СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ

Развитие микроэлектроники привело к появлению достаточно мощных, но при этом недорогих микроконтроллеров, с помощью которых радиолюбители могут разрабатывать собственные устройства. На рынке имеется множество различной литературы, посвященной различным микроконтроллерам. Значительная часть этих книг носит справочный характер, то есть в них большое внимание уделяется общим технологиям, применяемым в том или ином микроконтроллере, рассматривается система команд языка Ассемблер для данного семейства микроконтроллеров. Несомненно, данный тип книг очень полезен, но в них, как правило, не делается упора на разработку практических устройств, а ведь это следующий шаг при практическом изучении микроконтроллеров. Поэтому, на мой взгляд, имеется определенная потребность в практическом описании устройств, их схемах, исходных кодах прошивок с комментариями, а также способах их применения. Конечно, часть этой информации можно найти в Интернете. При написании своей книги я использовал материалы некоторых наиболее интересных ресурсов. Однако данная информация в большинстве своем отрывистая или неполная, например прошивка устройства представлена в виде скомпилированного hex-файла, без исходных кодов, что не позволяет изучить этот код и подправить его под свои нужды. Кроме того, часто примеры нужных устройств приводятся не под ту номенклатуру деталей, и прежде всего микроконтроллер, которые хотелось бы использовать. В таком случае также могут возникнуть сложности с внесением исправлений в схему устройства и исходный код.

В своей книге я делаю попытку исправить приведенные выше недостатки практических описаний устройств. Рассмотрев базовые основы программирования микроконтроллеров, я перейду к рассмотрению практических устройств. Все примеры будут содержать список необходимых для данного устройства деталей, схему, исходный код прошивки с комментариями, а также блок-схему, на которой будет представлен алгоритм работы программы. Данная блок-схема, во-первых, позволит лучше понимать, как работает программа, а во-вторых, при необходимости читатель сможет самостоятельно реализовать данный алгоритм на другом языке программирования. Обо всем этом мы поговорим в моей книге.

Все приведенные устройства построены на базе микроконтроллера ATTiny 13 фирмы Atmel. Выбор именно этого микроконтроллера не случаен. Конечно, в линейке Atmel имеется много интересных и более мощных контроллеров для решения различных задач. Од-

нако для разработки тех устройств, о которых пойдет речь в книге, данная микросхема является наиболее подходящей по следующим причинам: невысокая стоимость (около $1), наличие АЦП, требования к питанию от 2,7 до 5,5 В, объема энергонезависимой памяти в 1 Кб и оперативной в 64 байта вполне достаточно для решения большинства задач. Кроме того, маленький размер позволяет делать миниатюрные, портативные устройства.

Почему устройства безопасности? Будучи по роду своей деятельности специалистом по информационной безопасности, я немало внимания уделяю аспектам, связанным с физической безопасностью. Причем здесь речь идет не только о средствах, предназначенных для предотвращения хищений и другой криминальной активности, но и о пожарной безопасности, утечек газа, протечек воды и тому подобных угроз. Таким образом, я считаю, что устройства безопасности являются необходимыми, и поэтому решил написать эту книгу.

А зачем вообще что-то делать самому, ведь все можно купить? Сами радиолюбители и вообще люди, увлекающиеся различными хобби, вряд ли задаются подобным вопросом. А вот их близкие и знакомые довольно часто об этом спрашивают. Объясняют они это тем, что сейчас производители из Юго-Восточной Азии предлагают множество различных устройств по низким ценам. Казалось бы, зачем делать устройства самому, если их можно легко приобрести дешево через Интернет. Однако на практике я часто сталкиваюсь с ситуацией, когда готовые устройства либо не обладают нужным функционалом, либо имеют избыточные возможности, которые мешают их нормальному применению, или просто недостаточно качественно сделаны, что также мешает их долговременной эксплуатации. Таким образом, зачастую недорогое, специально разработанное под конкретные задачи устройство собственного изготовления может лучше решать поставленные задачи, чем покупное.

На этом, я думаю, вступление можно завершить. Думаю, мне удалось убедить читателя в полезности моей книги.

С чего начать

Новичку в области радиоэлектроники непросто разобраться в том, как работает даже самое простое устройство. Кроме того, при работе с микроконтроллерами необходимо наличие базовых знаний по программированию. Для того чтобы получить общее представление о работе электронных устройств, я рекомендовал бы прочесть книгу «Искусство схемотехники» П. Хоровица и У. Хилла [1]. В этой книге рассматриваются базовые аспекты электроники. О более специфич-

ной литературе, связанной с микроконтроллерами Atmel, мы будем говорить в соответствующих разделах. Ну а теперь поговорим о том, что нам потребуется для работы.

Необходимые инструменты

Радиолюбительство традиционно ассоциируется с паяльниками, припоем, печатными платами и прочим оборудованием. Все это нам, несомненно, потребуется при сборке готового устройства. Но для быстрого «прототипирования», то есть разработки прототипа макета будущего устройства, нам потребуются другие компоненты.

Прежде всего нам необходима беспаечная макетная плата. Применение такой платы позволяет проверить, наладить и протестировать схему еще до того, как устройство будет собрано на готовой печатной плате. Это дает возможность избежать ошибок при конструировании, а также быстро внести изменения в разрабатываемую схему и тут же проверить результат. Понятно, что макетная плата, безусловно, экономит массу времени и является очень полезной в мастерской радиолюбителя. Самый важный плюс беспаечной монтажной платы – это отсутствие процесса пайки при макетировании схемы. Данное обстоятельство значительно сокращает процесс макетирования и отладки устройств.

Беспаечная макетная плата состоит из пластмассового основания, в котором имеется набор токопроводящих контактных разъемов (рис. В.1).

Рис. В.1. Беспаечная макетная плата

Представленная на рисунке плата имеет горизонтальные ряды контактов, подключенных к общему проводнику. А вот по вертикали эти ряды контактов изолированы друг от друга. Таким образом, если нам нужен прототип устройства, мы размещаем детали, подключая их в разные ряды контактов. Для подключения микросхем имеется специальная бороздка посередине. Над ней крепятся микросхемы, при этом левый и правый ряды ножек изолированы друг от друга.

Еще одним полезным средством является набор монтажных перемычек, предназначенных для соединения удаленных друг от друга контактов. Конечно, можно использовать обычные провода, например можно «разобрать» кусок сетевого кабеля с витой парой. Однако такие проводки могут плохо держаться в макетной плате, в результате чего может возникать «дребезг контактов». Поэтому я бы рекомендовал использовать наборы готовых перемычек, например BBJ-65.

Для работы любого электронного устройства необходимо электропитание. Для большинства описываемых в книге устройств для прототипирования достаточно питания 5 В. Случаи, когда используются батарейки с напряжением 3 В, будут рассматриваться особо.

А для того чтобы подать на плату напряжение 5 В, проще всего взять старую зарядку от мобильного телефона соответствующего номинала и срезать с нее штекер, затем зачистить и залудить контакты проводов. Теперь необходимо надежно закрепить провода в плате, например с помощью изоленты. Затем, не забывая про полярность, с помощью перемычек выводим питание на собранный макет устройства.

Думаю, с прототипированием устройств все понятно, теперь перейдем к рассмотрению элементной базы.

Какие потребуются детали

Перед началом разработки устройств нам потребуется некоторый набор деталей. Для большинства из них этот набор будет идентичен, отличаясь лишь двумя-тремя компонентами.

Прежде всего нам нужен микроконтроллер. В последующих главах мы подробно рассмотрим всю номенклатуру микроконтроллеров линейки ATTiny от Atmel. Но для наших задач нам потребуется микроконтроллер ATTINY13A-PU. Обращаю внимание на индекс PU, продаются также детали с индексом SSU, однако для разработки устройств, приведенных в моей книге, необходимо использовать микроконтроллеры с индексом PU.

Неотъемлемой частью практически любого устройства являются резисторы. Прежде всего рекомендую запастись резисторами номиналом в 10 кОм, они будут использоваться в каждом устройстве. Также для индикации сигналов на светодиоды предлагается использовать резисторы 300–400 Ом. В приведенных устройствах для работы с диодами я, как правило, использую 330 Ом. Также нам потребуются сами светодиоды. Кроме того, для подачи звуковых сигналов необходимо приобрести несколько динамиков. Здесь подойдут самые простые угольные.

Для получения информации о различных событиях окружающей среды нам потребуются соответствующие сенсоры, однако о конкретных моделях мы будем говорить в главах, посвященных данным устройствам (таких как датчик дыма, температуры и т. д.).

Ну и для разработки конечного устройства нам потребуется монтажная плата. В отличие от беспаечной макетной платы, здесь нам обязательно потребуется пайка. Так как все описываемые в книге устройства достаточно простые, я предлагаю не заниматься травлением платы, а использовать готовую печатную плату. Такая плата содержит поле с изолированными друг от друга контактами, находящимися на расстоянии 2,54 мм. Для монтажа детали сначала впаиваются в контакты, а затем с помощью припоя соединяются друг с другом. Однако, если читателю больше нравится вариант с травлением платы, материалы по данной теме можно легко найти в Интернете.

Немного о программном обеспечении

Поговорив о необходимых деталях, теперь перейдем к не менее важному вопросу – какие программы нам потребуются. Прежде всего ответим на вопрос: зачем вообще нужно ПО при разработке устройств? У нас есть микроконтроллер, на который необходимо установить соответствующую прошивку. Для этого нам потребуется, во-первых, средство разработки и компиляции, во-вторых, средство эмуляции и, в-третьих, программатор, с помощью которого мы будем записывать прошивку на микроконтроллер.

В качестве основного средства разработки я предпочитаю использовать AVR Studio [2]. Это интегрированная среда разработки (IDE) для создания 8- и 32-битных AVR-проектов от компании Atmel, работающая в операционных системах семейства Windows. AVR Studio содержит компилятор ассемблера и C/C++ и симулятор, позволяющий отследить выполнение программы. Текущая версия 6 поддерживает все выпускаемые на сегодняшний день контроллеры AVR и средства разработки. AVR Studio содержит в себе менеджер

проектов, редактор исходного кода, инструменты виртуальной симуляции и внутрисхемной отладки, позволяет писать программы на ассемблере или на C/C++.

В своей книге я буду использовать AVR Studio в качестве среды разработки и компиляции. Для эмуляции работы мы будем использовать Proteus.

Пакет Proteus [3] предназначен для синтеза и моделирования непосредственно электронных схем. Работает также только под Windows.

Еще в состав пакета входит ARES – программа разработки печатных плат, однако в рамках данной книги мы ее рассматривать не будем. Вместе с программой устанавливается набор демонстрационных проектов для ознакомления.

AVR Studio и Proteus являются коммерческими решениями. Бесплатная ознакомительная версия характеризуется полной функциональностью, но не имеет возможности сохранения файлов.

И третий программный компонент – это программатор. Здесь, правда, нам потребуется не только софт, но и аппаратные средства. В качестве программного средства выступает небольшая бесплатная утилита AVRdude [4]. Здесь у многих бывалых специалистов могут возникнуть вопросы, почему я использую именно эту утилиту для прошивки микроконтроллеров, ведь существует множество других. Тут каждый волен использовать тот инструментарий, который ему больше нравится. Это же относится и к аппаратному программатору. Для разработки своих устройств я использовал ATMEL AVR DIP PROGRAMMER (рис. В.2).

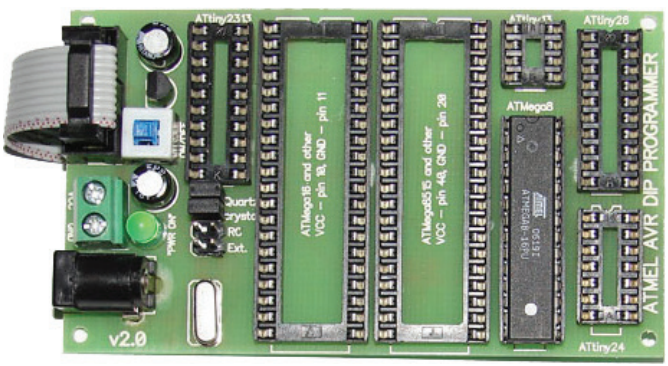

Рис. В.2. Программатор

Более подробно о том, как использовать описанные в этом разделе приложения и устройства, я расскажу далее в своей книге.

Некоторые особенности сборки готовых устройств

Итак, вы прошли весь путь от проектирования устройства до его сборки на реальной печатной плате. И теперь необходимо правильно продумать его форм-фактор. Все описанные в книге устройства являются малогабаритными, причем многие предполагают портативное использование и работу от батарейки. В связи с этим требования к корпусу становятся жестче, ведь нам необходимо, с одной стороны, небольшое (не многим более спичечной коробки), но, с другой – надежно работающее устройство.

Конечно, готовые корпуса можно приобрести в магазинах радиотоваров. Однако я столкнулся с тем, что имеющиеся в продаже корпуса или слишком большие, или же недостаточно надежны и удобны для установки платы, батарейки или вывода выключателей.

В связи с этим я хотел бы поделиться своим опытом по изготовлению корпусов для готовых устройств. Прежде всего необходимо определиться с тем, как устройство будет получать питание. Например, одно из моих устройств работало непосредственно от USB-порта. То есть его можно было включать в любой USB-порт, будь то компьютер или зарядное устройство от телефона, и оно начинало работать. Такое устройство вполне можно поместить в корпус от старой флешки или mp3-плеера. Также если детали на плате расположить соответствующим образом, то плата вполне помещается внутри гильзы от гладкоствольного охотничьего ружья, например 12-го калибра. Только гильза должна быть обязательно пластиковой, так как латунная может начать «коротить», если контакты попадут на ее корпус. Так что если у читателя есть знакомые охотники, то можно попробовать попросить несколько стрелянных гильз.

Если же устройство полностью портативное и предполагается часто его носить с собой, то здесь хорошим корпусом будет пластиковая сфера, которую используют для хранения игрушки в шоколадных яйцах, типа «Киндер-сюрприз». Небольшая овальная сфера помещается внутри яйца и используется для хранения игрушки. Так вот, в эту сферу можно поместить саму плату, а также батарейку типа CR2032, и при необходимости еще и динамик или другой элемент для вывода сигнала. Такой форм-фактор позволяет устройству быть более устойчивым при падениях и сотрясениях.

А кроме этого, корпуса можно делать из пластиковых электромонтажных коробов соответствующих размеров, просто отпиливая от короба кусочки нужно размера. Затем необходимо с одной стороны заклеить, например, кусочком пластика. После чего плату можно помещать в получившуюся коробочку.

Как видно, вариантов для изготовления корпуса устройства немало. Так что с помощью фантазии и напильника можно сделать не только надежное, но и оригинальное устройство.

Заключение

Итак, во вступлении я постарался рассказать о том, что мы собираемся делать, с помощью каких инструментов и как может выглядеть получившееся устройство. Думаю, начинающему радиолюбителю данной информации будет достаточно, для того чтобы начать искать необходимые инструменты, детали и программное обеспечение. На самом деле покупка деталей, необходимых для создания описанных в книге устройств, обойдется менее чем в тысячу рублей. Что касается инструментов, то начинающему вполне подойдет недорогой паяльник. Некоторые сложности могут вызвать программатор и плата для прошивки микроконтроллера, описанные в разделе 0.1.3, так как, например, плату мне пришлось заказывать через Интернет.

Однако в целом все это не так сложно, как может показаться. Далее мы поговорим о том, какие устройства мы будем делать.

Что мы будем делать

В этой главе мы обсудим, для чего нужны все те устройства, которые мы планируем собирать. Для специалистов по безопасности многое из того, что здесь написано, покажется очевидным, но для всех остальных эта информация может оказаться полезной.

Какие бывают устройства безопасности

Для обеспечения безопасности квартиры, дома или автомобиля нам необходимо прежде всего своевременно обнаруживать инциденты, которые могут привести к краже или разрушению охраняемого объекта. Только своевременное обнаружение позволяет предотвратить более тяжелые последствия. Угрозы могут быть природные, техногенные и антропогеннные. Природные – это различные землетрясения, наводнения и другие разрушительные природные явления. Техногенные – это прежде всего пожары, прорывы трубопроводов, утечки газа и прочее. К антропогенным угрозам мы причислим ущерб и разрушения, причиняемые действиями человека. Здесь прежде всего стоит отметить криминальную активность, такую как кражи и незаконное проникновение на охраняемый объект. Также к этому виду угроз относятся ошибки и непреднамеренные действия обслуживающего персонала.

Определимся с используемой терминологией. Большинство датчиков состоит из двух элементов: сенсора, который реагирует на соответствующую активность, и основного блока. В последний входят: микроконтроллер, принимающий решение о срабатывании в случае инцидента, и блок реагирования. Это может быть светодиод, динамик или какой-то сервопривод, выполняющий физическое действие.

Разрыв и замыкание цепи

Наверное, наиболее распространенный вид устройств – это различные датчики размыкания цепи. Это и датчики открытия двери или окна (герконы), датчики размыкания, используемые в системах зажигания автомобилей, и другие простые, но весьма полезные устройства.

В основу всех этих устройств положен принцип контроля состояния электроцепи. Пока цепь замкнута, устройство находится в состоянии покоя, выходной сигнал отсутствует. Но как только цепь обрывается и сигнал пропадает, датчик подает сигнал тревоги.

Датчики освещения

Датчик света в задачах безопасности обычно используется для включения аварийного освещения. Основной принцип данных устройств заключается в контроле освещенности в помещении. Как только уровень света становится ниже определенного значения, датчик срабатывает, включая систему аварийного освещения.

Еще одна более оригинальная сфера применения датчиков освещения, которую любят показывать в фильмах, – это контроль периметра. На фоторезистор сенсора света направляется узкий луч света. Если между резистором и источником света появляется преграда, срабатывает сигнализация. Как видно, принцип работы таких датчиков аналогичен предыдущему.

Датчики температуры и влажности

Измерение температуры важно не только для комфортного пребывания человека в помещении, но и для корректной работы различного оборудования. В случае если в вашей серверной вышла из строя система кондиционирования, температура начнет быстро увеличиваться, что может привести к перегреву оборудования.

Датчик влажности обычно размещают на полу в тех помещениях, где возможна протечка воды.

Принцип работы обоих устройств схож: при превышении определенного порога срабатывает сигнализация. Хотя некоторые системы

обнаружения протечек используют наличие тока в качестве сигнала для срабатывания.

Датчики движения и удара

Принцип работы пассивных инфракрасных охранных датчиков движения (извещателей) основан на улавливании приемником инфракрасного излучения человека и выдаче сигнала, если человек двигается.

Датчик удара срабатывает при сотрясении сенсора. В обоих случаях признаком срабатывания датчика будет являться изменение значений (дельта) уровня сигнала, приходящих от сенсора сотрясений к датчику.

Датчики дыма и газа

Датчики дыма и газа срабатывают при появлении в воздухе соответствующих примесей. Для их работы необходимы соответствующие сенсоры. Принцип работы этих сенсоров довольно прост. Это прибор, структура которого включает в себя инфракрасный светодиод, а также приемник, который отвечает за управление его работой. Первый элемент предназначен для выработки световых импульсов, второй – их приема. В случае появления дыма на приемник поступает искаженный световой импульс, что приводит к срабатыванию датчика.

Обнаружение утечек газа производится аналогичным образом. Инфракрасные сенсоры определяют концентрацию газа, полосы поглощения которого находятся в диапазоне инфракрасного электромагнитного спектра. Для различных видов газов существуют разные сенсоры. Подробнее эта тема будет рассмотрена при описании соответствующего устройства.

Таймеры

Помимо различных датчиков, использующих сенсоры, для решения задач безопасности нам могут потребоваться различные таймеры. Например, если устройство должно сработать по прошествии определенного времени. В этом случае нам потребуется сделать таймер. Причем возможны варианты, когда таймер должен работать в различных режимах в зависимости от задачи. Например, в одном режиме нам необходимо задавать интервал времени в секундах, а в другом – в минутах. Для этого нужен программируемый таймер, которому сначала передается режим, в котором он будет работать, а только потом сам временной интервал для срабатывания.

Сигнализации и другие устройства

Сигнализации в наших устройствах – это блок реагирования. Здесь нам наиболее интересны системы звуковой сигнализации, в которых в случае инцидента сигнал подается на динамик.

Также полезным устройством безопасности является кодовый замок. Данные замки бывают как классические, с клавиатурой, так и однокнопочные, где для ввода кода используется последовательность нажатий.

«Сюрпризы» в автомобиле

Для защиты автомобилей традиционно используется множество различных устройств. Основная цель их – это помешать злоумышленнику угнать машину. Данные устройства представляют собой либо ту же сигнализацию, которая в случае инцидента подает сигнал, либо «молчаливый» блокиратор, который просто не дает завести двигатель, либо комбинацию этих устройств. В контексте данной книги мы поговорим о создании интерфейса управления данными охранными системами. Однако темы блока реагирования для автомобильных охранных систем (то есть того, что нужно сделать, чтобы машина не поехала) я касаться не буду. Дело в том, что времена «гаражных умельцев» уходят в прошлое. Электрика и электроника в современном автомобиле устроены достаточно сложно, и, для того чтобы вносить в них какие-то изменения, необходимы высокая квалификация и хорошее знание устройства данной конкретной модели автомобиля. Кроме того, при внесении изменений в гарантийную машину вы рискуете лишиться гарантии. И наконец, каждая самодельная система защиты должна быть уникальной, так что не стоит в «открытом эфире» обсуждать возможные варианты. Угонщики ведь тоже могут прочитать эту книгу.

Заключение

Итак, в этой главе мы поговорили о том, какие устройства безопасности бывают, какие принципы положены в основу их работы. Конечно, полный список таких устройств не ограничивается описанными здесь. Существуют еще всевозможные RFID-метки, устройства беспроводной связи и другие решения. Но в своей книге я рассматриваю наиболее полезные и применимые с точки зрения работы с ATTiny13 устройства.

Теперь мы перейдем к подробному рассмотрению «мозга» любого нашего устройства.

1 ЧТО ТАКОЕ МИКРОКОНТРОЛЛЕР ATMEL ATTINY13

На сегодняшний день на рынке присутствует множество различных микроконтроллеров. Помимо Atmel, существуют также PIC, чьи контроллеры более подходят для тиражирования устройств, STM, ориентированны на более сложные, промышленные устройства и другие решения. Atmel из этого ряда выделяют дешевизна, по сравнению с другими микроконтроллерами, простота в освоении. К тому же, на мой взгляд, именно младшие модели контроллеров Atmel из линейки ATtiny наиболее подходят для начинающих радиолюбителей, которые еще не владеют всеми тонкостями работы с микроконтроллерами.

Но, прежде чем начать обсуждение продукции компании Atmel, я хотел бы поговорить о том, что такое микроконтроллер и зачем он нужен.

1.1. Микроконтроллер – мозг устройства

Микроконтроллер – это небольшая микросхема, на кристалле которой реализован функционал крошечного компьютера. То есть внутри одной микросхемы смонтированы процессор, оперативная и энергонезависимая память, периферийные устройства, аналогово-цифровой и цифроаналоговый преобразователи (АЦП и ЦАП). При этом данные элементы работают и взаимодействуют между собой и внешним миром с помощью специальной микропрограммы, которая хранится внутри микроконтроллера.

Основное назначение микроконтроллеров – это управление различными электронными устройствами. В контексте моей книги микроконтроллеры (МК) используются для управления устройствами безопасности. То есть именно он решает, какое значение входного параметра является поводом для выполнения тех или иных охранных действий. Например, именно МК решает, что надо включить сигнализацию, когда значение температуры, передаваемое термодатчиком, превысит некоторую пороговую величину.

Таким образом, можно без преувеличения сказать, что микроконтроллер является мозгом устройства (рис. 1.1).

Так выглядят современные микроконтроллеры.

В свою очередь, за правильность функционирования этого мозга отвечает программа, которая в него зашита. В обычных микросхемах функционал жестко зашивается при производстве и не может быть никак изменен. На микроконтроллер прошивку можно записывать многократно. Но не стоит считать МК «маленькими флешками». У МК гораздо меньше рабочих циклов записи, чем у USB-

накопителя. Не стоит более ста раз перепрошивать микроконтроллер, он может попросту не прошиться. В то же самое время данные на флешку можно записывать более 10 000 раз. Именно поэтому нам потребуется симулятор Proteus, с помощью которого можно будет протестировать работу прошивки и в определенной степени всего устройства, прежде чем начать прошивать живой микроконтроллер.

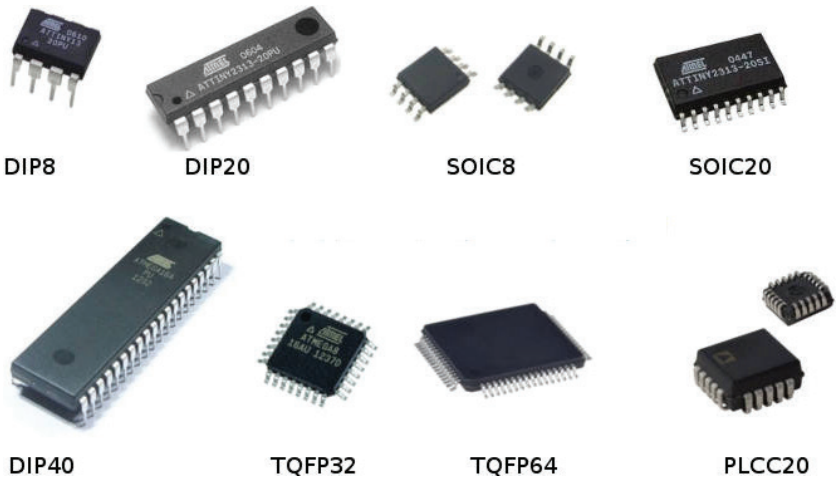

Рис. 1.1. Современные микроконтроллеры

Посмотрим, как логически устроен МК (рис. 1.2).

- Арифметико-логическое устройство (АЛУ) – предназначено для выполнения арифметических и логических операций, на самом деле в совокупности с регистрами общего назначения АЛУ выполняет функции процессора.
- Оперативно-запоминающее устройство (ОЗУ) – предназначено для временного хранения данных при работе микроконтроллера.
- Память программ – выполнена в виде перепрограммируемого постоянного запоминающего устройства и предназначена для записи микропрограммы управления микроконтроллером, так называемая прошивка.
- Память данных применяется в некоторых микроконтроллерах в качестве памяти для хранения всевозможных констант, табличных значений функций и т. д.

Микроконтроллер в своем составе может иметь и другие вспомогательные элементы.

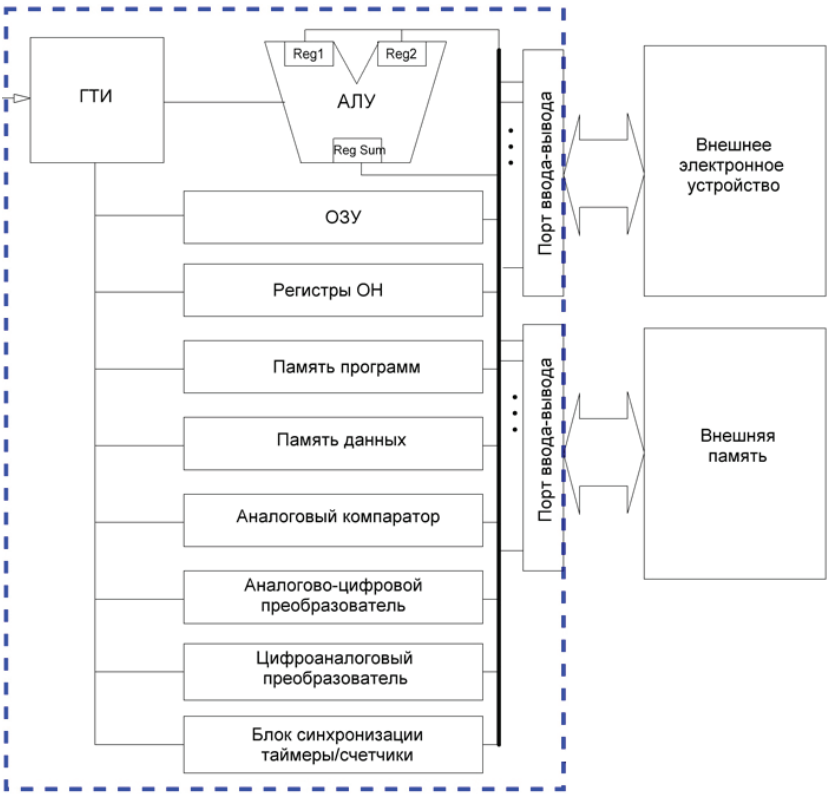

Рис. 1.2. Логическая схема устройства микроконтроллера

- Аналоговый компаратор – предназначен для сравнения двух аналоговых сигналов на его входах.
- Таймеры в микроконтроллерах применяются для осуществления различных задержек и установки различных интервалов времени в работе микроконтроллера.
- Аналогово-цифровой преобразователь (АЦП) необходим для ввода аналогового сигнала в микроконтроллер, и его функция – перевести аналоговый сигнал в цифровой.
- Цифроаналоговый преобразователь (ЦАП) выполняет обратную функцию, то есть сигнал из цифрового вида преобразуется в аналоговый вид.

Программный код, который зашивается в микроконтроллер, отвечает за взаимодействие всех этих элементов в рамках поставленной программистом задачи.

Думаю, что с тем, что такое микроконтроллер, в целом понятно, теперь перейдем к рассмотрению МК от Atmel.

1.1.1. Обзор микроконтроллеров Atmel AVR

Компания Atmel производит несколько типов микроконтроллеров. На их сайте можно найти следующее описание:

32-разрядные микроконтроллеры AVR UC3

Самые эффективные 32-разрядные микроконтроллеры.
Предназначение: системы общего назначения.
Основные параметры:
- флеш-память 16–512 Кб;
- корпуса с 48–144 выводами;
- рабочая частота до 66 МГц;
- производительность 1,5 MIPS/МГц.

Микроконтроллеры AVR XMEGA

8-разрядное решение высокой производительности.
Предназначение: системы общего назначения.
Основные параметры:
- флеш-память 16–384 Кб;
- корпуса с 32–100 выводами;
- рабочая частота до 32 МГц;
- производительность 1,0 MIPS/МГц.

Микроконтроллер megaAVR

Больше функций и периферийных устройств.
Предназначение: системы общего назначения.
Основные параметры:
- флеш-память 4–256 Кб;
- корпуса с 28–100 выводами;
- рабочая частота до 20 МГц;
- производительность 1,0 MIPS/МГц.

8-разрядные микроконтроллеры tinyAVR

Компактность и мощность.
Предназначение: системы общего назначения.
Основные параметры:
- рабочее напряжение 0,7 В;
- флеш-память 0,5–8 Кб;
- корпуса с 6–32 выводами;
- рабочая частота до 20 МГц;
- производительность 1,0 MIPS/МГц.

Управление аккумулятором

Предназначение:

- управление литий-ионными аккумуляторами;
- измерение уровня заряда аккумулятора;
- балансировка элементов аккумулятора;

Основные параметры:

- рабочее напряжение 1,8–25 В;
- флеш-память 8–40 Кб;
- корпуса с 28–48 выводами;
- рабочая частота до 8 МГц;
- производительность 1,0 MIPS/МГц.

Микроконтроллеры AVR для автомобильной электроники

Предназначение: интеллектуальное управление и надежная конструкция, соответствующие жестким требованиям к автомобильной электронике.

Как видно, Atmel выпускает множество различных микроконтроллеров, предназначенных как для общего применения, так и для решения специализированных задач. Для решения задач, описываемых в этой книге, нам вполне подойдут 8-битные МК семейства tinyAVR. Более мощные и, следовательно, более дорогие микроконтроллеры будут избыточны для этого.

1.1.2. ATtiny13 – маленькая, но шустрая

ATtiny – семейство AVR-микроконтроллеров, оптимизированных для приложений, требующих относительно большой производительности (до 1,0 MIPS и способных работать на частотах до 20,0 МГц), энергоэффективности и компактности.

Номенклатура микроконтроллеров ATtiny на сегодняшний день состоит из следующих устройств:

- ATtiny4, ATtiny5, ATtiny9, ATtiny10 – максимум 4 контакта ввода/вывода, рабочее напряжение 1,8–5,5 В, 32 байта SRAM, производительность до 12 MIPS (на частоте 12 МГц), Flash-память для хранения программ (1 Кбайт в ATtiny9/10 и 512 байт в ATtiny4/5), аналого-цифровой преобразователь (в ATtiny9/10).
- ATtiny13 – максимум 6 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 64 байта SRAM, 64 байта EEPROM, производительность до 20 MIPS (на частоте 20 МГц), 1 Кбайт Flash-памяти для хранения программ, аналого-цифровой преобразователь (ADC).

- ATtiny24, ATtiny 44, ATtiny 84 – максимум 12 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), температурный датчик (на кристалле), универсальный последовательный интерфейс (USI).
- ATtiny25, ATtiny 45, ATtiny 85 – максимум 6 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), универсальный последовательный интерфейс (USI).
- ATtiny261, ATtiny 461, ATtiny 8 61 – максимум 16 контактов ввода/вывода, рабочее напряжение 1,8–5,5 В, 128/256/512 байт SRAM и 128/256/512 байт EEPROM (соответственно), производительность до 20 MIPS (на частоте 20 МГц), 2/4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), универсальный последовательный интерфейс (USI).
- ATtiny48, ATtiny 88 – максимум 24/28 контактов ввода/вывода (в зависимости от корпуса), рабочее напряжение 1,8–5,5 В, 256/512 байт SRAM (соответственно), 64 байта EEPROM, производительность до 12 MIPS (на частоте 12 МГц), 4/8 Кбайт Flash-памяти для хранения программ (соответственно), аналого-цифровой преобразователь (ADC), последовательный внешний интерфейс (SPI).
- ATtiny43U – максимум 16 контактов ввода/вывода, рабочее напряжение 0,7–1,8 В, 256 байт SRAM, 64 байта EEPROM, производительность до 1 MIPS на мегагерц, 4 Кбайт Flash-памяти для хранения программ, аналого-цифровой преобразователь (ADC), температурный датчик (на кристалле), универсальный последовательный интерфейс (USI). Микросхема с низким энергопотреблением, встроенный преобразователь автоматически генерирует стабильное напряжение питания 3 В от низковольтного источника питания (не ниже 0,7 В).

Как видно, ATTiny13 является не самым мощным микроконтроллером в данном ряду. Однако небольшой размер, невысокие требования к энергопотреблению и дешевизна сделали его главным героем этой книги. Кроме того, ATTiny13 оптимально подходит для разработки тех устройств безопасности, о которых шла речь в преды-

дущей главе. Функционал ATtiny43U, к примеру, будет избыточен для большинства из них.

Далее мы немного поговорим о некоторых особенностях работы с данным микроконтроллером и затем перейдем к обсуждению большой темы, связанной с программированием данных МК (рис. 1.3).

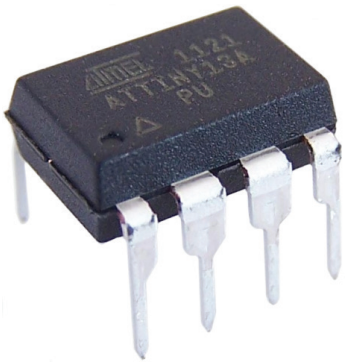

Рис. 1.3. Внешний вид МК ATtiny13

1.1.3. Особенности применения

МК ATtiny13 имеет корпус DIP или SMD. Учитывая небольшой размер самого микроконтроллера, а также наличие у него всего 8 внешних контактов, я бы рекомендовал использовать корпус DIP.

Кроме того, с микроконтроллерами в корпусе DIP очень легко работать благодаря большому шагу между выводами, что очень удобно как при пайке, так и при подключении микроконтроллера к плате через специальный разъем (панелька/кроватка) – для снижения риска повреждения компонента при пайке.

Также использование панелек позволяет при необходимости перепрошивать МК, например в случае модернизации кода программы или при использовании МК в другом устройстве.

В качестве источников питания можно использовать как старые зарядки от телефонов с выходным напряжением 5 В, так и круглые, плоские батарейки типа CR2032, выдающие напряжение 3 В.

В остальном этот микроконтроллер не требует каких-либо дополнительных настроек и компонентов.

1.1.4. Заключение

В этом разделе мы кратко обсудили, что такое микроконтроллер, зачем он нужен. Также поговорили о типах существующих на рынке

решений. Еще мы обсудили все особенности и преимущества ATtiny13. Теперь самое время перейти к описанию средств программного обеспечения для микроконтроллеров.

1.2. Ассемблер Atmel AVR

Для написания прошивок для МК ATtiny существует несколько языков программирования. Автору приходилось сталкиваться с компиляторами для языков высокого уровня C и Basic. Однако в своей книге для написания прошивок я буду использовать ассемблер.

1.2.1. Почему именно ассемблер?

У начинающих радиолюбителей, особенно не слишком хорошо знакомых с программированием, данный язык программирования вызывает страх и трепет, и они предпочитают использовать в своих устройствах код, написанный на языках высокого уровня. Конечно, такая точка зрения вполне имеет право на жизнь. В простых устройствах, где скомпилированная прошивка занимает не более 200–300 байт, использование высокоуровневых языков будет вполне уместно. Однако не стоит забывать, что при компиляции с языка высокого уровня неизбежно теряется часть памяти. Дело в том, что компилятор не всегда использует оптимальные приемы трансляции с языков высокого уровня в машинные коды. В результате один и тот же алгоритм, реализованный на ассемблере и на C, может в откомпилированном виде в первом случае занимать на 5–10 процентов меньше места, чем во втором. Кроме того, программы, написанные на ассемблере, зачастую работают быстрее, чем их аналоги на языках высокого уровня. Это также связано с тем, что компиляторы не оптимально транслируют высокоуровневую программу в машинные коды. Хотя при разработке описываемых в книге устройств вам вряд ли придется столкнуться с проблемами производительности МК.

С другой стороны, сложность ассемблера МК Atmel несколько надуманна. По крайней мере, по сравнению с аналогичными языками программирования в более серьезных процессорах, таких как Intel. Здесь нет такого разнообразия команд и типов регистров, поэтому освоить ассемблер Atmel будет значительно проще, чем, к примеру, программировать 64-битные приложения под Unix.

Так что не все так страшно. Ну а я со своей стороны постараюсь вам помочь, подробно и доходчиво описав основы программирования на данном языке.

1.2.2. Регистры

Для начала работы с ассемблером необходимо усвоить понятие регистра. Регистром является зарезервированная ячейка памяти. В отличие от памяти программ, адресное пространство памяти данных адресуется побайтно 8 разрядам (а не пословно 16 разрядам). Адресация МК Atmel полностью линейная, без какого-то деления на страницы, сегменты или банки, как это принято в некоторых других системах.

МК семейства Tiny (включая Tiny13) памяти данных как таковой не имеют, ограничиваясь лишь регистрами общего назначения (РОН) и регистрами ввода-вывода (РВВ).

Всего в МК 32 регистра общего назначения. В микроконтроллерах AVR все 32 РОН непосредственно доступны АЛУ. Благодаря этому любой РОН с R0 по R31 может использоваться во всех командах и как операнд-источник, и как операнд-приемник. Исключение составляют лишь пять арифметических и логических команд. При этом регистры R26-R31 используются для косвенной адресации. Пары этих 8-разрядных РОН образуют три 16-разрядных регистра X, Y, Z.

У МК есть три регистра ввода-вывода: DDRx, PORTx и PINx. Эти регистры, как следует из названия, предназначены для взаимодействия с портами ввода-вывода. Порты, в свою очередь, могут работать как входы и как выходы. Если порт работает как вход, то, для того чтобы считать значения, необходимо обратиться к регистру PINB или PIND – смотря с какого порта производим считывание. Если порт является выходом, то значения на линиях порта устанавливаются путем записи соответствующего значения в регистр порта PORTB или PORTD.

Самый важный момент работы с портом – это работа с регистром-защелкой, отвечающей за работу линий порта на вход или на выход. Название этого регистра DDRx, где x – буква порта. Для того чтобы сделать ножки выходами, мы должны записать в соответствующие биты «1». Например, мы хотим сделать ножку PB7 порта B входом, а остальные ножки – выходами, тогда для этого необходимо записать в регистр DDRB значение 0b01111111. Приставка 0b означает, что число записано в двоичном виде. При запуске регистры DDRx обнулены, т. е. все ножки являются входами.

Еще одним важным регистром, о котором следовало бы упомянуть, является регистр SREG. Он содержит специальные флаги (по сути, биты), которые могут принимать значения 0 или 1 в зависимости от того, в каком состоянии должен находиться флаг.

Ниже приводится описание всех флагов, входящих в SREG.

Бит 7. Флаг I. Общее разрешение прерываний

Для разрешения прерываний этот флаг должен быть установлен в 1. Если флаг сброшен, то прерывания запрещены независимо от состояния разрядов регистров маскирования отдельных прерываний. Флаг сбрасывается аппаратно после входа в прерывание и восстанавливается командой RETI для разрешения обработки следующих прерываний.

Бит 6. Флаг T. Хранение копируемого бита

Используется в качестве источника или приемника команд копирования битов BLD (Bit Load) и BST (Bit Store).

Бит 5. Флаг H. Флаг половинного переноса

Устанавливается в 1, если произошел перенос из младшей половины байта (т. е. из третьего разряда в четвертый) или заем из старшей половины байта при выполнении некоторых арифметических операций.

Бит 4. Флаг S. Флаг знак

Равен результату операции «Исключающее ИЛИ» (XOR) между флагами N и V. Соответственно, этот флаг устанавливается в 1, если результат выполнения арифметической операции меньше нуля.

Бит 3. Флаг V. Флаг переполнения дополнительного кода

Устанавливается в 1 при переполнении разрядной сетки знакового результата. Используется при работе со знаковыми числами (представленными в дополнительном коде).

Бит 2. Флаг N. Флаг отрицательного значения

Устанавливается в 1, если старший (седьмой) разряд результата операции равен единице. В противном случае флаг равен 0.

Бит 1. Флаг Z. Флаг нуля

Устанавливается в 1, если результат выполнения операции равен нулю.

Бит 0. Флаг C. Флаг переноса

Устанавливается в 1, если в результате выполнения операции произошел выход за границы байта.

Некоторые из этих флагов мы будем активно использовать в дальнейшем.

На этом пока закончим с регистрами. Читателю, не слишком хорошо знакомому с программированием, возможно, не все покажется сейчас понятным, однако в процессе написания программ я буду делать соответствующие пояснения, что позволит понять, как работать с регистрами.

1.2.3. Стек и переменные

Хранение данных в памяти может осуществляться различными способами: с помощью переменных, а также в стеке. В ATTiny каждой переменной, используемой в программе, должен соответствовать определенный регистр общего назначения.

Если с переменными в ассемблере все более-менее понятно, то о том, зачем нужен стек, мы поговорим подробнее.

Прежде всего разберемся с тем, что из себя представляет стек. Представим себе детскую пирамидку. Мы кладем сначала большой диск, затем поменьше, затем еще меньше и в конце концов самый маленький. Если же нам надо разобрать эту самую пирамидку, мы снимаем сначала самый маленький диск, затем второй по величине и последним снимаем самый большой. Так вот, пирамидка представляет собой классический стек – это структура, в которой элемент, который добавляется первым, удаляется последним, и наоборот: элемент, добавленный последним, удаляется первым. В литературе еще используется англоязычное сокращение LIFO (Last In – First Out: последним вошел – первым вышел).

Но причем здесь микроконтроллеры? Именно благодаря стеку есть возможность применять подпрограммы. Рассмотрим этот процесс подробнее. Вызов подпрограммы подразумевает, что после ее выполнения и выхода из подпрограммы выполнение основной программы продолжится со следующей за вызовом строки. А поскольку одна и та же подпрограмма может быть вызвана из разных мест основной программы или других подпрограмм, то адрес возврата каждый раз будет разным. Тут нам на помощь и приходит стек.

В момент вызова подпрограммы следующая после перехода команда сохраняется в верхушке стека. После завершения подпрограммы выполняется специальная команда возврата, и переход осуществляется по адресу, взятому из вершины стека. И таким образом продолжается выполнение программы с того места, откуда было совершен вызов подпрограммы. Так как в процессе работы программы таких переходов может быть большое количество, для хранения всех адресов возврата нам необходим стек.

Теперь поговорим о самих командах ассемблера.

1.2.4. Проверка и пропуск

Для того чтобы описываемые далее ассемблерные команды были более понятны начинающим радиолюбителям, я предлагаю параллельно описывать аналогичные команды языков высокого уровня, таких как C и Basic. Начнем с команд сравнения.

В языках высокого уровня за сравнение отвечает команда IF. В случае если выражение, следующее после IF, является истиной, далее выполняется подпрограмма, соответствующая выполнению данного выражения в команде IF.

В общем случае, без учета синтаксиса конкретных высокоуровневых языков, команда IF может иметь следующий вид:

```
IF (условие) {
…подпрограмма, соответствующая выполнению условия
}
```

В ассемблере существует несколько команд сравнения. Для тех устройств, которые мы будем разрабатывать далее, нам потребуются три из них.

```
CP Rd, Rr   Сравнение двух РОН
```

Данная команда сравнивает значения двух регистров. В случае их равенства флаг Z = 1.

```
CPI Rd, K   Сравнение РОН с константой
```

Данная команда сравнивает значения регистра с константой. В случае их равенства флаг Z = 1.

```
CPSE Rd, Rr  Сравнение двух РОН и пропуск следующей команды,
             если содержимое обоих РОН равно
```

В случае равенства значений двух регистров при выполнении будет пропущена следующая команда.

В связке с командами сравнения обычно используют команды перехода. Рассмотрим некоторые из них.

```
BREQ A  Переход по условию «равно»                        Если флаг Z = 1
BRNE A  Переход по условию «не равно»                     Если флаг Z = 0
BRSH A  Переход по условию «больше или равно»             Если флаг C = 0
BRLO A  Переход по условию «меньше»                       Если флаг C = 1
BRMI A  Переход по условию «отрицательное значение»       Если флаг N = 1
BRPL A  Переход по условию «положительное значение»       Если флаг N = 0
BRGE A  Переход по условию «больше или равно» (со знаком) Если флаги (N и V) = 0
```

Для того чтобы читателю стало понятнее, как работать с этими командами, я приведу несколько примеров.

Сравнение с константой.

```
…
cpi r16,168          ;сравниваем значение регистра r16 с 168
brlo sb1             ;если меньше, то переходим к следующей подпрограмме sb1
…
```

Сравнение двух регистров.

```
…
ldi r17, 0x00        ;загружаем значение 0x00 в регистр r17
ldi r18, 0x01        ; загружаем значение 0x01 в регистр r18
cp r17,r18           ;сравниваем значение регистра r17 с r18
brne sb2             ;если не равно, то переходим к следующей подпрограмме sb2
…
```

Приведенная во втором примере команда LDI будет подробно рассмотрена чуть позже.

1.2.5. Выполнение логических операций

Пожалуй, ни один язык программирования не обходится без логических операций, и практически во всех этих языках команды логических операций имеют схожий синтаксис. Поэтому важно понимать, как в принципе выполняются логические операции.

```
Логическое И: 0 И 0 =0, 0 И 1 = 0, 1 И 0 = 0, 1 И 1 = 1.
Логическое ИЛИ: 0 ИЛИ 0 =0, 0 ИЛИ 1 = 1, 1 ИЛИ 0 =1, 1 ИЛИ 1 =1.
Логическое НЕ: НЕ 0 = 1, НЕ 1 = 0.
```

Теперь посмотрим, какие команды ассемблера реализуют данные логические операции.

```
AND Rd, Rr    «Логическое И» двух РОН
ANDI Rd, K    «Логическое И» РОН и константы
EOR Rd, Rr    «Исключающее ИЛИ» двух РОН
OR Rd, Rr     «Логическое ИЛИ» двух РОН
ORI Rd, K     «Логическое ИЛИ» РОН и константы
TST Rd        Проверка РОН на отрицательное (нулевое) значение
```

На практике это выглядит следующим образом.

Для логического И:

```
…
ldi r17, 0xfe        ; загружаем значение 0x00 в регистр r17
ldi r18, 0x00        ; загружаем значение 0x01 в регистр r18
```

```
and r17,r18          ; логическое И регистров r17 с r18
…
```

Для логического ИЛИ:

```
…
ldi r17, 0x00        ; загружаем значение 0x00 в регистр r17
ori r17,0x01         ; логическое ИЛИ регистра r17 и 0x01
…
```

При первом прочтении логика данных операций новичкам может оказаться не совсем понятной, но в процессе работы понимание придет само собой.

1.2.6. Побитовые операции

Побитовые операции предназначены для последовательного сдвига всех битов влево или вправо. При этом в случае простого логического сдвига крайние биты слева или справа соответственно будут удаляться, а освободившиеся после сдвига будут заполняться нулями. При использовании сдвигов через перенос крайние биты будут соответственно переноситься в другой конец на освободившееся место. Так, при сдвиге седьмой бит будет перенесен на место нулевого, а все остальные просто сдвинутся влево.

```
 LSL Rd      ;Логический сдвиг влево
C< Rd7<Rd6<Rd5<Rd4<Rd3<Rd2<Rd1<Rd0<0
```

Старший разряд переходит в C, все остальные разряды сдвигаются влево. Младший разряд заполняется нулем.

```
 LSR Rd      ;Логический сдвиг вправо
0> Rd7>Rd6>Rd5>Rd4>Rd3>Rd2>Rd1>Rd0>C
```

Здесь наоборот – старший разряд заполняется нулем, все остальные разряды сдвигаются вправо. Младший разряд выталкивается в C.

```
 ROL Rd      ;Сдвиг влево через перенос
Rd7<Rd6<Rd5<Rd4<Rd3<Rd2<Rd1<Rd0
```

Старший бит из седьмого разряда переносится в нулевой.

```
 ROR Rd      ;Сдвиг вправо через перенос
Rd7>Rd6>Rd5>Rd4>Rd3>Rd2>Rd1>Rd0
```

Младший бит из нулевого разряда переносится в седьмой. Примеры здесь довольно простые.

```
…
ldi r18, 0x0a    ; загружаем значение 0x0a в регистр r18
rol r18          ; логический сдвиг влево
…
```

Для понимания значение 0a проще перевести в двоичный вид 00001010. После выполнения сдвига влево с переносом получаем 00010100 в двоичном виде, или в шестнадцатеричном – 14.

1.2.7. Арифметические операции

Из арифметических операций ATTiny умеет складывать и вычитать. Для выполнения наших задач этого будет вполне достаточно.

```
ADD Rd, Rr  Сложение двух РОН
ADC Rd, Rr  Сложение двух РОН с переносом
ADIW Rd, K  Сложение регистровой пары с константой
SUB Rd, Rr  Вычитание двух РОН
SUBI Rd, K  Вычитание константы из РОН
SBIW Rd, K  Вычитание константы из регистровой пары
DEC Rd      Декремент РОН
INC Rd      Инкремент РОН
```

Результат всегда загружается в Rd. В одном примере работа нескольких команд будет выглядеть так:

```
…
ldi r17, 0x06       ; загружаем значение 0x06 в регистр r17
ldi r18, 0x0f       ; загружаем значение 0x0f в регистр r18
add r17, r18        ; сложение регистров, результат в r17
subi r17, 0x05      ; вычитаем константу из регистра
dec r18             ; понижаем значение регистра на 1
…
```

Здесь достаточно простая арифметика. Сложение 06 и 0f даст 15, затем вычитание 5 даст шестнадцатеричное 10 в регистре r17. Потом значение r18 будет понижено на 1 – получаем 0e.

1.2.8. Пересылаем данные

Без операций пересылки данных не может обойтись ни одна программа. В языках высокого уровня пересылку производят с помощью присвоений.

Например:

a=10;
b=a

и другие аналогичные конструкции.

В ассемблере имеется множество различных команд присвоения. Я приведу лишь те, которые нам потребуются в работе.

```
MOV Rd, Rr  Пересылка между РОН
LDI Rd, K   Загрузка константы в РОН
IN Rd, P    Пересылка из РВВ в РОН
OUT P, Rr   Пересылка из РОН в РВВ
PUSH Rr     Сохранение байта в стеке
POP Rd      Извлечение байта из стека
```

```
…
ldi r16,0       ; загружаем значение 0 в регистр r16
in r16, PORTB   ;считываем значение из порта в регистр
out PINB,r16    ;выводим значение в порт
push r16        ; помещаем значение регистра в стек
…
```

В процессе написания программ, приведенных в книге, мы будем часто сталкиваться с этими командами, так что понимание принципов их работы придет довольно быстро.

1.2.9. Управление системой

Среди команд управления системой в языках высокого уровня можно найти аналог разве что для break. Команда break позволяет приостановить выполнение программы. Все остальные приведенные здесь команды предназначены именно для программирования устройств.

```
NOP     Нет операции
SLEEP   Переход в «спящий» режим
WDR     Сброс сторожевого таймера
BREAK   Приостановка программы (Используется только при отладке)
```

Примеры работы с этими командами мы также увидим при разработке прошивок для наших устройств.

1.2.10. Вызов процедур

Операция вызова процедур в ассемблере аналогична языкам высокого уровня. В программе создается метка, на которую затем производится переход.

```
RJMP A   Относительный безусловный переход
IJMP     Косвенный безусловный переход
RCALL A  Относительный вызов подпрограммы
ICALL    Косвенный вызов подпрограммы
RET      Возврат из подпрограммы
RETI     Возврат из подпрограммы обработки прерываний
```

В простейшем случае вызовы процедур могут иметь следующий вид:

```
…
call label  ;переход на метку label
…
label:      ; начало процедуры по метке label
…
ret         ; возврат из процедуры
…
```

Здесь основными инструментами, которые нам потребуются при разработке, являются RJMP, RCALL, RET и RETI. Разница между ними заключается в следующем: RJMP – это безусловный переход, который будет выполнен в любом случае и который не требует возврата, RET – это возврат из подпрограммы, переход в которую был вызван командой RCALL, IRET – возврат после вызова подпрограммы обработки прерываний.

1.2.11. Заключение

В этом разделе мы познакомились с основными командами ассемблера МК ATtiny. Конечно, набор команд не ограничивается лишь описанными здесь. Я описал лишь основные команды, которые нам потребуются для дальнейшей работы. Полный список всех команд микроконтроллера представлен в приложениях к книге в разделе «Команды ассемблера Atmel ATTiny13».

Для более глубокого изучения ассемблера я бы рекомендовал обратиться к литературе, представленной в приложениях к этой книге, в разделе «Библиография».

1.3. Пишем первую программу

После довольно продолжительного обсуждения теоретических аспектов самое время перейти к практической части и написать первую программу.

1.3.1. Электронный «Hello world»

В качестве примера предлагается собрать следующую несложную схему.

Суть работы устройства заключается в следующем: при нажатии на кнопку, подключенную к контакту PB3, загорается светодиод на выходе PB4.

Данный пример позволит нам лучше понять принципы построения алгоритмов при обработке входящих событий, таких как нажатие кнопки на устройстве.

Алгоритм работы программы будет построен следующим образом (рис. 1.4):

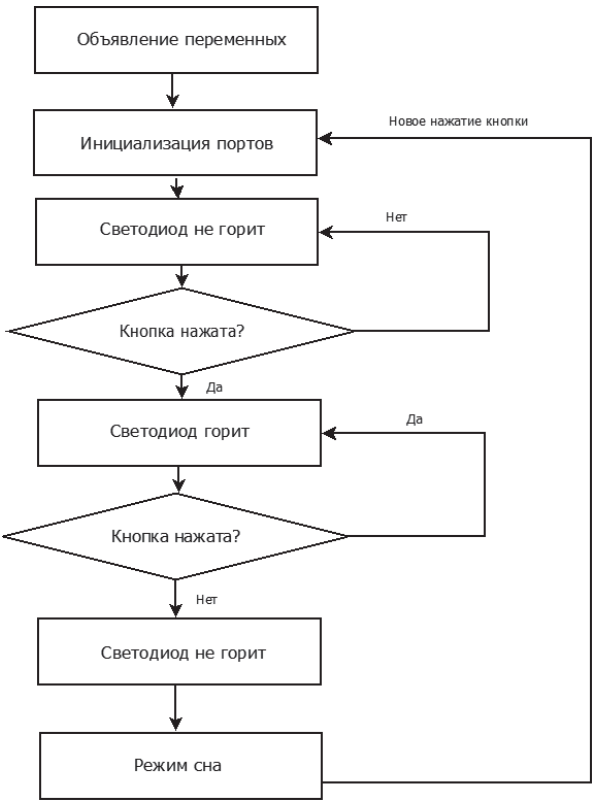

Рис. 1.4. Диаграмма для примера 1

Забегая вперед, отмечу, что подобные блок-схемы будут сопровождать описание каждого из описываемых в этой книге устройств.

Как видно на данной блок-схеме, весь алгоритм работы программы состоит из нескольких простых шагов. Сначала мы объявляем необходимые переменные и производим инициализацию портов. Далее мы переходим к рабочему блоку программы. Вначале у нас светодиод не горит, до тех пор, пока кнопка не нажата, мы находимся в бесконечном цикле. Как только мы нажали кнопку, светодиод загорается. Он будет гореть до тех пор, пока мы не отпустим кнопку. Далее микроконтроллер переходит в режим сна. Данный режим позволяет снизить затраты питания микроконтроллером, что актуально при использовании батареек или аккумуляторов.

Микроконтроллер выйдет из режима сна, когда произойдет какое-либо событие, в данном случае если мы снова нажмем кнопку. Теперь посмотрим, как реализовать этот алгоритм на языке ассемблера.

1.3.2. Основные элементы программы

Можно выделить следующие основные элементы блок-схемы:

1. Объявление переменных.
2. Инициализация портов.
3. Рабочий блок программы.
4. Режим сна.
5. Переход к п. 2 в случае новой активности.

Представим каждый из этих элементов в виде исходных кодов.

Для работы нашей программы нам потребуются одна переменная и два порта, один будет работать на вход, и один – на выход.

Указываем порты:

```
.equ  SW          =3        ;PB3  Этот контакт используется на вход (кнопка)
.equ  Led         =2        ;PB4  А этот вывод (светодиод)
```

Теперь объявим переменную. Напомню, что каждой переменной должен соответствовать регистр.

```
.def  temp        =r22      ; переменная для хранения временных значений
```

Далее нам необходимо произвести инициализацию переменных и портов. Обращаю внимание на необходимость принудительно обнулять значения переменных, так как в них может оказаться какой-нибудь «мусор».

```
reset:   ; метка reset
         clr ; обнуляем переменные
         cli ; сбрасываем прерывания

;-------------  инициализация портов  --------------
```

```
        ldi   temp,RAMEND; Загрузка в регистр r22(переменная temp) адреса
              верхней границы ОЗУ
        out   SPL,temp  ; Копирование значения из temp в регистр указателя стека SPL
        ldi   temp,0b11110111 ; Загрузка в temp числа 247 в двоичном виде
        out   DDRB,temp ; Копирование из temp в DDRB (PB4 – выход)
        com   temp  ; инверсия битов в temp
        out   PORTB,temp ; Копирование из temp в PORTB (PB3 – выход)
```

А вот и основной код программы, обеспечивающей функционал нашего устройства. Здесь два цикла. Первый цикл, в котором светодиод не горит, работает, пока мы не нажмем кнопку. Если *sbic PINB,SW* срабатывает, то следующая команда – переход к началу цикла пропускается. Затем второй цикл, в котором, наоборот, светодиод горит до тех пор, пока мы не отпустим кнопку.

```
;-------------  рабочий блок  --------------

main:      ; метка main
           clr  temp ; обнуляем temp
           cbi  PORTB,Led ; светодиод выключен
           sbic PINB,SW  ; если кнопка нажата, пропускаем следующую команду
           rjmp main ; переходим на main
main10:    sbi  PORTB,Led; светодиод горит
           sbis PINB,SW ; если кнопка отпущена, пропускаем следующую команду
           rjmp main10 ; переходим на main
```

Режим сна, про который я уже писал выше.

```
;-------------  переводим МК в режим сна  --------------

stop:
        sei
        ldi     temp,(1<<PCIE)    ; разрешаем прерывание
        out     GIMSK,temp        ;      PIN
        ldi     temp,(1<<PCIF)    ;
        out     GIFR,temp         ;      PB
        ldi     temp,(1<<PCINT3)  ;
        out     PCMSK,temp        ;      PB3
        ldi     temp,(1<<SE)|(1<<SM1)|(0<<SM0)
        out     MCUCR,temp
        sleep
        nop

        rjmp Reset    ; при пробуждении переходим к началу программы
```

Теперь посмотрим, как исходный код выглядит полностью. В таком виде его можно передать компилятору и затем записать на микроконтроллер.

```
.equ  SW          =3            ;PB3  Этот контакт используется на вход (кнопка)
.equ  Led         =2            ;PB4  А этот вывод (светодиод)
```

Теперь объявим переменную. Напомню, что каждой переменной должен соответствовать регистр.

```
.def  temp        =r22 ; переменная для хранения временных значений
```

Далее нам необходимо произвести инициализацию переменных и портов. Обращаю внимание на необходимость принудительно обнулять значения переменных, так как в них может оказаться какой-нибудь «мусор».

```
reset:  ; метка reset
        clr ; обнуляем переменные
        cli ; сбрасываем прерывания

;-------------  инициализация портов  --------------
        ldi     temp,RAMEND; Загрузка в регистр r22(переменная temp) адреса верхней
                границы ОЗУ
        out     SPL,temp  ; Копирование значения из temp в регистр указателя стека SPL
        ldi     temp,0b11110111 ; Загрузка в temp числа 247 в двоичном виде
        out     DDRB,temp ; Копирование из temp в DDRB (PB4 – выход)
        com     temp  ; инверсия битов в temp
        out     PORTB,temp ; Копирование из temp в PORTB (PB3 – выход)
```

А вот и основной код программы, обеспечивающей функционал нашего устройства. Здесь два цикла. Первый цикл, в котором светодиод не горит, работает, пока мы не нажмем кнопку. Если *sbic PINB,SW* срабатывает, то следующая команда – переход к началу цикла – пропускается. Затем второй цикл, в котором, наоборот, светодиод горит до тех пор, пока мы не отпустим кнопку.

```
;-------------  рабочий блок  --------------

main:   ; метка main
        clr   temp ; обнуляем temp
        cbi   PORTB,Led ; светодиод выключен
        sbic  PINB,SW  ; если кнопка нажата, пропускаем следующую команду
        rjmp  main ; переходим на main

main10: sbi   PORTB,Led; светодиод горит
        sbis  PINB,SW ; если кнопка отпущена, пропускаем следующую команду
        rjmp  main10 ; переходим на main
```

Режим сна, про который я уже писал выше.

```
;-------------  переводим МК в режим сна  --------------

stop:
        sei                     ;
        ldi     temp,(1<<PCIE)  ;     PIN
        out     GIMSK,temp      ;      PIN
        ldi     temp,(1<<PCIF)  ;
```

```
        out     GIFR,temp         ;     PB
        ldi     temp,(1<<PCINT3)  ;
        out     PCMSK,temp        ;      PB3
        ldi     temp,(1<<SE)|(1<<SM1)|(0<<SM0)
        out      MCUCR,temp
        sleep
        nop

        rjmp Reset    ; при пробуждении переходим к началу программы
```

1.3.3. Полный исходный код

Ниже приводится полный исходный код программы. В таком виде его можно использовать при компиляции. Подробнее об этом мы поговорим в следующей главе.

```
.include «tn13def.inc»

.equ  SW     =3
.equ  Led    =2

;-------------  объявляем переменные  --------------

.def  temp   =r22

;-------------  сегмент кода  --------------

.cseg
.org 0
rjmp reset

reset:
            clr
            cli

;-------------  инициализация портов  --------------
            ldi      temp,RAMEND
            out      SPL,temp
            ldi      temp,0b11110111
            out      DDRB,temp
            com      temp
            out      PORTB,temp

;-------------  рабочий блок  --------------

main:
            clr      temp
            cbi      PORTB,Led
            sbic     PINB,SW
            rjmp     main

main10:     sbi      PORTB,Led
            sbis     PINB,SW
            rjmp     main10

;-------------  переводим МК в режим сна  --------------
```

```
stop:
            sei
            ldi     temp,(1<<PCIE)
            out     GIMSK,temp
            ldi     temp,(1<<PCIF)
            out     GIFR,temp
            ldi     temp,(1<<PCINT3)
            out     PCMSK,temp
            ldi     temp,(1<<SE)|(1<<SM1)|(0<<SM0)
            out     MCUCR,temp
            sleep
            nop

            rjmp Reset
```

1.3.4. Заключение

В этой главе мы подробно рассмотрели язык ассемблера микроконтроллеров AVR и написали свою первую программу. Конечно, здесь я привел не все команды данного языка, так как эта книга не является справочным руководством по программированию. Поэтому для тех, кто хочет более глубоко изучить ассемблер AVR, я рекомендую обратиться к списку использованной литературы, приведенному в приложениях к книге.

В следующей главе мы подробно рассмотрим процесс компиляция и отладки кода, а затем и отладку программными средствами всего устройства. В качестве рабочего примера кода мы будем использовать программу, написанную в этой главе.