Юрий Ревич

ЗАНИМАТЕЛЬНАЯ МИКРОЭЛЕКТРОНИКА

Санкт-Петербург

«БХВ-Петербург»

2007

УДК 681.3.06
ББК 32.973.26-04
Р32

Ревич Ю. В.

Р32 Занимательная микроэлектроника. — СПб.: БХВ-Петербург, 2007. — 592 с.: ил.

ISBN 978-5-9775-0080-7

Книга на практических примерах рассказывает о том, как проектировать, отлаживать и изготавливать современные электронные устройства в домашних условиях. Теоретические основы, физические принципы работы электронных схем и различных типов радиоэлектронных компонентов иллюстрируются практическими примерами в виде законченных радиолюбительских конструкций и дополняются советами по технологии изготовления любительской аппаратуры. На доступном уровне излагаются теоретические основы цифровой техники — математическая логика и различные системы счисления. Вторая часть книги полностью посвящена программированию микроконтроллеров, как основы современной электроники. Особое внимание уделяется обмену данными микроэлектронных устройств с персональным компьютером, приводятся примеры программ на Delphi.

Для широкого круга радиолюбителей

УДК 681.3.06
ББК 32.973.26-04

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Оформление обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.03.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 47,73.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0080-7

Оглавление

К читателю

Как известно, каждый сходит с ума по-своему. Есть люди, «сдвинутые» на собирании спичечных этикеток или монет, есть те, кто прыгает с парашютом, лазает по городской канализации или спускается на плотах по горным рекам северного Урала и Сибири. Одна из самых распространенных разновидностей подобных психических сдвигов — радиолюбительство.

Когда-то радиолюбители были действительно только «любителями радио» — просто потому, что в 10—20-х годах XX века, когда появились первые энтузиасты, кроме радиоэлектроники, никакой другой электроники не существовало. О, в те времена это было жутко престижное хобби! Это сейчас мы привыкли к магнитолам в каждом автомобиле и двум-трем телевизорам в каждой квартире, не считая электронной почты с Интернетом. А тогда сама возможность слушать кого-то с другого конца Земли казалась чудом, подвластным разве что рукам волшебника. Даже настройка на нужную станцию простого промышленного приемника, ставшего к тридцатым годам уже довольно обычным атрибутом не только в богатой Америке, но и в СССР, поначалу вызывала не меньше вопросов, чем сейчас — установка Windows на персональный компьютер.

Но довольно быстро, начиная с 30—40-х годов, электроника стала «широко простирать руки свои в дела человеческие». Термин «радиолюбительство» сохранился, но под ним стали уже понимать отнюдь не только увлечение радиопередатчиками и приемниками. Первым делом выделилась в отдельное направление звукозапись и все с ней связанное — различные усилители и акустические устройства. Затем электроника вторгается в электротехнику и управление различными механизмами. Потом начался компьютерный бум и стало модным все, относящееся к информационным технологиям (вообще-то в них, по справедливости, следовало бы включить и радио с телевидением).

В настоящее время не больше пятой части объема журнала «Радио», выходящего в нашей стране с 1924 года, посвящено именно радио. Следовало бы

придумать иное название, но понятие «радиолюбитель» прижилось, и ныне означает любого, кто увлекается электроникой (по крайней мере по-русски, в английском, например, языке это не так). Поэтому не удивляйтесь, если вы в этой книге, адресованной начинающим (и просто желающим повысить свою квалификацию) радиолюбителям, о радио вообще не увидите ни слова. Зато довольно часто будут упомянуты компьютеры — по причинам, которые вы поймете, прочитав эту книгу.

Занятие радиолюбительством в нашей стране некоторое время назад было не просто популярным, а даже очень модным. Стоит привести тот факт, что на первом этапе развития телевидения в СССР, в начале 1930-х годов, половина приемных устройств на основе диска Нипкова (продававшегося в магазинах) была изготовлена населением самостоятельно. Развитию технического творчества способствовало много причин: и относительно высокий уровень технического образования, и бесплатный доступ к компонентам (да-да, купить в свободной продаже что-то электронное было очень сложно, а вот вынести с завода или из НИИ, — всегда пожалуйста), и, наконец, то, что промышленность явно не справлялась с обеспечением потребности населения в «продвинутых» электронных устройствах, а качество тех, что выпускались, чаще всего было ниже всякой критики. Дешевле, лучше и интереснее было сделать все самому. Поэтому в те времена стать меломаном (в смысле «любителем качественной звукозаписи») означало фактически, что человек сам вынужден был изучать азы электроники и браться за паяльник («махать паяльной косой», как любил выражаться один мой знакомый).

Положение, конечно, резко изменилось с приходом в страну дешевого и качественного ширпотреба с Запада и Востока, и теперь уже вряд ли кто будет самостоятельно изобретать, скажем, карманный «плеер-дебильник». Но, как ни странно, радиолюбительство не только не погибло, но даже расцвело, поскольку стали доступны практически любые, как импортные, так и отечественные компоненты (хотя и за деньги) и, что тоже немаловажно, исчерпывающая документация к ним. В 1970-е годы какой-нибудь «Справочник по транзисторам» сметали с прилавков со скоростью, которой могли бы позавидовать сами братья Стругацкие. И вот уже «Чип и Дип» открывает пятый за десять лет супермаркет в Москве и хвастается миллионом посетителей в год, и на Митинском рынке в выходной не протолкнуться…

С программированием сложилась несколько иная ситуация. Для программирования нужен как минимум компьютер, а в области вычислительной техники наша страна, как известно, в свое время сильно отстала. Потому у нас даже сами компьютеры собирали своими руками, а «там» компьютерное любительство началось с появлением знаменитого ПК Altair в 1975 году,

который за довольно большие по тем временам деньги (500 долл.) продавался в виде набора «сделай сам».

Сейчас положение выправилось и какой-нибудь компьютер доступен практически каждому. Потому и современная радиоэлектроника, которая «завязана» как на собственно электронику, так и на программирование, стала вполне доступной радиолюбителям и не требует больших денежных вложений.

Невозможно провести четкую границу и указать — вот это любительство, а здесь начинается профессионализм. По данным Microsoft, количество дилетантов в области программирования, использующих такие популярные средства написания программ, как Visual Basic или Delphi, превышает число профессионалов-программистов примерно в три раза. Скорее всего, эти данные неполные, т. к. по понятным причинам многие любители этих программных пакетов не покупают их официально, и в статистику не попадают.

То же самое относится к области электроники: большинство рыночных продуктов создается профессионалами. Но в очень многих областях деятельности, где стоит задача создания «одноразового» прибора (или программы), необходимого сию минуту, а не после многочисленных согласований и утверждений, профессионалов привлекать просто нецелесообразно. Гораздо лучший результат может быть достигнут, если специалист в данной области деятельности не поленится освоить азы электроники и возьмется за конструирование прибора самостоятельно.

Что нового?

Автор был приятно удивлен, когда после выхода «Занимательной электроники» оказалось, что его задумка о создании популярной книжки, рассказывающей об основах этой дисциплины, полностью оправдалась — и по сей день приятные и иногда даже незаслуженно восторженные отзывы продолжают с завидной регулярностью поступать в мой электронный почтовый ящик. Между тем, со временем стали очевидными и недостатки этой книги. После краткого заочного совещания с издателями было решено не перерабатывать старую книжку, а создать новую, в которой отчасти заимствовать предыдущие наработки, устранив повторы, неточности, излишние, иногда устаревшие, подробности, и частично написать текст заново, расширив разделы, посвященные основе современной электроники — микроконтроллерам.

В первой части, где рассматриваются базовые принципы электроники и построение «обычных» схем на дискретных компонентах и микросхемах, сохранены все важнейшие начальные сведения, касающиеся физических основ электроники и призванные ввести читателя в курс дела. Новый раздел

задумывался для того, чтобы на популярном уровне как можно полнее раскрыть для читателя мир микроконтроллеров. Изучив эти материалы, внимательный читатель сможет проектировать достаточно функциональные устройства для многих областей применения. И эти знания могут стать хорошей основой для дальнейшего обучения с целью профессионального овладения предметом.

Как читать?

А теперь дам совет, который может показаться несколько неожиданным. Сначала попробуйте мысленно ответить на один вопрос, и пусть он не покажется вам идиотским: «Какова величина тока в комнатной розетке?». То, что этот вопрос отнюдь не такой дурацкий, как кажется, доказывают результаты ответов на него, полученные после опроса группы студентов одного технического вуза (по специальности, не связанной напрямую с электротехникой или электроникой): из нескольких десятков опрошенных только двое смогли дать вразумительный ответ. Итак, если вы, читатель, замялись с ответом или просто не уверены в его правильности, то вот обещанный совет, причем адресованный любому, независимо от возраста: прежде чем продолжать читать дальше, возьмите учебники физики за седьмой и восьмой классы и перечитайте главы, посвященные электричеству. Можете также захватить главу, посвященную строению атома. Еще лучше обратиться не к школьным учебникам, а изучить соответствующие главы из книги Ландсберга [17], где все то же самое изложено куда более увлекательно и подробно. Тогда вам будет намного легче читать эту книгу дальше.

> ***Заметки на полях***
>
> Попутно бросим камень в огород Минобразования: по глубокому убеждению автора, полезной информации в указанных учебниках более чем достаточно, а 10-й и 11-й классы — совершенно пустое времяпрепровождение для того, кто не собирается становиться специалистом-физиком. Автор не компетентен говорить то же самое о других предметах, но есть основания полагать, что и там положение ничуть не лучше. Зажмурившись, проследим за полетом этого «камня» и с сожалением отметим, что школьников по-прежнему пичкают массой совершенно излишних сведений, и приступим наконец к делу.

Для лучшего понимания всех заумных материй, излагаемых в этой книге, читателю с самого начала следует хорошо осознать важное правило, которое заключается в эквивалентности программ и «железа»: любую программу можно реализовать аппаратными средствами, любые (цифровые) аппаратные средства можно заменить исполняющейся программой. Существует формальное доказательство этого утверждения, и оно (утверждение) часто используется

на практике, например, во многих микропроцессорах различные процедуры выполняются за счет «зашитых» в них программ. Именно этот принцип привел к тому, что микроконтроллеры стали универсальными электронными приборами, способными заменить почти любую электронную схему, реализованную «по старинке».

Несколько слов о том, как пользоваться книгой. Она рассчитана на тех, кто делает все своими руками и занимается конструированием дома (поэтому, например, я не рекомендую компоненты для поверхностного монтажа, т. к. платы под них своими руками изготовить достаточно сложно, и еще труднее их отлаживать). Книга отличается от большинства имеющихся руководств тем, что почти все описанные здесь схемы подробно до мелочей разобраны шаг за шагом, так, чтобы при повторении конструкции у вас не возникало вопросов, зачем нужен тот или иной резистор и почему его сопротивление именно такое.

Но эта книга не самодостаточна. Хотя некоторые технические характеристики популярных компонентов (в основном отечественных) приведены в *Приложении 3*, но это капля в море. Вам как минимум понадобятся различные справочники — по транзисторам, микросхемам, и особенно по микроконтроллерам. К счастью, сейчас не требуется всю эту литературу иметь под рукой, поскольку многое доступно через Интернет. По западным компонентам в Интернете можно найти абсолютно все самые подробные описания и рекомендации по использованию (т. н. Data Sheets и Application Notes), которые в «бумажном» варианте все равно не существуют. Однако они, естественно, на английском, что осложняет задачу новичка. И хотя некоторые технические описания переведены на русский и такие переводы тоже можно разыскать (например, на сайтах **gaw.ru**, **telesys.ru** находится крупнейший русскоязычный форум по электронике, где можно обменяться различными сведениями и получить квалифицированный совет), все же следует, по возможности, обзавестись русскоязычными «бумажными» справочниками и пособиями, например [1—3, 5—7, 9].

В отличие от большинства других радиолюбительских изданий, описания конструкций в книге не приводятся. Во-первых, повторить устройство в точности с теми компонентами, которые приведены в описании, как правило, не получается, да это совершенно и не требуется, так что в большинстве случаев плату придется все равно перерабатывать. Во-вторых, лично я никогда не повторял опубликованных конструкций в точности, стараясь улучшить или упростить схему, и в этой книге вы почти всегда найдете рекомендации по улучшению характеристик или расширению функциональности описанного прибора.

Наконец, есть и еще один момент, скорее методического порядка — разрабатывая печатную плату и конструкцию устройства самостоятельно, вы намного лучше вникаете в работу схемы, после чего отладка и регулировка ее значительно упрощаются. Мое глубокое убеждение состоит в том, что плату нужно делать самостоятельно, под выбранный корпус, а не подгонять его габариты под имеющуюся плату, в результате чего самодеятельные изделия иногда бывают весьма уродливыми.

Как разрабатывать схемы?

И, наконец, рискуя утомить читателя, все же скажу несколько слов о том, как вообще следует разрабатывать и отлаживать схемы. Самый эффективный метод — «сборка» нужной схемы из готовых и заранее отлаженных фрагментов. Эта операция совершенно аналогична тому, как программисты «собирают» программы из готовых и заранее отлаженных процедур (вот он, принцип эквивалентности программ и «железа» в действии!). Каждая такая процедура представляет собой «черный ящик», у которого есть входы и выходы для обмена с другими частями программы, причем в общем случае вы даже не знаете, как она устроена внутри — точно так же, как вы не знаете, что именно размещается внутри микросхемы.

Вы берете микросхему, подсоединяете к ней внешние элементы в соответствии с рекомендациями производителя, и получаете готовый узел, который соединяете с другими подобными узлами. Точно так же следует поступать в случае, если узел представляет собой уже не отдельную микросхему, а законченный фрагмент устройства. Наращивая иерархию отлаженных заранее узлов, вы сэкономите гораздо больше времени, чем при сборке схемы целиком и дальнейшем выяснении, куда что припаяно. Кроме всего прочего, при таком образе действий, когда схема разбивается на отдельные узлы-кирпичики, ее гораздо легче «удержать» в голове и мысленно анализировать ее работу.

При рисовании схемы обязательно обозначайте на ней конкретные типы и значения параметров элементов, не откладывайте это до выполнения практической отладки схемы. Изменить значения вы всегда сможете, но все, что можно посчитать, нужно определить заранее — это сохранит вам очень много времени. Не верьте печатному слову и все рекомендации из литературы проверяйте на макетах (в конце концов у вас образуется библиотека схем таких самостоятельно отлаженных узлов). Отладив все, обязательно нанесите на чертеж схемы полученные в результате точные значения компонентов (те, что еще требуют окончательной подгонки, обозначаются звездочкой), проверьте правильность соединения этих узлов и разводку питания, и только затем собирайте всю схему целиком (сначала

на макетной плате). И только убедившись в работоспособности макета схемы, переносите ее на настоящую рабочую плату.

Если вы разрабатываете серьезный прибор, который должен служить годами, постарайтесь заложить в разработку время и деньги, необходимые для выполнения следующих этапов:

- ❒ разработка технического задания, с возможно более подробным описанием требуемой функциональности;

> ***Совет***
>
> На этом этапе не стоит пренебрегать мелочами, особенно если вы работаете «на сторону», а не для себя. Так, будет очень печально, если вам заказали измеритель температуры, и в конце разработки выяснится, что он должен круглогодично работать на улице. Созданный вами на домашнем столе датчик, естественно, в таких условиях быстро выйдет из строя. Впрочем, подобные накладки чаще касаются технологии изготовления плат, конструкции и подбора деталей, а не собственно схемотехники, но лучше все это учесть заранее.

- ❒ разработка принципиальной схемы с отладкой отдельных узлов на макетах;
- ❒ изготовление полного макета и его отладка;
- ❒ разработка окончательной принципиальной схемы, подбор деталей и разработка печатной платы;
- ❒ изготовление и отладка опытного образца, корректировка печатной платы;
- ❒ изготовление окончательного варианта печатной платы, корпуса и монтаж прибора.

Приведенный идеальный вариант последовательности разработки редко осуществим на практике: либо времени не хватает, либо денег, либо того и другого. Есть одна известная фирма, которая занимается разработкой заказных электронных устройств, так там берут несколько «килобаксов» только за написание технического задания. И они правы! На практике же часто получается так, что макетный либо опытный образец сразу становится окончательным. И все же по мере возможности не пренебрегайте этими промежуточными этапами, — поверьте, так получится намного эффективнее, чем собрать все сразу, а потом в лучшем случае обнаружить, что ничего не работает, а в худшем — выветривать из комнаты очень неприятный и стойкий запах горелой пластмассы. Учтите, что почти ни одна новая схема никогда не работает сразу, будьте к этому готовы и заранее наберитесь терпения.

Итак, приступим.

Юрий Ревич <revich@homepc.ru>

Автор выражает благодарность Юрию Певзнеру за консультации по микроконтроллерам Atmel AVR. Отдельно хочется поблагодарить за теплое и внимательное отношение сотрудников издательства "БХВ-Петербург" Игоря Шишигина, зам. главного редактора, и Григория Добина, зав. редакцией, а также всех остальных известных и неизвестных мне работников издательства.

Схемы, чертежи и фотографии компонентов подготовлены автором. Все остальные иллюстрации взяты из официальных источников для прессы, за исключением фотографии первого транзистора из *главы 3* и портрета Клода Шеннона из *главы 7*, любезно предоставленных автору корпорацией Lucent Technologies Inc./Bell Labs в лице ее сотрудницы Франциски Мэттьюз (Francisca Matthews).

Часть I

ЭЛЕКТРОНИКА

БЕЗ ПРОГРАММИРОВАНИЯ

Глава 1

Основные физические законы в микроэлектронике

— Уйди-уйди! — закричал вампир. — Мы так не договаривались. Я боюсь электричества.

Кир Булычев «Вампир Полумракс»

В наше время нередко можно встретить «эксперта» по английской культуре, не знающего английского языка, или «программиста», не разбирающегося в математической логике. Не будем им уподобляться, тем более что практическая электроника совсем не требует знаний на уровне физико-математического факультета МГУ. Вполне работоспособные схемы можно создавать и проектировать, обладая лишь багажом сведений в пределах 8-го класса средней школы, но уж в базовых понятиях из области электричества желательно ориентироваться как можно свободнее. Мы и начнем с того, что проясним их для себя раз и навсегда.

Чем отличается ток от напряжения?

Дурацкий вопрос, скажете вы? Отнюдь. Опыт показал, что не так уж и много людей могут на него ответить правильно. Известную путаницу вносит и язык: в выражениях вроде «имеется в продаже источник постоянного тока 12 В» смысл искажен. На самом деле в данном случае имеется в виду, конечно, источник напряжения, а не тока, т. к. ток в вольтах не измеряется. Самое правильное будет сказать — «источник питания постоянного напряжения 12 вольт», а написать можно и «источник питания =12В» где символ «=» обозначает, что это именно постоянное напряжение, а не переменное. Впрочем, и в этой книге мы тоже иногда будем «ошибаться» — язык есть язык.

Чтобы разобраться во всем этом, для начала напомним строгие определения из учебника (зазубривать их — очень полезное занятие!). Итак, *ток*, точнее,

его величина, *есть количество заряда, протекающее через сечение проводника за единицу времени*: $I = Q/t$. Единица измерения тока — *ампер*, а ее размерность — кулоны в секунду (здесь и далее, кроме оговоренных случаев, мы будем употреблять систему единиц СИ). Знание сего факта пригодится нам позднее. Куда более запутанно выглядит определение напряжения, как *разности потенциалов между двумя точками пространства*. Измеряется она в *вольтах* и размерность этой единицы измерения — джоуль на кулон, т. е. $U = E/Q$. Почему это так, легко понять, вникнув в смысл строгого определения величины напряжения: *1 вольт есть такая разность потенциалов, при которой перемещение заряда в 1 кулон требует затраты энергии, равной 1 джоулю*.

В этой главе мы будем говорить о постоянном токе и напряжении. Все это наглядно можно представить себе, сравнив проводник с трубой, по которой течет вода. При таком сравнении величина тока есть количество (расход) протекающей воды за секунду. Это довольно точная аналогия, роль молекул воды играют бегущие по проводнику электроны. Тогда напряжение предстанет, как разность давлений на входе и выходе трубы, за счет которой поток приобретает способность к движению.

Чаще всего труба заканчивается открытым краном, так что давление на выходе равно атмосферному давлению, и его можно принять за нулевой уровень. Точно так же в электрических схемах существует общий провод (или «общая шина» — в просторечии для краткости ее часто называют «землей», хотя это и не совсем точно), с нулевым потенциалом, относительно которого отсчитываются все напряжения в схеме. Обычно (но не всегда!) за общий провод принимают минусовой вывод основного источника питания схемы.

Итак, вернемся к вопросу в заголовке: чем же отличается ток от напряжения? Правильный ответ будет звучать так: ток — это количество электричества, а напряжение — мера его потенциальной энергии, способности к движению.

Напряжение и ток обычно связаны между собой. Слово «обычно» я употребил потому, что в некоторых случаях — для источников напряжения или тока, о которых мы поговорим в этой главе далее — от этой связи стараются избавиться, хотя полностью это сделать никогда не удается. Если вернуться к аналогии с трубой, то легко представить, как при возрастании давления (напряжения) увеличивается количество протекающей жидкости (ток), т. е. зависимость тока от напряжения довольно наглядна. Сложнее уяснить обратную зависимость: как ток влияет на напряжение. Для этого нужно сначала понять, что такое *сопротивление*.

Сопротивление

Вплоть до середины XIX века физики не знали, как выглядит зависимость тока от напряжения. Этому есть одна важная причина. Попробуйте сами экспериментально выяснить, как выглядит график этой зависимости. Схема эксперимента приведена на рис. 1.1, а примерные результаты — на рис. 1.2.

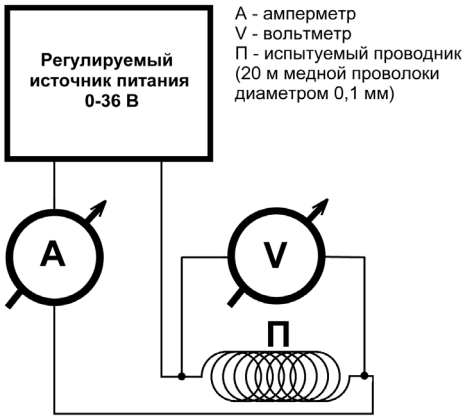


Рис. 1.1. Схема эксперимента по проверке закона Ома

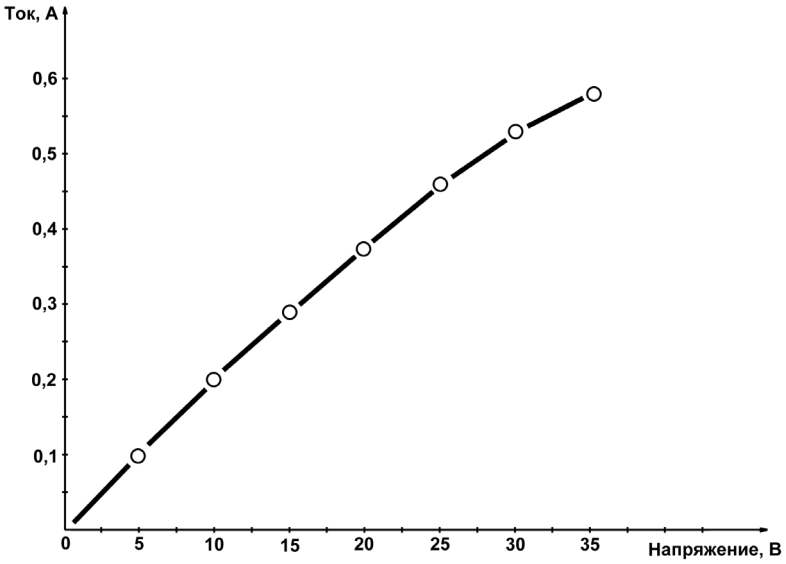


Рис. 1.2. Примерные результаты проверки закона Ома

Показанные на графике результаты весьма приблизительны, т. к. вид кривой будет сильно зависеть от того, как именно выполнен проводник (R1 на рис. 1.2): намотан ли он плотно или редко на толстый массивный каркас или на тонкий, а также от температуры в комнате, сквозняка и еще от множества других причин. Именно такое непостоянство и смущало физиков — меняется не только ход кривой (т. е. ток в общем случае непропорционален напряжению), но вид и форма этой зависимости весьма непостоянны и меняются как при изменении условий внешней среды, так и для различных материалов.

Понадобился гений Георга Ома, чтобы за всеми этими деревьями увидеть настоящий лес: а именно понять, что зависимость тока от напряжения описывается элементарно простой формулой: $I = U/R$. А все несуразности проистекают от того, что сама величина сопротивления R зависит от материала проводника и от условий внешней среды, в первую очередь от температуры. Так, в нашем эксперименте загиб кривой вниз происходит потому, что при прохождении тока проводник нагревается, а сопротивление меди с повышением температуры увеличивается (примерно на 0,4% на каждый градус). А вот сама величина этого нагрева зависит от всего, что угодно: намотайте провод поплотнее и заверните его в асбест, он будет нагреваться сильнее, а размотайте его и поместите на сквозняк — нагрев резко уменьшится.

В ознаменование заслуг Георга Ома единица измерения сопротивления так и называется — *ом*. Согласно формуле закона Ома, приведенной в предыдущем абзаце, *1 Ом есть сопротивление такого проводника, через который течет ток в 1 А при напряжении на его концах, равном 1 В*. Обратная сопротивлению величина называется *проводимостью* и измеряется в *сименсах*, названных так в честь другого ученого: 1 Сименс = 1/Ом. В электронике почти всегда оперируют величиной сопротивления, так что сименсы мы в основном оставим для физиков, хотя иногда прибегать к ним приходится.

Сопротивление проводника зависит от его геометрических размеров: оно увеличивается пропорционально длине и уменьшается пропорционально площади сечения: $R = \rho \cdot L/S$. Большое практическое значение имеет коэффициент пропорциональности ρ — т. н. удельное сопротивление материала проводника. При определенной температуре (обычно берется 20 °C) эта величина почти постоянна для каждого материала. «Почти» я тут написал потому, что на самом деле эта величина сильно зависит от химической чистоты и даже от способа изготовления материала проводника. Поэтому для проводников употребляют очень чистые металлы, скажем, обычный медный провод изготавливают из меди с количеством примесей не более 0,1% (как говорят, с чистотой в «три девятки»). Это позволяет уменьшить сопротивление такого провода и избежать лишних потерь на его нагрев.

Удельное сопротивление проводника, по определению, есть сопротивление (Ом) проводника единичной площади ($м^2$) и длины (м). Если подставить эти величины в предыдущую формулу, вы получите размерность для удельного сопротивления Ом·$м^2$/м или просто Ом·м. Практически в таких единицах измерять удельное сопротивление страшно неудобно, т. к. для металлов величина получается крайне маленькой — представляете сопротивление куба меди с ребром в 1 м?! На практике часто употребляют единицу в 100 раз больше: Ом·см. Эта величина часто приводится в справочниках, но и она не слишком удобна для практических расчетов. Так как диаметр проводников измеряют обычно в миллиметрах (а сечение, соответственно, в квадратных миллиметрах), то на практике наиболее удобна старинная внесистемная единица Ом·$мм^2$/м, которая равна сопротивлению проводника сечением в 1 квадратный миллиметр и длиной 1 метр. Для того чтобы выразить «официальный» Ом·м в этих единицах, нужно умножить его величину на 10^6, а для Ом·см — на 10^4. Посмотрев в справочнике величину удельного сопротивления меди (0,0175 Ом·$мм^2$/м при 20°C), мы легко можем вычислить, что сопротивление проводника с параметрами, приведенными на рис. 1.1, составляет около 45 Ом (проверьте!).

Заметки на полях

Надо сказать, что человечество весьма преуспело в изготовлении специальных материалов, имеющих коэффициент удельного сопротивления, мало зависящий от температуры. Это, прежде всего, специальные сплавы, константан и манганин, температурный коэффициент сопротивления (ТКС) которых в несколько сотен раз меньше, чем у чистых металлов. А для обычных стандартных углеродистых или металлопленочных резисторов ТКС составляет приблизительно 0,1% на градус или меньше, т. е. примерно в 4 раза лучше, чем у меди. Есть и специальные прецизионные резисторы (среди отечественных это, например, С2-14, С2-29В, С5-61, проволочные С5-54В и др.), у которых этот коэффициент значительно меньше. Есть и другие материалы, у которых температурный коэффициент, наоборот, весьма велик (несколько процентов на градус, и при этом, в отличие от металлов, отрицателен) — из них делают т. н. термисторы, которые применяют в качестве чувствительных датчиков температуры. Для точного измерения температуры тем не менее используют чистые металлы — чаще всего платину и медь.

Схема с двумя резисторами

Познакомившись с понятием сопротивления и его особенностями, вспомним, для чего мы все это делали. Ах, да, мы же хотели понять, как практически представить зависимость напряжения от тока! Но ведь мы пока не умеем произвольно изменять ток в проводнике, так? Напряжение изменять просто — нужно взять регулируемый источник питания, как это изображено

на рис. 1.1, или, на худой конец, набор батареек, при последовательном соединении которых (1, 2, 3 и более штук) мы получим некий набор напряжений. А вот источников тока (именно тока, а не напряжения) мы еще не имеем. Как же быть?

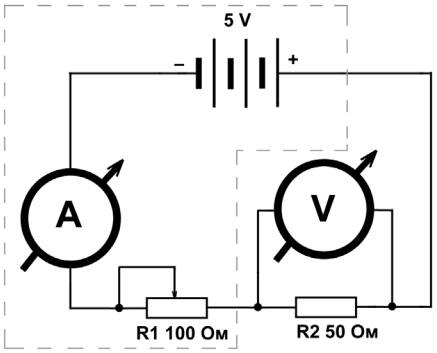


Рис. 1.3. Схема для изучения свойств цепи с двумя резисторами

Выход из этой ситуации показан на рис. 1.3 (заметьте, мы от схематического изображения проводника из длинной проволоки, имеющей некое сопротивление, перешли к стандартному обозначению резисторов, как это делается в настоящих электрических схемах, см. *Приложение 1*). Здесь нам уже не нужен регулируемый источник питания. Питается схема от батареи из трех гальванических элементов, например, типа D, соединенных последовательно (или одной типа 3336, см. *Приложение 2*). Каждый такой элемент (если он еще не был в эксплуатации) дает напряжение примерно 1,6 В, так что суммарное напряжение будет почти 5 В, как и указано на схеме (под нагрузкой и по мере истощения элементов напряжение немного упадет, но ошибка в данном случае не играет большой роли).

Как работает эта схема? Допустим, что движок переменного резистора R1 выведен в крайнее правое (по схеме) положение. Проследим путь тока от плюсового вывода батареи — амперметр, вывод движка резистора R1, крайний правый вывод R1, резистор R2, минусовой вывод батареи. Получается, что резистор R1 в схеме как бы не участвует, поскольку ток от плюсового вывода батареи сразу попадает на R2 (амперметр можно не принимать во внимание — далее мы узнаем, почему это так) и схема становится фактически такой же, как на рис. 1.1. Что покажут наши измерительные приборы? Вольтметр покажет напряжение батареи — 5 В, а показания амперметра легко вычислить по закону Ома: ток в цепи составит 5 В/50 Ом = 0,1 А

или 100 мА (напомним еще раз, что это значение приблизительное, т. к. напряжение батареи несколько меньше 5 В).

Теперь поставим движок R1 в среднее положение. Ток в цепи теперь пойдет от плюса батареи через амперметр, вывод движка R1, половину резистора R1, резистор R2 и далее, как и раньше, вернется к минусу батареи. Как изменятся показания приборов? Раньше резистор R1 в деле не участвовал, а теперь участвует, хоть и половинкой. Соответственно, общее сопротивление цепи станет уже не 50 Ом (один резистор R2), а 50 (R2) + 50 (половинка R1), т. е. 100 Ом. Амперметр покажет уже не 100 мА, а 5 В/100 Ом = 0,05 А или 50 мА — в два раза меньше. А вот что покажет вольтметр? Так сразу и не скажешь, не правда ли? Придется считать, для этого рассмотрим отдельно участок цепи, состоящий из R2 с присоединенным к нему вольтметром. Очевидно, что току у нас деться некуда — все то количество заряда, которое вышло из плюсового вывода батареи, пройдет через амперметр, через половинку R1, через R2 и вернется обратно в батарею. Значит, и на этом отдельном участке, состоящем из одного R2, ток будет равен тому, что показывает амперметр, т. е. 50 мА. Получается, как будто резистор R2 подключен к источнику тока!

Замечание

На самом деле это не совсем точно — часть тока, хотя и очень небольшая, все же пойдет через вольтметр, минуя R2. Но на практике, особенно для современных вольтметров, этим всегда пренебрегают (см. далее).

И это действительно так — источник напряжения с последовательно включенным резистором (в данном случае это половинка R1) представляет собой источник тока (хотя и плохой). Так каковы же будут показания вольтметра? Очень просто: из закона Ома следует, что $U = I \cdot R$, где R — сопротивление нужного нам участка цепи, т. е. R2, и в данном случае вольтметр покажет $0{,}05 \cdot 50 = 2{,}5$ В. Эта величина называется *падением напряжения*, в данном случае — падением напряжения на резисторе R2. Легко догадаться, даже не подключая вольтметр, что падение напряжения на резисторе R1 будет равно тоже 2,5 В, причем его можно вычислить двумя путями: как разницу между 5 В от батареи и падением на R2 (2,5 В), или по закону Ома, аналогично расчету для R2.

Замечание

И это не совсем точно — амперметр тоже имеет некоторое сопротивление и может быть представлен в виде еще одного последовательного резистора. Но, как и в случае вольтметра, этим на практике пренебрегают.

А что будет, если вывести движок переменника в крайнее левое положение? Я сразу приведу результат: амперметр покажет 33 мА, а вольтметр — 1,66 В. Пожалуйста, проверьте это самостоятельно! Если вы получите те же значения, то это будет означать, что вы усвоили закон Ома и теперь умеете отличать ток от напряжения.

Источники напряжения и тока

В схеме на рис. 1.3 мы можем выделить, как показано пунктиром, ее часть, включив туда батарейку и переменный резистор R1. Тогда этот резистор (вместе с сопротивлением амперметра, конечно) можно рассматривать, как *внутреннее сопротивление* источника электрической энергии, каковым выделенная часть схемы станет для нагрузки, роль которой будет играть R2. Любой источник, как легко догадаться, имеет свое внутреннее сопротивление (электронщики часто употребляют выражение «выходное сопротивление») — хотя бы потому, что у него внутри есть провода определенной толщины.

Но на самом деле не провода служат ограничивающим фактором. В *главе 2* мы узнаем, что такое мощность в строгом значении этого понятия, а пока, опираясь на интуицию, можно сообразить: чем мощнее источник, тем меньше у него должно быть свое внутреннее сопротивление, иначе все напряжение «сядет» на этом сопротивлении, и на долю нагрузки ничего не достанется. На практике так и происходит. Если вы попытаетесь запустить от набора батареек типа АА какой-нибудь энергоемкий прибор, питающийся от источника с низким напряжением (вроде настольного сканера или ноутбука), то устройство, конечно, не заработает, хотя формально напряжения должно хватать, — напряжение уменьшится почти до нуля. А вот от автомобильного аккумулятора, который гораздо мощнее, все получится, как надо.

Такой источник, у которого внутреннее сопротивление мало по отношению к нагрузке, называют еще *идеальным источником напряжения* (физики предпочитают название *идеальный источник ЭДС*, т. е. «электродвижущей силы», на практике, однако, это абстрактное понятие встречается реже, чем менее строгое, но всем понятное «напряжение»). К ним относятся, в первую очередь, все источники питания: от батареек до промышленной сети.

Наоборот, *идеальный источник тока*, как нетрудно догадаться, обязан обладать бесконечным внутренним сопротивлением — только тогда ток в цепи совсем не будет зависеть от нагрузки. Понять, как источник реального тока (не бесконечно малого) может обладать бесконечным выходным сопротивлением, довольно трудно, и в быту таких источников вы не встретите. Однако

уже обычный резистор, включенный последовательно с источником напряжения (не тока!), как R1 на рис. 1.3, при условии, что сопротивление нагрузки мало (R2<<R1), может служить хорошей моделью источника тока. Еще ближе к идеалу транзисторы в определенном включении, и мы с этим разберемся позднее.

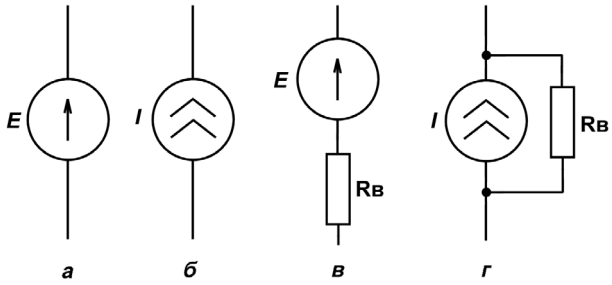


Рис. 1.4. Источники тока и напряжения:
а — обозначение идеального источника напряжения;
б — обозначение идеального источника тока;
в — эквивалентная схема реального источника напряжения;
г — эквивалентная схема реального источника тока

Источники напряжения и тока обозначаются на схемах так, как показано на рис. 1.4, *а* и *б*. Не перепутайте, логики в этих обозначениях немного, но так уж принято. А *эквивалентные схемы* (их еще называют *схемами замещения*) реальных источников приведены на рис. 1.4, *в* и *г*, где $R_в$ обозначает внутреннее сопротивление источника. Как можно использовать эти эквивалентные схемы при анализе реальных цепей? Для этого нужно окончательно разобраться, как рассчитываются схемы с параллельным и последовательным включением резисторов.

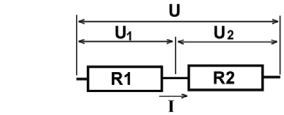
Параллельное и последовательное соединение резисторов и расчет схем

Схемы постоянного тока любой степени сложности всегда можно представить как совокупность резисторов и идеальных источников напряжения и тока. Для их расчета достаточно знать два очень простых закона, названных по имени физика XIX столетия Густава Роберта Кирхгофа (1824—1887).

Первый закон Кирхгофа формулируется так: *алгебраическая сумма токов в любом узле электрической цепи равна нулю*. Или еще проще: *сумма токов, направленных к данному узлу, равна сумме токов, направленных от него.*

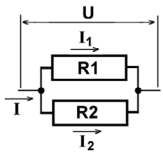
По сути он представляет одну из форм физических законов сохранения — ведь заряды не могут возникнуть из ничего, соответственно, сколько прибыло зарядов в некую точку, столько из нее обязано уйти.

Второй закон Кирхгофа гласит: *алгебраическая сумма падений напряжения вдоль любого замкнутого контура электрической цепи равна нулю*. Его легко проиллюстрировать на примере нашей схемы рис. 1.3 — там сумма падений напряжений на всех резисторах (включая внутреннее сопротивление батарейки, сопротивление амперметра, которым мы пренебрегали, и т. д.) равна напряжению батарейки. Иначе и быть не может — куда оно, напряжение батарейки, тогда денется?



Последовательное соединение резисторов

R = R1 + R2



Параллельное соединение резисторов

1/R = 1/R1 + 1/R2

Рис. 1.5. Последовательное и параллельное соединение резисторов

Из законов Кирхгофа вытекают очень часто применяющиеся на практике правила последовательного и параллельного соединения резисторов: при последовательном соединении складываются сопротивления резисторов, а при параллельном складываются их проводимости, которые по определению, данному ранее, есть величины, обратные сопротивлению (рис. 1.5). Понять, почему правила именно таковы, можно, если рассмотреть течение токов в обоих случаях.

- При последовательном соединении ток I через резисторы один и тот же, поэтому падения напряжения на них складываются ($U = U_1 + U_2$), что равносильно сложению сопротивлений.
- При параллельном соединении, наоборот, равны падения напряжений U, а складывать приходится токи ($I = I_1 + I_2$), что равносильно сложению

проводимостей. Если вы не поняли сказанное, то посидите над рис. 1.5 с карандашом и бумагой и выведите выражения закона Ома для каждого из резисторов — и все станет на свои места.

Из этих определений вытекает также несколько практических правил, которые полезно заучить:

- ❒ При последовательном соединении:
  - сумма двух резисторов имеет сопротивление всегда больше, чем сопротивление резистора с бóльшим номиналом (правило «больше большего»);
  - если номиналы резисторов равны, то суммарное сопротивление окажется вдвое больше каждого номинала;
  - если номиналы резисторов различаются во много раз, то общее сопротивление примерно равно большему номиналу. Типичный случай: в примере на рис. 1.3 мы игнорируем сопротивления проводов и амперметра, т. к. они много меньше сопротивлений резисторов.
- ❒ При параллельном соединении:
  - сумма двух резисторов имеет сопротивление всегда меньше, чем сопротивление резистора с меньшим номиналом (правило «меньше меньшего»);
  - если номиналы резисторов равны, то суммарное сопротивление будет вдвое меньше каждого номинала;
  - если номиналы резисторов различаются во много раз, то общее сопротивление примерно равно меньшему номиналу. Это также можно иллюстрировать на примере рис. 1.3, где мы игнорируем наличие вольтметра, включенного параллельно R2, т. к. его сопротивление намного больше сопротивления резистора.

Знание этих правил поможет вам быстро оценивать схему, не занимаясь алгебраическими упражнениями и не прибегая к помощи калькулятора. Даже если соотношение сопротивлений не попадает под перечисленные случаи, результат все равно можно оценить «на глаз» с достаточной точностью. При параллельном соединении, которое представляет большую сложность при расчетах, для такой оценки нужно прикинуть, какую долю меньшее сопротивление составляет от их арифметической суммы — именно во столько раз приблизительно снизится их общее сопротивление по отношению к меньшему.

Проверить это легко: рассмотрим ситуацию, когда сопротивления равны. В этом случае одно сопротивление составляет ½ часть их суммы, т. е. общее сопротивление должно снизиться вдвое, как и есть на самом деле. Возьмем

более сложный случай: одно сопротивление пусть имеет номинал 3,3 кОм, второе — 6,8 кОм. В соответствии с изложенным мы будем ожидать, что общее сопротивление должно быть на 30% меньше, чем 3,3 кОм, т. е. 2,2 кОм (3,3 составляет примерно одну треть от суммы 3,3+6,8, т. е. общее сопротивление должно быть меньше, чем 3,3, на треть от этого значения, равную 1,1 — в результате и получаем 2,2 кОм). Если мы проверим результат, полученный такой прикидкой в уме, точным расчетом, то мы получим в результате 2,22 кОм, что очень неплохо.

В большинстве случаев нам такой точности и не потребуется — помните, что и сами сопротивления имеют разброс по номиналу, и для обычных схем допуски на номиналы стандартных компонентов могут быть довольно значительными (по крайней мере, в правильно составленных схемах). Если же схема в некоторых случаях должна все же иметь какие-то строго определенные параметры, то с помощью стандартных компонентов вы все равно этого не добьетесь, т. к. параметры, образно выражаясь, будут «гулять» (в пределах допусков, естественно) от дуновения ветерка из форточки. В таких случаях надо применять прецизионные резисторы и конденсаторы, а во времязадающих цепях использовать кварцевые резонаторы. Но составлять схему так, чтобы она теряла работоспособность от замены резистора 1 кОм на 1,1 кОм — не наш метод!

Теперь понятно для чего служат эквивалентные схемы: вы просто включаете внутренние сопротивления в вашу цепь и учитываете их при расчетах, как будто они там специально поставлены. Отметим, что с помощью эквивалентных схем можно представить в принципе любой радиоэлектронный компонент — иногда это очень удобно.

Вольтметр и амперметр в измеряемой цепи

Теперь нам несложно понять, какое поведение ожидается от амперметра и вольтметра. Амперметр всегда включается в измеряемую цепь последовательно, ведь через него должен проходить тот же ток, что и во всей цепи. Но если он будет иметь большое собственное сопротивление, то внесет существенную погрешность, тогда на нем будет падать заметная часть напряжения, это уменьшит падение напряжения на остальных резисторах и суммарный ток.

По сути реальный амперметр является, как это не парадоксально, вольтметром — он измеряет падение напряжения на его собственном внутреннем сопротивлении, меняя значение которого (устанавливая т. н. *шунты* — специальные резисторы), вы переключаете диапазоны измерения. Потому сделать

его сопротивление равным нулю не получается, но удается сделать значение это достаточно малым, чтобы позволить себе пренебречь его влиянием.

Заметки на полях

Вот это-то замечательное свойство современных амперметров одновременно и является их самым слабым местом: достаточно перепутать и включить амперметр не последовательно, а параллельно источнику питания (подобно вольтметру), как через него, в полном соответствии с законом Ома, потечет огромный ток, ограниченный только возможностями источника. Действительно, типичное сопротивление амперметра составляет порядка нескольких миллиом, что даже при 5-вольтовом источнике дает токи в 1000 А и более! На самом деле никакой нормальный источник питания (включая даже бытовую электросеть) такого тока отдать не сможет, но того, что сможет, будет достаточно, чтобы прибор сгорел. Однако не отчаивайтесь — обычно в хороших мультиметрах внутри стоит плавкий предохранитель, а в самых качественных — даже самовосстанавливающийся. Если ваш прибор вдруг перестал показывать ток (а вы можете и не заметить, как случайно подсоединили его в режиме измерения тока к выводам питания), то прежде всего разберите его и проверьте этот самый предохранитель. Кстати, именно для того, чтобы дополнительно защитить мультиметр от описанных неприятностей, клемму для подключения щупа в режиме измерения тока всегда делают отдельно.

Рис. 1.6. Современный мультиметр

Наоборот, вольтметр подключается всегда параллельно, и потому, чтобы не вносить погрешности, должен иметь как можно большее сопротивление. По сути аналоговый вольтметр является амперметром, измеряя тот мизерный ток, который ответвляется из внешней цепи на это большое сопротивление. Однако это относится только к традиционным стрелочным приборам — современные вольтметры, построенные на интегральных схемах, ток от измеряемой цепи практически не потребляют, и потому много ближе к идеалу, чем амперметры. Это касается не только приборов, измеряющих напряжение (например, мультиметров в режиме измерения напряжения — рис. 1.6), но и других устройств, которые со стороны схемы выглядят, как вольтметры, например, осциллографов, различных аналогово-цифровых преобразователей и т. п.

Из-за этих свойств испортить мультиметр в режиме вольтметра гораздо труднее — если вы его по ошибке включите последовательно, то перестанет работать схема, а не прибор. Однако теоретически сжечь можно и вольтметр, если его включить на предел в 0,2 В, а подсоединить к сети 220 В. Поэтому, если вы не располагаете прибором с автоматическим выбором предела измерения, будьте внимательны, соблюдая и тип измеряемого напряжения (постоянное или переменное), и его возможный предел. На самом деле современные мультиметры обычно выдерживают многократное превышение предельного значения (например, 250 В на установленном пределе 2 В), но когда вы не знаете заранее, каково может быть напряжение в измеряемой точке, то начинать все же надо всегда с самого большого значения и постепенно его снижать.

Глава 2

Переменный ток, мощность и конденсаторы

— Роман Петрович, — сказал он. — Будьте любезны, включите, пожалуйста, рубильник.

А. и Б. Стругацкие
«Понедельник начинается в субботу»

Электрохимические (гальванические) элементы, с которыми мы экспериментировали в *главе 1*, есть источники постоянного напряжения. Определение «постоянное» не означает, что такое напряжение вообще не меняется. Отнюдь — типичный график зависимости напряжения от времени для гальванических элементов разных типов приведен на рис. 2.1 (это так называемые *разрядные кривые*). Причем зависит оно не только от времени. Отдельные пики на графиках относятся к моментам, когда нагрузка отключалась, при этом напряжение элемента скачкообразно росло, а затем, при подключении ее, снова падало — теперь вы знаете, что это происходит за счет внутреннего сопротивления источника, которое, как видно из графика, само может меняться по мере разряда элемента.

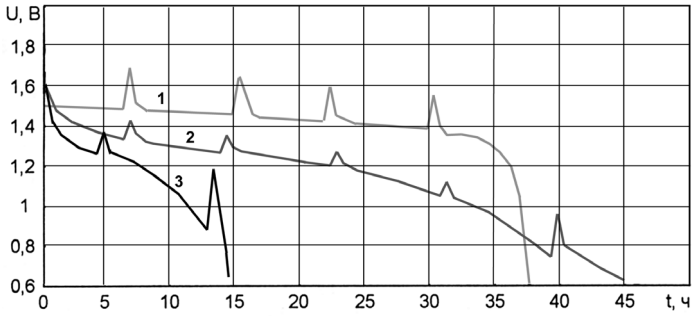


Рис. 2.1. Зависимость напряжения от времени для гальванических элементов различного типа при токе нагрузки 100 мА: 1 — литиевый; 2 — алкалайновый; 3 — марганец-цинковый. (По данным И. Подушкина, «Радио», № 2, 2004)

Заметки на полях

Этот график, между прочим, хорошо иллюстрирует то положение, что наиболее выгодными по соотношению цена/продолжительность работы являются щелочные («алкалайновые») элементы: обычные марганец-цинковые примерно в два раза дешевле, но имеют в три раза меньший срок службы, а единственное преимущество очень дорогих литиевых — в том, что их напряжение меньше снижается за все время разряда (зато потом быстро падает до нуля).

Переменное напряжение

Итак, постоянное напряжение на деле может быть совсем и не постоянным. Даже для самых лучших источников питания оно обязательно немножко «гуляет» — в зависимости от тока нагрузки и ее характера. Что же тогда называть переменным напряжением? Строгого определения, как ни странно, не существует — часто приводимое в учебниках выражение «напряжение, которое изменяется с течением времени», как видите, прекрасно подходит и к нашим батарейкам, хотя они являются типичными источниками постоянного напряжения. Поэтому мы договоримся переменными называть такие напряжения или токи, которые изменяются во времени, во-первых, периодически, во-вторых, делают это «сами по себе», без влияния со стороны нагрузки.

Замечание

Строго говоря, называть гальванические элементы батарейками неправильно — батареей называют источник, составленный из нескольких отдельных элементов. Но так уж повелось в разговорном языке, да потом не всегда точно известно, является ли данный элемент именно элементом или батареей (например, пальчиковые батарейки AA — это элемент, 9-вольтовая «Крона» — батарея).

Слово «периодически» означает, что, начиная с какого-то момента времени, форма графика такой величины в целом повторяется снова и снова. Время повтора называется *периодом переменной величины*. Выражение «повторяется в целом» означает, что изменения могут быть, но либо непринципиальные (скажем, за счет наложения шумов), либо период наступления этих изменений много больше периода самого сигнала.

Заметки на полях

Впрочем, и такое определение не будет строгим, — очевидное исключение представляют собой электрические колебания в устройствах для записи и воспроизведения звука, т. к. ни строгой периодичности, ни повторяемости вы там не найдете, если не рассматривать, конечно, звук одиночной струны или камертона. И тем не менее преобразованные в электричество звуковые колебания — типичный пример переменного тока. На чем и успокоимся, поскольку это далеко не единственный случай, когда очевидным вещам невозможно дать

строгого определения, скорее наоборот — надо еще сильно поискать в природе нечто такое, что можно было бы однозначно определить, не впадая в очевидные противоречия с реальностью. Специалист как раз и отличается от неспециалиста тем, что всегда *понимает, о чем речь*.

Как вы хорошо знаете из школьного курса физики, наиболее простым и наглядным примером переменной величины является величина, изменяющаяся во времени по синусоидальному закону. На рис. 2.2 приведен график подобной величины, построенный в условном масштабе. По оси ординат могут быть отложены как напряжение или ток, так и любой другой физический параметр. Отрезок времени T есть период изменения, а величина A носит название амплитуды и представляет собой максимальное значение нашей переменной в одном периоде (отметим, что для синусоидального закона минимальное значение — в области ниже оси абсцисс — строго равно максимальному).

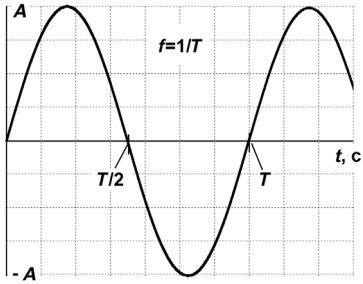


Рис. 2.2. График простого синусоидального колебания

Величина, обратная периоду, носит название частоты и обозначается буквой f (см. формулу на рис. 2.2 вверху). Для нее придумана специальная единица измерения — это хорошо всем знакомый герц (Гц), названный так в честь немецкого физика XIX века Генриха Герца, доказавшего существование радиоволн. Как следует из определения частоты, размерность герца есть единица, деленная на секунду: 1 Гц = 1/с, т. е. колебание с частотой 1 Гц имеет период повторения ровно 1 секунду. Соответственно, 1 кГц (килогерц) означает, что в одной секунде укладывается тысяча периодов, 1 МГц (мегагерц) — миллион периодов и т. п.

В дальнейшем под периодической величиной мы будем подразумевать напряжение (для тока все выглядит аналогично). Математический закон, описывающий поведение синусоидального напряжения (U) от времени (t), выглядит так:

$$U = A \cdot \sin(2\pi ft). \qquad (2.1)$$

Здесь π есть хорошо нам знакомое иррациональное число «пи», т. е. отношение длины окружности к диаметру, равное 3,1415… Произведение $2\pi f$ носит специальное название «круговая частота» и обозначается буквой ω. Круговая частота — это величина угла (измеряемого в радианах), пробегаемого нашей синусоидальной функцией за секунду. Так как мы не будем заниматься радиочастотной техникой, то углубляться в дальнейшие абстракции вроде представления переменных колебаний через комплексные числа, где понятие круговой частоты является ключевым, не стоит, для практических нужд нам хватит приведенных наглядных определений обычной частоты.

А что будет, если график немного подвигать вдоль оси абсцисс? Как видно из рис. 2.3 (кривая 2), это равносильно признанию того факта, что в нулевой момент времени наше колебание не равно нулю. На рис. 2.3 оно начинается с максимального значения амплитуды. При этом сдвигаются моменты времени, соответствующие целому и половине периода, а в уравнении появится еще одна величина, обозначаемая буквой φ и измеряемая в единицах угла — радианах:

$$U = A \sin (2\pi ft + \varphi). \qquad (2.2)$$

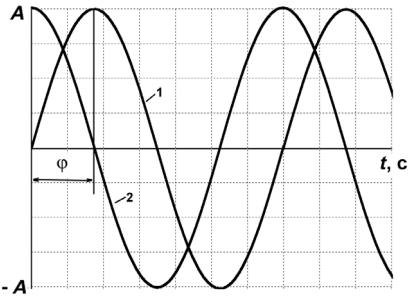


Рис. 2.3. График синусоидальных колебаний, различающихся по фазе:
1 — исходное колебание; 2 — сдвинутое на четверть периода

Величина φ носит название *фазы*. Взятое для одного отдельного колебания, значение фазы не имеет особого смысла, т. к. мы всегда можем сместить точку начала отсчета времени так, чтобы привести уравнение к виду (2.1), а, соответственно, график — к виду рис. 2.2, и при этом ничего не изменится. Все будет иначе, если мы имеем два связанных между собой колебания, скажем, напряжения в разных точках одной схемы. В этом случае нам может быть важно, как соотносятся их величины в каждый момент времени, и тогда фаза одного переменного напряжения относительно другого (называемая в этом случае сдвигом или *разностью фаз*) и будет характеризовать такое соотно-

шение. Для двух колебаний, представленных на рис. 2.3, сдвиг фаз равен 90° ($\pi/2$ радиан). Для наблюдения таких колебаний требуется многоканальный или многолучевой осциллограф — в обычном фаза колебания определяется только настройками синхронизации, и, рассматривая их по отдельности, разницы вы не увидите.

Интересно, что получится, если мы суммируем такие «сдвинутые» колебания? Не надо думать, что это есть лишь теоретическое упражнение — суммировать электрические колебания разного вида приходится довольно часто. Математически это будет выглядеть, как сложение формул (2.1) и (2.2):

$$U = A_1 \sin(2\pi f_1 t) + A_2 \sin(2\pi f_2 t + \varphi). \qquad (2.3)$$

Обратите внимание, что в общем случае амплитуды и частоты колебаний различны (на рис. 2.3 они одинаковы!).

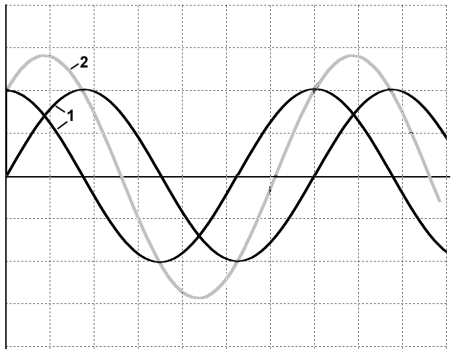


Рис. 2.4. Суммирование колебаний: 1 — исходные колебания; 2 — их сумма

Чтобы представить себе наглядно результат, надо проделать следующее: скопировать графики на миллиметровку, разделить период колебаний на несколько отрезков и для каждого из них сложить величины колебаний (естественно, с учетом знака), а затем по полученным значениям провести график. Так делали все — от школьников до ученых-математиков — еще лет двадцать назад. Теперь, конечно, удобнее проделать то же самое на компьютере: либо загрузить значения функций в Excel, либо (что, на мой взгляд, гораздо проще) написать программу, которая вычисляет значения по формуле (2.3) и строит соответствующие графики. Если сложить два колебания, которые были представлены на рис. 2.3, то получится результат, показанный на рис. 2.4. Обратим внимание на тот факт, что период результирующего колебания в точности равен периодам исходных, если они одинаковы, а вот амплитуда и фаза будут отличаться.

Результаты таких упражнений могут быть весьма неожиданными и вовсе неочевидными: скажем, при сложении двух синусоидальных колебаний с одинаковой частотой и амплитудой, как на рис. 2.3—2.4, но со сдвигом фаз в 180° (когда колебания находятся в противофазе), их сумма будет равна нулю на всем протяжении оси времени! А если амплитуды таких колебаний не равны друг другу, то в результате получится такое же колебание, амплитуда которого равна разности амплитуд исходных. Этот факт иногда используется для того, чтобы получить нестандартные напряжения с трансформатора с несколькими обмотками — если их обмотки подключить последовательно (начало одной к концу другой, см. *главу 4*), то напряжения суммируются, а если их включить встречно (начало одной к началу другой), то напряжения вычтутся, причем при строго одинаковых обмотках напряжение на выходе будет равно нулю!

Если у вас есть какой-нибудь низковольтный трансформатор под рукой, то можете поэкспериментировать с соединением вторичных обмоток, учитывая при этом, что начала обмоток будут иметь нечетные номера, а концы — четные. Только не ошибитесь, и не замкните что-нибудь с сетевой (первичной) обмоткой — это опасно и для вас, и для трансформатора, и для предохранителей в квартире. Так что если трансформатор вам незнаком, то необходимо сначала добыть его описание и определить, где у него сетевая обмотка.

Значения напряжения, естественно, можно измерять любым мультиметром, но вот вопрос на засыпку: что именно будет показывать вольтметр переменного тока? Ведь измеряемая величина все время, с частотой 50 раз в секунду, меняется от минимального отрицательного до максимального положительного значения, т. е. в среднем равна нулю. Тем не менее вольтметр нам покажет совершенно определенное значение. Для ответа на вопрос, какое именно, отвлечемся от колебаний и поговорим об еще одной важнейшей величине, которая характеризует электрический ток: о *мощности*.

Мощность

Согласно определению, *мощность есть энергия (работа), выделяемая в единицу времени*. Единица мощности называется ваттом (Вт). По определению, 1 ватт есть такая мощность, при которой за 1 секунду выделяется (или затрачивается — смотря с какой стороны поглядеть) 1 джоуль энергии. Для электрической цепи ее очень просто подсчитать по закону Джоуля-Ленца: P (ватт) = U (вольт) · I (ампер). Если подставить в формулу мощности выражения связи между током и напряжением по закону Ома, то можно вывести

еще два часто употребляющихся представления закона Джоуля-Ленца: $P = I^2 \cdot R$ и $P = U^2/R$.

Заметки на полях

Формулу закона Джоуля-Ленца очень просто вывести из определений тока и напряжения (см. *главу 1*). Действительно, размерность напряжения есть джоуль/кулон, а размерность тока — кулон/секунда. Если их перемножить, то кулоны сокращаются и получаются джоули в секунду, что, согласно данному ранее определению, и есть мощность. Обратите также внимание на одно важное следствие из этих формул: мощность в цепи пропорциональна квадрату тока и напряжения. Это означает, что если повысить напряжение на некоем резисторе вдвое, то мощность, выделяющаяся на нем, возрастет вчетверо. Отметьте также, что от величины сопротивления мощность зависит линейно: если вы при том же источнике питания уменьшите сопротивление вдвое, то мощность в нагрузке возрастет также вдвое. Это именно так, хотя факт, что, согласно закону Ома, ток в цепи увеличится также вдвое, мог бы нас привести к ошибочному выводу, будто в этом случае выделяющаяся мощность возрастет вчетверо — если вы внимательно проанализируете формулировки закона Джоуля-Ленца, то поймете, где здесь «зарыта собака».

В электрических цепях энергия выступает чаще всего в виде теплоты, поэтому электрическая мощность в подавляющем большинстве случаев физически означает просто количество тепла, которое выделяется в цепи (если в ней нет электромоторов или, скажем, источников света). Вот и ответ на вопрос, который мог бы задать пытливый читатель еще при чтении первой главы: куда расходуется энергия источника питания, «гоняющего» по цепи ток? Ответ: на нагрев сопротивлений нагрузки, включенных в цепь. И даже если нагрузка представляет собой, скажем, источник света (лампочку или светодиод), то бóльшая часть энергии все равно уходит в тепло: КПД лампы накаливания (т. е. та часть энергии, которая превращается в свет), как известно, не превышает нескольких процентов. У светодиодов эта величина значительно выше, но и там огромная часть энергии уходит в тепло. Кстати, из этого следует, например, что ваш компьютер последней модели, который потребляет далеко за сотню ватт, также всю эту энергию переводит в тепло — за исключением исчезающе малой ее части, которая расходуется на свечение экрана и вращение жесткого диска. Такова цена информации!

Если мощность, выделяемая в нагрузке, превысит некоторую допустимую величину, то нагрузка просто сгорит. Поэтому различные типы нагрузок характеризуют предельно допустимой мощностью, которую они могут рассеять без необратимых последствий. А сейчас зададимся вопросом: что означает мощность в цепях переменного тока?

Что показывает вольтметр в цепи переменного тока

Для того чтобы понять смысл этого вопроса, давайте внимательно рассмотрим график синусоидального напряжения на рис. 2.2. В каждый момент времени величина напряжения различна, соответственно будет разной и величина тока через резистор нагрузки, на который мы подадим такое напряжение. В моменты времени, обозначенные $T/2$ и T (т. е. кратные половине периода нашего колебания), напряжение на нагрузке вообще будет равно нулю (ток через резистор не течет), а в промежутках между ними — меняется вплоть до некоей максимальной величины, равной амплитудному значению A. Точно так же будет меняться ток через нагрузку, а следовательно, и выделяемая мощность. Но процесс выделения тепла крайне инерционен — даже такой маленький предмет, как волосок лампочки накаливания, за 1/100 секунды, которые проходят между пиками напряжения в промышленной сети частотой 50 Гц, не успевает заметно остыть. Поэтому нас чаще всего интересует именно средняя мощность за большой промежуток времени. Чему она будет равна?

Для того чтобы точно ответить на этот вопрос, нужно взять интеграл: средняя мощность за период есть интеграл по времени от квадрата функции напряжения. Здесь мы приведем только результат: величина средней мощности в цепи переменного тока определяется т. н. *действующим значением напряжения* ($U_д$), которое для синусоидального колебания связано с амплитудным его значением ($U_а$) следующей формулой: $U_а = U_д \cdot \sqrt{2}$. Аналогичная формула справедлива для тока. Когда говорят «переменное напряжение 220 В», то всегда имеется в виду именно действующее значение. При этом амплитудное значение равно примерно 311 вольт, что легко подсчитать, если умножить 220 на корень из двух. Это всегда нужно учитывать при выборе компонентов для работы в сетях переменного тока. Если взять диод, рассчитанный на 250 В, то он легко может выйти из строя при работе в обычной сети, в которой мгновенное значение превышает 300 В, хотя действующее значение и равно 220. А вот для компонентов, обладающих эффектом нагревания (лампочек, резисторов и т. п.) при расчете допустимой мощности, следует подставлять именно действующее значение.

Называть действующее значение «средним» неверно, правильнее — *среднеквадратическим* (по способу вычисления — через квадрат функции от времени). Но существует и понятие среднего значения, причем не одно, а даже два. Просто «среднее» (строго по смыслу названия, т. е. среднее арифметическое) — сумма всех мгновенных значений за период. Так как нижняя часть синусоиды (под осью абсцисс) строго симметрична верхней, то можно даже не брать интегралов, чтобы сообразить, что среднее значение синусоидального напряжения, показанного на рис. 2.2, в точности равно нулю (положительная

часть компенсирует отрицательную). Но такая величина малоинформативна, поэтому чаще используют средневыпрямленное *(среднеамплитудное)* значение, при котором знаки не учитываются (т. е. в интеграл подставляется абсолютная величина напряжения). Эта величина ($U_в$) связана с амплитудным значением ($U_а$) по формуле $U_а = \pi \cdot U_в/2$, т. е. $U_а \approx 1{,}57 \cdot U_в$.

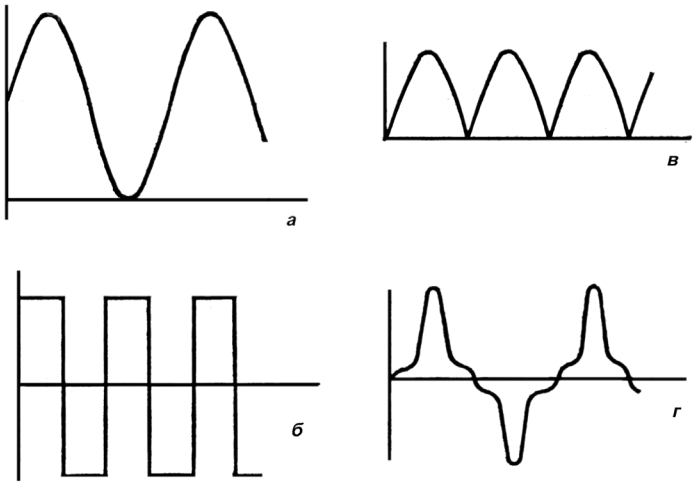


Рис. 2.5. Графики некоторых колебаний несинусоидальной формы

Кстати, для постоянного напряжения и тока действующее, среднее и среднеамплитудное значения совпадают и равны просто величине напряжения (тока). Однако на практике часто встречаются переменные колебания, форма которых отличается и от постоянной величины, и от строго синусоидальной. Осциллограммы некоторых из них показаны на рис. 2.5. Для таких сигналов приведенные соотношения для действующего и среднего значения недействительны! Самый простой случай изображен на рис. 2.5, *а*, где колебание представляет собой синусоиду, но сдвинутую вверх на величину амплитуды. Такой сигнал можно представить, как сумму постоянного напряжения величиной A (постоянная составляющая) и переменного синусоидального (переменная составляющая). Соответственно, среднее значение его будет равно A, а действующее $A + A / \sqrt{2}$. Для прямоугольного колебания (рис. 2.5, *б*) с равными по длительности положительными и отрицательными полуволнами (меандра) соотношения очень просты: действующее = среднеамплитудному = амплитудному, как и для постоянного тока, а вот среднее арифметическое значение равно, как и для синуса, нулю. Для случая рис. 2.5, *в*, который представляет собой синусоидальное напряжение, пропущенное через двухполупериодный

выпрямитель (см. *главу 4*), действующее и среднеамплитудное значения будут равны соответствующим значениям для синусоиды, а вот среднее будет равно не нулю, а совпадать со среднеамплитудным. Для последнего случая (рис. 2.5, *г*) указать все эти величины вообще непросто, т. к. они зависят от формы сигнала.

Но, даже выучив все это, вы все равно не сможете измерять величины напряжений и токов несинусоидальной формы с помощью мультиметра! Не забывайте об этом, как и о том, что для каждого мультиметра есть предельные значения частоты колебаний. Если вы включите мультиметр в цепь с иными параметрами, он может показать все, что угодно — «погоду на Марсе», по распространенному выражению. Измерительные приборы для переменного напряжения проградуированы в значениях действующего напряжения, но измеряют они, как правило, среднеамплитудное (по крайней мере, большинство, на подробностях мы не будем сейчас задерживаться), и сообразить, как именно пересчитать показания, далеко не всегда возможно. А для сигналов, как на рис. 2.5, *г*, это выливается в сущую головоломку на уровне задач для студентов мехмата.

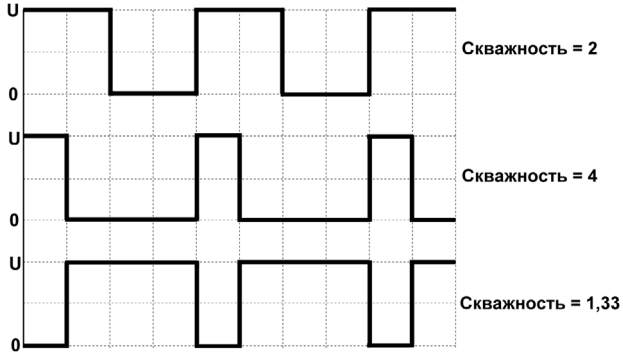


Рис. 2.6. Прямоугольные колебания с различной скважностью

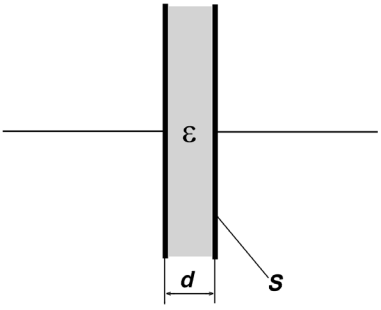
Для прямоугольных напряжений, представляющих собой меандр[1], подобный рис. 2.5, *б*, существует еще одна важная характеристика. Никто ведь не запрещает представить себе прямоугольное напряжение, в котором впадины короче или длиннее всплесков. В электронике термин «меандр» без дополнительных пояснений обычно означает именно симметричную форму прямоугольного напряжения, при которой впадины строго равны всплескам

[1] Меандр — тип геометрического узора с повторяющимися ломаными линиями (по названию извилистой реки Меандр в Малой Азии).

по длительности. Но, вообще говоря, это необязательно — на рис. 2.6 приведены два примера таких напряжений в сравнении с симметричным меандром. Параметр, характеризующий соотношение между длительностями частей периода, называется *скважностью*, и определяется, как отношение длительности всего периода к длительности его положительной части — именно так, а не наоборот, т. е. *величина скважности всегда больше единицы*. Для меандра скважность равна 2, для узких коротких импульсов она будет больше 2, для широких — меньше.

Конденсаторы

Все конденсаторы ведут свою родословную от лейденской банки, названной так по имени голландского города Лейдена, в котором трудился ученый середины XVIII века Питер ван Мушенбрук. Банка эта представляла собой большой стеклянный стакан, обклеенный изнутри и снаружи станиолем (тонкой оловянной фольгой, использовавшейся тогда для тех же целей, что и современная алюминиевая — металл алюминий еще не был известен). Так как банку заряжали от электростатической машины (другого источника электричества еще не придумали), которая запросто может выдавать напряжения в несколько сотен тысяч вольт, действие ее было весьма впечатляющим — в учебниках физики любят приводить случай, когда Мушенбрук продемонстрировал эффект от разряда своей банки через цепь гвардейцев, держащихся за руки. Ну не знали тогда, что электричество может и убить — гвардейцам сильно повезло, что емкость этого примитивного конденсатора была весьма невелика и запасенной энергии хватало только на то, чтобы люди почувствовали чувствительный удар током!



$$C = \frac{\varepsilon S}{4\pi d}$$

Рис. 2.7. Схематичное изображение плоского конденсатора и формула для расчета его емкости: C — емкость, Ф, S — площадь пластин, м^2, d — расстояние между пластинами, м, ε — диэлектрическая проницаемость

Схематичное изображение простейшего конденсатора показано на рис. 2.7. Из формулы, приведенной на рисунке (она носит специальное название «формула плоского конденсатора», потому что для конденсаторов иной геометрии соответствующее выражение будет другим), следует, что емкость тем больше, чем больше площадь пластин и чем меньше расстояние между ними.

Что же такое емкость? Согласно определению, *емкость есть отношение заряда (в кулонах) к разности потенциалов на пластинах (в вольтах)*: $C = Q/U$, т. е. размерность емкости есть кулон/вольт. Такая единица называется *фарадой*, по имени знаменитого английского физика и химика Майкла Фарадея (1791—1867). Следует подчеркнуть, что величина емкости есть независимая характеристика данного конденсатора — подобно тому, как номинальное сопротивление есть индивидуальная характеристика конкретного резистора — и характеризует количество энергии, которое может быть в нем запасено. Емкость в одну фараду весьма велика, обычно на практике прибегают к микрофарадам и еще более мелким единицам, скажем, емкость упомянутой лейденской банки составляла величину всего-навсего порядка 1 нФ.

Смысл понятия емкости раскрывается так: если напряжение от источника напряжения составляет 1 В, то емкость в одну нанофараду, как у лейденской банки, может запасти 10^{-9} кулон электричества. Если напряжение составит 10^5 вольт (типичная величина при заряде от электростатической машины, как в опытах Мушенбрука), то и запасенный на данной емкости заряд увеличится в той же степени — до 10^{-4} кулон. Любой конденсатор фиксированной емкости сохраняет это соотношение: заряд на нем тем больше, чем больше напряжение, а коэффициент пропорциональности в этой зависимости определяется номинальной емкостью.

Если замкнуть конденсатор на резистор, то в первый момент времени он будет работать, как источник напряжения с нулевым выходным сопротивлением и номинальным напряжением той величины, до которой конденсатор был заряжен. Таким образом ток через резистор в начальный момент времени определяется по обычному закону Ома. Скажем, в случае гвардейцев Мушенбрука характерное сопротивление цепи из нескольких человек, взявшихся за руки, составляет порядка 10^4 Ом, т. е. ток при начальном напряжении на конденсаторе 10^5 В составит 10 А, что примерно в 1000 раз превышает смертельное для человека значение тока! Выручило гвардейцев то, что такой импульс был крайне кратковременным, поскольку по мере разряда конденсатора, т. е. утекания заряда с пластин, напряжение быстро снижается (емкость-то остается неизменной, потому при снижении заряда, согласно формуле на рис. 2.7, падает и напряжение).

Заметки на полях

Интересно, что при фиксированном заряде (если цепь нагрузки конденсатора отсутствует) можно изменить напряжение на нем, меняя емкость — например, при раздвижении пластин плоского конденсатора емкость его падает (т. к. расстояние *d* между пластинами увеличивается), потому для сохранения заряда, согласно сказанному, напряжение должно увеличиться — что и происходит на деле (в эффектном школьном опыте между раздвигаемыми пластинами конденсатора проскакивает искра, когда напряжение превышает предельно допустимое напряжение пробоя для воздуха).

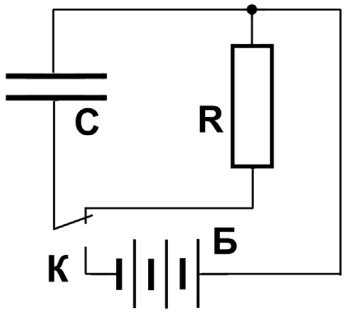


Рис. 2.8. Подключение конденсатора к нагрузке:
К — переключатель, Б — батарея, С — конденсатор,
R — сопротивление нагрузки

На рис. 2.8 изображено подключение конденсатора С к нагрузке R. Первоначально переключатель К ставят в нижнее по схеме положение и конденсатор заряжается до напряжения батареи Б. При переводе переключателя в верхнее положение конденсатор начинает разряжаться через сопротивление R и напряжение на нем снижается. Насколько быстро происходит падение напряжения при подключении нагрузки? Можно предположить, что чем больше емкость конденсатора и сопротивление резистора нагрузки, тем медленнее происходит падение напряжения. Правда ли это?

Это легко оценить через размерности связанных между собой электрических величин — тока, емкости и напряжения. В самом деле, в определение тока входит и время (напомним, что ток есть заряд, протекающий за единицу времени), которое нас и интересует. Если вспомнить, что размерность емкости есть кулон на вольт, то искомое время можно попробовать описать формулой: $t = CU/I$, где C — емкость, а U и I — ток и напряжение соответственно (проверьте размерность!).

Для случая, изображенного на рис. 2.8, эта формула справедлива на малых отрезках времени, пока ток I не падает значительно из-за уменьшения напря-

жения на нагрузке. Отметим, что данная формула полностью справедлива и на больших отрезках времени, если ток разряда (или заряда) конденсатора стабилизировать, что означает подключение его к источнику втекающего (при разряде) или вытекающего (при заряде) тока.

При фиксированной обычной нагрузке с сопротивлением *R* (помните, мы говорили, что простой резистор есть плохой источник тока?) так, конечно, не происходит — напряжение на конденсаторе падает по мере истощения заряда, соответственно ток через нагрузку также пропорционально снижается — в полном соответствии с законом Ома. Опять приходится брать интегралы, потому мы приведем только конечный результат: формула для расчета процесса снижения напряжения на емкости при разряде ее через резистор и соответствующий график показаны на рис. 2.9, *а*. На рис. 2.9, *б* показан аналогичный процесс, который происходит при заряде емкости через резистор.

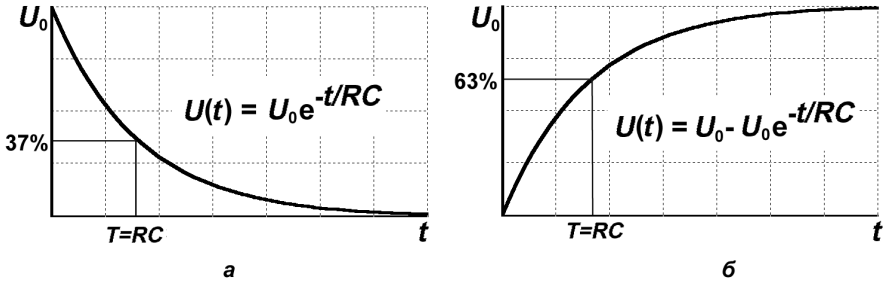


Рис. 2.9. Процессы при разряде (*а*) и заряде (*б*) конденсатора:
C — емкость, *R* — сопротивление нагрузки, *t* — время,
e — основание натуральных алгоритмов (2,718282)

Нужно отметить два момента. Во-первых, если сопротивление резистора R на рис. 2.8 (если включить в него как сопротивление проводов и ключа, так и — при заряде — внутреннее сопротивление батареи) не равно нулю, то получается, что процессы разряда и заряда по рис. 2.9 длятся бесконечно? Да, теоретически полностью конденсатор не разрядится и не зарядится никогда, но практически это почти не имеет значения, потому что напряжение на конденсаторе становится близким к нулю или к напряжению питания очень быстро.

Во-вторых, из формул на рис. 2.9 следует очень интересный вывод: если сопротивление R равно нулю, то время процесса разряда или заряда становится бесконечно малым, а ток через нагрузку, согласно закону Ома, бесконечно большим! Обратимся снова к рис. 2.8, нечто подобное должно происходить при переключении ключа К в положение заряда емкости от батареи. Естественно, в реальной жизни ни о каких бесконечных токах речи не идет, для этого

батарея должна иметь нулевое выходное сопротивление, т. е. бесконечно большую мощность (подумайте, почему эти утверждения равносильны?), а проводники должны обладать нулевым сопротивлением. Поэтому на практике процесс заряда от источника (и разряда при коротком замыкании пластин) происходит за малое, но конечное время, а ток, хоть и не бесконечно велик, но все же может достигать очень больших значений.

Значение тока в первый момент при включении конденсатора в цепь очень важно для практики. Например, под него надо рассчитывать кратковременную перегрузочную способность источника питания — иначе вы ничего не сможете включить в такой источник, потому что в первое же мгновение сработает защита, несмотря на то, что номинально мощности должно хватать. Как рассчитать этот ток? Для этого нужно представить, что *конденсатор при заряде в первый момент времени ведет себя так, как будто цепь в месте его установки замкнута накоротко* (это очень точное представление!). Тогда ток определится просто по закону Ома, в который подставляется сопротивление проводов и контактов (плюс, если это требуется, внутреннее сопротивление источника).

Интуитивно кажется, что должна существовать какая-то объективная характеристика цепи из конденсатора и сопротивления, которая позволяла бы описать процесс заряда-разряда во времени — независимо от напряжения на конденсаторе. Такая характеристика рассчитывается по формуле $T = RC$. Приведением единиц мы бы здесь занимались довольно долго, потому поверьте, что размерность произведения RC есть именно время в секундах. Эта величина называется *постоянной времени RC-цепи* и физически означает время, за которое напряжение на конденсаторе при разряде его через резистор (рис. 2.9, *а*) снижается на величину 0,63 от начального (т. е. до величины, равной доле $1/e$ от первоначального U_0, что и составляет примерно 37%). За следующий отрезок времени, равный RC, напряжение снизится еще на столько же от оставшегося и т. п. — в полном соответствии с законом экспоненты. Аналогично при заряде конденсатора (рис. 2.9, *б*), постоянная времени T означает время, за которое напряжение увеличится до доли $(1 - 1/e)$ до конечного значения U_0, т. е. до 63%. Произведение RC играет важную роль при расчетах различных схем.

Есть еще одно обстоятельство, которое следует из формулы для плоского конденсатора (см. рис. 2.7). В самом деле, там нет никаких ограничений на величины S и d — даже если развести пластины очень далеко, все же какую-то емкость, хотя и небольшую, конденсатор будет иметь. То же происходит при уменьшении площади пластин. Практически это означает, что небольшую емкость между собой имеют любые два проводника, независимо от их конфигурации и размеров, хотя эти емкости могут быть и исчезающе малы.

Этот факт имеет огромное значение на высоких частотах — в радиочастотной технике нередко конденсаторы образуют прямо из дорожек на печатной плате. А емкости между параллельными проводами в обычном проводе-«лапше» или кабеле из-за их большой длины могут оказаться значительными.

Если же учесть, что проводники имеют еще и собственное сопротивление, то мы приходим к выводу, что любую пару проводов можно представить в виде «размазанной» по длине (распределенной) *RC*-цепи — и это действительно так, со всеми вытекающими последствиями! Например, если подать на вход пары проводников в длинном кабеле перепад напряжения (фронт прямоугольного импульса), то на выходе мы получим картину, которая ничем не отличается от рис. 2.9, *б* — импульс «размажется», а если он короткий, то вообще может пропасть.

Заметки на полях

Впервые с эффектом распределенной емкости столкнулись еще при попытке прокладки первого трансатлантического кабеля в 1857 году — телеграфные сигналы (точки-тире) представляют собой именно такие прямоугольные импульсы, и при длине кабеля в 4000 км они по дороге искажались до неузнаваемости. За время до следующей попытки прокладки кабеля (1865) английскому физику У. Томпсону пришлось разработать теорию передачи сигналов по длинным линиям, за что он получил рыцарство от королевы Виктории и вошел в историю под именем лорда Кельвина, по названию городка Кельвин на западном побережье Ирландии, откуда начиналась прокладка кабеля.

В выражении для емкости на рис. 2.7 фигурирует постоянная ε, представляющая собой диэлектрическую проницаемость среды. Для воздуха и большинства обычных изолирующих материалов (полиэтилена, хлорвинила, лавсана, фторопласта) константа ε близка к величине ее для полного вакуума (ε_0). Значение ε_0 зависит от применяемой системы единиц измерения, и в международной системе единиц измерения СИ равна $8{,}854 \cdot 10^{-12}$ Ф/м. На практике удобно применять относительную диэлектрическую проницаемость конкретного материала: $\varepsilon_r = \varepsilon/\varepsilon_0$. Естественно, что для практических нужд производителей конденсаторов желательно, чтобы величина ε_r была как можно выше. Если вы заполните промежуток между пластинами, скажем, ацетоном или спиртом, то емкость такого конденсатора сразу возрастет раз в двадцать! К сожалению, чем выше ε_r, тем обычно больше и собственная проводимость материала, потому такой конденсатор быстро разрядится за счет собственных токов утечки через среду между пластинами.

Ясно, что производители конденсаторов стараются упаковать как можно бóльшую емкость в как можно меньшие размеры, пытаясь одновременно обеспечить токи утечки на приемлемом уровне. По этой причине количество практически используемых типов конденсаторов значительно больше, чем

сопротивлений — для каждого применения свой тип. Самым высоким соотношением емкость/габариты обладают электролитические (оксидные) конденсаторы, которые в настоящее время широко представлены серией, известной под отечественным наименованием К50-35 (импортные конденсаторы такого же типа обычно все равно продают под этим названием). Емкости их достигают 100 000 мкФ, а допустимые напряжения — до 600 В, но хорошо работают они только на низких частотах. Включаться «электролиты» должны только в определенной полярности. Эти конденсаторы обычно служат в качестве фильтров в источниках питания, хотя и иные применения не исключены.

Параллельное и последовательное включение конденсаторов

Как и резисторы, конденсаторы могут включаться последовательно или параллельно, однако расчет полученных величин производится противоположно правилам для резисторов: при параллельном соединении емкости складываются (по правилу «больше большего»), а при последовательном соединении складываются их обратные величины (правило «меньше меньшего»). К счастью, в отличие от резисторов, конденсаторы включают практически только параллельно — можно это представить так, как будто площади их пластин при этом складываются, следовательно, складываются и емкости. Последовательное же соединение емкостей само по себе не имеет практического смысла, и знание правил сложения для него необходимо лишь изредка при анализе цепей переменного тока.

Конденсаторы в цепи переменного тока

В дальнейшем мы будем иметь дело в основном с цепями постоянного тока или низкой частоты. Слова «низкой частоты» в предыдущей фразе нужно понимать условно, и вот почему: любой перепад напряжения (например, при включении или выключении питания) есть импульс высокой частоты, и тем она выше, чем быстрее происходит сам процесс снижения или повышения напряжения. Любое колебание, согласно теореме Фурье, великого французского математика, работавшего еще в конце XVIII века, можно представить как сумму гармонических (т. е. синусоидальных) колебаний. Возможен и обратный процесс — воспроизведение изначальной формы колебания через известную сумму гармоник. Если импульс строго прямоугольный, то самая высокая частота в такой сумме должна быть равна бесконечности, чего на деле, конечно, не бывает, поэтому реальные импульсы всегда не строго

прямоугольные. Прохождение прямоугольных импульсов через конденсаторы и резисторы мы разберем далее, а пока рассмотрим поведение конденсаторов в цепях с обычным синусоидальным переменным током.

Постоянный ток конденсатор не пропускает по определению, т. к. представляет собой разрыв в цепи. Однако переменный ток через него протекает, при этом происходит постоянный перезаряд конденсатора, поскольку напряжение все время изменяется по величине и полярности. Поэтому конденсатор в цепи переменного тока можно представить себе, как некий резистор: чем меньше емкость конденсатора и чем ниже частота, тем выше величина его сопротивления. Ее можно подсчитать по формуле $R = 1/2\pi fC$ (если емкость C выражена в фарадах, а частота f в герцах — сопротивление получится в омах). В пределе конденсатор очень малой емкости (что представляют собой, как мы уже выяснили, почти все пары проводников на свете) будет выглядеть, как полный разрыв в цепи и ток в этой цепи будет исчезающе мал.

Сам по себе конденсатор в такой цепи энергии не потребляет (в отличие от обычного резистора), поэтому его сопротивление переменному току называют *реактивным* — в то время как обычное резистивное сопротивление называют *активным*.

Замечание

Комплексную сумму активного и реактивного сопротивлений цепи иногда называют ее *импедансом* — это понятие эквивалентно обычному сопротивлению (и измеряется в омах), но используется при анализе высокочастотных схем.

Понять, почему так происходит, можно, если представить себе графики тока и напряжения в цепи с конденсатором — ток опережает напряжение по фазе ровно на 90°, поэтому их произведение, которое есть потребляемая мощность по закону Джоуля-Ленца, в среднем равно нулю — можете проверить! Однако если в цепи имеются еще и обычные резисторы (а, как мы знаем, они всегда присутствуют — взять хотя бы сопротивление проводов), то этот реактивный ток приведет ко вполне материальным потерям на их нагревание — именно поэтому линии электропередач выгоднее делать на постоянном токе.

Подробности

Кроме конденсаторов, реактивным сопротивлением обладают также *индуктивности* (в простейшем случае это катушка с проводом), только они по всему противоположны конденсаторам: ток в цепи, содержащей индуктивность, *отстает* от напряжения на 90°. Если конденсатор для постоянного тока представляет собой разрыв цепи, то индуктивность, наоборот — нулевое сопротивление, а с ростом частоты переменного тока реактивное сопротивление индуктивности растет. Индуктивности очень не «любят» в электронике, т. к. реальные изделия всегда далеки от идеальной индуктивности, имеют большие

габариты и с трудом поддаются автоматизации производства. В микроэлектронике их стараются избегать, за исключением трансформаторов и фильтров в источниках питания (см. *главу 4*), где применяют готовые *дроссели* (внешне очень похожие на резисторы), или намотанные вручную на ферритовые кольца. Измеряется индуктивность в *генри* (Гн).

При наличии реактивной нагрузки в цепи переменного тока полезная мощность (в нагрузке) может отличаться от величины произведения потребляемого тока на напряжение — она всегда меньше. Поэтому иногда различают мощность, выраженную в вольт-амперах (ВА), и мощность в ваттах (Вт), а отношение их называют коэффициентом мощности. Другое его общепринятое название — «косинус фи», потому что коэффициент мощности есть не что иное, как cos(φ), где φ — угол фазового сдвига тока относительно напряжения. При постоянном токе, а также в случае чисто активной нагрузки угол этот равен нулю, поэтому косинус равен единице. В другом предельном случае, когда нагрузка чисто реактивная, косинус равен нулю. В реальных цепях с электродвигателями или, скажем, мощными вторичными импульсными источниками питания (офис с большим количеством компьютеров) в качестве потребителей, коэффициент мощности может лежать в пределах 0,6—0,9. Следует подчеркнуть, что коэффициент мощности — это не КПД, как можно себе вообразить. Разница между вольт-амперами и ваттами никуда не теряется в физическом смысле, она всего лишь приводит к таким неприятным последствиям, как увеличение потерь в проводах, о котором мы упоминали (оно пропорционально именно вольт-амперам), а также возникновению разбаланса между фазами трехфазной промышленной сети, в результате чего через нулевой, обычно более тонкий, чем все остальные, провод начинают протекать значительные токи.

Дифференцирующие и интегрирующие цепи

Если подать на вход цепи, состоящей из резистора R и конденсатора C, прямоугольный импульс напряжения, то результат будет различным в зависимости от включения R и C. Переходные процессы в таких цепях подчиняются основным закономерностям, представленным на рис. 2.9, но имеют и свою специфику. На рис. 2.10 показаны два способа включения RC-цепочки в цепь с прямоугольными импульсами на входе (здесь они не такие, как на рис. 2.5, *б*, а однополярные, т. е. напряжение меняется по величине, но остается выше уровня «земли»). Такое включение называется *дифференцирующей цепочкой или фильтром высоких частот* (ФНЧ), потому что данная цепь пропускает высокочастотные составляющие, полностью отрезая постоянный ток. Чем больше постоянная времени *RC* в этой схеме, тем ниже частота, которая может быть пропущена без изменений, — в пределе импульсы пройдут

почти неизмененными. Наоборот, если постоянную времени уменьшать, то пики на графике будут все больше утончаться. Этим эффектом часто пользуются для выделения фронтов и спадов прямоугольных импульсов.

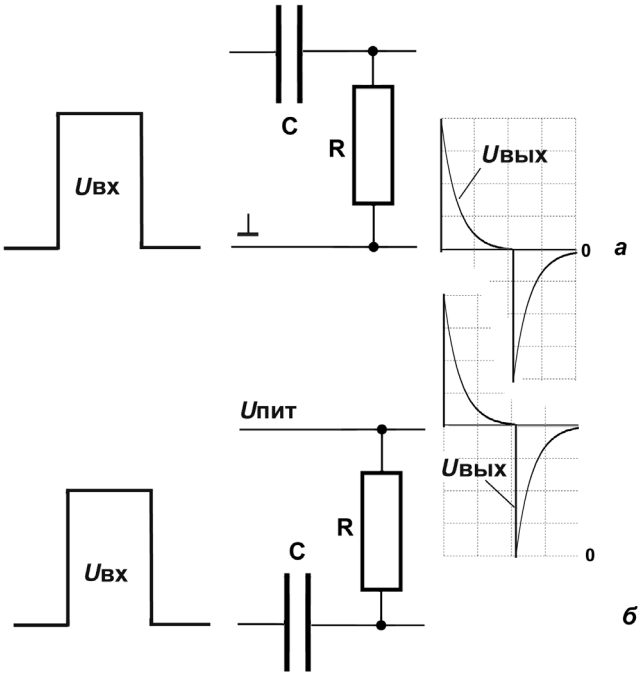


Рис. 2.10. Дифференцирующие цепочки:
а — при подключении резистора к нулевому потенциалу;
б — к потенциалу источника питания

Так как через конденсатор постоянная составляющая напряжения не проходит, то полученные импульсы «привязаны» к выходному потенциалу схемы — в зависимости от того, куда подключен резистор. На графиках рис. 2.10 резистор подключен либо к «земле» (*а*), либо к источнику питания (*б*), потому и для выходного напряжения базовым будет либо нулевой потенциал, либо потенциал источника (при этом амплитуда импульсов будет такой, как у входного напряжения). Этим широко пользуются при необходимости умножения напряжения (обратите внимание, что на рис. 2.10, *б* амплитуда положительного выходного импульса в два раза выше напряжения питания), или для формирования двуполярного напряжения из имеющегося однополярного. Иногда этот эффект вреден: подачей отрицательного или превышающего потенциал источника питания напряжения можно вывести из строя компоненты схемы.

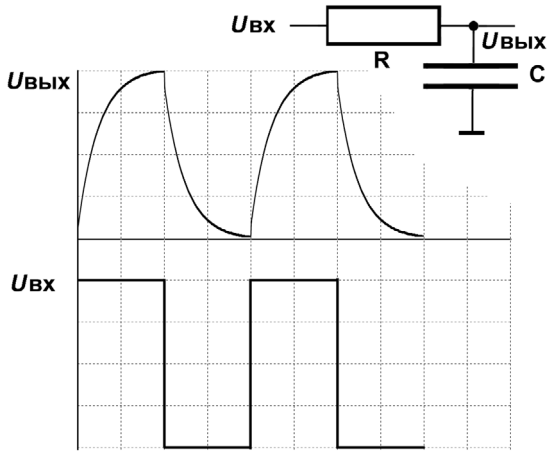


Рис. 2.11. Интегрирующая цепочка и график ее выходного напряжения, построенный в одном масштабе с входным

А *интегрирующая цепочка* (*фильтр нижних частот*, ФВЧ) получается из схем рис. 2.10, если в них R и C поменять местами. График выходного напряжения будет соответствовать показанному на рис. 2.11. Такие цепочки, наоборот, пропускают постоянную составляющую, в то время как высокие частоты будут отрезаться. Если в такой цепочке увеличивать постоянную времени *RC*, то график будет становиться все более плоским — в пределе пройдет только постоянная составляющая (которая здесь равна среднему значению исходного напряжения, т. е. ровно половине его амплитуды). Этим широко пользуются при конструировании вторичных источников питания, в которых нужно отфильтровать переменную составляющую сетевого напряжения. Интегрирующими свойствами обладает также обычный кабель из пары проводов, о котором мы упоминали ранее, потому-то и теряются высокие частоты при прохождении сигнала через него.

Сигналы

Электрический сигнал, как следует из названия, — это какое-то состояние электрической цепи, которое несет информацию. Различают источники сигналов и их приемники. Так как минимальное количество информации (1 бит) подразумевает по крайней мере два различимых состояния (подробнее об этом будет идти речь в *главе 7*), то и сигнал должен иметь как минимум два состояния. Самый простой сигнал — наличие или отсутствие постоянного напряжения в цепи, именно такими сигналами обмениваются логические

микросхемы. Однако на большое расстояние такой простейший сигнал не передашь, т. к. слишком сложно защититься от помех. Из-за них приемник легко может обнаружить наличие сигнала там, где на самом деле всего лишь помеха. Поэтому придумывают разные сложные методы, некоторые из них, например, предусматривают передачу переменного напряжения разной частоты или фазы (именно так устроены модемы).

Теория передачи сигналов тесно связана с теорией колебаний — одно только радио чего стоит! Подробнее о разных сигналах мы будем говорить в соответствующих главах, а сейчас нам важно только одно: когда мы говорим о сигналах, то подразумеваем, что соответствующее напряжение или ток не предназначено для совершения иной работы, кроме как заставить сработать приемник. Поэтому соответствующие передаваемые мощности здесь значительно меньше, чем при передаче электроэнергии для совершения полезной работы. Действительно, никто еще не придумал, как питать, скажем, спутники на орбите по радиолучу, а вот информацию передают вполне успешно даже за пределы Солнечной системы. В этом заключается основная разница между силовыми и сигнальными цепями. И понимание этого тонкого различия очень пригодится нам в дальнейшем.

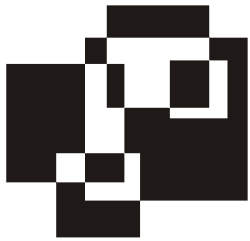
Переменный ток, как основа цивилизации

Кстати, отдельный вопрос — а почему нам вообще надо возиться с переменным током, как основой электропитания? Сколько можно было бы сэкономить на трансформаторах и сглаживающих конденсаторах, которые зачастую составляют бóльшую часть габаритов и стоимости схемы! Недаром схемотехники и дизайнеры в последнее время полюбили выносные блоки питания, встроенные в сетевую вилку — крайне некрасивое решение, которое просто переносит головную боль о габаритах с плеч разработчиков на плечи потребителей, зато позволяет не думать о выпрямителях, прочности изоляции, сертификатах электробезопасности и прочих трудностях преобразования силового переменного тока в постоянный.

Дело в том, что никаких других эффективных первичных генераторов электроэнергии (тех, что преобразуют энергию вращения ротора водяной или паровой турбины в электричество на электростанциях), кроме как работающих на переменном токе, не придумали. Интересно, что по причинам, указанным ранее в этой главе, многие линии электропередач в мире делают на постоянном (выпрямленном, т. е. пульсирующем) токе. Это позволяет во многом избежать реактивных потерь в проводах, но все же приходится сначала преоб-

разовывать переменный ток в постоянный, а затем выполнять обратное преобразование (которое куда сложнее) исключительно для того, чтобы состыковать имеющиеся линии электропитания со стандартными.

Аналогичная задача, только в меньших масштабах, стоит перед разработчиками источников бесперебойного питания (известных еще под английской аббревиатурой UPS). Питающий ток из сети нужно преобразовать в постоянный для зарядки низковольтного (12 или 24 В) резервного аккумулятора, а в случае пропадания сетевого питания это напряжение аккумулятора следует опять преобразовать к стандартному виду переменного сетевого напряжения (такое преобразование называется *инверсией*), причем желательно, чтобы форма его была максимально близка к синусоидальной. Приходится поломать голову, чтобы компьютер, питающийся через UPS, не заметил такого перехода!



Глава 3

Основные дискретные компоненты

> Полный список товаров занял бы несколько страниц, поэтому я приведу лишь некоторые: сковородки, шляпы, ведерные кофейники, рыболовные снасти, журналы и книги в мягких обложках, оружие и амуниция, всевозможные продукты питания, пончо, шпоры и седла, сигары, сигареты и табак, охотничьи и кухонные ножи, ковбойские сапоги и резиновые болотники, мужская и женская одежда, джинсы, открытки, авторучки, три полки с лекарствами...
>
> *Рекс Стаут «Смерть чужака»*

О двух важнейших электронных компонентах, которые вы встретите в любой, самой что ни на есть «микроэлектронной» схеме, мы уже говорили в предыдущих главах — это резисторы и конденсаторы. Но кроме них, в современной технике используется также много других типов компонентов, которые получили общее наименование *дискретные*. Грубо говоря, дискретные компоненты — это все, что не микросхемы. Хотя такое деление и достаточно условно: например, какой-нибудь оптрон (устройство, совмещающее в себе пару «светодиод — фотодиод» для передачи сигнала по оптическому каналу) относят обычно к дискретным компонентам, однако по сути это микросхема, и достаточно сложная в изготовлении.

Давайте разберемся немного в важнейших разновидностях дискретных компонентов. Сейчас немодно проектировать схемы на «рассыпухе», в большинстве случаев это и не имеет смысла, поскольку на интегральных микросхемах получается быстрее, дешевле и надежнее. Однако, во-первых, без дискретных элементов все равно во многих случаях не обойтись (посмотрите, сколько их на материнской плате вашего ПК, а ведь эти платы обычно вбирают

в себя все самое современное), во-вторых, микроэлектронные схемы работают по тем же законам, что и старинные, на отдельных элементах. А в-третьих, в радиолюбительской и полупрофессиональной практике часто бывает так, что гораздо удобнее применить, например, транзисторный ключ с парой резисторов, чем гоняться по торговым организациям за соответствующей микросхемой, и потом еще мучаться, раскладывая плату под какой-нибудь планарный корпус с шагом 0,127 мм (тем более, что резисторы, скорее всего, так или иначе потребуются).

Из всех полупроводниковых устройств исторически первыми были диоды.

Диоды

Диод — это простейший полупроводниковый прибор с двумя выводами, характеризующийся тем, что в одну сторону он проводит ток (т. е. представляет собой в идеале просто проводник с малым сопротивлением), в другую — нет (т. е. превращается в очень большое сопротивление) — одним словом, обладает односторонней проводимостью. Выводы диода, как повелось еще со времен ламповой техники, называют анодом (положительный) и катодом (отрицательный). Не всегда понятно, что означают слова «положительный» и «отрицательный» в приложении к некоторым включениям диодов, потому конкретизируем: если подать на анод положительное напряжение, то диод будет проводить ток. В обратном включении ток не пройдет.

Если подключить диод к регулируемому источнику напряжения, то он будет вести себя так, как показано на рис. 3.1, где представлена т. н. *вольт-амперная характеристика* диода. Из нее, в частности, следует, что в прямом включении (т. е. анодом к плюсу источника), после превышения некоторого напряжения ($U_{пр}$), прямой ток через диод ($I_{пр}$) растет неограниченно и будет лимитироваться только мощностью источника. На самом деле без нагрузки диоды, за редкими исключениями, не включают, и тогда в прямом включении ток ограничивается нагрузкой.

В обратном же включении (катодом к плюсу) ток через диод ($I_{обр}$) пренебрежимо мал и составляет от нескольких микро- или даже наноампер для обычных маломощных диодов, до единиц миллиампер для мощных выпрямительных. Причем для германиевых диодов обратный ток намного выше, чем для кремниевых, отчего их сейчас практически и не употребляют. Этот ток сильно зависит от температуры и может возрасти на несколько порядков (от нано- до микроампер) при повышении температуры от –50 до +50 °С, поэтому на графике его величина показана очень приблизительно (обратите внимание, что верхняя и нижняя половины графика по оси токов построены в разных масштабах).

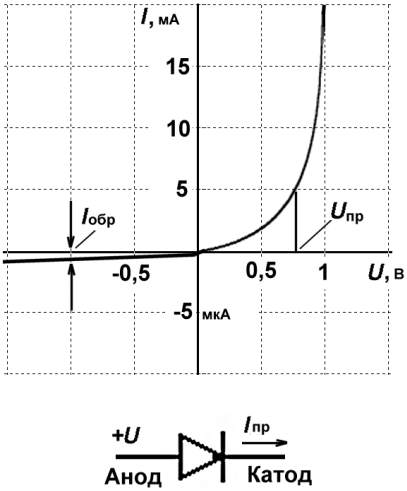


Рис. 3.1. Вольт-амперная характеристика диода

В отличие от обратного тока, прямое падение напряжения $U_{пр}$ гораздо меньше зависит как от типа и конструкции прибора, так и от температуры. *Для кремниевых диодов прямое падение напряжения $U_{пр}$ всегда можно считать равным примерно 0,6—0,7 В*, для германиевых или так называемых диодов Шоттки эта величина составляет 0,2—0,4 В. Для кремниевых диодов при изменении температуры на один градус $U_{пр}$ изменяется примерно на 2,3 мВ.

Если умножить указанное прямое падение напряжения на проходящий через диод в прямом включении ток, то мы получим тепловую мощность, которая выделяется на диоде. Именно она приводит диоды к выходу из строя — при превышении допустимого тока они просто сгорают. Впрочем, тепловые процессы инерционны, и в справочниках указывается обычно среднее значение допустимого тока, а мгновенное значение тока, в зависимости от длительности импульса, может превышать предельно допустимое в сотни раз! Обычное значение среднего предельно допустимого тока через маломощные диоды — десятки и сотни миллиампер. Мощные диоды (при токах 3—5 А и выше) часто приходится устанавливать на радиаторы.

Другая характеристика диодов — предельно допустимое обратное напряжение. Если оно превышено, то диоды также выходят из строя — электрически пробиваются и замыкаются накоротко. Обычная допустимая величина обратного напряжения для маломощных диодов — десятки вольт, для выпрямительных — сотни вольт, но есть диоды, которые выдерживают и десятки тысяч вольт. Далее мы увидим, что существуют приборы, для которых пробой

в обратном включении является рабочим режимом, — они называются стабилитронами.

Подробности

Физически диод состоит из небольшого кристаллика полупроводникового материала, в котором в процессе производства формируются две зоны с разными проводимостями, называемыми проводимостью *n*- и *p*-типа. *Ток всегда течет от p-зоны к n-зоне* (это стоит запомнить), в обратном направлении диод заперт. Более подробные сведения о физике процессов, происходящих в *p-n*-переходе, излагаются во множестве пособий, включая школьные учебники, но для практической деятельности почти не требуются.

Транзисторы

Транзистор — это электронный полупроводниковый прибор, предназначенный для усиления сигналов. Первым таким прибором в истории была электронная лампа (а еще до нее, кстати — электромагнитные реле, которые мы кратко рассмотрим далее). Лампа сумела сделать немало — именно в «ламповую» эпоху возникли радио и телевидение, компьютеры и звукозапись. Но только транзистор и появившиеся на его основе микросхемы сумели действительно перевернуть мир так, что электронные устройства вошли в наш повседневный быт и мы теперь уже не мыслим себя без них.

Транзисторы делятся на *биполярные* и *полевые* (или *униполярные*). Пока мы будем говорить только о биполярных транзисторах.

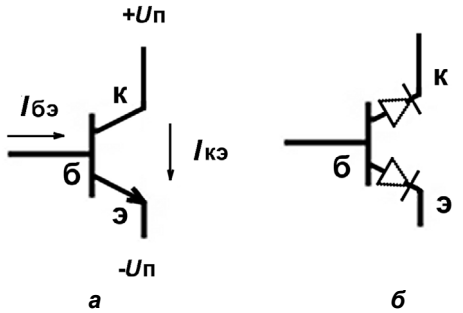


Рис. 3.2. Биполярный транзистор: *а* — рабочие полярности напряжений и направления токов в *n-p-n*-транзисторе (к — коллектор, б — база, э — эмиттер); *б* — условное представление транзистора, как состоящего из двух диодов

Физически биполярный транзистор — это структура из трех слоев полупроводника, разделенных двумя *p-n*-переходами. Поэтому можно себе представить, что он состоит как бы из двух диодов, один из слоев у которых общий,

и это весьма близко к действительности! Скомбинировать два диода можно, сложив их либо анодами, либо катодами, соответственно, различают *n-p-n-* и *p-n-p*-транзисторы, которые отличаются только полярностями соответствующих напряжений. Заменить *n-p-n*-прибор на аналогичный *p-n-p* можно, просто поменяв знаки напряжений во всей схеме на противоположные (и все полярные компоненты — диоды, электролитические конденсаторы — естественно, тоже надо перевернуть). Транзисторов *n-p-n*-типов выпускается гораздо больше, и употребляются они чаще, поэтому мы пока что будем вести речь исключительно о них, но помнить, что все сказанное справедливо и для *p-n-p*-структур, с учетом обратной их полярности. Правильные полярности и направления токов для *n-p-n*-транзистора показаны на рис. 3.2.

Рис. 3.3. Первый в истории транзистор (Фото Lucent Technologies Inc./Bell Labs)

Первый в истории транзистор был построен в знаменитых Лабораториях Белла (Bell Labs) Дж. Бардиным и У. Браттайном по идеям Уильяма Брэдфорда Шокли в 1947 году. В 1956 году все трое были удостоены Нобелевской премии. Кроме изобретения транзистора, У. Шокли известен также, как один из основателей знаменитой Кремниевой долины — технополиса в Калифорнии, где сегодня расположено большинство инновационных полупроводниковых и компьютерных фирм. Из фирмы Шокли, под названием Shockley Semiconductor Labs, вышли, в частности, Гордон Мур и Роберт Нойс — будущие основатели крупнейшего ныне производителя микропроцессоров фирмы Intel. Г. Мур еще известен, как автор знаменитого «закона Мура», а Р. Нойс — как изобретатель микросхемы (совместно с Д. Килби — подробнее см. *главу 6*).

Три вывода биполярного транзистора носят названия *коллектор*, *эмиттер* и *база*. Как ясно из рис. 3.2, *б*, база присоединена к среднему из трех полупроводниковых слоев. Так как, согласно показанной на рисунке полярности, потенциал базы более положителен, чем у эмиттера, то соответствующий диод всегда открыт для протекания тока. Парой страниц ранее мы убедились, что в этом случае на нем должно создаваться падение напряжения в 0,6 В. Именно так и есть — *в рабочем режиме напряжение между эмиттером и базой всегда составляет приблизительно 0,6 В*, причем на базе выше, чем на эмиттере (еще раз напомним, что для *p-n-p*-транзисторов напряжения обратные, хотя абсолютные величины их те же). А вот диод между коллектором и базой заперт обратным напряжением. Как же может работать такая структура?

Практически это можно себе представить, как если бы ток, втекающий в базу, управлял неким условным резистором, расположенным между коллектором и эмиттером (пусть вас не смущает помещенный там диод «коллектор-база», через него-то ток все равно не потечет). Если тока базы нет, т. е. выводы базы и эмиттера закорочены (здесь, главное, чтобы $U_{бэ}$ было бы близко к нулю), тогда промежуток «эмиттер-коллектор» представляет собой очень высокое сопротивление, и ток через коллектор пренебрежимо мал (сравним с обратным током диода). В таком состоянии транзистор находится в режиме *отсечки* (говорят, что прибор заперт или закрыт).

В противоположном режиме ток базы велик ($U_{бэ}$= 0,6—0,7 В, как мы говорили ранее, при этом ток, естественно, ограничен специальным сопротивлением), тогда промежуток «эмиттер-коллектор» представляет собой очень малое сопротивление. Это режим *насыщения*, когда транзистор полностью открыт (естественно, в коллекторной цепи, как и в базовой, должна присутствовать какая-то нагрузка, иначе транзистор в этом режиме может просто сгореть). Остаточное напряжение на коллекторе транзистора может при этом составлять порядка 0,3 В. Эти два режима представляют часто встречающийся случай, когда транзистор используется в качестве *ключа* (или, как говорят, «работает в ключевом режиме»), т. е. как обычный выключатель тока.

Ключевой режим работы биполярного транзистора

А в чем смысл такого режима, спросите вы? Смысл очень большой — ток базы может управлять током коллектора, который как минимум на порядок больше, т. е. налицо усиление сигнала по току (за счет, естественно, энергии источника питания). Насколько велико может быть такое усиление? В режиме «ключа» почти для всех обычных типов современных транзисторов можно

смело полагать коэффициент усиления по току (т. е. отношение максимально возможного тока коллектора к минимально возможному току базы $I_к/I_б$) равным нескольким десяткам — не ошибетесь. Если ток базы и будет больше нужного — не страшно, он никуда не денется, открыться сильнее транзистор все равно не сможет. Коэффициент усиления по току в ключевом режиме еще называют «коэффициентом усиления по току в режиме большого сигнала» и обозначают буквой β. Есть особые «дарлингтоновские» транзисторы, для которых β может составлять до 1000 и более (обычно они составные, поэтому напряжение $U_{бэ}$ у них заметно больше обычного: 1,2—1,5 В).

Рассмотрим подробнее ключевой режим работы транзистора ввиду его важности для практики. На рис. 3.4 показана простейшая схема включения транзистора в таком режиме, для наглядности — с лампочкой в качестве коллекторной нагрузки.

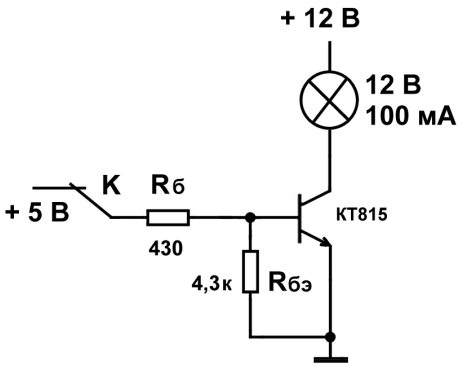


Рис. 3.4. Включение биполярного транзистора в ключевом режиме

Попробуем рассчитать необходимую величину резистора в базе. Как вы сейчас увидите, для транзисторных схем характерно, что напряжения в схеме никакой роли не играют, только токи: можно подключить коллекторную нагрузку хоть к напряжению 200 В, а базовый резистор питать от 5-вольтового источника, — если соотношение $\beta > I_к/I_б$ соблюдается, то транзистор (при условии, конечно, что он рассчитан на такое высокое напряжение) будет послушно переключать 200-вольтовую нагрузку, управляясь от источника 5 В. Таким образом, налицо усиление сигнала по напряжению!

В нашем примере выбрана небольшая автомобильная лампочка 12 В, 100 мА (примерно, как для подсветки приборной доски в «Жигулях»), а цепь базы питается от источника 5 В. Расчет элементарно прост: при 100 мА в коллекторе, в базе должно быть минимум 10 мА (не глядя в справочник, ориентиру-

емся на минимальное значение $\beta = 10$). Напряжение на базовом резисторе $R_б$ составит 5 В – 0,6 В = 4,4 В (о падении между базой и эмиттером забывать не следует), т. е. нужное сопротивление будет равно 440 Ом. Выбираем ближайшее меньшее из стандартного 5%-ного ряда и получаем 430 Ом. Все?

Нет, не все. Схема еще не совсем доделана. Она будет работать нормально, если вы будете поступать так: подключать базовый резистор к 5 В (лампочка горит), а затем *переключать его к «земле»* (лампочка гаснет). Но довольно часто встречается ситуация, когда напряжение на базовый резистор подается-то нормально, а вот при отключении его резистор не присоединяется к «земле», а просто «повисает в воздухе» (именно этот случай и показан на схеме в виде контактов выключателя К). Так мы не договаривались. Чтобы транзистор был в режиме отсечки, надо установить равные потенциалы базы и эмиттера, а какой потенциал будет у базы, если она «в воздухе»? Это только формально, что ноль, а на самом деле всякие наводки — электричества-то вокруг полно — и внутренние процессы в транзисторе формируют небольшой базовый ток. И транзистор не закроется полностью, лампочка будет слабо светиться!

Это очень неприятный эффект, который даже может привести к выходу транзистора из строя. Избежать его просто: следует замкнуть базу и эмиттер еще одним резистором $R_{бэ}$. Самое интересное, что рассчитывать его практически не нужно — лишь бы падение напряжения на нем при подаче напряжения на базу не составило меньше, чем 0,7 В. Его значение можно выбрать примерно в 10 раз больше, чем резистора $R_б$ (но если вы здесь поставите не 4,3 кОм, а, к примеру, 10 кОм, тоже не ошибетесь). Работать он будет так: если открывающее напряжение на $R_б$ подано, то он не оказывает никакого влияния на работу схемы, т. к. напряжение между базой и эмиттером все равно 0,6 В, и он только отбирает на себя очень небольшую часть базового тока (легко подсчитать, какую, поделив 0,6 на его значение 4,3 кОм, получится примерно 0,14 мА). А если напряжения нет, то $R_{бэ}$ обеспечивает надежное равенство потенциалов базы и эмиттера, независимо от того, подключен ли базовый резистор к «земле» или «висит в воздухе».

Я так подробно остановился на этом моменте потому, что о включении резистора $R_{бэ}$ при работе в ключевом режиме часто забывают. А ведь еще в 1950—60-х годах транзисторы по ТУ вообще запрещалось включать в режиме с «оборванной базой», т. к. первые промышленные типы их запросто могли выйти из строя!

Простейшая ключевая схема есть вариант т. н. *схемы с общим эмиттером* (ОЭ). Обратите внимание, что сигнал на коллекторе транзистора *инвертирован* (т. е. противоположен по фазе) по отношению ко входному сигналу.

Если на базе (точнее, на базовом резисторе) напряжение имеется — на коллекторе оно равно нулю, и наоборот! Это и имеют в виду, когда говорят, что *транзисторный каскад в схеме с общим эмиттером инвертирует сигнал* (справедливо не только для ключевого, но и для усилительного режима работы, о котором несколько слов далее). Сигнал при этом и на входе и на выходе должен измеряться относительно «земли». На нагрузке (лампочке), которая подключена к питанию, а не к общей для входа и выхода каскада «земле», все в порядке, т. е. она горит, когда на входе сигнал есть, «визуальный» сигнал не инвертирован.

Усилительный режим работы биполярного транзистора

Рассмотрим усилительный режим транзистора. В настоящее время его в реальных схемах воспроизводить почти не приходится, т. к. все современные усилители собирают из готовых микросхем, у которых все эти транзисторы находятся внутри. И все же понимание того, как они работают, никогда не помешает, да и транзисторы «россыпью» нередко еще приходится применять, поэтому мы рассмотрим работу различных усилительных каскадов довольно подробно.

Из написанного ранее ясно, что между режимами насыщения и отсечки должно существовать какое-то промежуточное состояние, например, когда лампочка на рис. 3.4 горит вполнакала. Действительно, в некотором диапазоне базовых токов (и соответствующих им напряжений, подающихся на базовый резистор) ток коллектора (и соответствующее ему напряжение на коллекторе) будет плавно меняться. Соотношение между токами здесь будет определяться величиной коэффициента усиления по току *для малого сигнала*, который обозначают $h_{21э}$. Такое странное на первый взгляд обозначение возникло от того, что первые транзисторы вызывали у инженеров отторжение и непонимание, тогда ученые предложили им математическую модель, чем, на мой взгляд, еще больше все запутали и усложнили. Обозначение $h_{21э}$ возникло из рассмотрения модели транзистора в виде четырехполюсника.

В первом приближении $h_{21э}$ можно считать равным коэффициенту β, хотя он всегда больше последнего. Учтите, что в справочниках иногда приводится именно $h_{21э}$, а иногда β, так что будьте внимательны. Разброс значений $h_{21э}$ для конкретных экземпляров весьма велик, поэтому в справочниках приводят граничные величины (от — до).

Схема с общим эмиттером

Поэкспериментировать с усилительным режимом транзистора и заодно научиться измерять $h_{21э}$ можно по схеме, приведенной на рис. 3.5. Переменный резистор должен иметь достаточно большое сопротивление, чтобы при выведенном в крайнее правое положение движке ток базы заведомо удовлетворял соотношению $I_б \cdot h_{21э} << I_к$ (ток коллектора в данном случае определяется нагрузкой). Если для транзистора (по справочнику) $h_{21э}$ составляет величину в среднем 50, а в коллекторе нагрузка 100 Ом, то переменный резистор разумно выбрать номиналом примерно 20—30 кОм и более. Выведя движок в крайнее правое по схеме положение, мы задаем минимально возможный ток базы. В этом положении следует включить питание и убедиться с помощью осциллографа или мультиметра, что транзистор близок к отсечке — напряжение на коллекторе $U_к$ будет почти равно напряжению питания (но не совсем — мы уже говорили, что для полной отсечки нужно соединить выводы базы и эмиттера между собой). Осторожно перемещая движок переменника, мы увидим, как напряжение на коллекторе будет падать (а на нагрузке, соответственно, расти). Когда напряжение на коллекторе станет почти равным нулю (т. е. транзистор перейдет в состояние насыщения), эксперимент следует прекратить, иначе можно выжечь диод «база-эмиттер» слишком большим прямым током (для предотвращения этой ситуации нужно последовательно с переменным поставить постоянный резистор небольшого номинала — на рис. 3.5 показан пунктиром).

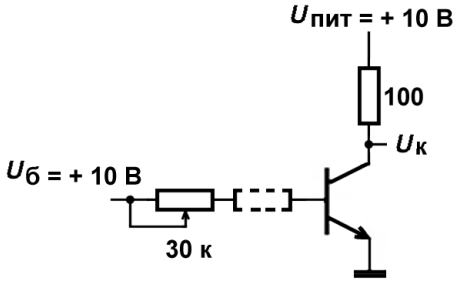


Рис. 3.5. Схема включения биполярного транзистора по схеме с общим эмиттером в усилительном режиме

Вернем движок переменника в состояние, когда напряжение на коллекторе примерно равно половине напряжения питания. Это так называемая рабочая точка транзистора в схеме с общим эмиттером. Если напряжение на базовом резисторе будет в определенных пределах колебаться, изменяя ток базы, то переменная составляющая напряжения на коллекторе будет повторять его

форму (с точностью до наоборот, т. е. инвертируя сигнал, как мы говорили ранее), но усиленную по напряжению и току. Это и есть усилительный режим транзистора. В какой степени входной сигнал может быть усилен? Все определяется знакомым нам коэффициентом $h_{21э}$. Его величину для данного экземпляра транзистора можно определить так: пусть при напряжении на коллекторе, равном половине напряжения источника питания (т. е. 5 В как на рис. 3.5), сопротивление базового резистора составляет 10 кОм. Ток коллектора (при коллекторной нагрузке 100 Ом) составит 50 мА. Ток базы составит (10 – 0,6) В/10 кОм, т. е. примерно 1 мА. Тогда их отношение и будет равно $h_{21э}$, в данном случае 50.

А каков коэффициент усиления такой схемы по напряжению? Это зависит от соотношения резисторов в базе и в коллекторе. Например, если величина базового резистора составляет 1 кОм, то изменение тока базы при изменении входного напряжения на 1 В составит 1 мА. А в пересчете через $h_{21э}$ это должно привести к изменению тока коллектора на 50 мА, что на нагрузке 100 Ом составит 5 В. Следовательно, усиление по напряжению при таком соотношении резисторов будет равно 5. Чем выше номинал резистора в базе (и ниже — нагрузки), тем меньше коэффициент усиления по напряжению. В пределе, если положить базовый резистор равным нулю, а коллекторный — бесконечности, то максимальный коэффициент усиления современных транзисторов по напряжению может составить величину порядка нескольких сотен (но не бесконечность — за счет того, что база имеет собственное входное сопротивление, а коллектор — собственное выходное). Обратите внимание на это обстоятельство: при повышении величины сопротивления в коллекторе коэффициент усиления увеличивается. В частности, это означает, что лучше вместо резистора включать источник тока, у которого выходное сопротивление очень велико. Именно так и поступают в аналоговых микросхемах, где создать источник тока в виде еще одного-двух транзисторов вместо нагрузочного резистора даже проще (см. *главу 6*).

В приведенном виде (см. рис. 3.5) схема по усилению исключительно плоха. В самом деле, все зависит от величины коэффициента $h_{21э}$, а он, во-первых, «гуляет» от транзистора к транзистору, во-вторых, очень сильно зависит от температуры (при повышении температуры повышается). Чтобы понять, как правильно построить усилительный транзисторный каскад со стабильными параметрами, нужно ознакомиться еще с одной схемой включения транзистора — схемой с *общим коллектором*.

Схема с общим коллектором

Схема с общим коллектором (ОК) показана на рис. 3.6. Учитывая, что напряжение базы и эмиттера никогда не отличается более чем на 0,6 В,

мы придем к выводу, что выходное напряжение такой схемы должно быть меньше входного именно на эту величину. Так и есть, схема с общим коллектором иначе называется *эмиттерным повторителем*, поскольку выходное напряжение повторяет входное (за вычетом все тех же 0,6 В). Каков же смысл этой схемы?

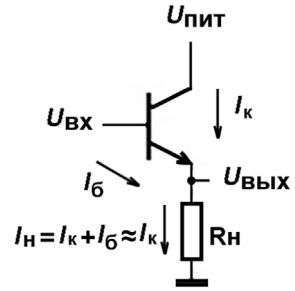


Рис. 3.6. Схема включения биполярного транзистора по схеме с общим коллектором

Схема на рис. 3.6 усиливает сигнал по току (в число раз, определяемое величиной $h_{21э}$), что равносильно увеличению собственного входного сопротивления схемы ровно в $h_{21э}$ по отношению к тому сопротивлению, которое находится в цепи эмиттера. Поэтому в этой схеме мы можем подавать на «голый» вывод базы напряжение без опасности сжечь переход «база-эмиттер». Иногда это полезно само по себе, если не слишком мощный источник (т. е. обладающий высоким выходным сопротивлением), нужно согласовать с мощной нагрузкой (в *главе 4* мы увидим, как это используется в источниках питания). Кстати, схема ОК *не инвертирует* сигнал, в отличие от схемы ОЭ.

Но главной особенностью схемы с общим коллектором является то, что ее характеристики исключительно стабильны и не зависят от конкретного транзистора, до тех пор, пока вы, разумеется, не выйдете за пределы возможного. Так, сопротивление нагрузки в эмиттере и входное напряжение схемы практически однозначно задают ток коллектора, — характеристики транзистора в этом деле никак не участвуют. Для объяснения данного факта заметим, что токи коллектора и эмиттера, т. е. ток через нагрузку, связаны между собой соотношением $I_н = I_к + I_б$, но ток базы мал по сравнению с током коллектора, потому мы им пренебрегаем и с достаточной степенью точности полагаем, что $I_н = I_к$. Но напряжение на нагрузке будет всегда равно входному напряжению минус $U_{бэ}$, которое, как мы уже выучили, всегда 0,6 В. Таким образом, ток в нагрузке есть $(U_{вх} – U_{бэ})/R_н$, и тогда окончательно получаем, что $I_к = (U_{вх} – U_{бэ})/R_н$.

Разумеется, мы по ходу дела приняли два допущения (что $I_б << I_к$ и что $U_{бэ}$ есть точно 0,6 В — и то, и другое не всегда именно так), но мы же давно договорились, что не будем высчитывать характеристики схем с точностью до процентов! Ограничение, которое накладывается транзистором, будет проявляться тут только, если мы попробуем делать $R_н$ все меньше и меньше, в конце концов либо ток коллектора, либо мощность, выделяемая на коллекторе (она равна $(U_{пит} – U_{вых}) \cdot I_к$), превысят предельно допустимые значения и тогда сгорит коллекторный переход или (если $I_к$ чем-то лимитирован) то же произойдет с переходом «база-эмиттер». Зато в допустимых пределах мы можем со схемой эмиттерного повторителя творить что угодно, и соотношение $I_к = (U_{вх} – U_{бэ})/R_н$ всегда будет выполняться.

Про такую схему говорят, что она *охвачена стопроцентной отрицательной обратной связью по напряжению*. Об обратной связи мы подробнее поговорим в *главе 6*, посвященной операционным усилителям, а сейчас нам важно, что такая обратная связь ведет к стабилизации параметров схемы и независимости их как от конкретного экземпляра транзистора, так и от температуры. Но ведь это именно то, чего нам так не хватало в классической схеме с общим эмиттером! Нельзя ли их как-то скомбинировать?

Стандартный усилительный каскад на транзисторе

Действительно, «правильный» усилительный каскад на транзисторе есть комбинация той и другой схемы, этот вариант показан на рис. 3.7. Для конкретности предположим, что $U_{пит}$ = 10 В, $U_{вх}$ = 5 В. Как правильно рассчитать сопротивления $R_э$ и $R_к$? Заметим, что схема обладает двумя выходами, из которых нас больше интересует выход 1 (выход усилителя напряжения, соответствующий выходу в схеме с общим эмиттером по рис. 3.5).

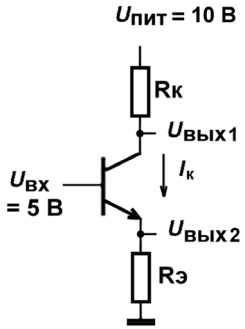


Рис. 3.7. Стандартный усилительный каскад на биполярном транзисторе

При нормальной работе каскада (для обеспечения максимально возможного размаха напряжения на выходе) разумно принять, чтобы в состоянии покоя, т. е. когда $U_{вх} = 5$ В, на выходе (на коллекторе транзистора) была половина напряжения питания (в нашем случае тоже примерно 5 В). Это напряжение зависит от коллекторного тока и от сопротивления нагрузки по этому выходу, которое равно в данном случае $R_к$. Как правило, сопротивление нагрузки $R_к$ нам задано, примем для определенности, что $R_к = 5{,}1$ кОм. Это означает, что в «хорошем» режиме, чтобы обеспечить $U_{вых1} = 5$ В, ток коллектора должен составлять 1 мА — посчитайте по закону Ома!

> ***Замечание***
>
> На самом деле средний ток коллектора в маломощном биполярном транзисторном каскаде и должен составлять величину порядка 1 мА. Если он много меньше, то в дело вступают шумы и прочие неидеальности транзистора, а когда много больше, то это неэкономно с точки зрения расходования энергии источника, и транзисторы нужно тогда выбирать более мощные, а у них намного больше шумы, утечки, они дороже, крупнее...

Но ток коллектора мы уже умеем рассчитывать, исходя из закономерностей для каскада ОК, он ведь равен $(U_{вх} - U_{бэ})/R_э$. Из этих условий получается, что резистор $R_э$ должен быть равен 4,3 кОм (мы всегда выбираем ближайшее значение из стандартного ряда сопротивлений, и больше не будем об этом упоминать). Мы не сильно нарушим законы природы, если просто положим в этой схеме $R_э = R_к = 5{,}1$ кОм (с точностью до десятых вольта выходные напряжения по обоим выходам будут равны — проверьте!).

Такая (очень хорошая и стабильная) схема нам не обеспечит никакого усиления по напряжению, это легко проверить, если при рассчитанных параметрах увеличить $U_{вх}$, скажем, на 1 В. Напряжение на эмиттере увеличится также на 1 В, общий ток коллектора-эмиттера возрастет на 0,2 мА (1 В/5 кОм), что изменит дополнительное падение напряжения на коллекторном резисторе (т. е. на нагрузке) также на 1 В в меньшую сторону (помните, что выходы инвертированы?). И никакого усиления не получится.

Зато! Мы в данном случае имеем схему, которая обладает двумя совершенно симметричными выходами: одним инвертирующим и другим, сигнал на котором точно совпадает по фазе с входным. Это дорогого стоит! Единственное, что портит картинку, — факт, что выходные сопротивления такой схемы сильно разнятся. Нагрузив нижний выход ($U_{вых2}$) какой-то еще нагрузкой (что равносильно присоединению параллельного резистора к $R_э$), мы изменим общий ток коллектора, и напряжение верхнего выхода ($U_{вых1}$) также изменится. А обратного не получается, если мы уменьшим $R_к$, нагрузив его, то $U_{вых1}$ изменится, но это практически никоим образом не скажется на $U_{вых2}$. (А куда денется разница? Ну, разумеется, «сядет» на транзисторе!)

Как нам обеспечить полную (или близкую к таковой) симметричность схемы усилителя — чуть далее. А пока нас занимает вопрос — как же добиться усиления по напряжению? У меня есть микрофон или гитарный звукосниматель с выходом 1 мВ. Хочу получить на выходе хотя бы 100 мВ, чтобы хватило для линейного входа усилителя — ну и? Оказывается, все просто, нужно только «поступиться принципами», как говаривала незабвенная Нина Андреева еще в советские времена.

Принципы заключаются в следующем: в рассчитанной схеме мы старались все сбалансировать и обеспечить оптимальный режим работы транзистора. Но оптимального ничего не бывает, ранее мы отмечали, что коэффициент усиления по напряжению каскада с общим эмиттером зависит от соотношения сопротивлений (т. е. токов в базе и коллекторе). Нарушив его по отношению к оптимальному для транзистора, мы можем что-то улучшить для себя.

Практически это делается так: мы предполагаем, что максимально возможная амплитуда на входе каскада (относительно среднего значения) не превысит, допустим, 1 В. Тогда напряжение на базе не должно быть меньше 1,7 В, иначе при минимальном сигнале транзистор запрется, и напряжение на выходе будет ограничено снизу. Примем его равным 2 В для надежности. Номинал эмиттерного резистора $R_э$ (при все том же оптимальном токе коллектора 1 мА) будет тогда равен 1,3 кОм (= (2 В – 0,7)/1 мА). Нагрузка коллектора ($R_к$) пусть останется прежней (5,1 кОм). Обратите внимание, что на выходе $U_{вых1}$ среднее напряжение — напряжение покоя — осталось то же самое (5 В), т. к. ток не изменился.

Тогда каждый вольт изменения напряжения на входе даст уже примерно 4 вольта изменения напряжения на выходе $U_{вых1}$, т. е. коэффициент усиления по напряжению составит 4 (и будет примерно равен соотношению резисторов в коллекторе и эмиттере). Мы можем в определенных пределах увеличить этот коэффициент, уменьшая номинал $R_э$ вплоть до нуля (и тем самым все больше дестабилизируя схему, как показано при описании схемы с общим эмиттером), и одновременно уменьшая диапазон усиливаемых входных напряжений. Интересным свойством рассмотренной схемы является то, что абсолютное значение напряжения питания здесь не важно — рассчитанный на одно питание каскад сохранит все свои свойства, кроме максимально допустимого выходного напряжения, и при другом.

Для усилителей переменного тока хорошим — и часто используемым — приемом является шунтирование эмиттерного резистора конденсатором большой емкости. В результате режим усилителя по постоянному току (точка покоя, т. е. напряжение на коллекторе) обеспечен, а при наличии переменного входного напряжения эмиттерный резистор по номиналу уменьшается (ведь параллельно к нему подключен конденсатор, сопротивление которого

тем меньше, чем выше частота, как мы узнали из *главы 2*), поэтому растет и коэффициент усиления напряжения всей схемы.

Дифференциальный каскад

Значительно улучшает схему комбинация двух одинаковых транзисторов в паре, соединенных эмиттерами — т. н. *дифференциальный усилительный каскад*. Дифференциальные каскады в силу их удобства широко применяли еще в эпоху недоступности микросхем (в том числе даже и в «ламповые» времена), но в настоящее время отдельно они практически не встречаются, а являются основой операционных усилителей. Тем не менее рассмотрим вкратце, как они работают.

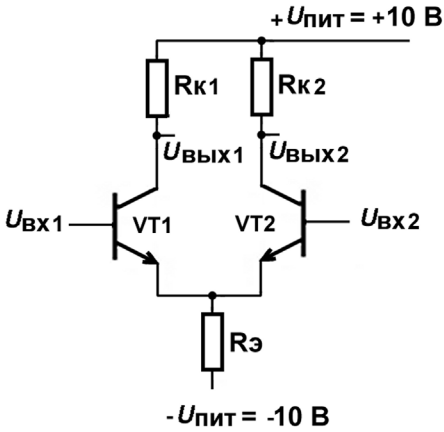


Рис. 3.8. Дифференциальный каскад на биполярных транзисторах

Дифференциальный каскад, показанный на рис. 3.8, предполагает два раздельных одинаковых питания (плюс и минус) относительно «земли», но для самого каскада это есть не более, чем условность — питание всего каскада можно рассматривать, как однополярное (равное 10 + 10 = 20 В, согласно рис. 3.8), просто входной сигнал должен находиться где-то между этими значениями. Ради удобства проектирования схем источник входного напряжения всегда привязывают к «земле», потенциал которой находится посередине (хотя и необязательно ровно посередине, но для удобства чаще поступают именно так) между потенциалами источников питания самого каскада, т. е. общее питание рассматривают, как разделенное на два — положительное и отрицательное (такое питание еще называют *двуполярным*). Относительно этой же общей «земли» мы будем также отсчитывать выходные напряжения $U_{вых1}$ и $U_{вых2}$.

Так как мы знаем, что база и эмиттер транзистора всегда «привязаны» друг к другу, то в этой схеме обе базы (в рабочем режиме) всегда будут иметь одинаковый потенциал. Поэтому если на них подавать один и тот же сигнал (базовые резисторы на рис. 3.8 не показаны), то ничего происходить не будет — току течь некуда, т. к. все потенциалы одинаковы. Вся конструкция из двух транзисторов будет смещаться относительно «земли» в соответствии с поданным сигналом, а на выходах ничего и не шелохнется (в идеале). Такой сигнал называют *синфазным*.

Иное дело, если сигналы на входах различаются, тогда они будут усиливаться. Такой сигнал называют *дифференциальным* (противофазным). Это основное свойство дифференциального усилителя, которое позволяет выделять небольшой сигнал на фоне довольно сильной помехи. Помеха одинаково — синфазно — действует на оба входа, а полезный сигнал усиливается.

Мы не будем здесь далее подробно разбирать работу этой схемы, только укажем некоторые ее особенности:

- входное сопротивление дифференциального каскада равно входному сопротивлению каскада с общим коллектором, т. е. достаточно велико;
- усиление по напряжению (для дифференциального сигнала) составляет 100 и более раз. Если вы хотите получить точно определенный коэффициент усиления, в каждый из эмиттеров нужно ввести по одинаковому резистору — тогда $K_{ус}$ будет определяться, как для каскада на рис. 3.7. Но обычно в таком режиме дифференциальный усилитель не используют. Основная область их применения — в системах с обратной связью, которая и задает необходимый коэффициент усиления (см. *главу 6*);
- выходы строго симметричны;
- резистор $R_{к1}$, если не требуется $U_{вых1}$, вообще можно исключить (или наоборот, смотря какой выход задействован).

Полевые транзисторы

Типы полевых транзисторов гораздо более разнообразны, чем биполярных (к полевым, кстати, и принадлежал самый первый прототип транзистора, изобретенный Шокли еще в 1946 году). Их существует более десятка только основных разновидностей, но всем им присущи общие черты, которые мы сейчас кратко и рассмотрим.

Простейший полевой транзистор с *p*-*n*-переходом показан на рис. 3.9 (в данном случае с *n*-каналом). Аналогичные базе, коллектору и эмиттеру выводы называются *затвором*, *стоком* и *истоком*. Если потенциал затвора равен

потенциалу истока (имеется в виду аналог замыкания цепи «база-эмиттер» у биполярного), то, в отличие от биполярного, такой полевой транзистор открыт. Но есть и еще одно существенное отличие: если биполярный транзистор при полном открывании имеет почти нулевое сопротивление цепи «коллектор-эмиттер», то полевой в этих условиях работает довольно стабильным источником тока, поскольку ток в цепи истока почти не зависит от напряжения на стоке. Сама величина тока определяется конкретным экземпляром транзистора и называется *начальным током стока*. Запереть же полевой транзистор удается подачей отрицательного (порядка 7—10 В) напряжения на затвор относительно истока. В промежуточном состоянии прибор с *n*-каналом находится в активном режиме, при этом ток стока зависит от напряжения на затворе.

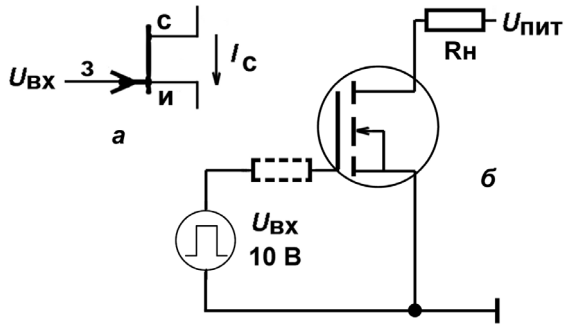


Рис. 3.9. Полевые транзисторы:
а — включение полевого транзистора с *p-n*-переходом и *n*-каналом;
б — полевой (MOSFET) транзистор с изолированным затвором в режиме ключа

Уникальной особенностью любого полевого транзистора является то, что в рабочем режиме он фактически не потребляет тока по входу затвора. Здесь достаточно лишь создать соответствующий потенциал, ведь диод «затвор-исток» в рабочем режиме смещен в обратном направлении и ток через него определяется только токами утечки, которые равны нано- и микроамперам, как говорилось ранее! В этом отношении полевой транзистор аналогичен электронной лампе.

В полевых транзисторах с изолированным затвором (т. н. МОП-транзисторах, от «металл — окисел — полупроводник» или, по-английски, MOS), последний вообще изолирован от цепи «сток-исток» (тонким слоем окисла кремния SiO_2), и там в принципе нет и не может быть никакого тока через цепь затвора. Правда, когда на затвор подается переменное напряжение (или короткий импульс), в дело вступает конденсатор, образованный затвором и истоком.

Как следует из *главы 2*, перезаряд этого конденсатора (его емкость может составлять десятки пикофарад) может приводить к значительному реактивному току в цепи затвора. На подобных транзисторах построены практически все современные логические микросхемы, отличающиеся практически нулевым потреблением тока в статическом режиме (см. *главу 8*).

Приведенные нами примеры не исчерпывают разнообразия типов полевых транзисторов. Например, т. н. MOSFET-транзисторы (см. рис. 3.9, *б*) управляются аналогично тому, как биполярный в схеме с общим эмиттером: при нулевом напряжении на затворе относительно истока транзистор заперт, при положительном напряжении порядка 5—10 В — полностью открыт, причем в открытом состоянии он представляет собой крайне малое сопротивление (у некоторых типов менее 0,01 Ом). Такие транзисторы имеют мощность от единиц до сотен ватт и используются, например, для управления шаговыми двигателями или в импульсных источниках питания.

Вообще «полевики» гораздо ближе к той модели транзистора, в которой промежутки «коллектор-эмиттер» или «сток-исток» представляются в виде управляемого сопротивления, т. к. у полевых транзисторов это *действительно* сопротивление. Условно говоря, со схемотехнической точки зрения биполярные транзисторы являются приборами для усиления тока, а полевые — для усиления напряжения.

Стабилитроны

Стабилитрон представляет собой обычный диод с вольт-амперной характеристикой, показанной на рис. 3.1, за одним исключением: при превышении некоторого обратного напряжения (индивидуального для каждого типа стабилитрона) он обратимо пробивается и начинает работать, как очень малое сопротивление. Это можно представить себе, как если бы обычное прямое падение напряжения, составляющее 0,6 В, увеличилось бы вдруг до большой величины. Стоит только снизить напряжение ниже оговоренного, стабилитрон опять запирается и больше не участвует в работе схемы. Напряжения стабилизации могут быть самыми разными (от 2 до 300 В). Учтите, что тепловая мощность, равная произведению тока через стабилитрон на его напряжение стабилизации, выделяется на нем самом, поэтому, чем выше напряжение стабилизации, тем ниже допустимый ток, который должен быть ограничен резистором нагрузки. В справочных данных также указывается обычно минимально допустимое значение тока, при котором стабилитрон еще «держит» нужное напряжение.

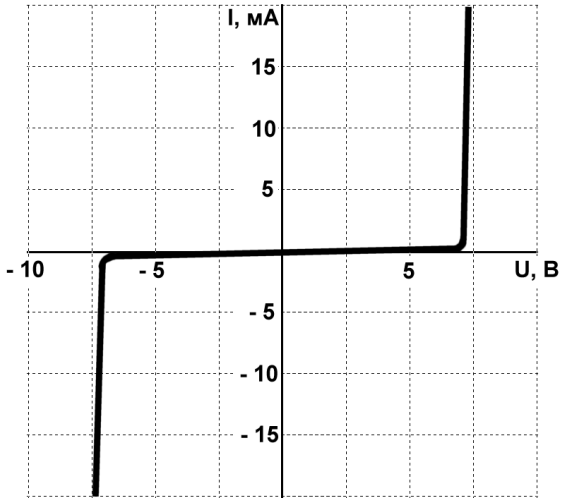


Рис. 3.10. Вольт-амперная характеристика двустороннего стабилитрона

Удобны двусторонние стабилитроны (которые представляют собой два обычных, включенных встречно-параллельно), обеспечивающие симметрию характеристик и в положительном и в отрицательном направлении включения. Вольт-амперная характеристика такого двустороннего стабилитрона типа КС170 показана на рис. 3.10. Отметьте, что характеристика в области пробоя все же имеет некоторый наклон, т. е. при возрастании тока через прибор напряжение на нем не остается строго постоянным, а растет (этот эффект обусловлен т. н. дифференциальным сопротивлением). К тому же напряжение стабилизации меняется с температурой.

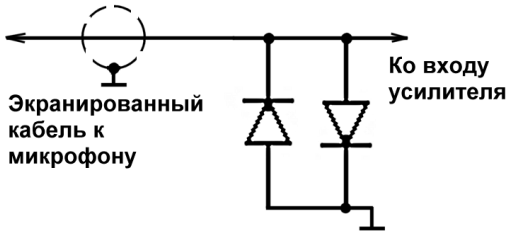


Рис. 3.11. Схема для защиты входа микрофонного усилителя

Как ясно из предыдущего, простейшим стабилитроном может быть обычный диод, включенный в прямом направлении, и его часто употребляют в таком качестве. Напряжение стабилизации составит при этом, естественно, 0,6 В

(для его увеличения можно включить последовательно два и более диодов). Как видно из вольт-амперной характеристики диода (см. рис. 3.1), стабильность пресловутого напряжения 0,6 В оставляет желать лучшего (и от тока зависит, и от температуры), но во многих случаях особой прецизионности и не требуется. На рис. 3.11 приведена схема ограничителя напряжения на двух диодах (если требуется более высокое напряжение ограничения, их можно заменить на стабилитроны или на один двусторонний стабилитрон). Эта схема удобна, например, для защиты высокоомного входа микрофонного усилителя. Нормальное напряжение с микрофона составляет несколько милливольт и диоды никак не влияют на работу схемы, поскольку таким маленьким напряжением не открываются. Но если микрофон присоединен через длинный кабель, то на входе могут создаваться помехи (от промышленного оборудования, от поднесенного к неподключенному входу пальца, или, скажем, от грозовых разрядов), которые сильно превышают указанные милливольты и могут вывести из строя нежные и чувствительные микрофонные усилители. В приведенной схеме такие помехи любой полярности замыкаются через диоды и входное напряжение не может превысить 0,6—0,7 В ни при каких условиях.

Заметки на полях

У внимательного читателя может возникнуть вопрос — ведь согласно вольт-амперной характеристике и стабилитрона и диода ток при превышении соответствующего напряжения растет очень быстро, так не сгорят ли эти входные диоды при наличии высоковольтной помехи? Отвечаем — энергия помехи обычно очень мала, поэтому ток хоть и может быть достаточно велик, но на протяжении очень короткого промежутка времени, а такое воздействие и диоды и стабилитроны выдерживают без последствий.

Стабилитроны в чистом виде хороши в качестве ограничителей и маломощных источников напряжения, а для формирования действительно стабильного напряжения (например, опорного для АЦП и ЦАП) применяются интегральные стабилизаторы, которые при наличии трех выводов (вход, выход и общий) дают на выходе стабильное напряжение. Они сродни обычным стабилизаторам напряжения, которые мы будем разбирать в *главе 4*, но значительно более стабильны и мало зависят от температуры. Например, интегральный стабилизатор типа MAX873, который в диапазоне 4—30 В на входе дает на выходе ровно 2,5 В, обладает еще и весьма высокой стабильностью. Даже если положить на него паяльник (тем самым нагрев его градусов до 200), то напряжение на выходе этого стабилизатора и не шелохнется. В современной интегральной технике источники опорного напряжения обычно встраивают прямо в нужные микросхемы, но часто предусматривают вход и для внешнего такого источника, потому что вы всегда можете захотеть изобрести что-нибудь получше.

Оптоэлектроника и светодиоды

Очень многие физические процессы обратимы. Типичный пример — если пластинка кварца изгибается под действием электрического поля, то принудительное изгибание пластинки должно привести к возникновению зарядов на ее концах — как и происходит в действительности, и этот эффект лежит в основе устройства кварцевых резонаторов для реализации высокоточных генераторов частоты (см. *главу 9*). Не давало покоя физикам и одно из первых обнаруженных свойств полупроводникового *p-n*-перехода — зависимость его проводимости от освещения. Этот эффект немедленно стал широко использоваться в различных датчиках освещенности (фотосопротивлениях, фотодиодах, фототранзисторах), которые пришли на замену хоть и весьма чувствительным, но крайне неудобным для широкого применения вакуумным фотоэлементам. Затем появился целый класс устройств — оптоэлектронные приборы.

Заметки на полях

Кстати, любой полупроводниковый диод в стеклянном корпусе является неплохим датчиком освещенности, его обратный ток сильно зависит от наличия света. Особенно этим отличаются старые германиевые диоды (типа Д2, Д9). Можете попробовать поэкспериментировать, только не забывайте два обстоятельства: во-первых, сам этот ток очень мал (обратное сопротивление диода весьма велико), что потребует хороших высокоомных усилителей, во-вторых, то, что от температуры этот обратный ток зависит еще больше, чем от света.

Оптоэлектроника

В оптоэлектронных приборах (оптронах) через светодиод (обычно инфракрасный, о них мы поговорим далее) пропускается зажигающий его ток, в результате чего в воспринимающем *p-n*-переходе фотодиода (или фототранзистора) ток резко возрастает. Между входным светодиодом и выходом при этом имеется изолирующая прокладка, которая позволяет гальванически развязать выводы входа и выхода.

Самый простой вариант такого прибора — диодная оптопара (рис. 3.12), которая обычно служит для электрически изолированной передачи линейных сигналов (например, звуковых колебаний или уровней постоянного тока в регулирующих устройствах). В ней обратный ток ($I_{вых}$) приемного диода линейно зависит от управляющего тока через светодиод ($I_{вх}$). Обратите внимание, что рабочая полярность для фотодиода обратная, чем для обычного, отчего у таких компонентов, если они выпускаются в отдельном корпусе, плюсом помечен катод, а не анод.

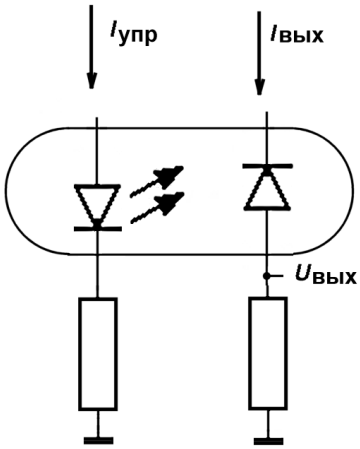


Рис. 3.12. Диодная оптопара

Встречаются и варианты оптоэлектронного реле: так, бесконтактное реле типа D24125 фирмы Crydom позволяет коммутировать переменное сетевое напряжение до 280 В при токе 125 А, путем подачи напряжения 3—5 В при токе 3 мА через управляющий светодиод (т. е. прямо от логической микросхемы). 10 мВт напрямую управляют мощностью примерно в 35 кВт (при полной гальванической развязке) — ей-богу, совершенно беспрецедентный случай, обычным электромагнитным реле недоступный! Тем не менее обычные электромагнитные реле также довольно широко применяются, и мы далее остановимся на них подробнее.

Набиравшая обороты космическая отрасль быстро сосредоточила усилия вокруг реализации другого эффекта: возможности генерации тока в полупроводниковом переходе под действием света, а также картинка искусственного спутника Земли с широко раскинутыми темно-синими панелями солнечных батарей теперь стала уже традиционной. Но вероятно можно таким образом и генерировать свет, если подавать на *p-n*-переход напряжение? Оказалось, что можно, но это было реализовано далеко не сразу.

Светодиоды

Первым «поддался» инфракрасный (невидимый глазом) и красно-зеленый участок спектра. К началу 80-х годов полупроводниковые светодиоды (LED — Light Emission Diode), излучающие в ИК-диапазоне, уже стали широко использоваться в дистанционных пультах управления, а красненькие и зелененькие сигнальные светодиоды, хоть и были тогда еще куда тусклее

традиционных лампочек накаливания, зато намного более долговечными и потребляли существенно меньше энергии.

В настоящее время все основные проблемы решены и освоен фактически весь видимый спектр, включая синий и даже ультрафиолетовый диапазон. Характерная особенность любых светодиодов — они излучают свет одной (точнее, близкой к этой одной) длины волны, из-за чего насыщенность излучаемого света превосходит все чаяния художников. Существует не менее двух десятков разновидностей светодиодов для разных длин волн, охватывающих все цвета видимого спектра (частично они перечислены в табл. 3.1, соответствующей продукции фирмы Kingbright).

Таблица 3.1

Длина волны, нм	Обозначение	Цвет свечения
700	H	Красный
660	SR	Красный
640	SU	Красный
625	I (E)	Чистый красный
610	N (SE)	Чистый оранжевый
590	Y (SY)	Желтый
565	G (SG, MG)	Зеленый
555	PG	Чистый зеленый
465	MB	Голубой
445	NB	Голубой
430	PB	Чистый синий

Светодиоды бывают обычной и повышенной яркости. Их выбор определяется практическими соображениями. Так, в большинстве случаев повышенная яркость не нужна и только будет слепить глаза, если светодиод установлен в качестве, скажем, индикатора наличия напряжения, причем регулировать такую яркость к тому же непросто. Очень тщательно следует подходить и к выбору корпуса: матовый (диффузный) рассеиватель обеспечивает меньшую яркость, зато светящуюся полусферу видно под углом почти 180° во все стороны.

Со схемотехнической точки зрения все светодиоды, независимо от цвета свечения, представляют собой обычные диоды, за одним исключением — прямое падение напряжения на них превышает обычные для кремниевых *p-n*-переходов 0,6 В и составляет: для красных и инфракрасных 1,5—1,8 В,

для желтых, зеленых и синих — 2—3 В. В остальном их включение не отличается от включения обычных диодов в прямом направлении. Светодиод есть прибор, управляемый током (а не напряжением, как лампа накаливания), поэтому должен иметь токоограничивающий резистор. Значение тока, при котором практически любой светодиод нормально светится, составляет 3—8 мА (хотя предельно допустимое может быть и 40 мА), на эту величину и следует рассчитывать схему управления светодиодами. При этом нужно учитывать, что яркость, воспринимаемая глазом, не зависит линейно от тока, поэтому вы можете и не заметить разницу в свечении при токе 5 или 10 мА, а разница между 30 и 40 мА будет еще менее заметной.

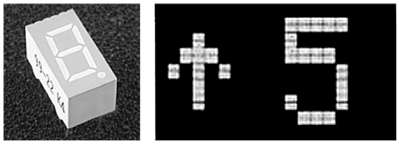
Иногда токоограничивающий резистор встраивают прямо в светодиод (в этом случае яркость свечения уже управляется напряжением, как у обычной лампочки, а не током) — это обычная практика для «мигающих» светодиодов со встроенным генератором частоты. Обычное предельное напряжение для таких светодиодов составляет 12—15 В.

Светодиоды делают разной формы: обычно они круглые, но встречаются также плоские, квадратные и даже треугольные. Широкое распространение сейчас имеют двухцветные светодиоды. Они бывают двух- и трехвыводные. С последними все понятно — это просто два разноцветных светодиода (зеленый и красный) в одном корпусе, управляющиеся раздельно. Подал ток на один — зажегся красный, на другой — зеленый, на оба — желтый (третий вывод общий), а манипулируя величиной токов, можно получить все промежуточные переходы. Но еще интереснее двухвыводный прибор, который представляет собой два разноцветных светодиода, включенные встречно-параллельно. Поэтому в них цвет свечения зависит от полярности тока: в одну сторону красный, в другую — зеленый. Самое интересное получается, если подать на такой светодиод переменное напряжение, тогда он светится желтым! Можно встретить в продаже и светодиоды белого свечения, которые все чаще служат в качестве экономичных и долговечных источников света.

Светодиодные индикаторы

Так как собственное падение напряжения на светодиодах невелико, то их можно включать последовательно, чем пользуются производители цифровых сегментных индикаторов. Но тут дело осложняется тем, что отдельный светодиод представляет собой фактически точечный источник света, и нарисовать с его помощью длинную светящуюся полоску непросто даже при наличии рассеивающей свет пластмассы (причем, как ни парадоксально, чем меньше габариты, тем хуже выглядят плоские светодиоды). Мелкие цифровые индикаторы (с длиной одного сегмента до 5—6 мм) содержат по одному

светодиоду в сегменте, а более крупные — по два и более. Это нужно учитывать при проектировании, так как семисегментный цифровой индикатор с высотой цифры 12,7 мм и более имеет падение напряжения на каждом сегменте, превышающее 4 В, и управлять им от пятивольтового контроллера напрямую затруднительно — номинальный запас в несколько десятых вольта легко «сожрется» собственным сопротивлением выхода контроллера и «проседанием» источника питания, отчего ваш индикатор вообще может и не загореться. Для таких случаев приходится идти на заведомые потери и питать индикаторы от повышенного напряжения через транзисторные ключи или специальные схемы управления индикаторами. Красота требует жертв! Набор семисегментных цифровых светодиодных индикаторов в четыре цифры в каком-нибудь мультиметре может потреблять до 100—200 мА тока, зато насколько он выглядит красивее по сравнению с почти не потребляющими, но совершенно «слепыми» черно-белыми жидкокристаллическими панелями!



а *б*

Рис. 3.13. Светодиодные индикаторы:
а — семисегментный; *б* — дисплей на основе матричного индикатора

Семисегментные индикаторы (рис. 3.13, *а*) бывают сдвоенными и строенными; кроме них, встречаются шестнадцатисегментные индикаторы, которые позволяют формировать буквы и специальные знаки. Такие индикаторы для удобства управления ими выполняют с общим анодом (тогда на индикатор подается общее питание, а зажигание сегментов производится коммутацией их к «земле») и с общим катодом (сегменты имеют общую «землю», а зажигание производится подачей тока на каждый сегмент). Почти всегда выпускаются идентичные внешне типы и той и другой конфигурации. Для формирования длинных строк используют матричные индикаторы (рис. 3.13, *б*), которые нередко встречаются в виде довольно больших дисплеев, содержащих несколько сотен точек.

ЖК-дисплеи

Жидкокристаллические (ЖК) индикаторы встречаются обычно в виде готовых ЖК-дисплеев для распространенных применений — например, для часов, магнитол, музыкальных центров, или в виде многоразрядного набора

цифр. Есть и матричные ЖК-дисплеи для формирования бегущей строки, многострочные — для текстовых сообщений и т. п., вплоть до полнофункциональных ЖК-матриц, в том числе цветных, тех, что используются в большинстве современных массовых устройств, от мобильных телефонов до широкоэкранных телевизионных панелей.

Все ЖК-дисплеи отличаются практически нулевым потреблением энергии в статическом режиме, энергия уходит только на переключение ЖК-ячейки. Правда, большинство матричных ЖК-дисплеев, предназначенных для демонстрации произвольных изображений (в том числе все цветные), не могут обойтись без подсветки, которая довольно энергоемка (так, в ноутбуках — более половины общего потребления). Но нас здесь интересуют лишь обычные ЖК-дисплеи, применяемые в качестве цифровых или цифробуквенных табло. Устройство ячейки такой простейшей (пассивной) матрицы или индикатора с зеркалом вместо подсветки показано на рис. 3.14.

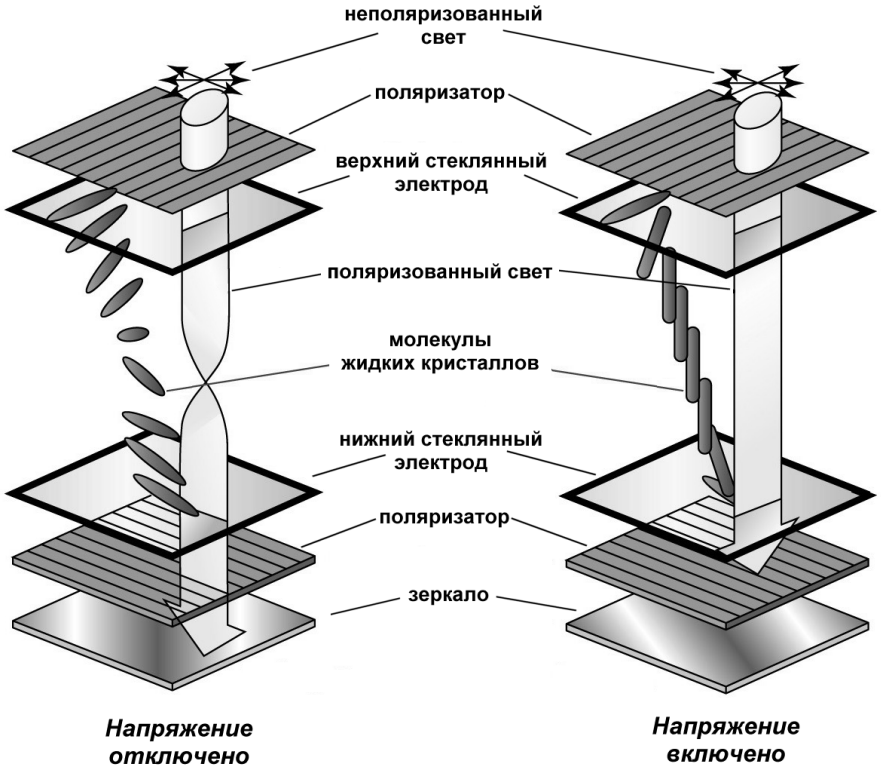


Рис. 3.14. Устройство пассивной ЖК-ячейки

Здесь слой жидких кристаллов толщиной несколько микрон находится между двумя стеклянными электродами, причем за счет специальной структуры поверхности стекла молекулы кристалла ориентированы параллельно плоскости этих электродов. Сверху и снизу такого «сэндвича» расположены пластины-поляризаторы, ориентированные перпендикулярно друг другу. Толщина слоя жидких кристаллов рассчитана так, что в исходном состоянии он поворачивает плоскость поляризации световой волны ровно на 90°. В результате в обесточенной ячейке (на рис. 3.14, слева) свет беспрепятственно проходит через весь «пирог», отражается от зеркала (оно сделано матовым, чтобы не отражало окружающих предметов) и возвращается обратно. Подобная матрица в обесточенном состоянии выглядит, как обычная стеклянная пластинка.

Когда вы подаете на электроды напряжение (на рис. 3.14, справа), то электрическое поле ориентирует молекулы жидкого кристалла вдоль его силовых линий, т. е. перпендикулярно плоскости электродов. Жидкий кристалл теряет свои свойства и перестает поворачивать плоскость поляризации света. За счет перпендикулярной ориентации поляризационных пластин весь «пирог» перестает пропускать свет. Образуется черная точка (или сегмент цифрового индикатора — в зависимости от конфигурации электродов).

Подобные монохромные ЖК-дисплеи всем хорошо знакомы, и используются в наручных и настольных часах, в портативных измерительных приборах, в дисплеях калькуляторов, плееров, магнитол, фотокамер. Величина напряжения сверх некоего, очень небольшого, предела (порядка 1—3 В), на «яркость» (точнее, на контрастность) такой ячейки практически не влияет. Поэтому таким способом получаются очень контрастные, выразительные монохромные цифробуквенные индикаторы и небольшие табло, для приличной разборчивости символов на которых достаточно лишь слабой внешней засветки.

Управлять сегментами такого индикатора, кстати, приходится с помощью разнополярного напряжения (это существенное, но не принципиальное неудобство), потому что однажды «засвеченный» сегмент может оставаться в таком состоянии часами даже после снятия напряжения с электродов, и возвращать в исходное состояние его приходится принудительно, подачей напряжения противоположной полярности.

Пассивные ЖК-матрицы как уже говорилось, отличаются практически нулевым потреблением энергии, но имеют малое быстродействие — система параллельных электродов по сути представляет собой отличный конденсатор, да еще и заполненный электролитом (жидкими кристаллами) как будто специально для увеличения его емкости. Вместе с неизбежно высоким сопротивлением тончайших прозрачных электродов ячейка образует отличный

фильтр низкой частоты. Поэтому время реакции при подаче импульса напряжения — сотня-другая миллисекунд. Для цифровых индикаторов это не имеет никакого значения, но для компьютерных и телевизионных дисплеев с сотнями тысяч и миллионами ячеек это никуда не годится, потому там необходимы активные матрицы, содержащие усилительные тонкопленочные транзисторы (TFT).

Управляют ЖК-дисплеями обычно от специальных микросхем-драйверов, с одной из таких микросхем мы познакомимся в *главе 10*. Следует отметить, что применение ЖК-индикаторов, на взгляд автора, оправданно лишь в автономных устройствах, где важно низкое потребление. В приборах, питающихся от сети, целесообразнее светодиодные индикаторы — они значительно красивее и эргономичнее. Однако сформировать на светодиодах произвольное изображение (например, даже просто отобразить названия месяцев и дней недели в часах-календаре) гораздо сложнее, чем на ЖК-дисплее, конфигураций которых выпускается значительно больше.

Электромагнитные реле

Конечно, выдающийся американский физик Джозеф Генри, помогая художнику Самюэлю Морзе в постройке телеграфа, и не думал ни о какой электронике, которая потом завоюет мир. Электромагнитное реле он изобрел даже не в рамках науки, которая, как известно, есть способ познания мира и чурается практики, а просто, чтобы «помочь товарищу», который, впрочем, наверняка платил неплохие деньги.

Так это было или иначе — важно, что электромагнитное реле стало одним из самых главных технологических изобретений XIX века. По популярности ему не затмить, конечно, электрического освещения, электрогенератора и электродвигателя, телеграфа, телефона и прочих достижений «века электричества», но факт, что именно этот не очень известный широкой публике приборчик еще недавно был одним из важнейших компонентов любой электрической системы. На нем даже строили компьютеры.

Реле стало первым в истории — задолго до ламп и транзисторов — усилителем электрических сигналов. С помощью реле напрямую не усилить предвыборную речь кандидата в президенты, но если текст закодировать нулями-единицами, как мы это будем делать далее, то реле справится с такой задачей ничуть не хуже любого другого устройства, — именно на этом свойстве было основано его применение в телеграфе Морзе.

Конечно, быстродействие реле, как ключевого элемента, оставляет желать лучшего — даже о килогерцах здесь речь не идет, обычная скорость срабатывания составляет для самых малогабаритных и быстродействующих реле

составляет десятки миллисекунд, что соответствует частотам в десятки герц. Но в режиме быстрого переключения реле использовать и не надо, для этого существуют другие электронные компоненты. Реле хороши там, где нужно надежно коммутировать нагрузку с минимальными потерями в контакте. Огромным преимуществом реле является не только полная гальваническая развязка между входом и выходом, но и низкое сопротивление контактов. По этой причине их применяли до самого последнего времени, например, для коммутации в измерительных схемах, где очень важно, чтобы сопротивление измерительных цепей было минимальным и стабильным. Учтите, что указываемые в справочниках параметры контактов (типа «переходное сопротивление не более 1 Ом») обычно сильно завышены, они рассчитаны на наихудший случай.

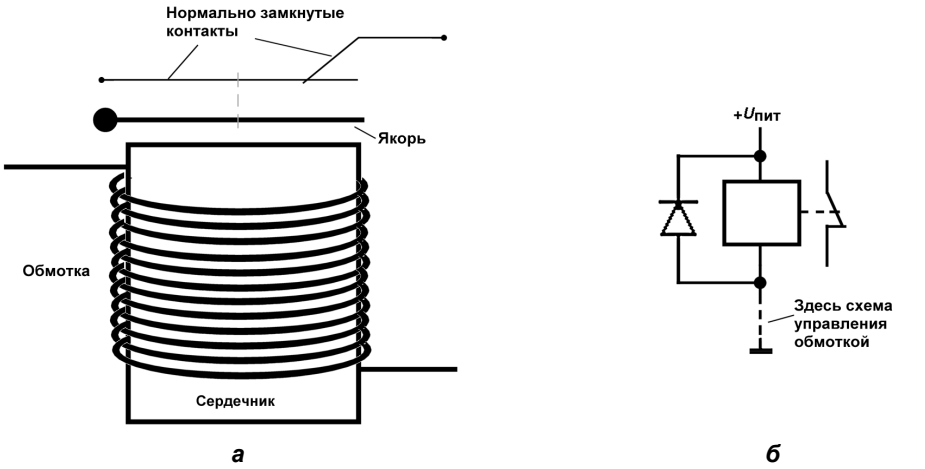


Рис. 3.15. Схематичное устройство (*а*) и рекомендуемая схема включения (*б*) электромагнитного реле

На рис. 3.15, *а* изображена схема простейшего электромагнитного реле, а на рис. 3.15, *б* — его подключение. Любое реле — независимо от конструкции — обязательно содержит три главных компонента: обмотку, якорь и контакты, последних может быть от одной пары до дюжины. Контакты бывают нормальнозамкнутые (тогда при срабатывании реле они размыкаются, см. рис. 3.15), нормальноразомкнутые (при срабатывании замыкаются) и перекидные.

Обмотка реле представляет собой катушку индуктивности (соленоид), около которой (или в которой) при подаче тока перемещается якорь, выполненный из ферромагнитного материала. Разумеется, вокруг этой базовой конструк-

ции за много лет были накручены различные «прибамбасы»: так, существуют реле, которые при каждой подаче импульса тока перебрасываются в противоположное положение, реле, контакт в которых может иметь три стабильных положения, т. е. трехпозиционные (замкнуто — нейтраль — замкнуто) и т. п., но мы их не будем рассматривать, потому что большинство функций таких специализированных реле давно выполняют логические микросхемы, и куда успешней.

Подробности

Несколько отличаются по конструкции т. н. герконовые реле, у которых якорем служат сами контакты. Слово «геркон» расшифровывается, как «герметизированный контакт». Герконы выпускаются и отдельно, они представляют собой стеклянную трубочку с двумя или тремя выводами от запаянного в нее контакта (простого или перекидного), защищенного таким образом от влияния внешней среды. Контакт под воздействием внешнего магнитного поля (например, при поднесении постоянного магнита) может замыкаться и размыкаться. Герконы часто служат в качестве датчиков положения. Герконовые реле обычно представляют собой такой геркон, на который намотана обмотка с теми или иными параметрами.

Главным и основным свойством, побуждающим инженера-электротехника и электроника прибегать к обычным реле в век господства транзисторов и микросхем, является полная (более полной и представить себе трудно) гальваническая развязка не только обмотки от коммутируемого напряжения, но, если пар контактов больше одной, то и различных коммутируемых напряжений друг от друга. Коммутация происходит чисто механическим способом, потому коэффициент усиления по мощности у реле ого-го-го какой! Например, обмотка реле РЭС9 потребляет 30 мА при 27 вольтах, что составляет меньше ватта, но может двумя парами контактов коммутировать нагрузки до 1 А при 220 вольтах переменного тока на каждый контакт в отдельности, т. е. в сумме почти полкиловатта! В этом отношении их могут «переплюнуть» только оптоэлектронные реле, о которых речь шла ранее.

Главный недостаток электромагнитных реле в сравнении с полупроводниковыми устройствами — энергетический порог, с которого начинается управление обмотками, весьма велик. Все же токи в 30—50 мА при напряжениях 15—27 вольт, т. е. мощности порядка ватта (это для малогабаритных реле — для реле покрупнее нужна еще большая мощность) — запредельны для современной электроники, и это слишком большая роскошь, если требуется всего только включить нагрузку в виде лампочки. В справочниках приводится либо величина тока через обмотку, либо величина рабочего напряжения, что равнозначно, потому что величина сопротивления обмотки тоже всегда указывается. Обычно одинаковые типы реле имеют разновидности с разными сопротивлениями обмоток (это определяется т. н. «паспортом реле»).

Заметки на полях

Другим недостатком обмоток реле, как нагрузки для полупроводниковых приборов, является то, что они представляют собой индуктивность. Для постоянного тока это просто сопротивление, но в момент переключения она может доставить немало неприятностей. В момент разрыва или замыкания управляющей цепи на обмотке реле возникает импульс напряжения (по полярности он препятствует направлению изменения тока в обмотке), и если индуктивность обмотки велика, а ее собственное (активное) сопротивление мало, то импульс этот может вывести из строя коммутирующий прибор (например, транзистор). В любом случае это создает сильные помехи остальным элементам схемы по шине питания. Поэтому при стандартном включении реле всегда рекомендуется устанавливать параллельно его обмотке диод (даже если коммутация происходит не от полупроводниковых источников, а от таких же реле) в таком направлении, чтобы в статическом режиме, когда все успокоилось и никто ничего не коммутирует, диод этот тока не пропускал (см. рис. 3.15, *б*). Тогда выброс напряжения ограничивается на уровне напряжения на открытом диоде, т. е. 0,6 В. Для управления подобными элементами (кроме реле, это, например, обмотки двигателей) в мощные коммутирующие транзисторы, подобные показанным на рис. 3.9, *б*, часто устанавливают защитные диоды еще в процессе их изготовления. Маломощные реле, управляемые от логических схем, также не требуют установки специальных диодов, роль которых играют защитные диоды микросхем (см. *главу 8*).

Следует учитывать еще вот какую особенность электромагнитных реле: ток (напряжение) срабатывания у них много превышает ток (напряжение) отпускания. Так, если в характеристиках указано, что номинальное напряжение реле составляет 27 В, то это напряжение, при котором замыкание нормально разомкнутых до этого контактов гарантируется. Но совершенно не обязательно (а иногда и не нужно) выдерживать это напряжение длительное время. Так, 27-вольтовые реле спокойно могут удерживать контакты в замкнутом состоянии вплоть до того момента, пока напряжение на их обмотке не снизится до 5—8 В. Это очень удобное свойство электромагнитных реле — называемое гистерезисом, — которое позволяет избежать дребезга при срабатывании-отключении и даже сэкономить на энергии при работе с ними. Так, на рис. 3.16, *а* приведена схема управления реле, которое в начальный момент времени подает на него нужное номинальное напряжение для срабатывания, а затем неограниченное время удерживает реле в сработавшем состоянии при пониженной величине тока через обмотку.

На рис. 3.16 также приведены еще две классические схемы. Первая (рис. 3.16, *б*) называется «схемой самоблокировки» и очень часто применяется в управлении различными мощными устройствами, например, электродвигателями станков. Мощные реле-пускатели для таких двигателей имеют даже специальную отдельную пару маломощных контактов, предназначенную для осуществления самоблокировки. В этих случаях ток через стандартные кнопки «Пуск» и «Стоп» не превышает тока через обмотку пускателя (который

составляет несколько десятков или сотен миллиампер), в то время, как мощность разрываемой цепи может составлять многие киловатты, притом цепи трехфазной со всякими дополнительными неприятностями типа огромных индуктивностей обмоток мощных двигателей.

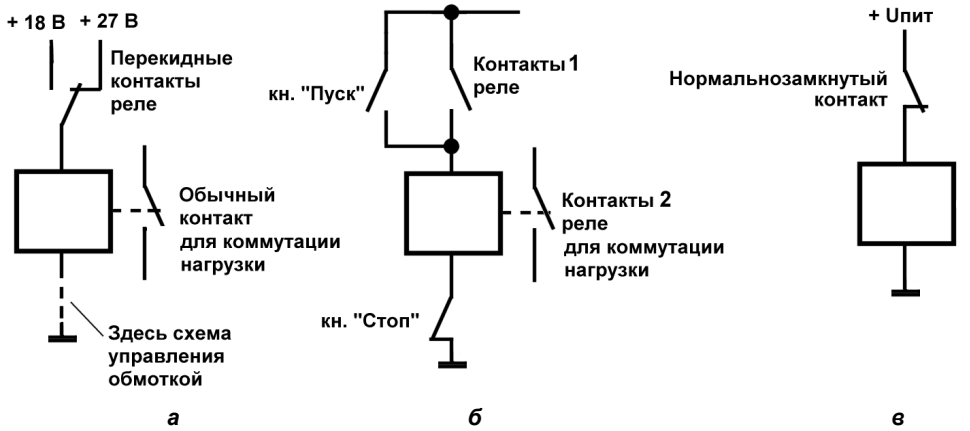
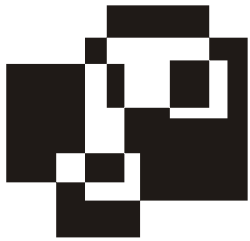


Рис. 3.16. Некоторые схемы включения реле: *а* — со снижением напряжения удержания; *б* — схема самоблокировки с кнопками «Пуск» и «Стоп»; *в* — схема классического электромеханического звонка

Другая схема (рис. 3.16, *в*) скорее забавна, и есть дань прошлому, когда никакой электроники не существовало. Это схема простейшего электрического звонка, которая может быть реализована на любом реле. Оно и само по себе при подключении по этой схеме задребезжит (правда, звук может быть самым разным, в зависимости от быстродействия и размеров реле, потому лучше употребить слово «зазуммерит»), но в обычном звонке якорь еще связывают со специальной тягой, которая в процессе работы стучит по металлической чашке, формируя звуковой сигнал. Есть и более простая конструкция электромеханического звонка, когда на обмотку реле просто подают переменное напряжение, от чего якорь вибрирует с его частотой (так устроены, например, звонки старинных телефонов с крутящимся диском), но нас тут интересует именно классическая схема, потому что в ней в чистом виде реализован другой основополагающий принцип электроники, так или иначе присутствующий в любых генераторах колебаний — принцип положительной обратной связи. Якорь в первый момент притягивается, в результате питание размыкается, якорь отпускает — питание замыкается, якорь притягивается и т. д. Частота генерируемых колебаний зависит исключительно от механической инерции деталей реле.



Глава 4

Правильное питание — залог здоровья

> Не так-то просто понять, как справиться с вредителями. Сначала их надо изучить, разобраться, как они устроены, чем питаются…
>
> *Реклама средств от насекомых*

Трансформаторы и фильтрующие конденсаторы зачастую составляют основную часть массы и габаритов многих современных микроэлектронных устройств. Однако реальной альтернативы обычным трансформаторным источникам питания, которые мы здесь будем рассматривать, всего две: либо электрохимические источники тока (батареи и аккумуляторы), либо импульсные источники питания (экзотику вроде солнечных батарей мы учитывать не будем).

Главное преимущество электрохимических источников (см. *Приложение 2*) — мобильность, в чем им замены нет. Главный недостаток — они не обеспечивают долговременной эксплуатации для подавляющего большинства электронных приборов, за исключением специально спроектированных малопотребляющих (вроде наручных часов) либо включающихся на непродолжительное время (пульты управления бытовой техникой) устройств. А для таких изделий, как плееры, цифровые фотоаппараты, мобильные телефоны и ноутбуки, емкость электрохимических источников явно недостаточна, к тому же общий срок службы их оставляет желать лучшего. Так что масса неудобств, которые приходится испытывать пользователям, есть вынужденная плата за мобильность. И одно из самых серьезных ограничений — отсутствие унификации зарядных устройств, хотя бы для аккумуляторов одного типа. Лично мне приходится таскать с собой в деревню и обратно пять типов зарядных устройств (два для разных мобильников, одно для фотоаппарата, одно для карманного компьютера и одно для шуруповерта), а ведь я далеко не самый «мобильный» из своих знакомых. Правда, положение потихоньку

выправляется — по крайней мере для мобильных телефонов и КПК зарядники постепенно унифицируются, хотя и недостаточно быстрыми темпами.

Остальные варианты мобильными не являются, и носят общее название вторичных источников питания, потому что они преобразуют энергию бытовой электросети в нужное напряжение постоянного тока. Главное преимущество импульсных источников — экономичность и значительно лучшие массогабаритные характеристики по сравнению с трансформаторными источниками. Поэтому практически все стационарные современные бытовые приборы снабжаются именно такими источниками — компьютеры, телевизоры, музыкальные центры и т. д. Главный их недостаток — сложность конструкции и вытекающая отсюда относительно высокая стоимость. Как правило, их целесообразно применять для относительно мощных приборов, с энергопотреблением 50—100 Вт и выше. Если вы попробуете создать импульсный источник, рассчитанный на 5—10 Вт, то вы в габаритах, стоимости и надежности скорее всего проиграете, даже с использованием серийно выпускающихся модулей.

Самостоятельно конструировать, изготавливать и настраивать импульсные источники принципиально сложнее обычных. В конце главы я приведу конструкцию небольшого самодельного импульсного преобразователя напряжения, но на практике в 99,9% случаев всегда можно найти подобный серийно выпускающийся аналог. А так мы в основном ограничимся обычными трансформаторными источниками с аналоговым регулированием. Кстати, импульсные источники тоже в большинстве своем содержат трансформатор, но он не является определяющим элементом.

Упомянем еще об одной альтернативе, которая была весьма модной в радиолюбительских кругах в советские времена — бестрансформаторные источники питания от сети. Вы можете наткнуться на нечто подобное, если перелистаете старые журналы «Радио». В связи с этим следует сказать только одно.

Никогда без крайней нужды не стройте прибора, работающего от сети переменного тока без трансформатора!

Это опасно для жизни — ваша схема будет всегда находиться под высоким напряжением относительно земли (без кавычек — т. е. водопроводных труб, батарей отопления и т. п.). Если ваша схема предназначена для управления мощной сетевой нагрузкой, то это управление следует обязательно осуществлять через гальванически развязывающие элементы — реле, электронные реле, трансформаторы и т. п., в остальных случаях в бестрансформаторных конструкциях нет никакой нужды.

Трансформаторы

Независимо от конкретной конструкции, трансформаторы всегда устроены по одной схеме: на замкнутом каркасе из металлических пластин или ленты находятся несколько обмоток. Самые распространенные разновидности трансформаторов — с Ш-образным и тороидальным сердечником схематично показаны на рис. 4.1. Если есть возможность, то лучше выбрать тороидальный трансформатор, т. к. у него меньшее магнитное поле рассеяния. В случае чего на него можно домотать недостающие обмотки или добавить витков к имеющимся. При выборе трансформатора следует предпочесть те, которые залиты компаундом (в старинных конструкциях употреблялся просто парафин). По крайней мере, катушка с обмотками должна прочно, без люфта, держаться на стержне, а сами пластины должны быть обязательно плотно сжаты специальной скобой (естественно, это относится в первую очередь к Ш-образным трансформаторам). Иначе трансформатор неизбежно будет во время работы гудеть.

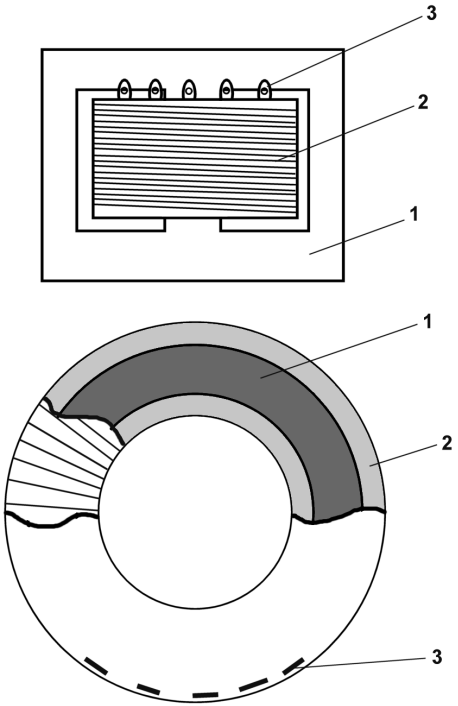


Рис. 4.1. Трансформаторы с Ш-образным и тороидальным сердечником:
1 — сердечник; 2 — обмотки; 3 — выводы обмоток

Одна из обмоток называется первичной — так как мы рассматриваем сетевые трансформаторы, то она всегда рассчитана на сетевое напряжение. Найти ее, если характеристики обмоток неизвестны, не очень сложно — она всегда имеет наибольшее сопротивление из всех, причем для малогабаритных трансформаторов это сопротивление может достигать сотен и даже тысяч ом. Иногда она поделена на две, которые перед включением нужно соединить (конец к началу), может иметь отводы для более точной подгонки напряжений или для обеспечения возможности переключения 220/120 В. Сравнивая сопротивления выводов между собой, можно найти эти отводы. Другой способ определения первичной обмотки — она всегда намотана наиболее тонким проводом (вообще, чем толще провод, тем меньше напряжение на обмотке, как мы увидим далее).

Остальные обмотки — вторичные, их можно соединять между собой в любой комбинации. Каждая обмотка имеет начало и конец. Для суммирования напряжений обмоток надо соединять конец одной обмотки с началом другой. Смысл понятий начала и конца обмоток очевиден: где начинали мотать обмотку, там начало. Если намотать следующую обмотку *в том же направлении* (а так всегда и поступают), то у нее начало будет там же, где и у первой. Если это фабричный трансформатор и выводы у него пронумерованы, то нечетные выводы принимаются за начала обмоток, а четные — за концы, т. е. при соединении двух обмоток с нумерацией выводов 1—2 и 5—6 для сложения напряжений нужно соединить вывод 2 первой обмотки с выводом 5 второй (или вывод 1 первой с выводом 6 второй), оставшиеся выводы 1—6 (или 5—2) будут, соответственно, началом и концом объединенной обмотки. Для серийно выпускающихся трансформаторов, а также у торгующих ими организаций имеются справочники по типовым разновидностям с указанием характеристик обмоток и нумерации их выводов.

Я надеюсь, что вам никогда не придется самим мотать сетевые трансформаторы, так что приведу только главное соотношение (его можно назвать «законом трансформатора»):

$$U_1/U_2 = n_1/n_2,$$

где U_1, U_2 — напряжение первичной и вторичной обмоток, n_1 , n_2 — число витков первичной и вторичной обмоток, соответственно.

Как видите, все необычайно просто. Если, скажем, первичная обмотка имеет 220 витков (это должен быть довольно мощный трансформатор, у маломощных число витков может составлять несколько тысяч), а вторичная — 22 витка, то при подключении к сети 220 В на вторичной обмотке будет 22 вольта. Токи находятся в обратном соотношении: если ток такой вторичной обмотки составляет 1 А, то первичная обмотка будет потреблять от сети 100 мА. Если

вторичных обмоток несколько, то для определения потребления тока от сети их токи нужно пересчитать на первичную обмотку в отдельности (число витков при этом знать необязательно, достаточно только напряжения), а затем сложить. Можно пойти и другим путем — суммировать мощности, потребляемые вторичными обмотками (которые равны произведениям токов на напряжения), а затем поделить полученную сумму на 220, в результате получим ток в первичной обмотке.

Заметки на полях

Кстати, из этого закона вытекает простой метод определения числа витков в обмотках трансформатора, если это зачем-то нужно: намотайте поверх имеющихся обмоток несколько витков любого провода, включите трансформатор и измерьте напряжение на этой импровизированной обмотке. Поделив число намотанных витков на полученное значение напряжения, вы определите величину числа витков на один вольт, которая едина для всех обмоток, а далее пересчитать полученный результат уже не составляет трудностей.

При определении напряжений вторичных обмоток учтите, что их нужно выбирать с запасом (это относится и к покупным, и к самодельным трансформаторам), поскольку под нагрузкой напряжение «садится», и это «просаживание» тем больше, чем меньше мощность трансформатора. Если вам задано минимально допустимое напряжение 7 В — выбирайте трансформатор с 9—12-вольтовой обмоткой, не ошибетесь. Мощность трансформатора можно подсчитать, если известно сечение его магнитопровода (для Ш-образных трансформаторов это сечение центрального стержня, на котором находится катушка с обмотками, для тороидального — просто поперечное сечение тора), по формуле $S = 1{,}15 \cdot \sqrt{P}$, где S — сечение в см^2, P — мощность в Вт.

Простейший нестабилизированный источник питания

Схема простейшего источника питания приведена на рис. 4.2. Именно по такой схеме устроены практически все распространенные ныне блоки питания, встроенные в сетевую вилку. Иногда в них вторичная обмотка имеет несколько отводов, и присутствует ползунковый переключатель, который коммутирует эти отводы, меняя выходное напряжение. Так как эти блоки весьма дешевы, то если вам не требуется большой мощности, спокойно можно покупать такой блок, разбирать его и встраивать в вашу аппаратуру (или даже не встраивать — хотя, на мой вкус, громоздкие «надолбы» на розетках отнюдь не украшают интерьер, все время хотят вывалиться и к тому же не во всякую розетку влезают). Нужно только обратить внимание на допустимый

ток, который указан на корпусе такого блока. Что касается номинального напряжения, то этот вопрос мы сейчас рассмотрим чуть подробнее.

Как работает эта схема? Здесь переменный синусоидальный ток со вторичной обмотки трансформатора (II) подается на конструкцию из четырех диодов, которая называется диодным мостом и представляет собой *двухполупериодный выпрямитель* (есть и другие способы двухполупериодного выпрямления, но этот самый распространенный). В мосте могут быть использованы любые типы выпрямительных диодов, лишь бы предельно допустимый ток их был не меньше необходимого (для указанных на схеме 1N4001 это 1 А), а предельно допустимое напряжение было не меньше половины амплитудного значения входного переменного напряжения (т. к. в данном случае это всего 7 В, то здесь подходят вообще все выпрямительные диоды). Мало того, такие мосты выпускаются уже в сборе, в одном корпусе, на котором иногда даже нарисовано, куда подключать переменное и откуда снимать постоянное напряжения (типичный пример из отечественных — КЦ407А).

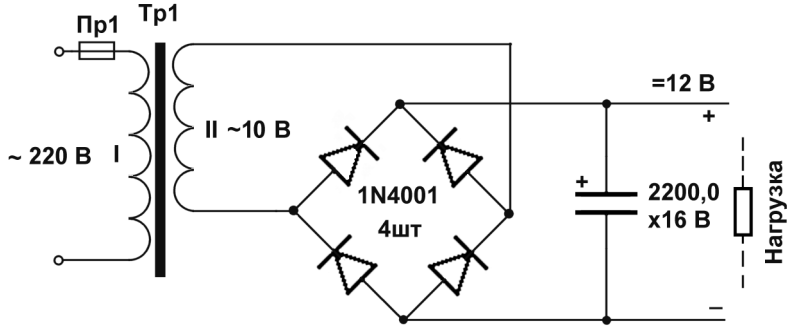


Рис. 4.2. Простейший нестабилизированный однополярный источник питания

Проследим за работой моста. Предположим, что на верхнем по схеме выводе вторичной обмотки в данный момент напряжение выше, чем на нижнем. Тогда ток в нагрузку (на рис. 4.2 она обозначена пунктиром) потечет через правый верхний диод моста, а возвратится в обмотку через левый нижний. Полярность на нагрузке, как видим, соблюдается. В следующем полупериоде, когда на верхнем выводе обмотки напряжение ниже, чем на нижнем, ток через нагрузку потечет, наоборот, через левый верхний диод и возвратится через правый нижний. Как видим, полярность опять соблюдается. Отсюда и название такого выпрямителя: двухполупериодный, т. е. он работает во время обоих полупериодов переменного тока.

Форма напряжения на выходе такого моста (в отсутствие конденсатора) соответствует пульсирующему напряжению, показанному на рис. 2.5, *а*. Естественно, такое пульсирующее напряжение нас не устраивает, мы хотим иметь настоящее постоянное напряжение без пульсаций, потому в схеме присутствует сглаживающий (фильтрующий) конденсатор, который вместе с выходным активным сопротивлением трансформатора и сопротивлением диодов представляет собой не что иное, как известный нам по *главе 2* интегрирующий фильтр низкой частоты. Все высокие частоты отфильтровываются, а на выходе получается «ровное» постоянное напряжение.

К сожалению, такая идиллия имеет место только при отсутствии нагрузки, к чему мы вернемся чуть позже, но сначала попробуем определить, какова величина постоянного напряжения на выходе фильтра.

В отсутствие нагрузки конденсатор с первых же полупериодов после включения питания заряжается до амплитудного значения пульсирующего напряжения, которое равно амплитудному значению напряжения на вторичной обмотке за вычетом падения напряжения на *двух* диодах, стоящих на пути тока. Так как в установившемся режиме через эти диоды ток весьма мал (только для подпитки собственных токов утечки конденсатора и диодов), то и падение напряжения на них мало и близко к нулю. Амплитудное значение напряжения на вторичной обмотке равно $10 \cdot \sqrt{2}$ = 14,1 В, так что на холостом ходу напряжение на выходе источника практически равно 14 В. Почему же на схеме написано 12 В?

При подключении нагрузки происходит сразу много всего. Во-первых, снижается напряжение на вторичной обмотке, поскольку трансформатор имеет конечную мощность. Во-вторых, увеличивается падение напряжения на диодах, которое может при максимально допустимом для них токе достигнуть 1 В на каждом. В-третьих, и в главных, во время «провалов» пульсирующего напряжения нагрузка питается только за счет того, что через нее разряжается конденсатор. Естественно, напряжение на нем при этом каждый раз немного снижается. Поэтому график выходного напряжения при подключенной нагрузке представляет собой уже не ровную постоянную линию, а выглядит примерно так, как показано на рис. 4.3 (причем снижение входного напряжения за счет «просаживания» трансформатора здесь не учитывается). Таким образом, выходное напряжение немного пульсирует — тем больше, чем больше ток в нагрузке, и тем меньше, чем больше емкость конденсатора. Именно поэтому в источниках применяют электролитические конденсаторы столь большой емкости. Наличие пульсаций также снижает постоянную составляющую выходного напряжения.

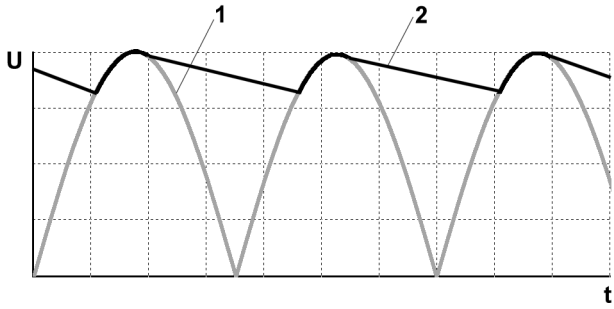


Рис. 4.3. Вид пульсаций на выходе нестабилизированного источника:
1 — исходное пульсирующее напряжение в отсутствие фильтрующего конденсатора;
2 — выходное напряжение при наличии фильтрующего конденсатора и нагрузки

Заметки на полях

В данной схеме избавиться от этих пульсаций полностью невозможно, как бы вы ни увеличивали емкость. Кстати, а как подсчитать нужную емкость? В принципе, это возможно, если задаться необходимым уровнем пульсаций, но мы здесь приведем только эмпирическое и весьма приблизительное правило: на каждый ампер нагрузки достаточно конденсатора от 1000 до 2200 мкФ. Первая величина ближе к тому случаю, когда на выходе такого источника планируется поставить стабилизатор напряжения, вторая — если такого стабилизатора не предполагается. Может показаться, что увеличением емкости конденсатора при заданной нагрузке можно в конце концов избавиться от пульсаций вообще, однако вы легко установите на практике, что увеличение емкости сверх некоторого значения далее пульсаций уже не снижает, помочь может только стабилизатор.

Указанные причины совместно приводят к тому, что под нагрузкой маломощные источники (типа тех, что со встроенной вилкой) могут выдавать в полтора-два раза меньшее напряжение, чем на холостом ходу. Поэтому не удивляйтесь, если вы приобрели такой блочок с указанным на шильдике номинальным напряжением 10 В, а мультиметр на холостом ходу показывает аж все 18!

Чтобы завершить описание простейшего источника, нужно сказать пару слов об указанном на схеме (см. рис. 4.2) предохранителе Пр. В упомянутых блоках со встроенной вилкой предохранитель часто отсутствует, и это вызвано, кроме стремления к удешевлению устройства, очевидно, тем обстоятельством, что маломощный трансформатор сам служит неплохим предохранителем — провод первичной обмотки у него настолько тонок, и сопротивление его настолько велико, что при превышении допустимого тока обмотка довольно быстро сгорает, отключая весь блок. (После чего его, естественно, остается только выбросить.) Но в стационарных устройствах и тем более в источниках большей мощности предохранитель должен быть обязательно.

Обычно его выбирают на ток в два-четыре раза больший, чем расчетный максимальный ток первичной обмотки.

Приведем еще одну полезную схему нестабилизированного источника, на этот раз двуполярного, т. е. выдающего два одинаковых напряжения относительно средней точки — «земли» (рис. 4.4). В принципе, она пояснений не требует, потому что очень похожа на однополярную, только возврат тока в обмотки от обеих нагрузок происходит непосредственно через общую «землю», минуя диодный мост. В качестве упражнения предлагаю вам самостоятельно разобраться, как работает эта схема. Вторичные обмотки (II и III) здесь, в сущности, представляют собой две одинаковые половины одной обмотки. Жирными точками около вторичных обмоток обозначены их начала, чтобы не перепутать порядок их соединения, если их наматывали раздельно.

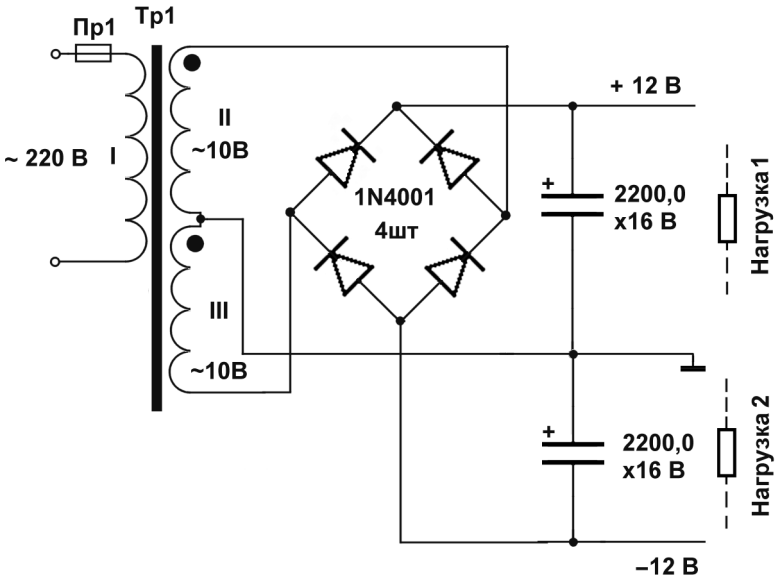


Рис. 4.4. Нестабилизированный двуполярный источник питания

Стабилизаторы

Простейший стабилизатор — это стабилитрон, который мы упоминали в *главе 3*. Если параллельно ему подключить нагрузку (рис. 4.5, *а*), то напряжение на ней будет стабилизировано до тех пор, пока ток через нее не будет слишком велик. Рассчитать работу этой схемы можно так: в отсутствие стабили-

трона напряжение в средней точке делителя из $R_{ст}$ (оно равно 200 Ом, как вы, наверное, догадались, т. к. при обозначении на схемах омы в большинстве случаев опускают, см. *главу 5*) и $R_н$ должно превышать номинальное напряжение стабилизации стабилитрона $U_{ст}$, иначе при его подключении ток через него не пойдет и стабилитрон не откроется. Так что максимальный ток, который мы можем получить в такой схеме, не превышает нескольких десятков миллиампер — в зависимости от мощности стабилитрона. Такой стабилизатор называют еще *параметрическим*.

Подробности

Вы зададите вопрос — а зачем здесь конденсатор? Ведь в нестабилизированном источнике, который мы рассмотрели ранее, и откуда поступает напряжение на этот стабилизатор, один фильтрующий конденсатор уже имеется, не так ли? Ответ простой: на выходе всех типов стабилизаторов всегда ставится конденсатор. Он позволяет сгладить наличие остаточных пульсаций, которые все равно просочатся на выход, т. к. стабилитрон имеет свое дифференциальное сопротивление, и при изменении входного напряжения или тока в нагрузке напряжение на нем также будет меняться, хоть и в значительно меньшей степени. Величина емкости здесь может быть значительно меньше, чем на выходе выпрямительного моста. Для интегральных стабилизаторов, которые мы будем рассматривать далее, установка конденсатора положена по рекомендациям производителя (и на входе, и на выходе) — иначе сложные внутренние схемы таких стабилизаторов с обратными связями могут «гудеть» — самовозбуждаться.

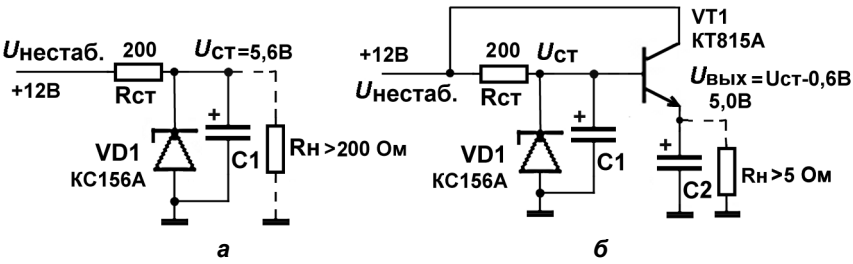


Рис. 4.5. Два параметрических стабилизатора:
а — самый простой на стабилитроне; *б* — с эмиттерным повторителем

Значительно интересней схема на рис. 4.5, *б*. Здесь транзистор включен эмиттерным повторителем (см. *главу 3*), который, во-первых, имеет высокое входное сопротивление (поэтому ток через стабилитрон практически не зависит от изменений тока в нагрузке), во-вторых, служит усилителем тока, т. е. мощностные возможности здесь определяются только транзистором. Конденсаторов здесь целых два: первый помогает сглаживать пульсации на стабилитроне, второй — дополнительно оставшиеся пульсации на выходе транзистора.

Подробности

Давайте попробуем рассчитать для простейшей параметрической схемы (рис. 4.5, *а*) т. н. *коэффициент стабилизации*: отношение изменения входного напряжения (в %) к изменению выходного (также в %). Для этого надо посмотреть в справочнике величину *дифференциального сопротивления* стабилитрона: для указанного КС156А — 46 Ом. Это означает, что при изменении тока через него на 1 мА изменение напряжения стабилизации составит 46 мВ. Теперь предположим, что входное напряжение изменяется на 1 В (8,3%), тогда изменение тока будет равно 1 В/200 Ом = 5 мА, отсюда изменение выходного напряжения будет 46·5 = 230 мВ или 4,6%. Коэффициент стабилизации тогда будет равен 8,3/4,3 ≈ 2. Конечно, это очень маленькая величина, потому простейшие параметрические стабилизаторы ставят только в редких случаях, когда входное напряжение дополнительно стабилизировано заранее.

Выходное сопротивление простейшего стабилизатора очень велико, поэтому выходное напряжение будет «гулять» независимо от входного при изменении тока нагрузки, от которого напрямую зависит ток через стабилитрон. Другое дело — схема на рис. 4.5, *б*, в которой ток через стабилитрон изменяется на величину β транзистора меньшую, чем ток в нагрузке. Статический коэффициент передачи тока для транзистора КТ815А равен (по справочнику) 40, поэтому при изменении тока нагрузки на 1 мА, ток через стабилитрон изменится всего на 0,025 мА, а напряжение стабилизации, соответственно, всего на 1,15 мВ, а не на 46 мВ, как ранее. Теоретический коэффициент стабилизации этой схемы по входному напряжению равен приблизительно 70. На практике стабилизирующие свойства данной схемы оказываются несколько хуже, т. к. следует учитывать нестабильность падения напряжения «база-эмиттер» транзистора.

При этом надо учитывать ограничения, накладываемые минимальным током через стабилитрон (5 мА для КС156А) и его максимальной допустимой мощностью (300 мВт). При выходном токе 1 А базовый ток транзистора должен составить не менее 25 мА, поэтому общий ток через резистор $R_{ст}$ не может быть меньше 30 мА (что и дает значение 200 Ом при минимальной разности напряжений «вход-выход» ~6 В). Максимально возможный выходной ток в такой схеме ~2 А, потому что минимальное значение $R_{ст}$ = 100 Ом. При отсутствии нагрузки ток через стабилитрон составит тогда 60 мА, а выделяющаяся на нем мощность при напряжении стабилизации ~5 В как раз и составит 0,3 Вт.

Да, кстати, а какая мощность выделится на «проходном» транзисторе VT1? Не такая уж и маленькая: при выходном токе 1 А она составит (12 В – 5 В) · 1 А = целых 7 Вт! Значит, транзистор явно придется ставить на радиатор. Отсюда виден главный недостаток подобных аналоговых стабилизаторов — низкий КПД. В данном случае он всего около сорока процентов (проверьте!), остальное рассеивается в пространстве. Мы можем его

несколько повысить, снижая входное напряжение, но только до определенного предела. Здесь этот предел равен примерно 8 В, иначе эта схема не справится. Помните, однако, что 8 В — это действительно нижний предел, а не среднее значение пульсирующего напряжения на выходе конденсатора фильтра, которое показывает вольтметр (если вы еще раз взглянете на рис. 4.3, то поймете о чем я). Иначе стабилизатор просто перестанет стабилизировать. Потому всегда следует иметь запас, и не маленький.

Заменой *n-p-n*-транзистора на *p-n-p* с соответствующей сменой всех полярностей (в том числе «переворотом» конденсаторов и стабилитрона) на обратные, мы получим стабилизатор отрицательного напряжения. На практике, однако, такие стабилизаторы давно уже не применяют. Гораздо более высокий коэффициент стабилизации, как по входному напряжению, так и по изменению тока нагрузки, дают интегральные стабилизаторы, которые к тому же гораздо проще в обращении.

Интегральные стабилизаторы

Совершенно естественным ходом мысли разработчиков было бы упаковать типовой узел, состоящий из стабилитрона, транзистора и резистора в одну микросхему. Однако выдающийся схемотехник и разработчик аналоговых микроэлектронных устройств Р. Видлар, о котором мы еще вспомним в связи с изобретением интегрального операционного усилителя, рассудил иначе. Действительно, такая простейшая схема обладает целым рядом недостатков, о которых мы говорили в предыдущем разделе. Для повышения коэффициента стабилизации наилучшим выходом было бы использовать принцип отрицательной обратной связи, с которым мы познакомимся в *главе 6*. Схему со стабилизирующей обратной связью не особенно трудно построить и на дискретных транзисторах, но с увеличением качества ее сложность и, соответственно, стоимость резко возрастают. А вот в производстве микросхем почти безразлично — пять транзисторов они содержат или тридцать пять. Кроме того, там все транзисторы находятся на одном кристалле, имеют одинаковую температуру и близкие характеристики, что недостижимо в дискретных схемах. Видлар этим воспользовался и сконструировал микросхему μA723, которая положила основу современным семействам интегральных стабилизаторов.

Наиболее широко распространена и доступна серия стабилизаторов LM78/79xx разработки фирмы National Semiconductor (имейте в виду, что семейство LM содержит и другие типы микросхем, и это название не должно вас смущать). Выпускается они сейчас очень многими производителями, тогда буквы могут быть другими, но цифры остаются теми же. Эти цифры означают вот что: первые две — наименование серии (78 — стабилизатор

положительного напряжения, 79 — отрицательного), вторые две — напряжение стабилизации (например, 7805 — стабилизатор напряжения +5 В). Выпускаются аналоги этой серии и в России, однако принцип наименования другой — это серия 142ЕНхх и др. Напряжения стабилизации в этой серии фиксированы, однако имеются и регулируемые стабилизаторы.

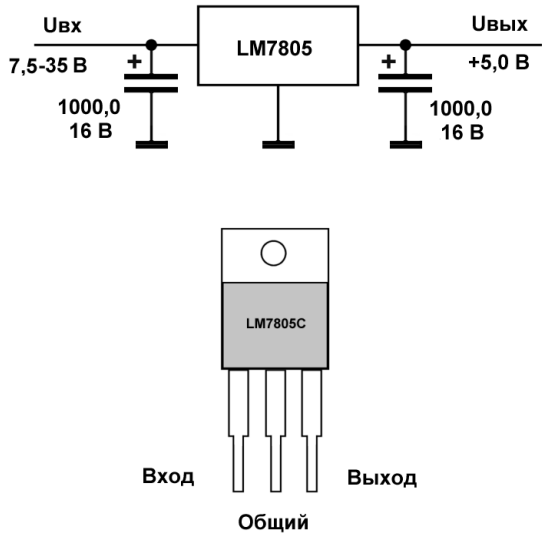


Рис. 4.6. Схема включения интегрального стабилизатора

На рис. 4.6 приведена типовая схема включения такого стабилизатора и вариант его внешнего вида. В корпусе TO-220, как на рисунке, такой стабилизатор может выдать ток до 2,4 А, если рассеиваемая мощность не превышает 20 Вт (с радиатором, см. *главу 5*). Но есть большой выбор и других корпусов, включая корпуса для поверхностного монтажа. Особенно удобен маленький корпус TO-92 (тогда в название вклинивается буква L: 78L05) — он позволяет стабилизировать питание отдельных узлов независимо друг от друга, избегая таким образом их взаимного влияния.

Коэффициент стабилизации по входному напряжению у серии LM равен приблизительно 100, а выходное напряжение меняется не более, чем на 1% при изменении нагрузки от минимальной (1—5 мА потребления) до максимально допустимой. Разумеется, серия 78/79xx — не единственная в своем роде, есть и другие, аналогичные по функциональности, среди них стоит отметить LM2931 — серию пятивольтовых стабилизаторов разной мощности, отличающуюся малым собственным потреблением (доли миллиампера) и, главное, способностью работать при предельно низких входных напряжени-

ях — всего на 0,2 В превышающих выходное (у LM78xx входное напряжение должно быть не менее, чем на 2 вольта выше выходного).

Кроме рассмотренных *линейных* устройств, существуют также преобразователи постоянного (DC/DC) напряжения, которые работают на эффекте умножения напряжения на конденсаторах (см. рис. 2.10, *б*) или аналогичном эффекте с использованием индуктивности.

В качестве примера приведу простейшие нестабилизированные модули фирмы Traco Power, которые часто применяют для получения двуполярного напряжения из однополярного. Так, одноваттный модуль TSM 0505D при входном напряжении 5 В±10% выдает два напряжения ±5 В при токе нагрузки до 100 мА, чего с большим запасом достаточно для питания нескольких операционных усилителей. Более сложные (и дорогие) преобразователи могут иметь стабилизированный выход, скажем, изделия серии TMR 0521 выдают на выходе те же два напряжения ±5 В (при токе нагрузки до 200 мА), но при входном напряжении от 4,5 до 9 В. Преобразователи Traco имеют полную гальваническую развязку вход-выход и довольно популярны, но характеристики их оставляют желать лучшего: особенно неприятным свойством этих конверторов является их работоспособность в ограниченном диапазоне мощности нагрузки (при снижении сопротивления нагрузки до нуля преобразователь практически перестает работать). Вариант использования подобных преобразователей для построения маломощного двуполярного источника приведен в *главе 17*.

Импульсные источники питания

Идея всех *импульсных источников* питания состоит в том, что при повышении частоты резко снижаются габариты трансформатора, и его можно изготовить, например, с ферритовым сердечником, который решительно не работает на промышленной частоте 50 Гц. Переменное напряжение при этом приходится формировать искусственно, что заметно усложняет схему, а определяющим габаритным фактором станет не трансформатор, а радиаторы ключевых переключающих элементов, функцию которых обычно выполняют MOSFET-транзисторы. КПД всего источника при этом заметно растет, и чем он мощнее — тем в большей степени.

Для сетевых импульсных источников питания целесообразно применять готовые модули (AC/DC-преобразователи), например, преобразователь CFM-2001S фирмы FABRIMEX (Швейцария) стоит около 30 долл. и при входном переменном напряжении от 85 до 264 В выдает на выходе постоянное напряжение 5 В при нагрузке до 4,4 А (более 20 Вт). Для целей DC/DC-преобразования также имеются готовые модули, но они не всегда обеспечи-

вают удовлетворительные характеристики, потому имеет смысл рассмотреть построение подобных преобразователей самостоятельно.

Самодельный импульсный преобразователь

Сейчас мы рассмотрим, как можно самостоятельно построить стабилизированный импульсный источник — преобразователь напряжения. Это может понадобиться на практике, если требуются нестандартные (например, повышенные) напряжения, кроме того, наш источник полностью разделяет (гальванически *развязывает*) входную и выходную цепи. Схема получится довольно громоздкая (хотя и не слишком большая по габаритам), и заниматься ее конструированием и отладкой стоит лишь в случае крайней необходимости. Однако характеристики такого преобразователя могут быть довольно высокими — по крайней мере, не хуже готовых изделий, и показанная схема хорошо иллюстрирует принципы работы такого рода устройств.

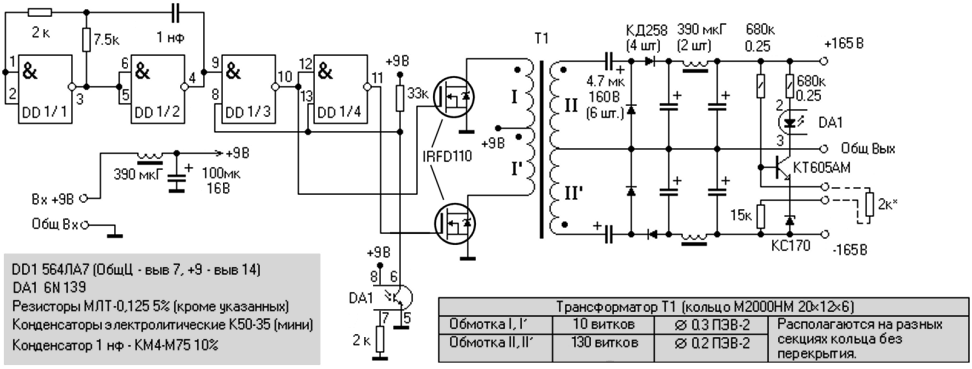


Рис. 4.7. Схема импульсного преобразователя с гальванической развязкой входа и выхода

Принципиальная схема преобразователя приведена на рис. 4.7. Он преобразует входное напряжение +9 В в два высоких напряжения ±165 В. Я специально выбрал такой крайний случай, далее я покажу, как изменением всего нескольких параметров схемы получить на выходе практически любую пару симметричных напряжений. Общая максимальная мощность схемы приблизительно 4 Вт (при указанном выходном напряжении максимальный нагрузочный ток до 12 мА по каждому из выходов). Она может быть повышена, если малогабаритные MOSFET-транзисторы IRFD110 заменить на более мощные (например, IRFZ44) и установить их на радиаторы. К сожалению, сильно снижать входное напряжение в данной схеме нельзя (не будут работать транзисторные MOSFET-ключи), а вот повышать (за счет некоторого

снижения КПД) можно, особенно при установке более мощных транзисторов. Реально данная схема при указанных на схеме элементах работает приблизительно в диапазоне входного напряжения от 8 до 12 В (при этом выходное остается равным номинальному с точностью примерно 2,5%).

Рассмотрим работу схемы. Единственный компонент, который мы еще не «проходили», — это логическая КМОП-микросхема К561ЛА7. Рассматривать мы ее будем в *главе 8*, а генератор прямоугольных импульсов, который на ней построен, — в *главе 9*. Сейчас нам достаточно знать, что она содержит внутри четыре логических элемента, и на выходе элементов D1/3 и D1/4 образуются противофазные прямоугольные импульсы, которые поочередно открывают транзисторные ключи с частотой примерно 60 кГц. В результате на вторичных обмотках трансформатора образуется высокое напряжение, которое дополнительно умножается вдвое на системе из диодов КД258, конденсаторов 4,7 мкФ и индуктивностей (дросселей) 390 мкГн.

Стабилизирующая часть схемы построена на приборе 6N139, который имеет внутри довольно сложную конструкцию и представляет собой транзисторный оптрон — подавая на вход (выводы 2, 3) напряжение, мы открываем гальванически развязанный от входа транзистор, и тогда на выходе (вывод 6) получаем напряжение, практически равное нулю. В результате все вместе работает так: если выходное напряжение схемы недопустимо повысилось, то ключ на транзисторе КТ605АМ открывается, на выходе оптрона появляется близкое к нулю напряжение, логические элементы D1/3 и D1/4 при этом запираются, и на ключи ничего не подается. Напряжение на выходе снижается, ключ КТ605АМ запирается, напряжение на выходе оптрона становится близким к напряжению питания, и импульсы опять поступают на трансформатор.

Трансформатор намотан на ферритовом кольце с характеристиками, указанными на схеме. Обмотки наматываются медным обмоточным проводом ПЭВ-2 парами совместно, причем обратите внимание, что у входной пары обмоток соединен конец одной с началом другой, а у выходной — начала обеих обмоток. Подбором дополнительного резистора 2 кОм (на схеме помечен звездочкой и соединен пунктиром) выходное напряжение устанавливается более точно. Дроссель по питанию +9 В (390 мкГн) служит для защиты внешних сетей от помех (см. *главу 5*). Учтите, что схема довольно заметно «фонит» в радиодиапазоне, потому ее надо заключать в металлический экран, который должен быть соединен с входной (обозначенной на схеме, как «Общ. Вх») «землей» в одной точке, вблизи входного контакта на плате.

Для того чтобы поменять выходное напряжение, следует, во-первых, изменить коэффициент резистивного делителя в базе ключа на КТ605АМ. При этом, конечно, надо снижать номинал верхнего по схеме резистора (680 кОм),

а не повышать — нижнего (15 кОм). Например, при выходном напряжении ±24 В номинал верхнего резистора должен составлять примерно 75—82 кОм. Но для хорошей работы преобразователя этого изменения недостаточно — для получения максимального КПД необходимо также изменить число витков во вторичных обмотках. Рассчитывать их следует так: желаемое выходное напряжение нужно умножить на коэффициент 1,3, затем полученную величину поделить на 9 (входное напряжение) и умножить на 10 (число витков в первичной обмотке). Например, при выходном напряжении, равном ±24 В, число витков в каждой из вторичных обмоток должно быть равно 35 (при этом и вторичную, и первичную обмотки можно намотать более толстым проводом). При пониженном выходном напряжении можно упростить схему, отказавшись от умножителя напряжения (убрав последовательно включенные конденсаторы, подключив диоды по схеме рис. 4.4 и увеличив соответственно число витков вторичной обмотки), при этом КПД повысится.

Подробности

Зачем в схеме обсуждаемого преобразователя вообще умножитель напряжения? Если вы проанализируете процессы, происходящие в трансформаторе, то обнаружите, что действующее значение напряжения на первичной обмотке равно напряжению питания — т. е. 9 В. Итого, чтобы получить после выпрямления и фильтрации значение напряжения 165 В, нам понадобилось бы как минимум $10 \cdot 165/9 \approx 180$ витков в каждой вторичной обмотке, а с запасом на потери и регулирование примерно на 20—30% больше, т. е. около 240. Такое число витков (в сумме около 500) намотать на кольце диаметром 20 мм физически сложно. А когда мы снижаем требования к напряжению, число витков уменьшается и умножитель, который отрицательно сказывается на КПД устройства, можно убрать.

Главным недостатком данной схемы с точки зрения КПД, однако, является не умножитель, а форма сигнала на первичных обмотках. Так как включение одного ключа и выключение другого совпадают во времени, существует момент, когда через обе обмотки течет сквозной ток. Это очень плохо сказывается на КПД устройства и ведет к излишним потерям на нагрев транзисторов. Для небольших мощностей, как здесь, этим эффектом можно пренебречь, но для больших его приходится учитывать и разносить моменты включения одного ключа и выключения другого во времени. Это делается обычно с помощью специализированных микросхем для управления ключами, хотя их несложно сымитировать на любом микроконтроллере.

Как правильно питаться

Общая схема грамотной разводки питания между источниками и потребителями в электронных устройствах приведена на рис. 4.8, *а*. На практике, если источник расположен в отдельном корпусе, то указанной на блок-схеме общей

точкой соединения «земли» служит выходная клемма «минус» этого корпуса. Если же вся конструкция — и источники и нагрузки — представляет собой набор плат в едином корпусе, то за общую точку удобно выбрать, скажем, минусовой вывод основного фильтрующего конденсатора.

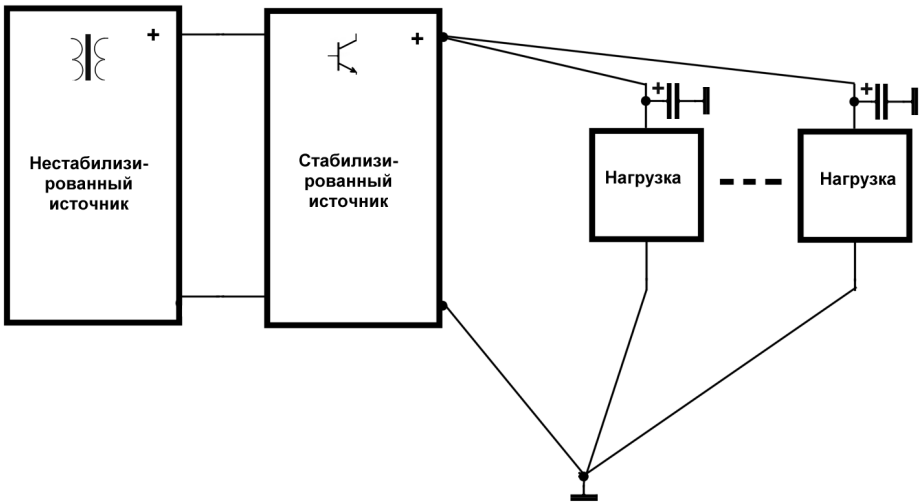


Рис. 4.8. Схемы разводки питания между источниками и потребителями

Смысл такой разводки заключается в том, чтобы токи от разных потребителей не протекали по одному и тому же проводу, поскольку это может вызвать их взаимное влияние и другие нежелательные явления. Характерный эффект под названием «захват частоты» можно наблюдать, если на двух разных, но с общим питанием, платах имеются генераторы (не кварцевые), работающие на близких или кратных частотах — вдруг по непонятным причинам они начинают работать на одной и той же частоте! Иногда от этого очень трудно избавиться, поэтому лучше сразу делать все правильно. Если же по каким-то причинам идеала по образцу рис. 4.8 достичь не получается (как в подавляющем большинстве практических случаев), то для нагрузки как можно ближе к выводу питания устанавливают т. н. «развязывающие» конденсаторы (они показаны на рис. 4.8). Причем если это отдельная плата, то конденсаторы ставят на ней, прямо около входного разъема, ни в коем случае не в дальнем конце платы! Кроме того, во всех случаях провода и проводники питания на плате должны быть как можно толще — если провод тонкий, то на нем самом за счет протекающего тока происходит падение напряжения, и разные потребители оказываются под разными потенциалами как по «земле», так и по питанию.

Заметки на полях

Кстати, о «земле» — почему я ее все время заключаю в кавычки? Дело в том, что в электротехнике существует совершенно определенное понятие земли — когда нечто находится под потенциалом земной поверхности, который принимается за истинный ноль напряжения. Под таким потенциалом по понятным причинам находятся, например, водопроводные трубы. Есть еще понятие «нулевого провода» (один из проводов в вашей домашней розетке всегда нулевой, второй называется «фазным») — теоретически он тоже находится под потенциалом земли, но практически соединяется (возможно) с истинной землей только где-то на электростанции, а за счет несбалансированности протекающего по нему тока потенциал его может «гулять», и довольно сильно. Поэтому правильно организованная бытовая электросеть всегда должна включать в себя третий провод, который будет истинным заземлением. Если у вас такого третьего провода нет (печально, но в нашей стране до сих пор строили именно так, и только в последние годы положение начинает выправляться), то в принципе его можно организовать путем присоединения к водопроводной трубе (СНиПы это допускают). Но это не только неудобно (представляете, сколько проводов придется растаскивать по всей квартире?), но иногда и опасно, т. к. в случае попадания фазного напряжения на такое «заземление», предохранитель не сразу сработает из-за наличия сопротивления между трубой и землей и кого-нибудь может основательно «тряхнуть», если в соседней квартире в этот момент мыть руки под краном. Если же вернуться к нашей схемотехнической «земле», то самое правильное называть ее «общим проводом», просто термин прижился, да и звучит короче.

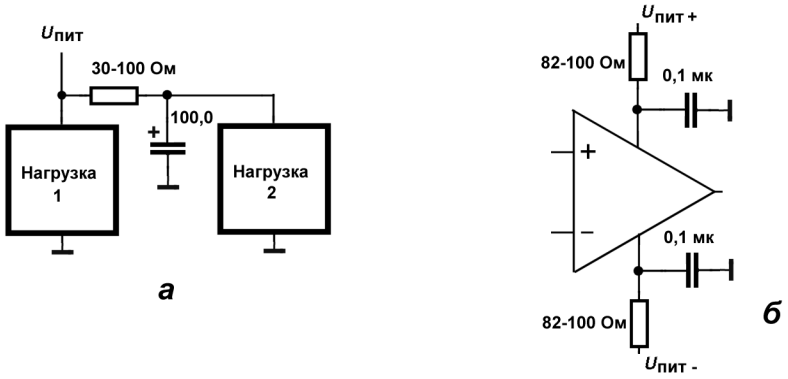
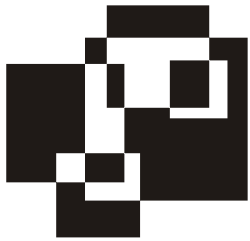


Рис. 4.9. Разводка питания: *а* — схема разделения нагрузок с помощью развязывающего фильтра; *б* — организация питания для быстродействующих и прецизионных усилителей

На рис. 4.9, *а* показана схема развязывающего фильтра для маломощной нагрузки (для одного электронного узла). Это может быть, например, входной каскад усиления микрофонного усилителя, который особо чувствителен к качеству питания, и его требуется развязать со следующими более мощными каскадами. На рис. 4.9, *б* показана правильная организация питания с такими фильтрами для быстродействующих или прецизионных измерительных усилителей, применяющихся, в частности, в измерительных схемах, о которых мы будем говорить в следующих главах.



Глава 5

Изготовление радиолюбительских конструкций

> Как любила повторять моя мама, если хочешь, чтобы что-то было сделано хорошо, сделай это сам.
>
> *Дик Френсис «Движущая сила»*

Есть такой эмпирический закон, известный под названием «закона Мэрфи», который имеет множество формулировок, но основная мысль, содержащаяся в нем, такова: «всегда полагайтесь на худший из возможных исходов». В моей практике этот закон не нарушался никогда: например, если некий прибор сломался, то обязательно следует предполагать, что поломка произошла как минимум в двух местах. И это невероятное предположение, противоречащее основным положениям теории надежности, обычно подтверждается на практике!

Наверное, вы хотите, чтобы ваши конструкции работали долго. Тогда имейте в виду, что в радиоэлектронике в полной мере оправдывается правило, которое заметили еще авиаконструкторы: красивый самолет имеет и лучшие летные качества. Аккуратно и эстетично смонтированный прибор будет работать лучше и надежнее — этому можно, кстати, отыскать вполне рациональные объяснения. Если, например, у вас соединительные провода между блоками имеют произвольную длину и толщину и кое-как запиханы в корпус прибора, напоминая мочалку для мытья посуды, то велика вероятность того, что вы зацепите тот или иной провод при сборке, и он просто оторвется, а если он слишком толстый и жесткий, то и цеплять не надо — пайка отломится при малейшей попытке отогнуть провод в сторону. Наоборот, слишком тонкий и мягкий провод будет цепляться за все подряд и обязательно попадет под крепежные винты.

Ни в коем случае не берите за образец сборку персональных компьютеров — там совершенно другая технологическая база, и спроектировано все настолько надежно, что хаотичное расположение кабелей в корпусе уже не может

помешать работоспособности (хотя в фирменно собранных ПК кабели все же убирают в аккуратные жгуты). «На коленке» такого не достичь, потому берите лучше пример с отечественной военной сборки, которая технологически немногим отличалась от «наколеночной», но, тем не менее, довольно надежно работала.

Радиолюбителю недоступны не только многослойные печатные платы, но часто даже обычные платы с металлизированными отверстиями. Однако если все сделано аккуратно и с соблюдением элементарных технологических правил, то ручная сборка ничуть не менее надежна, чем автоматизированная. Конечно, такой миниатюризации, когда в корпус мобильника вмещают процессор с чипсетом, высокочастотную приемную часть, память, контроллер дисплея и т. п. ручной сборкой достичь не удастся. И не надо к этому стремиться — применяйте более удобные для ручной пайки корпуса микросхем типа DIP (с выводами вниз, а не в плоскости самой микросхемы, см. рис. 12.1 в *главе 12*) и обычные резисторы и конденсаторы, с гибкими выводами, а не для поверхностного монтажа. Тогда все будет работать очень надежно.

Заметки на полях

Иногда микросхемы удобно ставить на панельки — не только дорогие, вроде микроконтроллеров или памяти, но даже и обычную логику. Это упрощает монтаж и позволит легко заменять их при необходимости. Следует только иметь в виду, что отечественные микросхемы в корпусах DIP выпускаются с шагом 2,5 мм, а импортные — 2,54 мм. Для выбора панелек это некритично, если число выводов в одном ряду не больше 16 — тогда они фактически взаимозаменяемы, в противном случае отечественные микросхемы могут не влезть в импортные панельки и наоборот. Для фирменных плат с металлизацией величина шага между выводами начинает сказываться уже для корпусов с четырьмя выводами в одном ряду. То же, кстати, относится и к некоторым другим компонентам, таким как клеммники, которые при внешней тождественности могут быть с шагом 5 или 5,08 мм. Если вы наберете ряд клеммников уже из трех-пяти штук, то при ошибке в раскладке они в плату не встанут.

Платы и пайка

Все схемы в настоящее время располагают на печатных платах. Название «печатные» произошло от того, что промышленные платы изготавливаются методом фотопечати. Однозначно следует отдавать ваши платы в промышленное изготовление, если вы делаете несколько экземпляров (чем больше, тем получится дешевле в расчете на один экземпляр) хорошо отработанного и обкатанного на макете устройства, так вы сильно экономите на последующей отладке, сборке, и, к тому же, надежность полученного устройства заметно выше и меньше зависит от квалификации монтажника. А если вы изготавливаете ваше изделие в одном экземпляре, то чаще всего затевать

подобную историю экономически нецелесообразно: времени уйдет масса, а стоимость раскладки и изготовления одной платы средних размеров даже в самых дешевых мастерских может составить сотни «вечнозеленых». Быстрее и дешевле аккуратно собрать схему на универсальной макетной плате, хотя это и приводит к значительной трате времени на монтаж и его проверку.

Изготовление плат

Существует немало описанных в литературе способов изготовления печатных плат в домашних условиях (достаточно поковыряться в старых подшивках журнала «Радио»). Вот один из самых простых.

Подготовьте рисунок проводников в натуральную величину — бумажный шаблон с четко обозначенными центрами отверстий. Раньше такие шаблоны приходилось рисовать карандашом на миллиметровке, теперь, располагая компьютером и принтером, можно сделать все гораздо аккуратнее и точнее. Вырежьте ножницами по металлу заготовку платы из фольгированного стеклотекстолита, соблюдая точные габаритные размеры. Затем плотно по всей поверхности наклейте на заготовку шаблон, используя простой резиновый клей — это позволит потом легко удалить бумагу и остатки клея с заготовки. Аккуратно накерните отверстия по шаблону, затем шаблон можно удалить.

Положите заготовку шаблоном вверх на деревянное основание, которое не жалко испортить, и закрепите ее струбцинкой. Чем плотнее заготовка прижмется к основанию, тем лучше. Затем микродрелью просверлите отверстия в помеченных местах, выбирая сверла соответствующего диаметра (для выводов большинства обычных компонентов — резисторов, диодов, маломощных транзисторов, микросхем — подойдет сверло 0,6—0,7 мм, остальные измерьте штангенциркулем и накиньте 0,1 мм). Учтите, что сверла на стеклотекстолите довольно быстро тупятся и приходят в негодность, потому следует иметь их запас. После сверления необходимо осторожно (чтобы не расширять отверстия) обработать края отверстий сверлом большего диаметра или зенковкой, чтобы убрать заусенцы. Наконец, обработайте поверхность с обеих сторон платы сначала обычной шкуркой, а затем нулевкой до зеркального блеска. На этом первый этап работы можно считать законченным.

Затем тщательно очистите рабочий стол и заготовку от стружек и пыли и обработайте с обеих сторон поверхность заготовки тампоном из хлопчатобумажной ткани (но не ваты!), смоченным бензином «Галоша». В дальнейшем старайтесь не касаться пальцами поверхности медного слоя, а берите заготовку пинцетом или, как компьютерный лазерный диск, за края.

Теперь вам понадобится водостойкий фломастер с тонким стержнем (не более 1 мм). Лучше, если фломастер новый — линия, проведенная даже очень

быстро, не должна прерываться. Проверьте его водостойкость, иначе вся работа может пойти насмарку (высохшая линия на бумаге при проведении по ней мокрым пальцем не должна иметь даже следов смазывания). К сожалению, все бытовые фломастеры теперь делаются на спирту, и это значительно снижает их водостойкость по сравнению с якобы вредными для здоровья старыми фломастерами на других органических растворителях бензоле и толуоле.

Этим фломастером следует сначала аккуратно обвести отверстия с обеих сторон платы, формируя контактные площадки. Они не должны быть слишком большими, иметь разрывы или непрокрашенные места. Потом только останется соединить эти площадки в соответствии с рисунком проводников (не забывайте, что вторая сторона выглядит зеркально по отношению к первой). Рисуя дорожки, не старайтесь их делать узкими, если место позволяет — лучше провести рядом несколько линий, сливающихся в одну, или использовать более толстый фломастер. И в любом случае следует шины питания, и особенно общего провода («земли»), делать как можно шире и стараться, чтобы питание проходило по одной стороне платы, а общий провод — по другой. Неплохо также распространить «землю» на свободную поверхность платы, где это возможно. Для некоторых аналоговых схем даже делают так: лицевую сторону платы (где расположены компоненты) оставляют целиком фольгированной, кроме протравленных мест под сквозные отверстия для выводов (это удобно делать зенковкой по готовой плате), и соединяют эту сторону с «землей», а остальные проводники располагают на другой стороне.

При изготовлении с помощью принтеров делается сначала все то же самое, кроме способа формирования изображения дорожек. Самое простое — напечатать изображение на поверхности слоя медной фольги с помощью принтера, имеющего прямой тракт подачи носителя, без перегибов. Однако обычные струйные принтеры для этой цели не годятся, так как чернила у них водорастворимые. Теоретически можно применить технологии термопереноса (а там и до литографии недалеко, правда?), однако все они очень дороги в смысле стоимости оборудования, и дешевле будет заказать обычную плату, потому что она тогда заодно получится с металлизированными отверстиями, недоступными в домашних условиях никаким технологиям.

Наибольшее распространение среди радиолюбителей получила технология с использованием обычных лазерных принтеров. При этом рисунок платы печатается в зеркальном изображении на каком-либо носителе, а затем переносится на плату с помощью горячего утюга (разумеется, тогда отверстия сверлятся после нанесения рисунка, а не до — оставьте маленькие просветы в центре рисунка каждой контактной площадки). Лучше взять утюг без отпаривателей, например, старый отечественный. Носителем может быть мело-

ванная глянцевая бумага (обложки от журналов), специальная тонкая принтерная бумага, прозрачная пленка (тоже специальная принтерная, обычная полиэфирная расплавится), алюминиевая фольга и др. Самое главное при этом — точно подобрать температуру утюга, чтобы тонер на основе расплавился и прилип к плате, но не растекся. Затем после остывания (лучше под грузом) основу удаляют. Обычную бумагу можно просто размочить в воде, а алюминиевую пленку вообще можно не удалять, т. к. она растворяется в травильном растворе, удалить надо только основу, на которую она была наклеена (без основы напечатать на ней ничего не удастся). Интересно, что при таком способе нанесения изображения узкие дорожки получаются лучше широких, правда, при печати на мелованной бумаге у них могут в итоге получаться «лохматые» края.

Нарисовав тем или иным способом проводники с обеих сторон, оставьте заготовку окончательно подсохнуть и подготовьте травильную ванну. Для этого лучше всего подходит фотографическая пластмассовая кювета. Ни в коем случае не металлическая! Из множества известных рецептов для травления меди в радиолюбительской практике лучшие результаты дает концентрированный раствор хлорного железа, который не выделяет в процессе работы газов и потому не повреждает рисунок фломастера. Он продается на рынках и радиолюбительских магазинах вроде «Чипа-Дипа», и его можно использовать многократно, только при хранении следует его плотно закрывать. Учтите, что все травильные растворы весьма агрессивно относятся к металлам и даже к не слишком качественной эмали на сантехнических приборах, поэтому нужно соблюдать предельную осторожность, чтобы не испортить раковину или ванну.

Окуните плату в подготовленный раствор. Работать лучше в резиновых перчатках, а манипулировать платой с помощью пинцета из пластмассы или нержавеющей стали, с гладкими губками (типа фотографического). Имейте в виду, что лимитирующая стадия процесса травления в хлорном железе — отвод продуктов травления от поверхности платы, поэтому в состоянии покоя плата снизу будет травиться гораздо быстрее, чем сверху, т. к. продукты реакции оседают на дно. Кювету нужно непрерывно покачивать и как можно чаще переворачивать плату, иначе могут остаться непротравленные участки, в то время как в других местах уже начнется процесс подтравливания дорожек. Важнее всего не пропустить момент, когда вся медь на непрокрашенных участках уже сошла. Если вы оставите плату на более долгий срок, считайте, что все испортили, т. к. краска долго не выдержит, и дорожки начнут протравливаться. Лучше всего в конце процесса периодически промывать плату в проточной воде и рассматривать ее на просвет.

После травления плату нужно тщательно промыть теплой водой, высушить и оставшуюся краску тщательно смыть ацетоном, меняя тампоны до удаления малейших следов фломастера. Наконец, все дорожки необходимо облудить. Для этого берется мощный паяльник (200 Вт), а плата целиком покрывается активным флюсом. При облуживании следует всего лишь легко касаться дорожек, чтобы долго их не прогревать, иначе они могут отслоиться. Затем плата еще раз промывается водой, высушивается и покрывается канифольным лаком, — теперь она готова к монтажу.

Пайка

Паяльник для пайки компонентов должен быть небольшой мощности (20—30 Вт), с тонким жалом, достаточно хорошо заточен и облужен, не перегреваться, но и не быть слишком холодным. Обязательно «красьте» канифольным флюсом всю плату, а не только места пайки. Для пайки удобен тонкий припой с канифолью внутри (слишком много канифоли не бывает!) — вы утыкаете одной рукой такую проволочку в место пайки, а другой прислоняете к этому месту кончик жала паяльника. Секунда — и пайка готова. Канифоль потом можно отмыть спиртом или спиртобензиновой смесью (не откладывайте этот процесс надолго, поскольку засохший канифольный лак удаляется значительно труднее). Однако в конструкциях «для себя» можно канифоль вообще не удалять, т. к. лак будет служить дополнительной изоляцией, помогать при доделках (которые неизбежны) и уменьшится риск засорить при промывке такие компоненты, как переменные резисторы.

После пайки выводы откусывают на требуемую длину: для промышленных плат с металлизированными отверстиями достаточно, чтобы места пайки выступали на 1 мм над поверхностью платы, для «доморощенных» необходимо оставлять несколько больше. Для плат собственного изготовления нужно не забывать, что сквозные отверстия не имеют металлизации и их следует пропаивать на обеих сторонах платы.

Не исключено, что вам попадутся отечественные или импортные детали, изготовленные давно, в первую очередь, это относится к сопротивлениям типа МЛТ, к некоторым типам конденсаторов и других компонентов. Я не знаю, какие материалы были тогда использованы, но выводы этих деталей при хранении чернеют (т. е. покрываются тонкой темной пленкой соединений типа сульфидов), и их пайка представляет определенные трудности. Такие компоненты вполне пригодны, только выводы нужно обработать: зачистить тонкой шкуркой-нулевкой, а затем облудить со всех сторон, стараясь не наносить лишнего припоя (иначе вывод может не влезть в предназначенное для него отверстие). Точно так же следует предварительно залуживать любые медные проводники, не покрытые припоем.

Заметки на полях

Снять лак с обмоточных проводов типа ПЭВ-2 и аналогичных можно шкуркой (только не резаком и не скальпелем, потому что зачистка будет некачественная, а кончик провода потом легко обламывается), или обжигом кончиков провода на зажигалке с последующим залуживанием с помощью активного флюса, вроде того, что описан далее. Но для ускорения процесса и получения стабильного результата до сей поры ничего лучше не придумано, чем старинный способ с использованием таблеток аспирина (ацетилсалициловой кислоты). Они легко плавятся паяльником, выделяя компоненты, которые размягчают лак и позволяют его счистить прямо кончиком паяльника с одновременным облуживанием.

В качестве активного флюса для облуживания дорожек, окислившихся выводов деталей, поверхностей из стали, грязной меди, латуни или, скажем, нихрома, удобно применять совершенно другую композицию. Из имеющихся в продаже можно рекомендовать «Паяльную кислоту» на основе хлористого цинка или «ХАФ» на основе хлористого аммония — оба они смываются водой.

Заметки на полях

Автор же вот уже в течение трех с лишним десятков лет использует самостоятельно приготовляемый активный флюс, который дает отличные результаты даже для нержавеющих сталей (для пайки которых обычно рекомендуют ортофосфорную кислоту). Приготавливается он следующим образом: нужно засыпать в пузырек примерно на одну треть его высоты порошок хлористого аммония и залить доверху смесью, состоящей из 70% глицерина и 30% воды. Взболтать эту смесь и оставить на одну-две недели. Если хлористый аммоний по истечении этого срока полностью растворится — досыпать еще, если нет — осадок не помешает. Насыщенным раствором удобно заполнить одноразовый шприц или полиэтиленовую пипетку с завинчивающейся крышечкой (например, от лекарства, которое закапывается в нос при гриппе). После применения остатки такого флюса обязательно смыть теплой водой под краном или стереть мокрой тряпочкой и тщательно высушить место пайки. Флюс совершенно нейтрален, не ядовит, безопасен для рук и не разъедает дерево, но чрезвычайно текуч и очень медленно испаряется, поэтому его остатки со стола и с других предметов следует тщательно удалять влажной тряпкой. Не следует употреблять его совместно с канифолью — они друг другу будут мешать и смывать остатки при этом гораздо труднее.

И еще один совет, который относится к распайке компонентов на платах промышленного изготовления. Дело в том, что в процессе производства контактные площадки и дорожки покрываются сплавами (типа «Розе»), имеющими очень низкую температуру плавления. Поэтому, припаивая к ним вывод некоего компонента, не следует удерживать этот вывод на весу трясущейся рукой с пинцетом — припой застынет тогда, когда тонкий слой сплава на поверхности дорожки еще будет жидким, и очень надежное по внешнему виду паяное соединение на поверку окажется просто блямбой припоя, слегка прижатой к контакту на плате за счет упругости вывода.

Макетные платы

Иногда под макетными платами понимают довольно сложные устройства с множеством зажимов, где схему можно собирать без помощи паяльника. Такие конструкции имеются в продаже. Но обычно, говоря о макетной плате, имеют в виду просто печатную плату, на которой предусмотрены места для установки компонентов (отверстия и контактные площадки), не соединенные проводниками вовсе или соединенные по некоей специальной универсальной схеме. Такая плата пригодна не только для собственно макетирования, но и для изготовления отдельных изделий в единичных экземплярах, что нередко практикуют и профессионалы.

Простейший вариант макетной платы — поле из металлизированных отверстий с двусторонними контактными площадками с шагом 2,5 (или 2,54) мм между ними. Некоторые варианты рисунка макетных плат показаны на рис. 5.1. Не поленитесь приобрести подобные платы — они продаются на радиорынках и том же «Чипе-Дипе». В крайнем случае их следует заказать, хотя это и дорого. Учтите, что абсолютно универсальной платы, пригодной для расположения любых компонентов, не существует, и в большинстве случаев имеющиеся приходится дорабатывать.

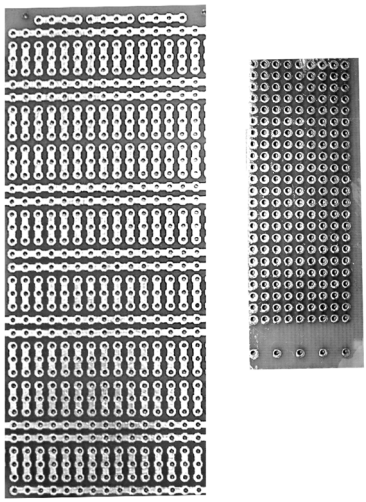


Рис. 5.1. Фрагменты различных макетных плат

Соединения между выводами компонентов на такой плате осуществляются в процессе сборки схемы с помощью отрезков обычного изолированного провода — лучше всего для этой цели употреблять т. н. «луженку», под кото-

рой понимается тонкий (сечением не более 0,5 мм) одножильный медный провод, покрытый припоем, в разноцветной хлорвиниловой изоляции. Такой провод имеет один «капитальный» недостаток — хлорвиниловая изоляция легко плавится при нагревании и «скукоживается» при пайке, обнажая концы на недопустимую длину. К сожалению, одножильных проводов для подобного монтажа в термостойкой (фторопластовой) изоляции я не встречал, хотя они, наверное, существуют в природе. Поэтому на практике удобнее гибкий фторопластовый (тефлоновый) провод типа МГТФ, хотя монтаж с его помощью получается не столь надежным из-за его гибкости.

При монтаже не следует стараться провести проводники «красиво» (по прямым перпендикулярным линиям) — наоборот, качество и надежность схемы будет выше, если все соединения разведены по кратчайшему пути. Необходимо, чтобы провода были припаяны «внатяг», а не змеились по плате. Короткие соединения, например перемычки, удобно делать неизолированными обрезками выводов от резисторов и диодов. Заметим, что не следует припаивать выводы деталей, особенно провода для внешних соединений платы, просто к контактной площадке или дорожке — их по мере возможности нужно просовывать в предусмотренное отверстие. В любом случае желательно прикреплять жгут внешних проводов к плате хомутиком, а по мере возможности межплатные соединения выполнять плоскими кабелями с игольчатыми разъемами типа IDC (какие используются для подсоединения жестких дисков с IDE-интерфейсом в компьютерах). Кабельные части разъемов (розетки) выпускаются на любое четное число контактов и легко заделываются с помощью специального инструмента.

Немного о резисторах и конденсаторах

Промышленные резисторы имеют строго определенные значения сопротивлений из стандартных рядов, выбранных так, чтобы при заданном допуске (например, 10%) границы возможных значений пересекались. Поэтому резисторы имеют такие «странные» номинальные значения: 3,9 или 5,1 кОм (а не естественные 4 и 5 кОм ровно). Современные резисторы маркируются цветным кодом, читать который — мука мученическая, учитывая особенно, что понятие, скажем, «золотистый» очень часто трактуется производителями весьма вольно, и отличить его от «оранжевого» или «желтого», к примеру, на темно-синем фоне, может только человек с большим опытом. Поэтому на практике проще и быстрее просто измерить сопротивление мультиметром.

В каждой декаде номиналы получаются из табличного ряда значений путем умножения на соответствующую степень десяти. Для маркировки резисторов,

не помеченных цветным кодом (например, старинных МЛТ) часто используют условные обозначения для каждого диапазона: буква R (или E) — обозначает омы, к — килоомы, м или M — мегомы. Эти буквы могут заменять десятичную точку: так, запись 1к2 есть то же самое, что и 1,2 кОм, а 3R3 (или 3E3) — то же самое, что 3,3 Ом. При обозначении на схемах омы в большинстве случаев вообще опускают, именно так мы будем поступать в этой книге, так что имейте в виду, что запись «360» на схеме означает просто 360 Ом.

ЧИП-резисторы для поверхностного монтажа маркируются по-другому: тремя цифрами, первые две из которых есть номинальное значение (без запятой!), а последняя справа — степень десяти. Так, надпись 103 означает $10 \cdot 10^3 = 10\,000$ Ом, т. е. 10 кОм, а надпись 272 — 2700 Ом, т. е. 2,7 кОм.

Аналогично маркируются конденсаторы (любые малогабаритные), только за основу шкалы там приняты пикофарады (10^{-12} Ф). Так что надпись 474, скажем, расшифровывается, как $47{\cdot}10^4 \cdot 10^{-12} = 0{,}47 \cdot 10^{-6}$ Ф или 0,47 мкФ. При обозначении на схемах единицу измерения (Ф) часто опускают, и пишут просто «мк» (мкФ), «н» или «n» (нФ), «п» или «p» (пФ). Пикофарады (подобно омам) могут вообще не указывать. Часто микрофарады обозначают просто лишним десятичным знаком (мы именно так и поступали в *главе 4*) — например, запись «100,0» означает 100 мкФ, в то время как просто «100» — это 100 пФ.

Корпуса

Проблема корпусов для радиоаппаратуры не стоит особенно остро — все крупные (и помельче) фирмы, торгующие компонентами, предлагают и различные корпуса. Беда тут примерно та же, что и с покупкой, скажем, обуви — вроде ее много на любой вкус и кошелек, да одни ботинки не смотрятся, в других кантик неподходящий, третьи цветом не вышли, четвертые в подъеме жмут… Короче, подобрать под конкретный прибор готовый корпус — задача весьма непростая. Потратив несколько десятков «баксов» на блестящее заморское изделие, очень не хочется браться за напильник, чтобы доводить его до ума, но приходится — здесь должно быть окно для индикатора, эту стенку вообще надо удалить, ибо тут будет стоять радиатор для мощного транзистора, тут требуются фигурные отверстия под разъемы... Тогда, спрашивается, зачем тратились? А если еще ошибешься, что нередко случается даже с опытными слесарями?

В общем, есть простой способ изготовления корпусов в домашних условиях под конкретные нужды, причем если «руки на месте», то такие готовые изделия будут выглядеть практически не хуже фабричных. Заключается способ

в том, что вы сначала рисуете эскизы всех стенок и перегородок, располагаете на экране компьютера (или просто карандашом на бумаге) все детали и платы, чтобы они не наезжали друг на друга, выверяете размеры (компьютер дает простор для такого рода творчества), а затем по готовым эскизам переносите размеры на фольгированный стеклотекстолит и вырезаете заготовки. Не забывайте давать припуски на толщину материала по нужным сторонам заготовок.

Лучше все отверстия сделать заранее, поскольку всегда удобнее работать с пластинкой, чем с готовой коробкой. Затем, прикладывая заготовки под прямым углом друг к другу, пропаиваете место стыка обычным припоем. Работать нужно самым мощным паяльником (200—400 Вт), припоем в прутках и водорастворимым активным флюсом. Сложность только одна, но существенная: припой сокращается в объеме при застывании, потому пластинки под прямым углом относительно друг друга надо прочно закреплять, иначе угол окажется совсем не прямым, а распаять будет уже очень трудно. Готовый корпус обтягивается самоклеящейся пленкой, например, под темное дерево. Если делать все аккуратно, получается классно!

Несколько замечаний по оформлению корпуса. Первое: если у вас в корпусе окно для индикаторов, то его надо делать из дымчатого, а не прозрачного пластика, а все, что за этим окном расположено, кроме, естественно, самих индикаторов (включая плату с компонентами), выкрасить в черный цвет из аэрозольного баллончика — это придаст оттенок «фирменности» вашему изделию. Ужасно выглядят конструкции, в которых через стекло виднеются пайки на печатной плате. Можно к тому же заклеить всю незадействованную поверхность окна изнутри черной липкой лентой. Если следовать этому совету, то можно не выпиливать окна точно по размеру индикатора, что довольно сложно сделать красиво, а выполнить из дымчатого оргстекла, например, всю переднюю панель.

Второе замечание касается нанесения надписей на переднюю панель. Наилучший способ — заказать панель с лазерной гравировкой. Но это дорого и хлопотно, поэтому хочется сделать самому. Ручной способ отвергаем с порога — ничто не может выглядеть кошмарнее, чем надписи, сделанные вручную. Никакие трафареты и гравировальные машинки здесь помочь не могут. Это вообще была одна из самых тяжелых проблем до последнего времени и не только для радиолюбителей, даже мелкосерийные приборы на советских заводах выпускались с гравированными вручную надписями. И это было не слишком эстетично.

К счастью, в последние годы в связи со всеобщей доступностью струйных принтеров проблема качественной печати любым размером шрифта, любым цветом и на любом фоне решена полностью. Делается это на специальной

основе, которая с одной стороны липкая и покрыта защитным слоем, как самоклеющаяся пленка, а с другой имеет особую пористую фактуру, хорошо удерживающую принтерные чернила. Она довольно дорогая, но десяти листочков вам хватит «на всю оставшуюся жизнь», если вы, конечно, не собираетесь налаживать крупносерийное производство. Если же такой пленки под рукой нет, то можно напечатать надписи просто на плотной мелованной бумаге (например, на обратной стороне обложки настенного календаря), а затем приклеить их двусторонним скотчем. Красивее всего, на мой взгляд, выглядят надписи, напечатанные с инверсией, т. е. белым цветом на черном фоне, только не забудьте закрасить белые торцы готовых к наклейке «лейблов» черным фломастером, иначе они будут очень бросаться в глаза.

Расчет радиаторов

Сразу скажем, что научно-обоснованной методики для расчета охлаждающих радиаторов не существует. По этому поводу можно написать не одну диссертацию или монографию (и написаны, и много), но стоит изменить конфигурацию охлаждающих ребер или стержней, расположить радиатор не вертикально, а горизонтально, приблизить к нему любую другую поверхность снизу, сверху или сбоку, как все изменится и иногда кардинально. Именно поэтому производители микропроцессоров или видеокарт предпочитают не рисковать, а снабжать свои изделия радиаторами с вентилятором — принудительный обдув, даже слабенький, повышает эффективность теплоотвода в десятки раз, хотя зачастую этого и не требуется. Последние модели компьютерных источников питания и материнских плат позволяют автоматически регулировать интенсивность обдува с целью снижения уровня шума, и некоторые такие конструкции вообще не запускают вентилятор, если процессор простаивает. В *главе 6* мы поговорим о том, как самостоятельно изготовить такой регулятор.

В критичных случаях, для снижения габаритов очень мощного устройства, конечно, можно вместо пассивного радиатора пристроить к вашей конструкции процессорный «кулер» с вентилятором. Правда, на практике мне этого делать никогда не приходилось, да и надежность конструкции снижается, т. к. за исправностью вентилятора приходится следить, а это неприемлемо для устройств, которые предназначены для автономной работы в течение длительного времени. Потому в радиолюбительских конструкциях мы обойдемся пассивными (без обдува) охлаждающими устройствами.

Здесь мы приведем только пару-другую эмпирических способов, которые оправдали себя на практике и годятся для того, чтобы рассчитывать именно пассивные радиаторы, устроенные примерно так, как показано на рис. 5.2.

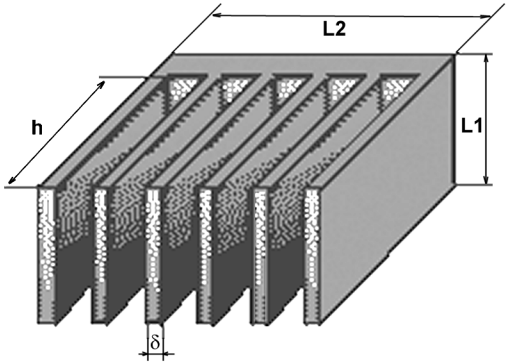


Рис. 5.2. Типичный пластинчатый радиатор

Сначала рассмотрим, как рассчитывать площадь радиаторов, исходя из их геометрии. Для такого расчета нужно к площади основания прибавить суммарную площадь его ребер (также с каждой стороны). Если нижней стороной радиатор прижимается к плате, то лучше считать рабочей только одну сторону основания, но мы предположим, что радиатор «висит» в воздухе (как часто и бывает) и поэтому площадь основания удваивается: $S_{осн} = 2 \cdot L_1 \cdot L_2$. Площадь одного ребра (тоже с двух сторон) $S_р = 2 \cdot L_1 \cdot h$, но к этой величине нужно еще прибавить боковые поверхности ребра, площадь которых равна $S_{бок} = 2 \cdot h \cdot \delta$. В данном случае ребер всего 6, поэтому общая площадь радиатора $S = S_{осн} + 6 \cdot S_р + 6 \cdot S_{бок}$. Пусть $L_1 = 3$ см, $L_2 = 5$ см, $h = 3$ см, $\delta = 0{,}2$ см, тогда общая площадь такого радиатора будет 145 см^2. Разумеется, это приближенный расчет (мы не учли, скажем, боковую поверхность основания), но для наших целей точнее и не надо.

Вот два эмпирических способа для расчета рассеиваемой мощности в зависимости от площади поверхности, и пусть меня не слишком строго осудят за то, что никаких особенных научных выкладок вы здесь не увидите.

Способ первый и наипростейший: площадь охлаждающего радиатора должна составлять 10 см^2 на каждый ватт выделяющейся мощности. Так что радиатор на рис. 5.2 с размерами, приведенными ранее, согласно этому правилу может рассеять 14,5 Вт мощности (как раз годится для простейшего источника питания, показанного на рис. 4.5, *б* или 4.6). И если позволяют размеры корпуса, то вполне можно ограничиться этим прикидочным расчетом.

Если же вы хотите подсчитать поточнее, то вот один из более сложных способов, который годится для пластинчатых радиаторов средних размеров ($L_1 = 20$—180 мм, $L_2 = 40$—125 мм).

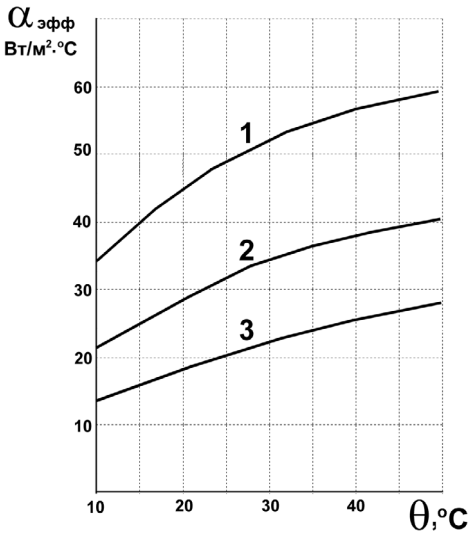


Рис. 5.3. Эффективный коэффициент теплоотдачи ребристого радиатора в условиях свободной конвекции при различной длине ребра:
1 — *h*=32 мм; 2 — *h*=20 мм; 3 — *h*=12,5 мм

Для оценки тепловой мощности радиатора можно использовать следующую зависимость: $W = \alpha_{эфф} \cdot \theta \cdot S$,

где: W — мощность, рассеиваемая радиатором, Вт; $\alpha_{эфф}$ — эффективный коэффициент теплоотдачи, Вт/м$^2 \cdot$ °С (см. график на рис. 5.3); θ — величина допустимого перегрева теплоотдающей поверхности, °С. $\theta = T_c - T_{o.c}$ (T_c — средняя температура поверхности радиатора, $T_{o.c}$ — температура окружающей среды), S — полная площадь теплоотдающей поверхности радиатора, м^2.

Обратите внимание, что площадь в эту формулу подставляется в квадратных метрах, а не сантиметрах.

Посчитаем мощность для радиатора, показанного на рис. 5.2 с размерами, приведенными ранее. Сначала зададимся желательным перегревом поверхности θ, выбрав не слишком большую величину, равную 30 °С. Можно полагать тогда, что при температуре окружающей среды 30°, температура поверхности радиатора составит 60°. Если учесть, что разница между температурами радиатора и кристалла транзистора или микросхемы при хорошем тепловом контакте (о котором далее) может составить примерно 5°, то это приемлемо практически для всех полупроводниковых приборов.

Высота ребер h у нас составляет 30 мм, поэтому пользуемся верхней кривой на графике рис. 5.3, откуда определяем, что величина коэффициента теплоотдачи $\alpha_{эфф} \approx 50$ Вт/м$^2 \cdot$ °С. После вычислений получим, что $W = 22$ Вт. Ранее

по простейшему правилу мы получили 14,5 Вт, т. е. проведя более точные расчеты, мы можем раза в полтора уменьшить площадь радиатора, тем самым сэкономив место в корпусе. Однако, повторим, если габариты позволяют, то лучше всегда иметь запас.

Радиатор (и его ребра) следует располагать вертикально (как на рис. 5.2), а поверхность его желательно покрасить в черный цвет. Я еще раз хочу напомнить, что все эти расчеты очень приблизительны, и даже сама методика может измениться, если вы поставите радиатор не вертикально, а горизонтально или снабдите его игольчатыми ребрами вместо пластинчатых. К тому же мы никак не учитываем здесь тепловое сопротивление переходов «кристалл-корпус» и «корпус-радиатор» (просто предположив, что разница температур составит 5°). Указанные методы дают неплохое приближение к истине, но если мы не обеспечим хороший тепловой контакт, все наши расчеты могут пойти насмарку.

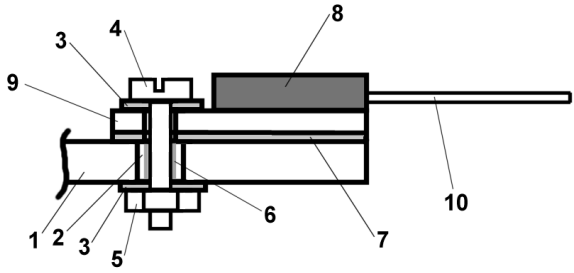


Рис. 5.4. Крепление транзистора в корпусе ТО-220 к радиатору при необходимости его изоляции: 1 — радиатор; 2 — отверстие в радиаторе; 3 — изолирующие шайбы; 4 — стягивающий винт; 5 — гайка; 6 — изолирующая трубка; 7 — слюдяная прокладка; 8 — пластмассовая часть корпуса транзистора; 9 — металлическая часть корпуса транзистора (коллектор); 10 — выводы транзистора

Просто плотно прижать винтом транзистор к радиатору, конечно, можно, но только в том случае, если поверхность радиатора в месте прижима идеально плоская и хорошо отшлифована. Практически этого никогда не бывает, поэтому радиатор в месте прижима смазывают специальной токопроводящей пастой. Ее можно купить в магазинах, а иногда тюбик с такой пастой прикладывают к «кулерам» для микропроцессоров. Смазывать поверхность надо тонким, но равномерным слоем.

Если на один радиатор ставятся два прибора, у которых корпуса находятся под разным напряжением, то под один из них нужно подложить изолирующую прокладку, под крепежные винты — изолирующие пластиковые шайбы, а на сами винты на длину, равную толщине радиатора в месте отверстия,

надеть отрезок изолирующей трубки (рис. 5.4). Самые качественные изолирующие прокладки — слюдяные, хороши прокладки из анодированного алюминия (но за ними надо внимательно следить, чтобы не процарапать тонкий слой изолирующего окисла) и из керамики (которые, впрочем, довольно хрупки и могут треснуть при слишком сильном нажиме). Кстати, за неимением фирменных прокладок можно использовать тонкую фторопластовую (но не полиэтиленовую, разумеется!) пленку, следя за тем, чтобы ее не прорвать. При установке на прокладку теплопроводящая паста наносится тонким слоем на обе поверхности — и на транзистор, и на радиатор.

Помехи

В заключение главы проясним ситуацию, связанную с сетевыми помехозащитными фильтрами. Вопреки распространенному мнению, такие фильтры чаще защищают от помех внешнюю сеть, а не сам прибор от внешних помех, проникающих из сети (исключение, конечно, составляют радиочастотные устройства). Если вы включите напрямую в сеть тиристорный регулятор, мощное электронное реле или импульсный блок питания (вроде компьютерного), то помех не избежать — как электрических по проводам сети, так и электромагнитных, распространяющихся в пространстве. Чем мощнее нагрузка, тем больше эти помехи. Особенно чувствительны к их воздействию АМ-приемники: мощный регулятор может подавить передачи Би-Би-Си не хуже советских глушилок.

Для того чтобы свести помехи импульсных приборов к минимуму, необходимо, во-первых, заземлить корпус прибора, во-вторых, на входе питания устройства вместе с нагрузкой поставить LC-фильтр. Это относится и к достаточно мощным преобразователям в интегральном исполнении.

Заметки на полях

Чтобы заземлить корпус, он, естественно, должен быть металлическим или металлизированным. Если же корпус чисто пластмассовый, то его нужно изнутри обклеить алюминиевой фольгой потолще (та, что для применения в микроволновых печах, конечно, не подойдет). Надежно обеспечить контакт вывода заземления с таким экраном непросто — это можно сделать, приклеив зачищенный на несколько сантиметров провод широким скотчем или соорудив прижимной контакт из упругой бронзы (например, из контакта старого мощного реле). Корпуса всех внешних разъемов, если они металлические, также следует надежно соединить с заземленным корпусом. Экран, как мы говорили ранее, соединяется с «землей» прибора (в одной точке), но если у вас сетевой блок питания, то экран тогда целесообразнее соединить с заземлением (зеленый провод) в сетевой вилке. Это может показаться бессмысленным ввиду отсутствия настоящей (без кавычек) земли в большинстве наших домов, но на самом деле совсем не глупо, если несколько приборов соединяются через один блок розеток с общим заземлением. В то же время для ряда схем, особенно измерительных, соединять экран с «землей» (общим проводом) схемы не следует — сами они помех не создают, а присоединение экрана к общему заземлению может ухудшить их работу.

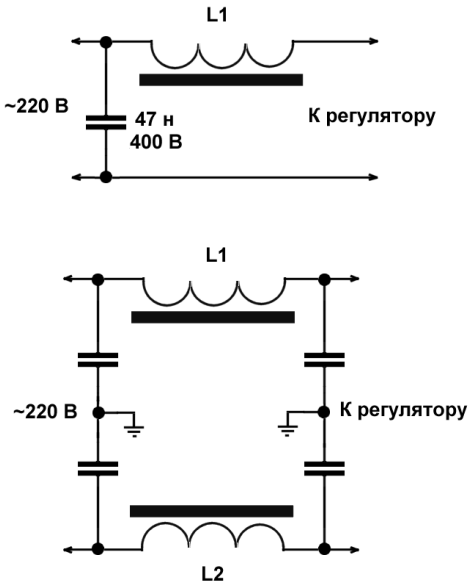
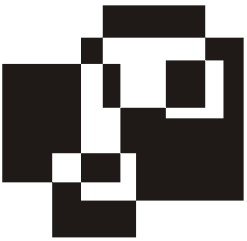


Рис. 5.5. Схемы фильтров сетевого питания для подавления помех

На рис. 5.5 приведены два варианта построения развязывающего LC-фильтра. Первый (вверху) вам уже знаком по схеме импульсного преобразователя (см. рис. 4.9). Второй, более сложный вариант (внизу), предназначен для схем помощнее, подобные фильтры входят, например, в удлинители типа «Пилот». При небольших токах берут готовые дроссели, как уже говорилось, они внешне очень похожи на резисторы. Для изготовления дросселей при больших токах (несколько ампер и более) нужно взять ферритовое кольцо марки 600—1000НН диаметром 15—24 мм и намотать на него виток к витку провод МГШВ сечением около 1 мм^2 до заполнения.

Во втором варианте фильтра дроссели L1 и L2 можно объединить, намотав их на одном кольце, причем если помехи будут подавляться плохо, то надо поменять местами начало и конец одной из обмоток. Конденсаторы — любые неполярные на напряжение не менее 400 В, среднюю точку их во втором варианте нужно подсоединить к заземлению (т. е. к уже заземленному корпусу). Если таковое отсутствует, то все равно надо присоединить эту точку к экрану корпуса прибора, но без настоящего заземления эффективность фильтра заметно ухудшится, — фактически он превратится в несколько улучшенный первый вариант.



Глава 6

Аналоговые микросхемы

> Интересно, о каких, собственно, микросхемах идет речь в твоем вопросе?
>
> *Форум радиолюбителей на shema.ru*

Самые первые микросхемы были совсем не такими, как сейчас. Они изготавливались гибридным способом: на изолирующую подложку напылялись алюминиевые проводники, приклеивались маленькие кристаллики отдельных транзисторов и диодов, малогабаритные резисторы и конденсаторы, и затем все это соединялось в нужную схему тонюсенькими золотыми проволочками — вручную, точечной сваркой под микроскопом. Можно себе представить, какова была цена таких устройств, которые тогда еще не назывались микросхемами, чаще употребляли название микромодули или микросборки. К гибридным микросхемам относятся и некоторые современные их типы, к примеру, оптоэлектронные, но, конечно, сейчас выводы отдельных деталей уже вручную не приваривают.

Слайсы, которые стали чипами

Ведущий специалист и один из основателей компании Fairchild Semiconductor Роберт Нойс позднее признавался, что ему стало жалко работников, терявших зрение на подобных операциях, и в 1959 году он выдвинул идею микросхемы — «слайса» или «чипа» (slice — ломтик, chip — щепка, осколок), где все соединения наносятся на кристалл прямо в процессе производства. Потом оказалось, что несколько ранее аналогичную идею выдвинул сотрудник Texas Instruments Джек Килби, однако у Нойса технология была разработана более детально (это была так называемая планарная технология с алюминиевыми межсоединениями, которая часто используется и по сей день). Спор о приоритете между Килби и Нойсом продолжался в течение десяти лет,

и в конце концов победила дружба: было решено считать Нойса и Килби изобретателями микросхемы совместно. В 2000 году Килби (Нойс скончался в 1990) получил за изобретение микросхемы Нобелевскую премию (одновременно с ним, но за другие достижения, ее получил и российский физик Жорес Алферов).

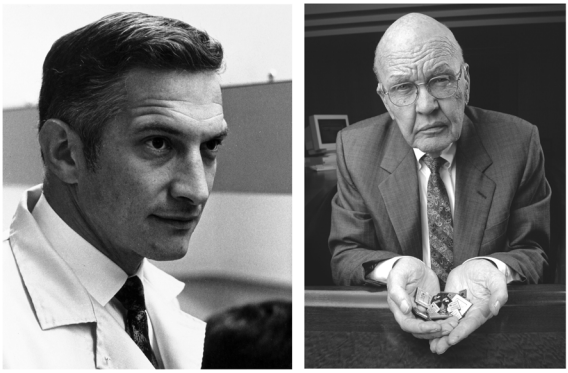


Рис. 6.1. Изобретатели микросхемы Роберт Нойс (Robert Noyce, 1927—1990, слева) и Джек Килби (Jack St. Clair Kilby, 1923—2005)

Что же дало внедрение интегральных микросхем, кроме очевидных преимуществ, таких как миниатюризация схем и сокращение числа операций при проектировании и изготовлении электронных устройств?

Рассмотрим прежде всего экономический аспект. Первым производителям чипов это было еще не очевидно, но экономически производство микросхем отличается от других производств. Если вы закажете архитектору проект загородного дома, то стоимость этого проекта будет сравнима со стоимостью самого дома. Даже если вы по этому проекту построите сто домов, то не так уж сильно выгадаете на стоимости каждого. Стоимость проекта поделится на сто, но выгода ваша будет измеряться процентами, потому что построить дом дешевле, чем стоят материалы и оплата труда рабочих нельзя, а они-то и составляют значительную часть затрат на строительство.

В производстве же микросхем картина меняется. Цена материалов, из которых они изготовлены, в пересчете на каждый чип настолько мала, что она составляет лишь несколько процентов от стоимости конечного изделия. Поэтому основная часть себестоимости микросхемы складывается из стоимости ее проектирования и производства, на котором она изготавливается (фабрика для производства полупроводниковых компонентов может обойтись в сумму порядка 2—4 млрд долл.). Ясно, что в этой ситуации определяющим факто-

ром конечной стоимости микросхемы будет количество, которое вы заказываете: если вам нужно меньше миллиона экземпляров, то с вами даже разговаривать не станут, а если вы будете продолжать настаивать, то один экземпляр обойдется вам во столько же, сколько и весь миллион. Именно массовость производства приводит к тому, что сложнейшие схемы, которые в дискретном виде занимали бы целые шкафы ценой в десятки тысяч долларов, продаются дешевле томика технической документации к ним.

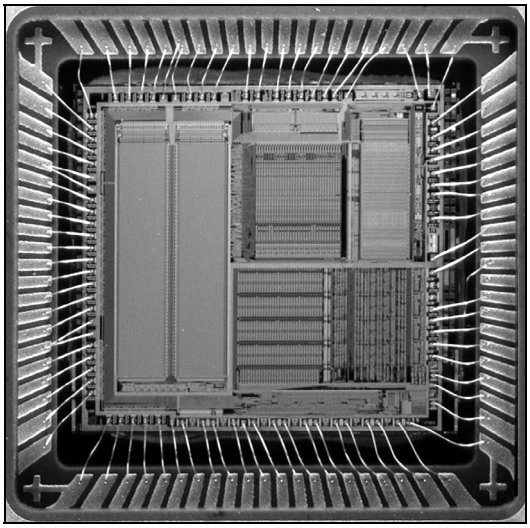


Рис. 6.2. Кристалл микропроцессора
(двойное поле слева — область встроенной памяти)

Вторая особенность экономики производства микросхем — то, что их цена мало зависит от сложности. Микросхема операционного усилителя содержит несколько десятков транзисторов, а микросхема микроконтроллера (рис. 6.2) — несколько десятков и сотен тысяч, однако их стоимости по меньшей мере сравнимы. Эта особенность тоже не имеет аналогов в дискретном мире, т. к. с увеличением сложности обычной схемы ее цена растет пропорционально количеству использованных деталей. Фактически единственный фактор, кроме стоимости проектирования, который ведет к увеличению себестоимости сложных микросхем по сравнению с более простыми — это процент выхода годных изделий, который снижается при увеличении размеров и числа элементов на кристалле. Если бы не это обстоятельство, то стоимость Pentium не намного бы превышала стоимость того же операционного усилителя. Однако в Pentium, извините, несколько десятков миллионов транзисторов! Все это позволило проектировщикам без увеличения стоимости и габаритов

реализовать в микросхемах такие функции, которые в дискретном виде было бы осуществить просто невозможно или крайне дорого.

Заметки на полях

Кстати, выход годных — одна из причин того, что кристаллы микросхем такие маленькие. В некоторых случаях разработчики даже рады были бы увеличить размеры, но тогда резко снижается и выход. Типичный пример — многолетняя борьба производителей цифровых фотоаппаратов за увеличение размера светочувствительной матрицы. Если бы удалось наладить массовый выпуск матриц размером с пленочный кадр (24×36 мм), то это одним махом решило бы множество проблем, но на момент написания этой книги только самые лучшие (и наиболее дорогие) любительские камеры имеют такие матрицы.

Еще одна особенность микросхем — высокая надежность. Дискретный аналог устройства типа аналого-цифрового преобразователя содержал бы столько паек, что какая-нибудь в конце концов обязательно вышла бы из строя. Между тем, если вы эксплуатируете микросхему в штатном режиме, то вероятность ее выхода из строя измеряется миллионными долями процента. Это настолько редкое явление, что его можно вообще не учитывать на практике. Если у вас сломался какой-то электронный прибор, ищите причину в контактах переключателей, в пайках внешних выводов, в заделке проводов в разъемах, но про возможность выхода из строя микросхемы забудьте. Разумеется, это, повторяю, относится к случаю эксплуатации в штатном режиме, если вы подали на микрофонный вход звуковой карты напряжение 220 В, то конечно, в первую очередь пострадает именно микросхема. Но сами по себе они практически не выходят из строя никогда.

Наконец, для схемотехников микросхемы обладают еще одним бесценным свойством: все компоненты в них изготавливаются в едином технологическом процессе и находятся в строго одинаковых температурных условиях. Это совершенно недостижимо для дискретных приборов — например, пары транзисторов, для которых желательно иметь идентичные характеристики, ранее приходилось подбирать вручную (такие уже подобранные пары специально выпускались промышленно) и иногда даже ставить их на медную пластину, чтобы обеспечить одинаковый температурный режим.

Рассмотрим типичный пример — так называемое токовое зеркало (рис. 6.3). Эта схема работает следующим образом. Левый по схеме транзистор представляет собой фактически диод, т. к. у него коллектор соединен с базой. Из характеристики диода (см. рис. 3.1) видно, что при изменении прямого тока на нем несколько меняется и напряжение (оно не равно точно 0,6 В). Это напряжение без изменений передается на базу второго, ведомого транзистора, в результате чего он выдает точно такой же ток — но только при условии, если характеристики транзисторов согласованы с высокой степенью точности.

Мало того, это соответствие должно сохраняться во всем диапазоне рабочих температур! Естественно, столь высокая идентичность характеристик практически недостижима для дискретных приборов, а для транзисторов, входящих в состав микросхемы, она получается сама по себе, без дополнительных усилий со стороны разработчиков.

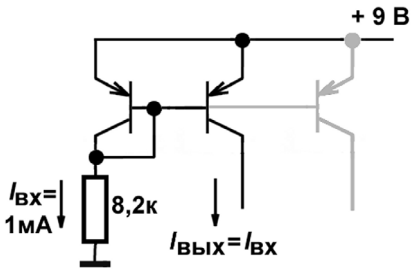


Рис. 6.3. Токовое зеркало

Подробности

Схемы подобных токовых зеркал получили широкое распространение в интегральных операционных усилителях в качестве нагрузки входного дифференциального каскада, что значительно лучше простых резисторов. Их применение вместо резисторов гарантирует повторяемость характеристик ОУ в широком диапазоне питающих напряжений. Отметим также, что ведомых транзисторов может быть много (на рис. 6.3 второй такой транзистор показан серым цветом), их число ограничивается только тем обстоятельством, что базовые токи вносят погрешность в работу схемы, отбирая часть входного тока на себя. Впрочем, и с этим можно успешно бороться.

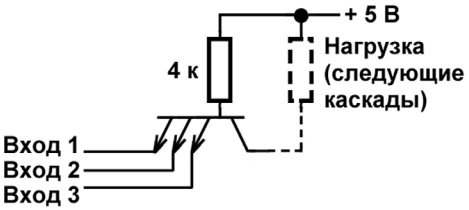


Рис. 6.4. Входной каскад элемента ТТЛ

Кстати, резисторы в микросхемах в некритичных случаях все равно предпочитают делать из транзисторов, поскольку сформировать обыкновенный резистор, как проводник с заданным сопротивлением, в процессе производства микросхем значительно труднее, чем соорудить, скажем, полевой транзистор

с заданным начальным током стока. В микросхемах могут использоваться такие разновидности транзисторных структур, которые в обычной дискретной жизни не имеют аналогов: скажем, многоэмиттерные или многоколлекторные транзисторы. Для примера на рис. 6.4 приведена схема входного каскада микросхемы транзисторно-транзисторной логики (ТТЛ), осуществляющей логическую функцию «ИЛИ» (подробнее об этом см. *главу 8*).

Эксплуатация микросхем

Возможно, вы слышали о том, что микросхемы боятся статического электричества. Действительно, потенциал заряда, накапливающегося во время ходьбы на нейлоновом халатике симпатичной монтажницы, одетой к тому же в синтетические юбочку, кофточку и колготки, может составлять тысячи вольт (правда, сама величина заряда невелика). Но необязательно носить синтетическую одежду — достаточно походить по полу, покрытому обычным линолеумом или недорогим паласом, чтобы накопить на себе потенциал ничуть не меньше. Такое напряжение, конечно, может вывести из строя микросхемы и не только их — особенно чувствительны к нему полевые транзисторы с изолированным затвором. Так как заряду на выводе затвора у них стекать некуда, то все накопленное на вас напряжение будет приложено к тоненькому (несколько микро- или даже нанометров) промежутку между затвором и каналом, и не исключено, что изолирующий слой оксида кремния не выдержит такого «надругательства».

Поэтому при монтаже всегда следует соблюдать несколько правил: не носить синтетическую одежду и не использовать синтетические покрытия для пола и монтажного стола (профессиональные монтажные столы вообще покрывают заземленным металлическим листом). Неплохую гарантию дает заземление корпуса паяльника, только на практике в домашних условиях это осуществить сложно. Можно также привести еще несколько рекомендаций:

- не хвататься руками за выводы микросхем без нужды, при необходимости их формования взять корпус в левую (для левшей — в правую) руку так, чтобы пальцы касались выводов питания;
- первыми всегда следует припаивать выводы питания микросхемы (для дискретных транзисторов — эмиттер или исток);
- перед началом монтажа, особенно если вы только что переодевались, желательно подержаться руками за заземленный металлический предмет (водопроводный кран);
- при стирке рабочей одежды обязательно использовать антистатик.

Хорошую защиту также дает метод, при котором вы не впаиваете микросхему в плату непосредственно, а устанавливаете ее на панельку.

Насколько эти меры необходимы в повседневности? Случаи выхода микросхем из строя от статического электричества все же довольно редки, т. к. производители эту опасность учитывают, и для критичных ситуаций принимают меры по защите выводов. Самой распространенной мерой является установка защитных диодов, по два на каждый вывод, так, что один из них присоединен катодом к плюсу питания, а другой — анодом к минусу (рис. 6.5). Подобный прием позволяет иногда защитить микросхему и от неправильного включения питания — если плюс и минус питания на схеме рис. 6.5 поменять местами, то весь ток пойдет через диоды, и напряжение питания упадет до двойного падения напряжения на диоде, правда, тут весь вопрос в том, насколько долго диоды смогут выдержать прямой ток от источника. Для большей надежности иногда ставят и еще один отдельный защитный диод — прямо от питания до питания.

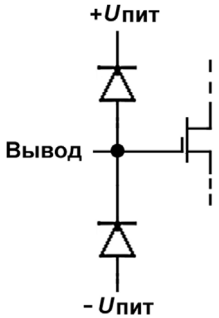


Рис. 6.5. Защита выводов микросхем от перенапряжения

Такой защитой снабжены наиболее капризные в этом отношении КМОП-микросхемы (см. *главу 8*). Но не все микросхемы имеют защиту, и не всегда она спасает, потому в этом отношении действует такое же правило, что и в обыденной жизни: ведь кирпичи тоже падают с крыш крайне редко. Особенно сами по себе. Но единственного такого случая во всей округе за последние 10 лет лично вам может оказаться более чем достаточно. Поэтому лучше не ходить под строящимися зданиями, не перебегать дорогу перед движущимся транспортом, не курить в постели, не пользоваться неисправными электроприборами и не хвататься за выводы микросхем голыми руками без нужды.

Операционные усилители

Операционные усилители — самые «главные» аналоговые микросхемы. Почти ни один современный аналоговый узел, как собранный на отдельных микросхемах, так и в составе других микросхем, без участия ОУ не обходится, исключение составляют лишь некоторые (не все) радиочастотные схемы.

Классическое определение гласит: *операционным усилителем (ОУ) называется дифференциальный усилитель постоянного тока (УПТ) с большим коэффициентом усиления*. Расшифруем. «Постоянного тока» — это не означает, что ОУ усиливают только сигналы частотой 0 Гц, это свидетельствует о том, что они могут усиливать сигналы, начиная с частоты 0 Гц. «С большим коэффициентом усиления» — это значит, что усиление действительно велико: хороший ОУ имеет коэффициент усиления порядка нескольких сотен тысяч или даже миллионов.

> ***Заметки на полях***
>
> Название «операционный» закрепилось за такими усилителями исторически, потому что во времена господства ламповой техники они использовались в основном для моделирования различных математических *операций* (интегрирования, дифференцирования, суммирования и пр.) в т. н. аналоговых вычислительных машинах. Других применений тех ОУ практически не было и быть не могло, потому что для достижения приемлемых характеристик не годилась не только ламповая, но и дискретно-транзисторная схемотехника. Настоящий переворот произошел только в середине 60-х годов после пионерских работ по конструированию интегральных ОУ уже упоминавшегося на этих страницах Роберта Видлара.

Разумеется, практически применять ОУ можно только в схемах с отрицательной обратной связью, за одним важным исключением, о котором чуть далее. В обычных схемах огромный коэффициент усиления приведет к тому, что без обратной связи такой усилитель будет находиться в состоянии, когда напряжение его выхода равно (или почти равно) одному из напряжений питания, положительному или отрицательному — такое состояние еще называют, по аналогии с транзисторами, *состоянием насыщения* выхода. В самом деле, чтобы получить на выходе напряжение 15 В, ОУ достаточно иметь на входе сигнал в несколько десятков микровольт, а такой сигнал всегда имеется — если это не наводка от промышленной сети или других источников, то достаточно и внутренних причин, о которых мы еще будем говорить.

А упомянутое исключение представляют так называемые *компараторы*: ОУ, которые предназначены для работы без отрицательной обратной связи и иногда даже наоборот, с положительной обратной связью. Они выполняют функцию точного сравнения уровней сигналов. Это одна из самых важных

областей использования ОУ, которая позволяет стыковать мир аналоговых и цифровых сигналов между собой. Например, ни одна из конструкций АЦП и ЦАП, которые мы будем рассматривать в *главе 10*, не обходится без компараторов.

Рассмотрим некоторые общие принципы построения аналоговых схем на ОУ.

Опасные связи

Согласно определению, *отрицательная обратная связь* — это связь выхода со входом, при которой часть выходного сигнала вычитается из входного. В противоположность отрицательной, в случае *положительной обратной связи* часть выходного сигнала суммируется с входным. Эти определения справедливы не только для усилителей и других электронных устройств, но и во всех других случаях, когда обратная связь имеет место. В общем случае их воздействие на некую систему можно описать так: наличие отрицательной обратной связи повышает ее устойчивость, наличие положительной — наоборот, ведет к неустойчивости.

Заметки на полях

Принцип действия обратных связей можно пояснить на примере классической взаимосвязи спроса и предложения в экономике. Предположим, у нас имеется некая фирма, которая состоит из производственных структур и каналов сбыта. На входе такой системы — задание на производство, на выходе — объем произведенной продукции. Сколько нужно производить товара? Естественно, столько, сколько его могут потребить. В идеальной системе происходит следующее: фирма производит один экземпляр товара и, как только его покупают, немедленно выдает на прилавок следующий экземпляр. Если фирма произведет два экземпляра, и один из них на прилавке задержится, то производство приостанавливается до тех пор, пока этот экземпляр не купят. Здесь мы наблюдаем типичное действие отрицательной обратной связи, роль которой играет спрос: лежащий на прилавке экземпляр товара как бы вычитается из задания на производство, и оно приостанавливается. Такая система очень устойчива и к тому же обладает множеством приятных свойств: не имеет перерасхода энергии и материалов, не приводит к перепроизводству или, в пределах мощности производства, наоборот, к дефициту.

Но в большинстве случаев в реальной жизни все обстоит гораздо сложнее — и прямых и обратных связей всегда существенно больше одной, реакция на спрос не может быть мгновенной, да и система не изолирована от всей остальной экономики. Что произойдет с нашей идеальной системой, если производство не может остановиться и возобновить работу мгновенно, или если сведения об изменении спроса поступают не сразу, а с некоторым запаздыванием? Предположим, фирма делает 10 экземпляров товара в день, и указанное запаздывание составляет также 1 день. Допустим, в какой-то из дней спрос упал на 2 штуки. Из-за запаздывания реакции на изменение спроса в этот день фирма произведет по-прежнему 10 штук, так что на следующее утро на прилавке их окажется 12. Если в этот день спрос, как и раньше, будет составлять 8 штук,

то к следующему утру на прилавке окажутся те же 12 экземпляров (8 произведенных — фирма отреагировала на изменение, плюс 4 оставшихся от предыдущего дня). В этот день фирма отреагирует и произведет всего 4 экземпляра. Но предположим, что в этот же день спрос внезапно возрос и составил 12 экземпляров, т. е. все имеющиеся раскуплены. На следующее утро на прилавке будет всего 4 штуки (произведенных накануне) и 8 из 12 гипотетических клиентов уйдут неудовлетворенными. Им предложат зайти через сутки, и на следующий день фирма вынуждена будет произвести 8 + 12 = 20 экземпляров товара! Легко продолжить эту цепочку рассуждений дальше и сообразить, что будет происходить с производством и удовлетворением спроса. Система будет «раскачиваться» все сильнее и сильнее, пока в дело не вступят естественные ограничения: объем производства не может быть меньше нуля и больше фактической мощности производства (в случае электронных систем роль таких ограничений выполняет напряжение питания или достижимая мощность выходного каскада усиления). Работоспособность же системы будет полностью нарушена, т. к. отрицательная обратная связь превратилась в положительную.

Отрицательная обратная связь в усилителях позволяет точно установить коэффициент усиления и приводит еще ко многим приятным улучшениям схемы. Попробуем разобраться, почему это так, и каково влияние характеристик реальных ОУ на параметры схемы.

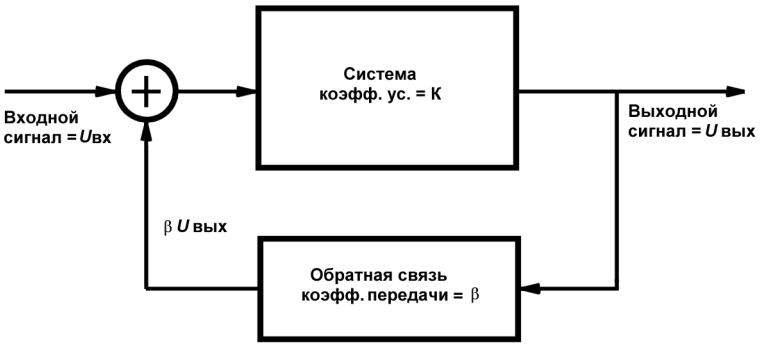


Рис. 6.6. Обобщенная схема системы с отрицательной обратной связью

На рис. 6.6 приведена обобщенная схема некоторой системы, охваченной отрицательной обратной связью. Коэффициент усиления К основной системы обычно больше единицы. Для ОУ это и есть его собственный коэффициент усиления, который может составлять сотни тысяч, как мы говорили. Коэффициент передачи по обратной связи β обычно, наоборот, меньше единицы (хотя ничто не мешает нам сделать его и больше единицы, тогда вся система будет не усиливать, а ослаблять сигнал).

Если разорвать петлю обратной связи, то сигнал на выходе $U_{вых}$ был бы равен $KU_{вх}$ (огромной величине — разумеется, в реальной системе напряжение

питание его бы ограничило, но для наших рассуждений это неважно). Но при действии обратной связи это не так. На вход выходной сигнал передается с коэффициентом ослабления β, и сигнал после сумматора, т. е. на входе основной системы, будет равен $U_{вх} - \beta U_{вых}$ (минус, т. к. обратная связь отрицательная). Этот сигнал передается на выход с коэффициентом K, т. е. $U_{вых} = K(U_{вх} - \beta U_{вых})$. Отсюда $U_{вых} = KU_{вх}/(1 + K\beta)$, т. к. коэффициент передачи $K_{ус}$ всей системы по определению равен $U_{вых}/U_{вх}$. В результате для него получаем следующую формулу:

$$K_{ус} = \frac{U_{вых}}{U_{вх}} = \frac{KU_{вх}}{U_{вх}(1 + K\beta)} = \frac{K}{(1 + K\beta)}. \quad (6.1)$$

Отсюда следует интересный вывод: если K много больше единицы (а в случае ОУ это действительно так с огромной степенью точности), то единицу в формуле (6.1) можно не принимать во внимание, и коэффициент передачи будет выражаться простым соотношением:

$$K_{ус} = 1/\beta. \quad (6.2)$$

Формула (6.2) означает, что *коэффициент передачи входного сигнала на выход будет определяться только параметрами обратной связи, и никак не зависит от характеристик ОУ*. Причем чем выше собственный коэффициент усиления системы K, тем точнее соблюдается это положение.

Введение отрицательной обратной связи приводит также еще к некоторым последствиям. Для практических целей достаточно их просто запомнить, не углубляясь в математические выкладки:

- входы ОУ не потребляют тока (входное сопротивление ОУ практически равно бесконечности, точнее — увеличивается по сравнению с ОУ без обратной связи в Kβ раз);
- ОУ с отрицательной обратной связью всегда «стремится» сделать так, чтобы потенциалы на его входах были равны между собой.

Характеристики конкретной схемы определяются соотношением собственного коэффициента усиления ОУ и коэффициента передачи системы с замкнутой обратной связью — чем выше это соотношение, тем ближе схема к идеалу. Интересно, что если на практике для обеспечения фактической независимости коэффициента усиления схемы от характеристик ОУ достаточно иметь собственный коэффициент усиления всего в несколько тысяч, то для получения, например, действительно высокого входного сопротивления (измеряемого гигаомами и выше), приходится увеличивать K до указанных величин в сотни тысяч и более.

Отметим также сразу, что введение обратной связи в указанной выше степени уменьшает и выходное сопротивление всего усилителя, которое становится

очень близким к нулю — точнее, примерно равным $R_{вых}/(1 + K\beta)$, где $R_{вых}$ — собственное выходное сопротивление ОУ, лежащее обычно в диапазоне сотен ом. Так что выходное сопротивление получается порядка 1 миллиома. Только не забывайте, что мощность выходного каскада ограничена, и если вы его перегрузите, то от падения напряжения на нагрузке вас уже никакая обратная связь, естественно, не спасет: ОУ просто не сможет отдать того тока, который требуется. Это ограничивает величину сопротивления нагрузки рядовых ОУ на уровне порядка килоом. Меньшие нагрузки обычно допустимы (вплоть до к. з.), но обратная связь уже работать не будет.

Заметки на полях

Из изложенных ранее рассуждений относительно экономической модели обратной связи ясно, что система с обратной связью может быть неустойчивой. Обсуждение теории устойчивости таких систем (скажем, известного метода Найквиста) увело бы нас слишком далеко, однако практические меры в основном сводятся к тому, чтобы ограничить коэффициент усиления исходной системы и/или глубину обратной связи на таких частотах, когда отрицательная обратная связь начинает превращаться в положительную. Другими словами, при амплитуде сигнала обратной связи, равной или большей значения входного сигнала, фазовый сдвиг между ними не должен достигать 180° (поглядите на графики суммирования синусоидальных сигналов в *главе 2*, чтобы лучше понять, в чем тут дело). Причем наибольшую опасность несет в себе режим с установленным коэффициентом усиления, равным единице (т. е. включение ОУ по схеме повторителя), т. к. на вход поступает бóльшая часть выходного сигнала. Роберт Видлар был сторонником того, чтобы переложить заботу о коррекции на плечи пользователей, и первые его конструкции ОУ, например, μA702, выпускавшийся в нашей стране под названием 140УД1[1] или получивший широкую известность μA709, имели специальные выводы для коррекции с помощью внешних резисторов и конденсаторов. Практически же этим никто не пользовался (подобно тому, как подавляющее большинство пользователей компьютерных программ работает с установками, введенными в них разработчиками по умолчанию) и такая возможность только приводила к необходимости введения в схему лишних компонентов, так что в настоящее время выводы для внешней коррекции сохранились лишь для некоторых моделей высокочастотных ОУ, где полоса частот действительно является критичным фактором.

Кстати, а каковы в свете всего изложенного могут быть рекомендации нашим предпринимателям из производственной фирмы? Они совершенно аналогичны методам для обеспечения стабильности ОУ: нужно ограничить глубину обратной связи и коэффициент усиления на высоких частотах. Проще говоря, им следует при наличии запаздывания не пытаться реагировать на каждый проданный или непроданный экземпляр, а выпускать некое среднее количество в сутки, изменяя его только, когда изменился средний объем продаж за промежуток времени, значительно больший суток — это и равносильно ограничению усиления на высоких частотах.

[1] Префикс «К» в названии отечественных микросхем, обозначающий их принадлежность к бытовому/коммерческому диапазону температур, мы будем в этой книге опускать, подробнее см. *главу 8*.

Базовые схемы усилителей на ОУ

Анализ схемы *неинвертирующего усилителя* (рис. 6.7, *а*) элементарно прост: исходя из приведенных правил $U_{ос} = U_{вх}$, т. е. $U_{вх} = U_{вых} \cdot R2/(R1 + R2)$. Тогда коэффициент усиления $K_{ус} = U_{вых}/U_{вх} = (R1 + R2)/R2 = 1 + R1/R2$.

Единица, которая плюсуется к отношению резисторов обратной связи в выражении для коэффициента усиления — очень важное дополнение, потому что если убрать в схеме неинвертирующего усилителя резистор R2 (т. е. принять его равным бесконечности), то отношение резисторов станет равным нулю, а $K_{ус}$ — равным единице. Соответствующая схема, показанная на рис. 6.7, *в*, и есть тот самый повторитель, которого так «боялся» Видлар. Зачем она нужна, если ничего не усиливает? Эта схема обладает одним бесценным свойством: ее входное сопротивление равно практически бесконечности, а выходное — нулю (в пределах, конечно, мощности выходного каскада, как мы уже говорили). Поэтому повторитель очень часто используют в случаях, когда нужно согласовать источник сигнала с высоким выходным сопротивлением с низкоомным приемником.

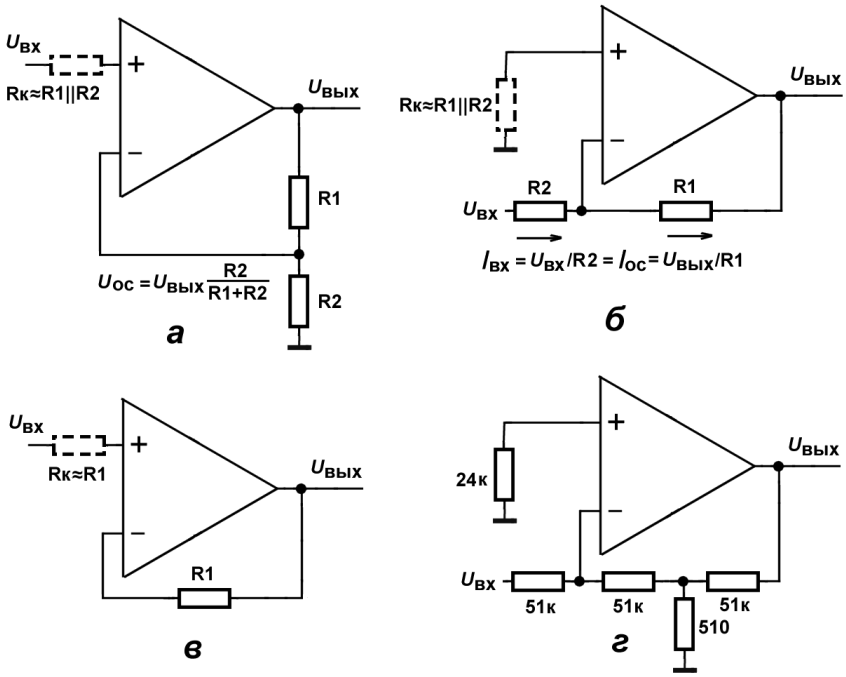


Рис. 6.7. Базовые схемы на ОУ:
а — неинвертирующий усилитель; *б* — инвертирующий усилитель; *в* — повторитель; *г* — инвертирующий усилитель с высоким коэффициентом усиления

В неинвертирующем усилителе обратная связь носит название «обратной связи по напряжению». В отличие от него, в *инвертирующем усилителе* (рис. 6.7, *б*) обратная связь имеет характер «обратной связи по току», и вот почему. Так как здесь неинвертирующий вход имеет потенциал «земли», то и инвертирующий тоже *всегда будет иметь такой же потенциал*. Будем считать, что питание у нас нормальное, симметрично-двуполярное. Тогда если в схеме рис. 6.7, *б* инвертирующий вход имеет всегда потенциал «земли», то от входа через резистор R2 потечет некий ток ($I_{вх}$). Так как мы договорились, что сам вход ОУ тока не потребляет, то этот ток должен куда-то деваться, и он, в полном соответствии с первым законом Кирхгофа, потечет через резистор R1 на выход ОУ. Таким образом, входной ток ($I_{вх}$) и ток обратной связи ($I_{ос}$) — это один и тот же ток. Причем потенциал выхода ОУ вынужденно станет противоположным по знаку потенциалу входа, иначе току некуда будет течь. Кстати, подавать именно нулевой потенциал на неинвертирующий вход совершенно необязательно, например, если у вас однополярный источник питания, то на неинвертирующий вход подается потенциал «искусственной средней точки».

Чему равен коэффициент усиления такой схемы? Так как $U_{вх}/R2 = U_{вых}/R1$, то $K_{ус} = U_{вых}/U_{вх} = R1/R2$. Без всяких дополнительных единиц, как в неинвертирующей схеме, т. е. R2 в данном случае есть необходимый элемент схемы и не может быть равным ни нулю (тогда вход ОУ просто замкнет выход источника на «землю»), ни бесконечности — за исключением того случая, если источник сигнала сам по себе представляет источник тока, а не напряжения. Вот тогда R2 из схемы можно (и нужно) исключить и подать токовый сигнал прямо на вход ОУ.

Заметьте, кстати, что похожее выражение для коэффициента усиления мы получали при рассмотрении транзисторного усилительного каскада (рис. 3.7), где усиление было равно отношению коллекторной нагрузки к сопротивлению в эмиттерной цепи. Это обусловлено тем, что в транзисторном каскаде также имеет место обратная связь (см. *главу 3*).

Подробности

Максимальное значение входного и выходного напряжений ОУ не всегда может быть равно положительному или отрицательному напряжению питания (как правило, оно меньше его на величину порядка 0,5—1,5 В). Однако многие современные изделия это все же позволяют и допустимое выходное (входное) напряжение у них достигает значений напряжения питания. Это свойство в западной технической документации обозначается как Rail-to-Rail (т. е. «от шины до шины») и на него нужно обращать внимание при выборе ОУ.

Если входное сопротивление неинвертирующего усилителя равно практически бесконечности, то у инвертирующего оно почти в точности равно R2.

Но входы реального ОУ все же потребляют ток, хотя и очень небольшой (называемый *током смещения*). Ток смещения на инвертирующем входе (в любой из двух схем) создаст падение напряжения на резисторе обратной связи и оно воспринимается как часть входного сигнала: если этот ток равен, к примеру, 0,2 мкА (казалось бы — так мало!), то при сопротивлении R1 = 1 МОм напряжение на выходе при отсутствии напряжения на входе достигнет 0,2 В. Как обычно, в подобных случаях важно не само по себе смещение, а его температурная нестабильность. Борьба с этим явлением может вестись в трех направлениях: во-первых, не следует использовать в цепочке обратной связи сопротивления большого номинала, стандартный диапазон их — от килоом до десятков килоом. Если же при необходимости сохранить достаточно высокое входное сопротивление инвертирующего усилителя при большом коэффициенте усиления применение высокоомных резисторов желательно, то предпочтительнее схема, показанная на рис. 6.7, *г*. В данном случае вся цепочка в обратной связи работает, как один резистор с номинальным сопротивлением 5,1 МОм, и коэффициент усиления равен 100 при входном сопротивлении 50 кОм.

Во-вторых, в схему следует вводить компенсирующий резистор $R_к$ (на рис. 6.7, *а*—*в* он показан пунктиром) — падение напряжения от тока смещения по неинвертирующему и инвертирующему входам на нем отчасти компенсируются. Тогда будет уже не столь важен сам ток смещения, сколько разница их, потребляемых по каждому из входов усилителя, которая определенно меньше каждого из токов. Кроме токов смещения, на работу реального ОУ влияет и т. н. *напряжение сдвига*, обусловленное неидентичностью параметров входных каскадов.

На практике, если эти явления критичны (а это далеко не всегда так), стоит подобрать более дорогой, но и более точный прецизионный ОУ. К рядовым «ширпотребовским» типам ОУ относятся старинные, но до сих пор производящиеся 140УД7 (μA741), 140УД20 (dial — сдвоенный, т. е. содержащий два ОУ в одном корпусе), LM321 (single — одинарный), LM358 (также сдвоенный), LM324 (quad — счетверенный). При этом обычные усилители (LM321, LM324, LM358) имеют широчайший диапазон напряжений питания (до ±16 В). Существует их модификация, выпускающаяся фирмой MAXIM/DALLAS, с добавлением буквы X к названию (LMX321), у которой напряжение питания снижено всего до 7 В (суммарно), однако выходное напряжение имеет полный размах (Rail-toRail) — фактически это совсем другие ОУ. Такие нюансы нередки, потому встретив знакомую микросхему, но с незнакомым индексом, обязательно следует проверить ее характеристики по документации на сайте производителя, иначе можно крупно «пролететь».

К прецизионным ОУ относятся, например, надежные и удобные MAX478 (сдвоенный) и MAX479 (счетверенный), также отличающиеся исключительно широким диапазоном допустимых напряжений питания: от ±2,2 до ±18 В. Они имеют высокие показатели по точности, но работают очень медленно и не допускают полного размаха напряжений по выходу. В настоящее время эти микросхемы не выпускаются (хотя их еще можно спокойно приобрести), причем адекватной замены у фирмы MAXIM нет, и лучше употреблять аналогичные изделия других фирм, например, серию AD820—AD824 фирмы Analog Devices, которая существенно быстрее и к тому же имеет полный Rail-to-Rail размах напряжения по выходу. По цоколевке они (как и большинство других ОУ) полностью взаимозаменяемы при условии идентичности корпуса. MAX4236 — пример прецизионного усилителя, который работает при напряжениях питания до 5,5 В, зато с полным Rail-to-Rail размахом напряжения по выходу, что хорошо стыкуется с цифровыми схемами, сейчас таких ОУ выпускается очень много. Особо высокими характеристиками, в том числе по быстродействию, отличаются относительно дорогие ОУ с цифровой стабилизацией: отечественный 149УД24, а также MAX420, MAX430, ICL7652 и др.

Дифференциальные усилители

Кроме всего прочего, ОУ имеют замечательное свойство подавлять синфазный входной сигнал. *Синфазный сигнал*, в отличие от обычного, дифференциального — это напряжение, которое действует на оба входа сразу (см. также *главу 3*). Это свойство приводит не только к возможности выделять полезный сигнал на фоне значительных наводок, но и, что иногда еще важнее, к подавлению нестабильности источника питания, поскольку изменение напряжения питания равносильно действию синфазного входного сигнала.

На рис. 6.8, *а* показана схема простейшего *дифференциального усилителя*. Делитель R3, R4 по неинвертирующему входу служит сразу двум целям: во-первых, он выравнивает входные сопротивления по входам (нетрудно показать, что т. к. потенциалы самих входов ОУ равны, то будут равны и входные сопротивления, естественно, при указанном на схеме равенстве соответствующих резисторов), во-вторых, что еще важнее, он делит входной сигнал в таком соотношении, чтобы коэффициенты усиления по инвертирующему и неинвертирущему входам сравнялись между собой. Именно при этом условии коэффициент ослабления синфазного сигнала (КОСС) будет максимальным. Для того чтобы получить действительно высокий КОСС (ослабление синфазного сигнала ~10 000 раз, т. е. на 80 дБ, о децибелах см. далее), согласование сопротивлений должно быть как можно более точным, и в такой схеме

следует применять прецизионные резисторы из ряда с погрешностью, не превышающей, по крайней мере, 0,1%, причем лучше всего их еще и дополнительно подобрать по строгому равенству номиналов. Тогда вы действительно сможете без проблем выделить полезный сигнал в 1 мВ на фоне наводки в 1 В.

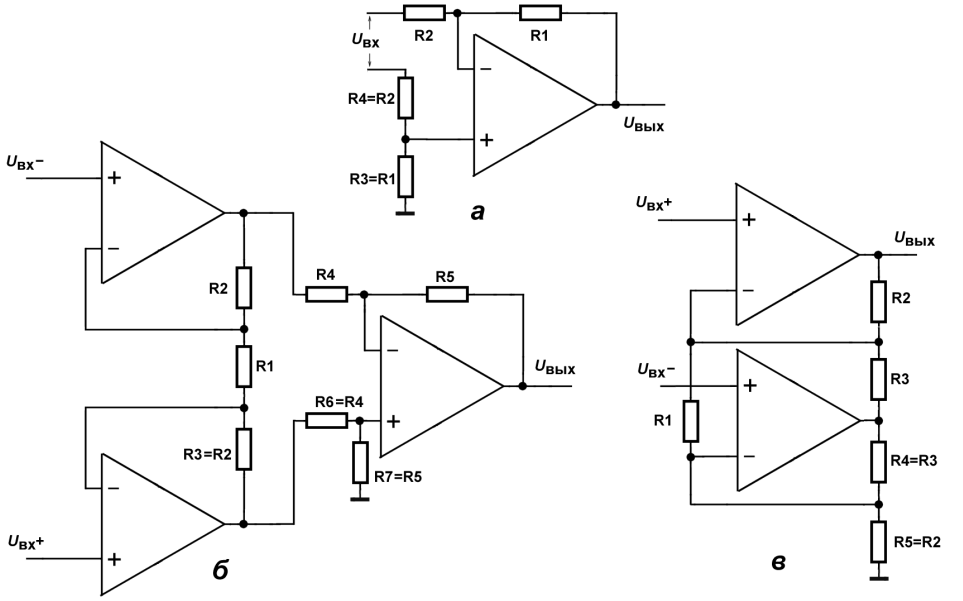


Рис. 6.8. Схемы дифференциальных усилителей:
а — простой дифференциальный усилитель;
б — классический инструментальный усилитель;
в — упрощенный инструментальный усилитель

Понятно, что заниматься подобными подборками при массовом производстве не с руки, да и входным сопротивлением наш простейший дифференциальный усилитель отличается не в лучшую сторону, потому на практике эту схему применяют редко. Ко всему прочему, в ней еще и почти невозможно изменять коэффициент усиления в процессе работы, если вдруг это понадобится, т. к. для этого потребуется менять одновременно два резистора, а куда денется в таком случае наше согласование?

Для того чтобы увеличить входное сопротивление, целесообразно добавить еще пару ОУ по каждому входу, включенных повторителями, как показано на рис. 6.8, *б*. Причем к увеличению габаритов и стоимости схемы это прак-

тически не приводит, т. к. специально для таких целей выпускают упоминавшиеся ранее сдвоенные и счетверенные ОУ в одном корпусе, почти не отличающиеся по цене от одинарных.

Так мы добьемся увеличения входного сопротивления по обеим входам почти до бесконечности, а что с КОСС? Если просто добавить повторители, то с ним ничего не произойдет и точное согласование резисторов по-прежнему будет необходимо. Выход из этой ситуации очень простой: достаточно установить еще один резистор (на схеме рис. 6.8, *б* он обозначен как R1). В результате получаем классическую схему т. н. *инструментального усилителя*. Здесь также целесообразны прецизионные резисторы (в целях обеспечения температурной стабильности), но подбора уже не требуется. Коэффициент усиления такого усилителя определяется по следующей формуле (при указанных на схеме соотношениях резисторов):

$$U_{\text{вых}} = (U_{\text{вх}+} - U_{\text{вх}-})\frac{\text{R5}}{\text{R4}}\left(1 + 2\frac{\text{R2}}{\text{R1}}\right).$$

Изменять его, не нарушая ничего в работе усилителя, можно одним резистором R1. Кстати, резисторы компенсации тока смещения здесь не нужны, т. к. эти токи по общим для системы инвертирующему и неинвертирующему входам взаимно компенсируют влияние друг друга, тем более, если ОУ расположены на одном кристалле.

Если мы люди не гордые, и большой КОСС нам не требуется (когда помеха мала по сравнению с полезным сигналом), то можно упростить схему инструментального усилителя. За исключением КОСС, схема на рис. 6.8, *в* обладает всеми достоинствами классической, но содержит на один ОУ меньше (значит, можно использовать сдвоенный, а не счетверенный чип), да и резисторов там поменьше. При указанных на схеме соотношениях резисторов выходное напряжение такого усилителя будет равно

$$U_{\text{вых}} = (U_{\text{вх}+} - U_{\text{вх}-})\left(2\frac{\text{R2}}{\text{R1}} + \frac{\text{R2}}{\text{R3}} + 1\right).$$

Заметки на полях

В подобных усилителях решительно не рекомендуется подгонять ноль выходного напряжения, нарушая баланс резисторов, например R4/R5 и R6/R7 в схеме рис. 6.8, *б*. В то же время иногда установка нуля необходима, т. к. начальное смещение выхода может быть, например, отрицательным (и не только из-за сдвига рабочей точки самих ОУ, но и по причине начального смещения у источника сигнала), и в случае, если весь диапазон изменения выходного напряжения должен располагаться в положительной области (скажем, при подаче его куда-нибудь на вход аналого-цифрового преобразователя, не «понимающего» отрицательных напряжений), вы можете потерять заметный «кусок» диапазона. Иногда для установки нуля рекомендуют воспользоваться корректирующими

выводами одного из входных ОУ, но для стабильности схемы это еще хуже, чем корректировка внешними резисторами, тем более что в сдвоенных и счетверенных вариантах эти выводы обычно отсутствуют, просто вследствие элементарной нехватки контактов корпуса. В действительности установку нуля лучше осуществлять со стороны входов, подмешивая к одному из входных напряжений через развязывающий резистор небольшое напряжение коррекции, как это делается в схемах сумматоров, к которым мы сейчас перейдем.

Другие распространенные схемы на ОУ

Как уже упоминалось, операционные усилители получили свое название от того, что они применялись для моделирования математических операций, которое выполнялось т. н. *аналоговыми вычислительными машинами*. Одной из основных схем в них был аналоговый сумматор, который представляет собой просто усилитель (инвертирующий или нет), на вход которого подается несколько напряжений через отдельные резисторы. При этом напряжения будут суммироваться с весами, пропорциональными значениям этих резисторов.

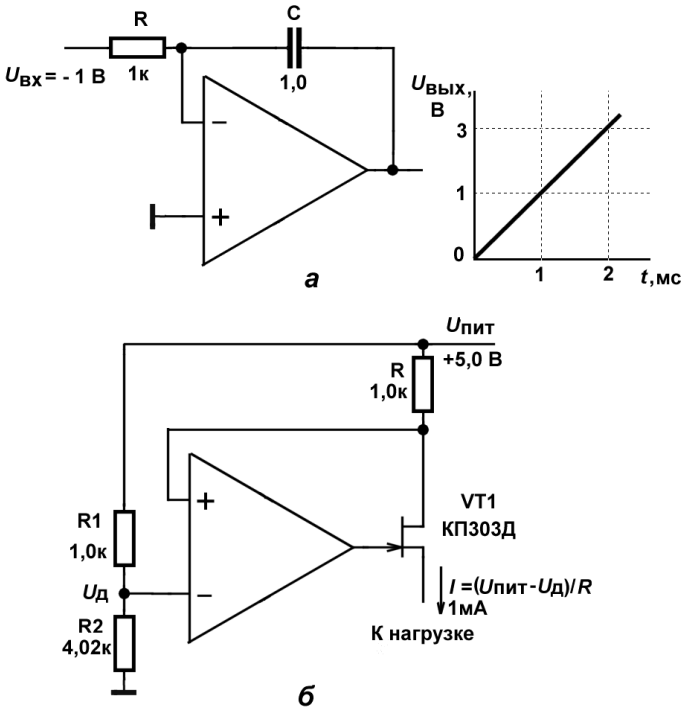


Рис. 6.9. Распространенные схемы на ОУ: *а* — интегратор; *б* — источник тока

Другой необходимой составляющей таких машин был интегратор на ОУ, схема которого приведена на рис. 6.9, *а*. Этот интегратор, в отличие от интегрирующей RC-цепочки из *главы 2*, действительно осуществляет операцию интегрирования в корректной форме. Например, если подать на его вход постоянное напряжение (отрицательное), то напряжение на выходе будет линейно возрастать (интеграл от константы есть прямая линия), с наклоном, равным $U_{вх}/RC$ (вольт в секунду). Входной сигнал можно подать и на неинвертирующий вход — получим неинвертирующий интегратор. Можно также объединить интегратор с сумматором, тогда интегрирование будет осуществляться по сумме входных напряжений с соответствующими весовыми коэффициентами. Интеграторы, как и сумматоры, используются и по сей день в различных схемах (см. *главу 10*).

Еще одна очень полезная схема (рис. 6.9, *б*) представляет собой почти идеальный источник тока с выходным сопротивлением, равным бесконечности. Здесь возможно однополярное питание, как и показано на схеме. Ток можно задавать как соотношением резисторов делителя R1, R2, так и резистором R. Обратите внимание, что отрицательная обратная связь подается на неинвертирующий выход ОУ, т. к. здесь использован полевой транзистор с *n*-каналом и стабилизируется его стоковое напряжение, которое есть инверсия напряжения на затворе. Если взять транзистор с *p*-каналом, то его в этой схеме нужно подключить наоборот: стоком в направлении нагрузки, а обратную связь, снимаемую с истока, подавать на инвертирующий вход. Для высокой стабильности тока в этой схеме требуется столь же высокая стабильность напряжения питания, поэтому если важна абсолютная величина тока, то схему (и делитель R1/R2, и резистор R, а не только делитель!) приходится питать от отдельного прецизионного стабилизатора. К счастью, стабильность в абсолютном понимании требуется не всегда, часто необходима стабильность некоей величины лишь относительно других параметров схемы. Кстати, от характеристик транзистора стабильность тока никак не зависит, единственное требование — чтобы начальный ток стока превышал установленный выходной ток схемы. Если применить не полевой, а биполярный транзистор, то будет иметь место некоторая зависимость выходного тока из-за изменений базового тока транзистора (т. к. коллекторный ток отличается от эмиттерного на величину тока базы), потому в таких источниках предпочтительнее именно полевые транзисторы.

Немало интересных практических применений ОУ вы можете найти в многочисленной литературе, например, в классических трудах [5] и [6], а также в Интернете. А сейчас мы рассмотрим две полезные схемы, которые хорошо иллюстрируют особенности использования ОУ на практике.

Регулятор оборотов вентилятора[2]

Крупный недостаток современных компьютеров заключается в том, что они шумят — приходится только удивляться периодически возникающим спорам по поводу нюансов звучания той или иной акустической системы, если уровень шума системного блока не опускается ниже 30—40 дБ. Определяющий вклад в этот шум вносят вентиляторы блока питания и процессора. Частично решить проблему можно, если заменить дешевые вентиляторы на более дорогие, с лучшей конфигурацией лопастей и более надежными подшипниками. Но чтобы снизить шум до предельно возможного уровня, следует применять устройства регулирования скорости вращения — зачем вентилятору «завывать» на полных оборотах, если температура находится в пределах допустимого? Многие современные чипсеты способны сами регулировать обороты, вместе с тем в эксплуатации полно дешевых машин, в которых такой регулировки нет.

Заметки на полях

Простейший прием для снижения шума — просто включить последовательно с вентилятором резистор. Производители «кулеров», естественно, «закладываются» на наихудшие температурные режимы, и типовой вентилятор для процессорного радиатора имеет порядка 2300—2700 об/мин. На практике, если у вас достаточно просторный корпус, их можно безболезненно снизить до примерно 1700 об/мин, для чего у обычного вентилятора 60—90 мм следует в разрыв питания (красный провод) включить резистор сопротивлением от 51 до 100 Ом и мощностью не менее 0,5 Вт. Величина сопротивления подбирается экспериментально, обороты и температура процессора контролируются с помощью соответствующей программы, обычно прилагаемой к каждой материнской плате. При экспериментах не торопитесь — дайте процессору выйти на стабильный температурный режим, еще лучше — нагрузите его какой-нибудь громоздкой задачей, вроде архивации крупного файла или текстового поиска среди большого количества документов.

На рис. 6.10 приведена схема пропорционального регулятора оборотов вентилятора с защитой от перегрева. Защита нужна потому, что 99% времени процессор занят менее чем наполовину, но в экстремальных задачах, и к тому же при повышенной температуре наружного воздуха, он может греться сильнее, тогда целесообразно запустить вентилятор на «полную катушку».

Датчиком температуры Rt служит *термистор* — полупроводниковый терморезистор, обладающий большим отрицательным температурным коэффициентом сопротивления (порядка 3—4% на каждый градус). Из-за нелинейности термисторы трудно использовать в качестве датчиков для измерения

[2] Конструкция опубликована автором в журнале «Радио», 2002, № 8.

температуры, но для не слишком точных регуляторов они подходят очень хорошо. Термистор (с отрицательным коэффициентом — не перепутайте с *позисторами*, которые имеют положительный коэффициент, но часто также продаются под названием «термисторы») годится абсолютно любого типа, но предпочтительнее те, что оформлены в корпусах, удобных для обеспечения хорошего теплового контакта с радиатором, например, M703 фирмы EPSOS, имеющие отверстие для крепежного винта, или отечественные фольговые термисторы ТРП, которые легко приклеивать.

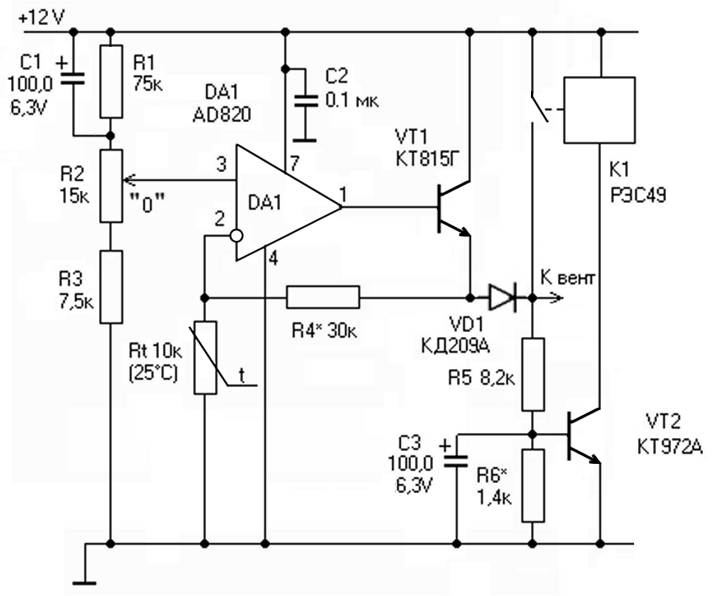


Рис. 6.10. Схема пропорционального регулятора оборотов вентилятора

Работает схема очень просто. Термистор здесь, как видите, включен в цепь отрицательной обратной связи ОУ, выходной каскад которого для повышения мощности дополнен эмиттерным повторителем на транзисторе VT1. При повышении температуры сопротивление термистора снижается и напряжение на выходе эмиттерного повторителя растет, соответственно увеличивается и число оборотов вентилятора. Если температура продолжает повышаться, срабатывает блок аварийного включения, собранный на резисторах R5, R6, транзисторе VT2 и реле K1. При превышении установленного порога транзистор открывается, и контакты реле подключают вентилятор напрямую к питанию 12 В. Схема при этом «защелкивается» — вывести ее из этого состояния можно только выключением питания.

Конденсатор C1 обеспечивает начальный запуск: когда радиатор холодный, напряжения на выходе схемы может не хватить для того, чтобы стронуть вентилятор с места, а некоторые процессорные платы могут вообще не запуститься, если вентилятор не крутится. При включении питания C1 разряжен, и, заряжаясь, закорачивает резистор R1, в результате чего на вентилятор первоначально подается повышенное напряжение, достаточное для запуска, а раскрученный вентилятор потом уже будет работать нормально и при пониженном напряжении. При этом конденсатор C3 предотвращает срабатывание схемы защиты (если она все же будет срабатывать при запуске, то его номинал следует увеличить).

Во избежание всяческих неприятностей в компьютер следует устанавливать уже отрегулированную схему. Она настраивается таким образом, чтобы при температуре радиатора около 60 °C напряжение на питании вентилятора достигало 10,5 В (хотя ОУ AD820 выдает полный размах вплоть до напряжения питания, выше примерно 10,8 В его увеличить не позволит цепь «база-эмиттер VT1 — диод VD1»). Соответственно, при таком напряжении уже должна срабатывать защита. Перед настройкой временно отключите сопротивление R5 делителя аварийного отключения и конденсатор C1 схемы начального запуска, подключите схему к источнику питания 12 В, поместите термистор Rt в среду с комнатной температурой и с помощью потенциометра R2 установите на эмиттере VT1 напряжение около 4—5 В (при установленном напряжении раскрученный вентилятор не должен останавливаться).

Затем поместите термистор в воду (завернув его в резиновый напальчник или поместив в узкий металлический стаканчик, что надежнее) с температурой 60—65 °C и подбором резистора обратной связи R4 установите на эмиттере VT1 напряжение около 10,5 В. Эту процедуру придется повторить несколько раз до получения нужных значений при обеих температурах. Затем подключите резистор R5 и, погружая термистор в среду с температурой выше 60 °C, подберите значение сопротивления R6 между базой и эмиттером VT2 так, чтобы аварийная схема срабатывала при достижении напряжения на вентиляторе ~10,5 В.

Если вы не найдете подходящий термистор Rt с сопротивлением 10 кОм, как на схеме (например, фольговые не встречаются с номиналом более 1 кОм), то его можно заменить на любой другой в пределах от 1 до 50 кОм, при этом R4 также надо соответственно изменить. ОУ типа AD820 можно заменить и на рядовые модели (140УД7), но при этом предельно достижимый уровень напряжения на выходе значительно снизится (примерно до 9,5 В). Транзистор VT1 — КТ815Г или КТ815Б, лучше подобрать экземпляр с коэффициентом передачи по току не менее 100. Вместо реле РЭС49 можно поставить любое малогабаритное на напряжение 12 В.

Схема собирается на небольшой макетной плате размерами примерно 30×100 мм и устанавливается в любом месте корпуса компьютера подальше от тепловыделяющих деталей. Прикрутив или приклеив вынесенный на скрученных проводах термистор к радиатору, далее необходимо разорвать цепь питания вентилятора (красный провод — не перепутайте с желтым, по которому идет сигнал числа оборотов), подключить его к выходу схемы, а также подключить схему к питанию 12 В (можно к любому желтому проводу из блока питания ПК, а можно и к красному проводу бывшего питания вентилятора со стороны материнской платы). В блок питания компьютера подобное устройство встраивается аналогично.

Терморегулятор для воды[3]

Обычное устройство для нагревания воды при отсутствии центрального горячего водоснабжения (например, в дачном домике) состоит из бака на 5—20 л со встроенным электронагревателем (ТЭНом) мощностью 1—2 кВт. Использовать его без терморегулятора неудобно — приходится внимательно следить за тем, чтобы вода не закипела, да и получается она либо слишком горячая, либо наоборот — недогретая.

На рис. 6.11 изображена схема термостата для нагревания воды. Она только на вид кажется сложной, на самом деле отличается от предыдущей схемы только тем, что работает не в пропорциональном (число оборотов плавно меняется с температурой), а в ключевом режиме (включено-выключено). Так как вода имеет большую тепловую инерционность, то пропорциональное регулирование тут ни к чему. Здесь мы познакомимся с компараторами (как мы знаем, это ОУ без обратной связи), а также с практическим применением оптоэлектронных (электронных) реле.

Множество разных деталей обусловлено тем, что схема имеет несколько режимов работы:

- ❒ автоматический термостатирующий;
- ❒ автоматический однократный с отключением по достижении нужной температуры («режим электрочайника»);
- ❒ ручной с подключением ТЭНа напрямую к сети.

Сначала отвлечемся от режимов и посмотрим, как работает основная схема регулирования. Здесь имеется точно такой же, как в регуляторе оборотов, термисторный датчик с отрицательным коэффициентом. После включения

[3] Конструкция опубликована автором в журнале «Радио», 2004, № 9.

питания, если температура еще ниже заданной, на выходе компаратора DA1 устанавливается уровень напряжения, близкий к нулю, причем усилительный транзистор здесь не нужен, поскольку компаратор 554СА3 специально приспособлен для подобных надобностей, и имеет на выходе довольно мощный (до 50 мА) транзистор с открытым коллекторным выводом.

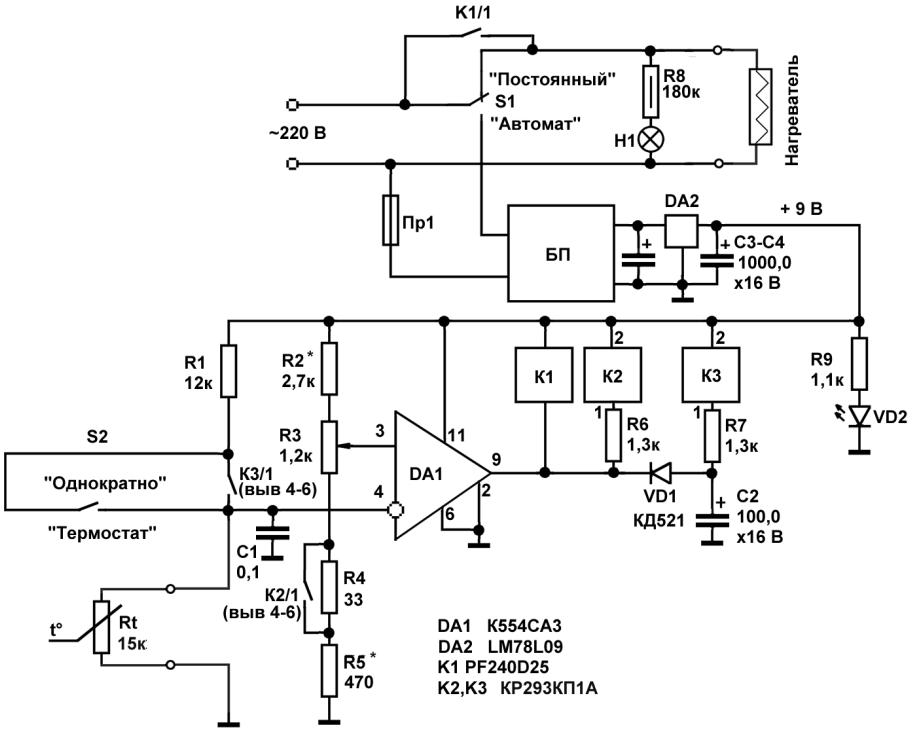


Рис. 6.11. Схема термостата для нагревания воды

В результате в первый момент срабатывает не только основное мощное реле К1, но и реле К2 (токоограничивающих резисторов в реле этого типа нет, и с этой целью установлены резисторы R6 и R7). Контакты его замкнуты, и резистор R4 не участвует в работе схемы. По мере увеличения температуры напряжение на датчике падает и в какой-то момент времени выходной транзистор компаратора разрывает цепь питания «обмотки» К1 — нагреватель обесточивается (на самом деле в электронных реле это не обмотка, а управляющий светодиод, как вы знаете из *главы 3*). В тот же момент времени отключается реле К2 и резистор R4 включается в цепь делителя R2, R3, R4, R5, еще больше увеличивая разницу напряжений между выводами компаратора.

По мере остывания воды напряжение на датчике повышается и в какой-то момент компаратор снова срабатывает, подключая нагрузку через реле К1. Контакты К2 при этом опять шунтируют резистор R4 и это тоже увеличивает разницу напряжений, но в теперь в другую сторону.

Подробности

Это обеспечивает т. н. *гистерезис* — небольшую разницу между напряжениями срабатывания и отпускания, которая необходима для того, чтобы схема не «дребезжала» в состоянии, близком к заданному порогу температуры. Наличие всей этой системы несколько увеличивает нестабильность поддержания температуры: при приведенных на схеме номиналах разница между температурой включения и выключения составит от 1 до 1,5° (например, при установленной температуре в 35° нагреватель включится, когда температура упадет до 34, а выключится — когда она достигнет 35,5°), однако нам более высокая стабильность в данном случае совершенно не требуется. В ключевых (пороговых) регуляторах гистерезис есть практически всегда, если нужно более точное регулирование, то целесообразнее пропорциональные регуляторы.

Теперь разберемся с режимами. Сначала рассмотрим «режим электрочайника» (автоматический однократный), для обеспечения которого в схему введено еще одно маломощное реле К3, включенное, как видите, довольно хитрым образом. Если тумблер S2 находится в положении «Автомат» (т. е. контакты его замкнуты), то реле К3 никак не участвует в работе схемы. Если же S2 переключить в режим «Однократный» (разомкнуть его контакты), то в момент достижения нужной температуры, вместе с отключением реле К1 (и, соответственно, нагрузки), реле К3, ранее включенное через диод VD1 и резистор R7 в ту же коллекторную цепь выходного транзистора микросхемы, также отключается, контакты его размыкаются и вывод 4 компаратора оказывается подключенным через датчик температуры к потенциалу земли.

Такое состояние схемы устойчиво и для возобновления работы в режиме стабилизации температуры необходимо либо на некоторое время отключить напряжение питания, либо тумблером S2 переключить схему в режим «Термостат». А конденсатор С2 вместе с диодом VD1 служат для «правильного» запуска схемы при включении питания: если тумблер К4 разомкнут, то контакты реле К3 должны замкнуться сразу после подачи напряжения питания, иначе компаратор не сработает. При подаче напряжения питания, как мы знаем, конденсатор представляет собой короткозамкнутый участок цепи, поэтому реле К3 на небольшое время, пока конденсатор заряжается (примерно 100 мс), замкнет контакты. Диод VD1 на это время запирается и предохраняет от срабатывания реле К1 и К2. В случае, если температура воды в момент включения превышает установленную, такое срабатывание реле будет кратковременным (только на время зарядки конденсатора С2). Если же температура ниже требуемой, то компаратор успеет сработать, диод VD1 откроется,

и реле К3 останется в замкнутом состоянии до момента отключения нагрузки. Кстати, опыт эксплуатации подобного устройства показал, что наиболее популярен именно режим «электрочайника», т. к. он позволяет экономить электроэнергию и не беспокоиться о том, что вы оставили включенный электроприбор без присмотра.

Ручной режим (резервный, на случай выхода автоматики из строя, чтобы при этом не остаться вовсе без горячей воды) обеспечивается просто: тумблер S1 в положении «Постоянно» подает сетевое питание напрямую на нагреватель (контакты К1 при этом шунтируются, схема обесточивается, а вся система работает так, будто никакой автоматики и не существует). В положении «Автомат» сетевое напряжение переключается на блок питания автоматики, а нагреватель теперь может включаться только контактами реле. Тумблер S1, естественно, должен выдерживать рабочий ток ТЭНа. Здесь подойдет импортный переключатель B1011, рассчитанный на ток до 16 А при напряжении 250 В или другой аналогичный. В крайнем случае можно использовать автомобильные переключатели, но это не очень корректно, т. к. на напряжения до 300 В они не рассчитаны.

Когда сетевое напряжение поступает на нагрузку (неважно, через тумблер или контакты реле), горит включенная параллельно ей неоновая лампочка H1, по которой можно контролировать работу схемы. Лампочка может быть любого типа, только при этом резистор R8 должен иметь мощность не менее 0,5 Вт, т. к. он работает при сетевом напряжении (обычные резисторы 0,125—0,25 Вт имеют предельно допустимое напряжение порядка 200 В). Отметим, что ставить светодиод здесь неудобно: нужно либо выбирать двухцветный встречно-паралельный, либо ставить выпрямительный мост, и мощность резистора придется еще больше увеличить — потребуется как минимум 1 Вт при сопротивлении 68 кОм, и он будет заметно греться.

Симисторное реле PF240D25 (разводка его выводов на схеме не показана, все нарисовано прямо на корпусе) в принципе допускает ток до 25 А, однако достаточно сильно греется уже при 10 А. Поэтому допустимую мощность ТЭНа лучше ограничить величиной 2 кВт, а в корпусе устройства сверху и снизу обязательно нужно предусмотреть вентиляционные отверстия. При этом реле К1 в рабочем положении корпуса должно быть расположено выше остальных деталей.

Если вы хотите добиться большей мощности, то лучше выбрать аналогичное реле типа D2425, которое имеет отверстия для установки на дополнительный радиатор. Электромагнитное реле ставить здесь не рекомендуется: придется включать мощное реле-пускатель через промежуточное реле, и они совместно отнюдь не будут услаждать ваш слух своим грохотом и жужжанием. А вот

реле К2 и К3 вполне можно заменить на маломощные электромеханические — например типа РЭС-60 или РЭС-49. Естественно, резисторы R6 и R7 в этом случае не нужны, а вот у конденсатора С2, возможно, придется раза в два увеличить емкость для более надежного включения устройства.

В положении тумблера S1 «Автомат» сетевое напряжение поступает на простейший нестабилизированный блок питания (квадрат с надписью БП на схеме рис. 6.11), схема которого не расшифрована, потому что полностью соответствует показанной на рис. 4.2. Как обычно, такую конструкцию можно извлечь из покупного блока со встроенной вилкой, мощности от него никакой не требуется (вся схема потребляет ток порядка 30 мА), поэтому можно выбирать любой на напряжение (под номинальной нагрузкой) от 10 до 15 В. Напряжение с него поступает на стабилизатор типа LM78L09 (в корпусе TO-92, можно заменить на отечественный 142ЕН8Б или на аналогичный иного производителя), откуда стабилизированное напряжение +9 В подается на схему. Светодиод VD2 сигнализирует о включении схемы автоматики, его лучше выбрать зеленого свечения, чтобы обеспечить контраст с неоновой лампочкой.

Заметки на полях

Самое сложное в процессе изготовления устройства — обеспечить надежную и долговечную изоляцию термистора от воды, но с сохранением хорошего теплового контакта. Хороший вариант — залить термистор в металлической трубочке эпоксидной смолой, прямо вместе с пайками к удлинительным проводам (последние дополнительно изолируются термоусадочной трубкой). Только при этом не следует забывать, что сама по себе эпоксидная смола не водостойка, а металл может корродировать. Такую конструкцию необходимо дополнительно покрыть каким-нибудь надежным и не выделяющим вредных веществ водостойким составом, вроде полиуретановых лаков или автомобильных эмалей горячей сушки. Другой вариант — «запечатать» датчик в зубную пластмассу (для чего может понадобиться помощь знакомого дантиста).

При указанных на схеме номиналах термостат обеспечивает установку заданной температуры в диапазоне примерно 35—85°. При термисторе с другим сопротивлением придется только пропорционально изменить номинал R1, больше ничего менять в схеме не надо. Настройка и калибровка схемы ничем не отличается от таковых для регулятора оборотов, кроме выбора диапазона температур. При настройке основную нагрузку можно не подсоединять, т. к. момент срабатывания и отключения вполне можно контролировать по неоновой лампочке, следует только учесть, что вовсе без нагрузки «неонка» может гореть даже при выключенном реле — из-за токов утечки через «контакты» (на самом деле там стоит тиристор, у которого ток утечки может достигать 10 мА) и вам даже может показаться, что система не работает. Если так, то придется все же подключить какую-то нагрузку, например лампочку

накаливания. В процессе калибровки надо обязательно обеспечить хорошее перемешивание воды!

Заметки на полях

Я настоятельно рекомендую теплоизолировать бак для воды, даже в отсутствие регулятора: просто обернув его старым ватным одеялом, вы можете экономить до 70—90% электроэнергии. Это касается не только данной конструкции, но и вообще всех водонагревателей. Можно сделать и «фирменную» теплоизоляцию из упаковочного пенопласта.

В заключение отметим, что схемы для построения термостатов невысокого класса, подобных двум описанным, существуют, разумеется, и в интегральном исполнении, обычно они при этом совмещены с полупроводниковым датчиком температуры, который часто имеет и отдельный выход, что обеспечивает возможность измерения температуры.

На этом мы с рассмотрением ОУ закончим и займемся звуком — это еще одна область, где аналоговые микросхемы доминируют над цифровыми (хотя и не всегда, как вы увидите в дальнейшем).

Звуковые усилители

В основе большинства усилителей звукового диапазона, предназначенных для работы на динамические громкоговорители-колонки (такие усилители часто именуют УМЗЧ — «усилитель мощности звуковой частоты», а кроме них, есть еще микрофонные, предварительные и тому подобные усилители, которые мы не будем здесь рассматривать), независимо от того, выполнены ли они на дискретных элементах, или в виде интегрального модуля, всегда лежит одна и та же базовая схема. В одном из упрощенных вариантов ее можно представить так, как показано на рис. 6.12. Разбирать мы ее подробно не будем, остановимся лишь на ключевых моментах, которые имеют значение для понимания работы интегральных усилителей.

Вход почти любого УМЗЧ, как и вход ОУ, представляет собой дифференциальный каскад. Так как звуковой сигнал в идеале является симметричной синусоидой, с которой удобно работать при симметричном двуполярном питании, то входной сигнал должен находиться где-то посередине между напряжениями питания. Чтобы обеспечить развязку по постоянному току, сигнал на вход обычно подают через фильтр высокой частоты (C1 и R1, в некоторых случаях обходятся и одним конденсатором).

На второй вход дифференциального каскада при этом подают сигнал обратной связи, стабилизирующий характеристики усилителя (в данном случае — через сопротивление R5). Если усилитель интегральный, то обратную связь

большей частью выносят вовне микросхемы, т. к. она обычно требует регулируемой коррекции (на схеме конденсатор C2) — ограничения усиления на высоких частотах, иначе готовый усилитель может «загудеть». При указанных на схеме соотношениях R5/R4 коэффициент усиления по напряжению устанавливается примерно равным 30, что позволяет усилить обычный выходной сигнал линейного выхода магнитолы или тюнера (0,7 В) до амплитуды, необходимой для «раскачки» мощной нагрузки.

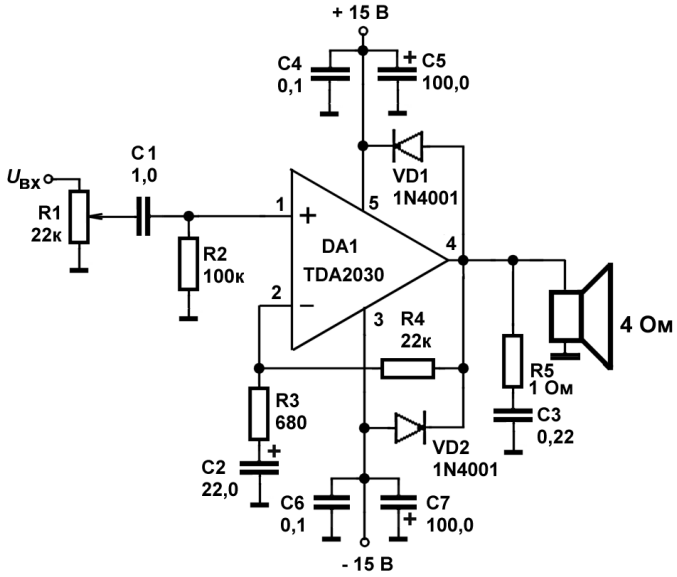


Рис. 6.12. Классическая базовая схема усилителя звуковой частоты

Оконечный каскад усиления мощности всех таких усилителей представляет собой т. н. «пушпульный» (от push-pull — «тяни-толкай», по-русски) каскад на паре *комплементарных* (т. е. «дополняющих друг друга») транзисторов, имеющих близкие характеристики, но разную полярность (*n-p-n* и *p-n-p*). На схеме вы видите довольно мощные приборы фирмы Motorola BDW93C/BDW94C (до 80 Вт), но существует много подобных пар отечественного производства: совсем «древних» КТ315/КТ361, маломощных КТ3102/КТ3107, средней мощности КТ815/КТ814, КТ817/КТ816, КТ972/КТ973 (с «супербетой»), наконец, более мощных КТ819/КТ818. Изредка используют и специальные пары мощных полевых транзисторов.

Чаще такой каскад встроен в микросхему, но иногда целесообразно «умощнить» выход интегрального усилителя дискретными транзисторами (или

в характеристиках микросхемы это прямо рекомендуется). По сути «пушпульный» каскад есть просто два эмиттерных повторителя разной полярности, работающих на одну нагрузку. При этом надо не забывать про падения напряжения «база-эмиттер», из-за чего каскад должен всегда иметь начальный сдвиг, приоткрывающий оба транзистора и обеспечивающий небольшой сквозной ток через них. На данной схеме для этого служит цепочка диодов.

> ***Замечание***
>
> Иногда встречаются и более сложные способы, причем для лучшей температурной стабильности каскада следует эти диоды располагать в контакте с радиатором мощных транзисторов. Если этого не делать, то возможен самопроизвольный выход транзисторов из строя — температурный коэффициент напряжения «база-эмиттер» отрицателен, и по мере нагревания транзисторы будут все больше «распахиваться», в свою очередь нагревая себя еще сильнее — вплоть до выгорания. В более сложных схемах такое удается предотвратить и иными способами.

Если смещения не делать, то выходное напряжение будет иметь искажения типа «ступенька» — из входной синусоиды за счет зоны нечувствительности в пределах ±0,6 В для обычных транзисторов (±1,2 В для транзисторов с «супербетой», т. н. «дарлингтоновских», состоящих из двух транзисторов, включенных последовательно) как бы вырезается «кусок» вблизи нулевого уровня.

За остальными подробностями я отправлю вас к классическим трудам [6] и [7], а мы займемся практическими конструкциями. Но сначала для общего образования рассмотрим одну единицу измерения, которая часто встречается при описании подобных схем.

О децибелах

В разговоре о таких вещах, как звуковые усилители, децибелы обойти нельзя. *Децибел* (одна десятая *белла*, названного так по имени изобретателя телефона А. Белла) есть единица измерения отношений величин. Перевести отношение в децибелы и обратно можно по формуле: K (дБ) = $20 \cdot \lg(A_1/A_0)$, где A_1/A_0 есть отношение значений некоторых величин (напряжений, токов, звукового давления и т. п.).

Децибелы удобны для характеристики изменения величин, меняющихся по степенному закону, их широко используют при расчетах фильтров, анализе частотных и амплитудных характеристик ОУ, или, скажем, в таких случаях, как измерение уровня звукового давления. График степенной функции, которая быстро возрастает или падает в обычных координатах, в широком диапазоне значений практически невозможно изобразить, а в логарифмическом масштабе (в децибелах) он будет выглядеть прямой линией (это часто встре-

чающиеся графики, где по осям отложены величины, возрастающие не линейно, а в геометрической прогрессии: 1, 10, 100, 1000…). Звуковое давление практически всегда измеряют в децибелах (относительно порога слышимости) — это связано с тем, что наше ухо реагирует именно на отношение громкостей, а не их абсолютный прирост.

Если отношение величин больше единицы, то величина в децибелах будет положительной, если меньше — отрицательной. Для перевода децибел в обычные относительные единицы и обратно необязательно выполнять расчет по указанной ранее формуле, достаточно запомнить несколько простых соотношений:

- 3дБ соответствует увеличению/уменьшению на треть;
- 6 дБ соответствует отношению в 2 раза;
- 10 дБ соответствует отношению в 3 раза;
- 20 дБ соответствует отношению в 10 раз.

Руководствуясь этими соотношениями, легко перевести любую величину: например, 73 дБ есть 20 + 20 + 20 + 10 + 3 дБ, т. е. 10·10·10·3·1,33 = 4000. Собственный коэффициент микросхемы звукового усилителя TDA2030 (см. далее) равен 30000, т. е. $3 \cdot 10^4$, или 10 + 4·20 = 90 дБ, а простейшей схемы по рис. 6.12 — около 66 дБ (2000). Коэффициент ослабления синфазного сигнала (КОСС), о котором шла речь ранее, также чаще всего измеряют в децибелах: так, его величина, равная –60 (3·20) дБ, означает, что синфазный сигнал ослабляется в 1000 раз. Крутизна характеристик простейших RC-фильтров низкой и высокой частоты из *главы 2* равна, соответственно, –6 и +6 дБ на октаву, что означает уменьшение/увеличение сигнала в 2 раза при двукратном изменении частоты.

Мощный УМЗЧ

Вооружившись такой терминологией, мы стали совсем умными, и можем приступить к делу. Первой разберем стандартную схему УМЗЧ на популярной микросхеме TDA2030 производства фирмы ST Microelectronics (рис. 6.13). В ней производитель гарантирует при выходной мощности 14 Вт на нагрузке 4 Ом искажения сигнала не более 0,5%. Если снизить требования к величине искажений, то при ±15 В питания из микросхемы можно «выжать» до 20 Вт. Предельно допустимое значение напряжения питания для TDA2030 достигает ±18 В (или 36 В однополярного), но, разумеется, при таком питании ее эксплуатировать не рекомендуется. Увеличение искажений при повышении выходной мощности, вероятно, связано с тем, что в чип встроена защита

от перегрева выходных транзисторов, которая ограничивает выходной ток, когда температура корпуса повышается.

Производитель гарантирует такие характеристики, как диапазон частот, которые передаются с заданным коэффициентом усиления и при заданных искажениях сигнала (40 Гц — 15 кГц), и коэффициент подавления влияния нестабильности источника питания на качество выходного сигнала (в 100—300 раз), что допускает питание от простейшего нестабилизированного источника (см. рис. 4.4). При указанных номиналах резисторов и конденсаторов устойчивость усилителя гарантируется и даже приводятся рекомендации по размерам охлаждающего радиатора.

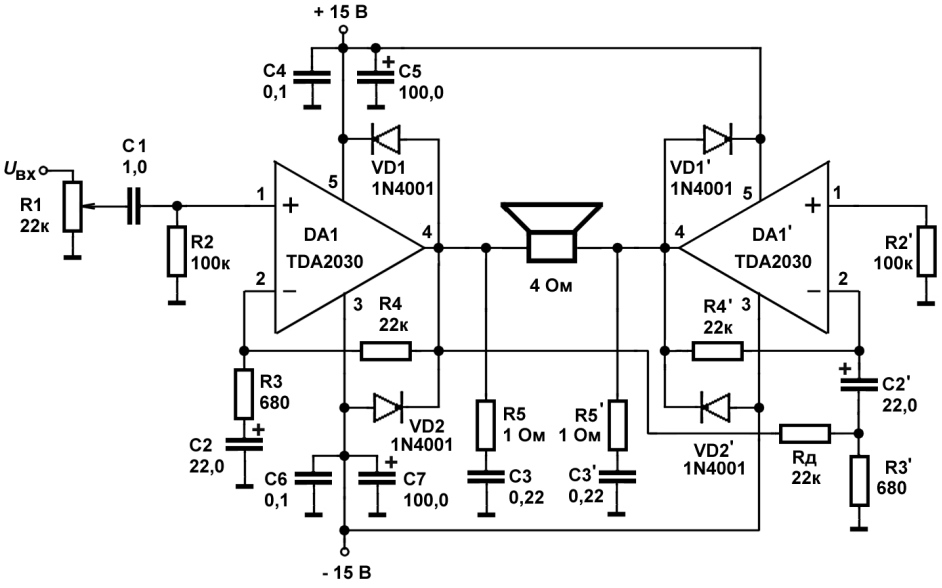


Рис. 6.13. Рекомендуемая схема усилителя звуковой частоты на микросхеме TDA2030

Собственно усилитель включает саму микросхему DA1, конденсаторы C1, C2 и резисторы R1—R4. Если внимательно присмотреться к этой схеме, то мы увидим, что структурно она ничем не отличается от нашей базовой схемы (см. рис. 6.12). Мало того, здесь даже установлен с помощью обратной связи тот же самый коэффициент усиления, примерно равный 30. Как будто взяли нашу схему и упаковали ее в отдельный корпус, обеспечив вывод наружу входов дифференциального усилителя, выхода двухтактного (push-pull) каскада усиления мощности и, естественно, выводов питания. На самом деле характеристики «фирменного» усилителя заметно выше: в микросхеме

TDA2030 коэффициент усиления по напряжению при разомкнутой цепи обратной связи, согласно документации производителя, равен примерно 30 000, а в предыдущей схеме он не более 2000—2500. Это, конечно, для «фирменной» схемы значительно увеличивает линейность усиления и уменьшает уровень искажений, аналогично работе обратной связи в ОУ.

Остальные элементы схемы — вспомогательные. Конденсаторы C4—C7 — развязывающие по питанию, их надо устанавливать прямо у выводов микросхемы. Причем разработчики учли, что емкость электролитических конденсаторов снижается с ростом частоты, поэтому в целях лучшей защиты от помех и повышения устойчивости схемы здесь рекомендуется устанавливать неполярные (например, керамические) конденсаторы (C4 и C6) параллельно с электролитическими (C5 и C7). Цепочка R5, C3 устанавливается для повышения линейности усилителя при работе на индуктивную нагрузку. Диоды VD1, VD2 служат для предотвращения возможного выхода из строя выходных каскадов микросхемы при индуктивных выбросах напряжения (например, при включении питания) — ох, до чего же нежные эти западные транзисторы!). Все электролитические конденсаторы — на напряжение не менее 16 В.

Если усилитель все же «загудит» (хотя и прямо об этом в тексте фирменной инструкции не сказано), здесь рекомендуется параллельно резистору обратной связи R4 установить цепочку из последовательно включенных резистора и конденсатора, которые ограничат полосу частот. При номиналах всех остальных компонентов, таких как указаны на схеме, резистор должен быть равен 2,2 кОм, а конденсатор — не менее 0,5 нФ. Увеличение емкости конденсатора сверх этой величины ведет к ограничению полосы частот, но и к повышению устойчивости схемы.

Сама микросхема TDA2030 выпускается в корпусе TO220, знакомом по мощным транзисторам, только имеет он не три вывода, а пять (см. *Приложение 3*). Разводка выводов приведена на схеме, а для того, чтобы определить их расположение, нужно положить микросхему маркировкой вверх, тогда вывод номер 1 будет находиться первым слева (в однорядных корпусах микросхем ключ для определения начала отсчета выводов часто отсутствует, но первый вывод всегда расположен именно так, как указано).

Заметки на полях

Рекомендованная в инструкции площадь охлаждающего радиатора для выходной мощности 14 Вт должна составлять 350—400 см^2, однако, на мой взгляд, эта величина завышена как минимум вдвое. Впрочем, подобное заключение я могу подтвердить, кроме весьма приблизительных расчетов из *главы 5*, только личным опытом и оно не должно быть воспринято, как руководство к действию — это совет из той самой серии «на ваш страх и риск». Скорее всего, разработчики из фирмы ST Microelectronics взяли запас специально, чтобы уменьшить уровень искажений при больших мощностях из-за встроенного механизма тепловой защиты.

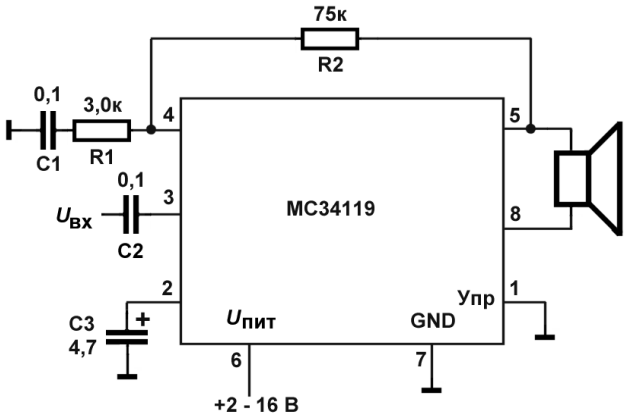


Рис. 6.14. Схема мостового усилителя звуковой частоты

На рис. 6.14 показано, как можно построить усилитель с удвоенной выходной мощностью при тех же напряжениях питания и используемых деталях. Это так называемая мостовая схема, которая представляет собой два идентичных усилителя, работающих на одну нагрузку в противофазе: когда на выходе одного усилителя положительный максимум напряжения, то на другом отрицательный. Таким образом, амплитуда и действующее значение напряжения на нагрузке возрастает ровно в два раза, соответственно растет и мощность, которая здесь составит при условии неискаженного сигнала почти 30 Вт. Для того чтобы усилители работали именно так, как указано, обычный (неинвертирующий) вход второго усилителя заземляется, а входной сигнал для него поступает на другой (инвертирующий) вход, туда же, куда и заведена его обратная связь. Сам этот входной сигнал берется с того места, куда поступает сигнал от первого усилителя (с левого по схеме вывода динамика) и ослабляется в той же степени, в которой оно было усилено первым усилителем, поскольку номиналы резисторов цепочки обратной связи R4, R3, задающей коэффициент усиления первого усилителя, и делителя Rд, R3' равны. Это означает, что на вход 2 второго усилителя поступает фактически то же самое входное напряжение, но, т. к. вход противоположной полярности, то на выходе второго усилителя повторится сигнал на выходе первого, только в противофазе, чего мы и добивались. Мощность источника питания, естественно, должна быть повышена.

Микроусилитель мощности

Не так уж редко возникает задача вывести звуковой сигнал на маломощный динамик или на головные наушники. Кроме очевидных применений вроде воспроизведения музыки, такой усилитель пригодился бы, скажем, в много-

численных конструкциях металлоискателей (их полно в Сети и в радиолюбительской литературе), в сигнальных устройствах. Одно из применений вы увидите в *главе 19*, когда мы заставим «разговаривать» микроконтроллер. Существует поистине необъятное множество типов микросхем от разных производителей, которые осуществляют усиление звукового сигнала с возможностью выхода на низкоомную нагрузку. Здесь мы остановимся на одной из самых популярных — MC34119 (выпускается не только фирмой Motorola, как можно было бы заключить из названия, но и другими производителями, возможно, с другими буквенными префиксами). Микросхема выпускается в обычном корпусе всего с восемью выводами (DIP-8) и никаких радиаторов не требует.

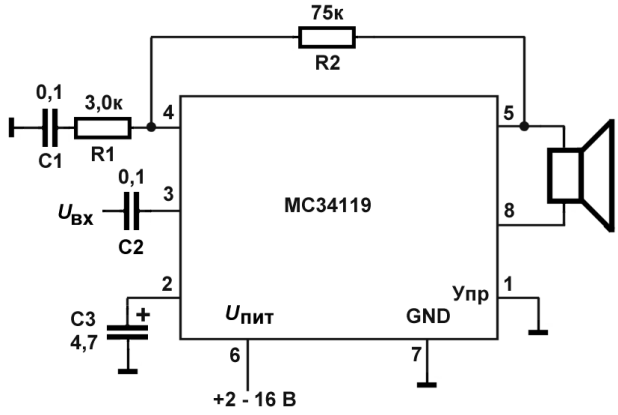


Рис. 6.15. Вариант типовой схемы включения микросхемы MC34119

Усилитель (рис. 6.15) обладает весьма неплохими характеристиками:

- напряжение питания 2—16 В (однополярное);
- сопротивление нагрузки 8 Ом (минимальное);
- частота единичного усиления: 1,5 МГц;
- выходная мощность 250 мВт (при напряжении питания 6 В и нагрузке 32 Ом);
- коэффициент гармоник 0,5—1%;
- время готовности после включения питания не более 0,36 с.

Самое главное — не надо думать, все уже придумано за вас. Коэффициент усиления задается двумя резисторами R1 и R2, и равен их отношению R2/R1 (в данном случае 25). Максимальная мощность в нагрузке 0,5 Вт обеспечива-

ется при нагрузке 32 Ом (головные наушники) при питании 12 В. В других сочетаниях нагрузки и питания такая мощность не достигается, в том числе потому, что недопустимо увеличиваются искажения. Обратите внимание, что динамик не имеет соединения с «землей» (что естественно для схемы с однополярным питанием). Имеется также интересная возможность выключения усилителя с помощью сигнала от логических микросхем (например, от микроконтроллера) — если подать на вывод 1 напряжение питания, микросхема выключится и будет потреблять ток не более нескольких десятков микроампер.

Отметьте, что по сути и микросхема TDA2030, и MC34119, и базовая схема по рис. 6.11, и даже разобранные нами в *главе 4* интегральные стабилизаторы, представляют собой не что иное, как узкоспециализированные ОУ — общие закономерности работы у них совершенно одинаковы. Что, если вдуматься, вполне логично, не так ли?

Глава 7

На пороге цифрового века

> Теперь в своей анкете в графе «Владение иностранными языками» вы можете гордо написать: «Бегло считаю по-японски до десяти…»
>
> *aikido-russia.ru*

Все началось, конечно, с Аристотеля, который жил в IV веке до нашей эры. Когда читаешь вступление к любой популярной книге, посвященной чему угодно — от изящных искусств до биологии, химии, физики и математики, — возникает впечатление, что Аристотель был каким-то сверхчеловеком. Тем не менее, авторы не врут, просто знаний было тогда накоплено еще не очень много, и обозреть их все — задача вполне посильная для человека острого ума и выдающихся способностей, каким Аристотель, несомненно, был.

Но главный урок Аристотеля, который заставляет даже пожалеть о том, что в современных колледжах и университетах прекратили преподавать латынь и греческий, в том, что древние рассматривали дисциплины во взаимосвязи. Науки, хоть и делились Аристотелем на практические (этику и политику) и теоретические (физику и логику) дисциплины, но они рассматривались, как составные части единой науки. И это, без сомнения, более верная позиция, которую даже несколько раз переоткрывали заново (синергетика, синтетическая теория эволюции), но тогда, когда уже было бесполезно: ученые, как строители Вавилонской башни, окончательно поделились на слабо понимающих друг друга *специалистов* по дисциплинам.

Заметки на полях

Аристотель, между прочим, четко разделял науку и ремесла («техно», по-гречески) — позиция, которая была странным образом утрачена уже почти на наших глазах, во второй половине XX века, когда в 1956 году Нобелевскую *(научную)* премию впервые дали за *технологическое* достижение — изобретение транзистора. И пошло-поехало — в некоторых источниках я встречал утверждения, что существование микропроцессоров есть выдающееся *научное* достижение.

И, раз уж мы заговорили на философские темы, уместно сделать еще одно замечание. Дело в том, что почти все «обычные» достижения технологического века (паровой двигатель, телеграф, телефон, самолет, телевизор, автомобиль и т. п.) в некотором смысле изобретать было не надо — идеи передачи речи или изображения на расстояние (волшебное зеркальце), или передвижения с большой скоростью по воздуху (ковер-самолет) были выдвинуты давно, вероятно, задолго даже до Аристотеля. Нужно было придумать только способы технического воплощения этих идей. Если завтра изобретут антигравитацию, вы, с детства читавшие фантастические романы, сильно ли удивитесь? А вот никаких таких «компьютеров» не существовало и в помине — в некотором смысле это единственное *настоящее изобретение*, основанное на чисто идеальных предпосылках, в материальном мире никаких аналогов не имевшее — кроме, конечно, самого человеческого разума. И люди — самые уважаемые, крупные математики — всерьез верили, что именно искусственный разум они и изобрели, не будучи в силах поверить, что это действительно абсолютно новая, ранее не существовавшая вещь, которой не надо искать аналогий в повседневности. Компьютеры к разуму имеют такое же отношение, как интернет-путешествие к реальной охотничьей экспедиции в Африку — это всего лишь *имитационное моделирование* отдельных немногочисленных сторон деятельности разума. Что совершенно не умаляет значимости самого изобретения, кстати, скорее наоборот.

Главной составной частью науки во времена Аристотеля считалась именно логика — искусство рассуждения. Она-то и послужила той основой, из которой выросла цифровая техника и все многообразие информационных технологий, которые окружают нас теперь на каждом шагу.

Булева алгебра

Законы аристотелевой логики, которые с его лихой подачи стали идентифицироваться с законами мышления вообще, неоднократно пытались привести в математическую форму. Некто Луллий в XШ веке попытался даже механизировать процесс логических рассуждений, построив «Всеобщий решатель задач» (несомненно, это была первая попытка построения «думающей машины»). Затем формализацией логики занимался Лейбниц и многие другие, пока, в конце концов, все не сошлось в двух работах английского математика Джорджа Буля, который жил и работал уже в середине XIX века.

Заметки на полях

Любопытно название второй из этих работ — «Исследование законов мышления», первая же работа называлась поскромнее, но без «мышления» и тут не обошлось — в названии фигурировало слово «рассуждения». Значит, и сам Буль, и все его предшественники в течение более чем двух тысяч лет, и еще сто с лишним лет после него — никто так и не усомнился, что в основе мышления лежит именно та логика, которая называется «аристотелевой». Это была такая, как сейчас модно говорить, *парадигма*. И лишь в XX веке, после работ Геделя и Тьюринга, и особенно в связи с благополучно провалившимися (как и у Луллия за 700 лет до того) попытками создания «искусственного интеллекта»,

до ученых наконец начало доходить, что мышление вовсе не имеет логической природы, а логика есть лишь удобный способ сделать свои рассуждения доступными окружающим, т. е. перевести их в вербальную форму.

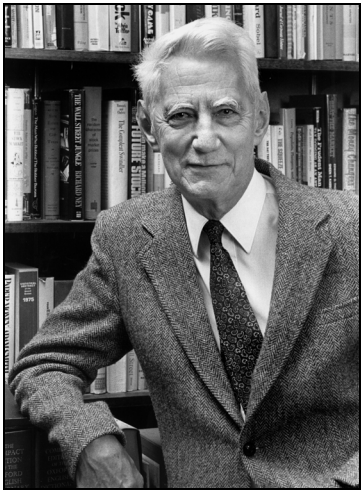


Рис. 7.1. Клод Элвуд Шеннон (Claude Elwood Shannon, 1916—2001). Фото Lucent Technologies Inc./Bell Labs

Главное для нашего повествования свойство логики обнаружил в своей магистерской диссертации от 1940 года великий Клод Шеннон (которому, как автору теории информации, мы вообще обязаны самим существованием цифрового века). Оказалось, что абстрактные булевы законы, не интересные, в общем, никому, кроме математиков (да и те сначала сомневались, стоит ли причислять логику к математическим дисциплинам), в точности совпадают с принципами функционирования реально существующих объектов — релейных электрических схем.

Что самое поразительное — все компоненты, необходимые для моделирования законов логики с помощью электрических устройств (реле, выключатели), были известны еще до опубликования Булем своих работ, но в течение еще почти ста лет никто не обращал на это внимание. Шеннон скромно утверждал, что случилось так, что до него просто никто не владел математикой и электротехникой одновременно. Не обратил на это внимание даже Чарльз Бэббидж, сконструировавший еще задолго до работ Буля механическую вычислительную («аналитическую») машину, а ведь был знаком и с самим Булем и с его работами!

Но довольно рассуждений, перейдем к практике.

Основные операции алгебры Буля

Булева алгебра имеет дело с абстрактными логическими переменными (операндами), для которых определены некоторые операции, подчиняющиеся определенным правилам:

- логическое сложение двух операндов (операция объединения, операция «ИЛИ» или «OR», обозначается обычным знаком сложения);
- логическое умножение двух операндов (операция пересечения, операция «И» или «AND», мы будем обозначать ее крестиком, чтобы отличить от обычного умножения)[1];
- отрицание для одного операнда (операция «НЕ» или «NOT», обозначается черточкой над символом операнда).

Остальные операции могут быть выведены из сочетания этих трех основных. Любая конкретная интерпретация булевых операндов — математическая или техническая — должна отвечать этим установленным правилам. Например, оказалось, что логическим операндам отвечают множества (отсюда названия операций «пересечение» и «объединение»). Но нас больше интересуют технические приложения, которые, однако, ничего не меняют в принципе: операция пересечения множеств совершенно адекватна операции «И» с логическими переменными, как и соответствующей манипуляции с выключателями в электрической сети, о чем далее.

В булевой алгебре многое совпадает с обычной (например, правила типа $A + B = B + A$; или $A + (B + C) = (A + B) + C$), но для нас важны как раз отличия. Вот они: $A + A = A$ (а не $2A$, как было бы в обычной алгебре), а также $A \times A = A$ (а не A^2). Последнее уравнение в обычной алгебре, впрочем, имело бы решение, причем сразу два: 0 и 1. Таким путем обычно и переходят к интерпретации булевых операндов, как логических переменных, которые могут иметь только два состояния: 1 и 0 или «правда» (True) и «ложь» (False). Тогда мы действительно можем с помощью определенных выше действий записывать некоторые словесные высказывания в виде уравнений и вычислять их значения, что дает иллюзию формального воспроизведения процесса мышления.

---

[1] В математике операция сложения (дизъюнкция) обозначается знаком $\vee$, а умножения (конъюнкция) — $\wedge$, но мы не будем их применять, т. к. запомнить, что есть что, тут непросто. Кроме того, операция умножения часто обозначается знаком «&», а сложения — « | », и эти обозначения нам встретятся далее.

Но сначала надо определить, как и в обычной алгебре, правила, которым подчиняются операции, т. е. таблицы логического сложения и умножения. Они таковы:

0 + 0 = 0	0 × 0 = 0
0 + 1 = 1	0 × 1 = 0
1 + 0 = 1	1 × 0 = 0
1 + 1 = 1	1 × 1 = 1

Операция отрицания «НЕ», понятно, меняет 1 на 0 и наоборот.

Примеры записи логических выражений обычно приводят для каких-нибудь бытовых высказываний, но мы не будем разбирать все эти любимые научно-популярными авторами схоластические доказательства утверждений вроде «все лебеди черные», а приведем более близкий к практике пример из области школьной математики.

Пусть высказывание состоит в следующем: «*x* меньше нуля *или x* больше единицы *и y* меньше двух». Как записать это высказывание? Введем следующие логические переменные: $A = (x < 0)$; $B = (x > 1)$; $C = (y < 2)$. Как мы видим, все они могут принимать только два значения — «правда» (если условие выполняется) и «ложь» (если не выполняется). Обозначим значение всего выражения через *D*. Тогда высказывание запишется в виде логического уравнения:

$$D = (A + B) \times C. \qquad (7.1)$$

Возможны другие варианты записи этого выражения:

- $D = (A$ ИЛИ $B)$ И C (по-русски);
- $D = (A$ OR $B)$ AND C (по-английски);
- $D = ((x < 0)$ **or** $(x > 1))$ **and** $(y < 2)$ (язык программирования Pascal);
- $D = ((x < 0) \mid (x > 1))$ & $(y < 2)$ (язык программирования C).

Рассмотрим подробнее возможные варианты решения уравнения 7.1. Пусть $x = 0{,}5$, $y = 1$. Чему будет равно *D* в этом случае? Очевидно, что выражение $(A + B)$ примет значение «ложь» (или 0), т. к. *x* не удовлетворяет ни одному из условий *A* и *B*. А переменная *C* примет значение «правда» (или 1), но результату это уже не поможет, т. к. произведение 0 на 1, согласно таблице логического умножения, равно 0. Таким образом, *D* в данном случае есть «ложь».

Если же принять значение $x = -0{,}5$, то *D* примет значение «правда». Интересный оборот примут события, если вместо «OR» между *A* и *B* подставить «AND» — легко догадаться, что выражение в скобках тогда не выполнится ни при каком значении *x*, т. к. условия «*x* меньше 0» и «*x* больше 1» взаимоисключающие. Потому результирующее условие *D* всегда будет принимать

значение 0, т. е. «ложь». Но вот если мы изменим выражение следующим образом:

$$D = \overline{(A \times B)} \times C, \tag{7.2}$$

т. е. инвертируем выражение в скобках с помощью операции «НЕ», то получим обратный результат: *D* всегда будет «правдой» (черточкой над символом или выражением, напомним, изображается инверсия). Интересно, что тот же самый результат мы получим, если запишем выражение следующим образом:

$$D = (\overline{A} + \overline{B}) \times C. \tag{7.3}$$

Это свойство выражается в т. н. правилах де Моргана (учителя Буля):

$\overline{A \times B} = \overline{A} + \overline{B}$;

$\overline{A + B} = \overline{A} \times \overline{B}$.

Отметим, что из таблиц логического умножения и сложения вытекает одно любопытное следствие. Дело в том, что ассоциация значения «ложь» с нулем, а «правды» — с единицей есть действие вполне произвольное, ничто не мешает нам поступить наоборот. В первом случае логика носит название «положительной», во втором — «отрицательной». Так вот, замена положительной логики на отрицательную приводит к тому, что все операции «ИЛИ» заменяются на «И» и наоборот (рассмотрите таблицы внимательно). А вот операция «НЕ» к такой замене индифферентна, т. к. 0 меняется на 1 в любой логике. В дальнейшем, если это специально не оговорено, мы всегда будем иметь в виду положительную логику.

Далее приведены несколько соотношений, которые вместе с правилами де Моргана помогают создавать и оптимизировать логические схемы. Некоторые из них очевидны, некоторые же — совсем нет.

$A \times B \times C = (A \times B) \times C = A \times (B \times C)$ (ассоциативный закон умножения);

$A + B + C = (A + B) + C = A + (B + C)$ (ассоциативный закон сложения);

$A \times A = A$; $A + A = A$;

$A + \overline{A} = 1$; $A \times \overline{A} = 0$;

$A \times 1 = A$; $A + 1 = 1$;

$A \times 0 = 0$; $A + 0 = A$;

$A + A \times B = A$;

$A \times (B + C) = A \times B + A \times C$;

$A + B \times C = (A + B) \times (A + B)$;

$\overline{1} = 0$; $0 = \overline{1}$.

Булева алгебра на выключателях и реле

Для того чтобы представить булевы переменные и операции над ними с помощью технических устройств (то, что сделал Клод Шеннон в своей диссертации), надо придумать схемы, которые воспроизводили бы эти операции согласно вышеизложенным правилам. Самый простые варианты таких схем показаны на рис. 7.2.

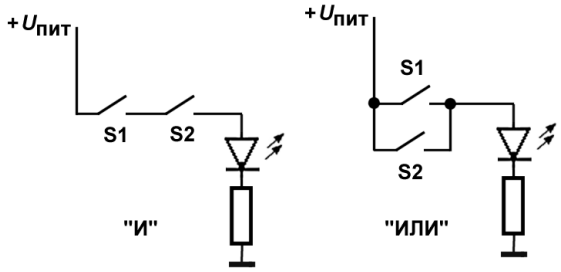


Рис. 7.2. Схемы реализации логических функций на кнопочных выключателях

Здесь операции «И» и «ИЛИ» выполняются обычными кнопками без фиксации. Каждая из них соответствует одной логической переменной, которая принимает значение «1», если контакты замкнуты, и «0» — если разомкнуты. На выходе значению «0» соответствует погасший светодиод, «1» — горящий. Легко понять, что работать эти схемы будут именно так, как указано в правилах для соответствующих логических операций. Для технических устройств, которые, как мы увидим далее, могут выполнять функции, отличающиеся от базового набора булевых операций, правила соответствия входов и выхода называются «таблицами истинности» (или «таблицами состояния») и оформляются в следующем виде:

«ИЛИ»		
Вх1	**Вх2**	**Вых**
0	0	0
0	1	1
1	0	1
1	1	1

«И»		
Вх1	**Вх2**	**Вых**
0	0	0
0	1	0
1	0	0
1	1	1

Если разобраться со схемами рис. 7.2 поглубже, то придется констатировать, что настоящими входными логическими переменными для них будут движения пальца, нажимающего на кнопку. В частности, операция «НЕ» здесь будет означать нажатие на кнопку с нормальнозамкнутыми контактами, а каскадное соединение таких схем для реализации сложных выражений предполагает наличие человека, транслирующего выходной сигнал одной схемы (состояние светодиода) во входной другой схемы (состояние контактов). Логично поставить вместо такого человека, «тупо» выполняющего предопределенные действия, техническое устройство. И здесь помогут уже хорошо нам известные электромагнитные реле.

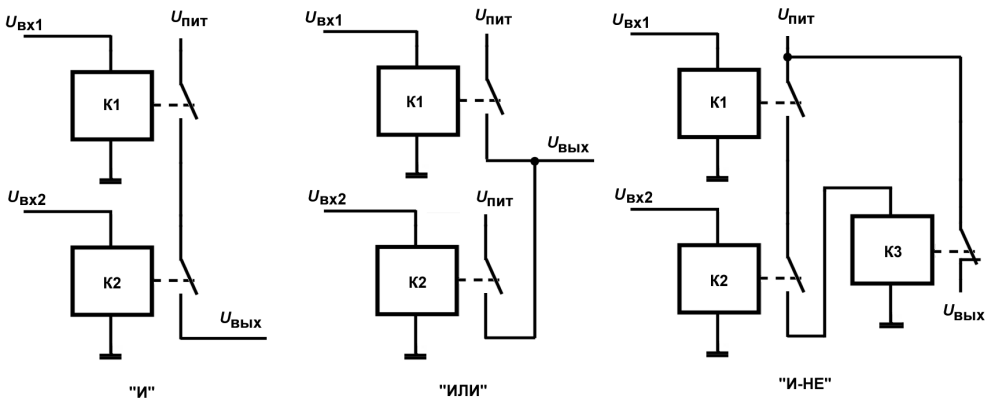


Рис. 7.3. Схемы реализации логических функций на реле

В схемах на рис. 7.3 как для входов, так и для выхода наличие напряжения соответствует логической единице, отсутствие его — логическому нулю. (Можно для наглядности подключить к выходу светодиод или лампочку, но суть дела от этого не изменится.) Способ подачи входного сигнала не указан, т. к. предполагается, что источник входного напряжения может быть самый разный (разумеется, его мощность должна быть достаточной, чтобы заставить реле сработать) — в том числе и такая же схема на реле.

Последний вариант представлен на рис. 7.3 справа, где изображена схема составного элемента «И-НЕ» на трех реле, в виде совокупности элемента «И» (такого же, как на рисунке слева) и элемента «НЕ» (инвертора), который есть не что иное, как одиночное реле с выходом через нормальнозамкнутые,

а не нормальноразомкнутые контакты. Таблица истинности для элемента «И-НЕ» будет выглядеть так:

«И-НЕ»		
Вх1	**Вх2**	**Вых**
0	0	1
0	1	1
1	0	1
1	1	0

Легко видеть, что она не адекватна таблице для «ИЛИ», как могло бы показаться на первый взгляд. Аналогично составляется элемент «ИЛИ-НЕ» — из схемы «ИЛИ», показанной на рис. 7.3 посередине, и инвертора. Таблица истинности для него будет такой:

«ИЛИ-НЕ»		
Вх1	**Вх2**	**Вых**
0	0	1
0	1	0
1	0	0
1	1	0

В большинстве современных серий логических микросхем используются именно элементы «И-НЕ» и «ИЛИ-НЕ», а не чистые «И» и «ИЛИ» (которые часто даже вовсе отсутствуют в составе некоторых серий, см. *главу 8*). Для того чтобы было проще разбираться в логических схемах, не заучивая таблицы истинности, работу элементов можно запомнить следующим образом: элемент «И» дает единицу на выходе только если на входах одновременно есть единица («оба одновременно»), элемент «ИЛИ» — если на любом из входов единица («хотя бы один»). Возможно, вам еще проще будет запомнить так: элемент «ИЛИ» дает единицу на выходе, если на входах «хотя бы одна единица», а элемент «И» дает ноль на выходе, если на входах «хотя бы один ноль». Рассмотренные же элементы с инверсией по выходу будут давать в тех же случаях обратные значения.

Заметки на полях

Интересно рассмотреть вопрос — а нельзя ли упростить схемы этих комбинированных элементов, исключив из них третье реле, выполняющее инверсию? В самом деле, большинство реле имеют перекидные контакты, так за чем же дело стало — меняем нормальноразомкнутые контакты на нормальнозамкнутые, и все! Легко заметить, что такая замена не будет адекватной, поскольку мы инвертируем здесь не общий выход элемента, а выходы каждого реле в отдельности, что равносильно инвертированию входов. Если обратиться к правилам де Моргана, то мы увидим, что такое изменение схемы приведет к тому, что элемент «И» превратится в «ИЛИ-НЕ», а «ИЛИ» — соответственно, в «И-НЕ». Иначе можно сказать так: мы получили желанный результат, но в отрицательной логике. Я советую читателю посидеть над этими соображениями и вывести таблицы истинности самостоятельно, чтобы убедиться, что все сказанное — правда. Второе полезное упражнение состоит в том, чтобы попытаться самому построить трехвходовые элементы, соответствующие уравнениям $A + B + C$ и $A \times B \times C$ (они будет состоять из трех реле).

Для тех, кто не разобрался как следует в этом по необходимости кратком изложении, среди прочих источников особенно порекомендую обратиться к [8] — книге, написанной очень простым и понятным языком, ориентированным на неподготовленного читателя, но вместе с тем излагающей предмет во всех подробностях.

Как мы считаем

О том, что мы считаем в десятичной системе потому, что у нас десять пальцев на двух руках, осведомлены, вероятно, все. Персонажи из мультфильмов студии «Пилот ТВ» — Хрюн Моржов и Степан Капуста — считают, наверное, в восьмеричной системе, так как у них пальцев по четыре. У древних ацтеков и майя в ходу была двадцатеричная система (вероятно потому, что закрытая обувь в их климате была не в моде). Вместе с тем, история показывает, что привязка к анатомическим особенностям строения человеческого тела совершенно необязательна. Со времен древних вавилонян у нас в быту сохранились остатки двенадцатеричной и шестидесятеричной систем, что выражается в количестве часов в сутках и минут в часах, или, скажем, в том, что столовые приборы традиционно считают дюжинами или полудюжинами (а не десятками и пятерками). Так что само по себе основание системы счисления не имеет значения, точнее, это дело привычки и удобства.

Число — одна из самых удивительных абстрактных сущностей. Нет никаких сомнений, что число, количество предметов — есть вполне объективно существующая характеристика, и в отличие от, к примеру, зрительных образов, она совершенно независима от самого факта наличия разума у считающего субъекта и даже от наличия самого субъекта — если бы (и когда) цивилизации

вообще не существовало, количество планет в Солнечной системе осталось бы тем же. И тем не менее материального воплощения числа не имеют — *количество*, представленное в виде комбинации пальцев рук и ног, зарубок на палочке (вспомните, как Робинзон Крузо вел свой календарь), разложенных на земле веточек, костяшек на счетах или — что для нас самое главное! — черточек или значков на бумаге, есть всего лишь физическая модель некоего идеального абстрактного понятия «числа». Умение считать в уме, которое отличает цивилизованного человека от дикаря, и состоит в том, что мы можем оторваться от такой материальной модели и оперировать непосредственно с абстракцией.

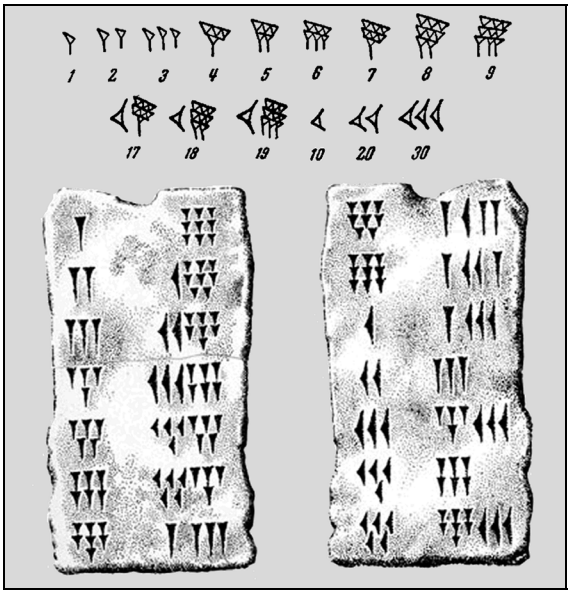


Рис. 7.4. Вавилонские глиняные таблички с записью чисел. Вверху перевод некоторых из них в десятичную систему

Заметки на полях

Раз уж мы опять ударились в философию, то не могу удержаться, чтобы не продолжить в том же духе: раз числа существуют объективно, то *где* они существуют? Этот вопрос совсем не так прост, потому что число есть лишь один из подобных объектов, несомненно присутствующих в природе, и тем не менее не имеющих материального воплощения — это и геометрические фигуры, и другие математические объекты, в том числе и булева алгебра вместе с ее операндами. Причем, если физические идеализации («абсолютно твердое» или «абсолютно упругое» тело) есть сущности, действительно выдуманные человеком с целью упрощения изучения свойств реальных тел, и вне человеческого знания не суще-

ствуют, то с математическими абстракциями вовсе не так: естественный спутник Земли всегда был *один*, даже когда самого человечества еще не существовало. Это послужило основанием для того, чтобы великий греческий философ Платон, из учений которого в той или иной степени проистекает вся современная западная философия, предположил существование некоего идеального мира («платоновского мира идей»), где все эти абстракции и «живут». Любопытно, что на этом основании Платона справедливо зачисляют в идеалисты, однако вышесказанное — хороший пример тому, что часто отождествляемые понятия «идеалистического» и «божественного» вовсе не одно и то же.

Позиционные и непозиционные системы счисления

Из понятия числа, как объективно существующей абстракции, вытекает, что его материальное представление может быть произвольным, лишь бы оно подчинялось тем же правилам, что и сами числа. Проще всего считать палочками (и в детском саду нас учат именно такому счету), в качестве которых могут выступать и пластмассовые стерженьки, и пальцы, и черточки на бумаге. Один — одна палочка, два — две палочки, десять — десять палочек. А сто палочек? Уже посчитать затруднительно, поэтому придумали сокращение записи: доходим до пяти палочек, ставим галочку, доходим до десяти — ставим крестик:

1	**2**	**5**	**7**	**10**	**11**
I	**II**	**V**	**VII**	**X**	**XI**

Узнаете? Конечно, это всем знакомая римская система, сохранившаяся до настоящих времен на циферблатах часов или в нумерации столетий. Она представляет собой пример *непозиционной* системы счисления, потому что значение определенного символа, *обозначающего* то или иное число, в ней не зависит от позиции относительно других символов — все значения в записи просто *суммируются*. Следовательно, записи «XVIII» и «IIIXV» в принципе должны означать одно и то же. На самом деле это не совсем так: в современной традиции принято в целях сокращения записи учитывать и позицию символа: скажем, в записи «IV» факт, что палочка стоит перед галочкой, а не после нее, означает придание ей отрицательного значения, т. е. в данном случае единица не прибавляется, а вычитается из пяти (то же самое относится и к записи девятки «IX»). Если вы человек наблюдательный, то могли заметить, что на часах четверку пишут почти всегда, как «IIII», а не как «IV», что, несомненно, более отвечает духу непозиционной системы. Однако при всех возможных отклонениях главным здесь остается факт, что в основе системы лежит операция *суммирования*.

Большие числа в римской системе записывать трудно. Поэтому были придуманы позиционные системы, к которым, в частности, принадлежала и упомянутая вавилонская шестидесятеричная (см. рис. 7.4).

Заметки на полях

В Европе позиционную систему переоткрыл (видимо) Архимед, затем от греков она была воспринята индусами и арабами, и на рубеже I и II тысячелетий н. э. опять попала в Европу[2] — с тех пор мы называем цифры арабскими, хотя по справедливости их следовало бы назвать индийскими. Это была уже современная десятичная система в том виде, в котором мы ее используем по сей день, у арабов отличается только написание цифр. С тем фактом, что заимствована она именно у арабов, связано не всеми осознаваемое несоответствие порядка записи цифр в числе и привычным нам порядком следования текста: арабы, как известно, пишут справа налево. Поэтому значение цифры в зависимости от позиции ее в записи числа возрастает именно справа налево, что в европейском языке нелогично — приходится заранее обозревать число целиком и готовить ему место в тексте.

Позиционные системы основаны не на простом суммировании входящих в них цифр, а на сложении их с весами, которые присваиваются автоматически в зависимости от положения цифры в записи. Так, запись «3» и в римской системе, и в арабской означает одно и то же, а вот запись «33» в римской системе означала бы шесть, а в арабской — совсем другое число, тридцать три.

Строгое определение позиционной системы является следующим: сначала выбирается некоторое число p, которое носит название *основания системы счисления*. Тогда любое число в такой системе может быть представлено следующим образом:

$$a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \ldots + a_1 p^1 + a_0 \cdot p^0 . \qquad (7.4)$$

В самой записи числа степени основания подразумеваются, а не пишутся (и для записи основания даже нет специального значка), т. е. запись будет представлять собой просто последовательность $a_n \ldots a_0$ (обратим внимание на то, что запись производится справа налево по старшинству — обычная математическая запись выглядела бы наоборот). Отдельные позиции в записи числа называются *разрядами*.

Заметки на полях

Еще один нюанс, дошедший до нас из древнегреческих времен, связан с тем, что греки и римляне не знали нуля. Именно поэтому первым годом нового века и тысячелетия считается 2001, а не 2000 год — год с двумя нулями относится к предыдущему столетию или тысячелетию — после последнего года до нашей

[2] Перевод соответствующего трактата арабского ученого аль Хорезми на латынь относится к 1120 году (на самом деле его звали Мухаммед аль Хорезми, т. е. «Муххамед из Хорезма»; между прочим, от его прозвища произошло слово *алгоритм*).

эры (минус первого) идет сразу первый год нашей эры, а не нулевой. Однако именно нулевой логично считать первым, вдумайтесь: ведь когда мы говорим «первые годы XX века», мы имеем в виду именно 1903 или 1905, а не 1913 или 1915. Но древние греки были совсем не такие дураки и ноль игнорировали не по скудоумию. Дело в том, что в последовательности объектов, нумерованных от нуля до, например, девяти, содержится не девять предметов, а десять! Чтобы избежать этой путаницы, в быту обычно нумеруют, начиная с единицы, тогда последний *номер* будет одновременно означать и *количество*. В электронике же и в программировании обычно принято нумеровать объекты, начиная с нуля, и всегда следует помнить, что номер и количество различаются на единицу (так, в байте 256 возможных символов, но номер последнего равен 255). На всякий случай всегда следует уточнять, откуда ведется нумерация, иначе можно попасть в неприятную ситуацию (скажем, элементы строки в языке Pascal нумеруются с единицы, а в языке C — с нуля).

Десятичная и другие системы счисления

В десятичной системе (т. е. в системе с основанием $p = 10$) полное представление четырехразрядного числа, например, 1024 таково: $1 \cdot 10^3 + 0 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$.

Так как любое число в нулевой степени равно единице, то степень в младшем разряде можно и не писать, но ради строгости мы ее будем воспроизводить, так как это позволяет нам лучше вникнуть в одно обстоятельство: степень старшего разряда всегда на единицу меньше, чем количество разрядов (нумерация степеней ведется с нуля).

Ну, а как можно представить число в системе счисления с другим основанием? Для любой системы с основанием p нужно не меньше (и не больше) чем p различных *цифр* — то есть *значков* для изображения чисел. Для десятичной системы их десять — это и есть известные всем символы от 0 до 9. Выбор начертания этих значков совершенно произволен — так, у арабов и по сей день 1 обозначается, как и у нас, палочкой. А вот цифра 2 обозначается знаком, похожим на латинскую строчную «r», причем тройка тоже имеет похожее начертание, и я плохо себе представляю, как Усама бен Ладен их там отличает. Впрочем, это дело привычки, у нас тоже значки «5» и «6» в некоторых случаях различить непросто, не говоря уж о сходстве между нулем «0» и буквой «O». В ручном написании текстов программ, а также в матричных компьютерных шрифтах, которые были в ходу до появления графического интерфейса, для этого ноль даже изображали перечеркнутым, наподобие знака диаметра: «∅». Попробуйте различить записи «15Ом» и «150м», если пробел забыли поставить в нужном месте — в случае матричных шрифтов или ручной записи, да и в любом случае, если символов «0» и «O» рядом не стоит,

это неразрешимая задача, если только из контекста не ясно, когда идет речь об омах, а когда — о метрах.

Чтобы древним вавилонянам, несчастным, не приходилось выучивать аж 60 разных начертаний знаков, они придумали логичную систему наподобие римской (еще раз обратите внимание на рис. 7.4) — действующую, впрочем, только в пределах первых шестидесяти чисел, а далее у них система становилась аналогичной современным.

Самые употребительные системы счисления в настоящее время, кроме десятичной, связаны с электроникой и потому имеют непосредственное значение для нашего повествования. Это знаменитая двоичная система и менее известная широкой публике, но также очень распространенная шестнадцатеричная.

Двоичная система

В двоичной системе необходимо всего два различных знака для цифр: 0 и 1. Это и вызвало столь большое ее распространение в электронике: смоделировать два состояния электронной схемы и затем их безошибочно различить неизмеримо проще, чем три, четыре и более, не говоря уж о десяти.

Что очень важно на практике, двоичная система прекрасно стыкуется как с представленными в предыдущем разделе логическими переменными «правда» и «ложь», так и с тем фактом, что величина, могущая принимать два и только два состояния, и получившая названия *бит*, есть естественная единица количества информации — меньше, чем один бит, информации не бывает. Это было установлено в 1948 году одновременно упоминавшимся Клодом Шенноном и Нобертом Винером, «отцом» кибернетики. Разряды двоичных чисел (то есть чисел, представленных в двоичной системе) также стали называть битами. (Bit, bite — по-английски «кусочек, частица чего либо». На самом деле это случайное совпадение: слово «бит» возникло от сокращения BInary digiT — «двоичная цифра».)

Заметки на полях

С троичным компьютером, который был на практике построен Н. Брусенцовым в МГУ на рубеже 60-х годов прошлого века (под названием «Сетунь»), связана отдельная история. При разработке первых компьютеров перед конструкторами встал вопрос об экономичности систем счисления с различными основаниями. Под экономичностью системы понимается тот запас чисел, который можно записать с помощью данного количества знаков. Чтобы записать 1000 чисел (от 0 до 999) в десятичной системе, нужно 30 знаков (по десять в каждом разряде), а в двоичной системе с помощью 30 знаков можно записать 2^{15}=32768 чисел, что гораздо больше 1000. Поэтому двоичная система явно экономичнее десятичной. В общем случае, если взять n знаков в системе с основанием p, то ко-

личество чисел, которые при этом можно записать, будет равно $p^{n/p}$. Легко найти максимум такой функции, который будет равен иррациональному числу *e*=2,718282…. Но поскольку система с основанием *e* может существовать только в воображении математиков, то самой экономичной считается система счисления с основанием 3, ближайшим к числу *e*. В компьютере, работающем по такой системе, число элементов, необходимых для представления числа определенной разрядности, минимально. Реализацию троичной системы в электронике можно представить себе, как схему с такими, например, состояниями: напряжение отсутствует (0), напряжение положительно (1), напряжение отрицательно (–1). Д. Кнут в своем труде [10] показывает, что троичная арифметика проще двоичной.

И все же брусенцовская «Сетунь» осталась историческим курьезом — слишком велики оказались сложности схемной реализации. Точнее, сам Николай Петрович Брусенцов как раз сложностей не испытывал, т. к. использовал для представления троичных цифр — тритов — трансформаторы, в которых наличие тока в обмотке в одном направлении принималось за 1, в другом — за –1, а отсутствие тока обозначало 0. Но реализовать на транзисторах такое представление значительно сложнее, чем двоичное. В наше время многоуровневые логические ячейки (правда, не троичные, а совместимые с двоичной логикой четвертичные) все же получили развитие — они служат для увеличения плотности упаковки информации в элементах флэш-памяти.

Стоит также отметить, что представление двоичных цифр с помощью уровней напряжения, как это делается в электронных устройствах, есть точно такая же модель числа, как раскладывание на земле палочек и проведение черточек на бумаге. В последних случаях мы оперируем с числами вручную, по правилам арифметики, а в электронных схемах это происходит в автоматическом режиме, без участия человека — вот и вся разница! Это очень важный момент, который следует хорошо осмыслить, если вы действительно хотите вникнуть в суть работы цифровых электронных схем.

Итак, запись числа в двоичной системе требует всего две цифры, начертание которых заимствовано из десятичной системы, и выглядит, как 0 и 1. Число, например, 1101 тогда будет выглядеть так:

$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13.$

Чтобы отличить запись числа в различных системах, часто внизу пишут основание системы:

$1101_2 = 13_{10}.$

Если система не указана, то имеется в виду обычно десятичная, но не всегда. Часто, когда из контекста понятно, что идет речь об электронных устройствах, не указывают не только основание «два», но и под словом «разрядность» имеют в виду количество именно двоичных, а не десятичных разрядов (таков, скажем, смысл термина «24-разрядный цвет»).

Шестнадцатеричная система

Шестнадцатеричная система имеет, как ясно из ее названия, основание шестнадцать. Для того чтобы получить шестнадцать различных значков, изобретать ничего нового не стали, а просто использовали те же цифры от 0 до 9 для первых десяти знаков, и заглавные латинские буквы от A до F — для знаков с одиннадцатого по шестнадцатый. Таким образом, известное нам число 13_{10} выразится в шестнадцатеричной системе, как просто D_{16}. Соответствие шестнадцатеричных знаков десятичным числам следует выучить наизусть: A — 10, B — 11, C — 12, D — 13, E — 14, F — 15. Значения больших чисел вычисляются по обычной формуле, например:

$A2FC_{16} = 10 \cdot 16^3 + 2 \cdot 16^2 + 15 \cdot 16^1 + 12 \cdot 16^0 = 40960 + 512 + 240 + 12 = 41724_{10}$.

Перевод из одной системы счисления в другую

Как следует из ранее изложенного, перевод в десятичную систему любых форматов не представляет сложности и при надлежащей тренировке может осуществляться даже в уме. Для того чтобы быстро переводить в десятичную систему двоичные числа (и, как мы увидим, и шестнадцатеричные тоже), рекомендую выучить наизусть таблицу степеней двойки до 16:

2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
2	4	8	16	32	64	128	256
2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
512	1024	2048	4096	8192	16384	32768	65536

На первое время достаточно запомнить верхний ряд, остальное выучится позже само.

Сложнее переводить из десятичной системы в двоичную, в учебниках описывается устрашающая процедура, основанная на делении столбиком. Я сейчас попробую вам показать способ, который позволяет переводить числа в двоичную систему несколько более простым методом, причем небольшие числа можно преобразовать даже в уме. Это, в сущности, то же самое деление, но без излишних сложностей и формальностей. Запомните сначала следующее правило: число, равное какой-либо степени двойки, имеет 1 в разряде с номером, на единицу большим степени, остальные все нули:

$2^1 = 2_{10} = 10_2$;

$2^2 = 4_{10} = 100_2$;

$2^3 = 8_{10} = 1000_2$ и т. д.

Способ состоит в следующем: пусть мы имеем, например, десятичное число 59. Подбираем наибольшую степень двойки из таблицы ранее, не превышающую этого числа: 32, что есть 5-я степень. Ставим 1 в шестом разряде: 100000. Вычитаем подобранную степень из исходного числа (59 – 32 = 27) и подбираем для остатка также степень, его не превышающую: 16 (2^4). Ставим единицу в 5-м разряде: 110000. Повторяем процедуру вычитания-подбора: 27 – 16 = 11, степень равна 8 (2^3), ставим единицу в 4-м разряде: 111000. Еще раз: 11 – 8 = 3, степень равна 2 (2^1), ставим единицу во 2-м разряде: 111010. Последнее вычитание дает 1, которую и ставим в младший разряд, окончательно получив $59_{10} = 111011_2$. Если бы исходное число было четным, к примеру 58, то в последнем вычитании мы бы получили 0, и число в двоичной системе также оканчивалось бы на ноль: $58_{10} = 111010_2$.

Кстати, полезно также обратить внимание, что числа, на единицу меньшие степени двойки, имеют количество разрядов, равное степени, и все эти разряды содержат единицы:

$2^1 – 1 = 1_{10} = 1_2$;

$2^2 – 1 = 3_{10} = 11_2$;

$2^3 – 1 = 7_{10} = 111_2$ и т. д.

Подобно тому, как наибольшее трехразрядное число в десятичной системе равно 999, и чисел таких всего $10^3 = 1000$ (от 000 до 999), в двоичной системе тех же трехразрядных чисел будет $2^3 = 8$ штук, в диапазоне от 000 до 111, т. е. от 0 до 7. Таким образом, наибольшее двоичное число с данным количеством разрядов будет всегда содержать все единицы во всех разрядах.

А вот из двоичной системы в шестнадцатеричную и обратно перевод очень прост: 16 есть 2^4 и без всяких вычислений можно утверждать, что одноразрядное шестнадцатеричное число будет иметь ровно 4 двоичных разряда. Поэтому перевод из двоичной системы в шестнадцатеричную осуществляется так: двоичное число разбивается на т. н. тетрады, т. е. группы по четыре разряда, а затем каждая тетрада переводится отдельно и результаты выписываются в том же порядке. Так как в тетраде всего 16 вариантов, то их опять же легко выучить наизусть:

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111
8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
1000	1001	1010	1011	1100	1101	1110	1111

Например, число 59_{10}, т. е. $0011\ 1011_2$, будет равно 3Bh.

Точно так же осуществляется и обратный перевод — каждое шестнадцатеричное число просто записывается в виде совокупности тетрад. Так, число $A2FC_{16}$ выразится, как $1010\ 0010\ 1111\ 1100_2$. Заметьте, что пробелы между тетрадами введены просто для удобства восприятия, подобно пробелам между тройками разрядов (классами) в записи больших десятичных чисел, и никакой иной нагрузки не несут. При записи двоичных чисел в тексте программ, как ассемблерных, так и программ на языках высокого уровня, естественно, эти пробелы ставить запрещается.

В качестве упражнения можете поразмыслить над вопросом, как переводить числа из двоичной в восьмеричную и четверичную системы, и наоборот: если вы справитесь с этим самостоятельно, считайте, что вы все поняли насчет систем счисления, применяемых в электронике.

Байты

Слово байт (byte) создано искусственно и представляет собой сокращение от BinarY digiT Eight, что буквально переводится, как «двоичная цифра восемь». На самом деле байт — это просто восьмиразрядное двоичное число. Соответственно, он имеет ровно два шестнадцатеричных разряда, или две двоичных тетрады. Такой байт был введен фирмой IBM в конце 50-х годов прошлого века, до этого (а в СССР — вплоть до 1969 года) применялись байты с другим количеством разрядов (5, 6 и 7).

Почему именно 8 разрядов? Да просто потому, что так удобно: число кратно степени двойки, т. е. легко масштабируется, скажем, шестнадцатиразрядное число — просто два байта, записанные подряд, подобно тетрадам в самом байте. В то же время оно относительно невелико и одновременно достаточно емко: имеет 256 значений, которых с лихвой хватает, к примеру, для представления всех печатных знаков европейских алфавитов (во всяком случае, до нашествия фирмы Microsoft с ее системой Windows хватало — что, конечно же, следует рассматривать, как шутку).

Поэтому в настоящее время байт — общепринятая единица измерения информации, т. к. основную единицу — бит — на практике применять неудобно из-за ее «мелковатости», числа получаются слишком большими. Применяют и меньшие единицы (кроме бита, это полубайт, или просто одно шестнадцатеричное число), и бóльшие — двухбайтовые (65 536 значений), четырехбайтовые (32 двоичных разряда или 4 294 967 296 значений) и даже восьмибайтовые числа. Все они часто называются *словами* — «word».

Однако на практике распространение получила запись больших количеств информации по системе, «слизанной» с десятичной: это килобайт (1024 или 2^{10} байта), мегабайт (1024 или 2^{10} килобайта) и гигабайт (1024 или 2^{10} мегабайта). Близостью значения 1024 к десятичной тысяче широко пользуются производители жестких дисков — так, емкость диска в 10 Гбайт может быть вовсе не 1024×1024 килобайта, как положено, а 1000×1000, что почти на пять процентов меньше. Пустячок — а ведь приятно обмануть покупателя, который потом будет с недоумением раздумывать, почему ему в настройках компьютера показывается вовсе не 10, а 9,54 Гбайта, и куда исчезли «родные» 460 Мбайт? Однако имейте в виду, что подобные фокусы все же выкидываются только в отношении наивных покупателей, а в серьезной технической документации кило- и мегабайты обычно означают то, что положено.

Подробности

В однобуквенных сокращениях принято обозначать байт большой буквой (Б), чтобы отличить его от бита (б), но в критичных случаях во избежание разночтений следует писать полностью: «кбайт», «кбит». В 1999 году Международная электротехническая комиссия (МЭК) с большим опозданием попыталась устранить неоднозначность в обозначениях кратности, введя специальные двоичные приставки *киби* (вместо «кило»), *меби* (вместо «мега») и *гиби* (вместо «гига»), означающие умножение на 1024 вместо 1000. Однако «килобайты» и «мегабайты» к тому времени настолько прижились, что эти не очень удачно звучащие обозначения так и не стали общепринятыми. Приставку «кило» в единицах информации иногда предлагают писать с большой буквы (Кбайт), чтобы подчеркнуть, что речь идет об умножении на 1024, а не на 1000, однако для приставок «мега» и «гига» (а также всех остальных) такого удобного приема уже нет. В большинстве случаев эти неоднозначности проблем не вызывают.

На практике можно встретить обозначение единиц информации из одной буквы (напр. «256 К памяти»), но злоупотреблять таким способом не следует, т. к. часто приходится гадать, идет ли тут речь о битах, байтах, словах или вообще *бодах* (см. далее). Мы вслед за фирменным описанием Atmel будем пользоваться такой нотацией (К или М без указания единиц), но исключительно для обозначения абсолютных чисел (например, «диапазон 4 М», что значит $4 \times 1024 \times 1024 = 4194304$), обозначающих обычно адреса в памяти программ, разбитые по двухбайтовым словам.

Впрочем, есть одна область, где традиционно употребляются именно биты (а также мегабиты и гигабиты), а не байты — это характеристики последовательных цифровых линий передач, к примеру, всем знакомая характеристика модемов в 56 К означает именно 56 кбит в секунду. Биты в секунду иногда называют *бодами* (по имени изобретателя телетайпного аппарата Эмиля Бодэ), но это не совсем точно. Употребление именно битов в сетях передачи данных связано с тем, что передача восьмиразрядными пакетами там применяется редко — скажем, стандарт RS-232, который мы будем еще разбирать, содержит восемь значащих разрядов (т. е. один байт), но помимо этого передается

как минимум еще два бита (стоповый и стартовый) — итого 10. В Интернете пакеты и вовсе могут иметь переменную длину, а модемы за одну посылку (как говорят связисты, за одну модуляцию) могут посылать от одного до 16 битов (вот число таких посылок в секунду и измеряется в бодах). Поэтому байтами в сетях информацию считают только, когда речь идет об отправленной или принятой информации, но не о той, которая реально передается.

Запись чисел в различных форматах

Шестнадцатеричный формат записи часто еще обозначают как HEX (hexadecimal), двоичный — как BIN (binary), а десятичный — как DEC (decimal). Кроме этого, в ходу еще т. н. двоично-десятичный формат BCD (binary-coded decimal), о котором далее. Так как с помощью матричных шрифтов на компьютерах с текстовыми дисплеями воспроизводить индексы было невозможно, то вместо того, чтобы обозначать основания системы цифрой справа внизу, их стали обозначать буквами: «B» (или «b») означает двоичную систему, «H» (или «h») — шестнадцатеричную, «D» (или «d») — десятичную. Отсутствие буквы также означает десятичную систему:

13 = 13d = 00001101b = 0Dh.

Такая запись принята, например, в языке ассемблера для процессоров Intel. Популярность языка C внесла в это дело некоторый разнобой: там десятичная система обозначается буквой «d» (или никак), двоичная буквой «b», а вот шестнадцатеричная буквой «x», причем запись во всех случаях предваряется нулем (чтобы не путать запись числа с идентификаторами переменных, которые всегда начинаются с буквы):

13 = 0d13 = 0b00001101 = 0x0D.

Такая же запись также принята в ассемблере для микроконтроллеров AVR, которыми мы будем пользоваться. Запись типа 0Dh ассемблер AVR не поддерживает, зато «понимает» представление HEX-формата, принятое в языке Pascal: $0D. В фирменном описании системы команд AVR можно даже встретить запись сразу с двумя обозначениями, по типу $0Dh (очевидно, специально для особо бестолковых).

Обратите внимание, что запись в HEX-формате обычно ведется в двухразрядном виде, даже если число имеет всего один значащий разряд, как в нашем случае, то в старшем пишется 0, то же самое относится и к двоичной записи, которая дополняется нулями до восьми разрядов. А вот для десятичного представления такой записи следует остерегаться.

Замечание

Чтобы еще больше «запутать» пользователя, разработчики AVR-ассемблера приняли для представления редко употребляемых восьмеричных чисел запись просто с ведущим нулем, без букв: например, 77 означает просто десятичное 77, а вот 077 будет означать $7 \cdot 8 + 7 = 63_{10}$. Нет, чтобы уж сделать и здесь, как в языке C, где восьмеричные числа записываются по аналогии со всеми остальными, как 0o*XX*.

Добавление незначащих нулей связано с тем обстоятельством, что запись и чтение чисел обычно ведется побайтно, а в байте именно два шестнадцатеричных разряда или восемь двоичных. Если число имеет разрядность больше байта, т. е. представляет собой *слово*, то для обозначения этого факта слева дописываются еще нули, так, шестнадцатиразрядное двоичное число 13 правильно записать, как 0000 0000 0000 1101b, а в HEX-формате оно будет иметь 4 разряда, т. е. запишется, как 000Dh. Поглядев на эти две записи, вы можете понять, для чего пользуются шестнадцатеричной системой — запись получается намного компактней.

Формат BCD

Электронные устройства «заточены» под двоичную и родственные им системы счисления, потому что основой являются два состояния — двоичная цифра. Так что соединив несколько устройств вместе с целью оперирования с многоразрядными числами, мы всегда будем получать именно двоичное число. При этом четыре двоичных разряда могут представлять шестнадцать различных состояний, и задействовать их для представления десятичных чисел было бы попросту неэкономично: часть возможного диапазона осталась бы неиспользованной. Подсчитайте сами: для представления числа с шестью десятичными разрядами в десятичном виде нужно $6 \times 4 = 24$ двоичных разряда, а для представления того же числа в двоичном виде с избытком хватит 20 разрядов ($2^{20} = 1\,048\,576$). А меньше, чем четыре двоичных разряда, для представления одного десятичного числа не хватит ($2^3 = 8$). К тому же с чисто двоичными числами, как мы увидим в дальнейшем, оперировать значительно проще.

И все же применять двоично-десятичный формат приходится всегда, когда речь идет о выводе чисел, например, на цифровой дисплей — двоичные или шестнадцатеричные числа человеку воспринимать, естественно, тяжело: представьте, что мы показываем температуру в виде 1E,D градуса по Цельсию! Многие ли сразу подсчитают, что это означает (примерно) 30,81 градуса?

Поэтому приходится преобразовывать шестнадцатеричные числа в десятичные и хранить их в таких же байтовых регистрах или ячейках памяти. Это можно делать двумя путями: в виде упакованного и неупакованного BCD.

Неупакованный формат попросту означает, что мы тратим на каждую десятичную цифру не тетраду, как необходимо, а целый байт. Зато при этом не возникает разночтений: 05h = 05_{10} и никаких проблем.

Однако ясно, что это крайне неэкономично — байтов требуется в два раза больше, а старший полубайт при этом все равно всегда ноль. Потому BCD-числа при хранении в регистрах всегда упаковывают, занимая и старший разряд второй десятичной цифрой: скажем, число 59 при этом и запишется, как просто 59. Однако это не 59h! 59 в шестнадцатеричной форме есть 3Bh, как мы установили ранее, а наше 59 процессор прочтет, как 5·16 + 9 = 89, что вообще ни в какие ворота не лезет! Поэтому перед проведением операций с упакованными BCD-числами их распаковывают, перемещая старший разряд в отдельный байт и заменяя в обоих байтах старшие полубайты нулями. Иногда для проведения операций с BCD в микропроцессоре или микроконтроллере предусмотрены специальные команды, так что самостоятельно заниматься упаковкой-распаковкой не требуется (такие инструкции есть, например, в системе команд знаменитого 8086, на котором был построен IBM PC). В качестве примера хранения чисел в BCD-формате можно привести значения часов, минут и секунд в энергонезависимых часах компьютера.

Двоичная арифметика

Правила двоичной арифметики значительно проще, чем десятичной, и включают две таблицы — сложения и умножения — несколько похожие на те же таблицы для логических переменных:

0 + 0 = 0	0 · 0 = 0
0 + 1 = 1	0 · 1 = 0
1 + 0 = 1	1 · 0 = 0
1 + 1 = 10	1 · 1 = 1

Как мы видим, правила обычного умножения одноразрядных двоичных величин совпадают с таковыми для логического умножения. Однако правила сложения отличаются, т. к. при сложении двух единиц результат равен 2 и у нас появляется перенос в следующий разряд. Так как умножение многоразрядных чисел сводится к сложению отдельных произведений, то там придется этот перенос учитывать (как это делается на практике, мы увидим при рассмотрении микроконтроллеров).

Возможно, вы удивитесь, но любая электронная схема (и микропроцессор не исключение) умеет только складывать. Свести умножение к сложению легко. А как свести к сложению вычитание и деление? Для этого придется познакомиться с тем, как в электронике представляют отрицательные числа.

Отрицательные числа

Самый простой метод представления отрицательных чисел — отвести один бит (логичнее всего старший) для хранения знака. По причинам, которые вы поймете далее, значение «1» в этом бите означает знак «минус», а «0» — знак «плюс». Как будут выглядеть двоичные числа в таком представлении?

В области положительных чисел ничего не изменится, кроме того, что их диапазон сократится вдвое: например, для числа в байтовом представлении вместо диапазона 0—255 мы получим всего лишь 0—127 (0000 0000 — 0111 1111). А отрицательные числа будут иметь тот же диапазон, только старший бит у них будет равен единице. Все просто, не правда ли?

Нет, неправда. Такое представление отрицательных чисел совершенно не соответствует обычной числовой оси, на которой влево от нуля идет минус единица, а затем числа по абсолютной величине увеличиваются. Здесь же мы получаем, во-первых, два разных нуля («обычный» 0000 0000, и «отрицательный» 1000 0000), во-вторых, оси отрицательных и положительных чисел никак не стыкуются, и выполнение арифметических операций превратится в головоломку. Поэтому поступим так: договоримся, что –1 соответствует число 255 (1111 1111), –2 — число 254 (1111 1110) и т. д., вниз до 128 (1000 0000), которое будет соответствовать –128 (и общий диапазон всех чисел получится от –128 до 127). Очевидно, что если вы в таком представлении хотите получить отрицательное число в обычном виде, то надо из значения числа (например, 240) вычесть максимальное значение диапазона (255) плюс 1 (256). Если отбросить знак, то результат такого вычитания (16 в данном случае) называется еще *дополнением до 2* (или просто *дополнительным кодом*) для исходного числа (а само исходное число 240 тогда будет дополнением до 2 для 16). Название «дополнение до 2» используется независимо от разрядности числа, потому что верхней границей всегда служит степень двойки (в десятичной системе аналогичная операция называется «дополнение до 10»).

Вычитание

Что произойдет в такой системе, если вычесть, например 2 из 1? Запишем это действие в двоичной системе обычным столбиком:

```
 00000001
-00000010
```

В первом разряде результата мы без проблем получаем 1, а уже для второго нам придется занимать 1 из старших, которые сплошь нули, поэтому пред-

ставим себе, что у нас будто бы есть девятый разряд, равный 1, из которого заем в конечном итоге и происходит:

```
(1)00000001
-  00000010
___________
   11111111
```

На самом деле девятиразрядное число 1 0000 0000 есть не что иное, как 256, т. е. то же самое максимальное значение +1, и мы здесь выполнили две операции: прибавили к вычитаемому эти самые 256, а затем выполнили вычитание, но уже в положительной области для всех участвующих чисел. А что результат? Он будет равен 255, т. е. тому самому числу, которое, как мы договорились, представляет собой –1. Таким образом, вычитание в такой системе происходит автоматически правильно, независимо от знака участвующих чисел.

Немного смущает только эта самая операция нахождения дополнения до 2, точнее, в данном случае до 256 — как ее осуществить на практике, если схема всего имеет 8 разрядов? В дальнейшем мы увидим, что на практике это не нужно: при вычитании и в микроконтроллерах, и в обычных электронных счетчиках все осуществляется автоматически.

Впрочем, в микропроцессорах есть обычно и отдельная команда, которая возвращает дополнение до 2. В большинстве ассемблеров она называется `NEG`, от слова «негативный», потому что просто-напросто меняет знак исходного числа, если мы договариваемся считать числа «со знаком». Разберем из любопытства, как ее можно было бы осуществить «вручную», не обращаясь в действительности к 9-му разряду. Для этого выпишем столбиком какое-нибудь число (далее для примера — 2 и 240), результаты операции нахождения его дополнения до 2, и результат еще одной манипуляции, которая представляет собой вычитание единицы из дополнения до 2, или, что то же самое, просто вычитания исходного числа из наивысшего числа диапазона (255):

Исходное число	2	`00000010`
Дополнение до 2	256 – 2 = 254	`11111110`
Дополнение до 1	255 – 2 = 253	`11111101`

Исходное число	240	`11110000`
Дополнение до 2	256 – 240 = 16	`00010000`
Дополнение до 1	255 – 240 = 15	`00001111`

Если мы сравним двоичные представления в верхней и нижней строках в каждом случае, то увидим, что они могут быть получены друг из друга путем инверсии каждого из битов. Эта операция называется нахождением *обратного кода* или *дополнения до единицы* (потому что число, из которого вычитается, содержит все 1 во всех разрядах; для десятичной системы аналогичная операция называется «дополнение до 9»). Для нахождения дополнения до 1 девятый разряд не требуется, да и схему можно построить так, чтобы никаких вычитаний не производить, а просто «переворачивать» биты. Так и делается, конечно, но не ищите в микроконтроллерах специальной операции инверсии битов — для этого вызывается именно команда нахождения дополнения до 1 (в AVR-ассемблере она обозначается, как `COM`, и определяется, как операция вычитания из FFh, а что уж там происходит на самом деле — тайна, покрытая мраком). Итак, для полного сведения вычитания к сложению надо проделать три операции:

1. Найти дополнение до 1 для вычитаемого (инвертировать его биты).
2. Прибавить к результату 1, чтобы найти дополнение до 2.
3. Сложить уменьшаемое и дополнение до 2 для вычитаемого.

Заметки на полях

Заметим, что все сложности с этими многочисленными дополнениями связаны с наличием нуля в ряду натуральных чисел — если бы его не было, дополнение было бы всего одно, и операция вычитания упростилась. Так может греки все же были в чем-то правы?

В заключение обратим внимание на еще одно замечательное свойство двоичных чисел, которое часто позволяет значительно облегчить операции умножения и деления: а именно умножению на 2 соответствует операция сдвига всех разрядов числа на один разряд влево, а операции деления на 2 — вправо. Крайние разряды (старший при умножении и младший при делении) в общем случае при этом должны теряться, но в микропроцессорах есть специальный регистр, в один из битов которого *(бит переноса)* эти «потерянные» разряды помещаются. Противоположные крайние разряды (младший при умножении и старший при делении) в общем случае замещаются нулями, но могут замещаться значением бита переноса, что позволяет без лишних проблем делить и умножать числа с разрядностью больше одного байта. Как можно догадаться, умножению и делению на более высокие степени двойки будет соответствовать операция сдвига в нужную сторону на иное (равное степени) число разрядов. Излишне говорить, что операцию сдвига разрядов в электронных схемах производить неизмеримо проще, чем операции деления и умножения. Тот же самый бит переноса используется для размещения переноса в старший разряд при сложении и займа при вычитании.

Глава 8

Математическая электроника или игра в квадратики

> Зеленые квадратики — это почтительность, а красные — ненависть. Чем больше красных квадратиков, тем больше люди вас боятся, крестьяне лучше работают, а у воинов снижаются боевые качества на 25%.
>
> *forum.ogl.ru*

Разновидностей *логических* (или *цифровых* — будем считать, что это синонимы) микросхем, используемых на практике, не так уж и много, и подавляющее большинство из них относится к одной из двух разновидностей — ТТЛ (TTL, Транзисторно-Транзисторная Логика) и КМОП (CMOS, Комплементарные [транзисторы типа] Металл-Окисел-Полупроводник). Различие между ними чисто технологическое, и функционально одноименные элементы из этих серий делают одно и то же и на схемах обозначаются одинаково. С точки же зрения электрических параметров они различаются, хотя и не настолько, чтобы можно было бы заявить об их полной несовместимости.

Транзисторно-транзисторная логика возникла раньше, во второй половине 60-х годов, и стала наследницей диодно-транзисторной логики (ДТЛ) и резисторно-транзисторной логики (РТЛ). Основной родовой признак ТТЛ — использование биполярных транзисторов, причем исключительно структуры *n-p-n*. КМОП же, как следует из ее названия, основана на полевых транзисторах с изолированным затвором структуры МОП, причем комплементарных, то есть обоих полярностей — и с *n*- и с *p*-каналом. Есть и другие цифровые серии, отличающиеся по технологии от этих двух основных, но они либо реализованы исключительно в составе больших интегральных схем (БИС), таких, например, как микросхемы памяти, либо имеют достаточно узкий и специфический диапазон применения, поэтому мы рассматривать их не будем.

ТТЛ-микросхемы мы тоже упомянули исключительно ввиду их еще недавней практической важности, а на практике, за небольшими исключениями, вы будете иметь дело только с КМОП-микросхемами. Все современные микроконтроллеры и другие цифровые микросхемы либо полностью построены на КМОП-технологиях, либо имеют КМОП-совместимые выходы и входы. Базовые серии ТТЛ были существенно более быстродействующими, чем КМОП, но современные микросхемы малой степени интеграции из серий AC (Advanced, т. е. «продвинутая» CMOS) и HC (High-speed, т. е. «высокоскоростная» CMOS) ничуть не уступают по быстродействию микросхемам ТТЛ при сохранении почти всех преимуществ и приятных особенностей КМОП. Которые мы сейчас и разберем.

Базовый логический элемент КМОП

Схемотехника базовых логических элементов КМОП приведена на рис. 8.1. На Западе такие элементы еще называют *вентилями* — чем можно оправдать такое название, мы увидим далее. Они довольно близки к представлению о том, каким должен быть идеальный логический элемент.

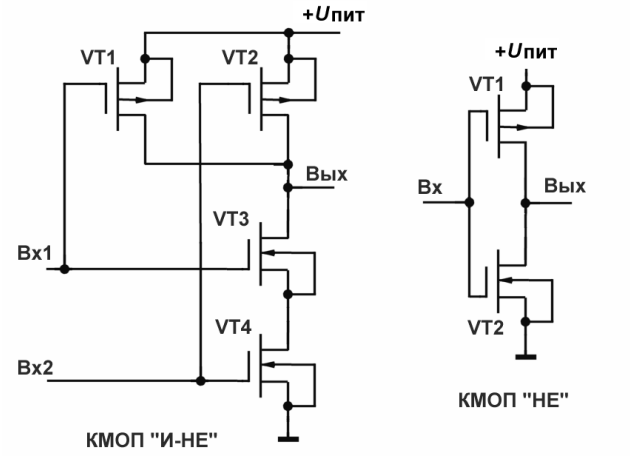


Рис. 8.1. Схемы базовых элементов КМОП

Как можно видеть из рис. 8.1, КМОП-элементы практически симметричны, как по входу, так и по выходу. Открытый полевой транзистор на выходе (либо *p*-типа для логической 1, либо *n*-типа для логического 0) фактически представляет собой, как мы знаем из *главы 3*, просто резистор, величина которого для разных КМОП-элементов может составлять от 100 до 1000 Ом. Причем

для дополнительной симметрии и повышения коэффициента усиления на выходе реальных элементов обычно ставят последовательно два инвертора, подобных показанному на рис. 8.1 справа (жалко, что ли транзисторов?). Не мешает даже то, что в нижнем плече для схемы «И-НЕ» стоят два транзистора последовательно (для схемы «ИЛИ-НЕ» они будут в верхнем плече, поскольку она полностью симметрична схеме «И-НЕ»). Обратите внимание, что выходной каскад инвертора построен не по схеме «пушпульного» каскада, т. е. это не истоковые повторители напряжения, а транзисторы в схеме с общим истоком, соединенные стоками, что позволяет получить дополнительный коэффициент усиления по напряжению.

На практике это приводит к следующим особенностям КМОП-микросхем:

- напряжение логической единицы практически равно напряжению питания, а напряжение логического нуля практически равно потенциалу «земли» (при ненагруженных выходах);
- порог переключения практически равен половине напряжения питания;
- входы в статическом режиме не потребляют тока, т. к. представляют собой изолированные затворы МОП-транзисторов;
- в статическом режиме весь элемент также не потребляет тока от источника питания.

Представляете: схема любой степени сложности, построенная с помощью КМОП-элементов, в «застывшем» состоянии или при малых рабочих частотах (не превышающих десятка-другого килогерц), практически не потребляет энергии! Отсюда ясно, как стали возможными такие фокусы, как наручные часы, которые способны идти от малюсенькой батарейки годами или sleep-режим микроконтроллеров, в котором они потребляют от 1 до 50 мкА (о нем см. *главу 17*). Другое следствие перечисленных особенностей — исключительная помехоустойчивость, достигающая половины напряжения питания.

Но это еще не все преимущества — КМОП-микросхемы базовой серии (о различных сериях см. далее), подобно многим операционным усилителям, могут работать в диапазоне напряжений питания от 3 до 15 В. Единственное, при снижении питания довольно резко (в разы) падает быстродействие.

Выходные транзисторы КМОП, как и любые другие полевые транзисторы, при перегрузке (например, в режиме короткого замыкания) работают, как источники тока: при напряжении питания 15 В этот ток для КМОП-элементов базовой серии составит около 30 мА, при 5 В — около 5 мА. Нагрузка при сохранении требований к логическим уровням (которые здесь обычно полагается иметь в пределах от 0 В до $0{,}1U_{\text{пит}}$ — логический ноль, и от $U_{\text{пит}}$ до $0{,}9U_{\text{пит}}$ — логическая единица) номинально ограничена величиной

примерно 1 кОм (т. е. ток порядка 1 мА). Но для некоторых разновидностей (как для выходов микроконтроллеров AVR) допустимый ток значительно выше, и может достигать 20—40 мА. Причем это штатный долгосрочный режим работы таких элементов, единственное, что при этом надо проверить: не превышается ли предельно допустимое значение рассеиваемой мощности для корпуса (0,5—0,7 Вт). В противном случае, возможно, придется ограничить число выходов, одновременно подключенных к низкоомной нагрузке.

Заметки на полях

ТТЛ-микросхемы значительно менее удобны на практике, поскольку для них характерно балансирование десятыми вольта: напряжение логического нуля составляет не более 0,8 В, напряжение порога переключения от 1,2 до 2 В, напряжение логической единицы не менее 2,4 В. Иногда вы и сейчас можете встретить подобные требования к логическим уровням (в целях совместимости). ТТЛ могут работать в довольно узком диапазоне напряжений питания: практически от 4,5 до 5,5 В, а нормы предполагают обычно от 4,75 до 5,25 В, т. е. 5 В ±5%. Максимально допустимое напряжение питания составляет для разных ТТЛ-серий от 6 до 7 В, при превышении его они обычно «горят ясным пламенем». Низкий и несимметричный относительно питания порог срабатывания элемента приводит и к невысокой помехоустойчивости. Другим крупным (и даже более серьезным, чем остальные) недостатком ТТЛ является высокое потребление (до 2,5 мА на один базовый элемент), так что приходится только удивляться, почему микросхемы ТТЛ, содержащие много таких элементов, не требуют охлаждающего радиатора. По всем этим причинам, даже если вы будете повторять старые схемы на ТТЛ-микросхемах, их рекомендуется заменять на современные АС или НС-элементы КМОП, с которыми они совместимы по выводам.

И тут мы плавно переходим к основному недостатку базовых КМОП-технологий — низкому (в сравнении ТТЛ) быстродействию. Это обусловлено тем, что изолированный затвор МОП-транзистора представляет собой конденсатор довольно большой емкости (в базовом элементе до 10—15 пФ). В совокупности с выходным резистивным сопротивлением предыдущей схемы такой конденсатор образует фильтр нижних частот. Обычно рассматривают не просто частотные свойства, а время задержки распространения сигнала на один логический элемент, которое может достигать у базовой серии КМОП величины 250 нс (сравните: у базовой серии ТТЛ — всего 10 нс), что соответствует одному периоду частоты 4 МГц. На практике при напряжении питания 5 В быстродействие базового КМОП не превышает 13 МГц. Попробуйте соорудить на логических элементах генератор прямоугольных сигналов по любой их схем, которые будут разобраны далее, и вы увидите, что уже при частоте 1 МГц форма сигнала будет скорее напоминать синусоиду, чем прямоугольник.

Другим следствием высокой входной емкости является то, что при переключении возникает импульс тока перезарядки этой емкости, т. е. чем выше

рабочая частота, тем больше потребляет микросхема, и при максимальных рабочих частотах ее потребление может сравниться с потреблением ТТЛ.

Развитие КМОП было, естественно, направлено в сторону устранения или хотя бы сглаживания этих недостатков. Однако, в отличие от ТТЛ, базовый вариант которой, представленный в отечественном варианте сериями 155 и 133, сейчас практически забыт (исключение см., например, *главу 19*, раздел «Аналоговая индикация»), оригинальная базовая серия 4000B[1] применяется и по сей день — в основном из-за неприхотливости и беспрецедентно широкого диапазона питающих напряжений (от 3 до 18 В), что позволяет без излишних проблем совмещать цифровые и аналоговые узлы в одной схеме.

Отечественные аналоги стандартной серии CD4000B — это «бытовая» серия К561 в корпусе типа DIP, или «военная» 564 в планарном корпусе, аналоге американского SOIC или SOT. Имеется и ряд уже упоминавшихся быстродействующих КМОП-элементов (в первую очередь серии AC и HC). Для быстродействующих серий пришлось пожертвовать расширенным диапазоном питания, например, номинальный диапазон напряжения питания для 74HC начинается, правда, от 2, но простирается всего до 6 В, отсюда и популярность старинной CD4000B. Для быстродействующих КМОП западное название серии (74) и разводка выводов микросхем совпадает со старой базовой ТТЛ (а не с CD4000B), что, безусловно, было продиктовано маркетинговыми соображениями, но сделало базовую серию несовместимой с быстродействующими по выводам. Отечественный аналог называется логичнее — 1561 или 1564, но разводка выводов, увы, в целях совместимости с западными также совпадает с ТТЛ, а не с базовой КМОП. Чтобы не запутаться в зарубежных наименованиях (что там ТТЛ, а что КМОП), можно применять простое правило: если в наименовании серии присутствует буква C (от «комплементарный», кроме HC и AC, есть и просто C), то это КМОП, все остальные многочисленные представители семейства 74 есть ТТЛ-микросхемы.

Подробности

Как мы договаривались в *главе 6*, префикс «К» в наименовании микросхем мы в дальнейшем будем опускать, но серия 561 в «военном» варианте (без буквы «К») не существует, и перепутать невозможно. «Военный» вариант на Западе называют промышленным — industrial, а «бытовой» — коммерческим — commercial (отсюда буква «К» в отечественном варианте), вместе с тем «у них там» имеется еще отдельно довольно редкий чисто «военный» — military. На практике в розницу попадает из западных микросхем только вариант commercial (отечественные можно сейчас часто купить и те и другие), industrial

[1] У крупнейшего производителя этих микросхем, фирмы Fairchild Semiconductor, принято название CD4000B, и мы тоже так ее будем называть. У других производителей могут отличаться буквы в наименовании (см. *Приложение 3*).

нужно специально заказывать. Коммерческие («бытовые») компоненты отличаются в основном тем, что гарантированно работают при температурах от 0 до 70 °С. Промышленный (industrial) диапазон обычно составляет от –40 до +85 °С (а иногда и значительно шире, транзисторы или микросхемы стабилизаторов питания, скажем, спокойно работают до +125 градусов и выше). Но не следует думать, что микросхема коммерческого диапазона сразу выйдет из строя, если вы ее охладите до –30° или нагреете до 100°. Вовсе нет, т. к. делаются они все обычно на одной линии (кроме каких-нибудь экстремально-космических применений, да и то не всегда), просто производитель не гарантирует, что в расширенном температурном диапазоне данная микросхема сохранит все оговоренные в описании характеристики.

Незадействованные входы элемента КМОП нужно обязательно подключать куда-нибудь — либо к «земле», либо к питанию, либо объединять с соседним входом — иначе наводки на столь высокоомном входе полностью нарушат работу схемы. Причем в целях снижения потребления следует делать это и по отношению к входам незадействованных *элементов* в том же корпусе (но не вообще ко всем *выводам*!). «Голый» вход КМОП из-за своей высокоомности может быть также причиной повышенной «смертности» чипов при воздействии статического электричества, однако на практике входы всегда шунтируют диодами, как показано на рис. 6.5.

Основные логические элементы

На рис. 8.2 показаны условные обозначения основных логических элементов на электрических схемах, причем нельзя не согласиться, что отечественные обозначения намного логичнее, легче запоминаются и проще выполняются графически, чем западные. Поэтому западные обозначения логических элементов у нас так и не прижились (как, кстати, и многие другие, например, обозначения резисторов и электролитических конденсаторов), и приведены здесь только для справки.

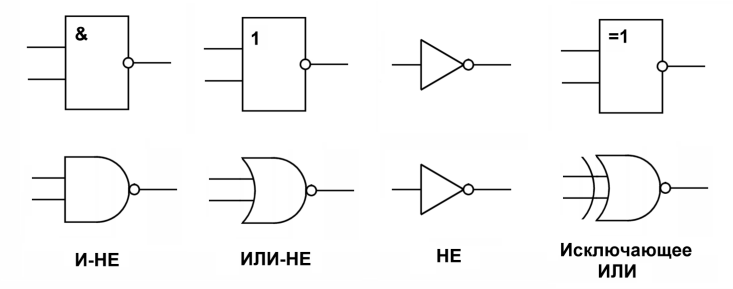


Рис. 8.2. Обозначения основных логических элементов на схемах: вверху — отечественное, внизу — западное

Крайний справа элемент под наименованием «Исключающее ИЛИ» нам еще неизвестен, но скоро мы его будем изучать. В табл. 8.1 приведена разводка выводов микросхем, содержащих логические элементы — она одинакова для всех трех наиболее употребляемых логических типов (напомним, что для серий ТТЛ и быстродействующей КМОП разводка будет другая). Естественно, все элементы из одного корпуса абсолютно идентичны и взаимозаменяемы, поэтому для таких микросхем номера выводов корпуса и расположение выводов питания на схеме обычно не указывают.

Таблица 8.1. *Разводка выводов КМОП-микросхем серии CD400B (561, 564) с четырьмя двухвходовыми логическими элементами*

«И-НЕ» CD4011, 561ЛА7 (корпус DIP-14), 564ЛА7 (корпус SOP-14) «ИЛИ-НЕ» CD4001, 561ЛЕ5 (корпус DIP-14), 564ЛЕ5 (корпус SOP-14) «Исключающее ИЛИ» CD4030, 561ЛП2 (корпус DIP-14), 564ЛП2 (корпус SOP-14)							
Вывод	1	2	3	4	5	6	7
Функция	Вх1/1	Вх1/2	Вых1	Вых2	Вх2/1	Вх2/2	Общ
Вывод	8	9	10	11	12	13	14
Функция	Вх3/1	Вх3/2	Вых3	Вых4	Вх4/1	Вх4/2	+ пит

Подробности

Мы будем использовать в схемах и простые одновходовые инверторы — это микросхема 561ЛН2, содержащая 6 таких инверторов в одном корпусе DIP-14. Разводка выводов у нее такая (первая цифра — вход, вторая — выход): 1—2, 3—4, 5—6, 9—8, 11—10, 13—12, питание обычное, т. е. «+» к выводу 14, «земля» — к выводу 7. Отметим, что точного импортного аналога этой микросхемы не существует, есть микросхема CD4049 в корпусе DIP-16, у которой разводка несколько другая, идентичная микросхеме, содержащей 6 просто буферных усилителей без инверсии (561ПУ4 или CD4050): питание (внимание!) — к выводу 1, «земля» — к выводу 8, сами же элементы расположены так: 3—2, 5—4, 7—6, 9—10, 11—12, 14—15, выводы 13 и 16 не задействованы (и, напомним, не должны никуда присоединяться!).

Есть, разумеется, и элементы с бóльшим числом входов, пример их использования мы увидим далее. Я не буду приводить здесь разводку выводов других типов логических микросхем, т. к. эти данные всегда можно найти в справочниках, например в [9]. Отдельно следует упомянуть, что многие микросхемы КМОП прекрасно коммутируют аналоговые сигналы, иногда даже специально делается отдельный вывод для подключения отрицательного напряжения питания, чтобы можно было пропускать двуполярное напряжение. Причем пропускание это осуществляется как в направлении от входа

к выходу, так и обратно (таковы микросхемы 561КТ3, КП1 и КП2 или, скажем, специально для этого предназначенные микросхемы серии 590КН*x*). Указанные микросхемы прекрасно работают также и с цифровыми сигналами, т. е. являются универсальными. Немного подробнее мы их рассмотрим далее.

Другой часто употребляемой разновидностью логических микросхем (в основном, правда, в составе больших интегральных схем, БИС) являются элементы, имеющие выход с открытым коллектором (или с открытым истоком). Такой выход, как мы помним, имеет компаратор 554СА3 (см. *главу 6*). Есть такие элементы и с чисто логическими функциями: в КМОП-серии это CD40107 (561ЛА10), а в ТТЛ — 7403 (133ЛА7). Как правило, они могут коммутировать значительный ток (до 50 мА, причем интересно, что для схемы с открытым истоком, типа ЛА10, нагрузочная способность по току с увеличением напряжения питания растет, а не падает).

Эти элементы удобны не только для коммутации мощной нагрузки, но и для объединения на общей шине в так называемое «проводное» или «монтажное» ИЛИ. В этом случае объединенные коллекторами (истоками) транзисторы разных устройств работают на общую нагрузку. В нормальном состоянии все они разомкнуты и на шине имеется потенциал логической единицы. Любое устройство может перевести шину в состояние логического нуля, замкнув выходной транзистор, при этом состояния всех остальных устройств уже не будут иметь значения (т. н. «захват шины»). Электрических конфликтов на такой шине возникнуть не может, т. к. ток от источника питания всегда ограничен нагрузочным резистором. Примером такой шины может служить интерфейс I^2C, который мы будем разбирать в *главе 16*.

Другой вариант построения выходов современных КМОП-элементов для коллективной работы представляет т. н. *выход с третьим состоянием*, когда оба транзистора, и по «плюсу» и по «минусу», могут быть разомкнуты. Так построены выходные каскады микроконтроллеров AVR, с которыми мы будем иметь дело в дальнейшем (см. *главу 12*).

Обработка двоичных сигналов с помощью логических элементов

В начале главы мы упоминали, что логические элементы носят еще название вентилей. На самом деле вентиль — это устройство для регулирования потока жидкости или газа. Каким же образом оправданно это название в приложении к нашим схемам? Оказывается, если на один из входов логического элемента подавать последовательность прямоугольных импульсов (некую аналогию потока жидкости), а на другой — логические уровни, то элемент будет себя вести совершенно аналогично вентилю.

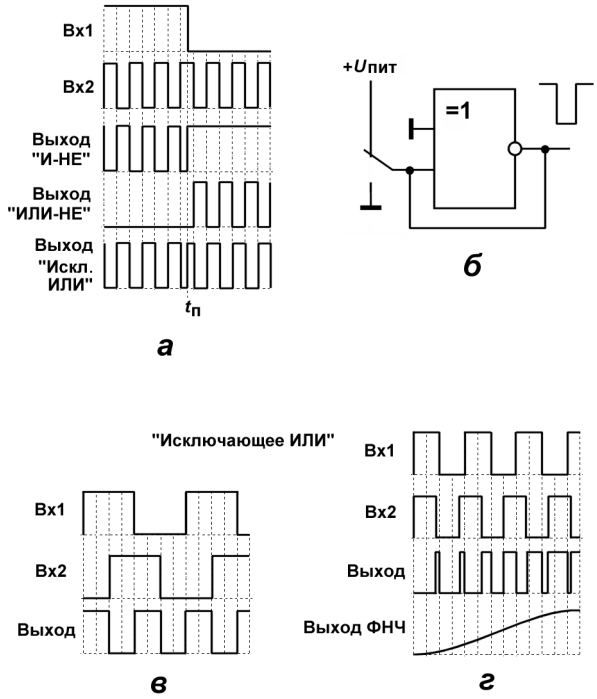


Рис. 8.3. Обработка цифровых сигналов при помощи логических элементов:
а — диаграммы прохождения сигналов через основные типы логических элементов;
б — «антидребезг» на основе элемента «Исключающее ИЛИ»;
в и *г* — использование элемента «Исключающее ИЛИ»
для выявления разности фаз (*в*) и частот (*г*) сигналов

Соответствующие диаграммы показаны на рис. 8.3, *а*. Из них вытекают следующие правила:

- для элемента «И-НЕ» логический уровень «1» является *разрешающим*, т. е. в этом случае последовательность на другом входе пропускается на выход без изменения (за исключением того, что она инвертируется, т. к. элемент у нас «И-НЕ», а не просто «И»). При логическом уровне «0» вентиль запирается, на выходе будет логическая единица;
- для элемента «ИЛИ-НЕ» ситуация полностью обратная: *разрешающим* является логический уровень «0», т. е. в этом случае последовательность на другом входе пропускается на выход (также с инверсией). При логическом уровне «1» вентиль запирается, на выходе будет логический ноль;
- для «Исключающего ИЛИ» все еще интересней: в зависимости от того, «0» на входе или «1»: относительно другого входа элемент ведет себя,

соответственно, как повторитель или как инвертор, что дает довольно широкие возможности для управления двоичными последовательностями. Почему так происходит?

Исключающее ИЛИ

Элемент «Исключающее ИЛИ» обладает рядом интересных свойств, которые вытекают из его таблицы истинности:

«Исключающее ИЛИ»		
Вх1	**Вх2**	**Вых**
0	0	0
0	1	1
1	0	1
1	1	0

Если сравнить эту таблицу с таблицами элементов «ИЛИ» и «И-НЕ» (см. *главу 7*), то можно заметить, что «Исключающее ИЛИ» есть логическое произведение этих элементов. Запомнить его таблицу истинности очень просто — он осуществляет функцию «несовпадения» (единица на выходе тогда, когда входы разные). Признаюсь, что я никогда не мог понять, почему в отечественной практике «Исключающее ИЛИ» обозначают значком «=1». По смыслу это обозначение больше подошло бы обратному элементу «совпадения», который представляет собой инверсию выхода «Исключающего ИЛИ» и носит название «Включающего ИЛИ». В любой логической серии есть специальные микросхемы «Исключающее ИЛИ» (561ЛП2), но функцию эту несложно воспроизвести и на базовых элементах, например, «И-НЕ».

Заметки на полях

Сама функция «Исключающее ИЛИ» (по-английски она называется XOR) имеет большое значение в логике и программировании. Например, часто употребляющаяся функция обнуления какого-то регистра в микроконтроллерах есть операция «Исключающее ИЛИ» этого регистра самого с собой (по определению, одинаковые входы дадут на выходе все 0). Другое интересное свойство этой функции — будучи применена к какому-то двоичному объекту дважды, она возвращает все, как было до операции. На этом принципе основано применение функции «Исключающее ИЛИ» в криптографии: первый раз вы складываете текст с секретным ключом, получая «абракадабру». Второй раз — на приемном конце — тот же ключ той же операцией применяется к этой «абракадабре»,

в результате чего получается исходный расшифрованный текст. Можно указать и еще одно распространенное применение «Исключающего ИЛИ», которое вы не раз встречали — получение «прозрачного» цвета в компьютерной графике. Эффект основан на том, что изготавливается т. н. XOR-маска. Там, где в XOR-маске были поля с нулевым значением бит (т. е. черного цвета), фон остается неизменным (если одна исходная величина равна нулю, то операция XOR будет повторять вторую исходную величину), в противном случае на фон с «дырой» накладывается то изображение, которое записано в XOR. Так, например, формируются «иконки» и «прозрачные» меню в Windows. Как видите, очень полезная функция.

На рис. 8.3, *б* показана интересная схема на основе элемента «Исключающее ИЛИ». Она устраняет неизбежный дребезг механических контактов, который может вызвать (более того, вызывает обязательно) многократное срабатывание некоторых электронных схем, например триггеров или счетчиков. С этим явлением борются разными путями — с помощью одновибраторов (см. далее), RS-триггеров (см. *главу 9*) и даже программно — в микроконтроллерах (см. *главу 13*).

При наличии свободного элемента «Исключающее ИЛИ» устранить дребезг, как видите, очень просто. Чтобы понять, как это работает, надо учесть, что подвижные контакты кнопки, тумблера или реле *никогда* не пролетают несколько раз расстояние от одного неподвижного контакта до другого — подвижной контакт только несколько (иногда до нескольких десятков) раз за короткое время оказывается «висящим в воздухе» (представьте себе, что он как бы подпрыгивает на неподвижном контакте, причем как при размыкании, так и при замыкании). При этом подача напряжения, соответствующего *противоположному* логическому уровню, не происходит, но «подпрыгивания» достаточно, чтобы на выходе микросхемы вызвать нечто похожее на дребезг выхода компаратора при наличии помехи. Но, согласно сказанному ранее, при наличии логического нуля на одном из входов «Исключающее ИЛИ» работает как повторитель. Если контакт был замкнут (надежно) с потенциалом питания (логическая единица), то на выходе будет также «1». Когда контакт в процессе дребезга разомкнется и «повиснет в воздухе», то потенциал на выходе все равно останется равным «1», т. к. поддерживается обратной связью, замыкающей выход со входом. Сколько бы контакт ни дребезжал таким образом, потенциал останется равным «1» до первого касания контактом «земли», тогда элемент перебросится в другое состояние и будет в нем пребывать опять независимо от того, дребезжит контакт или нет. Разумеется, можно и инвертировать сигнал, если присоединить второй вход к питанию, а не к «земле». В схеме по рис. 8.3, *б* обязательно требуется именно перекидной контакт, для простой кнопки с двумя выводами нужны иные способы.

Самое, однако, интересное будет, если на входы «Исключающего ИЛИ» подать две последовательности импульсов с разными частотами и/или фазами. На рис. 8.3, *в* показано, что произойдет, если обе последовательности имеют одинаковую частоту, но фазы при этом сдвинуты на полпериода. На выходе при этом возникнет колебание с удвоенной частотой! Попробуйте изменить фазу — вы увидите, что скважность результирующего колебания будет меняться, пока фазы не совпадут, и тогда сигнал на выходе исчезнет — одинаковые состояния выходов дают на выходе «Исключающего ИЛИ» всегда логический ноль. Это позволяет использовать такой элемент в качестве т. н. *фазового компаратора*, что широко применяется в фазовых модуляторах и демодуляторах сигнала.

Не менее интересный случай показан на рис. 8.3, *г*, где на входы подаются последовательности с *различающейся* частотой. Мы видим, что на выходе возникнет сигнал с меняющейся скважностью, причем легко показать, что период изменения скважности от минимума к максимуму и обратно будет в точности равен периоду сигнала с частотой, равной *разности* исходных частот. Если при этом поставить на выходе элемента фильтр низкой частоты (если разность частот невелика в сравнении с исходными частотами, то достаточно простой RC-цепочки), то мы получим синусоидальное колебание с частотой, равной этой разности! Это колебание можно подать, например, в качестве сигнала обратной связи на генератор, управляемый напряжением (ГУН), который тогда изменит частоту своего выходного сигнала так, чтобы она в точности совпадала со второй (опорной). Так, к примеру, делают схемы умножителей частоты, получая целый набор точных частот при наличии одного-единственного опорного кварцевого генератора.

Использование статической логики

На практике базовые логические элементы работают в основном в качестве управляемых вентилей, как описано ранее, для согласования положительной и отрицательной логики, а также в т. н. *комбинационных* схемах, которые в чистом виде реализуют логические уравнения (см. *главу 7*) разной степени сложности. Конечно, в массовых продуктах, кроме самых простых устройств, микросхемы малой степени интеграции сейчас почти не встречаются, т. к. им на смену пришли более функциональные и удобные ПЛИС и микроконтроллеры. Но для практики, особенно радиолюбительской, простые комбинационные схемы могут оказаться полезными. Мы рассмотрим один класс таких схем — дешифраторы.

Коды и шифры

Сначала внесем некоторую ясность в терминологию. Под словом «коды» ученые-криптографы чаще всего понимают словесный код: «первый, я третий, какие указания?». Типичным кодом также были уловки, которыми алхимики охраняли свои производственные секреты («возьми, сын мой, философской ртути и накаливай, пока она не превратится в зеленого льва…»). Такие тайные коды точными науками не рассматриваются, и применяются лишь в быту.

Другое дело — различные системы счисления, которые мы рассматривали в *главе 7*, по сути они также есть не что иное, как коды, в данном случае применяющиеся для обозначения чисел. На самом деле к таким кодам более применимо слово *шифры*. Наука криптография имеет дело немного с другими шифрами, но строгое определение этого понятия (использование математических приемов шифровки текста с возможностью его *однозначного восстановления* при знании ключа, см. ранее про «Исключающее ИЛИ») вполне применимо и к двоичным числам — к *двоичному коду*.

Кроме двоичного, шестнадцатеричного и т. д., есть и другие коды, и не только для чисел, но и для букв алфавита, и мы рассмотрим это чуть далее. А пока внесем все же ясность: в электронике кодом называют некую систему представления чисел и букв, которая позволяет однозначно перевести представленную информацию в любую другую подобную систему. Устройства, позволяющие осуществлять такой перевод, называют шифраторами и дешифраторами — совсем, как у шпионов. Ясно, что между шифратором и дешифратором нет строгого различия (что считать за исходную систему?), но, как правило, шифратором называют устройство, которое преобразует данные в двоичный код, а дешифратором — наоборот, из двоичного кода.

Двоичный код — это отнюдь не только двоичные числа, им можно закодировать все, что угодно, единственное требование — чтобы в самом коде участвовало лишь два знака, ноль и единица. В этом смысле широко известная азбука Морзе двоичным кодом является лишь по видимости — на самом деле двух символов (точки и тире) в нем недостаточно, там есть как минимум еще один знак — пауза. В цифровой технике паузу можно было бы и не считать за отдельный знак, когда она просто разделяет точки и тире между собой — если бы только не необходимость разделять буквы и слова. В азбуке Морзе количество точек и тире в коде отдельных символов не фиксировано, оно может быть равно всего одному, а может — двум, трем, вплоть до пяти, так что если вы просто формально запишете подряд точки и тире, как единицы и нули, то прочесть ничего не сможете, не зная, где именно в данном случае заканчивается одна буква и начинается другая.

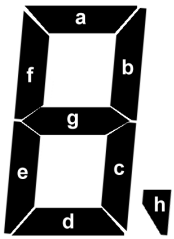
Более распространены в электронике т. н. *равномерные коды*, в которых число разрядов постоянно и определено заранее, благодаря чему разделители не требуются. Но бывает и иначе, например, почтовая кодировка UTF-8, основанная на 16-битовом Unicode, использует неравномерные коды. За экономию объема сообщения приходится расплачиваться довольно сложным алгоритмом кодирования и декодирования. Известная азбука Брайля для слепых — типичный пример двоичного кода, в котором число разрядов равно 6. По понятным причинам чаще употребляют коды, в которых число разрядов равно 8, и составляет целый байт, или более длинные, с числом разрядов, кратным 8. Типичные представители таких кодов — всем известный ASCII, который составляет основу любой современной компьютерной кодировки, а также Unicode — двухбайтовый код, позволяющий закодировать знаки всех алфавитов мира.

Управление цифровыми индикаторами

В электронике для разных целей необходимы и другие двоичные коды. Мы сейчас рассмотрим код, который для краткости будем называть *семисегментным* — т. е. тот, что служит для отображения численной информации на цифровых семисегментных индикаторах. Один разряд такого индикатора, как мы знаем из *главы 3*, состоит обычно из семи сегментов-полосок (на светодиодах или жидких кристаллах), расположенных в пространстве определенным образом. На рис. 8.4 показано расположение и общепринятые обозначения этих сегментов буквами (*a*, *b*, *c* и т. д., употребляются и большие, и маленькие буквы), а также соответствующий семисегментный код, в котором «1» обозначает светящийся сегмент, а «0» — несветящийся. Сегмент *h*, который представляет собой запятую, в формировании цифры не участвует, управляется отдельно и потому мы его не рассматриваем. Разумеется, на семисегментных индикаторах можно отобразить и некоторые другие символы (знак «минус», буквы C, A, F, П, H или P), но эти возможности мы также оставим пока в стороне. Для того чтобы отобразить больше символов, есть и другие типы цифровых индикаторов, например, шестнадцатисегментные или матричные, но управление ими сложнее и мы остановимся на семисегментных.

Как заставить электронную схему показывать цифру на семисегментном индикаторе? Ведь изначально мы чаще всего имеем цифру в двоичном (а не двоично-десятичном) представлении. Четырехразрядное двоичное число имеет диапазон от 0 до 16, что на одном семисегментном индикаторе отобразить невозможно даже в шестнадцатеричном исполнении, т. к. букву «B» нельзя будет отличить от восьмерки, а «D» — от нуля. Это означает, что, вообще говоря, мы должны провести нашу перекодировку в два этапа: сначала представить число в распакованном двоично-десятичном виде (см. *главу 7*), затем преобразовать четырехразрядный двоично-десятичный код в семисег-

ментный. Здесь мы ограничимся рассмотрением только последней задачи, т. к. первая практически всегда уже решена, поскольку двоично-десятичное число является результатом счета с помощью двоично-десятичного же счетчика (см. *главу 9*) или формируется в регистрах микроконтроллера программно.



DEC	BIN	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1

Рис. 8.4. Обозначения сегментов и таблица состояний семисегментного индикатора

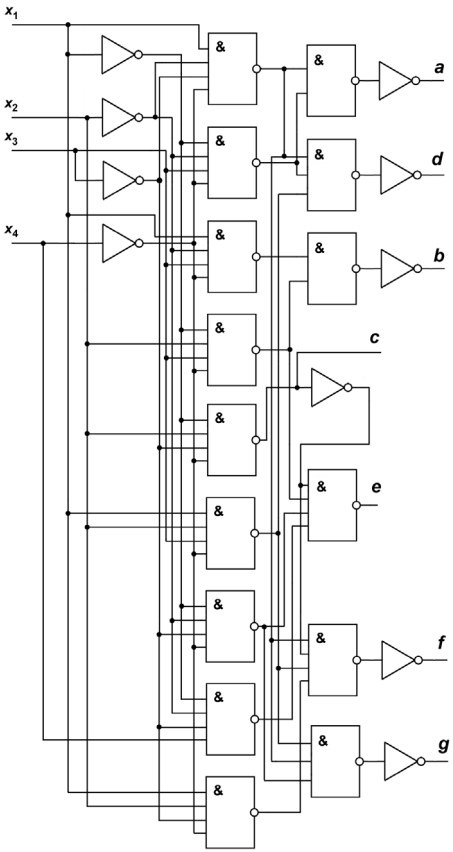


Рис. 8.5. Полная схема дешифратора для управления семисегментным индикатором

На рис. 8.5 для примера приведена схема дешифратора двоичного кода в семисегментный, созданная на базовых логических элементах и работающая в положительной логике (светящемуся сегменту соответствует высокий уровень напряжения, предполагается использование индикаторов с общим катодом). Здесь $x_1 \ldots x_4$ — входной двоично-десятичный код, причем x_1 соответствует младшему разряду. Как видите, схема довольно громоздкая: потребуется 10 инверторов, 11 четырехвходовых «И-НЕ», 2 трехвходовых «И-НЕ», 2 двухвходовых «И-НЕ». В зависимости от выбора реальной элементной базы (согласно приведенным в *главе 7* логическим соотношениям, одни и те же функции можно реализовать по-разному), число необходимых микросхем может составить порядка 10 корпусов и более. И это без учета того, что напрямую подключать светодиодные индикаторы к этой схеме нельзя, т. к. слишком велика нагрузка, которая может «просадить» логический уровень, например, на выводе сегмента «c». Потому здесь потребуются еще усилители-повторители выходного сигнала. А для жидкокристаллических индикаторов необходим генератор прямоугольных импульсов, т. к. они управляются переменным напряжением.

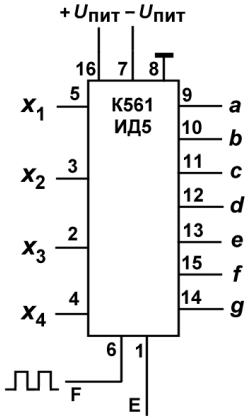


Рис. 8.6. Разводка выводов дешифратора 561ИД5

По этим причинам на практике всегда, конечно, ставят готовые дешифраторы, которые часто называют еще *драйверами* индикаторов. Один из самых простых таких дешифраторов — микросхема 561ИД5 (рис. 8.6), которая «заточена» под управление ЖК-дисплеями и обладает расширенной функциональностью по сравнению с простейшей схемой рис. 8.5. Она подает пульсирующий, как это полагается, сигнал на ЖК-сегменты, причем диапазон

питания может быть расширен в отрицательную сторону, так что микросхема может управлять большими индикаторами, для которых требуется повышенное напряжение. Кроме этого, в ней имеется возможность «защелки» состояния выходов, что важно при управлении индикацией состояния счетчиков: на вход «защелки» E следует подавать высокий уровень (логическую единицу), когда нужно, чтобы результаты сменились, в противном случае состояние выходов a—g «зависнет», независимо от изменения входных кодов. Можно индицировать буквы L, R, P, H и знак «минус» (схема по рис. 8.5 коды 0Ah—0Fh вовсе не использует, выходная комбинация будет совпадать с уже известными). Несложно приспособить эту микросхему и для управления светодиодными индикаторами — для этого придется поставить ключевые транзисторы или просто буферные усилители (микросхемы типа 561ЛН2 или 561ПУ4) по выходу. Для ЖК-индикаторов на вход F подают частоту в несколько десятков герц, для светодиодных — напряжение логической единицы.

Двоичный/десятичный дешифратор

Особый интерес, конечно, представляют дешифраторы двоичного кода в десятичный (а иногда и шифраторы — преобразователи десятичного кода в двоичный). Сначала только надо разобраться, что мы имеем в виду, когда говорим «десятичный код»? На самом деле рассмотренный семисегментный код тоже, по сути, есть десятичный код, представляющий рисунок цифр в определенной системе начертания знаков. Здесь же под десятичным кодом мы будем иметь в виду десятиразрядное двоичное число, которое имеет ровно десять различных состояний: когда в соответствующем разряде появляется единица, остальные при этом находятся в нулевом состоянии. Такое представление десятичного кода легко интерпретировать в виде табло с десятью лампочками, подсвечивающими в каждый момент времени только одну нужную цифру.

Для решения такой задачи обратимся к благословенным производителям микросхем, которые за нас уже все, как водится, придумали: это микросхема 561ИД1, разводка выводов которой приведена на рис. 8.7, *а*. Здесь цифры 1, 2, 4 и 8 внутри прямоугольника, обозначающего микросхему, соответствуют весам двоичных разрядов x_1—x_4, что общепринято для выводов микросхем, представляющих двоичное число. Снаружи цифрами 0—9 обозначены десятичные выходы.

Далее приведена таблица состояний для микросхемы 561ИД1, в том числе и для состояний выхода при входном коде, превышающем девятку (пустые клеточки означают нулевое состояние выхода). Заметим, что коды более 09h (1001) не задействованы (как и в «самодельном» дешифраторе по рис. 8.5),

т. к. при «бессмысленной» с точки зрения двоично-десятичного числа комбинации на входе выходы повторяют то восьмерку, то девятку.

Состояния дешифратора К561ИД1													
Входы				**Выходы**									
x_1	x_2	x_3	x_4	0	1	2	3	4	5	6	7	8	9
0	0	0	0	1									
0	0	0	1		1								
0	0	1	0			1							
0	0	1	1				1						
0	1	0	0					1					
0	1	0	1						1				
0	1	1	0							1			
0	1	1	1								1		
1	0	0	0									1	
1	0	0	1										1
1	0	1	0									1	
1	0	1	1										1
1	1	0	0									1	
1	1	0	1										1
1	1	1	0									1	
1	1	1	1										1

На двух дешифраторах ИД1 можно построить аналогичный преобразователь двоичного кода в шестнадцатеричный. Его схема приведена на рис. 8.7, *б*. При значении входного кода менее 8 работает только верхняя микросхема — по таблице легко убедиться, что подача единицы на разряд x_4 равносильна запрету на дешифрацию состояний входов x_1—x_3. Эту функцию выполняет инвертор, который во входном диапазоне чисел от 0 до 7 на выходе всегда имеет уровень «1» и запрещает функционирование второй — нижней — микросхемы. Когда же входной код принимает значения 8 и выше, то на входе второй микросхемы оказывается фактически код, соответствующий тому же диапазону 0—7 (из входного кода вычитается восьмерка), и она выдает состояния для выходов 08h—0Fh всей схемы. При этом верхняя микросхема, в свою очередь, заперта состоянием единицы на x_4 и неоднозначности не возникает. Выходы 8 и 9 у каждой из микросхем, естественно, не используются.

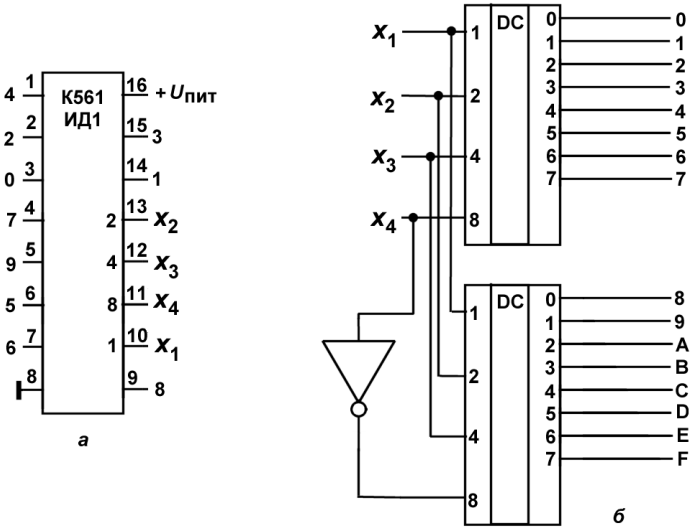


Рис. 8.7. Дешифратор 561ИД: *а* — разводка выводов; *б* — схема двоично/шестнадцатеричного дешифратора

Мультиплексоры/ демультиплексоры и ключи

Мультиплексоры/демультиплексоры — важный класс логических схем малой степени интеграции, о которых мы уже упоминали ранее, в связи с тем, что они прекрасно коммутируют не только цифровые, но и аналоговые сигналы. *Мультиплексором* называют схему, которая коммутирует единственный входной вывод напрямую с одним из нескольких выходных (как правило, четырех или восьми), в зависимости от поданного на нее двоичного кода (схема «1 → 8»). Соответственно, *демультиплексор* выполняет обратную операцию — пропускает сигнал с одного из нескольких выводов на единственный выходной (схема «8 → 1»).

Мультиплексоры в настоящее время делают на *ключах* — специальным образом включенных полевых транзисторах по технологии КМОП. Простейший такой ключ изображен на рис. 8.8, *а*. Он отличается тем, что может пропускать сигнал в обе стороны (на то транзисторы и *униполярные*), поэтому все КМОП-мультиплексоры одновременно являются также и демультиплексорами. Выпускаются также и микросхемы, содержащие просто наборы отдельных ключей, например, 590КН2 и аналогичные, мы еще с ними столкнемся.

Такие ключи часто входят в состав микросхем большей степени интеграции, например, в аналого-цифровых и цифроаналоговых преобразователях. Они практически заменили механические переключатели в коммутаторах телевизионных каналов, используются в цифровых переменных резисторах и т. д.

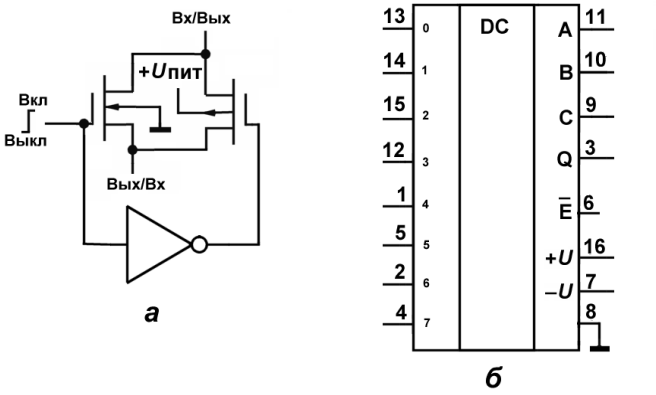


Рис. 8.8. Использование КМОП-ключей: *а* — простейший униполярный ключ; *б* — разводка выводов мультиплексора/демультиплексора 561КП2

На рис. 8.8 *б* приведена для примера схема разводки выводов микросхемы 561КП2, которая представляет собой восьмиканальный мультиплексор/демультиплексор (561КП1 делает то же самое, но содержит два четырехканальных мультиплексора). Эта микросхема коммутирует один из выводов, обозначенных как 0—7, к выводу Q, в зависимости от поданного на управляющие входы A—C двоичного кода. Очень важную функцию осуществляет вход E (с инверсией, т. е. активный уровень на нем — низкий) — это вход разрешения, если на нем присутствует высокий уровень, то все каналы размыкаются (недостающее 9-е состояние, подробнее см. *главу 19*).

Как видите, специально для коммутации переменных аналоговых сигналов у 561КП2 предусмотрено подключение отрицательного питания (выв. 7), в случае цифровых этот вывод просто соединяется с «землей». Размах питания между выводами 7 и 16 не может превышать предельно допустимого для однополярного питания 561-й серии значения 15 В, т. е. двуполярное питание возможно до ±7,5 В. Однако уровень сигнала управления (как по входам A—C, так и E) при этом отсчитывается от «цифровой земли», которая установлена потенциалом вывода 8. При этом аналоговый сигнал по амплитуде может достигать почти значений питания, однако для получения минимума искажений коммутируемые токи должны быть малыми.

Глава 9

Применение цифровых микросхем малой степени интеграции

> Перед тем, как съесть свой первый пейотный грибочек, Материалист поинтересовался у Поставщика (чернокожего джазиста): «А вообще-то эта дрянь опасна?».
>
> «Мать твою, — ответил он. — Индейцы тысячелетиями жрут ее в каждое полнолуние».
>
> *Роберт Антон Уилсон*
> *«Космический триггер»*

Из описания устройства логических элементов (см. *главу 8*) ясно, что любой логический вентиль есть, в сущности, не что иное, как усилитель. Только, в отличие от операционного усилителя, логический вентиль, во-первых, не имеет дифференциального входа, а во-вторых, обладает невысоким коэффициентом усиления по напряжению (порядка нескольких десятков для КМОП-элемента). Тем не менее не будет большой ошибкой представлять логический инвертор компаратором, у которого на неинвертирующий вход раз и навсегда подан определенный потенциал, примерно равный половине напряжения питания. И если ввести стабилизирующую обратную связь, которая выводит такой элемент в линейную область, то он вполне способен работать в аналоговом режиме.

Релаксационные схемы

Реально, конечно, аналоговые сигналы обрабатывать на логике не имеет никакого смысла, но это свойство логических вентилей широко используется на практике для построения т. н. *релаксационных* схем, продуцирующих самопроизвольные колебания, отличающиеся по форме от гармонических (прямоугольные,

импульсные, треугольные и т. д.). Такая схема характеризуется наличием одновременно положительной (ПОС) и отрицательной (ООС) обратных связей, причем теория гласит, что для получения устойчивых колебаний необходимо, чтобы действие ООС отставало от действия ПОС. Рассмотрим некоторые схемы такого рода.

Генераторы прямоугольных колебаний

Генератор прямоугольных колебаний называют еще *мультивибратором*. Существует много схем мультивибраторов, в том числе на цифровой логике (признаюсь, что мне даже невдомек, зачем в пособиях их обычно приводится так много, если они все равно делают в принципе одно и то же). Мы рассмотрим одну из них, выбранную с точки зрения минимального числа компонентов, и два ее варианта с управлением, разница между которыми заключается в используемых элементах.

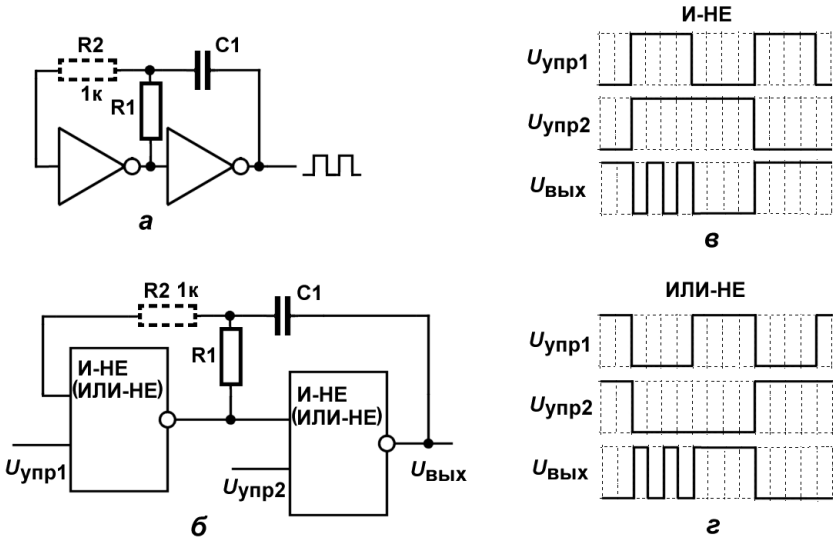


Рис. 9.1. Схемы мультивибратора на логических элементах:
а — базовая схема на инверторах; *б* — схема на двухвходовых элементах с управлением;
в — диаграмма состояний схемы на двухвходовых элементах «И-НЕ»;
г — диаграмма состояний схемы на двухвходовых элементах «ИЛИ-НЕ»

Схема по рис. 9.1, *а* — базовая. При включении питания она начинает работать сразу и выдает меандр с размахом от 0 до $U_{пит}$. Частота на выходе определяется параметрами R1 и C1: период колебаний $T \approx 2R1C1$. Резистор R2

в этом практически не участвует и нужен только для того, чтобы оградить защитные диоды микросхемы от перегрузки током разряда конденсатора С1. Величина его может изменяться от сотен ом до нескольких килоом. Величина же резистора R1 может изменяться от единиц килоом до 10 МОм, что позволяет избежать использования электролитических конденсаторов при малых частотах (напомним, что они очень нестабильны при работе во времязадающих цепях). Поэтому конденсатор С1 может применяться любой, с емкостью, начиная от нескольких десятков пикофарад, но только не электролитический. Практически указанные параметры элементов обеспечивают частоты от сотых долей герца до верхней границы рабочей частоты КМОП-микросхем в 1—2 МГц, а для быстродействующей КМОП-логики и выше, вплоть до 10 МГц и более.

Если в схеме рис. 9.1, *б* объединить входы логических элементов между собой, то она превратится в схему по рис. 9.1, *а* (чаще всего именно так базовую схему на практике и выполняют). Но нередко возникает задача остановить генерацию на время и при этом обеспечить совершенно определенный логический уровень на выходе генератора. Для этого предусматривают дополнительные входы. Диаграммы уровней на выходе в зависимости от состояния входов для разных типов логических элементов приведены на рис. 9.1, *в* и *г*.

Запоминать эти диаграммы нет необходимости, если обратиться к рис. 8.3, *а*. Из него следует, как описано в *главе 8*, что единица на входе «И-НЕ» и ноль на входе «ИЛИ-НЕ» являются *разрешающими* уровнями, следовательно, при этом наша схема будет функционировать как при объединении этих входов, т. е. подобно схеме на рис. 9.1, *а*. При запрещающих же уровнях на входе уровень на выходе будет устанавливаться так, как если бы никаких RC-цепочек не существовало.

Простейшие применения схемы с управлением — решение задачи приостановки генератора на время переходных процессов при включении питания, для чего к управляющему входу нужно подключить простейшую интегрирующую RC-цепочку. На рис. 9.2 показан другой вариант — схема звуковой сигнализации на микросхеме 561ЛА7 и одном транзисторе. Это пример случая, когда требуется определенный логический уровень при выключенной генерации, чтобы избежать протекания постоянного тока через динамик и не ставить при этом разделительный конденсатор.

Схема выдает сигнал около 500 Гц с периодом повторения около 0,5 с, если на управляющий вход подать сигнал высокого уровня. При низком уровне сигнала на этом входе, на выходе всей схемы также будет низкий уровень и постоянный ток через динамик не потечет. Транзисторный каскад лучше питать отдельным напряжением (например, нестабилизированным от входа

стабилизатора питания микросхемы), потому что тогда достаточно мощные импульсы тока через динамик будут фильтроваться стабилизатором и не окажут вредного воздействия на остальные элементы схемы. При питании цепи динамика и микросхемы от одного и того же источника лучше разделить их «развязывающим» RC-фильтром, как показано на рис. 9.2 пунктиром.

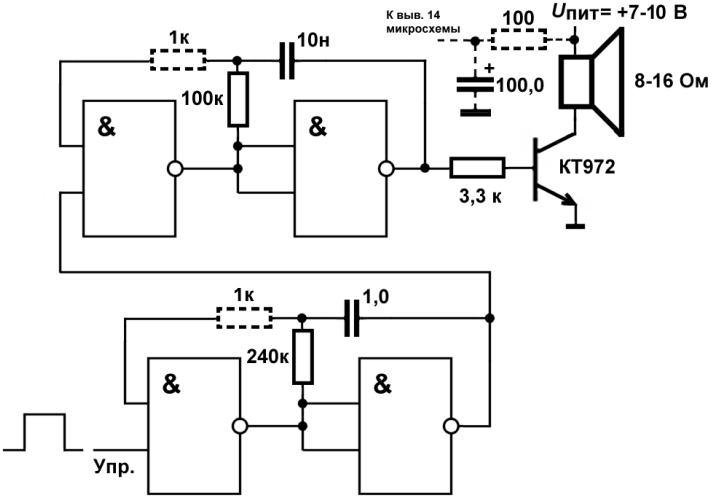


Рис. 9.2. Схема звуковой сигнализации с динамиком на выходе

Коллекторное напряжение насыщения транзистора КТ972 (это транзистор с «супербетой», см. *главу 3*) составляет около 1,5 В, поэтому при питании от источника 5 В звук может быть достаточно тихим. Вместо динамика можно поставить пьезоэлектрический звуковой излучатель, тогда подойдет маломощный транзистор с обычным коэффициентом передачи. А вот о пьезоэффекте мы сейчас подробнее и поговорим.

Кварцевые резонаторы

Точность поддержания частоты в схемах по рис. 9.1 невысока. Частота «уходит» примерно на 10—20% при изменении напряжения питания от 5 до 15 В и в достаточно большой степени зависит от температуры (высокостабильные резисторы и конденсаторы здесь не помогут и потому нецелесообразны). Чтобы избавиться от этого, необходимо использовать *кварцевый резонатор* (в просторечии — просто кварц). На кварцах работают все бытовые электронные часы, и вообще в любом современном бытовом электронном устройстве вы обязательно найдете кварц и иногда не один.

Подробности

Вкратце принцип работы кварца заключается в следующем: если приложить к кварцевому параллелепипеду, выпиленному из целого кристалла в определенной ориентации относительно его осей, напряжение, то кристалл деформируется (очень не намного, но все же достаточно, чтобы на этом принципе даже делать прецизионные манипуляторы, например, для электронных микроскопов). Это т. н. обратный пьезоэффект, имеет место и прямой — если такой кристалл деформировать, то у него на гранях появляется разность потенциалов. Получается, что если мы включим такой кристалл в схему с обратной связью, то она начнет генерировать, причем частота генерации будет зависеть исключительно от размеров кристалла — и ни от чего больше!

Как, спросите вы, даже от температуры не будет зависеть? Да от нее же зависит вообще все на свете — и геометрические размеры в первую очередь! Вот именно — пьезоэлектриков, как называют вещества, ведущие себя подобно кварцу, много, но используют именно кварц, так как он, помимо пьезоэлектрических свойств, обладает еще и одним из самых низких температурных коэффициентов расширения. В результате кварцевые генераторы без каких-либо дополнительных ухищрений обеспечивают нестабильность частоты порядка 10^{-5}, т. е. уход часов с таким генератором составляет не более 1 секунды в сутки. Именно распространение кварцевых генераторов привело к тому, что все измерения сейчас стараются свести к определению интервалов времени. Причем природа преподнесла здесь и еще один подарок: поскольку сам кварц является полным изолятором, то токов никаких через него не течет, и кварцевые генераторы в сочетании с КМОП-микросхемами почти не потребляют энергии.

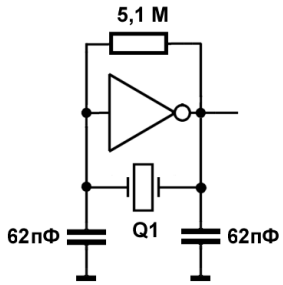


Рис. 9.3. Схема кварцевого генератора на КМОП-инверторе

Почти все кварцевые генераторы в микроэлектронной технике строят по одной и той же схеме, которая очень проста и требует всего одного инвертора, резистора и двух конденсаторов (рис. 9.3). Параметры элементов можно менять в довольно больших пределах — так, емкость конденсаторов может меняться от 20 до 200 пФ (причем они не обязательно должны быть одинаковыми), а сопротивление резистора — от 100 кОм до 10 МОм. Однако целесообразнее выбирать как можно меньшие емкости и как можно бóльшие сопротивления, иначе возрастает потребление от источника питания. Иногда для

дополнительного снижения потребления последовательно с кварцем со стороны выхода инвертора ставят еще один резистор в несколько сотен килоом. Естественно, инвертор при таких сопротивлениях может быть только КМОП-типа (ТТЛ-генераторы с кварцевым возбуждением строят по иным схемам). Частота кварца снизу практически не ограничена (для низких частот обычно употребляют т. н. часовой кварц с частотой 32 768 Гц), верхний же предел при использовании серии CD4000B ограничивается 1 МГц. Для более высоких частот потребуются быстродействующие КМОП-серии 74AC и 74HC (К1564). В качестве инвертора, естественно, пригоден и многовходовой логический элемент с объединенными входами.

Кварцы выпускают на определенные частоты, причем если нужна повышенная точность, то можно приобрести специализированные очень стабильные резонаторы с погрешностью до 10^{-7}, выпускаются и готовые генераторы на разные частоты (особенно большой выбор предлагает в этом отношении фирма, название которой обычно ассоциируется совсем с другими продуктами — Epson, приобретшая в свое время компанию, известную своей часовой торговой маркой Seiko). Установив вместо одного из постоянных конденсаторов подстроечный, частоту можно в очень небольших пределах (порядка 0,01% от номинала) менять, но сейчас этим почти никто не пользуется, т. к. подстройку лучше осуществить цифровым способом, или просто приобрести высокостабильный кварц.

Заметки на полях

Мало кто знает, что в случае если под рукой нет подходящего кварца, то схему на рис. 9.3 вполне можно «завести», просто заменив резонатор на малогабаритную индуктивность. Частоту можно грубо прикинуть, если учесть, что постоянная времени LC-контура равна $\sqrt{LC}$. Если в качестве величины *C* подставить сумму емкостей обоих конденсаторов, то частота будет примерно равна единице, деленной на удвоенную величину вычисленной постоянной времени. Естественно, главное преимущество кварца — высокая стабильность — при этом пропадет, зато могут резко снизиться габариты, т. к. кварцевые резонаторы далеко не всегда отличаются миниатюрностью, а серийно выпускаемые индуктивности обычно не крупнее малогабаритного резистора мощностью 0,125 Вт.

Формирователи импульсов

Большое значение на практике имеют формирователи коротких импульсов, называемые еще «схемами выделения фронтов». На рис. 9.4, *а* приведена схема, которая делает это, *как положено*. При поступлении положительного фронта на вход он сразу же переключает выход последнего элемента «И-НЕ» в состояние логического нуля. На выходе цепочки из трех инверторов также возникнет «0», который вернет выход в состояние «1», но это произойдет

не сразу, а спустя время, равное утроенной задержке срабатывания логических элементов. Поэтому на выходе возникнет короткая «иголка», длительность которой достаточна (задержка-то тройная!) для надежного срабатывания других элементов схемы. (Для КМОП длительность этого импульса составит несколько сотен наносекунд.) При желании можно выделить не фронт, а спад импульса (и получить при этом на выходе «иголку» положительной полярности), для этого потребуются элементы «ИЛИ-НЕ». А если использовать «Исключающее ИЛИ», то можно получать положительные импульсы при каждом переключении сигнала — и по фронту, и по спаду.

Замечание

В интуитивно понятном термине «фронт импульса» имеется некоторая неоднозначность, связанная с тем, что этим термином иногда обозначают только положительный перепад напряжения (т. е. переход из состояния «0» в «1»), чтобы отличить его от отрицательного (перехода из состояния «1» в «0»), который тогда называют «спадом импульса». В западной литературе соответствующие термины звучат, как «rising edge» и «falling edge» (буквально: «возрастающая кромка» и «падающая кромка»), что более соответствует смыслу.

Подобно тому, как термин «отрицательный перепад» отнюдь не означает наличия отрицательного напряжения относительно «земли», так и «полярность сигнала» в приложении к логическим уровням часто означает не полярность напряжения относительно той же «земли», а просто состояние логической единицы (положительный сигнал, высокий уровень) или логического нуля (отрицательный сигнал, низкий уровень).

Все здорово, но схема уж больно громоздкая для такой простой функции — целый корпус! На рис. 9.2 у нас был один корпус для какого сложного устройства, а тут — всего только выделение фронта. К тому же такие короткие импульсы очень сложно наблюдать на осциллографе. Поэтому на рис. 9.4, *б* и *в* приведены гораздо более экономичные схемы, которые делают то же самое, но *неправильно*. Почему неправильно? Потому что разработчики микросхем не рекомендуют использовать аналоговые узлы для построения цифровых схем. Вообще говоря, схемы генераторов (см. рис. 9.1) и одновибраторов (см. рис. 9.5) — тоже *неправильные*. Но они широко применяются, и нет причин для того, чтобы на тех же принципах не построить схемы выделения фронтов. Длительность импульса на выходе приведенных схем при указанных номиналах составит около 10 мкс.

Заметки на полях

В схемах генераторов на рис. 9.1 установлен дополнительный резистор (R2), ограничивающий ток через защитные диоды микросхемы. Дифференцирующая RC-цепочка, которая составляет основу этих схем, вырабатывает импульсы не только по нужному переключению сигнала, но и по противоположному, и при этом импульсы выходят за пределы питания, в чем вы можете убедиться, если взглянете на рис. 2.10. Здесь также применяется этот прием и потому в схемах

на рис. 9.4, *б* и *в* установлены необязательные ограничительные резисторы 1 кОм. Замечу, что во всех этих схемах (и в мультивибраторах, и в одновибраторах далее) можно обойтись и без токоограничивающих резисторов — как мы знаем, у диодов достаточно высокая перегрузочная способность, если только они не перегреваются. Обычно в мультивибраторах резистор ставят, т. к. они работают непрерывно, а в схемах выделения фронтов и одновибраторах, рассчитанных на периодическое срабатывание, опускают. В них такой резистор целесообразен лишь при больших выдержках времени, т. е. при низких частотах, когда емкость конденсатора времязадающей цепи велика.

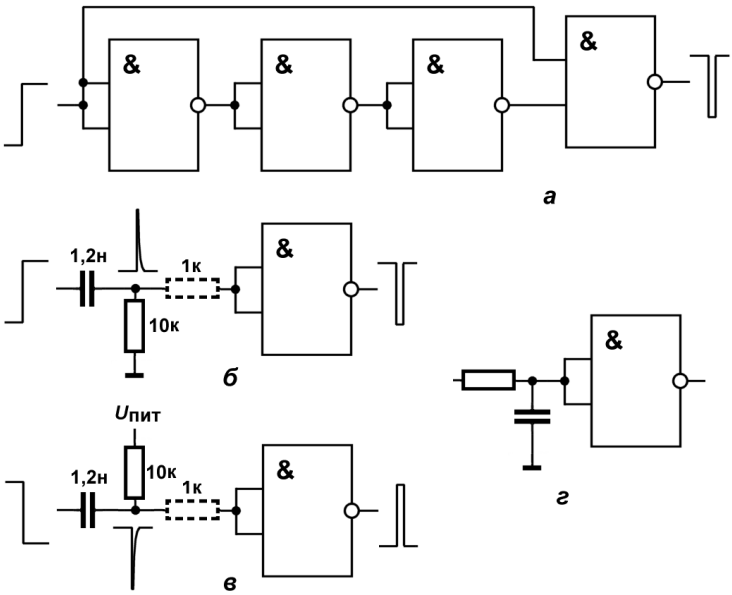


Рис. 9.4. Схемы формирователей импульсов:
а — стандартная схема формирователя коротких импульсов;
б, *в* — схемы с использованием дифференциальных RC-цепочек;
г — схема задержки

А на рис. 9.4, *г* показан пример интегрирующей цепочки для задержки импульса на фиксированное время. Диаграмм я не привожу, т. к. работа схемы понятна: импульс задерживается на время, необходимое для заряда конденсатора до порога срабатывания инвертора. Задний фронт импульса, соответственно, задерживается на время разряда. Если при этом входной импульс сравним по длительности с постоянной времени RC, то на выходе длительность импульса уменьшается, а при коротком входном импульсе выходной может вообще пропасть, поэтому такой схемой на практике пользуются очень редко, предпочитая ей чисто цифровые методы.

Одновибраторы

Одновибратор — это устройство, которое по внешнему сигналу выдает один-единственный импульс определенной длительности, не зависящей от длительности входного импульса. Запуск происходит либо по фронту, либо по спаду входного импульса и до возникновения на входе нового перепада напряжений той же полярности уровень на входе оказывать влияния на выход больше не будет. Как и в случае мультивибраторов, существует множество схемотехнических реализаций этого устройства. Мы изучим вариант, который получается небольшой модификацией схем выделения фронта, — нужно только ввести в них положительную обратную связь, которая будет фиксировать состояние выхода на время заряда конденсатора.

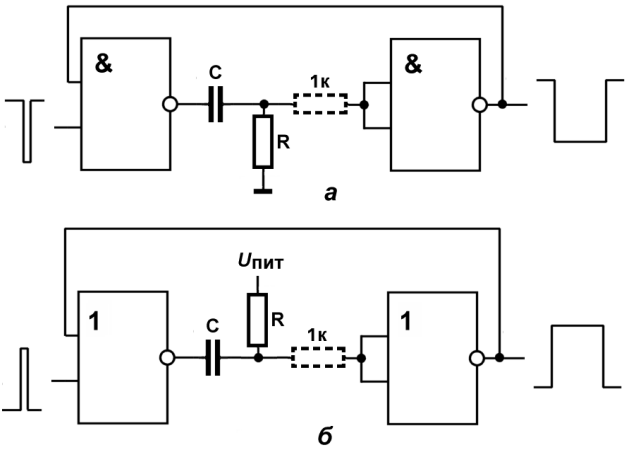


Рис. 9.5. Одновибраторы

Схема на рис. 9.5, *а* работает следующим образом: в состоянии покоя на выходе схемы имеется состояние логической единицы, т. к. вход второго (выходного) элемента «И-НЕ» заземлен через резистор. Так как на входе тоже логическая единица, то на выходе первого (входного) элемента «И-НЕ» — логический ноль, конденсатор разряжен. При поступлении на вход положительного уровня на выходе первого элемента типа «И-НЕ» возникает состояние логической единицы, которое через дифференцирующую цепочку RC передается на вход второго элемента, так что на выходе схемы и на втором входе первого элемента оказывается логический ноль. Это состояние схемы, уже независимо от уровня входного сигнала, будет устойчиво (обратная связь как бы «перехватила» и «зафиксировала» уровень нуля на выходе) — пока

конденсатор заряжается от выхода первого элемента через резистор R. Через время, примерно равное произведению *RC*, конденсатор зарядится до порога срабатывания выходного элемента «И-НЕ» и схема скачком перейдет обратно в состояние логической единицы по выходу.

Если по входу уже установлен уровень логической единицы (запускающий импульс закончился), то первый элемент также «перебросится» в начальное состояние и конденсатор С быстро разрядится через ограничительное сопротивление 1 кОм (если оно установлено) и входные защитные диоды второго элемента. Схема придет в начальное состояние в ожидании следующего запускающего импульса. Длительность импульса на выходе всегда будет примерно равна *RC*, даже в случае, если входной импульс длиннее (в этом случае конденсатор просто разрядится не сразу, а только тогда, когда закончится входной импульс). Совершенно аналогично работает схема на рис. 9.5, *б*, только с противоположными полярностями импульсов.

Главное применение одновибраторов — в качестве таймера, который формирует сигнал определенной длительности. Естественно, о высокой точности выдержки времени тут говорить не приходится, но часто это и не требуется. Например, если вы хотите ограничить по времени тревожный сигнал, подающийся с помощью устройства по рис. 9.2, то целесообразно управлять им от одновибратора, который запускается, скажем, нажатием кнопки. В одновибраторах для больших выдержек допустимы электролитические конденсаторы, хотя даже при использовании только керамических или полимерных типов с максимальными емкостями порядка 1—3 мкФ вполне достижимы выдержки в несколько десятков секунд.

Рассмотренные одновибраторы представляют собой схемы *без перезапуска*, т. е. длительность импульса не зависит от того, пришел ли еще раз входной импульс во время действия выходного или нет. Одновибраторы *с перезапуском*, в которых выходной импульс в этом случае продлевается (отсчет времени как бы начинается заново с нового импульса, когда бы он ни пришел), мы проектировать не будем, потому что они сложнее, и в этом случае предпочтительнее готовая микросхема (например, 561АГ1). Создать одновибратор (мультивибратор, кварцевый генератор) можно и на специальной микросхеме универсального таймера, известной под названием 555 с различными буквенными индексами (отечественный аналог —1006ВИ1).

Одновибратор может служить довольно эффективным средством подавления дребезга механических контактов, т. к. будет запускаться только от первого перепада уровней, причем независимо от того, «пролетают» подвижные контакты весь промежуток «туда-обратно» или нет (впрочем, на практике такого «пролета» и не случается). Главным его преимуществом в этом качестве,

несмотря на довольно сложную схему, является пригодность двухвыводной кнопки, а не переключающей, как в схемах по рис. 8.3, *б* или на RS-триггерах (см. далее). Вход одновибратора при этом соединяют с питанием (в схеме рис. 9.5, *а*) или с «землей» (на рис. 9.5, *б*) через резистор, а кнопкой замыкают этот вход, соответственно, на «землю» или на питание (пример подсоединения см. на рис. 17.1).

Одним из недостатков такого варианта является то, что приходится четко рассчитывать необходимую длительность импульса, иначе дребезг можно «пропустить». Второй недостаток — неясность ситуации с размыканием ранее замкнутой кнопки. Если кнопка удерживается в замкнутом состоянии дольше, чем длится импульс, то из-за дребезга при размыкании одновибратор может выдать импульс повторно. Для борьбы с этим явлением можно попробовать присоединить кнопку ко входу одновибратора не напрямую, а через одну из дифференцирующих цепочек по рис. 9.4, *б* или *в*, которые (в идеале) не должны пропускать через себя состояние вывода, «висящего в воздухе».

Заметки на полях

Укажем, кстати, на недопустимость использования часто рекомендуемых и заманчивых по своей простоте схем «антидребезга» на основе интегрирующей цепочки, т. е. элементарного ФНЧ. Опыт показывает, что такие схемы крайне ненадежны даже при совместно с т. н. *триггером Шмидта*, который представляет собой обычный логический элемент с гистерезисной характеристикой. Гистерезис в случае логических элементов чаще всего не защищает от дребезга вообще, т. к. помеха располагается обычно вблизи питания или «земли», за пределами зоны нечувствительности. А что касается ФНЧ, то даже если вы умудритесь подобрать параметры фильтра так, что данная конкретная кнопка, как вам кажется, не дребезжит, то это не гарантирует, что в случае очень короткого или, наоборот, долгого нажатия схема сработает как надо, или что другая аналогичная кнопка будет также нормально работать с теми же параметрами RC-фильтра.

Триггеры

Триггер — это устройство для записи и хранения информации в количестве одного бита[1]. (Существуют — по большей части в теории — и многостабильные триггеры, которые могут хранить более одного бита, но на практике они не используются, кроме очень экзотических конструкций, вроде упоминавшейся в *главе 7* ЭВМ «Сетунь»). Любая элементарная ячейка памяти, будь-то магнитный домен на пластинах жесткого диска, отражающая область

[1] Одно из главных «неэлектронных» значений слова «trigger» — спусковой крючок у огнестрельного оружия.

на поверхности CD-ROM или конденсаторная ячейка электронного ОЗУ, обязательно обладает триггерными свойствами, т. е. может хранить информацию спустя еще долгое время после того, как она была в нее введена.

Самый простой триггер можно получить, если в схемах одновибраторов на рис. 9.5 удалить RC-цепочку и соединить напрямую выход первого элемента со входом второго. Если схема находится в состоянии, когда на выходе уровень логической единицы, то кратковременная подача отрицательного уровня на вход, как и в случае одновибратора, перебросит выход в состояние логического нуля, но теперь уже нет конденсатора, который осуществляет отрицательную обратную связь и в конце концов возвращает схему в исходное состояние, потому в этом состоянии схема останется навечно, если мы что-то не предпримем.

Чтобы вернуть ее в исходное состояние, надо подать точно такой же сигнал, но на вход второго элемента, который (вход) в схеме одновибратора у нас отсутствует. Если мы его введем, то получим симметричную схему с двумя входами, которые обозначаются буквами R и S (от слов Reset и Set, т. е. «сброс» и «установка»). Такое устройство носит название *RS-триггера*. Оба варианта такой схемы на элементах «И-НЕ» и «ИЛИ-НЕ» показаны на рис. 9.6. Легко сообразить, что если поменять все обозначения местами (R на S, а прямой выход на инверсный), то в схеме ничего не изменится, но не все триггерные схемы обладают подобной симметрией.

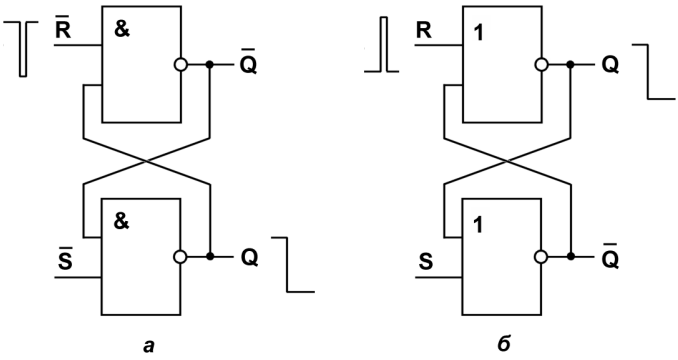


Рис. 9.6. Схемы триггеров на элементах «И-НЕ» (*а*) и «ИЛИ-НЕ» (*б*)

Нет нужды перебирать все состояния этих схем и приводить соответствующие таблицы истинности, нужно только твердо запомнить, что подача импульса на вход сброса R *всегда* устанавливает на выходе Q состояние логического нуля (естественно, на инверсном выходе $\overline{Q}$ при этом будет логическая

единица). Причем соответствующий вход у любого устройства, его имеющего — от микропроцессоров до счетчиков — является *асинхронным*, т. е. вся система обнуляется в момент подачи импульса по входу R независимо от того, что в этот момент она делает (говорят еще, что вход сброса имеет «наивысший приоритет»). Именно это происходит, скажем, когда вы нажимаете на кнопку Reset на системном блоке вашего компьютера.

Вход S, естественно, означает ровно противоположное — установку выхода Q в состояние логической единицы, но, в отличие от входа R, который всегда означает обнуление, вход S в различных устройствах может использоваться и в немного других целях, а чаще вообще отсутствует. Входы R и S могут управляться различными полярностями сигнала в зависимости от построения триггера — для схемы на элементах «И-НЕ» по рис. 9.6, *а* это низкий уровень, потому входы R и S обозначены с инверсией, согласно положительной логике (уровни, которые меняют состояние триггера, называются *активными*, так, для схемы по рис. 9.6, *а* активным является низкий уровень). «Более правильная» схема в этом смысле — на элементах «ИЛИ-НЕ» по рис. 9.6, *б*, где активный уровень — высокий.

В схемах RS-триггеров подача активного уровня на R-вход ничего не меняет, если выход Q уже был в состоянии логического нуля, то же самое справедливо для S-входа при выходе Q в состоянии логической единицы. Однако пока на соответствующем входе действует напряжение активного уровня, подача активного уровня на второй вход запрещена. Это не означает, что триггер при этом сгорит, просто он потеряет свои триггерные свойства — на обоих выходах установится один и тот же уровень, а после одновременного снятия активного уровня со входов состояние будет неопределенным (точнее, будет определяться тем элементом, который переключится чуть позже другого).

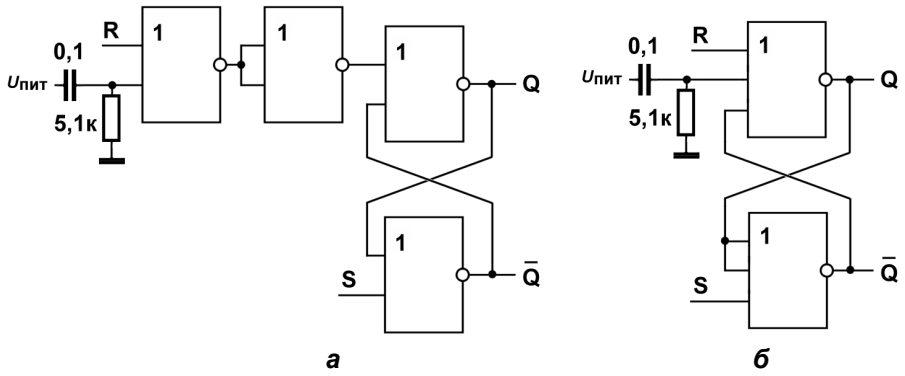


Рис. 9.7. Схемы триггеров с предустановкой при включении питания

Неопределенное состояние будет и после подачи питания, поэтому следует принимать специальные меры для установки схемы в нужное состояние после включения. Наиболее распространенной такой мерой является подача определенного уровня в начальный момент времени на один из требуемых входов с помощью RC-цепочки. Ввиду практической важности этого способа я приведу вариант соответствующей схемы, несмотря на ее очевидность (рис. 9.7, *а*). Лишние элементы необходимы для того, чтобы сохранить возможность произвольного сброса по отдельному R-входу, хотя на практике часто входы внешнего сброса и сброса по питанию объединены.

В этой схеме конденсатор в первый момент времени после подачи питания разряжен и на входе логического элемента оказывается положительный уровень, который устанавливает триггер в состояние «0» на выходе Q. Затем конденсатор заряжается и в дальнейшем RC-цепочка больше не оказывает влияния на работу схемы. Постоянную времени RC лучше выбирать побольше, чтобы к моменту зарядки конденсатора успели пройти все переходные процессы, на схемах по рис. 9.7 она равна примерно 0,5 мс. Естественно, при этом следует позаботиться, чтобы на «настоящих» RS-входах к моменту окончания заряда конденсатора был неактивный уровень, иначе все пойдет насмарку. Чтобы избежать нагромождения элементов, в этой схеме предпочтительнее использовать трехвходовые элементы (561ЛЕ10), как показано на рис. 9.7, *б*.

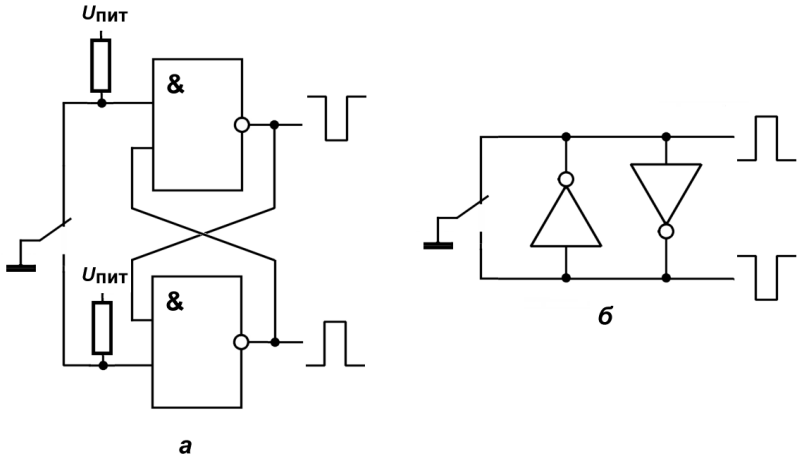


Рис. 9.8. Схемы «антидребезга» на RS-триггерах

Естественно, RS-триггеры выпускают и в интегральном исполнении (561ТР2 содержит четыре простых RS-триггера). Все более сложные триггеры, а также

счетчики в интегральном исполнении обязательно имеют отдельные R-, S- или хотя бы только R-асинхронные входы.

Использование RS-триггера является самым «капитальным» способом решения проблемы дребезга контактов. Стандартная схема включения показана на рис. 9.8, *а*, однако нет никакой нужды «городить» такую схему с резисторами, относительно которых еще нужно соображать, к чему их подключать (для варианта с «ИЛИ-НЕ» их пришлось бы присоединять к «земле»). На рис. 9.8, *б* показана упрощенная схема, которая работает точно так же и при этом в ней пригодны любые инверторы, в том числе и одновходовые.

Общий недостаток схем антидребезга как на RS-триггерах, так и на элементе «Исключающее ИЛИ» (см. рис 8.3, *б*) — необходимость переключающей кнопки с тремя выводами, которых на рынке предлагается гораздо меньше, чем обычных замыкающих и размыкающих с двумя контактами. Попробуйте приспособить двухвыводную кнопку к любой из указанных схем и вы сами придете к выводу, что это невозможно. Поэтому на практике часто приходится прибегать к схеме на одновибраторе (в том числе реализованной программными способами в микроконтроллерах), несмотря на все ее недостатки.

D-триггеры

D-триггеры получили свое название от слова «delay», что означает «задержка». На самом деле их существует две разновидности, формально различающиеся только тем, что первая (*статический* D-триггер или *триггер-защелка*) управляется уровнем сигнала, как и все схемы, рассмотренные ранее, а вторая (*динамический* D-триггер) управляется фронтом импульса. Фактически же это разные по устройству и области применения схемы, потому объединение их под одним названием представляется не совсем удачным. Так, микросхема 561ТМ3 содержит четыре статических триггера-защелки, а ТМ2 — два динамических D-триггера с дополнительными входами R и S. Если тип не указывается, то обычно по умолчанию предполагается, что речь идет о динамических D-триггерах.

Статический D-триггер легко получить из RS-триггера путем небольшого усложнения его схемы (рис. 9.9, *а*). При наличии на входе C уровня логической единицы входные сигналы будут пропускаться на вход RS-триггера и схема будет повторять на выходе Q уровни на входе D. Если же мы установим на входе C уровень логического нуля, то схема немедленно «зависнет» в состоянии выхода, соответствующем входному уровню непосредственно перед приходом отрицательного фронта на вход C — то есть запомнит его! Поэтому такой триггер и называют *защелкой* — при подаче на вход C короткого тактового импульса он как бы «защелкивает» состояние входа.

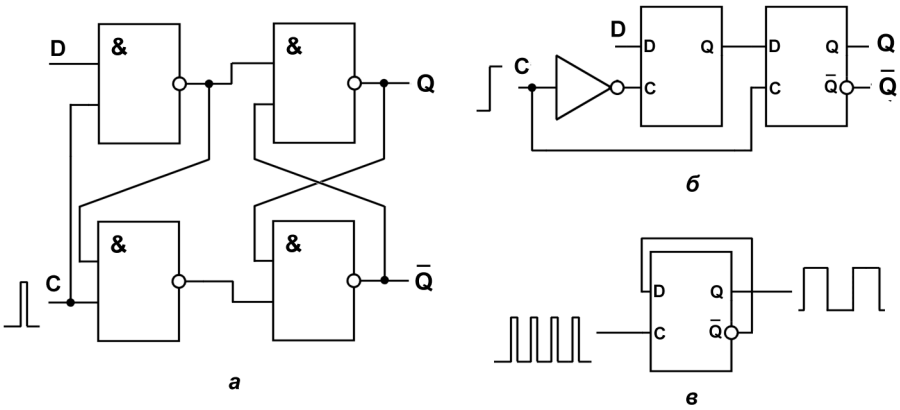


Рис. 9.9. D-триггеры: *а* — схема статического D-триггера; *б* — схема динамического D-триггера на основе двух статических; *в* — счетный триггер на основе динамического D-триггера

Динамические D-триггеры более универсальны и область применения у них куда шире, чем у статических. Динамический триггер сложнее по устройству. Один из способов построения динамического D-триггера из двух статических показан на рис. 9.9, *б*. Эта схема работает следующим образом: когда на общем входе C присутствует отрицательный уровень, состояние входа D переписывается на выход первого (слева) триггера, при этом второй триггер заперт. Сразу после положительного фронта на входе C это состояние переписывается во второй триггер и появляется на выходе Q, а первый триггер запирается. Таким образом, запоминание состояния общего D-входа происходит в точности в момент положительного перепада уровней и никогда больше. Если изменить местоположение инвертора и присоединить его ко входу второго триггера, а на первый триггер подавать тактовые импульсы напрямую, то срабатывание будет происходить по отрицательному фронту и такой тактовый вход будет считаться инверсным. Для того чтобы получить дополнительные входы принудительной установки триггера в нулевое и единичное состояния (R- и S-входы), нужно *оба* статических триггера реализовать на трехвходовых элементах и объединить соответствующие входы у обоих триггеров — устанавливать по входам R и S только выходной триггер недостаточно (подумайте, почему?).

Счетный триггер

На рис. 9.9, *в* показана самая простая схема *счетного* триггера на основе динамического D-триггера. Из сказанного ясно, как она работает: при каждом положительном перепаде на выход Q будет переписываться состояние про-

тивоположного выхода $\overline{Q}$, т. е. с приходом каждого тактового импульса система изменяет свое состояние на противоположное, в результате чего на выходе сформируется симметричный (независимо от скважности входных импульсов) меандр с частотой, вдвое меньшей, чем входная. Такой триггер можно считать делителем частоты на два или одноразрядным двоичным счетчиком — в зависимости от того, для чего он используется. В отличие от всех остальных типов триггеров (а кроме описанных, распространены еще и т. н. JK-триггеры, на которых мы здесь не останавливаемся), счетные триггеры в интегральном исполнении отдельно не выпускают (их легко получить, например, из D-триггеров), а изготавливают только готовые многоразрядные двоичные счетчики, из таких триггеров составленные. К рассмотрению счетчиков мы перейдем чуть далее, а пока кратко остановимся на регистрах.

Регистры

Регистрами называют устройства для хранения двоичных чисел. Количество разрядов в регистрах, выпускаемых отдельно, обычно не превышает восьми, но в составе других микросхем могут быть и регистры с большей разрядностью — вплоть до 128 бит в процессорах типа Pentium или Athlon.

Простейший одноразрядный регистр — это описанный в предыдущем разделе статический D-триггер. Большинство регистров в микроконтроллерах, а также ячеек статической памяти (SRAM) представляют собой именно такие триггеры-защелки. Вообще большинство типов электронных ЗУ, за исключением таких устройств, как магнитные или оптические диски, можно рассматривать как совокупность регистров. Например, четыре триггера-защелки, входящие в микросхему 561ТМ3, образуют четырехразрядный регистр с параллельной записью и считыванием, причем тактовый вход в этой микросхеме у всех четырех разрядов общий. Как и сам триггер, такой регистр называют защелкой.

Значительно чаще регистрами называют устройства, которые позволят записывать и считывать информацию не только раздельно в каждый разряд, но и последовательно, с помощью сдвига. Если регистр-защелка допускает только параллельную запись, то последовательный регистр имеет возможность записи через единственный вход, который является D-входом самого младшего разряда.

Последовательный регистр является неким обобщением конструкции D-триггера. Работу динамического D-триггера можно рассматривать, как процесс сдвига информации от входа через первый триггер ко второму при поступлении соответствующих перепадов на тактовом входе. В последовательном регистре, который в простейшем случае представляет собой просто

соединение таких триггеров друг за другом, происходит нечто подобное: с каждым фронтом тактового импульса информация сдвигается от младшего разряда к старшему, при этом в младший разряд записывается состояние входа. Считывать (и записывать) информацию при этом обычно можно и из каждого разряда в отдельности, как и в случае регистра-защелки, и через единый последовательный вход и выход. Такие регистры получили еще название сдвиговых (пример — 561ИР2). Они широко используются для последовательного ввода и вывода информации. Скажем, для вывода восьми бит через последовательный порт RS-232 достаточно записать их в такой регистр, а потом подать на него восемь тактовых импульсов с нужной частотой (см. *главу 16*).

Счетчики

Самый простой счетчик можно получить, если соединить последовательно ряд счетных триггеров, как показано на рис. 9.10, *а*. Схема обладает одной особенностью, в которой легко разобраться, если построить диаграмму работы этого счетчика, начиная с состояния, в котором все триггеры находятся в состоянии низкого уровня на выходе («0000»). В самом деле, при подаче первого же импульса триггеры перейдут в состояние со всеми единицами («1111»)! Если строить диаграмму дальше, то мы увидим, что последовательные состояния будут такими: «1110», «1101» и т. д. В этом легко узнать последовательный ряд чисел 15, 14, 13, т. е. счетчик получился вычитающим, а не суммирующим.

А как можно получить суммирующий счетчик? Очень просто — надо ко входу каждого следующего триггера подсоединить не прямой выход предыдущего, а инверсный. Если при этом тактовые импульсы подавать также через инвертор (рис. 9.10, *б*), тогда счетчик будет срабатывать по заднему (отрицательному) фронту входного импульса, а не по переднему (разумеется, можно просто выбрать триггеры с инверсным тактовым входом). В этом случае будет все в порядке — входные импульсы будут суммироваться (см. диаграмму) и мы получим ряд последовательных состояний: «0000», «0001», «00010», «0011» и т. д.

Заметки на полях

Удивительная все же штука — электроника! Сначала мы получили полную аналогию между абстрактной математической теорией — булевой алгеброй, — и состояниями переключателей на реле, теперь вот — между не менее абстрактным арифметическим счетом и последовательными состояниями счетчика на триггерах. Чем этот счетчик отличается от дикаря, раскладывающего на земле палочки? Ничем, кроме того, что он «раскладывает» не палочки, а уровни напряжений, причем выгодно отличается от первобытного сознания тем, что еще

и владеет позиционной системой счисления. Начинаешь понимать, почему ученые в середине прошлого века были так обольщены возможностями электронных схем, что даже заговорили о «машинном разуме». Но это уже другая тема…

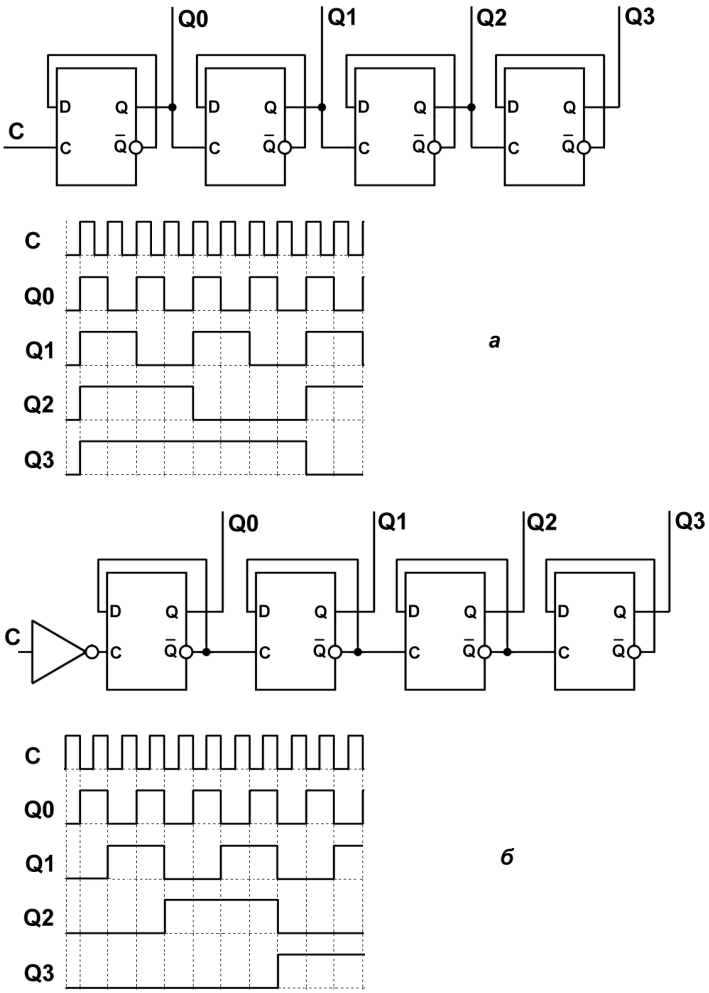


Рис. 9.10. Схемы счетчиков на D-триггерах: *а* — вычитающего; *б* — суммирующего

У счетчиков, построенных по такой простейшей схеме, есть один крупный недостаток: переключение триггеров происходит асинхронно, сигнал от входа должен пройти всю цепочку, пока на выходе также изменится уровень. Эти, казалось бы, незначительные задержки могут, однако, привести к значительным неприятностям, типа возникновения лишних «иголок» при дешиф-

рировании состояний выхода. А при больших частотах входных импульсов, на пределе возможностей конкретных логических элементов, фронты сигналов на выходах вообще могут приобрести совершенно хаотическое расположение относительно входного сигнала, так, что дешифрировать состояние счетчика будет невозможно. Поэтому практически все счетчики в интегральном исполнении делают по иным, синхронным, схемам, когда входной тактовый сигнал подается одновременно на все разряды, и фронты выстраиваются строго «по линеечке», независимо от задержек в том или ином триггере. Так устроены, например, два четырехразрядных счетчика, образующие микросхему 561ИЕ10.

Наиболее универсальные счетчики позволяют записывать информацию параллельно, как в регистрах. Тогда счетчик может начинать отсчет не с нулевого значения, а с некоего заданного числа. Таковы, например, счетчики 561ИЕ11 и 561ИЕ14. Подробно разбирать мы такие схемы не будем, т. к. самостоятельно их строить не придется, но для понимания того, как устроены счетчики-таймеры в микроконтроллерах, эта информация пригодится.

На практике счетчики используют не только по прямому назначению — для подсчета импульсов, — но и в качестве управляемых делителей частоты. На этом основано их применение в электронных часах. Обычный часовой кварц по технологическим причинам удобно делать на частоту 32 768 Гц. Пропустив частоту с генератора, построенного на таком кварце, через 16-разрядный счетчик-делитель (например, 561ИЕ16), мы получим на выходе колебания с периодом ровно в 1 с, которые удобны для дальнейшего формирования минут и часов. На практике из-за сложности суточного счета времени и особенно календарных дат, от дискретных счетчиков для таких целей давно отказались, и часы делают на специализированных микросхемах RTC (Real Time Clock — «часы реального времени») и микроконтроллерах, и тем, и другим мы еще будем заниматься. Но в основе работы таких интегральных часов все равно лежат счетчики-делители частоты — аппаратные или программные.

Глава 10

Откуда берутся цифры

Люди старый аналоговый телевизор на новый заменить не могут, а вы говорите про более дорогие цифровые...

«Время новостей», 08 июня 2004

Все природные явления носят непрерывный, аналоговый характер. По крайней мере, для нас все протекает так, как если бы явления природы были полностью непрерывными и характеризовались бы рядом действительных чисел, отстоящих друг от друга на бесконечно малые отрезки по числовой оси. Если же копнуть поглубже, то окажется, что все не так просто. Начнем с атомно-молекулярной структуры вещества и всей огромной совокупности явлений, которые являются следствием этого феномена. Открытие этой структуры в свое время немало потрясло ученых. Но если даже не вдаваться в атомные материи, то и на макроуровне тоже все не так однозначно, например, наш глаз по сути представляет собой светочувствительную матрицу, в которой около 125 миллионов светочувствительных палочек и около 6 миллионов цветочувствительных колбочек.

На практике, однако, эти рассуждения имеют мало значения, т. к. разрешение глаза на порядок превышает возможности искусственно созданной матрицы, да и сама дискретность там иной природы. Для большинства практических применений и звуковые и световые колебания можно считать имеющими чисто аналоговую, непрерывную природу. Но вот обрабатывать их, и особенно хранить, оказалось куда удобнее в цифровом виде.

Встает задача преобразования аналоговой величины в дискретную. Естественно, когда мы хотим, чтобы преобразованная информация опять предстала перед нами в форме, воспринимаемой нашими органами чувств, то мы вынуждены делать и обратное преобразование — цифроаналоговое. Правда, такое требуется не всегда: во многих случаях информацию можно оставить

в цифровом виде, так ее и отобразив — в виде совокупности цифр на семисегментном индикаторе, к примеру.

Заметки на полях

Интересно, что такой способ отображения, хотя и значительно более *корректный*, чем аналоговый (мы не теряем информации), но не всегда может оказаться более *правильным*. Если вы вглядитесь в пульт управления каким-нибудь сложным устройством — не обязательно атомной электростанцией, достаточно торпеды обычного автомобиля, — вы увидите, что большинство показывающих приборов там — стрелочные. Хотя, как вы понимаете, нет никаких проблем в современном автомобиле демонстрировать скорость, уровень топлива или температуру двигателя непосредственно в цифрах, но этого не делают сознательно, потому что в очень многих случаях человека не интересует точное значение того или иного параметра. Его интересует только отклонение от некоторого значения, или превышение некоторого порога, или тенденция изменения величины, но не сама эта величина, и не сам порог. Информация о том, что температура двигателя составляет 80 °С, для водителя совершенно лишняя, ему важно знать, что если вот эта стрелочка не достигла вот этой красненькой черточки — значит, все в порядке. Но бывают и другие случаи, например, отсчет пробега того же автомобиля имеет смысл, только будучи представленным именно в цифровом виде, поэтому еще на заре автомобилестроения пришлось придумывать разные — тогда еще, конечно, механические — счетчики, отображающие *число* пройденных километров. Все это следует учитывать при проектировании различных показывающих устройств, и при необходимости приходится даже идти на усложнение схемы, причем, что обидно, нередко с заведомой потерей информации или даже с ее искажением. Типичный пример из этой области — датчик количества топлива в том же автомобиле, который проектировщики традиционно заставляют врать, занижая показания, иначе слишком много водителей оказывалось бы на дороге с сухими баками в полукилометре от ближайшей заправочной станции.

Оцифровка

Основной принцип оцифровки любых сигналов очень прост и показан на рис. 10.1, *а*. В некоторые моменты времени t_1, t_2, t_3 мы берем мгновенное значение аналогового сигнала и как бы прикладываем к нему некоторую меру, линейку, проградуированную в двоичном масштабе. Обычная линейка у нас содержит крупные деления (метры), поделенные каждое на десять частей (дециметры), каждая из которых также поделена на десять частей (сантиметры), и т. д. Двоичная линейка содержала бы деления, поделенные пополам, затем еще раз пополам и т. д. — сколько хватит разрешающей способности. Если вся длина такой линейки составляет, допустим, 2,56 метра, а самое мелкое деление 1 см (т. е. мы можем померить ей длину с точностью не хуже 1 см, точнее, даже половины его), то таких делений будет ровно 256 и их можно представить двоичным числом размером 1 байт или 8 двоичных разрядов.

Ничего не изменится, если мы меряем не длину, а напряжение или сопротивление, только смысл понятия «линейка» будет несколько иной. Так мы получаем последовательные отсчеты величины сигнала x_1, x_2, x_3. Причем заметьте, что при выбранной разрешающей способности и числе разрядов мы можем померить аналоговую величину не больше некоторого значения, которое соответствует выбранному масштабу. Иначе придется или увеличивать число разрядов (длину линейки), или менять разрешающую способность в сторону ухудшения (растягивать линейку). Все изложенное и есть сущность работы аналого-цифрового преобразователя (АЦП).

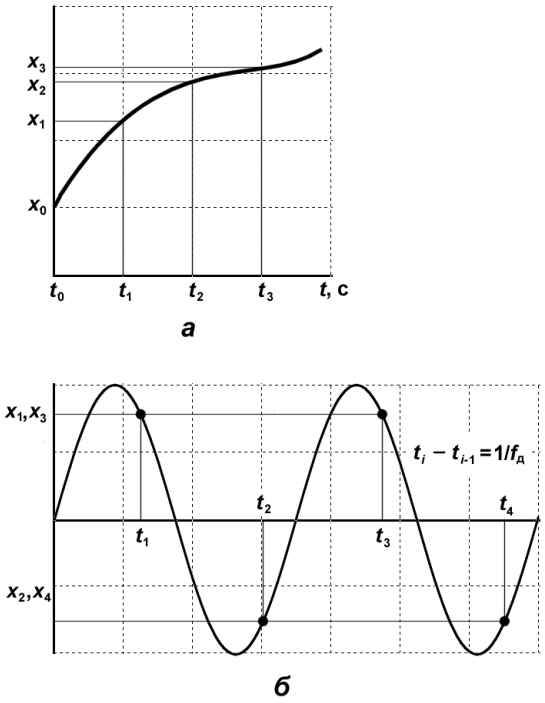


Рис. 10.1. Оцифровка аналоговых сигналов:
а — основной принцип; *б* — к теореме Котельникова—Найквиста

На рис. 10.1, *а* график демонстрирует этот процесс во времени. Если мы меряем какую-то меняющуюся во времени величину, то приходится производить измерения регулярно. Если стоит задача потом восстановить первоначальный сигнал, то эти измерения удобно проводить со строго равными промежутками времени между ними — иначе нам будет трудно узнать, какому измерению какой момент сигнала соответствует. Получаем массив чисел,

который и представляет наш исходный сигнал в цифровом виде. Зная частоту дискретизации (частоту оцифровки) и принятый масштаб (т. е. какому значению физической величины соответствует максимальное число в принятом диапазоне двоичных чисел), мы всегда можем восстановить исходный сигнал, просто отложив точки на графике и соединив их плавной линией.

Но что-то мы при этом теряем? Посмотрите на рис. 10.1, *б*, который иллюстрирует знаменитую теорему Котельникова (как водится, за рубежом она носит другое имя — Найквиста, на самом деле они оба придумали ее независимо друг от друга). На этом рисунке показана синусоида предельной частоты, которую мы еще можем восстановить, располагая массивом точек, полученных с частотой дискретизации $f_д$. Так как в выражении для синуса $A\sin(2\pi ft)$ имеется два независимых коэффициента (A — амплитуда и f — частота), то для того, чтобы вид кривой восстановить однозначно, нужно как минимум две точки на каждый период (если сами параметры синусоиды A и f не меняются во времени, то достаточно вообще двух точек на всем интервале времени; именно такой случай показан на графике рис. 10.1, *б*), т. е. *частота оцифровки должна быть как минимум в два раза больше, чем самая высокая частота в спектре исходного аналогового сигнала*. Это и есть теорема Котельникова—Найквиста.

Попробуйте сами нарисовать другую синусоиду без сдвига по фазе, проходящую через указанные на графике точки, и вы убедитесь, что это невозможно. В то же время можно нарисовать сколько угодно разных синусоид, проходящих через эти точки, если их частота в целое число раз выше частоты дискретизации $f_д$. В сумме эти синусоиды, или гармоники (т. е. члены разложения сигнала в ряд Фурье), дадут сигнал любой сложной формы, но восстановить их нельзя, и если такие гармоники присутствуют в исходном сигнале, то они пропадут навсегда. Следовательно, процесс оцифровки *равносилен действию ФНЧ с прямоугольным срезом характеристики на частоте, равной ровно половине частоты дискретизации*.

Займемся обратным преобразованием. В сущности, никакого преобразования цифра — аналог в цифроаналоговых преобразователях (ЦАП), которые мы будем здесь рассматривать, на самом деле не происходит: просто мы выражаем двоичное число в виде пропорциональной величины напряжения, т. е. занимаемся, с точки зрения теории, всего лишь преобразованием масштабов и физическим моделированием абстрактной величины — числа. Вся аналоговая шкала поделена на кванты — градации, соответствующие разрешающей способности нашей двоичной «линейки». Если максимальное значение сигнала равно, к примеру, 2,56 В, то при восьмиразрядном коде мы получим квант в 10 мВ, и что происходит с сигналом между этими значениями, и в промежутки времени между отсчетами, мы не знаем и узнать не можем.

Если взять ряд последовательных отсчетов некоего сигнала, (например, как на рис. 10.1, *а*), то мы в результате получим ступенчатую картину (рис. 10.2).

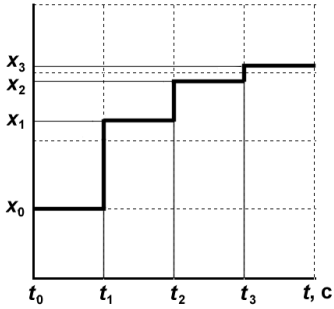


Рис. 10.2. Восстановление оцифрованного сигнала с рис. 10.1, *а*

Если вы сравните графики на рис. 10.1, *а* и 10.2, то увидите, что второй график представляет первый, мягко говоря, весьма приблизительно. Для того чтобы повысить степень достоверности полученной кривой, следует, во-первых, брать отсчеты почаще, во-вторых, увеличивать разрядность. Тогда ступеньки будут все становиться меньше и меньше, и есть надежда, что при некотором достаточно высоком разрешении, как по времени, так и по уровню, кривая станет, в конце концов, неотличима от непрерывной аналоговой линии. Есть и еще один способ получения гладкой кривой — пропустить полученный сигнал через ФНЧ, в результате чего ступеньки сгладятся. (Практически это равносильно вычислению промежуточных значений методом интерполяции, считая, что от отсчета к отсчету кривая меняется по линейному или какому-нибудь еще закону.) Конечно, ФНЧ — это лишь грубая полумера, и увеличения разрядности и частоты отсчетов не заменяет.

Все изложенное касается дискретизации аналоговых сигналов во времени. Но здесь нас будет больше занимать не временной ряд оцифрованных сигналов, а получение каждого отдельного значения этого ряда — как же реализовать на практике упомянутую ранее двоичную линейку?

ЦАП

Начнем мы с конца, т. е. с цифроаналоговых преобразователей. Будем считать, что на входе мы имеем числа в двоичной форме — неважно, результат оцифровки сигнала или синтезированный код. Нам его нужно преобразовать в аналоговый уровень напряжения в соответствии с выбранным масштабом.

Самый простой ЦАП — десятичный или шестнадцатеричный дешифратор-распределитель, подобный 561ИД1 (см. рис. 8.7). В самом деле, если на него подать четырехразрядный код, то на выходе мы получим значения в десятичной или шестнадцатеричной форме — для каждого значения кода на отдельном выводе. Присоединив к выходам этого дешифратора линейку светодиодов, получаем полосковый (шкальный) индикатор, который с разрешением в 10 или 16 ступеней на весь диапазон будет показывать уровень некоей величины. Иногда этого достаточно.

На самом деле это, конечно, еще не настоящий ЦАП, а только его часть — он не делает операции, показанной на рис. 10.2, а лишь отображает цифровую величину наглядно. Преобразовать выход дешифратора-распределителя в уровень напряжения теоретически несложно: для этого надо выстроить делитель из цепочки одинаковых резисторов, подключить его к источнику опорного напряжения и коммутировать отводы этого делителя ключами, управляемыми от дешифратора-распределителя. Для двух- или трехразрядного кода можно использовать описанные в *главе 8* мультиплексоры типа 561КП1 и 561КП2.

Но для большего числа разрядов такой ЦАП с непосредственным преобразованием превращается в совершенно чудовищную конструкцию. Для восьмиразрядного кода потребовалось бы 256 резисторов (строго одинаковых!), столько же ключей и дешифратор с таким же числом выходов, а ведь восьмиразрядный код — довольно грубая «линейка», разрешающая способность ее не превышает четверти процента. Поэтому на практике такой метод употребляют для построения АЦП, а не ЦАП (потому что, несмотря на сложность, он обладает одним уникальным свойством, о котором поговорим далее), а здесь мы даже не будем рисовать такую схему.

Рассмотрим один из самых распространенных методов, который позволяет осуществлять преобразование «код — напряжение» не прибегая к подобным «монструозным» конструкциям.

На рис. 10.3, *а* показан вариант реализации ЦАП на основе ОУ с коммутируемыми резисторами в цепи обратной связи. Самим нам строить такие ЦАП, конечно, не придется, но для любителей укажу, что в качестве коммутирующих ключей можно применить, например, малогабаритные электронные реле серии 293 или специализированные ключи из серии 590. Однако для осуществления переключающего контакта потребовалось бы ставить по два таких ключа на каждый разряд, потому в серии 561 предусмотрена специальная микросхема 561КТ3, которая содержит четыре одинаковых ключа, работающие именно так, как показано на данной схеме: если подать на вход управления сигнал логической единицы, то выход ключа коммутируется на вход,

а если сигнал управления равен логическому нулю, то выход замыкается на «землю».

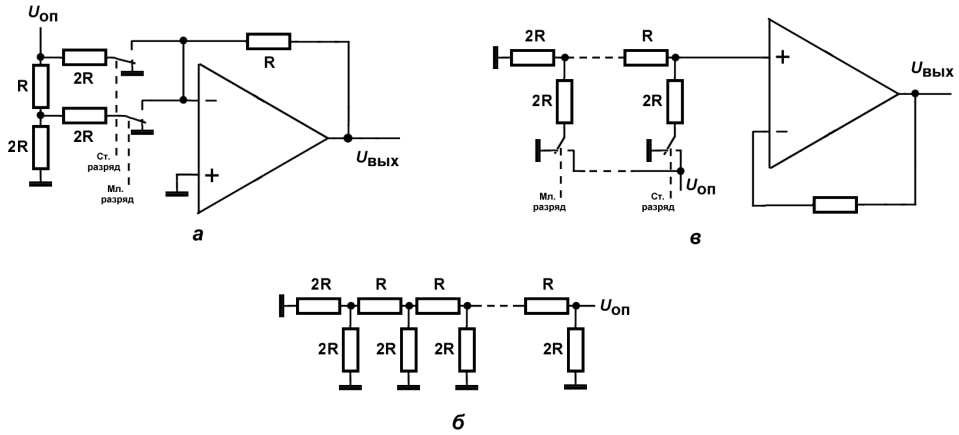


Рис. 10.3. Реализация ЦАП: *а* — двухразрядный ЦАП с отрицательным выходом; *б* — цепочка R-2R произвольной длины; *в* — ЦАП с положительным выходом

Для лучшего уяснения принципов я нарисовал всего лишь двухразрядный вариант. Два разряда — это четыре градации, т. е. выходное напряжение ОУ должно принимать четыре значения с равными промежутками. В данном случае эти напряжения равны 0, а также ¼, ½ и ¾ от опорного напряжения $U_{оп}$. Как это происходит?

Рассмотрим сначала цепочку резисторов с номиналами R и $2R$. Так как оба нижних по схеме резистора $2R$ в исходном состоянии присоединены к «земле», т. е. включены параллельно, то их суммарное сопротивление равно R. Тогда верхний по схеме резистор R и эти два резистора образуют делитель, напряжение на котором равно половине от $U_{оп}$. Предположим, на входах управления ключами оба разряда имеют нулевые значения, т. е. код принимает значения «00». В этом случае цепочка резисторов отсоединена от входа и на выходе ОУ будет напряжение, равное нулю. Пусть теперь код примет значение «01». В этом случае резистор с номиналом $2R$ младшего разряда переключается ко входу усилителя. Для самой цепочки «все равно» — к «земле» присоединен этот резистор или ко входу, потому что потенциал инвертирующего входа ОУ равен потенциалу неинвертирующего, т. е. тому же потенциалу «земли». Ко входу ОУ через сопротивление с номиналом $2R$ потечет ток, величина которого будет равна величине напряжения на его входе, т. е. $U_{оп}/2$, деленной на величину этого резистора ($2R$). Итого значение

тока будет $U_{оп}/4R$, и ток этот создаст на резисторе обратной связи ОУ, сопротивление которого также R, падение напряжения, равное $U_{оп}/4$. Можно считать и по-другому — рассматривать инвертирующий усилитель с коэффициентом усиления 0,5, что определяется отношением сопротивлений R/2R, и напряжением на входе $U_{оп}/2$. Итого на выходе всей схемы будет напряжение $U_{оп}/4$ (но с обратным знаком, т. к. усилитель инвертирующий).

Пусть теперь код принимает значение «10». Тогда все еще проще — ко входу ОУ подключается напряжение $U_{оп}$ через резистор $2R$. Коэффициент усиления тот же самый, так что на выходе будет напряжение $U_{оп}/2$. Самый сложный случай, когда код принимает значение «11» и подключаются оба резистора. При этом ОУ надо рассматривать как сумматор токов. Напряжение на выходе будет определяться суммой токов через резисторы $2R$, умноженной на величину сопротивления обратной связи R, т. е. будет равно $(U_{оп}/2R + U_{оп}/4R) \cdot R$, или просто $3U_{оп}/4$.

Способ построения цепочки R-2R с любым числом звеньев ясен (рис. 10.3, *б*). Крайние резисторы со значением $2R$ включены параллельно и в сумме дают сопротивление R, поэтому следующее звено оказывается состоящим из тех же номиналов по $2R$ и в сумме тоже даст R и т. д. Какой бы длины цепочку ни сделать, она будет делить входное напряжение в двоичном соотношении: на самом правом по схеме конце цепочки будет напряжение $U_{оп}$, на следующем отводе $U_{оп}/2$, на следующем $U_{оп}/4$ и т. д. Фактически это и есть наша двоичная линейка.

Так можно всего с помощью двух типономиналов резисторов, отличающихся ровно в два раза, строить ЦАП в принципе любой разрядности. Например, восьмиразрядный ЦАП будет содержать всего 16 резисторов и 8 ключей (с переключением), не считая резистора обратной связи, который у нас для наглядности был равен также R, но может быть любого удобного номинала. В интегральных ЦАП часто этот резистор вообще не устанавливают, а выносят соответствующие выводы наружу, так что можно легко получать любой масштаб напряжения по выходу. Например, если в нашей схеме сделать этот резистор равным $1{,}33R$, то на выходе мы получим напряжения, равные $U_{оп}$, $2U_{оп}/3$, $U_{оп}/3$ и 0. Правда, неудобство такой простейшей схемы заключается в том, что выходные напряжения будут с обратным знаком, но эта проблема легко решается: на рис. 10.3, *в* показан простейший вариант ЦАП с положительным выходом.

Большинство интегральных ЦАП построено на основе описанного принципа суммирования взвешенных токов или напряжений, хотя есть, конечно, и другие способы. Получив таким способом аналоговое напряжение из цифрового значения, мы можем теперь перейти к рассмотрению аналого-цифровых

преобразователей (АЦП), один из распространенных классов которых содержит указанные ЦАП.

АЦП

Номенклатура аналого-цифровых преобразователей существенно больше, чем ЦАП. Однако все разнообразие их типов можно свести к трем разновидностям: это АЦП параллельного действия, последовательного приближения и интегрирующие. Все эти типы АЦП встречаются на практике, т. к. обладают разными свойствами, и потому применимы в разных областях.

АЦП параллельного действия

АЦП параллельного действия — это зеркально отраженный простейший ЦАП на основе дешифратора, о котором шла речь ранее. В таких АЦП имеется делитель из k одинаковых резисторов, к каждой ступени которого подключен компаратор, сравнивающий напряжение на делителе с входным сигналом. Выходы компараторов образуют равномерный код, вроде того, что получается на выходе дешифратора-распределителя. Они подключены к шифратору с k входами, который преобразует этот код в двоичный с числом разрядов n, равным величине $\log(k)$ (округленной, естественно, до большего целого). Трудности тут те же, что и при построении основанных на подобном принципе ЦАП: для n-разрядного кода требуется $k = 2^n$ резисторов и компараторов, причем резисторов точно согласованных между собой, и компараторов также с как можно более идентичными характеристиками. Поэтому более чем 8-разрядные, такие АЦП практически не делают. А зачем их делают вообще? По одной простой причине — этот тип АЦП самый быстродействующий из всех, преобразование происходит фактически мгновенно и лимитируется только быстродействием применяемых компараторов и логики. Фактическое быстродействие АЦП такого типа может составлять десятки и сотни миллионов отсчетов в секунду (наиболее совершенных, например, MAX108 — даже до 1,5 млрд). Все остальные типы АЦП, как мы увидим, значительно медленнее.

АЦП последовательного приближения

АЦП последовательного приближения как раз и относятся к тем, что используют рассмотренные ранее ЦАП с коммутируемыми резисторами. Хотя самим в настоящее время такие АЦП строить не приходится, но для успешного их использования следует хорошо понимать, как они работают. Именно такого типа АЦП встроены в микроконтроллеры семейства AVR (см. *часть II* этой книги).

АЦП последовательного приближения работает по следующему принципу. Берется ЦАП нужной разрядности (именно поэтому мы рассматривали ЦАП раньше, чем АЦП). На его цифровые входы подается с некоего регистра код по определенному правилу, о котором далее. Выход ЦАП соединяется с одним из входов компаратора, на другой вход которого подается преобразуемое напряжение. Результат сравнения подается на схему управления, которая связана с этим самым регистром — формирователем кодов.

Есть несколько вариантов алгоритма преобразования. Самый простой выглядит следующим образом: сначала все разряды кода равны нулю. В первом такте самый старший разряд устанавливается в единицу. Если выход ЦАП при этом превысил входное напряжение, т. е. компаратор перебросился в противоположное состояние, то разряд возвращается в состояние логического нуля, в противном же случае он остается в состоянии логической единицы. В следующем такте процедуру повторяют для следующего по старшинству разряда. Такой метод позволяет за число тактов, равное числу разрядов, сформировать в регистре код, соответствующий входному напряжению. Способ довольно экономичен в смысле временных затрат, однако имеет один существенный недостаток: если за время преобразования входное напряжение меняется, то схема может «ошибаться», причем иногда вплоть до полного сбоя. Поэтому в такой схеме обязательно приходится предусматривать устройство выборки-хранения (УВХ), о которых далее.

В другой модификации этой же схемы формирование кодов осуществляет реверсивный счетчик, подобный 561ИЕ11, с нужным числом разрядов. Выход компаратора попросту подключают к выводу переключения направления счета. Изначально счетчик полностью сбрасывают, после чего подают на него тактовые импульсы. Как только счетчик досчитает до соответствующего значения кода, и выход ЦАП превысит входное напряжение, компаратор переключает направление счета, и счетчик отрабатывает назад. После окончания этого периода установления, в идеале, если напряжение на входе меняется мало, величина кода все время колеблется в пределах младшего разряда. Здесь выбросы не так страшны, но большое время установления и неизвестное заранее время реакции на быстрые изменения входного сигнала являются недостатком такого АЦП, получившего название *следящего*.

В большинстве случаев для АЦП последовательного приближения приходится ставить на входе устройства выборки-хранения (УВХ). В простейшем случае это аналоговый электронный ключ, на вход которого подается измеряемый сигнал, а на выходе стоит конденсатор. До начала измерения ключ открыт и напряжение на конденсаторе равно входному напряжению со всеми его изменениями. По команде начала измерения ключ запирается и в дальнейшем в качестве измеряемого фигурирует уже напряжение, запасенное

на конденсаторе, изменения на входе на измерительную схему не влияют. Все, казалось бы, просто, но наличие УВХ, прежде всего, достаточно сильно замедляет процесс, т. к. ключ имеет конечное сопротивление и конденсатор должен иметь время для зарядки. Кроме того, ключ вместе с конденсатором образует ФНЧ, который может искажать форму сигнала. К тому же, как бы ни было велико входное сопротивление компаратора, оно конечно, и притом ключ также имеет не бесконечно большое сопротивление в закрытом состоянии, присутствует в схеме обычно и элемент для сброса конденсатора, наконец, конденсатор также имеет собственные утечки, — все это вынуждает увеличивать емкость конденсатора и еще больше снижать быстродействие схемы. В интегральных АЦП подобного рода иногда даже предоставляется выбор между точностью (числом разрядов) и быстродействием.

Кроме выборки-хранения, в АЦП последовательного приближения требуется также время на вывод данных и подготовку к следующему циклу измерения. Все указанные причины приводят к тому, что наиболее распространенные 12-разрядные АЦП последовательного приближения имеют реальное быстродействие не выше 50—200 кГц. Как пример достаточно быстродействующей модели, приведем MAX1132, который имеет разрешение 16 бит при частоте выборок 200 кГц. АЦП последовательного приближения очень распространены и применяются там, где требуется средняя точность (до 12 разрядов) при среднем быстродействии.

Интегрирующие АЦП

Наиболее точные и одновременно самые медленные — интегрирующие АЦП. Разных типов интегрирующих АЦП вообще-то не меньше десятка, но здесь мы подробно рассмотрим только две их разновидности. Кстати, интегрирующие АЦП являются примером того, что цифровая техника вовсе не всегда достигает наивысшей точности в сравнении с аналоговой, т. к. центральный узел этих, как мы уже сказали, наиболее точных преобразователей — чисто аналоговый интегратор на ОУ.

Схема самого простого интегрирующего АЦП показана на рис. 10.4. Это так называемый АЦП *с однократным интегрированием*. В начале преобразования на С-вход динамического D-триггера поступает положительный фронт, который устанавливает выход Q в состояние логической единицы. Она является разрешающим уровнем для элемента «И-НЕ», и на вход счетчика поступают импульсы. Одновременно через выход $\overline{Q}$ запирается транзистор VT1. Конденсатор начинает заряжаться от источника стабильного тока. При равенстве значения входного измеряемого напряжения и напряжения на конденсаторе компаратор срабатывает и обнуляет триггер («ворота» на логическом

элементе «И-НЕ» запираются, транзистор открывается и разряжает конденсатор, счетчик обнуляется). Число импульсов, накопленных в счетчике к этому моменту, пропорционально входному напряжению.

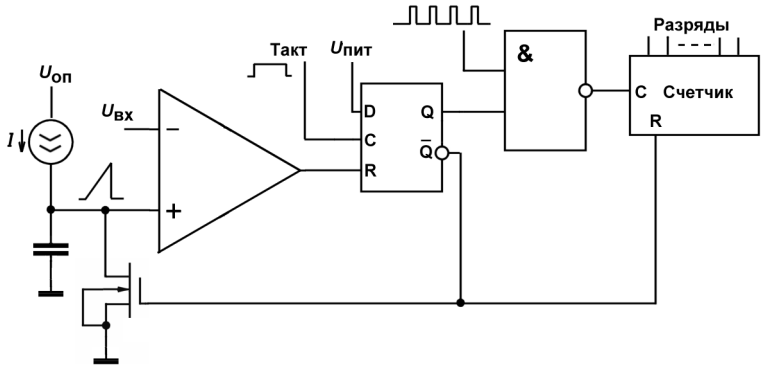


Рис. 10.4. АЦП однократного интегрирования

Источник тока вместе с конденсатором в данном случае образуют так называемый ГЛИН — генератор линейно изменяющегося напряжения. Схему можно упростить, если в качестве источника стабильного тока поставить простой резистор, питающийся от стабильного источника напряжения, но, т. к. форма кривой нарастания напряжения при этом не линейная, а экспоненциальная (см. рис. 2.9, *б*), то придется ограничиться небольшим диапазоном входных напряжений, где форма кривой еще близка к прямой линии. На практике так часто и поступают, поэтому источник тока я подробно не рисовал.

У схемы по рис. 10.4 множество недостатков и единственное достоинство — простота. При взгляде на нее непонятно, чего это я распинался насчет супервысоких характеристик интегрирующих АЦП. Главным недостатком однократного интегрирования является то, что результат преобразования тут зависит от всего на свете: от стабильности источника тока, ГЛИН (и каждого его элемента в отдельности, в первую очередь, конденсатора), порога компаратора, от неидеальности ключа для сброса и т. п. Еще хуже то, что схема в данном варианте срабатывает от мгновенного значения входного сигнала и потому весьма восприимчива к его дребезгу и вообще любым помехам. Если тактовая частота случайно окажется кратной частоте помехи (в первую очередь сетевой с частотой 50 Гц), то мы вообще можем получать каждый раз значения, весьма далекие от истины. (В теории добиться полной некратности частоты измерения и помехи можно только, если сделать тактовую

частоту изменяющейся по случайному закону. Так как отношения обычных чисел всегда образуют периодическую дробь, то на выходе такого АЦП мы получим биения выходной величины с частотой повторения периода этой дроби.) В то же время преобразование длится все равно достаточно долго, т. к. обычные значения тактовой частоты, при которых схема еще работает приемлемо, лежат в диапазоне максимум десятков килогерц (если, конечно, специально не использовать быстродействующие компараторы и логику).

Можно несколько улучшить такой АЦП: достаточно подать измеряемое напряжение на вход ГЛИН, а опорное — на компаратор. Тогда сигнал будет интегрироваться за время преобразования, причем интегрироваться очень точно, мы будем получать истинное среднее арифметическое значение сигнала за это время. Правда, сама функция преобразования при этом окажется обратной, т. е. время заряда (и значение выходного кода на счетчике) будет *обратно* пропорционально значению входного напряжения. Это неудобно, поскольку сильно усложняет обработку результата. По всем этим причинам АЦП с однократным интегрированием, несмотря на его простоту, в настоящее время не употребляют вообще и даже не выпускают в виде специализированных микросхем.

Заметки на полях

Единственное известное мне массовое применение АЦП, построенного именно по приведенной примитивной схеме (и при этом вполне справляющегося со своими обязанностями) — это схема считывания координат положения рукоятки джойстика на входе игрового порта ПК. Правда, я очень не уверен, что эта схема в современных материнских платах *в действительности* осталась той же, какой ее сделали еще в начале 80-х (впервые — в малоизвестной «домашней» модели IBM PC под названием PCj), но интерфейс для внешнего мира остался тем же.

Самое интересное, что все перечисленные недостатки можно преодолеть, как говорится, одним махом, путем небольшого усложнения схемы. Интегрирующие АЦП не получили бы такого распространения и заслуженной репутации «самых стабильных», если бы не это обстоятельство.

Идея метода, который называется *двойным* (или *двухстадийным*) интегрированием, показана на рис. 10.5. Посмотрим сначала на график, обозначенный цифрой 1. В первую часть цикла работы за фиксированное время такта $t_2 - t_1$ конденсатор интегратора заряжается током, который определяется входным напряжением $U_{вх}$. Во второй части этот конденсатор разряжается точно известным током, определяющимся опорным напряжением $U_{оп}$, до момента t_3, когда напряжение становится равным нулю. Чем больше входное напряжение, тем до большей величины зарядится конденсатор в первой части, и тем дольше он будет разряжаться во второй. Легко показать, что отношение

интервала времени $t_3 - t_2$ к известному времени такта $t_2 - t_1$ будет равно отношению входного напряжения $U_{вх}$ к опорному $U_{оп}$. Таким образом, измерив полученный интервал времени $t_3 - t_2$ обычным методом с помощью счетчика, как это сделано в схеме на рис. 10.4, мы получим на выходе код, пропорциональный входному напряжению.

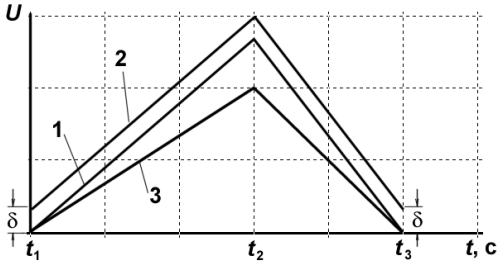


Рис. 10.5. Цикл работы АЦП двойного интегрирования: 1 — идеальный случай; 2 — при сдвиге порога компаратора; 3 — при изменении емкости конденсатора

На самом деле напряжение, до которого разряжается конденсатор, задается порогом компаратора и может в общем случае быть отличным от нуля на величину δ за счет «гуляния» порога, например, при изменении температуры. Но так как следующий цикл измерения начнется в точности с того же значения порога, то, как вы видите из графика 2, в данном случае имеет значение только изменение порога за время преобразования, которое обычно не превышает долей секунды. На результате не скажется и изменение емкости конденсатора (при тех же условиях), т. к. при этом наклон прямой и заряда и разряда изменится в одинаковой степени (график 3). В самых точных АЦП такого типа дополнительно проводят цикл «автокоррекции нуля», когда на вход подают нулевое напряжение и результат потом вычитают из значения кода, полученного в рабочем цикле. Мало того, здесь даже не требуется «кварцованная» частота и в теории всю схему можно «заводить» от любого RC-генератора — при условии, что время такта $t_2 - t_1$ и частота заполнения «ворот» для подсчета длительности результирующего интервала $t_3 - t_2$ задаются от одного и того же генератора.

Но чудес не бывает — точность и стабильность преобразования здесь полностью определяются точностью и стабильностью значения $U_{оп}$. Это общее условие для всех без исключения конструкций АЦП и ЦАП. Между прочим, обратите внимание, что $U_{вх}$ и $U_{оп}$ образуют в совокупности нечто вроде неинвертирующего и инвертирующего входа ОУ. Эта аналогия куда более полная, чем кажется, и манипулируя этими величинами, можно выделывать с выходным

кодом всякие штуки, в частности, подгонять масштаб преобразования к нужному диапазону. Другое облегчение, которое можно получить от этой связи, заключается в возможности проведения относительных измерений, когда входное и опорное напряжения получаются от одного источника и тем самым имеют одинаковую относительную погрешность. Получается нечто вроде явления ослабления синфазного сигнала в ОУ. В идеале тогда мы получаем очень точные измерения, однако идеал этот, к сожалению, редко достижим на практике.

Кстати, в интегрирующих АЦП такого рода для более полного подавления помех нужно делать первую часть цикла интегрирования именно кратным периоду помехи. Тогда в цикле укладывается целое число периодов помехи и она усредняется. Практически наибольшее влияние оказывает сетевая помеха частотой 50 Гц, поэтому частоту циклов стараются делать в «круглых» числах.

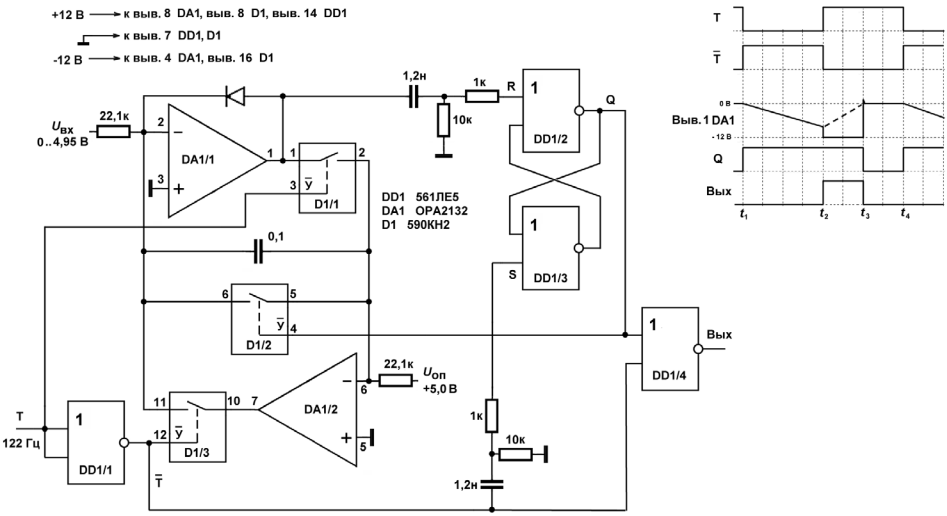


Рис. 10.6. Простой вариант АЦП двойного интегрирования (ПНВ)

Простой вариант практической схемы АЦП двойного интегрирования (преобразователь напряжение — время) приведен на рис. 10.6. Счетная часть на схеме не показана. Для понимания того, как работает схема, следует обратить внимание, что управляющий вход у ключей типа 590КН2 (D1) инверсный, т. е. при низком уровне на управляющем входе ключ открыт, при высоком — заперт.

Рассмотрим диаграмму работы (рис. 10.6, справа). В момент отрицательного перепада на тактовом входе T, RS-триггер устанавливается в единицу

по выходу Q. Так как на входе T в этот момент отрицательный уровень, ключ D1/1 открывается, остальные ключи заперты. Конденсатор подключается в обратную связь верхнего ОУ (DA1/1) и начинается цикл интегрирования входного напряжения (напряжение на конденсаторе возрастает по абсолютной величине, т. е. на выходе DA1/1 падает, т. к. интегратор инвертирующий). В момент окончания отрицательного полупериода тактовой частоты ключ D1/1 запирается, а D1/3 открывается, заряженный конденсатор оказывается подключенным в обратную связь второго ОУ (DA1/2).

Начинается цикл интегрирования опорного напряжения (изменение напряжения на конденсаторе показано на диаграмме пунктирной линией). Так как обратная связь в первом ОУ теперь отсутствует, то он сработает, как компаратор: сначала на его выходе установится напряжение, равное отрицательному питанию (или близкое к нему), а в момент равенства напряжения на конденсаторе нулю выход резко устремится от отрицательного к положительному питанию (но его ограничит на уровне примерно +0,6 В диод, включенный в обратную связь, который нужен для того, чтобы не затягивать переходной процесс). Положительный перепад передастся на обнуляющий вход RS-триггера и установит его выход Q в состояние логического нуля. При этом откроется ключ D1/2 и закоротит конденсатор, прерывая таким образом процесс интегрирования. На входе верхнего ОУ установится напряжение, равное нулю, а на выходе, вообще говоря, т. к. обратная связь по-прежнему отсутствует, оно станет неопределенным (на диаграмме оно показано условно в виде нулевого уровня). Это состояние длится до конца периода тактовой частоты, а с отрицательным перепадом на входе T ключи D1/3 и D1/2 закроются и все начнется сначала. На выходе схемы возникает положительный импульс напряжения, длительность которого $t_3 - t_2$ пропорциональна входному напряжению, согласно соотношению:

$$\frac{t_3 - t_2}{t_2 - t_1} = \frac{U_{\text{вх}}}{U_{\text{оп}}},$$

где промежуток времени $t_2 - t_1$ жестко задан внешним тактовым генератором.

Описанная схема рассчитана для получения разрешающей способности 12 разрядов или 4096 градаций. Максимальная частота отсчетов при тактовой частоте 1 МГц составит 122 Гц. Исходя из этого выбраны величины сопротивлений и емкость конденсатора. Точность преобразования напрямую зависит от стабильности резисторов, поэтому их нужно выбирать с точностью не хуже 0,1%, в этом случае абсолютная точность может достигнуть 10 разрядов без дополнительной калибровки. Однако $U_{\text{оп}}$ тоже должно иметь не меньшую стабильность.

По такому принципу устроены АЦП 572ПВ2 и 572ПВ5, которые мы будем подробнее рассматривать далее. Ранее были широко распространены ПНЧ — преобразователи напряжение-частота (в основном на основе микросхемы 555, см. *главу 9*), однако большинство их реализаций обладает тем же недостатком, что и однократный интегратор, т. е. их точность зависит от качества компонентов напрямую. Значительно более точные преобразователи (до 24 двоичных разрядов) получаются на основе интегрирующих преобразователей, которые также используют принцип двойного интегрирования, но на их выходе получается не интервал времени, который еще нужно сосчитать, а число-импульсный код, т. е. число импульсов за определенный промежуток времени, пропорциональное входному напряжению. АЦП такого типа называются еще *дельта-сигма-преобразователями* или *АЦП с уравновешиванием заряда*. Они широко распространены в интегральном исполнении, большинство наиболее высокоразрядных АЦП построены именно так.

Конструируем цифровой термометр

Цифровые термометры конструировать самостоятельно имеет практический смысл по крайней мере по двум причинам. Во-первых, фирменные приборы для жилых или производственных помещений обычно имеют невзрачный дизайн с ЖК-индикаторами и корпусами белого или «компьютерного» серого цвета. Во-вторых, рынок подобных бытовых устройств вообще достаточно беден, чему есть одна веская причина: сделать дешевый, достаточно точный и притом универсальный цифровой термометр, который, подобно традиционным спиртовым, можно и в воду опускать, и на мороз зимой выставить, очень непросто. Терпеливый радиолюбитель вполне может сделать конструкцию куда лучше фирменной — удобную, красивую и приспособленную под свои нужды, а «приставить» к такому термометру измеритель влажности, давления и еще чего угодно — вопрос только денег, и мы займемся этим позднее.

АЦП 572ПВ2 и ПВ5

Основой принципиальной схемы нашего термометра будет выпускающаяся уже более 20 лет очень удачная разработка 572ПВ2 (ICL7107), которая представляет собой АЦП двойного интегрирования с выходом в параллельном семисегментном коде с расчетом на 3,5 десятичных разряда.

Подробности

Что означает цифра 3,5 (в спецификациях нередко пишут в форме 3½) — не может же использоваться полразряда? Действительно, при полном выходном диапазоне этой микросхемы, который составляет число ±1999, нужно подклю-

чать 4 индикатора, однако старший из них будет индицировать только цифру «1» (и, при необходимости, знак «минус»). Число 3½, согласно договоренностям, достигнутым еще в 1970-е годы, и означает, что старший разряд служит только для индикации «0» или «1». Если прибор в старшем разряде индицирует больше знаков (обычно по образцу 3999 или 5999), то такое разрешение обозначается, как 3¾. Конечно, целая часть может меняться, в зависимости от общего числа разрядов (5½, 4¾, и т. д.). Заметим, что точность с разрешающей способностью, вообще говоря, не связана, и почти всегда ниже последней — например, у мультиметра, который лежит передо мной, разрешающая способность также 3,5, что эквивалентно почти 11 двоичным разрядам (1/2048), но погрешность при измерении напряжения 0,2%, что составляет всего 9 двоичных разрядов (1/512). При соблюдении некоторых (довольно жестких) требований к подаваемым сигналам и построению схемы точность обсуждаемого АЦП может быть эквивалентна разрешению — тем же 11 двоичным разрядам, т. е. приведенная погрешность составит 0,05%, что очень и очень неплохо.

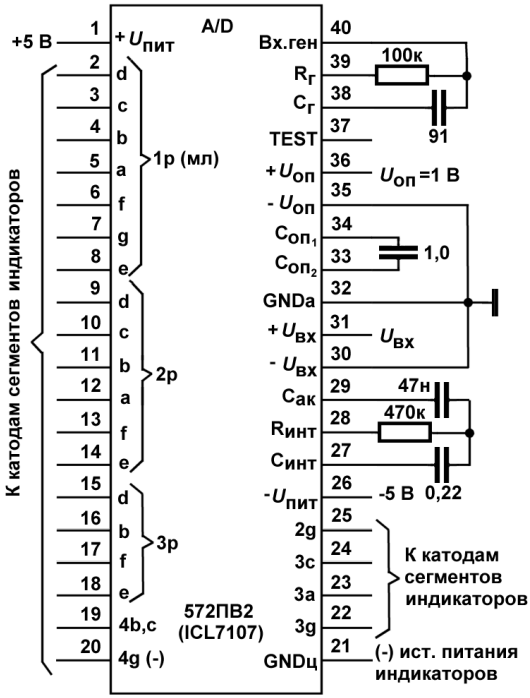


Рис. 10.7. Вариант типового включения микросхемы 572ПВ2 (ICL7107) в корпусе DIP-40

Основная (типовая) схема включения микросхемы 572ПВ2 показана на рис. 10.7. Микросхема имеет два собственных питания: положительное 5 В (от 4,5 до 6 В) и отрицательное, которое может варьироваться в довольно

большом диапазоне от –9 до –3,5 В (это обстоятельство позволяет при необходимости использовать для отрицательного питания не слишком стабильные преобразователи-инверторы, см. *главу 4*). Светодиодные индикаторы можно подключать напрямую, без каких-либо дополнительных резисторов (ток через сегмент при этом равен 5—8 мА), при этом им лучше обеспечить отдельное питание. Управление индикаторами здесь осуществляется коммутацией на «землю», поэтому нужен индикатор с общим анодом. Однако выходы управления дисплеем не являются выходами «с открытым коллектором» (точнее — истоком), а представляют собой обычный КМОП-выход, у которого вытекающий ток в состоянии логической единицы может составить примерно 0,5 мА, а при логическом нуле, как уже говорилось, он равен примерно 5—8 мА (для вывода 19, который управляет одновременно двумя сегментами при засветке символа «1» в старшем разряде, этот ток составляет 10—16 мА). Эти параметры следует учитывать при управлении индикаторами через внешние ключи, если требуется повышенное напряжение или ток. До 7 В амплитудного значения питания индикаторов, как показывает опыт, микросхема выдерживает и без ключей.

Выпускается совершенно идентичная по функциональности и практически совпадающая по разводке выводов микросхема 572ПВ5 (ICL7106), которая отличается только тем, что она предназначена для управления ЖК-индикаторами, а не светодиодными. Просто заменить ЖК-индикатор на светодиодный и наоборот нельзя потому, что, как вы знаете из *главы 3*, для управления ЖК-индикаторами требуется переменное напряжение, иначе отключенные сегменты «зависнут» в поглощающем свет состоянии. Поэтому при замене ПВ2 на ПВ5 отличие в схеме заключается в том, что вывод 21 представляет собой не «цифровую землю» (GNDц), а подсоединяется к общему выводу ЖК-индикатора. Отдельное питание тогда, естественно, не требуется.

Особый вопрос в этом случае представляет засветка запятой, если ее по ходу дела надо «передвигать» или просто «гасить». В варианте со светодиодами это несложно делать абсолютно автономно от микросхемы, отдельным ключом, а для ЖК придется для нее также обеспечить подобный режим управления с помощью переменного напряжения, иначе при подаче постоянного напряжения она просто засветится навсегда и к тому же будет резко выделяться большим контрастом. Разработчики рекомендуют для этой цели подключить к выходу 21 обычный КМОП-инвертор. При этом (как и в случае подключения внешнего генератора, см. далее) в качестве «цифровой земли» в ПВ5 следует использовать вывод 37 (TEST).

Для обеих микросхем опорное и входное напряжения не должны выходить за пределы, на 1 В отступающие от потенциалов $+U_{пит}$ и $-U_{пит}$. Для микросхемы

ПВ2, вообще говоря, требуется двуполярное питание, т. к. «цифровая земля» GNDц должна иметь общую точку с аналоговой частью для внутреннего согласования уровней управляющих сигналов. Однако можно обойтись одним питанием +5 В (подсоединив вход $-U_{пит}$ к «земле»), если опорное и измеряемое напряжения находятся в пределах от 1 до 4 В.

Есть и более современные варианты этих разработок, например, с очень малым потреблением, но параметры описанных микросхем и так достаточно хороши: при тактовой частоте 50 кГц время преобразования составляет 0,32 с (16000 периодов тактовой частоты), а потребление при этом не превышает 0,6 мА (не считая, конечно, потребления индикаторов в LED-варианте).

Удобство микросхем ПВ2 и ПВ5 заключается и в том, что они оперируют с двуполярными входными напряжениями, автоматически определяя и высвечивая знак. На схеме рис. 10.7 показан вариант с общими «землями». Диапазон входного измеряемого напряжения определяется опорным, с помощью которого и задается масштаб, при этом опорное должно находиться в пределах 0,1—1 В, а измеряемое может по абсолютной величине превышать его, в соответствии с разрешающей способностью, ровно в два раза. Если, например, опорное напряжение равно 1 В, то измеряемое может быть в пределах ±2 В (точнее ±1,999 В), а в общем случае выходной код определяется выражением $N = 1000 \cdot \frac{U_{вх}}{U_{оп}}$. Если значение входного напряжения превышает предел $+2U_{оп}$, младшие три разряда гаснут, а если снижается ниже $-2U_{оп}$ — гаснет все, кроме знака «минус».

Подробности

Оба входных напряжения — опорное и измеряемое — могут быть «плавающими», без общей «земли», единственное требование, чтобы их значения не выходили за пределы питания (а по абсолютной величине они, естественно, должны соответствовать указанным ранее требованиям). В этом случае вывод 32 («аналоговая земля») не используется. На этом выводе тогда присутствует напряжение, равное ($U_{+пит}$ – 2,8) В. При необходимости его можно выбрать в качестве опорного (не само напряжение относительно «земли», которая в данном случае есть довольно условное понятие, а именно разность между положительным питанием и выводом 32). Однако стабильность этого напряжения невелика, и так рекомендуется поступать только в уж очень экономичных схемах. Особенно это плохо в случае ПВ2, в которой выходные каскады за счет большого тока сильно (и неравномерно по времени из-за разного числа подключенных сегментов) нагревают кристалл и это напряжение начинает «плавать».

Тактовую частоту микросхем следует выбирать из ряда: 200, 100, 50 и 40 кГц, при этом частота помехи 50 Гц будет укладываться в длительность фазы интегрирования входного напряжения целое число раз и такая помеха будет интегрироваться полностью. Тактовую частоту можно задавать тремя способами: с помощью RC-цепочки, как показано на рис. 10.7, с помощью кварца,

подключаемого к выводам 39 и 40, а также внешним генератором, выход которого подключается в выводу 40 (в ПВ2 при общим проводом служит вывод 21 «цифровая земля», а в ПВ5 — вывод 37 «TEST»). На практике чаще встречается первый способ, при этом частота будет равна примерно $0{,}45R_{г}C_{г}$. В фирменной документации на этот счет есть некоторая неясность, т. к. рекомендуется $R_{г} = 100$ кОм при $C_{г} = 100$ пФ, и тогда согласно приведенной формуле частота должна составить 45 кГц. Это далеко и от 40 и от 50 кГц, рекомендуемых для частоты помехи 50 Гц, и не очень совпадает с 48 кГц, рекомендуемыми для помехи 60 Гц. Все отечественные описания микросхем ПВ2 и ПВ5 изящно обходят этот вопрос, просто повторяя фирменные рекомендации. Думается, что составители документации имели в виду все же помеху 60 Гц (т. е. тактовую частоту 48 кГц), поэтому в отечественном варианте следует снизить емкость $C_{г}$ до 91 пФ — так будет корректнее. Вообще, ошибка в ±5% тут вполне допустима.

Номиналы емкостей и резисторов на рис. 10.7 приведены для случая опорного напряжения, равного 1 В (и тактовой частоты 50 кГц). При опорном напряжении 0,1 В $C_{ак}$ нужно увеличить до 0,47 мкФ, $C_{инт}$ уменьшить до 0,1 мкФ, а $R_{инт}$ уменьшить до 47 кОм. В остальных случаях эти номиналы должны быть изменены в указанных пределах примерно пропорционально изменению опорного напряжения.

К выбору типов компонентов следует подходить весьма тщательно, от этого сильно зависит в первую очередь линейность преобразования. Резисторы все могут быть С1-4 (МЛТ). Конденсатор тактового генератора $C_{ген}$ может быть керамическим (типа КМ-5, КМ-6). Остальные конденсаторы ($C_{инт}$, $C_{оп}$, и $C_{ак}$) должны быть с органическим диэлектриком, лучше всего фторопластовые (К72П-6, К72-9) или полистироловые (К71-4, К71-5), но подойдут и полиэтилентерефталатные (К73-16, К73-17). Эти конденсаторы могут ужаснуть вас своими размерами, но ничего не поделаешь — такова плата за стабильность. Высокие конденсаторы (как К73-17) следует устанавливать лежа, хотя при этом площадь платы увеличивается, но зато конденсаторы не торчат над всеми остальными компонентами Это, кроме всего прочего, повышает надежность монтажа, т. к. меньше вероятность «выкорчевать» конденсатор, случайно положив поверх платы книгу «Занимательная микроэлектроника».

Практическая схема термометра

Теперь, вооружившись всеми этими знаниями, приступим наконец к нашему термометру. И сначала нам надо будет посчитать, что мы имеем на входе и что хотим при этом получить на выходе?

Начнем с выхода: температура традиционно демонстрируется в виде «XX,X», т. е. достаточно трех индикаторов. Таким образом, мы должны задействовать

только три младших разряда, при этом диапазон температур получится от –99,9 до +99,9°. Собственно говоря, такой диапазон чересчур широкий — измерять температуру ниже, скажем, –40 вряд ли когда-нибудь случится. Практически для «погодного» термометра хватило бы и диапазона от –50 до +50, но ничего не поделаешь. При таком подключении мы теряем ровно два точностных разряда, ужимая диапазон в 4 раза, ведь без какого-либо изменения в измерительной части никто нам не запретит подключить все четыре индикатора и демонстрировать температуру от –199,9 до 199,9°. Конечно, никто не запрещает вам использовать такой диапазон полностью, или, скажем, половину его со сдвигом, от –50 до +150° — все будет определяться возможностями калибровки. Мы здесь ограничимся тем, что калибровку будем выполнять от 0 до 50 °С. Однако считать придется на весь возможный диапазон исходя из того, что входное напряжение АЦП $U_{вх}$ при изменении температуры от 0 до +199,9° должно изменяться от 0 до $U_{оп}$ в соответствии с приведенной формулой для отображаемого числа N.

Так как мы собираемся делать более-менее точный прибор, то выберем медный датчик и прикинем, какое было бы желательно иметь его сопротивление. Обычные токи через датчик должны составлять порядка 1—3 мА, иначе медная катушка приемлемых размеров будет сама нагреваться. Проще всего в качестве датчика использовать обмотку малогабаритного реле из серий, например, РЭС-60, РЭС-80, РЭС-79 или РЭС-49 — какое окажется под рукой, и чем старше «возрастом», тем лучше, т. к. характеристики меди при хранении стабилизируются. Указанные мной реле имеют полностью герметизированный металлический корпус, остается только изолировать от внешней среды выводы.

У меня «под рукой» оказалось реле типа РЭС-60 с обмоткой 800±120 Ом (пасп. РС4.569.435-01). Изменения на диапазон 200° составят примерно 640 Ом (исходя из ТКС меди, равного примерно 0,4%/град). Выберем $U_{оп}$ = 0,5 В, тогда ток через датчик-обмотку должен составить 0,5 В/640 Ом ≈ 0,8 мА. Так как рабочие напряжения здесь не превышают по абсолютной величине 0,5 В, то мы сможем обойтись для АЦП одним питанием +5 В, нужно будет только максимально «сдвинуть» эти напряжения к середине питания.

Общая схема термометра показана на рис. 10.8. Рассмотрим сначала включение датчика. Для того чтобы при нуле градусов термометр показывал «0», нужно на вход АЦП подавать разность текущего напряжения на датчике и значения его при нулевой температуре. В данном случае это делается с помощью мостовой схемы.

Конструкция на сдвоенном ОУ MAX478 представляет собой два идентичных источника тока 0,8 мА. Оба ОУ DA1, полевые транзисторы VT1 и VT2 и ре-

зисторы R16—R19 образуют верхнюю половину моста, а нижняя состоит из датчика температуры Rt и опорного резистора R20, сопротивление которого должно быть равно сопротивлению датчика при 0°C (≈ 740 Ом, точное значение подбирается при калибровке, для удобства предусмотрен параллельный резистор). Разность этих напряжений подается на АЦП в качестве входного напряжения. Фильтр R22, C6 нужен для лучшего сглаживания помех (конденсатор C6 может быть керамическим).

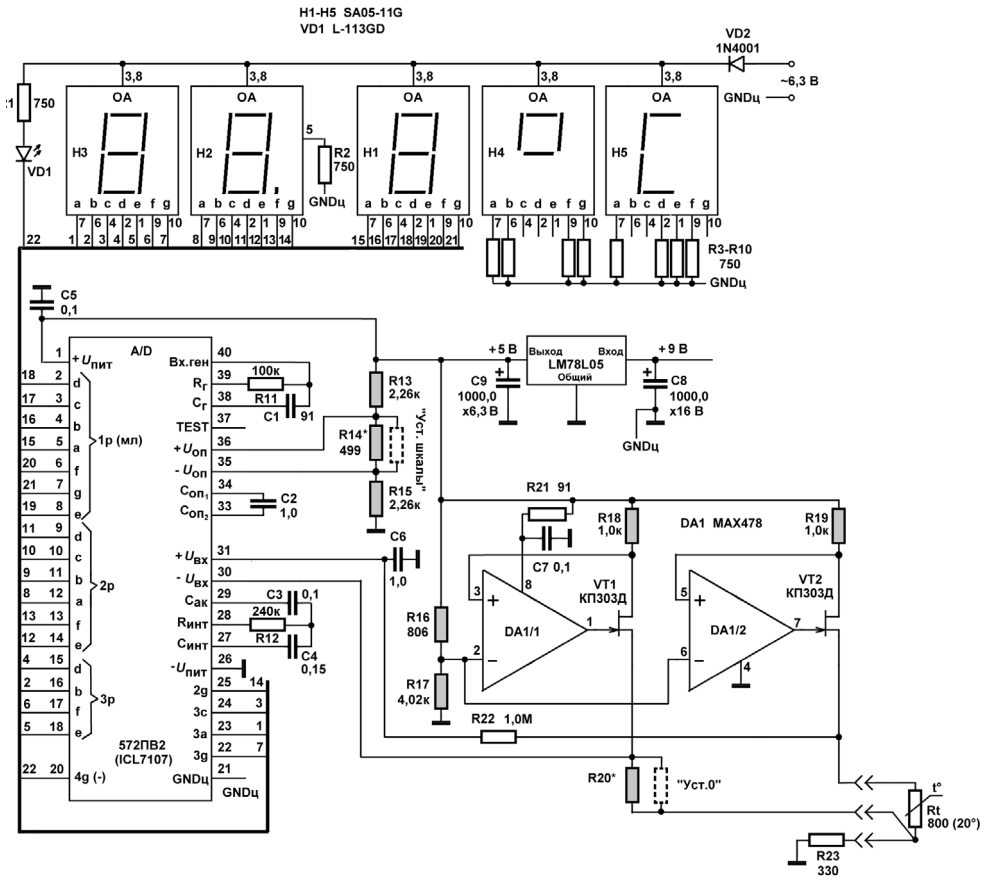


Рис. 10.8. Прецизионный цифровой термометр на микросхеме 572ПВ2

Обратим теперь внимание на хитрую схему включения самого датчика, которая носит название «трехпроводной» и позволяет избежать влияния соединительных проводов и, главное, помех, которые наводятся на них. Сами по себе провода влияют слабо, т. к. в данном случае достаточно, чтобы они имели

сопротивление, меньшее, чем 1/2000 сопротивления датчика, что составляет примерно 0,4 Ом (это даже слишком жесткое требование, т. к. не само сопротивление подводящих проводов вносит погрешность, а только его температурные изменения). Это вполне обеспечит провод МГТФ-0,35, если его суммарная длина не превысит 40 м.

Однако в трехпроводной схеме и столь малые изменения нивелируются тем, что два одинаковых провода, соединяющие опорный резистор с датчиком и датчик с источником тока, оказываются включенными в разные плечи моста, потому их изменения взаимно компенсируются. Наведенные на этих проводах помехи ведут себя точно так же. А третий провод, соединяющий датчик с «землей», оказывается включенным в оба плеча сразу, и создает чисто синфазную помеху. Дополнительный резистор R23, включенный в этот провод, «подтягивает» напряжение разбаланса моста к середине напряжения питания (падение напряжения на R23 составляет около 1 В). При возможном изменении напряжения питания опорное напряжение и сигнал с выхода моста будут меняться пропорционально, поэтому ошибки не возникнет.

Все резисторы, выделенные на схеме темным, должны иметь точность не хуже 1%, (например, С2-29В). Номиналы их, естественно, необязательно должны быть именно такими, как указано на схеме, и могут меняться в очень широких пределах, но соотношения должны быть выдержаны точно. При ином сопротивлении датчика соотношения этих резисторов, а также сопротивления резисторов R20 и R23 придется пересчитать, при этом желательно приблизительно сохранить значения напряжений в схеме, особенно это касается близости к середине напряжения питания.

Питание индикаторов в этой схеме обязательно должно осуществляться от отдельной обмотки трансформатора. Индикаторы зеленого свечения (с буквой G) можно заменить на любые другие, по вкусу, однако индикаторы бóльших размеров, чем указаны на схеме (с цифрой высотой более 0,5 дюйма), придется подключать через дополнительные ключи с повышенным напряжением питания. Так как мы четвертый разряд не используем, то не имеет смысла ставить целый индикатор для одного только знака минус, и его индикация производится с помощью одного плоского светодиода. Они бывают разных размеров, и чтобы схема выглядела красиво, следует подогнать светящуюся полоску по ширине сегментов индикатора. В данном случае светодиод L113 имеет размеры 5×2 мм, но сегменты заметно у́же, поэтому часть торцевой поверхности нужно аккуратно закрасить любой непрозрачной краской. Залить такой краской следует и боковые поверхности светодиода, иначе вместо минуса вы получите неопределенное светящееся пятно.

Если яркость «минуса», запятой (вывод 5 индикатора H2) и индикаторов H4—H5, постоянно демонстрирующих знак «°C», будет отличаться от яркости основных разрядов, нужно подобрать резисторы R1—R10. Источник питания следует рассчитывать на 250 мА по напряжению ~6,3 В (по напряжению +5 В потребление не достигает и 10 мА). Конечно, в целях экономии места, стоимости и потребления тока индикаторы H4—H5 можно исключить.

Датчик можно изготовить следующим образом. Берется трубка (лучше пластмассовая) длиной примерно 10 см и такого диаметра, чтобы все выводы реле, в том числе выводы обмотки с припаянными проводами, свободно помещались внутри. Места пайки на всякий случай следует изолировать термоусадочным кембриком. Затем нужно пропустить провода через трубку и обязательно в месте выхода из трубки также надеть на них отрезок кембрика, чтобы ограничить радиус перегиба. Потом необходимо залепить пластилином щели между корпусом реле и торцом трубки и залить внутренность ее эпоксидной смолой. Пластилин удаляется начисто с помощью чистого бензина. Готовый датчик следует покрыть атмосферо- и водостойкой эмалью или лаком.

Наладка схемы начинается с проверки правильности разводки индикаторов. Для этого вывод «TEST» следует замкнуть с напряжением питания — индикаторы должны загореться все, показав значение «888». Затем на место калибровочных резисторов R14 и R20 следует впаять резисторы большего номинала, а параллельно им — переменные резисторы с таким значением сопротивления, чтобы вместе они составляли номинал примерно на 5—10% больший расчетного.

Теперь можно приступать к процедуре калибровки. Набейте термос толченым льдом (зимой для этой цели лучше подойдет снег) пополам с водой — это будет первая калибровочная точка. Вторая может быть обеспечена просто теплой водой с температурой от 40 до 60 °C, причем поддерживать точную температуру необязательно, только за ней нужно все время следить (хотя, разумеется, наличие термостата предпочтительнее). Размещать датчик желательно так, чтобы и он, и эталонный термометр не касались стенок сосуда, причем и воду и смесь в термосе при этом следует обязательно перемешивать.

Помещая датчик в смесь льда и воды, с помощью резистора R20 устанавливают нулевые показания термометра. Затем датчик помещают в теплую воду вместе с образцовым термометром, и с помощью резистора R14 устанавливают показания, соответствующие показаниям этого термометра. Так как

у нас при 0° мост находится в равновесии, то в принципе корректировки нуля и крутизны независимы, и одной итерации достаточно, но на всякий случай следует несколько раз перенести датчик из нулевой температуры в теплую воду и обратно и при необходимости подкорректировать показания. Не забывайте, что каждый раз датчик следует выдерживать при соответствующей температуре не менее нескольких минут.

По окончании калибровки переменные резисторы заменяют на постоянные (на схеме они показаны пунктиром) — параллельные резисторы удобно впаивать в готовую схему прямо к выводам основных. Эти дополнительные резисторы могут быть типа С1-4 или МЛТ (при условии, что основной резистор не слишком отличается от окончательного номинала). Если все сделано аккуратно, то погрешность такого термометра в диапазоне от –50 до +50 °С не превысит его разрешающей способности, равной 0,1 °С.

Часть II

МИКРОКОНТРОЛЛЕРЫ

Глава 11

Анатомия микроконтроллера

Все жалуются на свою память, но никто не жалуется на свой разум.

Франсуа де Ларошфуко

Цифровые электронные устройства могут выполнять в автоматическом режиме довольно сложные функции. Устройства управления военной техникой в 40—50 годы XX века так и делали, для них строили специальные схемы на каждый раз, для каждой конкретной задачи, иногда очень «навороченные», и весьма остроумно придуманные. Эти схемы объединяли цифровые и аналоговые узлы, реализовывавшие различные функции, вплоть до решения в реальном времени сложнейших дифференциальных уравнений. Вы только представьте сложность задачи управления межконтинентальной баллистической ракетой, которая даже в те времена, когда не было ни спутников наведения, ни систем глобального позиционирования, обеспечивала точность попадания в радиусе нескольких десятков метров на расстоянии в тысячи километров!

Характерная черта таких устройств — они построены в принципе из одних и тех же основных элементов. Особенно это касается цифровой техники — со времен Клода Шеннона известно, что любая цифровая функция может быть реализована всего на нескольких базовых «кирпичиках», и мы видели в предыдущих главах, как на основе таких «кирпичиков» — логических элементов — строятся последовательно все более сложные устройства, вплоть до сумматоров и многофункциональных счетчиков, которые затем уже могут комбинироваться в схемы любой степени сложности. Возникает естественная мысль: а нельзя ли соорудить универсальное устройство, которое бы могло выполнять любые подобные функции, раз в какой-то глубинной основе своей они похожи?

К этой мысли человечество двигалось двумя совершенно разными путями. Один из них связан с никогда не покидавшей человечество мечтой о построении искусственного разума. Через арифмометр Паскаля, аналитическую машину Бэббиджа, математическую логику Буля, теоретические построения Тьюринга и Шеннона, через первые электромеханические компьютеры Конрада Цузе, Эйкена и Атанасова этот путь воплотился в ЭНИАКе — построенной в 1946 году электронной вычислительной машине, которая стала символом начала компьютерной эпохи (хотя, добавим, была не самой первой, и не единственной даже в те времена).

Ученые сразу поняли, каковы потенциальные возможности этого устройства — зародилось направление «искусственного интеллекта», стали обсуждаться проблемы автоматического перевода, шахматного компьютера, распознавания образов — в общем-то, многие из них не решены и до сих пор, несмотря на то, что мощность компьютеров возросла в миллионы раз, и вряд ли будут решены в ближайшее время[1]. А вот другая сторона этой революции до времени не обсуждалась столь широко: ведь по сути компьютер и есть то самое универсальное электронное устройство, которое может выполнить любую задачу — от наведения баллистической ракеты на цель до банального переключения режимов стиральной машины, нужно только иметь соответствующую программу. Но обсуждавшийся во введении принцип эквивалентности «железа» и программ, благодаря диссертации Шеннона понятный ученым и инженерам еще задолго до эпохи всеобщей компьютеризации, дошел до практики далеко не сразу, поскольку «железо» резко отставало от нужд практики. Как подумаешь, что вопрос «может ли машина мыслить» обсуждался во времена ЭВМ, по вычислительной мощности эквивалентных сегодняшнему двухдолларовому микроконтроллеру…

Более того, в этом вопросе движение было скорее с обратной стороны — нужды компьютерной отрасли вызвали бурное развитие электроники. И это логично, ведь первые ЭВМ были огромными, потребляли энергии, как небольшой завод, и требовали непрерывного обслуживания (плановое ежесуточное время работы первых советских ЭВМ — 16 часов, остальное — ремонт). Кому в те времена могла прийти мысль даже о том, чтобы дать компьютер каждому в персональное пользование, не то что управлять с его помощью стиральной машиной, правда? Революция произошла лишь с изобретением

[1] В 1950 году Алан Тьюринг опубликовал работу «Вычислительные машины и интеллект», в которой предположил, что «думающий» компьютер, который нельзя было бы отличить по поведению от человека, должен иметь объем памяти примерно в 10^{10} бит — чуть больше гигабайта. Примерно столько памяти сейчас рекомендуется для ПК, работающих с ОС Windows Vista.

микропроцессора в фирме Intel в 1971 году. С этого момента инженерам-электронщикам пришлось учить программирование.

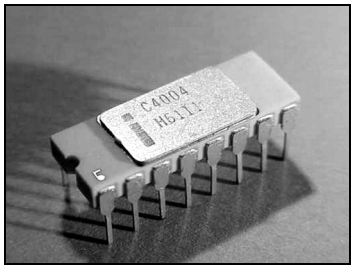


Рис. 11.1. Микропроцессор Intel 4004

Первоначально корпорация Intel не помышляла ни о каких процессорах и занималась разработкой и продажами микросхем памяти, на которые тогда как раз начиналось увеличение спроса. В 1969 г. в Intel появились несколько человек из Busicom — молодой японской компании, занимающейся производством калькуляторов. Им требовался набор из 12 интегральных схем в качестве основного элемента нового дешевого настольного калькулятора. Проект был разработан Масатоши Шима, который и представлял японскую сторону. Тед Хофф (Marcian E. «Ted» Hoff, p. 1937), руководитель отдела, занимавшегося разработкой применений для продукции Intel, ознакомившись с проектом, понял, что вместо того, чтобы создать калькулятор с некоторыми возможностями программирования, можно сделать наоборот, компьютер, программируемый для работы в качестве калькулятора. Развивая идею, в течение осени 1969 г. Хофф определился с архитектурой будущего микропроцессора. Весной в отдел Хоффа пришел (все из той же уже известной нам Fairchild) новый сотрудник Фредерик Фэггин (Federico Faggin), который и придумал название для всей системы: «семейство 4000». Семейство состояло из четырех 16-выводных микросхем: 4001 содержал ROM на 2 килобайта; 4002 — RAM с 4-битным выходным портом для загрузки программ; 4003 представлял собой 10-битный расширитель ввода/вывода с последовательным вводом и параллельным выводом для связи с клавиатурой, индикатором и другими внешними устройствами; наконец, 4004 был 4-битным ЦПУ (центральным процессорным устройством). Он состоял из 2300 транзисторов и работал с тактовой частотой 108 кГц. 15 ноября 1971 г. было объявлено о создании первого микропроцессора. Busicom приобрела разработку, заплатив Intel $60 000. Но в Intel решили возвратить Busicom эти деньги, чтобы вернуть себе права на микропроцессор. i4004 обладал вычислительной мощностью, сравнимой с первым электронным компьютером ЭНИАК. Свое первое практическое применение 4004-й нашел в таких системах, как устройства управления дорожными светофорами и анализаторы крови. Он использован в бортовой аппаратуре межпланетного зонда Pioneer-10, который поставил рекорд долгожительства среди подобных аппаратов: он был запущен NASA в 1972 г., а к 1 сентября 2001 г. Pioneer-10 удалился от Земли на 11,78 млрд км и все еще работал.

Еще раз повторим — для понимания того, как работают микропроцессорные системы, нужно очень твердо усвоить, что программирование процессора и составление логических схем есть в полном смысле слова один и тот же процесс, только выраженный на разных языках: либо в виде последовательности команд процессора, либо в виде схемы. Грубо говоря, при переходе на МК вы заменяете паяльник персональным компьютером (ПК), причем программировать много проще, потому что гораздо легче поправить ошибку. Типичная иллюстрация принципа эквивалентности: на процессоре 8086 операции с действительными числами выполнялись с помощью подпрограмм, но выполнение программы всегда медленнее, чем работа «железок». Поэтому к нему сначала добавили арифметический сопроцессор (8087), а потом (начиная с 486-х) и вовсе интегрировали блок обработки чисел «с плавающей точкой» внутрь процессора. В результате программы упростились, а процессор усложнился, но с точки зрения пользователя ничего (кроме ускорения работы) не произошло.

Заметки на полях

Легкостью программирования современных электронных узлов стали широко пользоваться производители оборудования, особенно в последние годы, чтобы оптимизировать свои прибыли. Делается это примерно так: разрабатывается некое устройство (видеокарта, пишущий привод и т. п.) с некими новыми возможностями. Естественно, оно продается несколько дороже аналогичных более примитивных устройств. Спустя некоторое время конкуренты догоняют, тогда это устройство снимается с продажи, а на рынок выбрасывается аналогичная модель с еще более расширенной функциональностью — по той же цене или еще несколько дороже. Другой вариант той же политики — одновременно выпустить линейку похожих устройств разного класса (например, по быстродействию) с ценой, отличающейся иногда в разы (характерно, например, для видеокарт). И простодушному покупателю невдомек, что разработан был лишь самый дорогой и «продвинутый» вариант, а все более дешевые отличаются лишь программой — «прошивкой» (ну, иногда еще отсутствием некоторых микросхем на плате). Выпуская модели с искусственно урезанной функциональностью, производитель имеет значительно бóльшую суммарную прибыль, т. к. фактически одно и то же устройство может продаваться годами без значительных инвестиций в его доработку, а также охватить все секторы рынка, от самых простых до эксклюзивных. Во многих случаях пользователь сам может превратить дешевое устройство в более дорогое, сменив лишь микропрограмму — классическим примером сезона 2005/2006 годов стала линейка пишущих приводов NEC 3540, 3550, 4550 и др.

С другой стороны, чем больше функций устройства реализовано программно, а не аппаратно, тем оно, во-первых, дешевле (сравните цены аппаратных модемов и софт-модемов), а во-вторых, тем легче исправляются недочеты при проектировании. Мой знакомый однажды, сменив прошивку дешевенькой «фотомыльницы», обнаружил, что она приобрела способность снимать чуть ли не в лунном свете, хотя до этого не вытягивала даже комнатное освещение!

Ну а теперь перейдем к рассмотрению того, как работает микропроцессор «вообще».

Как работает микропроцессор

Для того чтобы понять, как работает микропроцессор (МП), зададим себе вопрос: а как он должен работать? Есть теория (в основном созданная постфактум: после того, как первые ЭВМ были уже построены и функционировали), которая указывает, как именно строить алгоритмы, и что процессор в соответствии с этим должен делать. Мы, естественно, углубляться в это не будем, просто констатируем, что любой алгоритм есть последовательность неких действий, записанных в виде набора последовательно выполняемых команд (инструкций, операторов). При этом среди таких команд могут встречаться переходы, которые в некоторых случаях нарушают исходную последовательность выполнения операторов строго друг за другом. Среди прочих должны быть также команды ввода и вывода данных (программа должна как-то общаться с внешним миром?), а также команды выполнения арифметических и логических операций.

Команды необходимо где-то хранить, поэтому неотъемлемой частью всей системы должно быть устройство памяти программ. Где-то надо складывать и данные, как исходные, так и результаты работы программы, поэтому должно быть устройство памяти данных. Поскольку команды и данные в конечном счете все равно есть числа, то память может быть общая, только надо уметь отличать, где именно у нас команды, а где — данные. Это есть один из принципов фон Неймана, хотя и в микроконтроллерах, о которых мы будем говорить в дальнейшем, традиционно используют не фон-неймановскую, а так называемую *гарвардскую* архитектуру, когда память данных и программ разделены (это разделение, впрочем, может в определенных пределах нарушаться). Процессор, построенный по фон Нейману, более универсальный, например, он позволяет без особых проблем наращивать память и более эффективно ее перераспределять прямо по ходу работы. В то же время микроконтроллерам подобная гибкость не особенно требуется, на их основе, как правило, строятся узлы, выполняющие конкретную задачу и работающие по конкретной программе, так что нужную конфигурацию системы ничего не стоит предусмотреть заранее.

МП и МК

Кстати, а почему мы все время говорим так: то микропроцессоры, то микроконтроллеры? Микроконтроллер (МК) отличается от микропроцессора тем, что он предназначен для управления другими устройствами, и поэтому имеет встроенную развитую систему ввода/вывода, но, как правило, относительно более слабое арифметико-логическое устройство (АЛУ). Микроконтроллерам очень хорошо подходит термин, который в советское время имел, правда, несколько иное значение — «микроЭВМ», еще точнее звучит английское «computer-on-chip». В самом

деле, для построения простейшего вычислительного устройства, которое могло бы выполнять что-то полезное, обычный микропроцессор, от i4004 до Pentium и Core Duo, приходится дополнять памятью, BIOS, устройствами ввода/вывода, контроллером прерываний, тактовым генератором с таймерами и т. п. — всем тем, что сейчас стало объединяться в т. н. «чипсеты». «Голый» МП способен выполнить только одно: правильно включиться, ему даже программу загрузки неоткуда взять.
В то же время для МК микропроцессор — это только ядро, даже не самая большая часть кристалла. Для построения законченной системы на типовом МК не требуется вообще ничего, кроме источника питания и периферийных исполняющих устройств, которые позволяли бы человеку определить, что система работает. Обычный МК может без дополнительных компонентов общаться с другими МК, внешней памятью, специальными микросхемами (вроде часов реального времени или флэш-памяти), с компьютером, управлять небольшими (а иногда — и большими) матричными панелями, к нему можно напрямую подключать датчики физических величин (в том числе — чисто аналоговые, АЦП тоже входят в МК), кнопки, клавиатуры, светодиоды и индикаторы, короче — в микроконтроллерах сделано все, чтобы как можно меньше приходилось паять и задумываться над подбором элементов. За это приходится расплачиваться пониженным быстродействием (которое, впрочем, не так и требуется в типовых задачах для МК) и некоторым ограничением в отдельных функциях — по сравнению с универсальными, но в сотни раз более дорогими и громоздкими системами на «настоящих» МП. Вы можете мне не поверить, но процессоры для ПК, о которых мы столько слышим, занимают в общем количестве выпускаемых процессоров лишь 6%, — остальные 94% составляют микроконтроллеры различного назначения.

В соответствии со сказанным основной цикл работы процессора должен быть таким: выборка очередной команды (из памяти), при необходимости выборка исходных данных для нее, выполнение команды, размещение результатов в памяти (опять же если это необходимо). Вся работа в этом цикле должна происходить автоматически по командам некоторого устройства управления, содержащего тактовый генератор — системные часы, по которым все синхронизируется. Кроме того, где-то это все должно происходить — складирование данных, кода команды, выполнение действий и т. п., так что процессор должен содержать некий набор рабочих регистров (по сути — очень небольшую по объему сверхбыструю память), определенным образом связанных между собой, а также с устройством управления и АЛУ, которое неизбежно должно присутствовать.

Решающую роль в работе процессора играет счетчик команд. Он автоматически устанавливается на нуль в начале работы, что соответствует первой команде, и автоматически же инкрементируется (т. е. увеличивается на единицу) с каждой выполненной командой. Если по ходу дела порядок команд нарушается, например, встречается команда перехода (ветвления), то в счетчик

загружается соответствующий адрес команды от начала программы. К счастью, нам самим фактически не придется иметь дело со счетчиком команд, потому что все указания на этот счет содержатся в самой команде, и процессор все делает автоматически.

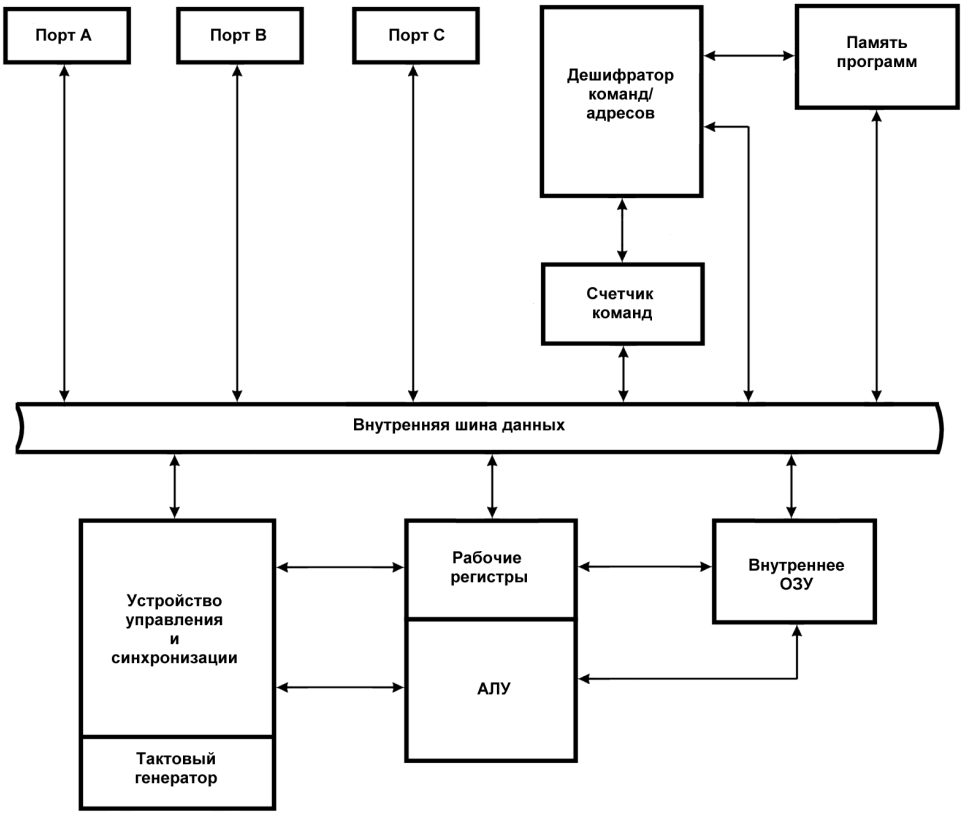


Рис. 11.2. Блок-схема типичного микропроцессора

Блок-схема типичного микропроцессора показана на рис. 11.2. Здесь мы включили в состав процессора память программ, которая у ПК-процессоров находится всегда отдельно — сами знаете, программы какого объема бывают в персональных компьютерах. В то же время в большинстве современных микроконтроллеров постоянное запоминающее устройство (ПЗУ) для программ входит в состав чипа и обычно составляет от 1 до 8 кбайт (но есть модели и со 256 кбайтами встроенной памяти!), чего для подавляющего большинства применений вполне достаточно, а если вдруг не хватит, то всегда можно подключить внешнюю память. Впрочем, внутреннее оперативное

запоминающее устройство (ОЗУ) того или иного объема имеется во всех современных процессорах, у процессоров для ПК это называется кэш-памятью (иногда — нескольких уровней). Типичный размер ОЗУ данных у микроконтроллеров — от 256 байт до 1 кбайта.

Подробности

В первых моделях микропроцессоров (включая и Intel-процессоры для ПК — от 8086 до 386) процессор выполнял команды строго последовательно: загрузить команду, определить, что ей нужны операнды, загрузить эти операнды (по адресу регистров, которые их должны содержать; адреса эти, как правило, хранятся сразу после собственно кода команды, или определены заранее), потом проделать нужные действия, складировать результаты… До нашего времени дошла архитектура суперпопулярных еще недавно микроконтроллеров 8051, выпускающихся и по сей день различными фирмами (Atmel, Philips), которые выполняли одну команду аж за 12 тактов (в современных системах, впрочем, это число несколько меньше). Для ускорения работы стали делить такты на части (например, срабатывать по переднему и заднему фронтам), но действительный прорыв произошел с внедрением конвейера. Со времен Генри Форда известно, что производительность конвейера зависит только от времени выполнения самой длинной операции — если поделить команды на этапы и выполнять их одновременно разными аппаратными узлами, то можно добиться существенного ускорения (хотя и не во всех случаях). Так, в рассматриваемых далее Atmel AVR конвейер двухступенчатый: когда очередная команда загружается и декодируется, предыдущая уже выполняется и пишет результаты. В AVR это позволило выполнять большинство команд за один такт (кроме команд ветвления, о чем подробнее будет рассказываться в *главе 13*).

Главное устройство в МП, которое связывает все узлы в единую систему, — внутренняя шина данных. По ней все остальные устройства обмениваются сигналами. Например, если МП требуется обратиться к внешней памяти, то при исполнении соответствующей команды на шину данных выставляется необходимый адрес, от устройства управления поступает через нее же запрос на обращение к нужным портам ввода/вывода. Если порты готовы, адрес поступает на выходы портов (т. е. на соответствующие выводы контроллера), затем по готовности принимающий порт выставляет на шину принятые из внешней памяти данные, которые загружаются в нужный регистр, после чего шина данных свободна. Для того чтобы все устройства не мешали друг другу, все это строго синхронизировано, при этом каждое устройство имеет, во-первых, собственный адрес, во-вторых, может находиться в трех состояниях: работать на ввод, на вывод или находиться в третьем состоянии, не мешая другим работать.

Под разрядностью МП обычно понимают разрядность чисел, с которыми работает АЛУ, соответственно, такую же разрядность имеют и рабочие регистры. Например, все ПК-процессоры от i386 до последних инкарнаций Pentium были 32-разрядными, последние модели от Intel и AMD стали 64-разрядными.

Большинство микроконтроллеров общего назначения 8-разрядные, но есть и 16- и 32-разрядные. При этом внутренняя шина данных может иметь и больше разрядов, например, чтобы одновременно передавать и адреса и данные.

Распределение рынка МК в первые годы тысячелетия было таким: немного меньше половины выпускаемых изделий составляют 8-разрядные кристаллы, а вторую половину поделили между собой 16- и 32-разрядные, причем доля последних неуклонно растет за счет 16-разрядных. Выпускаются даже 4-разрядные, потомки первого i4004, которые занимают не более 10% рынка, но что любопытно, эта доля снижается очень медленно.

Заметки на полях

Обычно тактовая частота универсальных МК невелика (хотя типичному инженеру 1980-х, когда ПК работали на частотах не выше 6 МГц, она показалась бы огромной) — порядка 8—16 МГц, иногда до 24 МГц или несколько более. И это всех устраивает: дело в том, что обычные МК и не предназначены для разработки быстродействующих схем. Если требуется быстродействие, то используется другой класс интегральных схем — ПЛИС, «программируемые логические интегральные схемы». Простейшая ПЛИС представляет собой набор никак не связанных между собой логических элементов (более сложные могут включать в себя и некоторые законченные узлы, вроде триггеров и генераторов), которые в процессе программирования такого чипа соединяются в нужную схему. Комбинационная логика работает гораздо быстрее тактируемых контроллеров, и для построения различных логических схем в настоящее время применяют только ПЛИС, от проектирования на «рассыпухе» в массовых масштабах уже давно отказались. Еще одно преимущество ПЛИС — статическое потребление энергии для некоторых серий составляет единицы микроватт, в отличие от МК, которые во включенном состоянии потребляют всегда (если не находятся в режиме энергосбережения). В совокупности с более универсальными и значительно более простыми в обращении, но менее быстрыми и экономичными микроконтроллерами, ПЛИС составляют основу большинства массовых электронных изделий, которые вы видите на прилавках. В этой книге мы, конечно, рассматривать ПЛИС не будем — в любительской практике, в основном из-за дороговизны соответствующего инструментария и высокого порога его освоения, они не используются, и для конструирования одиночных экземпляров приборов даже для профессиональных применений нецелесообразны. А вот если вы закажете разработку некоего прибора профессиональной «конторе», имеющей нужные инструменты и разработчиков с соответствующей квалификацией — почти всегда получите что-нибудь на базе ПЛИС, потому что в конечном итоге так оказывается дешевле.

Если подробности внутреннего функционирования МП нас не очень волнуют (достаточно иметь общее представление о структуре микропроцессорного ядра, чтобы понимать, что именно происходит при выполнении команд), то обмен с внешней средой нас как раз интересует во всех деталях. Для этого служат порты ввода/вывода (I/O-port, от Input/Output). В этом термине имеется некоторая неопределенность, т. к. «порт ввода/вывода» в МК с точки зрения его внутреннего устройства обозначает прежде всего некий регистр для

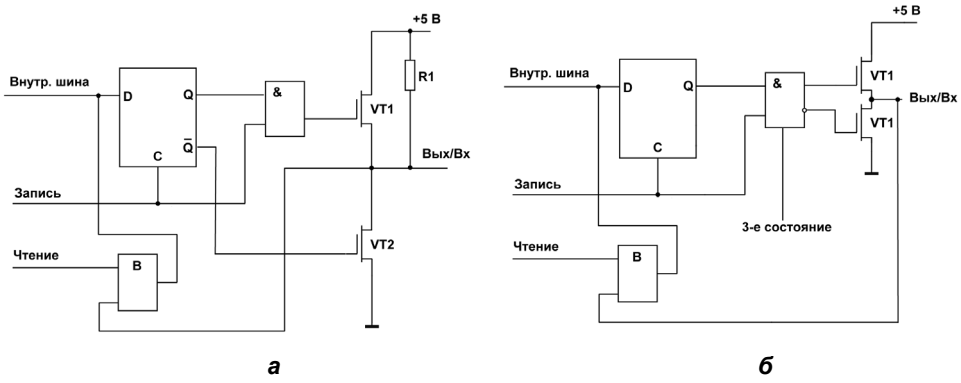
доступа к компонентам, внешним по отношению к вычислительному ядру. А это все узлы, которыми непосредственно управляет пользователь (от таймеров и последовательных портов до регистра флагов и управления прерываниями). Кроме разве что ОЗУ, доступ к которой обеспечивается специальными командами, все остальное в контроллере управляется через порты ввода/вывода.

Однако точно так же называются и внешние порты ввода/вывода, для обмена с «окружающей средой» (управляются они, естественно, внутренними портами ввода/вывода). На схеме рис. 11.2 они показаны в количестве трех (A, B и C). В разных МП их может быть и больше, и меньше. Еще важнее число выводов этих портов, которое чаще всего совпадает с разрядностью процессора (но не всегда, как это было у 8086, который имел внутреннюю 16-разрядную структуру, а внешне выглядел 8-разрядным). Если мы заставим 8-разрядные порты «общаться», например, с внешней памятью, то на двух из них можно выставить 16-разрядный адрес, а на оставшемся — принимать данные. А как быть, если портов два или вообще один? (К примеру, в микроконтроллере Atmel AVR 2313 портов формально два, но один усеченный, так что общее число линий составляет 15.) Для этого все внешние порты в МП всегда двунаправленные. Скажем, если портов два, то можно сначала выставить адрес, а затем переключить их на вход и принимать данные. Естественно, для этого порты должны позволять работу на общую шину, т. е. либо иметь третье состояние, либо выход с общим коллектором для объединения в «монтажное ИЛИ».

Варианты для обоих случаев организации выходной линии порта показаны на рис. 11.3, где приведены упрощенные схемы выходных линий микроконтроллеров семейства 8048 — широко когда-то использовавшегося предшественника популярного МК 8051 (например, 8048 был выбран в качестве контроллера клавиатуры в IBM PC). В самом 8051 построение портов несколько сложнее (в частности, вместо резистора там полевой транзистор), но для уяснения принципов работы это несущественно.

По первому варианту (рис. 11.3, *а*) в МК 8048 построены порты 1, 2 (всего там три порта). Когда в порт производится запись, то логический уровень поступает с прямого выхода защелки на статическом D-триггере на вход схемы «И», а с инверсного — на затвор транзистора VT2. Если этот уровень равен логическому нулю, то транзистор VT1 заперт, а VT2 открыт, на выходе также логический нуль. Если уровень равен логической единице, то на время действия импульса «Запись» транзистор VT1 открывается, а транзистор VT2 запирается (они одинаковой полярности). Если на выходе присутствует емкость (а она всегда имеется в виде распределенной емкости проводников и емкости входов других компонентов), то через VT1 протекает достаточно

большой ток заряда этой емкости, позволяющий сформировать хороший фронт перехода из «0» в «1». Как только импульс «Запись» заканчивается, оба транзистора отключаются, и логическая единица на выходе поддерживается резистором R1. Выходное сопротивление открытого транзистора VT1 примерно 5 кОм, а резистора — 50 кОм. Любое другое устройство, подключенное к этой шине, при работе на выход может лишь либо поддержать логическую единицу, включив свой подобный резистор параллельно R1, либо занять линию своим логическим нулем — это, как видите, и есть схема «монтажное ИЛИ». При работе на вход состояние линии просто считывается со входного буфера (элемент «В» на рис. 11.3, *а*).



а *б*

Рис. 11.3. Упрощенные схемы портов ввода/вывода МК 8048:
а — портов 1 и 2; *б* — порта 0

Второй же вариант, по которому устроен порт 0 (рис. 11.3, *б*), — это обычный выходной каскад КМОП с третьим состоянием, т. е. такой порт может работать на выход, только полностью занимая линию, остальные подключенные к линии устройства при этом должны «смиренно внимать» монополисту, воспринимая сигналы. Это обычно не создает особых трудностей и схемотехнически даже предпочтительно (ввиду симметрии выходных сигналов и высокого сопротивления для входных). Единственная сложность возникает при сопряжении такого порта с линией, работающей по первому варианту, т. к. при логической единице на выходе могут возникнуть электрические конфликты, если кто-то попытается выдать в линию логический нуль. Для обеспечения работы трехстабильного порта по схеме «монтажное ИЛИ» (в том числе для их параллельной работы) применяют хитрый прием: всю линию «подтягивают» к напряжению питания с помощью внешнего резистора (во многих МК существует встроенный отключаемый резистор, установленный аналогично R1 в схеме рис. 11.3, *а*), и нормальное состояние всех

участвующих трехстабильных портов — работа на вход в третьем состоянии, тогда на линии всегда будет логическая единица. На выход же линию переключают только когда надо выдать логический нуль, в этом случае, даже при одновременной активности нескольких портов, конфликтов не возникнет.

Лечение амнезии

В 1965 г. в Иллинойском университете был запущен один из самых передовых компьютеров по тому времени — ILLIAC-IV. Это был первый компьютер, в котором была применена быстрая память на микросхемах — каждый чип (производства Fairchild Semiconductor) имел емкость 256 бит, а всего было набрано 1 Мбайт. Стоимость этой памяти составила ощутимую часть от всей стоимости устройства, обошедшегося заказчику — NASA — в $31 млн. Через 10 лет один из первых персональных компьютеров Altair 8800 (1975 г.), продававшийся в виде набора «сделай сам», при стоимости порядка $500 имел всего 256 байт (именно байт, а не килобайт) памяти. В том же году для распространения языка Basic for Altair Биллом Гейтсом и Полом Алленом была создана фирма, получившая первоначальное название Micro-Soft. Одна из самых серьезных проблем, которую пришлось решать — нехватка памяти, потому что созданный ими интерпретатор Basic требовал аж 4 кбайтов!

Проблема объемов памяти и ее дороговизна преследовала разработчиков до самого последнего времени, еще в конце 90-х стоимость памяти для ПК можно было смело прикидывать на уровне $1/Мбайт, что при требовавшихся уже тогда для комфортной работы объемах ОЗУ порядка 128—256 Мбайт могло составлять значительную часть стоимости устройства. Сейчас гигабайтом памяти в настольном ПК и даже ноутбуке никого не удивишь, к тому же все современные ОС «умеют» автоматически дополнять недостающий объем ОЗУ за счет дискового пространства. Это привело, в частности, к кардинальным изменениям в самом подходе к программированию: если еще при программировании под DOS о компактности программ и экономии памяти в процессе работы нужно было специально заботиться, то теперь это практически не требуется.

Но в программировании для микроконтроллеров это все еще не так. Хотя гейтсовский интерпретатор Basic «влезет» в большинство современных однокристальных МК, но экономная программа легче отлаживается (а значит, содержит меньше ошибок) и быстрее выполняется. Три-четыре такта, потерянных на вызове процедуры, могут стать причиной какой-нибудь трудновылавливаемой ошибки времени выполнения, например, если за это время про-

изойдет вызов прерывания. Поэтому память в МК стоит экономить, даже если вы располагаете заведомо достаточным ее объемом.

Согласно упоминавшимся принципам фон Неймана, которые до сих пор являются основополагающими для разработчиков компьютерных систем, память должна быть организована иерархически, от памяти малого объема, но с высоким быстродействием, до медленной памяти большой емкости. Именно так и проектируют современные компьютеры, которые содержат очень быструю кэш-память на одном кристалле с процессором (а иногда два или три уровня такой памяти), потом идет быстродействующее ОЗУ (оперативное запоминающее устройство, та самая память, которая указывается в характеристиках ПК), а затем более медленные устройства типа жестких дисков или CD- и DVD-ROM.

Далее мы рассмотрим основные разновидности памяти, используемые как в составе микроконтроллеров, так и во внешних узлах. И начнем с того, что попробуем сами сконструировать устройство долговременной памяти — ПЗУ (постоянное запоминающее устройство). Как мы увидим, любая память в принципе есть не что иное, как преобразователь кодов.

Изобретаем простейшую ROM

Всем известное благодаря распространению оптических дисков сокращение ROM — Read Only Memory — и есть не что иное, как «западное» название ПЗУ. На самом деле это название («память только для чтения») не очень точно характеризует суть дела, отечественный термин «постоянное запоминающее устройство» более корректен, самое же правильное называть такую память «энергонезависимой». ПЗУ отличается от других типов памяти не тем, что его можно только читать, а записывать нельзя, а тем, что информация в нем не пропадает при выключении питания. Сама по себе невозможность записи теперь нехарактерна даже для компакт-дисков, однако название ROM сохранилось.

Тем не менее первыми разновидностями ПЗУ, изобретенными еще в 1956 году, были именно нестираемые кристаллы, которые носят наименование OTP ROM (One-Time Programmable ROM, «однократно программируемое ПЗУ»). До недавнего времени на них делали память программ МК для удешевления серийных устройств. Вы отлаживаете программу на перезаписываемой памяти, а в серию пускаете приборы с «прожигаемой» OTP ROM. И лишь в последние годы «прожигаемая» память стала постепенно вытесняться более удобной flash-памятью, когда последняя подешевела настолько, что смысл в использовании одноразовых кристаллов пропал. Они продолжают

выпускаться лишь за счет инерции производства: в 2007 году доля однократно программируемых и масочных (т. е. программируемых прямо на производстве) микроконтроллеров составит, по прогнозам, не более 1/4 всех выпускаемых чипов.

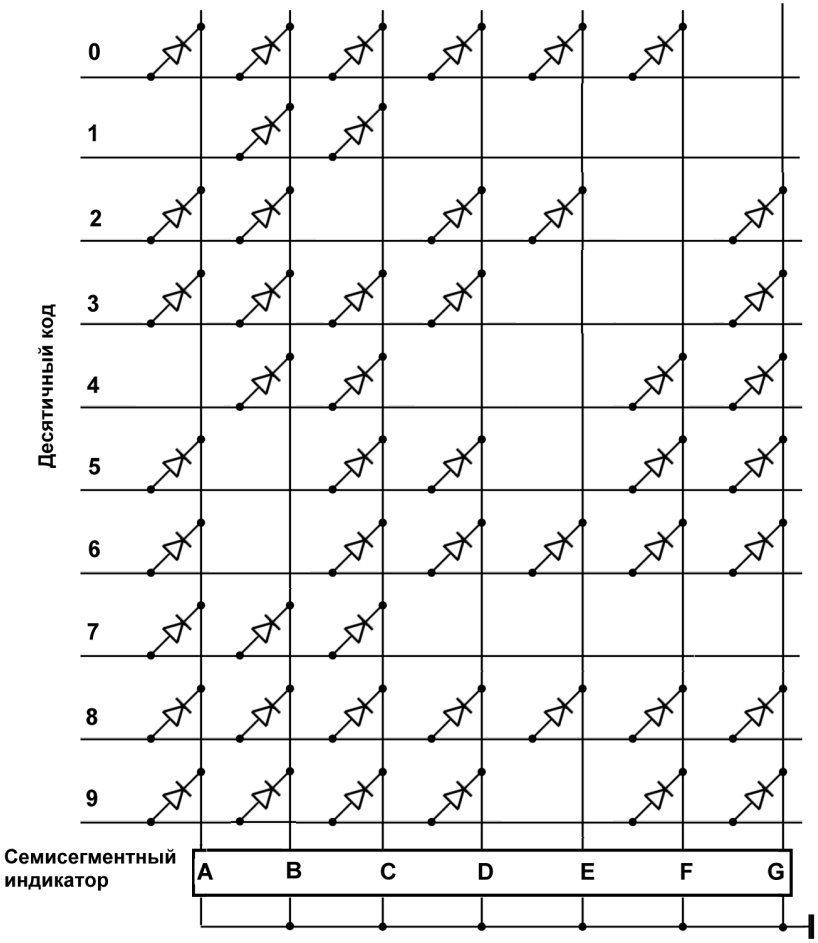


Рис. 11.4. Простейшее ПЗУ — преобразователь кода

Мы сконструируем подобие «прожигаемого» ПЗУ с помощью диодов. Простейший вариант такого ПЗУ представлен на рис. 11.4. В данном случае он представляет собой не что иное, как преобразователь из десятичного кода в семисегментный. Если на входе поставить дешифратор типа 561ИД1, то мы получим аналог микросхемы 561ИД5. Представьте себе, что первоначально

на всех пересечениях между строками и столбцами диоды присутствовали — это вариант незаполненной памяти, в которой записаны все единицы. Затем мы взяли и каким-то образом (например, подачей высокого напряжения) разрушили те диоды, которые нам не нужны, в результате чего получили нужную конфигурацию. Эта схема не содержит активных элементов и потому возможности ее ограничены, например, выходы устройства, подающего активный высокий уровень по входным линиям, должны «тащить» всю нагрузку по зажиганию сегментов. Обычная микросхема ПЗУ построена на транзисторных ячейках и поэтому без всяких хитростей принимает и выдает обычные логические уровни. К тому же она включает в себя и дешифрирующую логику, поэтому на вход подается двоичный, а не десятичный код.

Постойте, а при чем тут ПЗУ вообще? Дело в том, что любое ПЗУ можно представить, как универсальный преобразователь кодов, если рассматривать входной код, как адрес ячейки, а код, получающийся на выходе — как содержимое этой ячейки. Причем удобство состоит в том, что изначально в ПЗУ не записано ничего (одни нули или единицы), и мы можем реализовать на нем любую логическую функцию, все зависит только от емкости. В том числе, такую простую, как преобразователь кодов. Или такую сложную, как операционная система Windows. Последнее мы каждый раз и делаем, когда устанавливаем Windows на компьютер, причем в качестве ПЗУ выступает жесткий диск. Из этого примера отчетливо видно, что каким бы сложным ни был алгоритм, он все равно в конечном итоге сводится к совокупности однозначных логических уравнений, которые можно реализовать как через ПЗУ, так и с помощью устройства памяти любого другого типа.

Общее устройство памяти

Общее устройство однобитной ячейки памяти (любого типа) показано на рис. 11.5. Из нее видно, что память всегда имеет матричную структуру. В данном случае матрица имеет $8 \times 8 = 64$ ячейки. На рис. 11.5 показано, как производится вывод и загрузка информации в память с помощью мультиплексоров/демультиплексоров (вроде 561КП2, см. *главу 9*). Код, поступающий на мультиплексор слева (x_3—x_5) подключает к строке с номером, соответствующим этому коду, активирующий уровень напряжения (это может быть логическая единица или ноль, неважно). Код на верхнем мультиплексоре (x_0—x_2) выбирает аналогичным образом столбец, в результате к выходу этого мультиплексора подключается ячейка, стоящая на пересечении выбранных строки и столбца.

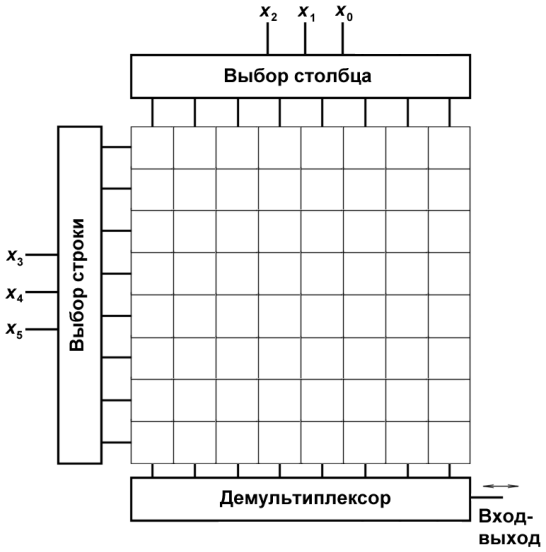


Рис. 11.5. Общее устройство ЗУ с однобитным выходом

Легко заметить, что сама по себе организация матрицы при таком однобитном доступе для внешнего мира не имеет значения. Если она будет выглядеть, как 4×16 или 32×2 или даже 64×1 — в любом случае код доступа (он называется *адресным* кодом) будет 6-разрядным, а выход один-единственный. Поэтому всем таким ЗУ приписывается организация N×1 бит, где N — общее число битов. Для того чтобы получить байтную организацию, надо просто взять 8 таких микросхем и добавить к адресной линии еще три разряда, которыми через отдельный мультиплексор можно управлять выборкой одной из этих микросхем (для этого каждая такая микросхема имеет специальный вывод, называемый «выбор кристалла» — chip select, или CS). В данном примере мы получим в сумме 9 адресных разрядов, что соответствует емкости памяти ($64 \times 8 = 512$ бит или 2^9). Один из битов можно использовать при этом для контроля четности, так что у нас получается хорошая модель типового модуля емкостью 256 байт, вроде тех, что были в упомянутом ILLIAC-IV. Большинство выпускаемых интегральных ЗУ также сложены из таких отдельных однобитных модулей (только в наше время уже значительно большей емкости) и имеют 8 или 16 параллельных выходов, но бывают кристаллы и с последовательным (побитным) доступом.

В качестве примера можно привести, скажем, ПЗУ с организацией 64К×16 типа AT27C1024 фирмы Atmel. Это однократно программируемое КМОП ПЗУ с напряжением питания 5 В и емкостью 1024 Мбита, что составляет

128 кбайт или 64 К двухбайтных слов (как мы увидим, такая организация очень удобна в качестве внешней памяти программ в контроллерах той же Atmel). Следует отметить, что в области микросхем памяти сложилась счастливая ситуация, когда все они, независимо от производителя и даже технологии, совпадают по выводам, разводка которых зависит только от организации матрицы (даже, как правило, не от объема!) и, соответственно, от применяемого корпуса (в данном случае — DIP-40). Для разных типов (RAM, ROM, EEPROM и т. д.) различается разводка выводов, управляющих процессом программирования, но можно спокойно заменять одну микросхему на другую (с той же организацией и, соответственно, в таком же корпусе) без переделки платы. Разводка выводов AT27C1024 показана на рис. 11.6.

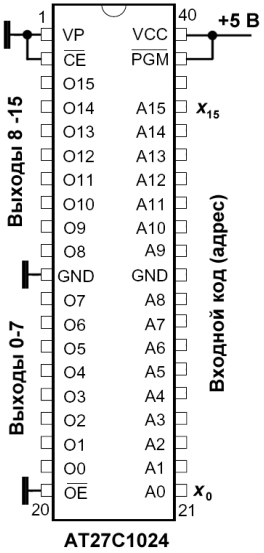


Рис. 11.6. Разводка выводов AT27C1024

RAM

Традиционное название энергозависимых типов памяти, как и в случае ROM, следует признать довольно неудачным. RAM значит Random Access Memory, т. е. «память с произвольным доступом», по-русски это звучит как ЗУПВ — «запоминающее устройство с произвольной выборкой». Главным же признаком класса является не «произвольная выборка», а то, что при выключении питания память стирается. EEPROM (о которой далее), к примеру, тоже

позволяет произвольную выборку и при записи, и при чтении. Но так сложилось исторически, и не нам нарушать традиции.

Устройства RAM делятся на две больших разновидности — статические и динамические ЗУПВ. Простейшее статическое ЗУПВ (SRAM, от слова «static») — это обычный триггер. И «защелки» из микросхемы 561ТМ3, и регистры типа 561ИР2, и даже счетчик с предзагрузкой типа ИЕ11 (см. *главу 9*), — все это статические ЗУПВ с различными дополнительными функциями или без них. Регистры и доступная пользователю область ОЗУ (оперативного запоминающего устройства) микроконтроллеров, — все они также относятся к классу SRAM, и мы с ними еще познакомимся довольно близко.

По счастью, с динамическими разновидностями RAM (DRAM) нам в схемотехническом плане иметь дело не придется, но ввиду практической важности этой разновидности (на DRAM построена вся оперативная память компьютеров) стоит остановиться на ней подробнее. Устройство ячейки обычной DRAM показано на рис. 11.7, из которого видно, что ячейка состоит всего из одного транзистора и одного конденсатора Последний на схеме (рис. 11.7, *а*) выглядит маленьким, но на самом деле занимает места во много раз больше транзистора (рис. 11.7, *б*), только в основном вглубь кристалла. Потому ячейки DRAM можно сделать очень малых размеров, а, следовательно, упаковать их много на один кристалл, не теряя быстродействия.

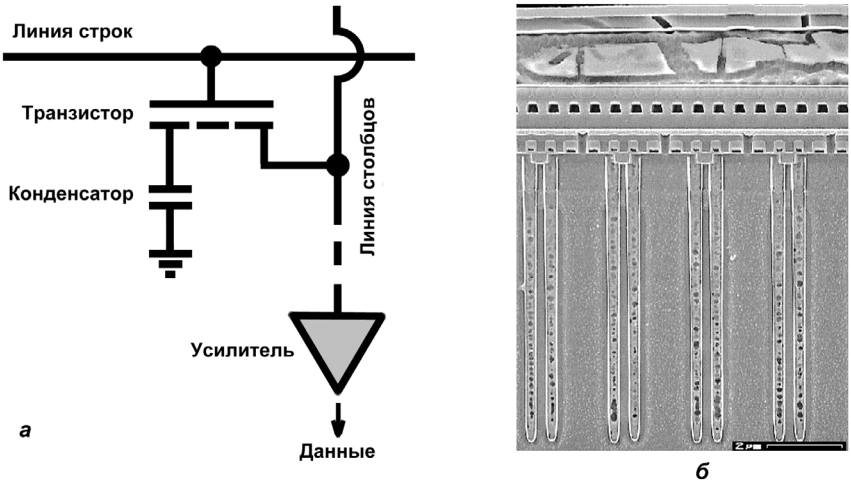


Рис. 11.7. Устройство ячейки DRAM:
а — схематическое устройство;
б — микрофотография среза кристалла DRAM
(вытянутые вниз структуры — накопительные конденсаторы)

Как происходит чтение данных с такой ячейки? Для этого вы подаете высокий уровень на линию строк (см. рис. 11.7), транзистор открывается и заряд, хранящийся на конденсаторе данной ячейки, поступает на вход усилителя, установленного на выходе столбца. Отсутствие заряда на обкладках соответствует логическому нулю на выходе, а его наличие — логической единице. Обратите внимание, что подача высокого уровня на линию строк откроет все транзисторы выбранной строки, и данные окажутся на выходе усилителей по всем столбцам сразу. Естественно, при этом все подключенные конденсаторы почти немедленно разрядятся (если они были заряжены), отчего процедура чтения из памяти обязана заканчиваться регенерацией данных (как оно в действительности и происходит, причем совершенно автоматически).

На практике регенерация в первых IBM PC и заключалась в осуществлении «фиктивной» операции чтения данных каждые 15 мкс с помощью системного таймера. Естественно, в таком решении было много подводных камней. Во-первых, регенерация всей памяти занимает много времени, в течение которого ПК неработоспособен. Потому-то сигнал на регенерацию и подавался с такой большой частотой, ведь каждый раз проверялась всего 1/256 памяти, так что полный цикл восстановления занимал около 3,8 мс. Во-вторых, такое решение потенциально опасно: любая зловредная программа спокойно может попросту остановить системный таймер, отчего компьютер уже через несколько миллисекунд обязан впасть в полный «ступор». И все современные микросхемы DRAM занимаются восстановлением данных самостоятельно, да еще и так, чтобы не мешать основной задаче — процессам чтения/записи.

Впервые принцип DRAM — хранение информации на конденсаторах с периодической регенерацией — применил еще Дж. Атанасов в самом первом электронном компьютере ABC (1941 г.). А зачем вообще нужна регенерация? Ввиду микроскопических размеров и, соответственно, емкости конденсатора в ячейке DRAM записанная информация хранится всего лишь сотые доли секунды. Несмотря на высококачественные диэлектрики с огромным электрическим сопротивлением, заряд, состоящий в рядовом случае всего из нескольких сотен, максимум тысяч электронов, успевает утечь так быстро, что вы и глазом моргнуть не успеете.

Огромный плюс DRAM — простота и дешевизна. В отличие от нее, ячейка SRAM, как вы знаете, представляет собой D-триггер, и содержит много логических элементов, занимая большую площадь кристалла. Потому SRAM много дороже, но зато не требует никакой регенерации. Фирма Dallas (ныне объединенная с MAXIM) одно время выпускала микросхемы энергонезависимой памяти (и некоторые другие устройства на их основе), представлявшие собой обычную SRAM со встроенной прямо в чип литиевой батарейкой.

EPROM, EEPROM и Flash

На заре возникновения памяти, сохраняющей данные при отключении питания (EPROM, Erasable Programmable ROM — «стираемая/программируемая ROM». По-русски иногда называют ППЗУ — «программируемое ПЗУ»), основным типом ее была память, стираемая ультрафиолетом: UV-EPROM (Ultra-Violet EPROM, УФ-ППЗУ). Причем часто приставку UV опускали, т. к. всем было понятно, о чем речь — альтернативой УФ-ППЗУ были фактически только однократно программируемые кристаллы OTP ROM, которые имелись обычно в виду под сокращениями ROM (или ПЗУ) просто, без добавлений. Микроконтроллеры с УФ-памятью программ были распространены еще в середине 1990-х. В рабочих образцах подобных устройств кварцевое окошечко заклеивали кусочком черной ленты, т. к. информация в UV-EPROM медленно разрушалась на солнечном свету.

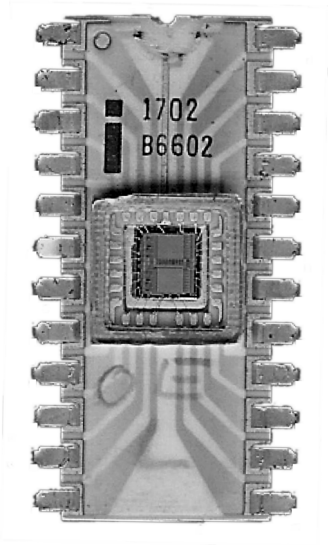


Рис. 11.8. Первая микросхема UV-EPROM фирмы Intel (1971) позволяла записать 256 байт информации

На рис. 11.9 показано устройство элементарной ячейки подобной EPROM, которая лежит в основе всех современных типов энергонезависимой памяти. Если исключить из нее то, что обозначено надписью «плавающий затвор», то мы получим самый обычный полевой транзистор — точно такой же, как тот, что входит в ячейку DRAM на рис. 11.7. Если подать на управляющий затвор такого транзистора положительное напряжение, то он откроется, и через него

потечет ток (это считается состоянием «логической единицы»). На рис. 11.9 вверху изображен как раз такой случай, когда плавающий затвор не оказывает никакого влияния на работу ячейки, например, такое состояние характерно для чистой flash-памяти, в которую еще ни разу ничего не записывали.

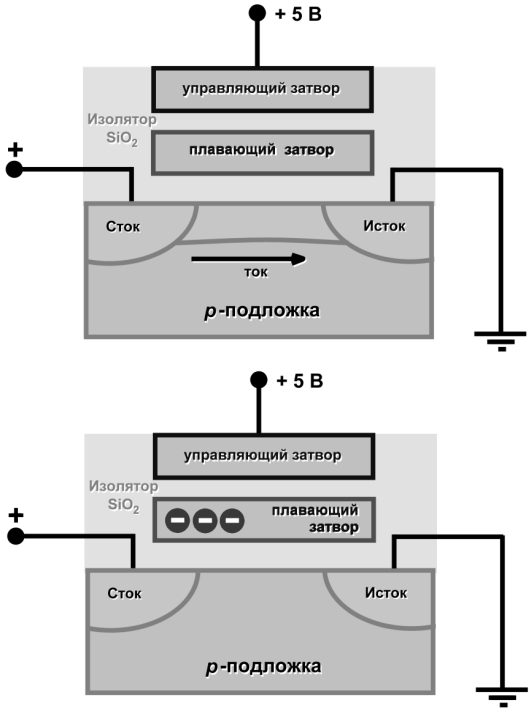


Рис. 11.9. Устройство элементарной ячейки EPROM

Если же мы каким-то образом (каким — поговорим отдельно) ухитримся разместить на плавающем затворе некоторое количество зарядов — свободных электронов, которые показаны на рис. 11.9 внизу в виде темных кружочков со значком минуса, то они будут экранировать действие управляющего электрода, и такой транзистор вообще перестанет проводить ток. Это состояние «логического нуля».

Замечание

Строго говоря, в NAND-чипах (о которых далее) логика обязана быть обратной: если в обычной EPROM запрограммированную ячейку вы не можете открыть подачей считывающего напряжения, то там наоборот — ее нельзя запереть снятием напряжения. Поэтому, в частности, чистая NAND-память выдает все нули, а не единицы, как EPROM. Но это нюансы, которые не меняют суть дела.

Так как плавающий затвор потому так и называется, что он «плавает» в толще изолятора (двуокиси кремния, SiO_2), то сообщенные ему однажды заряды в покое никуда деваться не могут. И записанная таким образом информация может храниться десятилетиями (до последнего времени производители обычно давали гарантию на 10 лет, но на практике в обычных условиях время хранения значительно больше).

Осталось всего ничего — придумать, как размещать заряды на изолированном от всех внешних влияний плавающем затворе. И не только размещать — ведь иногда память и стирать приходится, потому должен существовать способ их извлекать оттуда. В UV-EPROM слой окисла между плавающим затвором и подложкой был достаточно толстым (если величину 50 нм можно охарактеризовать словом «толстый», конечно), и работало все это довольно «грубо». При записи на управляющий затвор подавали достаточно высокое положительное напряжение — иногда до 36—40 В, а на сток транзистора — небольшое положительное. При этом электроны, которые двигались от истока к стоку, настолько ускорялись полем управляющего электрода, что просто «перепрыгивали» барьер в виде изолятора между подложкой и плавающим затвором. Такой процесс называется еще «инжекцией горячих электронов».

Ток заряда при этом достигал миллиампера — можете себе представить, каково было потребление всей схемы, если в ней одновременно заряжать хотя бы несколько тысяч ячеек. И хотя такой ток требовался на достаточно короткое время (впрочем, с точки зрения быстродействия схемы не такое уж и короткое — миллисекунды), но это было крупнейшим недостатком всех старых образцов подобной EPROM-памяти. Еще хуже другое, то, что и изолятор, и сам плавающий затвор такого «издевательства» долго не выдерживали, и постепенно деградировали, отчего число циклов стирания/записи было ограничено несколькими сотнями, максимум — тысячами. Во многих образцах flash-памяти даже более позднего времени была предусмотрена специальная схема для хранения карты «битых» ячеек — в точности так, как это делается для жестких дисков. В современных моделях с миллионами ячеек такая карта, кстати, тоже, как правило, имеется, однако число циклов стирания/записи теперь возросло до сотен тысяч. Как этого удалось добиться?

Сначала посмотрим, как осуществлялось в этой схеме стирание. В UV-EPROM при облучении ультрафиолетом фотоны высокой энергии сообщали электронам на плавающем затворе достаточный импульс для того, чтобы они «прыгали» обратно на подложку самостоятельно, без каких-либо электрических воздействий. Первые образцы электрически стираемой памяти (EEPROM, Electrically Erasable Programmable ROM — «электрически стираемое перепрограммируемое ПЗУ», ЭСППЗУ) были созданы в компании Intel в конце 1970-х при непосредственном участии будущего основателя Atmel

Джорджа Перлегоса. Он использовал «квантовый эффект туннелирования Фаулера — Нордхейма» (Fowler — Nordheim). За этим непонятным названием кроется довольно простое по сути (но очень сложное с физической точки зрения) явление: при достаточно тонкой пленке изолятора (ее толщину пришлось уменьшить с 50 до 10 нм) электроны, если их слегка «подтолкнуть» подачей не слишком высокого напряжения в нужном направлении, могут просачиваться через барьер, не «перепрыгивая» его. Сам процесс показан на рис. 11.10 вверху (обратите внимание на знак напряжения на управляющем электроде).

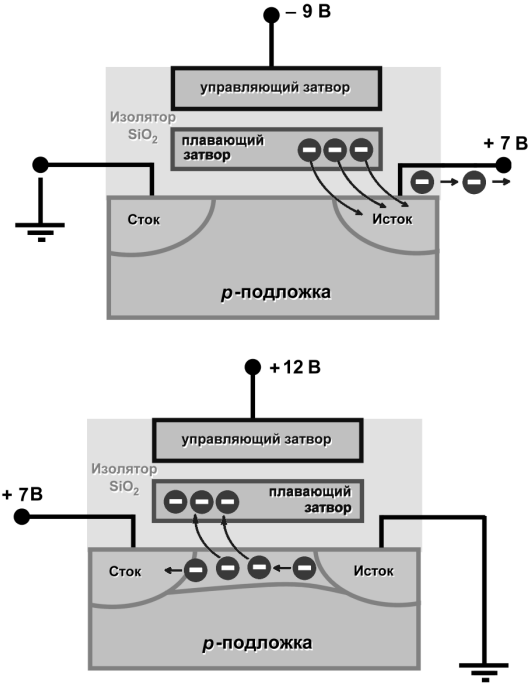


Рис. 11.10. Процесс стирания в элементарной ячейке EEPROM

Старые образцы EEPROM именно так и работали: запись производилась «горячей инжекцией», а стирание — «квантовым туннелированием». Оттого они были довольно сложны в эксплуатации — разработчики со стажем помнят, что первые микросхемы EEPROM требовали два, а то и три питающих напряжения, причем подавать их при записи и стирании требовалось в определенной последовательности. Мало того, цена таких чипов была в свете нынешних тенденций почти запредельной. Автор этих строк сам покупал в середине 1990-х полумегабитную (т. е. 64-килобайтную) энергонезависимую

память по цене 20 долл. за микросхему. Не забудьте еще про «битые» ячейки, возникновение которых в процессе эксплуатации приходилось все время отслеживать. Неудивительно, что на этом фоне разработчики предпочитали более дешевую, удобную, скоростную и надежную статическую память (SRAM), пристраивая к ней резервное питание от литиевых батареек, которые к тому времени уже достаточно подешевели.

Превращение EEPROM во Flash происходило по трем разным направлениям. В первую очередь — в направлении совершенствования конструкции самой ячейки. Для начала избавились от самой «противной» стадии — «горячей инжекции». Вместо нее запись стали осуществлять «квантовым туннелированием», как и при стирании. На рис. 11.10 внизу показан этот процесс: если при открытом транзисторе подать на управляющий затвор достаточно высокое (но значительно меньшее, чем при «горячей инжекции») напряжение, то часть электронов, двигающихся через открытый транзистор от истока к стоку, «просочится» через изолятор и окажется на плавающем затворе. Потребление тока при записи снизилось на несколько порядков. Изолятор, правда, пришлось сделать еще тоньше, что обусловило довольно большие трудности с внедрением этой технологии в производство.

Второе направление — ячейку сделали несколько сложнее, пристроив к ней второй транзистор (обычный, не двухзатворный), который разделил вывод стока и считывающую шину всей микросхемы. Благодаря этому (вместе с отказом от «горячей инжекции») удалось добиться значительного повышения долговечности — до сотен тысяч, а в настоящее время и до миллионов циклов записи/стирания (правда, последнее — при наличии схем коррекции ошибок, которые замедляют работу памяти). Кроме того, схемы формирования высокого напряжения и соответствующие генераторы импульсов записи/стирания перенесли внутрь микросхемы, отчего пользоваться этими типами памяти стало несравненно удобнее, т. к. они стали питаться от одного напряжения (5 или 3,3 В).

И, наконец, третье, чуть ли не самое главное усовершенствование заключалось в изменении организации доступа к ячейкам на кристалле, вследствие чего этот тип памяти и заслужил наименование — flash («молния»), ныне известное каждому владельцу цифровой камеры или карманного MP3-плеера. Так в середине 1980-х назвали разновидность EEPROM, в которой стирание и запись производились сразу целыми блоками — страницами. Процедура чтения из произвольной ячейки, впрочем, по понятным причинам замедлилась — для его ускорения приходится на кристаллах flash-памяти располагать промежуточную (буферную) SRAM. Для flash-накопителей это не имеет особого значения, т. к. там все равно данные читаются и пишутся сразу большими массивами, но для микроконтроллеров может оказаться неудобным.

Тем более там неудобен самый быстродействующий вариант технологии Flash — т. н. память типа NAND (от наименования логической функции «И-НЕ»), где читать и записывать память в принципе возможно только блоками по 512 байт (это обычная величина сектора на жестком диске, также читаемого и записываемого целиком за один раз — отсюда можно понять основное назначение NAND).

В МК обычно применяют традиционную (типа NOR) flash-память программ, в которой страницы относительно невелики по размерам (порядка 64—256 байт). Впрочем, если пользователь сам не взялся за изобретение программатора для такой микросхемы, он может о страничном характере памяти и не догадываться. А для пользовательских данных применяют EEPROM либо с возможностью чтения произвольного байта, либо секционированную на очень маленькие блоки (например, по 4 байта), что также для пользователя значения не имеет.

Развитие технологий flash-памяти имело огромное значение для удешевления и повышения доступности микроконтроллеров. В дальнейшем мы будем иметь дело с энергонезависимой памятью не только в виде встроенных в микроконтроллер памяти программ и данных, но и с отдельными микросхемами, позволяющими записывать довольно большие объемы информации.

Глава 12

Знакомство с микроконтроллером

> Примерно в середине 70-х один из сотрудников предложил мне идею того, что, по сути дела, являлось персональным компьютером. Смысл идеи сводился к оснащению процессора 8080 монитором и клавиатурой и последующей продаже его в качестве прибора для дома. Я спросил: «И что же с ним делать?» Он ответил только, что домохозяйка, например, смогла бы хранить там кулинарные рецепты. Я не увидел в этом никакой пользы, и мы к данному вопросу больше не возвращались.
>
> *Из воспоминаний Гордона Мура, основателя Fairchild и Intel*

Общее число существующих семейств микроконтроллеров оценивается приблизительно в 100 с лишним, причем ежегодно появляются все новые и новые. Каждое из этих семейств может включать десятки разных моделей. В 2002—2003 гг. в мире выпускалось ежегодно 3,2 млрд штук микроконтроллеров. Сравните — объем выпуска микропроцессоров для ПК в 2005—2006 гг. можно оценить в 200 млн единиц в год, т. е. всего около 6% рынка. В то время как в финансовом исчислении, по данным Ассоциации полупроводниковой промышленности США, мировой объем рынка процессоров для ПК в 2006 году равнялся 33 млрд долларов, а микроконтроллеров — всего 12 млрд. Типичная цена рядового МК — порядка 2—5 долл., отдельные их представители могут стоить как существенно меньше, так и больше, но в любом случае их цена не достигает сотен долларов, как для отдельных моделей микропроцессоров от Intel и AMD.

Если говорить о ведущих компаниях, выпускающих микроконтроллеры, то первое место среди производителей 8-разрядных чипов традиционно принадлежит Motorola. Компания Microchip с очень популярным среди радиолюбителей семейством PIC занимает третье место, а Atmel, о продукции которой

мы будем говорить далее — лишь шестое. Тем не менее эта формальная статистика еще ни о чем не говорит. Так, среди МК со встроенной flash-памятью Atmel принадлежит треть мирового рынка (и в 1995 году она была первой, кто вообще выпустил МК такой категории на рынок), при этом надо учитывать, что среди лидеров рассматриваемое семейство AVR-контроллеров самое молодое — первый Atmel AVR был выпущен в 1997 году. Так как рынок МК весьма консервативен (в рекламу на ТВ не вставишь что-нибудь вроде «в данном пылесосе используется последняя модель RISC-микроконтроллера с 32-разрядной шиной данных», поэтому мода тут играет далеко не ведущую роль), и шестое место в мировом масштабе можно рассматривать, как огромный успех.

Кроме 8-разрядных МК AVR, Atmel выпускает еще несколько их разновидностей, в том числе относящихся к таким популярным семействам, как ARM-процессоры и заметно модифицированные наследники старинного 8051, для которого в мире накоплен огромный объем программного обеспечения. Мы здесь ограничимся лишь 8-разрядными AVR, как одними из самых удобных в радиолюбительской и полупрофессиональной практике (например, для изготовления научного, производственного и другого спецоборудования в единичных экземплярах) — на этой почве они конкурируют лишь с упомянутыми PIC, однако для начального освоения опытные конструкторы рекомендуют именно AVR.

По вычислительной мощности ядро AVR-контроллеров, выполненное по RISC-архитектуре (Reduced Instruction System Command — «система с сокращенным набором команд»), заметно превышает 8-разрядные процессоры первых персональных компьютеров 80-х годов (i8086 в первых моделях IBM PC и 6502 в персоналках Apple), и сравнимо с производительностью 16-разрядного процессора i286, поскольку большинство инструкций выполняются за один такт, а рабочие частоты более высокие (обычно до 16 МГц, за небольшими исключениями, а у некоторых моделей — до 20 МГц). В AVR имеется 32 регистра общего назначения (часть из которых может также выполнять специализированные функции). Это позволяет в простых программах вообще не обращаться к памяти, что особенно удобно для начинающих.

Classic, Mega и Tuny

Линейка AVR делится на три семейства: Classic, Mega и Tuny. МК семейства Classic (они именовались, как AT90S<марка контроллера>) ныне уже не производятся, однако все еще распространены, т. к. их еще много на складах торгующих фирм и для них наработано значительное количество программ. Чтобы все это ПО пользователям не пришлось переписывать, фирма Atmel

позаботилась о преемственности — все МК семейства Classic (за исключением разве что самого первого и не очень удачного AT90S1200) имеют функциональные аналоги в семействе Mega, например, AT90S8515 — ATmega8515, AT90S8535 — ATmega8535 и т. п. (только AT90S2313 имеет аналог в семействе Tuny — ATtuny2313). На рис. 12.1 приведены варианты корпусов микроконтроллеров Atmel AVR и панельки для них.

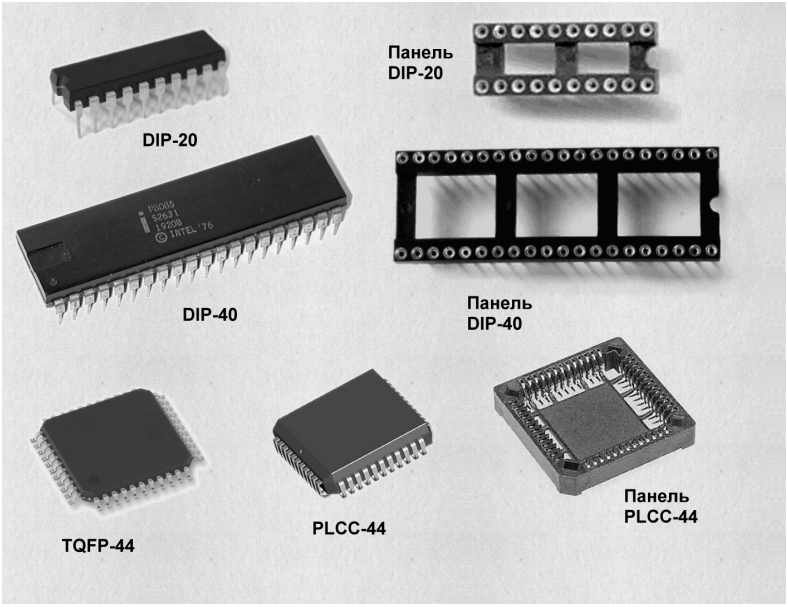


Рис. 12.1. Различные корпуса микроконтроллеров Atmel AVR и панельки для них

Полная совместимость обеспечивается специальным установочным битом (из набора т. н. Fuse-битов), при программировании которого Mega-процессор начинает функционировать как Classic (подробнее об этом — в *главе 13*). Для вновь разрабатываемых устройств обычно никакого смысла в режиме совместимости нет, однако такой прием в ряде случаев может оказаться полезным для начинающих, т. к. МК Classic устроены проще и не заставляют пользователя отвлекаться на ненужные подробности, не имеющие отношения к делу. Поэтому часть примеров в этой книге, особенно начального уровня, будут ориентированы на семейство Classic.

Семейство Tuny (что в буквальном переводе означает «легко запоминающийся») предназначено для наиболее простых устройств, часть таких МК не имеет возможности программирования по последовательному интерфейсу (а ATtuny10 даже является однократно программируемым), и потому мы их

не будем рассматривать в этой книге (за исключением ATtuny2313, точнее его «классического» аналога AT90S2313, оба они очень удобны для проектирования небольших, но функциональных устройств, и к тому же могут работать с частотой до 20 МГц). Это не значит, что Tuny следует избегать, среди них есть очень удобные и функциональные микросхемы. Но нельзя объять необъятное — здесь в основном мы будем рассматривать Mega.

Перечислим отличительные особенности Mega.

- Flash-память программ от 8 до 128 кбайт (у семейства Classic 2—8 кбайт).
- Статическое ОЗУ (SRAM) от 512 байт до 4 кбайт (для Classic 128—512 байт).
- EEPROM данных от 512 байт до 4 кбайт (для Classic 128—512 байт).
- Различные способы тактирования: от встроенного RC-генератора, внешней RC-цепочки, внешнего кварцевого резонатора, внешнего сигнала (у Classic — только от кварцевого резонатора или внешнего сигнала). Удобная возможность для удешевления и упрощения схем, хотя и усложняет начальное программирование кристалла, причем у некоторых старших моделей имеется возможность программного снижения частоты.
- Расширенные режимы пониженного энергопотребления.
- Наличие встроенного детектора снижения напряжения питания (Brown-Out Detector, BOD).
- Усовершенствованный полнодуплексный последовательный синхронно-асинхронный порт USART (на практике автором использовался исключительно в режиме обычного UART).
- Последовательный двухпроводной интерфейс TWI (по другому, I^2C — на практике автор никогда не мог его заставить работать лучше, чем программный имитатор I^2C, пригодный практически для всех моделей AVR).
- Инструкции аппаратного умножения 8-разрядных чисел (в семействе Classic отсутствуют).

Структура МК AVR

Для ознакомления с тем, как устроены МК AVR, возьмем «классический» AT90S8515. Он включает в себя все существенные узлы моделей МК AVR из середины линейки, т. е. как младших Mega, так и старших Tuny, за исключением имеющегося в некоторых моделях АЦП (как, например, в его близком родственнике AT90S8535/ATmega8535, который мы будем широко использовать). Выбор именно Classic обусловлен тем, что блок-схема AVR громоздкая

и без того, а в Mega присутствует еще много компонентов, которые для нас останутся второстепенными. Как и АЦП, эти компоненты (Brown-Out Detector, Fuse-биты, USART, дополнительные таймеры и т. п.) мы рассмотрим по ходу дела в дальнейшем.

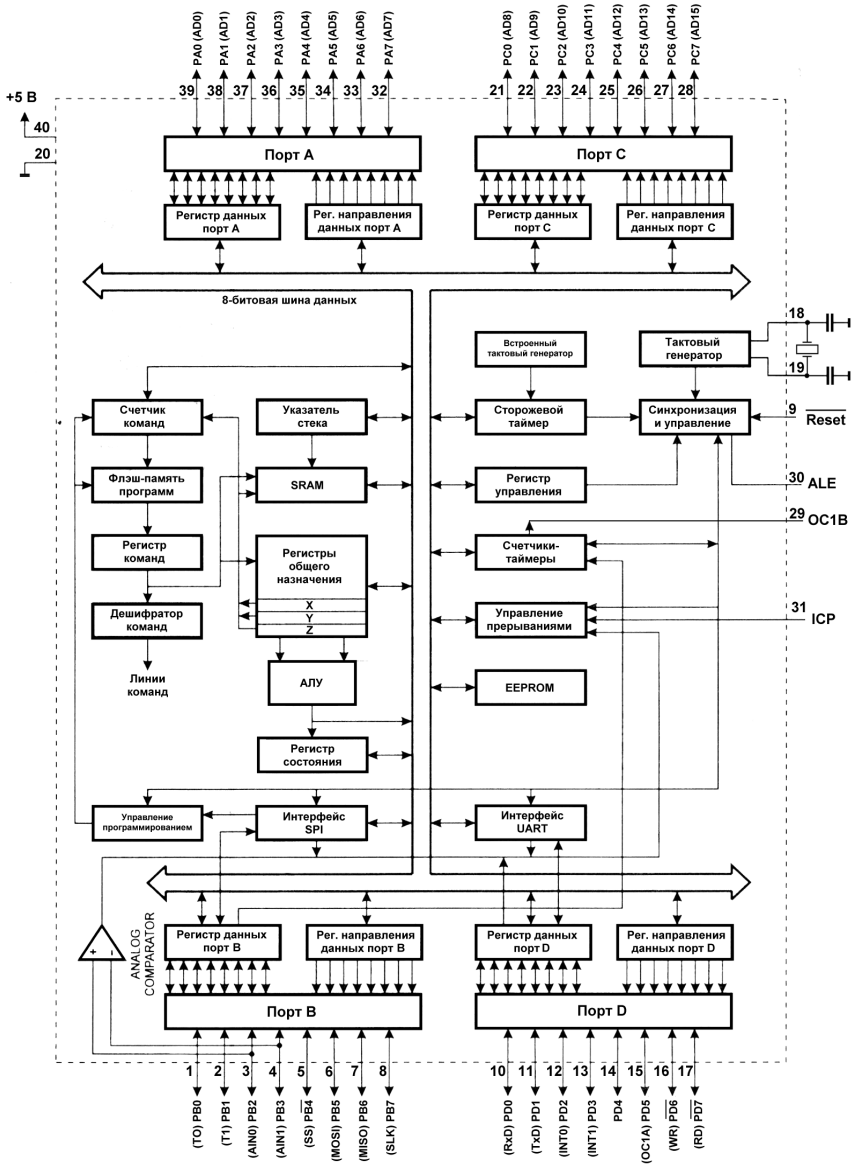


Рис. 12.2. Структура AT90S8515

На рис. 12.2 показана внутренняя структура МК AT90S8515. Нумерация выводов приведена для корпуса DIP-40, кроме этого, процессор выпускается в 44-выводных корпусах PLCC и TQFP (см. рис. 12.1). Даже беглого взгляда на рисунок достаточно, чтобы понять, что для детального рассмотрения структуры этого МК здесь просто не хватит места. Поэтому мы не будем переписывать фирменное описание (с некоторыми подробностями мы познакомимся по ходу дальнейшего изложения), а рассмотрим только некоторые ключевые узлы этой структуры и особенности их функционирования.

Параллельные порты ввода/вывода

Начнем с внешних портов. В этой модели их четыре, и если подсчитать необходимые выводы ($8 \times 4 = 32$), прибавить к ним обязательные выводы питания (контакты 20 и 40), тактового генератора (контакты 18 и 19) и вывод Reset (контакт 9), также присутствующий во всех моделях МК без исключения, то получится, что на все остальное остается три вывода. Это, конечно, недостаточно, поэтому почти все выводы портов, кроме своей основной функции (двунаправленного ввода/вывода) несут также и дополнительную. А как они (функции) разбираются между собой? А никак. Никакого специального переключения выводов портов не требуется, просто, если вы, к примеру, в своей программе инициализируете последовательный порт UART, то соответствующие выводы порта D (PD0 и PD1, выводы 10 и 11 микросхемы) будут работать именно в альтернативной функции, как ввод и вывод UART. При этом в промежутках между этим режимом выводов они могут выполнять функцию обычных двунаправленных выводов (хотя на практике это неудобно, потому что приходится применять схемотехнические меры для изоляции функций друг от друга, но иногда к этому прибегают). Аналогичная ситуация со всеми остальными альтернативными функциями — они начинают работать, когда в программе инициализированы соответствующие устройства МК.

По умолчанию все дополнительные устройства отключены, а порты работают на вход, причем находятся в состоянии с высоким *импедансом* (т. е. высоким входным сопротивлением). Работа на выход требует специального указания, для чего в программе нужно установить соответствующий нужному выводу бит в регистре направления данных (обозначается DDR*x*, где *x* — буква, обозначающая конкретный порт, например для порта A это будет DDRA). Если бит сброшен (т. е. равен логическому нулю), то вывод работает на вход (установка по умолчанию), если установлен (т. е. равен логической единице) — то на выход. Причем для установки выхода в состояние «1» нужно отдельно установить соответствующий бит в регистре данных порта (обозначается PORTx), а для установки в «0» — сбросить этот бит. Направ-

ление работы вывода (вход-выход, регистр DDR*x*), и его состояние (0—1, PORT*x*) путать не следует.

Регистр данных PORT*x* фактически есть просто выходной буфер, все, что в него записывается, тут же оказывается на выходе. Но если установить вывод порта на вход (т. е. записать в регистр направления логический ноль), как это сделано по умолчанию, то регистр данных DDR*x* будет играть несколько иную роль — установка его в «0» (так по умолчанию) означает, что вход находится в третьем состоянии с высоким сопротивлением, а установка в «1» подключит к выводу «подтягивающий» (pull-up) резистор сопротивлением около 35 кОм (подобно рис. 11.3, *а*; вернитесь к описанию работы подобного выходного каскада на «общую шину» в *главе 11* — и вы поймете, зачем он нужен).

Сразу отметим, что встроенный pull-up-резистор лучше употреблять при работе на «общую шину» в пределах одной платы, но в большинстве случаев, когда требуется такой резистор (например, если вы подключаете ко входу выносную кнопку с двумя выводами, которая коммутируется на «землю», или при работе на «общую шину» с удаленными устройствами), лучше устанавливать дополнительный внешний резистор параллельно этому внутреннему, с сопротивлением от 1 до 5 кОм (в критичных для потребления случаях его величину можно увеличить до 20—30 кОм). В ответственных устройствах такой резистор следует устанавливать на выводе 9 (Reset) и на выводах 6, 7 и 8, которые служат для программирования и подключены к программирующему разъему ISP (см. *главу 13*), поскольку их также следует «подтягивать» к напряжению питания (тем более что довольно часто они также используются в качестве обычных входных/выходных линий, что процессу программирования, как правило, не мешает).

Заметки на полях

Установка дополнительного резистора к питанию (как и конденсатора — к «земле») по входу Reset для более надежного сброса рекомендовалась для самых первых моделей AVR, однако в описаниях современных типов вы такого не встретите. Но даже для старых моделей не оговаривалась установка внешних резисторов по другим выводам. Однако для более надежной работы устройства в условиях помех автор настоятельно советует следовать изложенным рекомендациям — хуже от этого не будет. Сопротивление «родного» резистора (который на самом деле представляет собой, разумеется, полевой транзистор) слишком велико для того, чтобы электромагнитные помехи («наводки») на нем эффективно «садились», и у вас могут происходить ложные срабатывания, перезапуски системы, а при очень мощных помехах — даже порча программы в памяти программ. Это, конечно, не касается ситуации, когда порт в режиме входа подсоединен к низкоомному выходу другой микросхемы или, скажем, транзисторного ключа (см. *главу 12*), т. к. у них на выходе всегда имеется какой-то сигнал, и резистор тогда не потребуется вообще, лучше всего

ставить вывод микросхемы в третье состояние. Не забывайте только об этих резисторах при работе в режимах экономии питания — нужно тщательно проанализировать схему на предмет того, чтобы исключить ситуации, при котором через эти резисторы протекает ток.

Добавим еще, что, как вы, очевидно, уже поняли из изложенного, выводы портов в достаточной степени автономны, и их режим может устанавливаться независимо друг от друга. Кстати, а как прочесть уровень на выводе порта, если он находится в состоянии работы на вход? Возникает искушение прочесть данные из регистра данных PORT*x*, но это ничего не даст — вы прочтете только то, что там записано вами же. А для чтения того, что действительно имеется на входе (непосредственно на выводе микросхемы), предусмотрена другая возможность. Чтение происходит из буферного элемента, который в первом приближении соответствует элементу B на схемах по рис. 11.3, т. е. для чтения нужно обратиться к некоторому массиву, который обозначается PIN*x*. Обращение осуществляется так же, как и к отдельным битам обычных регистров (см. *главу 13*), но PIN*x* не есть регистр, это просто некий диапазон адресов, чтение по которым предоставляет доступ к информации из буферных элементов на входе порта. Записывать что-то по адресам PIN*x*, естественно, нельзя.

Прерывания

Как и в ПК, прерывания (interrupts) в микроконтроллерах бывают двух видов. Но если в ПК прерывания делятся на аппаратные (например, от таймера или клавиатуры) и программные (фактически не прерывания, а подпрограммы, записанные в BIOS — с распространением Windows это понятие почти исчезло из программистской практики), то в МК, естественно, все прерывания являются аппаратными, а делятся они на внутренние и внешние. Любое прерывание, а также вообще возможность их возникновения требует отдельно предварительного специального разрешения. Учтите, что для инициализации прерываний в программе необходимо сделать два действия: разрешить соответствующее прерывание (предполагаем, все прерывания вообще уже разрешены) и установить для него один из доступных режимов. И, конечно, написать подпрограмму-обработчик, иначе все это будет происходить вхолостую (подробнее см. *главу 13*).

Внутренние прерывания могут возникать от любого устройства, которое является дополнительным по отношению к ядру системы: от таймеров, от аналогового компаратора, от последовательного порта и т. д. Внутреннее прерывание — это событие, которое возникает в системе и прерывает выполнение основной программы. Система внутренних прерываний в AVR довольно раз-

ветвленная и представляет собой основной механизм взаимодействия устройств с ядром системы, и мы еще к этому вопросу будем неоднократно возвращаться.

На внешних прерываниях мы остановимся здесь подробнее. Внешних прерываний у МК типа AT90S8515 два, INT0 и INT1 (а вот в ATmega128 — целых восемь!), естественно, они могут возникать независимо друг от друга. Внешнее прерывание — событие, которое возникает при появлении сигнала на одном из входов, специально предназначенных для этого (в данном случае контакт 12 или 13). Различаются три вида событий, вызывающих прерывание, и их можно устанавливать в программе: это может быть низкий уровень напряжения, а также положительный или отрицательный фронт на соответствующем выводе. Любопытно, что прерывания по всем этим событиям выполняются, даже если соответствующий вывод порта сконфигурирован на выход.

Кратко рассмотрим особенности этих режимов. Прерывание по низкому уровню (режим установлен по умолчанию, для его инициализации достаточно разрешить соответствующее прерывание) возникает всякий раз, когда на соответствующем входе присутствует низкий уровень. «Всякий раз» — это значит, что действительно всякий, т. е. если отрицательный импульс длится какое-то время, то прерывание, закончившись (т. е. когда выполнится соответствующая процедура), повторится снова и снова, не давая основной программе работать. Поэтому обычно сразу же по возникновении такого внешнего прерывания его следует запретить (процедура обработки при этом, раз уж началась, выполнится до конца), и разрешить опять только тогда, когда внешнее воздействие должно уже закончиться (например, если это нажатие кнопки, то его стоит опять разрешить по таймеру через одну-две секунды).

В отличие от этого, прерывания по фронту или спаду выполняются один раз на импульс. Конечно, от дребезга контактов там никакой защиты нет и быть не может, потому что МК не способен отличить дребезг от серии коротких импульсов. Если это критично, нужно либо принимать внешние меры по защите от дребезга, либо использовать тот же способ, что и для прерывания по уровню — внутри процедуры обработчика прерывания первой же командой запретить само прерывание, а через некоторое время в другой процедуре (по таймеру, например, или по иному событию) опять его разрешить (способ «антидребезга», фактически идентичный применению одновибратора, см. *главу 9*). Но если по нажатию кнопки просто что-то устанавливается в некое состояние, то ничего страшного не случится, если это произойдет несколько раз подряд, только следует учесть, что окончание этого процесса совпадет с отпусканием кнопки.

Подробности

У внимательного читателя возникает законный вопрос — а зачем вообще нужен режим внешнего прерывания по уровню? Дело в том, что оно во всех моделях выполняется асинхронно, т. е. в тот момент, когда низкий уровень появился на выводе МК. Конечно, обнаружение прерывания может произойти только по окончании текущей команды, так что очень короткие импульсы могут «пропасть». Но прерывания INT0 и INT1 в режиме управления по фронту у большинства моделей определяются наоборот, только синхронно, т. е. в момент перепада уровней тактового сигнала контроллера, поэтому их длительность не должна быть короче одного периода тактового сигнала. Но это не самое главное: по большому счету разницы в этих режимах никакой бы не было, если бы не то обстоятельство, что синхронный режим требует непременно наличия этого самого тактового сигнала. И асинхронное внешнее прерывание, соответственно, может «разбудить» контроллер, находящийся в одном из режимов глубокого энергосбережения, когда тактовый генератор не работает, а синхронное — нет. И обычные МК, вроде разбираемого AT90S8515 семейства Classic (но не его Mega-аналога!), выводиться из глубокого «сна» могут только внешним прерыванием по уровню, которое не всегда удобно использовать. У большинства же моделей семейства Mega (из младших моделей — кроме ATmega8), имеется еще одно прерывание INT2, которое происходит только по фронтам (а не по уровню), и, в отличие от INT0 и INT1, только асинхронно. Это значительно повышает удобство работы с семейством Mega в режиме энергосбережения (на эту тему см. *главу 17*).

Таймеры-счетчики

В МК 8515 два таймера-счетчика: один 8-разрядный (Timer 0), второй — 16-разрядный (Timer 1). Оба счетчика представляют собой счетчики с предзагрузкой (вроде 561ИЕ11/14) и могут работать от тактовой частоты процессора непосредственно, или поделенной на 8, 64, 256 или 1024, представляя в таком режиме собой именно таймеры, т. е. устройства для отсчета времени.

Могут они работать и как обычные счетчики внешних импульсов с выводов T0 (вывод 1) для Timer 0 и T1 (вывод 2) для Timer 1 (по спаду или по фронту). Частота подсчитываемых импульсов не должна превышать половины частоты тактового генератора МК (причем при несимметричном меандре инструкция рекомендует и еще меньшее значение предельной частоты — 0,4 от тактовой), иначе счетчик вам посчитает «погоду на Марсе» — это сильное ограничение, поэтому, например, использовать МК для создания универсального частотомера неудобно. Поэтому все быстродействующие логические схемы конструируют не на микроконтроллерах, а на комбинационной логике — на ПЛИС.

При наступлении переполнения счетчика возникает событие, которое может вызывать соответствующее прерывание. Счетчик Timer 0 фактически этим и ограничивается. 16-разрядный счетчик Timer 1 — более «продвинутая» штука,

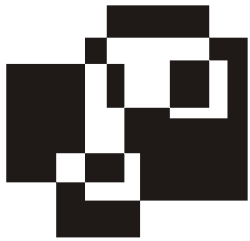
и может вызывать прерывания по совпадению с определенным заранее заданным числом (точнее, даже два отдельных прерывания с двумя разными числами A и B), при этом счетчик может обнуляться или продолжать счет, и на специальных выводах OSC1A (вывод 15) или OSC1B (вывод 29) при этом могут генерироваться импульсы (аппаратно, без участия программы). Этот режим нам понадобится, чтобы задавать определенную частоту прерываний — например, в часах для отсчета секунд (см. *главу 14*).

Также возможно прерывание по внешнему событию на специальном выводе ICP (вывод 31), при этом содержимое счетчика помещается в некий регистр ICR1, а счетчик может обнуляться и начинать счет заново или просто продолжать счет (режим измерения периода внешнего сигнала). Источником таких событий может быть встроенный аналоговый компаратор — это удобно для того, чтобы построить на основе МК частотомер, но, как мы видели ранее — только низкочастотный, не более 5—7 МГц. Некоторые подробности о прерываниях и таймерах вы узнаете также из *главы 17*, когда мы будем разбирать режимы энергосбережения.

В AT90S8535 и во всех Mega (кроме Mega8515) есть как минимум еще один, третий таймер, который также будучи, как и Timer0, 8-разрядным, может выполнять более интересные вещи (например, работать автономно от остальной системы, когда весь процессор находится в режиме экономии, что позволяет этот таймер использовать в качестве часов реального времени, RTC). Но и это не все: описанные таймеры могут работать в т. н. PWM-режиме, специально предназначенном для их работы в качестве широтно-импульсных модуляторов (ШИМ). Этот режим мы кратко рассмотрим в *главе 19* в связи с голосовой сигнализацией.

Кроме таймеров-счетчиков в любом AVR-контроллере есть сторожевой (Wathcdog) таймер. Он предназначен в основном для вывода МК из режима энергосбережения через определенный интервал времени, но может выполнять и простой перезапуск МК при включении питания, например, если работа программы зависит от прихода внешних сигналов, то при их потере (например, из-за сбоев на линии) МК может просто «повиснуть», а Wathcdog-таймер выведет его из этого состояния.

Остальные узлы МК семейства AVR мы рассмотрим по ходу изложения конкретных схем — так будет нагляднее. А тому, как состыковать последовательный интерфейс UART с компьютером, будет посвящена даже отдельная *глава 18*. Здесь же мы закончим краткое знакомство с микроконтроллером и перейдем к вопросу о том, как его программировать.



Глава 13

Персональный компьютер вместо паяльника

Всегда имеются в продаже свежие программные продукты!

Объявление на рынке

Готовой работающей программой называют код, содержащий пока еще не обнаруженные ошибки.

Народное

Слово «программирование» в отношении МК имеет двоякий смысл. Во-первых, это собственно процесс написания программы, как и для любых других устройств, содержащих процессор, от холодильников до персональных компьютеров или узлов управления космическими аппаратами. Во-вторых, этим словом также называют процесс загрузки программы в память программ. Чтобы не путаться, мы будем подразумевать под «программированием» МК именно запись программ в память (ее еще называют «прошивкой»), а подготовительный этап будем называть «созданием программы».

Заметки на полях

Отмечу, что каждый, кто хочет всерьез научиться программировать (неважно для чего, хоть для Веба, хоть для микроконтроллеров), должен начинать с изучения фундаментального труда Д. Кнута [10]. К сожалению, Кнут не практикующий программист, а профессор Стэндфордского университета, и как все профессоры, начинает с анализа, доказательств теорем и тому подобной «чепухи», зачем-то очень нужной математикам. Читатель, конечно, понимает, что в этих высказываниях есть значительная доля шутки, но правда заключается в том, что математик потребует доказательства там, где инженеру достаточно заверения авторитета, что такая-то формула справедлива, а доказательства и обоснования со ссылками на древнегреческих авторов его только отвлекают от главного. И все же я очень советую изучить по крайней мере те главы первого тома, которые посвящены машинным языкам, и алгоритмы арифметических

действий из второго тома (всего тома три, но я ссылаюсь только на два, так как третий посвящен исключительно алгоритмам сортировки, которые лежат за пределами любительской программистской практики).

Конечно, продираться через многочисленные математические изыски Кнута необязательно, если вы согласны ограничиться готовыми программными модулями, которые с течением времени научитесь править и подстраивать под свои нужды. Материал в этой книге не рассчитан на «крутых» профессионалов, и для его освоения и повторения приведенных здесь конструкций не надо обладать особыми знаниями. Но после ознакомления с работой Д. Кнута во многих вещах вам будет значительно проще разобраться.

Как программируются микроконтроллеры

В МК AVR встроенная память программ энергонезависимая, и называется «flash-памятью программ», так ее можно отличить от также энергонезависимой памяти для хранения пользовательских данных, которая именуется EEPROM. Если строго следовать терминологии, то flash — это, как вы знаете, разновидность EEPROM, но во всяком случае, встроенная память программ для AVR-контроллеров делается именно по flash-технологии, что позволяет значительно ускорить процесс программирования. Первоначально такое разделение на EEPROM и flash для, по сути дела, одинаковых по функционированию разделов памяти (и то и другое призвано хранить данные при отсутствии питания) приводило еще и к тому, что память программ допускала лишь около 1000 циклов перезаписи, в то время как EEPROM намного больше (порядка 100 000). Так было в семействе Classic и первоначальных выпусках семейств Tuny и Mega. И хотя на практике и того, и другого более чем достаточно, современные МК AVR допускают уже 10 000 циклов перезаписи для памяти программ.

Подробности

Дело в том, что модели из семейства Mega могут самопрограммироваться в процессе эксплуатации (загружать программу из внешних источников), и вот тогда тысячи циклов в теории может оказаться недостаточно. Если вас заинтересовала эта возможность, то имейте в виду, что для самопрограммирования все равно требуется первоначально занести в память программу-загрузчик. Употребляется этот способ тогда, когда контроллер должен выполнять существенно различные функции в зависимости от каких-то условий. В некотором роде это напоминает загрузку различных программ в обычных компьютерах (например, в карманных или в смартфонах). Здесь мы, конечно, разбирать эти подробности не будем.

С другой стороны, со временем разница между памятью программ и пользовательской EEPROM, совершенно очевидно, стирается: так, в семействе Tuny flash-память программируется побайтно, а не страницами, а в старших моделях

семейства Mega, наоборот, EEPROM программируется страницами (правда, небольшого размера, всего в 4 байта). Для конечного пользователя все эти нюансы не имеют ровным счетом никакого значения.

Встроенная память программ имеет объем от 1 кбайта у ATtuny11 до 128 кбайт у ATmega128 (как вы догадались, число в наименовании модели как раз и означает объем памяти программ). Если у пользователей системы Windows такие объемы вызовут лишь снисходительную улыбку, то советую им не «задаваться», поскольку Андрес Хейлсберг, автор Turbo Pascal, умудрился в первой версии этого продукта (1982) уложить в 33 280 байт интегрированную среду разработки, встроенный редактор и библиотеку времени выполнения. Так что возможностей AVR достаточно, чтобы построить неслабый персональный компьютер, который мог бы существенно превышать по возможностям модели IBM PC AT на процессоре i286 (напомним, что последний выполнял инструкцию в среднем за 2—4 такта, а AVR — за 1—2). Правда, i286 мог адресовать до 16 Мбайт динамического ОЗУ (у процессоров для ПК, напомним, нет встроенной памяти, кроме операционных регистров). Но для применений, которые требуют объемного программного кода, и AVR (кроме Tuny) могут выполнять программы из внешней памяти объемом до 4 Мбайт в отдельных моделях, причем скорость выборки команд зависит только от параметров ОЗУ и тактовой частоты AVR. Мы в данной книге эту возможность не рассматриваем — автору еще ни разу не удалось превысить лимит встроенной памяти выбранного МК.

Программаторы

Для того чтобы загрузить программу во flash-память, служат специальные программаторы, которые делятся на последовательные и параллельные (существуют еще стандартизированный согласно IEEE Std 1149.1-1990 интерфейс JTAG для отладки и тестирования микроконтроллерных устройств. С его помощью, например, выполняется всем известная «перешивка» готовых устройств. Но мы не будем здесь на этом останавливаться, т. к. в любительских конструкциях, да и во многих профессиональных, он никогда не употребляется).

Последовательные программаторы еще называют ISP (In-System Programmer, что переводится как «внутрисистемный программатор»), потому что они обычно не предполагают специального устройства для подсоединения и питания программируемой микросхемы, а заканчиваются обычным плоским кабелем с двухрядным штырьковым разъемом типа IDC или PLD (таким же, какие служат для подсоединения периферии к материнским платам ПК), который подсоединяется к специально предусмотренной штыревой части,

располагаемой прямо на плате вашего устройства (подробнее см. *главу 14*). Питание на программатор при этом поступает от самой схемы.

Такой способ программирования очень удобен для отладки, т. к. МК находится в «родном» окружении и сразу после программирования автоматически начинает работать. Причем внешние элементы (с некоторыми оговорками) процессу программирования обычно не мешают. Далее я расскажу, как с помощью этого способа программирования превратить любую схему в отладочный модуль, что позволяет избежать необходимости приобретения дорогих универсальных модулей и обойтись без изучения фирменной среды программирования AVR Studio. Недостатки способа — дополнительная площадь, которая будет занята «лишними» элементами на плате, а также некоторое снижение помехоустойчивости (о том, какие дополнительные меры следует принять, я обязательно расскажу, когда мы будем рассматривать конкретные схемы). Зато вы всегда без лишних сложностей сможете поправить ошибку в программе и дополнить ее функциональность хоть через месяц, хоть через год, что особенно важно для любительских конструкций, которые, как правило, выпускаются в одном-двух экземплярах.

Подробности

Последовательное программирование AVR есть, по сути, использование стандартного последовательного интерфейса SPI в специфических целях, причем следует учитывать, что в некоторых моделях (ATmega64 и ATmega128) выводы собственно SPI с выводами программирования не совпадают. Процесс последовательного программирования начинается с того, что на вывод SCK подается напряжение уровня «земли», после этого на Reset поступает короткий положительный импульс, и затем последний также оставляют в состоянии низкого уровня не менее, чем на 20 мс (можно и просто предварительно подключить эти выводы к низкому уровню, а затем включить питание контроллера). При этом микросхема переходит в режим ожидания команд программирования. Отсюда понятно, почему снижается помехоустойчивость: такое сочетание уровней вполне может возникнуть в процессе эксплуатации из-за наводок на выводы разъема программирования. У автора этих строк в контроллере, расположенном в цехе со множеством работающих механизмов, при срабатывании мощного пускателя в нескольких метрах от устройства стиралась память программ! Правда, исправить ситуацию оказалось довольно просто установкой дополнительных «подтягивающих» резисторов.

Последовательное программирование поддерживают не все контроллеры AVR, из семейства Tuny так можно программировать только ATtuny12 и ATtuny15. В отличие от последовательного, параллельный способ программирования применяется для кристалла вне схемы. Программаторы такого рода, как правило, в той или иной степени универсальны и подходят для множества различных программируемых чипов, начиная от микросхем ПЗУ (в том числе и компьютерной BIOS) и заканчивая большинством МК.

Например, показанный на рис. 13.1 популярный AutoProg поддерживает около 4000 типов микросхем. Такие программаторы, естественно, относительно дороги (не самый дорогой в своем классе AutoProg стоит почти 9000 руб.), что и понятно — представьте, сколько труда вложено в программное обеспечение с поддержкой такого количества чипов. Экономически целесообразны они для тех, кто много работает с различными семействами микросхем, а также при изготовлении серийной продукции с отлаженной программой, в которой разъемы ISP-систем будут только мешать. Для отладки универсальные программаторы неудобны, т. к. приходится часто вытаскивать и вставлять микросхему в панельку с риском ее повредить, а для некоторых типов корпусов это может быть вообще исключено, если нет соответствующего переходника. Если вы работаете только с одним типом микросхем, но хотите тиражировать отлаженное изделие без лишних разъемов на плате, то вам ничего не стоит изготовить отдельный программатор на основе ISP-системы. Правда, «оживить» чипы AVR, в которых по ошибке были неправильно запрограммированы fuse-биты, отвечающие за источник тактового сигнала (об этом см. в конце главы), можно только с помощью параллельного программатора.

Рис. 13.1. Параллельный программатор AutoProg

Надо сказать, что и тот и другой способ программирования не составляет никакого секрета и подробно излагается в описании любого AVR. Так что в принципе программатор может быть изготовлен самостоятельно как в виде аппаратного устройства, обычно подключаемого через COM-порт, так и в чисто программном виде, с подключением кристалла через LPT-порт. Причем

Atmel даже приводит в своих рекомендациях (Application Notes, которые наши электронщики обычно ласково зовут «аппнотами», доступны на сайте Atmel по адресу: **http://www.atmel.com/dyn/products/app_notes.asp?family_id=607**) типовую схему простейшего ISP-программатора (см. «аппноту» AVR910, которая в PDF-формате содержит схему, а asm-файлы включают исходные коды «прошивки»). Программу для ПК, впрочем, придется писать самому, на основе описания процесса программирования (или доставать где-нибудь готовую, например, извлекать из AVR Studio), поскольку программатор лишь транслирует команды.

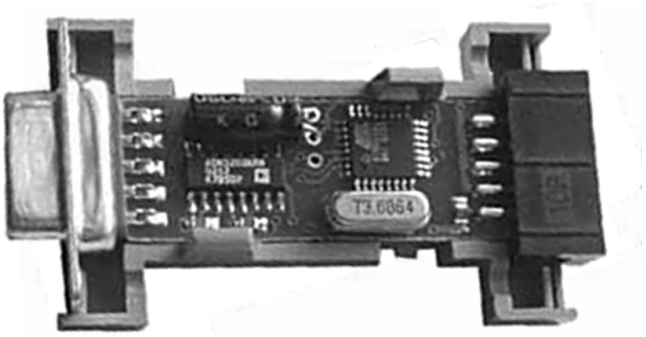


Рис. 13.2. Программатор AS-2 фирмы Argussoft в разобранном виде

Конечно, изготовление такого программатора самостоятельно — это занятие очень «на любителя», потому что количество затраченных усилий отнюдь не соответствует результату. Программатор можно заказать в самой Atmel, хотя это и нецелесообразно по ряду причин, т. к. долго и дорого (зато поддержка AVR Studio, о которой далее, обеспечена). Дешевенькое «наколеночное» устройство такого типа можно приобрести рублей за 300 через Интернет, а цена «более фирменных» ISP-систем не выходит за пределы примерно 30 долл. Автор этих строк много лет пользуется программатором AS-2 фирмы Argussoft, который непосредственно подсоединяется к COM-порту (на рис. 13.2 он показан без соединительного кабеля). Есть и аналогичная модификация AS-3, которая работает с USB. Программатор позволяет «прошивать» прямо в системе многие микросхемы Atmel (не только AVR), причем интерфейс управляющей программы на русском языке очень нагляден, что, в частности, дополнительно предохраняет от неверной установки Fuse-битов (см. раздел в конце этой главы). Поддерживается пакетная работа (когда стирание, запись и проверки объединяются в одну операцию), а также имеется полезная функция перепрошивки самого программатора, если он вдруг начинает сбоить.

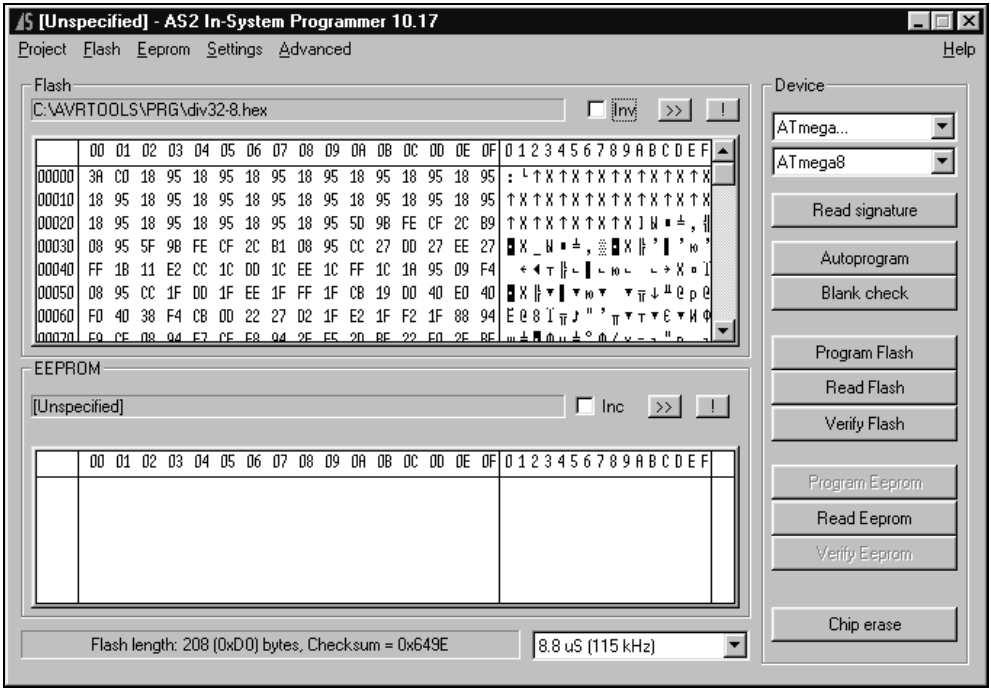
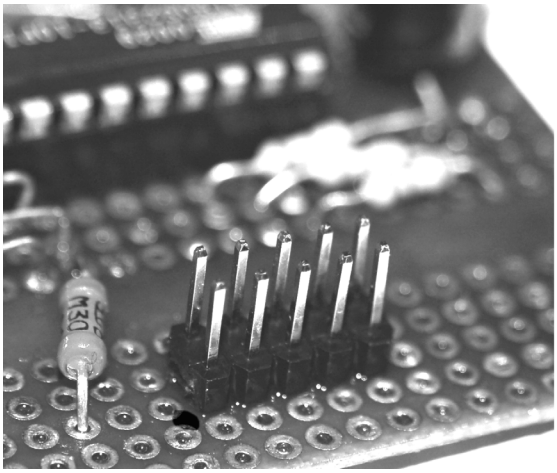


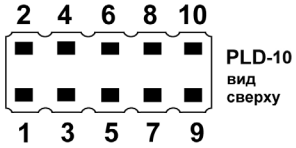
Рис. 13.3. Программа-загрузчик к программатору AS-2

Программирующий разъем — десятиконтактный игольчатый IDC или PLD (подробнее про наименования разъемов такого типа см. *главу 14*) и именно на него будут ориентированы все схемы, приведенные в этой книге. Это не значит, что я вас призываю покупать именно AS, разъем такого типа стал стандартом и им снабжены многие ISP, кроме упоминавшегося простейшего ISP от самой Atmel, где разъем минимальный — шестиконтактный (три линии собственно SPI, а также два питания и Reset). Впрочем, при необходимости всегда можно изготовить переходник, т. к. все последовательные программаторы работают по одним и тем же линиям (в десятиконтактном разъеме «лишние» контакты служат для разделения сигнальных линий «земляными», так же, как в соединительном кабеле IDE-интерфейса жестких дисков).

Внешний вид программирующего разъема PLD-10 на макетной плате и разводка его выводов показана на рис. 13.4. Обратите внимание, что отсчет выводов ведется не по кругу, как для микросхем, в нижнем ряду расположены все нечетные выводы, в верхнем — четные (это удобно, т. к. не приходится нарушать нумерацию у разъемов с различным количеством контактов, когда короткие могут получаться из длинных простым «отламыванием» лишней части). Естественно, при отсутствии чехла ответная часть может быть при-

соединена двумя способами, потому первый вывод на плате следует помечать (на фото видна черная точка, проставленная фломастером). Названия выводов соответствуют таковым у контроллера.





5	$\overline{\text{Reset}}$
4,6,8,10	GND
2	+5 В
7	SLK
9	MISO
1	MOSI

Рис. 13.4. Пример расположения программирующего разъема PLD-10 на макетной плате и его разводка выводов

С или ассемблер?

Ядро и система команд МК AVR с самого начала создавались в сотрудничестве с фирмой IAR Systems — производителем компиляторов для языков программирования C/C++. Поэтому структура контроллера максимально оптимизирована для того, чтобы можно было писать программы на языках высокого уровня. Так утверждает реклама, но верить подобным утверждениям стоит с некоторой оглядкой, т. к. все в конечном счете зависит от компилятора. Ведь задача перед ним стоит очень непростая: перевести строки на языке высокого уровня в команды контроллера, что для AVR ничуть не проще, чем для «настоящих» микропроцессоров, и потери тут неизбежны как в части компактности кода, так и времени его выполнения. Но это еще не самое главное.

Фирма IAR Systems в настоящее время предлагает серию пакетов Embedded Workbench для более чем двадцати типов МК различных фирм (под девизом «различные архитектуры — одно решение»). Здесь все рассчитано на то, чтобы человек, владеющий языком C, с минимальными потерями времени смог «пересесть» с одного на другой тип МК. В этом монструозном инструменте

все было бы, возможно, и здорово, если бы не цена, которая измеряется в тысячах американских «президентов». Конечно, кое-кто в нашей стране может не придать этому значения (и многие не придают), но сначала стоит рассмотреть саму целесообразность такого подхода, а она вызывает обоснованные сомнения у многих специалистов.

Упомянем также, что IAR Systems — не единственный разработчик компиляторов с языка С для AVR. Есть куда более изящные и не столь «навороченные» инструменты (например, ICC for AVR от фирмы ImageCraft, CodeVisionAVR от HP Infotech), но в любом случае реальная цена их начинается от сотен евро. И это оправданно, поскольку рабочие инструменты должны быть качественными, а потому дешевыми быть не могут. Но для полноты картины стоит упомянуть и бесплатный WinAVR (AVRGCC, **winavr.sourceforge.net**), который создан на основе компилятора GNU GCC и, соответственно, распространяется по лицензии GPL (вместе с исходными кодами), обладающий всеми недостатками и достоинствами «свободных» продуктов, на чем мы здесь не будем останавливаться. Этот продукт поддерживается AVR Studio, о которой далее.

С другой стороны, фирменный ассемблер (в том числе и вариант, работающий в графическом режиме под Windows) абсолютно бесплатен. Даже если вы пойдете на поводу у тех, кто не придает никакого значения факту наличия у программных продуктов некоторой стоимости или рискнете обратиться к WinAVR, то вам необходимо еще принять во внимание следующие соображения. Язык С надо специально и отдельно учить. И в любом случае никто вас также и не освободит от подробного изучения аппаратной части МК. Для начинающего все это в совокупности может оказаться слишком сложным.

А что, спросите вы, язык ассемблера учить не надо? Практически не надо, потому что ассемблер — это, вообще говоря, не язык, а просто несколько правил оформления последовательности команд МК в текстовом представлении в «понятную» компилятору программу. Основные правила изложены в этой главе далее. Фирменное описание ассемблера для AVR занимает несколько страничек (сравните с фолиантами для изучения С) и входит в AVR Studio, отдельно его можно скачать с сайта Atmel. Функциональность команд наизусть заучивать отнюдь не требуется, т. к. даже профессионалы во время работы кладут перед собой открытый справочник по командам. Причем для каждого типа МК и правила написания, и мнемоника одинаковых по содержанию команд может быть различной, но в принципе все ассемблеры похожи и переход от одного к другому не настолько трудоемок, как это стараются представить разработчики «многокилобаксовых» компиляторов с языка С. Куда больше времени занимает изучение самой структуры контроллеров и работы их отдельных узлов.

То, что для ассемблера выучить действительно потребуется — это реализация некоторых типовых приемов программирования, таких как различные циклы с условными и безусловными переходами, арифметические функции, деление на локальные и глобальные переменные и т. п. Это и служит основным аргументом в пользу языков высокого уровня, где эти вещи уже сделаны за вас. Я бы выразился так: выбор между С и ассемблером зависит от вашей предыдущей подготовки. Если вы вообще никогда не сталкивались с программированием, но разбираетесь в электронике и функционировании узлов МК, то с ассемблером вам будет освоиться намного легче, ведь вы просто напрямую взаимодействуете с этими самыми узлами. Если же, наоборот, вы легко программируете на С, и зачем-то вам потребовалось освоить контроллеры, то выбирайте С, собственно, для этого упомянутые компиляторы и создавались.

Заметки на полях

«Программирование без GOTO» — лозунг знаменитого Дейкстры, классика процедурного программирования, к ассемблеру, к сожалению, абсолютно неприложим. Дейкстра имел в виду, что большое количество операторов перехода на метку очень сильно усложняет чтение программы и ведет к лишним ошибкам (подобные программы Дейкстра называл «спагетти» — из-за многочисленных линий переходов, сопровождающих графическое представление такой программы в виде блок-схемы). Но после компиляции все эти хорошо знакомые знатокам Pascal циклы `while..do` или `repeat..until`, равно как и оператор выбора (`case`), и даже — с некоторыми нюансами — обращение к процедурам, все равно превращаются в набор условных или безусловных переходов на определенные адреса в памяти программ. В ассемблере это приходится делать, что называется, «ручками». Разумеется, считать адреса (как при программировании в машинных кодах в начале 1950-х) вручную не нужно, в программе просто ставится метка, как и в языках высокого уровня (только в ассемблере еще проще, чем даже в каноническом Бейсике — в случае процедуры-подпрограммы метка заодно служит ее именем). Тем не менее сложности тут могут быть, и все зависит от склада вашего ума, некоторым (как автору этих строк) в хитросплетениях условных и безусловных переходов разобраться не составляет особенного труда, и иногда такое представление кажется даже более наглядным. Другие от этого «стонут и плачут», и им, конечно, следует обращаться к языку С.

При первоначальном изучении микроконтроллеров я бы решительно рекомендовал ассемблер. Как писал упоминавшийся в начале главы классик программирования Дональд Кнут, «каждый, кто всерьез интересуется компьютерами, должен рано или поздно изучить по крайней мере один машинный язык». Ассемблер — это первично, а все остальное вторично. Но автор этих строк вовсе не является упрямым снобом и прекрасно понимает, что сколько-нибудь крупные проекты (когда число строк кода начинает измеряться тысячами и более) на ассемблере будет создавать весьма затруднительно,

по крайней мере, создавать «с нуля», не используя уже готовые модули. Но даже если вы в конце концов перейдете на C, советую эту книгу все же изучить до конца, чтобы познакомиться с AVR, а сам язык в приложении к AVR неплохо изложен в [4].

В этой книге вы встретите только ассемблерные программы. Мы ведь не будем писать для AVR операционные системы, и к тому же максимально попытаемся использовать готовые модули. А разобраться в работе этих модулей, когда они представлены в виде «голых» команд процессора, значительно проще. Не будем мы здесь и разбирать работу в AVR Studio — бесплатном пакете от Atmel, предназначенном для отладки программ[1] (впрочем, программы на ассемблере в нем можно и создавать, а вот на C — только отлаживать, если нет дополнительного компилятора из перечисленных ранее). Вообще-то этот пакет в некоторых случаях оказывается очень целесообразен, особенно в совокупности с различными отладочными модулями (т. н. «китами», Kit's, которые, в отличие от самого пакета, отнюдь не бесплатны). В нем удобно отлаживать сложные линейные (т. е. не использующие прерываний) процедуры, например, какое-нибудь деление четырехбайтных чисел. И если у вас хватит терпения и усидчивости, чтобы разобраться с еще одной «прослойкой» между вами и микросхемой, то вы ничего не потеряете. Но я не буду загромождать книгу не столь обязательными описаниями промежуточного софта, созданного исключительно для удобства, а покажу, как можно любую вашу схему превратить в отладочный модуль без лишних сложностей. Если вы заинтересовались AVR Studio, то вам сюда [3].

Заметки на полях

Автор этих строк — приверженец стиля разработки программ, который подозрительно напоминает широко известную в программистской среде «разработку, управляемую тестированием». Индустрия ПО приучает программистов к иному стилю: условно говоря, когда все создается «на бумаге». Сначала утверждается общий проект (для чего даже есть специальные люди — программные архитекторы), рисуются блок-схемы, разные программисты пишут отдельные куски кода (отлаживая их, конечно, но только «теоретически», без привязки к реальной рабочей среде), потом это объединяется в некий «проект», наконец, собирается альфа-версия и начинается тестирование готовой программы. Это привлекательный способ, особенно в случае разработки больших проектов, потому что он хорошо управляется в рамках стандартных бизнес-процессов. Именно в расчете на этот способ поточного производства программ создаются компиляторы со строжайшей проверкой типов данных и инструменты вроде AVR Studio, чтобы отловить мелкие ошибки по максимуму еще на начальной стадии. Но общие ошибки на уровне алгоритмов все равно отловить сразу не удается, и потом не удивляйтесь, откуда в конечном продукте низкоуровневые баги,

[1] Есть и еще более мощная штука — Algorithm Builder, который, в соответствии с новыми веяниями, позволяет проектировать программу в виде блок-схемы.

которые следовало бы исправить еще на стадии блок-схемы. Хороший пример дает скандал с гибелью американской станции Mars Climate Orbiter, произошедшей из-за нестыковки в единицах измерения длины различных модулей бортовой навигационной программы.

Способ «разработки, управляемой тестированием» (его еще иногда называют «экстремальным программированием») намного сложнее в организации, потому что предполагает определенный уровень подготовки и заинтересованности исполнителей, и с большим трудом может быть формализован в терминах календарных планов. При этом способе любой (в идеале) кусок кода тестируется сразу после написания прямо в рабочей среде, и конечный проект собирается из таких уже по максимуму отлаженных фрагментов. Остается только отладить их взаимодействие, что также выполняется немедленно по ходу сборки. Когда вы начинаете какую-то программу с неизвестным ранее алгоритмом — создайте сначала отдельную вспомогательную программу, где фрагменты этого алгоритма можно детально проверить. Это значительно дольше, чем писать программу сразу (и что еще важнее — здесь очень трудно рассчитать время заранее), но позволяет сэкономить кучу времени на последующем доведении программы до ума. Именно для такой цели и удобна система, которую я опишу в *главе 16*, когда вы имеете немедленную обратную связь от вашего устройства.

Обратите внимание на следующий момент: в этой концепции документация пишется не до, а *после* создания и отладки собственно программы. Это полезно вдвойне, поскольку документацию не приходится непрерывно править и в ней оказывается меньше ошибок, а программа при написании документации к ней лишний раз анализируется, и очень часто бывает, что на этой стадии в ней также обнаруживаются незамеченные ошибки и недоработки. Поэтому мой совет всем без исключения разработчикам таков: даже если вы делаете устройство в одном экземпляре и для собственного удовольствия, всегда потом составляйте детальное описание программы. Это пригодится не только для того, чтобы что-то «починить» или усовершенствовать спустя некоторое время, когда детали забудутся, но и для того, чтобы лучше «отловить» ошибки и добавить забытые, но необходимые «фичи». По аналогичным причинам текст программы обязательно следует сопровождать комментариями.

Обустройство ассемблера

Если у вас еще нет AVR-ассемблера (т. е. самой программы-компилятора), то, возможно, AVR Studio вам придется установить, т. к. это самый надежный способ добыть последнюю версию ассемблера. Скачайте AVR Studio с сайта Atmel (например, отсюда **http://atmel.ru/Software/Software.htm**, к сожалению, последние версии пакета стали довольно «монструозными» — более 40 Мбайт), установите его, разыщите файл avrasm32.exe или более современный avrasm2.exe (он находится в папке ...\Atmel\AVR Tools\ AvrAssembler) и скопируйте его в подготовленную заранее другую папку. Оттуда же целиком целесообразно скопировать папку Appnotes, во-первых, из-за собственно «аппнот», которые представляют собой рекомендации по реализации отдельных функций и выгодно отличаются от представленных

на сайте Atmel форматом ASM. По сути, это готовые модули для встраивания в вашу программу (на сайте они находятся в формате PDF, а исходные коды приходится скачивать отдельно). Учтите, что в них могут быть ошибки, о чем мы еще будем упоминать, и применять их надо с оглядкой.

Во-вторых, в этой папке находятся файлы макроопределений (с расширением inc) для всех моделей AVR, поддерживаемых данной версией ассемблера, даже для тех, что уже не выпускаются. Если не скопировать папку, их пришлось бы скачивать с сайта Atmel поодиночке, а для старых типов искать где-то на стороне. Эти файлы необходимы, иначе вы не сможете откомпилировать ни одной программы, поскольку сам ассемблер абсолютно не подозревает о существовании таких вещей, как `PortA` или регистр `DDRC`, а «знает» только числовые адреса соответствующих регистров. Соответствие между этими мнемоническими обозначениями и адресами и устанавливается с помощью inc-файлов, причем для разных моделей эти адреса могут различаться. Можно на всякий случай извлечь также описание AVR-ассемблера в формате CHM (на английском, естественно). После копирования саму AVR Studio можно удалить.

Ранее существовал и довольно примитивный ассемблер под Windows (wavrasm.exe), но затем, видимо, в корпорации решили его не развивать, обойдясь AVR Studio. Так что нам, сермяжным, придется обустроить среду для создания программ самостоятельно — это делается один раз, и потом вы вообще сможете забыть про существование avrasm.exe. Для этого потребуется еще как минимум текстовый редактор. Можно обойтись Блокнотом или многочисленными его более функциональными заменителями, вроде Edit Plus (MS Word не подойдет решительно), но тогда нам придется отдельно каждый раз запускать процесс компиляции из командной строки. Обычно это делается с помощью FAR или других клонов Norton Commander, что неудобно и сильно замедляет процесс, потому лучше использовать специальный редактор, который позволит запускать процесс компиляции прямо из своего окна, и к тому же имеет возможность «подсветки» синтаксиса. Так как специально для AVR редакторов никто не делает, они чаще всего по умолчанию «заточены» под Intel-ассемблер, но «подсветку» нередко можно настроить «под себя».

Заметки на полях

Редакторов для написания ассемблерного кода довольно много, как правило, они в той или иной степени «самодеятельные» и бесплатные (исключение — очень профессиональный и известный с давних времен, но платный MultiEdit). Тут важно только выбрать самый удобный, иначе можно попасть в ситуацию, когда будет еще хуже, чем с Блокнотом. Например, широко разрекламированный на множестве ресурсов ASMEdit (некоего Матвеева), первое, что у меня

сделал — еще в процессе инсталляции «повесил» Windows 98[2] до полной неработоспособности, а когда я все же «достучался» до исполняемого файла, то запустить его оказалось невозможным, т. к. окно свернулось в углу экрана в маленький квадратик и распахиваться не желало. Я разыскал более старую версию (ASMEdit 1.2), она установилась нормально (если не считать грамматических ошибок в инсталляторе), и тут выяснилось, что: а) настройка запуска компиляции из командной строки настолько сложна, что требует чуть ли не написания отдельной программы; б) настройка «подсветки» синтаксиса крайне примитивна. К тому же программа без спроса ассоциирует с собой расширение asm и «замусоривает» перечень ассоциаций еще полудюжиной расширений для неведомых целей, которые потом приходится вычищать вручную. Я это так подробно рассказываю потому, что у этого редактора есть одно удобное свойство: сообщения компилятора перенаправляются в окно редактора, как это было в wavrasm, и не требуется просматривать отдельные консольные окна. Если вам удастся «справиться» с ASMEdit, то вам сильно повезло.

Я перебрал в свое время несколько программ, но ни одна меня не устроила в такой степени, как творение некоего Анатолия Вознюка, которое я использую с 1999 года. Сам Анатолий знаменит также своим шедевром под названием Small CD Writer, давно скрывается инкогнито, сайт его больше не откликается, а программы не развиваются (в частности, Small CD Writer так и «не умеет» корректно писать на DVD). Но в разбираемом редакторе под названием ASM Editor (последняя версия 2.2), собственно, больше и развивать нечего. Доступен он на множестве «софтоотстойников» в Интернете, только не перепутайте его с упомянутым ASMEdit.

Для начала работы нам предварительно надо создать командный файл. Предположим, у вас файл avrasm32.exe находится в созданной вами папке c:\avrtools. Запустите Блокнот и введите в него следующий текст (соответственно измените путь, если папка другая):

```
c:\avrtools\avrasm32 -fI %1.asm
```

Сохраните этот файл под именем, например, asm32.bat в той же папке, что и avrasm32.exe. Теперь запустите ASM Editor, выберите в меню пункт **Service | Properties** и в появившемся окне найдите вкладку **Project**, а в ней — строку Assemble ASM file (рис. 13.5). В эту строку, как и показано на рисунке, внесите путь к нашему bat-файлу. Больше ничего настраивать не требуется, остальные пункты можно оставить, как есть, или очистить — по желанию. Теперь по нажатию сочетания клавиш <Alt>+<A>, вне зависимости от того, где находится файл с редактируемой программой, он скомпилируется, и результат (т. н. hex-файл, о чем далее), если нет ошибок, окажется в той же папке, где и исходный текст.

[2] Если у вас есть такая возможность, то для написания ассемблерных программ и программирования контроллеров лучше использовать ее, а не XP и не Vista — меньше проблем на вашу голову.

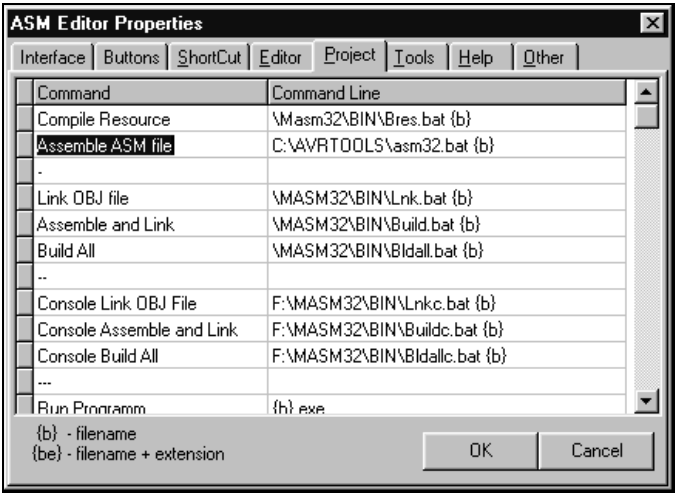


Рис. 13.5. Настройка редактора ASM Editor

Одновременно с компиляцией отдельно возникает консольное (с белыми надписями на черном фоне) окно с сообщениями компилятора. Сразу смотрите на последние строки, где должна быть надпись «Assembly complete with no errors» («ассемблирование выполнено без ошибок»). Если такая надпись есть, ищите готовый hex-файл в папке с исходным текстом. В противном случае hex-файла не окажется (а предыдущий его вариант, если он был, будет стерт), а в этом окне появятся номера строк в исходном тексте с ошибками и примерным описанием проблемы в данной строке (например, сообщение «Undefined variable reference» — «ссылка на неопределенную переменную», означает, что у вас в данной строке идентификатор, который нигде не определен).

Кроме этого, при успешной компиляции в окне сообщений можно посмотреть точный размер скомпилированной программы в словах (words), для получения размера в байтах надо умножить это число на два. Отметим сразу, что объем hex-файла размеру программы соответствовать не будет (подробнее см. далее), так что узнать объем занимаемой памяти можно только отсюда.

Структура программы AVR

Как вы уже знаете, при работе МК последовательно выполняет команды программы, имеющейся в памяти. Программист может менять порядок выполнения команд, организуя циклы и различные переходы. Одно из самых мощных средств программирования — вызов *подпрограмм* или *процедур* (в Pascal

подпрограммы делятся на процедуры и функции, а в языке С есть только функции, но это дела не меняет — в принципе это все одно и то же), т. е. кусков кода, которые могут использоваться неоднократно. Во всех ассемблерах вызов процедур предусмотрен обязательно.

Естественно, программу сначала нужно в память записать, причем записать совершенно определенным образом, так, чтобы МК «знал», откуда начинать при включении питания или после подачи импульса Reset. Это его «знание» в случае современных МК AVR также программируется, однако для простоты будем считать, что программа всегда начинает читаться с самой первой ячейки памяти программ, т. е. с нулевого адреса. (А вот для классического процессора семейства i86 это не так. По умолчанию он всегда начинал с команды, расположенной за 16 байт до конца первого мегабайта памяти, т. е. по адресу FFFF0h. К тому же, адрес этот можно изменять, но мы этим здесь заниматься не будем.) Исходя из этих обстоятельств, программа должна иметь определенную структуру.

По начальному (нулевому) адресу всегда располагается одна и та же команда безусловного перехода (по-английски `jump` — «прыжок»), которая записывается так:

```
rjmp RESET
```

Или так:

```
jmp RESET
```

Отметьте себе на память, что AVR-ассемблер, в отличие от канонического С, регистры букв не различает (одинаково правильной будет форма записи `Jmp`, `JMP` и `jmp`, так же как `Reset`, `RESET` и `reset`).

Форма написания (`jmp` или `rjmp`) зависит от типа контроллера: если в нем объем памяти программ меньше или равен 8 кбайт, то всегда (и не только в этом случае) используется команда `rjmp` (relative jump, т. е. «относительный безусловный переход»). Она занимает в памяти два байта, как и практически все остальные команды AVR, о чем мы еще поговорим. Код самой команды в этих двух байтах занимает старшую тетраду старшего байта (т. е. четыре бита), остальные 12 бит представляют собой адрес, куда переходить. В данном случае компилятор подставит адрес команды, следующей сразу за меткой `RESET`, с которой и начнется собственно выполнение программы. Метка с этим именем, естественно, всегда должна присутствовать, но может быть расположена уже в любом другом удобном месте программы, за исключением еще нескольких первых адресов, о назначении которых далее. Метка, кстати, может называться и не `RESET`, а любым другим именем, просто так принято для удобства чтения: хотите найти в любой программе ее начало — ищите метку `RESET`.

Вернемся к форме записи команды. 12 бит адреса могут представлять 4096 различных адресов. Так как единицей объема памяти программ служит слово из двух байтов (а не отдельный байт), как раз из-за того, что любая команда занимает не менее чем два байта, то общий объем адресуемой таким образом памяти и составит 8 кбайт. А вот если памяти больше, то приходится использовать команду `jmp` (абсолютный безусловный переход). Она состоит из четырех байт, в которых адрес займет 22 бита, и потому с ее помощью можно адресовать до 4 М (8 Мбайт) памяти. Мы условимся, что старшие модели семейства Mega применять не будем, и потому ограничимся командой `rjmp`.

Подведем некоторый итог: мы уже узнали многое о структуре типовой программы для AVR. Она должна начинаться с безусловного перехода на метку `RESET`, а само начало этой программы будет располагаться сразу после этой метки где-то в другом месте программы. А почему так странно? Нельзя ли начать прямо сразу с нулевого адреса, строка за строкой, зачем нужны какие-то переходы? Можно, отвечают нам авторы описаний AVR. Простейшая программа может начинаться с нулевого адреса без переходов, но только в одном случае: если мы обязываемся отказаться от прерываний. А без них контроллер теряет 90% своей функциональности. Конечно, можно придумать пример программы, которая бы делала что-нибудь полезное, но при этом работала совсем без прерываний. Но на практике без прерываний обходятся только тогда, когда штатных прерываний хватает (например, для отслеживания состояния большого количества кнопок), или когда без них программа получается компактнее (такие случаи мы еще встретим).

Обработка прерываний

AVR по умолчанию ожидает, что сразу после первой команды с адресом $0000 идет таблица т. н. *векторов прерываний*. Вектор — это просто отсылка по нужному адресу с помощью команды `rjmp`. Адрес обозначается меткой, может располагаться в любом месте программы, и содержит начало процедуры обработки прерывания. Первый вектор располагается по адресу $0001 (а для МК с памятью более 8 К — по адресу $0002, потому что по адресу $0001 находится вторая половина более длинной команды `jmp`), причем адрес этот означает номер двухбайтного слова в памяти, а не отдельного байта (оно в два раза меньше количества байт). Попутно заметим, адреса в SRAM (ОЗУ) именно байтовые, а не пословные (см. далее). На самом деле в режиме по умолчанию нам вообще не нужно думать про абсолютные адреса и их нумерацию: первая команда программы (`rjmp RESET`), которая еще называется *вектором сброса* (или *вектором начальной загрузки*), автоматически расположится по нулевому адресу, вторая — по адресу $0001 и т. д. Найдя какую-

нибудь команду перехода по метке, компилятор автоматически подставит абсолютные адреса.

Порядок векторов и их количество в таблице жестко задано в соответствии с типом МК. Потому самое первое, что вы должны сделать, приступая к программированию, — открыть руководство по применению выбранного типа контроллеров и скопировать оттуда эту таблицу. Можно попытаться сделать это прямо через буфер обмена из PDF-описания или преобразовав его в другой формат, так меньше вероятность что-то пропустить, только придется потом удалить указанные там абсолютные адреса, стоящие в начале каждой строки. Вот как будет выглядеть заготовка начала программы для МК ATmega8 (листинг 13.1).

Листинг 13.1

```
;=========прерывания================
rjmp RESET ; Reset Handler
rjmp EXT_INT0 ; IRQ0 Handler
rjmp EXT_INT1 ; IRQ1 Handler
rjmp TIM2_COMP ; Timer2 Compare Handler
rjmp TIM2_OVF ; Timer2 Overflow Handler
rjmp TIM1_CAPT ; Timer1 Capture Handler
rjmp TIM1_COMPA ; Timer1 CompareA Handler
rjmp TIM1_COMPB ; Timer1 CompareB Handler
rjmp TIM1_OVF ; Timer1 Overflow Handler
rjmp TIM0_OVF ; Timer0 Overflow Handler
rjmp SPI_STC ; SPI Transfer Complete Handler
rjmp USART_RXC ; USART RX Complete Handler
rjmp USART_UDRE ; UDR Empty Handler
rjmp USART_TXC ; USART TX Complete Handler
rjmp ADC ; ADC Conversion Complete Handler
rjmp EE_RDY ; EEPROM Ready Handler
rjmp ANA_COMP ; Analog Comparator Handler
rjmp TWSI ; Two-wire Serial Interface Handler
rjmp SPM_RDY ; Store Program Memory Ready Handler
;=======================================
```

Обратите внимание, что в ассемблерной программе каждый оператор занимает отдельную строку (в большинстве языков высокого уровня операторы можно записывать в одной строке, например, разделяя их знаками препинания; здесь это не допускается). Символы пробелов и табуляции могут встречаться в произвольном количестве, и они полностью игнорируются, за исключением двух случаев: нельзя разбивать идентификатор на части и,

наоборот, нельзя соединять разные идентификаторы между собой; хотя бы один пробел, знак табуляции (или знак препинания, если это предусмотрено форматом команды) между наименованиями инструкций, переменных, регистров и т. п. должен быть.

Имена меток, конечно, можно присваивать произвольно. Как вы уже поняли, после точки с запятой идет комментарий — произвольная строка, на которую компилятор не обращает внимания. На новую строку комментарий не переходит: в случае длинного комментария в начале каждой следующей строки также надо ставить точку с запятой.

Но постойте, мы что, обязаны использовать *все* прерывания? Конечно, нет. Для неиспользуемых прерываний вместо команды `rjmp <метка>` следует поставить `reti` — выход из прерывания («return interrupt»). На самом деле здесь возможна и команда `nop` — пустая операция («no operation»). Я пробовал из любопытства, — все отлично работает. Но руководство рекомендует именно `reti`, т. к. при случайной инициализации прерывания оно все равно не будет выполняться, а команда `nop` в этом случае, очень вероятно, привела бы к «зависанию» программы либо выполнению совсем других, не предусмотренных нами, операций.

Лично я оформляю начало программы следующим образом: копирую таблицу в указанном ранее виде, а затем в начале каждой строки (кроме первой) автоматически меняю с помощью функции поиска-замены редактора «`rjmp`» на «`reti; rjmp`», вот так (старую команду я закомментировал):

```
rjmp RESET ; Reset Handler
reti ;rjmp EXT_INT0 ; IRQ0 Handler
reti ;rjmp EXT_INT1 ; IRQ1 Handler
. . . . .
```

Теперь заготовка начала программы готова: при необходимости в дальнейшем какого-нибудь прерывания, мы удаляем из нужной строки фрагмент `reti;`, а затем где-то в программе перед началом процедуры обработки этого прерывания ставим нужную метку, например:

```
rjmp RESET ; Reset Handler
rjmp EXT_INT0 ; IRQ0 Handler
. . . . .
EXT_INT0: ;процедура обработки прерывания INT0
. . . . .
reti ;окончание процедуры обработки прерывания INT0
```

Обратите внимание, что после метки в программе (но не внутри команды, которая на нее ссылается!) должно идти двоеточие, иначе компилятор не «поймет», что перед ним именно метка. Кроме того, меток не касается правило написания операторов в отдельных строках (потому что метка не оператор, а адрес), после метки и двоеточия в программе сразу может идти оператор.

Процедура *RESET*

Теперь обратимся к процедуре RESET, т. е. к истинному началу программы, что там должно быть? Когда контроллер доходит до команды вызова подпрограммы (call или rcall, см. далее), или в нем происходит прерывание, он должен сохранить состояние программного счетчика с тем, чтобы потом «знать», куда вернуться. Это происходит в специальной области памяти, которая называется *стеком* (stack). Потому в любой программе на AVR-ассемблере для семейств Mega и Classic первыми после метки RESET должны идти следующие строки:

```
RESET:
      ldi   temp,low(RAMEND) ;загрузка указателя стека
      out   SPL,temp
      ldi   temp,high(RAMEND) ;загрузка указателя стека
        out   SPH,temp
```

Этими операторами вы указываете, где компилятору расположить программный стек, а именно в конце SRAM (что обозначается константой RAMEND, объявленной в соответствующем inc-файле, загляните). Служебные слова low и high означают, что от RAMEND нужно взять, соответственно, младший и старший байты. Для тех моделей, у которых объем SRAM не превышает 256 байт (из доступных в настоящее время это только модель 2313 во всех ее версиях), запись сокращается:

```
RESET:
      ldi   temp,RAMEND ;загрузка указателя стека
      out   SPL,temp
```

Заметки на полях

У многих представителей семейства Tuny (кроме, насколько я знаю, Tuny2313), в том числе тех, у которых объем SRAM также не превышает 256 байт, стек аппаратный, а не программный, потому там его расположение специально указывать не требуется (отметим, что в [2] в этой части допущена ошибка). По этой причине, в частности, для семейства Tuny не определены команды push и pop, которые для остальных моделей позволяют временно сохранять в стеке значение какого-либо регистра (а для некоторых процессоров, вроде i8086, в которых регистров общего назначения мало, это вообще одни из самых употребительных команд). Размер же аппаратного стека строго фиксирован, и в нем умещается только адрес возврата из подпрограммы или обработчика прерывания.

После задания стека желательно поставить следующие строки:

```
ldi temp,1<<ACD
out ACSR,temp   ;выкл. аналог. компаратор
```

Почему желательно, а не обязательно? Потому что по умолчанию аналоговый компаратор всегда включен и, соответственно, расходует питание. Если

вы его не используете, то зачем лишнее потребление? Правда, это практически не скажется на потреблении в нормальном режиме работы, т. к. доля компаратора здесь очень мала. Вставка этих строк становится критичной только в том случае, если применяются режимы энергосбережения. С другой стороны, если компаратор нужен (см. *главу 14*), то, конечно, эти строки следует исключить.

После всего этого в процедуре RESET обычно идет секция инициализации, где разрешаются прерывания, устанавливаются состояния выводов портов, инициализируются начальные значения переменных и т. п. Примеры мы еще встретим в тексте этой книги неоднократно. Эта секция обязательно должна заканчиваться командой sei — общим разрешением прерываний, т. к. по умолчанию они запрещены.

Теперь вроде бы все готово к работе, но что будет делать контроллер в ожидании прерываний? Ведь основная функциональность большинства микропрограмм сосредоточена именно в их обработчиках, и в простейшем случае контроллер попросту ничего не должен делать, пока не придет сигнал очередного прерывания. Поэтому простейшая программа должна заканчиваться пустым циклом:

```
. . . . .
sei ;разрешаем прерывания
STOP:
rjmp STOP
```

На самом деле в этом цикле можно делать и что-то полезное, например, войти в один из режимов энергосбережения, или отслеживать изменение состояния какого-то вывода, или момент прихода байта через UART и т. п. — в дальнейшем мы увидим примеры подобных действий. Именно внутри такого цикла работают немногочисленные программы, вообще не использующие прерываний.

Определения переменных, констант и подключение внешних файлов

Определения играют в процессе написания ассемблерной программы важную роль, не менее важную, чем секция объявления переменных в программе на языке высокого уровня. Сравните две записи:

```
. . . . .
mov   r16,r11
cpi   r16,$11
. . . . .
```

и

```
. . . . .
mov   temp,counter
cpi    temp,max_value
. . . . .
```

И та, и другая запись верна, но вторая значительно более «читабельна», правда? Одно дело иметь программу, в которой фигурируют регистры (еще слава богу, что не их абсолютные адреса) и числа, совсем другое — работать с «говорящими» именами переменных и констант. Идентификатором `temp` (от слова «temporary» — временный) обычно обозначают рабочую переменную, которая не предназначена для долговременного хранения данных, `counter` — в переводе «счетчик», `max_value` — «максимальное значение». Но если команды компилятор «знает» (мнемоника команд фиксирована и их смысл мы расшифруем немного далее), то как ему узнать, что мы имели в виду под этими самыми `temp` и `counter`?

Для этого служит секция определений имен переменных и констант, которую чаще всего располагают в самом начале текста программы, еще до векторов прерываний, или в отдельном внешнем файле, если определений много, и они очень загромождают текст. Именно такие макроопределения и содержат те самые файлы с расширением inc, которые мы копировали из AVR Studio, — для каждого контроллера свои. В результате чего мы можем писать `mov GIMSK,temp` вместо `mov $3F,r16`.

Для того чтобы компилятор нашел эти определения, inc-файл нужно подключить к нашей программе. Это делается с помощью директивы `include`, как в языке C, например (обратите внимание на точку перед `include`):

```
.include «2313def.inc»
```

В данном случае подключается файл с макроопределениями для контроллера AT90S2313. Подобной строкой должен начинаться текст любой программы. Причем файл должен соответствовать выбранному контроллеру, иначе программа может и не заработать: иногда на ошибку вам укажут при компиляции, иногда это выявится только во время исполнения.

Отметим, что имя файла здесь может быть абсолютно произвольным, — по директиве `include` компилятор, «не думая», разыщет файл с указанным именем (разумеется, он должен находиться в текущем каталоге, или для него должен быть указан полный путь), скопирует из него текст и вставит этот текст в том месте вашей программы, где расположена директива. И только потом начнет «разбираться». С директивами `include` надо быть аккуратным, если файл содержит команды, а не только определения, то вставлять его

нужно уже не в начале текста, а после всех векторов прерываний, иначе выполнение программы начнется с него.

Но имени `temp` (не говоря уж о `counter`) вы в inc-файлах не найдете — это нестандартное определение и относится уже к нашей «епархии». Создать свои определения можно с помощью директив `def` (definitions) для переменных и `equ` (equvalent) для констант вот так:

```
.equ    max_value = $11 ;максимальное значение = 17
.def    temp = r16 ;регистр r16 есть переменная temp
.def    counter = r05 ;регистр r05 есть переменная counter
```

Эти определения также обычно располагают в начале текста программы. Теперь компилятор разберется, что к чему в нашем примере ранее. Только учтите, что никаких проверок, кроме синтаксических, тут не выполняется, потому можно объявить два разных имени для одного регистра, и они будут восприниматься как синонимы:

```
.def    temp = r16 ;регистр r16 есть переменная temp
.def    counter = r16 ;регистр r16 есть переменная counter
```

Изменение `temp` будет автоматически означать изменение `counter` и наоборот. Иногда этим пользуются, если в разных частях программы один регистр применяется для разных по смыслу вещей (и вы можете встретить такие примеры в фирменных «аппнотах»). В общем случае к такой возможности следует прибегать лишь в исключительных случаях — слишком много ошибок можно наделать.

С помощью директивы `equ`, вообще говоря, можно определять довольно сложные выражения, но этим пользуются редко, гораздо чаще ее применяют для определения переменных, которые располагаются не в регистрах, а в области SRAM. Например, следующая последовательность директив и команд запишет содержимое регистра `counter` в SRAM по адресу `$60` (для большинства моделей, кроме старших Mega, это первый свободный адрес после занятых адресами регистров):

```
.equ memory_address = $60
. . . . .
clr ZH
ldi ZL,memory_address
st Z,counter
```

В заключение темы еще раз напомним, что имена переменных (как и команд, и меток) нечувствительны к регистру букв, что еще один плюс для AVR-ассемблера по отношению к языку C, где они по умолчанию различаются, и у начинающих это приводит к множеству ошибок.

Система команд AVR

С некоторыми командами мы уже познакомились, но еще не знаем толком, что они означают. Теперь я постараюсь разобрать основные команды и их использование. Подчеркну, что речь идет только о части самых употребительных и концептуально важных команд, потому что всего команд для AVR от 90 до 133 (в зависимости от контроллера), и только их подробное описание, например в [2], занимает почти 70 страниц, гораздо больше, чем описание собственно AVR-ассемблера. С теми командами, которые выпадут из рассмотрения здесь, мы познакомимся по ходу дела в дальнейшем. Выборочный перечень базовых команд с их описанием, достаточный для составления большинства законченных программ, вы также найдете в *Приложении 4*. Однако в любом случае при написании собственных программ вам будет необходимо иметь справочник по командам, в роли которого может выступать как [1, 2], так и англоязычный документ PDF с описанием конкретного контроллера, который легко добывается с сайта Atmel. Учтите, что в неофициальных пособиях могут быть ошибки, так в [1, 2] они встречаются в достаточном количестве, хотя в последних изданиях их, возможно, и исправили.

Формат команды

Прежде всего, заметим, что команды (инструкции, операции — мы будем употреблять эти слова как синонимы) контроллера записываются в определенном формате. Сначала идет собственно мнемоническое название команды, которое может состоять из двух (`ld`, `st`), трех (`mov`, `ldi`, `sei`, `add`) или четырех (`breq`, `brne`, `sbiw`) символов. Затем после пробела (любого их числа, допустимы также знаки табуляции), если это предусмотрено форматом команды, идут операнды: имена регистров или константы (а также выражения), над которыми производится операция (на самом деле это не имена, а адреса, но мы не будем забывать про включение inc-файлов и создание собственных определений, и оперировать только с именами, так будет гораздо меньше ошибок).

Если операндов два (больше не бывает), то они перечисляются через запятую (с любым числом пробелов до или после нее, или вообще без них). Причем тут действует важное правило: во всех ассемблерах второй операнд является первым по смыслу действия (т. е. источником), а первый — приемником (результатом). Например, команда `ldi temp,1` расшифровывается, как «загрузить единицу в регистр `temp`», а операция `sub temp,counter` вычитает `counter` из `temp`, а не наоборот, и результат помещает также в `temp`.

Заметки на полях

Такая запись (действие — приемник — источник) стандартна для всех ассемблеров, и называется *прямой польской записью*: когда сначала просят налить чай (действие), потом указывают, в какой стакан (первый операнд, он же результат), и только после этого сообщают, где находится чайник (второй операнд). В отличие от обычной *инфиксной*, когда сначала указывается стакан (первый операнд), в который нужно налить чай (действие), ах да, вот из этого чайника (второй операнд). Если кто-то пробовал пользоваться некоторыми старыми моделями настольных калькуляторов, то там использовалась *обратная* польская запись, когда сначала указываются оба операнда, а потом действие (стакан и чайник, затем указывается, что нужно налить из второго в первый). Это наиболее удобно для компьютерных устройств (потому что позволяет без лишних сложностей реализовать операции вычисления по формулам со скобками любой степени вложенности), но отнюдь не для человека, т. к. счет на таких калькуляторах требовал специальной тренировки. В то же время ассемблерный порядок операндов обычно не доставляет трудностей при обучении, и довольно логичен: сначала главный операнд, который будет изменяться, а потом тот, с помощью которого и производятся изменения.

Выходные файлы

Есть, конечно, команды, которые вообще не требуют операндов (как уже знакомые нам `nop` или `sei`), или используют всего один операнд (`clr`), но вне зависимости от этого каждая команда AVR будет занимать в памяти ровно два байта (кроме немногих команд, вроде упоминавшейся `jmp`, которые работают с операндами большого размера, оттого занимают четыре байта). Такое готовое представление — командное слово, или код операции (КОП) — и записывается в виде числа компилятором в выходной файл, имеющий расширение hex, который необходим программатору для дальнейшей записи в контроллер. Кроме HEX-файлов, есть еще и другие форматы записи готовых программ (самый известный — бинарный), но hex-формат для микроконтроллеров самый распространенный, и мы будем рассматривать только его.

Рассматриваемый HEX-формат придуман фирмой Intel (есть и другие «гексы»). Отличается такой формат тем, что содержит числа в текстовом представлении (в шестнадцатеричной записи). Поэтому в случае чего его можно даже править в обычном текстовом редакторе (что исключено для бинарных файлов, содержащих числа, а не символы). Кстати, точно в таком же формате осуществляется запись констант в EEPROM, если это требуется.

Рассмотрим HEX-формат подробнее. На рис. 13.6 представлен файл короткой программы, открытый в обычном Блокноте. На первый взгляд тут «сам черт ногу сломит», но на самом деле все достаточно просто, хотя чтение затрудняется тем, что строки не поделены на отдельные байты. Разбираться

будет проще, если вы скопируете этот файл под другим именем и расставите в нем пробелы после каждой пары символов. Мы же рассмотрим данный файл как есть.

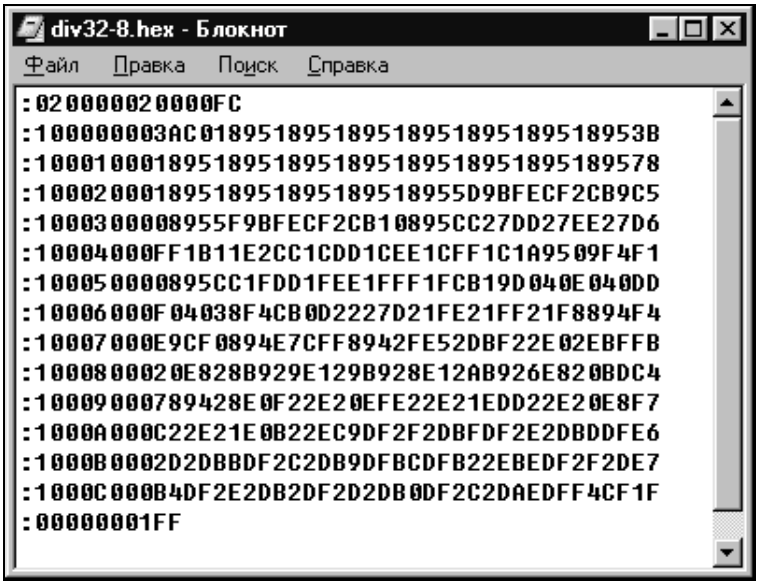


Рис. 13.6. Файл формата HEX в Блокноте

Основную часть файла занимают информационные строки, содержащие непосредственно КОП. Они состоят из ряда служебных полей и собственно данных. Каждая строка начинается двоеточием, после которой идет число байт в строке. Кроме первой и последней, везде стоит, как видите, число 10 (десятичное 16), т. е. в каждой строке будет ровно 16 информационных байт (исключая служебные). После количества байт идет два байта адреса памяти, указывающие куда писать (в первой строке 0000, во второй это будет, естественно, 0010, т. е. предыдущий адрес плюс 16, и т. д.). Наконец, после адреса идет еще один служебный байт, обозначающий тип данных, который в информационных строках равен 00 (а в первой и последней — 02 и 01, о чем далее). Только после этого начинаются собственно байты данных, которые означают соответствующие КОП, записанные пословно (КОП для AVR, напоминаю, занимают в основном два байта, и память в этих МК также организована пословно), причем так, что младший байт идет первым. Таким образом, запись в первой информационной строке «3AC0» в привычном нам «арабском» порядке, когда самый старший разряд располагается слева,

должна выглядеть, как C03A (если помните, я в *главе 7* упоминал об этой несуразице).

В первой строке служебный байт типа данных равен 02, и это означает, что данные в ней представляют сегмент памяти, с которого должна начинаться запись (в данном случае 0000). Заканчивается hex-файл всегда строкой «:00000001FF» — значение типа данных 01 означает конец записи, и данных больше не ожидается. А что означает FF?

Самым последним байтом в каждой строке идет контрольная сумма (дополнение до 2) для всех остальных байт строки, включая служебные. Алгоритм ее вычисления очень простой: нужно вычесть из числа 256 значения всех байтов строки (можно не обращая внимание на перенос), и взять младший байт результата. Соответственно, проверка целостности строки еще проще: нужно сложить значения всех байтов (включая контрольную сумму), и младший байт результата должен равняться нулю. Так, в первой строке число информационных байтов всего два (первое значение в строке), плюс служебный байт типа данных, равный также двум, итого контрольная сумма всегда равна 256 – 2 – 2 = 252 = $FC. В последней строке одни нули, кроме типа данных, равного 1, соответственно, контрольная сумма равна 256 – 1 = 255 = $FF.

Теперь попробуем немного расшифровать данные. Первое слово в первой информационной строке, как мы выяснили, равно $C03A. Если мы возьмем фирменное описание команд, то обнаружим, что значению старшей тетрады в КОП, равной $C (1100 в двоичной системе), соответствует команда `rjmp`, но ведь так и должно быть, мы же договорились, что любая программа начинается с безусловного перехода на метку `RESET`. Теперь очевидно, что остальные биты в этом значении ($03A) представляют абсолютный адрес в программе, где в ее тексте стояла метка `RESET`. Попробуем его найти, для этого вспомним, что адреса отсчитываются по словам, а не по байтам, т. е. число $3A (58) надо умножить на 2 (получится 116 = $74), и искать в этой области. Разыщем строку с адресом $0070, отсчитаем три пары байт от начала данных, и найдем там фрагмент «F894», который в нормальной записи будет выглядеть, как $94F8, а это, как легко убедиться по справочнику, есть код команды `cli`, запрещающей прерывания (которая в начале программы лишняя, т. к. они все равно запрещены, но, видимо, поставлена на всякий случай). Следующая команда будет начинаться с байта $E5, и первая тетрада в ней обозначает код команды `ldi` (1110 — проверьте!), а пятерка, очевидно, есть фрагмент адреса конца памяти (`RAMEND`), который в силу довольно сложного формата записи получается на самом деле равным $025F (см. значение младшего байта, равное $2F). Такое значение соответствует значению `RAMEND`, определенному в inc-файлах для МК

с 512 байтами встроенного ОЗУ[3]. Все, как и должно быть. Если мы обратим внимание опять на первую-вторую строки с данными, то увидим повторяющийся фрагмент «1895», который, как легко догадаться из материала предыдущего раздела, должен быть командой `reti` (если проверите по справочнику, то так оно и окажется).

Как видите, разобраться довольно сложно, но при некотором навыке и наличии под рукой таблицы двоичных кодов команд вполне возможно. Именно так работает программа, которая превращает код обратно в текст — *дизассемблер* (он входит в AVR Studio). А зачем это может понадобиться на практике? Дело в том, что в памяти программ часто хранят те константы, которые предположительно не будут изменяться в процессе эксплуатации, например, устанавливаемые по умолчанию значения какой-то величины. Но, разумеется, по истечении некоторого времени эти константы обязательно захочется изменить. И если у вас текст программы по каким-то причинам отсутствует (например, программа взята из публикации в журнале или скачана с радиолюбительского сайта), а загрузочный hex-файл имеется, то всегда можно «хакнуть» исходный код, и немного подправить его под свои нужды.

Команды перехода (передачи управления)

В языках высокого уровня была всего одна команда перехода на метку (GOTO), и то Дейкстра на нее «набросился». А в ассемблере AVR таких команд пруд пруди, целых 33 шутки! Зачем? На самом деле без доброй половины из них, если не больше, можно обойтись во всех жизненных случаях, т. к. они в значительной степени взаимозаменяемы. Разнообразие это, если угодно, дань памяти великому программисту — для повышения читаемости программ. Мы рассмотрим только ключевые команды из этого перечня.

С командами безусловного перехода `rjmp` и `jmp` мы уже познакомились достаточно подробно, так что сразу перейдем к вызову процедур `rcall` и `call` (официальный язык фирменных руководств Atmel предпочитает вместо процедуры упоминавшийся нами консервативный термин — подпрограмма, subroutine). Синтаксис у них точно такой же, как у команд безусловного перехода, и, по сути, это тот же самый переход по метке. И разница между этими двумя командами аналогичная: `call` работает в пределах 64 К адресов памяти (или до 8 М в соответствующем контроллере, поддерживающем такой

[3] $025F = 607, т. е. всего адресов 608, из которых 96 ($5F) занимают регистры, итого получается 512 ($0200) незанятых байтовых ячеек, составляющих ОЗУ.

объем адресного пространства), занимает 4 байта и выполняется за 4 цикла, а `rcall` годится только для МК с объемом памяти не более 8 кбайт, но занимает 2 байта и выполняется за 3 цикла. Мы в дальнейшем будем пользоваться только командой `rcall`.

Отличаются они от команд безусловного перехода тем, что здесь в момент перехода к процедуре контроллер автоматически сохраняет в стеке адрес текущей команды для того, чтобы потом знать, куда вернуться (потому длительность выполнения этих команд на такт больше, чем для просто перехода). А как МК «узнает», *когда* именно нужно возвращаться? Для этого каждая процедура-подпрограмма оформляется специальным образом: не отличаясь поначалу ничем от любого другого участка программного кода, обозначенного меткой, в месте возврата она должна содержать специальную команду — `ret` (от return — «возврат»). По этой команде МК извлекает из стека сохраненное содержимое счетчика команд и продолжает выполнение прерванной основной программы.

Заметим, что стек — одно из самых употребительных понятий в программировании. Наличие программного стека позволяет, например, организовать привычное для языков высокого уровня разделение переменных на локальные и глобальные. Во всех последующих программах в этой книге мы будем пользоваться только глобальными переменными, и при нехватке регистров общего назначения для них мы будем использовать ячейки SRAM. (Привычку употреблять преимущественно глобальные переменные автор даже перенес в свой стиль программирования на Delphi, как вы увидите из *главы 18*.) Однако в больших проектах локальные переменные зачастую бывает просто необходимы (например, когда одна и та же процедура вызывается в разных местах с исходными данными, хранящимися в различных регистрах). Механизм организации локальных переменных с помощью стека приведен в листинге 13.2 (он в точности такой же, как это происходит при компиляции в «настоящих» языках программирования).

Листинг 13.2

```
push var_1 ;переменная var_1 в программе помещается в стек
push var_2 ;переменная var_2 в программе помещается в стек
rcall procedure ;вызывается процедура
pop var_2 ;после нее результат извлекается из стека
pop var_1 ;второй результат извлекается из стека
. . . . .
procedure: ;в процедуре извлекается локальная
pop var_loc2 ;переменная var_loc2 со значением var_2
```

```
pop var_loc1 ;и переменная var_loc1 со значением var_1
. . . . . ;расчеты, расчеты...
push var_loc1 ;результат - в стек
push var_loc2 ;результат - в стек
ret ;возврат из процедуры
```

В качестве переменных `var_1` и `var_2` возможны любые регистры, при этом действия внутри процедуры всегда будут совершаться с заданными регистрами `var_loc`. Обратите внимание, что когда таких переменных несколько, важно соблюдать правильный порядок их помещения в стек и извлечения оттуда, согласно принципу «первым вошел — последним вышел» (в программах на языках высокого уровня за порядком переменных в стеке следят специальные форматы вызова функций типа `stdcall` и подобные).

Аналогично происходит обработка прерываний, только специальной команды, как вы знаете, там нет, вызов производится обычным переходом `rjmp` или `jmp`, но поскольку он осуществляется с определенного адреса (там, где стоит вектор прерывания), то контроллер делает то же самое: сохраняет в стеке адрес командного счетчика, на котором выполнение основной программы было грубо нарушено, начинает выполнять прерывание и ожидает команды возврата, здесь она записывается как `reti` (return interrupt).

Еще одна важнейшая группа команд ветвления программы — команды перехода по состоянию отдельного бита в указанном регистре (`sbrs`, `sbic` и т. п.). Они очень удобны для организации процедур, аналогичных оператору выбора CASE в языках высокого уровня, но, к сожалению, обладают непривычной логикой: «пропустить следующую команду, если условие выполняется» и для новичка могут показаться слишком заумными. В качестве примера приведу довольно сложную по логике работы, но характерную для микроконтроллерной техники процедуру, в которой задача формулируется так: при наступлении некоторого условия мигать попеременно зеленым и красным светодиодами (СД).

Предположим, что условие мигания задается состоянием бита 3 в некоем рабочем регистре, который назовем регистром флагов — `Flag` (не путать с «официальным» регистром флагов `SREG`, о котором далее). Если бит 3 регистра `Flag` равен единице (установлен), надо мигать, если нет (сброшен) — оба СД погашены.

Красный СД подсоединен к выходу порта B номер 5, а зеленый — к выходу порта C номер 7 (разумеется, это могут быть любые другие выводы других портов).

Текущее состояние СД задается битом 4 в том же регистре флагов `Flag`.

Алгоритм работы такой программы на языке Pascal (листинг 13.3) описывается типичным вложенным оператором выбора.

Листинг 13.3

```
case <бит 3 рег. Flag> of
  0: <погасить оба СД>
  1: case <бит 4 рег. Flag> of
     1: <устанавливаем Port C, вых. 7>; //горим зеленым <сбрасываем
Port B, вых. 5>; //гасим красный
       <сбрасываем бит 4 Flag>; {следующий раз горим красным}
     0: <устанавливаем Port B, вых. 5>; //горим красным
       <сбрасываем Port C, вых. 7>; //гасим зеленый
       <устанавливаем бит 4 Flag>; //следующий раз горим зеленым
     end;
end;
```

Разобравшийся в алгоритме читатель уже, несомненно, задает вопрос — а как обеспечить цикличность? Для этого подобный код включают в обработчик события по таймеру с секундным, например, интервалом. Причем, что характерно, и в микроконтроллере, и в операционной системе Windows это происходит абсолютно одинаково: инициализируется системный таймер (в МК для этого надо разрешить соответствующее прерывание), задается интервал его срабатывания (в Windows это одна команда, в МК их несколько больше) и — вперед! Но с таймерами мы будем разбираться далее по ходу дела, а пока посмотрим, как тот же алгоритм реализовывается в МК (листинг 13.4).

Листинг 13.4

```
      sbrs   Flag,3 ;если флаг 3 стоит, будем мигать
      rjmp   dark ;иначе будем гасить
      sbrs   Flag,4 ;если флаг 4 стоит, будем гореть зеленым
      rjmp   set4 ;иначе красным
      cbr    Flag,0b00010000 ;следующий раз горим красным
      cbi    PortB,5 ;гасим зеленый
      sbi    PortC,7 ;горим зеленым
      rjmp   continue ;все готово
set4:       ;если флаг 4 не стоит, будем гореть красным
      sbr    Flag,0b00010000 ;следующий раз горим зеленым
      cbi    PortC,7 ;гасим красный
      sbi    PortB,5 ;горим красным
      rjmp   continue ;все готово
```

```
dark:
      cbi    PortC,7 ;гасим оба
      cbi    PortB,5
continue:
. . . . .
```

С используемыми здесь командами установки и сброса отдельных бит (`sbi`, `sbr` и т. п.) мы плотнее познакомимся чуть позже, а сейчас задержимся на ключевой команде всего алгоритма — `sbrs`, что расшифровывается, как Skip if Bit in Register is Set («пропустить, если бит в регистре установлен»). Имеется в виду, что по состоянию бита (если он установлен в единицу) пропустить нужно следующую команду. В качестве последней обычно также выступает одна из команд ветвления, как здесь, но далеко не всегда. Удобно, например, организовывать выход из прерывания или подпрограммы по какому-то условию, если поставить следующей после `sbrs` команду `reti` или, соответственно, `ret` (примеры мы еще встретим).

Противоположная по логике процедура записывается, как sbrc (Skip if Bit in Register is Cleared — «пропустить, если бит в регистре очищен», т. е. равен нулю). Наконец, есть еще пара аналогичных команд — `sbis` и `sbic`, которые применяются, когда нужно отследить состояние бита в регистре ввода/вывода (I/O), а не в регистре общего назначения. Все эти команды мы будем активно применять в дальнейшем.

Наконец, в самой обширной группе команд ветвления имена начинаются с букв br (от слова branch — «ветка»). Это команды условного перехода, которые считаются одними из самых главных в любой системе программирования, т. к. позволяют организовывать циклы — базовое понятие программистских наук. По смыслу все эти команды сводятся к банальному `if ... then` («если … то»). Мы будем пользоваться лишь частью этих команд, потому что они во многом взаимозаменяемы, и здесь подробно разберем только одну пару команд. Смысл остальных вам будет понятен по ходу дела.

Это наиболее часто употребляемая пара `brne` (Branch if Not Equal, «перейти, если не равно») и `breq` (Branch if Equal, «перейти, если равно»). Уже из смысла этих команд понятно, как они пишутся: после команды следует метка, на которую нужно перейти. Вопрос только такой: откуда здесь берется собственно условие? Для этого все команды ветвления обязательно употребляют в паре с одной из команд, устанавливающих флаг нуля Z в регистре флагов `SREG`. Обычно (и это наиболее наглядно) для этой цели служит команда `cp` (от compare — «сравнить»), которая сравнивает регистры, или `cpi` («сравнить с непосредственным значением»). Например, вот так будет вы-

глядеть простейший цикл, в котором переменная temp последовательно принимает значения от 1 до 10:

```
   clr temp ;обнулить temp
back_loop:
   inc temp ;увеличиваем temp на 1
   <что-то делаем, необязательно с помощью temp>
   cpi temp,10
   brne back_loop
```

Обратите внимание: если надо, чтобы temp начинала с нулевого значения, то фрагмент «что-то делаем» следует вставить до команды inc, но тогда последним рабочим значением temp в цикле будет 9, а не 10, а после выхода из процедуры — все равно 10.

Другой нюанс заключается в том, что удобная команда сравнения с непосредственным числом cpi работает только для регистров с номерами от 16 до 31 (как и многие другие команды работы с непосредственными значениями, например, ldi). По этой причине рабочие переменные (temp, счетчики) всегда желательно выбирать из этой половины регистрового файла (в примерах в этой книге, как и в «аппнотах», кстати, обычно temp — это r16, хотя и не всегда). Положение осложняется тем, что регистры из старшей половины наиболее дефицитны: последние шесть из них объединены в пары для работы с памятью (см. далее) и некоторых других операций, r24 и r25 задействованы в команде adiw и т. п. Если переменных не хватает, то регистр из первой половины регистрового файла (допустим, это r15) в аналогичном цикле приходится использовать с парой команд:

```
ldi temp,10
cp r15,temp
```

Иногда проще построить декрементный цикл, в котором переменная уменьшается от заданного значения до нуля:

```
   ldi temp,10 ;загружаем 10 в temp
back_loop:
   dec temp ;уменьшаем temp на 1
   <что-то делаем с помощью temp>
   brne back_loop
```

Как видите, здесь вообще ничего сравнивать не требуется, потому что команда dec при достижении нуля сама установит флаг Z (то же относится к команде tst temp, которая эквивалентна команде cpi temp,0). И если даже выбрать регистр из первой половины, то лишняя команда понадобится не в каждом цикле, а только один раз для загрузки предварительного значения:

```
   ldi temp,10 ;загружаем 10 в temp
   mov r15,temp ;загружаем 10 в r15 и далее его используем
   ;в команде dec
```

Один важный нюанс в работе всех команд перехода заключается в том, что они могут занимать непредсказуемое количество циклов (1 или 2), в зависимости от того, выполняется условие или нет. В AVR реализован конвейер команд, который «тупо» полагает, что следующей будет выполняться команда сразу после команды перехода. Естественно, если ветвление необходимо (в большинстве случаев), конвейер останавливается на один лишний такт, в течение которого происходит выборка адреса перехода. Однако этот недостаток с лихвой компенсируется тем, что за счет конвейера почти все остальные команды выполняются за один такт — недостижимая мечта для многих других типов контроллеров (например, в популярном до сих пор семействе 8051, выпущенном еще в начале 80-х, команда выполняется как минимум за 12 тактов, хотя некоторые современные клоны этого семейства могут работать и быстрее).

Арифметика и логика в интерпретации AVR

Арифметические операции для AVR на первый взгляд могут показаться реализованными довольно странно для пользователя, привыкшего к бытовому представлению об арифметике, но на самом деле получается очень стройная система.

Не вызывают никаких возражений только очевидные операции: `add R1,R2` (сложить два регистра, записать результат в первый) и `sub R1,R2` (вычесть второй из первого, записать результат в первый). Но если вдуматься, то вопросов возникает множество: а что будет, если сумма превышает 255? Или разность меньше нуля? Куда девать остатки? Оказывается, все продумано: для учета переноса есть специальные команды `adc` и `sbc` соответственно. Корректная операция сложения двух 16-разрядных чисел будет занимать две команды:

```
add RL1,RL2
adc RH1,RH2
```

Здесь `RL1` и `RL2` содержат младшие (low) байты слагаемых, а `RH1` и `RH2` — старшие (high). Если при первой операции результат превысит 255, то перенос запишется в специальный флаг переноса (в регистре флагов `SREG` обозначается буквой C) и учтется при второй операции. Аналогично выглядит операция вычитания.

Постойте, но мы же вовсе не хотели складывать 16-разрядные числа! Мы хотели всего лишь сделать так, чтобы в результате сложения 8-разрядных чисел получился правильный результат, пусть он займет два байта. На что нам тогда старший байт второго слагаемого, если его вообще в природе не существует? Конечно, можно сделать его фиктивным, загрузив в некий регистр

нулевое значение, но это только кажется, что регистров у AVR много (аж 32 штуки), на самом деле они довольно быстро расходуются под переменные и разные другие надобности, и занимать целый регистр под фиктивную операцию, пусть даже на один раз, как-то некрасиво. Потому «экономная» операция сложения 8-разрядных чисел будет выглядеть таким образом:

```
add  RL1,R2
brcc  add_8
inc  RH1
add_8:
. . . . .
```

Исходные слагаемые находятся в `RL1` и `R2`, а результат будет в `RH1:RL1`. Отметим, что в старшем разряде (`RH1`) в результате может оказаться только либо 0, либо 1, т. к. сумма двух восьмиразрядных чисел не может превысить число 510 (255 + 255), именно потому флаг переноса C представляет собой один-единственный бит в регистре флагов. В этой процедуре команда `brcc` представляет собой операцию условного перехода на метку `add_8` по условию, что флаг переноса равен нулю (BRanch if Carry Cleared). Таким образом, если флаг не равен нулю, выполнится операция увеличения значения регистра `RH1` на единицу `inc RH1`, в противном случае она будет пропущена.

Внимательный читатель, несомненно, уже заметил ошибку в программе: а чему равно значение `RH1` до выполнения нашей процедуры? Вдруг оно совсем не ноль, и тогда нельзя говорить о корректном результате. Поэтому правильней было бы дополнить нашу процедуру еще одним оператором, который расположить раньше всех остальных: `clr RH1` (т. е. очистить `RH1`).

Заметим, что во всех случаях процедура разрастается во времени: было две команды, стало четыре, причем тут имеется еще и неявное замедление, поскольку все представленные арифметические команды выполняются за один такт, а команда ветвления `brcc` может занимать такт (если C = 1), а может и два (если C = 0). Итого мы выиграли один регистр, а потеряли два или три такта. И так всегда: есть процедуры, оптимизированные по времени, есть — по количеству команд, а могут быть и по количеству занимаемых регистров. Идеала, как и везде в жизни, тут не добиться, приходится идти на компромисс.

Если вы посмотрите таблицу команд в *Приложении 4*, то можете обратить внимание, что из восьмибитовых арифметических операций с константой доступно только вычитание (`SUBI, SUCI`), напрашивающейся операции сложения с константой нет. Это не упущение автора при формировании выборочной таблицы, а действительная особенность системы команд AVR. Это можно обойти при желании вычитанием отрицательного числа, но на практике не очень-то требуется, потому что разработчики семейства AVR решили

облегчить жизнь пользователям, добавив в перечень арифметических команд две очень удобные команды `adiw` и `sbiw`, которые, по сути, делают то же самое, что пары команд `add`/`adc` (`sub`/`sbc`), только за одну операцию, и притом с константой, а не с регистром. Единственный их недостаток — работают они только с определенными парами регистров: с четырьмя парами, начиная с `r25-r24`. Три старших пары (`r26-27`, `r28-29` и `r30-31`) носят еще название X, Y и Z, и мы их будем «проходить» далее в этой главе, они задействованы в операциях обмена данными с SRAM. Но, к счастью, точно так же работает и пара `r24-r25`, которая более нигде не употребляется в объединенном качестве, и это очень удобно. Независимо от используемой пары, старшим считается регистр с бóльшим номером, а операцию нужно проводить с младшим, при этом перенос учтется автоматически. Например, в результате выполнения последовательности команд

```
clr r25
ldi r24,100
adiw r24,200
```

в регистрах `r25:r24` окажется записано число 300 (в `r25` будет записано $01, что эквивалентно 256 для 16-разрядного числа, а в `r24` окажется $2C = 44). Аналогично работает и процедура вычитания константы `sbiw`.

С арифметикой многоразрядных чисел мы познакомимся в *главе 15*, там же попробуем освоить умножение и деление. Заметим, что все эти операции корректно работают с числами со знаком (см. *главу 7*), но вы удивитесь, узнав, насколько редко такая задача возникает на практике. А с дробными числами (с «плавающей запятой») AVR работать напрямую не «умеют», и тут приходится хитрить, о чем мы подробнее поговорим в *главе 15*.

Кроме собственно арифметических, к этой группе команд еще иногда относят некоторые логические операции, например логический сдвиг, хотя чаще их помещают в группу инструкций для операций с битами. Самая простая такая операция — сдвиг всех разрядов регистра влево (`lsl`) или вправо (`lsr`) на одну позицию. Сдвиги приходится применять довольно часто, потому что, как вы знаете из *главы 7*, такая операция равносильна умножению (соответственно, делению) на 2. Для того чтобы разряды не терялись при сдвиге, предусмотрены разновидности этих операций: `rol` и `ror`. Они учитывают все тот же флаг переноса C, и его можно перенести в другой регистр. Например, в результате выполнения последовательности команд

```
lsl R1
rol R2
```

регистр `R1` будет умножен на 2, а старший его разряд (неважно, ноль он или единица) окажется в младшем разряде `R2`. Причем обратите внимание, что `R2` при этом также умножается на 2, и, следовательно, его младший разряд без

учета переноса всегда окажется равным нулю (четные числа в двоичной системе оканчиваются на 0).

Здесь же имеет смысл рассмотреть инструкции установки отдельных битов. Команды, устанавливающие значения битов в регистрах общего назначения (`sbr` и `cbr`) иногда относят к группе арифметических операций, а команды, устанавливающие биты в регистрах ввода/вывода (`sbi` и `cbi`) — к группе битовых операций. Правда, в некоторых пособиях их относят к одной группе операций с битами, что, конечно, логичней. Но почему такой разнобой, если они, по сути, делают одно и то же?

Механизм работы этих команд существенно различается. Очевиднее всего устроены `sbi` и `cbi`: так, уже знакомая нам команда `sbi PortB,5` установит в единичное состояние разряд номер 5 порта B. Если этот разряд порта сконфигурирован на выход, то единица появится непосредственно на соответствующем выводе микросхемы (например, для микросхемы ATmega8 это вывод 19, для 8515 или 8535 любого семейства это вывод 6 и т. п.), если на вход, то эта операция управляет подключением «подтягивающего» резистора.

Гораздо сложнее устроены команды установки бит в регистрах общего назначения. Взгляните на фрагмент ранее приведенного кода, где вы встретите команду `sbr Flag,0b00010000`, устанавливающую бит номер 4 в единицу. Почему так сложно, да еще и в двоичной системе? Дело в том, что команды эти на самом деле не устанавливают никаких битов, а просто осуществляют некую логическую операцию между значением регистра и заданным байтом, который в этом случае называют *битовой маской*. Указанная команда расшифровывается как `Flag OR 0b00010000`. Если вы вернетесь к *главе 7*, то сообразите, что при такой операции четвертый бит (нумерация начинается с нуля) установится в единицу, какое бы значение он ни имел ранее, а все остальные останутся в старом состоянии. Понятно также, почему я предпочел использовать двоичное представление маски: так легче отсчитывать биты. Так, разумеется, можно за один раз установить хоть все биты в регистре — в этом отличие команды `sbr` от `sbi`, которая устанавливает только по одному биту.

Аналогично работает команда сброса бита `cbr Flag,0b00010000`. Только для того, чтобы ее можно было записывать в точности в том же виде, что и `sbr`, логическая операция, которую она осуществляет, сложнее: `Flag AND (NOT 0b00010000)`. Здесь сначала в маске инвертируются все биты (реально это производится вычитанием ее из числа $FF, т. е. нахождением дополнения до 1), а затем полученное число и значение регистра комбинируются операцией логического умножения (а не сложения, как в предыдущем случае). В результате четвертый бит обнулится обязательно, а все остальные останутся

в неприкосновенности. Кстати, обнулять все биты некоего регистра R удобнее и нагляднее не командой `cbr R,$FF`, а с помощью инструкций `clr R` или `ldi R,0`.

Призываю вас об этой разнице между регистрами общего назначения и регистрами ввода/вывода не забывать. Я и сам до сих пор «попадаюсь» на том, что в случае необходимости обнуления бита 1 в рабочем регистре `temp` записываю `cbr temp,1` (аналогично верной команде `cbi PortB,1`), хотя такая операция обнулит не первый, а нулевой бит в `temp`. А операция `cbr R,0` (как и операция `sbr R,0`) вообще ничего не делает, и такая запись бессмысленна.

Команды переноса данных

Последнее, что мы рассмотрим в этом разделе — команды, которые употребляются для переноса данных из одной области памяти в другую (здесь память рассматривается в широком смысле этого слова, и в нее включаются также и регистры). Некоторые команды из этой группы нам уже знакомы — это `ldi` и `mov`. Первая загружает в регистр непосредственное число (но действует только для регистров, начиная с `r16`), а вторая — значение другого регистра. Отметим еще, что для `ldi` очень часто применяется форма записи, которую можно пояснить следующим практическим примером:

```
ldi temp,(1<<RXEN|1<<TXEN|1<<RXB8|1<<TXB8)
```

Смысл этого выражения в том, что мы устанавливаем в единицы для регистра `temp` только биты с указанными именами (естественно, последние должны быть где-то определены, в данном случае это сделано в inc-файлах). Обратите внимание, что в отличие от команды `sbi`, остальные биты будут одновременно обнулены, а не останутся при своих значениях. Такая форма очень удобна для задания поименованных битов в процессе инициализации служебных регистров (при использовании «законной» команды `sbi` нужные биты пришлось бы указывать в числах, и предварительно обнулять регистр). В данном случае мы хотим инициализировать последовательный порт UART, и приведенная форма записи означает, что устанавливается разрешение приема (`RXEN`) и передачи (`TXEN`), причем и для того и для другого устанавливается 8-битный режим (`RXB8` и `TXB8`). Подробнее мы в этом разберемся в *главе 16*, когда будем «проходить» программирование UART, а пока заметим, что еще не все доделали: установили биты в регистре `temp`, но он-то какое отношение имеет к последовательному порту?

Потому следующей по тексту программы командой мы обязаны перенести установленное значение в соответствующий регистр ввода/вывода, для которого, собственно, поименованные биты и имеют значение, называющийся `UCR`. Это делается традиционной для всех ассемблеров командой `out`:

```
out  UCR,temp
```

Замечу, что указанные команды по отношению к UART справедливы для семейства Classic, для Mega это делается несколько сложнее (а Tuny, кроме модели 2313, вообще UART не имеют), но сейчас это неважно. В паре к команде `out` идет симметричная команда `in` (чтения данных из I/O-регистра).

> ***Заметки на полях***
>
> Здесь уместно обратить внимание читателей на то, что в момент подобного рода установок надо тщательно следить за тем, чтобы значение рабочего регистра `temp` не могло измениться между командами его установки и переносом значения в I/O-регистр. А как это может случиться, спросите вы? Да запросто, — если разрешены прерывания, в обработчиках которых наверняка также будет присутствовать `temp`, то они могут «вклиниться» между командами. Пусть вероятность такого события крайне мала, и оно может не происходить годами, но программист обязан об этом помнить, и грамотно составленная программа во время таких установок прерывания запрещает. В секции `RESET` это обеспечивается тем, что по умолчанию прерывания запрещены, а команда на их общее разрешение (`sei`) ставится после всех установок. Куда меньше проблем вызывают сами обработчики, т. к. по умолчанию во время обработки прерывания другое, произошедшее позднее, ожидает своей очереди (автоматически прерывания разрешатся только на выходе из текущего обработчика по команде `reti`). Но если вам зачем-то нужно разрешить критичным прерываниям «вклиниваться» во время обработки некритичных (для этого следует явно разрешить прерывания внутри обработчика данного прерывания командой `sei`), то такая проблема снова возникает. Именно поэтому, в частности, не рекомендуется употреблять синонимы для имен рабочих регистров: слишком легко забыть, что `temp` и `counter`, к примеру, указывают на один регистр, и в разных процедурах, вложенных одна в другую, они будут произвольно менять свое значение. Если же все-таки запретить прерывания нельзя или не хочется (например, во время выполнения длинного цикла), то уберечься от ошибок можно, если в начале обработчика прерывания сохранять критичные регистры в стеке командой `push`, а в конце — извлекать их командой `pop`. Но тут также возникает немало возможностей наделать трудновылавливаемые ошибки в программе, поскольку отследить содержимое стека тоже не всегда просто.

Следующими по важности командами переноса данных будут команды загрузки и чтения SRAM — `ld` и `st`. С этими командами связаны такие «жуткие» понятия, как «прямая» и «косвенная» адресации, а также «относительная косвенная адресация» и прочая подобная «лабуда» из арсенала разработчиков микросхем и теоретиков программирования. Эти понятия на практике абсолютно не требуются, и только затуманивают мозги, потому что зачастую относятся к совершенно разным командам и узлам кристалла (это единственный раздел в описаниях МК, который спокойно можно пропускать при чтении, все остальные изучать очень полезно). Мы постараемся от термина «адресация» вообще отказаться и разберем здесь три основных режима чтения/записи SRAM: простой, а также с преддекрементом и постинкрементом. Все три употребляются очень часто, хотя два последних режима работают не во всех типах AVR.

Во всех случаях в чтении и записи SRAM используются регистры X, Y и Z — т. е. пары r27:r26, r29:r28 и r31:r30 соответственно, которые по отдельности еще именуют XH, XL, YH, YL, ZH, ZL в том же порядке (т. е. старшим в каждой паре служит регистр с бóльшим номером). Если обмен данными производится между памятью и другим регистром общего назначения, то достаточно только одной из этих пар (любой), если же между областями памяти — целесообразно задействовать две. Независимо от того, какую из пар мы используем, чтение и запись происходят по идентичным схемам, меняются только имена регистров.

Покажем основной порядок действий при чтении из памяти для регистра Z (r31:r30). Чтение одной ячейки с заданным адресом Address, коррекция ее значения и последующая запись выполняются так:

```
ldi ZH,High(Address) ;старший байт адреса RAM
ldi ZL,Low(Address) ;младший байт адреса RAM
ld temp,Z ;читаем ячейку в temp
inc temp ;например, увеличиваем значение на 1
st Z,temp ;и снова записываем
```

Заметки на полях

При всех подобных манипуляциях нужно внимательно следить за двумя вещами: во-первых, за тем, чтобы не залезть в несуществующую область памяти (если объем SRAM составляет 512 байт, как в большинстве моделей, которые мы будем использовать, то ZH в данном примере может иметь значения только 0 или 1). Во-вторых, не забыть, что младшие адреса SRAM заняты регистрами (в большинстве моделей под это зарезервированы первые 96 адресов от $00 до $5F). И запись, например, по адресу $0000 (ZH=0, ZL=0) равносильна записи в регистр r0. Во избежание коллизий я по возможности поступаю следующим образом: резервирую пару ZH, ZL только под запись/чтение в память, и устанавливаю с самого начала регистр ZH в единицу. Тогда при любом значении ZL мы будем «шарить» только в старших 256 байтах из 512, чего для любых практических нужд обычно достаточно (а если недостаточно, то, скорее всего, все равно придется задействовать внешнюю память), а случайно пересечься с регистрами нет никакой возможности. Обязательно нужно также помнить, что и последние адреса памяти также нельзя занимать: мы сами дали в начале программы команду задействовать их под стек.

Режимы с преддекрементом и постинкрементом удобны, когда нужно прочесть или записать в память целый фрагмент (эти команды недействительны для большинства МК семейства Tuny). Схема действий аналогичная, только при этом команды выглядят так:

```
st -Z,temp ;с преддекрементом, запись в ячейку с адресом Z-1, после
;выполнения команды Z = Z-1
st Z+,temp ;с постинкрементом, запись в ячейку с адресом Z, после
;выполнения команды Z = Z+1
```

Аналогично выглядят команды чтения:

```
ld temp,-Z ;с преддекрементом, чтение из ячейки с адресом Z-1, после
;выполнения команды Z = Z-1
ld temp,Z+ ;с постинкрементом, чтение из ячейки с адресом Z, после
;выполнения команды Z = Z+1
```

Вот как можно в цикле записать 16 ячеек памяти подряд одним и тем же значением из temp, начиная с нулевого адреса старших 256 байтов памяти:

```
       ldi ZH,1
       clr ZL
LoopW:
       st  Z+,temp ;сложили в память
       cpi ZL,16 ;счетчик до 16
       brne LoopW
```

Еще одна важная, хотя и нечасто употребляемая команда переноса данных — инструкция lpm, которая позволяет прочесть произвольный байт из памяти программ. Напомню, что архитектура у большинства разновидностей МК, в том числе и AVR, гарвардская, когда память программ отделена от памяти данных, и в память программ контроллер самостоятельно ничего писать не может (кроме случая самопрограммирования, но это экзотика). Потому хранение в памяти программ тех констант, которые никогда не будут изменяться, прямо рекомендуется разработчиками AVR, за много лет так и не сумевшими окончательно решить проблему безопасного хранения данных в EEPROM (о чем мы еще будем долго и много говорить).

Вот типичная задача такого рода: пусть контроллер осуществляет управление семисегментным индикатором в динамическом режиме, когда в каждом такте приходится выводить разные цифры. Выстраивать рисунки (битовые маски) этих цифр каждый раз замучаешься, и программа получится очень громоздкая, к тому же совершенно нечитаемая. Проще их «нарисовать» один раз и расположить по порядку (от 0 до 9) прямо в любом месте программы (удобно сразу после векторов прерываний). Дать понять компилятору, что это особая область памяти, которая его не касается, и должна быть перенесена без изменений, можно с помощью директивы db, а чтобы потом можно было найти эту область, ее следует пометить обычной меткой:

```
N_mask: ;маски цифр на семисегментном индикаторе
.db 0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,
0b01101101,0b01111101,0b00000111,0b01111111,0b01101111
```

Заметим, что таким образом можно формировать hex-файлы для предварительной записи констант в EEPROM, хотя мы будем пользоваться иным, более корректным методом (см. *главу 15*). Теперь можно в нужном месте использовать команду lpm следующим хитрым образом:

```
ldi ZH,High(N_mask*2) ;загружаем адрес начала маски
ldi ZL,Low(N_mask*2)
```

```
add ZL,5 ;адрес маски цифры «5»
lpm ;маска окажется в регистре r0
```

Надо сказать, что современные МК семейства Mega поддерживают и более простой формат команды `lpm`, аналогичный обычной `ld`, но универсальности ради я привык к традиционному формату, который поддерживают все МК AVR. Здесь в регистр Z (и только Z!) заносится адрес начала массива констант, причем учитывается, что память программ имеет двухбайтную организацию, а в Z надо заносить побайтные адреса, отчего появляется множитель 2. По этому адресу, как мы договорились, располагается маска цифры «0». Для загрузки цифры «5» прибавляем к адресу это значение и вызываем команду `lpm`. Полученное значение маски окажется в самом первом регистре общего назначения `r0`, так что при таких действиях его не следует занимать под какие-то переменные.

О Fuse-битах

Это «несчастье» свалилось на нашу голову с появлением семейств Tuny и Mega и привело к многочисленным проклятиям на голову фирмы Atmel со стороны армии любителей, которые стали один за другим «запарывать» кристаллы при программировании. Теперь уже все привыкли и обзавелись соответствующим софтом, а сначала было довольно трудно. Положение усугублялось тем, что в описании этих сущностей действовала извращенная логика: как мы знаем, любая чистая EEPROM (по принципу ее устройства) содержит единицы, и слово «запрограммированный» по отношению к такой ячейке означает, что в нее записали нули. Поэтому разработчики программатора AS-2 даже специально написали в окне программирования конфигурационных ячеек (такое название более правильное, чем fuse-бит, буквально означающее «предохранительный бит») памятку на этот счет (рис. 13.7).

На рис. 13.7 приведено безопасное рабочее состояние конфигурационных ячеек для ATmega8535, причем выпуклая кнопка означает единичное состояние ячейки, а нажатая — нулевое (и не путайтесь с этим самым «запрограммированным» состоянием!). Для разных моделей набор fuse-битов различный, но означают они одно и то же, потому мы разберем типовое их состояние на этом примере. Перед первым программированием нового кристалла просто один раз установите эти ячейки в нужное состояние.

Переписывать фирменные руководства я не буду, остановлюсь только на самых необходимых моментах. По умолчанию любая микросхема семейств Mega или Tuny запрограммирована на работу от внутренней RC-цепочки, за что разработчикам большое спасибо, иначе было бы невозможным первичное программирование по SPI, а только через параллельный программатор.

Для работы с обычным кварцем, присоединенным по типовой схеме, требуется установить все ячейки CKSEL0—3 в единицы, что согласно логике контроллера означает *незапрограммированное* их состояние. Это и ведет к критической ошибке. Решив при поверхностном чтении очень невнятно написанного, и к тому же по-английски, руководства, что установка всех единиц означает запрограммировать все ячейки, пользователь смело устанавливает их на самом деле в нули, отчего микросхема переходит в состояние работы от внешнего генератора и разбудить ее через SPI-интерфейс уже невозможно. Легче всего в этом случае переустановить fuse-биты с помощью параллельного программатора, либо за неимением такового, попробовать-таки подключить внешний генератор (его можно собрать по одной из схем из *главы 9*), как описано в руководстве.

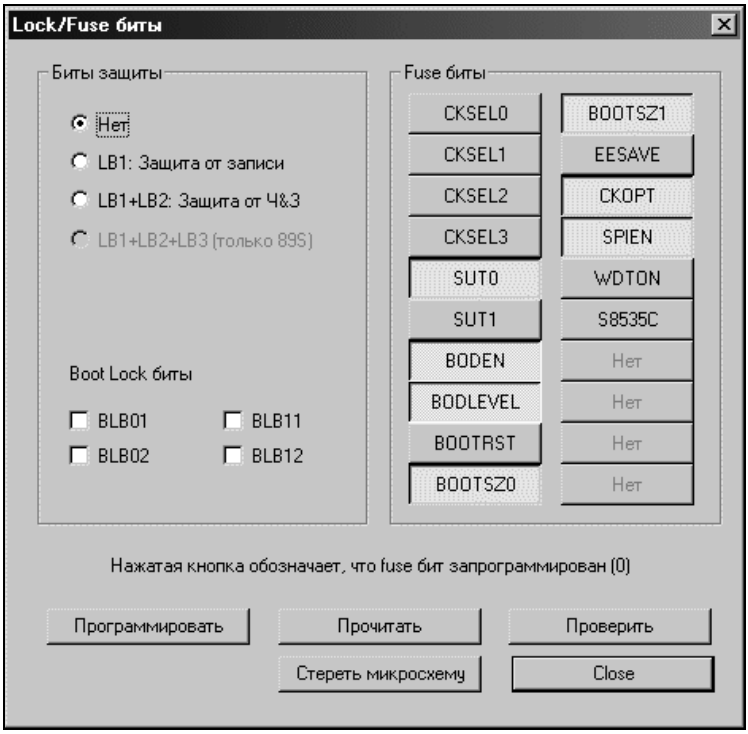


Рис. 13.7. Окно типового состояния конфигурационных ячеек в нормальном режиме работы ATmega8535

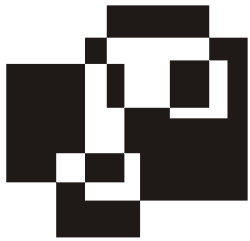
Это самая крупная ошибка, которую можно допустить, но есть и помельче, правда, легче исправляемые. Ячейка SPIEN разрешает/запрещает последовательное программирование по SPI и должна оставаться в нулевом состоянии.

Ячейка S8535C (в других моделях она, соответственно, будет иметь другое название, или вовсе отсутствовать) очень важна, она определяет режим совместимости с семейством Classic (в данном случае с AT90S8535). Если ее установить в нулевое состояние, то МК семейства Mega (а также и единственный представитель Tuny — модель 2313) перейдет в режим совместимости, про все «навороты» можно забыть (кроме, конечно, самих конфигурационных ячеек), и без изменений использовать наработанные старые программы. В режиме совместимости следует учесть, что состояния МК нельзя перемешивать: если fuse-бит совместимости запрограммирован, то программа компилируется полностью, как для семейства Classic (в том числе с помощью соответствующего inc-файла), иначе она может не заработать. Например, AT90S8535 имеет 17 прерываний, а ATmega8535 — 21, и те же самые прерывания могут оказаться на других местах.

Еще одна важная ячейка — EESAVE, которая на рис. 13.7 установлена в единицу (режим по умолчанию), но ее целесообразно перевести в нулевое состояние, тогда при программировании памяти программ не будет стираться содержимое EEPROM. Ячейки SUT определяют длительность задержки сброса, и в большинстве случаев принципиального значения их состояние не имеет.

Наконец, для нас будет иметь значение состояние ячеек BODEN и BODLEVEL. Первая, будучи установлена в ноль, разрешает работу т. н. схемы BOD (Brown-out Detection), которая сбрасывает контроллер при снижении питания ниже допустимого порога. Ячейка BODLEVEL и определяет этот самый порог: при установленной в ноль ячейке он равен 4 В, при единице — 2,7 В. При питании 5 В надо выбирать первое значение, при 3,3 В — второе. Это предохраняет контроллер от «недопустимых операций» при выключении питания, но для обеспечения полной сохранности содержимого EEPROM таких мер оказывается недостаточно и приходится принимать дополнительные.

Ячейки, название которых начинается с BOOT, определяют режим начальной загрузки. Как я уже упоминал, в современных AVR можно изменять начальный адрес программы и расположение векторов прерываний. Эти ячейки, как и все остальные, следует оставить в исходном состоянии. В том числе это касается и битов защиты программы, которые на практике никакой защиты не дают, т. к. при необходимости легко обходятся. Зато неприятностей могут доставить массу, поскольку раз запрограммировав их, исправить что-то уже будет очень трудно, а для любителя почти невозможно.



Глава 14

Проба пера: настольные часы

> Вы никогда не задумывались, почему на обычных часах стрелки идут слева направо? Вот если бы солнечные, а затем и механические часы были изобретены в Южном полушарии, все было бы наоборот.
>
> *Сетевой Журнал Русского Ословодства*

Сконструировать свои первые настольные часы «для дома, для семьи» меня заставила судьба. ЖК-индикаторы в работающих от сети стационарных конструкциях я полагаю неуместными — они «слепые» даже днем, а надо, чтобы часы было видно и ночью. Между тем, пытаясь в конце 1990-х приобрести настольные часы в связи с переездом на новую квартиру, я попал в какой-то неудачный момент, когда во всей Москве не было часов со светящимися индикаторами: старинные советские изделия, в которых малюсенькие голубенькие циферки были еле видны за густой сеткой анода, уже исчезли из продажи, а импортные на светодиодах, как говорится, «не завезли». В результате пришлось делать самому такие, какие нравится.

И хорошо, что я их сделал, потому что то, что, наконец, появилось в продаже позднее, проигрывает моей самодельной конструкции, по крайней мере, по двум статьям: по точности хода и цвету свечения индикаторов (ядовито-красный без какой-либо возможности выбора). Правда, фирменные конструкции несколько меньше размерами и обладают функциями будильника, но последние мне без надобности (никогда не пользовался будильниками), а реализовать их при необходимости несложно. Так что, как видите, часы конструировать самостоятельно стоит.

Отдельный вопрос — на какой же именно элементной базе все это делать? Во-первых, существуют, естественно, специальные микросхемы для часов.

Начал я именно с экспериментов с такими схемами (отечественного выпуска), и даже получил вполне работоспособную конструкцию. Но мне очень не понравилась их негибкость и приспособленность под определенную разновидность индикаторов, отчего пришлось «упражняться» со схемами усиления и преобразования уровней. Конструкция получилась довольно громоздкой — зачем тогда вообще специальная микросхема?

Потому первые свои законченные часы я сконструировал на универсальном микроконтроллере «в лоб», заставив МК посекундно «тикать» таймером и управлять индикаторами. Это была отличная практика написания программ для МК, и, как сейчас вижу, никаких принципиальных ошибок я не наделал. Хотя я потом создал еще пару конструкций, но и эти, самые первые часы безотказно работают уже вот без малого девятый год. Именно такую, не очень сложную для понимания, конструкцию мы и разберем в этой главе.

Заметки на полях

Предваряя темы в следующих главах, стоит отметить, что на самом деле — для массового производства — так часы, конечно, не делают. В данной конструкции, например, затруднительно воспроизвести календарь — и потому, что его алгоритм весьма громоздок (и в нем легко наделать скрытых ошибок), и потому, что он требует значительно больше индикаторов. Притом цифровыми семисегментными тут, вообще говоря, не обойдешься (день недели не удастся показать). Функции будильника также заметно усложнят схему. Наконец, — и это самое важное, — такие часы сами по себе будут работать неплохо, но если заставить микроконтроллер делать что-то еще полезное, он может попросту не справиться: начать сбоить в отсчете времени, что для часов недопустимо. К тому же такие часы практически невозможно использовать с батарейным питанием, в резервном режиме они могут идти в лучшем случае пару недель.

По этим причинам в более серьезных устройствах (например, когда в дальнейшем мы попробуем объединить часы с различными датчиками) «велосипедов» лучше не изобретать, а выбрать одну из широко распространенных неспециализированных микросхем часов, называемых еще RTC (Real Time Clock), которые включают календарь и функции будильника (а иногда и нескольких), таймера, могут выдавать во внешний мир определенную частоту, потребляют очень мало (типичная величина — 0,8 мкА), иногда обладают встроенным прямо в чип часовым кварцем (и даже с возможностью подстройки). Еще один плюс такой конструкции — часовые кварцы 32 768 Гц, как правило, точнее обычных, тех, что служат для тактирования МК. Выпускаются RTC с самыми разнообразными интерфейсами: от параллельного до I^2C. Именно такие микросхемы применяются, например, в компьютерах. Особенно преуспели в этом деле две фирмы: Dallas (он же MAXIM) и бывшая Seiko (ныне Epson). Далее мы разберем конструкции на основе таких микросхем.

Ну а теперь перейдем непосредственно к конструкции простейших часов, и начнем с выбора подходящего для этой цели МК.

Выбор микроконтроллера и общее построение схемы

Для выбора МК из предлагаемых фирмой Atmel просто подсчитаем, сколько нам требуется выводов. Во-первых, надо управлять четырьмя разрядами индикации (ЧЧ:ММ). Это мы будем делать в режиме *динамической индикации*, когда в каждый отдельный момент времени напряжение питания подается только на один разряд индикаторов. В это же время на сегменты, которые все соединены между собой параллельно, подается код, соответствующий именно этому разряду. В следующем такте код меняется, а напряжение подается на следующий разряд, и так далее. При четырех разрядах непосредственное управление предполагает $7 \times 4 = 28$ задействованных выводов, а динамическое — всего $7 + 4 = 11$. Чтобы мигание было незаметно для глаза, полный цикл смены разрядов должен повторяться с частотой не менее 70—100 Гц.

Затем нам надо засвечивать разделительный символ — в часах это традиционно двоеточие. Его, конечно, можно засветить постоянно, но лучше, когда оно мигает с не слишком высокой частотой (иногда можно увидеть конструкции, где разделительное двоеточие мигает быстро-быстро — это, конечно, недоработка, оно должно показывать недостающие на дисплее секунды). Наконец, нам надо часы устанавливать. Для этого минимально необходимо две кнопки (включение режима установки и собственно установка). Итого получилось по минимуму 14 выводов.

По тем временам я остановился на МК AT90S2313 — он выпускается в 20-выводном корпусе (см. рис. 12.1 вверху), в котором минимум 5 выводов должно быть занято под системные нужды (два питания, Reset[1] и два вывода для подключения кварца). Итого нам остается на все про все 15 выводов, что нас устраивает. Мы даже вроде бы получаем один резервный вывод, но далее увидим, что на самом деле под все желательные дополнительные функции выводов нам будет не хватать и придется изворачиваться (сейчас я бы, скорее всего, остановился на ATmega8, у которого 28 выводов корпуса, чтобы не экономить, но для изучения особенностей AVR дефицит даже полезнее). Естественно, если вы захотите повторить схему, и не достанете 2313 Classic, то придется заменить его на ATtuny2313, соответствующим образом установив fuse-бит совместимости (см. *главу 13*). Так как корпус у него тот же самый, то конструкция ничем отличаться не будет.

Теперь общая схема. Выбираем индикаторы большого размера (высота цифр — 1" или 25,4 мм), с общим анодом, т. е. типа SA10, если брать продукцию

[1] В МК семейства Mega и Tuny вывод Reset можно использовать и под другие нужды, но в «классике» это было еще не так, да и неудобно это.

Kingbright. Лично я предпочитаю желтого свечения (например, SC10-21Y), но это не имеет значения. Так как падение напряжения у них может достигать 4 В, то от того же источника, что требует МК (5 В), питать их нельзя.

Следовательно нам потребуется два напряжения питания: стабилизированное +5 В и нестабилизированное (пусть будет +12 В). Управлять разрядами индикаторов мы будем от транзисторных ключей с преобразованием уровня (когда на выходе МК уровень +5 В, ключ подает +12 В на анод индикатора), а сегменты от простых транзисторных ключей — при уровне +5 В вывод сегмента коммутируется на «землю» (так как питание индикаторов повышенное, то, к сожалению, управлять прямо от выводов процессора не получится). В обоих случаях управление получается в положительной логике: включенному индикатору и сегменту соответствует логическая единица (что совершенно не принципиально, но удобно для простоты понимания работы схемы). Резисторы в управлении сегментами примем равными 470 Ом, тогда пиковый ток через сегмент составит примерно 20 мА, а средний — 5 мА (при динамическом управлении 4-мя разрядами). Всех «восьмерок» у нас быть не может, максимальное число одновременно горящих разрядов равно 24 («20:08»), потому общее максимальное потребление схемы составит $24 \times 5 = 120$ мА, плюс ~10 мА схема управления, итого 130 мА.

Теперь обязательно подумаем о том, чтобы часы продолжали идти при сбоях в электрической сети. Нет ничего ужасней бытового прибора, который не может сохранить установки даже при секундном пропадании напряжения питания, вероятно, вы не раз с такими мучались. Конструкторов, делающих музыкальные центры, магнитофоны, микроволновые печи и электроплиты, в которых часы при малейшем сбое в подаче электроэнергии приходится устанавливать заново, следует расстреливать без суда и следствия.

Режим энергосбережения с глубоким «засыпанием» МК не подходит, поскольку тогда все «замирает» и его применение обессмысливается, ведь нам нужно, чтобы часы не просто сохраняли значение времени, а продолжали идти и при отключении от сети. При питании в пределах 4—5 В МК типа 2313 потребляет около 5 мА, так что можно рассчитывать на непрерывную работу от щелочной («алкалайновой») батарейки типа AA с емкостью порядка 2 А·ч в течение не менее 2—3 недель. Для обеспечения работы понадобятся три таких элемента, соединенных последовательно, тогда их общее напряжение составит 4,5 В.

Заметки на полях

В устройствах на специализированных микросхемах RTC можно использовать режимы энергосбережения МК, и дело обстоит значительно лучше: часы идут отдельно до тех пор, пока есть хоть какое-то питание (типичное минимально

допустимое значение для RTC — 2 В). В результате при грамотном проектировании можно обеспечить время работы от батарейки в сотни раз большее, чем у нас. Но мы все же пока ограничимся простейшим вариантом — настольные часы и не предназначены для работы в автономном режиме, а для того, чтобы перенести их из комнаты в комнату или «пережить» отключение электричества на пару часов, возможностей нашей системы вполне хватит.

Для обеспечения такого режима нам понадобится *монитор питания* — схема, которая отслеживает наличие входного напряжения, и переключатель с сетевого питания на батарейки. Чтобы сделать схему совсем «юзабельной», добавим также небольшой узел для сигнализации о необходимости замены резервной батарейки — пусть это будет наше ноу-хау, т. к. в подобных сетевых приборах такого почти ни у кого нет. Хотя есть специальные микросхемы, которые «мониторят» питание, и мы будем их в дальнейшем использовать, здесь в целях максимального упрощения схемы мы без них обойдемся. Схему такого узла удобно реализовать, «не отходя от процессора», на встроенном компараторе. Но тогда нужно задействовать аж 18 выводов (12 под индикацию, 2 кнопки, 2 входа компаратора, 1 для его выхода и еще 1 для монитора питания), а ставить процессор большего размера только для этой цели не хочется. И еще больше не хочется добавлять какие-то внешние схемы — все только потому, что мы захотели контролировать батарейку, которая, может быть, сядет этак лет через пять?

Поэтому мы поступим так: задействуем один из входов компаратора также и под вторую кнопку, как обычный вывод порта. А на второй вход компаратора «повесим» дополнительно функцию монитора — сигнализировать о пропадании внешнего питания. Остается придумать, как обеспечить сигнализацию разряда батареи — тут мы сделаем просто: пусть разделительный символ (двоеточие) мигает, когда все нормально, а когда батарея разряжена — горит все время. Таким образом мы получим наиболее экономичную схему с минимумом внешних элементов.

Теперь поглядим на схему разводки выводов AT90S2313 (рис. 14.1) и выберем, что и к чему мы будем коммутировать. Ко входу внешнего прерывания INT1 (7) удобно подключить кнопку, которая будет вводить часы в режим установки. От порта D (портов A и C в этом микроконтроллере нет) осталось шесть разрядов, четыре из которых мы задействуем под управление разрядами индикаторов: PD0 (2), PD1 (3), PD2 (6) и PD4 (8). Из восьми выводов порта B два заняты под входы компаратора AIN+ (выв. 12 — к нему мы подсоединим опорный источник для контроля батареи и также с него будем снимать информацию о состоянии питающего напряжения и второй кнопки) и AIN- (выв. 13 — к нему подключим батарейку). Для управления миганием разделительного двоеточия удобно использовать вывод OC1 (15), который управляется автоматически от таймера (см. *главу 12*). Под управление сегментами

мы задействуем оставшиеся выводы: PD5 (9), PD6 (11), PB2 (14) и PB4—PB7 (16—19). То, что выводы для управления индикаторами расположены не по порядку — это, конечно, не здорово, нам фактически придется управлять каждым разрядом по отдельности, но обойдемся.

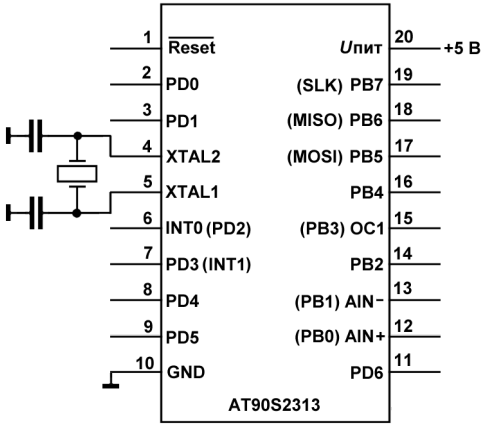


Рис. 14.1. Разводка выводов МК AT90S2313 (функции показаны применительно к нашей задаче)

Схема

Вот, собственно, и все предварительные наметки, можно рисовать схему платы управления (рис. 14.2). Схема проста, правда, некоторую громоздкость ей придают ключи управления индикаторами, однако все равно ее можно без труда уместить на плату примерно 70×100 мм, а при некоторых усилиях — и на меньшую.

Игольчатый разъем X1 типа IDC с 10 контактами — программирующий (в *главе 13* мы договаривались, что я буду разводить его в соответствии с 10-контактным ISP-программатором, там же подобный разъем описан подробнее, см. рис. 13.4). Все остальные внешние соединения, кроме питания, — через такой же разъем, но с 16 контактами, два из которых — «земля» и питание.

Подробности

Так как игольчатые разъемы типа IDC с шагом 2,54 мм встречаются в практике изготовления микроэлектронных устройств довольно часто, то стоит разобраться в их маркировке. Начнем с того, что наименование IDC в случае штыревых разъемов для установки на плату относится только к разъемам в кожухе с ключом (именно такие используются для подсоединения жесткого диска

в ПК). Бескорпусные подобные разъемы носят название PLD для двухрядных (или PLS для однорядных) типов и более удобны в радиолюбительской практике, т. к. длинные разъемы легко «ломаются» в нужном месте, обеспечивая необходимое число выводов (правда, при этом приходится как-то обозначать на плате первый вывод, чтобы не перепутать ориентацию при включении, см. рис. 13.4). Разметка на плате для обоих типов разъемов (с кожухом и без) одинакова, т. к. все равно приходится учитывать место, которое займет кабельная розетка при ее подсоединении, и мы в этой книге для определенности остановимся на IDC-типе. Разумеется, розетка для установки на плоский кабель (с использованием соответствующего инструмента), может иметь только фиксированное число контактов (из ряда 6, 10, 14, 16, 20, 22, 24, 26, 30, 34, 36, 40, 44, 50, 60…), что нужно учитывать при проектировании.

Цифра после обозначения разъема (IDC-10 или PLD-10), естественно, обозначает число контактов разъема, а следующая буква символизирует его конфигурацию: M (male, «папа») для штыревой части, и F (female, «мама») — для гнездовой. Далее может следовать еще одна буква, которая обозначает ориентацию: S для прямых выводов (разъем перпендикулярен плате), R для повернутых под углом 90° (разъем параллелен плате). Таким образом, приведенное на схеме рис. 14.2 обозначение IDC-10MS означает штыревой («папа») разъем в кожухе с ключом, с 10 прямыми выводами. Соответствующая этому разъему кабельная часть обозначится, как IDC-10F. Бескорпусные PLD-разъемы бывают, естественно, только штыревые, потому для них буквы M и F не указываются (а повернутые под углом 90° дополняются буквой R).

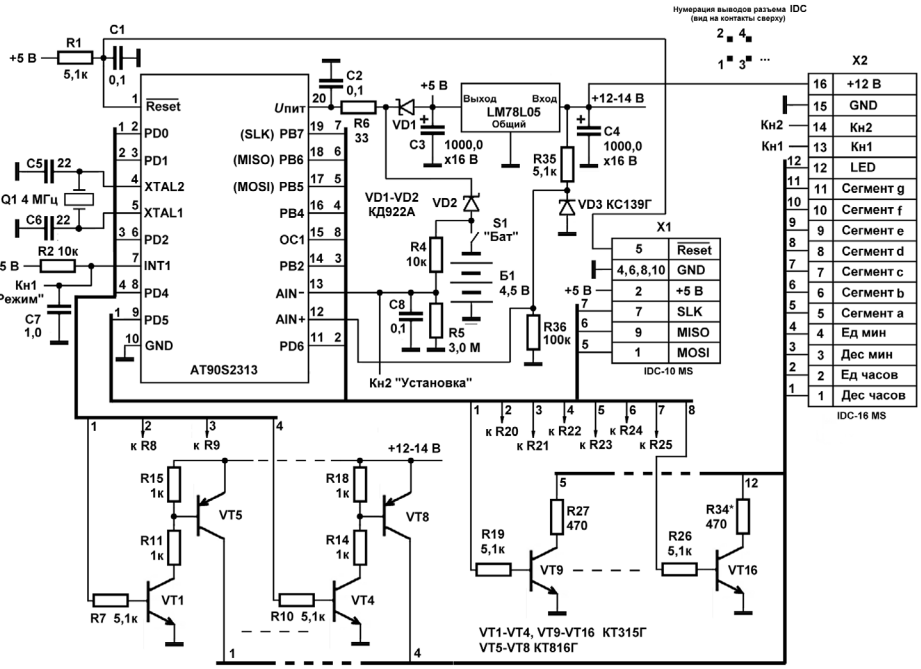


Рис. 14.2. Схема часов на МК AT90S2313 (плата управления)

Обратите внимание, что программирующие выводы (кроме Reset) здесь работают в двояком режиме. В нормальном режиме эти выводы работают как выходы на достаточно низкоомную (5,1 кОм) нагрузку. Не помешает ли это процессу программирования? Нет, не помешает — такая нагрузка для программатора вполне приемлема. Более того, «чистые» (нигде не задействованные) выводы программирования все равно следует нагружать «подтягивающими» резисторами, иначе не исключены сбои (об этом мы говорили в *главе 12*). Здесь же роль гасящей помехи нагрузки играют базовые резисторы ключей управления транзисторами, и дополнительные меры не требуются.

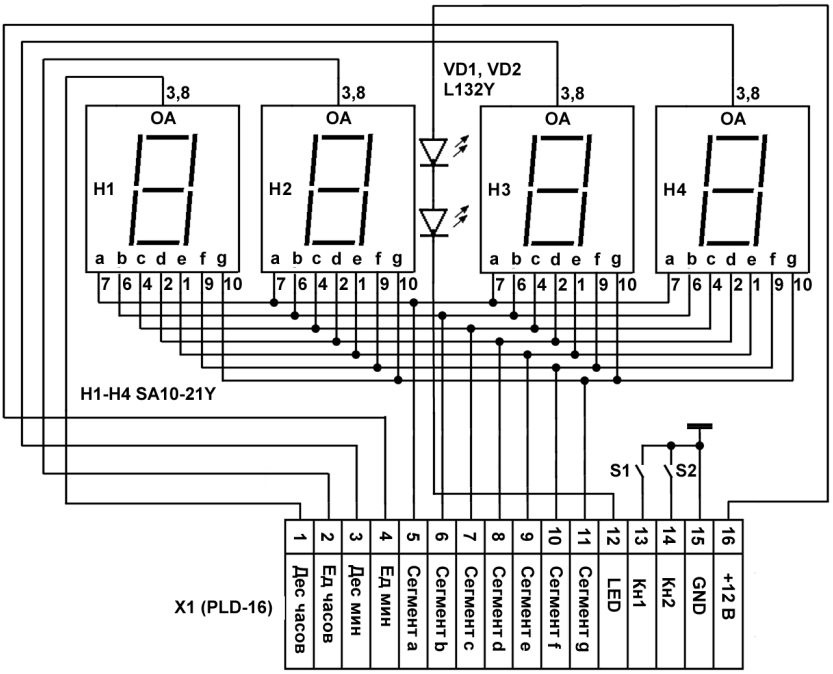


Рис. 14.3. Схема часов на МК AT90S2313 (плата индикации)

Плату индикации делаем отдельно (рис. 14.3). На ней мы располагаем четыре индикатора и две управляющих кнопки (о них далее), а также в точности такой же разъем IDC-16, как и на плате контроллера, причем он должен находиться на стороне платы, противоположной индикаторам. Разводка у него также должна быть идентичной. Эти разъемы мы соединим плоским кабелем. Изготовить такой плоский кабель с разъемами IDC-16F самостоятельно без специального инструмента практически невозможно, потому либо придется такой инструмент приобрести, либо попросить вам установить разъемы на

кабель в любой фирме, которая занимается сборкой и ремонтом компьютеров. Можно употребить и готовый кабель даже с бóльшим числом линий, если на плате установить разъемы PLD (т. е. при отсутствии кожуха). Это решение не очень красивое, т. к. при этом кабельная часть разъема будет выходить за пределы ответной на плате, и это нужно предусмотреть в разводке, иначе большой разъем может во что-нибудь упереться.

Разберем немного работу схемы. При включении питания цепочка R1C1 формирует надежный сигнал Reset. Напомню (см. *главу 12*), что ставить эту цепочку необязательно — производитель МК гарантирует нормальный Reset и без каких-либо внешних элементов, однако для лучшей защиты от помех это не повредит, ведь часы у нас должны работать по идее годами в круглосуточном режиме. После установления питания диод VD2 «запрет» батарею, которая имеет напряжение заведомо ниже, чем на выходе стабилизатора. Оба диода с переходом Шоттки, падение напряжения на них не превышает 0,2—0,4 В.

Теперь разберемся с нашими компараторными «примочками». В нормальном режиме кнопка Кн2 разомкнута и на работу схемы не влияет. Напряжение батареи фактически напрямую (делитель R4/R5 делит сигнал в отношении 300/301 и эта ошибка не имеет значения) попадает на инвертирующий вход компаратора. Это напряжение сравнивается с напряжением на стабилитроне VD3, равном примерно 3,9 В (стабилитрон обязательно должен быть маломощный, типа КС139Г в стеклянном корпусе, или соответствующий импортный, в другом случае сопротивление резистора R35 надо снизить примерно в два-три раза). Когда напряжение батареи упадет ниже этого уровня (выбранного с некоторым запасом, поскольку при 3 В МК еще может нормально работать, но часть напряжения батареи упадет на диоде VD2, кроме того, следует учитывать, смена батарейки может произойти не сразу), то компаратор перебросится в состояние логической единицы по выходу.

Программа (см. далее) это зарегистрирует и разделительная точка (пара светодиодов VD1 и VD2, рис. 14.3) перестанет мигать и будет гореть постоянно. Восстановление произойдет сразу, как только батарею сменят на свежую. Та же реакция будет, если просто отключить батарею тумблером «Бат» (S1 на рис. 14.2) или удалить ее. Для того, чтобы в этих случаях вход компаратора не оказывался «висящим в воздухе», и предназначен резистор R5. Ток через него настолько мал (около 1,5 мкА), что на разряд батареи это не оказывает влияния. C8 защищает вход от наведенных на этом резисторе помех.

При пропадании внешнего питания диод VD1 запирается, а VD2 открывается и напряжение батареи поступает на питание МК. Резистор R6 вместе с развязывающим конденсатором C2 предназначены для большей устойчивости работы МК в момент перепада напряжений при переключении питания, для той же цели служит конденсатор C7, установленный параллельно кнопке Кн1

(иначе при перепадах напряжения может спонтанно возникать прерывание, и часы войдут в режим установки, о котором см. далее). Одновременно с переключением питания становится равным нулю напряжение на стабилитроне, а т. к. при этом стабилитрон представляет собой обрыв в цепи, то установлен резистор R36, который служит тем же целям, что и R5. Компаратор работать перестает (точнее, он всегда будет показывать «нормальную» батарею), но нас это не волнует, т. к. индикации все равно нет. Тумблер «Бат» нужен для отключения батареи в случае, если вы хотите остановить часы надолго, а вот тумблер для включения сетевого питания тут совершенно не требуется (разве что на время отладки).

Программа

Полный текст программы часов приведен в *Приложении 5* (раздел «Программа для часов», листинг П5.1). Все подробности даны в виде комментариев к тексту программы, здесь мы разберем только общее построение и принцип работы.

При включении питания процессора все регистры обнулены, программа начинает работу с команды по метке RESET. Здесь она устанавливает соответствующие порты на выход (все, кроме двух входов компаратора и входа кнопки Кн1), затем делает нужные установки для таймеров и разрешает соответствующие прерывания.

Timer 0 у нас будет по событию переполнения управлять разрядами в режиме динамической индикации. При заданной частоте на входе Timer 0, равной 1/8 от тактовой (4 МГц), частота управления разрядами получится равной 4 МГц/8 = 500 кГц/256 (емкость счетчика), т. е. чуть меньше 2 кГц, а сами разряды (4 шт.) будут изменяться с частотой почти 500 Гц, что однозначно превышает порог заметности мигания.

Заметки на полях

Заметим, что при проектировании питания подобных устройств следует учитывать еще одно обстоятельство: в динамическом режиме для питания индикаторов пульсирующее напряжение использовать нельзя (как в схеме со статической индикацией вроде термометра из *главы 10*), т. к. обязательно возникнут биения между частотами питающего напряжения и переключения разрядов, и яркость свечения будет пульсировать. Потому напряжение +12 В необязательно должно быть стабилизированным, но совершенно необходим сглаживающий фильтр (некоторая неизбежно возникающая — см. *главу 4* — величина пульсаций, конечно, может присутствовать). На самом деле в данной конструкции это условие соблюдается автоматически, т. к. те же +12 В подаются и на вход стабилизатора +5 В, но в дальнейшем мы встретим конструкции, в которых питание индикаторов осуществляется от отдельной обмотки трансформатора, и там об этом забывать не следует.

16-разрядный Timer 1 у нас будет управлять собственно отсчетом времени по прерыванию сравнения. Для этого в регистры сравнения загружается число 62 500, а предварительный коэффициент деления задается равным 1/64, тогда прерывание таймера будет возникать с частотой 4 МГц /64/62500 = 1 Гц. На практике число для сравнения подгоняется под конкретный кварц, и обычно оказывается почему-то меньше теоретической величины 62 500 (так, в моем случае оно было равно 62 486).

Подробности

Как быстро подобрать коэффициент деления? Можно воспользоваться высокоточным частотомером (мультиметры, позволяющие измерять частоту, не подойдут решительно, а большинство радиолюбительских частотомеров пригодны лишь для ориентировочной прикидки) для измерения длительности секундного импульса на выводе OC1. При отсутствии такого прибора следует воспользоваться следующим приемом: установить часы с каким-то определенным коэффициентом (скажем, с теоретическим значением 62 500), например, по компьютерному времени, которое несложно выставить через Интернет очень точно. Так как небольшая ошибка все равно может сохраниться (см. далее процедуру установки), то после установки отметьте точную разницу в секундах между моментом смены показаний минут нашей конструкции и компьютерных часов, и запишите ее. Потом выдержите часы достаточно длительный промежуток времени (чем длиннее, тем точнее), в течение которого они не должны «сбоить» с необходимостью переустановки времени. Снова точно установите компьютерное время и опять запишите разницу в момент смены минут.

Таким образом вы получите величину ухода часов. Пусть, например, она составляет 200 с месяц в сторону отставания. Это значит, что у нас секундный интервал длиннее необходимого на $200/2592000 = 7{,}7 \cdot 10^{-5}$ часть, т. е. на 77 мкс (число 2 592 000 есть число секунд за 30 дней, проверьте). Эту же величину мы можем получить и с помощью частотомера. Настолько следует повысить частоту «тиков» таймера, для чего нужно уменьшить наш коэффициент деления на величину $62500 \cdot 7{,}7 \cdot 10^{-5} \approx 5$, т. е. в регистры таймера необходимо записать число 62 495. Отметьте, что, несмотря на кажущуюся достаточно высокую величину коэффициента деления 62 500, изменение его всего на единицу изменит ход часов на целых 40 секунд в месяц, т. е. более чем на секунду в сутки — это является следствием использования 16-разрядных счетчиков-таймеров, и крупнейшим недостатком применения МК для отсчета времени. Далее мы увидим, что для более тонкой подстройки нам придется изощряться, придумывая всякие хитрости (см. *главу 19*).

Кроме этого, в процедуре инициализации разрешается прерывание от кнопки Кн1 (INT1). Для кнопки Кн2 (объединенной с одним из входов компаратора) отдельного прерывания не требуется, ее состояние отслеживается непосредственно в процессе установки (см. далее). По окончании установок разрешаются прерывания (команда `sei`), и далее программа переходит к выполнению бесконечного цикла, во время которого производится мониторинг состояния определенных узлов.

Основная логика работы часов следующая. Каждую секунду, когда происходит прерывание Timer 1, счетчик секунд `sek` увеличивается на единицу (см. процедуру обработки прерывания TIM1 по метке mtime). Если его значение не равно 60, то больше ничего не происходит, если равно, то регистр `sek` обнуляется, и далее по цепочке обновляются значения текущего времени, хранящиеся в регистрах `emin`, `dmin`, `ehh` и `dhh` (см. их определения в начале программы).

Прерывание по переполнению Timer 0 для управления разрядами происходит независимо от прерывания Timer 1 и использует установленные в последнем значения часов. По Timer 0 обнуляются все выходы всех портов, управляющие индикацией, затем проверяется значение счетчика POS, отсчитывающего последовательные номера разрядов (от 0 до 3). Чтобы не тратить время на всякие проверки и обнуления, для организации счетчика до четырех здесь учитывается тот факт, что число 4 совпадает с числом комбинаций первых двух бит. Тогда для последовательного непрерывного счета (0—1—2—3—0—1…) достаточно каждый раз увеличивать счетчик на единицу (см. команду `inc POS` в конце процедуры), а в начале ее лишь обнулять старшие шесть бит (команда `andi POS,3`). Далее в зависимости от значения счетчика (`cpi POS,…`) устанавливаем питание нужного индикатора (`sbi PortD,…`) и вызываем процедуру установки маски сегментов `SEG_SET`, причем устанавливаемая маска определяется значением данного разряда в часах.

В процедуре `SEG_SET` и собственно процедурах установки маски (`OUT_x`) я предлагаю вам разобраться самостоятельно. Единственный вопрос, который у вас здесь может возникнуть — почему не применить удобный способ непосредственного задания маски рисунков цифр через загрузку констант командой `lpm`, как я писал в *главе 13* — вместо многочисленных «тупых» процедур отдельно для каждой цифры? Дело в том, что маску удобно использовать, если у вас выводы управления разрядами идут подряд (к примеру, когда биты PortD 0—7 соответствуют битам маски 0—7). Тогда маску достаточно «приложить» к регистру порта, и программа резко сокращается. А здесь это сделать нельзя, т. к. «перестраивание» маски под выводы различных портов займет не меньше места, чем простая и понятная прямая установка выводов.

Процедура установки часов работает следующим образом. При коротком нажатии на Кн1 возникает прерывание INT1 (процедура по метке `INTT1`), в котором первым делом проверяется, есть ли сетевое питание (бит 1 регистра `Flag`, см. далее), иначе и сама установка не требуется. Далее, как обычно, запрещается само прерывание INT1 во избежание дребезга. Разрешается оно в прерывании Timer 1 (см. начало текста процедуры `TIM1`), которое, как мы уже знаем, происходит каждую секунду. Таким образом время нечувстви-

тельности, в течение которого можно отпустить кнопку без последствий (без перескока на произвольный разряд), составляет случайную величину от 0 до 1 с. На самом это не совсем верное решение, и сделано так только для простоты — следовало бы пропустить одну секунду, и только потом разрешать, иначе вероятность дребезга все-таки остается большой.

Далее в прерывании INT1 устанавливается отдельный счетчик разрядов `set_up`, который будет считать от 1 до 4 (если он больше, то выходим из режима установки), и признак режима установки (бит 0 регистра `Flag`). Если этот признак установлен, то разряд, соответствующий установленному номеру в счетчике `set_up`, будет мигать. Это достигается с помощью вспомогательного счетчика `count` (см. процедуру `TIM1` по метке `CONT_1`). В этом же месте программы отслеживается состояние Кн2: если она нажата и удерживается, то каждую секунду происходит увеличение значения выбранного разряда на единицу, в тех пределах, в которых это допускается (для единиц минут — от 0 до 9, для десятков минут — от 0 до 5, для десятков часов — от 0 до 2, причем предел единиц часов зависит от значения десятков), далее значение опять обращается в ноль. Отпустив кнопку Кн2, вы фиксируете установленное значение, а нажав кратковременно на Кн1, переходите к следующему разряду. После прохождения всех разрядов, при последнем (пятом) нажатии Кн1, режим установки отменяется, т. е. бит 0 регистра `Flag` сбрасывается (см. процедуру по прерыванию INT1).

Немаловажная особенность этой конструкции — то, что во время установки счет времени прекращается (команды `rjmp END_TIM1` в процедуре `TIM1`, это было сделано по неопытности, на самом деле там можно было поставить просто `reti`, но для наглядности я ничего менять не стал), а при выходе из режима установки счетчик секунд устанавливается в состояние 59 (команда `ldi sek,59`), т. е. счет сразу же начинается с новой минуты. Окончание установки — довольно важный момент, который можно организовать по-разному, но данный способ самый удобный, т. к. вам достаточно дождаться окончания текущей минуты по образцовым часам, и в этот момент сделать последнее нажатие, выйдя из режима установки, чтобы довольно точно синхронизировать время. Сравните, например, как неудачно выполнена ручная установка часов в Windows, где часы продолжают идти и во время установки. А если бы мы обнуляли счетчик секунд, вместо его установки в максимальное значение, то нам пришлось каждый раз устанавливать число минут на единицу большее, что неудобно.

Теперь об обеспечении режима автономной работы. Программа контроллера в непрерывном цикле опрашивает значение логического уровня на выводе номер 12 PinB,0, он же AIN+ (3,9 В эквивалентно логической единице), и когда оно становится равным нулю, принимает меры к снижению потребления,

в первую очередь за счет отключения внешних портов (см. процедуру `Disable`). Как только внешнее питание восстанавливается, автоматически возобновляется нормальный режим работы (`Restore`).

При перебрасывании компаратора в любою сторону происходит прерывание `ACOMPI`. В нем вывод 15 (OC1) отключается от таймера Timer 1, и устанавливается навсегда в логическую единицу, если состояние компаратора есть логическая единица (когда истощается или отключается батарейка). Тогда двоеточие горит постоянно. И наоборот, вывод этот опять подключается к автоматическому миганию, когда компаратор перебрасывается обратно в ноль.

Детали и конструкция

Для блока питания используем «внутренности» блока со встроенной вилкой, с номинальным напряжением питания 10 В и током не менее 500 мА (такие продаются для некоторых игровых консолей). Напряжение на холостом ходу у него будет составлять примерно 13—14 В, под нагрузкой 130 мА оно сядет как раз примерно до 11—12 В.

В качестве кнопок Кн1 и Кн2 с легким нажатием удобны обычные микропереключатели (известные в отечественном варианте под названием МП-1), но со специальной металлической лапкой-рычагом, которая предназначена для того, чтобы уменьшить усилие нажатия и увеличить его зону (вообще-то такие кнопки предназначены для применения в качестве концевых выключателей). Подойдут импортные кнопки типа SM5 (рис. 14.4). Тогда нам не придется портить внешний вид фальшпанели кнопками или устанавливать их где-то сзади, а можно поставить их прямо на плату индикаторов и просверлить в дымчатом оргстекле напротив них маленькие отверстия, через которые кнопку можно нажимать зубочисткой или другим острым предметом. Чтобы отверстие в оргстекле выглядело «фирменно», сверлить следует осторожно, на малых оборотах, затем вручную сверлом или зенковкой сделать аккуратную фаску с лицевой стороны и обработать отверстие маслом, чтобы оно не белело. Подобное решение еще хорошо тем, что случайное нажатие кнопок — беда почти всех бытовых электронных устройств — совершенно исключено.

После изготовления платы индикации сначала следует установить с обратной стороны разъем, а затем «обдуть» лицевую часть платы черной автомобильной эмалью из баллончика, не слишком густо (достаточно одного слоя), чтобы краска не затекла в отверстия. Потом на черную плату уже монтируются индикаторы, светодиоды разделительной точки и кнопки. Светодиоды нужно выбирать, естественно, того же цвета, что и индикаторы. Имейте в виду, что

сама по себе характеристика «желтый» или «зеленый» еще ни о чем не говорит, только в табл. 3.1, два зеленых цвета и три красных, а у разных изделий различных фирм их может быть еще больше. И чтобы разница не бросалась в глаза, приготовьтесь к тому, что покупать придется несколько разновидностей и подбирать их оттенок.

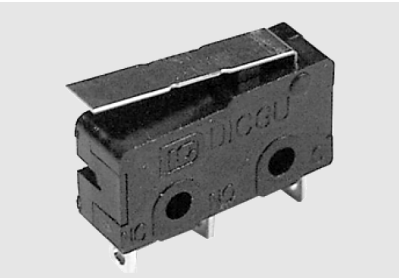


Рис. 14.4. Кнопка SM5 с лапкой-рычагом

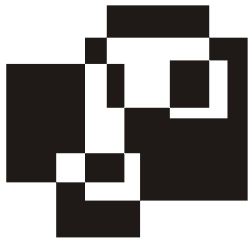
Под индикаторы указанного типоразмера 1″ подойдут светодиоды диаметром 3 мм, обычные 5-миллиметровые будут слишком выделяться, а под меньшие индикаторы потребуются светодиоды с еще меньшим диаметром. «Суперяркие» светодиоды сюда решительно не годятся. Светодиод при этом желательно иметь с диффузным рассеиванием, чтобы его было одинаково видно со всех углов зрения. Так что вопрос подбора LED может оказаться непростым — в упомянутой конструкции мне так и не удалось сначала подобрать «диффузный» тип, удовлетворяющий всем перечисленным требованиям (они оказались чересчур «лимонного» оттенка), и пришлось устанавливать светодиоды в прозрачном корпусе. Для каждого типа светодиодов придется подобрать резистор R34 (см. рис. 14.2) согласно необходимой яркости (для прозрачных номинал его будет больше, для диффузных — меньше). Устанавливать эту пару диодов следует не прямо друг над другом, а с некоторым наклоном, соответственно наклону цифры индикатора. Неплохо будут также выглядеть прямоугольные (5×2 мм) светодиоды, также под наклоном, только их боковые грани придется закрасить густой черной краской или аккуратно обернуть их качественной липкой лентой.

Я останавливаюсь на всех этих подробностях потому, что они имеют решающее значение для того, будет ли ваша конструкция выглядеть фирменно, или напоминать продукт творчества членов кружка юных техников из деревни Гадюкино. Затрачивать столько сил и средств на конструирование и пренебречь при этом нюансами внешнего вида просто не имеет смысла — если вы, конечно, конструируете бытовой прибор, а не утилитарную схему для

рабочих нужд. Но и в последнем случае гораздо приятнее брать в руки аккуратную и удобную в работе конструкцию, а не «голую» плату с болтающимися проводами.

Когда мы соединим платы управления и индикации кабелем и подключим питание, схема заработать сразу не может, потому что нужно запрограммировать МК. Для этого вы должны подключить к разъему X1 программатор и загрузить hex-файл с программой. Часы должны «затикать» светодиодами и показать все нули на индикаторах. Потом можно браться за установку времени.

Без сомнения, вы легко сможете доделать эту конструкцию, добавив в нее, к примеру, функции будильника. Причем это можно сделать в принципе даже без переделки схемы, если «повесить» функции установки и включения будильника на те же кнопки, разделив их с простой установкой за счет отсчета времени удержания кнопки (т. е. между нажатием и отпусканием). Сложнее, правда, будет обеспечить выход на «пищалку», но ее можно «повесить» на тот же вывод «мигалки» управления разделительными светодиодами, если при срабатывании будильника заполнять включенное состояние мигалки частотой 2 кГц, предназначенной для управления разрядами — например, переключая с этой частотой вывод OC1 то на вход, то на выход (при этом в обычном режиме «пищалка» будет еле слышно тикать). Но, разумеется, никто вас не заставляет применять именно МК 2313 — возьмите модель 8515, где выводов гораздо больше, и все окажется куда проще. Тем более, что в этом случае можно придумать и еще что-то, например, добавить маленькие разряды секундомера в углу передней панели, а будильник дополнить «милицейской» мигалкой, переключая красный и синий светодиоды попеременно.



Глава 15

Вычисления в МК и использование АЦП

> В давние, давние времена компьютеры занимались только своими прямыми обязанностями: они считали.
>
> *Вадим Житников «Компьютеры, математика и свобода»*

Обычные операции сложения и вычитания для 8-разрядных чисел (которые на поверку все равно оказываются 16-разрядными операциями) мы уже «проходили» в *главе 13*. Здесь мы попытаемся понять, как в 8-разрядном МК можно работать с многоразрядными и дробными величинами, в том числе осуществлять операции деления и умножения.

Подумаем сначала, — а с какими числами приходится работать на практике? Если говорить о целых числах, то большинство реальных нужд вполне укладывается в трехбайтовое значение (2^{24} или 16 777 216). Этот же диапазон дает достаточное для практики значение точности (7 десятичных разрядов), большие числа обычно округляют и записывают в виде «мантисса — порядок», с добавкой степени 10. То же касается и разрядности дробных чисел. При этом следует учесть, что любая арифметическая операция дает погрешности округления, которые могут накапливаться. Углубляться в этот достаточно сложный вопрос мы не будем, нам достаточно того факта, что оперируя с трехбайтовыми числами, как результатом обычных операций деления и умножения, мы не выходим за пределы погрешности в шестом знаке, что значительно превышает разрешение рядовых 10-разрядных АЦП — с числом градаций 1024, означающем ошибку уже в третьем, максимум (в реальности так не бывает) в четвертом знаке.

Потому, хотя в типовых приложениях для микропроцессоров оперируют либо 16-, либо 32-разрядными двоичными числами, мы в большинстве случаев ограничимся трехбайтовыми (24-разрядными) числами — нет смысла занимать

дефицитный регистр (причем в арифметических операциях — и не один), если он все равно всегда будет равен нулю. Однако возможность получения полных 32 разрядов также не следует упускать из виду, т. к. в некоторых случаях это может понадобиться.

Процедуры умножения для многобайтовых чисел

Чтобы более четко разделить задачи с различной разрядностью, приводимые в «аппнотах» процедуры для наших целей придется творчески переработать, тем более, что в них имеются ошибки. Ошибки эти мы разбирать подробно не будем, да и вообще не будем останавливаться на внутреннем механизме работы таких процедур — к чему углубляться в теорию вычислений? Желающих я отсылаю к упоминавшемуся фундаментальному труду [10]. Здесь мы займемся лишь практическими алгоритмами.

На рис. 15.1 приведена блок-схема алгоритма перемножения двух 16-разрядных беззнаковых чисел (MPY16U), скопированная из pdf-файла Application note AVR200. Жирным выделено необходимое исправление, то же следует сделать в алгоритме перемножения двух 8-разрядных чисел (MPY8U).

Ассемблерный текст процедур умножения можно найти в той же «аппноте», только представленной в виде asm-файла (его можно скачать с сайта Atmel по адресу, приведенному в *главе 13*, если в таблице с перечнем Application notes щелкнуть по значку диска, а не PDF-документа). Для внесения изменений найдите в тексте процедуру `mpy16u`, переставьте метку `m16u_1` вместо команды `brcc noad8`, перед которой она стоит, двумя командами ранее — перед командой `lsr mp16uH`. Аналогичную манипуляцию надо проделать в процедуре `mpy8u` с меткой `m8u_1`, если вы желаете воспользоваться 8-разрядным умножением.

Рассмотрите внимательно текст процедуры 16-разрядного умножения в «аппноте» 200 и обратите внимание на две особенности. Во-первых, для представления результата в общем случае требуется 4 байта (регистра), т. е. результат будет 32-разрядным числом, что понятно. Поэтому, во-вторых, разработчики используют в целях экономии одни и те же регистры для множителя и младших байтов результата, но называют их разными именами. Такой прием мы уже обсуждали, и ясно, для чего он применяется — для большей внятности текста процедуры. И тем не менее, вообще говоря, это недопустимо — слишком легко забыть про то, что под разными именами кроется один и тот же регистр, и использовать его еще где-то (не будете же вы, в самом

деле, держать неиспользуемыми аж целых 7 регистров только для того, чтобы один раз где-то что-то перемножить, правда?). Потому мы в дальнейшем обойдемся без таких фокусов — лучше написать внятные комментарии, а имена оставить уникальные, даже если они и не окажутся «говорящими».

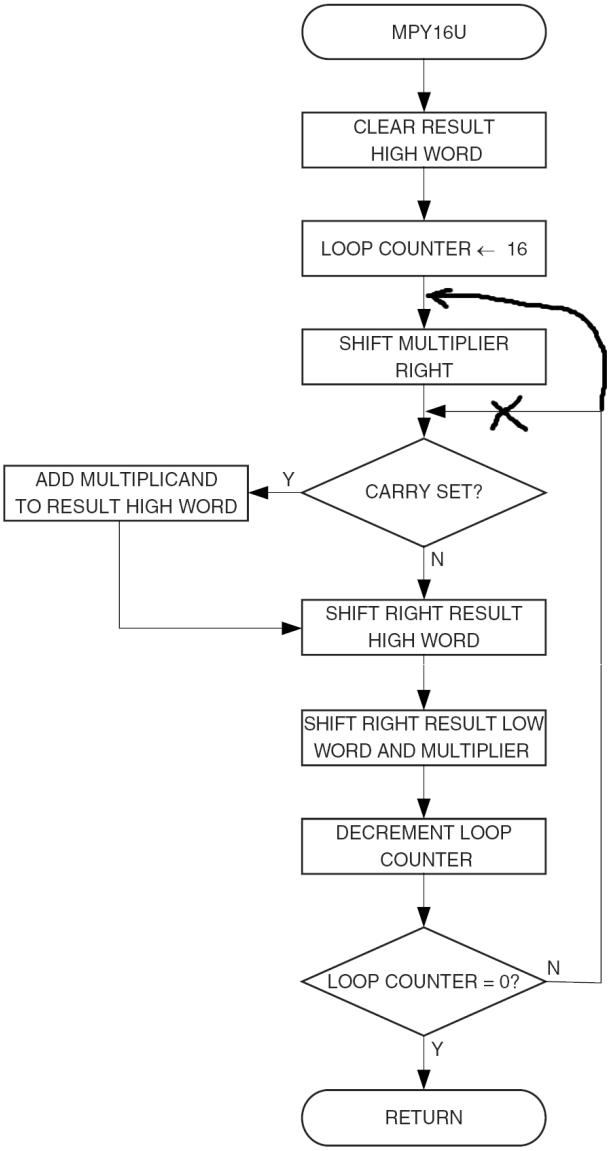


Рис. 15.1. Процедура перемножения двух 16-разрядных чисел из pdf-файла Atmel Application note AVR200 с исправленной ошибкой

На практике, как мы говорили ранее, 32-разрядное число (максимум 4 294 967 296, т. е. более 9 десятичных разрядов) с точки зрения точности в большинстве случаев избыточно. Если мы ограничимся 24 разрядами результата, то нам придется пожертвовать частью диапазона исходных чисел так, чтобы сумма двоичных разрядов сомножителей не превышала 24. Например, можно перемножать два 12-разрядных числа (в пределах 0—4095 каждое) или 10-разрядное (скажем, результат измерения АЦП) на 14-разрядный коэффициент (до 16 383). Так как при умножении точность не теряется, то этого оказывается более чем достаточным, чтобы обработать большинство практических величин.

Процедура перемножения двух таких величин в исходном 16-разрядном виде, с представлением результата в трехбайтовой форме может быть легко получена из исправленной нами процедуры MPY16U по «аппноте» 200, но я решил воспользоваться тем обстоятельством, что для контроллеров семейства Mega определены аппаратные операции умножения (в *Приложении 4* я их не привожу). Тогда алгоритм сильно упрощается, причем он легко модифицируется как для 32-, так и для 24-разрядного результата. Таким образом, для Tuny и Classic по-прежнему следует пользоваться обычными процедурами из «аппноты» (исправленными), а алгоритм для Mega приведен в листинге 15.1 (в названиях исходных переменных отражен факт основного назначения такой процедуры — для умножения неких данных на некий коэффициент). Сокращения LSB и MSB, которые нам еще встретятся не раз, означают least (most) significant bit — младший (старший) значащий разряд, по-русски МЗР и СЗР соответственно.

Листинг 15.1

```
.def dataL = r4 ;multiplicand low byte
.def dataH = r5 ;multiplicand high byte
.def KoeffL = r2 ;multiplier low byte
.def koeffH = r3 ;multiplier high byte
.def temp = r16 ;result byte 0 (LSB — младший разряд)
.def temp2 = r17 ;result byte 1
.def temp3 = r18 ;result byte 2 (MSB — старший разряд)
. . . . .
;*****************************************************************
;умножение двух 16-разрядных величин, только для Mega
;исходные величины dataH:dataL и KoeffH:KoeffL
;результат 3 байта temp2:temp1:temp
;*****************************************************************
Mu1616:
clr temp2 ;очистить старший
```

```
mul dataL,KoeffL ;умножаем младшие
mov temp,r0 ;в r0 младший результата операции mul
mov temp1,r1 ;в r01 старший результата операции mul
mul dataH,KoeffL ;умножаем старший на младший
add temp1,r0 ;в r0 младший результата операции mul
adc temp2,r1 ;в r01 старший результата операции mul
mul dataL,KoeffH ;умножаем младший на старший
add temp1,r0 ;в r0 младший результата операции mul
adc temp2,r1 ;в r01 старший результата операции mul
mul dataH,KoeffH ;умножаем старший на старший
add temp2,r0 ;4-й разряд нам тут не требуется, но он — в r01
ret
;****************************************************************
```

Как видите, эта процедура легко модифицируется под любую разрядность результата, если нужно получить полный 32-разрядный диапазон, просто добавьте еще один регистр для старшего разряда (`temp3`, к примеру) и одну строку кода перед командой `ret`:

```
adc temp3,r01
```

Естественно, можно просто обозначить `r01` через `temp3`, тогда и добавлять ничего не придется.

Процедуры деления для многобайтовых чисел

Деление — значительно более громоздкая процедура, чем умножение, требует больше регистров и занимает больше времени (MPY16U из «аппноты» занимает 153 такта, по уверению разработчиков, а аналогичная операция деления двух 16-разрядных чисел — от 235 до 251 тактов). Операции деления двух чисел (и для 8-, и для 16-разрядных) приведены в той же «аппноте» 200, и на этот раз без ошибок, но они не всегда удобны на практике: часто нам приходится делить результат какой-то ранее проведенной операции умножения или сложения, а он нередко выходит за пределы двух байтов.

Потому пришлось разрабатывать свои операции. Например, часто встречается необходимость вычислить среднее значение для уточнения результата по сумме отдельных измерений. Если даже само измерение укладывается в 16 разрядов, то сумма нескольких таких результатов уже должна занимать 3 байта. В то же время делитель — число измерений — может быть и относительно небольшим, и укладываться в один байт. В листинге 15.2 я привожу процедуру деления 32-разрядных чисел (на всякий случай) на однобайтное

число, которая представляет собой модификацию оригинальной процедуры из Application notes 200. Как и ранее, названия переменных отражают назначение процедуры — деление состояния некоего 4-байтового счетчика на число циклов счета (определения регистров-переменных не приводятся, комментарии сохранены из оригинального текста «аппноты», они соответствуют блок-схеме алгоритма, размещенной в pdf-файле).

Листинг 15.2

```
;*****************************************************************
;div32x8» — 32/8 деление беззнаковых чисел
;Делимое и результат в count_HH (старший), countTH,
;countTM, countTL (младший)
;делитель в cikle
;требуется четыре временных регистра dremL — dremHH
;из диапазона r16-r31 для хранения остатка
;*****************************************************************
div32x8:
     clr  dremL     ;clear remainder Low byte
     clr  dremM ;clear remainder
     clr  dremH ;clear remainder
     sub  dremHH,dremHH ;clear remainder High byte and carry
     ldi  cnt,33     ;init loop counter
d_1: rol  countTL ;shift left dividend
     rol     countTM
     rol     countTH
     rol     count_HH
     dec     cnt ;decrement counter
     brne    d_2 ;if done
     ret ;return
d_2: rol     dremL ;shift dividend into remainder
     rol     dremM
     rol     dremH
     rol     dremHH
     sub     dremL,cikle ;remainder = remainder — divisor
     sbci    dremM,0     ;
     sbci    dremH,0     ;
     sbci    dremHH,0     ;
     brcc    d_3 ;if result negative
     add     dremL,cikle ;restore remainder
     clr temp
     adc     dremM,temp
     adc     dremH,temp
     adc     dremHH,temp
```

```
     clc ;clear carry to be shifted into result
     rjmp     d_1 ;else
d_3: sec ;set carry to be shifted into result
     rjmp     d_1
;*********************конец 32x8
```

Многие подобные задачи на деление удается решить значительно более простым и менее громоздким методом, если заранее подгадать так, чтобы делитель оказался кратным степени двойки. Тогда все деление сводится, как мы знаем, к сдвигу разрядов вправо столько раз, какова степень двойки. Для примера предположим, что мы некую величину измерили 64 раза, и хотим узнать среднее. Пусть сумма укладывается в 2 байта, тогда вся процедура деления будет такой:

```
;деление на 64
     clr count_data ;счетчик до 6
div64L:
     lsr dataH ;сдвинули старший
     ror dataL ;сдвинули младший с переносом
     inc count_data
     cpi count_data,6
     brne div64L
```

Не правда ли, гораздо изящнее и понятнее? Попробуем от радости решить задачку, которая на первый взгляд требует, по крайней мере, знания высшей алгебры — умножить некое число на дробный коэффициент (вещественное число с «плавающей запятой»). Теоретически для этого требуется представить исходные числа в виде «мантисса — порядок», сложить порядки и перемножить мантиссы (см. [10]). Нам же неохота возиться с этим представлением, т. к. мы не проектируем универсальный компьютер, и в подавляющем большинстве реальных задач все конечные результаты у нас представляют собой целые числа.

На самом деле эта задача решается очень просто, если ее свести к последовательному умножению и делению целых чисел, представив реальное число в виде целой дроби с оговоренной точностью. Например, число 0,48576 можно представить как 48 576/100 000. И если нам требуется на такой коэффициент умножить, к примеру, результат какого-то измерения, равный 976, то тогда можно действовать, не выходя за рамки диапазона целых чисел: сначала умножить 976 на 48 576 (получится заведомо целое число 47 410 176), а потом поделить результат на 10^5, чисто механически перенеся запятую на пять разрядов. Получится 474,10176 или, если отбросить дробную часть, 474. Большая точность нам и не требуется, т. к. и исходное число было трехразрядным.

Улавливаете, к чему я клоню? С числами в десятичном виде хорошо работать руками, просто отсчитывая разряды. Нам же делить на сто тысяч в 8-разрядном МК крайне неудобно — представляете, насколько громоздкая процедура получится? Наше ноу-хау будет состоять в том, что мы для того, чтобы «вогнать» дробное число в целый диапазон, будем использовать не десятичную дробь, а двоичную — деление тогда сведется к описанной «механической» процедуре сдвига, аналогичной переносу запятой в десятичном виде.

Итак, чтобы умножить 976 на коэффициент 0,48576, следует сначала последний вручную умножить, например, на 2^{16} = 65 536, и тем самым получить числитель соответствующей двоичной дроби (у которой знаменатель равен 65 536) — он будет равен 31834,76736, или, с округлением до целого 31 835. Такой точности хватит, если исходные числа не выходят, как у нас, за пределы трех-четырех десятичных разрядов. Теперь мы в контроллере должны умножить исходную величину 976 на константу 31 835 и полученное число 31 070 960 (оно оказывается 4-байтовым — $01DA1AF0, потому нашу `Mu1616` придется чуть модифицировать, как сказано при ее описании ранее) сдвигаем на 16 разрядов вправо:

```
;в ddHH:  ddH:ddM:ddL число $01DA1AF0,
;его надо сдвинуть на 16 разрядов
     clr cnt
div16L:  ;деление на 65536
     lsr  ddHH ;сдвинули старший
     ror  ddH ;сдвинули 3-й
     ror  ddM ;сдвинули 2-й
     ror  ddL ;сдвинули младший
     inc  cnt
     cpi  cnt,16
     brne div16L ;сдвинули-поделили на 2 в 16
```

В результате, как вы можете легко проверить, старшие байты будут нулевыми, а в `ddM:ddL` окажется число 474 — тот же самый результат. Но и это еще не все, такая процедура приведена скорее для иллюстрации общего принципа. Ее можно еще больше упростить, если обратить внимание на то, что сдвиг на восемь разрядов есть просто перенос значения одного байта в соседний (в старший, если сдвиг влево, и в младший — если вправо). Итого получится, что для сдвига на 16 разрядов вправо нам надо всего-навсего отбросить два младших байта, и взять из исходного числа два старших `ddHH:ddH` — это и будет результат. Проверьте — $01DA и есть 474. Никаких других действий вообще не требуется!

Если степень знаменателя дроби, как в данном случае, кратна 8, то действительно никакого деления, даже в виде сдвига, не требуется, но чаще всего это

не так. Однако и тут приведенный принцип может помочь: например, при делении на 2^{15} (что может потребоваться, если, например, в нашем примере константа больше единицы) вместо пятнадцатикратного сдвига вправо результат можно сдвинуть на один разряд влево (фактически умножив число на два), а потом уже выделить из него старшие два байта. Итого процедура будет состоять из четырех операций сдвига и займет четыре такта. А в виде циклического сдвига на 15, как ранее, нам требуется в каждом цикле сделать четыре операции сдвига и одну увеличения счетчика: $15 \times 5 = 75$ простых однотактных операций, и еще 15 операций сравнения, из которых 14 займут два такта — итого 104 такта. А решение «в лоб» на основе операций целочисленного деления в несколько раз превышало бы и эту величину. Существенная разница, правда? Вот такая специальная арифметика в МК.

Операции с числами в формате BCD

Это важная группа операций, ведь значительная часть устройств на основе МК предназначена для демонстрации чисел в том или ином виде. Это, естественно, можно делать только в десятичном формате, в то время как внутреннее представление чисел в регистрах двоичное. В некоторых микропроцессорных системах (в их число входит семейство *x*51 от Intel и, кстати, *x*86) даже имеется специальная инструкция для т. н. *двоично-десятичной коррекции*, которая позволяет получить верный результат при сложении двоично-десятичных чисел в упакованном формате (о BCD-форматах см. *главу 7*). Но в системе команд AVR такой инструкции нет, и, в общем-то, она все равно не очень-то полезна, т. к. математические операции в любом случае удобнее выполнять в «родной» двоичной форме, а для представления на дисплее числа так или иначе приходится «распаковывать». В ПК этим незаметно для пользователя занимаются процедуры на языках высокого уровня (да так успешно, что приходится скорее озадачиваться обратной проблемой — представлением десятичных чисел в двоичной/шестнадцатеричной форме), ну а на уровне ассемблера десятичные преобразования приходится делать, что называется, ручками.

Заметки на полях

Традиционная область использования команд двоично-десятичной коррекции, в том числе и в процессорах *x*86 — манипуляции со значением времени, полученным из микросхем RTC, в которых часы, минуты и секунды всегда хранятся в упакованном BCD-формате. Как вы увидите далее, такой формат хранения довольно удобен на практике. Однако область применения микроконтроллерных систем далеко не исчерпывается подсчетом и демонстрацией времени, потому нам придется выйти за рамки однобайтовых кодов, для которых, собственно, инструкция коррекции и создавалась. Уже для двухбайтовых чисел ее применение вызывает только лишние сложности.

В области BCD-преобразований есть три основные задачи:

- Преобразование двоичного/шестнадцатеричного числа в упакованный BCD-формат.
- Распаковка упакованного BCD-формата для непосредственного представления десятичных чисел с целью их вывода на дисплей.
- Обратное преобразование упакованного BCD-формата в двоичный/шестнадцатеричный для выполнения над ним, например, арифметических действий.

Некоторые процедуры для этой цели приведены в фирменной Application notes 204. При их использовании нужно учесть ряд моментов. Так, процедура bin2BCD8 для преобразования однобайтового числа в BCD работает только для чисел от 0 до 99 (для больших чисел нужен еще один байт, точнее, тетрада — в ней будет храниться старший разряд). В «аппноте» процедура представлена в универсальном виде, пригодном (при небольшой модификации) и для получения упакованного BCD, и для изначально распакованного (результат в двух отдельных байтах). Чтобы не путаться, приведу здесь ее вариант (листинг 15.3), который заодно более экономичный по количеству используемых регистров. Исходное hex-число содержится в регистре temp, распакованный результат — в temp1:temp. Как и в предыдущих случаях, комментарии сохранены из исходного текста.

Листинг 15.3

```
;преобразование 8-разрядного hex в неупакованный BCD
;вход hex= temp, выход BCD  temp1-старший; temp – младший
;эта процедура работает только для исходного hex от 0 до 99
bin2bcd8:
     clr      temp1     ;clear result MSD
bBCD8_1:subi  temp,10   ;input = input – 10
     brcs     bBCD8_2   ;abort if carry set
     inc      temp1     ;inc MSD
     rjmp     bBCD8_1   ;loop again
bBCD8_2:subi  temp,-10  ;compensate extra subtraction
ret
```

В листинге 15.4 приведено одно из решений обратной задачи — преобразования упакованного BCD (например, тех же значений часов, минут и секунд из RTC) в hex-число, после чего с ним можно производить арифметические действия. По сравнению с «фирменной» BCD2bin8 эта процедура хоть и немного длиннее, но понятнее и более предсказуема по времени выполнения («фирменная» может занимать от 3 до 48 тактов).

Листинг 15.4

```
;на входе в temp упакованное BCD-значение
;на выходе в temp hex-значение
;temp1 — вспомогательный регистр для промежуточного хранения temp
;действительна только для семейства Mega
HEX_BCD:
     mov temp1,temp
     andi temp,0b11110000  ;распаковываем — старший
     swap temp  ;старший в младшей тетраде
     mul temp,mult10  ;умножаем на 10, в r0 результат умножения
     mov temp,temp1  ;возвращаемся к исходному
     andi temp,0b00001111  ;младший
     add temp,r0  ;получили hex
ret
```

Более громоздкая задача — преобразование многоразрядных чисел. Преобразовывать BCD-числа, состоящие более чем из одного байта, обратно в HEX-формат приходится крайне редко, зато задача прямого преобразования возникает на каждом шагу. Я здесь приведу отсутствующую в «аппноте» 204 процедуру конвертации чисел, выходящих за рамки 16-разрядного диапазона. Например, такая задача может возникнуть при конструировании многоразрядных счетчиков. Ограничимся диапазоном в 7 десятичных знаков (9 999 999), тогда исходное число будет укладываться в 3 байта (24 разряда). В целях универсальности в процедуре, которая приводится далее в листинге 15.5, на выходе получается отдельно неупакованный (сразу для индикации) и упакованный десятичный формат. Сократить число необходимых регистров можно, если большую часть результатов сразу записывать в SRAM — в дальнейшем мы так и будем поступать, а здесь для наглядности работаем только с регистрами.

Отметим, что процедура bin2BCD24 сделана на основе «фирменной» bin2BCD16 и, как и последняя, использует хитрый прием с записью значений в регистры по адресам памяти: так можно производить над адресами разные манипуляции, меняя регистры (аналогично адресной арифметике в языке C). Как и в других случаях, сохранена часть оригинальных комментариев из исходной «фирменной» процедуры.

Листинг 15.5

```
;процедура преобразования 3-байтового hex в упакованный (4 регистра)
;и неупакованный (7 регистров) BCD
;исходное значение в регистрах
```

```
.def   Count0  = r25
.def   Count1  = r26
.def  Count2  = r27
;на выходе упакованный BCD в регистрах
.def  tBCD0  =r13     ;BCD value digits 1 and 0
.def  tBCD1  =r14     ;BCD value digits 3 and 2
.def  tBCD2  =r15     ;BCD value digit 4,5
.def  tBCD3  =r16     ;BCD value digit 6
;на выходе неупакованный BCD в регистрах
.def  N1  = r1 ;младший
.def  N2  = r2
.def  N3  = r3
.def  N4  = r4
.def  N5  = r5
.def  N6  = r6
.def  N7  = r7 ;старший
;вспомогательные регистры
.def  cnt16a  =r18  ;счетчик цикла
.def  tmp16a  =r19  ;временное значение
;адреса регистров в памяти
.equ  AtBCD0  =13   ;address of tBCD0
.equ  AtBCD3  =16   ;address of tBCD3
bin2BCD24:
     ldi  cnt16a,24  ;Init loop counter
     clr  tBCD3
     clr  tBCD2      ;clear result (4 bytes)
     clr  tBCD1
     clr  tBCD0
     clr  ZH     ;clear ZH (not needed for AT90Sxx0x)
bBCDx_1:lsl     Count0     ;shift input value
     rol  Count1  ;through all bytes
     rol  Count2  ;through all bytes
     rol  tBCD0
     rol  tBCD1
     rol  tBCD2
     rol  tBCD3
     dec  cnt16a  ;decrement loop counter
     brne bBCDx_2 ;if counter not zero
;распаковка
     ldi temp,0b00001111
     mov N1,tBCD0
     and N1,temp
     mov N2,tBCD0
     swap N2
     and N2,temp
```

```
     mov N3,tBCD1
     and N3,temp
     mov N4,tBCD1
     swap N4
     and N4,temp
     mov N5,tBCD2
     and N5,temp
     mov N6,tBCD2
     swap N6
     and N6,temp
     mov N7,tBCD3
     ret ; return
bBCDx_2:ldi  r30,AtBCD3+1  ;Z points to result MSB + 1
bBCDx_3:
     ld    tmp16a,-Z    ;get (Z) with pre-decrement
     subi  tmp16a,-$03  ;add 0x03
     sbrc  tmp16a,3     ;if bit 3 not clear
     st    Z,tmp16a     ;store back
     ld    tmp16a,Z     ;get (Z)
     subi  tmp16a,-$30  ;add 0x30
     sbrc  tmp16a,7     ;if bit 7 not clear
     st    Z,tmp16a     ;store back
     cpi   ZL,AtBCD0    ;done all three?
     brne  bBCDx_3      ;loop again if not
     rjmp  bBCDx_1
```

Использование встроенного АЦП

Встроенный АЦП последовательного приближения входит в состав почти всех МК семейства Mega и большинства МК семейства Tuny, кроме простейших младших моделей. В семействе Classic был только один тип МК со встроенным АЦП — AT90S8535 — несколько доработанный вариант популярного AT90S8515. На примере его Mega-версии под названием ATmega8535 мы в дальнейшем и разберем работу встроенного АЦП, но сначала стоит сделать несколько общих замечаний.

Все встроенные АЦП многоканальные и 10-разрядные (за небольшим исключением — например, в ATmega8 из 6 каналов только четыре имеют разрешение 10 разрядов, а оставшиеся два — 8). Многоканальность означает, что имеется только одно ядро преобразователя, которое по желанию программиста может подключаться к одному из входов через аналоговый мультиплексор, наподобие разобранного в *главе 8* 561КП2. Если вы, как чаще всего и бывает, задействуете лишь часть входов, то остальные могут использоваться как обычные порты ввода/вывода. В разных моделях число каналов колеб-

лется от 4 до 16, причем в некоторых из них выводы АЦП можно коммутировать попарно так, чтобы получить АЦП с дифференциальным входом (тогда измеряется разность напряжений между этими входами, а не абсолютное значение относительно «земли»). Добавим еще, что в некоторых моделях все или часть входов в дифференциальном режиме могут иметь добавочный коэффициент усиления (10, 20 или 200).

Все эти «примочки» дополнительно снижают и без того не слишком высокую точность АЦП, которая номинально составляет для несимметричного (недифференциального) входа ±2 LSB, плюс еще 0,5 LSB за счет нелинейности по всей шкале. Фактически такой АЦП с точки зрения абсолютной точности соответствует 8-разрядному. При соблюдении всех условий эту точность, впрочем, можно повысить, правда, условия довольно жесткие и включают в себя как «правильную» разводку выводов АЦП, так и, например, требование остановки цифровых узлов на время измерения, чтобы исключить наводки (специальный режим ADC Noise Reduction). В дифференциальном режиме есть свои специальные приемы повышения точности. В общем, как и всегда в таких случаях, для получения хорошего результата аналого-цифрового преобразования требуются определенные усилия и некоторый опыт.

Чтобы не углубляться в детали этого процесса и не загромождать программу, мы в дальнейшем поступим проще — предпримем ряд мер, чтобы обеспечить стабильность результата, а абсолютную ошибку скомпенсируем за счет калибровки, которая все равно потребуется. Для начала давайте посчитаем, какие, собственно, ошибки нас могут устроить. Максимально достижимая точность с помощью 10-разрядного преобразователя составляет 0,1% (1/1024, или ±0,5 LSB) *приведенной погрешности* (т. е. погрешности от всей шкалы измерения). Для бытовых измерений это достаточно высокая величина, например, большинство портативных мультиметров имеют точность раз в пять хуже, обладая погрешностью порядка 0,5%. АЦП в 10 разрядов может, например, обеспечить точность измерения температуры 0,1° для стоградусной шкалы (от –50 до +50°).

На самом деле нам такая *точность* не требуется — все равно термометр, подвешенный за окном или на стенке комнаты, никогда не покажет точную температуру, насколько бы он ни был точным сам по себе. На него будут влиять сквозняки, солнечные лучи, осветительные приборы, конвекция воздуха по нагретой стенке, тепловое излучение от оконных проемов — одним словом, все то, что определяет т. н. *методическую* погрешность. И для большинства бытовых измерений абсолютной точности в 8 разрядов (~0,4%) хватает, как говорится, «выше крыши». Это относится не только к температуре, но и к подавляющему большинству других бытовых измерений. В большин-

стве случаев нам важно обеспечить не абсолютную точность, а, во-первых, *стабильность* показаний (чтобы в одинаковых условиях прибор показывал одно и то же, и показания можно было бы сравнивать между собой), и, во-вторых, достаточную *разрешающую способность*, т. е. оптимальную цену деления прибора.

Заметки на полях

Необходимость последнего параметра можно проиллюстрировать на примере наручных часов — практически все они содержат секундную стрелку (или демонстрируют секунды на дисплее), хотя секундомер в жизни требуется не часто, да и уход таких часов от истинного времени (т. е. их абсолютная точность) может составлять минуты, что нас совсем не «напрягает». Просто без секундной стрелки нам как-то неуютно. Точно так же при измерении температуры следует демонстрировать десятые градуса, хотя термометр, повешенный, например, на высоте четвертого этажа, может показать на пару-другую градусов больше, чем термометр на уровне земли. Впрочем, излишняя разрешающая способность тоже ни к чему — если мы бы захотели демонстрировать ту же температуру с сотыми градуса, то они бы попросту мелькали на дисплее, не неся никакой информации.

После такого экскурса в теорию измерений мы можем сделать вывод, что погрешности встроенного АЦП нам в большинстве случаев хватит и без особых ухищрений, важно только, чтобы показания не «дребезжали». Цифровые помехи со стороны ядра МК, как показывает опыт, имеют значительно меньшее влияние на результат, чем внешние, потому режим Noise Reduction нам не потребуется. Уменьшение дребезга почти до нуля достигается тем, что, во-первых, на входе канала ставится фильтр низкой частоты для устранения неизбежных в совмещенных аналого-цифровых схемах наводок на внешние цепи. Обычно достаточно керамического конденсатора порядка 0,1—1 мкФ, хотя в критичных случаях фирменное руководство рекомендует еще последовательно с ним включать индуктивность (порядка 10 мкГн), которую, добавим, для простоты можно заменить на резистор (несколько единиц или десятков килоом). Во-вторых, мы будем измерять несколько раз, и значения отдельных измерений усреднять — это самый эффективный способ повышения стабильности показаний, который я рекомендую для всех случаев, даже и тогда, когда соблюдены все фирменные рекомендации по повышению точности измерений (и в этом случае — особенно!). Это хоть и загромождает программу, но полученный эффект оправдывает такое усложнение.

Наконец, остановимся на источнике опорного напряжения, который, как мы знаем из *главы 10*, влияет на точность АЦП напрямую. Встроенные АЦП в МК AVR могут использовать три источника опорного напряжения на выбор: внешний, встроенный и напряжение питания аналоговой части (оно всегда в таких случаях отдельное от питания цифровой, хотя в простейших случаях это может быть один и тот же источник).

Встроенным источником опорного напряжения 2,56 В я пользоваться не рекомендую, прежде всего потому, что его величина может «гулять» в значительных пределах (до ±0,3 В), и зависит к тому же от напряжения питания, что в достаточной степени обессмысливает его использование. Единственным аргументом «за» является сама величина 2,56 В, что позволяет без сложных арифметических преобразований получать на выходе число измеряемых милливольт. Выходное значение АЦП (для несимметричного входа) выражается формулой:

$$N = 1024\frac{U_{\text{вх}}}{U_{\text{оп}}}.$$

Поэтому при $U_{\text{оп}}$ = 256 мВ, выходная величина N будет представлять учетверенное значение входного напряжения в милливольтах. Его легко привести к целому числу милливольт, просто сдвинув результат на два разряда вправо.

Однако такое измерение будет достаточно неточным и с искусственно пониженным разрешением (мы «легким движением руки» зачем-то превращаем 10-разрядный АЦП в 8-разрядный). Поэтому во всех случаях, когда требуется обеспечить абсолютную точность (например, при работе АЦП в составе мультиметра, где нас интересуют именно абсолютные значения в вольтах), следует использовать внешний точный источник опорного напряжения, тем более что они вполне доступны, хотя и не всегда дешевы (так, один из самых дорогих — прецизионный MAX873 с напряжением 2,5 В имеет разброс напряжения 1,5—3 мВ при температурной стабильности 2,5—7 мВ во всем диапазоне температур, и стоит порядка 10 долл.). Важным преимуществом такого способа служит возможность выбора опорного напряжения из более удобных величин (например, 2,048 В), что позволит не терять разрешение встроенного АЦП.

Если же нам требуется не измерять напряжение в абсолютных вольтах, а получать какие-то иные физические величины, то при работе от встроенного источника мы к тому же не можем воспользоваться способом повышения точности путем относительных измерений (запитав внешнюю измерительную схему от того же источника, чтобы скомпенсировать его изменения, например, с температурой). При этом нам в любом случае понадобится довольно сложная арифметика для пересчета показаний в физические величины, и тогда проще всего выбрать в качестве опорного источник аналогового питания, т. к. это только повысит достоверность измерений и сделает схему проще и дешевле.

Пару слов о самой организации измерений. АЦП последовательного приближения должен управляться определенной тактовой частотой, для чего в его состав входит делитель тактовой частоты самого МК, подобный предвари-

тельному делителю у таймеров. Устанавливать максимально возможную частоту (которая равна половине от тактовой) не рекомендуется, а лучше подбирать коэффициент деления так, чтобы тактовая частота АЦП укладывалась в промежуток от 50 до 200 кГц. Например, для тактовой частоты МК 4 МГц подойдет коэффициент деления 32, тогда частота АЦП составит 125 кГц. Преобразование может идти в непрерывном режиме (после окончания преобразования сразу начинается следующее), запускаться автоматически по некоторым прерываниям (не для всех типов AVR), или каждый раз запускаться по команде. Мы будем применять только последний «ручной» режим, т. к. нам для осреднения результатов тогда удобно точно отсчитывать число преобразований. В таком режиме на одно преобразование уходит 14 тактов, поэтому для приведенного примера с частотой 125 кГц время преобразования составит приблизительно 9 мс.

В любом случае по окончании процесса преобразования вызывается прерывание АЦП, и результат измерения читается из соответствующих регистров. Так как число 10-разрядное, то оно займет два байта, у которых старшие 6 разрядов равны нулю. Это удобно, т. к. мы можем без опасений суммировать до 64 (2^6) результатов, не привлекая дополнительных переменных, и затем простым сдвигом, как мы обсуждали ранее, вычислять среднее.

Измеритель температуры и давления на AVR

Для иллюстрации практического использования встроенного АЦП мы сконструируем измеритель температуры и атмосферного давления. Для измерения температуры мы заимствуем аналоговую часть схемы термометра из *главы 10*, перенеся ее сюда практически без изменений, за исключением того, что здесь мы запитаем схему от двуполярного источника ±5 В, чтобы обеспечить более удобный нам диапазон входных напряжений АЦП, начинающийся от 0 В в положительную сторону. Это позволит нам включить АЦП в несимметричном режиме, а не в дифференциальном, что упрощает схему и обеспечивает максимальное разрешение.

С датчиком атмосферного давления все еще проще — ряд фирм выпускают готовые датчики давления. Мы выберем барометрический датчик MPX4115 фирмы Motorola, питающийся от напряжения 5 В и имеющий удобный диапазон выхода примерно от 0,2 до 4,6 В. Крупный недостаток таких датчиков с нашей точки зрения — то, что погрешность привязана к абсолютной шкале (в данном случае от 15 до 115 кПа, что составляет примерно 11 и 860 мм рт. ст. соответственно) и составляет не менее 1,5%. Это без учета заводского раз-

броса (устраняется калибровкой) и зависимости выходного напряжения от напряжения питания (устраняется путем относительных измерений — питанием АЦП и датчика от одного источника). Но даже при этих условиях 1,5% от всей шкалы в 850 мм рт. ст. составит более 12 мм рт. ст. Это, конечно, недопустимо высокая погрешность для измерения атмосферного давления, которое на практике меняется в десятикратно меньших пределах — для большей части России, кроме горных местностей, можно выбирать диапазон от 700 до 800 мм рт. ст., даже с запасом. На самом деле это не должно нас пугать — как показал опыт, такой диапазон нас устраивает с точки зрения разрешения (одному мм рт. ст. будет соответствовать около одного разряда АЦП), а стабильность датчика оказывается вполне на высоте и обеспечивает при надлежащей калибровке разброс в пределах ±1 мм рт. ст.

При этом учтем, что большая абсолютная точность нам не требуется, как и в случае температуры — для небольших высот над уровнем моря можно считать, что при изменении высоты на каждые 10 м давление меняется примерно на 1 мм рт. ст., так что в пределах такого города, как Москва, с естественными перепадами высот 50 и более метров, оно само по себе будет «гулять» в пределах 5 мм рт. ст., даже без учета этажности зданий. И нам все равно целесообразно будет подогнать результат «по месту» так, чтобы не иметь крупных расхождений с прогнозом погоды по телевизору, иначе от показаний прибора будет мало пользы.

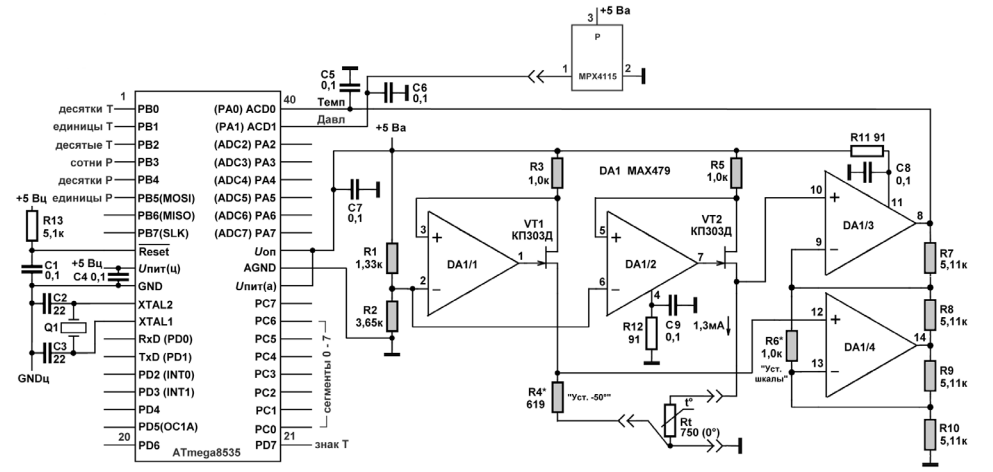


Рис. 15.2. Схема измерителя температуры и давления на МК ATmega8535

Схема

Схема такого прибора будет выглядеть так, как показано на рис. 15.2. Чтобы не загромождать схему, здесь не показан узел индикации, т. к. он аналогичен тому, что используется в часах из *главы 14*, за исключением того, что должен содержать не четыре, а шесть разрядов (показания в формате «33,3»° и «760» мм рт. ст.). К ним можно добавить постоянно горящие индикаторы, показывающие единицы измерения (см. рис. 15.3, где они изготовлены на основе шестнадцатисегментных индикаторов типа PSA-05). Так как здесь выводов портов хватает, то можно назначить для управления разряды подряд (например, разряды порта C от PC0 до PC6 для управления сегментами и порта B от PB0 до PB5 для управления разрядами) и использовать для вывода цифры прием с формированием маски в виде констант (см. *главу 13*), что заметно сократит программу. Кроме того, надо не забыть знак температуры, который удобно изготовить из отдельного плоского светодиода. В остальном принцип индикации точно такой же, как в часах, и мы остановимся на подробностях чуть далее, когда будем разбирать программу.

Рис. 15.3. Расположение индикаторов измерителя температуры и давления

Не показан на схеме и программирующий разъем, который полностью одинаков для любой схемы на AVR и показан на рис. 13.4 и 15.2 (соответствующие выводы для ATmega8535 подписаны на схеме рис. 15.2). То, что вывод MOSI (вывод 6) совпадает с выводом индикации единиц давления, вас смущать уже не должно. Однако незадействованные в других функциях выводы программирования (в данном случае MISO и SLK, выводы 7 и 8) следует подсоединить к питанию +5 Вц «подтягивающими» резисторами номиналом от 1 до 10 кОм (на схеме не показаны), так же, как и вывод Reset, только, естественно, без каких-либо конденсаторов (на схеме для вывода Reset указан номинал резистора 5,1 кОм). Как и RC-цепочка для Reset, «подтягивающие» резисторы для выводов программирования в принципе не требуются, однако их следует устанавливать. В тех случаях, когда схема представляет собой временный макет, без этих деталей можно обойтись, однако в работающей

схеме без них могут быть неприятности, о чем мы уже говорили в *главе 12*. Если разъем программирования вообще не предусматривается, то устанавливать резисторы к выводам программирования не нужно.

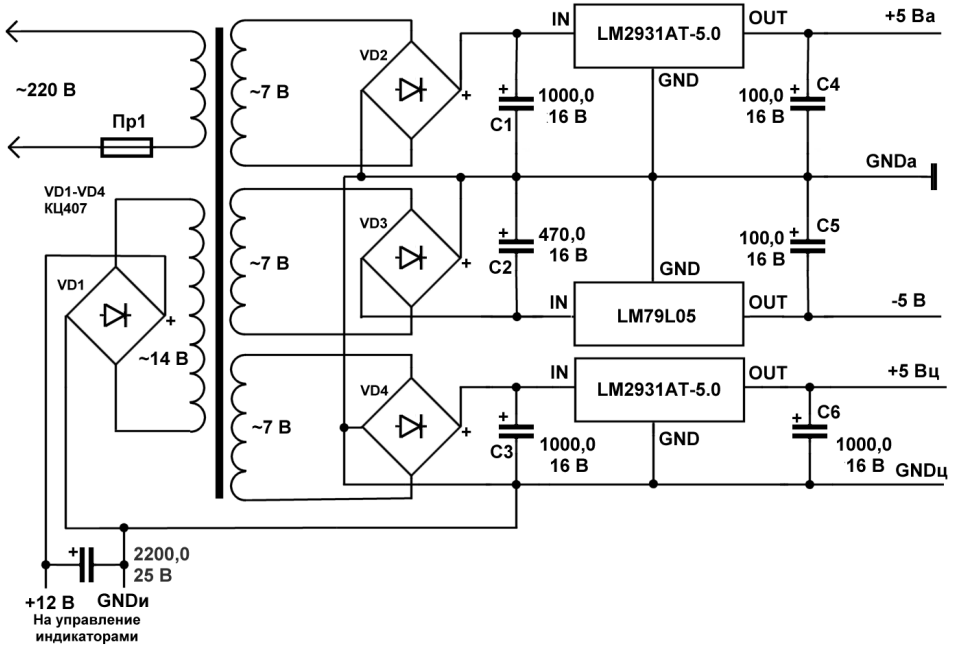


Рис. 15.4. Схема источника питания для измерителя температуры и давления

Схема источника питания показана на рис. 15.4. Измеритель имеет четыре питания (+5 Вц, ±5 Ва и +12 В для индикации) и три «земли», причем обычным значком «⊥» здесь обозначена аналоговая «земля» CNDa. Линия цифровой «земли» обозначена GNDц, кроме этого, имеется еще общий провод индикаторов GNDи. Все три «земли» соединяются только на плате источника питания. Отмечу, что готовый трансформатор с характеристиками, указанными на схеме, вы можете не найти. Поэтому смело выбирайте тороидальный трансформатор мощностью порядка 10—15 Вт на напряжение вторичной обмотки 10—14 В (для индикаторов), измерьте на нем число витков на вольт (как описано в *главе 4*), и домотайте три одинаковых обмотки на 7—8 В каждая поверх существующих, проводом не меньше, чем 0,3 мм в диаметре. Удобнее всего их мотать одновременно сложенным втрое проводом заранее рассчитанной длины.

Теперь немного разберемся с температурой. Сопротивление датчика составляет 760 Ом при 0 °С (≈ 610 Ом при –50°) и имеет крутизну примерно 3 Ом/° (о датчике см. *главу 10*). Величины резисторов в аналоговой части измерителя подогнаны так, чтобы обеспечить ток через датчик 1,3 мА. Таким образом напряжение на датчике в диапазоне температур от –50° до +50 °С будет меняться на 400 мВ, т. е. на выходе дифференциального усилителя (с учетом его коэффициента усиления около 12) диапазон напряжений составит примерно 4,9 В. Таким образом мы будем использовать весь диапазон АЦП (от 0 до $U_{оп}$) в полной мере с некоторым запасом. Резистор R4 устанавливает нижнюю границу диапазона, и здесь его нужно выбирать равным не сопротивлению датчика при 0°, как в схеме по рис. 10.8, а его сопротивлению при минимальной требуемой температуре. При указанных на схеме номиналах нижняя граница диапазона температур будет около –47°, а верхняя — около 55 °С. Для медного датчика с другим сопротивлением следует пересчитать коэффициент усиления усилителя (соответствующая формула приведена в *главе 6*, см. рис. 6.8). Это можно делать приблизительно — окончательную калибровку под реальный датчик мы будем производить путем изменения коэффициентов пересчета в программе МК.

Программа

Чтобы перейти к обсуждению непосредственно программы измерителя, нам нужно решить еще один принципиальный вопрос. Передаточная характеристика любого измерителя температуры, показывающего ее в градусах Цельсия, должна «ломаться» в нуле — ниже и выше абсолютные значения показаний возрастают. Так как мы тут действуем в области положительных напряжений, то этот вопрос придется решать самостоятельно (в АЦП типа 572ПВ2, напомним, определение абсолютной величины и индикация знака производилась автоматически).

Это несложно сделать, если представить формулу пересчета значений температуры в виде уравнения:

$$N = K(x - Z),$$

где N — число на индикаторе, x — текущий код АЦП, Z — код АЦП, соответствующий нулю градусов Цельсия (при наших установках он должен соответствовать примерно середине диапазона).

Чтобы величина по данной формуле всегда получалась положительная, нам придется сначала определять, что больше — x или Z, и вычитать из большего меньшее. Заодно при этой операции сравнения мы определяем значение знака. Если мы предположим, что в регистрах `AregH:AregL` содержится значение

текущего кода АЦП *x*, а в регистрах `KoeffH:KoeffL` значение коэффициента *Z*, то алгоритм будет выглядеть так, как иллюстрирует листинг 15.6.

Листинг 15.6

```
. . . . .
;вычисление знака:
      cp     AregL,KoeffL ;сравниваем x и Z
      cpc    AregH,KoeffH
      brsh b0
      ;если x меньше Z
      sub    KoeffL,AregL
      sbc    KoeffH,AregH
      mov    AregL,KoeffL ;меняем местами, чтобы температура
      mov    AregH,KoeffH ;оказалась опять в AregH:AregL
      sbi    PortD,7 ; знак -
      rjmp   m0
b0: ;если x больше Z
      sub    AregL,KoeffL
      sbc    AregH,KoeffH
      cbi    PortD,7 ;знак +
m0:
<умножение на коэффициент K>
. . . . .
```

Здесь разряд 7 порта D (вывод 21) управляет плоским светодиодом, который горит, если температура меньше нуля, и погашен в противном случае.

Давление занимает только положительную область значений, поэтому там такой сложной процедуры не понадобится. Если вы посмотрите на характеристику датчика в фирменном описании, то выясните, что он работает не с начала шкалы — нулевому напряжению на выходе (и, соответственно, нулевому коду АЦП) будет соответствовать некоторое значение давления. В результате можно ожидать, что в формуле пересчета значений давления, представленной в виде

$$N = K(x + Z),$$

все величины будут находиться в положительной области.

Физический смысл коэффициента *K* — крутизна характеристики датчиков в координатах «входной код АЦП — число на индикаторах». Умножение на *K* мы будем производить описанным методом — через представление его в виде двоичной дроби (за основу берется 2^{10} = 1024, этого будет достаточно). Вычисление ориентировочных значений коэффициентов *K* и *Z* поясняется далее, при описании процедуры калибровки.

Теперь можно окинуть взглядом собственно программу, которая целиком приведена в *Приложении 5* (раздел «Программа измерителя температуры и давления»). Как вы видите из таблицы прерываний, здесь используется всего один, самый простой Timer 0, который срабатывает с частотой около 2000 раз в секунду. В его обработчике по метке `TIM0` и заключена бóльшая часть функциональности.

В каждом цикле сначала проверяется счетчик `cRazr`, который отсчитывает разряды индикаторов (от 0 до 5). В соответствии с его значением происходит формирование кода индицируемого знака (по алгоритму вызова константы-маски знака, описанному в *главе 13*) и затем на нужный разряд подается питание.

Заметки на полях

Как видите, здесь формирование знака реализовано не очень красиво, и довольно громоздким способом — просто передачей управления на нужную процедуру в зависимости от значения счетчика. Сами же процедуры структурно одинаковы (меняются лишь адреса в памяти, из которых считываются значения разрядов индикатора и номера разрядов порта управления `PortB`). Программу в этой части можно слегка сократить (если просто вывести одинаковые операторы в отдельную процедуру, и задействовать локальные переменные), но я не стал этого делать, т. к. принципиально это ничего не изменит: места в памяти у нас достаточно, а программа, на мой взгляд, тем лучше читается, чем в ней меньше структурных блоков. (Упоминавшийся в *главе 13* Дейкстра, несомненно, схватился бы за сердце, услышав такое, но тем не менее это чистая правда — *весь* алгоритм окинуть взглядом легче, когда он максимально структурирован, но каждый отдельный фрагмент его проще понять, если не приходится «рыскать» по всему листингу.)

Интерес же представляет другой момент во всем этом — а нельзя ли было бы кардинально решить проблему, учитывая тот факт, что разряды считаются подряд (от нулевого до пятого), в регистре `PortB` они также расположены подряд, и ячейки SRAM, содержащие значения цифр, также идут подряд (начиная с `TdH`, см. секцию констант и определений)? Очень хочется как-то «свернуть» все шесть повторяющихся фрагментов в один, т. к. все равно все увязано со значением счетчика `cRazr`. Отсчитывать адрес, где хранится текущая цифра, несложно, просто прибавляя к начальному адресу (`TdH`) значение счетчика. В основном же это естественное желание упирается в тот факт, что невозможно простым способом перевести двоичное значение некоего регистра (в данном случае `cRazr`) в номер устанавливаемого бита в регистре `PortB`. Чтобы заменить «ручное» задание нужного бита в регистре `PortB` (см. пару команд `ldi temp,1<<RazrPdL/ out PORTB,temp`) на автоматическое в соответствии со значением `cRazr`, понадобится двоично-десятичный дешифратор, подобный по функциям микросхеме 561ИД1 (см. *главу 8*), программная реализация которого будет еще более громоздкой, чем данный алгоритм. Мы еще вернемся к этому вопросу в *главе 17*, когда нам понадобится управлять большим количеством индикаторов.

После формирования цифры программа переходит к довольно запутанному, на первый взгляд, алгоритму работы АЦП. На самом деле он не так уж и сложен. Управляют этим процессом две переменных: счетчик циклов `countCyk` и счетчик преобразований `count`. Первый из них увеличивается на единицу каждый раз, когда происходит прерывание таймера. Когда его величина достигает 32 (т. е. когда устанавливается единица в бите 5, см. команду `sbrs countCyk,5`), то значение счетчика сбрасывается для следующего цикла, и происходит запуск преобразования АЦП, причем для канала, соответствующего значению бита в регистре `Flag`, указывающего, что именно мы измеряем сейчас: температуру или давление. Таким образом измерения равномерно распределяются по времени.

Сами преобразования отсчитываются счетчиком `count` до 64 (т. е. цикл одного измерения занимает чуть более секунды: $32 \times 64 = 2048$ прерываний таймера, а в секунду их происходит примерно 1953). Когда это значение достигается, то мы переходим к обработке результатов по описанным ранее алгоритмам: сумма измерений делится на 64 (т. о. мы получаем среднее за секунду), затем вычитается или прибавляется значение «подставки», т. е. коэффициента *Z*, и полученная величина умножается на коэффициент *K*, точнее — на его целый эквивалент, полученный умножением на 1024. Произведение делится на это число и преобразуется к распакованному двоично-десятичному виду, отдельные цифры которого размещаются в памяти для последующей индикации. Как только очередной такой цикл заканчивается, меняется значение бита в регистре `Flag`, и таким образом давление и температура измеряются попеременно. В целом выходит, что значение каждой из величин меняется примерно раз в две секунды, и представляет собой среднее за половину этого периода.

Собственно результат измерения читается в прерывании АЦП (процедура по метке `readADC`), которое происходит автоматически по окончании каждого преобразования. В нем увеличивается значение счетчика `count`, извлекается из памяти предыдущее значение суммы показаний (в зависимости от регистра `Flag` — температуры или давления), считываются значения АЦП, суммируются и записываются обратно в память. Практически весь алгоритм мы описали — осталось только понять, как получить значения коэффициентов преобразования *K* и *Z* и затем выполнить точную калибровку.

Калибровка

Для того чтобы прибор заработал, в него необходимо ввести предварительные значения коэффициентов преобразования *K* и *Z*, причем желательно такие, чтобы они были достаточно близки к настоящим, и измеритель не пока-

зал бы нам «погоду на Марсе». В программе «зашиты» некие значения коэффициентов (см. процедуру `Reset`, секцию «Запись коэффициентов» в самом конце программы), которые подойдут вам, если вы не меняли характеристики схемы по рис. 15.2 и использовали тот же самый датчик давления. Как они получены?

Схема датчика температуры должна выдавать, как мы говорили, значение от 0 до 5 В в диапазоне температур примерно от –47 до 55 градусов. Следовательно, на 102 градуса у нас приходится 1024 градации АЦП, и крутизна характеристики составит 1020/1024 = 0,996 десятых градуса на единицу кода АЦП. Для вычислений в МК эту величину мы хотим умножить на 1024, так что можно было бы и не делить, ориентировочное значение коэффициента *K* так и будет 1020.

Величину *Z*, соответствующую 0 °С, вычислить также несложно. Мы полагаем, что нулю показаний соответствует температура –47°, тогда значение кода в нуле должно составить величину 470, поделенную на крутизну: 470/0,996 = 471.

Теперь разберемся с давлением. «Если повар нам не врет», то диапазон датчика, соответствующий изменению напряжения на его выходе от 0 до 4,6 В, составляет примерно 850 мм рт. ст. Это будет соответствовать изменению кодов примерно от 0 до 940 единиц, т. е. крутизна *K* равна 850/950 = 0,895 мм рт. ст. на единицу кода. В приведенном для наших расчетов виде это составит 0,895 × 1024 = 916. «Подставка» *Z* есть значение кода на нижней границе диапазона датчика, которая равна 11 мм рт. ст., соответственно, *Z* = 11/0,895 = 12 единиц. Полученные величины и «зашиваем» в программу.

После этого нужно сразу провести калибровку по температуре. Для этого следует запустить прибор и поместить датчик температуры в воду, записав для двух значений температур (как можно ближе к 0°, но не ниже его, и около 30—35 °С) показания датчика (t) и реальные значения температуры по образцовому термометру (t'). Они, естественно, будут различаться. Для расчета новых (правильных) значений коэффициентов K' и Z' достаточно решить относительно них систему уравнений:

$$t_1' = K'(x_1 - Z');$$
$$t_1 = K(x_1 - Z);$$
$$t_2' = K'(x_2 - Z');$$
$$t_2 = K(x_2 - Z).$$

Здесь величины со штрихами относятся к правильным (новым) значениям, а без штрихов — к старым, причем значение коэффициента *K* нужно подставлять в изначальной форме (а не умноженным на 1024). Система четырех

уравнений содержит четыре неизвестных, два из которых (величины кодов x_1 и x_2) вспомогательные. Если вы забыли, как решаются такие простые системы, купите любой справочник по математике для средней школы (или книжку по использованию Excel в алгебраических расчетах). Вычисленные значения (не забудьте *K* умножить на 1024) «забейте» в программу и перепрограммируйте контроллер.

Аналогично калибруется канал давления, только коэффициент *Z* в уравнениях не вычитается, а прибавляется к *x*. Но самое сложное здесь — получить действительные значения давления. Далеко не все научные лаборатории располагают образцовыми манометрами для измерения столь малых давлений с необходимой точностью. Поэтому самый простой, хотя и долгий метод — сравнивать показания датчика с данными по давлению, которые публикуются в Интернете. Есть сайты, которые публикуют погоду каждые 3 часа (это т. н. метеорологический интервал). Лучшие и наиболее популярные из них — **weather.yandex.ru** и **gismeteo.ru**. Причем лучше не ограничиваться данными одной какой-то службы, а обращаться к нескольким, отбрасывая явные ошибки и усредняя правдоподобные данные, с учетом того, что они публикуются с некоторым запаздыванием (отметьте показания прибора, например, в 9:00, а в Интернет лезьте примерно в 11:00). Данные радио и телевидения использовать нежелательно, т. к. текущие значения могут сообщаться с опозданием на полсуток, либо вообще отсутствовать, а по завтрашнему прогнозу, естественно, вы ничего не откалибруете.

Для получения двух точек дождитесь, пока давление на улице не станет достаточно низким, а затем, наоборот, высоким — экстремальные значения давления в регионе Москвы составляют примерно 720 и 770 мм рт. ст.[1] Чем дальше будут отстоять друг от друга значения, тем точнее калибровка. Для повышения точности можно усреднить коэффициенты, рассчитанные по нескольким парам значений давления, но это стоит делать только, если у вас хватит терпения вести наблюдения в течение нескольких месяцев, когда будет пройдено несколько минимумов и максимумов. Средние значения давления при калибровке лучше не учитывать, т. к. ошибка ее из-за узкого интервала и так достаточно велика.

Хранение констант в EEPROM

Полученные коэффициенты пересчета кода в физические величины мы «зашивали» прямо в программу МК. Излишне говорить, что это приемлемый

[1] Как раз сейчас, когда я редактирую эти строки, давление в Москве составило уникально низкую величину — 712 мм рт. ст. Но это, конечно, крайне редкая аномалия.

метод лишь тогда, когда изготавливается один-единственный экземпляр прибора, который стоит лично у вас на столе. Изготовить пару-другую экземпляров и подарить их кому-то уже не получится, поскольку при необходимости поправить коэффициенты пересчета владельцу придется обращаться к вам. Да и вообще, метод калибровки, при котором прибор требуется разобрать и переписать заново все его содержимое, выглядит как-то … некрасиво.

Логично придумать способ хранения констант, которые могут быть изменены в процессе эксплуатации, отдельно от программы. Для этой цели и служит энергонезависимая память данных, называемая EEPROM. Большинство МК семейства AVR имеют не менее 512 байт такой памяти, а младшие модели семейства Tuny — 64—128 байт. Для подавляющего большинства применений этого более чем достаточно. Не задерживаясь сейчас на вопросе, как осуществлять перезапись констант в процессе эксплуатации (этому вопросу будет посвящена *глава 16*), рассмотрим детали обращения с EEPROM.

Сохранность данных в EEPROM

Как мы уже говорили (см. *главу 11*), EEPROM и flash-память программ принципиально не отличаются, и предназначены для хранения данных в отсутствие питания. Однако между ними есть кардинальное различие: EEPROM может быть перезаписана в любой момент программой самого МК. В этом слабость всей системы: при снижении питания ниже определенных величин МК начинает совершать непредсказуемые операции, и EEPROM с большой вероятностью может быть повреждена. Для защиты от этой «напасти» (и вообще от выполнения каких-то операций, которые иногда могут навредить внешним устройствам) в AVR предусмотрена система BOD (см. *главу 13*), которая при снижении напряжения питания ниже определенного порога (4 или 2,7 В) «загоняет» МК в состояние сброса. Это помогает, но, как показывает опыт, для абсолютной защищенности данных в EEPROM, к сожалению, встроенной системы BOD недостаточно. Возможно, она недостаточно быстродействующая или в ней не слишком надежно фиксируется момент срабатывания, но факты свидетельствуют, что даже при включенной BOD данные все же могут быть повреждены.

Не исключено, что система BOD все время совершенствуется, но автор предпочитает не экспериментировать и использует самый надежный и проверенный способ с внешним монитором питания. Это небольшая микросхема (как правило, трехвыводная), которая при снижении питания ниже допустимого закорачивает свой выход на «землю». Если питание в пределах нормы, то выход находится в состоянии «разрыва» и никак не влияет на работу схемы. Присоединив этот выход к выводу Reset, мы получаем надежный предохранитель (рис. 15.5).

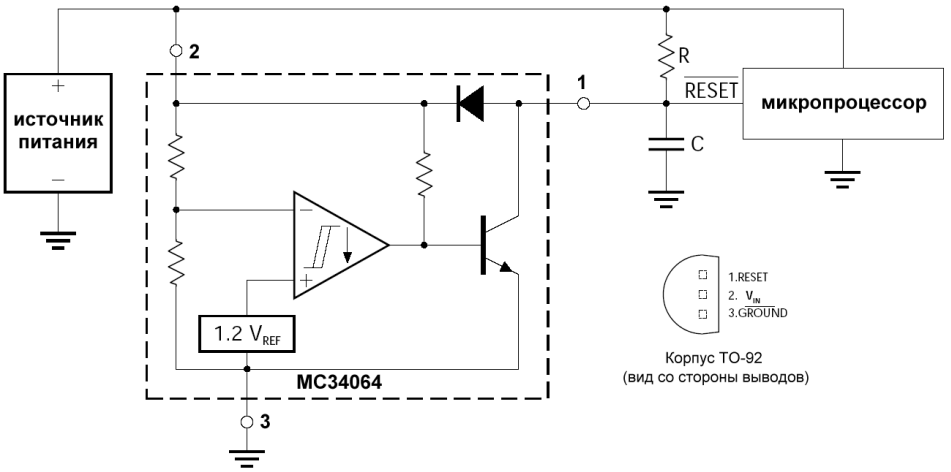


Рис. 15.5. Подсоединение внешнего монитора питания MC34064 к МК (схема из руководства фирмы Motorola)

Для напряжений питания 5 В один из самых популярных мониторов питания — микросхема MC34064, которая имеет встроенный порог срабатывания 4,6 В, выпускается в корпусе TO-92 с гибкими выводами и обладает достаточно малым собственным потреблением (менее 0,5 мА). Время срабатывания у нее составляет при плавном снижении напряжения порядка 200 нс, что достаточно для предотвращения выполнения «вредных» команд.

Если у вас питание автономное (от батарей), то к выбору монитора питания следует подходить довольно тщательно — так, чтобы не приводить схему в неработоспособное состояние тогда, когда батареи еще не исчерпали свой ресурс. При напряжении питания схемы 3,3 В пригодны приборы DS1816-10, MAX809S, при напряжении питания 3,0 В — DS1816-20 или MAX803R, а также некоторые другие.

Отметим, что рекомендуемый в фирменных описаниях способ защиты EEPROM вводом МК в состояние пониженного энергопотребления (см. *главу 17*) довольно сложно осуществить на практике. Если же вы все же умудритесь его задействовать, то следует учитывать, особенно в случае батарейного питания, что при резком снижении потребления напряжение источника немедленно повысится, что при относительно малом значении гистерезиса монитора питания (для MC34064 — 20 мВ) обязательно вызовет «дребезг» схемы. Увеличить гистерезис можно включением еще нескольких резисторов, но лучше обойтись вводом в режим сброса, как более простым и надежным способом.

Запись и чтение EEPROM

Запись и чтение данных в EEPROM осуществляется через специальные регистры ввода/вывода: регистр данных EEDR, регистр адреса EEAR (если объем EEPROM более 256 байт, то он делится на два — EEARH и EEARL) и регистр управления EECR. Основная особенность этого процесса — медленность процедуры записи, которая для разных моделей AVR может длиться от 2 до 9 мс, в тысячи раз дольше, чем выполнение обычных команд (обратим внимание, что в отличие от записи чтение осуществляется всего за один машинный цикл, даже быстрее, чем из обычной SRAM).

Для удобства проведения всех подобных процедур, которые могут длиться достаточно долго, в AVR предусмотрено соответствующее прерывание. В данном случае прерывание EEPROM может генерироваться по окончании очередного цикла записи, когда память свободна. Использовать его удобно, если нам требуется производить запись достаточно больших массивов: например, для 100 байтов запись может длиться почти секунду, и тормозить МК на весь этот период было бы неразумно. Тогда основная схема действий будет такой: разрешить прерывание EEPROM, и «внутри него» произвести запись очередного байта. Когда массив заканчивается, прерывания EEPROM запрещаются.

Такой метод можно назвать «правильным», но он заметно сложнее простого «лобового» метода, рекомендуемого, кстати, и в фирменном описании. Простой метод состоит в том, что мы запускаем бесконечный цикл ожидания, пока EEPROM освободится, и только тогда выполняем запись (или чтение) данных. В этом случае, если нам нужно записать всего один байт, МК вообще не будет затормаживаться (перед первой записью память свободна), и лишь при записи нескольких байтов подряд будет возникать упомянутая задержка. Факт задержки стоит учесть на будущее, когда нам придется стыковать запись в EEPROM с процедурами приема данных из последовательного порта, а во всех остальных ситуациях это практически не играет никакого значения: как вы увидите, в простейшем случае запись в EEPROM в процессе эксплуатации нам вообще не потребуется.

Процедура записи в EEPROM, которую мы будем использовать (листинг 15.7), ничем не отличается от приводимой в фирменных описаниях контроллеров, и я ее привожу в удобных для нас обозначениях регистров.

Листинг 15.7

```
WriteEEP: ;в ZH:ZL – адрес EEPROM куда писать
;в temp – записываемый байт
     sbic EECR,EEWE ; ждем очистки бита
```

```
     rjmp WriteEEP ;разрешения записи EEWE
     out EEARH,ZH ;старший адреса
     out EEARL,ZL ;младший адреса
     out EEDR,temp ;данные
       sbi EECR,EEMWE ;установить флаг разрешения записи
       sbi EECR,EEWE ;установить бит разрешения
ret ;(конец WriteEEP)
```

Установленный нами бит разрешения EEWE в регистре управления сбросится автоматически, когда запись закончится — этого сброса мы и ожидаем в начале процедуры. Естественно, в самый первый раз никакого ожидания на самом деле не потребуется. На всякий случай то же самое рекомендуется делать и при чтении, но практически всегда (если только мы не читаем непосредственно после записи), это не будет задерживать программу дольше, чем на время выполнения команды sbic, т. е. на два машинных цикла. Так как при чтении не требуется устанавливать никаких флагов, то процедура получается несколько короче (листинг 15.8).

Листинг 15.8

```
ReadEEP: ;в ZH:ZL — адрес откуда читать
;возврат temp — прочтенный байт
     sbic EECR,EEWE ;ожидание очистки флага записи
     rjmp ReadEEP
     out EEARH,ZH ;старший адреса
     out EEARL,ZL ;младший адреса
     sbi EECR,EERE     ;бит чтения
     in temp,EEDR ;чтение
ret ;конец ReadEEP
```

В этих процедурах регистр Z не играет никакой выделенной роли, а просто выбран в качестве удобной пары регистров, и может быть заменен на любую другую пару. Отметим еще, что на время записи следует запрещать прерывания, однако в наших программах далее это будет обеспечиваться автоматически.

Первичная запись констант в EEPROM

В принципе можно избежать процедуры записи вообще, если просто записать в EEPROM необходимые константы в процессе программирования. Это нужно делать отдельно от записи программы во Flash, с помощью специально подготовленного hex-файла. Но это ничем не будет отличаться от ситуации,

когда константы хранятся в тексте программы, только программировать МК придется значительно дольше, особенно при отладке. Гораздо грамотнее будет не пожалеть труда и составить программу так, чтобы она сама записывала нужные константы «по умолчанию». Как это правильно сделать?

Разумеется, это следует сделать при запуске МК, в процедуре Reset. Но записывать константы каждый раз при включении питания не только не имеет смысла (тогда проще их опять же хранить в тексте), но и еще более неудобно для пользователя, чем установка часов, о которой шла речь в *главе 14* — в дальнейшем мы научимся отдельно от программы записывать коэффициенты, не меняя текст программы, и хочется, чтобы это не требовалось делать после каждого сбоя питания. Тогда при удаче (если схема спроектирована верно и EEPROM надежно защищена от сбоев) автоматическая запись будет производиться один-единственный раз: при первом запуске контроллера, сразу после загрузки в его память программы, которую мы сейчас создадим.

Для этого нам потребуется как-то узнавать, есть ли уже в EEPROM какие-то данные, или нет, и правильно ли они записаны. Можно учесть тот факт, что в пустой EEPROM всегда записаны одни единицы (любой считанный байт будет равен $FF), но в общем случае это ненадежно. Наиболее универсальный способ — выделить для этого один какой-то байт в EEPROM, и всегда придавать ему определенное значение, а при загрузке МК его проверять. Это не гарантирует 100%-ной надежности при сбоях (т. к. данные в незащищенной EEPROM могут меняться произвольно, в том числе и с сохранением значения отдельных байтов), но мы будем считать, что от сбоев защищены «двойной броней» (из внешнего монитора питания и встроенной схемы BOD), и нам важно только распознать ситуацию, когда требуется первичная запись в еще не заполненную память. Приборы, которые я проектировал таким образом, работали, не выключаясь годами, без единого сбоя загруженных констант.

Итак, общая схема алгоритма такая: читаем контрольный байт из EEPROM, если он равен заданной величине (обычно я выбираю чередование единиц и нулей: $AA), то это значит, что коэффициенты уже записаны. Если же нет, то записывает значения «по умолчанию», в том числе и значение этого контрольного байта. Далее в любом случае переходим к процедуре чтения из EEPROM и перегрузки записанных констант в SRAM, откуда они при необходимости извлекаются точно так же, как ранее в процедурах расчета физических величин. Так мы сможем ничего не менять в основной программе, описанной ранее в этой главе, а лишь дописать некий текст в секции начальной загрузки.

Пусть значения коэффициентов записываются в EEPROM с самого начала (с адреса 0:0, в том же порядке, в котором они расположены в SRAM), а по адресу $10 записывается контрольный байт, равный $AA. Тогда в программе, приведенной в *Приложении 5*, в конце процедуры начальной загрузки по метке RESET вместо всего фрагмента, начинающегося с заголовка «запись коэффициентов» до команды sei (обязательно перед ней, а не после) добавляется текст листинга 15.9.

Листинг 15.9

```
;чтение коэффициентов из EEPROM ===========
     clr ZH ;ст. адрес =0
     ldi ZL,$10 ;адрес контрольного байта
     rcall ReadEEP
     cpi temp,$AA ;если он равен $AA
     breq mm_RK ;то на чтение в ОЗУ
rcall ZapisK ;иначе запись значений по умолчанию
mm_RK: ;извлечение коэфф. из EEPROM в SRAM
     clr ZL ;начальный адрес EEPROM 0:0
      ldi YL,tZH ;начальный адрес SRAM, см. основной текст
LoopRK:
     rcall ReadEEP ;читаем байт
     st Y+,temp ;складываем в ОЗУ
     inc ZL ;следующий адрес
     cpi ZL,8 ;всего 4 коэффициента, 8 байт
     brne LoopRK
```

Процедура записи коэффициентов по умолчанию, обозначенная как ZapisK (листинг 15.10), может быть вставлена в любом месте программы.

Листинг 15.10

```
ZapisK:
;запись предварительных коэффициентов по умолчанию
     clr ZH ;с нулевого адреса в EEPR
     clr ZL
; Z tempr=471
     ldi temp,High(471) ;ст.
     rcall WriteEEP
     inc ZL
     ldi temp,Low(471) ;мл.
     rcall WriteEEP
```

```
     inc ZL
; K tempr=1020
     ldi temp,High(1020) ;ст.
     rcall WriteEEP
     inc ZL
     ldi temp,Low(1020) ;мл.
     rcall WriteEEP
     inc ZL
;Z press=12
     ldi temp,0x00 ;ст.
     rcall WriteEEP
     inc ZL
     ldi temp,12 ;мл.
     rcall WriteEEP
     inc ZL
; K prs=916
     ldi temp,High(916) ;ст.
     rcall WriteEEP
     inc ZL
     ldi     temp,Low(916) ;мл.
     rcall WriteEEP

     ldi ZL,$10
     ldi temp,$AA ;все Ok, записываем
     rcall WriteEEP ;контрольный байт
ret
```

Манипулируя значением контрольного байта, можно даже определить, предварительные у нас коэффициенты записаны, или уже окончательные после калибровки, если вдруг возникает такая задача.

Конечно, иногда может понадобиться запись какой-то константы по ходу работы программы: например, если вы делаете электронный регулятор уровня какой-то величины (громкости, освещения, яркости свечения), то будет очень правильно записывать текущее значение в EEPROM, чтобы при следующем включении восстанавливалось установленное состояние, и пользователю не приходилось бы делать регулировку заново. Только при этом следует учесть, что EEPROM все же не RAM, и запись в нее, во-первых, имеет ограниченное (хотя и большое — до 100 000) число циклов, во-вторых, протекает на много порядков медленнее, а в-третьих, ведет к повышенному расходу энергии. Потому использовать EEPROM как ОЗУ, конечно, не стоит.

Кроме записи констант, наиболее часто EEPROM служит для хранения, например, заводского номера и названия прибора, фамилии конструктора-программиста или названия фирмы-изготовителя, и всякой другой полезной информации (ср. данные, которые извлекает операционная система ПК при подсоединении устройства plug&play, например, через USB). Можно заполнять различные поля, вплоть до серийного номера, и вести базу выпущенных экземпляров. Несложно сделать и так, чтобы эта информация выдавалась «наверх» автоматически при подсоединении прибора к компьютеру с загруженной программой, и текущие значения параметров выводились в отдельном окне — тогда можно обойтись без громоздкой и «прожорливой» индикации и получить компактный компьютерный «прибамбас». Только вот как все эти данные извлекать и при необходимости изменять, не затрагивая самой программы? Для этого существуют последовательные интерфейсы, к рассмотрению которых мы сейчас и перейдем.

Глава 16

Некоторые последовательные интерфейсы МК

Звонок в офис интернет-провайдера:
— Алло! Это Интернет?
— Да, слушаем Вас!
— Соедините с www.yahoo.com.

smeshok.com

Все современные интерфейсы, предназначенные для обмена данными между некоторыми устройствами (USB, FireWare, Serial ATA, Ethernet и т. п.), — последовательные. Исключение составляют до сих пор распространенные IDE (ATA)-интерфейсы жестких дисков и совершенно уже устаревший, но еще «живой» интерфейс LPT, который когда-то использовался не только для подсоединения принтеров, но даже цифровых камер и сканеров.

Почему так? Казалось бы, преимущество параллельной передачи данных перед последовательной видно невооруженным взглядом — в то время как по одному проводу за такт передается всего один бит, по восьми проводам — сразу целый байт. Однако такое естественное представление справедливо только для относительно небольших скоростей обмена. Когда речь заходит о скоростях, превышающих единицы Мбайт/с (десятки Мбит/с), преимущества параллельной передачи становятся вовсе не столь однозначными. Ведь в параллельной линии отдельные проводники всегда немного разные, отчего при увеличении длины кабеля и скорости передачи биты, передаваемые по разным проводам, начинают «разъезжаться» по времени: одни приходят чуть раньше, другие чуть позднее. По научному это называется *фазовым сдвигом*. Этот самый сдвиг сказывается при достаточно высоких скоростях уже на очень небольших расстояниях, например, при стандартной ныне тактовой частоте системной шины ПК 533 МГц (и тем более при 1066 МГц), материнскую плату приходится проектировать так, чтобы проводники, связывающие процессор и память, были строго параллельными и имели одинаковую длину. Учитывая, что число одних только линий данных доходит до 128, можно себе

представить, какая головоломная задача встает перед конструкторами. Несравненно проще повышать частоту последовательного канала, ведь там за каждый такт передается всего один бит, и сам такт мы теоретически можем сделать сколь угодно коротким, т. к. все зависит только от быстродействия оборудования. Оказывается выгоднее заложить максимум функциональности в микросхемы, чем иметь дело с толстенными «шлангами» с сотней проводов внутри.

В микроконтроллерных устройствах с нашими объемами данных, конечно, скорость передачи нас волнует в последнюю очередь, но вот количество соединительных проводов — очень критичный фактор. Поэтому все внешние устройства, с которыми мы будем иметь дело в этой книге, будут иметь последовательные интерфейсы. Из всех доступных мы рассмотрим два, которых достаточно для удовлетворения первостепенных нужд: это классический интерфейс UART и универсальный I^2C.

UART и RS-232

Сначала разберемся в терминах, которые имеют отношение к предмету разговора. В компьютерах есть COM-порт (в противном случае его всегда можно эмулировать через USB, как мы увидим в *главе 18*), часто ошибочно называемый портом RS-232. Правильно сказать так: COM-порт передает данные, основываясь на стандарте последовательного интерфейса RS-232. Последний, кроме собственно протокола передачи, стандартизирует также и электрические параметры и даже всем знакомые разъемы DB-9 и DB-25. UART (Universal Asynchronous Receiver-Transmitter, «универсальный асинхронный приемопередатчик») есть основная часть любого устройства, поддерживающего RS-232, но и не только его (недаром он «универсальный»), например, стандарты RS-485 и RS-422 также реализовываются через UART. Мы будем здесь рассматривать только RS-232, как самый простой.

Кроме UART, в состав RS-232 (в том числе в COM-порт ПК) входит схема преобразования логических уровней в уровни RS-232, где биты передаются разнополярными уровнями напряжения, притом инвертированными относительно UART. В UART действует положительная логика с обычными логическими уровнями, где логической единице соответствует высокий уровень (+3 или +5 В), а логическому нулю — низкий (0 В). У RS-232 наоборот, логическая единица есть отрицательный уровень от –3 до –12 В, а логический ноль — положительный уровень от +3 до +12 В. Преобразователь уровня в МК, естественно, не входит, так что для состыковки с компьютером придется его «изобретать». Этот вопрос мы в подробностях рассмотрим в *главе 18*, поэтому схемы в этой главе будут лишены узла сопряжения с ПК.

Для того чтобы доделать устройства до конца, вам придется еще выбрать наиболее подходящую для ваших условий схему сопряжения из указанных в *главе 18*.

Перед тем как обсуждать UART, рассмотрим подробнее, как, собственно, происходит обмен. Стандарт RS-232 — один из самых первых протоколов передачи данных между устройствами, он был утвержден еще в 1969 году, и к компьютерам (тем более ПК) тогда еще не имел никакого отношения. Идея этого интерфейса заключается в передаче целого байта по одному проводу в виде последовательных импульсов, каждый из которых может быть «0» или «1». Если в определенные моменты времени считывать состояние линии, то можно восстановить то, что было послано.

Однако эта простая идея натыкается на определенные трудности. Для приемника и передатчика, связанных между собой тремя проводами («земля» и два сигнальных провода «туда» и «обратно»), приходится задавать скорость передачи и приема, которая должна быть одинакова для устройств на обеих концах линии. Эти скорости стандартизированы, и выбираются из ряда 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 128000, 256000 (более медленные скорости я опустил). Число это обозначает количество передаваемых/принимаемых бит в секунду. Отметим, что стандарт RS-232E устанавливает максимальную скорость передачи 115200, однако функции Windows позволяют установить и более высокую скорость. Но не все схемы преобразования уровней могут пропустить через себя такие сигналы, и это следует учитывать при проектировании.

Проблема состоит в том, что приемник и передатчик — это физически совершенно разные системы, и скорости эти для них не могут быть строго одинаковыми в принципе (из-за разброса параметров тактовых генераторов), и даже если их каким-то образом синхронизировать в начальный момент, то они в любом случае быстро «разъедутся».

Заметки на полях

Такая же идея лежит в основе всех последовательных интерфейсов, они различаются только способами синхронизации. Например, в интерфейсе SPI, в том числе в его варианте для программирования МК, синхронизирующие импульсы передаются по отдельной, специально выделенной линии. Это облегчает задачу синхронизации, но требует большего количества проводов (не менее четырех, включая «землю»). А модемы или, к примеру, устройства Ethernet могут работать вообще всего по двум проводам, благодаря довольно сложному протоколу. Интерфейсы UART и I^2C, которые мы будем изучать, требуют *трехпроводного* соединения (включая «землю»), однако различаются по назначению линий.

В RS-232 передача каждого байта всегда сопровождается начальным (стартовым) битом, который служит для синхронизации. После него идут восемь

(или девять — при проверке на четность) информационных битов, а затем стоповые биты, которых может быть один, два и более, но это уже не имеет принципиального значения (почему — мы сейчас увидим).

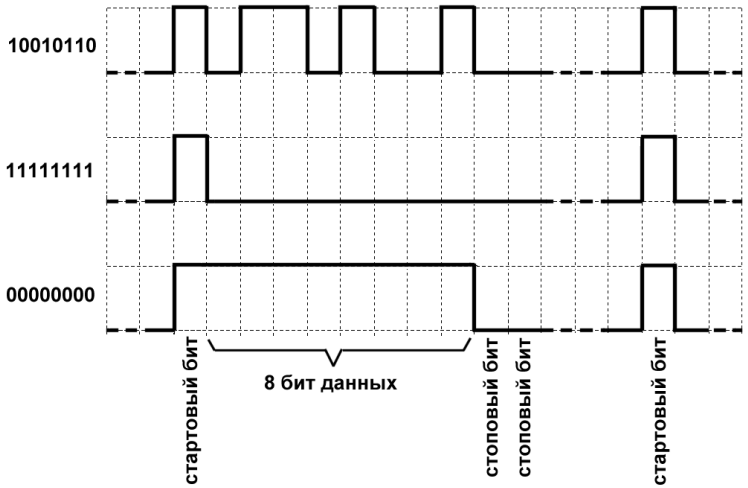


Рис. 16.1. Диаграмма передачи данных по последовательному интерфейсу RS-232 в формате 8n2

Общая диаграмма передачи таких последовательностей показана на рис. 16.1. Хитрость заключается в том, что состояния линии передачи, соответствующие стартовому и стоповому битам, имеют разные уровни: стартовый бит передается положительным уровнем напряжения (логическим нулем), а стоповый — отрицательным уровнем (логической единицей), потому фронт стартового бита всегда однозначно распознается.

Подробности

Подавляющую часть времени линия находится в состоянии ожидания, т. е. имеет уровень логической единицы (отрицательный — этим фактом мы воспользуемся в *главе 18* для создания простейшего преобразователя уровней.) Потому выбор уровней стартового и стопового бита не был полностью произвольным: такая комбинация уровней имеет большой смысл со схемотехнической точки зрения. Во-первых, в качестве преобразователя уровня удобно использовать транзистор, который инвертирует сигнал, и тогда специально об этом думать не приходится. Во-вторых, со стороны UART, где логика обратная, стоповый бит должен иметь высокий уровень, что соответствует состоянию запертого транзистора «с открытым коллектором» (или «с открытым истоком»). Так как бóльшую часть времени этот выходной транзистор оказывается выключен, то вывод не потребляет тока.

В момент передачи стартового бита и происходит синхронизация. Приемник отсчитывает время от фронта стартового бита, равное примерно половине периода заданной частоты обмена (чтобы попасть в середину стартового бита), и три раза подряд проверяет состояние линии (оно должно быть логическим нулем). Если все три состояния совпали, то принимается решение, что действительно пришел стартовый бит. Тогда восемь (или девять, если это задано заранее) раз подряд с заданным периодом регистрируется состояние линии (т. н. процедура *восстановления данных*). После этого линия переходит в состояние стопового бита и может в нем пребывать сколь угодно долго, пока не придет следующий стартовый бит.

Обычный формат данных, по которому работает львиная доля всех устройств, обозначается 8n1, что читается так: 8 информационных бит, no parity, 1 стоповый бит. «No parity» означает, что проверка на четность не выполняется. На диаграмме рис. 16.1 показана передача некоего произвольного кода, а также передача байтов, состоящих из всех единиц и из всех нулей в формате (для наглядности) 8n2. А в каком случае это важно, два стоповых бита передается или один (ведь, по сути, в состоянии линии при этом ничего не меняется)?

Задание минимального числа стоповых битов производится для того, чтобы приемник «знал», сколько времени минимально ему нужно ожидать следующего стартового бита (как минимум, это может быть, естественно, один период частоты обмена, т. е. один стоповый бит). Если по истечении этого времени стартовый бит не придет, приемник может регистрировать так называемый timeout, т. е. «перерыв», по-русски, и заняться своими делами. Мы никакими «тайм-аутами» себе голову заморачивать здесь не будем (передача одного байта с нашими скоростями будет занимать порядка миллисекунды, и за это время мы успеем перехватить данные даже без использования прерывания), и нам в принципе все равно, сколько стоповых битов будет. Но во избежание излишних сложностей следует их устанавливать у передатчика и у приемника всегда одинаково. Заметим, что если линия «зависнет» в состоянии логического нуля (высокого уровня напряжения), то это может восприниматься устройством, как состояние «обрыва» линии — не очень удобный механизм, и в микроконтроллерах он через UART не поддерживается.

Из описанного алгоритма работы понятно, что погрешность несовпадения скоростей обмена может быть такой, чтобы фронты не «разъезжались» за время передачи/приема всех десяти-двенадцати бит более чем на полпериода, т. е. в принципе фактическая разница частот тактовых импульсов может достигать 4—5%. На практике их стараются все же сделать как можно ближе к стандартным величинам, но это не всегда возможно.

Заметки на полях

А какова надежность при передаче таким способом? Приемник RS-232 дополнительно снабжают схемой, которая фиксирует уровень не один раз за период действия бита, а трижды, при этом за окончательный результат принимается уровень двух одинаковых из трех полученных состояний линии, таким образом удается избежать случайных помех. Дополнительная проверка целостности данных (контроль четности) и/или программные способы (вычисление контрольных сумм и т. п.) нам не потребуется, т. к. скорости обмена малы и ошибки маловероятны. Данные о разновидностях соединительных кабелей вы найдете в *главе 18*.

Для работы в обе стороны нужны две линии, которые у каждого приемопередатчика обозначаются RxD (приемная) и TxD (передающая). В каждый момент времени может работать только одна из линий, т. е. приемопередатчик либо передает, либо принимает данные, но не одновременно (это т. н. *полудуплексный* режим — так сделано потому, что у UART-микросхем традиционно один регистр и на прием, и на передачу).

Замечание

Для AVR на самом деле это не так — UART может одновременно принимать и передавать данные. Но адрес регистра данных для приема и передачи один и тот же, потому со стороны выглядит, как будто регистры приема и передачи есть один регистр. В самых первых микросхемах UART это действительно так и было.

Кроме RxD и TxD, в разъемах RS-232 присутствуют также и другие линии, о чем подробнее мы поговорим в *главе 18*. Отметим, что специально устанавливать состояния выводов порта (на вход или на выход), которые используются, как TxD и RxD, не требуется, как только вы «заведете» UART, они автоматически сконфигурируются, как надо. Только, в отличие от выводов программирования, их не рекомендуется задействовать еще для каких-то функций.

В AVR семейства Tuny (кроме модели 2313, которая все же, если позволительно так выразиться, «не совсем» Tuny) UART отсутствует, а в большинстве моделей семейства Mega этот порт реализован в виде более функционального USART («синхронно-асинхронного»), в некоторых моделях их даже несколько. USART полностью совместим с UART (кроме наименований некоторых регистров), и отличается от UART тем, что, во-первых, может самостоятельно обрабатывать девятибитовые посылки с контролем четности (не требуя программной реализации этого контроля), во-вторых, может иметь длину слова от 5 до 9 бит (UART только 8 или 9). Самое же главное его отличие (из-за которого он и получил свое название) в том, что его можно использовать в синхронном режиме, передавая по специальной дополнительной линии XCK тактовые импульсы (в результате чего USART почти перестает

отличаться от SPI, кроме того, что последний может работать значительно быстрее). Еще одна особенность USART — возможность работы в режиме мультипроцессорного обмена. Мы все эти режимы применять не будем, потому в дальнейшем будем вести речь только о UART, т. е. о работе в асинхронном режиме.

Прием и передача данных через UART

Перед работой с UART его следует установить в нужный режим, а также задать скорость обмена. Делается это, например, таким образом:

```
;для семейства Classic при частоте 4 МГц
     ldi    temp,25 ;скорость передачи 9600 при 4 МГц
     out    UBRR,temp ;устанавливаем
     ldi    temp,(1<<RXEN|1<<TXEN|1<<RXB8|1<<TXB8)
     out    UCR,temp ;разрешение приема/передачи 8 бит
```

Число BAUD для делителя частоты (в данном случае 25) можно определить из таблиц, которые имеются в каждом описании соответствующего контроллера (там же приводится и ошибка для выбранного значения частоты), или рассчитать по формуле: BAUD = $f_{рез}$/16(UBRR+1). Для семейства Mega процедура несколько усложняется, потому что регистров в USART больше:

```
;для семейства Mega при частоте 16 МГц
     ldi temp,103 ;9600 при 16 МГц
     out UBRRL, temp
     ldi temp,(1<<RXEN)|(1<<TXEN) ;разрешение приема/передачи
     out UCSRB,temp
     ldi temp, (1<<URSEL)|(3<<UCSZ0) ;формат 8n1
     out UCSRC,temp
```

Чем выше тактовая частота МК $f_{рез}$, тем точнее может быть установлена скорость. При частоте кварца 4 МГц мы с приемлемой точностью можем получить скорости обмена не более 28 800 бод. Правда, при выборе специального кварца (например, 3,6864 МГц) можно получить с нулевой ошибкой весь набор скоростей вплоть до 115 200, но зато для других целей такие частоты неудобны. Для получения скоростей передачи выше указанных (стандартно COM-порт позволяет установить скорости, как указано ранее до 256 кбод) придется увеличивать частоту. Так, при кварце 8 МГц и общем коэффициенте деления, равном единице, мы получим скорость 250 000, что отличается от стандартных 256 000 на приемлемые 2,4%.

Теперь перейдем к собственно процессу приему/передачи данных. В общем случае наша задача формулируется так: контроллер все время ожидает команды от компьютера, и при получении ее должен послать в ответ несколько байтов. Такой процесс можно организовать различным образом.

С UART связаны прерывания, причем в силу важности предмета тут их аж целых три: «передача завершена» (TX Complete), «регистр передатчика пуст» (TX UDR Empty) и «прием завершен» (RX Complete). Для их использования можно поступить следующим образом (примеры приведены для семейства Classic). Сначала вы инициализируете прерывание «прием закончен» (для чего надо установить бит RXCIE в регистре UCR). Возникновение этого прерывания означает, что в регистре данных UDR имеется принятый байт. Процедуру обработки этого прерывания иллюстрирует листинг 16.1.

Листинг 16.1

```
UART_RXC:
  in temp,UDR ;принятый байт — в переменной temp
  cbi UCR,RXCIE ;запрещаем прерывание «прием закончен»
. . . . .
       <анализируем команду, если это не та команда — опять разреша-
       ем прерывание «прием закончен» и выходим из процедуры
  sbi UCR,RXCIE
  reti
       В противном случае готовим данные, самый первый посылаемый
       байт должен быть в переменной temp>
. . . . .
  sbi UCR,UDRIE ;разрешение прерывания «регистр данных пуст»
reti
```

Далее у нас почти немедленно возникает прерывание «регистр данных пуст». Обработчик этого прерывания состоит в том, что мы посылаем байт, содержащийся в переменной temp, и готовим данные для следующей посылки (листинг 16.2).

Листинг 16.2

```
UART_DRE:
  out UDR,temp ;посылаем байт
  cbi UCR,UDRIE ;запрещаем прерывание «регистр данных пуст»
. . . . .
       <готовим данные, следующий байт — в temp. Если же был отправ-
       лен последний нужный байт, то опять разрешаем прерывание
       «прием закончен» и далее выходим из процедуры, иначе выполня-
       ем следующий оператор:>
  sbi  UCR,UDRIE ;разрешаем прерывание «регистр данных пуст»
reti
```

Для семейства Mega (USART) вместо UCR в текст примеров надо подставить UCSRB. Обратим внимание на то, что после обработки первого прерывания переменная temp здесь может содержать подготовленный для отправки байт, и не должна в промежутках между прерываниями использоваться еще где-то. В противном случае ее надо сохранять, например, в стеке, или все же отвести для этого дела специальный регистр. Как видите, все довольно сложно.

Однако, как и в случае записи в EEPROM (см. *главу 15*), поскольку эти события (прием и передача) происходят относительно редко, на практике мы не будем использовать прерывания, и процедуры резко упростятся (на примере USART — листинг 16.3).

Листинг 16.3

```
Out_com:  ;посылка байта из temp с ожиданием готовности
      sbis   UCSRA,UDRE ;ждем готовности буфера передатчика
      rjmp   out_com
      out   UDR,temp ;собственно посылка байта
ret ;возврат из процедуры Out_com
In_com: ;прием байта в temp с ожиданием готовности
      sbis   UCSRA,RXC ;ждем готовности буфера приемника
      rjmp   in_com
      in     temp,UDR ;собственно прием байта
ret ;возврат из процедуры In_com
```

Для семейства Classic надо заменить все UCSRA на USR. Для сформулированной ранее задачи непрерывного ожидания внешних команд обращение к процедуре In_com при этом вставляется в пустой цикл в конце программы:

```
Cykle:
      rcall  In_com
. . . . .
       <анализируем полученный в temp байт, и что-то с ним делаем,
например, посылаем ответ через процедуру Out_com>
. . . . .
rjmp   Cykle ;зацикливаем программу
```

При таком способе контроллер большую часть времени ожидает приема, непрерывно выполняя проверку бита RXC (в процедуре In_com), этот процесс прерывается только на время выполнения «настоящих» прерываний. Прерывания все равно должны выполняться много быстрее, чем байт, в случае, если он пришел, успевает в UDR смениться следующим (пауза составляет около 1 мс при скорости 9600, и за это время успеет выполниться порядка нескольких тысяч команд), так что мы ничего не потеряем. А процедура посылки

Out_com сама по себе может выполняться долго (как и в случае с записью EEPROM, кроме самого первого обращения: задержка будет, если посылать несколько байт подряд). Но для программиста процедура также в основном будет заключаться в том, что контроллер будет ожидать очистки UDR, и т. к. это не прерывание, то ожидание в любой момент может быть прервано реальным прерыванием, и мы ничего не теряем (даже если длительность прерывания превысит время посылки байта, то это лишь вызовет небольшую паузу в передаче).

Но чтобы ничего действительно не потерять, при таком способе следует быть внимательным: так, нужно следить за использованием temp внутри возникшего прерывания, а лучше на момент посылки данных вообще прерывания запретить. Правда, если мы будем применять процедуру Out_com внутри процедуры прерывания, куда другое прерывание «влезть» не может, то temp меняться заведомо не будет, но тогда при посылке нескольких байтов контроллер будет терять значительное время на ожидание, и это может нарушить работу других прерываний. Если это критично, то следует перейти к более сложной процедуре с использованием прерываний UART.

В общем и целом все эти нюансы следует иметь в виду, но на практике они почти не доставляют сложностей, за исключением одного момента, который мы еще обсудим в *главе 17*: если вам необходим переход в режим энергосбережения, то его объявление останавливает МК немедленно, как только встретится соответствующая команда. Если при этом в регистре данных UART оставался недоотправленный байт, то он так и не будет отправлен. Простыми задержками (например, выполнением пустого цикла) перед остановкой МК с этим явлением бороться неудобно (как мы говорили, нужно выполнить несколько тысяч команд). Лучше всего в таких случаях дождаться момента, когда регистр передатчика вновь окажется пуст (выполнением того же цикла непрерывной проверки UDRE, как в процедуре Out_com), и только тогда переходить к объявлению режима энергосбережения.

Отладка программ с помощью UART

Эти процедуры помогут мне выполнить давнее обещание и рассказать о том, как любую схему превратить в отладочный стенд. Вы просто временно расставляете в нужных местах программы контрольные точки в виде пар операторов:

```
move temp,RegX
rcall Out_com
```

Здесь RegX — регистр, значение которого хочется отследить в реальном времени. Если это регистр ввода/вывода, то вместо move надо использовать

инструкцию `in`. Подсоединив схему к компьютеру (см. *главу 18*), вы будете получать на ПК значения требуемого регистра при каждом прохождении программой этой контрольной точки. Иногда это может нарушить нормальную работу программы, как мы говорили ранее, но даже с учетом этого обстоятельства такой способ много нагляднее, быстрее и дешевле, чем использование дорогих отладочных модулей в совокупности с AVR Studio.

Если вы исследуете программу, в которой работа с UART не предусмотрена, то ничего не стоит вставить его инициализацию туда временно, и также временно вывести проводочками выводы RxD и TxD на небольшой отладочный стендик, состоящий из одного-единственного преобразователя уровней UART/RS-232. Единственное неудобство — при перестановке контрольных точек программу придется каждый раз перекомпилировать и заново записывать ее в МК, но это все равно потребуется при ее правке. По этим причинам я стараюсь иметь компьютеры с двумя COM-портами: к одному из них подключается программатор, к другому — выход схемы. Если у вас есть редактор текста, позволяющий запускать компиляцию прямо из него, как описывалось в *главе 13*, то процесс отладки микропрограммы становится не сложнее, чем работа в среде Turbo Pascal, Delphi или Visual Basic. Правда, многое еще зависит от удобства программы, которая принимает данные в ПК. Этот вопрос мы также обсудим в *главе 18*.

Запись констант через UART

Научившись таким образом принимать и передавать данные через UART, мы можем внести изменения в нашу программу измерителя с тем, чтобы загружать коэффициенты в EEPROM без перепрограммирования системы. Для начала придется изменить схему самого измерителя, добавив к ней модуль преобразователя UART/RS-232, который будет подсоединяться к выводам 14 (RxD) и 15 (TxD) контроллера ATmega8535 (см. рис. 15.2). Саму схему мы рисовать пока не будем, различные варианты ее построения мы подробно обсудим в *главе 18*.

Инициализацию UART удобно производить в той же секции начальной загрузки, например, сразу после инициализации таймеров. Вставьте туда фрагмент задания скорости и режима, приведенный ранее (естественно, в варианте для семейства Mega, но с коэффициентом 25, а не 103, как в примере, т. к. у нас кварц 4 МГц), потом процедуры `Out_com` и `In_com` (например, в начало текста, сразу после векторов прерываний), а затем вместо пустого цикла в конце программы впишите следующий код:

```
Gcykle:
     rcall in_com
```

```
    cpi temp,0xE0 ;записать коэффициенты + 8 байт
    breq proc_E0
    cpi temp,0xE2 ;читать коэффициенты 8 байт
    breq proc_E2
rjmp Gcykle
```

Работает этот кусок программы так, как описано ранее: программа в основном непрерывно опрашивает бит RXC, «отвлекаясь» только на выполнение настоящего прерывания (Timer 0 в данном случае, см. *главу 15*). И только если через UART был принят какой-то байт (он окажется в temp), программа переходит к его последовательной проверке на одно из заданных значений.

Заметки на полях

В оформлении процедуры заложен один потенциальный баг: между выполнением приема байта (процедура in_com) и его анализом (cpi temp...) может «вклиниться» прерывание, и содержимое temp будет, скорее всего, испорчено. Чтобы избежать этой ошибки, можно поступить двояко: либо запретить на время все прерывания (по крайней мере, пока идет анализ), либо каждый обработчик прерывания начинать с команды push temp и заканчивать командой pop temp (что в больших программах может быть довольно сложно осуществить на практике). Между тем, учитывая относительную редкость обращения через UART, вероятность такого совпадения чрезвычайно мала (в данном случае ее можно оценить, как отношение среднего времени выполнения цикла анализа к промежутку между прерываниями Timer 0, что составит величину порядка 0,1% — один шанс из 1000). И за все время работы по подобной схеме автор ни разу не получил такого сбоя. Потому я не стал усложнять программу, но, строго говоря, это неправильно, и грамотный преподаватель программирования обязательно сделал бы замечание. Чтобы соблюсти все правила, можно, например, в процедуру in_com вставить команду запрещения прерываний cli сразу по выходу из цикла опроса (перед оператором out UDR,temp), а команду разрешения прерываний sei — в текст основной программы сразу после метки Gcykle. Только в этом случае следует помнить, что использование процедуры in_com (например, для отладки) всегда должно сопровождаться командой sei, иначе МК просто «сдохнет» в какой-то момент (зависнет). Кроме того, прерывания будут тогда запрещены и при выполнении команд, поступающих с компьютера, а это в общем случае не всегда желательно. Чтобы освободить себя от подобных размышлений, я и отказался от этой возможности. В крайнем случае команда с компьютера пропадет, ничего страшного — то же самое может быть при простом сбое обмена. Тем не менее помнить о том, что подобные баги возможны, следует всегда, в других случаях это может оказаться очень критичным.

В данном случае мы договариваемся, что значение $E0 означает команду на перезапись коэффициентов в памяти, а значение $E2 — чтение ранее записанных значений. Естественно, в первом случае программа обязана ожидать «сверху» дополнительно еще 8 байт значений, а во втором — наоборот, прочесть записанные в EEPROM коэффициенты и выдать их «наверх». Если же принятый байт не равен ни одной из этих величин, то программа спокойно

возвращается по метке Gcykle и продолжает опрос бита RXC до следующего отправленного ей байта.

Для единообразия записи текста процедуры приема и отправки не вызываются напрямую, а дополнительно структурированы. Процедура приема организуется так:

```
proc_E0: ;записать коэффициенты +8 байт
     rcall WriteKoeff
rjmp Gcykle
```

Метка proc_E0, как и метка proc_E2 далее, должны располагаться сразу после основного цикла Gcykle (потому что команда rjmp имеет ограниченное пространство действия, см. *главу 13*). Далее, где-то (в конце программы, например) записываем, наконец, собственно процедуру WriteKoeff приема коэффициентов и записи их в память. В ней мы учтем, что коэффициенты нельзя писать сразу в EEPROM, так как запись байта длится дольше, чем его прием через UART, и во избежание их потери необходим некий буфер. Но нам и не нужно его специально изобретать, т. к. коэффициенты все равно дублируются в SRAM, куда мы их первоначально и запишем. Если бы мы этого не сделали, то пришлось бы перезапускать контроллер после записи[1]. Сказанное иллюстрирует листинг 16.4.

Листинг 16.4

```
WriteKoeff: ;записать коэффициенты +8 байт
cli ;запрещаем прерывания
       ldi ZH,1
       ldi ZL,tZH ;начальный адрес SRAM
LoopWR:
       rcall in_com ;принимаем следующий байт
       st Z+,temp ;сложили в SRAM
       cpi ZL,pKL+1 ;до адреса pKL ровно 8 байт,
                    ;см. листинг в Приложении 5
       brne LoopWR
;теперь коэффициенты находятся в SRAM, складываем в EEPROM
       clr ZH
       clr ZL ;адрес EEPROM = 0:0
       ldi YH,1
       ldi YL,tZH ; начальный адрес SRAM
```

[1] Кстати, у многих начинающих программистов возникает вопрос — а можно ли перезапустить контроллер программно? Для этого нужно воспользоваться сторожевым таймером — см. *главу 17*.

```
LoopWE:
       ld temp,Y+ ;забираем из SRAM
       rcall WriteEEP ;переписываем в EEPROM
       inc ZL ;следующий адрес
       cpi ZL,8
       brne LoopWE
       ldi temp,$AA ;все Ok, посылаем ответ
       rcall out_com
sei ;разрешаем прерывания
ret
```

Если все благополучно, по окончании процедуры в компьютер будет послан байт со значением $AA. Если такой байт не получен, значит, что-то, например, потерялось по дороге, или произошел еще какой-то сбой.

Процедура чтения коэффициентов вызывается так:

```
proc_E2: ;читать все коэффициенты 8 байт из EEPROM
       rcall ReadKoeff
rjmp Gcykle
```

А собственно процедура чтения (листинг 16.5) будет гораздо короче, т. к. не требуется спешить с приемом байтов и, соответственно, обращаться к SRAM (вообще-то нам безразлично, откуда получать коэффициенты, так что будем читать из оригинала — из EEPROM).

Листинг 16.5

```
ReadKoeff:  ;читать коэффициенты 8 байт из EEPROM
cli
       clr ZH
       clr ZL
LoopRE:
       rcall ReadEEP
       rcall out_com
       inc ZL
       cpi ZL,8 ;счетчик до 8
       brne LoopRE
sei
ret
```

Разобранный нами последовательный порт UART хорош своей изумительной простотой. UART в той или иной форме содержат практически все современные контроллеры, кроме самых простых, вроде семейства Tuny. Эта простота,

однако, оборачивается и некоторыми недостатками. Во-первых, UART может работать только с заранее оговоренной скоростью обмена. Это неудобно, когда вы заблаговременно не знаете характеристики линии: при соединении с компьютером на столе можно задавать скорость и 115 200, а при необходимости передачи по километровому кабелю и скорость 9600, которую мы тут выбрали, окажется чересчур высокой (подробнее об этом см. *главу 18*).

Если не брать во внимание способ синхронизации по отдельной линии, как в USART (лишний провод нам ни к чему), то в принципе можно придумать способ, когда устройства соединяются на самой быстрой скорости, на которой это возможно: так, например, работает модем, который со стороны компьютера есть тот же UART, отличающийся только формой передачи сигнала по линии (в нем единицы и нули передаются не уровнями напряжения, а посылками различной частоты или фазы), отчего может работать на значительно бóльших расстояниях. Не так уж сложно решить и задачу автоопределения на одной стороне скорости обмена, заданной на другой: нужно попробовать соединиться на различных скоростях, и когда заранее оговоренный байт (или их последовательность) будет принят верно, значит, мы достигли нужной скорости.

Можно ли, однако, решить задачу так, чтобы скорость передачи задавалась с одного конца, как в синхронном обмене, но при этом избежать лишнего провода? Оказывается, вполне возможно, если работать на небольших скоростях. Но UART имеет еще один капитальный недостаток: он предназначен только для соединения не более чем двух устройств между собой. Режим мультипроцессорного обмена USART, о котором мы упоминали, есть попытка решить эту проблему, но лучше выстроить сразу такой протокол, при котором несколько устройств могут быть соединены между собой, и без помех обмениваться данными в нужном направлении. Все последовательные интерфейсы, кроме «чистого» UART, построены именно таким образом, включая и SPI, и USB и многие другие. В том числе и протокол I^2C, который мы сейчас и разберем.

Последовательный интерфейс I^2C

Собственно термин I^2C принадлежит фирме Philips, которая придумала этот интерфейс, а в описаниях AVR «местный» вариант I^2C называют TWI (от two-wire, «двухпроводной»). Мы не будем вдаваться в тонкости различий этих протоколов, потому что они нам, по большому счету, безразличны — главное, что они полностью совместимы, и все внешние устройства, имеющие интерфейс I^2C, будут работать с AVR. Потому во избежание путаницы мы всегда будем употреблять более распространенный термин I^2C.

Этот интерфейс использует два сигнальных провода, как и UART (плюс, конечно, «землю», поэтому физически это трехпроводной интерфейс, а не двухпроводной, как его часто называют), только по одному из них (SCL) всегда передаются синхронизирующие импульсы, а собственно информация — по второму (SDA). Информация в каждый данный момент времени передается только одним устройством и только в одну сторону. С помощью I^2C может быть (теоретически) соединено до 128 устройств, так, как показано на рис. 16.2. «Подтягивающие» резисторы должны иметь номинал порядка единиц или десятков килоом (чем выше скорость передачи, тем меньше). В качестве их можно использовать встроенные резисторы выходных линий портов AVR, но автор не рекомендует это делать, поскольку их номинал слишком велик для обычных скоростей передачи (см. далее).

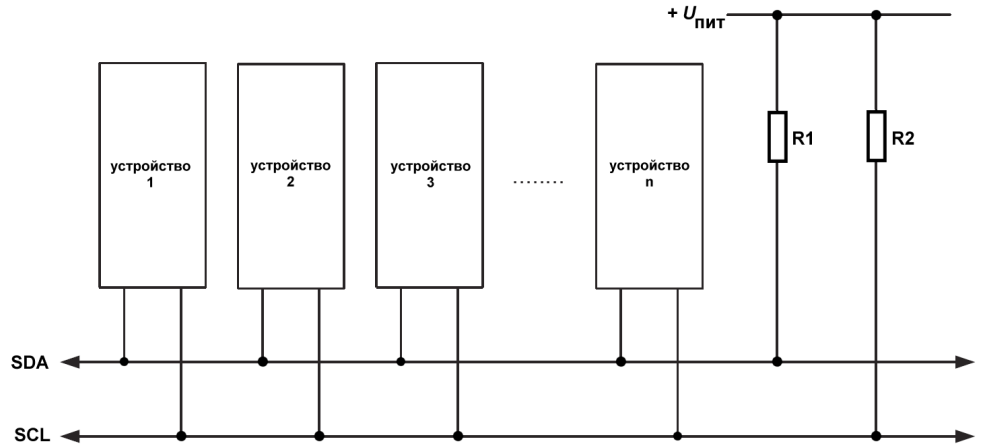


Рис. 16.2. Соединение устройств по интерфейсу I^2C (общий провод не показан)

Обратите внимание, что все устройства в этом случае обязаны иметь выход с «открытым коллектором», а привязка к шине питания обеспечивается парой внешних резисторов. Как мы знаем, выходы портов AVR построены иначе: по схеме с симметричным КМОП-выходом и третьим состоянием. Чтобы обеспечить совместимость с «открытым коллектором», здесь реализуют хитрый прием: состояние разрыва (выключенного транзистора на выходе) имитируется установкой выхода в третье состояние, т. е. фактически в режим вывода порта на вход, а включенное состояние — установкой вывода порта на выход и при этом обязательно в состояние логического нуля.

Разумеется, чтобы различить несколько устройств, каждое из них обязано иметь индивидуальный адрес. Он задается 7-битным кодом (восьмой бит байта

адреса служит для других целей, как мы увидим), потому-то всего таких устройств на одной линии может быть 128. Адрес этот часто задается еще изготовителем, хотя в самом AVR он может быть, разумеется, задан программно, но для наших целей это не потребуется, и вот почему.

Для того чтобы избежать конфликтов, во всех таких протоколах какое-то одно устройство может выставить себя главным (Master, ведущий). Оно может тогда инициализировать процесс обмена данными, выставляя на шину адрес того устройства, от которого желательно получить информацию — ведомого (Slave, что значит «слуга»). Все остальные устройства при этом «молчат», и таким образом налаживается двусторонний обмен. В принципе «мастером» может объявить себя любое устройство, но в простейшем случае это, естественно, микроконтроллер, а ведомыми при этом выступают другие микросхемы. При этом ведущему необязательно иметь свой собственный адрес, т. к. только он обращается к другим, к нему же не обращается никто, и выделять его с помощью специального адреса тогда не требуется.

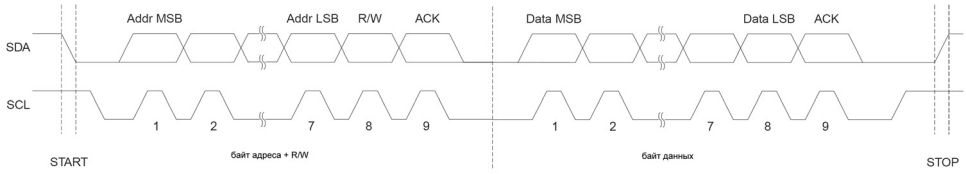


Рис. 16.3. Обмен информацией по интерфейсу I^2C

Типовой вариант обмена информацией по интерфейсу I^2C показан на рис. 16.3. Кратко расшифруем эту диаграмму. Любой сеанс передачи по протоколу I^2C начинается с состояния линии, именуемого Start (когда состояние линии SDA меняется с логической единицы на нуль при *высоком* уровне на линии SCL). Start может выдаваться неоднократно (тогда он называется «повторный старт»). Заканчивается сеанс сигналом Stop (состояние линии SDA меняется с логического нуля на единицу при высоком уровне на линии SCL). Между этими сигналами линия считается занятой, и только ведущий (тот, который выдал сигнал Start) может управлять ей (подробнее см. в [2]). Сама информация передается уровнями на линии SDA (в обычной положительной логике), причем смена состояний может происходить только при *низком* уровне на SCL, при высоком уровне на ней происходит считывание значения бита. Любая смена уровней SDA при высоком уровне SCL будет воспринята либо как Start, либо как Stop.

Процесс обмена всегда начинается с передачи ведущим байта, содержащего 7-битовый адрес (начиная со старшего разряда). Восьмой (младший!) бит

называется R/W и несет информацию о направлении обмена: если он равен «0», то далее ведущий будет передавать информацию (W), если равен «1» — читать (R), т. е. ожидать данные от ведомого. Все посылки (и адресные, и содержащие данные) всегда сопровождаются девятым битом, который носит название «бит квитирования». Во время действия этого девятого тактового импульса адресуемое устройство (т. е. ведомый, который имеет нужный адрес при передаче адреса, или ведущий, если данные направлены к нему, и т. п.) обязан сформировать ответ (ACK) низким уровнем на линии SDA. Если такого ответа нет (NACK), то можно считать, что данные не приняты, и фиксировать сбой на линии. Иногда устройства не требуют отсылки бита ACK, и это учтено в процедурах, которые рассмотрены далее.

Заметим, что сигналы SCL совершенно необязательно должны представлять собой равномерный меандр со скважностью 2 — период их следования в принципе ничем не ограничен, кроме «терпения» приемника, который, естественно, ждет сигнала какое-то ограниченное время (иначе при нарушении протокола программа может зависнуть). Более подробно мы разбирать протокол не будем, так как вы легко можете найти его изложение в описании любого устройства, которое этот протокол поддерживает (в том числе и в описаниях AVR, изложенных по-русски в книге [2]).

Как видим, организовать обмен по протоколу I^2C непросто, но это есть цена за универсальность и простоту электрической схемы. Большинство современных устройств с интерфейсом I^2C могут работать с тактовой частотой до 400 кГц, но в силу не слишком высокой помехоустойчивости такой линии максимальные частоты лучше использовать только тогда, когда микросхемы установлены на одной плате недалеко друг от друга. При соединении проводами (например, МК с каким-нибудь датчиком) лучше ограничиться частотами до 100 кГц, а при длинных линиях связи (провода в полметра длиной и более) частоту обмена стоит снижать до 10—30 кГц.

Организовать обмен по интерфейсу I^2C можно различными способами, и еще недавно это была исключительно программная эмуляция протокола. AVR семейства Mega (и только этого семейства) имеют I^2C (TWI), реализованный аппаратно. Реализация эта, впрочем, не очень удобна, потому что не избавляет от необходимости «ручного» отслеживания различных этапов обработки сигнала, в результате чего программа получается не менее громоздкой, чем при программной эмуляции. Еще один способ — использование прерывания, которое связано с TWI, тогда можно разгрузить контроллер от многочисленных задержек (передача одного байта длится примерно 0,1 мс). В дальнейшем, чтобы не распыляться, мы будем применять более универсальную программную эмуляцию, которая имеет и некоторые преимущества: позволяет

произвольно выбирать выводы для соединения (какие удобно, а не какие заданы аппаратной реализацией) и годится для абсолютно всех МК AVR, а не только для тех Mega, что имеют встроенный TWI. Единственное, чего мы лишимся — возможности «будить» контроллер, находящийся в «спящем» режиме (см. *главу 17*) обращением к нему через TWI, но нам это будет не нужно, т. к. контроллер всегда у нас находится в режиме ведущего. В фирменных Application Notes есть изложение процедуры программной эмуляции, но, как водится, с ошибкой в реализации.

Программная эмуляция протокола I²C

Наша задача будет формулироваться так: есть контроллер, и есть некое (одно или более) внешнее устройство. Нам надо прочесть/записать данные. Контроллер тут всегда будет выступать, как Master, а устройство — как Slave. Для того чтобы программно эмулировать протокол I²C, нам тогда придется сначала решить вопрос о том, как формировать тактирующую последовательность на линии SCL.

В принципе это можно сделать с помощью таймера, но на самом деле это неудобно: и таймеры обычно заняты более полезными делами, и нет тут у нас каких-то жестких требований ни к стабильности, ни к форме сигнала. Потому мы воспользуется древним, как сами микропроцессоры, способом формирования временной задержки с помощью пустого цикла. Так как время выполнения всех команд точно известно, то этот способ для формирования импульса определенной длительности ничуть не хуже любого другого, а когда в МК еще не было никаких таймеров, он вообще был единственным доступным способом для таких целей.

Для формирования импульса воспользуемся счетчиком `cnt` (пусть это будет регистр из числа старших — от `r16` до `r31`). Пустой цикл, повторенный `NN` раз, тогда запишется так:

```
       ldi cnt,NN
cyk_delay:  dec cnt
       brne cyk_delay
```

Посчитаем, чему должно равняться число NN. Пусть мы хотим обеспечить скорость передачи для I²C около 100 кГц, тогда длительность одного импульса (полпериода тактовой частоты) должна равняться примерно 5 мкс. Сам цикл занимает 3 такта (команда `dec` 1 такт + команда `brne` с переходом — 2 такта), т. е., например, при частоте кварцевого генератора 4 МГц он будет

длиться 0,75 мкс. Итого, чтобы получить при этой частоте импульс в 5 мкс, нам надо повторить цикл 6—7 раз. Точно подогнать частоту не удастся, но нам это, как мы говорили, и не требуется: опыт показывает, что при ошибке даже в два-три раза работоспособность I^2C практически не нарушается.

Чтобы не отводить отдельный регистр только для такой частной задачи, как счет циклов в задержке, стоит дополнить цикл процедурами сохранения в стеке значения счетчика, тогда этот регистр можно безопасно использовать где-то еще. И вся процедура будет выглядеть так:

```
delay: ;~5 мкс (кварц 4 МГц)
       push cnt
       ldi cnt,6
cyk_delay: dec cnt
       brne cyk_delay
       pop cnt
ret
```

Используя эту процедуру, можно сформировать весь протокол. Чтобы не загромождать текст этой главы, я вынес полный текст процедур обмена по I^2C в *Приложение 5* (раздел «Процедуры обмена по интерфейсу I^2C», листинг П5.3). Подробно расшифровывать я его не буду, т. к. он полностью соответствует описанию протокола.

Указанный в *Приложении 5* текст, кроме общих процедур посылки и приема байта (бесхитростно названных `write` и `read`), содержит процедуры для двух конкретных устройств: энергонезависимой памяти с интерфейсом I^2C (типа AT24) и часов реального времени (RTC) с таким же интерфейсом DS1307. Эти микросхемы имеют заданные I^2C-адреса — при записи $A0 (10100000) у памяти и $D0 (11010000) у часов (соответственно, $A1 и $D1 при чтении, подробности см. далее). Сейчас мы займемся проектированием устройства, использующего эти возможности.

Как и сказано в *Приложении 5*, текст приведенной в листинге П5.3 программы следует скопировать и сохранить в виде отдельного подключаемого файла. Мы будем предполагать, что такой файл называется i2c.prg. Директиву `.include "i2c.prg"` следует включать в текст программы обязательно *после* таблицы векторов прерываний, т. к., в отличие от файла макроопределений (m8535def.inc в данном случае), наш включаемый файл содержит команды, а не только инструкции компилятору. В принципе можно просто вставить текст из файла в основную программу (это и делает компилятор, когда встречает директиву `include`), только программа тогда станет совсем «нечитаемой».

Запись данных во внешнюю flash-память

Задача, которую мы сейчас будем решать, формулируется так: предположим, мы хотим, чтобы данные с нашего измерителя температуры и давления не терялись, а каждые три часа записывались в энергонезависимую память. Разумеется, встроенной памяти нам надолго не хватит, потому придется прибегнуть к внешней. Схему измерителя (см. рис. 15.2) придется минимально доработать — так, как показано на рис. 16.4.

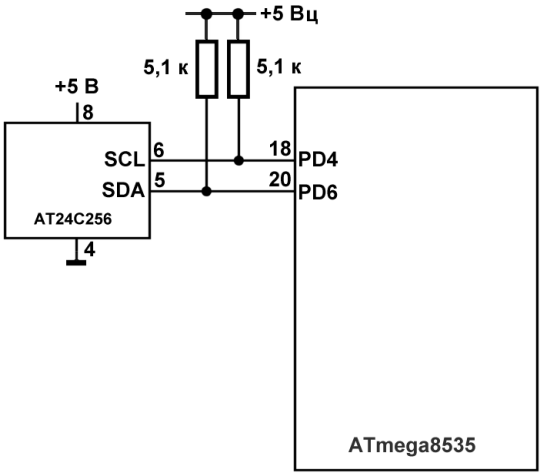


Рис. 16.4. Присоединение внешней EEPROM к измерителю температуры и давления

Здесь применяется энергонезависимая память типа AT24C256. Она имеет структуру EEPROM (т. е. с индивидуальной адресацией каждого байта), но чтобы отличить ее от встроенной EEPROM, в дальнейшем мы будем внешнюю память называть flash (хотя это и не совсем корректно). Последнее число в обозначении означает объем памяти в килобитах, в данном случае это 256 Кбит или 32 768 байтовых ячеек (32 кбайт). Объем памяти в 32 кбайт кажется смешным в сравнении с современными разновидностями flash, которые уже достигают объемов 8 Гбайт, но, во-первых, для наших целей, как вы увидите, этого будет достаточно. Во-вторых, память принципиально бóльших объемов с интерфейсом I^2C не выпускают — слишком он медленный.

Заметки на полях

Чтобы прочесть 32 кбайта со скоростями I^2C, потребуется примерно 0,5 мс на каждый байт, т. е. около 16 с. Потому максимальный объем памяти с таким интерфейсом фирмы Atmel, к примеру, составляет 1 Мбит (AT24C1024). Память с большими объемами представляет собой, во-первых, действительно flash-память (с блочным доступом), во-вторых, выпускается с интерфейсами побыстрее (как, к примеру, AT26DF321 объемом 4 Мбайта с 66-мегагерцевым интерфейсом SPI). Максимальный объем одного кристалла flash-памяти, достигнутый на момент написания этих строк — 8 Гбит (Samsung), более емкие устройства (flash-карты) представляют собой модули, собранные из нескольких подобных микросхем. Микросхемы с параллельным интерфейсом имеют стандартную разводку выводов и совместимы по выводам с любыми другими микросхемами памяти.

Кстати, найти в продаже микросхемы EEPROM с последовательным доступом (в том числе и использованную в нашей схеме AT24C256) в корпусе DIP довольно сложно. AT24C256 (как и упоминающаяся далее AT24C512 и другие) чаще встречается в миниатюрном корпусе SOIC с 8 выводами. Так как присоединять в простейшем случае приходится всего 4 вывода (2 питания и 2 сигнальных), то даже при ручной разводке это не доставляет больших сложностей.

Время чтения можно сократить, если попробовать «выжать» из I^2C все, на что он способен, но непринципиально. Да нам это и не требуется, потому что все равно эти данные мы будем передавать через UART примерно с такими же по порядку величины скоростями.

Адресация тут двухбайтовая, потому под адрес задействуется два регистра (`AddrL` и `AddrH`). Мы выбираем `r24` и `r25` (см. текст процедур в *Приложении 5*) — почему именно эти, вы увидите далее. Записываемые данные будут храниться в регистре `DATA`. Эти регистры являются входными переменными как для процедуры записи, так и чтения.

Теперь давайте определимся, что именно мы будем записывать, и на сколько нам хватит этой памяти. Базовый кадр данных у нас будет состоять из четырех байтов значений давления и температуры. Мы можем, конечно, писать и в распакованном BCD-виде, взяв подготовленные для индикации значения, но зачем загромождать память (кадр тогда состоял бы из 6 байтов, не четырех), если коэффициенты пересчета мы знаем (они у нас хранятся в EEPROM), и пересчитать всегда сможем. Если договориться на четырех байтах, то в наши 32 кбайта мы сможем вместить 8192 измерения (на самом деле чуть меньше, как мы увидим, но это несущественно), то есть при трехчасовом цикле (8 измерений в сутки) памяти нам хватит на 1024 суток, или почти на 3 года записей!

Как видите, даже такой объем вполне приемлемый. Если хотите увеличить еще в два раза — возьмите память AT24C512, ее можно поставить сюда без изменений в схеме (и в программе, кроме задания максимального адреса).

Схемотехника серии AT24 предполагает возможность установки параллельно четырех или восьми таких микросхем (с заданием индивидуального I^2C-адреса для каждой), так что при желании объем можно увеличить еще в четыре-восемь раз. Причем использовать, например, две AT24C512 целесообразнее, чем одну ATC1024, так как для последней адресация усложняется (адрес для объема 128 кбайт содержит 17 бит и выходит за рамки 2-байтового).

Подробности

Микросхемы серии AT24 имеют два (или три для микросхем с буквой B в конце обозначения, например, AT24C256B) специальных вывода A0 и A1 (выводы 1 и 2 для 8-выводных корпусов), которые задают индивидуальный I^2C-адрес. Если эти выводы ни к чему не подсоединять (как в нашей схеме), то считается, что они подсоединены к логическому нулю. Тогда I^2C-адрес микросхемы при записи будет 10100000 в двоичной форме или $A0 в шестнадцатеричной (см. листинг процедур I^2C в *Приложении 5*). Если на указанные выводы адреса подавать сигналы, то старшие 7 бит адреса такой микросхемы будут определяться формулой 10100A1A0. Таким образом, переходом от одной микросхемы к другой можно управлять, если подавать на эти выводы сигнал по дополнительным линиям, которые фактически будут 17-м и 18-м битами адреса.

Для того чтобы записывать исходные значения температуры и давления, нам их придется где-то хранить отдельно, отведя для этого специальные ячейки в SRAM. Сама запись производится очень просто: с каждым байтом мы увеличиваем на единицу содержимое счетчика адресов `AddrH:AddrL` (командой `adiw` — именно для этого и выбирались регистры `r24` и `r25`, чтобы ее можно было использовать), «забиваем» нужный байт в регистр `DATA`, и вызываем процедуру `WriteFlash`.

Но тут встает две проблемы. Прежде всего, нужно решить, что делать, когда память закончится. Тогда следует либо обнулять ячейки и начинать запись заново, поверх младших адресов, либо, что гораздо красивее, остановить запись, пока содержимое ее не будет прочитано и адрес принудительно не будет обнулен. Поэтому потребуется какой-то флаг, сигнализирующий о том, что настал конец памяти. Причем отвести для этого флага, например, бит в регистре `Flag`, будет недостаточно: а что будет при сбое питания? Нам придется хранить где-то во встроенной EEPROM и этот флаг и, главное, текущий адрес памяти, иначе данные будут пропадать после каждого отключения питания. А для прибора, который может писать три года подряд, это «несолидно».

А как отсчитывать время, когда производить запись? Для того чтобы метеоданные были полноценными, их нужно привязать ко времени. И тут мы неизбежно приходим к тому, чтобы объединить часы с нашим измерителем. Этим мы займемся чуть далее, потому что использовать сам контроллер в качестве часов, как мы это делали в *главе 14*, здесь нецелесообразно, слишком много он всего делает такого, что может вызвать сбой в отсчете времени.

Придется задействовать внешние часы, но подключение RTC заметно сложнее, чем памяти, потому мы рассмотрим этот вопрос позднее.

А пока, чтобы отработать процедуры обмена по I^2C, договоримся, что запись в память у нас будет производиться по прерыванию Timer 1, который больше все равно в измерителе ничем не занят. При 4 МГц тактовой частоты и максимально возможном коэффициенте ее деления 1024, можно заставить Timer 1 срабатывать каждые, например, 15 с, для чего в регистр сравнения придется записать число 58 594 (проверьте!). С такой частотой память, конечно, заполнится очень быстро (32 кбайта — менее чем за 1,5 суток), но это, наоборот, удобно, если стоит задача проверить все наши процедуры.

Итак, записываем в секции определений программы измерителя, там, где адреса SRAM:

```
. . . . .
;Hex-данные — «сырые», без пересчета
.equ Thex = 0x0A ;0A,0B — старший и младший байты температуры
.equ Phex = 0x0C ;0C,0D — старший и младший байты давления
.equ FEnRAM   = $0E ;флаг, если равен $FF, то писать во flash
. . . . .
```

Отдельно запишем адреса в EEPROM (первые восемь у нас заняты коэффициентами):

```
.equ   FEnEE  = 0x10 ;флаг если равен $FF, то писать во flash
.equ   EaddrL = 0x11 ;младший байт тек. адреса
.equ   EaddrH = 0x12 ;старший байт тек. адреса
```

Обратите внимание, что запись во flash разрешена, если байт `FEnEE` равен $FF, т. е. в самом начале, когда EEPROM еще пуста, запись по умолчанию разрешается. В процедуре обработки данных дописываем процедуры сохранения «сырых» значений температуры и давления по указанным адресам. Они у нас содержатся в регистрах `AregH:AregL`. В начале обработки данных по температуре, после имеющегося оператора `rjmp prs` дописываем:

```
      ldi ZL,Thex ;запоминаем температуру
      st Z+,AregH
      st Z,AregL
```

А там, где начинается расчет давления, после оператора `rjmp contPT` записываем:

```
      ldi ZL,Phex ;запоминаем давление
      st Z+,AregH
      st Z,AregL
```

Теперь инициализируем таймер. В загрузочную секцию вместо строк инициализации Timer 0 (`ldi temp,(1<<TOIE0)` и `out TIMSK,temp`) добавляем:

```
;++++++++Set Timer 1
      ldi temp,high(58594)
```

```
out OCR1AH,temp
ldi temp,low(58594)
out OCR1AL,temp
ldi temp,0b01000000
out TCCR1A,temp ;переключающий режим для вывода PD5-OC1A
ldi temp,0b00001101
out TCCR1B,temp ;1/1024 очистить после совпадения
ldi temp, (1<<TOIE0)|(1<<OCIE1A) ;разреш. прерывания
;по совпадению для Timer 1 и переполнению Timer 0
out TIMSK,temp
```

К выводу OC1A (вывод 19 для ATmega8535) можно присоединить светодиод, который будет попеременно гореть и гаснуть с периодом 30 с, показывая, что запись работает.

Далее в секции начальной загрузки инициализируем регистры адреса. Получится довольно сложная процедура (листинг 16.6), которая должна проверять значения адреса в EEPROM, и если он есть (т. е. память не пуста и там не записаны все единицы), то еще и сравнивать его с последним возможным адресом (32767 или 7FFFh).

Листинг 16.6

```
;=========инициализация адреса flash
     clr ZH ;старший EEPROM
     ldi ZL,EaddrL ;младший EEPROM
     rcall ReadEEP
     mov AddrL,temp
     ldi ZL,EaddrH
     rcall ReadEEP
     mov AddrH,temp ;теперь в AddrH:AddrL адрес из EEPROM
     ldi temp,0xFF ;если все FF, то память была пуста
     cp AddrL,temp
     ldi temp,0xFF
     cpc AddrH, temp
     brne cont_1
     clr AddrH ;если пуста, то присваиваем адрес = 0
     clr AddrL
     clr ZH ;старший EEPROM
     ldi ZL,EaddrL  ;младший EEPROM
     mov temp,AddrL
     rcall WriteEEP ;и записываем его опять в EEPROM
     inc ZL
     mov temp,AddrH
     rcall WriteEEP
```

```
cont_1: ;теперь проверку на последний адрес $7FFF
     ldi temp,0xFF
     cp AddrL,temp
     ldi temp,0x7F
     cpc AddrH, temp
     brne cont_2
     sbr Flag,4 ;4 бит регистра Flag = конец памяти
cont_2: ;загрузка байта разрешения записи flash
     clr ZH ;старший EEPROM
     ldi ZL,FEnEE
     rcall ReadEEP
     ldi ZH,1 ;старший RAM
     ldi ZL,FEnRAM ;младший RAM
     st Z,temp ;сохраняем значение флага
```

Отдельный бит «конец памяти» в регистре Flag (бит 2, т. е. устанавливается он командой `sbr Flag,4`, см. *главу 13*) нам понадобится позднее, для того, чтобы можно было временно запретить запись во flash внешней командой, не сбрасывая значения адреса и независимо от того, достигнут конец памяти или нет.

Теперь в секции прерываний заменим `reti` на `rjmp TIM1_COMPA` в строке для прерывания Timer1 Compare A (шестое сверху, не считая RESET), и напишем его обработчик (листинг 16.7).

Листинг 16.7

```
TIM1_COMPA: ;15 секунд
;проверять разрешение записи во flash
     ldi ZH,1 ;старший RAM
     ldi ZL,FEnRAM
     ld temp,Z
     cpi temp,$FF
     breq flag_WF
     reti ;если запрещено, то выходим из прерывания
flag_WF:
     ldi ZL,Thex ;адрес значения в SRAM
     ld DATA,Z+ ; старший T
     rcall WriteFlash ;пишем во flash
     adiw AddrL,1
     ld DATA,Z+   ;младший T
     rcall WriteFlash
     adiw AddrL,1
     ld DATA,Z+   ;старший P
```

```
     rcall WriteFlash
     adiw AddrL,1
     ld DATA,Z+   ;младший P
     rcall WriteFlash
;проверяем адрес на 7FFF
     ldi temp,0xFF
     cp AddrL,temp
     ldi temp,0x7F
     cpc AddrH, temp
     breq clr_FE если равен, на clr_FE
     adiw AddrL,1  ;иначе сохраняем след. адрес:
     clr ZH
     ldi ZL,EaddrL ;в EEPROM
     mov temp,AddrL
     rcall WriteEEP
     inc ZL
     mov temp,AddrH
     rcall WriteEEP
reti ;выход из прерывания
clr_FE ;если конец памяти:
     clr temp
     ldi ZH,1 ;старший RAM
     ldi ZL,FEnRAM
     st Z,temp ;сбрасываем разрешение записи
     clr ZH ;старший EEPR
     ldi ZL,FEnEE
     rcall WriteEEP
     sbr Flag,4 ;бит «конец памяти»
     clr temp
     out TCCR1A,temp ;отмена переключающего
                     ;режима для вывода PD5-OC1A
reti ; на выход из прерывания
```

Как мы видим, здесь каждые 15 с идет запись в EEPROM текущего адреса (того, по которому должна производиться следующая запись), т. е. если в какой-то момент питание пропадет, то при следующей загрузке запись все равно начнется с текущего адреса. При достижении конца памяти отключится переключающий режим для вывода OC1A, и светодиод перестанет мигать, сигнализируя о конце памяти.

Отметим, что запись идет через каждые четыре байта, т. е. для заполнения 32 кбайтов внешней памяти придется 8192 раза обновить содержимое ячеек. Таким образом, для достижения теоретического предела по количеству циклов записи в EEPROM (100 тыс.) нужно как минимум двенадцать раз заполнить внешнюю память. На самом же деле число допустимых циклов еще

намного больше (автор специально запускал запись каждые три секунды на пару месяцев, но сбоев так и не добился), потому можно не опасаться, что мы исчерпаем ресурс встроенной EEPROM.

> ***Заметки на полях***
>
> Обратите внимание, что в процедурах I²C имеются псевдонимы: DATA и ClkA есть те же регистры, что и temp1 и temp2 в основной программе. Я предостерегал вас от такой практики, но в данном случае ничего страшного не произойдет, т. к. использование этих регистров разнесено во времени — обращение к I²C никогда не сможет произойти во время расчетов, где задействованы temp1 и temp2. В дальнейшем эти регистры нам где-нибудь еще пригодятся.

Чтение данных из памяти через UART

Теперь займемся процедурами чтения содержимого внешней памяти и осуществления установок разрешения-запрещения записи. Естественно, это придется делать через компьютер (а куда еще читать?), и мы используем уже инициализированный нами UART. Нам потребуется реализовать четыре процедуры:

- Запретить запись во flash.
- Разрешить запись во flash.
- Прочесть содержимое flash.
- Обнулить адрес flash, чтобы начать запись с начала.

Добавим в текст программы, в основной цикл (по метке Gcykle, обязательно после команды rcall in_com) следующие строки:

```
. . . . . .
     cpi temp,0xF0 ;запись во flash разрешить
     breq proc_F0
     cpi temp,0xF1 ;запись во flash запретить
     breq proc_F1
     cpi temp,0xF2 ;читать flash
     breq proc_F2
     cpi temp,0xF8 ;запись во flash с начала, обнулить адрес
     breq proc_F8
. . . . . .
```

Посылая соответствующие команды ($F0 и т. д.) с компьютера, мы будем вызывать соответствующую процедуру. Проще всего оформить процедуры разрешения и запрещения так, как в листинге 16.8.

Листинг 16.8

```
proc_F0: ;F0 запись flash разрешить
     rcall EnableFlash
rjmp Gcykle
proc_F1: ;F1 запись flash запретить
     rcall DisFlash
rjmp Gcykle
. . . . .
EnableFlash:
cli
;сначала проверяем бит «конец памяти»
     sbrc Flag,2
     rjmp exit_FE
;проверяем байт разрешения, если нет, пишем его в память
     ldi ZH,1 ;старший RAM
     ldi ZL,FEnRAM
     ld temp,Z
     cpi temp,$FF
     breq exit_FA
     ldi temp,$FF
     st Z,temp
     clr ZH ;старший EEPR
     ldi ZL,FEnEE
     rcall WriteEEP
     ldi temp,$AA ;все Ok
     rcall out_com
sei
ret
exit_FA:
     ldi temp,$FA ;ответ в комп. — уже разрешен
     rcall out_com
sei
ret
exit_FE:
     ldi temp,$FE ;ответ в комп. — конец памяти
     rcall out_com
sei
ret

DisFlash:
cli
     clr temp
     ldi ZH,1 ;старший RAM
     ldi ZL,FEnRAM
```

```
     st Z,temp
     clr ZH ;старший EEPR
     ldi ZL,FEnEE
     rcall WriteEEP
     ldi temp,$AA ;ответ в комп. — все Ok
     rcall out_com
sei
ret
```

Мы используем запрещение прерываний, потому что процедуры достаточно долгие и запросто можно «испортить» temp по ходу дела, а здесь это уже недопустимо. Кроме того, они включают запись в EEPROM, во время которой прерывания надо все равно запрещать. Из этих процедур мы также видим, зачем нам понадобился отдельный флаг «конец памяти» — если он установлен, то разрешить запись будет нельзя. Это можно будет сделать только одновременно со сбросом адреса, что необратимо, и данные после этого уже прочесть будет нельзя. Потому мы сначала займемся их чтением (листинг 16.9).

Листинг 16.9

```
proc_F2: ;F2 читать flash
     rcall ReadFullFlash
rjmp Gcykle
. . . . .
ReadFullFlash:
cli
     mov YH,AddrH ;сохраняем текущий адрес в Y
     mov YL,AddrL
     clr AddrL ;чтение начнем с начала памяти
     clr AddrH
loopRF:
     cp AddrL,YL ;не дошли ли до текущего
     cpc AddrH,YH
     breq end_RF ;если дошли, то конец чтения
     rcall ReadFlash ;собственно чтение
     mov temp,DATA ;данные из DATA в temp
     rcall out_com ;передаем наружу
     adiw AddrL,1 ;следующий адрес
     rjmp loopRF
end_RF:
     mov AddrH,YH ; восстанавливаем текущий адрес
     mov AddrL,YL
sei
ret
```

Процедура эта будет долгой, если записан сколько-нибудь существенный кусок в памяти (для передачи 32 кбайт со скоростью 9600 потребуется порядка полминуты, да еще и чтение по I^2C), и на все это время прерывания будут запрещены. Для нашего измерителя это выльется только в исчезновение на это время индикации, но могут быть ситуации, когда следует предотвратить выключение контроллера на такое время — например, чтобы не потерять данные, когда настанет момент очередной записи. В дальнейшем мы учтем этот момент (хотя это, как вы увидите, сильно усложнит программу). А пока нам осталось только описать сброс адреса (листинг 16.10).

Листинг 16.10

```
proc_F8:;F8 clear address
     rcall ClearAddr
rjmp Gcykle
. . . . .
ClearAddr:
cli
     cbr Flag,4 ;обнуляем бит конец памяти
     clr AddrH ;обнуляем адрес
     clr AddrL
     clr ZH ;и записываем его в EEPROM
     ldi ZL,EaddrL
     mov temp,AddrL ;можно и просто clr temp
     rcall WriteEEP
     inc ZL
     mov temp,AddrH
     rcall WriteEEP
     ldi temp,$AA ;ответ в комп. все Ok
     rcall out_com
sei
ret
```

Теперь у нас есть измеритель температуры и давления, записывающий данные во внешнюю память, откуда они могут быть прочитаны в любой момент. В *главе 18* мы займемся вопросом приема данных через компьютер. А здесь нам осталось только объединить измеритель с часами, чтобы можно было задавать точный и достаточно длительный интервал измерения. И для этой цели нам также пригодится интерфейс I^2C.

Часы с интерфейсом I^2C

Моделей микросхем RTC, о которых мы говорили в начале *главы 14*, существует множество. Все они внутри устроены примерно одинаково, и имеют встроенный счет времени и календарь, а также функции будильника и/или

таймера. Подавляющее большинство RTC имеют возможность автономной работы от батарейки в течение длительного времени, без потери однажды установленного времени. Такие часы обычно снабжены кварцем на 32 768 Гц, иногда даже встроенным в микросхему. Кроме этого, значительная часть моделей имеет дополнительный выход (иногда и не один), на котором формируется некая частота, задаваемая программно. Этот выход можно использовать для управления прерыванием микроконтроллера, и таким образом организовать счет времени и его индикацию.

Еще одна особенность микросхем RTC — величины времени в них традиционно представлены в десятичном виде (т. е. в упакованном BCD-формате). Именно так выдаются значения времени в RTC, встроенных в ПК. Например, число минут, равное 59, так и выдается, как байт со значением 59, но, как мы уже говорили, это не $59, что в десятичной системе есть 89! Соответствующее шестнадцатеричное число записалось бы как $3B. BCD-представление удобно для непосредственной индикации, но при выполнении арифметических операций со временем (или, например, операций сравнения) его приходится преобразовывать к обычному двоичному виду. На самом деле это почти не доставляет неудобств, скорее наоборот.

Для наших целей выберем модель RTC под названием DS1307. Это простейшие часы с I^2C-интерфейсом, в 8-выводном корпусе с внешним резонатором на 32 768 Гц, 5-вольтовым питанием и возможностью подключения резервной батарейки на 3 В (т. е. обычной «таблетки»). Схема переключения питания на батарейку встроенная и не требует внешних элементов. Имеется вывод для прерывания МК, который может программироваться с различным коэффициентом деления частоты кварца. Мы его запрограммируем на выдачу импульсов с периодом 1 с, по внешнему прерыванию от этих импульсов в МК будем считать секунды, обновлять значение времени и производить всякие другие полезные действия — точно так же, как мы это делали в часах из *главы 14*, только там отсчет времени производился внутренним таймером. Но здесь мы можем быть уверены, что при любых сбоях в МК время у нас будет отсчитываться верно.

Схема подсоединения DS1307 к нашему измерителю приведена на рис. 16.5. Обратите внимание, что выводы интерфейса I^2C (5 и 6) здесь те же самые, что и для памяти. Выход программируемой частоты SQW у нас подсоединен к выводу внешнего прерывания МК. SQW мы должны запрограммировать на выдачу сигнала с периодом 1 с.

Основное неудобство обращения с часами DS1307 — отсутствие состояния «по умолчанию», поэтому внутренние регистры могут при включении питания иметь произвольные значения. В частности, в этих часах в одном из регистров (том же, что хранит значения секунд) предусмотрен бит CH, который

может погружать часы в «спячку» — если он установлен в единицу, то не работает генератор и даже невозможно определить правильность подключения. Есть и бит (в регистре управления), который отключает выход частоты на прерывания МК. По этим причинам после первого включения (если батарейка подсоединена — то только после первого), часы приходится инициализировать. Логика разработчиков проста: зачем кому-то нужны часы, которые не установлены на правильное время? Ну а если их устанавливать, то нетрудно и установить эти биты.

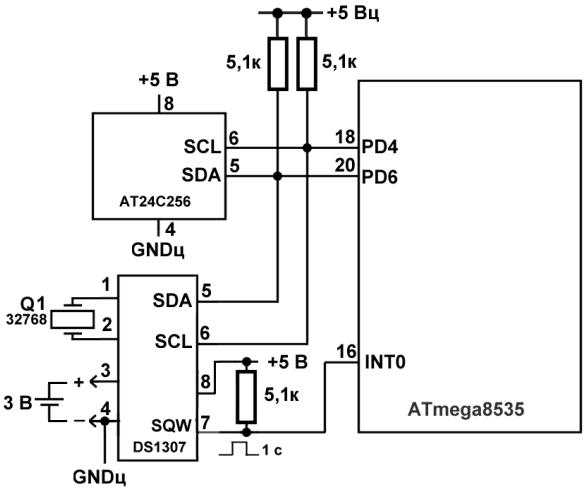


Рис. 16.5. Присоединение часов DS1307 к измерителю температуры и давления

Так что сначала нам придется написать процедуру инициализации часов. Для этого в регистре управления DS1307 (он имеет номер 7) нужно установить бит 4, который разрешает выход частоты для прерывания, и обнулить младшие два бита в этом регистре, что означает частоту на этом выходе 1 Гц (подробности см. в описании DS1307, которое можно скачать с сайта **maxim-ic.com**). Но это еще не все: ранее мы говорили, что необходимо вообще завести часы, установив бит, который отвечает за работу задающего генератора. Это бит номер 7 в регистре секунд — здесь используется тот факт, что максимальное значение секунд равно 59 (напомним, что оно в BCD-форме, потому это равносильно значению $59), и старший бит всегда будет равен нулю. А если мы его установим, то часы стоят, и значение секунд не имеет значения. Потому мы совместим сброс этого бита с установкой секунд в нужное значение (соответствующий регистр самый первый, и имеет адрес $00). Сказанное иллюстрирует листинг 16.11.

Листинг 16.11

```
IniSek:  ;секунды — в temp, если бит 7=1
       ;то остановить, иначе завести часы
       sbis PinC,pSDA   ;линия занята
       rcall err_i2c
   ldi ClkA,0 ;адрес регистра секунд
   mov DATA,temp
   rcall write_i2c
   brcs  stopW
   ldi temp,$AA ;все отлично
   rcall out_com
ret
IniClk: ;установить выход SQW
   ldi ClkA,7 ;адрес регистра управления
   ldi DATA,0b00010000 ;выход SQW с частотой 1 Гц
   rcall write_i2c
   brcs  stopW
   ldi temp,$AA ;все отлично
   rcall out_com
ret
stopW:
       ldi temp,$EE ;подтверждение не получено
       rcall out_com
ret
err_i2c:
       ldi temp,$AE ;линия занята
       rcall out_com
sei
ret
```

Напомню, что процедура `write_i2c` (как и использующаяся далее `read_i2c`) для доступа к часам уже имеется в файле i2c.prg (см. *Приложение 5*). Процедурой `IniSek` мы можем при желании и остановить, и запустить часы. Если нужно остановить, то следует `temp` придать значение, большее 127. Если `temp` меньше 128, то в часы запишется значение секунд, и они пойдут. При обнаружении ошибок в компьютер (без запроса с его стороны) выдается определенный код: $AE, если линия занята, и $EE, если подтверждение со стороны часов (ACK) не получено. Если все в порядке, то выдается код $AA. Те же самые вызовы для выдачи кодов у нас будут в других процедурах обращения к часам.

Раздельные процедуры нам понадобились потому, что иногда часы идут, а выход на прерывание МК у них может оказаться отключенным. Тогда нам

надо только его подключить, а время сбивать не следует. А когда вызывать эти процедуры? Прежде всего, при включении контроллера: мы помним, что при самом первом запуске часы следует заводить обязательно. Но могут быть и сбои при перебоях с питанием (на практике бывало так, что выход SQW при работе от батарейки самопроизвольно отключался). Для того чтобы правильно организовать процедуру, нам следует сначала выяснить, в каком состоянии часы находятся (листинг 16.12).

Листинг 16.12

```
ReadSet:
       sbis PinC,pSDA    ;линия занята
       rcall err_i2c
   ldi ClkA,7 ;адрес регистра управления
   rcall read_i2c
   mov temp,data ;в temp значение регистра управления
   ldi ClkA,0 ;адрес регистра секунд
   rcall read_i2c  ;в data значение регистра секунд
brcs  stopW
ret
```

Записав все эти процедуры в любом месте программы (но поблизости друг от друга, чтобы обеспечить беспроблемный переход на метку `stopW`), мы включаем в процедуру начального запуска такой фрагмент (листинг 16.13).

Листинг 16.13

```
;===========инициализация часов ==========================
       rcall ReadSet ;прочли установочные байты
               ;в temp регистр установок, в data секунды
       cpi DATA,$80 ;если больше или равно 128
       brsh setsek ;то завести часы
       cpi temp,$10 ;если выход не установлен
       brne st_clk;тогда только его установка
       rjmp setRAM
setsek:
       clr temp
       rcall IniSek ;устанавливаем секунды = 0
st_clk:
       rcall IniClk ;установка выхода
setRAM:
      rcall RclockIni ;в любом случае чтение часов в память
. . . . .
```

Значение $10 регистр установок должен иметь, если мы ранее уже устанавливали часы. Процедура чтения значений часов RclockIni в память у нас отсутствует, и мы поспешим исправить это, включив в текст туда же, где находятся остальные процедуры для часов, еще две: ReadClk для чтения BCD-значений и RclockIni для преобразования их в распакованный формат (листинг 16.14). Предварительно зададим место в SRAM, куда мы будем складывать значения всех разрядов времени (включая календарь), и отдельно только часы и минуты, но распакованные (они могут пригодиться для индикации).

Листинг 16.14

```
;SRAM старший байт адреса SRAM=0x01
.equ Sek   = 0x10 ;текущие секунды BCD-значение
.equ Min   = 0x11 ;текущие минуты
.equ Hour  = 0x12 ;текущие часы
.equ Date  = 0x13 ;текущая дата
.equ Month = 0x14 ;текущий месяц
.equ Year = 0x15 ;текущий год
;распакованные часы
.equ DdH = 0x16 ; часы старш. дес.
.equ DeH = 0x17 ; часы младший дес.
.equ DdM = 0x18 ;мин старш. дес.
.equ DeM = 0x19 ;мин младш. дес.
;<начиная с адреса $20 у нас хранятся коэффициенты>
. . . . .
RclockIni: ;инициализация часов
     rcall ReadClk    ;сложили часы в память
     ldi ZH,0x01;
     ldi ZL,Sek ;адрес секунд в памяти
     ld temp,Z ;извлекаем из памяти упакованные Sek
     mov count_sek,temp
     andi temp,0b11110000  ;распаковываем — старший
     swap temp ;старший в младшей тетраде
     ldi data,10
     mov mult10,data ;в mult10 всегда будет 10
     mul temp,mult10 ;умножаем на 10 в r1:r0 результат умножения
     andi count_sek,0b00001111  ;младший
     add count_sek,r0 ;получили hex-секунды
          ldi ZL,Hour ;распакованные в память
          ld temp,Z
          mov data,temp
          andi temp,0b00001111  ;младший часов
          ldi ZL,DeH
          st Z,temp
```

```
          andi data,0b11110000  ; старший часов
          swap data ;старший в младшей тетраде
          ldi ZL,DdH
          st Z,data
          ldi ZL,Min ;распакованные в память
          ld temp,Z
          mov data,temp
          andi temp,0b00001111  ;младший минут
          ldi ZL,DeM
          st Z,temp
          andi data,0b11110000  ; старший минут
          swap data ;старший в младшей тетраде
          ldi ZL,DdM
          st Z,data
ret
ReadClk: ;чтение часов
       ldi ZH,1 ;старший RAM
       ldi ZL,Sek ;адрес секунд в памяти
       ldi ClkA,0 ;адрес секунд в часах
       sbis PinC,pSDA
       rcall err_i2c
       rcall start
       ldi   DATA,0b11010000 ;I2C-адрес часов+запись
       rcall write
       brcs  stopR ;C=1 если ошибка
       mov   DATA,ClkA ;адрес регистра секунд
       rcall write
       brcs  stopR ; C=1 если ошибка
       rcall start
       ldi DATA,0b11010001 ;адрес часов+чтение
       rcall write
       brcs  stopR ;C=1 если ошибка
       set   ;CK
       rcall read  ;читаем секунды
       brcs  stopR ;C=1 если ошибка
       st Z+,DATA ;записываем секунды в память
       rcall read  ;читаем минуты
       brcs  stopR ;C=1 если ошибка
       st Z+,DATA ;записываем минуты
       rcall read ;читаем часы
       brcs  stopR ;C=1 если ошибка
       st Z+,DATA ;пишем часы в память
       rcall read ;день недели читаем, но никуда не пишем
       brcs  stopR ;C=1 если ошибка
       rcall read ;дата — читаем
```

```
        brcs  stopR ; C=1 если ошибка
        st Z+,DATA ;дату записываем
        rcall read ;месяц читаем
        brcs  stopR ; C=1 если ошибка
        st Z+,DATA ;месяц записываем
        clt ;НЕ давать ACK – конец чтения
        rcall read ;год читаем
        brcs  stopR ; C=1 если ошибка
        st Z+,DATA ;год записываем
        rcall stop
ret
stopR:
        ldi temp,$EE ;подтверждение не получено
        rcall out_com
ret
```

Здесь нам пришлось оформить процедуру чтения из часов отдельно, прямым обращением к процедурам чтения через I^2C, т. к. часы имеют специальный и очень удобный протокол. Если вы им один раз даете команду на чтение (значение адреса `0b11010001`), то они начинают выдавать последовательно все значения регистров, начиная с того, к которому было последнее обращение прошлый раз. Здесь мы начинаем с регистра секунд и заканчиваем регистром года. Чтобы остановить выдачу, надо в последнем чтении и не выдавать подтверждение (ACK).

Прочитанные значения складываются в память (в исходном BCD-виде) и отдельно, в процедуре `RclockIni`, распаковываются для индикации. Об индикации мы тут подробно говорить не будем, вы уже знаете, как ее организовать (для этого надо добавить еще четыре разряда ЧЧ:ММ в обработчик прерывания по таймеру `TIM0`, см. окончательный вариант измерителя в конце этой главы), остановимся на применении полученных значений времени для наших целей своевременной записи температуры и давления.

Сначала нам еще надо обеспечить ход времени в МК (в память МК должны все время попадать текущие значения времени) и научиться устанавливать часы: пока мы их только «заводили» и устанавливали секунды. Для счета времени установим отдельный регистр-счетчик секунд (не читать же каждую секунду значения часов) и запомним, что его нельзя трогать:

```
.def count_sek = r26 ;счетчик секунд
```

Теперь начнем с последней задачи: как установить нужное время? Для этого напишем процедуру (листинг 16.15), которая будет вызываться из компьютера по команде `$A1`. А по команде `$A2` будем читать значение часов.

Листинг 16.15

```
Gcykle
. . . . .
     cpi temp,0xA1 ;установить RTC + 6 байт BCD, начиная с секунд
     breq proc_A1
     cpi temp,0xA2 ;читать часы в комп.
     breq proc_A2
. . . . .
rjmp Gcykle
proc_A1: ;A1  установка часов
       rcall SetTime
rjmp Gcykle
proc_A2: ;A2 читать часы в комп. из памяти
       rcall ReadTime;
rjmp Gcykle
. . . . .
      ;Процедура преобразования BCD в HEX, специально для времени
HEX_time: ;на входе в ZL адрес секунды, часы или минуты
;на выходе в temp hex-значение,
             ld temp,Z ;
             andi temp,0b11110000  ;распаковываем — старший
             swap temp ;старший в младшей тетраде
             mul temp,mult10 ;умножаем на 10 в r0 результат
             ld temp,Z ;
             andi temp,0b00001111  ;младший
             add temp,r0 ;получили hex
ret
Sclock: ;получить из компьютера 6 байт и записать в память
       ldi ZH, 0x01 ;старший RAM
       ldi ZL,Sek ;Ram
     rcall in_com
     st Z+,temp ;sek
     rcall in_com
     st Z+,temp ;min
     rcall in_com
     st Z+,temp ;hour
     rcall in_com
     st Z+,temp ;data
     rcall in_com
     st Z+,temp ;month
     rcall in_com
     st Z,temp ;year
push cnt ;сохраняем cnt на всякий случай
       rcall SetClk ;переписываем в часы
```

```
pop cnt
ret

Setclk: ;установить часы
       sbis PinC,pSDA ;линия занята
       rcall err_i2c
       ldi ZH,0x01
       ldi ZL,Sek ;адрес секунд в памяти
   ldi ClkA,0 ;регистр секунд
   ld DATA,Z+ ;извлекаем секунды
   rcall write_i2c ;секунды записываем
   brcs  stopS
   ldi ClkA,1 ;регистр минут
   ld DATA,Z+ ;извлекаем минуты
   rcall write_i2c ;минуты записываем
   brcs  stopS
   ldi ClkA,2 ;регистр часов
   ld DATA,Z+
   rcall write_i2c ;записываем часы
   brcs  stopS
   ldi ClkA,4 ;регистр даты (день недели пропускаем)
   ld DATA,Z+
   rcall write_i2c ;записываем дату
   brcs  stopS
   ldi ClkA,5 ;регистр месяца
   ld DATA,Z+
   rcall write_i2c ;месяц записываем
   brcs  stopS
   ldi ClkA,6 ;регистр года
   ld DATA,Z
   rcall write_i2c ;год записываем
   brcs  stopS
   ldi ClkA,7 ;регистр установок — на всякий случай
   ldi DATA,0b00010000
   rcall write_i2c
   brcs  stopS
   ldi temp,$AA   ;все отлично
   rcall out_com
 ret
stopS:
       ldi temp,$EE ;подтверждение не получено
       rcall out_com
ret
SetTime: ;установка текущих значений в МК
cli
```

```
       rcall Sclock ;записали из компьютера BCD-значения
             ldi ZL,Sek ;упакованные секунды
             rcall HEX_time ;имеем hex-секунды в temp
             mov count_sek,temp ;переписываем в счетчик
      ;далее распаковываем для индикации: часы-минуты
             ldi ZL,Hour ;распакованные в память
             ld temp,Z
             mov data,temp
             andi temp,0b00001111  ;младший часов
             ldi ZL,DeH
             st Z,temp
             andi data,0b11110000  ; старший часов
             swap data ;старший в младшей тетраде
             ldi ZL,DdH
             st Z,data

             ldi ZL,Min ;распакованные в память
             ld temp,Z
             mov data,temp
             andi temp,0b00001111  ;младший минут
             ldi ZL,DeM
             st Z,temp
             andi data,0b11110000  ; старший минут
             swap data ;старший в младшей тетраде
             ldi ZL,DdM
             st Z,data
sei
ret

ReadTime: ;чтения часов из памяти в порядке ЧЧ:ММ ДД.мм.ГГ
cli
       rcall ReadClk ;сначала читаем из часов
       ldi ZH,1 ;старший RAM
       ldi ZL,Hour
       ld temp,Z
       rcall out_com ;hour
       ldi ZL,Min
       ld temp,Z ;
       rcall out_com ;min
       ldi ZL,Sek
       ld temp,Z ;
       rcall out_com ;sek
       ldi ZL,Date
       ld temp,Z+ ;
       rcall out_com ;data
```

```
        ld temp,Z+ ;
        rcall out_com ;month
        ld temp,Z ;
        rcall out_com ;year
sei
ret
```

Как видите, довольно длинно получилось, но ничего не поделаешь. Теперь мы находимся в следующей ситуации: часы установлены и идут сами по себе, в памяти МК имеются значения времени, которые туда записали при установке, есть еще регистр `count_sek`, в котором отдельно хранятся значения секунд в нормальном (а не BCD) цифровом формате. Осталось заставить МК отсчитывать время — сам по себе контроллер никогда не «узнает», который сейчас час.

Для этого мы и припасли прерывание от часов, которое происходит раз в секунду. В принципе мы могли бы каждое это прерывание читать значения времени из часов процедурой `ReadClk`, но это неудобно, т. к. процедура длинная и будет тормозить индикацию. Даже в ПК так не делали — там время отсчитывается BIOS при включенном компьютере самостоятельно. И нет никакой нужды этим заниматься, если мы можем считать время в МК: синхронизацию значений мы при включении питания или при установке часов делаем, а синхронизация хода часов обеспечена тем, что прерывания управляются от RTC. А считать секунды, минуты и часы совсем нетрудно и много времени не займет. Календарь же нам вести в МК не требуется, мы его правильный отсчет получим при чтении из устройства за счет того, что предварительно обновляем значения в памяти процедурой `ReadClk` (см. процедуру ReadTime в листинге 16.15).

Итак, вычеркнем опять из начального запуска процедуру инициализации Timer 1 (всю секцию `Set Timer 1`, вернув вместо нее `ldi temp,(1<<TOIE0)` и `out TIMSK,temp` для инициализации только Timer 0, см. первоначальный текст в *Приложении 5*), уберем из текста обработчик прерывания `TIM1_COMPA` и вместо ссылки `rjmp TIM1_COMPA` в секции прерываний опять поставим команду `reti`.

Вместо этого в секции прерываний для внешнего прерывания INT0 (во второй строке, сразу после `rjmp RESET`) заменим `reti` на `rjmp EXT_INT0`, а в начальную загрузку впишем инициализацию внешнего прерывания `INT0`:

```
;====== внешнее прерывание INT0
ldi temp,(1<<ISC01) ;прерывание. INT0 по спаду
out MCUCR,temp
ldi temp,(1<<INT0) ;разрешение. INT0
out GICR,temp
```

```
ldi temp,$FF ;на всякий случай сбросить все флаги прерываний
out GIFR,temp
```

Теперь, если часы работают, у нас каждую секунду будет происходить прерывание INT0. В нем мы сначала займемся счетом времени, а потом записью во внешнюю flash каждые три часа. Для этого нам придется организовать довольно громоздкую процедуру сравнения времени с заданным. В нашем измерителе мы будем писать с т. н. метеорологическим интервалом (каждые три часа, начиная с 0 часов).

Но писать в память в определенные моменты времени — это еще не все. Метеоданные имеют смысл только, если они привязаны к абсолютному времени. Если же мы будем просто писать в память, как сейчас, то при чтении данных мы никогда не узнаем, когда именно была произведена первая запись. Но даже если мы запишем время включения прибора на бумажке (точно зная интервал, остальные кадры нетрудно привязать к абсолютному времени), то учесть отключения питания мы все равно не сможем. Зачем тогда было придумывать такой хитрый механизм сохранения адреса при сбоях?

Но и писать в память время каждого измерения нецелесообразно — оно займет минимум 5 байт, в нашем случае больше, чем сами данные. Потому мы поступим следующим образом: при начальной загрузке устанавливаем некий флаг (назовем его «флаг первичной записи»), который покажет, что это первая запись после включения питания. Если этот флаг установлен, то мы будем писать время в виде отдельного кадра, а точнее — двух кадров, потому что в один 4-байтовый кадр время + дата у нас не уместится. Можно в принципе и сэкономить, но сделать размер вспомогательного кадра времени кратным кадру данных удобно с точки зрения отсчета адресов во flash. Два кадра займут 8 байт, пять из них есть значение времени, а оставшиеся три мы используем так: будем придавать самым первым двум определенное значение ($FA). Тогда считывающая программа, встретив два $FA подряд, будет «знать», что перед ней кадры времени, а не данных, и их нужно интерпретировать соответствующим образом.

Тут мы учитываем тот факт, что ни данные (10-битовые), ни значения времени не могут содержать байтов, имеющих величину, когда старшая тетрада равна $F. Так что в принципе хватило бы и одного такого байта, но для надежности мы их вставим два подряд (благо их количество позволяет), и у нас даже еще один байт останется в запасе. И его мы также используем: будем писать в него значение регистра `MCUCSR`, в котором содержатся сведения о том, откуда ранее пришла команда на сброс. Отдельные биты в этом байте сбоев (БС) означают следующее:

- Bit 3 — Watchdog Reset Flag (БС = 08) устанавливается, если сброс был от сторожевого таймера;

- Bit 2 — Brown-out Reset Flag (БС = 04) устанавливается, если был сброс от снижения питания ниже 4 В;
- Bit 1 — External Reset Flag (БС = 02) устанавливается, если сброс был от внешнего сигнала Reset (характерно для перепрограммирования);
- Bit 0 — Power-on Reset Flag (БС = 01) устанавливается, если было включение питания МК.

Сейчас нам эта информация не очень нужна, но в дальнейшем, когда мы используем для повышения надежности работы сторожевой таймер, она пригодится для сбора статистики сбоев. После записи кадров времени флаг первичной записи сбрасывается.

Таким образом, после каждого включения МК у нас будет записываться кадр времени, и мы всегда сможем привязать данные к абсолютному времени и дате, даже если в записи был длительный перерыв. Это немного уменьшит полезный объем памяти, но т. к. сбои происходят относительно редко, подобное уменьшение можно не принимать во внимание.

Есть еще один момент, который связан с процедурой чтения данных из flash-памяти: коли мы считаем время в МК отдельно, то при такой длительной процедуре счет неизбежно собьется. Чтобы это исправить, нам требуется в самом конце процедуры `ReadFullFlash` инициализировать часы заново:

```
rcall RclockIni ;заново инициализируем часы
```

Окончательный вариант программы измерителя с часами, суммирующий все, описанное ранее, довольно велик по объему (он содержит порядка 1300 строк, без учета включаемого файла i2c.prg), потому я его в книге не привожу. Его можно воссоздать, если последовательно делать в исходной программе измерителя из *Приложения 5* (раздел «Измеритель температуры и давления на AVR», листинг П5.2) все рекомендованные мной изменения, начиная с листинга 15.9. Не доверяющие своей внимательности и просто ленивые могут его скачать с моей домашней странички по адресу **http://revich.lib.ru/AVR/ctp.zip**. В архиве содержатся оба необходимых файла: собственно программа ctp.asm и файл с процедурами I^2C, который называется i2c.prg и полностью совпадает с тем текстом, что приведен в *Приложении 5* и обсуждался ранее. Распакуйте их в одну папку и не забудьте еще приложить фирменный файл макроопределений m8535def.inc, после чего программу можно компилировать.

Заметки на полях

В этой программе есть потенциальная ошибка, хотя и не очень серьезная: если мы обратимся к какой-либо «длительной» процедуре (в данном случае это чтение содержимого flash), то при совпадении ее по времени с необходимостью записи данных последняя осуществлена не будет, и данные пропадут. Чтобы

такое исключить полностью, надо отслеживать время и вблизи значения часа, кратного трем, запрещать такие процедуры. Можно поступить еще проще — устанавливать при чтении флаг первичной записи, и тогда пропущенная запись не приведет к сбою при анализе информации. Но здесь я не стал в такие моменты углубляться, т. к. совпадение все же крайне маловероятно (чтение занимает максимум полминуты, можно и вручную отследить момент), да и навредить оно вряд ли сможет, т. к. обычно после чтения оператором счетчик обнуляется и запись начинается заново.

Схема измерителя совпадает с приведенной на рис. 15.2, если добавить к ней изменения, представленные на рис. 16.5 и индикацию. Для индикации часов в программе предусмотрены незанятые выводы порта A (с 4 по 7). Их следует присоединить к индикаторам по схеме, аналогичной остальным (как в часах из *главы 14*). Если вы не хотите использовать индикацию часов, то лучше вернуть процедуру по прерыванию Timer 0 в то состояние, которое она имеет в программе измерителя без часов (см. *Приложение 5*), тогда на каждый разряд будет приходиться относительно большая часть времени индикации. Еще один момент в схеме, который мы пока не обсуждали полностью — это соединение с компьютером, напомню, что его мы будем разбирать в *главе 18*.

Обращаю ваше внимание также, что в тексте программы ctp.asm, процедуре записи во flash (строка 496), есть закомментированный оператор `rjmp mm1`. Если его раскомментировать, то запись будет происходить каждую минуту, что позволит выполнить проверку записи во flash, в том числе отследить корректность последнего адреса.

Глава 17

«Зеленые» микросхемы

> Поскольку значительное количество энергии тратится на разговоры, прекращение этого вида деятельности может оказаться весьма благотворным.
>
> *Виктор Санчес «Практическое использование техник Карлоса Кастанеды»*

Глупо тратить энергию только на поддержание работоспособности МК: как и ПК, основное время современные контроллеры тратят на ожидание прерывания или внешних команд. Даже самая громоздкая процедура (не считая медленных по самому принципу действия алгоритмов передачи данных к внешним устройствам и записи в EEPROM), состоящая, к примеру, из тысячи команд процессора, будет выполнена в МК с весьма невысокой тактовой частотой 4 МГц примерно за треть миллисекунды, а то и быстрее. Остальное время контроллер будет простаивать, но потреблять практически столько же, сколько в рабочем режиме. Хочется придумать механизм, который бы позволял «будить» процессор только тогда, когда это требуется. Оказывается, это очень непросто.

Потребляет типичный микроконтроллерный прибор немного: даже с самосветящимися LED-индикаторами наш измеритель температуры и давления тратит не более нескольких ватт, из которых на долю контроллера приходятся лишь доли ватта. Оставшееся расходуется на индикацию и при желании это потребление можно предельно снизить, если использовать ЖК-индикаторы. Потому к таким устройствам не придерется даже самый упертый «зеленый» (в отличие от процессоров для ПК с их гигагерцевыми частотами, часто совсем неоправданными).

И на практике режимы энергосбережения в большинстве случаев целесообразны только тогда, когда ваша схема работает от автономного источника.

Посчитайте: типичный AVR семейства Mega потребляет около 15 мА при напряжении питания 5 В. Таким образом, щелочные батарейки самого распространенного для таких устройств типа AA (см. *Приложение 2*) при емкости примерно 2000 мА·ч проработают всего 133 часа или чуть больше 5,5 суток — это не считая потребления внешних устройств.

При использовании режима энергосбережения следует очень тщательно продумывать и схему, и саму процедуру перехода в этот режим. Может случиться, что какие-то внешние устройства у вас потребляют больше, чем сам контроллер, тогда все ваши усилия пойдут насмарку. Например, в схеме часов из *главы 14* (там энергосбережение нецелесообразно, но я использую его схему, как наглядный пример) разряды порта D и B управляют транзисторными ключами. Если мы введем МК в режим энергосбережения, то процессор остановится, но состояния портов останутся такими, какими они были к моменту остановки. И если вдруг они случайно оказались в состоянии высокого уровня, то на каждый вывод придется немного менее чем по 1 мА вытекающего тока. При всех включенных выводах это около 10 мА, что сравнимо с потреблением самого контроллера. Какое уж тут энергосбережение…

О режимах энергосбережения AVR

В контроллерах AVR доступны несколько разных режимов энергосбережения (до пяти). Все они вызываются единой командой `sleep`, а результат ее выполнения различается в зависимости от предварительных установок. Интересно, что фирменное описание рекомендует производить установки и разрешать режим «сна» непосредственно перед подачей команды `sleep` — иначе МК, по словам разработчиков, может уйти в «сон» самопроизвольно. Это неудобно, но придется такой рекомендации следовать.

Что касается самих режимов, то принципиально отличаются от остальных лишь режим Idle (ждущий) и ADC Noise reduction. В первом из них отключение касается лишь центрального процессора, а все остальные устройства продолжают функционировать. При этом потребление микросхемы уменьшается всего лишь на треть, и на фоне общего расхода энергии всей схемы с учетом внешних устройств такая экономия теряется. Режим Idle удобен тем, что «просыпание» происходит мгновенно по любому прерыванию. В режиме ADC Noise reduction (подавления шумов АЦП — мы его не задействовали), кроме отключения процессорного ядра, отключаются также порты ввода/вывода, в остальном он аналогичен Idle. Его используют только по прямому назначению, т. к. питание также практически не экономится, а выводить МК из этого режима сложнее, чем в случае Idle.

С точки зрения энергосбережения более интересны остальные режимы, которые не очень сильно отличаются друг от друга. Во всех моделях AVR без исключения имеется наиболее универсальный режим Power Down, при котором отключаются все внутренние тактовые сигналы. Потребление МК при этом снижается до нескольких микроампер. Вывести МК из такого состояния можно лишь внешним прерыванием (и то не всяким, о чем далее) или сбросом, т. е. выключением/включением питания, подачей внешнего сигнала Reset или сигналом от сторожевого таймера (последним мы займемся в конце этой главы). Вывод из состояния «сна» в режиме Power Down занимает довольно много времени, и это время еще требуется контролировать с помощью конфигурационных битов.

Потому мы здесь будем рассматривать его модификацию под названием Standbye. В этом режиме выключается все, кроме тактового генератора, что несколько повышает потребление (до десятков микроампер), зато вывод из «сна» занимает всего 6 машинных тактов. Так как Standbye имеется не во всех AVR семейства Mega, а в семействах Classic и Tuny его нет вообще, то для этих МК можно без переделок задавать обычный режим Power Down, не забывая только, что пробуждение тогда может длиться до более чем 10 мс (это важно, например, если МК управляется случайными сигналами и промежуток между ними может быть короче времени «пробуждения» — сигнал попросту потеряется).

Внешние прерывания, которыми можно вывести МК из режима Standbye, — это прерывания INT0, INT1 и INT2 (если последнее имеется в данной модели). Однако сначала требуется тщательно изучить их режимы, т. к. они делятся на синхронные и асинхронные. Мы уже знаем из *главы 12*, что синхронные прерывания — это те, которые обнаруживаются лишь в момент прихода фронта тактового импульса. Их преимущество в практически мгновенном обнаружении сигнала на прерывание, но в режиме «сна», когда тактовые сигналы отключены, они, естественно, не могут быть обнаружены. К синхронным относятся прерывания по фронту и по спаду INT0 и INT1. Режим прерывания по низкому уровню асинхронный, потому он пригоден для «пробуждения» МК. Кроме этого, очень удобно, если МК поддерживает прерывание INT2, поскольку оно имеет режимы только по фронту или по спаду, которые обнаруживаются асинхронно, в отличие от INT0 и INT1 (из основных представителей семейства Mega INT2 нет только у ATmega8).

Очень важный момент здесь — ответ на вопрос: а что, собственно, будет делать МК, если его «разбудить» по прерыванию? Если это происходит в результате сброса (от любого источника), то выполняется, естественно, процедура начальной загрузки, как при включении питания. А «разбуженный» внешним прерыванием контроллер сделает совсем другое: *в первую очередь*

он выполнит процедуру прерывания, которое его разбудило, а после этого перейдет к выполнению команды, следующей после `sleep`, на которой выполнение программы было прервано уходом в состояние «сна». Такая определенность очень важна. Хочу обратить также внимание, что команда `sleep` игнорируется, если она расположена внутри обработчика прерывания, она допустима лишь в тексте основной программы. Этот момент в фирменных руководствах явно не прописан, потому обратите на него особое внимание.

Измеритель давления и температуры в автономном режиме

Покажем возможную схему действий на примере нашего измерителя температуры и давления из *главы 15* (без часов и записи в память). Схема его была приведена на рис. 15.2, и мы помним, что там не показана индикация (она такая же, как у часов на рис. 14.2, только используются другие разряды портов). В принципе можно было бы использовать ЖК-индикаторы, и тогда срок службы батарей сильно увеличивается (и их количество можно уменьшить, см. далее), но это потребует заметного изменения системы управления индикаторами (на ЖК, напомню, надо подавать переменное напряжение). К тому же подобрать удобный и эстетичный ЖК-индикатор для нашей конкретной цели — тоже отдельная и непростая задача. Чтобы не уходить в сторону от темы, я решил не останавливаться на этом вопросе: оставим LED-индикаторы, как более яркую иллюстрацию особенностей программирования режима энергосбережения.

Не забудем, что аналоговая часть потребляет сама по себе не менее 3 мА, потому ее придется отключать принудительно. Вариант батарейного источника питания для измерителя в автономном режиме, сделанный с учетом этого обстоятельства, показан на рис. 17.1. Один набор батарей служит здесь для обеспечения всех напряжений, причем напряжение аналоговой части (положительное и отрицательное) формируется с помощью инвертора, входящего в состав микросхемы P6BU-0505Z фирмы PEAK Electronic. Входное напряжение для этого DC/DC-преобразователя берется от стабилизатора цифровой части схемы.

Микросхема P6BU-0505Z может быть заменена на аналогичные изделия других фирм, только следите за характеристиками: так, преобразователи TMR-3 фирмы TRACO имеют встроенную возможность отключения, могут работать от нестабилизированного входного напряжения (т. е. прямо от батарей), но если присмотреться внимательно, то окажется, что они плохо работают при малых токах потребления (менее 20% от номинала) и даже в режиме StandBye потребляют до 10 мА, что, конечно, неприемлемо в нашем случае.

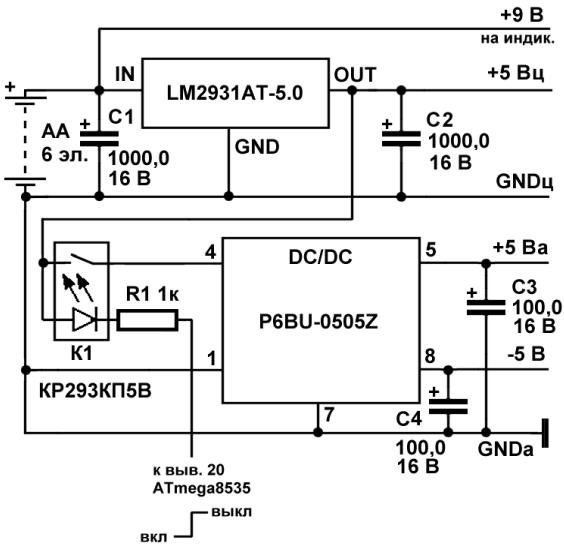


Рис. 17.1. Батарейный источник питания для измерителя температуры и давления

Выбранная микросхема потребляет на холостом ходу гораздо меньше (порядка 1 мА), но встроенной возможности отключения не имеет, потому приходится вводить отдельный ключ (электронное реле КР293КП5В с контактами на замыкание), отключающий аналоговую часть при переходе в режим «сна». Для включения/отключения используем разряд 6 порта D (вывод 20 микросхемы ATmega8535), который придется устанавливать на выход и с нулевым уровнем, иначе схема не заработает.

Если пренебречь разделением аналогового и цифрового положительного питания, то P6BU-0505Z можно заменить на доступный и дешевый (порядка 1 долл.) преобразователь напряжения +5 В в одно напряжение –5 В под названием ICL7660 (он же MAX1044, он же 1168ЕП1), который включается примерно по той же схеме, но требует дополнительно еще двух конденсаторов. Главный недостаток этой микросхемы — невысокая стабильность выходного напряжения, из-за чего точность измерений заметно снизится.

Индикаторы питаются нестабилизированным напряжением от 6 элементов питания, т. к. минимальное напряжение, при котором индикаторы еще горят, равно приблизительно 6 В. Среднее же напряжение во время эксплуатации будет около 8 В, соответственно ограничительные резисторы тока сегментов (R27—R34 на рис. 14.2), которые в сетевой схеме были равны 470 Ом, следует уменьшить как минимум вдвое — примерно до 200—220 Ом.

Для щелочных батареек типа AA общее время работы до исчерпания их ресурса составит около 10 часов (потребление тока шестью одновременно включенными индикаторами по схеме рис. 14.2, с учетом уменьшения сопротивлений составит максимум 200 мА), что для наших целей приемлемо. Если хотите увеличить время работы, и габариты позволяют, то следует выбрать более емкие батареи — так, со щелочными батареями типа D (особенно модели Duracell Ultra) наш прибор проработает около недели.

Задачу сформулируем так: пусть по первому нажатию внешней кнопки контроллер «просыпается», а по второму — «засыпает». Кроме этого, введем режим автоматического «засыпания» по истечении некоторого промежутка времени. Чтобы пользоваться прибором было удобно, этот промежуток должен быть достаточно большим: не менее минуты, так что нам придется потрудиться (встроенный таймер, как нам известно, обеспечит лишь интервал порядка 16 с). Разумеется, при выключении контроллера должна отключаться и индикация.

Заметки на полях

Одно замечание: никогда не проектируйте устройств, в которых режим энергосбережения не выключается отдельной кнопкой! Посмотрите, как неудобно пользоваться мобильными телефонами, в которых для включения подсветки экрана надо обязательно совершить какое-то действие. В идеале устройство должно содержать возможность отключения энергосбережения вообще, хотя в простейших случаях это не всегда удобно, и внешней кнопки включения/отключения достаточно, но если устройство управляется через экранное меню или от компьютера, то оно обязательно должно иметь команду полного отключения режима энергосбережения (или, по крайней мере, установки достаточно большого — более 10 минут — интервала отключения). Если подобных возможностей нет, это сразу говорит о неряшливости разработчиков.

Использование режима энергосбережения

В случае использования прерывания по низкому уровню, как мы уже знаем из *главы 12*, при первом же возникновении его следует запретить, иначе оно будет происходить непрерывно, пока действует низкий уровень. Это не очень удобно, когда прерывания управляются от внешних сигналов в форме меандра (как от часов из *главы 16*), потому что возникает вопрос — а когда разрешать его снова? Если перед уходом в «сон», то, в силу длительности состояния низкого уровня на выводе прерывания, оно тут же произойдет опять, и МК вообще никогда (в течение текущего полупериода) не «заснет». И управление «засыпанием» и прерываниями очень сильно усложняется. В таких случаях лучше будет задействовать прерывание INT2, которое не нужно запрещать (дребезг в таких случаях исключен), и соответствующую модель контроллера.

Но для нашей ситуации с управлением от кнопки это безразлично, поскольку дребезг при нажатии все равно приводит к необходимости запрета даже в прерываниях по фронту, а проблемы разрешения заново не возникает: непосредственно перед «засыпанием» или по таймеру. Если в момент разрешения кнопка окажется все еще нажата, то МК просто тут же проснется (или заснет) заново, ничего страшного. И мы, не мудрствуя лукаво, используем здесь универсальное для всех моделей AVR прерывание INT0 по низкому уровню.

Кнопка (без фиксации) подсоединяется к выводу INT0 (PD2, вывод 16 для ATmega8535). Если кнопка имеет перекидной контакт (т. е. три вывода), она подсоединяется к «земле» и питанию так, как показано на рис. 8.3, *б* (вместо элемента «исключающее ИЛИ» выступает, естественно, наш МК). Чаще встречаются кнопки с двумя выводами, тогда их подсоединяют так, как показано на рис. 17.2. В обоих случаях при ненажатой кнопке на выводе должен быть потенциал питания, т. е. высокий уровень, а при нажимании вывод коммутируется на «землю».

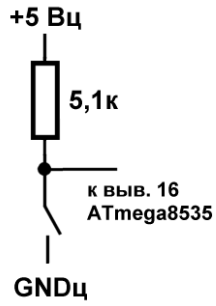


Рис. 17.2. Подсоединение кнопки с двумя выводами к МК ATmega8535

Заметки на полях

Еще раз напомню (см. *главу 12*), что в принципе от резистора на схеме рис. 17.2 можно отказаться, поскольку специально для таких случаев в AVR предусмотрено подключение внутреннего «подтягивающего» резистора. Однако для надежности ставить его все же следует (так же, как и по выводам программирования, см. описание схемы к рис. 14.2 в *главе 14*), т. к. номинал встроенного «подтягивающего» резистора достаточно велик (минимум 35 кОм, согласно руководству), то на нем могут возникать наводки, которые приведут к ложным срабатываниям кнопки. Вы можете попробовать исключить резистор из схемы и убедиться в этом сами — ложные срабатывания появятся обязательно. И уж наверняка ложное прерывание будет возникать в ситуации, описанной в *главе 14* для кнопки Кн1, когда питание внезапно переключается с сети на батареи. Так как мы тут такой режим не используем, то конденсатор параллельно кнопке (см. схему подключения Кн1 на рис. 14.2) ставить необязательно.

Доработка программы

Доработку программы измерителя (напоминаю, что за основу берем листинг П5.2, приведенный в *Приложении 5*, в разделе «Программа измерителя температуры и давления») начнем с того, что, во-первых, включим питание аналоговой части. Для этого в разделе начальной загрузки, там, где устанавливаются порты («установка портов вход/выход»), вместо команды `ldi temp, 0b10000000` (перед `out DDRD,temp`) запишем `ldi temp,0b11000000`. Теперь вывод 20 сконфигурирован на выход и для надежности следует добавить еще оператор `cbi PortD,6` (т. к. включается ключ низким уровнем).

Во-вторых, инициализируем таймер (Timer 1). В пробной программе записи во flash из *главы 16* мы его настраивали ровно на 15 с, а здесь выберем вдвое меньший интервал — 7,5 с. В загрузочную секцию вместо строк инициализации Timer 0 (`ldi temp,(1<<TOIE0)` и `out TIMSK,temp`) добавляем строки листинга 17.1.

Листинг 17.1

```
;++++++++Set Timer 1
     ldi temp,high(29297)
     out OCR1AH,temp
     ldi temp,low(29297)
     out OCR1AL,temp
     ldi temp,0b00001101
     out TCCR1B,temp ;1/1024; очистить после совпадения
     ldi temp, (1<<TOIE0)|(1<<OCIE1A) ;разреш. прерывания
        ;по совпадению для Timer 1 и переполнению Timer 0
     out TIMSK,temp
```

Переключающий режим для вывода `PD5-OC1A` здесь мы не используем. Кроме этого, введем переменную `count_min`, с помощью которой будем считать интервалы в 7,5 с. В секции объявления переменных добавим:

```
.def count_min = r23; счетчик 7,5-секундных интервалов
```

А в секции начальных установок его не забудем обнулить:

```
clr count_min
```

Далее введем специальный флаг `sleep` (пусть будет бит 7 в регистре `Flag`), который будет сигнализировать о режиме. Если этот бит установлен — пора «спать», если обнулен — работаем, как ни в чем не бывало. По умолчанию он обнулен (см. секцию «начальная установка переменных») и нам ничего не грозит, если мы вставим в основной цикл запуск режима энергосбережения по схеме, согласно листингу 17.2.

Листинг 17.2

```
Gcykle:
  sbrs Flag,7 ;если бит 7 установлен, то засыпаем
rjmp Gcykle ;иначе бесконечный цикл
cli ;на всякий случай запрещаем прерывания
;все порты на вход, и нули в разряды, кроме PortD,6
  clr temp
  out DDRB,temp
  out DDRC,temp
  out DDRD,temp
  out PortB,temp
  out PortC,temp
  ldi temp,0b01000000 ;выключение питания на всякий случай
  out PortD,temp
  ldi temp,0b11100000 ;разрешение Sleep, режим Standby
                      ;прерывание по уровню
  out MCUCR,temp
  ldi temp,(1<<INT0) ;разрешение INT0
  out GICR,temp
sei ;разрешаем прерывания
Sleep ;наконец, спим
  cbr Flag, $80 :по выходу из сна сбрасываем флаг sleep
  clr count_min ;отсчет времени
; сначала установка портов вход-выход обратно
cli ;на всякий случай запрещаем прерывания
  ldi temp,0b00111111 ;разряды
  out DDRB,temp
  ldi temp,0b01111111 ;сегменты
  out DDRC,temp
  ldi temp,0b11000000 ;знак минус и питание
  out DDRD,temp
  clr temp
  out PortD,temp ;включить аналоговое питание
  out MCUCR,temp ;запрещаем режим Sleep
  out TCNT1H,temp ;очищаем счетные регистры таймера
  out TCNT1L,temp
  ldi temp,0b00001101 ;запускаем таймер
  out TCCR1B,temp ;1/1024 очистить после совпадения
sei ;разрешаем прерывания
rjmp Gcykle ;бесконечный цикл
```

Теперь самое сложное: разобраться с прерываниями и установкой флага `sleep`. Не забудьте заменить `reti` на `rjmp TIM1_COMPA` в таблице прерываний,

в строке для прерывания Timer1 Compare A (шестое сверху, не считая RESET). Прерывание таймера иллюстрирует листинг 17.3.

Листинг 17.3

```
TIM1_COMPA:
  inc count_min
  cpi count_min,1 ;через 7,5 с разрешаем INT0
  brne schet_time
  ldi temp,(1<<INT0) ;разрешение INT0
  out GICR,temp ;GIMSK и GIGR — синонимы
schet_time: ;здесь отсчет времени
  sbrs count_min,3 ;если бит 3 = 1, то прошло 8 интервалов =1 мин
reti ;иначе выходим
  clr count_min ;в след. раз — сначала
  sbr Flag,$80 ;устанавливаем бит sleep
  clr temp ;останавливаем таймер
  out TCCR1B,temp
reti ;выходим
```

Если бы мы не связывались с кнопкой, то на этом можно было бы закончить — через 1 минуту после включения у нас измеритель уходит в «сон», из которого его можно вывести только выключением-включением питания или подачей сигнала Reset. Но мы хотим все делать грамотно, потому используем кнопку. Прерывание INT0 тогда будет выглядеть так, как показано в листинге 17.4 (также не забудьте заменить в таблице прерываний во второй строке `reti` на `rjmp EXT_INT0`).

Листинг 17.4

```
EXT_INT0:
  clr temp
  out GICR,temp ;запрещаем внешние прерывания
sbrs Flag,7 ;если были во сне, то больше ничего не делаем
reti
  clr temp ;иначе готовимся ко сну
  out TCCR1B,temp ;останавливаем таймер
  clr temp ;чистим счетные регистры таймера
  out TCNT1H,temp
  out TCNT1L,temp
  ldi temp,0b00001101 ;заново запускаем
  out TCCR1B,temp
ldi count_min,7 ;на интервал 7,5 с
reti
```

Теперь у нас измеритель будет работать как задумывали: после включения через 1 мин. он уходит в режим энергосбережения, когда индикаторы гаснут, и потребление минимизируется. Если нажать на кнопку, то МК «проснется» и выполнит все процедуры после команды Sleep. Если ничего не делать, то через минуту измеритель опять «заснет». Если нажать до истечения этого срока на кнопку (но не ранее, чем через 7,5 с), то он «заснет» через 7,5 с после нажатия. В принципе таймер можно было бы и не останавливать перед засыпанием, но так мы более уверены, что он начнет отсчет с самого начала.

Время задержки вы легко можете регулировать, просто меняя число, которое загружается в регистры `OCR1AH/L`. Оно рассчитывается исходя из формулы: время задержки в секундах равно частоте кварца в герцах, деленному на коэффициент предварительного деления (1024) и на это число. Например, если вместо 29 297 загрузить 11 719, то пауза до засыпания по нажатию кнопки станет равной 3 с, а время работы сократится менее чем до полуминуты. Чтобы увеличить время «бодрствования» вдвое, команду `sbrs count_min,3` в прерывании таймера нужно заменить на `sbrs count_min,4`.

Заметки на полях

В прерывании кнопки для исключения случайного попадания в конец отсчитываемого интервала времени используется очистка регистров таймера. Тогда таймер начнет считать сначала, пока не достигнет заданного числа. Если вам потребуется задать отсчитываемый интервал очень точно, то следует почистить также счетчик предделителя (в семействе Classic такой возможности не было). Для этого в регистре `SFIOR` нужно записать единицу (не ноль!) в бит, соответствующий таймеру: для Timer 1 (а также Timer 0, так как у них предделитель общий) это будет бит 0 под названием `PSR10`. Этого не нужно делать при установленном счете напрямую (с коэффициентом деления тактовой частоты 1/1), в остальных случаях при запуске таймера в произвольный момент времени счетчик предделителя начнет со случайного числа (за исключением момента начального запуска при включении питания). Чем меньше коэффициент деления и чем больше число, отсчитываемое таймером, тем меньше относительная погрешность, но абсолютная ошибка всегда может достигнуть величины интервала между отчетами таймера (при коэффициенте 1/1 она попросту неустранима, только и всего).

Вторая особенность таймеров семейства Mega — наличие асинхронного режима работы для одного из таймеров (в большинстве моделей это Timer 2). Тогда его можно завести от независимого источника импульсов (от внешней частоты или низкочастотного кварца, в том числе часового 32 768 Гц), и он может считать независимо от работы всей остальной схемы. Теоретически эту функцию можно использовать как RTC, но на практике это неудобно (нет ни счета времени, ни календаря), зато ее очень удобно применять для вывода МК из «сна» по времени. В режиме энергосбережения под названием Power Save МК будет «просыпаться» каждое прерывание от Timer 2, и подсчетом этих прерываний его можно окончательно «разбудить» автоматически через нужный промежуток времени.

Использование сторожевого таймера

Сторожевой (watchdog) таймер — одно из самых полезных устройств в составе микроконтроллеров AVR. Причем это неочевидно: в нормальном режиме работы, когда все настроено идеально, он вовсе не нужен. Но представьте себе такую ситуацию: МК настроен на прием данных от компьютера, причем по простейшей схеме из *главы 16*, с непрерывным опросом бита UDRE. К примеру, он ожидает шесть байтов, как в нашей программе, но на пятом байте ПК внезапно ломается (кто-то прошел и ногой выдернул провод из COM-порта). Что будет с контроллером? Естественно, он повиснет в ожидании байта, и из этого состояния его не удастся вывести никаким способом, кроме полного сброса. Еще более опасны в этом отношении наши процедуры опроса линии по интерфейсу I^2C — при программировании для ПК нам бы «голову оторвали» за такую организацию процесса.

И тем не менее здесь это нормальный способ программирования — нецелесообразно усложнять программу на порядки только для того, чтобы исключить все возможные ситуации, которые, может быть, за время «жизни» прибора вообще ни разу не произойдут. В ПК все иначе: во-первых, там есть удобные инструменты для таких случаев, во-вторых, там от одной программы могут зависеть и другие. Вот если бы мы на МК сооружали устройство управления космическим кораблем — тогда другое дело…

Но все же, как быть в таких случаях — ставить специальную кнопку Reset (как в компьютере) или писать в инструкции «если прибор не реагирует, то выключите и включите питание»? Вот тут-то на помощь и приходит сторожевой таймер, который, будучи включен, выполняет одну-единственную операцию: считает импульсы от собственного генератора (абсолютно независимо от всей остальной схемы МК), и когда досчитает до заданного их числа, не обращая внимания ни на что, попросту сбрасывает процессор, как будто был подан сигнал Reset. Самая длительная выдержка, которую можно получить от сторожевого таймера, составляет примерно 2 с (с большим разбросом, т. к. задающий генератор простейшего RC-типа).

Ну и что с этим делать, спросите вы? Нельзя же сбрасывать МК каждые две секунды «на всякий случай» и начинать работу заново, правда? Но этого и не требуется: достаточно завести сторожевой таймер, а потом сбрасывать его в исходное состояние, не дожидаясь, пока он сбросит контроллер. Это можно делать специально по таймеру или в любой другой периодически протекающей процедуре, которая должна выполняться раньше, чем таймер успеет сбросить процессор. Тогда, если МК завис по любой причине (даже просто из-за ошибки в программе), таймер сработает и приведет все в начальное

состояние. В том числе, кстати, «разбудит» МК, даже если тот находится в самом глубоком «сне» (в режиме Power Down).

Для примера разберем такой случай возможного использования сторожевого таймера. В измерителе с часами, который был описан в *главе 16*, опасная ситуация может возникнуть, если по каким-то причинам не «придет» прерывание от часов. Это может произойти, например, если сами часы «повисли» (у них ведь тоже достаточно сложный алгоритм) или просто «потерялся» фронт очередного импульса. А так как мы настраиваемся на круглогодичную работу этого прибора, то подобную ситуацию следует рассматривать как вероятную. В конце концов, космическая частица может прилететь и все нарушить — согласно некоторым исследованиям, сбои такого рода неизбежны с вероятностью примерно 1 сбой на 1000 часов работы (правда, автор этих строк лично ничего такого не наблюдал, но и специально вопрос не исследовал). В этом случае неплохо выполнить процедуру включения прибора заново: в начале программы мы в том числе инициализируем и часы, а уж если они совсем сломались, тут ничего не поделаешь.

Для этого перед началом основного цикла инициализируем сторожевой таймер:

```
;запускаем WDT на 2 сек:
wdr ;команда на сброс — так рекомендуется
ldi temp,(1<<WDCE)|(1<<WDE)
out WDTCR,temp
ldi temp,(1<<WDP0)|(1<<WDP1)|(1<<WDP2)|(0<<WDCE)|(1<<WDE)
out WDTCR,temp
```

Теперь осталось по каждому прерыванию от часов (INT0) просто сбрасывать сторожевой таймер:

```
EXT_INT0:
wdr ;сброс сторожевого таймера
. . . . .
```

Так как прерывание должно возникать каждую секунду, то мы сбрасываем таймер заведомо раньше, чем он сработает, и он начнет отсчет выдержки сначала. Если же что-то (часы или программа) «повиснет», то произойдет общий сброс МК, и он начнет работать опять. Причем после чтения данных из flash мы сможем это обнаружить: если помните, мы в кадр времени записывали байт сбоев, в котором установленный бит 3 означал, что сброс произошел именно от сторожевого таймера.

В этом деле есть единственный «тонкий» момент: если у нас где-то в программе имеется процедура с запрещением прерываний, выполняющаяся дольше, чем 2 с, то сторожевой таймер на время ее выполнения, естественно,

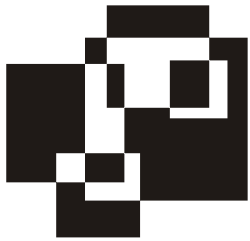
следует выключить (не просто сбросить, а вообще запретить его работу). У нас такая процедура есть — это чтение данных из памяти, при котором даже индикация выключается. Вызов этой процедуры придется переписать так, как показано в листинге 17.5.

Листинг 17.5

```
proc_F2: ;F2 читать flash
cli ;запрещаем прерывания
;выключить WD:
     wdr ; Reset WDT
     in temp, WDTCR
     ori temp, (1<<WDCE)|(1<<WDE)
     out WDTCR, temp
     ldi temp, (0<<WDE); выключить WDT
     out WDTCR, temp

     rcall ReadFullFlash ;читаем данные
       ;запускаем WDT обратно, 2 с
       wdr ;команда на сброс
       ldi temp,(1<<WDCE)|(1<<WDE)
       out WDTCR,temp
       ldi temp,(1<<WDP0)|(1<<WDP1)|(1<<WDP2)|(0<<WDCE)|(1<<WDE)
       out WDTCR,temp
sei ;разрешаем прерывания — необязательно, уже есть в ReadFullFlash
rjmp Gcykle
```

Заметьте, что после такой длительной процедуры можно прерывания и не разрешать (для этого придется убрать разрешение и из самой процедуры `ReadFullFlash`) — а вдруг мы чего-то нарушили в работе? Тогда контроллер просто перезапустится с нуля, и работа восстановится.



Глава 18

Персональный компьютер и системы на МК

> Обычный программист пишет код, чтобы отработать свою зарплату. Великий хакер готов заниматься программированием для собственного удовольствия, поэтому его искренне радует, когда кто-то готов за это заплатить.
>
> *Пол Грэхем*

Если вы умеете паять и знаете, с какого конца отсчитываются выводы у микросхемы — в современной электронике это даже не полдела. В лучшем случае вы сможете повторить часть тех конструкций, что описаны в этой книге, публикуются в журнале «Радио» или размещаются на сайте **shema.ru**. Но ничего серьезного вам создать не удастся, пока вы не научитесь самостоятельно писать «верхние» программы, т. е. такие, с помощью которых персональный компьютер (ПК) «общается» с вашим устройством.

Каналом связи с ПК имеет смысл снабжать практически все конструкции — я уже отмечал, что огромная доля современной электроники модифицируется простой заменой микропрограммы («перешивкой»), это стало даже стандартным маркетинговым приемом. Но дело даже не в этом: возьмите довольно простые часы, которые мы разбирали в *главе 14*. Не правда ли, они бы ничего не потеряли, если бы были снабжены дополнительной возможностью установки времени через ПК, а не только кнопочками? Нажал экранную кнопку «Установить», и время из компьютера переписалось в часы. Секундное дело — вместо того, чтобы вручную «ковыряться» с кнопками.

Беда в том, что для подобных вещей почти невозможно приспособить какую-то готовую программу. «Почти», потому что далее я предложу вам программу, с помощью которой можно работать со всеми устройствами, упоминающимися в этой книге. Однако, во-первых, мне пришлось эту программу написать *самому* — за вас. Во-вторых, будучи универсальной, она, как водится,

для каждого конкретного случая неудобна (лучше всего эта программа подходит для отладки, с какой целью и создавалась). В-третьих, я ее создавал «под себя», и она не может охватить всех возможных вариантов, например, она не работает с девятибитовыми посылками с проверкой четности или с синхронными протоколами. Во всех этих случаях вам придется писать программу самому, причем, как правило, каждый раз заново.

Правда, в этом деле есть два смягчающих обстоятельства. Во-первых, создание таких программ заметно проще, чем написание офисных или веб-приложений. Вам не потребуется знание SQL, Java или Ajax — достаточно общего понимания того, как работает Windows. С другой стороны, вам и не удастся обойтись стандартными компонентами Delphi или Visual Basic — придется привлекать нестандартные компоненты или использовать функции Windows API, обращение с которыми может вызвать приступы «истерического бешенства» похуже, чем любая Java. Но в целом создание типовых приложений для обслуживания электронных приборов — все же сравнительно несложная и к тому же достаточно консервативная область искусства программирования.

Начнем с того, что разберем технические аспекты взаимодействия ПК с микроконтроллерными устройствами.

Соединение ПК и МК

Я уже упоминал о том, что средой обмена данными между UART служит RS-232, где логика отрицательная (логическая единица есть низкий уровень напряжения, см. рис. 16.1), а сами уровни двуполярные, причем с большим допуском: низкий уровень представлен напряжением от –12 до –3 В, а высокий от +3 до +12 В. В этих пределах любая линия передачи по этому стандарту надежно работает.

Как мы говорили в *главе 16*, приемник RS-232 дополнительно снабжают схемой, которая фиксирует уровень не один раз за период действия бита, а трижды, при этом за окончательный результат принимается уровень двух одинаковых из трех полученных состояний линии (*мажоритарная* схема), таким образом удается избежать случайных помех. И хотя длина линии связи по стандарту не должна превышать 15 м, но на практике это могут быть много бóльшие величины. Если скорость передачи не выбирать слишком высокой, то RS-232 может уверенно работать на расстояниях в десятки и даже сотни метров (автору этих строк удавалось без дополнительных ухищрений наладить обмен с компьютером на скорости 4800 по кабелю, правда, довольно толстому, длиной около полукилометра). В табл. 18.1 приведены ориентиро-

вочные эмпирические данные по длине неэкранированной линии связи для различных скоростей передачи.

Таблица 18.1

Скорость, бод	Длина кабеля (неэкранированного), м
9600	75
2400	150
110	900

Эти данные ни в коем случае не могут считаться официальными — слишком много влияющих факторов (уровень помех, толщина проводов, их взаимное расположение в кабеле, фактические уровни напряжения, выходное/входное сопротивление портов и т. п.). Для экранированного кабеля длину можно увеличить примерно в полтора-два раза.

> ***Замечание***
>
> В правильно построенной экранированной линии экран не должен быть одним из токоведущих проводов, т. е. контакты GND соединяются отдельным проводом в кабеле, а экран либо соединяется с GND только на одной стороне — там, где имеется более качественное настоящее заземление, либо — в случае, если сигнальная «земля» GND является «плавающей» относительно «настоящей» земли — вообще только с заземлением.

При «несанкционированной» длине кабеля связи, особенно на больших скоростях передачи, следует применять меры по дополнительной проверке целостности данных: контроль четности, и/или программные способы (вычисление контрольных сумм и т. п.).

RS-232 также стандартизирует всем известные разъемы типа DB. Полный список всех контактов для двух стандартных разъемов типа DB (9- и 25-контактного) приведен в табл. 18.2. Нумерация контактов DB-разъема обычно приведена прямо на нем. Отметим, что 25-контактный разъем был создан первоначально в расчете на развитие стандарта, но позднее стало ясно, что развития не предвидится, и создали 9-контактный, которого достаточно для всех нужд, и в настоящее время практически только он и используется. Кабельная часть для RS-232 обычно представляет собой гнездовую часть разъема («маму»), а приборная — штыревую («папу»). Потому, к примеру, разъем LPT, который тоже есть 25-контактный DB, перепутать с COM-портом невозможно: для LPT на компьютере установлена гнездовая часть, а для COM-порта — штыревая.

Таблица 18.2

COM 9(25)	Обозначение	Направление	Сигнал
1 (8)	DCD	Вход	Детектор принимаемого сигнала с линии (Data Carrier Detect)
2 (3)	RxD	Вход	Принимаемые данные (Receive Data)
3 (2)	TxD	Выход	Передаваемые данные (Transmit Data)
4 (20)	DTR	Выход	Готовность выходных данных (Data Terminal Ready)
5 (7)	GND	–	Общий (Ground)
6 (6)	DSR	Вход	Готовность данных (Data Set Ready)
7 (4)	RTS	Выход	Запрос для передачи данных (Request To Send)
8 (5)	CTS	Вход	Разрешение для передачи данных (Clear To Send)
9 (22)	RI	Вход	Индикатор вызова (Ring Indicator)

Смысл дополнительных линий в том, что они могут применяться для организации различных синхронных протоколов обмена (протоколов с handshakes — «рукопожатием»). В «чистый» UART они не входят, в контроллере их организуют выводами обычных портов (но они входят в отдельные микросхемы UART для реализации полного протокола RS-232, а также в USART). Большинство устройств их не задействует. Однако любое устройство, использующее «рукопожатия», можно подключить к устройству, не имеющему этой функции (потеряв, конечно, возможности синхронизации), если соединить на каждой стороне между собой выводы RTS и CTS, а также выводы DSR, DCD и DTR.

Для нормальной совместной работы приемника и передатчика выводы RxD и TxD, естественно, нужно соединять накрест — TxD одного устройства с RxD второго, и наоборот (то же относится и к RTS, CTS и т. д.). Кабели RS-232, которые устроены именно таким образом, называются еще нуль-модемными (в отличие от простых удлинительных). Их стандартная конфигурация показана на рис. 18.1. В варианте рис. 18.1, *в* дополнительные выводы подключены именно так, как описано ранее, для возможности соединения устройств с «рукопожатием».

Выходные линии RTS и DTR иногда могут служить и для «незаконных» целей — питания устройств, подсоединенных к COM-порту. Именно так устроены, например, компьютерные мыши, работающие через COM. Разумеется,

для работы такого устройства требуется установить эти линии в нужное состояние. Тех, кто интересуется этим вопросом, я отсылаю к моей книге [11], а здесь мы на этом не будем останавливаться, т. к. сейчас подобных устройств никто не проектирует. Мы займемся более актуальным вопросом: как обеспечить преобразование уровней UART в уровни RS-232 со стороны микроконтроллерного устройства?

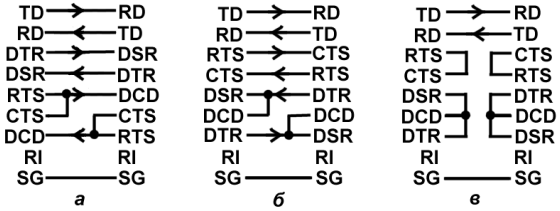


Рис. 18.1. Схемы нуль-модемных кабелей RS-232:
а, *б* — различные полные варианты, *в* — минимальный вариант

Преобразователи уровней UART в уровни RS-232

Простейшая схема преобразователя уровня показана на рис. 18.2. В ней мы учли отмеченный ранее факт, что линия TxD со стороны компьютера бóльшую часть времени пребывает в состоянии низкого уровня, и мы запасаем это напряжение на конденсаторе через диод, а потом расходуем его при передаче. Это несколько снижает входное сопротивление линии RxD устройства (и повышает выходное сопротивление TxD), но в принципе прекрасно работает, даже если байты идут туда-сюда сплошным потоком.

«Официальный» путь состоит в том, чтобы применять специальные микросхемы приемопередатчиков RS-232 (правильнее их было бы называть преобразователями уровня), это, например, MAX202, MAX232, ADM202 и подобные, которые содержат внутри преобразователь — инвертор напряжения, подобный тому, что мы применяли для питания измерителя в автономном режиме (см. *главу 17*). Вариант построения такой схемы показан на рис. 18.3. Выходные уровни вывода TxD здесь при интенсивном обмене составят не менее ±7 В.

Одной из этих схем следует дополнить наш измеритель температуры и давления, чтобы получить возможность соединения его с ПК. Если выбран разъем DB-9M (штыревая часть, как на самом ПК), спроектированный для установки на плату (т. е. типа DRB), то вы сможете соединить ваше устройство

с компьютером только одним способом: с помощью симметричного нульмодемного кабеля, который имеет на обоих концах гнездовые части. Удлинительный кабель RS-232, в котором линии передачи не перекрещиваются, имеет на одном из концов гнездовую, на другом — штыревую часть, и с его помощью подсоединить компьютер не удастся. Можно, конечно, спроектировать устройство в расчете на удлинительный кабель (тогда надо поставить разъем DRB-9F и поменять местами выводы RxD и TxD).

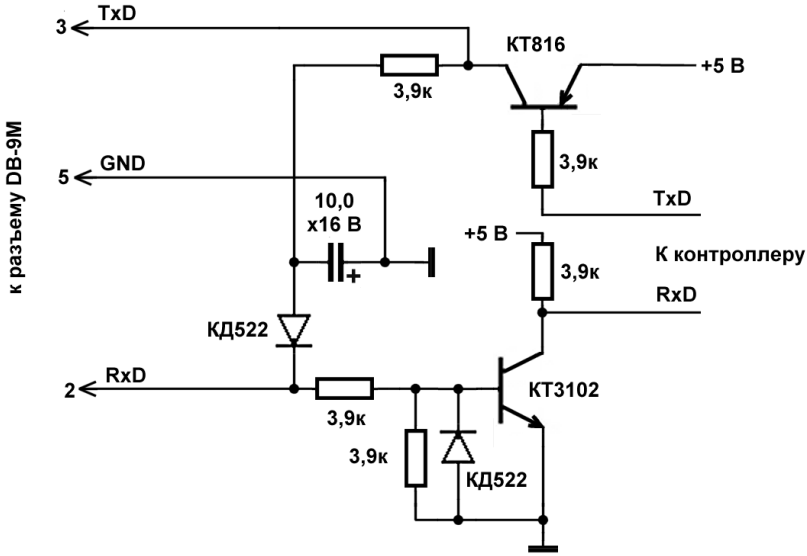


Рис. 18.2. Простейший вариант самодельного преобразователя уровней RS-232 — UART при соединении контроллера с компьютером

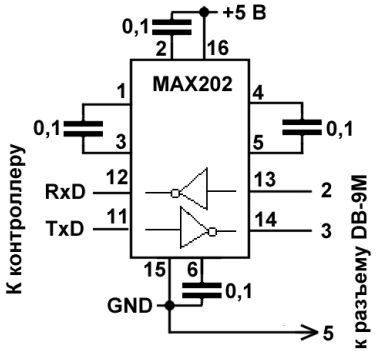


Рис. 18.3. Вариант одноканального преобразователя уровней RS-232 — UART на микросхеме MAX202

Применение таких приемопередатчиков не решает одной проблемы — гальванической развязки устройства с COM-портом. А это очень даже может понадобиться, поскольку на корпусе компьютера «висит» обычно вполне приличный потенциал.

Заметки на полях

По этой причине, кстати, нужно внешнее металлическое обрамление разъемов DB-9 соединять с «землей» и со стороны компьютера, и со стороны прибора — оно первое входит в соприкосновение и потенциалы выравниваются до того, как успевают соприкоснуться контакты разъема. Заставлять пользователей подключать устройства исключительно при выключенном компьютере — «прошлый век».

Автор этих строк однажды чуть не убил одно несчастное животное (до сих пор в кошмарах вспоминается), когда проектировал прибор для измерения внутричерепного давления у обезьян. Главная причина всей этой «катавасии» в отсутствии, разумеется, нормального заземления в наших постройках, но даже при его наличии развязка все равно не помешает.

Один из вариантов такой развязки, реализованный на относительно быстродействующем оптроне типа 6N139, показан на рис. 18.4. Верхняя часть схемы (оптрон D1) служит для передачи сигналов от контроллера к компьютеру. Сигнал TxD с контроллера должен иметь положительный уровень не ниже 4,5 В под нагрузкой, в противном случае следует увеличить номинал резистора R1.

Приемная часть построена на оптроне D2. Ток через входной светодиод оптрона идет во время положительного уровня напряжения на линии TxD COM-порта, а диод VD3 защищает этот светодиод от обратного напряжения.

Здесь питание той части схемы, которая подает сигнал к компьютеру, обеспечивается преобразователем напряжения TMA0505D фирмы TRACO, имеющим гальваническую развязку между входом и выходом. Он обеспечивает преобразование положительного напряжения +5 В в два напряжения ±5 В.

Схема спроектирована в расчете на то, что оптоизолятор выполнен в форме удлинительного кабеля для COM-порта. Разъем X2 типа DRB и остальные детали, кроме разъема X1, монтируют на макетной плате размерами 30×60 мм. С противоположной от разъема стороны распаивают трехжильный плоский кабель примерно 0,5 м длиной, закрепляют его на плате и соединяют с разъемом X1. Разъем X1 может быть не только DB, а любого удобного типа, например, IDC. После проверки плату затягивают в отрезок термоусадочного кембрика подходящего диаметра. Разъем X2 вместе с платой соединяется прямо с COM-портом, ответная часть разъема X1 располагается на плате устройства.

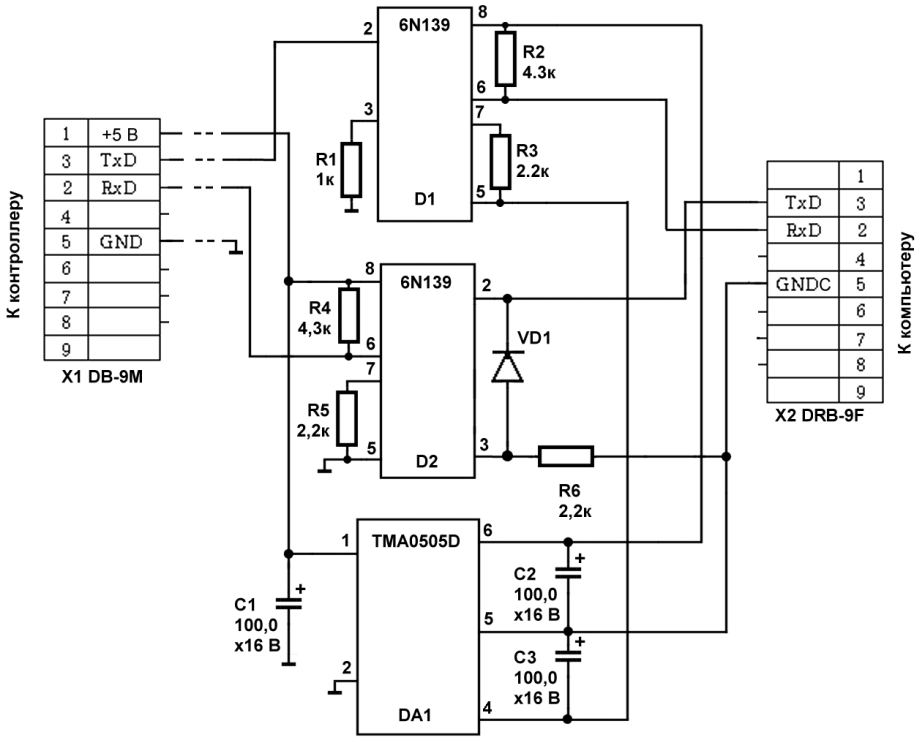


Рис. 18.4. Вариант одноканального преобразователя уровней RS-232 — UART с гальванической развязкой

Для того чтобы вывести напряжение +5 В, пришлось тянуть отдельную линию (контакт 1 разъема X1). Поэтому, если вы собираетесь подключать ПК к компьютеру таким способом, вам придется использовать только этот кабель. В большинстве случаев вместо этого проще будет установить всю схему на плату устройства. Тогда, естественно, разъем X1 следует исключить, но учтите, что у нас в схеме линии RxD и TxD уже перекрещиваются и соединять придется не нуль-модемным, а удлинительным кабелем (что и обеспечивает разъем DRB-9F). Если есть желание унифицировать соединение (нуль-модемные кабели более распространены, чем удлинительные), то следует взять разъем DRB-9M, а линии RxD и TxD в нем поменять местами.

Подключение через USB

Еще не так давно считалось, что протокол обмена данными по USB настолько сложен, что его реализовать под силу только далеко не рядовым специалистам. Но спрос рождает предложение. Вероятно, самое удобное на сегодняшний

день решение для эмуляции COM-порта через USB предлагает английская (точнее шотландская) фирма Future Technology Devices Intl. Limited — FTDI. Читатель, несомненно, в курсе того, что подобные готовые «переходники» COM/USB имеются в продаже вместе с уже настроенным драйвером, но мы немного остановимся на том, как спроектировать такой интерфейс самому и как с ним обращаться.

Наилучший для практики способ построения последовательного порта через USB-интерфейс — на микросхеме FT232BM[1]. С возможностями ее и других USB-микросхем этой фирмы можно ознакомиться из хорошей подборки статей на сайте компании «ЭФО» [12]. Самое главное преимущество этой микросхемы — наличие драйверов для Windows (притом бесплатно распространяемых), которые обеспечат, в том числе, полную эмуляцию последовательного COM-порта со скоростями до 1 Мбит в секунду. На рис. 18.5 без лишних объяснений я привожу схему устройства сопряжения USB/RS-232, которая без изменений честно заимствована из фирменной документации FTDI [13]. В русифицированном варианте эта схема приведена в [14], где она также заимствована из размещенной на сайте екатеринбургской фирмы «Институт радиотехники» (**http://www.institute-rt.ru/common/statyi/conv1/index.html**) статьи А. Лысенко, Р. Назмутдинова, И. Малыгина в журнале «Радио» (2002, № 6 и 7).

Согласно уверениям производителя, если вы просто припаяете микросхему FT232BM без дополнительного программирования внешней EEPROM (микросхема 93C46 на схеме), в которой должны храниться идентификаторы устройства и прочая служебная информация, и даже вообще без нее, то устройство все равно будет работать, хотя могут возникнуть сложности с подключением других подобных устройств. Если же есть желание EEPROM запрограммировать, то специально этим заниматься не требуется, при установке драйвера типа D2XX (как указано далее) это можно сделать прямо на готовой плате через специальную фирменную утилиту EditEEPROM. Есть, по слухам, некоторые особенности с обеспечением скоростного режима этих микросхем, но вдаваться в подробности в рамках этой книги не имеет смысла.

Имейте в виду, что максимальную скорость обмена здесь ограничивает не интерфейс, а применяемые компоненты. Так, в схеме по рис. 18.5 преобразователь MAX213 или ADM213 могут обеспечить 115 кбод, микросхема SP213 — 500 кбод, а 1 Мбод вы получите только при выборе MAX3245. Правда, при этом встанет необходимость еще и запрограммировать как UART, так и виртуальный COM-порт на такие скорости. На самом деле схема, приведенная

[1] Имейте в виду, что подключение снятой с производства FT232AM (ее полное название FT8U232AM) несколько отличается.

на рис. 18.5, целесообразна только для устройств с уже готовым интерфейсом RS-232. Если вы устройство целиком проектируете самостоятельно, то нет никакого смысла преобразовывать уровни UART в уровни RS-232 и обратно, дважды устанавливая приемопередатчик — в этом случае его из схемы на рис. 18.5 нужно исключить, а вместо него линии RxD и TxD подсоединить прямо к контроллеру. Остальные линии можно не подключать, вывод CTS# микросхемы FT232BM при этом следует заземлить.

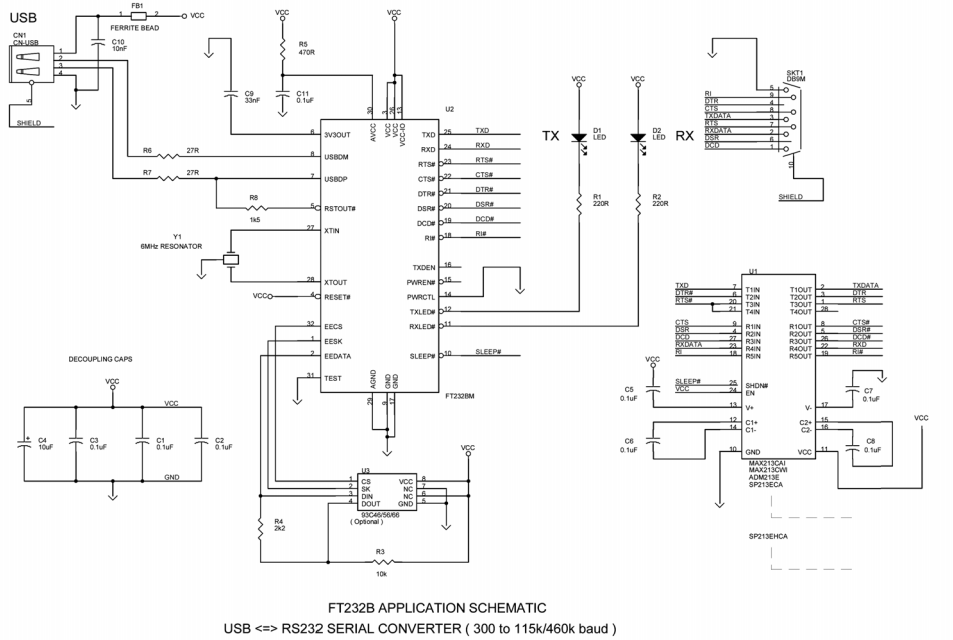


Рис. 18.5. Рекомендуемая производителем схема преобразователя USB — RS-232 с использованием микросхемы FT232BM

Как я уже говорил, к устройствам FTDI прилагаются бесплатные и свободно распространяемые драйверы под все основные ОС, в том числе и под Windows семейств NT и 9*x*. Разновидностей таких драйверов две — это VCP и D2XX-драйверы.

VCP означает Virtual Communication Port, этот драйвер просто-напросто транслирует все стандартные функции Win32 API, которые мы будем использовать далее, в необходимые команды для USB, и через микросхему FT232BM или аналогичные ей, находящиеся в устройстве, опять преобразует их в битовые последовательности UART. При подключении устройства

в системе возникает новый виртуальный COM-порт, и все описанные далее программы без переделок будут работать через него. Это касается и случая, когда у вас покупной «шнурок» USB/RS-232, и когда сами «изобретаете» интерфейс по схеме рис. 18.5. Мало того, в Windows XP такой драйвер уже встроен, и там ничего вообще устанавливать не надо. Единственная разница между таким подключением и обычным в том, что виртуальный COM, в отличие от реального, не будет виден в системе, пока вы свое устройство не подключите к USB.

Один вопрос, впрочем, может возникнуть в случае применения VCP-драйверов, и касается он упомянутых скорости передачи — если уж USB, то хочется реализовать хоть малую долю возможностей этой шины. Между тем стандартно COM-порт может иметь скорость максимум в 256 кбод. Есть приемы установки для COM-порта нестандартной скорости обмена, но вопрос о том, насколько это корректно для данного случая, лично для меня, ввиду новизны ситуации, пока открыт (официальная документация не слишком уверенно утверждает, что допустимы только стандартные скорости). Второй — менее серьезный — недостаток драйверов VCP заключается в том, что вы не можете через него работать со встроенной в FT-устройство EEPROM, в которой записан идентификатор устройства и прочие необходимые для USB «прибамбасы».

В этих случаях следует выбрать D2XX-драйвер, работа с которым отличается, и похожа на работу с разобранным далее компонентом AfComPort. Программы, конечно, придется переписывать. С установкой D2XX-драйвера связана одна совершенно «идиотская» трудность (так и хочется написать — «характерная для продуктов Microsoft») — как указывалось ранее, в XP VCP-драйвер уже есть, и для установки D2XX-драйвера нужно устраивать некоторые «пляски с бубном», цитирую из русского переложения фирменных рекомендаций по этому поводу (**http://www.efo.ru/doc/Ftdi/Ftdi.pl?1046**):

«Немного сложнее обстоит дело в случае операционной системы Windows XP, которая уже имеет в своем составе сертифицированные VCP-драйверы FTDI. При попытках присоединить к компьютеру новое USB-устройство со стандартными идентификаторами FTDI (например, любой DLP-модуль) система по умолчанию, не спрашивая пользователя, самостоятельно установит VCP-драйверы. Пользователю, желающему работать с D2XX-драйверами, необходимо в этот момент вспомнить, что очень полезно воспитывать в себе терпение и воспользоваться утилитой ftxprcvr.exe, входящей в состав дистрибутива D2XX-драйверов для Windows XP. Утилита ftxprcvr.exe, используя установившиеся по умолчанию VCP-драйверы, перепрограммирует внешнюю EEPROM в присоединенном устройстве и задаст

новые значения идентификаторов (VID=0403 и PID=6006). После этого необходимо повторить процедуру установки D2XX-драйверов с начала, т. е. отключить и снова присоединить устройство. Теперь система даст возможность пользователю указать директорию для установки D2XX-драйверов.»

Перепрограммировать EEPROM в устройстве USB на основе микросхем FT232BM через свою программу не обязательно, как уже упоминалось, для этого есть фирменная утилита EditEEPROM (**http://www.efo.ru/doc/Ftdi/Ftdi.pl?798**). Приводить примеры программ для работы с D2XX-драйвером я не вижу никакого смысла, т. к. фирменная документация уже не однажды переписана и переведена на русский, см., например, цитированную ранее статью или книгу [14]. На сайте екатеринбургской фирмы «Институт радиотехники» есть, в том числе, работающий пример проекта на Delphi (**http://www.institute-rt.ru/ftdi/d2xxappl.zip**). А вот сами драйверы с этого сайта скачивать не следует, на момент написания этих строк там лежит устаревший вариант, лучше обратиться к первоисточнику (**http://www.ftdichip.com**) или на упоминавшийся сайт компании «ЭФО» (**http://www.efo.ru/doc/Ftdi/Ftdi.pl?784**).

На этом мы с вопросом эмуляции COM через USB закончим и обратимся непосредственно к программам, которые взаимодействуют с COM-портом, неважно настоящим или эмулированным, но через стандартные функции API.

Программа COM2000

Обещанная универсальная программа для доступа к микроконтроллерным устройствам через COM-порт под названием COM2000 находится на моей домашней страничке по адресу: **http://revich.lib.ru/comcom.zip**. Устанавливать ничего не требуется, просто распакуйте содержащий два файла архив в любую папку. Сама программа содержится в файле com2000.exe. Файл помощи help2000.htm можно открыть как изнутри программы (через меню со знаком вопроса или клавишей <F1>), так и обычным способом в браузере, что удобнее. Собственно, в этом файле все рассказано, здесь я только немного подробнее опишу основные возможности программы.

На рис. 18.6 представлено основное и единственное окно программы COM2000. Когда-то подобные программы называли *эмуляторами терминала* (сейчас это название забыто вместе с самим понятием «терминалы»). Основная их функциональность заключается в постоянном ожидании приема данных по заданному порту с заданной скоростью (на рис. 18.6 установлен порт COM1 и скорость 9600, см. статусную строку внизу). Принятые данные побайтно выводятся на экран, причем отображение их может осуществляться

тремя различными способами (в соответствии с выбором из показанного на рис. 18.6 меню в пункте **Receive**): в шестнадцатеричной форме, в десятичной и в виде текстового символа, соответствующего значению принятого байта. К последней возможности нужно относиться с осторожностью — Windows не «любит» встречать в текстовых компонентах несуществующие символы (вроде символа с номером 0) и программа может «рухнуть». Так что текстовый режим следует выбирать только, если вы ожидаете именно текст.

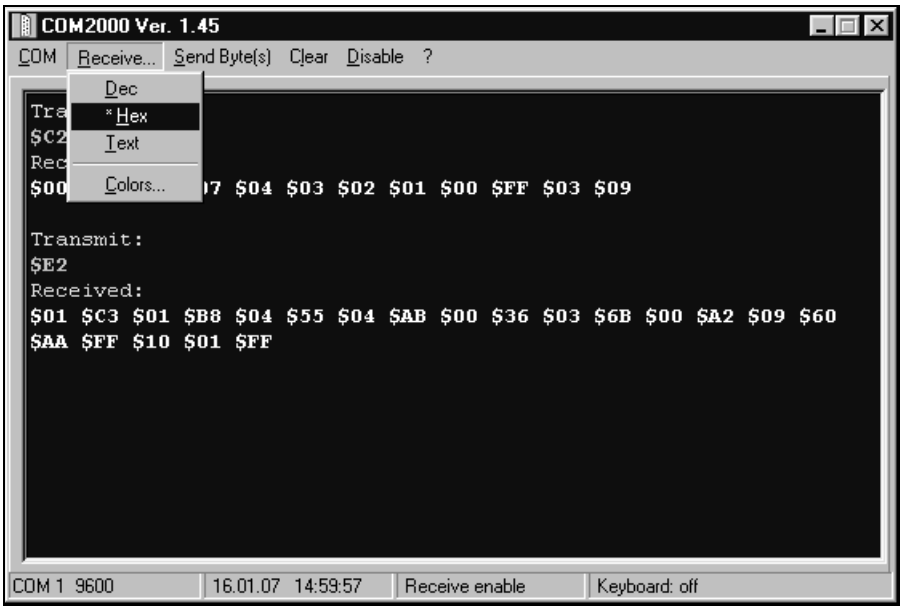


Рис. 18.6. Программа COM2000

На рис. 18.6 показан пример приема байтов в шестнадцатеричной форме в ответ на посланные команды (во втором случае, видном на экране полностью, это команда $E2). Посылать команды можно выбором из меню **Send Byte(s)** также одной из трех возможностей: с клавиатуры (пункт **Keyboard**, <Ctrl>+<K>), непосредственным вводом значений (**Value**, <Ctrl>+<V>) или из файла (**From file**, <Ctrl>+<F>). Посылка с клавиатуры означает то же, что и прием в текстовой форме: при нажатии буквенной клавиши посылается ее код в виде байта с соответствующим значением. В Windows с ее путаницей в отношении виртуальных кодов клавиш эта возможность почти потеряла значение, но до сих пор встречаются устройства, в инструкции к которым команды записаны именно в виде символов (а не их номеров в таблице ASCII). Для совместимости с этими устройствами и сохранена такая возможность.

Если вы выбрали ввод с клавиатуры, то внизу в статусной строке надпись Keyboard Off сменится надписью Keyboard On. Не забудьте обратиться к пункту меню **Keyboard** или нажать <Ctrl>+<K> еще раз, чтобы выключить отсылку символов с клавиатуры после ввода, иначе они будут отсылаться и дальше при любом нажатии клавиш.

В обычном режиме используются две другие возможности, в основном вторая — посылка байтов с конкретным значением. При обращении к меню **Send Byte(s) | Value** (<Ctrl>+<V>) вы вызовете на экран однострочный редактор, в поле которого можно ввести нужное значение байтов, причем сразу много (до 32). Байты можно вводить в десятичном или шестнадцатеричном виде (с предваряющим знаком $) вперемешку, разделяя их пробелами. В выпадающем списке редактора запоминаются ранее отосланные вами строки (в том числе там есть несколько значений по умолчанию, для образца). После ввода значений нужно либо нажать на <Enter>, либо дважды щелкнуть мышью в окне редактора с введенной строкой значений. Обратите внимание, что проверка значения не производится, и при превышении диапазона посылаемый байт усекается до 8 разрядов, например значение 257 будет послано, как 257 – 256 = 1. Проверяется только корректность записи, например, при попытке послать 0A без предваряющего $, вам будет выдано сообщение об ошибке.

Аналогично осуществляется посылка из файла, которая хороша тогда, когда нужно послать много байтов, и вводить их в однострочный редактор неудобно. Тогда следует создать текстовый файл, в котором содержится строка со значениями, составленная по точно таким же правилам, что действуют для непосредственной посылки, и выбрать этот файл через меню **Send Byte(s) | From file** (<Ctrl>+<F>).

Заметим, что непрерывный прием можно отключить, если выбрать пункт меню **Disable** (он изменится на **Enable** и для включения его следует нажать еще раз). Это полезно, когда устройство (вроде GPS-навигатора) выдает данные непрерывно, и не хочется «забивать» экран ненужными данными. Только будьте внимательны: если режим непрерывного приема отключен, вы можете забыть об этом и подумать, что прибор внезапно перестал работать. Пункт меню **Clear** предназначен для очистки экрана.

Важная особенность программы — непрерывное ведение log-файла, который создается при первом запуске и далее только дополняется. В него записывается все, что отображается на экране, плюс при каждом запуске программы пишется еще текущая дата и время. log-файл полезен, если вы хотите сохранить принятые данные. Со временем он увеличивается до «неподъемных» размеров, и чтобы удалить ненужные данные, просто сотрите com.log, и он создастся заново при следующем запуске.

В программе можно, естественно, выбирать COM-порт (от COM1 до COM4) и скорость обмена (пункт меню **COM**). Кроме этого, можно менять оформление программы (цвет фона и надписей) через пункт меню **Receive | Colors**. Оформление и заданные режимы запоминаются к следующему сеансу.

Программа COM2000 очень удобна для отладки, о чем рассказывалось в разделе *главы 16* «Отладка программ с помощью UART». Вы соединяете компьютер через один COM-порт с программатором, в свою очередь, соединенным со схемой, а через второй — с выходом UART схемы. Держа на экране открытыми одновременно три окна (редактор, программу для загрузки через программатор и COM2000), вы получаете возможность почти в реальном времени править программу и немедленно проверять ее работоспособность, временно расставляя контрольные вызовы функции `out_com` в нужных местах.

Работа с COM-портом в Delphi

В подавляющем большинстве случаев для создания даже таких относительно «навороченных» программ, как COM2000, можно обойтись стандартным Турбо Паскалем под DOS. Это было бы целесообразно, однако пользование DOS-программами на современном ПК крайне неудобно и порождает множество проблем совместимости (не говоря уж о многозадачности — конечно, работать в режиме немедленной проверки в DOS далеко не так удобно, как в Windows). Потому мы остановимся на Delphi, как одном из самых популярных в нашей стране средств быстрого создания приложений (RAD, Rapid Application Development). Можно использовать и Visual Basic (в некоторых отношениях это даже более удобно, т. к. в силу происхождения он лучше интегрирован с Windows), однако мой личный опыт общения с этим продуктом отрицательный — значительную часть времени приходится тратить на то, чтобы бороться с внутренними проблемами самого VB. Язык Delphi (Object Pascal) проще и понятнее новичку. Для тех, кто привык к C, можно рекомендовать Borland C++ Builder — это почти то же самое, что и Delphi, даже с общей библиотекой компонентов, только язык другой.

Мы будем ориентироваться на Delphi 7 — последнюю версию для Win32. Программы работают без оговорок во всех версиях Windows (начиная с 98-й). Насколько программы для этой платформы работоспособны в Vista — на момент написания этих строк мне еще не удалось проверить, но, по слухам, все более-менее в порядке (правда, установление этого *порядка* все требует некоторых «плясок с бубном»), а на крайний случай там имеется режим совместимости с более ранними версиями, который работает, говорят, лучше, чем в Windows XP. Потому, надеюсь, с изучением платформы .NET можно повременить.

В дальнейшем я буду предполагать, что читатель имеет некоторые навыки работы в Delphi, поскольку рассмотрение данного вопроса выходит за рамки этой книги. Остальным я рекомендую обратиться к [15] и [16], а также к моей книге [11].

Работа через функции Win32 API

Собственно передача и прием данных через COM-порт неоднократно описаны во множестве публикаций и теоретически в них ничего сложного нет. На практике, однако, могут возникнуть проблемы различного уровня сложности, особенно касающиеся непрерывного приема данных в реальном времени. Далее я описываю только проверенные способы работы (многие публикации в Интернете содержат ошибки, и к тому же не описывают ситуацию полностью).

Начнем мы с самого главного — с отдельной процедуры инициализации порта, которую назовем `IniCOM`. COM-порт представляется с точки зрения системы в виде файла, потому его сначала нужно открыть («создать файл»). Но этого недостаточно — необходимо проверить его работоспособность, и факт, не является ли запрошенный COM-порт модемом.

Объявим следующие переменные:

```
var
  Form1: TForm1;
  hCOM: hFile=0;
  pDCB: TDCB;
  comtime: TCOMMTIMEOUTS;
  xb: byte;
  xn: dword;
  ab: array[1..32768] of byte;
  st,stcom: string;
  ttime,told: TDateTime;
. . . . .
```

Размер буфера `ab` может быть произвольным в зависимости от количества ожидаемых данных (в данном случае он подогнан для чтения данных из используемой нами внешней flash-памяти). К началу выполнения процедуры `IniCOM` у нас в строке `stcom` должно содержаться название порта, например, «COM1». Задаваемые параметры: прием по схеме 8n1, скорость 9600. Текст процедуры приведен в листинге 18.1.

Листинг 18.1

```
procedure IniCOM;
var i:integer;
begin
```

```
{инициализация COM – номер в строке stcom}
  hCOM:=CreateFile(Pchar(stcom),
  GENERIC_READ+GENERIC_WRITE,0,nil,OPEN_EXISTING,0,0);
  if (hCom = INVALID_HANDLE_VALUE) then
  begin
    st:=stcom+' не найден';
    Application.MessageBox(Pchar(st),'Error',MB_OK);
    exit;
  end;
  if GetCommState(hCOM,pDCB)
  then  st:=stcom+': baud=9600 parity=N data=8 stop=1';
  if BuildCommDCB(Pchar(st),pDCB) then SetCommState(hCOM,pDCB)
  else
  begin
    st:=stcom+' занят или заданы неверные параметры';
    Application.MessageBox(Pchar(st),'Error',MB_OK);
    exit;
  end;
  GetCommTimeouts(hCom,comtime); {устанавливаем задержки:}
  comtime.WriteTotalTimeoutMultiplier:=1;
  comtime.WriteTotalTimeoutConstant:=10;
  comtime.ReadIntervalTimeout:=10;
  comtime.ReadTotalTimeoutMultiplier:=1;
  comtime.ReadTotalTimeoutConstant:=2000; {ждем чтения 2 с}
  SetCommTimeouts(hCom,comtime);
  ab[1]:=ord('A');  {будем посылать инициализацию модема}
  ab[2]:=ord('T');
  ab[3]:=13;{CR}
  ab[4]:=10;{LF}
  WriteFile(hCOM,ab,4,xn,nil);
  if ReadFile(hCOM,ab,10,xn,nil) then {ответ модема 10 знаков}
  begin
     st:=»;
     for i:=1 to 10 do st:=st+chr(ab[i]);
     if pos('OK',st)<>0 then
     begin
       st:=stcom+' занят модемом';
       Application.MessageBox(Pchar(st),'Error',MB_OK);
       CloseHandle(hCOM);
       hCOM:=0;
       Form1.Label7.Caption:='COM?';
       exit;
     end;
  end;
  Form1.Label7.Caption:=stcom+' 9600';
end;
```

Сначала текст тут мало отличается от того, что описано во всех стандартных рекомендациях по программированию порта. Единственный момент, который несколько выходит за рамки стандарта: мы не устанавливаем поля структуры `DCB` напрямую, а используем функцию `BuildCommDCB`. Я это делаю отчасти потому, что структура `DCB` в Delphi транслируется из API не полностью (сравните ее описание в Windows.pas и в Win32.hlp), и хотя для данного случая, разумеется, все нужные поля имеются, но функция `BuildCommDCB` все равно удобнее.

После стандартных установок мы сразу выполняем два действия, о которых упоминают далеко не всегда. Во-первых, мы устанавливаем все возможные задержки (timeout) для разных вариантов приема и передачи. В параметрах, которые заканчиваются на «Multiplier» можно для простоты всегда ставить «1» (если больше, то процедуры чтения/записи будут отслеживать еще и скорость поступления байтов, что нам не требуется). А остальные из этих параметров делают следующее: если задержка посылки через порт больше, чем `WriteTotalTimeoutConstant` (в миллисекундах), то будет прервана передача, а при задержке между поступающими байтами больше, чем `ReadIntervalTimeout`, и при задержке всей процедуры чтения (в данном случае — самый главный параметр) больше, чем `ReadTotalTimeoutConstant`, будет прерван прием. Последний параметр мы установили равным двум секундам. При выборе этих параметров следует иметь в виду, что один байт при скорости 9600 передается/принимается примерно за 1 мс. Если эти параметры вообще не устанавливать (оставить их в значении «0», как по умолчанию), то при отсутствии принимаемых байтов процедура чтения через **ReadFile** просто зациклится и «повесит» всю программу.

Во-вторых, мы определяем, не является ли установленный нами порт модемом. Так как мы рассматриваем «чистый» RS-232, то для нас модемный порт все равно как бы занят. Определяем модем мы очень просто: посылаем в выбранный порт символьный код инициализации, который одинаков для всех модемов: «AT<CR><LF>» (65 84 13 10). В ответ мы от модема должны получить строку «AT<CR><LF><CR><LF>OK<CR><LF>» (65 84 13 10 13 10 79 75 13 10), но все такие подробности нам не требуются, подозреваю, что строка для разных модемов может немного отличаться. Но в любом случае в ней должны содержаться символы «OK», если модем, конечно, свободен (с занятым модемом я предоставляю читателю разобраться самостоятельно). В последнем операторе, если порт инициализировался нормально, выводим в `Label` номер порта и скорость.

Эту процедуру мы выполним сразу при запуске (для COM1). Заодно напишем процедуру (листинг 18.2) закрытия порта (ведь порт все время занят, пока программа запущена).

Листинг 18.2

```
procedure TForm1.FormCreate(Sender: TObject);
begin
{инициализация COM1 при запуске}
  stcom:='COM1';
  IniCOM;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin {уничтожаем COM}
    CloseHandle(hCOM);
end;
```

Кроме этого, конечно, следует создать меню для выбора порта по ходу работы с программой (аналогичное меню **COM** в моей программе COM2000), но мы не будем здесь на этом останавливаться, т. к. на основе приведенной процедуры его создать несложно.

Итак, порт открыт, инициализирован с нужными параметрами — что дальше? Далее все очень непросто, потому что мы здесь не можем, как в DOS, зациклить программу в ожидании поступившего байта или нажатия клавиши. Windows и сама представляет собой бесконечный цикл в ожидании сообщений (событий) — в точности такой же, как мы организовывали у себя в контроллере в ожидании прерываний. И вот этими сообщениями мы и должны ее снабдить.

Сначала разберем простейший способ обмена данными: когда мы точно знаем, что нам придет в ответ и в каком количестве. Например, посмотрим, как можно организовать процедуру приема значения часов в ответ на команду `$A2` (описание того, как это работает в контроллере, см. *главу 16*). Расставим на форме следующие компоненты: кнопку **Button1**, а также **Label** и напротив него **StaticText**, и то и другое в количестве 6 штук. В компоненты `Label` запишем, соответственно, по порядку: «Часы», «Минуты», «Секунды», «Дата», «Месяц», «Год» (в таком же порядке их выдает наш контроллер). Листинг 18.3 иллюстрирует, как будет выглядеть сама процедура, когда по нажатию кнопки **Button1** в МК выдается команда `$A2`, а потом принятые значения выводятся в **Static Text** (т. к. они в BCD-формате, то приходится переводить в HEX-представление).

Листинг 18.3

```
procedure TForm1.Button1Click(Sender: TObject);
begin    {запрос}
  if (hCOM=0) or (hCOM=INVALID_HANDLE_VALUE) then exit;
```

```
                    {если порт еще не инициализирован — выход}
  PurgeComm(hCOM,PURGE_RXCLEAR); {очищаем буфер}
  xb:=$A2;
  WriteFile(hCOM,xb,1,xn,nil);
  told:=Time;
  if ReadFile(hCOM,ab,6,xn,nil) then {читаем 6 байтов в массив ab}
  begin
    ttime:=Time;
    if SecondsBetween(told,ttime)>0 then
    begin
      Application.MessageBox('Устройство не
                          обнаружено','Error',MB_OK);
      exit;
    end;
    if xn<>6 then
    begin
      Application.MessageBox('Неправильный формат
                              данных','Error',MB_OK);
      exit;
    end;
    StaticText1.Caption:=IntToHex(ab[1],2); //часы
    StaticText2.Caption:=IntToHex(ab[2],2); //минуты
    StaticText3.Caption:=IntToHex(ab[3],2); //секунды
    StaticText4.Caption:=IntToHex(ab[4],2); //дата
    StaticText5.Caption:=IntToHex(ab[5],2); //месяц
    StaticText6.Caption:=IntToHex(ab[6],2); //год
  end else {если не сработало}
  begin
    Application.MessageBox('COM сломался','Error',MB_OK);
    exit;
  end;
end;
```

Результат будет выглядеть примерно так, как показано на рис. 18.7.

Здесь процедура `PurgeComm` нужна для очистки приемного буфера, на случай, если там случайно задержались какие-то байты (те, кто работал с COM-портом в DOS, часто об этом забывают, т. к. там никаких буферов не было). Второй момент, который нужно прокомментировать, связан с возможным отсутствием нужного устройства на втором конце линии (или его неработоспособностью — включить забыли). Мы здесь, как видите, все делаем очень просто: читаем системное время до и после вызова функции `ReadFile`, и выясняем — если прошло более 1 с (а timeout у нас задан в 2 с), то «устройство

не обнаружено». Опыт показывает, что это самый надежный метод. Если же связь каким-то образом прервется посередине посылки, то мы получим меньше байтов, чем заказывали, и программа выдаст сообщение «Неправильный формат данных». Не забудьте, что при работе с функциями времени в Delphi надо добавить ссылку на модуль DateUtils.

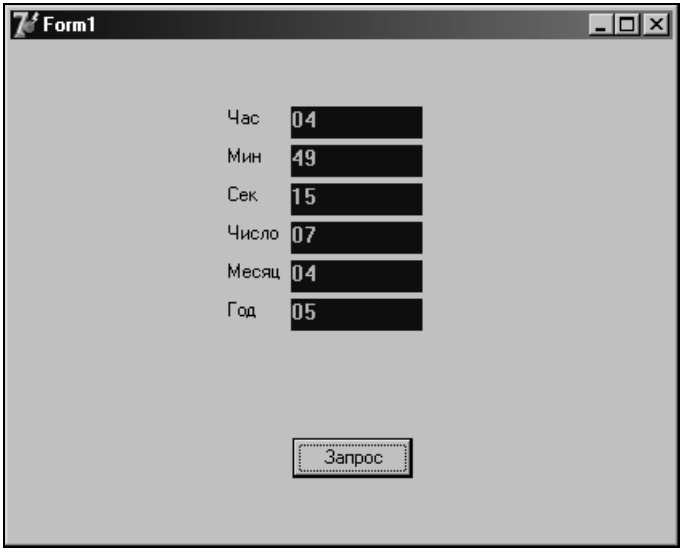


Рис. 18.7. Результат приема значений времени из часов-измерителя

По той же схеме можно организовать взаимодействие с измерителем для всех наших команд, кроме команды чтения данных из flash-памяти. С ней сложности возникнут оттого, что мы точно не знаем, сколько данных нам придет, и когда пора заканчивать. В принципе можно обойтись тем же приемом с проверкой времени, т. к. данные поступают сплошным потоком, и когда поток этот прервется, то можно заканчивать. Организация процедуры тогда в некотором роде напоминает использование сторожевого таймера в МК: мы задаем в процедуре чтения максимальное количество принимаемых байтов за один прием (например, 1024 — столько придет примерно за секунду), и пока данные идут, в цикле принимаем их, «скидываем» временно в какой-нибудь массив (потом будем обрабатывать) и обнуляем разницу между `told` и `ttime`. Если она не обнулилась вовремя и превысила 1 с — прием окончен.

Это требует довольно тонкой организации процесса и вообще не универсально: а как быть, если мы не знаем точно момента прихода данных? Ну, например, пишем программу для взаимодействия с упоминавшимся GPS-навигатором,

который выдает данные (и точно неизвестно, сколько именно) каждые несколько секунд?

Для этой цели можно организовать прием данных в отдельном потоке. Мы здесь не будем разбирать этот способ, потому что из-за «заумности» соответствующих функций API самое простое, что там приходится делать, — это создавать сам параллельный поток (по сути очень простое действие, которое почему-то вызывает дрожь у новичков, в значительной степени из-за неудобства этого процесса в Delphi). А вот собственно прием данных и их передача в основную программу там получается неоправданно сложной и запутанной процедурой. Потому я не буду на этом останавливаться (интересующихся подробностями отсылаю опять же к книге [11]), а сразу покажу, как можно создать подобную программу на основе дополнительного стороннего компонента, специально созданного для приема через COM-порт. Именно такой способ используется в программе COM2000.

Использование драйвера AsyncFree

Компонентов для работы с COM-портом довольно много, есть среди них платные и бесплатные. Мы будем использовать один из самых удачных и профессионально сделанных компонентов для COM-порта — свободно распространяемый AsyncFree некоего Петра Вониса (Petr Vones), судя по электронному адресу из Чехии. Компонент доступен бесплатно, с исходными кодами, и скачанный архив включает в себя, в том числе, и файлы dpk, что упрощает процедуру установки до предела — нужно просто щелкнуть мышью на том из dpk-файлов, который соответствует имеющейся у вас версии Delphi, и компонент установится самостоятельно, без утомительных процедур ручной инсталляции. Хотя к самому компоненту приложена ссылка на (отличный, кстати) сайт Delphree Open Source Initiative (**http://delphree.clexpert.com**), однако на нем я нашел только старую версию AsyncFree под Delphi 5, а скачивать последние версии лучше отсюда: **http://sourceforge.net/project/showfiles.php?group_id=20226**.

Принцип работы компонента заключается как раз в создании параллельного потока, в котором байты принимаются по мере их поступления и накапливаются в буфере независимо от деятельности основной программы. Специально следить за приходом данных не требуется, поскольку все делается автоматически, нам остается только отловить принятые данные. Недостаток этого способа в том, что данные принимаются кучей в одной процедуре, и приходится отдельно разбираться, что же мы приняли в данный момент, и где заканчиваются одни данные и начинаются другие. Но при использовании параллельного потока это пришлось бы делать в любом случае, а тут все организовано за вас, «ручками» ничего делать не приходится.

После установки компонент будет находиться в палитре компонентов на вкладке **AsyncFree**. На самом деле там образуется много компонентов, но нам требуется только самый первый из них под названием **AfComPort**. Установим его на форму. В перечень переменных добавим:

```
. . . . .
FlagCOM:boolean=False;
FlagSend:integer=0;
tall:integer;
. . . . .
```

Переменные `told` и `ttime` нам здесь не понадобятся. На форму добавим `Label7`, в который будем выводить установленный порт и скорость передачи, и Timer. Проверьте, чтобы у таймера интервал составлял 1000 мс (так по умолчанию). Кроме этого, установим компонент `ComboBox` (выпадающий список), у которого в свойстве `Items` сразу запишем строки для выбора COM-порта («COM1», «COM2» и т. п.). Аналогичный список можно создать для выбора скорости передачи, но если речь идет о конкретном приборе, то это необязательно (а вот COM выбирать, скорее всего, придется).

Начнем с того, что перепишем процедуру `IniCOM`, что иллюстрирует листинг 18.4 (как и в предыдущем случае, к моменту ее вызова в `stcom` должна находиться строка с номером порта, например, «COM1»).

Листинг 18.4

```
procedure IniCOM;
var i, err :integer;
begin
  FlagCOM:=False;
  Form1.Label7.Caption:='COM?';
{инициализация COM – номер в строке stcom}
  Form1.AfComPort1.Close; {закрываем старый COM, если был}
  val(stcom[length(stcom)],i,err); {извлекаем номер порта}
  if err=0 then  Form1.AfComPort1.ComNumber:=i else exit;
                    {здесь требуется число, а не строка}
  Form1.AfComPort1.BaudRate:=br9600; {скорость 9600}
  try
    Form1.AfComPort1.Open; {пробуем открыть}
  except
   if not Form1.AfComPort1.Active then {если не открылся}
   begin
      st:=stcom+' does not be present or occupied.';
      Application.MessageBox(Pchar(st),'Error',MB_OK);
```

```
      exit {выход из процедуры — неудача}
   end;
  end;
  ab[1]:=ord('A');  {будем посылать инициализацию модема}
  ab[2]:=ord('T');
  ab[3]:=13;{CR}
  ab[4]:=10;{LF}
  for i:=1 to 4 do Form1.AfComPort1.WriteData(ab[i],1);
  {ответ не сразу:}
  Form1.Timer1.Enabled:=True;
  tall:=0;
  while tall<1 do application.ProcessMessages; {пауза в 1 с}
  Form1.Timer1.Enabled:=False;
  st:=Form1.AfComPort1.ReadString; {ответ модема 10 знаков}
  if pos('OK',st)<>0 then  {модем}
  begin
       st:=stcom+' занят модемом';
       Application.MessageBox(Pchar(st),'Error',MB_OK);
       exit;
  end else  {все нормально, COM открыт}
  begin
       Form1.Label7.Caption:=stcom+' 9600';
       FlagCOM:=True;
  end;
end;
```

Как видим, процедура создания порта много понятнее, чем в случае прямого обращения к API — все через привычную установку свойств компонента. `FlagCOM` играет у нас роль индикатора, доступен порт или нет. Если он остался при значении `False`, то процедуру следует повторить с другим значением в строке `stcom` (каковую мы задаем с помощью `ComboBox`, см. далее). При определении модема применен хитрый способ задания паузы — вместо обычного оператора `Sleep`, который тормозит программу, мы использовали таймер. Чтобы это сработало, надо в обработчике события `OnTimer`: все время увеличивать переменную `tall`. Полностью процедура по таймеру приводится далее, т. к. `tall` нам понадобится не только для этого.

Как только мы обратились к процедуре `AfComPort1.Open`, у нас немедленно будет создан параллельный поток и весь прием пойдет через него. Поэтому, чтобы при определении модема принятые байты не обрабатывались, нужно не забыть добавить в процедуру приема выход по условию `FlagCOM=False`.

Для создания этой процедуры обычным способом — через инспектор объектов — создадим обработчик события AfComPort1DataRecived[2] (листинг 18.5).

Листинг 18.5

```
procedure TForm1.AfComPort1DataRecived(Sender: TObject; Count:
Integer);
{чтение очередного байта по сообщению wmCOMPORT}
var i:integer;
begin
  if FlagCOM=False then exit; {если модем еще не опрошен}
  if count<>0 then {если что-то принято}
  begin
    AfComPort1.ReadData(ab,count); {читаем буфер в массив}
    xn:=xn+count; {число принятых байт}
    tall:=0; {обнуляем время}
  end;
end;
```

На самом деле условие count<>0 не требуется, оно введено просто ради порядка (иначе бы процедура просто не была бы вызвана). По выходу из процедуры в переменной xn будет накапливаться количество принятых байтов. Осталось только дописать остальные процедуры (листинг 18.6).

Листинг 18.6

```
procedure TForm1.Button1Click(Sender: TObject);
begin    {запрос}
  if FlagCOM=False then exit;
  {если порт еще не инициализирован — выход}
  AfComPort1.PurgeRX; {очищаем буфер порта на всякий случай}
  xb:=$A2;
  AfComPort1.WriteData(xb,1); {посылаем команду}
  FlagSend:=$A2; {обозначаем посылку запроса времени}
  tall:=0; {обнуляем время}
  xn:=0; {счетчик принятых байтов}
  Timer1.Enabled:=True;  {запускаем таймер}
end;
```

[2] Написание слова «receive» — настоящая проблема для любого, кто не является носителем английского языка. Каких только вариантов не встретишь на просторах Сети! Как видим, наш чех Петя Вонис также не избежал общей участи.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
{инициализация COM1 при запуске}
  stcom:='COM1';
  IniCOM;
end;
procedure TForm1.ComboBox1Select(Sender: TObject);
begin
  stcom:=ComboBox1.Text;  {устанавливаем порт COM1,2,3,4}
  IniCOM;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  AfComPort1.Close; {закрытие порта}
end;
```

Теперь нам осталось разобраться с тем, что мы там напринимали. Это позволит сделать установленное нами значение `FlagSend` и таймер. В таймере переменную `tall` мы будем увеличивать на единицу, а в процедуре приема мы все время ее обнуляем, так что пока она равна нулю, можно полагать, что прием еще не закончился. Как только она станет больше единицы (прошло более секунды с момента последнего принятого байта или прием вообще не происходил), мы начинаем что-то делать, но только в том случае, если флаг `FlagCOM` установлен (`True`), иначе это вообще был не прием, а опрос модема. Сказанное иллюстрирует листинг 18.7.

Листинг 18.7

```
procedure TForm1.Timer1Timer(Sender: TObject);
var i:integer;
begin   {таймер}
  inc tall
  if FlagCOM=False then exit;
  if tall>1 then
  begin
    Timer1.Enabled:=False; {выключаем таймер}
    if xn=0 then {если счетчик = 0, то ничего не принято}
    begin
       Application.MessageBox('Устройство не
                              обнаружено','Error',MB_OK);
       exit {выход из процедуры – неудача}
     end else
     begin {иначе обрабатываем данные}
```

```
      if FlagSend=$A2 then {если был запрос времени}
      begin
        if xn<>6 then
        begin
             Application.MessageBox('Неправильный формат
                              данных','Error',MB_OK);
             exit;
         end;
        StaticText1.Caption:=IntToHex(ab[1],2); //часы
        StaticText2.Caption:=IntToHex(ab[2],2); //минуты
        StaticText3.Caption:=IntToHex(ab[3],2); //секунды
        StaticText4.Caption:=IntToHex(ab[4],2); //дата
        StaticText5.Caption:=IntToHex(ab[5],2); //месяц
        StaticText6.Caption:=IntToHex(ab[6],2); //год
      end;
    end;
  end;
end; {конец таймера}
```

По аналогии вы легко добавите процедуры, соответствующие всем остальным командам, предусмотренным в программе нашего измерителя. Пользуясь функцией `DateTime`, легко создать процедуру, которая будет загружать из компьютера точное время (только с форматом `TDateTime` придется немного попотеть, см. по этому поводу [15] и [16]). Не забывайте принимать и анализировать возвращаемые байты для процедур записи. При длинной процедуре приема данных из flash, когда число байтов заранее неизвестно, суммарное значение счетчика `xn` покажет, сколько именно байтов принято. Причем если это число не кратно четырем, то можно смело утверждать, что целостность данных была нарушена. И не забудьте увеличить размер массива `ab`, если у вас энергонезависимая память большей емкости!

Наряду с такой программой для обслуживания прибора имеет смысл также написать отдельное приложение, которое обрабатывает скачанные данные по температуре и давлению, переводит их в физические величины и представляет их в вид масштабируемого графика, отфильтровывая и записывая отдельно для просмотра байты сбоев с соответствующими датами.

Глава 19

Практические схемы на AVR

> … и телефонный аппарат (клубок катодов, спаек, клемм, сопротивлений) безмолвствует.
>
> *И. Бродский «Посвящается Ялте»*

Возможности МК серии AVR весьма велики, вплоть до того, что на старших моделях можно выстраивать полноценные системы управления, например, жидкокристаллическими дисплеями. Среди продукции Atmel есть контроллер USB на AVR-ядре, и с его помощью можно делать настоящие USB-устройства, не прибегая к эмуляции COM-порта. Но и младшие модели AVR вполне пригодны для серьезных вещей, достаточно упомянуть, что такое устройство, как компьютерная клавиатура, изначально была спроектирована на куда менее мощном, чем почти любой современный AVR, контроллере 8048.

Здесь мы рассмотрим способы построения некоторых типовых узлов с помощью AVR, не задаваясь вопросом конструирования законченных приборов. Довести до ума и встроить в необходимые вам устройства эти узлы вы сможете сами, я лишь покажу, как это делается в принципе. И начнем мы с самого, пожалуй, интересного — попробуем заставить МК воспроизводить цифровой звук.

Заставить камни заговорить

На самом деле способность воспроизводить звук встроена во все МК AVR изначально. Для этого надо лишь иметь «исходник» — ранее записанный звуковой файл с определенными параметрами. Такое устройство можно использовать, как узел голосовой сигнализации — например, если укомплектовать им наш измеритель с часами, и он сможет вслух сообщать время и температуру. Для этого придется сделать немного — разделить в памяти

звуковые сэмплы, произносящие различные слова, и комбинировать их по ходу дела в нужном порядке. Именно так работают системы автоматического оповещения, например, о задолженности по телефонным счетам. Здесь мы в подробности влезать не будем, а покажем, как вообще организовать на AVR режим воспроизведения цифрового звука.

Что такое цифровой звук, вы уже знаете из *главы 10* — это последовательность отсчетов сигнала с определенной частотой (называемой *частотой дискретизации* или *частотой оцифровки*, *битрейтом*) и с определенным разрешением по напряжению *(квантованием)*. Сначала давайте поймем, как в принципе можно воспроизводить такой звук.

Предположим, что мы имеем в качестве источника набор байтов, напрямую представляющий исходную оцифрованную последовательность (сжатые форматы мы не рассматриваем, поскольку это увело бы нас далеко за рамки темы книги). «Лобовой» метод понятен из той же *главы 10*: взять ЦАП, подать ему на вход оцифрованный звук с той же частотой, с которой его оцифровывали, а к его выходу подключить фильтр, усилитель и динамик. Но вот, например, производители сотовых телефонов просто замучили своей «полифонией»: сидя в маршрутке, невольно вздрагиваешь, когда у соседки в сумочке вдруг то ребенок заплачет, то котенок замяукает. Не многие, однако, задавались вопросом — а как это делается? Неужели в мобильник встраивают настоящий цифровой тракт, со всеми этими ЦАПами, фильтрами и усилителями? Вовсе нет. Главная идея, которая заложена в простой реализации воспроизведения цифрового звука, называется «усилитель в режиме D», от слова digital — цифровой.

В усилителях класса D цифровой звук вообще не переводится в аналоговую форму какими-то специальными устройствами. Наоборот, там обычный звук представляется в виде последовательности прямоугольных импульсов, пропорциональных по длительности интенсивности сигнала. Для усиления таких импульсов не нужно никаких ухищрений — чаще всего используют комплементарную пару транзисторов, подобно тому, как усиливается сигнал на выходе логических КМОП-микросхем. Выгода заключается в том, что теоретически КПД такого импульсного усилителя может быть равен 100%, ведь какой-то из транзисторов пары всегда заперт, а второй транзистор в это время полностью открыт и мощности на них не выделяется. Это, конечно, в теории, потому что из *главы 3* вы знаете, что падение напряжения на открытом транзисторе все же имеет место, да и переключение не происходит мгновенно. Но вопросы КПД нас тут не интересуют, т. к. мы не собираемся конструировать 100-ваттные усилители, и указанная схема нас привлекает не столько КПД, сколько простотой и компактностью.

Представление синусоидального сигнала в виде последовательности импульсов различной длительности называется ШИМ-модуляцией (по-английски, PWM — Pulse-Wide Modulation). Фокус заключается в том, что для извлечения исходного синусоидального сигнала из ШИМ не требуется никаких специальных сложных приборов — достаточно обычного резисторно-конденсаторного ФНЧ с подходящей частотой среза (о ФНЧ см. *главу 2*). В результате весь звуковой тракт упрощается до предела (рис. 19.1).

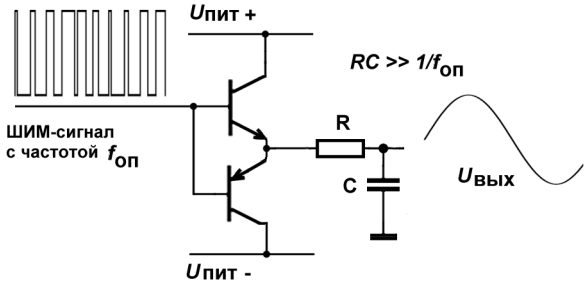


Рис. 19.1. Принцип работы выходной части усилителя в режиме D

Заметки на полях

А как сформировать входной сигнал для такого усилителя, если у нас в наличии имеется лишь аналоговая звуковая волна? Нужно ли ее оцифровывать? Совсем нет: исходный аналоговый сигнал поступает на один вход компаратора, а на второй его вход подается напряжение треугольной формы и подходящей амплитуды. Тогда на выходе компаратора мы получим ШИМ-сигнал. Работа счетчика-таймера, показанная на рис. 19.2 далее, делает в точности то же самое, но в цифровой форме.

Для того чтобы получить ШИМ-сигнал из уже оцифрованного звука, у нас есть такая «штука», как микроконтроллер, причем уже специально приспособленный для подобных целей. Если вы уже имеете книгу [1] или [2], или скачивали фирменный PDF-документ с описанием какого-то из контроллеров AVR, и при этом ваше любопытство зашло столь далеко, что вы эти источники даже немного пролистали, то, несомненно, заметили, что в описаниях таймеров PWM-режиму уделяется довольно много места — больше, чем всем остальным режимам вместе взятым. Это потому, что PWM-режим сложнее простого счета. Но на самом деле идея, которая в него заложена, очень проста: мы загружаем в регистр сравнения очередное число, взятое из звуковой последовательности, и запускаем таймер на счет с нуля, а когда он дойдет до верхнего предела, то сразу реверсируется и начинает считать обратно до нуля. В контроллерах Tuny вместо реверсирования счетчик сбрасывают и начи-

нают отсчет заново. В семействе Mega для формирования сигнала PWM есть и тот, и другой и еще некоторые режимы работы таймеров (например, с переменным битрейтом).

В момент, когда числа в счетчике таймера и в регистре сравнения равны между собой, в режиме PWM автоматически переключается знакомый нам выход, связанный с выбранным таймером (в *главе 14* это был выход OC1, который управлял миганием двоеточия). Только в данном случае он не переключается туда-сюда с каждым прерыванием от таймера, а находится в состоянии логического нуля, когда число в таймере больше, чем в регистре сравнения, и в состоянии логической единицы — когда меньше. В результате на один цикл счета «туда-обратно» мы получаем один период ШИМ-сигнала, в котором длительность состояния логической единицы строго пропорциональна числу в регистре сравнения. Меняя к следующему циклу это число на очередную выборку из звуковой последовательности, мы в результате получаем то, что требовалось: входной импульсный сигнал для усилителя в режиме D. Общая схема процесса показана на рис. 19.2 (на примере с использованием Timer 1). Кстати, отметим, что этот режим может применяться также, например, просто для формирования сигнала с определенной скважностью, не равной двум.

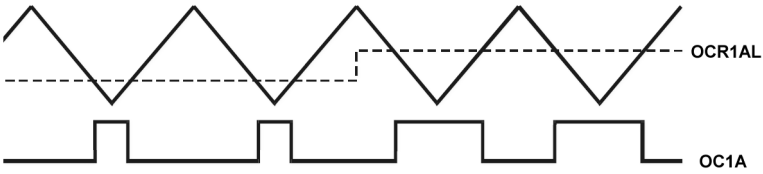


Рис. 19.2. Принцип работы счетчика-таймера в режиме PWM

Теперь надо понять, какие характеристики исходного оцифрованного сигнала нам нужны и какие параметры таймера необходимо устанавливать. Хотя мы будем использовать Timer 1, но задействовать все 16 разрядов в таком режиме он не может (счет в реверсивном режиме возможен максимум с 10 разрядами, а использовать режимы с переменной разрядностью мы не будем). Нам же будет достаточно и 8 — это означает, что глубина квантования исходного звука должна быть также 8 разрядов. Баха не очень сыграешь, но для передачи разборчивой речи достаточно.

Теперь разберемся с частотой оцифровки. Тактовую частоту МК для такой схемы лучше выбирать максимально возможной, для большинства AVR это 16 МГц (чтобы еще повысить качество звука, можно специально взять модель 2313, у которой максимальная частота 20 МГц, но мы будем ориентиро-

ваться на 16 МГц). Легко подсчитать, что реверсивный 8-разрядный счетчик будет считать туда и обратно с частотой $f_{такт}/510$, т. е. при такой тактовой частоте получится около 32 кГц. Это и будет несущая частота $f_{оп}$ на выходе ШИМ, что удовлетворительно, т. к. она выходит за пределы слышимого диапазона. Однако требуемая частота оцифровки исходного звука может быть все же заметно ниже (что удобно в целях экономии памяти). Пусть она составляет 4 кГц (это может возмутить аудиофилов, но для передачи речи это нормальный показатель).

Тогда можно сразу выбрать характеристики RC-фильтра: чтобы отфильтровать 32 кГц простой RC-цепочкой, нам желательно, чтобы частота среза не превышала частоту оцифровки, т. е. 4 кГц. Тогда 32 кГц затухнут в 8 раз по сравнению с верхней частотой диапазона оцифровки, и мы их влияние не почувствуем. Параметры фильтра рассчитываются по формуле $f_{ср} = 1/2\pi RC$, и нашим требованиям удовлетворяют параметры R = 10 кОм и C = 3,9 нФ.

Для хранения звука используем память с I^2C-интерфейсом AT24C512. Одному отсчету тут будет соответствовать ровно один байт, одной секунде звучания — 4 кбайт. Итого 65 536 байт дадут нам около 16 с звучания. Этого достаточно, чтобы произнести стандартную предвыборную речь кандидата в президенты, если предварительно ее отредактировать и выбросить все фразы, не несущие смысловой нагрузки.

Все параметры схемы мы рассчитали, можно приступать к проектированию. В качестве звукового усилителя возьмем описанный в *главе 6* микроусилитель MC34119. Выбор усилителя не имеет большого значения, но данная микросхема «умеет» работать с однополярным напряжением 5 В и это удобно. Общая схема соединений показана на рис. 19.3. Полную схему включения MC34119 см. на рис. 6.15.

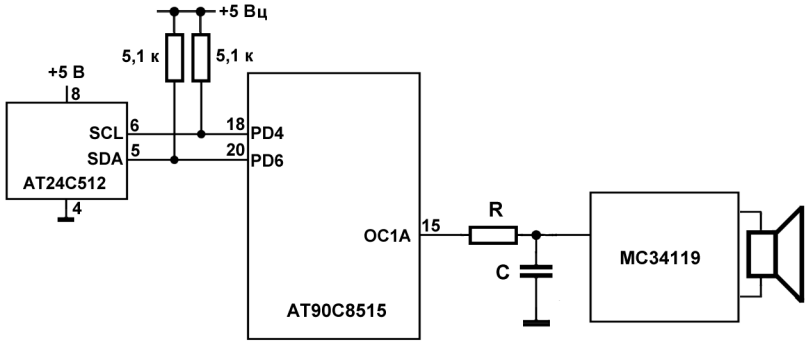


Рис. 19.3. Принципиальная схема для использования AVR в режиме голосовой сигнализации

Программа для вывода звука

Здесь мы для простоты выберем AT90C8515 семейства Classic (а точнее, ATmega8515 в режиме совместимости с AT90C8515, потому что оригинал может работать максимум на 8 МГц тактовой частоты, а мы рассчитывали на 16 МГц). Это проще для нашего рассмотрения, поскольку в семействе Classic имеется лишь один режим PWM для таймеров, а в Mega их много и это лишь путает. В крайнем случае, разобраться в том, как дополнить программу выбором нужного нам в данном случае режима Phase Correct PWM вы сможете самостоятельно. Для работы с интерфейсом I^2C в программе используется тот же самый, что и в *главе 16*, включаемый файл i2c.prg (см. *Приложение 5*, листинг П5.3).

В целях компактности из него можно для данного случая удалить процедуры, относящиеся к RTC (`write_i2c` и `read_i2c`), но будьте осторожны, чтобы не удалить что-то нужное. Кроме того, следует обратить внимание на величину задержки в процедуре `delay` — там у нас установлена величина 5 мкс в расчете на 4 МГц. При 16 МГц задержка укоротится вчетверо, и память будет работать на пределе (400 кГц), что не очень хорошо. Хотя обычно (если линия передачи не слишком длинная) память справляется, но помнить об этом параметре в случае возникновения каких-то сбоев следует. Но и снижать скорость шины до тех величин, что у нас были в измерителе не следует, т. к. чтение может не успеть за битрейтом (см. далее).

Программа для данного случая содержится в листинге 19.1.

Листинг 19.1

```
;==== программа вывода цифрового звука ====
;процессор mega8515 в режиме 8515, частота 16 МГц
.include "8535def.inc"
.equ T_Sample = 193;предварительное значение для Timer 0 при 4 кГц
.equ bSample = 0 ;бит готовности к чтению
;регистры temp и DATA определены в "I2C.prg" (Приложение 5)
.def FLAGS = r19
;-------- прерывания ------
rjmp RESET ; начальный загрузчик
reti
reti
reti
reti
reti
reti
rjmp TIM0 ;обработчик прерывания переполнения Timer 0
```

```
reti
reti
reti
reti
reti

.include "I2C.prg"
RESET:
       ldi   temp,0b00100000
       out   DDRD,temp ;OC1A — на выход
       ldi   temp,(1<<COM1A1)|(1<<PWM10) ;инициализация PWM
       out   TCCR1A,temp
       ldi   temp,1<<CS10 ;включаем Timer1, 1/1
       out   TCCR1B,temp
       ldi   temp,(1<<CS01)|(1<<CS11) ;включаем Timer0, 1/64
       out   TCCR0,temp
       out   TCNT0,T_Sample ;T_Sample=6, заряжаем таймер 0 на 4
кГц
       ldi temp,(1<<TOIE0) ;разрешаем прерывание Timer0
       out TIMSK,temp
       clr   temp ;очищаем все регистры
       out   OCR1AH,temp
       out   OCR1AL,temp
       out   OCR1BH,temp
       out   OCR1BL,temp
       ldi   XH,high(Nbytes)
       ldi   XL,low(Nbytes) ;зарядка счетчика выводимых байт
       ;вместо Nbytes подставить объем записи в байтах, не более
64К
       ldi   YH,high(ADrWord)
       ldi   YL,low(ADrWord)
       ;вместо ADrWord можно подставить начальный адрес
       ;во flash-памяти, он может быть отличен от 0:0
       sei ; разрешаем прерывания
loop_voice: ;читаем байт для последующего вывода
       rcall read_i2c ;чтение памяти,
                  ;в YL,YH — адрес, в DATA — полученных байтов
       sleep ;Idle-mode, проснется по прерыванию Timer0
       sbrs  FLAGS,bSample      ;бит встает в 1 в обработчике
прер. TIM0
       rjmp  loop_voice ;если еще не установлен, то на начало
цикла
       adiw  YL,1 ;иначе увеличиваем адрес на 1
       cbr   FLAGS,1<<bSample ;сбрасываем бит готовности к чтению
       sleep ;Idle-mode, проснется по прерыванию Timer0
       in    temp,TCCR0 ;проверяем, не остановлен ли таймер
```

```
        tst   temp
        brne  loop_voice ;если не остановлен, то следующий цикл
        ;иначе вывод звука закончен — делаем что-то еще
. . . . .
;========= обработчик прерывания от Timer 0 ============
TIM0:
        out   TCNT0,T_Sample ;перезаряжаем Timer0
        clr   temp
        out   OCR1AH,temp
        out   OCR1AL,DATA  ;занесение байта в PWM
        sbr   FLAGS,1<<bSample ;бит готовности к чтению — в 1
        sbiw  XL,1 ;уменьшаем счетчик прочитанных байтов
        brne  rt_pwm_ ;если он равен 0
        out   TCCR0,XL ;то выключаем таймер 0
        out   TCCR1B,XL ;и Timer 1 также
rt_pwm_:
        reti ;возврат из обработчика Timer0
```

В начале программы мы устанавливаем Timer 1 в PWM-режим и задаем ему переключающий режим по выходу OC1A такой, чтобы там устанавливался низкий уровень, а также запускаем его с входной частотой, равной тактовой. Прерываний от Timer 1 никаких не требуется, он будет работать непрерывно, пока мы его не выключим. Управление битрейтом мы будем осуществлять по прерыванию переполнения Timer 0. Если каждый раз в него записывать некоторое число, то можно регулировать частоту таких прерываний. Включим его с частотой на входе, равной 1/64 тактовой (последняя должна быть равна 16 МГц), тогда минимальная частота на выходе будет равна 976 Гц (976,5625 Гц = 16 МГц/64/256). Мы же здесь хотим частоту как можно ближе к 4 кГц (не следует окончательно портить звук еще и изменением битрейта), поэтому мы будем записывать каждый раз в таймер число 193, и он будет считать от 193 до 255, т. е. отсчитывать 62 такта, тогда прерывание будет происходить с частотой почти ровно 4 кГц. Меняя эти параметры (указанное число и частоту на входе Timer 0), можно устанавливать другой битрейт, ограниченный в данном случае скоростью чтения из памяти по интерфейсу I^2C (при максимальной частоте шины 400 кГц эта скорость составит около 9 кбайт/с). При более скоростной памяти битрейт будет ограничен в принципе лишь скоростью работы таймера в PWM-режиме (32 кГц), но при воспроизведении такого звука могут возникнуть сложности из-за искажений. Более глубоко в этот вопрос влезать здесь нет смысла.

В процедуре прерывания мы загружаем очередной байт в Timer 1 и будем устанавливать некий флаг (`bSample` в регистре флагов), а в основной про-

грамме заведем непрерывный цикл, в котором, если этот флаг установлен, производится чтение из памяти следующего байта.

В этой программе число воспроизводимых байтов ограничено 65 536 (64 кбайт), т. к. для упрощения мы считаем их в 16-разрядном регистре `X`, но при необходимости несложно добавить еще один регистр счетчика адреса и задействовать бóльшую емкость памяти (правда, для длинных клипов придется переходить на другие типы интерфейса, см. *главу 16*). В листинге 19.1 указаны теоретические начальный адрес (`ADrWord`) и объем записи (`Nbytes`), которые нужно для вашей задачи заменить на конкретные числа. Кроме того, по окончании звукового фрагмента программа просто остановится. Несложно сделать так, например, чтобы она «закольцевалась»: для этого вместо выключения таймера просто заново занесите значение `Nbytes` в регистры `XH` и `XL`. В общем, приспосабливайте программу для ваших нужд, как можете.

И еще несколько слов о том, откуда берутся исходные звуковые сэмплы. Для этого нужно записать в компьютере звук (моно!) в формате WAV, и обработать его в любом звуковом редакторе, который позволяет регулировать битрейт и глубину оцифровки (например, Sound Forge). Исходным материалом может быть как ваш собственный голос, записанный через микрофон, так и готовый звуковой клип. Формат WAV — чистый оцифрованный звук, и его можно напрямую перекачивать в нашу память. Проще всего для этого воспользоваться каким-нибудь универсальным программатором, но несложно модифицировать данную программу так, чтобы МК сам мог записывать клипы из компьютера через UART. Все необходимые сведения для создания такой программы в этой книге есть.

Аналоговая индикация

Аналоговая индикация может быть во многих случаях более естественным методом для создания человеко-машинных интерфейсов, чем цифровая. Как я уже указывал в *главе 10*, большинство показывающих приборов на пультах управления сложными системами имеют стрелочные или шкальные индикаторы, т. к. точное значение некоего параметра человека интересует не так уж часто. Это касается даже часов: модели со стрелками не есть просто дань стилю «ретро», в некоторых ситуациях (например, когда вы кого-то ждете), они удобнее, чем с цифровым индикатором. Все определяется задачей: от медицинского термометра мы ждем точного значения температуры, от датчика температуры двигателя — лишь оценки относительно некоего порога. Вот когда нужна такая оценка, аналоговый индикатор окажется лучше цифрового.

Существуют готовые микросхемы для управления шкальными индикаторами. Они представляют собой специализированные дешифраторы. Например, К155ИД11[1] при подключении 8 светодиодов, расположенных в ряд, формирует светящийся столбик, высота которого соответствует трехразрядному входному двоичному коду. Для каскадного включения таких микросхем предусмотрены дополнительные входы и выходы (разрешения и переноса), потому с их помощью можно создавать и более длинные шкалы. На практике для большинства применений достаточно 16 градаций.

Более современный аналог К1003ПП1 (UAA180) менее удобен, т. к. управляет 12 светодиодами — ни то ни се (24 при каскадном включении двух микросхем — много, 12 при одной микросхеме — мало). Впрочем, никто не мешает вам задействовать только часть разрядов, да и индикация с 24 разрядами, а то и более, тоже иногда требуется. Отметим, что К1003ПП2, которая вроде бы управляет 16 светодиодами, для практических целей не годится, т. к. высвечивает не столбик, а только один из светодиодов в линейке, что и некрасиво, и малоинформативно. Разумеется, есть и другие подобные микросхемы, но мы не будем на этом задерживаться.

Заметки на полях

Учтите, что приобрести готовую прилично выглядящую шкалу, представляющую собой линейку LED, крайне непросто. Основная проблема состоит в том, что близкорасположенные светодиоды засвечивают друг друга, и в стандартном технологическом процессе их приходится разделять промежутком, чтобы компаунд затекал в зазоры достаточно надежно. Выглядит такая шкала безобразно: лучше уж взять матричный индикатор и сформировать шкалу на его основе, хотя это усложняет управление и тоже не совсем то, что требуется. Наилучший способ — сформировать линейку из плоских светодиодов, вручную окрасить их боковые грани и затем установить на плату. Светодиоды обязательно должны быть с диффузным (матовым) рассеивателем, иначе равномерной засветки вы не добьетесь. В идеале их следует также отбирать по яркости свечения, в противном случае шкала будет неравномерной, хотя на практике это довольно сложно осуществить. Для окраски граней обойтись краской из баллончика не удастся, т. к. она будет просвечивать, особенно на ребрах, даже при покрытии в несколько слоев. Потому необходимо взять обычную темную и достаточно густую нитрокраску (можно даже дать постоять ей на воздухе, чтобы дополнительно загустела) и окрасить весь светодиод методом окунания. Когда краска полностью засохнет (высохнув, она значительно уменьшится в объеме), аккуратно сошлифуйте мелкой шкуркой ее с торца светодиода.

[1] Я ранее (см. *главу 8*) обещал префикс «К» в наименовании микросхем опускать, но в данном случае серия 155, в отличие от 561, встречается и в «военной» модификации (без буквы «К»), а конкретно эта микросхема имеется только в «бытовом» варианте, потому я ее обозначаю полностью. Не существует, кстати, и ее прямого зарубежного аналога.

Большинство проблем, связанных со шкальной индикацией, с помощью К155ИД11 или ее многочисленных аналогов решить можно. Причем следует учесть, что на самом деле 8-разрядная линейка индикаторов имеет не 8, а 9 состояний (восемь зажженных LED плюс состояние, когда все погашены). На выходе К155ИД11, в частности, число зажженных LED на единицу больше входного кода: когда на входе 000, горит один, самый первый светодиод шкалы. Чтобы его тоже погасить, надо подать отдельно сигнал логического нуля на вход переноса (тогда микросхема перестает реагировать на входной код вообще). Поэтому иногда проще уменьшить число ступенек шкалы до 7 (а для 16-разрядного индикатора — до 15). Аналогичная проблема с числом состояний касается всех дешифраторов подобного рода.

При этом придется заняться компрессией, т. к. обычные АЦП все же имеют разрядность много большую, чем требуется в этом случае. Компрессию совершить элементарно просто: достаточно сдвинуть исходное число на столько разрядов вправо, сколько нужно, чтобы результат содержал четыре (для шкалы с 16 состояниями) или три (для 8 состояний) разряда. Предварительно надо позаботиться, чтобы исходное число занимало всю нужную шкалу, т. е. подогнать масштаб.

Приведем пример того, как можно подключить шкальный индикатор к нашему измерителю с помощью микросхемы К155ИД11. Нашей целью будет индикация атмосферного давления в расчете на шкалу с 15 градациями (16 состояниями) в диапазоне от 710 до 785 мм рт. ст. (т. е. по 5 мм рт. ст. на одну градацию шкалы). При этом состоянию 710 и менее должны соответствовать все погашенные LED, от 711 до 715 — один горящий, от 716 до 720 — два горящих и т. д.

Схема на рис. 19.4 составлена в предположении, что от цифровой индикации мы отказались, и порт C, занятый ранее сегментами, у нас освободился. Младшие разряды этого порта мы и задействуем для управления линейкой светодиодов. Остальные соединения на схеме не показаны (см. рис. 15.2). Ради простоты индикацию температуры опустим. Вывод E (разрешения) микросхем управляется старшим разрядом четырехбитового числа с вывода PC3. Логика К155ИД11 такова, что если на E уровень логического нуля, то работа микросхемы DD2 запрещена, если «1» — запрещается работа микросхемы DD3. Так как во втором случае на выходе P верхней микросхемы уровень логического нуля, то нижняя микросхема зажжет все светодиоды. Ограничительных резисторов для светодиодов не требуется, они встроены в микросхему, хотя яркость в этом случае может быть непредсказуемой (а вот К1003ПП1 «умеет», в том числе, управлять яркостью).

Программу придется переделать таким образом. Расчеты физических величин нам уже не требуются, но вот преобразование масштабов провести при-

дется. Потому на подготовительном этапе нужно выяснить значение коэффициентов *K* и *Z* таких, чтобы по уравнению зависимости выходного кода от значений, прочитанных из АЦП, приведенному в *главе 15*, у нас значение выходного кода, равного нулю, соответствовало 710 мм рт. ст. Таким образом, коэффициент *Z*, который нужно вычесть из кода АЦП, будет равен значению кода при 710 мм рт. ст., или, как несложно рассчитать, примерно 840 (с учетом того, что датчик работает не с нуля давления, см. *главу 15*).

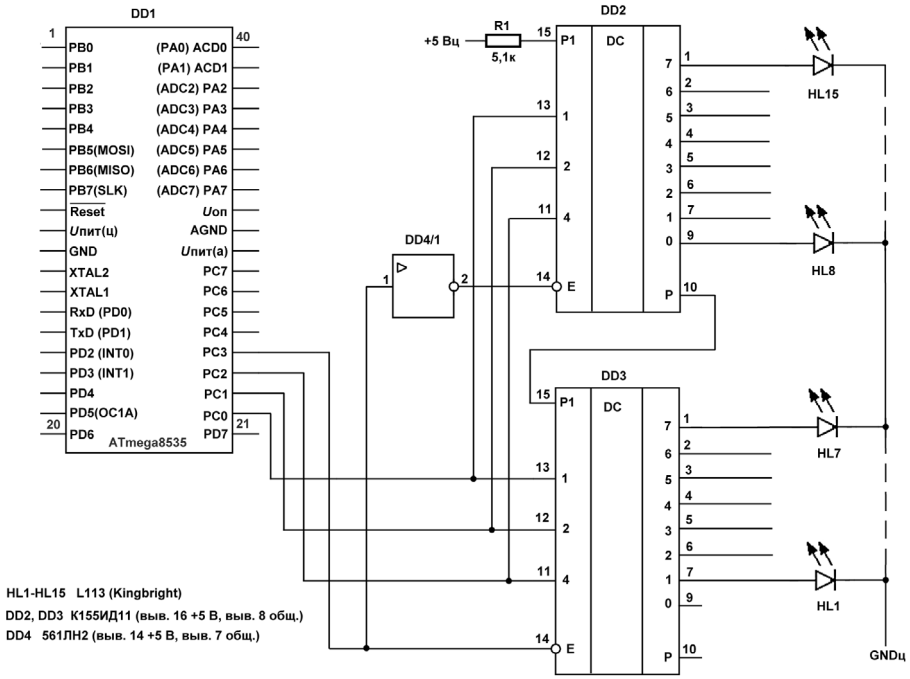


Рис. 19.4. Схема шкальной индикации для измерителя давления и температуры

Значение же, соответствующее 785 мм рт. ст., должно соответствовать какому-нибудь «круглому» двоичному числу (не очень важно, какому, т. к. мы потом его все равно урежем до 4 бит). Из характеристик датчика мы знаем, что максимальная шкала АЦП в 10 бит соответствует давлению около 850 мм рт. ст. Нас же интересует шкала всего в 75 мм (от 710 до 785), что составит около 90 единиц кода. Потому мы смело можем выбрать, например, 128 для верхнего предела шкалы (что соответствует 7 битам). Тогда коэффициент *K* (который ранее составлял 0,895 мм рт. ст. на единицу кода), теперь будет примерно 128/90 = 1,422. Оба коэффициента, естественно, должны уточняться при калибровке.

Сама процедура расчета не нуждается в переделке (меняются только значения коэффициентов, остальное можно оставить, как есть, хотя если внимательно ее рассмотрите, то увидите, что есть резервы для сокращения необходимых ресурсов, например, задействованных регистров). Единственное, что следует учесть — вычисленные значения хранятся в регистрах `AregH:AregL` (см. *Приложение 5*), но 7-битовый результат, конечно окажется только в регистре `AregL`, а регистр `AregH` всегда будет равен нулю. После расчета, вместо преобразования в двоично-десятичный код (`rcall bin2BCD16`), мы должны записать:

```
. . . . .
lsr AregL
lsr AregL
lsr AregL ;теперь результат усечен до 4 бит
in temp.PortC ;значение разрядов PortC в temp
cbr temp,15 ;обнуляем младшие 4 бита
ori temp,AregL ;устанавливаем младшие 4 бита
out PortC,temp ;выводим
. . . . .
```

Здесь операции очистки младших битов и сохранения старших требуются для того, чтобы не вмешиваться в процессы, которыми (предположительно) могут управлять старшие биты порта C. В остальном программу читатель может доделать самостоятельно — она упростится, даже если ввести еще индикацию температуры. При полном отказе от цифровой индикации исчезнет необходимость в громоздкой процедуре управления разрядами (индикация становится статической), и также сократится число необходимых ячеек в SRAM — ясно, что хранить четырех- и даже восьмибитовые значения величин ни к чему. А в физические величины, если потребуется, исходные значения параметров можно пересчитать и в компьютере, сохраняя их в энергонезависимой памяти.

В заключение отмечу, что в сконструированном мной однажды приборе для некоей яхты была задействована более сложная и более красивая шкальная индикация: при значении параметра ниже некоторого порога индикаторы меняли цвет на красный, который сменялся зеленым при повышении параметра до нормы. Естественно, готовые микросхемы (типа К155ИД11 и подобные) такую задачу решить не позволяют, и пришлось городить всю процедуру внутри контроллера, управляя светодиодами напрямую. Задачу изменения цвета удалось решить, применив двухвыводные двухцветные LED (L117), которые обоими выводами коммутировались к выводам МК (для одного цвета действовало сочетание состояний 0 и 1, для другого — 1 и 0, а погашен LED оказывался при одинаковом состоянии выводов).

Подстройка внешних часов

Кварцевые резонаторы имеют весьма высокую точность: ±1 с в сутки для электронных часов равносильны, например, ошибке в 1 м на 86 км при измерении длины. Но, не говоря уж о специальных технических применениях, такой точности при измерении времени недостаточно даже в быту — часы с такой ошибкой будут уходить на полминуты в месяц и их придется подводить каждые два-три месяца, как минимум. Если для стрелочных часов такая процедура ничего сложного не представляет, то для электронных она выливается в довольно занудные манипуляции с кнопками или необходимость (как в нашем измерителе с часами) коррекции времени через подключение к компьютеру.

Манипуляций хочется избежать, а иногда и просто необходимо обеспечить ход часов более точно. Как это сделать? Как мы видели в *главе 14*, 16-разрядный таймер при отсчете секунд дает достаточно грубую подстройку того же порядка, что и ошибка, т. е. для начала надо придумать способ более тонкой подстройки. Кроме того, когда речь идет о внешних часах, то мы вообще не можем вмешаться в процесс счета времени. Значит, кроме самой поправки, необходимо придумать еще механизм того, как ее применять к внешним часам. Все эти проблемы можно решить, например, следующим способом.

Для начала о самой поправке: можно, конечно, попросту ввести задержку вроде той, что служит для формирования сигналов I^2C (собственно, так и отсчитывали время в контроллерах, когда в них еще не было таймеров). Но это не очень хорошо по той же причине, по которой такие паузы лучше не использовать в Windows, где много процессов протекают параллельно. Либо это задержит весь контроллер, либо (если прерывания разрешены) задержка может быть непредсказуемой длительности. Потому лучше задействовать таймер (скажем, Timer 2, который имеется во всех Mega, а Timer 1 оставим для других надобностей), только запускать его так, чтобы он формировал прерывания достаточно короткой длительности. Например, ежесуточная коррекция ступеньками по 10 мс нас будет вполне устраивать (при точном подборе задержки ошибка составит не более 1 с в три месяца, если не учитывать «гуляние» частоты кварца самой по себе).

Самое сложное здесь — понять, как автоматически ввести коррекцию в обе стороны (и когда часы спешат, и когда отстают). Я придумал следующий механизм: мы задаем некий коэффициент коррекции в виде байта $K_{ккор}$. Если этот байт равен $FF (расчет на то, что именно такое значение содержится в чистой EEPROM), то коррекцию вообще не проводим, или загружаем значение по умолчанию (как и с коэффициентами).

Если часы спешат, то значение $K_{ккор}$ должно находиться в пределах 1—127 (старший бит равен нулю), и оно определяет задержку, которую мы проводим с помощью таймера, настроенного на 10 мс, раз в сутки. Для этого надо остановить часы, пропустить столько циклов таймера, сколько задано в $K_{ккор}$, и запустить их заново с того же значения. Максимальная задержка составит 1,27 с в сутки, что намного перекрывает возможную ошибку кварца.

Сложнее, если часы отстают. Чтобы отличить эту ситуацию от предыдущей, мы задаем старший бит $K_{ккор}$ равным единице, но в пределах 155—254 (т. е. коррекция возможна в пределах 1 с). Почему именно в этих пределах, мы сейчас поймем. Величина задержки программой будет вычисляться, как разность между 255 и $K_{ккор}$ (задержка, равная нулю, не имеет смысла, отсюда максимальное значение 254). В МК мы останавливаем часы, включаем таймер, делаем столько сравнений, сколько задано этой разностью, затем устанавливаем секунды = 1 и запускаем часы опять (теперь они идут немного вперед). Отсюда понятно, почему мы не можем задать $K_{ккор}$ меньше 155 — при таком алгоритме значение, к примеру, 128 привело бы к еще большему отставанию. Следовательно, чем больше задержка (разность между 255 и $K_{ккор}$), тем меньше коррекция в данном случае: если нам нужно коррекцию свести к нулю, то следует задать значение $K_{ккор}$ = 155. Отметим, что остановка и пуск часов сами по себе займут время порядка миллисекунд, но ошибка будет невелика, и ее можно не учитывать.

Для коррекции задействуем Timer 2 с коэффициентом 1:256, получаем на выходе 15 625 Гц (при тактовой частоте 4 МГц). Если использовать прерывание по сравнению с величиной 156, получаем примерно 100 Гц (или 10 мс). $K_{ккор}$ мы будем хранить в EEPROM (по адресу `KorrEE`), и переписывать его в RAM (по адресу `KorrRAM`) — эти адреса выбираются из любых свободных. Процедура для инициализации регистра сравнения таймера и коэффициента коррекции приведена в листинге 19.2 (в секторе начальной загрузки, о самой программе см. *главу 16*).

Листинг 19.2

```
;команду ldi temp,(1<<TOIE0) заменяем на
      ldi    temp,(1<<TOIE0)|(1<<OCIE2)
;добавляем:
      ldi temp,156
       out OCR2,temp ;получаем  100 Гц  Timer2
. . . . .
;коэффициент коррекции ============================
      clr    ZH;addr eepr
      ldi    ZL,KorrEE
```

```
      rcall  ReadEEP
      cpi temp,$FF
      brne corr_K
      ldi temp,100 ;если = FF, то по умолчанию 100
corr_K:
       ldi ZH,1
       ldi ZL,KorrRAM
       st Z,temp
```

Величина задержки по умолчанию определяется на основе предварительных изысканий по уходу конкретного кварца (здесь часы спешили примерно на 1 с в сутки). Не забудем заменить в четвертой сверху строке таблицы прерываний `reti` на `rjmp TIM2_COMP`. Отведем для счета прерываний регистр `count_msek` (пусть будет `r12`). Бит 4 регистра `Flag` будет сигнализировать о том, отстают часы или спешат (если отстают, то бит установлен). Значение этого бита и регистра `count_msek` будем устанавливать в момент, когда будет вызываться процедура коррекции (см. далее), там же часы останавливаются. Сам обработчик приведен в листинге 19.3.

Листинг 19.3

```
TIM2_COMP:
       dec count_msek
       brne end_t  ;если еще не 0, то на выход
       sbrs Flag,4
       rjmp k_minus ;если спешат, то пропустить
;если отстают
       cbr Flag,8 ;очищаем бит к следующему разу
;устанавливаем часы на 01 сек. и заводим их
       ldi temp,1
       rcall IniSek
       ldi count_sek,1 ; меняем значение в регистре секунд
       ldi ZH,1
       ldi ZL,Sek
       st Z,count_sek
       rjmp end_korr
k_minus:
;если спешат, просто заводим часы обратно
       clr temp
       rcall IniSek
end_t:
reti ;конец прерывания коррекции
```

Теперь собственно процедура вызова коррекции: будем выполнять ее в полночь. Она довольно громоздкая, потому что приходится определять момент, когда полночь настала. Мы можем вклинить ее туда, где часы проверяются на кратность трем (0 минут и 0 секунд нам обеспечены). Однако сразу после этого производится запись во flash (а иногда и не производится), и она нам будет непредсказуемо тормозить коррекцию. Потому будем ее проводить в одну минуту первого (00:01:00). Сразу после метки `sek_0` вписываем фрагмент кода, содержащийся в листинге 19.4.

Листинг 19.4

```
sek_0:
       ldi ZL,1;загружаем минуты
       ld temp,Z+ ;
       cpi temp,1 ;сравниваем минуты, если 1- коррекция
       breq min_1
       cpi temp,0 ;сравниваем минуты, если 0 — запись
       breq mm0 ;на проверку трехчасового цикла
       reti ;иначе выходим
min_1:
       ld temp,Z ;загружаем часы
       cpi temp,0 ;сравниваем часы = 0
       breq hour_0 ;если равны 0, то на коррекцию
       reti ;иначе выходим
hour_0:
          ldi ZL,KorrRAM ;коэффициент коррекции
          ld count_msek,Z ;в счетчик
          ldi temp,128
          cp count_msek,temp ;определяем его величину
          brlo k_plus
       com count_msek ;если больше 127, то вычитаем из 255
       sbr Flag,8 ;значит отстают
k_plus: ;если меньше — часы отстают
       ldi temp,$80
       rcall IniSek ;останавливаем часы
       ldi temp,0b00000110 ;заводим таймер
       out TCCR2,temp ;Timer2 1:256
reti
mm0: ;далее по тексту программы
```

Разумеется, если вы используете сторожевой таймер, то его следует останавливать на время проведения этой операции (и запускать в прерывании Timer 2 по окончании коррекции.) В правильно организованной программе, кроме того, необходимо также запрещать за некоторое время до наступления момента

коррекции длинные процедуры с запрещением прерываний (вроде чтения данных из Flash), иначе коррекцию можно пропустить.

Измерение частоты

Частоту можно измерять, как известно, двумя способами: либо подсчетом числа импульсов измеряемой частоты за определенный промежуток времени, либо, наоборот, подсчетом числа импульсов известной частоты за период (или несколько периодов) измеряемого сигнала. В первом случае мы получаем именно значение частоты (если промежуток времени равен 1 с, то сразу в герцах), а во втором — обратную величину, значение периода. Первый способ удобнее для измерения высоких частот, второй — низких.

С помощью контроллеров МК частоту можно измерять несколькими путями. В том числе есть специальный режим работы таймеров с «захватом» (capture) внешнего перепада уровней и генерацией прерывания по этому поводу. Он удобен для измерения периода низких (с точки зрения МК) частот по второму способу. Здесь я покажу метод прямого измерения (по способу подсчета импульсов) достаточно высокой частоты, причем с подстройкой измерительного интервала для получения более точного результата прямо в физических величинах — герцах.

Предположим, измеряемая частота находится в диапазоне до 4 МГц, и нам желательно измерить ее с разрешением до 1 Гц. Прежде всего отметим, что тактовая частота контроллера должна превышать измеряемую не менее, чем в два раза — таково требование руководства. Обнаружение изменения внешнего сигнала производится по фронту тактового, и если период измеряемого сигнала слишком короткий, то в регистрации могут быть пропуски.

> ***Замечание***
>
> Отметим, что перепад уровней внешнего сигнала регистрирует автономная асинхронная схема. Потому из изложенного в руководстве не следует, что тактовая частота должна быть выше измеряемой именно в два раза — достаточно простого превышения. И по моему опыту AVR прекрасно измеряют частоту всего в полтора раза ниже тактовой. Однако окончательное суждение по этому вопросу я оставляю на усмотрение читателей — изложение в руководстве не очень толковое (не до конца прояснен вопрос с задержками регистрации фронта сигнала), и, конечно, лучше «на всякий случай» следовать фирменным рекомендациям.

Так что нам потребуется МК с тактовой частотой не менее 8 МГц. Выберем AT90C2313 (Classic — при необходимости легко модифицировать алгоритм к любому AVR) с частотой 8 МГц. Для измерения мы используем два таймера — один 16-разрядный (Timer 1) для отсчета собственно внешней частоты, и второй (Timer 0) для отсчета измерительного интервала.

Заметки на полях

Результат измерения частоты 4 МГц с точностью до герца в принципе займет не менее трех 8-битовых регистров. Но реальное их число, которое требуется задействовать, будет зависеть от нижнего предела измеряемой частоты. В самом деле, предположим, что частота может меняться не более чем на 256 Гц. Тогда старшие два регистра всегда будут показывать одно и то же число (и точно известно, какое), а все изменения будут регистрироваться только в самом младшем регистре счетчика. Если же частота 4 МГц не меняется более чем на 65 кГц, то можно оставить только два регистра (собственно таймера). Здесь важно только, чтобы в процессе изменений частота не «переваливала» за границу, когда старший регистр тоже меняется (что в нашем случае произойдет, например, если средняя частота колеблется около значения 2^{22} = 4 194 304), иначе возникнет неоднозначность (которую, впрочем, также в некоторых случаях можно учесть). Но мы не будем в этот вопрос углубляться, а тупо предположим, что частота в пределах емкости трех регистров (т. е. с большим запасом — до 16,7 МГц) может быть любой.

Для измерения нам потребуется ввести прерывание Timer 1 по переполнению, в котором третий регистр (назовем его `count3`) будет всякий раз увеличиваться на единицу. Входной сигнал подадим на вход T1 (вывод 9 для 2313), с которого внешние импульсы поступают прямо на счетчик таймера, если ему задать соответствующий режим.

Теперь разберемся с формированием измерительного интервала. При 8 МГц тактовой частоты и коэффициенте предделителя для Timer 0, равном 1/256, прерывания будут происходить с частотой 122,07 Гц. Нам же требуется 1 с (1 Гц), потому мы введем счетчик (`count_sek`) и будем его с каждым прерыванием увеличивать, пока он не отсчитает ровно 122 таких прерывания. После этого можно фиксировать число импульсов, сосчитанное к тому времени в регистрах Timer 1. Но если кварцевый резонатор идеально точный, то секунда получится чуть меньше настоящей (неучтенные 0,07 Гц дадут ошибку 576 мкс со знаком «минус»), и перед чтением значений мы введем задержку для компенсации этой недостачи, с помощью которой наш частотомер можно еще дополнительно калибровать (т. е. учесть исходную неточность кварца). В начале и в конце интервала будем переключать разряд 6 порта D (вывод 11), чтобы контролировать измерительный интервал в процессе калибровки. На рис. 19.5 представлен МК AT90S2313 с обозначением выводов для нашей цели.

Программа частотомера приведена в листинге 19.5 (определение регистров я опускаю, в данном случае их потребуется всего три — `temp`, `count3` и `count_sek`). В секции прерываний введем прерывания Timer 0 и Timer 1 по переполнению (по меткам `TIM0` и `TIM1`). Инициализация таймеров в секции начальной загрузки сводится к разрешению прерываний, но обязательно здесь же нужно очистить счетные регистры таймеров и все дополнительные

регистры (к сожалению, очистка предделителя таймеров в серии Classic не предусмотрена) и только потом запускать Timer 0.

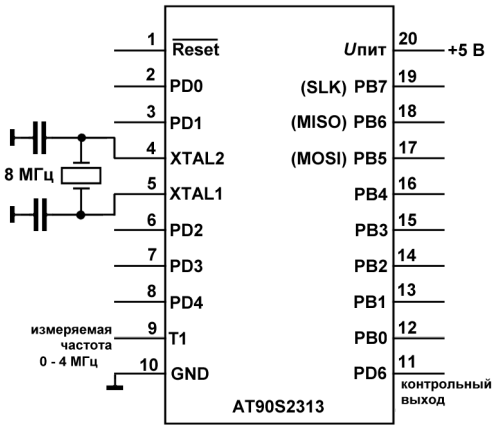


Рис. 19.5. Подключение AT90S2313 для измерения частоты

Листинг 19.5

```
ldi    temp,(1<<TOIE0)|(1<<TOIE1) ;разр. прер. Timer0 и Timer1
out    TIMSK,temp
clr temp
out TCNT1H,temp
out TCNT1L,temp  ;очищаем Timer1
out TCNT0,temp ;очищаем Timer0
clr count3
clr count_sek ;очищаем счетчик прерываний
ldi temp,0b00000100;
out TCCR0,temp ;запускаем Timer0 div 1:256
```

Прерывание Timer 1 будет очень простое (листинг 19.6).

Листинг 19.6

```
TIM1:
 inc count3
reti
```

Теперь рассмотрим самое главное прерывание, Timer 0 (листинг 19.7).

Листинг 19.7

```
TIM0: ;таймер 122,07 Гц
      inc  count_sek
      cpi count_sek,122    ;получаем 0.999424 с
      breq corr_1 ;если секунда прошла, то на коррекцию счета

      cpi count_sek,1
      brne corr_1
;в самом первом цикле запускаем Timer 1:
      ldi    temp,0b00000111;внешний сигнал T1 (выв. 9) по фронту
      out TCCR1B,temp ;запускаем Timer1
corr_1:      ;1 сек + коррекция
      clr temp
      out TCCR0,temp ;останавливаем Timer0
      ;задержка на ~600 мкс для коррекции интервала
      ldi ZH,high(1200)  ;8 МГц, цикл 4 такта
      ldi ZL,low(1200)
loop:
      sbiw ZL,1
       brne loop
;переключение контр. выв 11 PinD6  период 1 с
       sbis PinD,6
       rjmp set_1
       cbi PortD,6
       rjmp Set_0
Set_1:
       sbi PortD,6
Set_0:
       clr temp
       out TCCR1B,temp ;останавливаем Timer1
;читаем данные
       in temp,TCNT1L
       <пишем младший байт в память>
       in temp,TCNT1H
       <пишем второй байт в память>
       <пишем старший байт count3 в память>
;очищаем все регистры
       clr temp
       out TCNT1H,temp
       out TCNT1L,temp  ;очищаем Timer1
       out TCNT0,temp ;очищаем Timer0
       clr count3
       clr count_sek ;очищаем счетчик прерываний
;запускаем таймер 0 опять
       ldi temp,0b00000100;
       out TCCR0,temp ;запускаем Timer0 div 1:256
reti
```

Обратите внимание, что читать данные из регистров таймера нужно в указанном порядке: сначала старший, потом младший, а записывать в обратном порядке. Это сделано специально: при чтении из старшего регистра `TCNT1H` данные из младшего `TCNT1L` одновременно переписываются в специальный регистр временного хранения, и ситуации, когда в промежутке между командами чтения младший разряд может измениться (при запущенном таймере), не возникает. То же самое, только в обратном порядке, происходит и при записи. Запись данных я не расшифровывал, потому что это может быть и запись в SRAM с последующим выводом на индикацию, и запись во внешнюю энергонезависимую память для последующего чтения из компьютера (или одновременно и то и другое). Простейший частотомер можно сделать, если организовать автоматическую передачу данных через UART в компьютер, который и занимается отображением и записью информации.

Из-за инструкции `sbiw`, которая занимает два такта (а не один, как инструкция `dec`, которую мы использовали в процедуре `delay` для интерфейса I^2C, см. *главу 16*), здесь один цикл задержки равен четырем тактам, или 0,5 мкс при тактовой частоте 8 МГц. Меняя число циклов задержки, можно подстроить длительность секундного интервала в интервале от 576 мкс в сторону уменьшения (задержка равна нулю) до целых 131 мс в сторону увеличения. 576 мкс может показаться слишком маленьким значением, но этого достаточно для подстройки стандартного кварца, в крайнем случае, можно отобрать экземпляр из нескольких. Калибровка осуществляется измерением длительности импульса на контрольном вводе 11 МК с помощью точного частотомера (профессионального лабораторного прибора, а не любительского, который имеет недостаточную точность).

У AT90S2313 недостаточно выводов, чтобы напрямую обеспечить динамическую индикацию для нашей цели (7 десятичных разрядов), поэтому можно либо только использовать данные непосредственно, либо управлять разрядами через внешние дешифраторы (скажем, 561ИД5 позволяет управлять семисегментным индикатором четырехразрядным двоичным кодом с выводов PD0—PD3, а управление переключением семи и даже восьми разрядов можно тогда осуществить через полностью свободный порт B). Можно, конечно, выбрать другой контроллер. Динамическая индикация практически никак не помешает счету, так как таймер считает абсолютно независимо от остальных схем.

Частотомер получится довольно низкочастотный (если только не использовать внешние счетчики-делители с ущербом для разрешающей способности измерений). Но самой сложной проблемой для построения настоящего частотомера будет формирование из исходного сигнала произвольной формы последовательности «чистых» импульсов нужной формы и амплитуды.

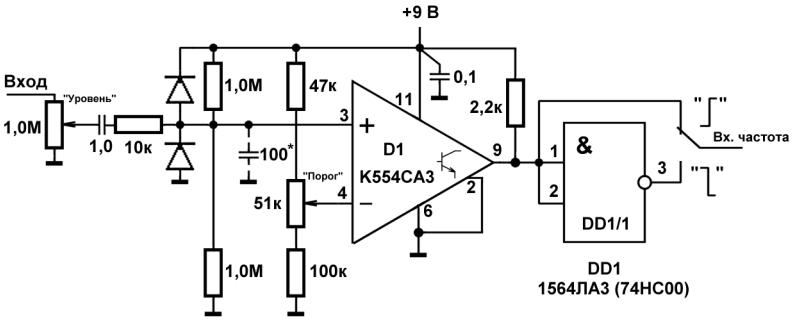


Рис. 19.6. Формирователь входных импульсов для частотомера 0—4 МГц

Без подробного обсуждения привожу одну из возможных схем такого формирователя импульсов (рис. 19.6), работающего на частотах до 4 МГц. 554СА3 заменяется на 521СА3 (импортный аналог — LM311). Конденсатор 100 пФ служит для фильтрации высокочастотных помех и его емкость подбирается при регулировке. Предпочтителен более современный (и несколько более быстродействующий) LM6511, совпадающий с указанными по выводам. Переключатель полярности сигнала в принципе не требуется (его можно реализовать программно простым переключением режимов Timer 1), но таким образом можно сэкономить вывод МК, который пришлось бы подсоединять к переключателю.

Объединение систем на МК

Как я уже упоминал в *главе 16*, последовательный порт USART может работать в режиме мультипроцессорного обмена. Однако реализован он довольно сложно и малопригоден для организации такого часто встречающегося варианта, когда есть главный компьютер и несколько равноправных систем на МК, с которым хочется организовать двустороннюю связь через единый COM-порт. Сейчас мы разберем одну из возможностей организации такого обмена.

Обмен такого рода не может обойтись без присвоения индивидуального адреса устройству, т. к. их надо как-то различать. Все подобные протоколы (I^2C хотя бы) различаются лишь способом доставки и форматом этого адреса. В нашем же случае придется еще придумать, как обеспечить «прозрачное» переключение каналов обмена во избежание конфликтов (в USART это достигается односторонностью обмена — по линии от главного МК к ведомым передается только 9-битовый адрес, а обратно — только 8-битовые данные, нам же нужен двусторонний обмен).

Для реализации такого варианта мы сконструируем специальную плату-коммутатор на основе отдельного МК (возьмем тот же AT90S2313). Идея состоит в том, что мы выделяем специальные команды-адреса, которые воспринимает только этот МК, и в соответствии с ними переключает канал обмена на нужное устройство. Если переключать с помощью мультиплексоров/демультиплексоров 561КП2 (см. рис. 8.8), то можно адресовать до восьми устройств. В качестве команд адресации удобно выбрать числа от 0 до 7, тогда они прямо будут соответствовать коду, который требуется подать на мультиплексоры.

Естественно, среди команд управления устройствами и передаваемых к устройству данных байты с таким значением должны полностью отсутствовать (например, установку часов напрямую таким способом не передашь), и это накладывает ограничения, но не очень серьезные. В каждом конкретном случае можно что-нибудь придумать: например, дополнять данные со значением меньше 8 старшим битом, равным единице, или еще что-то в этом роде (с похожими проблемами сталкиваются при передаче произвольных данных — вложений — по электронной почте, и ничего, как видите, справляются). Разумеется, можно задействовать и дополнительные линии COM-порта для адресации коммутирующего МК отдельно от остальных, или использовать девятибитовые посылки адреса (так, как это делается в USART), чтобы отличить их от данных и т. п. Здесь я приведу только самый простой вариант.

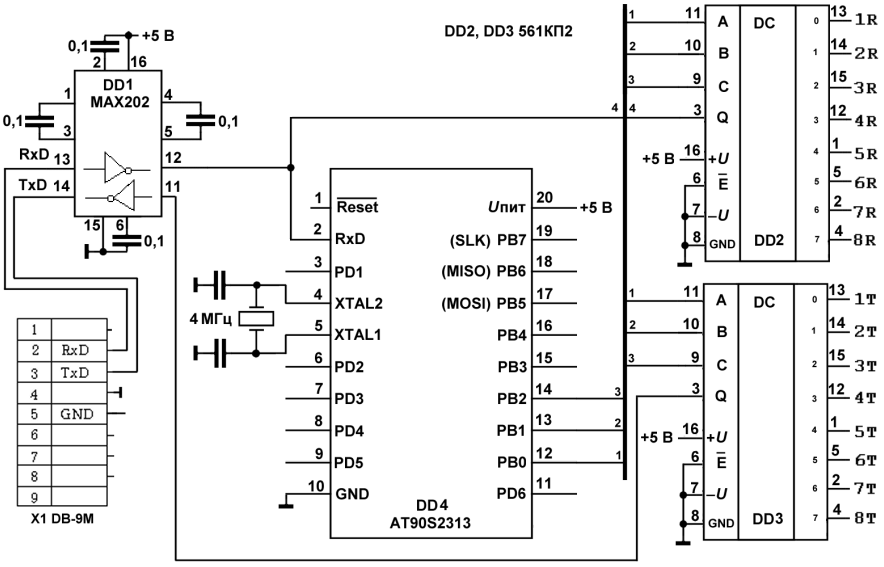


Рис. 19.7. Коммутатор UART на 8 каналов

Схема такого коммутатора показана на рис. 19.7. Выводы коммутатора, помеченные номером с буквой R, следует присоединить к выводам RxD устройств, а их выводы TxD (строго в том же порядке) следует соединить с выводами коммутатора, помеченными номером с буквой T. Программа коммутатора (листинг 19.8) очень проста и даже не содержит таблицы векторов прерываний, поэтому я привожу ее целиком.

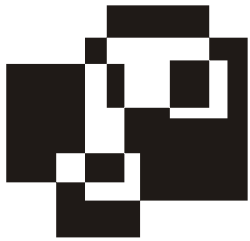
Листинг 19.8

```
;Тестовый коммутатор
;Кварц 4 МГц
.include «2313def.inc»
;========
 .def      temp  = r16
;========== программа
       ldi temp,low(RAMEND) ;загрузка указателя стека
       out SPL,temp
       ldi temp,(1<<ACD) ;выкл. аналог. компаратор
       out ACSR,temp
       ldi temp,(1<<RXEN|1<<TXEN|1<<RXB8|1<<TXB8)
       out UCR,temp ;разрешение приема/передачи 8 бит
       ldi temp,25
       out UBRR,temp ;скорость передачи 9600
       ldi temp,0b00000111 ;устанавливаем PB0-PB2 выходы
       out DDRB,temp
       clr temp
       out PortB,temp ;по умолчанию адресуется устройство 0

G_cykle:
       rcall   in_com
       cpi temp,9 ;если принятый байт больше или равен 9
       brsh G_cykle ;то ничего не делаем
       out PortB,temp ;иначе выводим его в порт B
rjmp G_cykle
in_com:   ;прием байта в temp с ожид. готовности
      sbis   USR,RXC
      rjmp   in_com
      in     temp,UDR
ret
```

Больше ничего делать не требуется — «верхняя» программа всегда начинает с того, что посылает номер устройства *n* от 0 до 7, мультиплексор коммутирует выходы *n*R и *n*T к выводам RxD и TxD устройства с номером *n*, и далее «общение» с ним происходит совершенно «прозрачно», как будто остальных устройств не существует.

ПРИЛОЖЕНИЯ



Приложение 1

Принятые условные обозначения

Физические величины и их единицы измерения по умолчанию

I — ток, ампер (А);

U — напряжение, вольт (В, V);

R — электрическое сопротивление, ом (Ом, Ohm);

E — энергия, джоуль (Дж);

P — электрическая мощность, ватт (Вт, W);

W — тепловая мощность, ватт (Вт, W);

C — электрическая емкость, микрофарад (мкФ, μF);

Q — заряд, кулон (Кл);

t — время, секунда (с);

T,θ — температура, градус Цельсия (°С);

L — длина, метр (м);

S — площадь ($м^2$);

ρ — удельное сопротивление ($ом \times м/мм^2$).

Примечание

Все наименования единиц измерения, которые названы по фамилиям ученых, в сокращении пишутся с большой буквы (вольт — В, ом — Ом, ватт — Вт, ампер — А), а все остальные — с маленькой (секунда — с, метр — м).

Приставки и множители для образования десятичных кратных и дольных единиц

В табл. П.1.1 приведены приставки и множители для образования десятичных кратных и дольных единиц.

Таблица П1.1

Приставка			Множитель
Наименование	Обозначение		
	русское	международное	
экса	Э	E	10^{18}
пета	П	P	10^{15}
тера	Т	T	10^{12}
гига	Г	G	10^{9}
мега	М	M	10^{6}
кило	к	k	10^{3}
гекто	г	h	10^{2}
дека	да	da	10
деци	д	d	10^{-1}
санти	с	c	10^{-}
милли	м	m	10^{-3}
микро	мк	μ	10^{-6}
нано	н	n	10^{-9}
пико	п	p	10^{-12}
фемто	ф	f	10^{-15}
атто	а	a	10^{-18}

Некоторые буквенные обозначения в электрических схемах

R — резистор;

C — конденсатор;

VT — транзистор;

VD — диод;

H — цифровой индикатор;

K — тумблер, кнопка;

P — реле;

DA — аналоговая микросхема;

DD — цифровая микросхема;

D — микросхема, содержащая аналоговые и цифровые узлы;

Q — кварцевый резонатор;

L — лампа (газоразрядная или накаливания).

Некоторые символические обозначения в электрических схемах

В табл. П1.2 приведены символические обозначения в электрических схемах.

Таблица П1.2

Обозначение	Наименование
	Постоянный резистор
	Переменный резистор
	Подстроечный резистор
	Резистор с зависимостью от температуры (терморезистор, термистор)
	Конденсатор
	Полярный (электролитический) конденсатор
	Диод

Таблица П1.2 *(продолжение)*

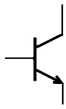
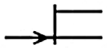
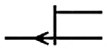
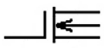
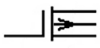
Обозначение	Наименование
	Стабилитрон
	Двусторонний стабилитрон
	Тиристор
	Светодиод
	Фотодиод
	Биполярный транзистор (*n*-*p*-*n*)
	Биполярный транзистор (*p*-*n*-*p*)
	Полевой транзистор с *p*-*n*-переходом и *n*-каналом
	Полевой транзистор с *p*-*n*-переходом и *p*-каналом
	Полевой транзистор с изолированным затвором и *n*-каналом (*n*-МОП)
	Полевой транзистор с изолированным затвором и *p*-каналом (*p*-МОП)
	Кварцевый или керамический резонатор
	Нормальнозамкнутый контакт (реле, тумблера, кнопки)
	Нормальноразомкнутый контакт (реле, тумблера, кнопки)
	Реле с нормальноразомкнутым контактом

Таблица П1.2 *(окончание)*

Обозначение	Наименование
	Лампа накаливания
	Лампа газоразрядная (неоновая)
	Динамическая звуковая головка
	Общий провод
	«Земля»

Символические обозначения мощности резисторов на схемах

В табл. П1.3 приведены символические обозначения мощности резисторов на схемах.

Таблица П1.3

Обозначение	Мощность резистора
	0,125 Вт
	0,25 Вт
	0,5 Вт
	1 Вт
	2 Вт
	10 Вт

Приложение 2

Стандартные обозначения и размеры некоторых гальванических элементов

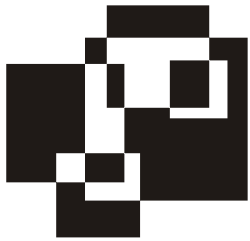
В табл. П2.1 приведены стандартные разновидности бытовых одноразовых электрохимических элементов питания. Элементы одного типоразмера взаимозаменяемы (а также заменяются на аналогичные типоразмеры аккумуляторов), однако могут различаться по энергоемкости. Щелочные (с буквой «L» или надписью «alcaline») элементы приблизительно в три раза превышают по емкости «обычные». Кроме указанных в таблице «пальчиковых» элементов, широко распространены «таблетки» — серебряно-цинковые или угольные малогабаритные элементы с напряжением питания обычно около 3 В, предназначенные для питания маломощных устройств.

Таблица П2.1

Напряжение, В	Обозначение (типоразмер)			Размеры, мм		Примерная емкость, мА·ч**
	ANSI	Международное (МЭК)*	Отечественное (ГОСТ)	Диаметр (высота× ширина)	Длина (толщина)	
4, 5	—	3R12, 3LR12	3336	62×67	22	—
9	F8	6F22	«Крона»	17,5×26	48	600
1,5	AAA	R03, LR03	286	10,5	45	1000
1,5	AA	R6, LR6	316	14,5	50,5	2000
1,5	C	R14, LR14	343	26,2	50	6000
1,5	D	R20, LR20	373	34,2	61,8	15—20 000

* Буква L означает щелочной элемент.

** Для щелочных элементов.



Приложение 3

Справочные данные некоторых компонентов

Далее приведены справочные данные некоторых компонентов, которые мы использовали в схемах, размещенных в этой книге. Разумеется, поместить данные всех упомянутых в тексте компонентов, и к тому же в полном объеме невозможно, поэтому были выбраны лишь самые характерные и часто упоминающиеся полупроводниковые приборы, таким образом, что они в совокупности образуют некий базовый набор, достаточный, чтобы повторить примерно 9 из 10 радиолюбительских конструкций. Данные о некоторых типах приводятся также при описании соответствующей конструкции в тексте глав. Сведения далее «профильтрованы» на предмет реальной информативности: к примеру, нет никакой необходимости размещать информацию о прямом падении напряжения на кремниевых диодах, т. к. оно одно и то же для всех без исключения типов, в то же время такие, к примеру, параметры, как емкость коллекторного перехода или граничное напряжение транзисторов просто бесполезны для радиолюбителя и потому также не приводятся.

Такие характеристики, как предельно допустимая мощность рассеяния для транзисторов малой (но не средней и не большой!) мощности, также на практике требуются очень редко (можете спокойно принимать ее равной примерно 0,2—0,7 Вт, в зависимости от корпуса). Не привожу я и данные логических микросхем, т. к. их характеристики в целом одинаковы и описаны в *главе 8*, а разводка выводов приведена на ряде схем и в тексте. Исключение сделано только для сводной таблицы (табл. П3.1) соответствия отечественных и импортных КМОП-микросхем (базовой серии), потому что эту информацию разыскать не всегда просто.

Источники, откуда взяты приведенные данные, слишком многочисленны (это и официальные справочники, и справочные данные, размещенные в литературе

и в Интернете, и информация из каталогов торгующих организаций, и фирменная документация, как в виде бумажных каталогов, так и в формате PDF), чтобы их перечислить, — вы не поверите, сколько бумажной «макулатуры» пришлось «перелопатить» и сколько интернет-часов потратить с целью найти, систематизировать и привести к более-менее единому «знаменателю» данные даже о таком небольшом количестве компонентов. Отечественные производители (и некоторые зарубежные тоже) просто «сговорились» в своем стремлении скрыть от пользователя как можно больше критически важных параметров. Часто для компонентов, аналогичных по назначению, но от разных производителей, приводится разный набор параметров, из-за чего сравнение приборов становится довольно нетривиальной задачей. Этот недостаток я по мере возможности также постарался устранить.

Заметки на полях

Яркий пример такой ситуации, к нашему изложению, правда, отношения не имеющий: характеристики чувствительности светочувствительных матриц для цифровых фотоаппаратов приводятся в единицах ISO, как для фотопленок, а для цифровых видеокамер — в единицах освещенности, хотя матрицы часто одни и те же! В нашем же случае можно привести пример статического коэффициента передачи тока для транзисторов, который у одних приборов приводится для режима, близкого к насыщению, у других — для активного режима, есть и третьи, не очень понятные случаи, поэтому я величину $U_к$, при которой производились измерения β, просто опускаю, т. к. это только запутывает неискушенного пользователя, не давая никакой реальной информации, т. к. справочное значение коэффициента все равно есть величина ориентировочная.

Следует учесть, что предпочтение тут отдано отечественным компонентам, т. к. нахождение справочных данных для них представляет наибольшие трудности. Конечно, информация о производителе того или иного компонента, особенно отечественного, ориентировочная: кроме приведенных, это могут быть и другие фирмы, а отечественные заводы могли прекратить выпуск продукции с тех пор, когда была собрана информация.

Обращаю внимание, что это издание ни в коем случае не является официальным, и поэтому помещенные данные не могут служить основанием для претензий к тому или иному производителю в случае их несоответствия реальным характеристикам того или иного компонента.

Соответствие наименований зарубежных и отечественных микросхем КМОП

Таблица П3.1

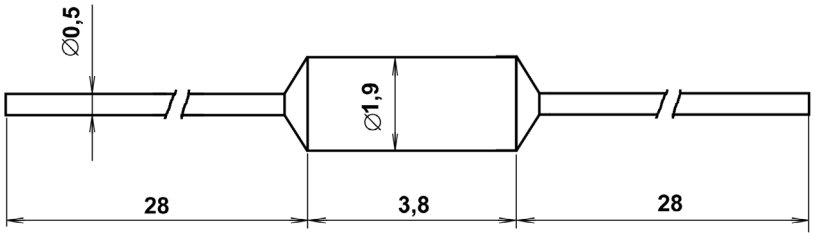
4000	К561	4000	К561	4000	К561
CD4000	ЛП4	CD4023	ЛА9	CD4061	РУ2
CD4001	ЛЕ5	CD4025	ЛЕ10	CD4066	КТ3
CD4002	ЛЕ6	CD4027	ТВ1	CD4093	ТЛ1
CD4006	ИР10	CD4028	ИД1	CD4098	АГ1
CD4007	ЛП1	CD4029	ИЕ14	CD40107	ЛА10
CD4008	ИМ1	CD4030	ЛП2	CD40108	ИР12
CD4009	ПУ2	CD4034	ИР6	CD40109	ПУ6
CD4010	ПУ3	CD4035	ИР9	CD40181	ИП3
CD4011	ЛА7	CD4039	РП1	CD40182	ИП4
CD4012	ЛА8	CD4042	ТМ3	MC14502	ЛН1
CD4013	ТМ2	CD4043	ТР2	MC14516	ИЕ11
CD4015	ИР2	CD4046	ГГ1	MC14520	ИЕ10
CD4016	КТ1	CD4049	ЛН2	MC14531	СА1
CD4017	ИЕ8	CD4050	ПУ4	MC14554	ИП5
CD4018	ИЕ19	CD4051	КП2	MC14580	ИР12
CD4019	ЛС2	CD4052	КП1	MC14581	ИП3
CD4020	ИЕ16	CD4056	ИД5	MC14582	ИП4
CD4022	ИЕ9	CD4059	ИЕ15	MC14585	ИП2

Примечание

CD4xxxx — наименование серии 4000, принятое в фирме Fairchild Semiconductor, MC14xxxx — в фирме Motorola. У других фирм могут быть свои префиксы, например, HCC или HCF — у фирмы ST Microelectronics. Сами обозначения микросхем остаются одинаковыми независимо от производителя.

Диоды

КД521



Кремниевые диоды.

Производитель: неизвестен (СНГ).

Назначение: предназначены для применения в импульсных устройствах.

Выпускаются в стеклянном корпусе с гибкими выводами. Тип и полярность диодов обозначается одной широкой и двумя узкими цветными полосками на корпусе со стороны положительного (анодного) вывода: КД521А — полосы синие; КД521Б — серые; КД521В — желтые; КД521Г — белые; КД521Д — зеленые. В табл. П3.2 приведены предельные значения обратного напряжения.

Таблица П3.2

Тип	КД521А	КД521Б	КД521В	КД521Г	КД521Д
Макс. постоянное обратное напряжение, В	75	60	50	30	12

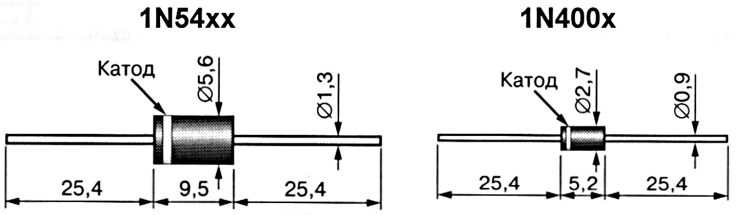
Постоянный или средний прямой ток при T = –60…+50 °С, не более … 50 мА.

Перегрузка по прямому току не более 5 мин при T = –60…+50 °С … 200 мА.

Постоянный обратный ток при $U_{обр} = U_{max}$:

- при T = –60 и +25 °С … 1 мкА;
- при T = +125 °С … 100 мкА.

1Nxxxx



Кремниевые диоды.

Производитель: Fairchild Semiconductor (США) и др.

Назначение: выпрямление переменного тока.

В табл. П3.3 приведены предельные значения токов и напряжений.

Таблица П3.3

***U*, В** ***I*, А**	**50**	**100**	**200**	**400**	**600**	**800**	**1000**
1,0	1N4001	1N4002	1N4003	1N4004	1N4005	1N4006	1N4007
3,0	1N5400	1N5401	1N5402	1N5404	1N5406	1N5407	1N5408

Примечание 1. Частота выпрямления до 1 МГц.

Примечание 2. Для диодов 1N54xx максимальная рассеиваемая мощность 6,25 Вт, макс. импульсный ток (одиночная синусоидальная полуволна 8,3 мс) –200 А.

КД 202

Кремниевые диоды.

Производитель: неизвестен (СНГ).

Назначение: выпрямление переменного тока.

В табл. П3.4 приведены предельные значения обратного напряжения.

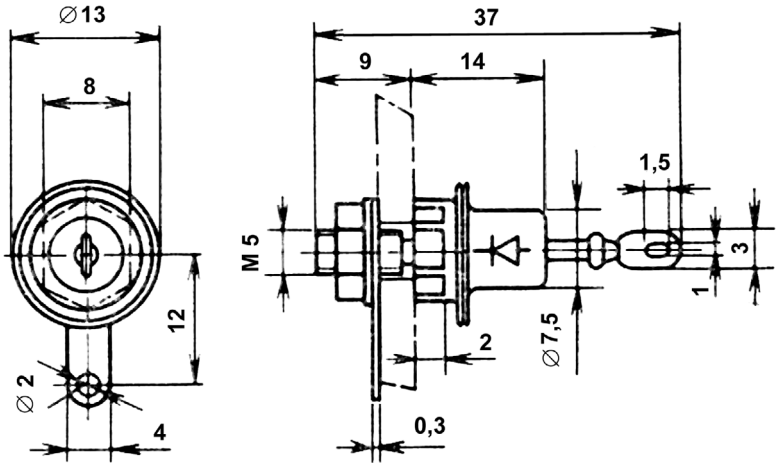


Таблица П3.4

Тип	КД202 А	КД202 В	КД202 Д	КД202 Ж	КД202 К	КД202 М	КД202 Р
Макс. обратное напряжение, В	50	100	200	300	400	500	600

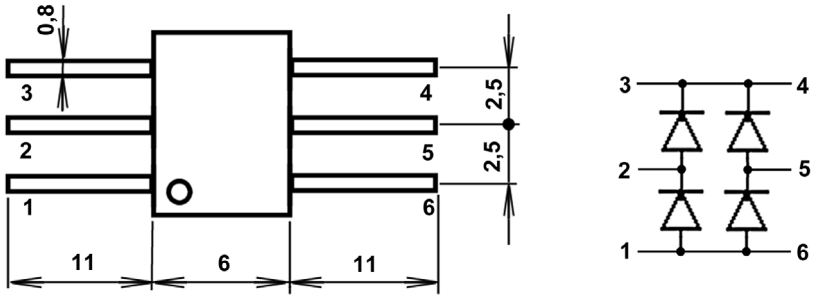
Постоянное обратное напряжение … 0,7 $U_{\text{обр. макс.}}$

Постоянный (средний) прямой ток при T = –60 до +75 °С … 5 А.

Макс. импульсный ток (одиночная синусоидальная полуволна 10 мс) … 30 А.

Частота выпрямления до 1,2 кГц.

КЦ 407А



Кремниевый выпрямительный мост.

Производитель: неизвестен (СНГ).

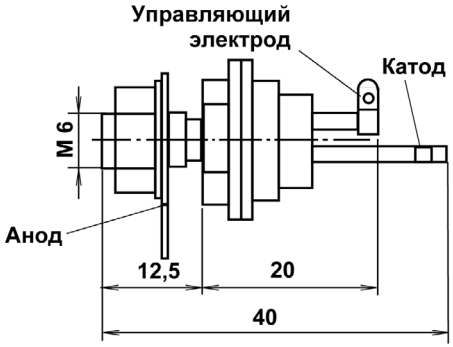
Назначение: выпрямление переменного тока.

Максимальное обратное напряжение … 400 В.

Средний выпрямленный ток при *T* = –60…+55 °С, не более … 500 мА.

Частота выпрямления до 20 кГц.

КУ202Н



Кремниевый тиристор.

Производитель: неизвестен (СНГ).

Назначение: ключи средней мощности.

Напряжение в открытом состоянии … 1,5 В.

Импульсное отпирающее напряжение на управляющем электроде
при $t_{отп}$ > 10 мс … 7 В.

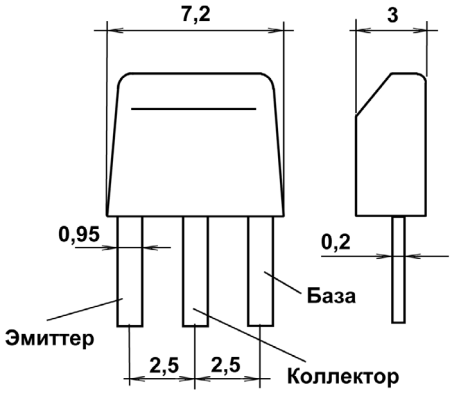
Импульсный отпирающий ток управляющего электрода
при $t_{отп}$ > 10 мс, не более … 200 мА.

Постоянное прямое напряжение в закрытом состоянии, не более … 400 В.

Постоянный ток в открытом состоянии при *T* < 50 °С, не более … 10 А.

Транзисторы

КТ315, КТ361



Транзисторы кремниевые малой мощности высокочастотные.

Производитель: «Кремний» (г. Брянск, Россия), НИИПП (г. Томск, Россия).

Структура: *n-p-n* (КТ315), *p-n-p* (КТ361).

Тип прибора указывается на корпусе в виде буквы номинала: для КТ315 буква слева, для КТ361 увеличенная и посередине корпуса.

Назначение: усилители НЧ и ВЧ.

Тип корпуса: КТ-13.

Статический коэффициент передачи тока ($I_к$ = 1 мА):

КТ315А, КТ315В … 30—120;
КТ315Б, КТ315Г, КТ315Е,
КТ361Б, КТ361Г, КТ361Е … 50—350;
КТ315Д, КТ361А, КТ361Д … 20—90;
КТ315Ж … 30—250;
КТ315И … >30;
КТ361В … 40—160.

Макс. постоянное напряжение коллектор-эмиттер, В:

КТ315А, КТ361А … 25;
КТ315Б, КТ315Ж, КТ361Б … 20;
КТ315В, КТ315Д, КТ361В, КТ361Д … 40;

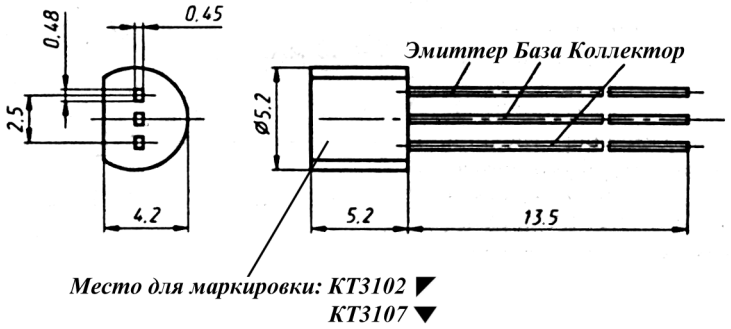
КТ315Г, КТ315Е, КТ361Г, КТ361Е … 35;
КТ315И … 60.

Максимальный постоянный ток коллектора, мА:

КТ315А, КТ315Б, КТ315В,
КТ315Г, КТ315Д, КТ315Е … 100;
КТ361 (все буквы), КТ315Ж, КТ315И … 50.

Граничная частота … 250 МГц.

КТ3102, КТ3107



Транзисторы кремниевые малой мощности высокочастотные.

Производитель: АО «Светлана» (г. Санкт-Петербург, Россия), АО «Кремний» (г. Брянск, Россия), Нальчикский завод полупроводниковых приборов (г. Нальчик, Россия).

Структура: *n-p-n* (КТ3102), *p-n-p* (КТ3107).

Тип прибора указывается на корпусе или непосредственно, или на торце корпуса наносится метка: КТ3102А — темно-красная, КТ3102Б — желтая, КТ3102В — темно-зеленая, КТ3102Г — голубая, КТ3102Д — синяя, КТ3102Е — белая, КТ3102Ж — две темно-красные, КТ3102И — две желтые, КТ3102К — две темно-зеленые, КТ3107А — голубая и розовая, КТ3107Б — голубая и желтая; КТ3107В — голубая и синяя; КТ3107Г — голубая и бежевая; КТ3107Д — голубая и оранжевая; КТ3107Е — голубая и цвета электрик; КТ3107Ж — голубая и салатовая; КТ3107И — голубая и зеленая; КТ3107К — голубая и красная; КТ3107Л — голубая и серая. Другой способ маркировки состоит в том, что тип транзистора указывается условным значком-треугольником (см. рисунок корпуса). Иногда вместо этих значков ставится просто цифра (2 или 7). Кроме этого, на маркировочной поверхности ставится

буква, соответствующая группе (А, Б, В и т. п.) и иногда еще дополнительно два значка, условно означающие год и месяц выпуска (например, М 6 означает июнь 2000).

Назначение: для низкочастотных устройств с малым уровнем шумов, переключающих, усилительных и генераторных устройств средней и высокой частоты.

Тип корпуса: КТ-26 (ТО-92).

Статический коэффициент передачи тока при ($I_э$ = 2 мА; T = +25 °С):

КТ3102А, КТ3102Ж … 100—250;
КТ3102Б, КТ3102В, КТ3102Г, КТ3102И, КТ3102К … 200—500;
КТ3102Г, КТ3102Е … 400—1000;
КТ3107А, КТ3107В … 70—140;
КТ3107Б, КТ3107Г, КТ3107Е … 120—220;
КТ3107Д, КТ3107Ж, КТ3107И … 180—460;
КТ3107К, КТ3107Л … 380—800.

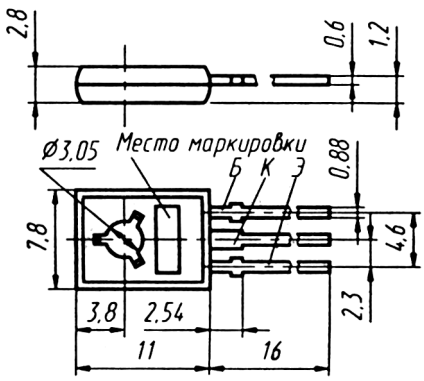
Макс. постоянное напряжение коллектор-эмиттер, В:

КТ3102А, КТ3102Б, КТ3102Ж, КТ3102И … 50;
КТ3102В, КТ3102Д, КТ3102К … 30;
КТ3102Г, КТ3102Е, КТ3107Е, КТ3107Ж, КТ3107Л … 20;
КТ3107А, КТ3107Б, КТ3107И … 45;
КТ3107В, КТ3107Г, КТ3107Д, КТ3107К … 25.

Максимальный постоянный ток коллектора, мА … 100.

Граничная частота … 200 МГц (КТ3102А—Е ...150 МГц).

КТ814, КТ815, КТ816, КТ817



Транзисторы кремниевые средней мощности низкочастотные.

Производитель: АО «Кремний» (г. Брянск, Россия), «Искра» (г. Ульяновск, Россия).

Структура: *n-p-n* (КТ815, КТ817), *p-n-p* (КТ814, КТ816).

Тип прибора указывается на корпусе.

Назначение: усилители низкой частоты, импульсные устройства.

Тип корпуса: КТ-27 (ТО-126).

Статический коэффициент передачи тока при ($I_э$ = 1 А; T = +25 °С):

КТ814А-В, КТ815А—В … 40;
КТ814Г, КТ815Г … 30;
КТ816А-Г, КТ817А—Г … 25.

Макс. постоянное напряжение коллектор-эмиттер, В:

КТ814А, КТ816А, КТ817А … 25;
КТ814Б, КТ815А … 40;
КТ816Б, КТ817Б … 45;
КТ815Б … 50;
КТ814В, КТ816В, КТ817В … 60;
КТ815В … 70;
КТ814Г, КТ816Г, КТ817Г … 80;
КТ815Г … 100.

Максимальный постоянный ток коллектора, А:

КТ814, КТ815 … 1,5;
КТ816, КТ817 … 3.

Постоянная рассеиваемая мощность коллектора, Вт:

Без теплоотвода КТ814 – КТ817 … 1;
С теплоотводом КТ814, КТ815 … 10;
С теплоотводом КТ816, КТ817 … 25.

Граничная частота … 3 МГц.

КТ972, КТ973

Чертеж корпуса и разводку выводов см. КТ814—КТ817.

Транзисторы кремниевые средней мощности высокочастотные структуры Дарлингтона.

Производитель: «Транзистор» (г. Минск, Белоруссия).

Структура: *n-p-n* (КТ972), *p-n-p* (КТ973).

Тип прибора указывается на корпусе (зависит от производителя).

Назначение: управление импульсными устройствами.

Тип корпуса: КТ-27 (ТО-126).

Статический коэффициент передачи тока при ($I_э$ = 1 А; T = +25 °С):

КТ972А,Б, КТ973А,Б … 750.

Макс. постоянное напряжение коллектор-эмиттер, В:

КТ972А, КТ973А … 60;
КТ972Б, КТ973Б … 45.

Максимальный постоянный ток коллектора … 4 А.

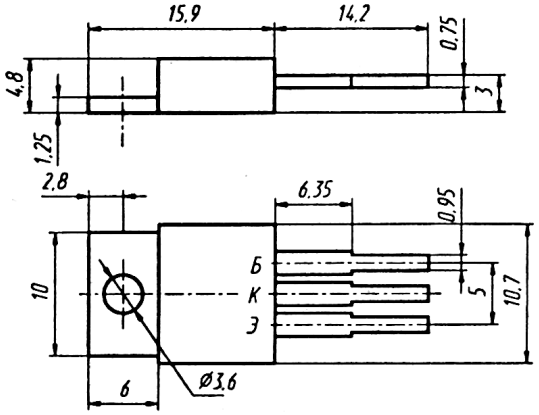
Постоянная рассеиваемая мощность коллектора (с теплоотводом) … 8 Вт.

Напряжение насыщения коллектор-эмиттер … 1,5 В.

Напряжение насыщения база-эмиттер … 2,5 В.

Граничная частота … 100 МГц.

КТ818, КТ819



Транзисторы кремниевые большой мощности низкочастотные.

Производитель: АО «Кремний» (г. Брянск, Россия).

Структура: *n-p-n* (КТ819), *p-n-p* (КТ818).

Тип прибора указывается на корпусе.

Назначение: усилители НЧ и переключающие устройства.

Тип корпуса: КТ-28 (ТО-220). Есть модификации в металлическом корпусе типа ТО-3, а также ТО-218 с шагом выводов 5,5 мм.

Минимальный статический коэффициент передачи тока при ($I_э$ = 5 А; T = +25 °С):

КТ818А, КТ818В, КТ819А, КТ819В … 15;
КТ818Б, КТ819Б … 20;
КТ818Г, КТ819Г … 12.

Макс. постоянное напряжение коллектор-эмиттер, В:

КТ818А, КТ819А … 40;
КТ818Б, КТ819Б … 50;
КТ818В, КТ819В … 70;
КТ818Г … 90;
КТ819Г … 100.

Максимальный постоянный ток коллектора … 10 А.

Постоянная рассеиваемая мощность коллектора (с теплоотводом) … 60 Вт.

Граничная частота … 3 МГц.

КТ829

Чертеж корпуса и разводку выводов см. КТ818, КТ819.

Транзистор кремниевый большой мощности низкочастотный структуры Дарлингтона.

Производитель: АО «Элиз» (г. Фрязино, Россия).

Структура: *n-p-n*.

Тип прибора указывается на корпусе.

Назначение: усилители НЧ и переключающие устройства.

Тип корпуса: КТ-28 (ТО-220).

Статический коэффициент передачи тока при ($I_э$ = 3 А; T = +25 °С) … 750.

Макс. постоянное напряжение коллектор-эмиттер, В:

КТ829А … 100;

КТ829Б … 80;

КТ829В … 60;

КТ829Г … 45.

Максимальный постоянный ток коллектора … 8 А.

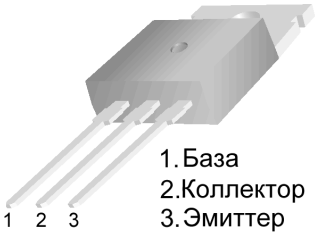
Постоянная рассеиваемая мощность коллектора (с теплоотводом) … 60 Вт.

Напряжение насыщения коллектор-эмиттер … 2 В.

Напряжение насыщения база-эмиттер … 2,5 В.

Граничная частота … 10 МГц.

BDW93, BDW94



Транзисторы кремниевые большой мощности низкочастотные структуры Дарлингтона.

Производитель: Fairchild Semiconductor (США), ST Microelectronics (Европа).

Структура: *n-p-n* (BDW93), *p-n-p* (BDW94).

Тип прибора указывается на корпусе.

Тип корпуса: TO-220.

Минимальный статический коэффициент передачи тока:

при $I_к$ = 3 А … 1000.
при $I_к$ = 10 А … 100.

Максимальный статический коэффициент передачи тока … 20 000.

Макс. постоянное напряжение коллектор-эмиттер, В:

BDW93, BDW94 … 45;

BDW93A, BDW94A … 60;

BDW93B, BDW94B … 80;

BDW93C, BDW94C … 100.

Максимальный постоянный ток коллектора … 12 А.

Постоянная рассеиваемая мощность коллектора (с теплоотводом):

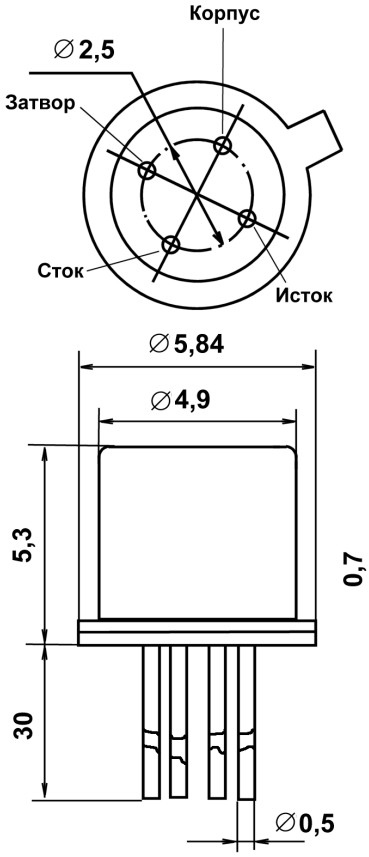
при T = +25 °C … 60 Вт;
при T = +100 °C … 35 Вт.

Напряжение насыщения коллектор-эмиттер (макс.) … 3 В.

Напряжение насыщения база-эмиттер (макс.) … 4 В.

Граничная частота[1] … 1 МГц.

КП303



Транзистор кремниевый полевой малой мощности с затвором на основе *p*-*n*-перехода и каналом *n*-типа.

[1] На основе косвенных данных.

Производитель: АООТ «Элекс» (г. Александров, Россия).

Тип прибора указывается на корпусе.

Назначение: общего назначения.

Тип корпуса: КТ-1-12 (ТО-17).

Начальный ток стока (при $U_{зи}$ = 0), мА:

КП303А, КП303Б … 0,5—2,5;
КП303В, КП303И … 1,5—5;
КП303Г … 3—12;
КП303Д … 3—9;
КП303Е … 5—20;
КП303Ж … 0,3—3.

Напряжение отсечки (при Iс = 10 мкА), В:

КП303А, КП303Б … 0,5—3;
КП303В … 1—4;
КП303Г, КП303Д, КП303Е … < 8;
КП303Ж … 0,3—3;
КП303И … 0,5—2.

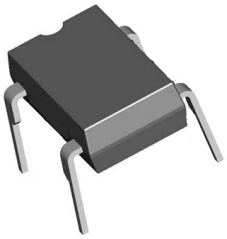
Максимальное напряжение сток-исток, В … 25.

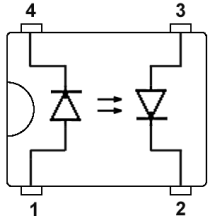
Максимальное напряжение затвор-сток, В … 30.

Максимальный постоянный ток стока … 20 мА.

Электронные реле и оптроны

АОД130





Диодный оптрон.

Производитель: з-д «Протон» (Россия).

Тип прибора указывается на корпусе.

Тип корпуса: модифицированный DIP-8.

Постоянное прямое входное напряжение, В …1,5.

Максимальный входной ток, мА … 20.

Максимальный импульсный входной ток, мА …100.

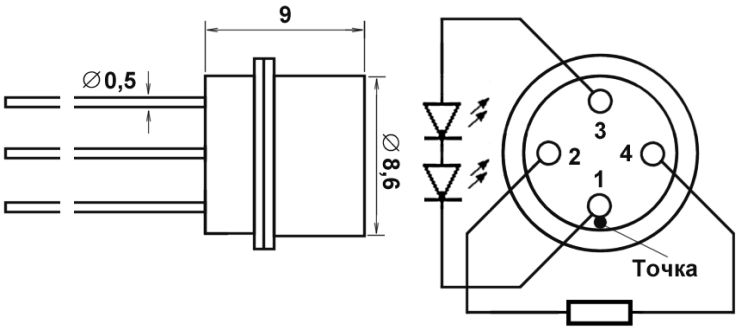
Максимальное входное обратное напряжение, В … 3,5.

Максимальное выходное обратное напряжение, В … 30.

Коэффициент передачи тока, % … 1.

Максимальное напряжение изоляции, В … 1500.

АОР124Б



Резисторный оптрон со светодиодом на входе.

Производитель: неизвестен (Россия).

Тип прибора указывается на корпусе.

Тип корпуса: советский вариант ТО-39.

Постоянное прямое входное напряжение, В … 3,8.

Максимальный входной ток, мА … 15.

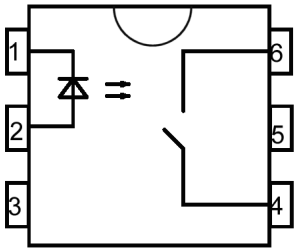
Максимальный импульсный входной ток, мА …100.

Максимальное входное обратное напряжение, В … 6.

Максимальный выходной ток, мА … 9.

Максимальное напряжение изоляции, В … 1000.

КР293КП1 (5П14)



Электронное реле малой мощности.

Производитель: з-д «Протон» (Россия).

Тип прибора указывается на корпусе.

Тип корпуса: DIP-6.

Постоянное прямое входное напряжение, В …1,5.

Максимальный входной ток, мА … 25.

Минимальный входной ток, мА … 5.

Максимальный импульсный входной ток, мА …150.

Максимальное коммутируемое напряжение, В:

КР293КП1А … 60;

КР293КП1Б … 230;

КР293КП1И … 400.

Максимальный коммутируемый ток, мА:

КР293КП1А … 250;

КР293КП1Б … 100;

КР293КП1И … 80.

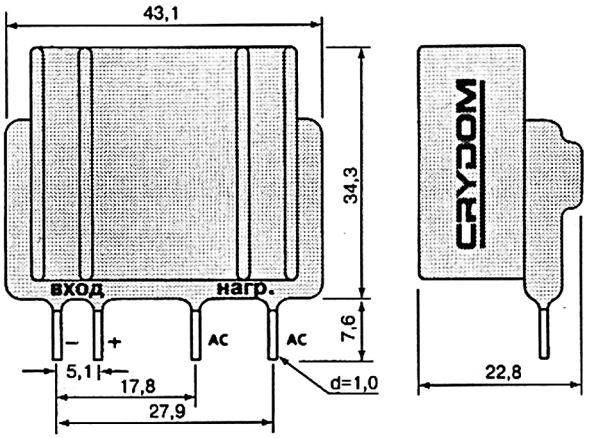
Максимальное напряжение изоляции, В … 1500.

PF240D25

Электронное реле большой мощности.

Производитель: CRYDOM (San Diego, США).

Тип прибора указывается на корпусе.



Управляющее входное напряжение, В …3—15.

Входной ток при упр. напряж. 5 В, мА … 15.

Максимальное коммутируемое напряжение, В … 12—280 (действ. значение).

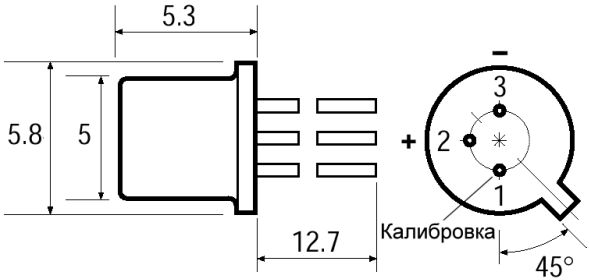
Коммутируемый ток, А … 0,06—25.

Перегрузка по току (1 полупериод 50—60 Гц), А … 250.

Примечание. При коммутируемых токах более 10 А требуется принудительный обдув корпуса.

Микросхемы

1019ЕМ1



Термодатчик с линейной зависимостью выходного напряжения от температуры.

Производитель: АООТ «Фотон» (г. Ташкент, Узбекистан).

Тип прибора указывается на корпусе.

Тип корпуса: TO-18.

Ток питания, мА … 0,5...1,5.

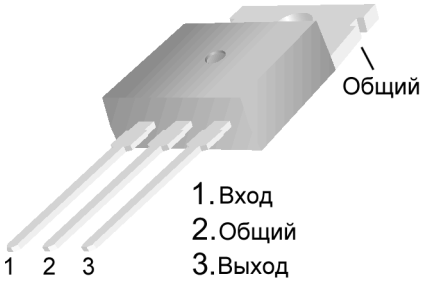
Выходное напряжение, мВ при токе питания 1 мА и температуре:

298 К (25 °С) … 2952—3012;
398 К (125 °С) … 3932...4032;
263 К (–10 °С) … 2582...2682 для К1019ЕМ1А;
228 К (–45 °С) … 2232...2332 для К1019ЕМ1.

Рабочий температурный интервал, °С:

К1019ЕМ1 … от –45 до +125;
К1019ЕМ1А … от –10 до +125.

7805, 7809, 7812, 7815, 7905, 7909, 7912, 7915



Стабилизаторы постоянного тока большой мощности.

Производитель: National Semiconductor (США), Fairchild Semiconductor (США), ST Microelectronics (Европа).

Отечественные аналоги:

КР142ЕН5А (7805), КР142ЕН8А (7809), КР142ЕН8Б (7812), КР142ЕН8В (7815), КР1162ЕН5 (7905), КР1162ЕН9 (7909), КР1162ЕН12 (7912), КР1162ЕН15 (7915).

Тип прибора указывается на корпусе.

Тип корпуса: TO-220.

Выходное напряжение, В:

7805 … +5 ± 0,2;
7809 … +9 ± 0,36;

7812 … +12 ± 0,5;
7815 … +15 ± 0,6;
7905 … –5 ± 0,2;
7909 … –9 ± 0,36;
7912 … –12 ± 0,5;
7915 … –15 ± 0,6.

Максимальный постоянный выходной ток (внутреннее ограничение), А:

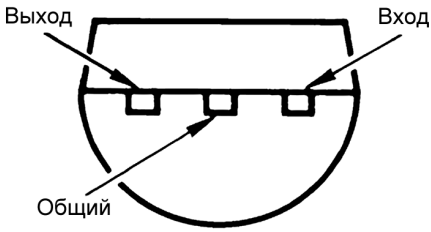
7805, 7905 … 2,1;
7809, 7909 … 1,8;
7812, 7912 … 1,5;
7815, 7915 … 1,2.

Максимальное входное напряжение, В … 35 (для 78xx), –35 (для 79xx).

Максимальная рассеиваемая мощность, Вт (*T* = 0 …+75 °С):

без теплоотвода … 3;
с теплоотводом … 20.

78L05, 78L09, 78L12, 78L15, 79L05, 79L09, 79L12, 79L15



Стабилизаторы постоянного тока малой мощности.

Производитель: National Semiconductor (США), Fairchild Semiconductor (США), ST Microelectronics (Европа).

Отечественные аналоги:

КР1157ЕН5 (78L05), КР1157ЕН9 (78L09), КР1157ЕН12 (78L12), КР1157ЕН15 (78L15), КР1168ЕН5 (79L05), КР1168ЕН9 (79L09), КР1168ЕН12 (79L12), КР1168ЕН15 (79L15).

Тип прибора указывается на корпусе.

Тип корпуса: TO-92.

Выходное напряжение, В:

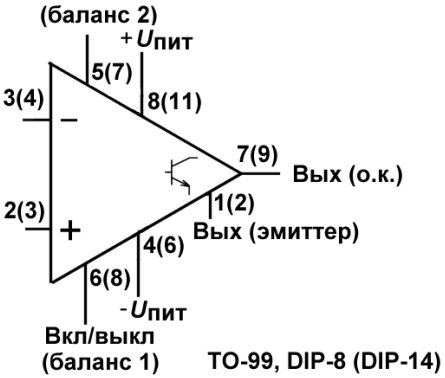
78L05 … +5 ± 0,2;
78L09 … +9 ± 0,36;
78L12 … +12 ± 0,5;
78L15 … +15 ± 0,6;
79L05 … –5 ± 0,2;
79L09 … –9 ± 0,36;
79L12 … –12 ± 0,5;
79L15 … –15 ± 0,6.

Максимальный постоянный выходной ток, мА … 100.

Максимальное входное напряжение, В … 35 (для 78xx), –35 (для 79xx).

Максимальная рассеиваемая мощность, Вт:
при T = 0 °C … 1 Вт;
при T = +75 °C … 0,5 Вт.

LM311 (521СА3, 554СА3)



Быстродействующий компаратор.

Производитель: National Semiconductor (США).

Тип прибора указывается на корпусе.

Тип корпуса: DIP-8, TO-99 (металлический круглый), DIP-14.

Максимальное напряжение питания, В … ±18 или +36.

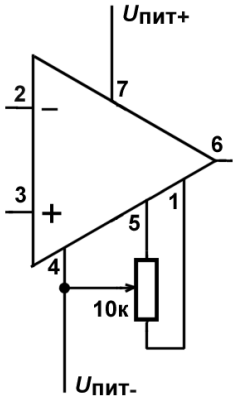
Максимальное выходное напряжение (коллектор – отриц. напр. пит.), В … 50.

Максимальный выходной ток, мА … 50.

Входной ток смещения, нА … <10.

Входное напряжение сдвига при сопротивлении источника <10 кОм, мВ … <2.

μA741 (140УД7)



Операционный усилитель общего назначения.

Производитель: Philips Semiconductors (Европа), ST Microelectronics (Европа) и др.

Тип прибора указывается на корпусе.

Тип корпуса: DIP-8.

Максимальный ток потребления ($R_н = \infty$, $U_{пит} = \pm 15$ В), мА … 2,8.

Напряжение питания, В … от ±5 до ±18.

Максимальное входное напряжение, В … ±15.

Максимальное выходное напряжение ($U_{пит} = \pm 15$ В), В:

Rн = 10 кОм … ±14;
Rн = 2 кОм … ±13.

Выходное сопротивление, Ом … 75.

Коэффициент усиления … 50 000—200 000.

Входной ток смещения, нА … 80—500.

Разность входных токов смещения, нА … 20—200.

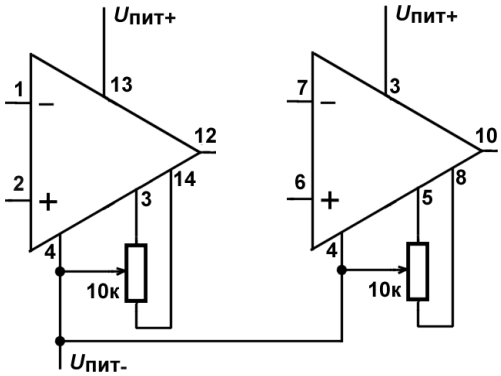
Входное напряжение сдвига при сопротивлении источника <10 кОм, мВ … 2.

Температурный дрейф напряжения сдвига (ТКЕ), мкВ/°С … 10.

Частота единичного усиления, МГц … 1.

Скорость нарастания, В/мкс … 0,5.

μA747 (140УД20)



Два операционных усилителя типа μA741 в одном корпусе.

Производитель: Texas Instruments (США) и др.

Тип прибора указывается на корпусе.

Тип корпуса: DIP-14.

Максимальный ток потребления ($R_н = ∞$, $U_{пит} = ±15$ В), мА … 5,5.

Остальные характеристики — см. μA741.

MAX478

Два прецизионных операционных усилителя в одном корпусе.

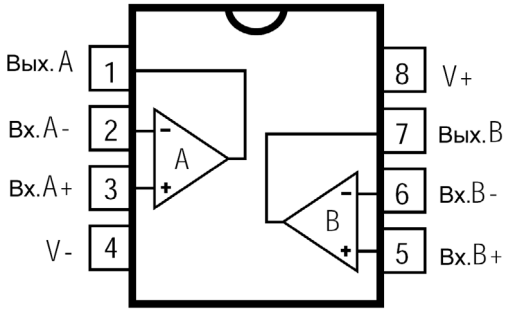
Производитель: MAXIM/DALLAS (США).

Тип прибора указывается на корпусе.

Тип корпуса: DIP-8 (SO-8).

Максимальный ток потребления ($R_н = ∞$, $U_{пит} = ±15$ В), мкА … 50.

Напряжение питания, В … от ±2,2 до ±18.



Максимальное входное напряжение, В … $\pm U_{пит}$.

Максимальное выходное напряжение ($U_{пит}$ = ±15 В, $R_н$ = 2 кОм), В … … ±12,7.

Коэффициент усиления … 300 000—1200 000.

Входной ток смещения, нА … 5.

Разность входных токов смещения, нА … 0,25.

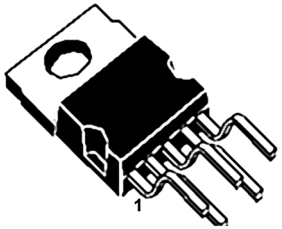
Входное напряжение сдвига при сопротивлении источника <10 кОм, мкВ … 100.

Температурный дрейф напряжения сдвига (ТКЕ), мкВ/°С … 0,9.

Частота единичного усиления, кГц … 60.

Скорость нарастания, В/мкс … 0,02.

TDA3020



Усилитель мощности звуковой частоты.

Производитель: ST Microelectronics (Европа).

Тип прибора указывается на корпусе.

Тип корпуса: пятивыводной TO-220.

Напряжение питания, В … от ±6 до ±18.

Максимальный выходной ток, А … 3,5.

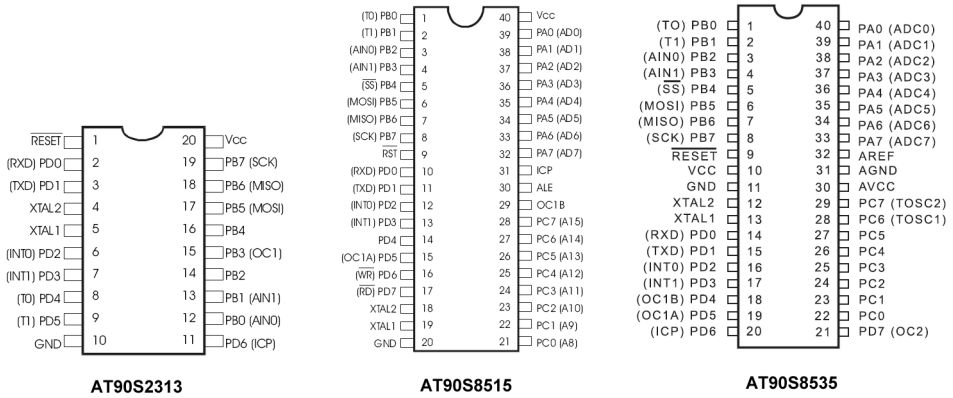
Максимальная рассеиваемая мощность, Вт … 20.

Входной ток смещения, мкА … 2.

Входное напряжение сдвига, мВ … 20.

Ширина полосы пропускания по уровню 3 дБ, Гц … 10—140 000.

AT90S2313, AT90S8515, AT90S8535



Микроконтроллеры AVR. Микросхемы с буквой «L» в наименовании обладают пониженным энергопотреблением. Как вы знаете из *главы 12*, в настоящее время указанные типы микроконтроллеров AVR «Classic» заменяются на более развитые версии, в названиях которых присутствуют приставки «Mega» или «Tuny» (например, ATmega8515), но электрические характеристики у них одни и те же, за исключением быстродействия (до 16 МГц, а в случае ATtuny2313 до 20 МГц) и в некоторых случаях расширенного диапазона питания (до 1,8 В).

Производитель: Atmel (США).

Тип прибора указывается на корпусе.

Тип корпуса: DIP-20 (AT90S2313), DIP-40 (AT90S8515, AT90S8535) и др.

Основные электрические параметры

Напряжение питания, В … 2,7—6,0.

Входное напряжение лог. 0, не более, В … $0{,}2U_{пит}$.

Входное напряжение лог. 1, не менее, В … $0{,}6U_{пит}$.

Входное напряжение лог. 1 выводов XTAL, не менее, В … $0{,}8U_{пит.}$

Входное напряжение лог. 1 вывода Reset, не менее, В … $U_{пит}$.

Максимальное входное напряжение лог. 1, В … $U_{пит}$ + 0,5 В.

Минимальное входное напряжение лог. 0, В … –0,5 В.

Выходное напряжение лог. 0, не более, В … 0,6.

Выходное напряжение лог. 1, не менее, В:

$U_{пит}$ = 5 В … 4,2;
$U_{пит}$ = 3 В … 2,3.

Потребляемый ток в режиме:

$U_{пит}$ = 3 В, $F_т$ = 4 МГц … 3,5 мА;
Power Down … 1 мкА;
Power Down + WDT … 50 мкА.

Входной ток утечки в режиме входа, мкА … 8.

Нагрузочный резистор по входу, кОм … 35—120.

Входное напряжение сдвига аналогового компаратора, мВ … 20.

Входной ток смещения аналогового компаратора, нА … 10.

Максимальный ток через вывод питания или «земли», мА:

AT90S2313, AT90S8515 … 140;
AT90S8535 … 200.

Максимально допустимый ток одного вывода порта, мА … 40.

Максимальный постоянный ток одного вывода порта, мА … 20.

Максимально допустимый суммарный ток всех выводов порта, мА:

для AT90S2313 … 80 (каждый из портов B и D);
для AT90S8515 … 80 (порт A), 80 (порты B, C и D в сумме);
для AT90S8535 … 100 (порт A), 100 (порты B, C и D в сумме).

Максимальная частота тактового генератора, МГц:

при $U_{пит}$ = 2,7—6,0 В … 10 (AT90S2313), 4 (AT90S8515, AT90S8535);
при $U_{пит}$ = 4,0—6,0 В … 16 (AT90S2313), 8 (AT90S8515, AT90S8535).

Приложение 4

Базовые команды Atmel AVR

Система команд микроконтроллеров Atmel AVR довольно обширна и включает в себя от 90 до 133 команд, в зависимости от разновидности микроконтроллера. Далее приводится выборочный перечень наиболее часто употребляемых команд по группам. Приведенных команд в принципе достаточно для того, чтобы составить большинство законченных программ для МК AVR, хотя многие полезные, но редко употребляемые команды (такие как `EOR`, `COM`, `NEG` и др.), здесь отсутствуют. Поэтому для полноценной работы следует иметь полный справочник по командам. Краткие таблицы команд прилагаются ко всем описаниям МК, полный перечень команд на русском имеется в пособиях [1] и [2] (берегитесь неточностей, которые встречаются в первых изданиях этих пособий!). Официальный перечень команд на английском (AVR Instruction Set) можно скачать с сайта **atmel.com** в виде PDF-документа.

Будьте внимательны при работе с командами: следует обращать внимание на то, что некоторые команды могут быть применены только к определенным регистрам (к примеру, операции с константами работают только со старшими 16 регистрами, начиная с `R16`), а сами константы не всегда могут иметь полный диапазон значений. Поэтому следует внимательно смотреть характеристики команд, прежде чем использовать их в программе. Команды, помеченные серым цветом, действительны не для всех моделей AVR (для семейства Mega, как правило, действительны все, но стоит уточнить по описанию конкретного контроллера).

В табл. П4.1—П4.6 приняты следующие сокращения: РОН (регистр общего назначения), РВВ — регистр ввода/вывода, РС — счетчик команд (программный счетчик, Program Counter). Буквой `C` в регистре флагов `SREG` обозначается флаг переноса (устанавливается при возникновении переноса при арифметических операциях), буквой `Z` — флаг нуля (устанавливается по равенству операндов при сравнении). Последний не путать с парой регистров `R31:R30`,

которые задействованы в командах переноса данных, и также обозначаются буквой Z. В этих командах используются также обозначения X (пара регистров R27:R26) и Y (пара регистров R29:R28).

Арифметические и логические команды

Арифметические и логические команды приведены в табл. П4.1.

Таблица П4.1

Команда	Операнды	Описание	Операция
ADD	Rd, Rr	Сложение двух РОН без учета переноса	Rd ← Rd + Rr d = 0..31 r = 0..31
ADC	Rd, Rr	Сложение двух РОН с учетом переноса	Rd ← Rd + Rr + C d = 0..31 r = 0..31
ADIW	Rd, K	Сложение регистровой пары с константой	Rd+1:Rd ← Rd+1:Rd + K d = 24,26,28,30 K = 0..63
SUB	Rd, Rr	Вычитание двух РОН без учета переноса	Rd ← Rd – Rr d = 0..31 r = 0..31
SBC	Rd, Rr	Вычитание двух РОН с учетом переноса	Rd ← Rd – Rr – C d = 0..31 r = 0..31
SBIW	Rd, K	Вычитание константы из регистровой пары	Rd+1:Rd ← Rd+1:Rd – K d = 24,26,28,30 K = 0..63
SUBI	Rd, K	Вычитание константы из регистра	Rd ← Rd – K d = 16..31 K = 0..255
SBCI	Rd, K	Вычитание константы из регистра с учетом переноса	Rd ← Rd – K – C d = 16..31 K = 0..255
INC	Rd	Увеличить на 1	Rd ← Rd + 1 d = 0..31
DEC	Rd	Уменьшить на 1	Rd ← Rd – 1 d = 0..31
CLR	Rd	Очистить регистр (операция «исключающее ИЛИ» регистра с самим собой)	Rd ← Rd ⊕ Rd d = 0..31

Таблица П4.1 *(окончание)*

Команда	Операнды	Описание	Операция
AND	Rd, Rr	Логическое И	Rd ← Rd ∧ Rr d = 0..31 r = 0..31
ANDI	Rd, K	Логическое И с константой	Rd ← Rd ∧ K d = 16..31 K = 0..255
OR	Rd, Rr	Логическое ИЛИ	Rd ← Rd v Rr d = 0..31 r = 0..31
ORI	Rd, K	Логическое ИЛИ с константой	Rd ← Rd v K d = 16..31 K = 0..255

Команды операций с битами

Команды операций с битами приведены в табл. П4.2.

Таблица П4.2

Команда	Операнды	Описание	Операция
SBR	Rd,K	Установить биты РОН по маске (то же, что и команда ORI)	Rd ← Rd v K d = 16..31 K = 0..255
CBR	Rd,K	Сбросить биты РОН по маске	Rd ← Rd ∧ ($FF – K) d = 16..31 K = 0..255
SBI	A, b	Установить бит в РВВ	A(b) ← 1 A = 0..31[1] b = 0..7
CBI	A, b	Очистить бит в РВВ	A(b) ← 0 A = 0..31[1] b = 0..7
LSL	Rd	Логический сдвиг влево (с установкой переноса)	Rd(n+1) ← Rd(n), Rd(0) ← 0,C ← Rd(7) d = 0..31
LSR	Rd	Логический сдвиг вправо (с установкой переноса)	Rd(n) ← Rd(n+1), Rd(7) ← 0,C ← Rd(0) d = 0..31

Таблица П4.2 *(окончание)*

Команда	Операнды	Описание	Операция
ROL	Rd	Логический сдвиг влево через перенос	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7) d = 0..31
ROR	Rd	Логический сдвиг вправо через перенос	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0) d = 0..31
CLI	Нет	Запретить все прерывания (очистить флаг прерываний I в SREG)	I ← 0
SEI	Нет	Разрешить все прерывания (установить флаг прерываний I в SREG)	I ← 1
CLC	Нет	Очистить бит переноса C	C ← 0
SEC	Нет	Установить бит переноса C	C ← 1
BCLR	s	Очистить флаг s в регистре SREG	SREG(s) ← 0 s = 1..7
BSET	s	Установить флаг s в регистре SREG	SREG(s) ← 1 s = 1..7

[1] Команды SBI и CBI действительны только для РВВ по первым 32 адресам (0..31).

Команды сравнения

В операциях сравнения с регистрами производятся те же действия, что и в соответствующих арифметических и логических операциях, однако результат никуда не помещается (и, соответственно, операнды не портятся), лишь устанавливаются соответствующие флаги (C и Z) в регистре флагов SREG. Значением этих флагов в дальнейшем определяется работа тех команд условного перехода, которые употребляются в паре с командами сравнения (исключение составляет опущенная здесь и редко употребляемая команда CPSE, которая содержит сравнение и переход «в одном флаконе»). Описание команд сравнения приведено в табл. П4.3.

Таблица П4.3

Команда	Операнды	Описание	Операция
CP	Rd,Rr	Сравнение двух регистров	Rd ← Rd – Rr d = 0..31 r = 0..31
CPC	Rd,Rr	Сравнение двух регистров с учетом переноса	Rd ← Rd – Rr – C d = 0..31 r = 0..31
CPI	Rd,K	Сравнение регистра с константой	Rd – K d = 16..31 K = 0..255
TST	Rd	Проверка на 0 или отрицательное значение (операция «логическое И» регистра с самим собой)	Rd ← Rd ∧ Rd d = 0..31

Команды передачи управления

Команды передачи управления делятся, с одной стороны, на команды безусловного перехода и похожие на них команды вызова подпрограмм (последние от первых отличаются тем, что явно размещают в стеке содержимое счетчика команд для последующего возврата из подпрограммы), с другой стороны, — на команды условного перехода, т. е. нарушения последовательности выполнения операторов по какому-то условию. Большинство таких команд оперируют с адресом в памяти (К) оператора, на который производится переход. В тексте ассемблерных программ абсолютные или относительные числа, обозначающие адрес, в команды передачи управления не подставляются, вместо них указывают метки, которые затем компилятор интерпретирует, как абсолютный адрес. Команды, начинающиеся с буквы «B» (от Branch — «ветка») предполагают предварительный вызов одной из команд сравнения. Описание команд передачи управления приведено в табл. П4.4—П4.5.

Команды безусловного перехода и вызова подпрограмм

Таблица П4.4

Команда	Операнды	Описание	Операция
CALL	K	Абсолютный вызов подпрограммы	STACK ← PC + 2 PC ← K K = 0..65536[1]

Таблица П4.4 *(окончание)*

Команда	Операнды	Описание	Операция
RCALL	K	Относительный вызов подпрограммы	STACK ← PC + 1 PC ← PC + K + 1 K = –2048..2048
JMP	K	Абсолютный переход	PC ← k K = 0..4 M
RJMP	K	Относительный переход	PC ← PC + K + 1 K = –2048..2048
RET	нет	Возврат из подпрограммы	PC ← STACK
RETI	нет	Возврат из подпрограммы обработки прерывания	PC ← STACK

[1] Для устройств с максимально возможным объемом памяти программ до 64 К слов (128 кбайт).

Команды условного перехода

Таблица П4.5

Команда	Операнды	Описание	Операция
SBRC	Rr,b	Пропустить след. команду, если разряд РОН сброшен	If Rr(b) = 0 then PC ←PC + 2 (or 3[1]) else PC ←PC + 1 r = 0..31 b = 0..7
SBRS	Rr,b	Пропустить след. команду, если разряд РОН установлен	If Rr(b) = 1 then PC ←PC + 2 (or 3) else PC ←PC + 1 r = 0..31 b = 0..7
SBIC	A,b	Пропустить след. команду, если разряд РВВ сброшен	If A(b) = 0 then PC ←PC + 2 (or 3[1]) else PC ←PC + 1 A = 0..31[2] b = 0..7
SBIS	A,b	Пропустить след. команду, если разряд РВВ установлен	If A(b) = 1 then PC ←PC + 2 (or 3[1]) else PC ←PC + 1 A = 0..31[2] b = 0..7
BRNE	k	Перейти, если не равно	If Rd ≠ Rr (Z = 0) then PC ← PC + k + 1, else PC ← PC + 1 K = –63…63

Таблица П4.5 *(окончание)*

Команда	Операнды	Описание	Операция
BREQ	k	Перейти, если равно	If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1 K = –64…63
BRSH	k	Перейти, если больше или равно	If Rd ≥Rr (C = 0) then PC ← PC + k + 1, else PC ← PC + 1 K = –64…63
BRLO	k	Перейти, если меньше	If Rd <Rr (C = 1) then PC ← PC + k + 1, else PC ← PC + 1 K = –64…63
BRCC	k	Перейти, если нет переноса	If C = 0 then PC ← PC + k + 1, else PC ← PC + 1 K = –64…63

[1] Значение 2 — если следующая команда занимает одно слово (два байта) и 3 — если следующая команда занимает два слова (четыре байта).

[2] Команды SBIC и SBIS действительны только для РВВ по первым 32 адресам (0..31).

Команды переноса данных

В табл. П4.6 приведено описание команд переноса данных.

Таблица П4.6

Команда	Операнды	Описание	Операция
MOV	Rd,Rr	Перенос данных между РОН	Rd ←Rr d = 0..31 r = 0..31
LDI	Rd,K	Загрузка константы в РОН	Rd ← K d = 16..31 K = 0..255
LD (1)	Rd,p	Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в p	Rd ←(p) d = 0..31 (исключая p) p = X, Y, Z

Таблица П4.6 *(окончание)*

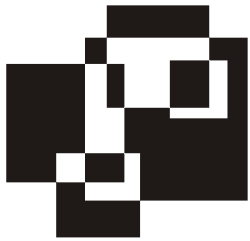
Команда	Операнды	Описание	Операция
LD (2)	Rd,p+	Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в p, с постинкрементом адреса	Rd ←(p), p = p + 1 d = 0..31 (исключая p) p = X, Y, Z
LD (3)	Rd,-p	Чтение значения в РОН из памяти данных (SRAM) по адресу, содержащемуся в p, с преддекрементом адреса	p = p – 1, Rd ←(p) d = 0..31 (исключая p) p = X, Y, Z
ST (1)	p,Rr	Запись значения в память данных (SRAM) из РОН по адресу, содержащемуся в p	(p) ←Rr r = 0..31 (исключая p) p = X, Y, Z
ST (2)	p+,Rr	Запись значения в память данных (SRAM) из РОН по адресу, содержащемуся в p, с постинкрементом адреса	(p) ←Rr, p = p + 1 r = 0..31 (исключая p) p = X, Y, Z
ST (3)	-p,Rr	Запись значения в память данных (SRAM) из РОН по адресу, содержащемуся в p, с преддекрементом адреса	p = p – 1, (p) ←Rr r = 0..31 (исключая p) p = X, Y, Z
LPM	нет	Загрузка данных из памяти программ в регистр R0 по адресу (байтовому), находящемуся в регистре Z	R0 ←(Z)
IN	Rr,A	Загрузка значения РВВ в РОН	Rr ←(A) r = 0..31 A = 0..63
OUT	A,Rr	Вывод значения РОН в РВВ	(A) ←Rr r = 0..31 A = 0..63
PUSH	Rr	Сохранить значение РОН в стеке	STACK ← Rr r = 0..31
POP	Rd	Извлечь значение верхушки стека в РОН	Rd ← STACK d = 0..31

Команды управления системой

В табл. П4.7 приведены команды управления системой.

Таблица П4.7

Команда	Операнды	Описание	Операция
NOP	нет	Нет операции	—
SLEEP	нет	Переход в «спящий» режим	—
WDR	нет	Сброс сторожевого таймера	—



Приложение 5

Тексты программ

Далее приведены полные тексты микропрограмм для конструкций, разбираемых в книге. Каждая программа подробно описана в указанных далее главах. Для воспроизведения сканируйте исходный текст с помощью Fine Reader, проверьте и сохраните в виде текстового файла с расширением asm. После этого программу можно компилировать и загружать в память контроллера, как описано в *главе 13*.

Программа для часов

Описание схемы и программы часов см. *главу 14*. Программа (листинг П5.1) занимает в памяти МК 298 слов (596 байт).

Листинг П5.1

```
;============== Программа часов ==================
;AT90S2313 4 МГц

.include "2313def.inc"

;==== Управление сегментами ====
.equ segA = 5 ; PD5 (pin 9)
.equ segB = 6 ; PD6 (pin 11)
.equ segC = 2 ; PB2 (pin 14)
.equ segD = 4 ; PB4 (pin 16)
.equ segE = 5 ; PB5 (pin 17)
.equ segF = 6 ; PB6 (pin 18)
.equ segG = 7 ; PB7 (pin 19)

;==== Управление разрядами ====
.equ em = 0 ;  PD0 (pin2)
```

```
.equ dm = 1 ;  PD1 (pin 3)
.equ eh = 2 ;  PD2 (pin 6)
.equ dh = 4 ;  PD4 (pin 8)

;==== Рабочие переменные ====
.def POS    = r16 ;отсчет разрядов для динамической индикации
.def sek    = r17 ;число секунд
.def temp   = r18 ;рабочая переменная
.def emin   = r19 ;число единиц минут
.def dmin   = r20 ;число десятков минут
.def ehh    = r21 ;число единиц часов
.def dhh    = r22 ;число десятков часов
.def set_up = r23 ;отсчет разрядов при установке
.def count  = r24 ;счетчик для мигания
.def Flag   = r25 ;флаги режимов

;в регистре Flag: если бит0=1, то идет установка часов
;если бит1=1, то внешнее напряжение пропало

;======== Прерывания =========
rjmp RESET ;процедура Reset
reti
rjmp INTT1 ;прерывание INT1 по нажатию Кн1
reti
rjmp TIM1 ;прерывание Timer1 по сравнению CompareA (счет времени)
reti
rjmp TIM0 ;прерывание Timer0 по переполнению (управл. разрядами)
reti
reti
reti
rjmp ACOMPI ;прерывание аналогового компаратора

;========== Программа =================
;===== прерывание по нажатию кнопки Кн1 ====
INTT1:
      sbrc Flag,1 ;если бит1=1 напряжения нет
      rjmp END_SET ;какая установка - на выход
      clr temp
      out GIMSK,temp ;запрещаем INT1
      sbr Flag,1 ;устанавливаем бит 0 флага - идет установка
      inc set_up ;отсчет, какой разряд устанавливаем
      clr count ;очищаем счетчик мигания
      cpi set_up,5 ;если счетчик разрядов
      brne END_SET ;еще не больше 4, то на выход
      clr set_up ;если больше, то обнуляем счетчик
```

```
      ldi sek,59 ;и устанавливаем число секунд = 59
      cbr Flag,1 ;сбрасываем бит 0 флага - конец установки
END_SET:
reti ;конец прерывания INT1

;===== прерывание Timer1 - ход часов ====
TIM1:
      sbrc Flag,1 ;если бит1=1 напряжения нет
      rjmp mtime ; сразу посылаем на счет времени
      sbrs Flag,0 ;если бит0=1 идет установка
      rjmp mtime ;иначе сразу на счет времени

      ldi temp,0b10000000 ;на всякий случай
      out GIFR,temp ;очищаем прерывание INT1
      cpi count,0 ; если счетчик =0
      breq CONT_1 ;то на продолжение
      ldi temp,1<<INT1 ;иначе разрешаем прерывание INT1
       out GIMSK,temp

CONT_1: ;установка
      inc count  ;увеличиваем счетчик мигания на 1
      cpi set_up,1 ;если первый разряд, устанавливаем его
      brne mm1 ;иначе на следующий
      sbis PinB,0 ;если Кн2 нажата
      inc emin ;увеличиваем число единиц минут на 1
      cpi emin,10 ;если число единиц минут еще не равно 10
      brne END_TIM1 ;то на выход из процедуры
      clr emin ;если =10, то обнуляем
      rjmp END_TIM1 ;на выход
mm1: cpi set_up,2 ;если второй разряд, устанавливаем его
      brne mm2 ;иначе на следующий
      sbis PinB,0 ;если Кн2 нажата
      inc dmin ;увеличиваем число десятков минут на 1
      cpi dmin,6 ;если число десятков минут еще не равно 6
      brne END_TIM1 ;то на выход из процедуры
      clr dmin ;если =6, то обнуляем
      rjmp END_TIM1 ;на выход
mm2: cpi set_up,3   ;если третий разряд, устанавливаем его
      brne mm3 ;иначе на следующий
      sbis PinB,0 ;если Кн2 нажата
      inc ehh ;увеличиваем число единиц часов на 1
      cpi ehh,4 ;если число единиц часов еще меньше 4
      brlo END_TIM1 ;то на выход из процедуры
      cpi dhh,2 ;если число десятков часов не равно 2
      brne mm21 ; то на метку mm21
```

```
      clr ehh ;иначе обнуляем число единиц часов
      rjmp END_TIM1 ;на выход из процедуры
mm21: cpi ehh,10 ;если число единиц часов не равно 10
      brne END_TIM1 ;то на выход
      clr ehh ;иначе обнуляем число единиц часов
      rjmp END_TIM1 ;на выход
mm3:  sbis PinB,0 ;четвертый разряд
      inc dhh ;увеличиваем число десятков часов на 1
      cpi dhh,3 ;если оно еще не равно 3
      brne END_TIM1 ;то на выход
      clr dhh ;иначе обнуляем
      rjmp END_TIM1 ;на выход

      ;счет времени
mtime:  inc sek ;увеличиваем число секунд на 1
      cpi sek,60 ;если оно не равно 60
      brne END_TIM1 ;то на выход

      clr sek ;иначе обнуляем секунды
      inc emin ;и увеличиваем единиц минут
      cpi emin,10 ;если число единиц минут не равно 10
      brne END_TIM1 ;то на выход

      clr emin ;иначе обнуляем единиц минут
      inc dmin ;и увеличиваем число десятков минут
      cpi dmin,6 ;если оно не равно 6
      brne END_TIM1 ;то на выход

      clr dmin ;иначе обнуляем число десятков минут
      inc ehh ;и увеличиваем число единиц часов
      cpi ehh,4 ;если число единиц часов не равно 4
      brlo END_TIM1 ;то однозначно на выход
      cpi dhh,2 ;если число десятков часов при этом не равно 2
      brne mhh ;то на метку mhh
      clr ehh ;иначе обнуляем число единиц часов
      clr dhh ;и сразу число десятков часов тоже (это полночь)
      rjmp END_TIM1 ;на выход
mhh:
      cpi ehh,10 ;если число единиц часов не равно 10
      brne END_TIM1 ;на выход
      clr ehh ;иначе обнуляем единиц часов
      inc dhh ;увеличиваем число десятков часов

END_TIM1:
reti ;выход из прерывания Timer1
```

```
;===== прерывание Timer0 управление разрядами =====
TIM0:
      sbrc Flag,1 ;если бит1=1 напряжения нет
      rjmp END_TIM0 ;тогда сразу на выход

      clr temp ;все очищаем
      out PortD,temp
      out PortB,temp

      andi POS,3 ;счет разрядов от 0 до 3

      sbis PinB,0 ; пока нажата кнопка Кн2
      clr count; разряд при установке не мигает

      cpi POS,0 ;если счетчик разрядов не 0
      brne m0 ;то на метку m0 - следующий разряд
      cpi set_up,1 ; если set_up=1, мигает этот разряд
      brne sm_1 ;иначе на включение разряда
      sbrs count,0 ;если бит 0 в count=0
sm_1:  sbi PortD,em ;то уст. PD0 в 1, разряд единиц мин. светится
      ;иначе он останется темным, т. е. мигает с частотой count
      mov temp,emin ;в temp - число единиц минут
      rcall SEG_SET ;устанавливаем сегменты
      rjmp END_0 ; на выход

m0:      cpi POS,1 ;если счетчик разрядов не 1
      brne m1 ;то на сл. разряд
      cpi set_up,2 ;если set_up=2, то мигает этот разряд
      brne sm_2 ; иначе на включение разряда
      sbrs count,0 ;если бит 0 в count=0
sm_2:  sbi PortD,dm ;то уст. PD1 в 1, разряд десятков мин. светится
      mov temp,dmin ;в temp - число десятков минут
      rcall SEG_SET ;устанавливаем сегменты
      rjmp END_0 ; на выход

m1:    cpi POS,2 ;если счетчик разрядов не 2
      brne m2 ;то на сл. разряд
      cpi set_up,3 ;если set_up=3, то мигает этот разряд
      brne sm_3 ; иначе на включение разряда
      sbrs count,0 ;если бит 0 в count=0
sm_3:  sbi PortD,eh ;то уст. PD2 в 1, разряд единиц часов светится
      mov temp,ehh ;в temp - число единиц часов
      rcall SEG_SET ;устанавливаем сегменты
      rjmp END_0 ; на выход
```

```
m2: ;обработка 4 разряда десятков часов
      cpi set_up,4  ;если set_up=4, то мигает этот разряд
      brne sm_4 ;иначе на включение разряда
      sbrs count,0 ;если бит 0 в count=0
sm_4:  sbi PortD,dh ;то уст. PD4 в 1, разряд десятков часов светится
      mov temp,dhh ;в temp - число десятков часов
      rcall SEG_SET ;устанавливаем сегменты

END_0:
      inc POS
END_TIM0:
reti ;выход из прерывания Timer0

;======= компаратор - батарейка =========
ACOMPI:
      sbis ACSR,ACO ;если бит ACO =1, то продолжим
      rjmp COMP_1 ;иначе на установку мигания
      ldi temp,0b11000000
      out TCCR1A,temp ; двоеточие включено постоянно
      rjmp END_comp ;на выход
COMP_1:
      ldi temp,0b01000000
      out TCCR1A,temp ; двоеточие мигает
END_comp:
reti ;выход из процедуры компаратора

;=============== процедура Reset ======================
RESET:
cli ;запрещаем прерывания на время установки
ldi temp,low(RAMEND) ;загрузка указателя стека
out SPL,temp

ldi temp,0b01110111 ;порт D разряды 0,1,2,4,5,6 на выход
out DDRD,temp
ldi temp, 0b11111100 ;порт B все на выход, кроме 0,1 (компаратор)
out DDRB,temp

;---- таймеры  ----
ldi temp,0b00000010 ; запуск Timer0 входная частота 1:8
out TCCR0,temp ;управление разрядами по переполнению,
;частота около 2 кГц (4/8=0,5 МГц/256)

ldi temp,high(62500) ;старший байт
out OCR1AH,temp; вых. частота 1 Гц при вх. частоте 1:64
```

```
ldi temp,low(62500) ;младший байт
out OCR1AL,temp

ldi temp,0b01000000
out TCCR1A,temp ;включаем переключающую моду для вывода OC1 (15)
ldi temp,0b01001011 ;запуск 1/64 timer1 с обнулением при сравнении
out TCCR1B,temp;

;------ прерывания --------
ldi temp,1<<INT1 ;разрешаем прерывание INT1
out GIMSK,temp
ldi temp,(1<<TOIE0)|(1<<OCIE1A);разр. прерываний Tim0 и Tim1
out TIMSK,temp
ldi temp,(1<<ACIE) ;разр. прерывания компаратора при переключении
out ACSR,temp

sei ;разрешаем прерывания

;======= основной цикл ========

G_cycle: ;все время будем отслеживать пропадание внешнего питания
      sbrs Flag,0 ;если бит0=1 идет установка
      rjmp G_cycle ;если так, то ничего не делаем
      sbic PinB,0 ;если напряжение есть
      rcall Restore ;вызываем процедуру восстановления выходов
      sbis PinB,0 ;если напряжения нет
      rcall Disable ;вызываем процедуру отключения выходов
rjmp G_cycle ;зацикливаем программу

;======= конец основного цикла ========

Disable: ;процедура отключения
      sbrc Flag,2 ;если бит1=1, то уже все сделали
      rjmp END_dis ;выход из процедуры
      cli ;запрещаем прерывания на время установок
      sbr Flag,2 ; устанавливаем бит1 Flag
      ;обрываем внешние соединения и обнуляем все порты:
      clr temp
      out PortD,temp
      out PortB,temp
      out DDRD,temp
      out DDRD,temp
      sei ;разрешаем прерывания
END_dis:
ret ;конец процедуры отключения выходов
```

```
Restore: ;процедура восстановления
      sbrs Flag,2 ;если бит1=0, то уже все остальное сделали
      rjmp end_res ;выход из процедуры
      ;восстановление
      cli ;запрещаем прерывания на время установок
      cbr Flag,2 ;сбрасываем бит1 Flag
      ;восстанавливаем внешние соединения:
      clr temp
      ldi temp,0b01110111 ;порт D разряды 0,1,2,4,5,6 на выход
      out DDRD,temp
      ldi temp, 0b11111100 ;порт B все на выход, кроме 0,1
      out DDRB,temp
      sei ;разрешаем прерывания
END_res:
ret ;конец процедуры восстановления выходов

SEG_SET: ; процедура установки сегментов
      cpi temp,0 ;ищем число, равное temp
      brne ss_0
      rcall OUT_0 ;и посылаем на нужную процедуру вывода сегментов
      rjmp END_seg
ss_0: cpi temp,1
      brne ss_1
      rcall OUT_1
      rjmp END_seg
ss_1: cpi temp,2
      brne ss_2
      rcall OUT_2
      rjmp END_seg
ss_2: cpi temp,3
      brne ss_3
      rcall OUT_3
      rjmp END_seg
ss_3: cpi temp,4
      brne ss_4
      rcall OUT_4
      rjmp END_seg
ss_4: cpi temp,5
      brne ss_5
      rcall OUT_5
      rjmp END_seg
ss_5: cpi temp,6
      brne ss_6
      rcall OUT_6
      rjmp END_seg
```

```
ss_6: cpi temp,7
      brne ss_7
      rcall OUT_7
      rjmp END_seg
ss_7: cpi temp,8
      brne ss_8
      rcall OUT_8
      rjmp END_seg
ss_8: cpi temp,9
      rcall OUT_9
END_seg:
ret ;конец процедуры установки сегментов
;процедуры для отображения цифр на семисегментном индикаторе
;ABCDEF(0);BC(1);ABDEG(2);ABCDG(3);BCFG(4);ACDFG(5)
;ACDEFG(6);ABC(7);ABCDEFG(8);ABCDFG(9)
OUT_0:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segC
      sbi PortB,segD
      sbi PortB,segE
      sbi PortB,segF
      ret
OUT_1:
      sbi PortD,segB
      sbi PortB,segC
      ret
OUT_2:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segG
      sbi PortB,segE
      sbi PortB,segD
      ret
OUT_3:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segG
      sbi PortB,segC
      sbi PortB,segD
      ret
OUT_4:
      sbi PortD,segB
      sbi PortB,segC
      sbi PortB,segF
      sbi PortB,segG
      ret
```

```
OUT_5:
      sbi PortD,segA
      sbi PortB,segF
      sbi PortB,segG
      sbi PortB,segC
      sbi PortB,segD
      ret
OUT_6:
      sbi PortD,segA
      sbi PortB,segF
      sbi PortB,segG
      sbi PortB,segC
      sbi PortB,segE
      sbi PortB,segD
      ret
OUT_7:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segC
      ret
OUT_8:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segC
      sbi PortB,segD
      sbi PortB,segE
      sbi PortB,segF
      sbi PortB,segG
      ret
OUT_9:
      sbi PortD,segA
      sbi PortD,segB
      sbi PortB,segC
      sbi PortB,segD
      sbi PortB,segF
      sbi PortB,segG
      ret
```

Программа измерителя температуры и давления

Программа измерителя температуры и давления, описанного в *главе 15*, приведена в листинге П5.2.

Листинг П5.2

```
;ATmega8535 кварц 4000 кГц
.include "m8535def.inc"
;============= Определения и константы ====================
;Управление индикацией
.equ    segG = 6 ; port C pin 28
.equ    segF = 5 ; pin 27
.equ    segE = 4 ; pin 26
.equ    segD = 3 ; pin 25
.equ    segC = 2 ; pin 24
.equ    segB = 1 ; pin 23
.equ    segA = 0 ; pin 22

.equ    RazrTdH = 0 ;  PortB,0 pin 1 десятки градуса
.equ    RazrTdM = 1 ;  PortB,1 pin 2 ед. градусов
.equ    RazrTdL = 2 ;  PortB,2 pin 3 десятые градусов
.equ    RazrPdH = 3 ;  PortB,3 pin 4 сотни давления
.equ    RazrPdM = 4 ;  PortB,4 pin 5 дес. давления
.equ    RazrPdL = 5 ;  PortB,5 pin 6 ед. давления

;адреса SRAM старший байт адреса SRAM=0x01
.equ Tram = 0x0 ; 0x0,0x1 -  старш. и младш. байты температуры АЦП
.equ Pram = 0x2 ; 0x2,0x3 – старш. и младший. байты давления АЦП
.equ TdH =  0x04 ;температура старший дес.
.equ TdM =  0x05 ;темп. средний дес.
.equ TdL =  0x06 ;темп. младший дес.
.equ PdH =  0x07 ;давление старший дес.
.equ PdM =  0x08 ;давление средний дес.
.equ PdL =  0x09 ;давление младший дес.

.equ tZH =  0x20 ;ст. разряд коэффициента Z температуры
.equ tZL =  0x21 ;мл. разряд коэффициента Z температуры
.equ tKH =  0x22 ;ст. разряд коэффициента К температуры
.equ tKL = 0x23; мл. разряд коэффициента К температуры
.equ pZH = 0x24; ст. разряд коэффициента Z давления
.equ pZL = 0x25; мл. разряд коэффициента Z давления
.equ pKH = 0x26; ст. разряд коэффициента К давления
.equ pKL = 0x27; мл. разряд коэффициента К давления

.equ AtBCD0 = 5 ;адрес ResL для процедуры преобразования Bin-Dec
.equ AtBCD2 = 6 ;адрес ResH

;переменные
.def  AregH =  r2 ;результат измерения hex, старший байт
```

```
.def  AregL =  r3 ;результат измерения hex, младший байт
.def  KoeffH = r4 ;коэффициент, старший байт
.def  KoeffL = r5 ;коэффициент, младший байт
.def  ResL =   r6 ;результат упакованный BCD, младший байт
.def  ResH =   r7 ;результат упакованный BCD, старший байт

.def  temp =     r16 ;рабочий регистр
.def  temp1 =    r17 ;вспомогательная  переменная
.def  temp2 =    r18 ;вспомогательная  переменная
.def  count =    r19 ;счетчик преобразований - до 64
.def  countCyk = r20 ;счетчик до 32,
.def  Flag  =    r21 ;регистр флагов, биты:
                    ;измерение: 0 – температуры, 1 – давления,
.def cRazr =     r22; счетчик до 6 мультиплексирования разрядов

;==================== прерывания ======================
rjmp RESET ;Reset Handle
      reti ;IRQ0 Handler
      reti ;IRQ1 Handler
      reti ;Timer2 Compare Handler
      reti ;Timer2 Overflow Handler
      reti ;Timer1 Capture Handler
      reti ;Timer1 Compare A Handler
      reti ;Timer1 Compare B Handler
      reti ;TIM1_OVF ; Timer1 Overflow Handler
rjmp TIM0 ; Timer0 Overflow Handler
      reti ;SPI Transfer Complete Handler
      reti ;USART RX Complete Handler
      reti ;UDR Empty Handler
      reti ;USART TX Complete Handler
rjmp readADC ;reti ;ADC ; ADC Conversion Complete Handler
      reti ;EEPROM Ready Handler
       reti ;Analog Comparator Handler
       reti ;Two-wire Serial Interface Handler
       reti ;IRQ2 Handler
       reti ;Timer0 Compare Handler
       reti ;Store Program Memory Ready Handler
;=========================================================
OUT_N: ;маски цифр
.db
0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,
0b01111101,0b00000111,0b01111111,0b01101111
;seg ABCDEF(0);seg BC(1);seg ABDEG(2);seg ABCDG(3);seg BCFG(4);seg
ACDFG(5)
;seg ACDEFG(6);seg ABC(7);seg ABCDEFG(8);seg ABCDFG(9)
```

```
TIM0:
;сначала индикация
      inc  cRazr   ;счетчик разрядов
      cpi  cRazr,6 ; всего 6 разрядов
      brne Set_razr
      clr cRazr ;если равен 6, очищаем
Set_razr:
      cpi  cRazr,0 ; десятки градусов
      brne P1 ;если нет - на другие разряды
           ldi r28,TdH ;установка YL - старш. темп.
      ld   temp,Y ;в temp - значение десятков градусов
      ldi ZH,HIGH(OUT_N*2) ;адрес констант в памяти - в Z
           ldi ZL,LOW(OUT_N*2)
           add ZL,temp ;адрес маски цифры, равной temp
           lpm ;в r0 - маска
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrTdH ;устанавливаем разряд
      out PORTB,temp  ;установили разряды
      rjmp rADC
P1:
      cpi      cRazr,1 ;единицы град
      brne P2
      ldi  r28,TdM ;установка адреса - ср. темп.
      ld   temp,Y
      ldi ZH,HIGH(OUT_N*2)
           ldi ZL,LOW(OUT_N*2)
           add ZL,temp
           lpm
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrTdM ;устанавливаем разряд
      out PORTB,temp  ;установили разряды
      rjmp rADC
P2:
      cpi  cRazr,2 ;дробные град
      brne P3
      ldi  r28,TdL ;установка Y - мл. темп.
      ld   temp,Y
      ldi ZH,HIGH(OUT_N*2)
           ldi ZL,LOW(OUT_N*2)
           add ZL,temp
           lpm
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrTdL ;устанавливаем разряд
      out PORTB,temp  ;установили разряды
      rjmp rADC
```

```
P3:
      cpi  cRazr,3 ;сотни давления
      brne P4
      ldi  r28,PdH ;установка Y - сотн prs
      ld   temp,Y
      ldi ZH,HIGH(OUT_N*2)
           ldi ZL,LOW(OUT_N*2)
           add ZL,temp
           lpm
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrPdH ;устанавливаем разряд
      out PORTB,temp  ;установили разряды
      rjmp rADC
P4:
      cpi  cRazr,4 ;десятки давления
      brne P5
      ldi  r28,PdM ;установка Y - дес. prs
      ld   temp,Y
      ldi ZH,HIGH(OUT_N*2)
      ldi ZL,LOW(OUT_N*2)
      add ZL,temp
      lpm
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrPdM ;устанавливаем разряд
      out PORTB,temp  ;установили разряды
      rjmp rADC
P5:
      cpi  cRazr,5 ;единицы давления
      brne rADC
      ldi  r28,PdL ;установка Y – ед. prs
      ld   temp,Y
      ldi ZH,HIGH(OUT_N*2)
      ldi ZL,LOW(OUT_N*2)
      add ZL,temp
      lpm
      out PortC,r0 ;установили сегменты
      ldi temp,1<<RazrPdL ;устанавливаем разряд
      out PORTB,temp  ;установили разряды


rADC: inc  countCyk   ;обработка АЦП
      sbrs countCyk,5 ;если бит 5 в countCyk равен 1,
             ;то прошло 32 такта таймера 0,
      ;будем запускать АЦП (см. readADC)
      reti
```

```
      clr  countCyk
      inc  count
      cpi count,65 ;если прошло 64 чтения, то сразу на обработку
      breq endADC
      sbrc Flag,0  ;если бит 0 флага - читаем температуру
      ldi  temp,0
      sbrc Flag,1 ;если бит 1 флага - читаем давление
      ldi  temp,1
      out  ADMUX, temp ;установили АЦП канал
      sbi  ADCSRA,ADSC  ;запуск новое преобразование
      reti

endADC:
;расчет по 64 значениям
      clr  count
      sbrc Flag,0
      ldi  r28,Tram ;установка адреса - T
      sbrc Flag,1
      ldi  r28,Pram ;установка адреса - P
      ld   AregH,Y+ ;загрузка суммы из памяти
      ld   AregL,Y
div64L:                    ;деление на 64
      lsr  AregH ;сдвинули старший
      ror  AregL;сдвинули младший
      inc  count
      cpi count,6
      brne  div64L ;сдвинули-поделили на 64

      subi  r28,1  ;в Y опять адрес Tram или Pram
      clr temp
      st  Y+,temp
      st  Y,temp ;очистили память для следующего цикла

      sbrs Flag,0 ;расчет температуры
      rjmp prs ;иначе давления
         ldi r28,tZH ;установка адреса коэфф. Z
         ld  temp,Y+
         mov KoeffH,temp
         ld  temp,Y
         mov KoeffL,temp  ;получили коэфф. Z температуры

;вычисление знака:
            cp   AregL,KoeffL
            cpc  AregH,KoeffH
            brsh b0
```

```
            sub  KoeffL,AregL
            sbc  KoeffH,AregH
            mov  AregL,KoeffL
            mov  AregH,KoeffH
            sbi  PortD,7 ; знак -
            rjmp m0
b0: ;если больше
            sub  AregL,KoeffL
            sbc  AregH,KoeffH
            cbi  PortD,7 ;знак +

m0: ;умножение на коэфф. K
            ldi  r28,tKH ;установка адреса коэфф. K
            ld   temp,Y+
            mov  KoeffH,temp
            ld   temp,Y
            mov  KoeffL,temp  ;получили K температуры
            rcall Mp16       ;умножили
    ;деление  на 1024
          mov AregL,temp1
          mov AregH,temp2 ;на 256
          lsr AregH ;сдвинули старший
          ror AregL;сдвинули младший
          lsr AregH ;сдвинули старший
          ror  AregL;сдвинули младший; еще на 4
          rjmp contPT
prs:
      sbrs Flag,1 ;расчет давления
      rjmp contPT
          ldi r28,pZH ;установка адреса коэфф. Z
          ld  temp,Y+
          mov KoeffH,temp
          ld  temp,Y
          mov KoeffL,temp    ;получили коэфф. Z давления

            adc AregL,KoeffL
            add AregH,KoeffH ;прибавили к величине

            ldi r28,pKH ;установка адреса коэфф. K
            ld  temp,Y+
            mov KoeffH,temp
            ld  temp,Y
            mov KoeffL,temp    ;получили коэфф. K давления
            rcall Mp16     ;умножили
    ;деление на 1024
```

```
            mov AregL,temp1
            mov AregH,temp2 ;на 256
            lsr AregH ;сдвинули старший
            ror AregL;сдвинули младший
            lsr AregH ;сдвинули старший
            ror AregL;сдвинули младший; еще на 4

contPT:

      sbrc  Flag,0
      ldi   r28,TdH ;установка адреса - T
      sbrc  Flag,1
      ldi   r28,PdH ;установка адреса - P
      rcall bin2BCD16 ;преобраз в дв. дес.
      st    Y+,ResH  ; запоминаем в памяти старший BCD
      mov   temp,ResL ;младший распаковываем
      swap temp
          andi      temp,0b00001111
      st    Y+,temp
      mov   temp,ResL
      andi  temp,0b00001111
      st    Y,temp ;и тоже сохраняем в памяти
 ;установка флагов и переменных для следующего цикла
      clr  count
      sbrc Flag,0
      rjmp _F0
      clr Flag
      sbr Flag,0x1;был бит 1, устанавливаем бит 0
reti
_F0:
      clr Flag ;был бит 0, устанавливаем бит 1
      sbr Flag,0x2
reti ; TIME0

readADC:    ;чтение АЦП по прерыванию
      sbrc Flag,0
      ldi  r28,Tram ;установка адреса - T
      sbrc Flag,1
      ldi  r28,Pram ;установка адреса - P
      ld   AregH,Y+
      ld   AregL,Y
      in temp1,ADCL;получаем младший
          in temp,ADCH ;старший
          add AregL,temp1 ;суммируем
          adc AregH,temp
      dec r28
```

```
      st  Y+,AregH ;запоминаем сумму
      st  Y,AregL ;в памяти
 reti ;ADC

Mp16: ;умножение двух 16-разрядных величин
;AregL multiplicand low byte
;AregH multiplicand high byte
;KoeffL multiplier low byte
;KoeffH multiplier high byte
;temp result byte 0 (LSB)
;temp1 result byte 1
;temp2 result byte 2
      clr temp2 ;очистить старший
      mul AregL,KoeffL ;умножаем младшие
      mov temp,r0 ;в r0 младший результата операции mul
      mov temp1,r1 ;в r01 старший результата операции mul
      mul AregH,KoeffL ;умножаем старший на младший
      add temp1,r0 ;в r0 младший результата операции mul
      adc temp2,r1 ;в r01 старший результата операции mul
      mul AregL,KoeffH ;умножаем младший на старший
      add temp1,r0 ;в r0 младший результата операции mul
      adc temp2,r1 ;в r01 старший результата операции mul
      mul AregH,KoeffH ;умножаем старший на старший
      add temp2,r0 ;4-й разряд нам тут не требуется, но он в r01
ret

bin2BCD16: ;преобразование 16-разрядного hex в упакованный BCD

;input: hex value low=AregL hex value high = AregH
;output: BCD value digits 1 and 0 ResL
;BCD value digits 2 and 3(=0) ResH
      ldi temp1,16 ;Init loop counter
      clr ResH
      clr ResL
      clr ZH       ;clear ZH (not needed for AT90Sxx0x)
bBCDx_1:lsl AregL  ;shift input value
      rol   AregH  ;through all bytes
      rol   ResL            ;
      rol   ResH
      dec   temp1  ;decrement loop counter
      brne  bBCDx_2  ;if counter not zero
      ret            ;   return
bBCDx_2:ldi r30,AtBCD2+1 ;Z points to result MSB + 1
bBCDx_3:
      ld   temp,-Z   ;get (Z) with pre-decrement
      subi temp,-$03 ;add 0x03
```

```
      sbrc temp,3    ;if bit 3 not clear
      st   Z,temp    ;store back
      ld   temp,Z    ;get (Z)
      subi temp,-$30 ;add 0x30
      sbrc temp,7    ;if bit 7 not clear
      st   Z,temp    ;store back
      cpi  ZL,AtBCD0 ;done all three?
      brne bBCDx_3   ;loop again if not
      rjmp bBCDx_1

bin2bcd8: ;преобразование 8-разрядного hex в неупакованный BCD
;вход hex= temp, выход BCD  temp1 - старш.; temp - младш.
; эта процедура работает только для исходного меньше 100
      clr    temp1   ;clear result MSD
bBCD8_1:subi temp,10 ;input = input - 10
      brcs   bBCD8_2 ;abort if carry set
      inc    temp1   ;inc MSD
      rjmp   bBCD8_1 ;loop again
bBCD8_2:subi temp,-10;compensate extra subtraction
      ret

RESET:  ;точка запуска программы после включения
      ldi   temp,low(RAMEND) ;загрузка указателя стека
      out   SPL,temp
      ldi   temp,high(RAMEND) ;загрузка указателя стека
      out   SPH,temp
      ldi   temp,1<<ACD
      out   ACSR,temp   ;выкл. аналог. компаратор на всякий

;установка портов вход-выход
      ldi temp,0b00111111 ; разряды
      out DDRB,temp
      ldi temp,0b01111111 ;сегменты
      out DDRC,temp
      ldi temp,0b10000000 ;знак "минус"
      out DDRD,temp

;установка АЦП
      ldi temp,1<<ADEN|1<<ADIE|1<<ADPS2|1<<ADPS0
      ;start ADC 1/32 такт = 128 кГц; interrupt enable
      out ADCSRA,temp

;установка таймера
      ldi temp,0b00000010
      out TCCR0,temp
```

```
      ;Timer0 on div 1:8 управление разрядами 2000 Гц

;прерывания
      ldi temp,(1<<TOIE0) ;разр. прер. Timer 0
      out TIMSK,temp
      ldi temp,255 ;сбросить все прерывательные флаги
      out TIFR,temp
      out GIFR,temp

;начальная установка переменных
      ldi r29,1 ;YH=1,пишем в RAM, начиная с 01:00
      clr count
      clr countCyk
      clr cRazr
      clr Flag
      sbr Flag,0x01 ;сначала измеряем температуру

;обнуление рабочих ячеек
       clr temp
       ldi r28,Tram ; Tempr
       st  Y+,temp
       st  Y,temp
       ldi r28,Pram ; Prs
       st  Y+,temp
       st  Y,temp

;запись коэффициентов
       ldi  r28,tZH ;начальный адрес
; Z Tempr=471
       ldi  temp,High(471) ;ст.
       st   Y+,temp
       ldi  temp,Low(471) ;мл.
       st   Y+,temp

; K Tempr=1020
       ldi temp,High(1020) ;ст.
       st  Y+,temp
       ldi temp,Low(1020) ;мл.
       st  Y+,temp

; Z prs=12
       ldi temp,High(12) ;ст. на самом деле = 0
       st  Y+,temp
       ldi temp,Low(12) ;мл.
       st  Y+,temp
```

```
; K prs=916
       ldi  temp,High(916) ;ст.
       st   Y+,temp
       ldi  temp,Low(916) ;мл.
       st   Y,temp

sei ;разрешаем прерывания
G_cykle:           ;основной цикл
      rjmp G_cykle
```

Процедуры обмена по интерфейсу I²C

Протокол обмена I²C и пример его использования описан в *главе 16*. Процедуры составлены для передачи со скоростью около 100 кГц при тактовой частоте контроллера 4 МГц. При другой частоте контроллера или иной скорости передачи число циклов в процедуре `delay`, равное 6, следует пропорционально изменить. Например, для частоты кварца, равной 16 МГц и скорости передачи, пониженной до 30 кГц, команду `ldi cnt,6` следует заменить на команду `ldi cnt,75` (приблизительно). Ошибка в ±50% в скорости передачи обычно роли не играет. При наличии сбоев увеличивайте число циклов до тех пор, пока связь не установится.

Программа содержит процедуры для двух конкретных устройств: энергонезависимой памяти с интерфейсом I²C (типа AT24) и часов реального времени (RTC) с таким же интерфейсом (например, DS1307). Процедуры WriteFlash/ReadFlash предназначены для обмена с памятью, процедуры write_i2c/read_i2c для обмена с часами. Примеры их использования см. в *главе 16*. Для других устройств легко построить собственную процедуру по аналогии, если использовать универсальные процедуры формирования протокола (start, write, read и stop), находящиеся в данном тексте. Исключите ненужные процедуры перед компиляцией программы, чтобы не загромождать память МК.

Текст листинга П5.3 целесообразно скопировать в отдельный файл, и назвать его, например, i2c.prg. Готовый файл i2c.prg также доступен в архиве по адресу **http://revich.lib.ru/AVR/ctp.zip**, где еще содержится программа для измерителя температуры и давления, обсуждаемая в *главе 16*. При необходимости использовать эти процедуры в другом устройстве следует отредактировать начальные строки, которые задают выводы МК, задействованные в процессе обмена. В данном случае это биты порта D номер 4 (SCL) и 6 (SDA). Кроме этого, конечно, следует изменить регистры для переменных, если это требуется (и при необходимости число циклов, если частота тактового гене-

ратора другая). Файл подключается к вашей программе с помощью директивы `.include "i2c.prg"` *после* таблицы векторов прерываний (см. *главу 16*).

Листинг П5.3

```
;файл I2C.prg
;Процедуры чтения и записи по интерфейсу I2C
;+---------------------- порт D ------------------------+
.equ pSCL = 4
.equ pSDA = 6
;-----------------------------------------------------
.def DATA  = r17
.def ClkA  = r18
.def cnt   = r23
.def AddrL = r24    ;адреса EEPROM
.def AddrH = r25

;----------------------- запись EEPROM -------------------------
WriteFlash:  ;в AddrL,AddrH - адрес, данные в DATA
          ;на выходе если бит c = 1 в регистре флагов, то ошибка
      cbi    PORTD,pSDA
      cbi    PORTD,pSCL
      ldi    cnt,120      ;120 попыток прописать
loop120f:
      push   DATA
      rcall  start
      ldi    DATA,0xA0  ;addr device=0,r/w=0
      rcall  write
      brcs   rt_writef  ;C=1 ERROR
      mov    DATA,AddrH ;set HI address
      rcall  write
      brcs   rt_writef  ;C=1 ERROR
      mov    DATA,AddrL ;set LO address
      rcall  write
      brcs   rt_writef  ;C=1 ERROR
      pop    DATA      ;set data to DATA
      rcall  write
      brcs   rt_f       ;C=1 ERROR
      rcall  stop
      brcs   rt_f       ;C=1 ERROR
      ret

;----------------------- чтение EEPROM -------------------------
ReadFlash: ;в AddrL,AddrH - адрес, данные в DATA
           ;если бит c = 1 в регистре флагов, то ошибка
```

```
      cbi  PORTD,pSDA
      cbi  PORTD,pSCL
ldi   cnt,120
loop_read_f:
      rcall start
      ldi   DATA,0xA0  ;addr device=0,r/w=0
      rcall write
      brcs  rt__f  ;C=1 ERROR
      mov   DATA,AddrH ;set HI address
      rcall write
      brcs  rt__f  ;C=1 ERROR
      mov   DATA,AddrL ;set LO address
      rcall write
      brcs  rt__f  ;C=1 ERROR
      rcall start
      ldi   DATA,0xA1  ;addr device=0,r/w=1
      rcall write
      brcs  rt__f  ;C=1 ERROR
      clt   ;  no put ACK
      rcall read
      rcall stop
      brcs  rt__f  ;C=1 ERROR
      ret
rt__f:
      dec   cnt
      brne  loop_read_f
      ret


rt_writef:
      pop  DATA
rt_f:
      brcc Ok_wr_f
      dec  cnt
      brne loop120f
Ok_wr_f:
      ret

;----------------------- запись RTC -------------------------
write_i2c:  ;в ClkA - адрес, данные в DATA
         ;если бит c = 1 в регистре флагов, то ошибка
      cbi  PORTD,pSDA
      cbi  PORTD,pSCL
      ldi  cnt,120 ;120 попыток прописать
```

```
loop120:
      push DATA
      rcall      start
      ldi  DATA,0b11010000 ;addr device,r/w=0
      rcall write
      brcs  rt_write  ;C=1 ERROR
      mov   DATA,ClkA ;set HI address
      rcall write
      brcs  rt_write  ;C=1 ERROR
      pop   DATA      ;set data to DATA
      rcall write
      brcs  rt_       ;C=1 ERROR
      rcall stop
      brcs  rt_       ;C=1 ERROR
      ret

;----------------------- чтение RTC -------------------------
read_i2c: ;ClkA - адрес, данные в DATA
          ;если бит c = 1 в регистре флагов, то ошибка
      cbi PORTD,pSDA
      cbi PORTD,pSCL
ldi      cnt,120
loop_read_:
      rcall start
      ldi   DATA,0b11010000  ;addr device,r/w=0
      rcall write
      brcs  rt__  ;C=1 ERROR
      mov   DATA,ClkA ;set HI address
      rcall write
      brcs  rt__  ;C=1 ERROR
      rcall start
      ldi   DATA,0b11010001  ;addr device,r/w=1
      rcall write
      brcs  rt__  ;C=1 ERROR
      clt   ;  no put ACK
      rcall read
      rcall stop
      brcs  rt__  ;C=1 ERROR
      ret
rt__:
      dec   cnt
      brne  loop_read_
      ret
rt_write:
      pop   DATA
```

```
rt_:
      brcc  Ok_wr_
      dec   cnt
      brne  loop120
Ok_wr_:
      ret
;-----------------------------------------------------

write:   ;запись байта из DATA
      push  DATA
      push  cnt
      ldi   cnt,8 ;счетчик бит
x42:
      rol   DATA
      brcs  se1
      sbi   DDRD,pSDA
      rjmp  del_wr
se1:
      cbi   DDRD,pSDA
del_wr:
      cbi   DDRD,pSCL
      rcall delay
      sbi   DDRD,pSCL
      rcall delay
      dec   cnt
      brne  x42 ;следующий бит
      cbi   DDRD,pSDA ; освободить pSDA для ACK
      rcall delay
      cbi   DDRD,pSCL
      rcall delay
      clc
      sbic  PIND,pSDA ;читаем в бит C состояние ACK
      sec   ;ACK не пришел
      sbi   DDRD,pSCL
      rcall delay
      pop   cnt
      pop   DATA
ret

read: ;чтение в DATA, бит t=1 -> ответить ACK, t=0 не отвечать ACK
      ldi   DATA,1
loop_read:
      sbi   DDRD,pSCL   ;SCL=0
      cbi   DDRD,pSDA   ;SDA=1
      rcall delay
```

```
      cbi   DDRD,pSCL   ;SCL=1
      rcall delay
      clc
      sbic  PIND,pSDA   ;читать SDA в бит C
      sec
      rol   DATA
      brcc  loop_read
      ;отсылаем ACK ()
      sbi   DDRD,pSCL    ;SCL=0
      rcall delay
      brts  se0
      cbi   DDRD,pSDA   ;не отвечать ACK (t) , SDA=1
      rjmp  rd_
se0:
      sbi   DDRD,pSDA   ;отвечать ACK (t) , SDA=0
rd_:
      clc
      rcall delay
      cbi   DDRD,pSCL    ;SCL=1
      rcall delay
ret

start:
      cbi   DDRD,pSDA
      cbi   DDRD,pSCL
      rcall delay
      sbis  PINC,pSDA
      rjmp  start
      sbis  PINC,pSCL
      rjmp  start
      sbi   DDRD,pSDA ;0=SDA
      rcall delay
      sbi   DDRD,pSCL ;0=SCL
      rcall delay
ret

stop:
      sbi   DDRD,pSDA
      sbi   DDRD,pSCL
      rcall delay
      cbi   DDRD,pSCL ;1=SCL
      rcall delay
      cbi   DDRD,pSDA ;1=SDA
      rcall delay
      clc
```

```
      sbic  PIND,pSDA
      ret
      sbic  PIND,pSCL
      ret
      sec
ret

delay:              ;~5 мкс (кварц 4 МГц)
      push  cnt
      ldi   cnt,6
cyk_delay:  dec cnt
      brne  cyk_delay
      pop   cnt
ret
```

Приложение 6

Словарь часто встречающихся терминов

В табл. П6.1 и П6.2 приведен перевод некоторых терминов, часто встречающихся в технической документации. Термины, вошедшие в русский язык в оригинальном звучании или близком к нему (transistor, resistor, logic, timer, emitter и т. п.) и потому понятные без перевода, за некоторыми исключениями в таблицах не приводятся. Не приводятся также термины и сокращения, подробно рассмотренные в тексте соответствующих глав (SRAM, DRAM, EEPROM и т. п.).

Таблица П6.1

Русско-английский	
Блок (узел, устройство) unit **центральный процессорный блок** central processor unit, CPU	**Период (импульсов)** cycle
Внешний external	**Питание** power **источник питания** power supply
Внутренний internal	**Плата** board
Восьмеричный octal	**Поддержка** support
Вход input	**Показатель** rate
Вывод (компонента) pin, lead	**Полоса (частот)** band **ширина полосы** bandwidth
Выпрямитель rectifier	**Полупроводник** semiconductor
Выход output	**Поправка** correction
Вычитание subtraction	**Последовательный** serial

Таблица П6.1 *(продолжение)*

Русско-английский	
Генератор тактирующих импульсов clock	**Предел** limit
Данные data	**Преобразователь** converter **аналого-цифровой~** analog-to-digital converter, ADC
Двоичный binary	**Проверка, контроль** check
Действующий (значение напряжения) effective	**Провод** wire **гибкий~ (шнур)** cord
Деление division	**Проводник** conductor
Делитель divisor	**Произвольный** random
Десятичный decimal	**Прямой** direct
Диапазон range, scale	**Регулировать** adjuist, control
Доступ access	**Регулировка** adjuistment
Дрейф drift	**Режим (работы)** mode
Емкость capacity, capacitance	**Синхронизация** clock
Задержка delay	**Сложение** adding
Заряд charge	**Смещение** offset
Затвор gate	**Соединение** connect
Земля ground	**Соединитель (разъем)** connector
Измерение measuring	**Состояние** state
Индуктивность (катушка индуктивности) coil	**Стирание** erase
Исток, источник source	**Сток** drain
Канал channel **~передачи данных** data transfer channel	**Сторожевой (таймер)** watchdog
Кнопка button, key	**Схема** circuit
Конденсатор capacitor	**Счетчик** counter

Таблица П6.1 *(окончание)*

Русско-английский	
Корпус case, package	**Ток** current **~базы** base current **втекающий~** sink current **вытекающий~** source current **~насыщения** saturation current **переменный~** alternating current, AC **постоянный~** direct current, DC **~смещения** bias current **сила тока** amperage
Коэффициент усиления gain **~по напряжению** voltage gain	**Точность (погрешность)** accuracy
Мост bridge **~выпрямительный** rectifier bridge	**Умножение** multiplication
Мощность power	**Умножитель** multiplier
Набор kit	**Управление** control **центральное устройство управления** mean control unit, MCU
Напряжение voltage **высокий уровень~** high voltage **низкий уровень~** low voltage **~питания** supply voltage **~смещения** bias	**Усилитель** amplifier
Ноль zero	**Установка** set **начальная~ (переустановка)** reset
Объединение (каналов) multiplex	**Устройство** device
Отношение ratio	**Утечка** leakage
Пайка soldering	**Хранение** storage
Память memory	**Частота** frequency
Панель (для микросхем) socket	**Шестнадцатеричный** hexadecimal
Параллельный parallel	**Шина** bus
Переключатель switch	**Элемент (гальванический)** cell, battery

Таблица П6.2

Англо-русский	
AC (alternating current) переменный ток	**Gain** коэффициент усиления
Access доступ	**Gate** затвор (полевого транзистора); логический элемент, вентиль (AND gate)
Accuracy точность (погрешность)	**Ground** земля
ADC (analog-to-digital converter) аналого-цифровой преобразователь	**Hexadecimal** шестнадцатеричный
Adding сложение	**Input** вход
Adjuist регулировать	**Internal** внутренний
Adjuistment регулировка	**Key** кнопка
Amperage сила тока	**Kit** набор
Amplifier усилитель	**Lead** вывод (компонента)
Band полоса (частот)	**Leakage** утечка
Bandwidth ширина полосы	**Limit** предел
Battery элемент (гальванический)	**Loop** контур обратной связи; цикл (в программе)
Bias смещение; напряжение смещения	**MCU (mean control unit)** центральное устройство управления
Binary двоичный	**Measuring** измерение
Board плата	**Memory** память
Bridge мост **rectifier~** выпрямительный мост	**Mode** режим (работы)
Bus шина	**Mount** монтировать
Button кнопка, клавиша	**Multiplex** объединение (каналов)
Capacitor конденсатор	**Multiplication** умножение
Capacity, capacitance емкость	**Multiplier** умножитель
Case корпус	**Octal** восьмеричный
Cell ячейка, элемент (гальванический)	**Offset** смещение
Channel канал **data transfer~** канал передачи данных	**Output** выход

Таблица П6.2 *(продолжение)*

Англо-русский	
Charge заряд	**Package** корпус
Check проверка, контроль	**Parallel** параллельный
Circuit схема	**Pin** вывод (компонента)
Clock синхронизация; генератор тактирующих импульсов	**Power supply** источник питания
Coil индуктивность (катушка индуктивности)	**Power** мощность
Conductor проводник	**Power** питание
Connect соединение	**Random** произвольный, случайный
Connector соединитель (разъем)	**Range** диапазон
Control управлять, регулировать, управление	**Rate** показатель
Converter преобразователь	**Ratio** отношение
Cord гибкий провод (шнур)	**Rectifier** выпрямитель
Correction поправка	**Reset** переустановка; начальная установка
Counter счетчик	**Scale** диапазон
CPU (central processor unit) центральный процессорный блок	**Semiconductor** полупроводник
Current ток **base~** ток базы **bias~** ток смещения **saturation~** ток насыщения **sink~** втекающий ток **source~** вытекающий ток	**Serial** последовательный
Cycle период (импульсов)	**Set** установка
Data данные	**Socket** панель (для микросхем)
DC (direct current) постоянный ток	**Soldering** пайка
Decimal десятичный	**Source** исток**,** источник
Delay задержка	**State** состояние
Device устройство	**Storage** хранение

Таблица П6.2 *(окончание)*

Англо-русский	
Direct прямой	**Subtraction** вычитание
Division деление	**Support** поддержка
Divisor делитель	**Switch** переключатель
Drain сток	**Unit** блок (узел, устройство)
Drift дрейф	**Value** значение
Effective действующий (значение напряжения)	**Voltage** напряжение **high~** высокий уровень напряжения **low**~ низкий уровень напряжения **supply~** напряжение питания **~gain** коэффициент усиления по напряжению
Erase стирание	**Watchdog** сторожевой (таймер)
External внешний	**Wire** провод
Frequency частота	**Zero** ноль

Литература

1. Евстифеев А. В. Микроконтроллеры AVR семейства Classic фирмы ATMEL. — М.: Додэка-XXI, 2002—2006.
2. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL. — М.: Додэка-XXI, 2004—2006.
3. Баранов В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. Изд. 2-е, испр. — М.: Додэка-XXI, 2006.
4. Шпак Ю. А. Программирование на языке C для AVR и PIC микроконтроллеров. — М.: МК-Пресс, 2006.
5. Гутников В. С. Интегральная электроника в измерительных устройствах. Изд. 2-е. — Л.: Энергоатомиздат, 1988 (1-е издание доступно в Интернете на сайте: **http://www.fayly.ru**).
6. Титце У., Шенк К. Полупроводниковая схемотехника: Справочное руководство. — М.: Мир, 1982 (доступна в Интернете: **http://www.cs.ua/rad/lib/titc/titsh.zip**).
7. Хоровиц П., Хилл У. Искусство схемотехники, в 3 т. Пер. с англ. — М.: Мир, изд. 1983, 2001, 2003.
8. Петцольд Ч. Код. — М.: ИТД «Русская редакция», 2001.
9. Шило В. Л. Популярные цифровые микросхемы: Справочник. — М.: Радио и связь, 1988.
10. Кнут Дональд Э. Искусство программирования. Т. 1. Основные алгоритмы. Т. 2. Получисленные алгоритмы. — Изд-во «Вильямс», 2005.
11. Ревич Ю. Нестандартные приемы программирования на Delphi. — СПб.: БХВ-Петербург, 2005.
12. Статьи по применению микросхем FTDI (**http://www.efo.ru/cgi-bin/go?778**).

13. FT232BM Designers Guide Version 2.0 (**ftp://ftp.efo.ru/pub/ftdichip/Documents/dg232v20.pdf**).
14. Агуров П. Интерфейс USB. Практика использования и программирования. — СПб.: БХВ-Петербург, 2004.
15. Фаронов В. В. Система программирования Delphi в подлиннике. — СПб.: BHV, 2003.
16. Осипов Д. Delphi. Профессиональное программирование. — СПб.: Символ-Плюс, 2006.
17. Элементарный учебник физики, Т. 2, 3 / Под ред. Г. С. Ландсберга. — М.: Наука, 1971.

Предметный указатель

Т