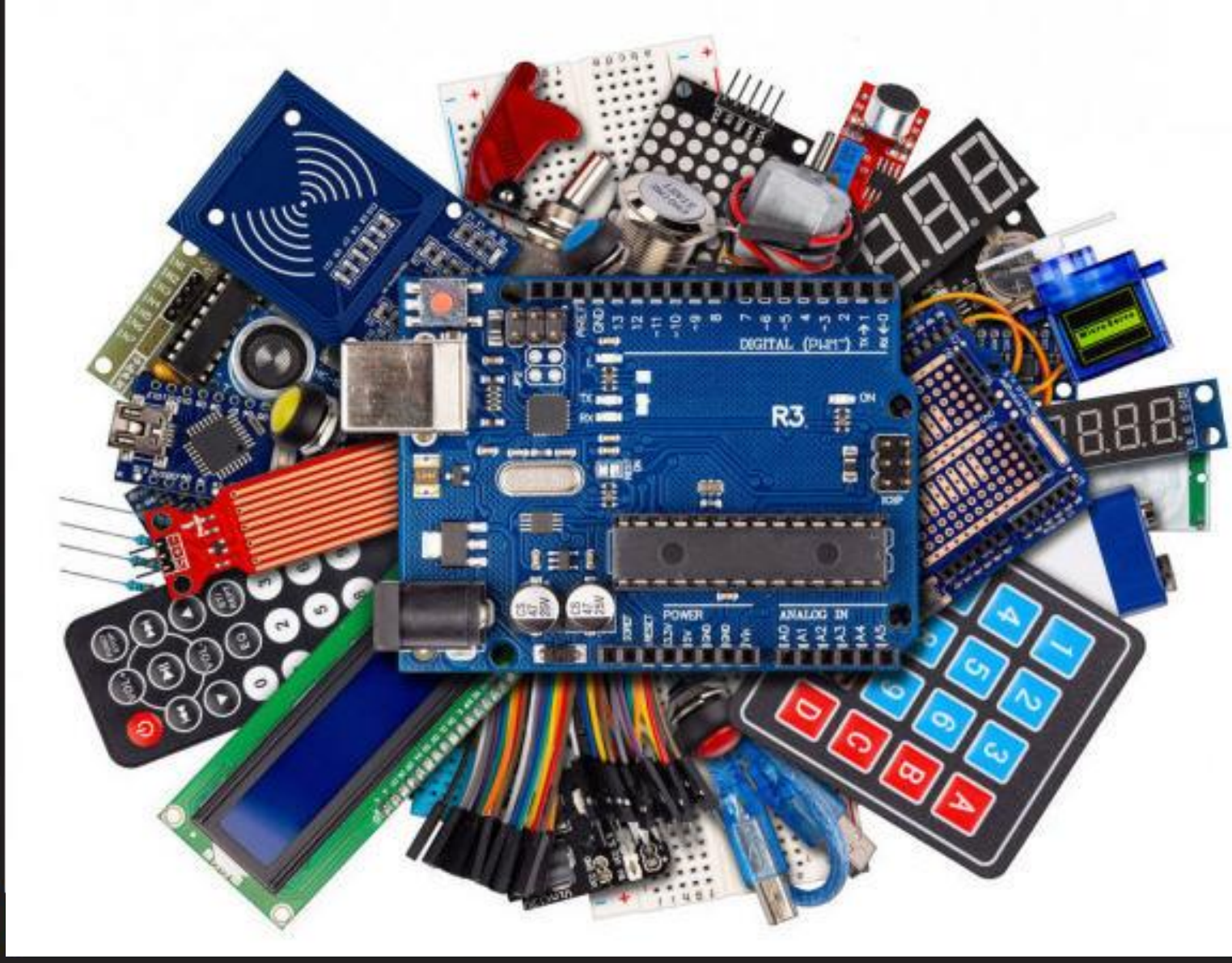


ОСНОВЫ ARDUINO

Майк МакРобертс

Основы ARDUINO

Майкл МакРобертс

Об авторе

Майк МакРобертс обнаружил Arduino в 2008 году, когда искал способы подключить датчик температуры к ПК, чтобы сделать детектор облаков для другого своего хобби - астрофотографии. После небольшого исследования, Arduino показался очевидным выбором, и детектор облаков был успешно изготовлен. У Майка началось увлечение Arduino. С тех пор он реализовал бесчисленное количество проектов с использованием Arduino. Он регулярно проводит семинары по Arduino в Соединенном Королевстве для хакпейсов, предприятий и других организаций.

Он также основал онлайн-бизнес по производству стартовых комплектов Arduino и компонентов под названием Earthshine Electronics.

Его следующий проект в использовании схемы на базе Arduino для отправки высотного воздушного шара к границе космоса, чтобы делать кадры и видео, черт возьми, с помощью ребят из UKHAS и CUSF.

Хобби Майка электроникой началось, когда он был ребенком, и комплекты электроники 100-в-1 от Radio Shack вошли в его список рождественских подарков. Он начал программировать как хобби, когда в подростковом возрасте приобрел компьютер Sinclair ZX81. С тех пор он никогда не оставался без компьютера.Он является членом London Hackspace и наблюдательным сотрудником Орпингтонского астрономического общества, а также регулярно участвует в работе форума Arduino. Он также любит скрываться в IRC в каналах Arduino, high altitude и london-hack-space (как «earthshine»), в Twitter как @TheArduinoGuy, а также в Google+ под именем «Mike McRoberts» (ищите аватар с изображением Футболка «А ты Ардуино?»). Когда он не возится с Arduinos, ему нравится заниматься астрономией, астрофотографией, ездой на мотоцикле и парусным спортом.

краткое содержание

СОДЕРЖАНИЕ

Вступление

Я впервые обнаружил Arduino в 2008 году, когда искал способы подключить датчики температуры к моему ПК, чтобы я мог сделать детектор облаков. Я хотел опробовать концепцию обнаружения облаков,о которой читал на метеорологическом форуме, и, поскольку она была экспериментальной, я не хотел тратить на нее много денег в случае неудачи. На рынке было много решений, но больше всего мне понравился Arduino. Это не только казалось простым и дешевым способом подключения необходимых мне датчиков, но и его можно было использовать для других интересных вещей. Тысячи проектов в блогах, видеосайтах и форумах показывали удивительные вещи, которые люди делали со своими Arduino. Казалось, было огромное чувство общности, когда все пытались помочь друг другу.

Было очевидно, что я могу повеселиться с Arduino. Однако я не хотел рыться в поисках информации по веб-сайтам. Я хотел купить книгу на эту тему, что-нибудь, что я мог бы держать в руке и читать в поезде на работу. Осмотревшись, я нашел одну книгу. К сожалению, она был очень простой и устаревшей и не оправдала моих ожиданий. Мне нужна была практическая книга, которая научила бы меня программированию и электронике, чтобы я занимался практическими вещами, вместо того, чтобы сначала пролезать страницы теории. Такой книги в то время просто не существовало.

Затем я основал Earthshine Electronics, чтобы продавать комплекты на базе Arduino. Чтобы дело пошло с этим комплектом, я подготовил небольшой буклет с учебными пособиями, чтобы люди начали работать. Этот небольшой буклет стал чрезвычайно популярным, и я получил сотни запросов от людей, которые спрашивали, когда я добавлю еще проекты или продам ли я печатную версию. На самом деле, я уже подумал, что было бы здорово написать исчерпывающую книгу для начинающих, набитую проектами и написанную в удобном для понимания стиле. Так появилась эта книга. Эта книга оказалась настолько успешной в обучении людей Arduino, что с тех пор она была обновлена до этого второго издания с улучшениями и обновленными разделами, относящимися к изменениям в мире Arduino с тех пор, как я начал этим заниматься.

Я написал эту книгу, исходя из предположения, что вы никогда раньше не занимались программированием или электроникой. Я также предполагаю, что вам неинтересно читать много теории, прежде чем вы приступите к созданию чего-либо с вашим Arduino. С самого начала книги вы сразу же погрузитесь в создание простого проекта. Оттуда вы будете работать в общей сложности над 50 проектами, пока не станете уверенными и опытными в разработке Arduino. Я считаю, что лучший способ чему-либо научиться - это учиться на ходу и заниматься практикой.

Книга работает так: первый проект знакомит с основными понятиями программирования Arduino и электроники. Каждый последующий проект основан на предыдущих проектах. К тому времени, когда вы завершите все 50 проектов, вы будете уверенны и опытны в создании собственных проектов. Вы сможете адаптировать свои новые навыки и знания, чтобы подключить к Arduino что угодно и создавать отличные проекты для развлечения или облегчения своей жизни.

Каждый проект начинается со списка необходимых частей. Я также привожу принципиальную схему, показывающую, как именно соединить Arduino и детали вместе с помощью перемычек. Для создания изображений деталей и макетов книги я использовал отличную программу с открытым исходным кодом Fritzing (http://fritzing.org). Программа позволяет дизайнерам документировать свои прототипы, а затем переходить к созданию печатных плат для производства.

После того, как вы создали свою схему, я предоставляю листинг кода для ввода в редактор

Arduino (IDE), который затем можно загрузить в ваш Arduino, чтобы проект заработал. У вас будет полностью рабочий проект. Только после того, как вы создали свой проект и убедились, что он работает, я объясню, как он работает. Вам объяснят электронику таким образом, чтобы вы знали, как работают компоненты и как правильно их подключить к Arduino.

Затем код будет объяснен вам шаг за шагом, чтобы вы точно поняли, что делает каждый раздел кода. Анализируя схему и код, вы поймете, как работает весь проект, и затем сможете применить полученные навыки и знания к более сложным проектам, а затем к своим собственным проектам в будущем. Стиль преподавания в этой книге очень прост. Даже если у вас нет абсолютно никакого опыта ни в программировании, ни в электронике, вы сможете легко усвоить и понимать понятия по ходу дела. Что еще более важно, вы получите удовольствие. Arduino - отличный и интересный продукт с открытым исходным кодом. С помощью этой книги вы поймете, насколько это просто.

Майк МакРобертс

ГЛАВА 1

Начало

С момента запуска проекта Arduino в 2005 году по всему миру было продано более 500 000 плат. Количество проданных неофициальных плат-клонов, без сомнения, превышает количество официальных плат, и вполне вероятно, что более миллиона плат Arduino или их вариантов находятся в свободном доступе. Его популярность постоянно растет, поскольку все больше и больше людей осознают удивительный потенциал этого невероятного проекта с открытым исходным кодом и его способность быстро и легко создавать крутые проекты с относительно неглубокой кривой обучения.

Самым большим преимуществом Arduino перед другими платформами разработки микроконтроллеров является простота использования, при которой люди, не являющиеся техническими специалистами, могут овладеть основами и создавать своисобственные проекты за относительно короткий промежуток времени.

Художники, в частности, считают, чтоэто идеальный способ быстро и без специальных знаний в области электроники создавать интерактивные произведения искусства. Существует огромное сообщество людей, использующих Arduinos и делящихся своими кодами и схемами, чтобы другие могли использовать их и форум Arduino - это то место, куда вы можете обратиться, если вам нужны быстрые ответы. Однако, несмотря на огромное количество информации, доступной в Интернете для новичков, большая часть этой информации разбросана по различным источникам, что затрудняет получение новичками нужной информации. Здесь и вписывается эта книга. На страницах, которые вы собираетесь прочитать, есть 50 проектов, которые разработаны, чтобы шаг за шагом познакомить вас с миром электроники и легко запрограммировать вашу Arduino. Я считаю, что лучший способ чему-либо научиться - это сразу же начать делать это. Вот почему эта книга не утомит вас страницами теории, прежде чем вы начнете использовать свой Arduino. Я знаю, каково это, когда вы впервые получаете Arduino или любой новый гаджет: вы хотите подключить его, подключить светодиод и сразу же заставить его мигать, а не сначала читать страницы руководств. Автор понимает это волнение, и поэтому мы сразу же погрузимся в подключение вещей к нашему Arduino, загрузку кода и начало работы. Я считаю, что это лучший способ изучить предмет, и особенно такой предмет, как физические вычисления, в чем и заключается суть Arduino.

О чем эта книга

Книга начинается с введения в Arduino, как установить программу, загрузить свой первый эскиз и убедиться, что ваш Arduino и программа работают правильно. Затем мы расскажем про Arduino IDE (интегрированную среду разработки) и то, как ее использовать, прежде чем мы перейдем непосредственно к некоторым проектам, переходя от самых простых вещей к сложным. Каждый проект начинается с описания того, как настроить устройство и какой код необходим для его работы.
Затем мы отдельно опишем код и , а также подробно объясним, как это работает. Все будет объяснено в виде понятных и простых шагов. Книга содержит множество схем и фотографий, чтобы как можно проще было проверить, правильно ли вы следуете проекту.В книге вы встретите некоторые термины и понятия, которые поначалу можете не понять. Не волнуйтесь; они станут ясны по мере того, как вы будете работать над проектами.

Что вам понадобится

Чтобы следовать проектам, описанным в этой книге, вам потребуются различные компоненты. Для начала я предлагаю вам начать с покупки компонентов для проектов, описанных в первых нескольких главах. По мере прохождения книги вы можете приобрести детали, необходимые для последующих проектов.Есть несколько других предметов, которые вам понадобятся или могут оказаться полезными. Конечно, вам нужно будет приобрести плату Arduino или одну из многих плат-клонов на рынке, таких как Freeduino, Seeeduino (да, действительно их три), Boarduino, Sanguino, Roboduino или любой другой «duino». Все они полностью совместимы с Arduino IDE, Arduino Shields и всеми остальными, что вы можете использовать с официальной платой Arduino. Помните, что Arduino - это проект с открытым исходным кодом; поэтому любой может создать клон или другой вариант Arduino. Для проектов в этой книге мы будем использовать Arduino Uno, хотя любая из доступных плат Arduino будет работать так же хорошо. Вам понадобится доступ к Интернету, чтобы загрузить Arduino IDE, программу, используемое для написания вашего кода Arduino и загрузки его на плату, а также для загрузки примеров кода из папки с кодами (если вы не хотите вводить их самостоятельно), а также любые библиотеки, которые могут потребоваться для работы вашего проекта. Наконец, самое главное, что вам понадобится -это энтузиазм и желание учиться. Arduino разработан как простой и дешевый способ познакомиться с электроникой микроконтроллера, и нет ничего слишком сложного для изучения, если вы готовы попробовать. Эта книга поможет вам в этом путешествии и познакомит вас с этим увлекательным и творческим хобби.

Что такое Arduino?

В Википедии говорится: «Arduino - это одноплатный микроконтроллер, разработанный, чтобы сделать процесс использования электроники в мультидисциплинарных проектах более доступным. Аппаратное обеспечение состоит из простой платы с открытым исходным кодом, разработанной на базе 8-разрядного микроконтроллера Atmel AVR, хотя новая модель была разработана на основе 32-разрядного микроконтроллера Atmel ARM. Программное обеспечение состоит из компилятора стандартного языка программирования и загрузчика, который выполняется на микроконтроллере ».

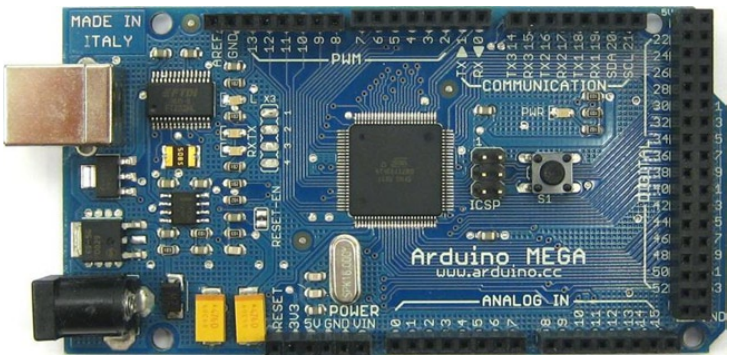


Рис.1-1. *Arduino Mega*

Говоря простым языком, Arduino - это крошечный компьютер, который можно запрограммировать для обработки входных и выходных сигналов между устройством и внешними компонентами, которые вы к нему подключаете. Arduino - это так называемая физическая или встраиваемая вычислительная платформа.
Например, простое использование Arduino - это включить свет на установленный период времени, скажем, на 30 секунд, после нажатия кнопки. В этом примере к Arduino будет подключена лампа, а также кнопка. Arduino будет терпеливо ждать, пока будет нажата кнопка.
Когда вы нажимаете кнопку, Arduino включает лампу и начинает отсчет. После того, как он отсчитывал 30 секунд, он выключает лампу, а затем продолжает ждать следующего нажатия кнопки. Вы можете использовать эту установку, например, для управления лампой в шкафу.
Вы можете расширить это понятие, чтобы устройство обнаруживало, когда дверца шкафа была открыта или произошло какое-либо другое событие, и автоматически включала лампу, выключая ее по истечении заданного периода времени. Вы можете пойти еще дальше и подключить пассивный инфракрасный датчик (PIR) для обнаружения движения и включения лампы при срабатывании триггера.
Это несколько простых примеров того, как вы можете использовать Arduino.
Arduino можно использовать для разработки автономных интерактивных объектов, или он может быть подключен к компьютеру, сети или даже Интернету для извлечения и отправки данных на Arduino и обратно, а затем воздействовать на эти данные. Например, его можно использовать для отправки набора данных, полученных от датчиков, на веб-сайт для отображения в виде графика.
Arduino может быть подключен к светодиодам, точечно-матричным дисплеям (см. Рисунок 1-2), кнопкам, переключателям, двигателям, датчикам температуры, датчикам давления, датчикам расстояния, приемникам GPS, модулям Ethernet или WiFi или ко всему, что выводит данные. или можно контролировать. Если заглянуть в Интернет, можно найти множество проектов, в которых Arduino использовалась для чтения данных с огромного множества устройств или управления ими.

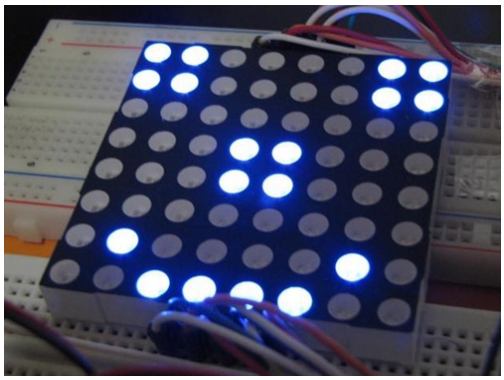


Рис. 1-2. *Точечно-матричный дисплей, управляемый Arduino*

Плата Arduino состоит из микропроцессора Atmel AVR, генератора вырабатывающий заданную частоту и стабилизатора напряжения 5 В. Он также имеет разъем USB, позволяющий подключать его к ПК для загрузки или получения данных. Плата имеет разъемы,через которые осуществляется доступ к пинам ввода / вывода микроконтроллера, чтобы вы могли подключать эти пины к другим схемам или датчикам и т. д.
Чтобы запрограммировать Arduino (заставить его делать то, что вы хотите), используется Arduino IDE, часть бесплатного программного обеспечения, которое позволяет вам программировать на языке, который понимает Arduino. Язык основан на C / C ++ и даже может быть расширен с помощью библиотек C ++. IDE позволяет вам писать программы, которая представляет собой набор пошаговых инструкций, которые вы затем загружаете в Arduino. Затем ваш Arduino будет выполнять эти инструкции и взаимодействовать со всем, что вы к нему подключили. В мире Arduino программы известны как «скетчи».

Аппаратное и программное обеспечение Arduino имеют открытый исходный код, что означает, что код, схемы, дизайн и т. д. находятся в свободном доступе. Поэтому, существует множество плат-клонов на базе Arduino, которые можно купить или изготовить самому. В самом деле, ничто не мешает вам приобрести соответствующие компоненты и создать свой собственный Arduino на макете или на своей собственной печатной плате. Единственное условие, которое делает команда Arduino, заключается в том, что вы не можете использовать слово Arduino, поскольку оно зарезервировано для официальной платы. Arduino также может быть расширен за счет использования «шилдов», которые представляют собой печатные платы, содержащие другие устройства (например, приемники GPS, ЖК-дисплеи, модули Ethernet и т. д.), которые вы можете просто подключить к верхней части вашего Arduino, чтобы получить дополнительную функциональность. Шилды также расширяют выводы (места на вашем Arduino, где вы можете выводить или вводить данные) до верхней части их собственной печатной платы, поэтому у вас по-прежнему есть доступ ко всем из них. Вам не обязательно использоватьшилд, если вы этого не хотите,так как вы можете сделать точно такую же схему, используя макет, какой-нибудь Stripboard или Veroboard (платы, состоящие из полосок меди в сетке для проектов с домашней пайкой), или сделав свою собственную печатную плату. Большинство проектов в этой книге выполнено с использованием схем на макетной плате.

Поскольку дизайн является открытым исходным кодом, клонированная плата, такая как Freeduino, может быть на 100 процентов совместима с Arduino и, следовательно, с любой программой,шилдами и т. д.. Кроме того, Due (который является подлинным Arduino) имеет некоторые проблемы, такие как его работа с напряжением 3 В, которая может работать не со всеми шилдми.

Доступно множество различных вариантов Arduino. Самая распространенная из них - Uno, выпущенная в 2010 году (в настоящее время находится в редакции 3), и это плата, которую вы, скорее всего, увидите, как используемую в подавляющем большинстве проектов Arduino в Интернете. Последние дополнения к линейке продуктов - это Arduino Leonardo и Arduino Due, которые являются первым вторжением команды Arduino в использование процессоров ARM вместо процессоров архитектуры AVR. Due имеет 32-битный процессор вместо обычного 8-битного процессора, работают на частоте 84 МГц и имеют 512 КБ флеш-памяти.

Вероятно, самый универсальный Arduino и, следовательно, причина его популярности - Uno (до Uno самым популярным был Duemilanove). Это связано с тем, что они используют стандартный 28-контактный чип, подключенный к гнезду IC (интегральной схемы). Прелесть этой схемы в том, что если вы делаете что-то аккуратное с помощью Arduino, а затем хотите превратить его во что-то постоянное, вместо использования относительно дорогой платы Arduino вы можете просто использовать Arduino для разработки своего устройства и программирования чипа, а затем вытащить микросхему из платы и поместить ее на свою печатную плату в собственном устройстве. Затем за пару фунтов или баксов вы можете заменить чип AVR в вашем Arduino на новый. Чип должен быть предварительно запрограммирован с помощью загрузчика Arduino (программа,зашитое в чип, чтобы его можно было использовать с Arduino IDE), но вы можете либо приобрести программатор AVR, чтобы записать загрузчик самостоятельно, либо вы можете купить готовый запрограммированный чип.

Более новый Arduino Uno имеет на борту программируемый USB-чип, который позволяет вам прошивать чип таким образом, что при подключении устройства к компьютеру он будет отображаться как USB-устройство. Это позволяет использовать Arduino в качестве интерфейса для создания собственных USB-устройств. Однако это расширенная функция и не для слабонервных.

Если вы выполните поиск Arduino в Интернете, вы будете поражены огромным количеством веб-сайтов, посвященных Arduino или на которых кто-то использовал Arduino для создания классного проекта. Arduino - удивительное устройство, которое позволит вам создавать что угодно, от интерактивных произведений искусства (см. Рис. 1-3) до роботов. С большим энтузиазмом по поводу того, как научиться программировать Arduino и заставить его взаимодействовать с другими компонентами, а также немного воображения, вы можете построить все, что только сможете придумать.

Рис. 1-3. *Арт-инсталляция Антроса Ричарда В. Гилбанка, управляемая с помощью Arduino*

Эта книга даст вам необходимые навыки, чтобы начать заниматься этим увлекательным и творческим хобби. Итак, теперь, когда вы знаете, что такое Arduino, давайте подключим его к нашему компьютеру и начнем его использовать.

Настройка вашего Arduino

В этом разделе объясняется, как настроить Arduino и IDE в первый раз. Даны инструкции как для Windows, так и для Mac. Если вы используете Linux, обратитесь к инструкциям по началу работы на веб-сайте Arduino по адресу http://playground.arduino.cc/learning/linux. Если у вас другой тип платы, обратитесь к соответствующей странице в руководстве по началу работы на веб-сайте Arduino.

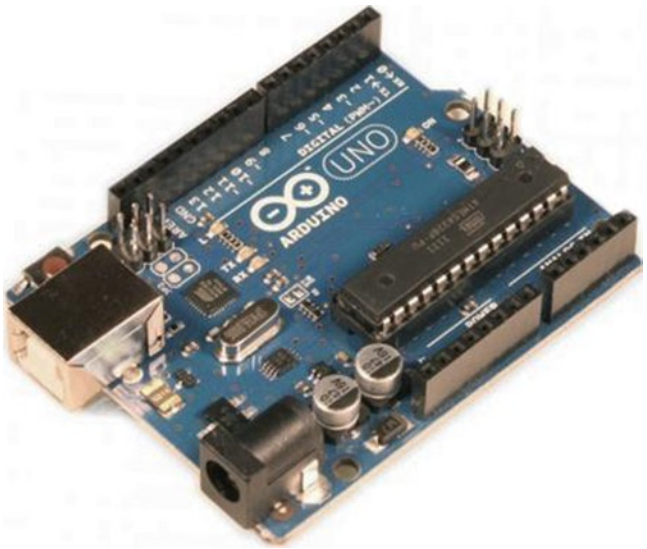


Рис. 1-4. *Arduino Uno*

Вам также понадобится USB-кабель (с разъемом от A до B), который является кабелем того же типа, который используется в большинстве современных USB-принтеров. Если у вас есть Arduino Nano, вам понадобится кабель USB A - Mini-B.

Затем вам нужно загрузить Arduino IDE. Это программа, которое будет использоваться для написания своих программ (или скетчей) и загрузки их на вашу плату. Для получения последней версии IDE перейдите на страницу загрузки Arduino по адресу

http://arduino.cc/en/Main/Software и получите версию, соответствующую вашей операционной системе.

Теперь вам нужно подключить плату Arduino перед установкой драйверов и программного обеспечения. Подключите USB-кабель к Arduino, а другой конец подключите к USB-разъему на вашем компьютере. Вы увидите, что на вашей плате загорится зеленый индикатор питания (помеченный PWR), чтобы показать вам, что на нее есть питание. Если вы работаете в Windows, он попытается установить драйверы для Arduino. Эта автоматическая попытка не удастся, и вы получите сообщение, что «Программное обеспечение драйвера устройства не было успешно установлено» (Рисунок 1-5); не беспокойся об этом.

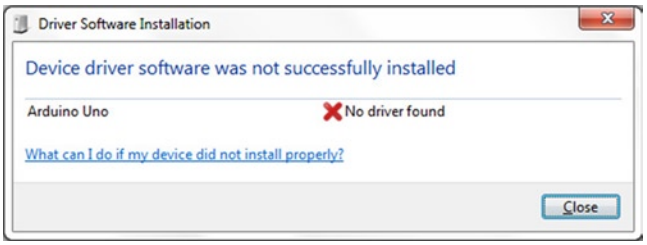


Рис. 1-5. *Автоматическая попытка Windows установить драйверы не удается. Это нормально*

Щелкните меню «Пуск», затем щелкните «Панель управления». Перейдите в раздел «Система и безопасность», нажмите «Система» и откройте Диспетчер устройств. В списке оборудования под «Другие устройства» вы должны увидеть что-то похожее на рис. 1-6, на котором у вас есть «Arduino Uno» с желтым значком опасности над ним.

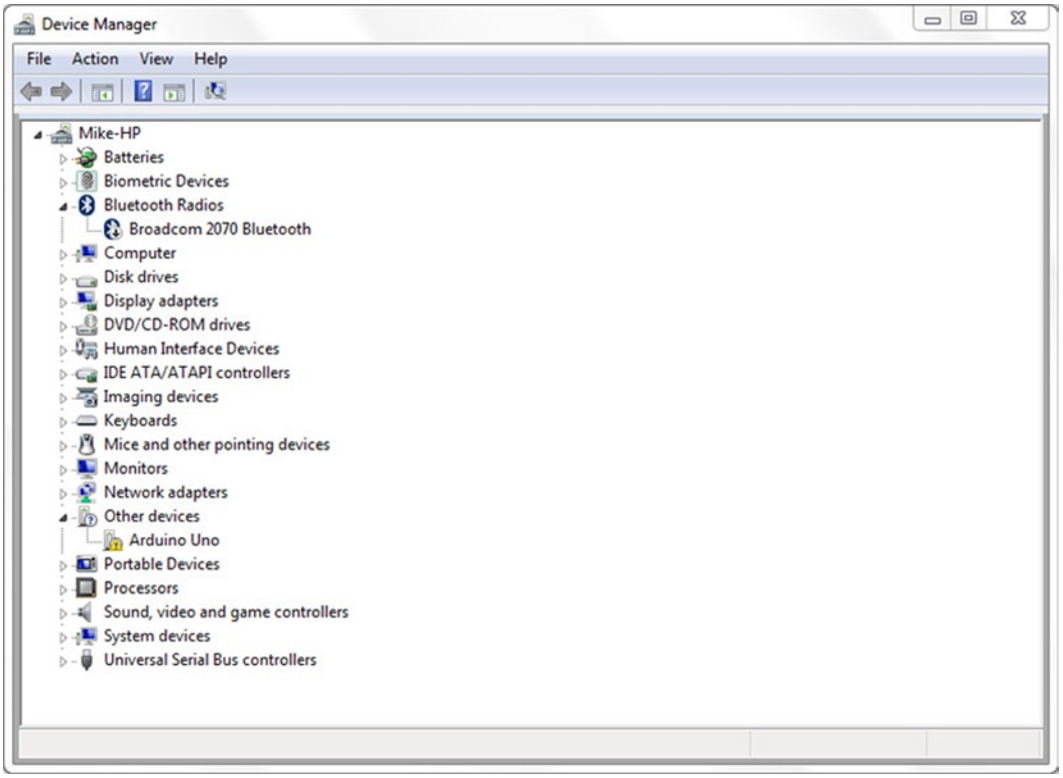


Рис. 1-6. *Диспетчер устройств Windows*

Щелкните правой кнопкой мыши значок Arduino Uno в списке и выберите «Обновить программное обеспечение драйвера» (рисунок 1-7).

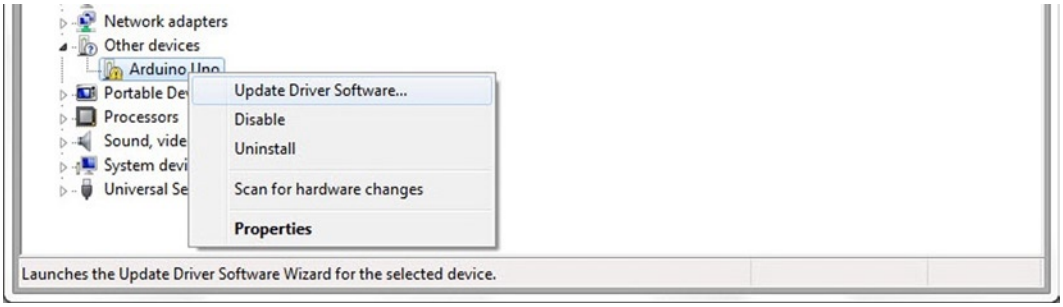


Рис. 1-7.

Теперь выберите «Найти на моем компьютере драйверы».

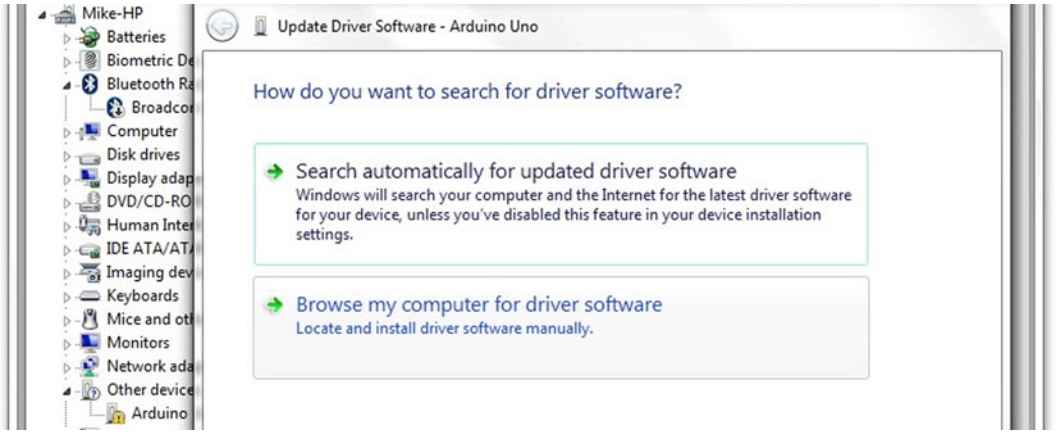


Рис. 1-8. *Нажмите «Найти на моем компьютере драйверы»*

Затем перейдите в папку с драйверами установки Arduino и нажмите кнопку «Далее». Windows завершит установку драйвера. Если вы получили сообщение «Windows не может проверить издателя этого программного обеспечения драйвера», нажмите «Все равно установить этот драйвер».Теперь, когда драйверы установлены, вы готовы открыть Arduino IDE. В Windows дважды щелкните файл arduino.exe в распакованной папке Arduino. IDE откроется и представит вам пустой эскиз, как на рис. 1-9.

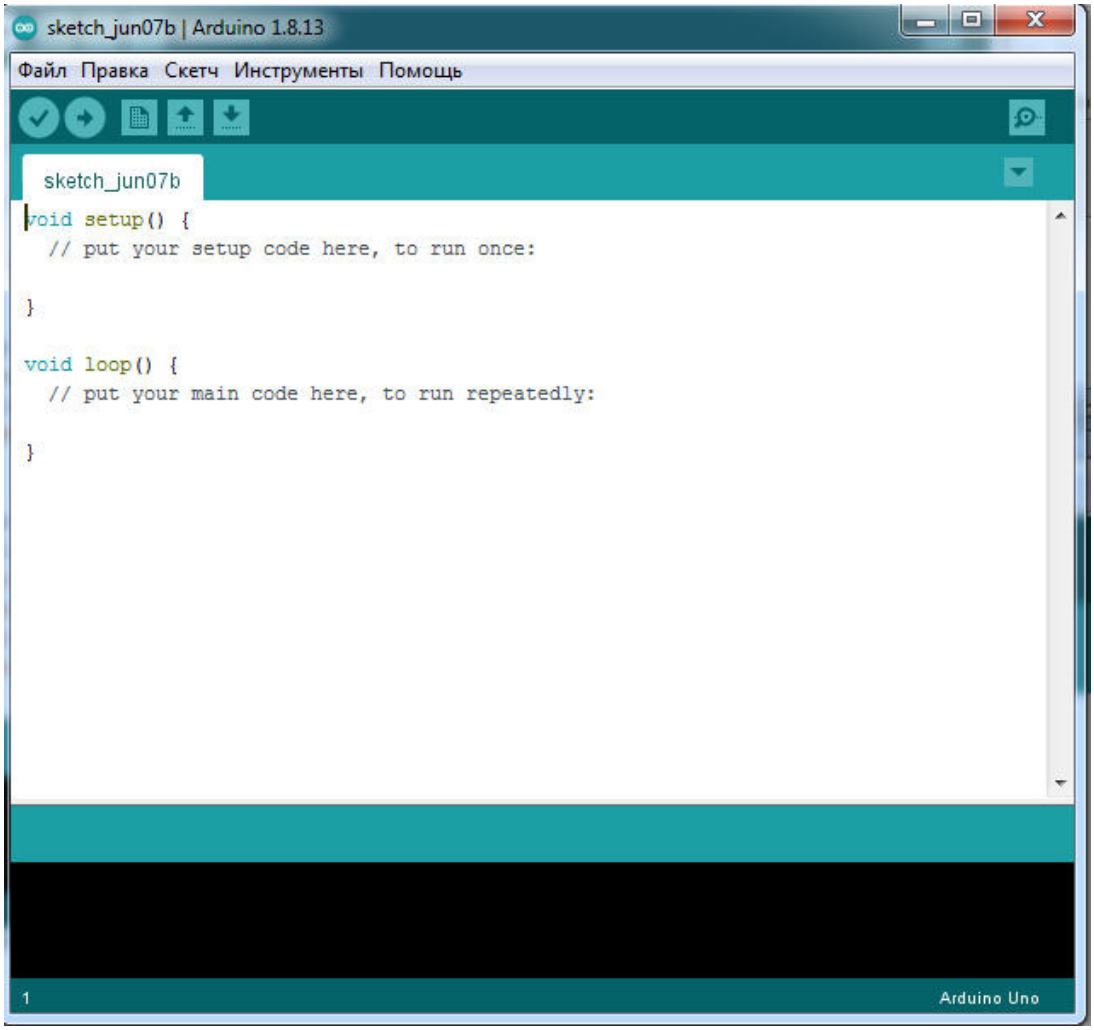


Рис. 1-9. *Arduino IDE*

Затем откройте пример скетча, чтобы протестировать IDE и Arduino. Щелкните Файл, затем Примеры, затем 01.Basics,и, наконец, Blink (см. рис. 1-10).

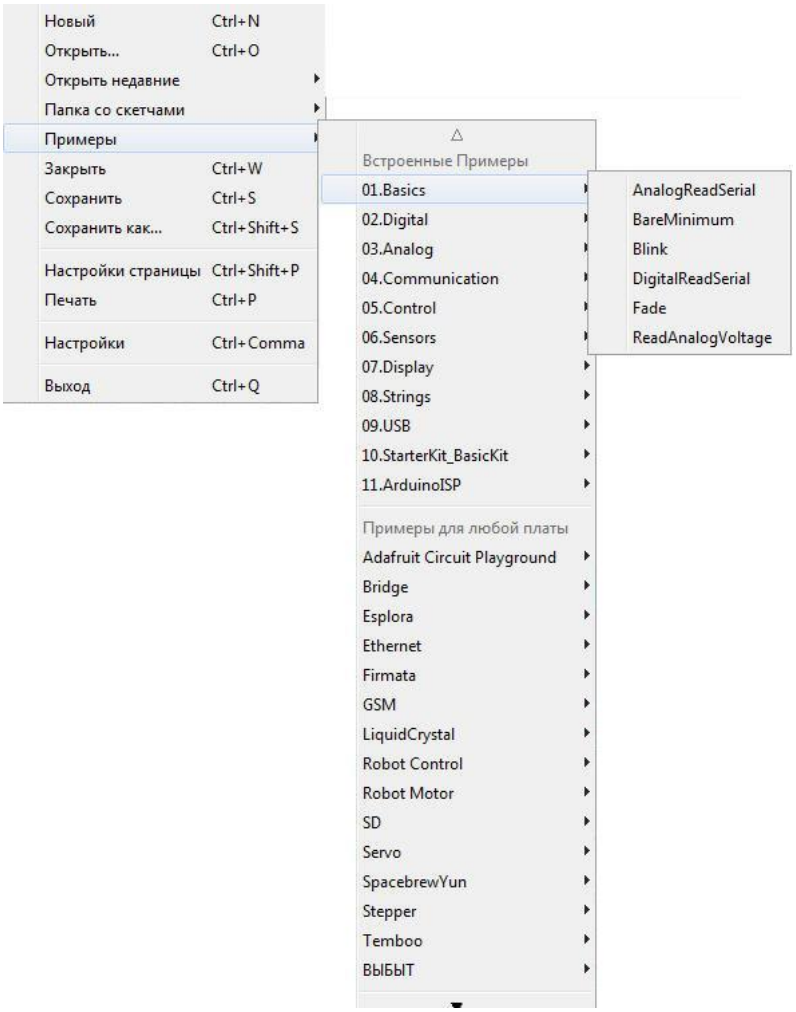


Рис. 1-10. *Файловое меню Arduino. Выберите скетч Blink*

Загрузится скетч-пример Blink в IDE и будет выглядеть примерно так, как показано на рис. 1-11.

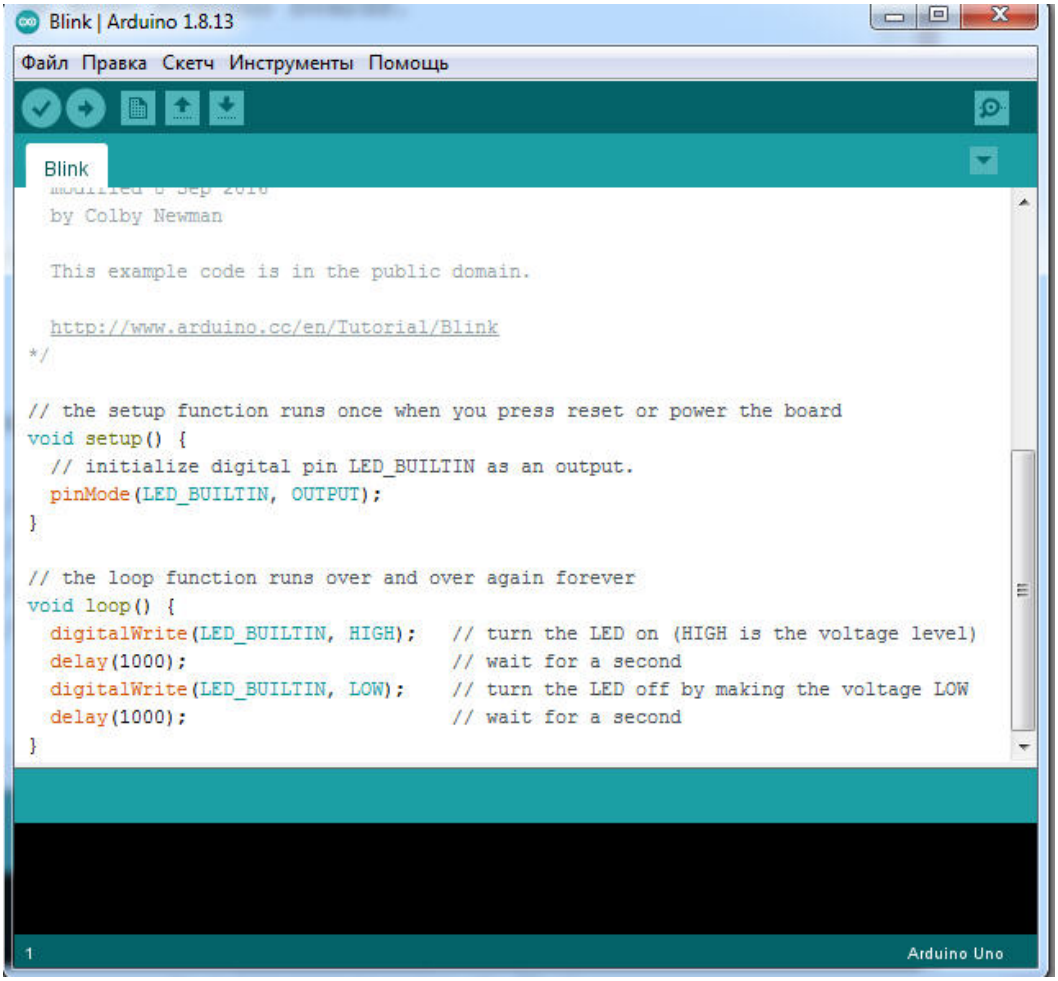


Рис. 1-11. *IDE с загруженным скетчем Blink*

Затем вам нужно будет выбрать свою ПЛАТУ из списка (см. Рис. 1-12) в Инструменты ➤ плата. Для Arduino Uno выберите это в верхней части списка.

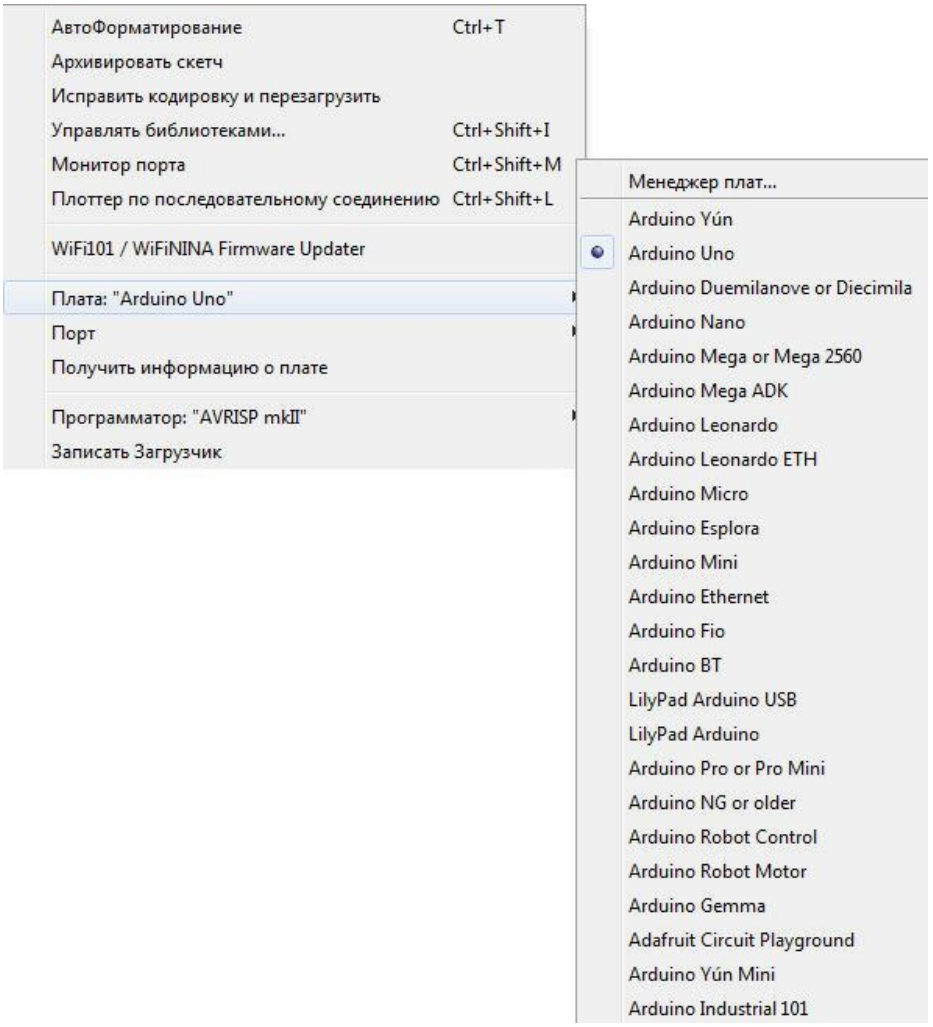


Рис. 1-12. *Выберите тип вашей платы*

Выберите последовательное устройство платы Arduino в Инструменты Tools Последовательный порт (см. Рисунок 1-13). Если вы не уверены, какой у вас порт, отключите Arduino и проверьте доступные порты, затем снова подключите Arduino и посмотрите, какой порт теперь появился (вам может потребоваться закрыть и снова открыть меню, чтобы он отобразился).
F

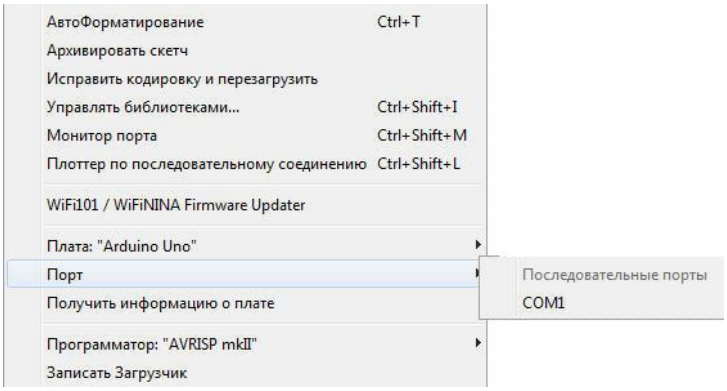


Рис. 1-13. *Выберите порт*

Загрузите свой первый скетч

Теперь, когда вы установили драйверы и IDE и выбрали соответствующие плату и порты, вы можете загрузить пример скетча Blink в Arduino, чтобы проверить, все ли работает правильно, прежде чем переходить к первому проекту. После того, как вы загрузили скетч Blink в Arduino IDE, вы можете загрузить его в Arduino, просто нажав кнопку «Загрузить» (вторая кнопка слева, которая представляет собой стрелку вправо) и посмотрите на свой Arduino (если у вас есть Arduino Mini, NG или другая плата, вам может потребоваться нажать кнопку сброса на плате до нажатия кнопки загрузки).
IDE скажет «Компиляция скетча. . . », Который затем изменится на« Загрузка. . . . » Затем индикаторы RX и TX должны начать мигать, показывая, что данные передаются с вашего компьютера на плату. После успешной загрузки скетча в строке состояния IDE появятся слова «Done uploading», а индикаторы RX и TX перестанут мигать.
Через несколько секунд вы должны увидеть, как светодиод на контакте 13 (крошечный светодиод над светодиодами TX и RX) начинает мигать с интервалом в одну секунду. Если это так, значит, вы только что успешно подключили Arduino, установили драйверы и программное обеспечение и загрузили пример скетча. Эскиз Blink - это очень простой эскиз, в котором мигает светодиод 13, представляющий собой крошечный оранжевый светодиод, припаянный к плате и также подключенный к цифровому выводу 13 микроконтроллера (см. Рисунок 1-14).

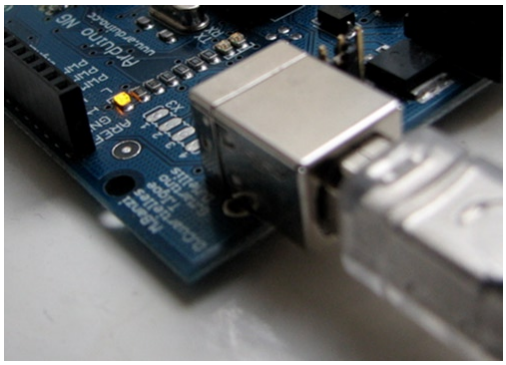


Рис. 1-14. *Мигание светодиода 13*

Прежде чем мы перейдем к проекту 1, давайте взглянем на IDE Arduino, и я объясню, что делает каждая из частей программы.

Arduino IDE

IDE Arduino IDE (интегрированная среда разработки) Arduino - это то, что вы будете использовать для написания кода для вашего Arduino, его проверки и загрузки на свою плату. Когда вы откроете IDE Arduino, она будет похожа на версию для Windows на изображении ниже (рис. 1-15).

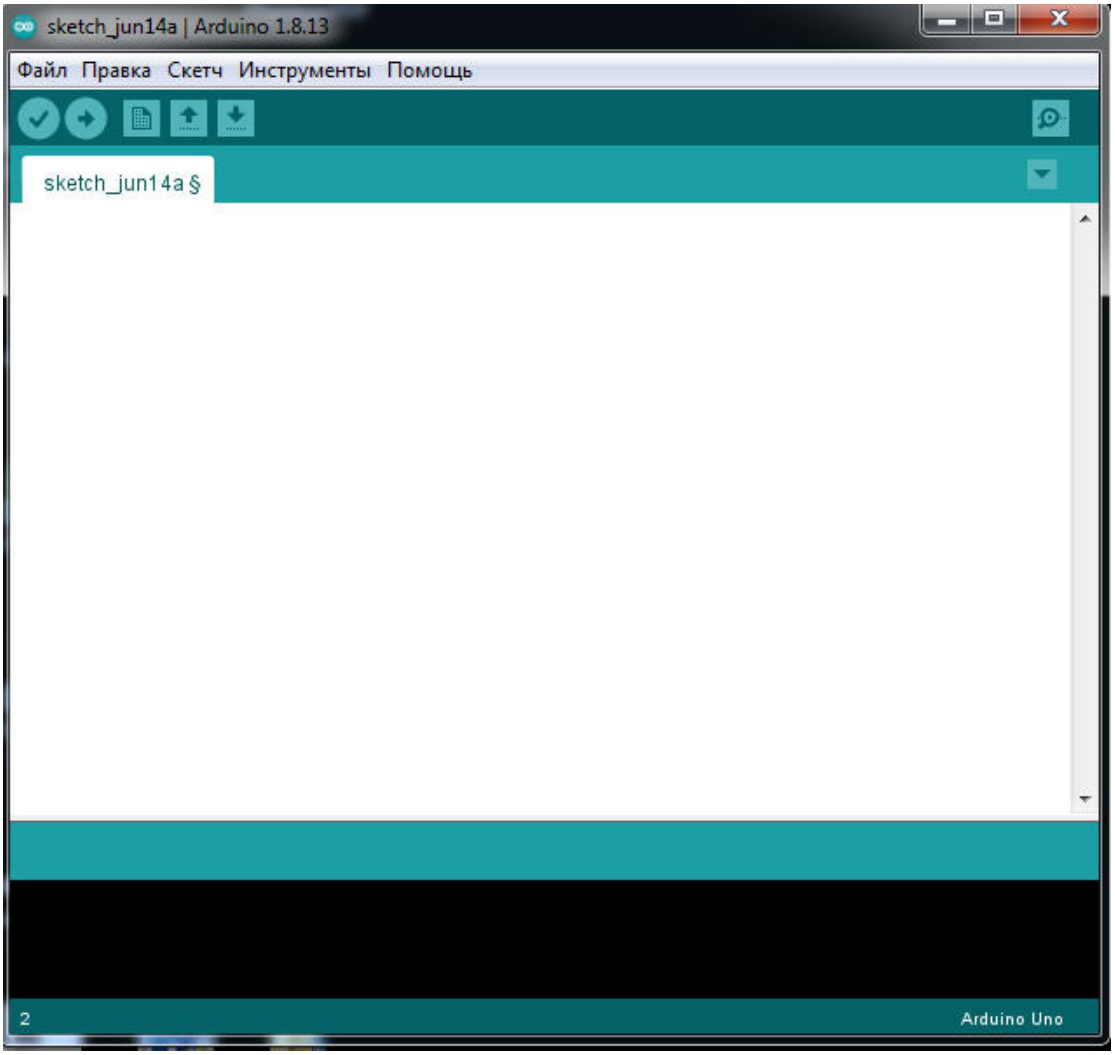


Рис. 1-15. *Так выглядит IDE при открытии приложения*

IDE разделена на четыре части: меню «Файл» в верхней части программы (или в верхней части экрана в OSX), панель инструментов под ним, код или окно эскиза в центре и окно сообщения внизу. . Панель инструментов состоит из шести кнопок, а под панелью инструментов находится вкладка или набор вкладок с именем файла скетча внутри вкладки. В дальнем правом углу есть еще одна кнопка, которая вызывает окно Serial Monitor.Вверху находится меню файлов с раскрывающимися меню под заголовками Arduino, Файл,Правка,Скетч,Инструменты и Помощь. Кнопки на панели инструментов (см. Рис. 1-16) обеспечивают удобный доступ к наиболее часто используемым функциям в этом меню файла.

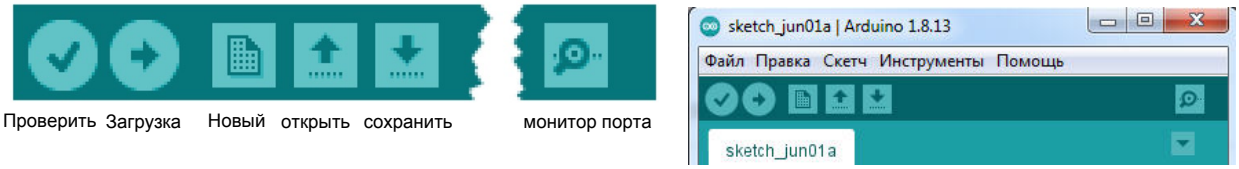


Рис. 1-16. *. Панель инструментов.*

Кнопки панели инструментов перечислены на Рисунке 1-16. Функции каждой из кнопок следующие:

Таб. 1-1. *Функции кнопок панели инструментов*

Проверить	Проверяет код на наличие ошибок
Загрузка	Загружает текущий скетч в Arduino
Новый	Создает новый пустой скетч
Открыть	Показывает список скетчей в вашем Sketchbook для открытия
Сохранить	Сохраняет текущий скетч в ваш Sketchbook
Монитор порта	Отображает последовательные данные, отправляемые с Arduino

Кнопка «**Проверить**» используется для проверки правильности кода и отсутствия ошибок перед его загрузкой на плату Arduino.
Кнопка «**Загрузка**» загрузит код из текущего окна скетча в ваш Arduino. Перед загрузкой необходимо убедиться, что у вас выбраны соответствующие плата и порт (в меню «Инструменты»). Важно, чтобы вы сохранили свой скетч перед загрузкой его на плату на случай, если из-за странной ошибки ваша система зависнет или IDE выйдет из строя. Также рекомендуется проверить код перед загрузкой, чтобы убедиться в отсутствии ошибок, которые необходимо сначала отладить.
Кнопка «**Новый**» создаст совершенно новый и пустой скетч, в который вы можете ввести свой код. IDE попросит вас ввести имя и местоположение для вашего скетча (попробуйте использовать местоположение по умолчанию, если возможно), а затем предоставит вам пустой скетч, готовый для кодирования. Вкладка в верхней части скетча теперь будет содержать имя, которое вы дали новому скетчу.
Кнопка «**Открыть**» представит вам список скетчей, хранящихся в вашем альбоме, а также список примеров скетчей, которые вы можете опробовать с различными периферийными устройствами после подключения. Примеры набросков неоценимы для новичков, которые могут использовать их в качестве основы для собственного скетча. Откройте соответствующий скетч для подключаемого устройства,а затем измените код в соответствии с вашими потребностями.
Кнопка «**Сохранить**» сохранит код из окна скетча в файл скетча. После завершения вы получите сообщение«Сохранение готово» в нижней части окна кода.

Монитор порта - очень полезный инструмент, особенно для отладки вашего кода. Монитор отображает последовательные данные, отправляемые с вашего Arduino (USB или последовательная плата). Вы также можете отправить последовательные данные обратно в Arduino с помощью Монитора порта. Если вы нажмете кнопку Монитор порта, вы увидите изображение, подобное изображенному на Рис. 1-17.

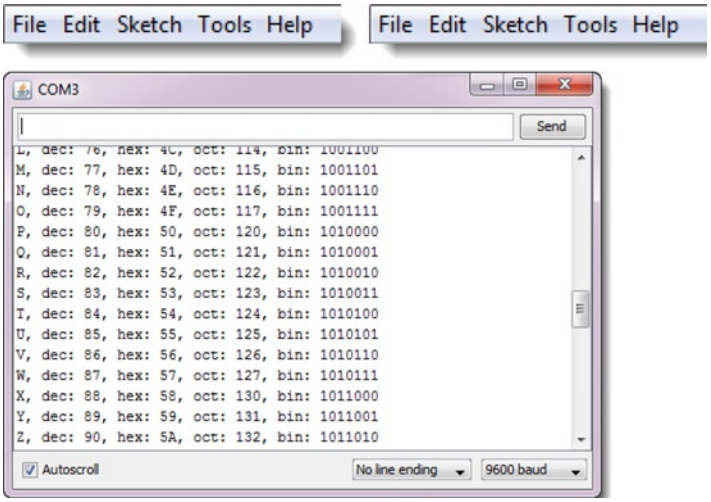


Рис. 1-17. *Используемый монитор порта*

В правом нижнем углу вы можете выбрать скорость передачи, с которой последовательные данные будут отправляться в / из Arduino. Скорость передачи - это скорость в секунду с которой биты (данные) отправляются на / с платы. По умолчанию установлено значение 9600 бод, что означает, что если вы отправляете текстовый роман по линии последовательной связи (в данном случае через USB-кабель), то 1200 букв или символов романа будут отправляться в секунду (9600 бит / 8 бит на символ = 1200 байтов или символов - биты и байты будут объяснены позже).
Вверху находится пустое текстовое поле для ввода текста для отправки обратно в Arduino и кнопка «Отправить» для отправки текста в этом поле. Обратите внимание, что Монитор порта не будет получать никаких последовательных данных, если вы не настроили код внутри своего скетча для отправки последовательных данных из Arduino. Точно так же Arduino не получит никаких данных, если вы не закодировали его для этого.
В левом нижнем углу есть поле для галочки, где вы можете выбрать, хотите ли вы, чтобы данные в окне Монитора порта автоматически прокручивались или нет.
Поле слева от меню скорости передачи влияет на данные, отправляемые с Монитора порта обратно в Arduino.
По умолчанию установлено значение «без окончания строки». Это означает, что когда вы вводите данные в текстовое поле на Мониторе порта и нажимаете «отправить», данные будут отправлены как есть. Если щелкнуть раскрывающееся меню, появятся еще три параметра: «Новая строка» и «Возврат каретки». и Оба NL + Cr. При выборе одного из них монитор последовательного порта (МПП) добавит код ascii для новой строки, возврата каретки или обоих в конце любых данных, введенных в окно МПП, когда вы нажмете кнопку «Отправить». Имейте это в виду при обработке данных, отправленных с МПП обратно на Arduino.
Наконец, основная область - это место, где будут отображаться ваши данные. На изображении выше Arduino выполняет скетч ASCIITable, который можно найти в примерах связи. Эта программа выводит символы ASCII из Arduino через последовательный порт (USB-кабель) на ПК, где их затем отображает МПП.
Когда вы научитесь обмениваться данными через последовательный порт с Arduino и обратно, вы можете использовать другие программы, такие как Processing, Flash, MaxMSP и т. д. для связи между Arduino и вашим ПК. Мы будем использовать МПП позже в наших проектах, когда мы будем считывать данные с датчиков

Под окном сообщений в левом нижнем углу вы увидите число. Это текущая строка, на которой находится курсор в окне кода. Если у вас есть код в вашем окне, и вы перемещаетесь вниз по строкам кода (используя клавишу ↓ на клавиатуре), вы увидите, что число увеличивается по мере продвижения вниз по строкам кода. Это полезно для поиска ошибок, отмеченных сообщениями об ошибках.

В верхней части окна IDE вы увидите различные меню, щелкнув по которым можно получить доступ к дополнительным пунктам меню (см. Рис. 1-18).

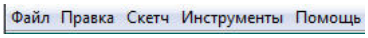


Рис. 1-18. *Меню IDE*

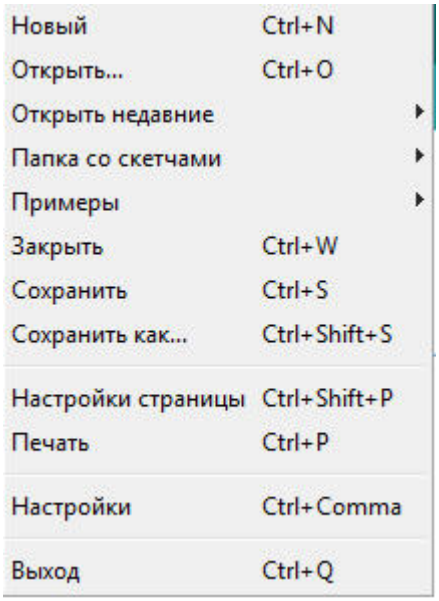


Рис. 1-20. *Меню «Файл»*

Следующее меню - это меню «Файл». (см. рис. 1-20). Здесь вы получаете доступ к параметрам для создания нового скетча, просматриваете скетчи, хранящиеся в вашей папке с скетчами, примеры файлов, параметры для сохранения вашего скетча (или «Сохранить как», если вы хотите дать ему другое имя). У вас также есть возможность загрузить свой скетч в Arduino, загрузить с помощью программатора (мы не будем использовать эту функцию), а также параметры печати для распечатки вашего кода.

Внизу находится опция «Настройки». В ней находится окно настроек, в котором вы можете изменить различные параметры IDE, например, где хранится ваша папка со скетчами по умолчанию и т. д. Наконец, есть параметр «Выход», который позволяет выйти из программы.

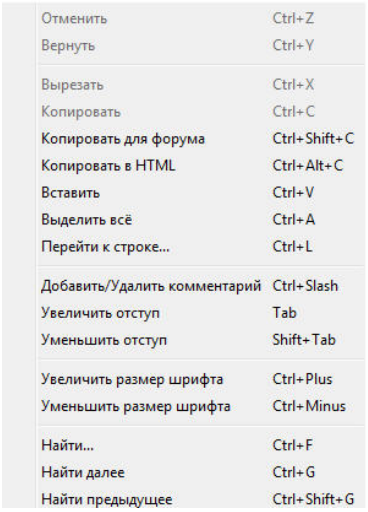


Рис. 1-21. *Меню«Правка»*

Далее идет меню «Правка» (см. Рис. 1-21). Здесь вы получаете опции, позволяющие вырезать, копировать и вставлять разделы кода. Выделяйте весь свой код, а также находите в нем определенные слова или фразы. Комментируйте свой код, чтобы объяснить, как он работает), а также увеличивайте или уменьшайте отступы. Также включены полезные опции «Отменить» и «Вернуть», которые пригодятся, если вы допустили ошибку.
Параметр «Копировать для форума» скопирует код в окно Sketch, но в формате, который при вставке в форум Arduino (или большинство других форумов в этом отношении) будет отображаться так же, как в среде IDE, вместе с раскраской синтаксиса и так далее.

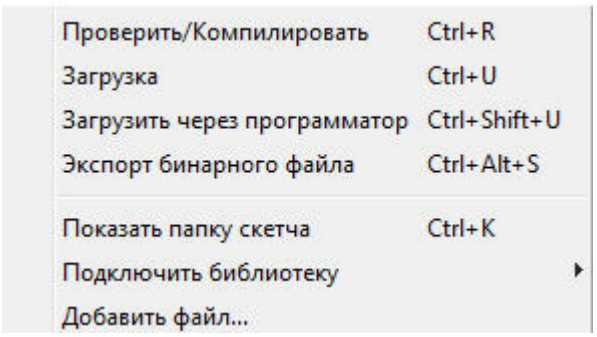


Рис. 1-22. Меню скетча

Наше следующее меню - это меню Sketch (см. Рис. 1-22), которое дает нам доступ к функциям Проверить/компилировать и некоторым другим полезным функциям, которые вы будете использовать позже. К ним относится опция«подключить библиотеку», при нажатии на которую открывается список доступных библиотек, хранящихся в папке ваших библиотек.

Библиотека - это набор кода, который вы можете включить в свой эскиз для улучшения функциональности вашего проекта.Это способ не дать вам заново изобретать велосипед, повторно используя код, уже созданный кем-то другим, с которыми вы можете столкнуться при использовании Arduino.

Например, вы найдете одну из библиотек - Stepper, которая представляет собой набор функций, которые вы можете использовать в своем коде для управления шаговым двигателем. Кто-то любезно уже создал все необходимые функции, необходимые для управления шаговым двигателем, и, включив библиотеку Stepper в наш скетч, мы можем использовать эти функции для управления двигателем по своему желанию. Сохраняя часто используемый код в библиотеке, вы можете многократно использовать этот код в разных проектах, а также скрывать сложные части кода от пользователя. Позже мы более подробно рассмотрим использование библиотек.

Наконец, в меню Sketch есть опция «Показать папку скетча», которая открывает папку, в которой хранится ваш скетч. Кроме того, есть опция «Добавить файл...», которая позволит вам добавить еще один исходный файл в ваш скетч. Эта функция позволяет разбивать большие скетчи на файлы меньшего размера, а затем добавлять их в основной скетч.

Следующее меню в среде IDE - это меню «Инструменты» (см. Рис. 1-23). Здесь есть возможность выбрать используемую плату и последовательный порт, как мы это делали при первой настройке Arduino. Также у нас есть функция автоформатирования, которая форматирует ваш код, чтобы он выглядел лучше.

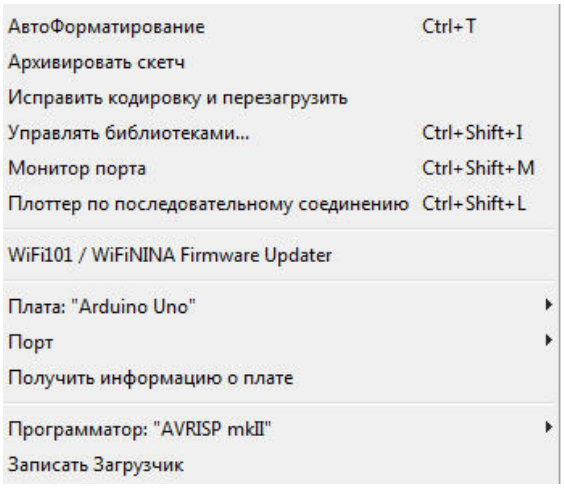


Рис. 1-23. *Меню Инструменты*

Опция «Архивировать скетч» позволит вам сжать эскиз в ZIP-файл и спросит вас, где вы хотите его сохранить. Параметр «Исправить кодировку и перезагрузить» предназначен для преобразования кода, созданного в более старых версиях среды IDE, в более новый формат.
Кнопка «программатор:» позволит вам выбрать программатор, если вы используете внешнее устройство для загрузки кода в Arduino или хотите записать код на чип в вашем собственном проекте. Мы просто будем использовать USB-кабель, который мы купили с нашим Arduino.
Наконец, параметр «Записать загрузчик» можно использовать для записи загрузчика Arduino (фрагмента кода на чипе, чтобы сделать его совместимым с IDE Arduino) на чип. Эту опцию можно использовать только в том случае, если у вас есть программатор AVR и вы заменили чип в Arduino или купили пустые чипы для использования в собственном встроенном проекте. Если вы не планируете прожигать много микросхем, обычно дешевле и проще просто купить микросхему ATmega (см. Рис. 1-24) с уже запрограммированным загрузчиком Arduino. Во многих интернет-магазинах есть предварительно запрограммированные чипы, и их можно купить довольно дешево. Чип, используемый в Arduino Uno, - это Atmel ATmega328.

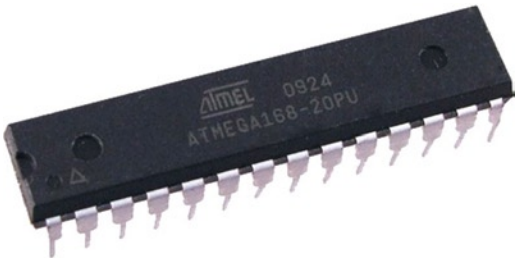


***Рис. 1-24.** Микросхема Atmel ATmega. Сердце вашего Arduino.*

Последнее меню - это меню «Справка», где вы можете найти справочные меню для получения дополнительной информации об IDE или ссылки на справочные страницы веб-сайта Arduino и другие полезные страницы. IDE Arduino довольно проста, и вы узнаете, как ее быстро и легко использовать, пока мы работаем над проектами. По мере того, как вы становитесь более опытными в использовании Arduino и программировании на C (язык программирования, который мы используем для кодирования на Arduino), вы можете обнаружить, что IDE Arduino слишком проста и захотите использовать что-то с лучшими функциями. Действительно, многие опытные программисты Arduino вообще не используют IDE, а вместо этого используют профессиональные программы IDE (некоторые из которые бесплатны), такие как Eclipse,ArduIDE, GNU / Emacs, AVR-GCC, AVR Studio и даже XCode от Apple.
Итак, теперь, когда у вас установлено программное обеспечение Arduino, плата подключена и работает, и у вас есть базовое понимание того, как использовать IDE, давайте сразу перейдем к Project 1 - LED Flasher.

Резюме

В этой главе вы узнали, что такое Arduino, немного о различных вариантах Arduino, что вы можете с ним делать и какие основные компоненты составляют плату Arduino. Затем вы узнали, как установить и настроить программное обеспечение и драйверы для Arduino, как выбрать правильный последовательный порт и загрузить тестовый скетч на свой Arduino, чтобы убедиться, что все работает правильно.
Затем мы перешли к IDE: как ее использовать и каково назначение каждой из кнопок и меню, включая окно последовательного монитора. Это основные понятия, необходимые для понимания того, как настроить программное обеспечение для работы с оборудованием Arduino. В следующей главе мы применим эти понятия на практике, используя IDE для написания нашего кода и загрузки его на нашу плату Arduino.

Глава 2

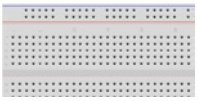
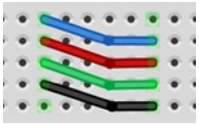
Проекты на LED

Теперь вы будете работать над первыми четырьмя проектами. Во всех этих проектах светодиодное освещение используется по-разному. Вы узнаете об управлении выходами с Arduino, а также о простых входах, таких как нажатие кнопок. Что касается аппаратного обеспечения, вы узнаете о светодиодах, кнопках и резисторах, включая подтягивающие и стягивающие резисторы, которые важны для правильного считывания устройств ввода. Попутно вы познакомитесь с понятиями программирования на языке Arduino. Давайте начнем с проекта «Hello World», который заставляет ваш Arduino мигать внешним светодиодом.

Проект 1 - Светодиодная мигалка

Для самого первого проекта мы собираемся повторить эскиз мигания светодиода, который мы использовали на этапе тестирования в главе 1. Мы подключим светодиод к одному из цифровых выводов, а не используем светодиод 13, припаянный к плате. Мы также узнаем, как именно работает аппаратное и программное обеспечение для этого проекта, по ходу дела, узнавая немного об электронике и кодировании на языке Arduino одновременно.

Таб. 2-1. *Детали, необходимые для проекта 1*

Макетная плата	
5 мм LED	
Резистор* 10 Ом	
перемычки	

** Это значение может отличаться в зависимости от того, какой светодиод вы используете. По тексту будет объяснение как это решить.*

Требуемые детали

Наилучший тип макета для большинства проектов, описанных в этой книге, - это макетная плата на 840 точек. Это довольно стандартная плата, которая имеет размеры примерно 16,5 на 5,5 см и имеют 840 точек на плате. Обычно у них есть маленькие «ласточкины хвосты» сбоку, позволяющие соединить несколько из них вместе для создания платы более крупных размеров,что нужно для больших проектов. Однако для этого проекта подойдет макетная плата любого размера.

Светодиод должен быть диаметром 5 мм любого цвета. Вам нужно будет знать ток и напряжение (иногда называемые прямым током и прямым напряжением) светодиода, поскольку это позволит вам рассчитать необходимое значение резистора. Мы определим это значение позже в проекте.

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Затем возьмите плату, светодиод, резистор и провода и подключите все, как показано на рисунке 2-1.

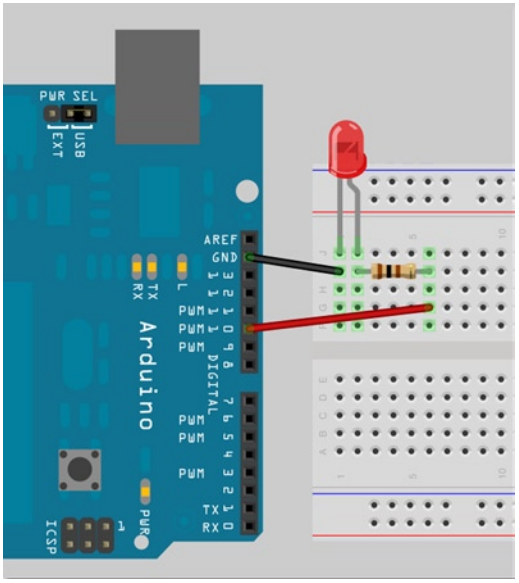


Рис. 2-1. Схема для Проекта 1 - LED мигалка

Неважно, используете ли вы провода разного цвета или разные отверстия на макетной плате, если компоненты и провода подключены в том же порядке, что и на картинке. Будьте осторожны при установке компонентов в макетную плату. Неправильная установка компонентов может привести к повреждению платы. Убедитесь, что анод светодиода (обычно более длинный вывод) подключен к резистору, а катод - на землю. Когда вы уверены, что все подключено правильно, включите Arduino, подключив USB-кабель.

Введите код

Откройте IDE Arduino и введите код из листинга 2-1 или из папки с кодами:

Листинг 2-1. Код для проекта 1

```
// Project 1 - LED мигалка
int ledPin = 10;
void setup() {
         pinMode(ledPin, OUTPUT);
}
void loop() {
        digitalWrite(ledPin, HIGH);
        delay(1000);
        digitalWrite(ledPin, LOW);
        delay(1000);
}
```

Теперь нажмите кнопку "Проверить" в верхней части IDE, чтобы убедиться, что в вашем коде нет ошибок. В случае успеха теперь вы можете нажать кнопку «Загрузка», чтобы загрузить код в свой Arduino. Если вы все сделали правильно, вы должны видеть, как светодиод на плате мигает каждую секунду.Теперь давайте взглянем на код и и выясним, как он работает.

Проект 1 - LED Flasher - Обзор кода

Давайте посмотрим на код этого проекта. Наша первая строка

```
// Project 1 - LED мигалка
```

Это просто комментарий в нашем коде, который игнорируется компилятором. Компиля́тор – это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов. Все, что следует за // (двойная косая черта) в строке, компилятор игнорирует. Это позволяет вам добавлять примечания для себя и всех остальных, которые могут читать код, объясняющий, как он работает.

Комментарии необходимы в вашем коде, чтобы помочь вам понять, что происходит и как работает ваш код. Позже, когда ваши проекты станут более сложными, а ваш код расширится до сотен или, может быть, тысяч строк, комментарии будут иметь важное значение. Вы можете придумать потрясающий фрагмент кода, но если вы вернетесь и посмотрите на этот код через несколько дней, недель или месяцев, вы можете забыть, как все это работает. Комментарии помогут вам вспомнить , позволят другим понять,что происходит в вашем коде.

Вы также можете помещать комментарии в блок, используя разделители / * и * /, например:

```
/ * Весь текст внутри косой черты и
звездочки является комментарием и
будет проигнорирован компилятором *
/
```

IDE автоматически изменит цвет любого закомментированного текста на серый. Следующая строка программы

```
int ledPin = 10;
```

Это так называемая переменная. Переменная - это место для хранения данных. Представьте себе переменную ввиде небольшого ящика, в котором вы можете хранить вещи. Переменная называется переменной, потому что вы можете изменить ее содержимое. В этом случае вы настраиваете переменную типа int или integer. Целое число - это число в диапазоне от -32 768 до 32 767. Затем вы присвоили этому целому числу имя ledPin и присвоили ему значение 10. Мы можем назвать это число как угодно. Но поскольку мы хотим, чтобы имя нашей переменной было описательным, мы называем его ledPin, чтобы показать, что эта переменная использует цифровой вывод 10 платы Arduino для подключения нашего светодиода. В конце этого оператора стоит точка с запятой. Это символ, указывающий компилятору, что этот оператор завершен.

Хотя мы можем называть наши переменные как угодно, каждое имя переменной в C должно начинаться с буквы; остальная часть имени может состоять из букв, цифр и символов подчеркивания. C распознает символы верхнего и нижнего регистра как разные. Наконец, вы не можете использовать ключевые слова C, такие как main, while, switch и т. д., в качестве имен переменных. Ключевые слова - это константы, переменные и имена функций, которые определены как часть языка Arduino. Не используйте имя переменной, которое совпадает с ключевым словом. Все ключевые слова в скетче будут выделены красным цветом. Итак, вы создали область в памяти для хранения числа целого типа и сохранили в этой области число 10. Далее у нас есть наша функция setup ():

```
void setup() {
        pinMode(ledPin, OUTPUT);
}
```

Скетч Arduino должен иметь функции setup () и loop (). Функция setup () запускается один раз и только один раз в начале программы, и именно здесь будут общие инструкции для подготовки программы перед запуском основного цикла, такие как настройка режимов вывода, установка скорости последовательной передачи данных и т. д. В основном, функция - это блок кода, собранный в один удобный блок. Например, если мы создали нашу собственную функцию для выполнения целого ряда сложных математических операций, в которых было много строк кода, мы могли бы запускать этот код столько раз, сколько захотим, просто вызывая имя функции вместо того, чтобы писать код заново раз. Позже мы рассмотрим функции более подробно, когда начнем создавать свои собственные. В случае нашей программы функция setup () должна выполнить только один оператор. Функция начинается с

```
void setup()
```

Здесь мы сообщаем компилятору, что наша функция называется setup, что она не возвращает данных (void) и что мы не передаем ей никаких параметров (пустые скобки). Если бы наша функция вернула целочисленное значение, и у нас также были бы целые значения для передачи ей (например,функции для обработки), то это выглядело бы примерно так

```
int myFunc(int x, int y)
```

В этом случае мы создали функцию (или блок кода) под названием myFunc. Этой функции были переданы два целых числа с именами x и y. После завершения функции она вернет целочисленное значение в точку после того, как наша функция была вызвана в программе. Весь код внутри функции заключен в фигурные скобки. Символ { начинает блок кода, а символ } закрывает блок. Все, что находится между этими двумя символами, - это код, принадлежащий функции. Мы более подробно рассмотрим функции в Проекте 4 в этой главе. Все, что вам нужно знать, что в этой программе у нас есть две функции, и первая функция называется setup; его цель - настроить все необходимое для работы нашей программы до запуска основного цикла программы.

```
void setup() {
        pinMode(ledPin, OUTPUT);
}
```

Наша функция настройки имеет только один оператор - pinMode. Здесь мы говорим Arduino, что хотим установить режим одного из наших цифровых выводов как режим вывода, а не ввода. В скобках указываем номер пина и режим (OUTPUT - ВЫХОД). Номер нашего пина - ledPin, которому в нашей программе ранее было присвоено значение 10. Следовательно, этот оператор просто сообщает Arduino, что цифровой вывод 10 должен быть установлен в режим OUTPUT (ВЫХОД). Поскольку функция setup () выполняется только одинраз, мы переходим к основному циклу функции.

```
void loop() {
        digitalWrite(ledPin, HIGH);
        delay(1000);
        digitalWrite(ledPin, LOW);
        delay(1000);
}
```

Функция loop () является основной функцией программы и вызывается постоянно, пока включен наш Arduino. Каждый оператор в функции loop () (в фигурных скобках) выполняется один за другим, шаг за шагом, пока не будет достигнута нижняя часть функции; затем Arduino снова запускает цикл в верхней части функции и так до бесконечности, или до тех пор, пока вы не выключите Arduino или не нажмете кнопку сброса.

В этом проекте мы хотим, чтобы светодиод включился, оставался включенным в течение одной секунды, выключился и оставался выключенным в течение одной секунды, а затем процесс бы повторился. Следовательно, команды, указывающие Arduino на выполнение этого действия, содержатся в функции loop (), поскольку мы хотим, чтобы они повторялись снова и снова. Первое утверждение

```
digitalWrite(ledPin, HIGH);  // HIGH - высокий уровень
```

... записывает значение HIGH или LOW на цифровой вывод в операторе (в данном случае ledPin, который является цифровым выводом 10). Когда вы устанавливаете цифровой вывод на HIGH, вы отправляете на него 5 вольт. Когда вы устанавливаете его на LOW, вывод становится 0 вольт или землей. Таким образом, этот оператор отправляет 5 В на цифровой вывод 10 и включает светодиод. После этого оператор

```
delay(1000);
```

... просто говорит Arduino подождать 1000 миллисекунд (в секунде 1000 миллисекунд) перед выполнением следующего оператора, который

```
digitalWrite(ledPin, LOW);
```

... отключит питание, поступающее на цифровой контакт 10, и, следовательно, погасит светодиод. Затем идет еще один оператор задержки еще на 1000 миллисекунд, а затем функция завершается. Однако, поскольку это наша основная функция loop (), функция снова запустится с самого начала. Еще раз проследив пошагово структуру программы, мы увидим, что она очень проста:

```
// Project 1 - LED Flasher
int ledPin = 10;
void setup() {
        pinMode(ledPin, OUTPUT);
}
void loop() {
        digitalWrite(ledPin, HIGH);
        delay(1000);
        digitalWrite(ledPin, LOW);
        delay(1000);
}
```

Мы начинаем с присвоения переменной с именем ledPin значения 10. Затем мы переходим к функции setup (), где просто устанавливаем режим для цифрового вывода 10 в качестве выхода. В основном программном цикле выставляем на цифровой выводе 10 HIGH,т.е 5В. Затем мы ждем секунду, а затем отключаем 5 В на выводе 10 с помощью LOW , прежде чем ждать еще секунду.

Затем цикл начинается снова с самого начала, и светодиод будет постоянно включаться и выключаться, пока на Arduino есть питание.

Теперь, когда вы это знаете, вы можете изменить код, чтобы включить или выключить светодиод на другой периоды времени. Например, если мы хотим, чтобы светодиод оставался включенным в течение двух секунд, а затем погас на полсекунды, мы могли бы сделатьтак:

```
void loop() {
        digitalWrite(ledPin,
        HIGH);delay(2000);
        digitalWrite(ledPin, LOW);
        delay(500);
}
```

Возможно, вы хотите, чтобы светодиодный индикатор оставался выключенным в течение пяти секунд, а затем кратковременно горел (250 мс), как светодиодный индикатор в автосигнализации; тогда вы могли бы написать следующее :

```
void loop() {
        digitalWrite(ledPin, HIGH);
        delay(250);
        digitalWrite(ledPin, LOW);
        delay(5000);
}
```

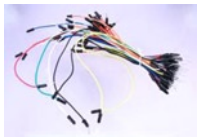
Или сделать так, чтобы светодиодный индикатор мигал очень быстро.

```
void loop() {
        digitalWrite(ledPin, HIGH);
        delay(50);
        digitalWrite(ledPin, LOW);
        delay(50);
}
```

Изменяя время включения и выключения светодиода, вы можете создать любой желаемый эффект. Прежде чем перейти к чему-то более интересному, давайте взглянем на наше устройство и посмотрим, как оно работает.

Проект 1 — LED мигалка – Обзор схемы

Компоненты для нашего проекта 1

Макетная плата	
5mm LED	
Резистор 100 Ом*	
Перемычки	

*от (100 до 270) Ом

Макетная плата представляет собой беспаечное устройство многоразового пользования, которое обычно используется для создания прототипов электронной схемы или для экспериментов со схемами. Плата состоит из ряда отверстий в виде сетки. Эти отверстия под платой соединены полосами проводящего металла с зажимами. Эти полосы обычно расположены так, как показано на рис. 2-2.

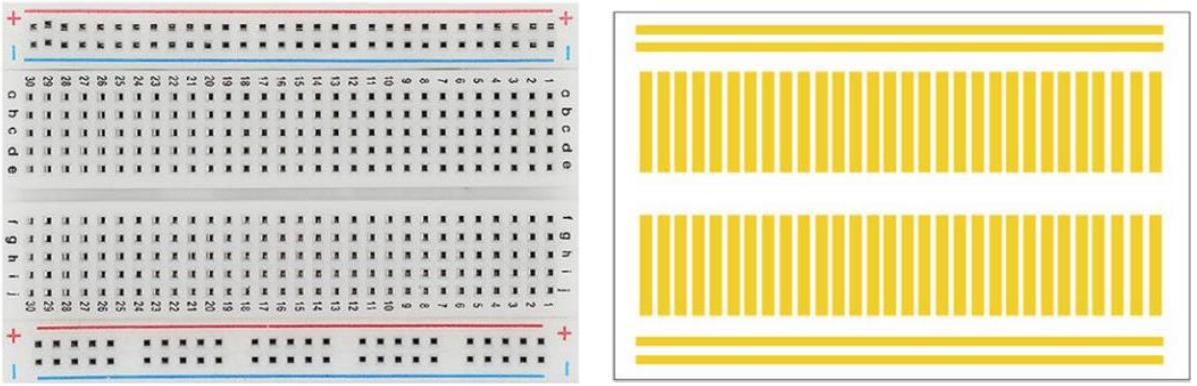


Рис. 2-2. *Так раскладываются металлические полосы в макетной плате*

Цветные линии вдоль верхней и нижней части проходят параллельно плате и предназначены для подключения к плюсу (красная) и минусу (синяя) вашего источника питания.

Посередине макетной платы есть зазор, позволяющий разместить микросхему (см. Рисунок 2-3).

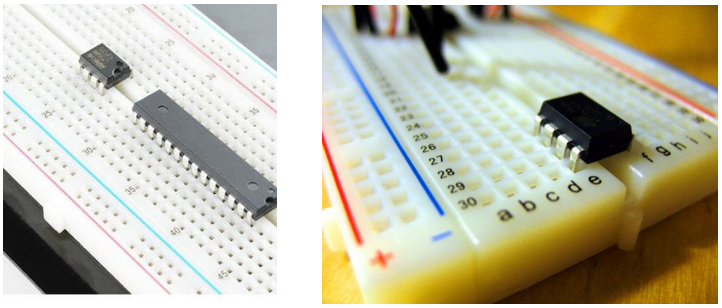


Рис. 2-3. *Микросхема,установленная на макетной плате*

Следующий компонент - резистор. Резистор - это компонент, которое вызывает сопротивление электрическому току и, вызывает падение напряжения на его выводах. Вы можете представить резистор как водопроводную трубу, которая намного тоньше, чем труба, подключенная к нему.
По мере того, как вода (электрический ток) попадает в резистор, труба становится тоньше, и поэтому ток, выходящий с другого конца, уменьшается. Мы используем резисторы для уменьшения напряжения или тока в электрической цепи.
Сопротивление измеряется в единицах, называемых Ом. Символом для обозначения Ом является греческий символ омеги . В этом случае цифровой вывод 10 выдает 5 вольт постоянного тока при 40 мА (миллиампер), в соответствии с таблицей данных Atmega, а для наших светодиодов требуется напряжение 2 В и ток 35 мА, соответственно даташиту. Поэтому нам нужно использовать резистор, который снизит 5 В до 2 В и ток с 40 мА до 35 мА, если мы хотим отображать светодиод с максимальной яркостью.

■ **Примечание** Никогда не используйте резистор НИЖЕ рассчитанного значения. Вы допустите слишком большой ток через светодиод и навсегда его повредите. Вы также можете повредить другие части вашей схемы.

Формула для определения сопротивления резистора:

$$R = (V_S - V_L) / I$$

Где V_S - напряжение питания, V_L - напряжение свечения светодиода, а I - ток светодиода. Итак, для нашего примера у нас есть светодиод с напряжением 2 вольта и током 35 мА (миллиампер), подключенный к цифровому выводу от Arduino, который выдает 5 вольт; следовательно, необходимое сопротивление резистора будет:

$$R = (5 - 2) / 0.035$$

что дает значение 85,71 Ом.

Резисторы бывают стандартных номиналов, и наиболее близкое номинальное значение составляет 100 Ом. Всегда выбирайте следующий резистор стандартного номинала, который больше необходимого. При более низком значении, через резистор будет протекать слишком большой ток, и резистор и / или связанные с ним компоненты могут быть повреждены. Итак, как нам найти резистор 100 Ом? Резистор слишком мал, чтобы на нем можно было писать номинальное значение сопротивления, поэтому используется цветовой код. На резисторе обычно находятся четыре цветные полосы, и с помощью цветового кода в Таблице 2-2 вы можете узнать номинал резистора или какой цветовой код будет иметь конкретное сопротивление.

Таб. 2-2. *Цветовые коды резисторов*

Цвет	1 диапазон	2 диапазон	3 диапозон множитель	4 диапозон допуск
черный	0	0	$x10^0=1$	
коричневый	1	1	$x10^1$	±1%
красный	2	2	$x10^2$	±2%
оранжевый	3	3	$x10^3$	
желтый	4	4	$x10^4$	
зеленый	5	5	$x10^5$	±0.5%
синий	6	6	$x10^6$	±0.25%
фиолетовый	7	7	$x10^7$	±0.1%
серый	8	8	$x10^8$	±0.05%
белый	9	9	$x10^9$	
Золотой			$x10^{-1}$	±5%
серебряный			$x10^{-2}$	±10%
пустой				±20%

Нам нужен резистор 100 Ом, поэтому, если вы посмотрите таблицу цветовой кодировки, вы увидите, что нам нужна 1 в первой полосе, которая является коричневой, за которой следует 0 в следующей полосе, которая является черной, и затем нам нужно умножить это на 10^1 (другими словами, добавить 1 ноль), что является коричневым в 3-й полосе. Последняя полоса указывает допуск резистора. Если у вашего резистора золотая полоса, он имеет допуск ± 5%. Следовательно, если у вас есть светодиод, который требует два вольта и 35 мА, вам понадобится резистор с коричневой, черной и коричневой полосой.

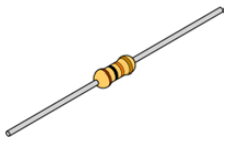


Рис. 2-4. *Резистор 10 кОт с допуском 5%*

Если вам нужен резистор 10 кОм (или 10 000 Ом) (см. Рисунок 2-4), вам понадобится комбинация коричневого, черного и оранжевого цветов ($1, 0, 10^3$). 10^3 - добавить три нуля. Если вам нужен резистор 570 кОм (570 000 Ом), цвета будут зелеными, фиолетовыми, желтыми - 4 нуля) и так далее. Итак, если вы нашли этот резистор и захотели узнать, какое у него значение, чтобы вы могли хранить его в хорошо помеченном ящике для хранения резисторов,вы можете посмотреть на таблицу.

Последний компонент - это светодиод -LED, который расшифровывается как Light Emitting Diode - диод, испускающий свет ...при пропускании через него электрического тока. Условное обозначение, цоколевка представлены на рис.2.5.
Диод - это устройство, которое пропускает ток в одном направлении,так что LED также обладает этим свойством.

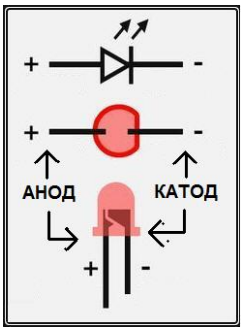
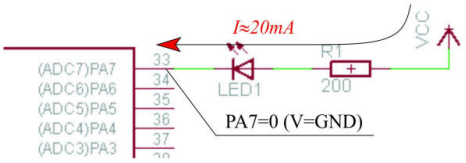


Рис. 2.5

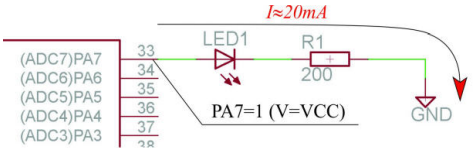
При подключении светодиода к МК возможны два следующих случая:
1) светодиод подключен катодом к микроконтроллеру (вариант 1);
2) светодиод подключен анодом к микроконтроллеру (вариант 2).

рассмотрим вариант 1



Для того чтобы светодиод начал излучать свет, необходимо создать разность потенциалов между анодом и катодом. Анод подключен к положительному выводу источника питания (VCC). Т.о. когда вывод PA7 сбрасывается в «0», на нем образуется нулевой потенциал (GND) и через светодиод начинает протекать ток. Если вывод PA7 = 1 (VCC), то разности потенциалов не будет – и светодиод не будет излучать свет.

Рассмотрим вариант 2:



Для того чтобы светодиод начал излучать свет, необходимо создать разность потенциалов между анодом и катодом. Катод подключен к отрицательному выводу источника питания (GND). При сравнении с предыдущей схемой становится ясно, что когда разряд PA7 сбрасывается в «0» – светодиод не будет излучать свет, а когда PA7 = 1 - будет.

Теперь, когда вы знаете, как работают компоненты и как работает код в этом проекте, давайте попробуем кое-что поинтереснее.

Project 2 – S.O.S. Сообщение азбукой Морзе

Для этого проекта мы собираемся оставить ту же схему, что и в проекте 1, но будем использовать другой код, чтобы светодиод отображал сообщение азбукой Морзе. В этом случае мы собираемся заставить светодиод сигнализировать S.O.S., который является международным сигналом бедствия кодом Морзе. Код Морзе - это тип кодировки символов, который передает буквы и цифры с использованием шаблонов включения и выключения. Поэтому он хорошо подходит для нашей цифровой системы, поскольку мы можем включать и выключать светодиод в нужном порядке, чтобы разобрать слово или серию символов. В этом случае мы будем сигнализировать S.O.S., который в алфавите кода Морзе состоит из трех точек (короткие вспышки), за которыми следуют три тире (длинные вспышки), а затем снова три точки.

Таким образом, теперь мы можем закодировать наш скетч так, чтобы светодиод включался и выключался в этой примере, сигнализируя SOS.

Ввод кода

Листинг 2-2. Код для проекта 2

```
// подключение LED к цифровому пину 10
int ledPin = 10;

// запускаем один раз при запуске скетча
void setup()
{
        // устанавливаем цифровой вывод как выход
        pinMode(ledPin, OUTPUT);
}

// бегаем снова и снова
void loop()
{
       // 3 точки
       for (int x=0; x<3; x++) {
              digitalWrite(ledPin, HIGH);  // включаем светодиод
              delay(150);                  // ждем 150 мс
              digitalWrite(ledPin, LOW);   // выключает светодиод
              delay(100);                  // ждем 100 мс
        }

  // задержка 100 мс для паузы между буквами
       delay(100);
  // 3 тире
        for (int x=0; x<3; x++) {
              digitalWrite(ledPin, HIGH);  // включаем светодиод
              delay(400);                  // ждем 400 мс
              digitalWrite(ledPin, LOW);   // выключаем светодиод
              delay(100);                  // ждем 100 мс
        }
```

```
  // задержка 100 мс для паузы между буквами
  delay(100);

  // снова 3 точки
        for (int x=0; x<3; x++) {
              digitalWrite(ledPin, HIGH);   // включаем светодиод
              delay(150);                   // ждем 150 мс
              digitalWrite(ledPin, LOW);    // выключаем светодиод
              delay(100);                   // ждем 100 мс
        }

        // ждем 5 секунд перед повторением сигнала SOS
        delay(5000);
}
```

Создайте новый скетч в IDE Arduino и введите код из листинга 2-2. Убедитесь, что ваш код не содержит ошибок, а затем загрузите его в Arduino. Если все пойдет хорошо, вы увидите, как светодиодный индикатор мигает сигналом SOS кодом Морзе, пауза 5 секунд, затем повтор.

Итак, давайте взглянем на этот код и разберемся, как он работает.

Проект 2 - S.O.S. Сигнализатор азбуки Морзе - Обзор кода

Таким образом, первая часть кода идентична последнему проекту, в котором мы инициализируем переменную, а затем устанавливаем вывод 10 как выход. В основном цикле кода мы можем видеть такие же операторы для включения и выключения светодиодов на заданный период времени, но на этот раз операторы находятся в трех отдельных блоках кода.

Первый блок выводит три точки:

```
for (int x=0; x<3; x++) {
        digitalWrite(ledPin, HIGH);
        delay(150);
        digitalWrite(ledPin, LOW);
        delay(100);
}
```

Мы видим, что светодиод включается на 150 мс, а затем выключается на 100 мс, и мы видим, что эти операторы заключены в набор фигурных скобок и, следовательно, находятся в отдельном блоке кода. Блок - это набор из одного или нескольких операторов, заключенных в фигурные скобки. Но когда мы запускаем скетч, мы видим, что свет мигает три раза, а не один раз.

Это делается с помощью цикла for.

```
for (int x=0; x<3; x++) {
```

Этот оператор заставляет код в его блоке выполняться три раза. Есть три выражения, которые мы можем передать в цикл for. Это инициализация, условие, приращение. Выражение инициализации вычисляется первым и только один раз. Каждый раз при прохождении цикла проверяется условие; если это правда, выполняется блок операторов, а затем выполняется выражение приращения (если оно есть).
Затем управление возвращается к проверке условия, и процесс повторяется до тех пор, пока условие не станет ложным при проверке; затем цикл заканчивается.Итак, сначала нам нужно инициализировать переменную, чтобы она была начальным номером цикла. В этом случае мы устанавливаем переменную x и устанавливаем ее равной нулю.

```
int x=0;
```

Затем мы устанавливаем условие, определяющее, сколько раз будет выполняться код в цикле.

```
x<3;
```

В этом случае код будет зацикливаться, если x меньше (<) 3.Выражение инициализации всегда выполняется, но если при переходе в цикл for условие ложно, тело цикла и выражение приращения не выполняются. Символ <- это так называемый оператор сравнения. Эти операторы используются для принятия решений в нашем коде и для сравнения двух значений. Используемые символы:

== (равно)

!= (не равно)

< (меньше чем)

> (больше чем)

<= меньше или равно)

>= ((больше или равно)

В нашем коде мы сравниваем x со значением 3, чтобы увидеть, меньше ли оно 3. Если x меньше 3, то код в блоке будет выполняться. В противном случае цикл завершится.

Оператор условия:

```
x++
```

Это инструкция для увеличения значения x на 1. Мы также могли бы ввести x = x + 1, что присвоило бы x значение x + 1. Обратите внимание, что нет необходимости ставить точку с запятой после оператора x++ .

Итак, наш цикл for инициализирует значение x равным 0, затем проверяет, выполняется ли условие, то есть x меньше 3; если это так, он запускает код в теле цикла for. Затем он увеличивает x на значение выражения приращения.

Пока выполняется условие цикла, цикл будет повторяться.

Итак, теперь мы знаем, как работает цикл for, мы можем видеть в нашем коде, что есть три цикла for, один изкоторых повторяется три раза и отображает точки. Следующий повторяется три раза и отображает тире. Потом снова повторение точек.

Следует отметить, что переменная x имеет локальную «область видимости», что означает, что она может быть видна только коду внутри своего собственного блока . Переменная, определенная в функции for () или в блоке, доступна в этой функции for () или в блоке. Переменные, определенные вне какой-либо функции, глобально видимы в оставшейся части файла, в котором они определены. Они не видны отдельно скомпилированным частям программы, если они не объявлены там с помощью команды extern. Если вы попытаетесь получить доступ к x вне цикла for, вы получите сообщение об ошибке. Между каждым циклом for есть небольшая задержка, чтобы сделать паузу между буквами S.O.S. Наконец, код ждет пять секунд, прежде чем снова запустится основной цикл программы.

Теперь перейдем к использованию нескольких светодиодов.

Проект 3 - Светофоры

Теперь мы создадим набор светофоров на основе системы, которая изменит цвет с зеленого на красный, на желтый и обратно через заданный промежуток времени с использованием системы с четырьмя состояниями. Мы будем использовать детали, перечисленные в таблице 2-3. Этот проект можно использовать на модельной железной дороге для изготовления набора рабочих светофоров или для детского игрушечного городка.

Таб. 2-3. *Детали, необходимые для проекта 3*

Макетная плата	
Красный LED	
Желтый LED	
Зеленый LED3 x	
Резистор* 3 x 150 Ом	
Перемычки	

**или любое другое значение, которое требуется для вашего светодиода*

Требуемые детали

В списке частей Таблицы 2-3 вы найдете все, что вам нужно для этого проекта. Мы будем использовать три светодиода и три токоограничивающих резистора.

Подключение

Подключите вашу схему, как показано на рисунке 2-6. На этот раз мы соединили аноды каждого светодиода с цифровыми выводами 8, 9 и 10 платы Arduino, через токоограничивающие резисторы 150 Ом .

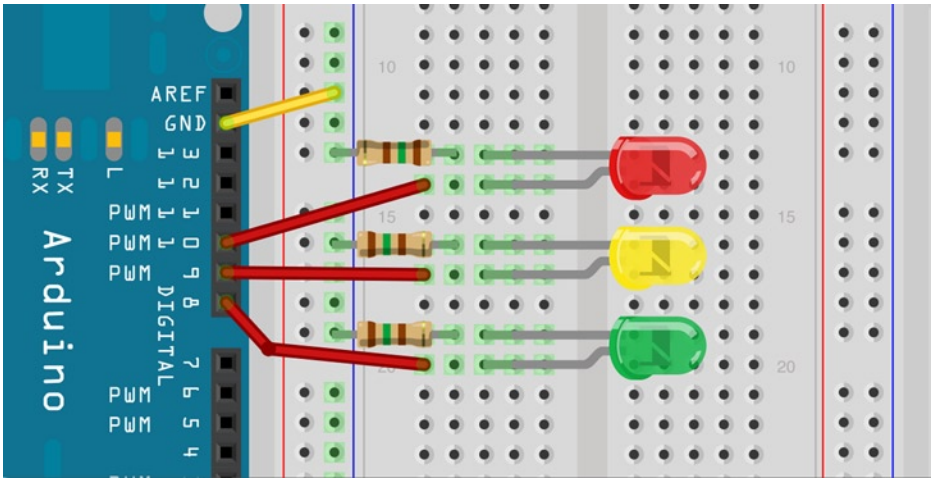


Рис. 2-6. *Подключение для третьего проекта*

Для этой простой схемы не имеет значения, подключен ли резистор между цифровым выводом и анодом или между катодом и землей, если он включен последовательно со светодиодом.

Ввод кода

Введите код из листинга 2-3, проверьте его и загрузите в свой Arduino. Теперь светодиоды будут переходить в четыре состояния, имитирующие схему светофора, как показано на рис. 2-7. Если вы следовали проектам 1 и 2, то и код, и схема для проекта 3 будут очевидны. Я думаю,что вы разберетесь с работой кода и поэтому только приведем его в листинге 2-3.

Листинг 2-3. Код для проекта 3

```
// Project 3 - Traffic Lights

int ledDelay = 10000; // задержка
int redPin = 10;  //красный LED к 10 выводу
int yellowPin = 9;  //желтый LED к 9 выводу
int greenPin = 8;  //зеленый LED к 8 выводу

void setup() {
       pinMode(redPin, OUTPUT);
       pinMode(yellowPin, OUTPUT);
       pinMode(greenPin, OUTPUT);
}

void loop() {

       digitalWrite(redPin, HIGH); //включаем красный свет
       delay(ledDelay); // ждем 5 секунд
```

```
        digitalWrite(yellowPin, HIGH); // включаем желтый
        delay(2000); // ждем 2 секунды

        digitalWrite(greenPin, HIGH); // включаем зеленый
        digitalWrite(redPin, LOW); // выключаем красный
        digitalWrite(yellowPin, LOW); // выключаем желтый
        delay(ledDelay); // ждем ledDelay миллисекунд

            digitalWrite(yellowPin, HIGH); // включается желтый
            digitalWrite(greenPin, LOW); // выключаем зеленый
            (2000); // wait 2 seconds

        digitalWrite(yellowPin, LOW); // ждем 2 секунды
        //  наш цикл повторяется

}
```

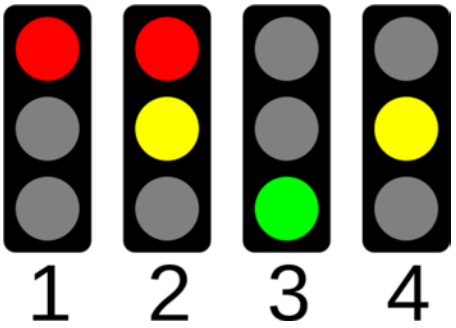


Рис. 2-7. *Рисунок 2-7. Четыре состояния светофора*

В следующем проекте мы построим проект 3, включив набор пешеходных огней и добавив кнопку, чтобы сделать свет интерактивным. При этом мы изучим изменения состояния входа (INPUT).

Проект 4 - Интерактивные светофоры

На этот раз мы собираемся расширить предыдущий проект, включив в него набор пешеходных огней и кнопку для пешеходов, используемую для запросов на переход дороги. Arduino будет реагировать на нажатие кнопки, изменяя состояние огней, чтобы машины останавливались и позволяли пешеходу безопасно перейти дорогу.Впервые мы сможем взаимодействовать с Arduino и заставлять его что-то делать, когда мы изменяем состояние кнопки, за которой наблюдает Arduino (т. е. нажимаем кнопку, чтобы изменить состояние с открытого на закрытое). В этом проекте мы также узнаем, как создавать собственные функции в коде.
С этого момента при перечислении необходимых деталей мы больше не будем перечислять макетную плату и перемычки. Просто предположите, что они вам всегда понадобятся.

Требуемые детали

Список деталей в Таблице 2-4 почти идентичен в Проекте 3. Однако мы добавили два дополнительных светодиода и их соответствующие токоограничивающие резисторы, чтобы представить огни пешеходного перехода. Мы также включили кнопку, чтобы вы могли управлять светом,а также узнаем, как считывать состояние входного вывода.

Table 2-4. *Требуемые детали для проекта 4*

2 x красных LED	
Желтый LED	
2 x зеленых LED	
Резистор 10 кОм	
5 x 5 токоограничивающих резисторов	
Кнопка	

Пять резисторов для светодиодов предназначены для ограничения тока , идущего к светодиоду. Выберите резистор соответствующего номинала для вашего проекта (см. Главу 1). Для кнопки вам понадобится резистор 10 кОм. Это так называемый «стягивающий резистор». Мы рассмотрим подтягивающие и стягивающие резисторы позже в этой главе.

Подключение

Подключите вашу схему, как показано на Рисунке 2-8. Дважды проверьте проводку, прежде чем подавать питание на Arduino. Не забудьте отключить Arduino от источника питания при подключении схемы.

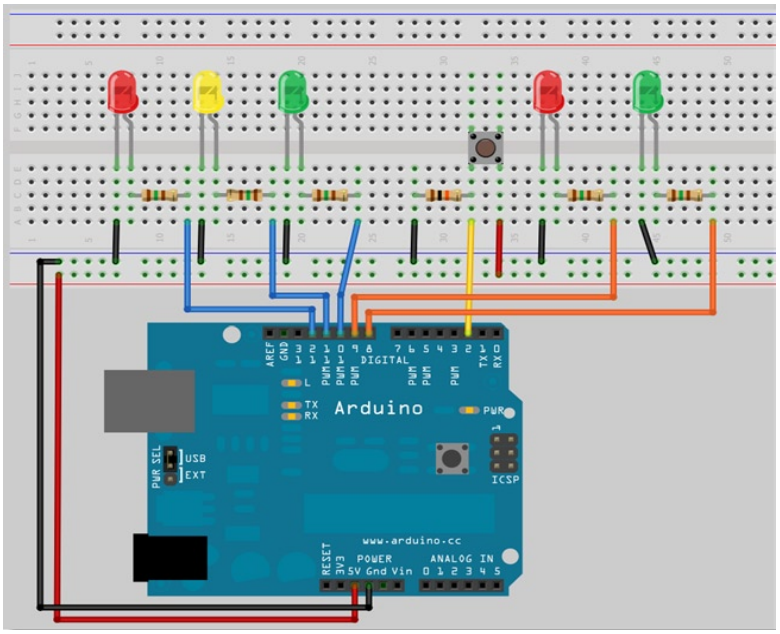


Рис. 2-8. *Система светофоров с пешеходным переходом и кнопкой запроса*

Ввод кода

Введите код из листинга 2-4, проверьте и загрузите его.

Когда вы запустите программу, вы увидите, что автомобильный светофор загорится зеленым, чтобы позволить машинам проехать, а пешеходный свет загорится красным.

Когда вы нажимаете кнопку, программа проверяет, прошло ли не менее пяти секунд с момента последней замены света (чтобы разрешить движение транспорта), и, если они есть, передает выполнение кода созданной нами функции с именем changeLights. (). В этой функции фары автомобиля меняют цвет с зеленого на желтый, затем на красный, а затем на зеленый цвет светофора.

По прошествии периода времени, установленного в переменной crossTime (время, достаточное для того, чтобы пешеходы могли перейти дорогу), зеленый свет будет мигать, предупреждая пешеходов о том, что им нужно поторопиться, поскольку огонь вот-вот снова станет красными. Затем цвет пешеходного света снова меняется на красный, а огни транспортного средства меняют цвет с красного на желтый, затем на зеленый, и движение может возобновиться.

Листинг 2-4. Код для проекта 4

```
// Project 4 - Интерактивные светофоры

int carRed = 12; // подключаем красный LED
int carYellow = 11; // .... желтый
int carGreen = 10; //  ... зеленый
int pedRed = 9; // назначаем пешеходные огни
int pedGreen = 8;
int button = 2; // пин кнопки
int crossTime = 5000; // время, разрешенное для перехода
unsigned long changeTime = 0; // время завершения перехода
```

```
void setup() {
       pinMode(carRed, OUTPUT);
       pinMode(carYellow, OUTPUT);
       pinMode(carGreen, OUTPUT);
       pinMode(pedRed, OUTPUT);
       pinMode(pedGreen, OUTPUT);
       pinMode(button, INPUT); // кнопка на выводе 2
       // включаем зеленый свет
       digitalWrite(carGreen, HIGH);
       digitalWrite(pedRed, HIGH);
}

void loop() {
       int state = digitalRead(button);
       /* проверяем, нажата ли кнопка и прошло ли более 5 секунд с момента последнего нажатия кнопки*/
       if (state == HIGH && (millis() - changeTime) > 5000) {
              // Вызов функции для изменения освещения
              changeLights();
       }
}

void changeLights() {
       digitalWrite(carGreen, LOW); // зеленый выключен
       digitalWrite(carYellow, HIGH); // желтый горит
       delay(2000); // ждем 2 секунды

       digitalWrite(carYellow, LOW); // желтый не горит
       digitalWrite(carRed, HIGH); // красный горит
       delay(1000); // ждем 1 секунду

       digitalWrite(pedRed, LOW); // выкл. кнопкой красный
       digitalWrite(pedGreen, HIGH); // вкл. кнопкой зеленый
       delay(crossTime); // ждем заданного периода времени

       // мигаем педалью зеленым
       for (int x=0; x<10; x++) {
              digitalWrite(pedGreen, HIGH);
              delay(250);
              digitalWrite(pedGreen, LOW);
              delay(250);
       }
       // включаем кнопкой красный
       digitalWrite(pedRed, HIGH);
       delay(500);

       digitalWrite(carYellow, HIGH); // желтый горит
       digitalWrite(carRed, LOW); // красный выключен
       delay(1000);
       digitalWrite(carGreen, HIGH);
       digitalWrite(carYellow, LOW); // желтый выкл.
```

```
        // записываем время с момента последней смены огней
        changeTime = millis();
         // затем возвращаемся в основной цикл программы
}
```

Код в этом проекте аналогичен предыдущему проекту. Однако было введено несколько новых утверждений и понятий, поэтому давайте рассмотрим их.

Проект 4 - Интерактивные светофоры - Обзор кода

Большую часть кода в этом проекте взята из предыдущих проектов. Однако давайте взглянем на несколько новых ключевых слов и концепций, которые были представлены в этом скетче.

```
unsigned long changeTime = 0;
```

Здесь у нас есть новый тип данных для переменной. Ранее мы создали целочисленные типы данных, которыемогут хранить числа в диапазоне от -32 768 до 32 767. На этот раз мы создали тип данных long, который может хранить число от -2147 483 648 до 2 147 483 647. Однако мы указали беззнаковое long, что означает, что переменная не может хранить отрицательные числа, что дает нам диапазон от 0 до 4 294 967 295. Если бы мы использовали целое число для хранения времени с момента последней смены источников света, мы бы получили максимальноевремя в 32 секунды, прежде чем целочисленная переменная достигнет числа, большего, чем она могла бы сохранить.

Поскольку пешеходный переход вряд ли будет использоваться каждые 32 секунды, мы не хотим, чтобы наша программа работала неправильно из-за «переполнения» нашей переменной, когда она пытается сохранить число, слишком большое для типа данных переменной. Вот почему мы используем беззнаковый длинный тип данных, так как теперь мы получаем огромное время между нажатиями кнопок.

4,294,967,295 * 1мс = 4,294,967 секунд

4,294,967 секунд = 71,582 минут

71,582 минут = 1,193 часов

1,193 часов = 49 дней

Поскольку кнопка на пешеходном переходе неизбежно будет нажата хотя бы раз в 49 дней, у нас не должно возникнуть проблем с этим типом данных.Вы вполне можете спросить, почему у нас нет только одного типа данных, который может хранить огромные числа все время, и покончить с этим. Причина заключается в том, что переменные занимают место в памяти, и чем больше число, тем больше памяти используется для хранения переменных. На вашем домашнем ПК или ноутбуке вам вообще не придется беспокоиться об этом, но на небольшом микроконтроллере, таком как Atmega328, который использует Arduino, важно, чтобы мы использовали только наименьший тип переменных данных, необходимый для нашей цели.

Существуют различные типы данных, которые мы можем использовать в качестве скетчей, а именно:

Таб. 2-5. *Типы данных*

Тип данных	Размер	Диапазон номеров
пустое ключевое слово	N/A	N/A
boolean	1 byte	0 to 1 (Истина или ложь)
байт	1 byte	0 to 255
символ	1 byte	–128 to 127
беззнаковый символ	1 byte	0 to 255

(*продолжение*)

Table 2-5. (*продолжение*)

Тип данных	Размер	Диапазон номеров
int	2 byte	–32,768 to 32,767
unsigned int	2 byte	0 to 65,535
word	2 byte	0 to 65,535
long	4 byte	–2,147,483,648 to 2,147,483,647
unsigned long	4 byte	0 to 4,294,967,295
float	4 byte	–3.4028235E+38 to 3.4028235E+38
double	4 byte	–3.4028235E+38 to 3.4028235E+38
string	1 byte + x	Массивы символов
array	1 byte + x	Набор переменных

Каждый тип данных использует определенный объем памяти на Arduino, как вы можете видеть на диаграммевыше. Некоторые переменные используют только один байт памяти, а другие - четыре или более. Вы не можете копировать данные из одного типа данных в другой; например, если x был int, а y был string, то x = y не будет работать, поскольку эти два типа данных различны.
Atmega168 имеет 1 КБ (1024 байта), а Atmega328 имеет 2 КБ (2048 байтов) SRAM. Это не так уж и много, и в больших программах с большим количеством переменных вам может легко не хватить памяти, если вы неоптимизируете использование соответствующих типов данных. Из приведенного выше списка мыможем ясно видеть, что использование нами типа данных int расточительно, поскольку он использует двабайта и может хранить число до 32 767. Поскольку мы использовали int для хранения номера нашего цифрового вывода, который может достигать 13 на нашей Arduino (и до 54 на Arduino Mega), мы использовали больше памяти, чем было необходимо. Мы могли бы сэкономить память, используя байтовый тип данных, который может хранить число от 0 до 255, что более чем достаточно для хранениянномера вывода ввода / вывода.

Далее у нас есть

```
pinMode(button, INPUT);
```

... что сообщает Arduino, что мы хотим использовать цифровой вывод 2 (кнопка = 2), как INPUT. Мы собираемся использовать вывод 2 для проверки нажатий кнопок, поэтому его режим должен быть установлен на ввод. В основном цикле программы мы проверяем состояние цифрового вывода 2 с помощью этого оператора:

```
int state = digitalRead(button);
```

Он инициализирует целое число, называемое «state - состоянием» (да, это расточительно, и мы должны использовать логическое значение), а затем устанавливает значение состояния как значение цифрового вывода 2. Оператор digitalRead считывает состояние цифрового вывода и возвращает его к целому числу, которому мы его присвоили. Затем мы можем проверить значение состояния, чтобы увидеть, была ли нажата кнопка или нет.

```
if (state == HIGH && (millis() - changeTime) > 5000) {
    // Вызов функции для изменения освещения
    changeLights();
  }
```

Оператор if является примером структуры управления, и его цель - проверить, выполнено ли определенное условие или нет, и если да, то выполнить код в своем блоке. Например, если мы хотим включить светодиод, если переменная с именем x поднялась выше значения 500, мы могли бы написать

```
if (x>500) {digitalWrite(ledPin, HIGH);
```

Когда мы считываем цифровой вывод с помощью команды digitalRead, состояние вывода будет либо HIGH, либо LOW. Итак, команда if в нашем скетче выглядит так:

```
if (state == HIGH && (millis() - changeTime) > 5000)
```

Здесь мы проверяем выполнение двух условий. Во-первых, переменная, называемая состоянием, имеет высокий уровень. Если кнопка была нажата, состояние будет высоким, поскольку мы уже установили его как значение, считываемое с цифрового вывода 2. Мы также проверяем, что значение millis () - changeTime больше 5000 (с помощью оператора логического И &&). Функция millis () встроена в язык Arduino и возвращает количество миллисекунд, прошедших с момента запуска Arduino текущей программы. Наша переменная changeTime изначально не будет иметь значения, но после запуска функции changeLights () вы установите для нее в конце этой функции текущее значение millis (). Вычитая значение переменной changeTime из текущего значения millis (), вы можете проверить, прошло ли пять секунд с момента последней установки changeTime. Вычисление millis () - changeTime помещается в отдельный набор круглых скобок, чтобы гарантировать, что вы сравниваете значение состояния и что это результат этого вычисления, а не значение millis () отдельно.

Символ && между

```
state == HIGH
```

и вычислением является примером логического оператора. В данном случае это означает И. Чтобы понять, что мы имеем в виду, давайте взглянем на все логические операторы.

&&	Логическое И
||	Логическое ИЛИ
!	НЕТ

Это логические операторы, которые можно использовать для проверки различных условий в операторах if. && означает истину, если оба операнда верны, например:

```
if (x==5 && y==10)  {....
```

Этот оператор if запустит свой код, только если x равно 5, а также если y равно 10. || означает истину, если любой из операндов истинен, например:

```
if (x==5 || y==10) {.....
```

Это будет выполняться, если x равно 5 или если y равно 10.! Оператор or NOT (или нет) означает истину, если операнд ложен, например:

```
if (!x) {.......
```

Будет работать, если x ложно, т.е. равно нулю.
Вы также можете «вложить» условия в круглые скобки, например

```
if (x==5 && (y==10 || z==25)) {.......
```

В этом случае условия в скобках обрабатываются отдельно и обрабатываются как одно условие, а затем сравниваются со вторым условием. Итак, если мы нарисуем простую таблицу истинности (Таблица 2-6) для этого утверждения, мы сможем увидеть, как это работает,где TRUE - ИСТИНА; FALSE - ЛОЖЬ.

Таб. 2-6. *Таблица истинности для условия (x==5 && (y==10 || z==25))*

x	y	z	True/False?
4	9	25	FALSE
5	10	24	TRUE
7	10	25	FALSE
5	10	25	TRUE

Утверждение в команде if

```
changeLights();
```

Это пример вызова функции. Функция - это просто отдельный блок кода, которому присвоено имя. Однако функциям также можно передавать параметры и / или возвращать данные. В этом случае мы не передали в функцию никаких данных, и функция не вернула никаких данных. Позже мы более подробно рассмотрим передачу параметров и возврат данных из функций в главе 3 проекта 10.
Когда changeLights (); вызывается, выполнение кода переходит от текущей строки к функции, выполняет код внутри этой функции, а затем возвращается к точке в коде после того, как функция была вызвана.
Итак, в этом случае, если условия в операторе if выполнены, программа выполняет код внутри функции, а затем возвращается к следующей строке после changeLights (); в операторе if. Код внутри функции просто меняет свет "автомобиля" на красный, на желтый, а затем включает зеленый пешеходный свет. По истечении периода времени, установленного переменной crossTime, свет мигнет несколько раз, чтобы предупредить пешехода о том, что его время скоро истечет, затем пешеходный свет станет красным, а свет транспортного средства перейдет с красного на зеленый, через желтый , и вернется в нормальное состояние. Основной цикл программы просто постоянно проверяет, была ли нажата кнопка пешехода или нет, и если она была, и (&&) время с момента последней замены света больше пяти секунд, он снова вызывает функцию changeLights (). В этой программе не было никакой пользы от помещения кода в отдельную функцию, кроме как сделать код более чистым и объяснить вам понятие функций. Их истинные преимущества обнаруживаются только тогда, когда функции передаются параметры и / или возвращаются данные, и мы рассмотрим это позже, когда снова будем использовать функции. Уменьшение размера кода также является истинным преимуществом, которое может быть достигнуто за счет инкапсуляции кода, который используется более чем в одном месте кода - например, создание точек в коде генерации SOS. Это преимущество приходит без необходимости в передаваемых параметрах или возвращаемых значениях.
Далее, в Проекте 5, мы собираемся использовать намного больше светодиодов, поскольку мы создадим эффект погони за светодиодами в стиле Knight Rider.

Проект 4 - Интерактивные светофоры - Обзор компонентов

Новое компонент, представленный в Project 4, - это кнопка. Как вы можете видеть, посмотрев на схему, кнопка не подключена напрямую между линией питания и входным контактом. Между кнопкой и земляной шиной также есть резистор. Это так называемый понижающий резистор, который необходим для правильной работы кнопки. Теперь мы сделаем небольшое отступление, чтобы объяснить роль подтягивающих и понижающих резисторов.

Логические состояния

Логическая схема предназначена для включения или выключения выхода. Они представлены двоичными числами 1 и 0. Для входов и выходов Arduino выключенное (или нулевое) состояние - это напряжение, близкое к нулю, т.е. фактически это земля.

а состояние включения (или 1) представлено высоким уровнем, близким к напряжению питания. Простейшим представлением логической схемы является выключатель (см. рис. 2-9).

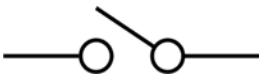


Рис. 2-9. *Электронный символ выключателя*

Выход логической схемы тоже можно считать своеобразным выключателем. Напряжение логической 1 должно быть как можно ближе быть к 5В,а логического нуля - к нулевому напряжению. Если это условие не соблюдается,то эта часть схемы может считаться «плавающей», то есть она не находится ни в высоком, ни в низком состоянии. Плавающее состояние восприимчиво к электрическому шуму, и шум в цифровой схеме можно интерпретировать как случайные единицы и нули. Здесь можно использовать подтягивающие или стягивающие резисторы, чтобы гарантировать высокое или низкое состояние логической схемы.

Стягивающие резисторы

Стягивающие резисторы подтягивают вывод к логически низкому значению. Они подключаются между землей и соответствующим контактом устройства. Пример стягивающего резистора в цифровой схеме можно увидеть на рис. 2-10.
Кнопка подключена между источником напряжения VCC и выводом контроллера PB6. В такой схеме, когда кнопка замкнута, вход схемы имеет высокое логическое значение, но когда кнопка разомкнута, стягивающий резистор понижает входное напряжение до земли (значение логического нуля), предотвращая неопределенное состояние на входе. Стягивающий резистор должен иметь большее сопротивление, чем импеданс логической схемы. На рис. 2.12 представлена схема подключения резистора в программе fritzing.

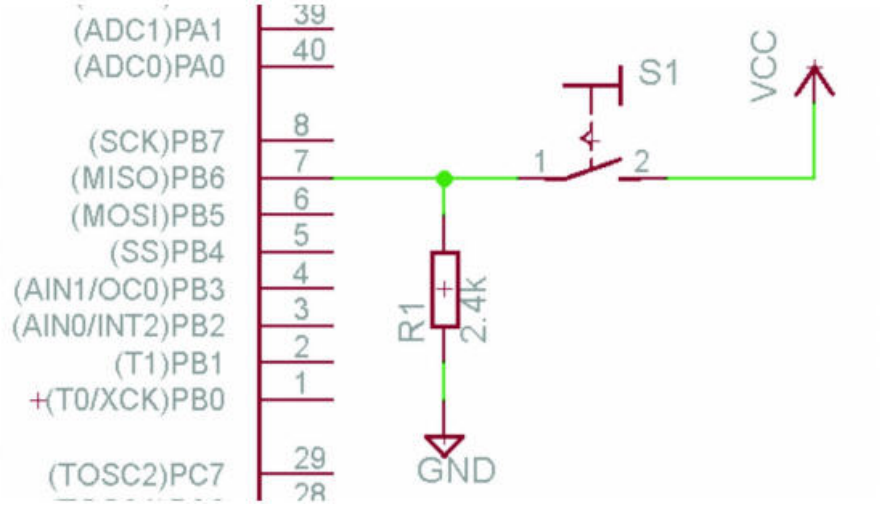


Рис. 2-10. *Схема подключения стягивающего резистора*

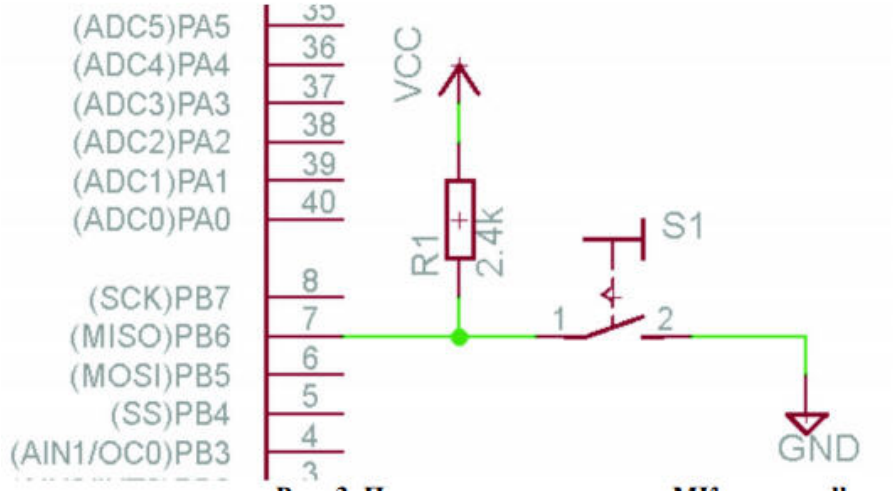


Рис. 2-11. *Схема подключения подтягивающего резистора*

Подтягивающий резистор

pull-up resistor — подтягивающий к питанию резистор. Подтягивающие резисторы необходимы для того, чтобы вывод МК при разомкнутой кнопке имел строго определенный потенциал, а не «болтался в воздухе». Без этого резистора вывод МК будет находиться в третьем состоянии (потенциал на выводе не определен), а при попытке определения состояния вывода, будет считана или «1», или «0» (зависит от внешних наводок на плате).

Рассмотрим схему на рис.2.12 с pull-up резистором более детально.

Когда кнопка разомкнута, вывод PB6 через подтягивающий резистор R1 имеет потенциал VCC (PB6 = 1). При замыкании кнопки на потенциал вывода МК будут оказывать влияние VCC (через R1 = 2.4k) и GND (сопротивления практически нет). Т.о. потенциал GND пересилит VCC и при чтении состояния вывода мы получим PB6=0

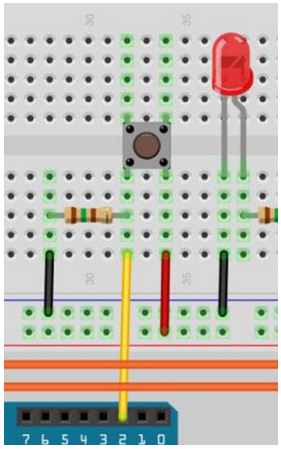


Рис. 2-12.Стягивающий резистор из проекта 4

Вывод(пин) как вход(INPUT) и как выход(OUTPUT).

В книге мы встречаем такие понятия как присвоение(инициализация) выводу микроконтроллера (МК),режима "вход" и режима "выход". Под выводом здесь понимается вывод порта,например на рис.2.10 и рис.2.11 мы видим фрагменты портов PA0-PA5 и PB3-PB7. К ним можно подключить кнопку,микрофон,светодиод или зуммер или еще что-нибудь.

Если к выводу порта подключены кнопка,микрофон мы посредством программы задаем выводу режим "вход -INPUT".

Образно говоря,компоненты в этом режиме "управляют" микроконтроллером,заставляя его совершить какое то действие,например зажечь светодиод,запустить двигатель или зуммер и т.д. Здесь кнопка,микрофон выступают как входные компоненты,а светодиод,двигатель,зуммер - как выходные. Поэтому последние "сидят" на выводах портов МК,находящихся в режиме "выход - OUTPUT".

Функция pinMode () настраивает контакты ввода-вывода устройства. Требуется два аргумента: первый для указания того, какой вывод нужно настроить, а второй для указания направления: INPUT или OUTPUT.

При включении все цифровые выводы ввода / вывода на Arduino конфигурируются как входы.

Внутренние подтягивающие резисторы Arduino

У МК есть подтягивающие резисторы, подключенные к цифровым выводам (аналоговые выводы также имеют подтягивающие резисторы). Они имеют значение 20 кОм и должны быть активированы программой для их использования. Чтобы активировать внутренний подтягивающийрезистор на выводе, вам сначала нужно изменить pinMode вывода на INPUT, а затем записать HIGH на этот вывод с помощью команды digitalWrite. Например:

```
pinMode(pin, INPUT);
digitalWrite(pin, HIGH);
```

Если вы измените pinMode с INPUT на OUTPUT после активации внутренних подтягивающих резисторов, то пин останется в состоянии ВЫСОКИЙ. Это также работает в обратном порядке: на выходном пине, который находился в состоянии ВЫСОКИЙ, который впоследствии был переключен в режим ВХОДА, будут включены внутренние подтягивающие резисторы.Теперь, когда вы понимаете использование подтягивающих и стягивающих резисторов, которые будут использоваться на протяжении всей книги, давайте перейдем к проекту 5 и создадим некоторые световые эффекты с помощью наших светодиодов.

Резюме

Наши первые четыре проекта охватили много вопросов. Теперь вы знаете основы считывания входных данных и включения и выключения светодиодов. Вы начинаете накапливать свои знания в области электроники, понимая, как работают светодиоды и резисторы, как резисторы можно использовать для ограничения тока и как их можно использовать для повышения или понижения входного сигнала в соответствии с вашими потребностями.Теперь вы также можете взять резистор и вычислить его значение в омах, просто взглянув на его цветные полосы.Ваше понимание языка программирования Arduino идет полным ходом, и вы познакомились с рядом команд и концепций. Навыки, полученные в главе 2, являются основой даже для самого сложного проекта Arduino. В главе 3 мы продолжим использовать светодиоды для создания различных эффектов и при этом изучите огромное количество команд и понятий. Эти знания подготовят вас к изучению более сложных предметов, которые будут рассмотрены позже в книге.

Предметы и понятия, затронутые в главе 2:

- Важность комментариев в коде
- Переменные и их типы
- Назначение функций setup () и loop ()
- Понятие о функциях и способы их создания
- Установка pinMode цифрового вывода
- Запись HIGH или LOW значения на вывод
- Как создать задержку на указанное количество миллисекунд
- Макетные платы и способы их использования
- Что такое резистор и как его использовать для ограничения тока
- Как рассчитать необходимое сопротивление резистора для светодиода
- Как рассчитать номинал резистора по цветным полосам
- Что такое светодиод и как он работает
- Как заставить код повторяться с помощью цикла for
- Операторы сравнения

- Простая математика в коде
- Разница между локальной и глобальной областью действия
- Подтягивающие и стягивающие резисторы и способы их использования
- Как читать нажатие кнопки
- Принятие решений с помощью оператора `if`
- Изменение режима вывода между `INPUT` и `OUTPUT`
- Функция `millis()` и как ее использовать
- Логические операторы и как их использовать для принятия логических решений

ГЛАВА 3

Светодиодные эффекты

В главе 2 вы изучили основы ввода и вывода, некоторую элементарную электронику и целый ряд концепций программирования. В этой главе мы собираемся продолжить работу со светодиодами, создавая на их основе необычные эффекты. Мы познакомимся со многими важными понятиями кодирования, такими как массивы, математические функции и последовательная связь, которые обеспечат необходимые навыки программирования для решения более сложных проектов, представленных далее в этой книге.

Проект 5 - Эффект светодиодной погони

Теперь мы собираемся использовать цепочку из 10 светодиодов , чтобы создать эффект светодиодной погони.

Требуемые детали

Таб. 3-1. *Детали, необходимые для проекта 5*

10 x 5mm красных светодиодов	
10 x токоограничивающих резистора	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. А теперь возьмите макетную плату, светодиоды,резисторы и перемычки и подключите все, как показано на рис. 3-1. Тщательно проверьте схему перед подключением питания на Arduino.

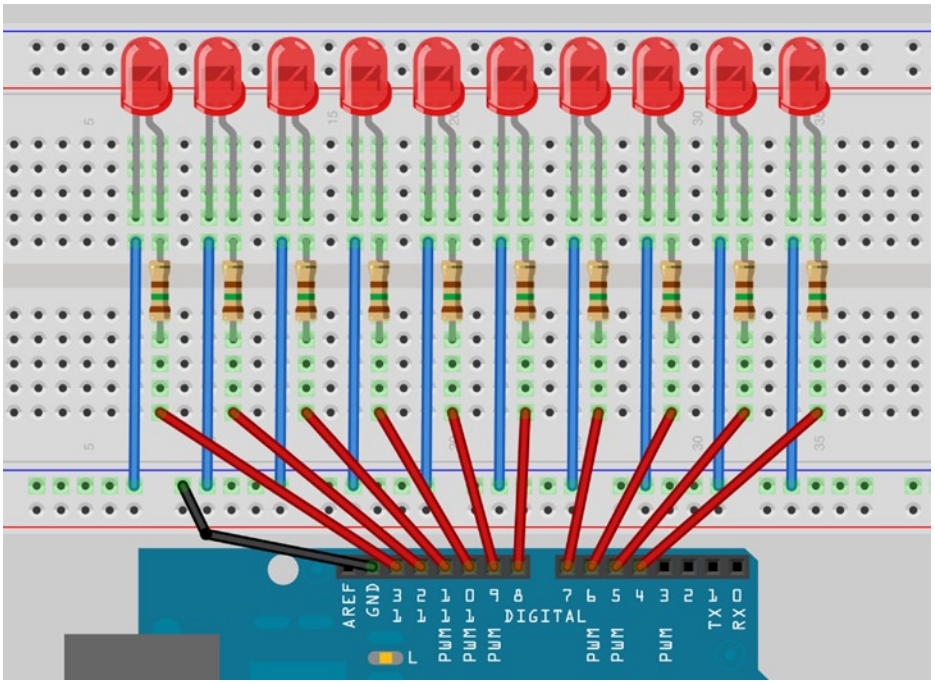


Рис. 3-1. *Схема для Проекта 5 - Эффект светодиодной погони*

Ввод кода

Откройте IDE Arduino и введите код из листинга 3-1 или из папки с кодами:

Листинг 3-1. Код для проекта 5

```
// Project 5 - LED Chase Effect
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};     //Создаем массив для выводов светодиодов
int ledDelay = 65;                                       // задержка между изменениями
int direction = 1;
int currentLED = 0;
unsigned long changeTime;

void setup() {
        for (int x=0; x<10; x++) {                       // устанавливаем все пины на вывод
                pinMode(ledPin[x], OUTPUT);
        }
        changeTime = millis();
}

void loop() {
        if ((millis() - changeTime) > ledDelay) {// если с последнего изменения было ledDelay мс
                changeLED();
                changeTime = millis();
        }
}
```

```
void changeLED() {
        for (int x=0; x<10; x++) {                    // выключить все светодиоды
                digitalWrite(ledPin[x], LOW);
        }
        digitalWrite(ledPin[currentLED], HIGH);       // включаем текущий светодиод
        currentLED += direction;                      // увеличиваем на значение направления
        // меняем направление, если дойдем до конца
        if (currentLED == 9) {direction = -1;
        }if (currentLED == 0) {direction = 1;}
}
```

Теперь нажмите кнопку "Проверить" в верхней части IDE, чтобы убедиться, что в вашем коде нет ошибок. В случае успеха теперь вы можете нажать кнопку «Загрузка», чтобы загрузить код в свой Arduino. Если все сделано правильно, вы должны увидеть, как светящиеся светодиоды движутся друг за другом , а затем возвращаются в исходное положение. Мы ввели новое понятие в код этого проекта в виде массивов. Давайте посмотрим на код Проекта 5 и посмотрим, как он работает.

Проект 5 - Эффект светодиодной погони - Обзор кода

Наша самая первая строка в этом скетче

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

который объявляет переменную ledPin как массив элементов типа данных byte. Все элементы имеют одно и то же имя (в данном случае ledPin), но могут быть выбраны индивидуально по порядковому номеру, как квартиры в многоквартирном доме. [] после имени переменной в объявлении сообщает компилятору, что это переменная массива, а не простая (неиндексированная) переменная. Затем мы инициализировали массив с 10 значениями, которые представляют собой цифровые выводы с 4 по 13.
Чтобы получить доступ к элементу массива, мы следуем за именем массива с номером индекса этого элемента в квадратных скобках. Номер индекса между [] не обязательно должен быть константой - это может быть переменная или выражение. Массивы имеют нулевой индекс, что просто означает, что индекс первого элемента равен нулю, а не единице. Итак, в нашем 10-элементном массиве номера индексов от 0 до 9. В этом случае третий элемент (ledPin [2]) имеет значение 6, а седьмой элемент (ledPin [6]) - значение 10. Вы должны сообщить компилятору размер массива, если вы сначала не инициализируете его данными в объявлении. В нашем скетче мы не выбирали размер явно, так как компилятор сам может подсчитать значения, которые мы присвоили массиву, чтобы определить, что размер равен 10 элементам. Если бы мы объявили массив, но не инициализировали его значениями одновременно, нам нужно было бы объявить размер, например, мы могли бы сделать это:

```
byte ledPin[10];
```

а затем загружали данные в элементы позже. Чтобы получить значение из массива, мы должны сделать что-то вроде этого:

```
x = ledpin[5];
```

В этом примере x теперь будет содержать значение 8. Чтобы вернуться к нашей программе, мы начали с объявления и инициализации массива с 10 значениями, которые являются цифровыми номерами выводов, используемых для выходов наших 10 светодиодов.В нашем основном цикле мы проверяем, что с момента последней смены светодиодов прошло не менее ledDelay миллисекунд, и если это так, управление передается нашей функции. Причина, по которой мы собираемся передать управление только функции changeLED () таким образом, а не использовать команды delay (), состоит в том, чтобы позволить другому коду, если необходимо, запускаться в основном цикле программы, пока этот код занимает меньше времени чем ledDelay для запуска.

Созданная нами функция

```
void changeLED() {
        // выключаем все светодиоды
        for (int x=0; x<10; x++) {
                digitalWrite(ledPin[x], LOW);
        }
        // включаем текущий светодиод
        digitalWrite(ledPin[currentLED], HIGH);
        // увеличиваем на значение направления
        currentLED += direction;
        // меняем направление, если дойдем до конца
        if (currentLED == 9) {direction = -1;
        }if (currentLED == 0) {direction = 1;}
}
```

и задача этой функции - выключить все светодиоды, а затем включить текущий светодиод (это делается так быстро, что вы этого не заметите), который сохраняется в переменной currentLED.
Затем к этой переменной добавляется направление. Поскольку направление может быть только 1 или -1, тогда число будет либо увеличиваться (+1), либо уменьшаться на единицу (currentLED + (- 1)).
Затем у нас есть оператор if, чтобы увидеть, достигли ли мы конца ряда светодиодов, и если это так, мы затем меняем переменную направления.
Изменяя значение ledDelay, вы можете заставить светодиодный индикатор гонять вперед и назад с разной скоростью. Попробуйте разные значения, чтобы увидеть, что произойдет. Вы должны остановить программу и вручную изменить значение ledDelay, а затем загрузить измененный код, чтобы увидеть любые изменения Однако было бы неплохо иметь возможность регулировать скорость во время работы программы? Поэтому давайте сделаем именно это в следующем проекте, представив способ взаимодействия с программой и регулировки скорости с помощью потенциометра.

Проект 6 - Интерактивный светодиодный эффект погони

Не нарушая соединении схемы на макетной плате (рис. 3.1) мы добавим в эту схему потенциометр, который позволит нам изменять скорость пробега при изменении его сопротивления.

Требуемые детали

Используйте схему из Проекта 5 и добавьте потенциометр.

Таб. 3-2. *Детали, необходимые для проекта 6*

Потенциометр 4,7 кОм	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Теперь добавьте в схему потенциометр в соответствии с рис. 3-2:

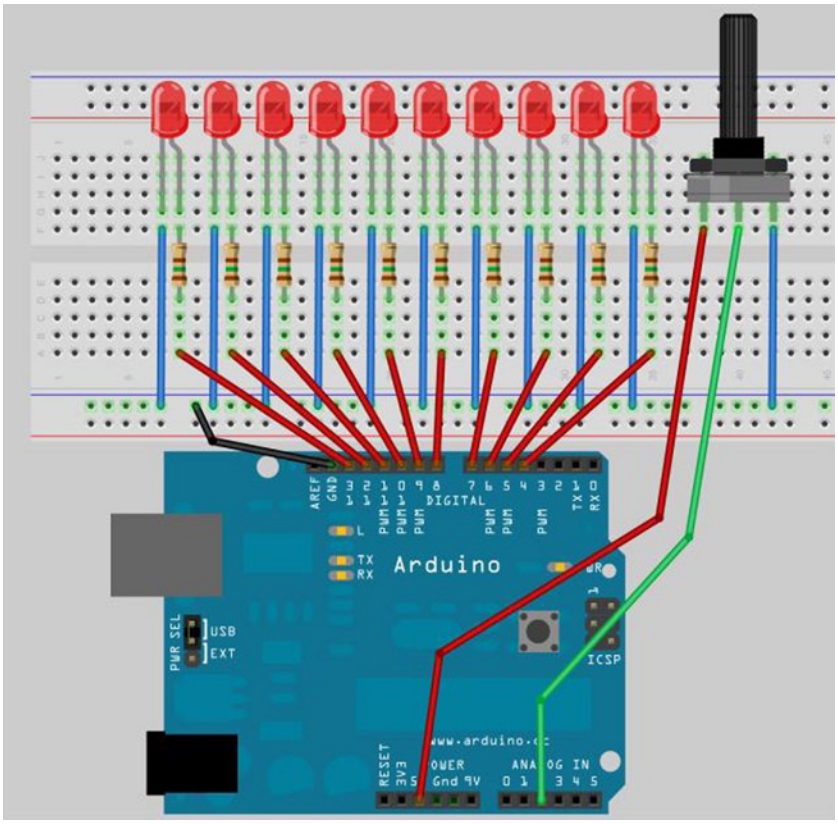


Рис. 3-2. Схема для Project 6 - Интерактивный LED эффект

Ввод кода

Откройте IDE Arduino и введите код из листинга 3-2 или из папки с кодами:

Листинг 3-2. Код для проекта 6

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};   // Создаем массив  светодиодов
int ledDelay;      //  задержка между изменениями
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2; // выбираем входной контакт для потенциометра
```

```
void setup() {
        for (int x=0; x<10; x++) {        // устанавливаем все пины на вывод
                pinMode(ledPin[x], OUTPUT);
        }
        changeTime = millis();
}

void loop() {
        ledDelay = analogRead(potPin);       // считываем значение с потенциометра
        if ((millis() - changeTime) > ledDelay) { // если с момента последнего изменения было ledDelay мс
                changeLED();
                changeTime = millis();
        }
}

void changeLED() {
        for (int x=0; x<10; x++) {                    // выключаем все светодиоды
                digitalWrite(ledPin[x], LOW);
        }
        digitalWrite(ledPin[currentLED], HIGH);    // включаем текущий светодиод
        currentLED += direction;               // увеличиваем на значение направления
                                                    // меняем направление, если дойдем до конца
         if (currentLED == 9) {direction = -1;}
        if (currentLED == 0) {direction = 1;}
}
```

На этот раз, когда вы проверяете и загружаете свой код, вы должны увидеть бегущий светящийся светодиод. Но, поворачивая ручку потенциометра, вы изменяете значение ledDelay и ускоряете или замедляете эффект.

Давайте посмотрим, как это работает и узнаем, что такое потенциометр.

Проект 6 - Интерактивный светодиодный эффект погони - Обзор кода

Код этого проекта почти идентичен коду предыдущего проекта. Мы просто добавили потенциометр к нашей схеме, и в коде есть дополнения, позволяющие нам считывать значения с потенциометра и использовать их для регулировки скорости бегущего светодиода.

Сначала мы объявляем переменную для номера вывода (ползунка) потенциометра.

```
int potPin = 2;
```

поскольку наш потенциометр подключен к аналоговому выводу 2. Чтобы считать значение с аналогового вывода, мы используем команду analogRead. Arduino имеет 6 аналоговых входов / выходов с 10-битным аналого-цифровым преобразователем (биты мы обсудим позже). Это означает, что аналоговый вывод может считывать напряжения от 0 до 5 вольт в целочисленных значениях от 0 (0 вольт) до 1023 (5 вольт). Это дает разрешение 5 В / 1024 единиц или 0,0049 В (4,9 мВ) на единицу.Нам нужно установить нашу задержку с помощью потенциометра, поэтому мы просто будем использовать прямые значения, считанные с вывода, чтобы настроить задержку между 0 и 1023 миллисекундами (или чуть более 1 секунды). Мы делаем это, напрямую считывая значение вывода потенциометра в ledDelay. Обратите внимание, что нам не нужно устанавливать аналоговый вывод как вход или выход, как это делается с цифровым выводом (пином).

```
ledDelay = analogRead(potPin);
```

Это делается во время нашего основного цикла, и поэтому он постоянно читается и корректируется. Поворачивая ручку, вы можете настроить значение задержки от 0 до 1023 миллисекунды (или чуть более секунды) и, следовательно, полностью управлять скоростью эффекта.
Теперь разберемся, что такое потенциометр и как он работает.

Проект 6 - Интерактивный светодиодный эффект погони - Обзор схемы

Единственным дополнительным компонентом, используемым в этом проекте, будет потенциометр 4,7 (4700 Ом) (см. Рисунок 3-3).

Рис. 3-3. *Потенциометр*

. Потенциометр - это просто фиксированный резистор с ползунком, который можно использовать для разделения общего сопротивления на две части, которые в сумме дают общее (фиксированное) значение. В этом проекте мы используем потенциометр 4,7 кОм или 4700 Ом, что означает, что его диапазон составляетот 0 до 4700 Ом.
Потенциометр имеет три ножки. Подключив всего две ножки, потенциометр становится переменным резистором. Подключив все три ножки и приложив к ним напряжение, потенциометр становится делителем напряжения. Вот как мы использовали его в нашей схеме. Один крайний вывод подключен к земле, другой - к 5 В, а центральный вывод - к нашему аналоговому выводу.
Регулируя ручкой, с ползунка будет доступно напряжение от 0 до 5 В; мы можем прочитать значение этого напряжения на аналоговом выводе 2 и использовать его значение для изменения скорости задержки светового эффекта.

упражнение

Теперь попробуйте проделать следующие два упражнения. У вас есть все необходимые знания для написания кода, чтобы вы могли сделать следующее:

- Упражнение 1. Заставьте светодиоды сбегать с концов полосы,встречаться в середине и опять разбегаться.
- Упражнение 2: Создайте эффект прыгающего мяча, установив LED полосу вертикально.
  При включении загорается самый нижний LED, затем он «подпрыгивает» до верхнего LED, а затем снова вниз; в следующее время пусть он подскакивает» только до 9-го светодиода, затем обратно вниз, затем до 8-го и т. д., чтобы смоделировать прыгающий мяч,теряющий импульс после каждого отскока.

Проект 7 - Пульсирующая лампа

Теперь мы углубимся в более продвинутый метод управления светодиодами. Пока мы просто включали или выключали светодиод. Как насчет того, чтобы иметь возможность регулировать его яркость? Можем ли мы сделать это с помощью Arduino? Конечно можно.

Требуемые детали

В Таблице 3-3 перечислены детали, необходимые для Проекта 7.

Таб. 3-3. *Детали, необходимые для проекта 7*

Зеленый 5mm LED	
Токоограничивающий резистор	

Подключение

Схема для этого проекта представляет собой зеленый светодиод, подключенный через токоограничивающий резистор между землей и цифровым выводом 11. См. рис. 3-4.

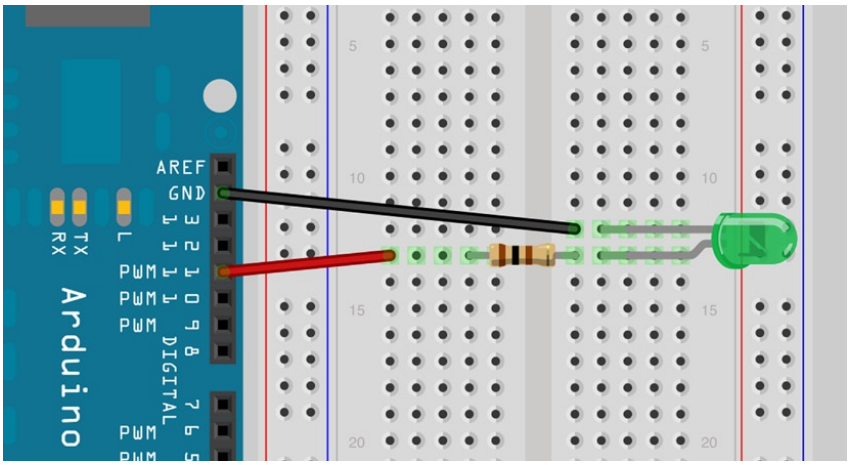


Рис. 3-4. *Схема для проекта 7 - Пульсирующая лампа.*

Введите код

Откройте IDE Arduino и введите код из листинга 3-3:

Листинг 3-3. Код для проекта 7

```
// Проект 7 - Пульсирующая лампа
int ledPin = 11;
float sinVal;
int ledVal;

void setup() {
        pinMode(ledPin, OUTPUT);
}

void loop() {
        for (int x=0; x<180; x++) {
        // convert degrees to radians then obtain sin value
        sinVal = (sin(x*(3.1412/180)));
        ledVal = int(sinVal*255);
        analogWrite(ledPin, ledVal);
        delay(25);
        }
}
```

Проверить и загрузить. Теперь вы увидите, что ваш светодиод постепенно загорается и гаснет. Вместо простого включения / выключения теперь мы получили пульсирующую лампу. Давайте узнаем, как это работает

Проект 7 - Пульсирующая лампа - Обзор кода

Код этого проекта очень прост, но требует некоторых пояснений.
Сначала мы настраиваем переменные для вывода светодиода, числа с плавающей запятой (тип данных с плавающей запятой) для значения синусоидального сигнала и ledVal, который будет содержать целочисленное значение для отправки на вывод 11 платы Arduino.
Идея заключается в том, что мы создаем синусоидальный сигнал, и яркость светодиода должна соответствовать частоте этого сигнала, что заставляет свет пульсировать, а не мигать с резкими переходами яркости.
Мы используем функцию sin (), которая представляет собой математическую функцию для синуса угла. Нам нужно задать функции угол, выраженный в радианах. У нас есть цикл for от 0 до 179; мы не берем число больше 179, так как это приведет нас к отрицательным значениям, а значение яркости, которое нам нужно передать на вывод 11, должно быть только от 0 до 255.
Функция sin () требует, чтобы угол был в радианах, а не в градусах, поэтому уравнение x * (3.1412 / 180) преобразует градусный угол в радианы. Затем мы передаем результат в ledVal, умножая его на 255, чтобы получить наше значение.
Результатом функции sin () будет число от -1 до 1, поэтому нам нужно умножить это на 255, чтобы получить максимальную яркость. Мы «преобразуем» значение sinVal с плавающей запятой в целое число с помощью int () в инструкции

```
ledVal = int(sinVal*255);
```

Затем мы отправляем это значение на цифровой вывод 11, используя оператор

```
analogWrite(ledPin, ledVal);
```

Используя функцию Arduino analogWrite(pin, value),где value - значение мы можем обеспечить плавное загорание и затухание светодиода.
Для этого мы используем метод, называемый широтно-импульсной модуляцией (ШИМ). ШИМ - это метод получения аналогового сигнала с помощью цифровых средств.
Изменяя рабочий цикл (рабочий цикл - это процент периода, когда сигнал активен), мы можем имитировать аналоговое напряжение (см.рис.). Когда нам нужно среднее напряжение, мы используем рабочий цикл равным 50%. Точно так же, если мы хотим подавать низкое или высокое напряжения, мы используем рабочий цикл на уровне 10% и 90% соответственно. В этом проекте ШИМ - это процесс управления напряжением, подаваемой на светодиод:

Arduino UNO имеет шесть выводов (3, 5, 6, 9, 10 и 11) в качестве ШИМ. Эти выводы управляются встроенными таймерами, которые автоматически переключают выводы с частотой около 490 Гц. В этом проекте мы используем 11 вывод.
Функция analogWrite () принимает в качестве параметра номер вывода и значение вывода. Здесь, значение вывода может быть от 0 до 255, а рабочий цикл сопоставлен с 0% и 100%.

Позже мы вернемся к ШИМ, чтобы использовать его для создания звуков с помощью пьезозумера.

Проект 8 - лампа настроения RGB

В последнем проекте мы увидели, что можем регулировать яркость светодиода, используя возможности ШИМ микросхемы Atmega. Теперь мы воспользуемся этой возможностью, используя красный, зеленый и синий светодиоды для смешивания цветов, чтобы создать любой желаемый цвет. Исходя из этого, мы создадим лампу настроения, подобную той, которую вы часто видите в магазинах.

Требуемые детали

На этот раз мы будем использовать три светодиода: красный, зеленый и синий.

Table 3-4. *Детали, необходимые для проекта 7*

Красный 5mm LED	
Зеленый 5mm LED	
Синий 5mm LED	
3 токоограничивающих резистора	

Подключение

Подключите три светодиода, как показано на рис. 3-5. Возьмите лист бумаги размером A5 , скатайте его в цилиндр, затем закрепите скотчем, чтобы он оставался таким. Затем поместите цилиндр поверх трех светодиодов. Он будет объединять цвета в один.

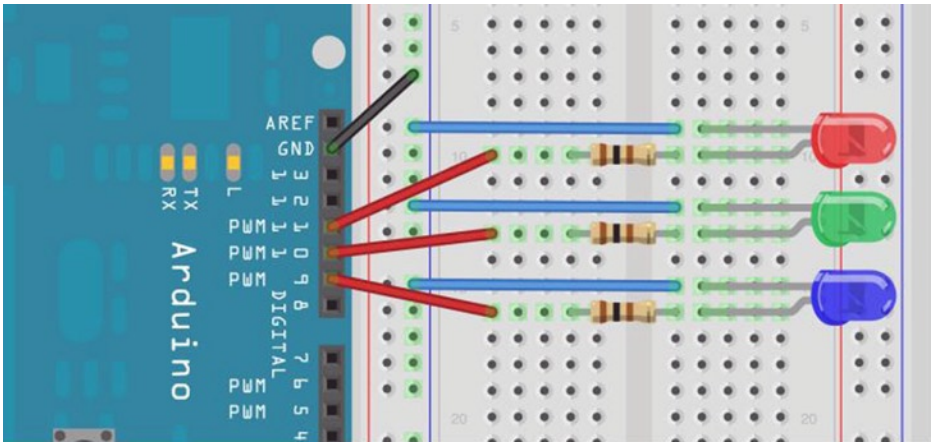


Рис. 3-5. *Схема для проекта 8 - лампа настроения RGB*

Ввод кода

Откройте IDE Arduino и введите код из листинга 3-4:

Листинг 3-4. Код для проекта 8

```
// Проект 8 - Лампа настроения
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;

void setup()
{
        Serial.begin(9600);
        randomSeed(analogRead(0));

        RGB1[0] = 0;
        RGB1[1] = 0;
        RGB1[2] = 0;
```

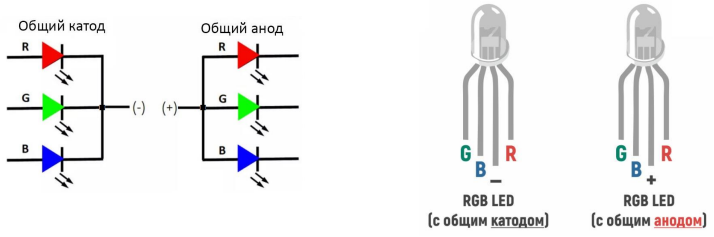
```
        RGB2[0] = random(256);
        RGB2[1] = random(256);
        RGB2[2] = random(256);
}
void loop()
{
        randomSeed(analogRead(0));

        for (int x=0; x<3; x++) {
                INC[x] = (RGB1[x] - RGB2[x]) / 256; }
        for (int x=0; x<256; x++) {
                red = int(RGB1[0]);
                green = int(RGB1[1]);
                blue = int(RGB1[2]);

                analogWrite (RedPin, red);
                analogWrite (GreenPin, green);
                analogWrite (BluePin, blue);
                delay(100);
                RGB1[0] -= INC[0];
                RGB1[1] -= INC[1];
                RGB1[2] -= INC[2];
        }
        for (int x=0; x<3; x++) {
                RGB2[x] = random(556)-300;
                RGB2[x] = constrain(RGB2[x], 0, 255);
                delay(1000);
         }
}
```

Когда вы запустите его, вы увидите, что цвета медленно меняются. Вы только что сделали свою лампу настроения.

Проект 8 -Лампа настроения -Обзор кода

Светодиоды, из которых состоит лампа настроения, красного, зеленого и синего цвета. Смешивая программным путем значения цветов, мы можем получить множество цветов. Утверждают, что можно имитировать 16,7 миллиона различных цветов.

В качестве альтернативы можно использовать светодиод RGB в виде отдельного компонента. Это 5-миллиметровый светодиод с четырьмя ножками. Одна ножка - это либо общий анод , либо общий катод, а три других подключаются к противоположному выводу красного, зеленого и синего светодиодов . По сути, это три светодиода в одном корпусе. В нашем проекте следует использовать RGB светодиод с общим катодом. На рисунке изображены RGB светодиоды :



Таб. 3-5. *Цвета, доступные при включении или выключении светодиодов в различных комбинациях*

Красный	Зеленый	Синий	Цвет
255	0	0	Красный
0	255	0	Зеленый
0	0	255	Синий
255	255	0	Желтый
0	255	255	Пурпурный
255	0	255	Сине-зеленый
255	255	255	Белый
0	0	0	Черный

Регулируя яркость с помощью ШИМ, мы можем получить любой другой цвет между ними. Помещая светодиоды близко друг к другу и смешивая их значения, спектры света трех сложенных вместе цветов образуют один цвет (см. Рисунок 3-6). Рассеивая свет с помощью бумажного цилиндра, мы обеспечиваем хорошее смешивание цветов. Светодиоды можно поместить в любой объект, который будет рассеивать свет, или вы можете отразить свет от отражающего диффузора. Попробуйте поместить лампочки в шарик для настольного тенниса или небольшую белую пластиковую бутылку (чем тоньше пластик, тем лучше).

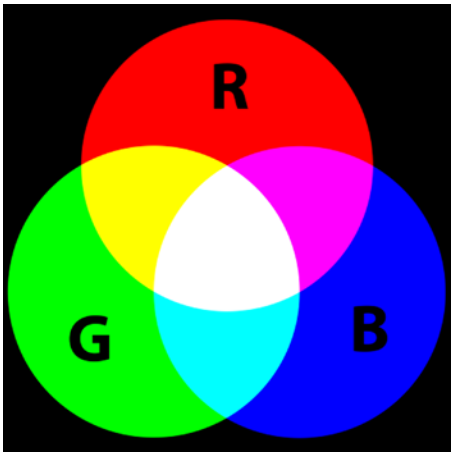


Рис. 3-6. *Смешивание R, G и B для получения разных цветов*

Общий диапазон цветов, который мы можем получить с помощью ШИМ с диапазоном от 0 до 255, составляет 16 777, 216 цветов (256 x 256 x 256), что намного больше, чем нам когда-либо понадобится.

В коде мы начинаем с объявления некоторых массивов с плавающей запятой, а также некоторых целочисленных переменных, которые будут хранить наши значения RGB, а также значение приращения.

```
float RGB1[3];
float RGB2[3];
float INC[3];

int red, green, blue;
```

В функции настройки у нас есть

```
randomSeed(analogRead(0));
```

Команда randomSeed используется для создания случайных (фактически псевдослучайных) чисел. Программа не может создавать действительно случайные числа, поэтому она использует математическую функцию, которая перед повторением генерирует очень длинную последовательность псевдослучайных значений. Установив «начальное число», вы можете указать компьютеру, где в последовательности начать возвращать случайные числа. В этом случае значение, которое мы даем randomSeed, является значением, считываемым с аналогового вывода 0. Поскольку у нас ничего не подключено к аналоговому выводу 0, все, что мы будем читать, - это случайное число, созданное аналоговым шумом. После того, как мы установили «начальное число» для нашего случайного числа, мы можем создать его с помощью функции random (). Затем у нас есть два набора значений RGB, хранящихся в трехэлементном массиве. RGB1 - это значение RGB, с которого должна начинаться лампа (в данном случаевсе нули или выключены).

```
RGB1[0] = 0;
RGB1[1] = 0;
RGB1[2] = 0;
```

Массив RGB2 - это набор случайных значений RGB, к которым мы хотим, чтобы лампа перешла на

```
RGB2[0] = random(256);
RGB2[1] = random(256);
RGB2[2] = random(256);
```

В этом случае мы установили значения RGB на случайное число, заданное с помощью random (256), что даст нам число от 0 до 255 включительно (поскольку число всегда будет в диапазоне от нуля и выше).Если вы передадите в функцию random () одно число, она вернет значение от 0 до 1, меньшее, чем это число; random (1000) вернет число от 0 до 999. Если вы укажете два числа в качестве параметров, тогда она вернет случайное число между меньшим числом включительно и максимальным числом минус 1 (-1).Например, random (10,100) вернет случайное число от 10 до 99.

В основном цикле программы мы сначала смотрим на начальное и конечное значения RGB и выясняем, какое значение необходимо в качестве приращения для перехода от одного значения к другому за 256 шагов (поскольку значение ШИМ может быть только между 0 и 255). Мы делаем это с

```
for (int x=0; x<3; x++) {
        INC[x] = (RGB1[x] - RGB2[x]) / 256;
}
```

Этот цикл for устанавливает значения INCrement для каналов R, G и B, вычисляя разницу между двумя значениями яркости и разделив ее на 256.

Затем у нас есть еще один цикл for

```
for (int x=0; x<256; x++) {

        red = int(RGB1[0]);
        green = int(RGB1[1]);
        blue = int(RGB1[2]);

        analogWrite (RedPin, red);
        analogWrite (GreenPin, green);
        analogWrite (BluePin, blue);
        delay(100);
```

```
        RGB1[0] -= INC[0];
        RGB1[1] -= INC[1];
        RGB1[2] -= INC[2];
  }
```

и он устанавливает значения красного, зеленого и синего в значения в массиве RGB1, записывает эти значения на выводы 9, 10 и 11, затем вычитает значение приращения и затем повторяет этот процесс 256 раз, чтобы медленно исчезнуть с одного случайного цвета к следующему. Задержка в 100 мс между каждым шагом обеспечивает медленное и устойчивое продвижение. Вы можете, конечно, отрегулировать это значение, если хотите, чтобы оно было медленное или быстрое, или вы можете добавить потенциометр, чтобы пользователь мог устанавливать скорость. После 256 медленных шагов от одного случайного цвета к другому массив RGB1 будет иметь те же значения (почти), что и массив RGB2. Теперь нам нужно выбрать другой набор из трех случайных значений, готовый к следующему разу.

Мы делаем этос другим циклом for

```
for (int x=0; x<3; x++) {
        RGB2[x] = random(556)-300;
        RGB2[x] = constrain(RGB2[x], 0, 255);
        delay(1000);
 }
```

Случайное число выбирается путем выбора случайного числа от 0 до 556 (256 + 300), а затем вычитания 300. Причина, по которой мы это делаем, состоит в том, чтобы время от времени пытаться форсировать основные цвета, чтобы гарантировать, что мы не всегда просто получаем пастельные тона. У нас есть 300 шансов из 556 получить отрицательное число и, следовательно, вызвать смещение в сторону одного или нескольких из двух других цветовых каналов. Следующая команда проверяет, что числа, отправленные на выводы ШИМ, не являются отрицательными, используя функцию constrain ().
Функция ограничения требует трех параметров; x, a и b, как в ограничении (x, a, b), где x - это число, которое мы хотим ограничить, a и b нижний и верхний предел диапазона соответственно. Итак, функция ограничения смотрит на значение x и проверяет, находится ли оно в диапазоне от a до b. Если он меньше, чем a, он устанавливает его в a; если он больше, чем b, он устанавливает его на b. В нашем случае мы убеждаемся, что число находится в диапазоне от 0 до 255, что соответствует диапазону нашего вывода ШИМ

Поскольку мы используем случайное (556) -300 для наших значений RGB, некоторые из этих значений будут ниже нуля, а функция ограничения гарантирует, что значение, отправленное в ШИМ, не ниже нуля.

Принудительное смещение в сторону одного или нескольких каналов гарантирует, что мы получаем более яркие и менее пастельные оттенки цвета, а также гарантирует, что время от времени один или несколько каналов полностью отключаются, что дает более интересную смену освещения (или настроения).

упражнение

посмотрите, можете ли вы изменить код, чтобы цвета циклически сменяли цвета радуги, а не случайные цвета.

Проект 9 - Светодиодный эффект огня

Project 9 будет использовать светодиоды и мерцающий случайный световой эффект, снова используя ШИМ, чтобы воссоздать эффект мерцающего пламени.Если вы, например, разместите эти светодиоды внутри модельного дома на макете модельной железной дороги, вы можете создать особый эффект горящего дома или поместить его в искусственный камин в своем доме, чтобы создать эффект огня. . Это простой пример того, как светодиоды можно использовать для создания спецэффектов для фильмов, театральных постановок, модельных диорам и т.д.

Требуемые детали

На этот раз мы будем использовать три светодиода: красный и два желтых.

Таб. 3-6. *Детали, необходимые для проекта 7*

Красный 5mm LED	
2 x желтый 5mm LED	
3 x токоогр. резистора	

Подключение

Выключите Arduino, затем подключите три светодиода, как показано на рисунке 3-7. По сути, это та же схема, что и в проекте 8, но с использованием одного красного и двух желтых светодиодов . Опять же, эффект лучше всего виден, когда свет рассеивается с помощью бумажного цилиндра или когда свет отражается от зеркала на поверхность, на которую вы хотите проецировать эффект.

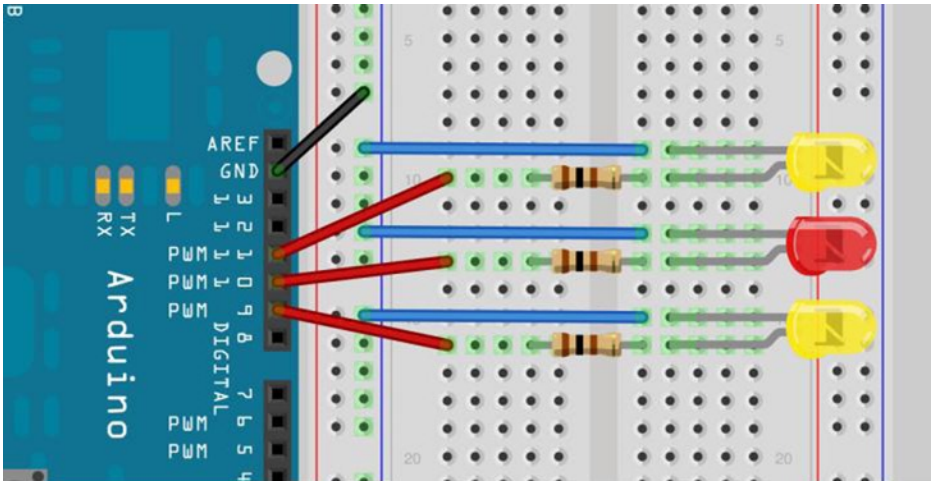


Рис. 3-7.*Схема для проекта 9 – LED эффект огня*

Введем код

Откройте IDE Arduino и введите код из листинга 3-5:

Листинг 3-5. Код для проекта 9

```
// Project 9 - LED Fire Effect
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;
```

```
void setup()
{
        pinMode(ledPin1, OUTPUT);
        pinMode(ledPin2, OUTPUT);
        pinMode(ledPin3, OUTPUT);
}

void loop()
{
        analogWrite(ledPin1, random(120)+135);
        analogWrite(ledPin2, random(120)+135);
        analogWrite(ledPin3, random(120)+135);
        delay(random(100));
}
```

Теперь нажмите кнопку «Проверить / компилировать» в верхней части IDE, чтобы убедиться, что в вашем коде нет ошибок. В случае успеха теперь вы можете нажать кнопку «Загрузить», чтобы загрузить код в свой Arduino.Если вы все сделали правильно, теперь вы должны увидеть, как светодиоды мерцают случайным образом, чтобы имитировать пламя или эффект огня.Теперь давайте взглянем на код и узнаем, как он работает.

Проект 9 - Светодиодный эффект огня - Обзор кода

Итак, давайте посмотрим на код этого проекта. Сначала мы объявляем и инициализируем некоторые целочисленные переменные, которые будут содержать значения цифровых выводов, к которым мы собираемся подключить наши светодиоды.

```
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 11;
```

Затем мы настраиваем их как выходы.

```
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
```

Затем основной цикл программы отправляет случайное значение от 0 до 120, а затем добавляет к нему 135, чтобы получить полную яркость светодиода, на выводы 9, 10 и 11 ШИМ.

```
analogWrite(ledPin1, random(120)+135);
analogWrite(ledPin2, random(120)+135);
analogWrite(ledPin3, random(120)+135);
```

Наконец, у нас есть случайная задержка от нуля до 100 мс.

```
delay(random(100));
```

Затем основной цикл запускается снова, вызывая эффект мерцающего света, который вы можете видеть. Отразите свет от белой карты или зеркала на стене, и вы увидите очень реалистичный эффект пламени. Поскольку устройство простое, мы сразу перейдем к Проекту 10.

упражнение

Теперь попробуйте проделать следующие два упражнения:

- Упражнение 1. Сможете ли вы воссоздать эффект вспышек света от сварочного аппарата,используя один или два синий/или белый светодиоды.
- Упражнение 2: Используя синий и красный светодиоды, воссоздайте эффект огней автомобиля скорой помощи.

Проект 10 - Лампа настроения с последовательным управлением

Для проекта 10 мы вернемся к схеме из проекта 8, RGB Лампа настроения, но теперь углубимся в мир последовательной связи и будем управлять нашей лампой, отправляя команды с ПК на Arduino с помощью монитора последовательного порта (МПП) в Arduino IDE. Этот проект также знакомит с тем, как мы манипулируем текстовыми строками. Итак, соберите устройство, как в Проекте 8, и введите новый код.

Ввод кода

Откройте вашу Arduino IDE и введите код из листинга 3-6:

Листинг 3-6. Код для проекта 10

```
// Проект 10 - Лампа настроения с последовательным управлением
buffer[18];
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;

void setup()
{
        Serial.begin(9600);
        while(Serial.available())
                Serial.read();
        pinMode(RedPin, OUTPUT);
        pinMode(GreenPin, OUTPUT);
        pinMode(BluePin, OUTPUT);
}

void loop()
{
        if (Serial.available() > 0) {
                int index=0;
                delay(100); // позвольте буферу заполниться
                int numChar = Serial.available();
                if (numChar>15) {
                        numChar=15;
                }
```

```
                while (numChar--) {
                        buffer[index++] = Serial.read();
                }
                splitString(buffer);
        }
}

void splitString(char* data) {
        Serial.print("Data entered: ");
        Serial.println(data);
        char* parameter;
        parameter = strtok (data, " ,");  // Обратите внимание на пробел перед запятой в ","
        while (parameter != NULL) {
                setLED(parameter);
                    parameter = strtok (NULL, " ,");  // пробел перед запятой в ","
}

         // Очистить текстовый и последовательный буферы
        for (int x=0; x<16; x++) {
                buffer[x]='\0';
        }
        while(Serial.available())
                Serial.read();}

void setLED(char* data) {
        if ((data[0] == 'r') || (data[0] == 'R')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(RedPin, Ans);
                Serial.print("Red is set to: ");
                Serial.println(Ans);
        }
        if ((data[0] == 'g') || (data[0] == 'G')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(GreenPin, Ans);
                Serial.print("Green is set to: ");
                Serial.println(Ans);
        }
        if ((data[0] == 'b') || (data[0] == 'B')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(BluePin, Ans);
                Serial.print("Blue is set to: ");
                Serial.println(Ans);
        }
}
```

После проверки кода загрузите его в Arduino.Теперь при загрузке программы вроде ничего не происходит. Это потому, что программа ждет вашего ввода. Запустите МПП, щелкнув его значок на панели задач Arduino IDE.

В текстовом окне МПП теперь вы можете вручную ввести значения R, G и B для каждого из трех светодиодов; Цвет светодиодов изменится на введенный вами цвет. Например, если вы введете R255, красный светодиод будет гореть на полную мощность. Если вы введете R255, G255, то красный и зеленый светодиоды также будут отображаться с полной яркостью.Теперь введите R127, G100, B255, и вы получите красивый пурпурный цвет. Если вы наберете r0, g0, b0, все светодиоды погаснут. Вводимый текст предназначен для приема как строчных, так и прописных букв R, G и B, а затем значения от 0 до 255.Любые значения больше 255 будут уменьшены до 255. Вы можете ввести запятую или пробел между параметрами, и вы можете ввести 1, 2 или 3 значения светодиодов одновременно. Например,

```
r255 b100

r127 b127 g127

G255, B0

B127, R0, G255
```

и т.д.

Проект 10 - Лампа настроения с последовательным управлением - Обзор кода

В этом проекте представлен целый ряд новых понятий, включая последовательную связь, указатели и манипуляции со строками. Так что держитесь за шляпы; это потребует много объяснений.

Сначала мы настраиваем массив char (символов) для хранения нашей текстовой строки. Мы сделали его длиной 18 символов, что превышает максимальное значение, равное 16, которое мы разрешаем, чтобы избежать ошибок «переполнения буфера».

```
char buffer[18];
```

Затем мы устанавливаем целые числа для хранения значений красного, зеленого и синего, а также значений для цифровых выводов.

```
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
```

В нашей функции настройки мы устанавливаем три цифровых вывода в качестве выходов. Но до этого у нас есть команда Serial.begin.

```
void setup()
{
        Serial.begin(9600);
        while(Serial.available())
                Serial.read();
        pinMode(RedPin, OUTPUT);
        pinMode(GreenPin, OUTPUT);
        pinMode(BluePin, OUTPUT);
}
```

Serial.begin сообщает Arduino о необходимости начать последовательную связь, а число в скобках, в данном случае 9600, устанавливает скорость передачи (символов в секунду), с которой будет обмениваться данными последовательная линия.

Следующая строка,

```
while(Serial.available())
        Serial.read();
```

удалит все символы, которые могут быть в последовательной строке, так что она будет пустой и готовой к вводу / выводу.

Она делает это, проверяя, доступны ли какие-либо последовательные данные (в нашем случае, ненужные данные), и, если да, считывает их. При чтении данных удаляются все данные, хранящиеся в последовательном буфере. Линия последовательной связи - это просто способ связи Arduino с внешним миром, в данном случае с ПК и МПП Arduino IDE.В основном цикле у нас есть оператор if. Проверяется условие

```
if (Serial.available() > 0) {
```

Команда Serial.available проверяет, были ли получены какие-либо символы в последовательной линии. Если были получены какие-либо символы, то условие выполняется, и код в блоке кода операторов if теперь выполняется. Команда Serial.available () возвращает количество символов, ожидающих чтения.

```
if (Serial.available() > 0) {
        int index=0;
        delay(100); // позвольте буферу заполниться
        int numChar = Serial.available();
        if (numChar>15) {
                numChar=15;
        }
        while (numChar--) {
                buffer[index++] = Serial.read();
        }
        splitString(buffer);
}
```

Объявляется целое число, называемое индексом, и инициализируется нулем. Мы будем использовать индекс, чтобы отслеживать нашу позицию в массиве символов.Затем мы устанавливаем задержку 100 мс. Это необходимо для того, чтобы дать время для получения полной команды, прежде чем мы продолжим и обработаем данные. Если мы этого не сделаем,возможно, что код начнет обрабатывать текстовую строку до того, как мы получим все данные. Линия последовательной связи очень медленная по сравнению со скоростью, с которой выполняется остальная часть кода. Когда вы отправляете строку символов, функция Serial.available немедленно принимает значение больше нуля, и начинает выполняться инструкция «if». Если бы у нас не было оператора delay (100), Arduino мог бы начать выполнение кода в операторе if до того, как была получена вся текстовая строка, а обработанные последовательные данные могут быть только первыми несколькими символами строки введенного текста.

После того, как мы ожидаем получения полной текстовой строки в течение 100 мс, мы объявляем и инициализируем целое число numChar как количество символов в текстовой строке. Например, если мы отправили этот текст в МПП:

```
R255, G255, B255
```

Тогда значение numChar будет 17. Это 17, а не 16, поскольку в конце каждой строки текста есть невидимый символ, называемый символом NULL. Это символ «ничего», который просто сообщает Arduino о том, что достигнут конец строки текста.

Следующий оператор if проверяет, больше ли значение numChar 15 или нет, и если да, то устанавливает его равным 15. Это гарантирует, что мы не переполним массив char buffer [18];

После этого наступает время оператора while, с чем мы раньше не сталкивались. Мы уже использовали цикл for, который будет повторяться заданное количество раз. Оператор while также является циклом, но выполняется только тогда, когда условие истинно.

Синтаксис:

```
while(expression)
{заявление(я)
}
```

В нашем коде цикл while

```
while (numChar--) {
      buffer[index++] = Serial.read();
}
```

Проверяемое им условие - это просто numChar, другими словами, он проверяет, что значение, хранящееся в целочисленном numChar, не равно нулю. numChar - после него. Это так называемый постдекремент. Это просто означает, что numChar уменьшается ПОСЛЕ проверки на while(). Если бы мы использовали –numChar, значение в numChar будет уменьшено (из него вычтена единица) перед оценкой. В нашем случае цикл while проверяет значение numChar, а затем вычитает из него единицу. Если значение numChar не было нулем до декремента, он выполняет код в своем блоке кода. numChar устанавливается равным длине текстовой строки, которую мы ввели в окно последовательного монитора. Таким образом, код в цикле while будет выполняться столько раз. Код в цикле while:

```
buffer[index++] = Serial.read();
```

Он устанавливает каждый элемент буферного массива для каждого символа, считываемого из последовательной линии. Другими словами, он заполняет буферный массив буквами, которые мы ввели в текстовое окно МПП. Команда Serial.read () считывает входящие последовательные данные по одному байту за раз. Итак, теперь, когда наш массив символов был заполнен символами, которые мы ввели в последовательный монитор, цикл while завершится, когда numChar достигнет нуля (то есть длины строки).

После цикла while у нас есть

```
splitString(buffer);
```

Это вызов созданной нами функции splitString (). Функция выглядит так:

```
void splitString(char* data) {
        Serial.print("Data entered: ");
        Serial.println(data);
        char* parameter;
        parameter = strtok (data, " ,");
        while (parameter != NULL) {
                setLED(parameter);
                parameter = strtok (NULL, " ,");
        }

        // Очистить текстовый и последовательный буферы
        for (int x=0; x<16; x++) {
                buffer[x]='\0';
        }
        Serial.flush();
}
```

Мы не планируем возвращать какие-либо данные из функции, поэтому для нее установлен тип данных void. Мы передаем функции один параметр, и это тип данных char, который мы назвали данными. Вместо того, чтобы передавать все элементы в массиве функции, C передает указатель (другими словами, место в памяти) на первый элемент массива. (Это называется передачей по ссылке, а не по значению.) Итак, наше объявление функции должно принимать аргумент-указатель. Мы сообщаем компилятору, что это переменная-указатель, добавляя звездочку * перед именем переменной.

Указатели в двух словах

Указатели - довольно сложная тема в C, поэтому мы не будем вдаваться в эти подробности. Если вам нужно знать больше, обратитесь к книге по программированию на C. Все, что вам сейчас нужно знать, это то, что, объявляя «данные» как указатель, это просто переменная, указывающая на другую переменную.

Тип в объявлении указателя сообщает, на какой тип данных указывает указатель.

char * mytext; объявляет mytext как указатель, который может указывать на данные типа char(символьные).

int *nextnumber; объявляет nextnumber как указатель, который может указывать на целые числа.

Указатели должны быть инициализированы, прежде чем их можно будет использовать. Мы можем установить указатель на местоположение другой переменной, массива или элемента массива, используя оператор & (адрес), чтобы получить адрес переменной или элемента, на который мы хотим указать. Мы можем присвоить одну переменную-указатель другой переменной-указателю того же типа. Тогда они оба укажут на одно и то же. Мы можем получить значение, хранящееся в той ячейке памяти, на которую в настоящий момент указывает указатель, используя оператор * (разыменование). Например:

```
char mychar;       // объявляет простую переменную, которая может содержать символ.
char buff[] = {'a','b','c','d'}; // объявляет массив символов
char *mytext;   // объявляет переменную, которая может указывать на символьные данные
mytext = &buff[1];  //  инициализирует mytext, чтобы он указывал на расположение buff [1] в памяти;
mychar = *mytext; //  извлекает символ, на который указывает mytext (хранится в buff [1])
                  // и копирует этот символ (в данном случае 'b') в mychar.
```

Увеличение или уменьшение указателя заставляет его указывать на ячейку памяти, следующую или предшествующую местоположению, на которое он указывал. Поэтому, если он указывает на элемент массива, увеличение указателя заставляет его указывать на следующий элемент массива.

```
mytext++;              // mytext теперь указывает на buff [2];
```

Мы можем изменить содержимое памяти, на которую указывает указатель, разыменовав указатель в левой части оператора присваивания.

*mytext = 'g'; stores a 'g'в том месте, где mytext указывает на (buff[2], после строк выше).

Специальное значение (NULL) используется для представления недействительного (пустого) значения указателя. Если значение указателя равно NULL, попытки получить или сохранить значение, на которое он указывает, не определены (другими словами, ваша программа может перестать работать). Функции поиска часто возвращают значение указателя NULL, чтобы указать, что они не могут найти то, что вы просили их искать.

Поскольку указатели часто указывают на массивы, язык позволяет вам индексировать указатель так же, как массив.

mytext [1] относится к значению, хранящемуся в первом месте после того места, на которое в настоящий момент указывает mytext. Если mytext указывает на buff [2], mytext [1] относится к содержимому buff [2 + 1] (buff [3]), который в данном случае содержит 'd'. mytext [0] совпадает с * mytext.

Когда мы вызывали splitString, мы отправляли ему содержимое «buffer - буфера» (фактически указатель на него, как мы видели выше).

```
splitString(buffer);
```

Итак, мы вызвали функцию и передали ей (по ссылке) все содержимое массива символов буфера.
Первая команда

```
Serial.print("Data entered: ");
```

и это наш способ отправки данных обратно с Arduino на ПК. В этом случае команда печати отправляет все, что находится в скобках, на ПК через USB-кабель, где мы можем прочитать это в окне последовательного монитора. Мы отправили слова «Data entered:Введенные данные:». Текст должен быть заключен в кавычки «».
Следующая строка похожа

```
Serial.println(data);
```

и снова мы отправили данные обратно на ПК; на этот раз мы отправляем переменную-указатель на символ с именем data.Переменная символьного указателя, которую мы назвали «data», указывает на содержимое символьного массива «buffer», который мы передали функции. Итак, если наша введенная текстовая строка была

```
R255 G127 B56
```

Тогда

```
Serial.println(data);
```

Команда отправит эту текстовую строку обратно на ПК и распечатает ее в окне последовательного монитора.На этотраз команда печати имеет в конце ln, чтобы сделать ее println. Это просто означает «печать», а затем переход кследующей строке.
Когда мы печатаем с помощью команды print, курсор (точка, в которой появится следующий символ) остается в концевсего, что мы напечатали. Когда мы используем команду println, за нашим текстом следуют возврат каретки и перевод строки, заставляя курсор опускаться на следующую строку после того, как наш текст будет напечатан.

```
Serial.print("Data entered: ");
Serial.println(data);
```

Если мы посмотрим на наши две команды печати, первая напечатает «Data entered: », а затем курсор останется в конце этого текста. Следующая команда печати напечатает «data» или, другими словами,содержимое массива,называемого «буфером», а затем выдаст перевод строки или опустит курсор на следующую строку.
Если после этого мы выполним еще один оператор print или println, все, что будет напечатано в окне МПП,появится наследующей строке под последней.
Затем мы создаем новую переменную-указатель char, называемую параметром

```
Char* parameter;
```

и поскольку мы собираемся использовать эту переменную для доступа к элементам массива «данных», она должна быть того же типа, отсюда и символ *. Вы не можете передавать данные из переменной одного типа данных в другую, поскольку данные должны быть сначала преобразованы. Эта переменная - еще один пример переменной, имеющей «локальную область видимости». Его можно «увидеть» только по коду внутри этой функции. Если вы попытаетесь получить доступ к переменной параметра вне функции splitString, вы получите сообщение об ошибке.
Затем мы используем команду strtok, которая является очень полезной командой и позволяет нам манипулировать текстовыми строками.
Strtok получил свое название от String и Token, поскольку его цель - разделить строку с помощью разделителей. В нашем случае искомый разделитель - это пробел или запятая. Он используется для разделения текстовых строк на более мелкие строки, называемые токенами.
Мы передаем массив «data» команде strtok в качестве первого аргумента, а разделители (заключенные в кавычки) - в качестве второго аргумента.
Следовательно

```
parameter = strtok (data, " ,");
```

И в этой точке строка разбивается. Таким образом, мы используем его, чтобы установить «parameter» (параметр) как часть строки с точностью до пробела или запятой.

Итак, если бы наша текстовая строка была

```
R127 G56 B98
```

Тогда после этого оператора значение «parameter» будет

```
R127
```

поскольку команда strtok разбила бы строку до первого появления пробела запятой.

После того, как мы установили переменную «параметр» для той части текстовой строки, которую мы хотим вырезать (т. е. до первого пробела или запятой), мы затем вводим цикл while, условием которого является то, что параметр не является пустым (т.е. мы не достигли конца строки), используя

```
while (parameter != NULL) {
```

Внутри цикла мы вызываем нашу вторую функцию

```
setLED(parameter);
```

Мы рассмотрим ее позже. Затем он устанавливает переменную «parameter» в следующую часть строки до следующего пробела или запятой. Мы делаем это, передавая strtok параметр NULL

```
parameter = strtok (NULL, " ,");
```

Это указывает команде strtok продолжить работу с того места, где она остановилась в последний раз.

Итак, вся эта часть функции

```
char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
        setLED(parameter);
        parameter = strtok (NULL, " ,");
}
```

просто удаляет каждую часть текстовой строки, разделенную пробелами или запятыми, и отправляет эту часть строки следующей функции, называемой setLED ().

Последняя часть этой функции просто заполняет буферный массив символом NULL, что делается с помощью символа \ 0, а затем сбрасывает последовательные данные из последовательного буфера, который затем готов для ввода следующего набора данных.

```
// Clear the text and serial buffers
for (int x=0; x<16; x++) {
        buffer[x]='\0';
}
while(Serial.available())
 Serial.read();
```

Функция setLED будет брать каждую часть текстовой строки и устанавливать для соответствующего светодиода выбранный нами цвет. Итак, если вводимая нами текстовая строка

```
G125 B55
```

затем функция splitString () разбивает это на два отдельных компонента

```
G125
B55
```

и отправляет эту сокращенную текстовую строку в функцию setLED (), которая ее прочитает, решит, какой светодиод мы выбрали, и установит для него соответствующее значение яркости.

Итак, давайте взглянем на вторую функцию, которая называется setLED ().

```
void setLED(char* data) {
        if ((data[0] == 'r') || (data[0] == 'R')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(RedPin, Ans);
                Serial.print("Red is set to: ");
                Serial.println(Ans);
        }
        if ((data[0] == 'g') || (data[0] == 'G')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(GreenPin, Ans);
                Serial.print("Green is set to: ");
                Serial.println(Ans);
        }
        if ((data[0] == 'b') || (data[0] == 'B')) {
                int Ans = strtol(data+1, NULL, 10);
                Ans = constrain(Ans,0,255);
                analogWrite(BluePin, Ans);
                Serial.print("Blue is set to: ");
                Serial.println(Ans);
        }
}
```

Мы видим, что эта функция содержит три очень похожих оператора if. Поэтому мы рассмотрим только один из них, так как два других почти идентичны.

```
if ((data[0] == 'r') || (data[0] == 'R')) {
    int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
    analogWrite(RedPin, Ans);
    Serial.print("Red is set to: ");
    Serial.println(Ans);
}
```

Оператор if проверяет, является ли первый символ в строке (data [0]) буквой r или R (символы верхнего и нижнего регистра совершенно разные, если речь идет о C). Мы используем команду логического ИЛИ с символом || чтобы проверить, является ли буква буквой r OR (ИЛИ) R, так как можно вводить r как в верхнем, так и в нижнем регистре.

Если это r или R, тогда оператор if знает, что мы хотим изменить яркость красного светодиода, и поэтому код внутри выполняется. Сначала мы объявляем целое число с именем Ans (которое имеет локальную область видимости только для функции setLED) и используем команду strtol (строка в длинное целое число) для преобразования символов после буквы R в целое число.

Команда strtol принимает три параметра. Это строка, которую мы ему передаем, указатель на переменную, где strtol () может сохранять символ после числовой строки (которую мы не используем, поскольку мы уже удалили строку с помощью команды strtok и, следовательно, передаем указатель NULL), а затем «base» (основание), которое в нашем случае является основанием 10. Строка, которую мы ему передаем, будет содержать десятичные цифры (в отличие от двоичных, восьмеричных или шестнадцатеричных цифр, которые будут основанием 2, 8 и 16). соответственно). Другими словами, мы объявляем целое число и устанавливаем его в значение текстовой строки после буквы R (или ее числовой части).

Затем мы используем команду ограничения, чтобы убедиться, что Ans идет от 0 до 255 и не более. Затем мы выполняем команду analogWrite для красного LED и отправляем ему значение Ans. Затем код отправляет сообщение «Red is set to:» (Красный установлен на:), за которым следует значение Ans обратно на МПП. Два других оператора if делают то же самое, но для зеленого и синего светодиодов.

В этом проекте мы рассмотрели много вопросов и много новых понятий. Чтобы убедиться, что вы точно понимаете, что происходит в этом коде, я собираюсь установить код проекта рядом с псевдокодом (компьютерный язык, переведенный на язык, понятный людям). См. Таблицу 3-7.

Таб. 3-7. *Объяснение кода в проекте 10 с использованием псевдокода*

Язык программирования C	Псевдокод
//Проект 10 - RGB-лампа с последов. управлением	Комментарий с номером и названием проекта
char buffer[18];	Объявить символьный массив из 18 букв
int red, green, blue;	Объявите 3 целых числа, которые называются красным, зеленым и синим
int RedPin = 11;	An integer for which pin to use for red LED
int GreenPin = 10;	" " Зеленый
int BluePin = 9;	" " Синий
void setup()	Функция настройки
{	
Serial.begin(9600);	Установите последовательную связь для
Serial.flush();	работы со скоростью 9600 символов в
pinMode(RedPin, OUTPUT);	секундуFlush the serial line
	Установите вывод красного LED в качестве вывода
pinMode(GreenPin, OUTPUT);	
pinMode(BluePin, OUTPUT);	То же самое для зеленого
}	И синий
void loop()	Основной цикл программы
{	
if (Serial.available() > 0) {	Если данные отправляются по последовательной линии...
int index=0;	Объявить целое число с именем index и установить в 0
delay(100); // let the buffer fill up	Подождите 100 миллисекунд
int numChar = Serial.available();	Установите numChar на длину входящих данных из последовательного

(*продолжение*)

Таб. 3-7. (*продолжение*)

Язык программирования C	Псевдокод
if (numChar>15) {	Если numchar больше 15 символов...
numChar=15;	Сделайте 15 и не более
}	
while (numChar--) {	{Пока numChar не равно нулю (вычтите из него 1)
buffer[index++] = Serial.read();	Установить элемент [index] на считываемое значение (добавить 1 в индекс)
}	
splitString(buffer);	Вызвать функцию splitString и отправить ей данные в буфер
}	
}	
void splitString(char* data) {	Функция splitString ссылается на данные буфера
Serial.print("Data entered: ");	Распечатать «Введенные данные:»
Serial.println(data);	Распечатайте значение данных, а затем выпустите строку
char* parameter;	Объявить параметр типа данных char
parameter = strtok (data, " ,");	Установите текст до первого пробела или запятой
while (parameter != NULL)	Пока содержимое параметра не пустое ..
{setLED(parameter);	Вызов функции setLED
parameter = strtok (NULL, " ,");	Установить параметр в следующую часть текстовой строки
}	
// Clear the text and serial buffers	Другой комментарий
for (int x=0; x<16; x++) {	Следующую строчку сделаем 16 раз
buffer[x]='\0';	Установите каждый элемент буфера в NULL (пустой)
}	
Serial.flush();	Очистите серийный порт
}	
void setLED(char* data) {	Функция splitString ссылается на данные буфера
if ((data[0] == 'r') || (data[0] == 'R')) {	Если первая буква r или R…
int Ans = strtol(data+1, NULL, 10);	Установить целое число Ans в число в следующей части текста
Ans = constrain(Ans,0,255);	Убедитесь, что это значение от 0 до 255.
analogWrite(RedPin, Ans);	Запишите это значение на красный пин

(продолжение)

Таб. 3-7. (*продолжение*)

Язык программирования C	Псевдокод
Serial.print("Red is set to: ");	Распечатайте «Красный установлен на:»
Serial.println(Ans);	И тогда значение Ans
}	
if ((data[0] == 'g') || (data[0] == 'G')) {	Если первая буква g или G...
int Ans = strtol(data+1, NULL, 10);	Установить целое число Ans в число в следующей части текста
Ans = constrain(Ans,0,255);	Убедитесь, что это значение от 0 до 255.
analogWrite(GreenPin, Ans);	Запишите это значение на зеленый пин
Serial.print("Green is set to: ");	Распечатайте «Зеленый установлен на:»
Serial.println(Ans);	И тогда значение Ans
}	
if ((data[0] == 'b') || (data[0] == 'B')) {	Если первая буква *b* или *B*...
int Ans = strtol(data+1, NULL, 10);	Установить целое число Ans в число в следующей части текста
Ans = constrain(Ans,0,255);	Убедитесь, что это значение от 0 до 255.
analogWrite(BluePin, Ans);	Запишите это значение на синий пин
Serial.print("Blue is set to: ");	Распечатайте «Синий установлен на:»
Serial.println(Ans);	И тогда значение Ans
}	
}	

Надеюсь, вы сможете использовать этот «псевдокод», чтобы точно понимать, что происходит в коде этого проекта.Теперь мы на некоторое время оставим светодиоды и посмотрим, как создавать звуки из нашего Arduino с помощью пьезозуммера.

Резюме

В главе 3 представлено много новых команд и понятий программирования. Вы узнали о массивах и о том, каких использовать, как считывать аналоговые значения с вывода, как использовать выводы ШИМ и основы последовательной связи.
Знание того, как отправлять и читать данные по последовательной линии, означает, что вы можете использовать Arduino для связи со всеми типами последовательных устройств и других устройств с помощью простых протоколов связи. Позже в этой книге вы вернетесь к последовательной связи.

Предметы и понятия, рассматриваемые в главе 3:

- Массивы и способы их использования
- такое потенциометр (или переменный резистор) и как им пользоваться
- Считывание значений напряжения с аналогового входного вывода
- Как использовать функцию математического синуса (sin)
- Преобразование градусов в радианы
- Концепция приведения переменной к другому типу
- Широтно-импульсная модуляция (ШИМ) и как ее использовать с `analogWrite ()`
- Создание цветных источников света с использованием различных значений RGB
- Генерация случайных чисел с помощью `random ()` и `randomSeed ()`
- Как различные световые эффекты могут быть созданы с помощью одной и той же схемы, но разного кода
- Концепция последовательной связи
- Установка скорости последовательной передачи данных с помощью `Serial.begin ()`
- Отправка команд с помощью МПП
- Использование массива для создания текстовых строк
- Проверка передачи данных по последовательной линии с помощью `Serial.available`
- Создание цикла при выполнении условия с помощью команды while ()
- Чтение данных из последовательной линии с помощью `Serial.read ()`
- Основная концепция указателей
- Отправка данных на МПП с помощью `Serial.print ()` или `Serial.println ()`
- Управление текстовыми строками с помощью функции strtok ()
- Преобразование строки в длинное целое число с помощью `strtol ()`
- Ограничение значения переменных с помощью функции constrain ()

ГЛАВА 4

Простые оповещатели и датчики

Теперь мы собираемся заставить нашу Arduino издавать несколько простых звуков с помощью пьезозуммера. Возможность издавать звуки с помощью Arduino - отличный способ добавить сигналы тревоги, предупреждающие звуковые сигналы для устройств, которое мы будем создавать. Начиная с версии 0018 IDE Arduino, была добавлена новая команда, позволяющая легко создавать тональные сигналы. Мы также узнаем, как использовать пьезоэлектрический датчик и научимся считывать с него данные. Наконец, мы рассмотрим датчик освещенности. Вы узнаете, как издавать простые звуки, и по ходу освоите основы считывания показаний аналоговых датчиков.

Мы узнаем, как использовать команду tone () в Проекте 11, создав простую сигнализацию, похожую на автомобильную.

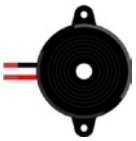
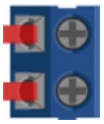
Проект 11 - Сигнализация с пьезозуммером

Мы используем пьезозуммер для подачи звукового сигнала тревоги,по принципу, который мы использовали в Проекте 7, чтобы сделать пульсирующую лампу. Но на этот раз мы заменим светодиод на пьезозуммер или пьезодиск. Полный список необходимых деталей можно найти в Таблице 4-1.

Выбирая пьезозуммер, убедитесь, что это не активный зуммер, то есть зуммер со встроенным генератором. Нам нужен пассивный пьезозуммер, который использует переменное напряжение, а не постоянное.

Требуемые детали

Таб. 4-1. *Требуемые детали для проекта 11*

Пьезозуммер (или пьезодиск)	
2-х винтовой зажим	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Теперь возьмите пьезозуммер и зажмите его провода в винтовой терминал. Подключите винтовой зажим к макетной плате и подключите его к Arduino, как показано на рисунке 4-1. Затем снова подключите Arduino к USB-кабелю и включите его.

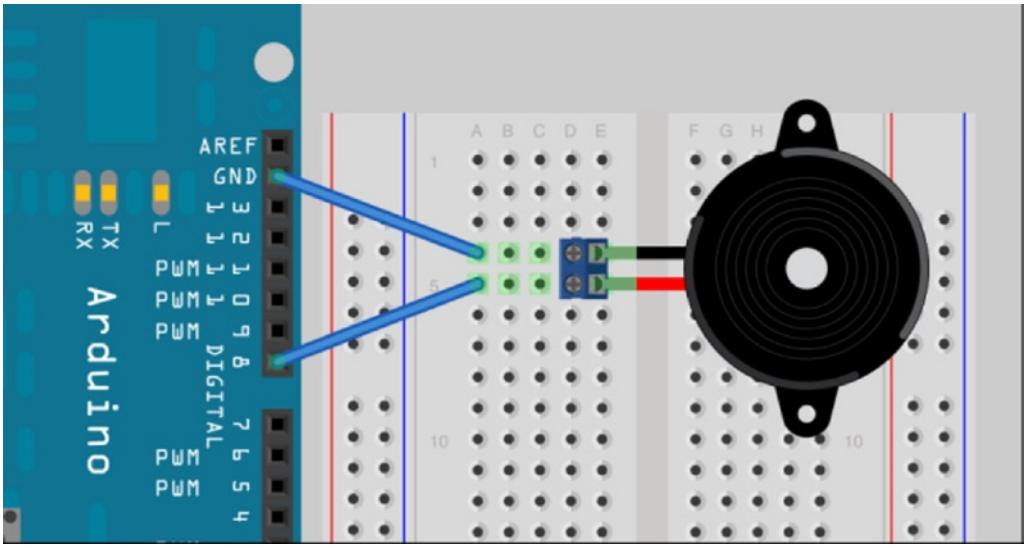


Рис. 4-1. *Схема по Проекту 11 - Пьезозвуковая сигнализация*

Введите код

Откройте IDE Arduino и введите код из листинга 4-1:

Листинг 4-1. Код для проекта 11

```
// Проект 11 - Пьезозвуковая сигнализация


float sinVal;
int toneVal;

void setup() {
        pinMode(8, OUTPUT);
}

void loop() {
        for (int x=0; x<180; x++) {
                // преобразовать градусы в радианы, затем получить значение sin
                sinVal = (sin(x*(3.1412/180)));
```

```
                // генерировать частоту из значения sin
                toneVal = 2000+(int(sinVal*1000));
                tone(8, toneVal);
                delay(2);
        }
}
```

После того, как вы загрузили код, произойдет небольшая задержка, а затем пьезо начнет издавать звуки. Если все работает по плану, вы услышите повышающуюся и понижающую сигнализацию типа сирены, похожую на автомобильную сигнализацию. Код для проекта 11 почти идентичен коду для проекта 7, поэтому давайте посмотрим, как он работает.

Проект 11 - Пьезозвуковая сигнализация - Обзор кодов

Сначала мы устанавливаем две переменные

```
float sinVal;
int toneVal;
```

Переменная sinVal float будет содержать значение синуса, которое будет вызывать повышение и понижение тона так же, как пульсирующая лампа в Project 7. Переменная toneVal будет использоваться для преобразования значения sinVal в нужную нам частоту.

В функции настройки мы теперь устанавливаем цифровой вывод 8 на выход.

```
void setup() {
        pinMode(8, OUTPUT);
}
```

Затем переходим к основному циклу. Как и в Project 7, мы устанавливаем цикл for от 0 до 179, чтобы гарантировать, что значение синуса не станет отрицательным.

```
for (int x=0; x<180; x++) {
```

Преобразуем значение x в радианы, как в Проекте 7:

```
sinVal = (sin(x*(3.1412/180)));
```

Затем это значение преобразуется в частоту, подходящую для звукового сигнала.

```
toneVal = 2000+(int(sinVal*1000));
```

Берем 2000 и складываем sinVal, умноженный на 1000. Это дает нам хороший диапазон частот для нарастающего и падающего тона.

Затем мы используем команду tone () для генерации частоты на пьезоэхолоте.

```
tone(8, toneVal);
```

Для команды tone () требуются два или три параметра, таким образом:

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Пин - это цифровой вывод, используемый для вывода на пьезо, а частота - это частота тона в герцах. Существует также необязательный параметр длительности в миллисекундах для длины тона. Если длительность не указана, звук будет продолжать воспроизводиться до тех пор, пока вы не сыграете другой тон или пока не воспользуетесь командой noTone (pin), чтобы прекратить генерацию звука на указанном пине.

Наконец, мы запускаем задержку в 2 миллисекунды между изменениями частоты, чтобы гарантировать, что синусоидальный сигнал нарастает и падает с требуемой скоростью.

```
delay(2);
```

Вам может быть интересно, почему мы не поместили 2 миллисекунды в параметр длительности команды tone (), например:

```
tone(8, toneVal, 2);
```

Это связано с тем, что наш цикл for настолько короткий, что он в любом случае изменит частоту менее чем за 2 миллисекунды, что делает параметр длительности бесполезным. Следовательно, после генерации тона устанавливается задержка в 2 миллисекунды, чтобы гарантировать, что он будет воспроизводиться не менее 2 миллисекунд, прежде чем цикл for повторится и тон снова изменится.

Мы используем этот принцип генерации сигналов тревоги позже, когда узнаете, как подключать датчикик Arduino.Затем мы можем активировать сигнал тревоги при достижении порогового значения датчика, например, если кто-то подойдет слишком близко к ультразвуковому извещателю или если температура станет слишком высокой.

Если вы измените значения 2,000 и 1,000 в расчете toneVal и длительность задержки, вы можете сгенерировать разные звуковые сигналы. Поэкспериментируйте и посмотрите, какие звуки будут издаваться.

Проект 11 - Пьезозвуковая сигнализация - Обзор устройства

В этом проекте мы использовали два новых компонента, винтовой зажим и пьезозуммер. Мы использовали винтовой зажим, поскольку провода от пьезозуммера или диска слишком тонкие и мягкие, чтобы их можно было вставить в макетную плату(см. рис.). Винтовой зажим имеет выводы (штыри), чтобы вы могли вставить его вмакетную плату.

Другой новый компонент - пьезозуммер или пьезодиск (см. Рис. 4-2).

Пьезопищалка конструктивно представлена металлической пластиной с нанесенным на нее напылением из токопроводящей керамики. Пластина и напыление выступают в роли контактов. При подаче напряжения на зуммер он начинает деформироваться. При этом происходят удары о металлическую пластинку, которая и производит “шум” нужной частоты.

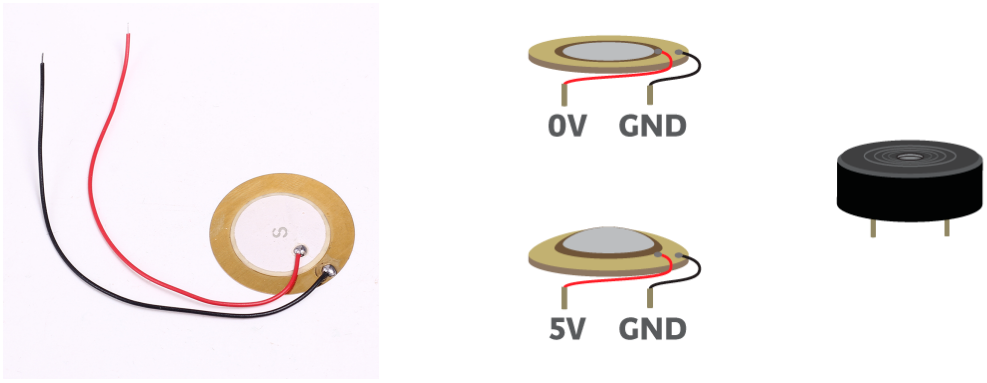
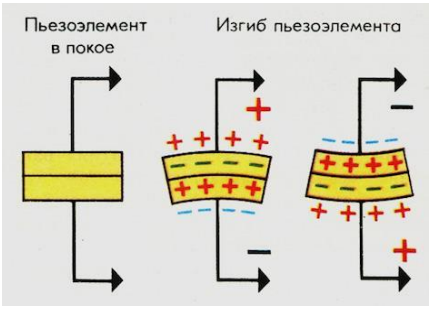
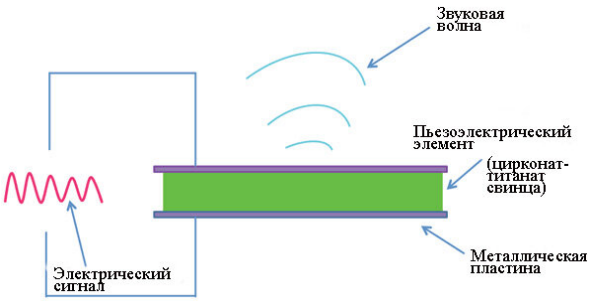


Рис. 4-2. Пьезодиск





Пьезоэлектрические материалы, особенно кристаллы и определенная керамика, обладают способностью производить электричество при приложении к ним механического воздействия (см. рис.). Эффект находит полезныеприменения в производстве и обнаружении звука, генерации высоких напряжений, генерации электронныхчастот, микровесов и сверхтонкой фокусировки оптических сборок.

Эффект также обратим: если электричество в форме напряжения приложить к пьезоэлектрическому материалу, это приведет к изменению формы материала (в некоторых случаях на 0,1%).

Чтобы воспроизвести звук от пьезодиска,напряжение с какой-то частотой,начиная со 100 Гц заставляет материал изгибаться (имеется в виду керамика) и взаимодействовать с металлической пластиной, вызывая «щелчок».

Из-за изменения частоты импульсов диск будет изгибаться сотни или тысячи раз в секунду, вызывая жужжащий звук. Изменяя частоту щелчков и время между ними, можно создавать определенные звуки.

Вы также можете использовать способность пьезо создавать потенциал напряжения для измерения движения или вибрации. Фактически, пьезодиски используются в качестве контактных микрофонов для гитар или ударных. Мы будем использовать эту особенность пьезодиска в Проекте 13, когда будем делать датчик детонации. Прежде чем мы перейдем к этому, мы сделаем еще один проект с пьезоэлементом на выходе.

Проект 12 – Пьезозуммер - проигрыватель мелодии

Вместо того, чтобы использовать пьезозуммер для воспроизведения раздражающих звуковых сигналов, как насчет того, чтобы использовать его для воспроизведения мелодии? Мы собираемся заставить нашу Arduino играть «Oh My Darling Clementine». Оставьте схему в точности такой, как это было в Project 11. Мы просто изменим код.

Введите код

Откройте IDE Arduino и введите код из листинга 4-2:

Listing 4-2. Код для проекта 12

```
// Проект 12 - Пьезозуммер - проигрыватель мелодии

#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494

#define WHOLE 1
#define HALF 0.5
#define QUARTER 0.25
#define EIGHTH 0.125
#define SIXTEENTH 0.0625

int tune[] = {
  NOTE_F3, NOTE_F3, NOTE_F3,  NOTE_C3,
  NOTE_A3, NOTE_A3, NOTE_A3,  NOTE_F3,
  NOTE_F3, NOTE_A3, NOTE_C4,  NOTE_C4,  NOTE_AS3, NOTE_A3, NOTE_G3,
  NOTE_G3, NOTE_A3, NOTE_AS3, NOTE_AS3,
  NOTE_A3, NOTE_G3, NOTE_A3,  NOTE_F3,
  NOTE_F3, NOTE_A3, NOTE_G3,  NOTE_C3,  NOTE_E3,  NOTE_G3, NOTE_F3
};

float duration[] = {
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER+EIGHTH, EIGHTH, EIGHTH, EIGHTH, HALF,
  EIGHTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER+EIGHTH, EIGHTH, EIGHTH, SIXTEENTH, HALF
};
```

```
int length;

void setup() {
        pinMode(8, OUTPUT);
        length = sizeof(tune) / sizeof(tune[0]);
}

void loop() {
        for (int x=0; x<length; x++) {
                tone(8, tune[x]);
                delay(1500 * duration[x]);
                noTone(8);
        }
        delay(5000);
}
```

После того, как вы загрузите код, произойдет небольшая задержка, и пьезозуммер начнет играть мелодию. Вы можете узнать это как «О, моя дорогая Клементина». В этом проекте задействовано несколько новых концепций, так что давайте рассмотрим их.

Проект 12 - Пьезозуммер - проигрыватель мелодии - Обзор кода

Первое, что вы видите, глядя на код Project 12, - это длинный список директив define. Директива define - это что-то новое для нас, но при этом она очень проста и очень полезна. #define просто определяет токен (имя) и его значение.

Например, рассмотрим следующее:

```
#define PI 3.14159265358979323846264338327950288419716939937510
```

Это позволит вам заменять Пи в любом вычислении вместо того, чтобы вводить Пи с точностью до 50 десятичных знаков.Вот еще несколько примеров:

```
#define TRUE  1
#define FALSE 0
```

Этот код означает, что вы можете указать в коде TRUE или FALSE вместо 0 или 1. Это делает логические утверждения намного более понятными для чтения человеком.

Предположим, вы написали код для отображения фигуры на светодиодном точечно-матричном дисплее, и разрешение дисплея составляет 8 x 32. Вы можете создать директивы определения для высоты и ширины дисплея, таким образом:

```
#define DISPLAY_HEIGHT 8
#define DISPLAY_WIDTH 32
```

Теперь, когда вы ссылаетесь на высоту и ширину дисплея в своем коде, вы можете использовать DISPLAY_HEIGHT и DISPLAY_WIDTH вместо чисел 8 и 32.

Это дает три основных преимущества вместо простого использования чисел.

- Во-первых, код теперь становится намного проще для понимания, поскольку вы изменили значения высоты и ширины дисплея на токены, которые делают эти числа более понятными для третьих лиц.
- Во-вторых, если вы позже примените другой дисплей , скажем, дисплей 16 x 64, все, что вам нужно будет сделать, это изменить два значения в директивах define вместо того,чтобы изменять числа, которые могут быть сотнями строк кода, в которых отображается разрешение дисплея. Изменяя значения в директиве define в начале программы, новые значения автоматически используются во всем остальном коде.

- В-третьих, с #defines третьему лицу, читающему код, больше не нужно знать особенности прикрепленного дисплея, чтобы распознать, какие части кода имеют дело с шириной и высотой дисплея.

В случае Проекта 12 мы создаем целый набор директив define, в которых токены - это ноты от C3 до B4, а значения - это частоты, необходимые для создания этой ноты. Первая нота нашей мелодии - F3, а соответствующая ей частота - 173 герца.

Это средняя фа на музыкальной шкале. Не все определенные ноты используются в нашей мелодии, но я включил их на тот случай, если вы захотите написать свою собственную мелодию.

Следующие пять директив определяют длину нот. Ноты могут быть целым тактом, половиной, четвертью, восьмой или шестнадцатой долей такта. Числа мы будем использовать для умножения длины полосы в миллисекундах, чтобы получить длину каждой ноты. Например, четвертная нота равна 0,25 (или четверть единицы), и поэтому мы умножаем длину такта, в нашем случае 1500 миллисекунд, на 0,25, чтобы получить длину четвертной ноты.

```
1500 x QUARTER = 375 milliseconds
```

Затем мы определяем целочисленный массив с именем tune [] и заполняем его нотами для «Oh My Darling Clementine».

```
int tune[] = {
  NOTE_F3, NOTE_F3, NOTE_F3,  NOTE_C3,
  NOTE_A3, NOTE_A3, NOTE_A3,  NOTE_F3,
  NOTE_F3, NOTE_A3, NOTE_C4,  NOTE_C4, NOTE_AS3, NOTE_A3, NOTE_G3,
  NOTE_G3, NOTE_A3, NOTE_AS3, NOTE_AS3,
  NOTE_A3, NOTE_G3, NOTE_A3,  NOTE_F3,
  NOTE_F3, NOTE_A3, NOTE_G3,  NOTE_C3, NOTE_E3,  NOTE_G3, NOTE_F3
};
```

После этого мы создаем еще один массив, на этот раз плавающий, который будет содержать продолжительность каждой ноты во время ее воспроизведения.

```
float duration[] = {
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER+EIGHTH, EIGHTH, EIGHTH, EIGHTH, HALF,
  EIGHTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER, QUARTER,
  EIGHTH+SIXTEENTH, SIXTEENTH, QUARTER+EIGHTH, EIGHTH, EIGHTH, SIXTEENTH, HALF
};
```

Как вы можете видеть, глядя на оба этих массива, использование директив `define` для определения нот и длины нот значительно упрощает чтение и понимание массива, чем если бы он был заполнен серией чисел. Затем мы создаем целое число, называемое длиной

```
int length;
```

Она будет использоваться для вычисления и сохранения длины массива (т. е. количества нот в мелодии).

В нашей программе настройки мы сначала устанавливаем цифровой вывод 8 на выход

```
pinMode(8, OUTPUT);
```

затем инициализируем целочисленную длину количеством нот в массиве

```
length = sizeof(tune) / sizeof(tune[0]);
```

и это делается с помощью функции sizeof ().

Функция sizeof () возвращает количество байтов в переданном ей параметре. В Arduino целое число состоит из двух байтов. Байт состоит из 8 бит. Это касается области двоичной арифметики, и для этого проекта нам не нужно беспокоиться о битах и байтах. Мы рассмотрим их позже в книге.

Итак, наша мелодия состоит из 26 нот. Итак, массив tunes [] состоит из 26 элементов. Чтобы вычислить это, мы получаем размер (в байтах) всего массива

```
sizeof(tune)
```

и разделим это на количество байтов в одном элементе.

```
sizeof(tune[0])
```

В нашем случае это эквивалентно

```
26 / 2 = 13
```

Если вы замените мелодию в проекте своей собственной, то длина будет рассчитана как количество нот в вашей мелодии.

Функция sizeof () полезна при вычислении длины различных типов данных и особенно полезна, если вы должны были перенести свой код на другое устройство, в котором длина типов данных отличается от длины типов данных на Arduino.

В основном цикле мы настраиваем цикл for, который повторяет, сколько раз в мелодии есть ноты.

```
for (int x=0; x<length; x++) {
```

затем мы сыграем следующую ноту в массиве tune [] на выводе 8

```
tone(8, tune[x]);
```

затем подождем соответствующее время, чтобы нота сыграла

```
delay(1500 * duration[x]);
```

Задержка составляет 1500 миллисекунд, умноженных на длину ноты (например, 0,25 для четвертой ноты, 0,125 для восьмой ноты и т. д.).

Перед тем, как сыграть следующую ноту, мы приостанавливаем звук, генерируемый на выводе 8.

```
noTone(8);
```

Это необходимо для того, чтобы при одновременном воспроизведении двух одинаковых нот их можно было различить как отдельные ноты. Без функции noTone () ноты вместо этого сливались бы в одну длинную ноту.

Наконец, после завершения цикла for мы запускаем 5-секундную задержку перед повторением мелодии.

```
delay(5000);
```

Чтобы создать ноты для этой мелодии, я использовал общедоступные ноты для песни «Oh My Darling Clementine» в Интернете и ввел ноты в массив tune [], а затем длины нот в массиве duration []. Обратите внимание, что я добавил длину нот, чтобы получить ноты, разделенные точками (например, ЧЕТВЕРТЬ + ВОСЬМАЯ). Сделав что-то подобное, вы можете создать любую мелодию, какую захотите.

Если вы хотите ускорить или замедлить темп мелодии, измените значение 1500 в функции задержки на большее или меньшее

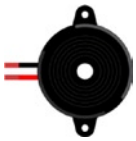
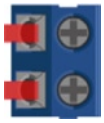
Теперь мы используем пьезодиск для другой цели, а именно его способности производить разность потенциалов, когда диск сжимается или ударяется. Используя эту функцию, мы сделаем датчик детонации в Проекте 13.

Проект 13 - пьезодатчик детонации

Пьезодиск работает, когда электрическое поле (напряжение) прикладывается к керамическому материалу в диске, заставляя его менять форму и, следовательно, издавать звук (щелчок). Диск также работает в обратном направлении: когда диск ударяется или сжимается, сила, действующая на материал, вызывает генерацию электрического заряда (напряжения). Мы можем считывать этот ток с помощью Arduino, и мы сделаем на этой основе датчик детонации.

Требуемые детали

Table 4-2. *Требуемые детали для проекта 13*

Пьезозуммер (или пьезодиск)	
2- винтовой зажим	
Светодиод 5мм (любой цвет)	
Резистор 1 Мегаом	
резистор 150 Ом	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Затем соедините детали так, чтобы получилась схема, показанная на Рисунке 4-3. Пьезодиск лучше подходит для этого проекта, чем пьезозуммер. Резистор на пьезоэлементе предназначен для разряда любой пьезоемкости, накопленной на нем во время его использования.

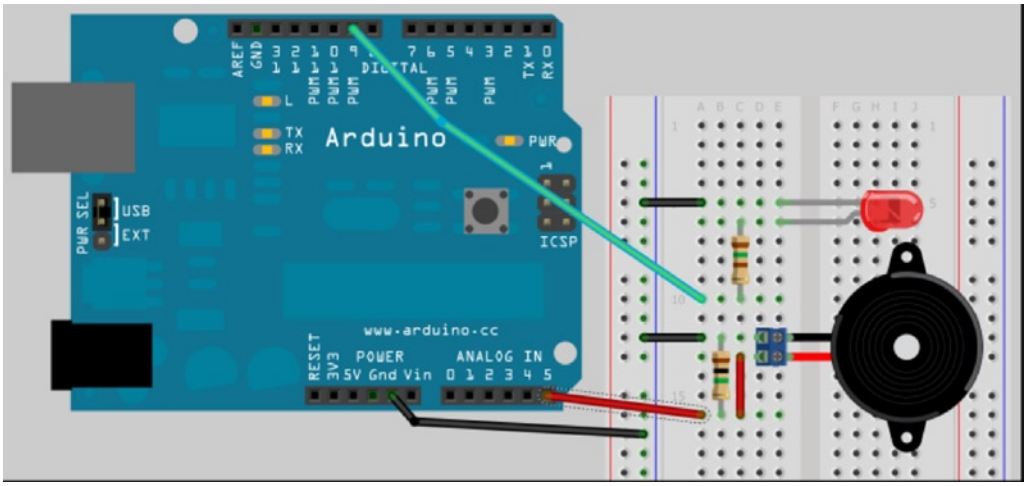


Рис. 4-3. *Схема для проекта 13 - Пьезодатчик детонации*

Введите код

Откройте IDE Arduino и введите код из листинга 4-3:

Listing 4-3. Код для проекта 13

```
// Проект 13 - Пьезоэлектрический датчик детонации

int ledPin = 9; // Светодиод на цифровом выводе 9
int piezoPin = 5; // Пьезодатчик на аналоговом выводе 5
int threshold = 120; // Значение датчика, которое нужно достичь перед активацией
int sensorValue = 0;  // Переменная для хранения значения, считанного с датчика
float ledValue = 0; // Яркость светодиода

void setup() {
        pinMode(ledPin, OUTPUT); // Устанавливаем ledPin на ВЫХОД
        // Дважды мигаем светодиодом, чтобы показать, что программа запущена
        digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
        digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
}

void loop() {
        sensorValue = analogRead(piezoPin);  // Считываем значение с датчика

  if (sensorValue >= threshold) { // Если детонация обнаружена, установить яркость на максимум
          ledValue = 255;
        }
        analogWrite(ledPin, int(ledValue) ); // Записываем значение яркости  светодиода
        ledValue = ledValue - 0.05; // Постепенно уменьшаем яркость светодиода
   if (ledValue <= 0) { ledValue = 0;}  // Убедимся, что значение не опускается ниже нуля
}
```

После того, как вы загрузили свой код, светодиод быстро мигнет дважды, показывая, что программа запущена.Теперь вы можете постучать по датчику (сначала положив его на поверхность) или сжать его пальцами. Каждый раз, когда Arduino обнаруживает стук или сжатие, светодиод загорается, а затем плавно гаснет. Пороговое значение в коде было установлено для пьезодиска, который я использовал при сборке проекта. Возможно, вам потребуется установить большее или меньшее значение в зависимости от типа и размера пьезоэлектрического преобразователя, который вы использовали в своем проекте. Нижний порог - более чувствительный, а верхний - менее чувствительный.

Проект 13 - Пьезоэлектрический датчик детонации - Обзор кода

Мы не получим никаких новых команд кода в этом проекте, но мы все равно рассмотрим, как это работает.

Сначала мы настраиваем необходимые переменные для нашей программы. Они говорят сами за себя.

```
int ledPin = 9; // Светодиод на цифровом выводе 9
int piezoPin = 5; // Пьезо на аналоговом выводе 5
int threshold = 120; // Значение датчика, которое нужно достичь перед активацией
int sensorValue = 0;  // Переменная для хранения значения, считанного с датчика
float ledValue = 0; // Яркость светодиода на нуле
```

В функции настройки ledPin устанавливается на выход, а светодиод быстро мигнет дважды, что является визуальным индикатором того, что программа начала работать.

```
void setup() {
        pinMode(ledPin, OUTPUT);
        digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
        digitalWrite(ledPin, HIGH); delay(150); digitalWrite(ledPin, LOW); delay(150);
}
```

В нашем основном цикле мы сначала считываем аналоговое значение с вывода 5, к которому подключен пьезоэлектрический преобразователь.

```
sensorValue = analogRead(piezoPin);
```

Затем код проверяет, больше ли это значение или равно (> =) порогу, который мы установили, то есть, чтобы определить, действительно ли это удар или сжатие (пьезоэлемент очень чувствителен, как вы можете увидеть, установив порог до очень низкого значения). Если это так, то он устанавливает для ledValue значение 255, что является максимальным напряжением на выводе 9 ШИМ.

```
if (sensorValue >= threshold) {
        ledValue = 255;
}
```

Затем мы записываем это значение на вывод 9 PWM. Поскольку ledValue является числом с плавающей запятой, мы «преобразуем» его в целое число, поскольку функция analogWrite может принимать только целое число, а не плавающее значение. (Приведение просто означает «преобразование в другой тип данных».)

```
analogWrite(ledPin, int(ledValue) );
```

Затем мы уменьшаем значение ledValue, которое является плавающим, на 0,05.

```
ledValue = ledValue - 0.05;
```

Мы хотим, чтобы светодиод плавно затемнялся, и поэтому мы используем значение с плавающей запятой для хранения значения яркости светодиода вместо целого числа. Таким образом, мы можем уменьшить его значение на небольшую величину (в нашем случае 0,05), что означает, что потребуется некоторое время, пока основной цикл повторится, чтобы значение ledValue достигло нуля, когда светодиод будет самым тусклым и выключенным. . Если вы хотите, чтобы светодиод гас медленнее или быстрее, увеличьте или уменьшите это значение.

Наконец, мы не хотим, чтобы ledValue опускался ниже нуля, поскольку вывод 9 ШИМ может выводить только значение от 0 до 255, поэтому мы проверяем, меньше ли оно или равно нулю, и если оно меньше, мы меняем его обратно на нуль.

```
if (ledValue <= 0) { ledValue = 0;}
```

Затем основной цикл повторяется, слегка уменьшая яркость светодиода каждый раз, пока светодиод не погаснет или не будет обнаружен другой стук, а яркость снова не будет установлена на максимум.

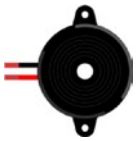
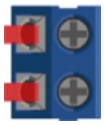
А теперь познакомимся с новым датчиком - светозависимый резистор или LDR.

Проект 14 - Датчик освещенности

Теперь мы рассмотрим новый компонент, известный как светозависимый резистор или LDR. Как следует из названия, устройство представляет собой резистор, сопротивление которого зависит от света. В темноте резистор будет иметь очень высокое сопротивление. Когда свет попадает на LDR, сопротивление уменьшается. Чем больше света, тем меньше будет сопротивление. Считывая значение с датчика, мы можем определить степень освещенности. В этом проекте мы будем использовать LDR для обнаружения света и пьезозуммер, который будет сигнализировать о количестве обнаруженного света.

Требуемые детали

Table 4-3. *Требуемые детали для проекта 14*

Пьезозуммер (или пьезодиск)	
2- винтовой терминал	
Светозависимый резистор	
Резистор 10 кОм	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Затем соедините детали так, чтобы получилась схема, показанная на Рисунке 4-4. Перед повторным подключением питания к Arduino проверьте все свои подключения.

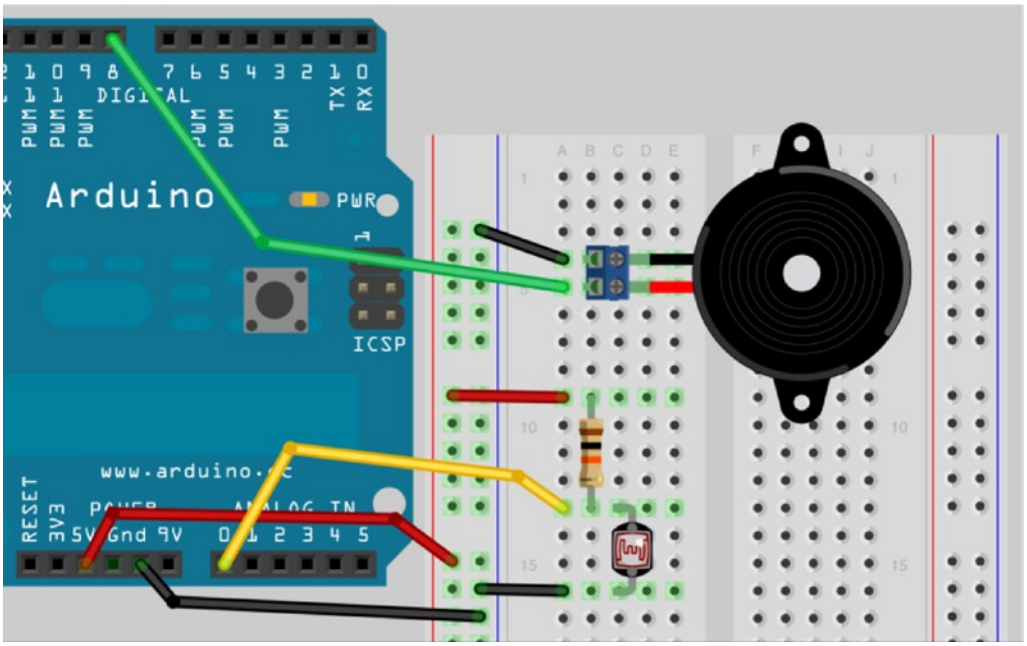


Рис. 4-4. *Схема для проекта 14 - Датчик освещенности*

LDR можно вставить любым способом; у него нет полярности. Я обнаружил, что резистор на 10 кОм хорошо работает с моим LDR. Возможно, вам придется подобрать резистор, который подойдет для вашего LDR. Значение может варьироваться от 1 до 10 кОм. Всегда полезно иметь набор резисторов разных номиналов в коробке с компонентами.

Введите код

Теперь запустите вашу Arduino IDE и введите короткий и простой код из листинга 4-4 ниже.

Листинг 4-4. . Код для проекта 14

```
// Проект 14 - Датчик освещенности

int piezoPin = 8;  // Пьезодатчик на выводе 8
int ldrPin = 0;   // LDR на аналоговом выводе 0
int ldrValue = 0;  // Значение читается из LDR

void setup() {
 // здесь нечего делать
}
```

```
void loop() {
        ldrValue = analogRead(ldrPin); // считываем значение из LDR
        tone(piezoPin,1000); // воспроизводим тон 1000 Гц от пьезозуммера
        delay(25);  // Подождем немного
        noTone(piezoPin);  // выключаем звук
           delay(ldrValue); // ждем количество миллисекунд в ldrValue
}
```

Когда вы загрузите этот код в Arduino, Arduino начнет издавать короткие звуковые сигналы. Паузы между звуковыми сигналами будут большими, если LDR находится в тени, и будут короткими, если его будет освещать яркий свет. Это даст эффект счетчика Гейгера, но с нашим детектором, обнаруживающим фотонные частицы вместо ионизирующего излучения.
Код для Проекта 14 очень прост, и вы сможете самостоятельно разобраться, как он работает, без посторонней помощи.
Однако мы рассмотрим, как работает LDR и почему важен дополнительный резистор.

Проект 14 - Датчик освещенности - Обзор устройства

Новым компонентом в этом проекте является светозависимый резистор (LDR), также известный как фотоэлемент или фоторезистор. LDR бывают разных форм и размеров (см. Рис. 4-5) и имеют разные диапазоны сопротивления.

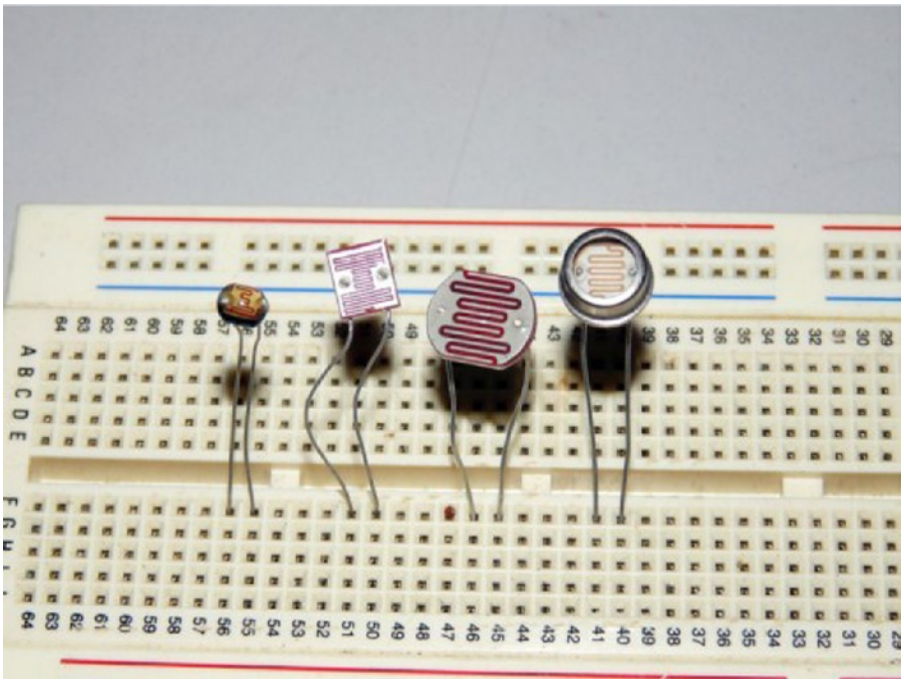


Рис. 4-5. *Различные виды LDR*

Новшество в нашей схеме - это делитель напряжения . Используя два последовательно подключенных резистора и снимая напряжение только с одного из них, мы можем уменьшить напряжение Vin. В нашем случае у нас есть резистор фиксированного значения (10 кОм или около того) и переменный резистор в форме LDR. Как оказалось, потенциометр, использованный в проекте 6, также был делителем напряжения. Давайте посмотрим на стандартный делитель напряжения с использованием резисторов R1 и R2 и посмотрим, как она работает (см.рис. 4-6) .

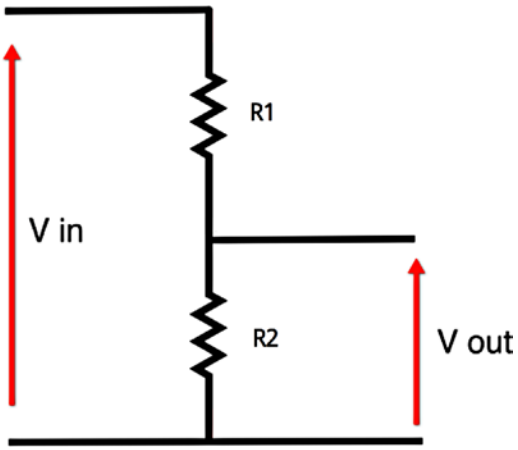


Рис. 4-6. *Делитель напряжения*

Здесь представлена формула делителя напряжения

$$Vout = \frac{R2}{R2 + R1} \times Vin$$

Итак, если бы в схеме на рисунке 4-6 у нас были резисторы 100 Ом (или 0,1 кОм) как для R1, так и для R2, и 5 В для входа в Vin, наша формула была бы

$$\frac{100}{100 + 100} \times 5 = 2.5 \text{ вольт}$$

Давайте сделаем это снова с резисторами 470 Ом.

$$\frac{470}{470 + 470} \times 5 = 2.5 \text{ вольт}$$

Другими словами, при равных R1 и R2 выходное напряжение Uout равно половине входного Uin

Таблица 4-4 показывает, какие напряжения мы получим из нашей цепи при изменении сопротивления.

Таб. 4-4. *Значения Vout для примера LDR с 5 В в качестве Vin*

Уровень освещенности	R1	R2 (LDR)	Vout
Самый темный	10kΩ	100kΩ	4.55v
25%	10kΩ	73kΩ	4.40v
50%	10kΩ	45kΩ	4.09v
75%	10kΩ	28kΩ	3.68v
Lightest	10kΩ	10kΩ	2.5v

Как видите, по мере увеличения яркости в помещении напряжение на Vout уменьшается. В результате значение, которое мы считываем на датчике, становится меньше, а задержка после звукового сигнала меньше, что приводит к повышению частоты звукового сигнала. Если бы вы поменяли местами резистор и LDR, напряжение увеличилось бы по мере того, как больше света попадало на LDR.

Резюме

В главе 4 вы узнали, как создавать музыку, звуки будильника, предупреждающие сигналы и т. д. с вашего Arduino. У этих звуков есть много полезных применений. Вы можете, например, сделать свой будильник. Наконец, используя LDR для обнаружения света, вы можете включать ночник, когда окружающий свет падает ниже определенного порога.

Предметы и понятия, затронутые в главе 4:

- Что такое пьезоэлектрический преобразователь и как он работает
- Как создавать звуки с помощью функции tone ()
- Как остановить генерацию тона с помощью функции noTone ()
- Команда • #define и то, как она упрощает отладку и понимание кода
- Получение размера массива (в байтах) с помощью функции sizeof ()
- Что такое LDR (светозависимый резистор), как он работает и как считывать с него значения
- Понятие о делителях напряжения и способы их использования

ГЛАВА 5

Управление двигателем DC

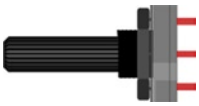
Пришло время перейти к управлению двигателем постоянного тока. Если вы когда-нибудь планируете построить робота или какое-либо устройство, которое требует использования двигателя, то знания, которые мы собираемся получить, будут иметь важное значение. Для управления двигателем требуются токи выше, чем Arduino может безопасно обеспечить со своих выходов, поэтому нам нужно будет использовать транзисторы и диоды, чтобы обеспечить достаточный ток для двигателя и диодов для защиты Arduino.
Обзор устройства объяснит, как они работают. В нашем первом проекте мы будем управлять двигателем, используя очень простой метод, а затем перейдем к использованию очень популярной микросхемы драйвера двигателя L293D. Позже в книге мы также узнаем, как использовать их для управления шаговым двигателем.

Проект 15 - Простое управление двигателем

Сначала мы будем просто управлять скоростью двигателя постоянного тока (DC) в одном направлении, используя силовой транзистор, диод и внешний источник питания для питания двигателя и потенциометр для управления скоростью. Любой подходящий силовой транзистор NPN, предназначенный для сильноточных нагрузок, может заменить транзистор TIP120. Однако я настоятельно рекомендую вам использовать силовой транзистор Дарлингтона - он имеет очень высокий коэфициент усиления. Убедитесь, что вы выбрали транзистор, который может выдерживать напряжение и ток, которые потребляет ваш двигатель. Может потребоваться установить радиатор на транзистор, если он потребляет больше ампера.
Внешний источник питания может представлять собой набор батарей или внешний источник постоянного тока. Источник питания должен иметь достаточное напряжение и ток для управления двигателем. Напряжение не должно превышать требуемого для двигателя. В целях тестирования я использовал источник питания постоянного тока, обеспечивающий 5 В при 500 мА. Этого было достаточно для двигателя постоянного тока 5 В, который я использовал. Если вы используете источник питания с напряжением выше, чем может выдержать двигатель, вы можете повредить его. В таблице 5-1 перечислены детали, необходимые для следующего проекта.

Требуемые детали

Table 5-1. *Требуемые детали для Проекта 15*

DC мотор	
Потенциометр 10 кОм	
TIP120 транзистор*	
1N4001диод	
*Jack разьем	
Внешний источник питания	
Резистор 1K	

**или подходящий эквивалент*

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Теперь возьмите необходимые детали и соедините их, как показано на рис. 5-1. Для этого проекта важно, чтобы вы проверили и дважды проверили, все ли ваши соединения в порядке, прежде чем подавать питание на схему, поскольку невыполнение этого может привести к повреждению ваших компонентов или даже вашего Arduino. В частности, диод необходим для защиты Arduino от обратной ЭДС, о чем мы поговорим позже.

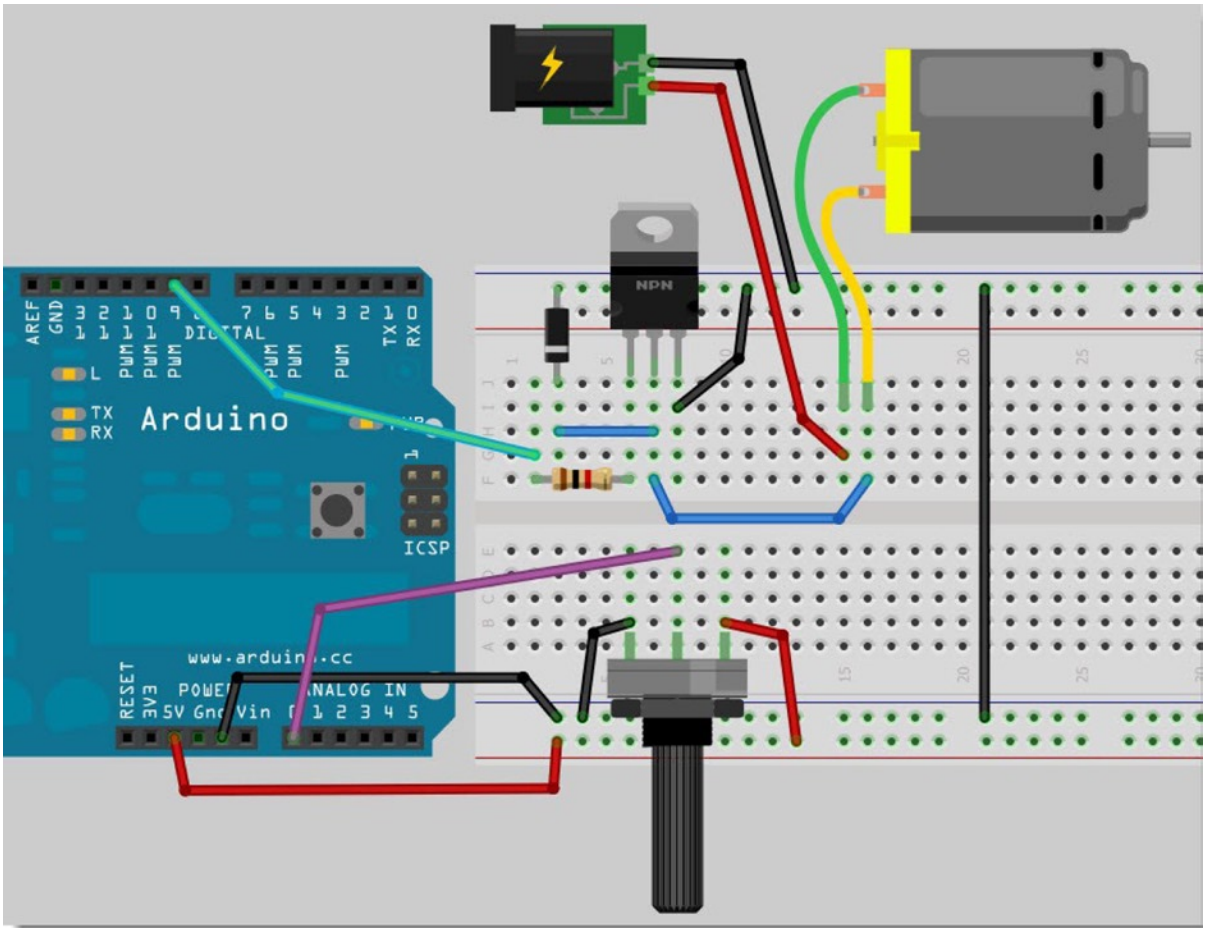


Рис. 5-1. *Схема для проекта 15 - Простое управление двигателем*

Введите код

Откройте IDE Arduino и введите код из листинга 5-1:

Листинг 5-1. Код для проекта 15

```
// Проект 15 - Простое управление двигателем

int potPin = 0;           // Аналоговый вход 0 подключен к средн. выводу потенциометру
int transistorPin = 9;    // Вывод 9 ШИМ подключен к базе транзистора
int potValue = 0;         // значение, возвращаемое потенциометром
```

```
void setup() {
  // установите вывод транзистора как выход:
  pinMode(transistorPin, OUTPUT);
}

void loop() {
  // считываем потенциометр, преобразуем его в 0-255:
  potValue = analogRead(potPin) / 4;
  // используем это для управления транзистором:
  analogWrite(transistorPin, potValue);
}
```

Перед загрузкой кода отключите внешний источник питания от двигателя и убедитесь, что потенциометр полностью повернут против часовой стрелки. Теперь загрузите код в Arduino. После загрузки кода подключите внешний источник питания. Теперь вы можете повернуть потенциометр, чтобы управлять скоростью двигателя.

Проект 15 - Простое управление двигателем - Обзор кода

Сначала мы объявляем три переменные, которые будут содержать значение для аналогового вывода, подключенного к потенциометру, вывода ШИМ, подключенного к базе транзистора, и одну переменную для хранения значения, считанного потенциометром с аналогового вывода 0.

```
int potPin = 0;           // Аналоговый вход 0 подключен к потенциометру
int transistorPin = 9;    // Вывод 9 ШИМ подключен к базе транзистора
int potValue = 0;         // значение, возвращаемое потенциометром
```

В функции setup () мы устанавливаем режим пина транзистора на вывод.

```
void setup() {
  // устанавливаем пин транзистора как выход:
  pinMode(transistorPin, OUTPUT);
}
```

В основном цикле potValue устанавливается равным значению, считанному с аналогового вывода 0 (potPin), а затем деленному на 4.

```
potValue = analogRead(potPin) / 4;
```

Нам нужно разделить считываемое значение на 4, поскольку аналоговое значение будет варьироваться от 0 для 0 вольт до 1023 для 5 вольт.
Значение, которое нам нужно записать на вывод транзистора, может находиться в диапазоне от 0 до 255, поэтому мы делим значение аналогового вывода 0 (максимум 1023) на 4, чтобы получить максимальное значение 255 для установки цифрового вывода 9 (с помощью analogWrite, поэтому мы используем ШИМ). Затем код записывает на вывод транзистора значение потенциометра.

Затем код записывает на вывод транзистора значение потенциометра.

```
analogWrite(transistorPin, potValue);
```

Другими словами, когда вы вращаете потенциометр, считываются различные значения в диапазоне от 0 до 1023, и они преобразуются в диапазон от 0 до 255, а затем это значение записывается (через ШИМ) на цифровой вывод 11, который изменяет скорость мотора. Поверните ручку потенциометра до упора влево, и мотор выключится; поверните его вправо, и он разгонится до максимальной скорости.

Код очень простой, и мы не узнали ничего нового. Давайте теперь посмотрим на устройство, используемое в этом проекте, и посмотрим, как все это работает.

Проект 15 - Простое управление двигателем - Обзор устройства

Схема по существу разделена на две части. Секция 1 - это наш потенциометр,крайные выводы которого подключены к 5 В и заземлению, при этом центральный вывод подключен к аналоговому выводу 0 платы Arduino. При вращении ручки потенциометра делитель напряжения,который образует потенциометр изменяет напряжение от 0 до 5 В,значение которого с центрального вывода, считывается с помощью аналогового вывода 0.

Вторая часть - компонент управляющий скоростью вращения вала двигателя. Цифровые выводы на Arduino выдают максимум 40 мА (миллиампер). Однако в таблице данных для МК указано, что максимальный гарантированный выходной ток составляет всего 20 мА. Вы никогда не должны работать на абсолютном максимуме. Двигателю постоянного тока может потребоваться около 500 мА для работы на полной скорости, и это явно слишком много для Arduino. Если бы мы попытались запустить двигатель непосредственно с вывода на Arduino, это могло бы серьезно и необратимо повредить его.

Следовательно, нам нужно найти способ обеспечить его более высоким током. Поэтому мы получаем питание от внешнего источника питания, который дает нам ток, достаточный для питания двигателя. Мы могли бы использовать выход 5 В от Arduino, который может обеспечить до 800 мА при подключении к внешнему источнику питания и меньше при питании от USB.

Однако платы Arduino дороги, и их слишком легко повредить при подключении к сильноточной нагрузке, такой как двигатели постоянного тока. Поэтому мы будем использовать внешний источник питания.

Кроме того, вашему двигателю может потребоваться напряжение 9 В или 12 В или выше, и это превышает все, что может предоставить Arduino.Однако этот проект управляет скоростью двигателя, поэтому нам нужен способ управления мощностью, чтобы изменять скорость. Здесь на помощь приходит транзистор TIP-120.

Транзисторы

Транзистор - это усилитель мощности, который также можно использовать как цифровой переключатель. В нашей схеме мы используем его как цифровой переключатель.

Цоколевка транзистора который мы используем в этом проекте изображена на рис. 5-2.

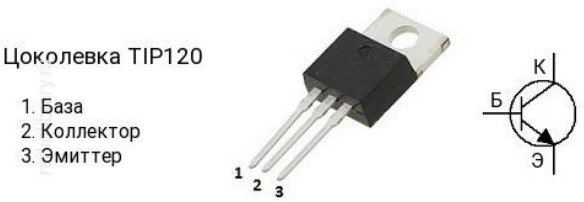


Рис. 5-2. *Обозначение и цоколевка транзистора TIP120*

Транзистор TIP120 можно заменить на 2N6043, 2N6043G, 2N6044, 2N6044G, 2N6045, 2N6045G, 2N6387, 2N6387G, 2N6388, 2N6388G, 2N6530, 2N6532, 2SB601, 2SB601-K, 2SB601-L, 2SB601-M, 2SD1415, 2SD1415A, 2SD2495, 2SD2495-O, 2SD2495-P, 2SD2495-Y, 2SD560, 2SD560-KB, 2SD560-LB, 2SD560-MB, 2SD633, 2SD634, 2SD635, BDT63, BDT63A, BDT63B, BDT63C, BDT65, BDT65A, BDT65B, BDT65C, BDW40, BDW41, BDW41G, BDW42, BDW42G, BDW43, KSB601, KSB601-O, KSB601-R, KSB601-Y, KSD560, KSD560-O, KSD560-R, KSD560-Y, MJF122, MJF122G, MJF6388, MJF6388G, TIP100, TIP100G, TIP101, TIP101G, TIP102, TIP102G, TIP120G, TIP121, TIP121G, TIP122, TIP122G, TIP130, TIP130G, TIP131, TIP131G, TIP132, TIP132G, TIP142T, TTD1415B

Я не буду описывать работу транзистора как такового,т.к. это материал для целой книги. Заинтересованных я отсылаю к соответствующей литературе,где более подробно и доходчиво описывается работа транзистора, схемы его включения и проекты на его основе.

Моторы

Моторы используют электромагнитные силы для обеспечения движения, преобразования электроэнергии в механическую.
Магнитные поля создают силу, которая может перемещать объекты. Каждый магнит имеет магнитное поле с северным и южным полюсом. Если вы попробуете приблизить два северных полюса двух магнитов, они будут отталкиваться. То же самое произойдет, если вы попытаетесь приблизить два южных полюса. Если полюса одинаковые, они будут отталкиваться друг от друга. Если же вы приблизите северный полюс одного с южному полюсу другого магнита, они притянутся с определенным усилием. То есть, противоположные полюса притягиваются друг к другу.
Электродвигатель использует свойства магнитов отталкиваться и притягиваться для генерации движения. В обычном электродвигателе два магнита: постоянный и переменный. Переменный магнит называется электромагнитом. Электромагнит создается с помощью пропускания электрического тока через проводник. Постоянный магнит постоянно имеет магнитное поле (северный и южный полюса), а электромагнит генерирует магнитное поле только, когда через него пропускают электрический ток. Сила магнитного поля электромагнита может быть увеличена с помощью повышения тока, проходящего через проводник или методом формирования нескольких обмоток проводника.
В электродвигателе электромагнит устанавливается на ось таким образом, что он может свободно вращаться внутри магнитного поля постоянного магнита. В момент, когда через проводник проходит электрический ток, переменное магнитное поле взаимодействует со статическим магнитным полем магнита, возникают силы отталкивания и притяжения. Это заставляет электромагнит вращаться, возникает движение.
Основные узлы электродвигателя:

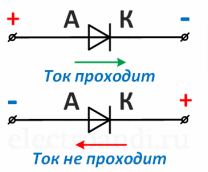
- Постоянный магнит (магниты), в случае, когда он установлен неподвижно, называется статором;

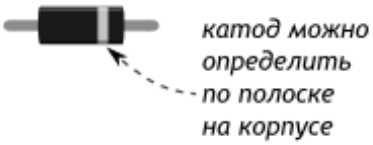
- внутри статора есть катушка, которая установлена на оси и вращается, называется ротором.

Двигатели постоянного тока (DC motor) во многом являются самыми простыми электродвигателями. Большинство «щеточных» двигателей работают одинаково. Есть ротор и статор. Есть магниты на статоре и катушка на роторе с магнитным полем, которое генерируется с помощью подачи на нее силы тока. Есть щетки внутри мотора, которые заставляют двигаться ротор.
При использовании источника постоянного тока, для управления подобным двигателем практически ничего не надо. Скорость его вращениязависит от силы тока, которая поступает на катушки от источника питания к коммутатору.
Для вращения оси двигателя в противоположном направлении, достаточно подключить контакты от источника питания к двигателю наоборот.

Диоды

Диод — это электронный элемент, который пропускает ток в одном направлении и не пропускает в другом. На рисунке ниже видно как течёт ток через диод.
Диоды относятся к классу полупроводников и считаются активными электронным компонентам (резисторы и конденсаторы — пассивными).
У диода есть два вывода катод и анод. Существует простой способ запомнить, что подключать к катоду плюс или минус. В слове "анод" столько же букв сколько в слове "плюс", соответственно к аноду подключаем плюс, а в слове "катод" столько же букв сколько в слове "минус", то есть к катоду подключаем минус.



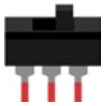


Проект 16 - Использование микросхемы драйвера двигателя L293D

В нашем предыдущем проекте мы использовали транзистор для управления двигателем. В этом проекте мы собираемся использовать микросхему драйвера двигателя под названием L293D. Преимущество этого чипа в том, что мы можем управлять двумя двигателями одновременно, а также управлять направлением вращения валов. Чип также можно использовать для управления шаговым двигателем в Проекте 28.Также доступен совместимый по выводам чип, известный как SN754410, который также может быть использован и имеет более высокий номинальный ток. Микросхема имеет собственные внутренние диоды, поэтому в этом проекте они нам не нужны.

Требуемые детали

Table 5-2. *Требуемые детали проекта 16*

DC мотор	
L293D или SN754410	
Потенциометр 10 кОм	
Ползунковый переключатель	
Резистор 10 кОм	
Радиатор	
2 конденсатора по 0,1 мкФ	

Подключение

Во-первых, убедитесь, что ваш Arduino выключен, отсоединив его от USB-кабеля. Теперь возьмите необходимые детали и соедините их, как показано на Рисунке 5-3. Опять же, тщательно проверьте цепь перед ее включением. L293D может нагреваться при использовании, поэтому рекомендуется использовать радиатор. Приклейте радиатор к верхней части микросхемы с помощью специального эпоксидного термоклея, двусторонней теплопроводной ленты или используйте прикрепляемый радиатор. Чем больше радиатор, тем лучше. Имейте в виду что температура может стать достаточно высокой, чтобы расплавить пластик макетной платы или любые соприкасающиеся с ней провода. Не прикасайтесь к радиатору, так как вы можете обжечься и не оставляете схему под напряжением и без присмотра.

.

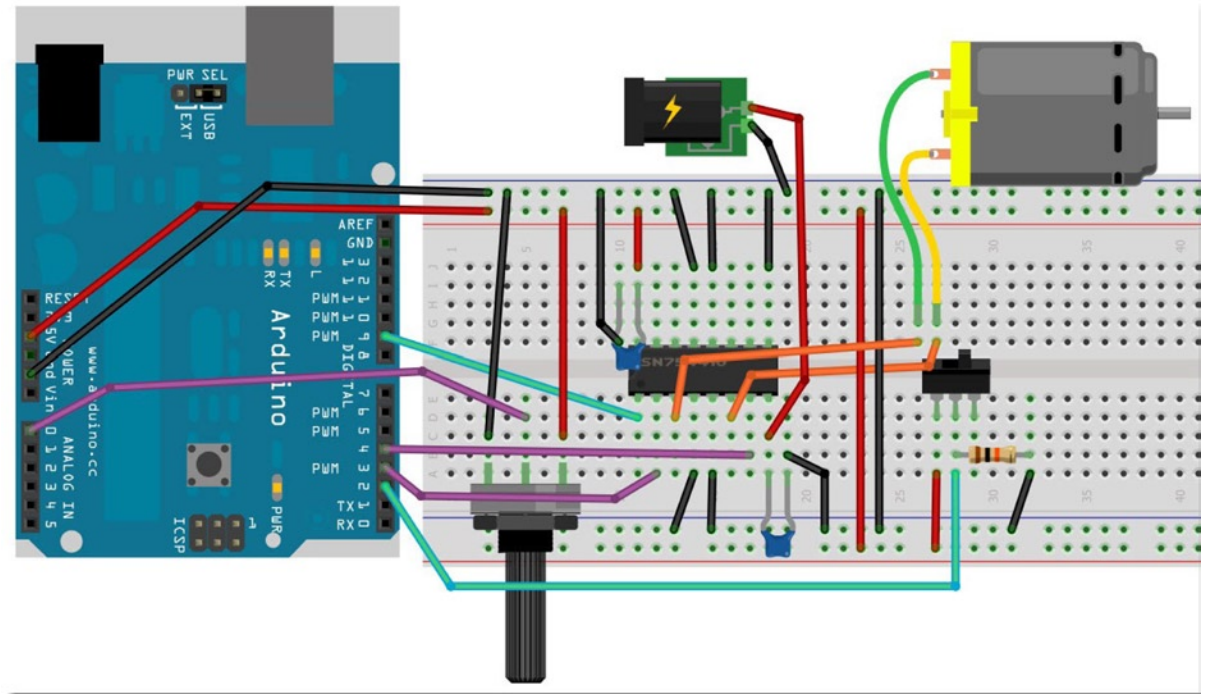


Рис. 5-3. *Подключения для проекта 16*

Введите код

Убедившись, что ваша схема подключена правильно, загрузите код из Листинга 5-2. На этом этапе не подключайте внешний источник питания.

Листинг 5-2. Код для проекта 16

```
// Проект 16 - Использование микросхемы драйвера двигателя L293D
#define switchPin 2 // switch input
#define motorPin1 3 // L293D Вход 1
#define motorPin2 4 // L293D Вход 2
#define speedPin 9  // L293D включить пин 1
#define potPin 0    // Потенциометр на аналоговом выводе 0
int Mspeed = 0;     // переменная для хранения текущего значения скорости

void setup() {
//устанавливаем контакт переключателя как INPUT
pinMode(switchPin, INPUT);

// устанавливаем оставшиеся контакты как выходы
pinMode(motorPin1, OUTPUT);
pinMode(motorPin2, OUTPUT);
pinMode(speedPin, OUTPUT);
}
```

```
void loop() {
  Mspeed = analogRead(potPin)/4;   // считываем значение скорости с потенциометра
  analogWrite (speedPin, Mspeed);  //  запись скорости в Enable 1 pin
  if (digitalRead(switchPin)) { // Если переключатель ВЫСОКИЙ,вращать вал двигателя по часовой стрелке
    digitalWrite(motorPin1, LOW);  // устанавливаем Вход 1 L293D на низкий уровень
    digitalWrite(motorPin2, HIGH); // устанавливаем Вход 2 L293D в высокий уровень
  }
      else {  // если переключатель в НИЗКОМ состоянии, вращать вал двигателя против часовой стрелки
        digitalWrite(motorPin1, HIGH); // устанавливаем Вход 1 L293D на низкий уровень
        digitalWrite(motorPin2, LOW);  // устанавливаем Вход 2 L293D в высокий уровень
  }
}
```

После завершения загрузки кода установите потенциометр в среднее положение и подключите внешний источник питания. Теперь двигатель будет вращаться, и вы можете регулировать его скорость ручкой потенциометра. Чтобы изменить направление вращения двигателя, сначала установите минимальную скорость, затем переведите ползунок переключателя в противоположное положение. Теперь двигатель будет вращаться в противоположном направлении.Опять же, будьте осторожны с этим чипом, так как при включении он сильно нагревается.

Проект 16 - Использование микросхемы драйвера двигателя L293D - Обзор кода

Код этого проекта очень простой. Сначала мы определяем выводы, которые собираемся использовать на Arduino.

```
#define switchPin 2 // switch input
#define motorPin1 3 // L293D Input 1
#define motorPin2 4 // L293D Input 2
#define speedPin 9  // L293D enable pin 1
#define potPin 0    // Потенциометр на аналоговом выводе 0
```

Затем установите целое число, чтобы удерживать значение скорости, считываемое с потенциометра.

```
int Mspeed = 0;  // переменная для хранения текущего значения скорости
```

Затем в функции настройки мы устанавливаем соответствующие выводы на входы или выходы.

```
pinMode(switchPin, INPUT);

pinMode(motorPin1, OUTPUT);
pinMode(motorPin2, OUTPUT);
pinMode(speedPin, OUTPUT);
```

В основном цикле мы сначала считываем значение с потенциометра, подключенного к аналоговому выводу 0, и сохраняем его в Mspeed.

```
Mspeed = analogRead(potPin)/4; // считываем значение скорости с потенциометра
```

Затем мы устанавливаем значение ШИМ на выводе 9 ШИМ на соответствующую скорость.

```
analogWrite (speedPin, Mspeed); //  запись скорости в Enable 1 pin
```

Оператор if решает, является ли значение, считываемое с вывода переключателя, HIGH или LOW. Если он ВЫСОКИЙ, то выход 1 на L293D установлен на НИЗКИЙ, а выход 2 установлен на ВЫСОКИЙ. Это будет то же самое, что выход 2, имеющий положительное напряжение, а выход 1 - на земле, заставляя двигатель вращаться в одном направлении.

```
if (digitalRead(switchPin)) {      // Если переключатель ВЫСОКИЙ, вращение по часовой стрелке
    digitalWrite(motorPin1, LOW);  // устанавливаем Вход 1 L293D на низкий уровень
     digitalWrite(motorPin2, HIGH); // устанавливаем Вход 2 L293D на высокий уровень
  }
```

Если контакт переключателя находится в положении НИЗКИЙ, то выход 1 устанавливается на ВЫСОКИЙ, а выход 2 - на НИЗКИЙ, меняя направление вращения двигателя на обратное.

```
else {   // если переключатель в НИЗКОМ состоянии,вращение двигателя против часовой стрелки
  digitalWrite(motorPin1, HIGH); // устанавливаем Вход 1 L293D на низкий уровень
 digitalWrite(motorPin2, LOW);  // устанавливаем Вход 2 L293D в высокий уровень
}
```

Цикл теперь будет повторяться, проверяя новое значение скорости или новое выбранное направление и устанавливая соответствующие пины скорости и направления. Как видите, использование микросхемы драйвера двигателя вовсе не так страшно, как вы могли подумать вначале. Попытка воссоздать вышеуказанную схему и код без этого была бы намного сложнее. Никогда не пугайтесь микросхем, поскольку они обычно намного проще, чем кажется на первый взгляд. Внимательное чтение их таблиц данных раскроет их секреты. Давайте посмотрим, как работают новые компоненты, представленные в этом проекте.

Проект 16 - Использование микросхемы драйвера двигателя L293D - Обзор устройства

Новый компонент, с которым мы столкнулись в Проекте 16, - это ИС драйвера двигателя. Это будет либо L293D, либо SN754410, в зависимости от того, что вы выбрали (доступны и другие микросхемы, и при небольшом поиске в Интернете можно найти другие драйверы двигателя, совместимые с ножками). Рекомендуется использовать L293D, так как SN754410 греется и требуется радиатор.
L293D - это так называемый двойной H-мост. H-образный мост - это полезная и простая электронная схема. Вы можете сделать свой собственный H-мост, используя переключатели. См. Рисунок 5-4.

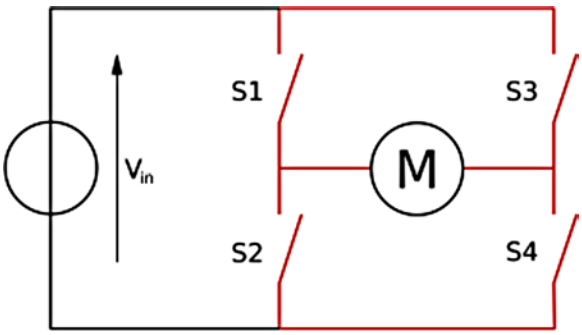


Рис. 5-4. *H-образный мост из переключателей*

На рисунке 5-4 двигатель подключен к четырем переключателям.
Конфигурация называется H-образным мостом, потому что он напоминает букву H с нагрузкой моста в центре. Теперь посмотрим на рисунок 5-5.

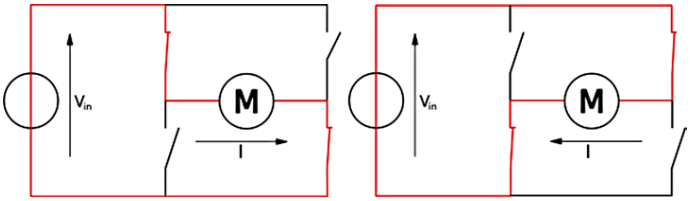


Рис. 5-5. *Изменение направления двигателя в H-мосте*

С левой стороны у нас замкнуты верхний левый и нижний правый переключатели. В закрытом состоянии ток будет течь через двигатель слева направо, и двигатель будет вращаться. Если мы затем разомкнем эти переключатели и вместо этого замкнем верхний правый и нижний левый переключатели, ток будет течь через двигатель в противоположном направлении и, следовательно, заставит его вращаться в противоположном направлении. Так работает H-мост.

ИС драйвера двигателя состоит из двух H-мостов и вместо переключателей использует транзисторы. Так же, как транзистор в проекте 15 использовался в качестве переключателя, транзисторы внутри H-моста включаются и выключаются в той же конфигурации, что и на рис. 5-5, чтобы ваш двигатель вращался в любом направлении. Чип представляет собой двойной H-мост, так как внутри он имеет два H-моста; Таким образом, вы можете управлять двумя двигателями и их направлениями одновременно.

При желании вы можете сделать свой собственный H-мост из транзисторов и диодов, и он будет выполнять ту же работу, что и L293D. L293D имеет то преимущество, что он крошечный, и все эти транзисторы и диоды расположены внутри небольшого пространства.

L293D имеет два пина источника питания в дополнение к четырем пинам земли. Пин 16 (Vcc1) - это напряжение питания для логической схемы микросхемы, и он должен подаваться от источникапитания 5V Arduino. Пин 8 (Vcc2) - это напряжение питания для схемы питания микросхемы (которая фактически управляет двигателем). Земли обоих источников питания и все четыре пина земли микросхемы должны быть соединены вместе. Два H-моста на микросхеме работают независимо, разделяя только два источника питания. Контакт 1 служит для включения H-моста, управляемого контактами 2 и 7. Контакт 9 служит для включения H-моста, управляемого контактами 10 и 15.

Если вход разрешения подключен к низкому логическому уровню, все четыре переключателя соответствующего H-моста будут выключены (разомкнуты). Если разрешение имеет высокий логический уровень (> 2,4 В), пара входов для соответствующего H-моста будет управлять, какой из 4 переключателей H-моста включен, следующим образом:

H-мост 1-2 (включение управляется пином 1)

пин 2	пин 7	Выходной пин 3	Выходной пин 6	Описание
L	L	Подключается к земле	Подключается к земле	Мотор выключен (тормоз)
L	H	Подключается к земле	Подключается к Vcc2	Двигатель включен (в одном направлении)
H	L	Подключается к Vcc2	Подключается к земле	Двигатель включен (в одном направлении)
H	H	Подключается к Vcc2	Подключается к Vcc2	Мотор выключен (тормоз)

H-мост 3-4 (включение управляется пином 9)

пин 10	пин 15	Выходной пин 11	Выходной пин 14	Описание
L	L	Подключается к земле	Подключается к земле	Мотор выключен (тормоз)
L	H	Подключается к земле	Подключается к Vcc2	Двигатель включен (в одном направлении)
H	L	Подключается к Vcc2	Подключается к земле	Двигатель включен (в одном направлении)
H	H	Подключается к Vcc2	Подключается к Vcc2	Мотор выключен (тормоз)

Максимальное напряжение питания для Vcc2 (питание двигателя) составляет 36 вольт. Если напряжение Vcc2 ниже 4,5 вольт, на выходах могут происходить странные вещи, и микросхемы могут нагреваться. Транзисторы - не идеальные переключатели. Когда транзисторы в L293D включены, на них будет падение напряжения, поэтому напряжение на двигателе обычно будет на 1,5–2,5 В меньше, чем Vcc2. (От 1,5 до 2,5 вольт). Чем выше ток двигателя, тем больше мощности теряется в транзисторах. Это один из недостатков. При работе с более мощными двигателями необходимы более совершенные микросхемы драйверов и радиаторы.

упражнение

теперь попробуйте проделать следующее упражнение

- Упражнение 1: вместо одного двигателя попробуйте подключить два с двумя переключателями направления и потенциометрами для управления направлением и скоростью. На рисунке 5-6 показаны выводы микросхемы.

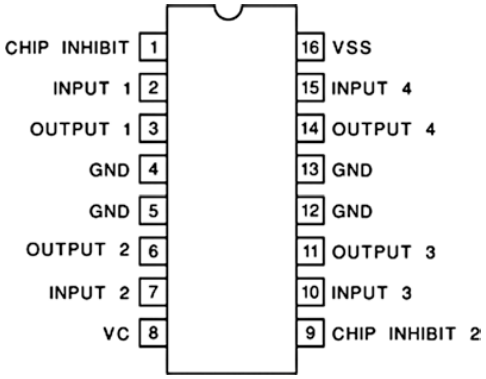


Рис. 5-6. *Распиновка L293D (или SN754410)*

Резюме

В главе 5 показано, как использовать моторы в ваших проектах. Он также познакомил вас с важными концепциями, касающимися транзисторов и диодов; вы будете часто использовать оба этих компонента всвоих проектах, особенно если вы управляете устройствами, которые требуют большего напряжения илитока чем может обеспечить Arduino.
Теперь вы также знаете, как построить H-образный мост и как использовать его для изменения направлениядвигателя. Вы также использовали популярный драйвер двигателя L293D IC; вы будете работать с этимчипом позже в этой книге, когда будете рассматривать двигатель другого типа. Навыки, необходимые дляуправления двигателем, жизненно важны, если вы собираетесь построить робота или любоерадиоуправляемое транспортное средство с использованием Arduino.

Предметы и понятия, затронутые в главе 5

- Использование транзистора для питания высокомощных устройств
- Управление скоростью двигателя с помощью ШИМ
- Ток, подаваемый цифровым выводом Arduin
- Использование внешнего источника питания необходимо для устройств большой мощности
- Понятие о транзисторах и как они работают
- Моторы и как они работают
- Использование диода для предотвращения обратной ЭДС
- Как работает диод
- Как двигатели могут быть использованы для выработки электроэнергии
- Насколько важны диоды для защиты схемы
- Как запитать двигатель с помощью микросхемы драйвера двигателя L293D
- Почему радиаторы необходимы для отвода тепла от ИС
- Концепция H-образного моста и его использование для изменения направления вращения двигателя

ГЛАВА 6

Двоичные счетчики и ввод / вывод регистра сдвига

Регистр обычно представляет собой сборку из D-триггеров — элементарных ячеек памяти,которые могут хранить небольшой объем данных для быстрого доступа к ним. Он есть внутри каждого контроллера и МК, включая и микроконтроллер Atmega328, который входит в состав платы Arduino Uno. Записывать данные в регистр можно либо последовательно, либо параллельно. Регистры первого типа называются сдвиговыми, второго типа — параллельными.

Регистр называется сдвиговым, потому что при добавлении каждого нового бита в него, мы как бы сдвигаем все остальные в сторону.

В параллельном режиме данные из регистра можно считывать одновременно из всех ячеек. Именно это свойство помогает нам работать с кучей светодиодов.

Регистры можно соединять в цепочку. В таком случае, вытесненный бит не будет пропадать без следа, а отправится в начало следующего регистра. При этом увеличивается число доступных выводов.

Плата Arduino содержит ограниченное число выводов и при сложном проекте их не хватает для полноценной работы. К примеру, для подключения 7-сегментного индикатора необходимо задействовать восемь выводов, два индикатора займут уже 16 выводов. Сдвиговый регистр позволяет сэкономить число используемых выводов, беря часть управления выводами на себя.

Проект 17 - 8-битный двоичный счетчик сдвигового регистра

Требуемые детали

Детали, необходимые для Проекта 6 показаны в Таблице 6-1.

Таб. 6-1. *Детали, необходимые для проекта 17*

1 x 74HC595 регистр сдвига IC	
8 x Резисторы 560 Ом *	
8 x светодиодов 5 мм	
Конденсатор 0,1 мкФ	

**или подходящий эквивалент*

Подключение

Внимательно,изучите схему подключения.

Вам понадобится конденсатор, который будет находиться между выводом источника питания и землей. Убедитесь, что его номинальное напряжение выше, чем используемое напряжение питания. Выводы должны быть короткими, а конденсатор должен как можно ближе к микросхеме. Назначение этого конденсатора - уменьшить влияние электрических помех на схему.

После того, как вы все подключили, как показано на рис. 6-1, проверьте правильность подключения, а также правильность расположения ИС и светодиодов.

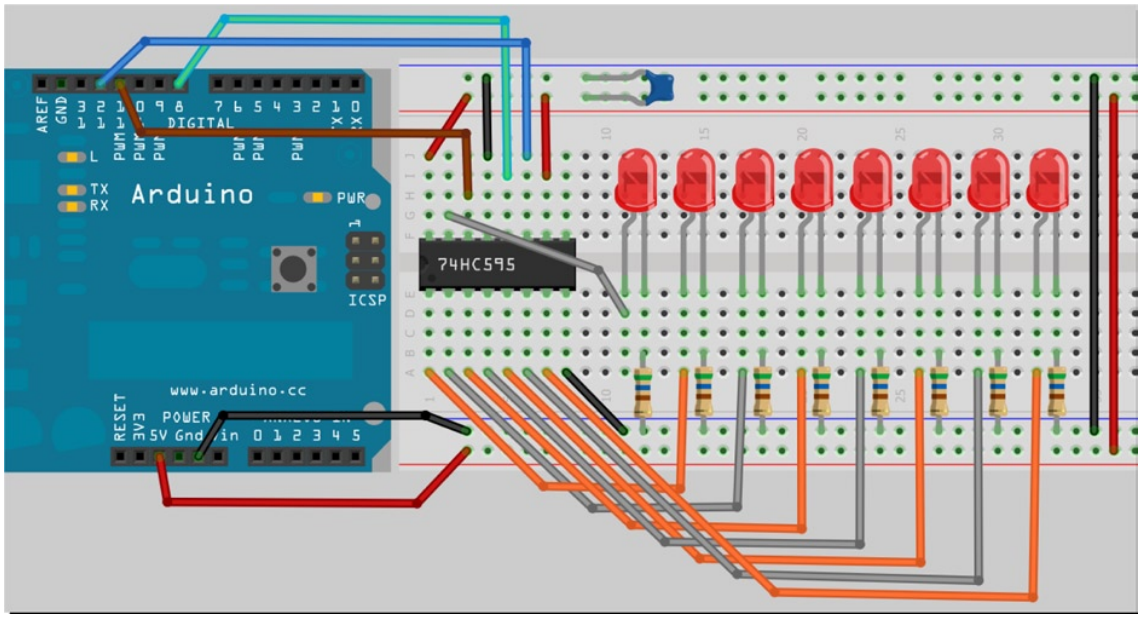


Рис. 6-1. *Схема для проекта 17 - 8-битный двоичный счетчик сдвигового регистра*

Введите код

Введите следующий код в листинге 6-1 и загрузите его в свой Arduino. После запуска кода вы увидите, что светодиоды включаются и выключаются по отдельности, поскольку они запускаются в двоичном формате каждую секунду от 0 до 255 друг за другом.

Листинг 6-1. Код для проекта 17

```
// Проект 17

Pin = 8;  //Пин подключен к выводу 12 74HC595 (latch - защелка)
Pin = 12; //Пин подключен к выводу 11 74HC595 (clock - такты)
Pin = 11;  //Пин подключен к выводу 14 of 74HC595 (data - данные)
```

```
void setup() {
        //устанавливаем пины как выходы
        pinMode(latchPin, OUTPUT);
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);
}

void loop() {
        //считаем от 0 до 255
        for (int i = 0; i < 256; i++) {
                shiftDataOut(i);
                //устанавливаем latchPin на низкий, а затем на высокий уровень для отправки данных
                digitalWrite(latchPin, LOW);
                digitalWrite(latchPin, HIGH);
                delay(1000);
        }
}

void shiftDataOut(byte dataOut) {
  // Сначала сдвигаем 8 бит LSB, синхронизируя каждый бит с нарастающим фронтом линии синхронизации
  boolean pinState;

        for (int i=0; i<=7; i++)  {// для каждого бита в dataOut отправляем бит
                digitalWrite(clockPin, LOW);//установить для clockPin значение LOW перед отправкой бита

                // если значение DataOut и (логическое И) битовая маска
                // верны, устанавливаем pinState в 1 (HIGH)
                if ( dataOut & (1<<i) ) {
                        pinState = HIGH;
                }
                else {
                        pinState = LOW;
                }

                //устанавливает для dataPin значение HIGH или LOW в зависимости от pinState
                digitalWrite(dataPin, pinState); //отправляем бит перед нарастанием тактового сигнала
                digitalWrite(clockPin, HIGH);
        }
  digitalWrite(clockPin, LOW); //прекращаем перенос данных
}
```

Двоичная система счисления

Прежде чем мы рассмотрим код и устройство для Project 17, давайте взглянем на двоичную систему счисления, поскольку очень важно понимать двоичную систему для успешного программирования МК.

Люди используют десятичную или десятичную систему счисления, потому что у нас наруках 10 пальцев. У компьютеров нет пальцев, поэтому лучший способ для компьютера считать - это состояние ВКЛЮЧЕНО или ВЫКЛЮЧЕНО (1 или 0). Логическое устройство, такое как компьютер, может определить, есть ли напряжение (1) или нет (0), и поэтому использует двоичную систему счисления или систему счисления с основанием 2, поскольку эту систему счисления можно легко представить в виде электронной схемы с состоянием высокого или низкого напряжения.

```
000,001,002,003,004,005,006,007,008,009
010,011,012,013,014,015,016,017,018,019
020,021,023 .........
```

В двоичном формате происходит то же самое, за исключением того, что старшая цифра равна 1. Таким образом, добавление 1 к 1 приводит к сбросу цифры на ноль и добавлению 1 к столбцу слева.

```
000, 001
010, 011
100, 101...
```

8-битное число (или байт) представлено в таблице 6-2.

Таб. 6-2. *Примеры двоичных чисел*

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	0	0	1	0	1	1

Число выше в двоичном формате - 01001011; в десятичной системе это 75.

Это работает следующим образом:

1 x 1 = 1

1 x 2 = 2

1 x 8 = 8

1 x 64 = 64

Сложите все это вместе, и вы получите 75.

Вот еще несколько примеров (см. Таблицу 6-3.)

Table 6-3. *Примеры двоичных чисел*

Dec	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
75	0	1	0	0	1	0	1	1
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
12	0	0	0	0	1	1	0	0
27	0	0	0	1	1	0	1	1
100	0	1	1	0	0	1	0	0
127	0	1	1	1	1	1	1	1
255	1	1	1	1	1	1	1	1

Итак,когда мы разобрались с двоичным кодом, рассмотрим компоненты, прежде чем совершим обзор кода.

■ **Совет** вы можете использовать GOOGlE для преобразования десятичного числа в двоичное и наоборот.

Например, чтобы преобразовать 171 десятичное число в двоичное, введите 171 в двоичном формате в поле поиска GOOGlE, которое вернет 171 = 0b10101011, префикс 0b показывает, что число является двоичным числом, а не десятичным. так что ответ 10101011.

Проект 17 - 8-разрядный двоичный счетчик сдвигового регистра - Обзор устройства

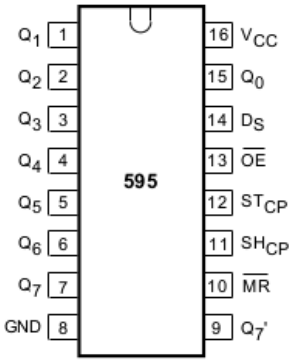
Мы используем сдвиговый регистр, в частности, сдвиговый регистр типа 74HC595. Этот тип регистра сдвига представляет собой 8-битный регистр сдвига с последовательным входом, последовательным или параллельным выходом с выходными защелками. Это означает, что мы можем отправлять данные в сдвиговый регистр последовательно или параллельно. Последовательно означает 1 бит за раз. Параллельный означает множество битов (в данном случае 8) одновременно.
Таким образом, вы передаете данные регистра сдвига (в виде единиц и нулей) по одному биту за раз. Каждый бит шунтируется при вводе следующего бита. Когда вводится девятый бит, первый введенный бит будет выведен за пределы регистра сдвига и потерян (если мы что-то с ним не сделаем - см. Следующий проект). Когда вы поднимаете сигнал фиксации с LOW на HIGH,текущее содержимое сдвигового регистра копируется в выходные защелки. (Копирование данных в выходные защелки не влияет на содержимое сдвигового регистра.)

Этот тип сдвигового регистра обычно используется для последовательного преобразования данных в параллельный. Другие регистры сдвига с параллельными входами и последовательными выходами используются для параллельного преобразования данных в последовательные. Некоторые регистры сдвига могут делать и то, и другое. Поскольку выводимые данные - это единицы и нули (или 0 В и 5 В), мы можем использовать их для включения и выключения группы из 8 светодиодов.
Регистр сдвига для этого проекта требует всего три вывода от Arduino. Выходы Arduino и входы 595 показаны в Таблице 6-4). Мы не используем функции пинов 10 (общий сброс) и 13 (вход разрешения выхода). Тем не менее, они все равно должны быть установлены на высокий (вывод 10) или низкий (вывод 13) уровни, чтобы проект работал.

Таб. 6-4. *Используемые пины*

Пины Arduino	Пины 595	Описание
8	12	Тактовый вход регистра памяти
11	14	Последовательный ввод данных
12	11	Тактовый вход регистра сдвига

Мы будем называть пин 11 тактовым выводом (CLOCK), пин 14 - выводом данных (DATA), а пин 12 - выводом защелки (LATCH).
Представьте, что защелка похожа на камеру, которая делает снимок содержимого сдвигового регистра, когда пин защелки переходит с LOW на HIGH, и выводит этот снимок на 8 контактов (QA-QH или Q_0 до Q_7 в зависимости от вашей таблицы данных - см. Рис. 6-2). Такт - это просто импульс , а вывод данных - это то место, куда мы отправляем данные из Arduino в 595.



- Выходы Q_0 - Q_7 (15 и 1..7)
- GND (8) — земля
- Q7′ (9) — выход регистра, который необходимо соединить с Q0 следующего регистра для создания цепочки
- MR (10) - сброс значений регистра. Сброс происходит при LOW на что указывает верхняя черта
- D_S - вход последовательных данных (DATA)восемь бит
- SH_{CP} (11) - тактовый вход регистра сдвига (CLOCK)
- ST_{CP} (12) - “защелка” (LATCH) (12), позволяет (после сессии передачи данных) устанавливать уровни на всех восьми выходных ножках (Q0-Q7) одновременно

Рис. 6-2. *Распиновка микросхемы 595*

Чтобы использовать сдвиговый регистр, пин защелки и тактовый пин должны быть установлены в LOW. Пин защелки будет оставаться в НИЗКОМ состоянии до тех пор, пока не будут установлены все 8 бит. Это позволяет вводить данные в регистр хранения (место внутри ИС для хранения 1 или 0).

Затем мы подаем сигнал HIGH или LOW на вывод данных, а затем устанавливаем вывод синхронизации на HIGH.

Установив тактовый вывод на HIGH, мы копируем данные, представленные на выводе данных, в регистр хранения. Как только это будет сделано, мы снова устанавливаем тактовый пин на НИЗКИЙ, а затем представляем следующий бит данных на выводе данных. После того, как мы проделали это 8 раз, мы отправили полное 8-битное число в 595. Теперь вывод защелки поднят в 1, который копирует данные из регистра хранения в регистр сдвига и выводит их на Q0 по Q7 (вывод 15, 1-7).

Очень кратко принцип работы регистра можно описать так: у нас имеется байт (8 бит) данных снимаемый последовательно бит за битом с какого-либо цифрового вывода Arduino,который должен появиться на выходных ножках регистра - Q0-Q7.

Открываем “защелку”,т.е.ставим ей низкий уровень, передаем данные по линии DATA синхронно с тактовыми импульсами CLOCK. После 8 импульса на линии DATA закрываем “защелку” (установкой 1),что будет означать - закончить передавать данные и переслать на выход. После “защелкивания” состояние каждого бита переданного байта будет моментально установлено на соответствующих ножках регистра - Q0 - Q7.

Рис.6.3. Выход 595 показан в логическом анализаторе

Я подключил логический анализатор к моему 595 и диаграмма на рисунке 6-3 показывает вывод Arduino на 595.

Хорошо видно, как открылась защелка (Latch) падением из единицы в ноль, как восемь раз тикнули синхроимпульсы Clock с разным значением данных (Data) при них.

Из этой диаграммы видно, что двоичное число 00110111 (чтение справа налево) или десятичное 55 было отправлено на ИС 595.

Вместо вступления

У нас есть светодиод или реле, подключенное к Arduino. Мы хотим, чтобы он или оно включалось и выключалось, когда надо, по команде от нашей программы. Для этого мы подключаем его на любой свободный пин, указываем номер пина в программе, инициализируем его и пользуемся. Теперь представим, что светодиода два, нет, лучше десять, и еще десять реле, а еще десять кнопок и штук пять энкодеров, и все это не считая остального - датчиков, дисплеев и так далее. Ардуино Уно, с ее 20 пинами, уже давно не хватает и заканчиваются ножки на Меге. Но тут появилась задача добавить в проект десяток семисегментных индикатора, пять шаговиков, каждому из которых необходимо еще по два пина, и пару джойстиков, чтобы всем мог управлять оператор. Количество требуемых ножек перевалило за сотню, а в планах еще много чего. Да, если пойти по пути развития и усложнения своих проектов, не довольствуясь одним только blink, рано или поздно такое случится, пинов под все нужды может не хватить даже у самого большого контроллера-многоножки.

Для простого и надежного решения проблемы с дефицитом портов ввода-вывода и придуманы так называемые сдвиговые регистры - микросхемы, превращающие всего три пина Ардуино в десятки и сотни, преобразовывая последовательный интерфейс в параллельный или наоборот.

В природе существуют много технических исполнений сдвиговых регистров. В нашем проекте рассматривается самых распространенный: 74HC595 - выходной т.е он имеет 8 выходов.

Где же обещанные десятки и сотни выходов, спросите вы? И я отвечу - у регистров есть замечательная способность соединяться как вагоны в поезде, передавая данные друг другу посредством регистра Q0', который необходимо соединить с Q0 следующего регистра для создания цепочки. На языке схемотехники это называется “каскад”. Максимальное количество регистров в каскаде теоретически не ограничено и при правильном соединении их можно стыковать очень и очень много - скореевсего, раньше закончится память у Ардуино или фантазия и терпение у изготовителя.

Почему популярны именно эти сдвиговые регистры? Они просты, удобны, быстры, надежны, дешевы, их легко найти в продаже. В случае чего их легко заменить и не жалко выбросить.

Проект 17 - 8-битный двоичный счетчик сдвигового регистра - Обзор кода

Код для Project 6 на первый взгляд выглядит довольно устрашающе. Но если разложить его на составные части, вы увидите, что он не такой сложный, как кажется.

Сначала инициализируются три переменные:

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11;
```

... которые затем устанавливаются на выходы:

```
pinMode(latchPin, OUTPUT);
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);
```

Основной цикл запускает цикл for, отсчитывающий от 0 до 255. На каждой итерации цикла для latchPin устанавливается значение LOW, чтобы разрешить ввод данных, затем вызывается функция shiftDataOut, которая передает значение i в цикле for функции. Затем latchPin устанавливается в HIGH, передавая данные из регистра сдвига на выходные защелки и выходные выводы Q0-Q7. Наконец, есть задержка в полсекунды перед началом следующей итерации цикла.

```
void loop() {
  //считаем от 0 до 255
  for (int i = 0; i < 256; i++) {
    //устанавливаем latchPin на низкий уровень, чтобы разрешить поток данных
    digitalWrite(latchPin, LOW);
    shiftDataOut(i);
    //устанавливаем latchPin на высокий уровень для блокировки и отправки данных
    digitalWrite(latchPin, HIGH);
    delay(500);
  }
}
```

Функция shiftDataOut принимает в качестве параметра байт (8-битное число), которое будет нашим числом между 0 и 255. Мы выбрали байт для этого использования, поскольку его длина составляет ровно 8 бит, и нам нужно отправить только 8 битов в регистр сдвига.

```
void shiftDataOut(byte dataOut) {
```

Затем инициализируется логическая переменная pinState с сохранением состояния, в котором мы хотим, чтобы соответствующий вывод находился при отправке данных (1 или 0).

```
boolean pinState;
```

После этого мы готовы посылать 8 бит последовательно по одному бит за раз. Устанавливается цикл for, который повторяется 8 раз.

```
for (int i=0; i<=7; i++)  {
```

На выводе синхронизации устанавливается низкий уровень перед отправкой бита данных.

```
digitalWrite(clockPin, LOW);
```

Теперь оператор if / else определяет, следует ли установить для переменной pinState значение 1 или 0.

```
if ( dataOut & (1<<i) ) {
      pinState = HIGH;
}
else {
      pinState = LOW;
}
```

Условие для оператора if:

```
dataOut & (1<<i).
```

Это пример того, что называется «битовой маской», и теперь мы используем побитовые операторы. Это логические операторы, похожие на логические операторы, которые мы использовали в предыдущих проектах. Однако побитовые операторы действуют с числами на битовом уровне.В этом случае мы используем поразрядный оператор и (&) для выполнения логической операции над двумя числами. Первое число - это dataOut, а второе - результат (1 << i). Прежде чем мы продолжим, давайте взглянем на побитовые операторы.

Побитовые операторы

Поразрядные операторы выполняют вычисления переменных на битовом уровне. Существует 6 распространенных побитовых операторов, а именно:

```
&       побитовый and
|       побитовый or
^       побитовый xor
~       побитовый not
<<      побитовый left
>>      побитовый right
```

Побитовые операторы можно использовать только между целыми числами. Каждый оператор выполняет расчет на основе набора логических правил. Давайте внимательно посмотрим на оператор поразрядного И (&).

Побитовый AND (&) Побитовое И (&)

Поразрядный оператор И действует в соответствии с этим правилом: -

Если оба входа равны 1, результирующие выходы равны 1, в противном случае выход равен 0.

Другой способ взглянуть на это:

```
0 0 1 1       Operand1
0 1 0 1       Operand2

_______
0 0 0 1       (Operand1 & Operand2)
```

Тип int - это 16-битное значение, поэтому использование & между двумя выражениями int вызывает одновременное выполнение 16 операций И, как в таком разделе кода:

```
int x = 77;   //binary:    0000000001001101
int y = 121;  //binary:    0000000001111001
int z = x & y;//результат: 0000000001001001
```

In this case 77 & 121 = 73

Посмотрим на оставшиеся операторы.

Bitwise OR (|) Побитовое ИЛИ (|)

Если один или оба входа равны 1, результат равен 1, в противном случае - 0.

```
0 0 1 1       Operand1
0 1 0 1       Operand2

_____________
0 1 1 1       (Operand1 | Operand2)
```

Bitwise XOR (^) Побитовое исключающее ИЛИ (^)

Если только 1 из входов равен 1, то выход равен 1. Если оба входа равны 1, то выход 0.

```
0 0 1 1       Operand1
0 1 0 1       Operand2

_________
0 1 1 0       (Operand1 ^ Operand2)
```

Bitwise NOT (~) Побитовое НЕ (~)

Побитовый оператор НЕ применяется к одному операнду справа от него.

Выход становится противоположным входу. Нули преобразуются в единицы, а единицы в нули.

```
0 0 1 1       Operand1

_________
1 1 0 0       ~Operand1
```

Bitshift Left (<<), Bitshift Right (>>) Битовый сдвиг влево (<<), битовый сдвиг вправо (>>)

Операторы битового сдвига перемещают все биты целого числа влево или вправо на количество битов, указанное правым операндом.

```
variable << number_of_bits     переменная << число_битов
```

Например:

```
byte x = 9 ;     // binary: 00001001
byte y = x << 3; // binary: 01001000 (или 72 десятичное)
```

Любые биты, сдвинутые с конца строки, теряются навсегда. Вы можете использовать левый битовый сдвиг для умножения числа на степени 2, а правый битовый сдвиг - для деления на степени 2 .

Теперь, когда мы взглянули на операторы битового сдвига, давайте вернемся к нашему коду.

Проект 17 - Обзор кода (продолжение)

Условие оператора if / else было

```
dataOut & (1 << i)
```

И теперь мы знаем, что это побитовая операция И (&). Правый операнд в скобках - это операция сдвига влево. Это «битовая маска». 74HC595 принимает данные только по одному биту за раз. Поэтому нам необходимо преобразовать 8-битное число в dataOut в одноразрядное число, представляющее каждый из 8 битов по очереди. Битовая маска позволяет нам гарантировать, что для переменной pinState установлено значение 1 или 0 в зависимости от результата вычисления битовой маски. Правый операнд - это номер 1 бит, сдвинутый i раз. Поскольку цикл for заставляет значение i перейти от 0 до 7, мы можем видеть, что 1 бит, сдвинутый i раз, каждый раз в цикле, приведет к этим двоичным числам(см. Таблицу 6-6).

Таб. 6-6. *Результаты 1 << i*

Результ. 1 << i	Результат (1 << i) в двоичном формате
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

Итак, вы можете видеть, что в результате этой операции 1 перемещается справа налево.

Теперь правила оператора & гласят, что

Если оба входа равны 1, результирующие выходы равны 1, в противном случае выход равен 0.

Итак, состояние

```
dataOut & (1<<i)
```

приведет к 1, если соответствующий бит в том же месте, что и битовая маска, равен 1, в противном случае он будет равен нулю. Например, если значение dataOut было десятичным 139 или 10001011 двоичным, то каждая итерация цикла приведет к значениям в таблице 6-7 (см. Таблицу 6-7).

Таб. 6-7. *Результаты b10001011 << i*

Значение I	Результаты b10001011 << i
0	00000001
1	00000010
2	00000000
3	00001000
4	00000000
5	00000000
6	00000000
7	10000000

Таким образом, каждый раз, когда в позиции I появляется 1 (чтение справа налево), значение выходит как ненулевое значение (или ИСТИНА), и каждый раз, когда в позиции I появляется 0, значение выходит в 0 (или ЛОЖЬ).

Следовательно, условие if будет выполнять свой код в блоке, если значение больше 0 (или, другими словами, если бит в этой позиции равен 1) или «else» (если бит в этой позицииравен 0), оно выполнит код в блоке else.

Итак, посмотрим на оператор if / else еще раз

```
if ( dataOut & (1<<i) ) {
  pinState = HIGH;
 }
 else {
  pinState = LOW;
 }
```

И сравнивая это с таблицей истинности в Таблице 6-7, мы видим, что для каждого бита в значении dataOut, имеющего значение 1, pinState будет установлено в HIGH (1), а для каждого значения 0 он будетустановлен в LOW (0).

Следующий фрагмент кода записывает состояние HIGH или LOW на вывод данных, а затем устанавливает тактовый вывод на HIGH, чтобы записать этот бит в регистр хранения.

```
digitalWrite(dataPin, pinState);
digitalWrite(clockPin, HIGH);
```

Наконец, тактовый вывод (Clock) устанавливается на низкий уровень, чтобы больше не записывать бит.

```
digitalWrite(clockPin, LOW);
```

Таким образом, простыми словами, этот раздел кода просматривает каждый из 8 битов значения в dataOut один за другим и устанавливает для вывода данных значение HIGH или LOW соответственно, а затем записывает это значение в регистр хранения.Это просто отправка 8-битного числа на 595 по одному биту за раз. Затем основной цикл устанавливает пин защелки (latch) в 1, чтобы отправлять эти 8 бит одновременно на выводы 15 и с 1 по 7 (от QA до QH) сдвигового регистра, тем самым заставляя наши 8 светодиодов визуально отображать двоичное число, хранящееся в регистре сдвига.

Ваш мозг может заболеть после Проекта 17, поэтому отдохните, вытяните ноги и сделайте еще один глубокий вдох, прежде чем погрузиться в Проект 18, в котором мы теперь будем использовать два сдвиговых регистра, соединенных гирляндой.

Проект 18 - Сдвоенные 8-битные двоичные счетчики

В Проекте 18 мы будем использовать компоненты, перечисленные в Таблице 6-8, для каскадирования ИС 74HC595 для создания сдвоенного двоичного счетчика.

Требуемые детали

Таб. 6-8. *Детали, необходимые для проекта 18*

1 x 74HC595 регистр сдвига	
16 резисторов	
8 x красных LED	
8 x зеленых LED	
Конденсатор 0,1 мкФ	

Подключение

Первый 595 подключен так же, как и в Project 17. Второй 595 имеет + 5 В и провода заземления, идущие к тем же контактам, что и первый 595.

Те же выходы, что и на первом 595, идущем к первому набору светодиодов, переходят от второй ИС ко второму набору светодиодов.

Внимательно изучите схемы на рисунках 6-4 и 6-5.

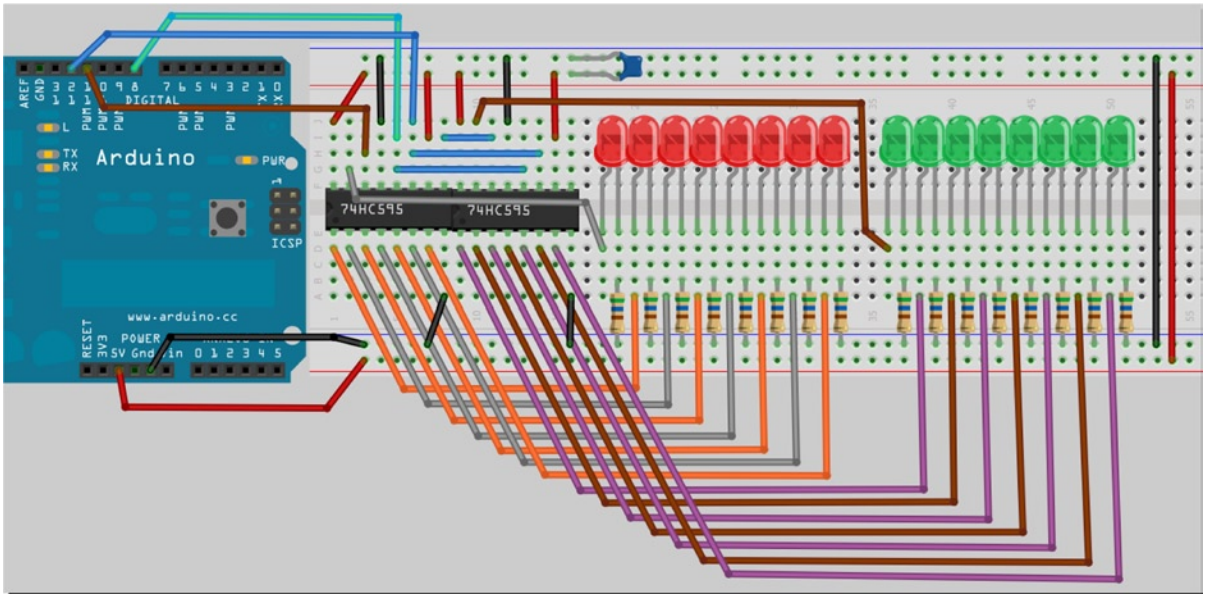


Рис. 6-4. *Схема соединений для проекта 18*

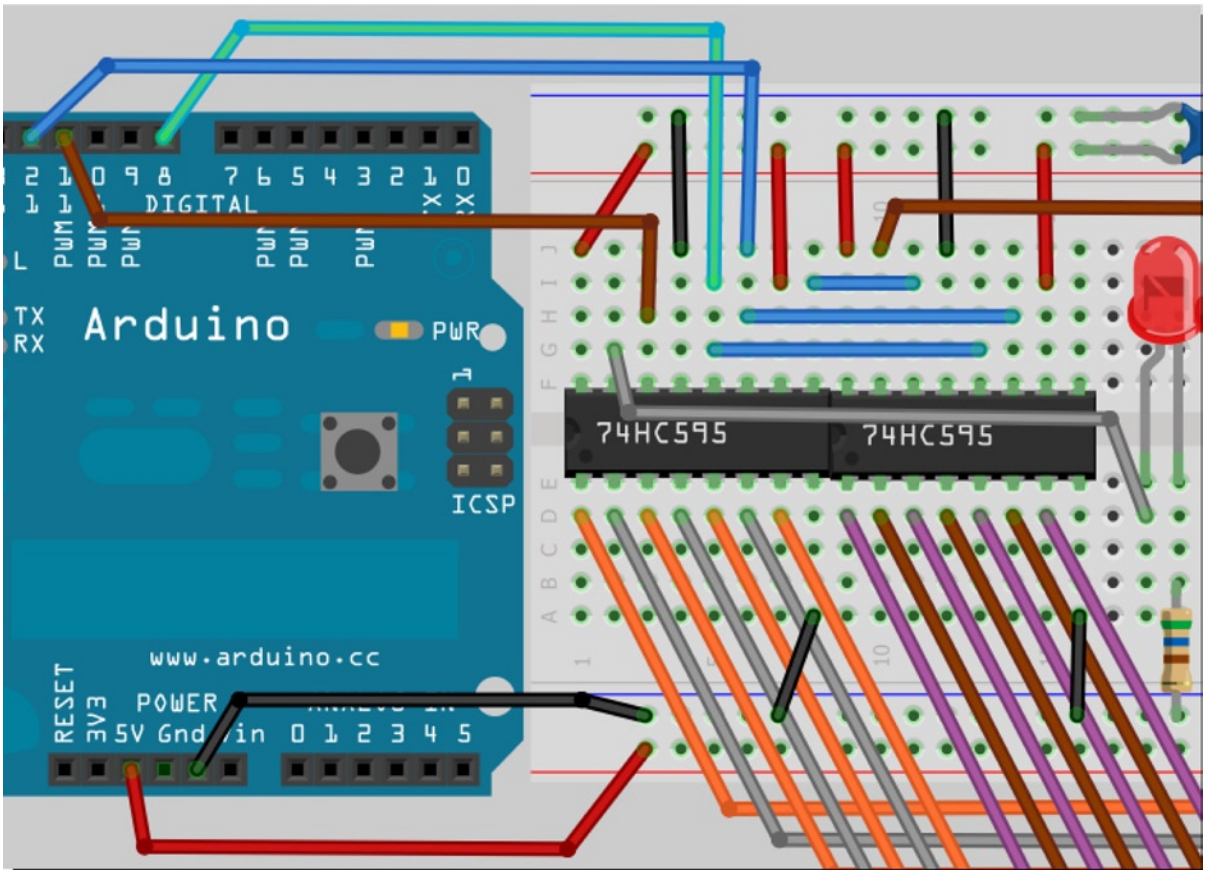


Рис. 6-5. *The circuit for Project 18. Close up of the wiring of the ICs*

Введите код

Enter the following code in Listing 6-2 and upload it to your Arduino. When you run this code, you will see the green set of LEDs count up (in binary) from 0 to 255 and the red LEDs count down from 255 to 0 at the same time.

Листинг 6-2. Код для проекта 18

```
// Проект 18

int latchPin = 8;  //Пин подключен к пину 12 74HC595 (защелка)(Latch)
int clockPin = 12; //Пин подключен к пину 11 of 74HC595 (Clock)
int dataPin = 11;  //Пин подключен к пину 14 of 74HC595 (Data)

void setup() {
        //устанавливаем пины для вывода
        pinMode(latchPin, OUTPUT);
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);
}
```

```
void loop() {
        for (int i = 0; i < 255; i++) {                          //считаем от 0 до 255
digitalWrite(latchPin, LOW); //устанавливаем latchPin на низкий уровень, чтобы разрешить поток данных
                shiftOut(dataPin, clockPin, LSBFIRST, i);        // сдвигаем первые 8 бит
                  shiftOut(dataPin, clockPin, LSBFIRST, 255-i); // shiftOut вторые 8 бит//установить
                  latchPin to high to lock and send data
                digitalWrite(latchPin, HIGH);
                delay(250 );
        }
}
```

Код проекта 18 и обзор устройства

Код проекта 18 очень похож на код проекта 17. Мы просто добавили вторую инструкцию для сдвига еще 8 бит во второй регистр сдвига. Однако вы сразу заметите, что, несмотря на то, что это более сложный проект, чем проект 17, у нас меньше кода. Как так? Здесь функция shiftDataOut была заменена на shiftOut () и что эта функция окрашена в красный цвет в среде Arduino IDE. Это связано стем, что команда shiftOut является неотъемлемой частью языка Arduino, поскольку сдвиг 8 бит данных в регистры сдвига или микросхемы контроллеров светодиодов настолько часто используется в мире Arduino, что возможность делать это была введена в языке.

Синтаксис функции shiftOut следующий:

shiftOut(dataPin, clockPin, bitOrder, value)

Параметры dataPin и clickPin были установлены в начале программы, и вы устанавливаете их pinMode на OUTPUT в цикле setup (). BitOrder решает, сдвигаются ли биты, начиная с самого старшего бита (крайний левый бит) или сначала с самого младшего бита (бит в крайнем правом из 8).

Мы используем слова MSBFIRST для первого старшего бита или LSBFIRST для наименее значимого первого бита . Мы хотим сначала отправить младший бит, поэтому мы будем использовать LSBFIRST. Наконец, последнее значение - это цифра, которую мы отправляем. Это байт, так как он составляет максимум 8 бит (0–255).

В Project 17 мы создали нашу собственную функцию shiftDataOut, которая делает то же самое, что и shiftOut. Причина, по которой я заставил вас проделать эту дополнительную работу, заключалась в том, чтобы вы могли точно видеть, что происходит и как работает сдвиговый регистр. Это важные знания для использования регистров сдвига. Однако с этого момента мы будем использовать функцию shiftOut.

В основном цикле процедура shiftOut отправляет 8 битов на 595. В основном цикле мы поместили два набора вызовов shiftOut, один из которых отправляет значение I, а другой - 255-i. Мы дважды вызываем shiftOut, прежде чем установить защелку в 1. Она отправит два набора по 8 бит, или всего 16 бит, на 595 микросхем, прежде чем защелка будет установлена в 1, чтобы скопировать содержимое регистра сдвига на выходные контакты, которые, в свою очередь, заставят светодиоды включаться или выключаться.

Второй 595 подключается точно так же, как и первый. Пины clock и latch привязаны к одним и тем же пинам первого 595. Однако у нас есть провод, идущий от пина 9 IC 1 к пину 14 IC 2. Пин 9 - для вывода данных, а пин 14 - для ввода данных.

Данные вводятся на вывод 14 первой микросхемы Arduino. Вторая микросхема 595 «гирляндно» подключена к первой микросхеме с помощью вывода 9 (Q7')на микросхеме 1, которая выводит данные, на вывод 14 на второй микросхеме, который является входом данных.

Что происходит, когда вы вводите 9-й бит и выше, данные в IC 1 переходят с его вывода данных на вывод данных второй IC. Итак, после того, как все 16 бит были отправлены по линии данных от Arduino, первые 8 отправленных бит перешли бы из первого чипа во второй. Второй чип 595 будет содержать ПЕРВЫЕ 8 отправленных бит, а первый чип 595 будет содержать ВТОРОЙ 8 бит или биты с 9 по 16.

Таким образом можно последовательно соединить почти неограниченное количество из 595 микросхем.

УПРАЖНЕНИЕ

упражнение. используя ту же схему для проекта 18 и всех 16 светодиодов, воссоздайте световой эффект KNIGHT RIDER (или CYION), заставляя светодиоды подпрыгивать вперед и назад через все 16 светодиодов.

Резюме

В главе 6 мы много говорили об использовании внешней ИС, чтобы дать нам дополнительные выходные выводы на нашем Arduino.
Хотя мы могли бы реализовать эти проекты без внешней ИС, регистры сдвига значительно облегчили нам жизнь. Если бы мы не использовали регистры сдвига, код для этих проектов был бы намного сложнее. Использование ИС, предназначенной для приема последовательных данных и их параллельного вывода, идеально подходит для управления линиями светодиодов таким образом и уменьшило сложность, а не увеличило ее.
Никогда не пугайтесь использования внешних микросхем. Медленное и методичное чтение таблиц данных покажет, как они работают.
В главе 7 мы продолжим использовать регистры сдвига, но на этот раз мы собираемся использовать их для управления светодиодными матричными дисплеями, которые содержат не менее 64 светодиодов на блок. Мы можем управлять большим количеством светодиодов одновременно, используя замечательную технику, известную как мультиплексирование, о которой мы узнаем в следующей главе.

Предметы и понятия, затронутые в главе 6

- Двоичная система счисления и способы преобразования в десятичную и обратно
- Как использовать регистр сдвига для ввода последовательных данных и вывода параллельных данных
- Использование внешней ИС для уменьшения сложности проекта
- Отправка параметра в вызов функции
- Использование логических переменных
- Понятие и использование побитовых операторов
- Использование побитовых операторов для создания битовой маски
- Как каскадировать (или последовательно подключать) два или более регистров сдвига
- Использование функции shiftOut ().

ГЛАВА 7

Светодиодные дисплеи

До сих пор вы имели дело с отдельными 5-мм светодиодами. Светодиоды также могут быть получены в корпусе, известном как точечно-матричный дисплей, наиболее популярным из которых является матрица из 8×8 светодиодов или 64 светодиода. Вы также можете получить двухцветные точечные матричные дисплеи (например, красный и зеленый) или даже точечные матричные дисплеи RGB, которые могут отображать любой цвет и содержат в общей сложности 192 светодиода в одном корпусе дисплея. В этой главе вы познакомитесь со стандартным одноцветным дисплеем с точечной матрицей 8×8 и узнаете, как отображать изображения и текст.Мы начнем с простой демонстрации создания анимированного изображения на дисплее 8×8, а затем перейдем к более сложным проектам отображения. Попутно мы узнаем о таком важном понятии как мультиплексирование.

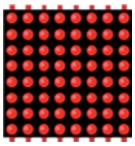
Проект 19 - Светодиодный точечно-матричный дисплей - Базовая анимация

В этом проекте мы опять используем две IC регистра сдвига 595. Они будут связаны со строками и столбцами точечного матричного дисплея. Затем мы представим на дисплее простой объект или спрайт и оживим его. Основная цель этого проекта - показать как работает матричный дисплей и познакомиться с понятием мультиплексирования.

Требуемые детали

Для этого проекта вам потребуются детали, перечисленные в Таб. 7-1.

Таб. 7-1. *Детали, необходимые для проекта 19*

2 × 74HC595 регистр сдвига IC	
8 резисторов (510 Ом)	
Матричный дисплей 8×8 с ОК	
конденсатор 0.1 мФ	

Номинал токоограничивающих резисторов будет зависеть от используемого светодиода. Ознакомьтесь с этим руководством, чтобы узнать, какой из них выбрать: https://www.sparkfun.com/tutorials/219

Подключение

Внимательно изучите схему соединении. Важно, чтобы вы не подключали Arduino к USB-кабелю или источнику питания, пока схема не будет завершена; иначе вы рискуете повредить IC или матричный дисплей. Производите соединения внимательно и не спеша.

Схема соединений на рис. 7-1 относится к конкретному модулю точечной матрицы, который я использовал при создании этого проекта - миниатюрному модулю с красной точечной матрицей 8×8. Однако ваш дисплей может (и, скорее всего,будет) иметь другое расположение пинов, чем у того, который я использовал. Вы должны прочитатьтехническое описание дисплея, который вы купили, чтобы убедиться, что соединения вашего устройства произведены без ошибок.

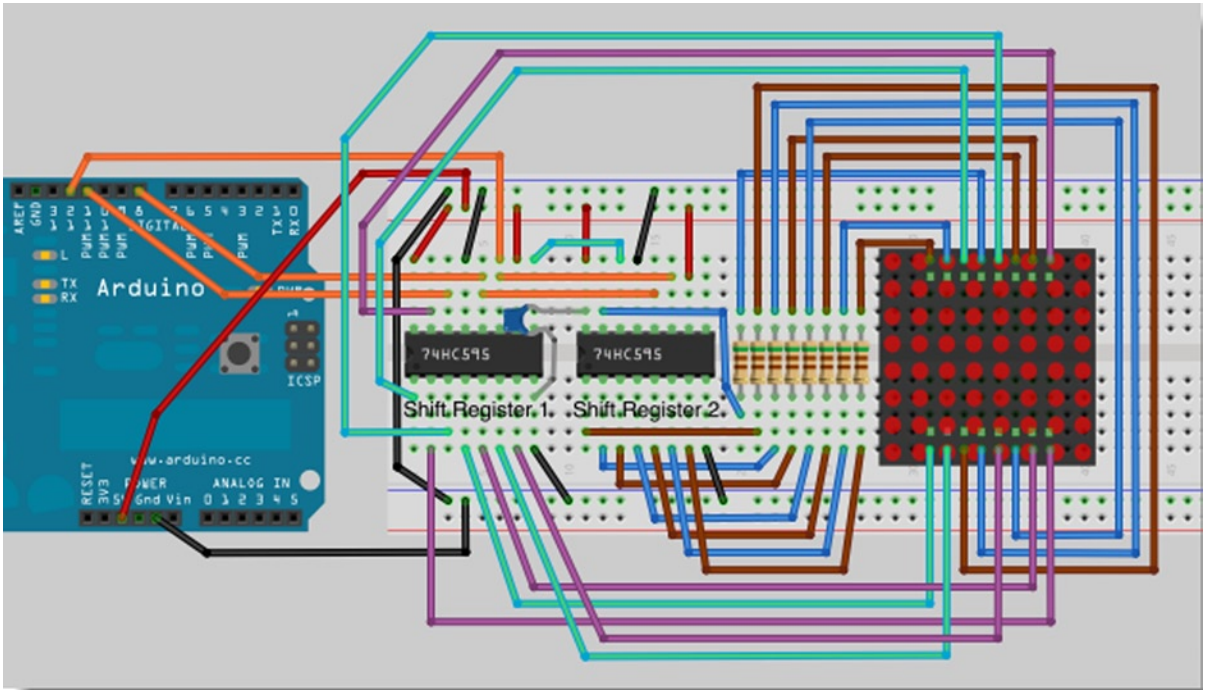


Рис. 7-1. *Схема для Проекта 19 - LED точечно-матричный дисплей - Базовая анимация*

Чтобы упростить эту задачу, в таб. 7-2 показано, какие выводы из регистров сдвига должны перейти к каким выводам на точечно-матричном дисплее. Выводы матрицы соответствуют моему конкретному дисплею (как показано на рисунке 7-2), поэтому отредактируйте схему соединении соответствующим образом, чтобы она соответствовала тому дисплею, который у вас есть.

Table 7-2. *Схема соединении точечно-матричного дисплея с IC*

Ряд	Пин регистра 1	Пин матрич. дисплея
Row 1	Pin 15	9
	Pin 1	14
Row 3	Pin 2	8
Row 4	Pin 3	12
Row 5	Pin 4	1
Row 6	Pin 5	7
Row 7	Pin 6	2
Row 8	Pin 7	5
Столбец	**Пин регистра 2**	**Пин матрич. дисплея**
Column 1	Pin 15	13
Column 2	Pin 1	3
Column 3	Pin 2	4
Column 4	Pin 3	10
Column 5	Pin 4	6
Column 6	Pin 5	11
Column 7	Pin 6	15
Column 8	Pin 7	16

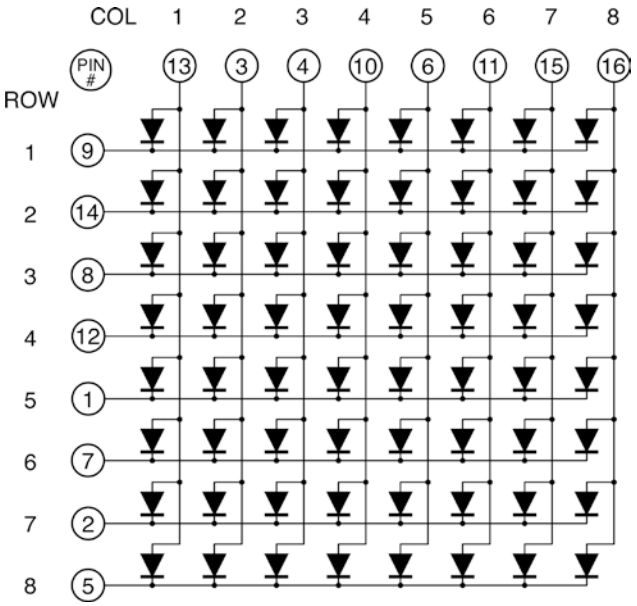


Рис. 7-2. *Типовая схема для светодиодного точечно-матричного дисплея* 8×8

Схема точечно-матричного дисплея, используемого для создания этого проекта, представлена на рисунке 7-2. Токоограничивающие резисторы размещены на связях между сдвиговым регистром и дисплеем. Вам необходимо тщательно выбрать значения токоограничивающих резисторов, чтобы они соответствовали используемому вами дисплею. См. Руководство по Sparkfun по адресу https://www.sparkfun.com/tutorials/219, чтобы узнать о формуле для расчета правильного значения. Вам нужно будет прочитать техническое описание имеющегося у вас дисплея и выполнить аналогичное упражнение, чтобы определить, какие выводы из регистра сдвига идут к каким контактам на светодиодном дисплее.

Введите код

Убедившись в правильности подключения, введите код из Листинга 7-1 и загрузите его в свой Arduino. Вам также потребуется загрузить библиотеку TimerOne. Его можно загрузить с веб-сайта Arduino поадресу www.arduino.cc/playground/Code/Timer1. После того, как вы загрузили библиотеку, распакуйте ее и поместите всю папку TimerOne в папку оборудования / библиотеки внутри установки Arduino (Arduino / СОДЕРЖАНИЕ / Resources / Java / библиотеки на Mac). Убедитесь, что папка называется просто TimerOne, удалите все остальные символы.

Это пример внешней библиотеки. В Arduino IDE предварительно загружено множество библиотек, таких как Ethernet, LiquidCrystal, Servo и т. д. Библиотека TimerOne - это внешняя библиотека, которую вам просто нужно загрузить и установить в папку с библиотеками, чтобы она работала (вам нужно будет перезапустить IDE до того, как она будет распознана).

Листинг 7-1. Код для проекта 19

```
// Проект 19
#include <TimerOne.h> // Сначала установим библиотеку, иначе код не будет работать

int latchPin = 8;     //Пин подключен к выводу 12 of 74HC595 (Latch)
int clockPin = 12;    //Пин подключен к выводу 11 74HC595 (Clock)
int dataPin = 11;     //Пин подключен к выводу 14 74HC595 (Data)

byte led[8];          // 8-элементный целочисленный массив без знака для хранения спрайта

void setup() {
  // устанавливаем 3 цифровых вывода на выходы
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  // Загружаем двоичное представление изображения в массив
  led[0] = B11111111;
  led[1] = B10000001;
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
  led[5] = B10111101;
  led[6] = B10000001;
  led[7] = B11111111;

  // устанавливаем таймер длиной 10000 микросекунд (1/100 секунды)
  // и присоединяем к таймеру прерывания функцию screenUpdate
  Timer1.initialize(10000);
  Timer1.attachInterrupt(screenUpdate);
}
```

```
//  инвертируем каждую строку двоичного изображения и ждем 1/4 секунды
void loop() {

  for (int i=0; i<8; i++) {
    led[i]= ~led[i];
  }
  delay (250);
}

// Отображаем изображение
void screenUpdate() {
 byte row = B10000000; // row 1
 for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, led[k] ); // массив светодиодов
      shiftOut(dataPin, clockPin, LSBFIRST, ~row);  // двоичный номер строки (активный минимум)

      // фиксируем от низкого к высокому для вывода данных
      digitalWrite(latchPin, LOW);
      digitalWrite(latchPin, HIGH);

      // битовый сдвиг вправо
      row = row >> 1;

 }
// Отключаем все строки до следующего прерывания
      shiftOut(dataPin, clockPin, LSBFIRST, 0);
      shiftOut(dataPin, clockPin, LSBFIRST, ~0);

      // защелкиваем от низкого к высокому для вывода данных
      digitalWrite(latchPin, LOW);
      digitalWrite(latchPin, HIGH);
}
```

После запуска кода вы увидите концентрические квадраты на дисплее. Примерно каждые четверть секунды дисплей будет инвертироваться, чтобы придать изображению базовый анимационный эффект. Помните, что для работы этого проекта вам понадобится библиотека Timer1.

Проект 19 - Светодиодная точечная матрица - Базовая анимация - Обзор схемы

Мы изучим LED матрицу, прежде чем будем разбираться, как работает код. Это упростит понимание кода.

Мы разобрали работу 74HC595 в предыдущих проектах. Единственным дополнением к ней является светодиодная матрица 8 × 8.

Светодиодные дисплеи (т.е. LED-дисплеи) часто представляют собой матрицу, состоящей из группы светодиодов – с общими анодами в рядах и общими катодами в столбцах см. рис.7.2 (или наоборот).

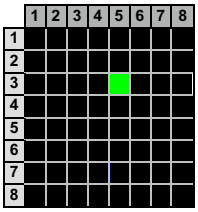
Типичный одноцветный матричный блок 8 × 8 будет иметь 16 контактов, по 8 для каждого ряда и 8 для каждого столбца.

Чтобы включить «столбцовый» светодиод, ему нужно отправить значение LOW. А чтобы включить «рядный» светодиод, ему нужно отправить значение HIGH. Если светодиоды, будь они хоть в рядах, хоть в столбцах, будут получать только значение HIGH или только значение LOW, разницы потенциалов не возникнет, поэтому в итоге светодиоды так и останутся не включенными.

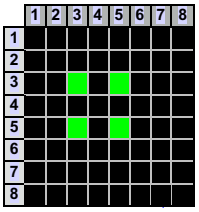
Контакты, которые при помощи функции pinMode() задали как выходные (OUTPUT), по умолчанию имеют значение LOW.
Причина, по которой светодиоды соединены вместе по строкам и по столбцам, состоит в том, чтобы минимизировать количество требуемых контактов.
Если бы это было не так, то для одноцветного блока с точечной матрицей 8×8 потребовалось бы 65 контактов, по одному для каждого светодиода и общая анодная или катодная шина. При соединении строк истолбцов вместе требуется всего 16 контактов.

Предположим, стоит задача зажечь светодиод $R3C5$. Для этого нам потребуется подать высокий уровень сигнала на строку $R3$, а столбец $C5$ соединить с землей.

Теперь загорится светодиод в 5-м столбце и 3-м ряду.



Теперь представим, что мы хотим также зажечь светодиод в столбце 3, строке 5. Итак, мы прикладываем положительное напряжение к 5 ряду и заземляем 3 столбец.
Теперь загорится светодиод встолбце 3, строке 5. Но подождите ... светодиоды в столбце 3, строке 3 и столбце 5, строке 5 такжезагорелись.



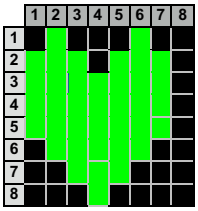
Это связано с тем, что вы подаете питание на ряды 3 и 5 и заземляете столбцы 3 и 5. Казалось бы, невозможно зажечь только два необходимых светодиода с соединенными вместе строками и столбцами. Очевидно, что помочь может динамическая индикация. Если мы будем включать точки R3C3 и R3C5 по-очереди очень быстро, то сможем использовать инерцию зрения — способность интерпретировать быстро сменяющиеся изображения как одно целое.

Мультиплексирование

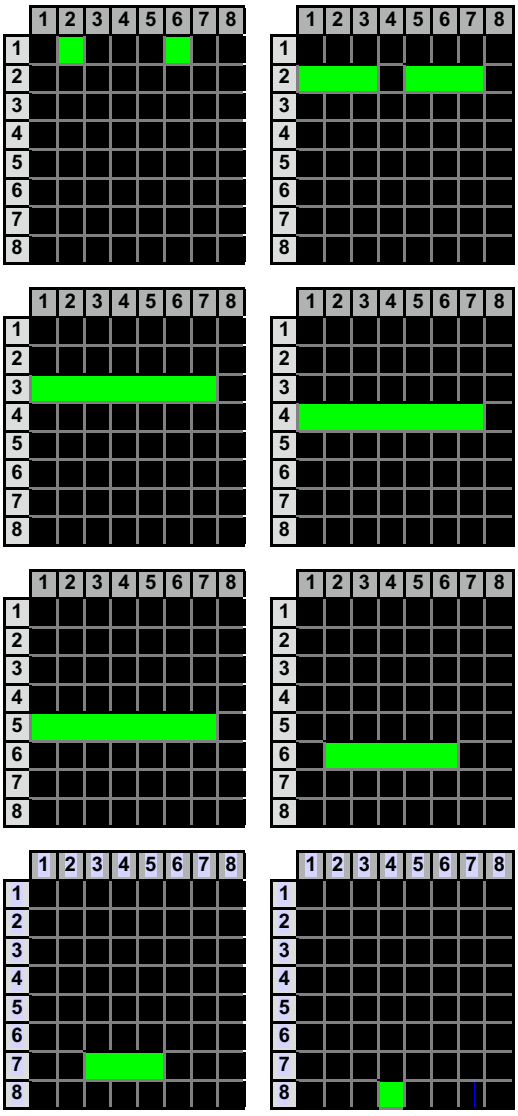
Применимо к нашей схеме мы используем мультиплексирование. С использованием этой технологии в каждый момент времени мы будем переключать только одну строку светодиодной матрицы, а далее у нас будет задействован непрерывный цикл по всем 8 строкам. Для человеческого глаза это будет незаметно.

Используя этот метод, вы обойдете проблему отображения отдельных светодиодов без включения других светодиодов в том же столбце или строке.

Например, вы хотите вывести на дисплей следующее изображение:



Тогда каждая строка будет светиться по очереди так:



Сканируя строки сверху вниз и зажигая соответствующие светодиоды в каждом столбце этой строки и делая это очень быстро (более 100 Гц), человеческий глаз будет воспринимать изображение как устойчивое.

Проект 19 - Светодиодная точечная матрица - Базовая анимация - Обзор кода

В коде этого проекта используется функция IC ATmega, называемая аппаратным таймером. По сути, это таймер на микросхеме, который можно использовать для запуска события. Эти события называются прерываниями, потому что они заставляют процессор прерывать код, который он обрабатывает, и переходит к обработке функции (подпрограммы обслуживания прерывания (ISR)) для обработки события.

Когда он завершает обработку ISR, он возобновляет обработку программы с того места, где она была, когда произошло прерывание. Это очень похоже на то, как человек прерывает то, что он делает, чтобы ответить на телефонный звонок, а затем возвращается к тому, что он делал. В нашем случае мы устанавливаем таймер для генерации события каждые 10000 микросекунд, то есть каждые 100-е секунды.

Мы используем TimerOne. TimerOne упрощает создание ISR. Вы просто указываете функции интервал (в данном случае 10000 микросекунд) и имя функции, которую хотите активировать каждый раз, когда срабатывает прерывание (в данном случае это функция screenUpdate ()). Обратите внимание, что процедуры обслуживания прерываний должны быть короткими (меньше времени между прерываниями), иначе процессор никогда не вернется к основному коду. (Или, что еще хуже, переполнится стек.) TimerOne - это внешняя библиотека, поэтому вам необходимо включить ее в свой код. Это легко сделать с помощью команды include:

```
#include <TimerOne.h>
```

Затем объявляются пины, используемые для взаимодействия со сдвиговыми регистрами:

```
int latchPin = 8;          //Пин подключен к выводу 12 74HC595 (Latch-защелка)
int clockPin = 12;         //Пин подключен к выводу 11 74HC595 (Clock-такты)
int dataPin = 11;         //Пин подключен к выводу 14 74HC595 (Data-данные)
```

Затем создаем массив типа byte, состоящий из восьми элементов. Массив led [8] будет использоваться для хранения изображения, которое мы будем отображать на точечно-матричном дисплее:

```
byte led[8];            // 8-элементный байтовый массив для хранения спрайта
```

В программе настройки вы устанавливаете пины защелки, синхронизации и данных в качестве выходов:

```
void setup() {
        pinMode(latchPin, OUTPUT);  // устанавливаем 3 цифровых пина на выходы
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);
```

После того, как пины будут установлены на выходы, в матрицу светодиодов будут загружены 8-битные двоичные изображения, которые будут отображаться в каждой строке матричного дисплея 8×8:

```
led[0] = B11111111;                 // вводим двоичное представление изображения
led[1] = B10000001;                 // в массив
led[2] = B10111101;
led[3] = B10100101;
led[4] = B10100101;
led[5] = B10111101;
```

```
led[6] = B10000001;
led[7] = B11111111;
```

Посмотрев на массив выше, вы можете различить изображение, которое будет отображаться, которое представляет собой прямоугольник внутри прямоугольника.Единицы указывают, где будет гореть LED, и 0, где он будет выключен. Конечно, вы можете сами создать любой спрайт 8×8, который вам нужен.
После этого используется объект Timer1. Во-первых, таймер должен быть инициализирован с частотой, с которой он будет активирован. В этом случае вы устанавливаете период на 10000 микросекунд или $1/100$ секунды. После инициализации прерывания вам необходимо присоединить к прерыванию функцию, которая будет выполняться каждый раз при достижении периода времени.
Это функция screenUpdate (), которая запускается каждые $1/100$ секунды:

```
// устанавливаем таймер длиной 10000 микросекунд (1/100 секунды)
Timer1.initialize(10000);
// присоединяем функцию screenUpdate к таймеру прерывания
Timer1.attachInterrupt(screenUpdate);
```

В основном цикле цикл for проходит по каждому из восьми элементов массива светодиодов и инвертирует содержимое с помощью побитового оператора ~ или NOT, превращая все 1 в 0 и все 0 в 1. Затем он ждет 250 миллисекунд перед повторением.

```
for (int i=0; i<8; i++) {
led[i]= ~led[i]; // инвертируем каждую строку двоичного изображения
}
delay(250);
```

Теперь у вас есть функция screenUpdate (). Это функция, которая активируется прерыванием каждые 100 долей секунды. Вся эта процедура очень важна, потому что она отвечает за правильное включение светодиодов в матричном массиве и отображение изображения, которое вы хотите передать. Это очень простая, но эффективная функция.

```
void screenUpdate() {           // функция для вывода изображения
// Display the image
byte row = B10000000;                                   // ряд 1
 for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, led[k] ); // массив светодиодов (инвертированный)
      shiftOut(dataPin, clockPin, LSBFIRST, row);     // двоичное число строки

      digitalWrite(latchPin, LOW);     // фиксируем от низкого к высокому для вывода данных
      digitalWrite(latchPin, HIGH);
      row = row >> 1;                                   //битовый сдвиг вправо
 }
// Clear the matrix
row = B10000000;                                        // ряд 1
 for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, 255);     // массив светодиодов (инвертированный)
      shiftOut(dataPin, clockPin, LSBFIRST, row);     // двоичное число строки

      digitalWrite(latchPin, LOW);        // фиксируем от низкого к высокому для вывода данных
      digitalWrite(latchPin, HIGH);
      row = row >> 1;                                   // битовый сдвиг вправо
 }
}
```

Объявляется байт с именем row и инициализируется значением B10000000:

```
byte row = B10000000;                                    // ряд 1
```

Теперь мы циклически просматриваем светодиодный массив и отправляем эти данные в регистры сдвига (которые обрабатываются с помощью побитового НЕ ~, чтобы убедиться, что столбцы, которые мы хотим отобразить, выключены или заземлены), за которой следует строка:

```
for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, led[k] ); // массив светодиодов (инвертированный)
      shiftOut(dataPin, clockPin, LSBFIRST, row);     // двоичное число строки

      digitalWrite(latchPin, LOW);    // защелкиваем от низкого к высокому для вывода данных
      digitalWrite(latchPin, HIGH);
      row = row >> 1;                // битовый сдвиг вправо
 }
```

После того, как мы сдвинули 8 бит текущей строки, значение в строке сдвигается на одно место вправо, так что отображается следующая строка (т.е. отображается только строка с цифрой 1). Мы узнали о команде битового сдвига в Главе 6.

```
row = row >> 1;                                          // битовый сдвиг вправо
```

Затем мы повторяем описанное выше, но на этот раз заземляем все строки, чтобы экран очистился. Это важно для того, чтобы дисплей не оставался включенным слишком долго, пока он ожидает выполнения следующей ISR, так как это приведет к тому, что наша последняя строка будет ярче, чем остальные.

```
row = B10000000;                                         // ряд 1
 for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, 255);   // массив светодиодов (инвертированный)
      shiftOut(dataPin, clockPin, LSBFIRST, row);     // двоичное число строки

      digitalWrite(latchPin, LOW);                    // фиксируем от низкого к высокому
      digitalWrite(latchPin, HIGH);
      row = row >> 1;                                 // битовый сдвиг вправо
 }
```

Помните из обзора устройства, что процедура мультиплексирования отображает только одну строку за раз, выключает ее и затем отображает следующую строку. Это мерцание происходит на частоте 100 Гц, что слишком быстро для человеческого глаза.

Итак, основная концепция здесь заключается в том, что у вас есть процедура прерывания, которая выполняется каждые 100 долей секунды. В этой процедуре вы просто просматриваете содержимое массива экранных буферов (в данном случае led []) и отображаете его на матричном модуле по одной строке за раз, но делается это так быстро, что кажется, что все зажигается сразу.

Основной цикл программы - просто изменить содержимое массива экранных буферов, а все остальное сделает ISR.

Анимация в этом проекте очень проста, но, манипулируя единицами и нулями в буфере, мы можем заставить все, что угодно, от фигур до прокручиваемого текста, появиться на модуле точечной матрицы. Давайте попробуем этот вариант в следующем проекте; мы создадим анимированный спрайт с прокруткой.

Проект 20 - Светодиодный точечно-матричный дисплей - Спрайт с прокруткой

Мы собираемся использовать ту же схему, но с небольшими изменениями в коде, чтобы создать многокадровую анимацию, которая также прокручивается справа налево. При этом мы познакомимся с понятиями многомерных массивов. Мы также узнаем о небольшом трюке, чтобы получить побитовое вращение (или круговой сдвиг). Для начала мы будем использовать ту же схему, что и в Project 19.

Введите код

Введите и загрузите код из Листинга 7-2.

Листинг 7-2. Код для проекта 20

```
// Проект 20
#include <TimerOne.h>

int latchPin = 8;  //Пин подключен к выводу 12 74HC595(Latch - защелка)
int clockPin = 12; //Пин подключен к контакту 11 74HC595 (Clock - такты)
int dataPin = 11;  //Пин подключен к выводу 14 74HC595 (Data-данные)
byte frame = 0;    // переменная для хранения текущего отображаемого кадра
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},            // 8 кадров анимации

                   {0, 56, 124, 186, 146, 130, 68, 56},
                   {0, 56, 116, 242, 242, 130, 68, 56},
                   {0, 56, 68, 226, 242, 226, 68, 56},
                   {0, 56, 68, 130, 242, 242, 116, 56},
                   {0, 56, 68, 130, 146, 186, 124, 56},
                   {0, 56, 68, 130, 158, 158, 92, 56},
                   {0, 56, 68, 142, 158, 142, 68, 56} };

void setup() {
        pinMode(latchPin, OUTPUT);                // устанавливаем 3 цифровых пина на выходы
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);

        Timer1.initialize(10000);             // устанавливаем таймер длиной 10000 микросекунд
        Timer1.attachInterrupt(screenUpdate);      // присоединяем функцию screenUpdate
}

void loop() {
        for (int i=0; i<8; i++) {                 // перебираем все 8 кадров анимации
                for (int j=0; j<8; j++) {         // перебираем 8 строк в кадре
                        led[i][j]= led[i][j] << 1 | led[i][j] >> 7;    // побитовое вращение
                }
        frame++;                        // переходим к следующему кадру анимации
        if (frame>7) { frame =0;}    // убедитесь, что мы вернулись к кадру 0 после 7
            delay(100);                               // ждем немного между кадрами
}
```

```
void screenUpdate() {                              // функция для вывода изображения
   byte row = B10000000;                           // ряд 1
   for (byte k = 0; k < 8; k++) {

      shiftOut(dataPin, clockPin, LSBFIRST, led[frame][k] );   // массив светодиодов
      shiftOut(dataPin, clockPin, LSBFIRST, ~row); // выбор строки (активный минимум)

      // создаем переход от низкого к высокому на latchPin для передачи вывода на дисплей
      digitalWrite(latchPin, LOW);
      digitalWrite(latchPin, HIGH);
      row = row >> 1;                              // битовый сдвиг вправо
   }

   // выключаем все строки до следующего прерывания таймера, чтобы последняя строка не
   оставалась дольше других
   shiftOut(dataPin, clockPin, LSBFIRST, 0 ); // столбец не имеет значения со всеми строками
   shiftOut(dataPin, clockPin, LSBFIRST, ~0);      // не выбираем строку

   // создаем переход от низкого к высокому на latchPin для передачи вывода на дисплей
   digitalWrite(latchPin, LOW);
   digitalWrite(latchPin, HIGH);
}
```

Когда мы запустим Проект 20, мы увидим, как движется анимированное колесо. Схема не изменилась, поэтому мы не будем её обсуждать. Посмотрим, как работает этот код.

Проект 20 - Светодиодная точечная матрица - Спрайт с прокруткой - Обзор кода

Мы снова загружаем библиотеку TimerOne и устанавливаем три пина, которые управляют регистрами сдвига:

```
#include <TimerOne.h>

int latchPin = 8;  //Pin connected to Pin 12 of 74HC595 (Latch)
int clockPin = 12; //Pin connected to Pin 11 of 74HC595 (Clock)
int dataPin = 11;  //Pin connected to Pin 14 of 74HC595 (Data)
```

Затем мы объявляем переменную типа byte и инициализируем ее нулем. Это сохранит номер текущего отображаемого кадра восьмикадровой анимации:

```
byte frame = 0;    // переменная для хранения текущего отображаемого кадра
```

Затем мы настраиваем двумерный массив типа byte:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},       // 8 кадров анимации
                   {0, 56, 124, 186, 146, 130, 68, 56},
                   {0, 56, 116, 242, 242, 130, 68, 56},
                   {0, 56, 68, 226, 242, 226, 68, 56},
                   {0, 56, 68, 130, 242, 242, 116, 56},
                   {0, 56, 68, 130, 146, 186, 124, 56},
                   {0, 56, 68, 130, 158, 158, 92, 56},
                   {0, 56, 68, 142, 158, 142, 68, 56} };
```

Массивы были представлены в главе 3. Массив - это набор переменных, доступ к которым осуществляется с помощью номера индекса. Этот массив отличается тем, что в нем есть два набора чисел для элементов. В главе 3 мы объявили одномерный массив следующим образом:

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

Здесь нам нужно создать двумерный массив с двумя наборами индексных номеров. В этом случае наш массив составляет 8 X 8 или всего 64 элемента. Двумерный массив почти такой же, как двумерная таблица, поскольку мы можем получить доступ к одной ячейке, указав номер строки и столбца соответственно. Таблица 7-3 показывает нам, как получить доступ к элементам в вашем массиве.

Таб. 7-3. *Элементы в нашем массиве*

	0	1	2	3	4	5	6	7
0	0	56	92	158	158	130	68	56
1	0	56	124	186	146	130	68	56
2	0	56	116	242	242	130	68	56
3	0	56	68	226	242	226	68	56
4	0	56	68	130	242	242	116	56
5	0	56	68	130	146	186	124	56
6	0	56	68	130	158	158	92	56
7	0	56	68	142	158	142	68	56

Строки представляют собой первое число индекса массива, то есть байт led [7] [..], а столбцы представляют второе число индекса массива, то есть байт led [..] [7]. Чтобы получить доступ к числу 158 в 6-й строке и 4-м столбце, мы должны использовать байт led [5] [3]. Помните, что индексы начинаются с 0.

Обратите внимание, что при объявлении массива мы воспользовались возможностью одновременно инициализировать его данными.

Чтобы сделать это с двумерным массивом, мы помещаем все данные в фигурные скобки, а каждый набор данных из второго индекса в свою фигурную скобку с запятой после нее, например:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},
                   {0, 56, 124, 186, 146, 130, 68, 56},
                   {0, 56, 116, 242, 242, 130, 68, 56}, // и т.д.
```

Двумерный массив будет хранить восемь кадров вашей анимации. Первый индекс массива будет ссылаться на кадр анимации, а второй индекс будет составлять из 8 строк 8-битных чисел схему включения и выключения светодиодов. Для экономии места в коде числа преобразованы из двоичных в десятичные. Если бы мы могли видеть двоичные числа,то увидели бы следующую анимацию на рис. 7-3.

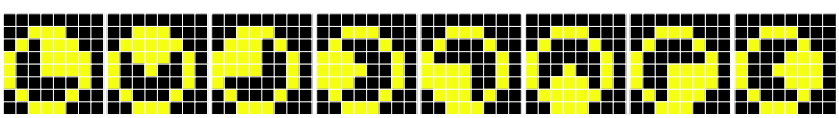


Рис. 7-3. *Анимация катящегося колеса*

Конечно, мы можем изменить эту анимацию на все, что захотим, а также увеличить или уменьшить количество кадров. Нарисуйте свою анимацию на миллиметровой бумаге, а затем преобразуйте строки в 8-битные двоичные числа и поместите их в свой массив.
В функции настройки мы снова устанавливаем три пина для вывода, создаем объект таймера длиной 10000 микросекунд и присоединяем функцию screenUpdate () к прерыванию:

```
void setup() {
        pinMode(latchPin, OUTPUT);                  // устанавливаем 3 цифровых пина на выходы
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);

        Timer1.initialize(10000);             // устанавливаем таймер длиной 10000 микросекунд
        Timer1.attachInterrupt(screenUpdate);  // присоединяем функцию screenUpdate
}
```

В основном цикле, как и в Project 19, мы просматриваем восемь строк спрайта. Однако этот цикл находится внутри другого цикла, который повторяется восемь раз, и этот цикл контролирует, какой кадр вы хотите отобразить:

```
void loop() {
        for (int i=0; i<8; i++) {           // перебираем все 8 кадров анимации
                for (int j=0; j<8; j++) {   // перебираем 8 строк в кадре
```

Затем мы получаем каждый элемент в массиве один за другим и сдвигаем значение влево на один. Однако, используя небольшой логический трюк, мы гарантируем, что любой бит, смещенный с левой стороны, перекатывается обратно в правую сторону. Это делается с помощью следующей команды:

```
led[i][j]= led[i][j] << 1 | led[i][j] >> 7; // побитовое вращение
```

Здесь текущий элемент массива, выбранный целыми числами i и j, сдвигается на одну позицию влево. Однако затем мы берем этот результат и логически ИЛИ складываем число со значением led [i] [j], которое было сдвинуто на семь позиций вправо. Давайте посмотрим, как это работает

Допустим, текущее значение led [i] [j] - 156. Это двоичное число 10011100. Если это число сдвинуть влево на единицу, вы получите 00111000.Теперь вы берете то же число, 156, и сдвигаете его вправо семь раз. Теперь у вас 00000001. Другими словами, вы переместили крайнюю левую двоичную цифру из левой части в правую.Теперь вы выполняете побитовую операцию логического ИЛИ над двумя числами. Помните, что побитовое сложение ИЛИ даст единицу, если в любой цифре есть единица, например:

```
00111000  |
00000001  =
--------
00111001
```

Итак, мы переместили число влево на одну позицию, и логическим OR'ed - ИЛИ сложилии это с тем же номером, сдвинули вправо на семь позиций.

Как вы можете видеть выше, результат такой же, как при смещении числа влево на единицу и смещении любой цифры, которая сместилась слева, обратно на правую сторону. Это известно как побитовое вращение или круговой сдвиг, и этот метод часто используется в цифровой криптографии.
Вы можете выполнить побитовое вращение для целочисленного типа данных без знака любой длины, используя вычисление

```
i << n | i >> (a - n);
```

где n - количество цифр, на которое мы хотим повернуть число, а a - длина в битах вашей исходной цифры.

Затем мы увеличиваем значение кадра на единицу, проверяем, что оно не больше семи, и, если да, снова устанавливаем нулевое значение. Оно будет циклически проходить через каждый из восьми кадров анимации один за другим, пока мы не дойдем до конца кадров, а затем повторяем. Наконец, есть задержка в100 миллисекунд.

```
frame++;                              // переходим к следующему кадру анимации
if (frame>7) { frame =0;}             // убедитесь, что мы вернулись к кадру 0 после 7
delay(100);                           // ждем немного между кадрами
```

Затем мы запускаем функции screenUpdate () и shiftOut, как и в предыдущих проектах на основе регистра сдвига.

В следующем проекте мы снова будем использовать светодиодную матрицу, но на этот раз мы не будем использовать регистры сдвига,а будем использовать популярный чип MAX7219.

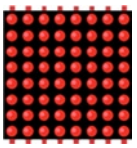
Проект 21 - LED точечно-матричный дисплей - Прокручиваемое сообщение

Есть много разных способов управления светодиодами. Использование регистров сдвига - это один из способов, и у них есть свои преимущества. Тем не менее, существует множество доступных ИС, которые специально разработаны для управления светодиодными дисплеями и значительно облегчают нам жизнь. Одна из самых популярных микросхем драйверов светодиодов в сообществе Arduino - это 8-разрядные микросхемы драйвера светодиодных дисплеев MAX7219 с последовательным интерфейсом, изготовленные компанией Maxim. Эти микросхемы предназначены для управления 7-сегментными цифровыми светодиодными дисплеями до 8 цифр, гистограммами или матричными светодиодными дисплеями 8 × 8, для чего мы и будем их использовать. IDE Arduino поставляется с библиотекой под названием Matrix, а также с некоторыми примерами кода, специально написанными для микросхем MAX7219. Использование этой библиотеки упростит использование этих микросхем. Однако в этом проекте мы не собираемся использовать какие-либо внешние библиотеки.
Вместо этого мы поступим сложным образом и будем писать каждый фрагмент кода самостоятельно. Таким образом, вы узнаете, как именно работает микросхема MAX7219, и сможемперенести эти навыки на использование любой другой микросхемы драйвера светодиода по нашему желанию.

Требуемые детали

Мы используем чип LED драйвер MAX7219. В качестве альтернативы можно использовать AS1107, который в значительной степени идентичен MAX7219 и будет работать без изменений вашего кода или схемы. Точечно-матричный дисплей 8x8 должен быть с ОК , поскольку мы выводим символы по одной строке за раз на сегментных линиях,которые предназначены для управления анодами светодиодов. См. полный список деталей в Таблице 7-4.

Таб. 7-4. *Требуемые детали для проекта 21*

Конденсатор 0,1 мФ	
MAX7219 (или AS1107)	
Резистор (39кОм)	
Матричный дисплей 8 × 8 (ОК)	

Подключение

Внимательно изучите схему соединении на рис. 7-4. Убедитесь, что ваш Arduino выключен при подключении проводов. Соединения от MAX7219 к точечно-матричному дисплею на рис. 7-4 построены для блока дисплея, который я использовал для создания этого проекта. Распиновка вашего дисплея вполне может быть другой. Поэтому вместо того, чтобы полагаться на соединения, показанные на Рисунке 7-4, подключите выводы, выходящие из MAX7219, к соответствующим выводам столбцов и строк на вашем дисплее (см. Схему выводов в Таблице 7-5). Чтение по горизонтали покажет, какие два устройства подключены и к каким контактам. На дисплее столбцы - это катоды, а строки - аноды. На моем дисплее я обнаружил, что строка 1 находится внизу, а строка 8 - вверху. Возможно, вам придется изменить порядок на вашем собственном дисплее, если вы обнаружите, что буквы перевернуты или инвертированы. Подключите 5 В от Arduino к положительной шине макета, а землю к шине заземления.

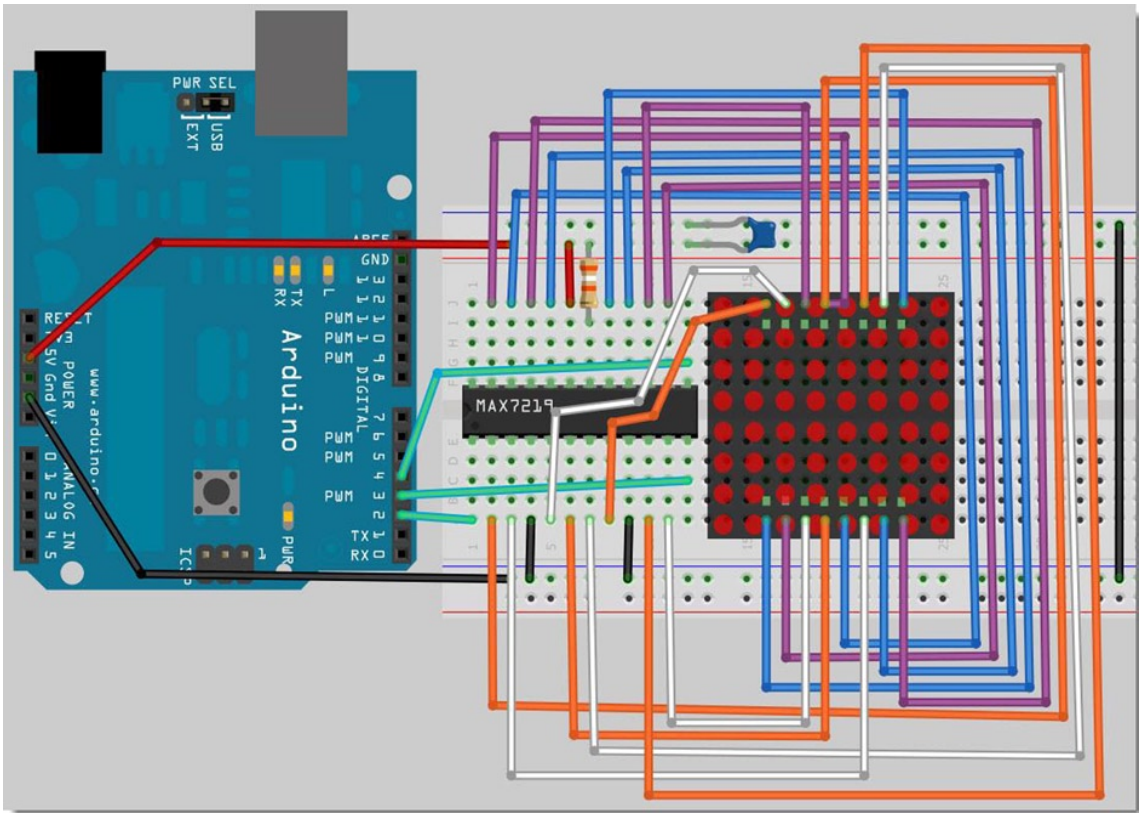


Рис. 7-4. *Схема для проекта 21*

Таб. 7-5. *Распиновка между Arduino, IC и точечно-матричным дисплеем*

Arduino	MAX7219	Display	Other
Digital 2	1 (DIN)		
Digital 3	12 (LOAD)		
Digital 4	13 (CLK)		
	4, 9		Gnd
	19		+5v
	18 (ISET)		Резистор с +5v
	2 (DIG 0)	Столбец 1	
	11 (DIG 1)	Столбец 2	
	6 (DIG 2)	Столбец 3	
	7 (DIG 3)	Столбец 4	
	3 (DIG 4)	Столбец 5	
	10 (DIG 5)	Столбец 6	
	5 (DIG 6)	Столбец 7	
	8 (DIG 7)	Столбец 8	
	22 (SEG DP)	Ряд 1	
	14 (SEG A)	Ряд 2	
	16 (SEG B)	Ряд 3	
	20 (SEG C)	Ряд 4	
	23 (SEG D)	Ряд 5	
	21 (SEG E)	Ряд 6	
	15 (SEG F)	Ряд 7	
	17 (SEG G)	Ряд 8	

Перед включением Arduino проверьте свои соединения.

Введите код

Введите и загрузите код из Листинга 7-3.

Листинг 7-3. Код для проекта 21

```
#include <avr/pgmspace.h>
#include <TimerOne.h>

int DataPin = 2;   // Пин 1 на MAX
int LoadPin = 3;   // Пин 12 на MAX
int ClockPin = 4;  // Пин 13 на MAX
byte buffer[8];
```

```
#define SCAN_LIMIT_REG 0x0B
#define DECODE_MODE_REG 0x09
#define SHUTDOWN_REG 0x0C
#define INTENSITY_REG 0x0A

static byte font[][8] PROGMEM = {
// TТолько печатаемые символы ASCII (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
{B00001010, B00001010, B00001010, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00001010, B00011111, B00001010, B00011111, B00001010, B00011111, B00001010},
{B00000111, B00001100, B00010100, B00001100, B00000110, B00000101, B00000110, B00011100},
{B00011001, B00011010, B00000010, B00000100, B00000100, B00001000, B00001011, B00010011},
{B00000110, B00001010, B00010010, B00010100, B00001001, B00010110, B00010110, B00001001},
{B00000100, B00000100, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000010, B00000100, B00001000, B00001000, B00001000, B00001000, B00000100, B00000010},
{B00001000, B00000100, B00000010, B00000010, B00000010, B00000010, B00000100, B00001000},
{B00010101, B00001110, B00011111, B00001110, B00010101, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000100, B00000100, B00011111, B00000100, B00000100, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000110, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00000000, B00001110, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000100},
{B00000001, B00000010, B00000010, B00000100, B00000100, B00001000, B00001000, B00010000},
{B00001110, B00010001, B00010011, B00010001, B00010101, B00010001, B00011001, B00001110},
{B00000100, B00001100, B00010100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00001000, B00010000, B00011111},
{B00001110, B00010001, B00000001, B00001110, B00000001, B00000001, B00010001, B00001110},
{B00010000, B00010000, B00010100, B00010100, B00011111, B00000100, B00000100, B00000100},
{B00011111, B00010000, B00010000, B00011110, B00000001, B00000001, B00000001, B00011110},
{B00000111, B00001000, B00010000, B00011110, B00010001, B00010001, B00010001, B00001110},
{B00011111, B00000001, B00000001, B00000001, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00001110, B00010001, B00010001, B00010001, B00001110},
{B00001110, B00010001, B00010001, B00001111, B00000001, B00000001, B00000001, B00000001},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00000000},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00001000},
{B00000001, B00000010, B00000100, B00001000, B00001000, B00000100, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00011110, B00000000, B00011110, B00000000, B00000000},
{B00010000, B00001000, B00000100, B00000010, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00000100, B00000000, B00000100},
{B00001110, B00010001, B00010001, B00010101, B00010101, B00010001, B00010001, B00011110},
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
{B00011110, B00010001, B00010001, B00011110, B00010001, B00010001, B00010001, B00011110},
{B00000111, B00001000, B00010000, B00010000, B00010000, B00010000, B00001000, B00000111},
{B00011100, B00010010, B00010001, B00010001, B00010001, B00010001, B00010010, B00011100},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00011111},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010000, B00010000, B00010111, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001, B00010001},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00010100, B00001000},
{B00010001, B00010010, B00010100, B00011000, B00010100, B00010010, B00010001, B00010001},
{B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00011111},
```

```
{B00010001, B00011011, B00011111, B00010101, B00010001, B00010001, B00010001, B00010001},
{B00010001, B00011001, B00011001, B00010101, B00010101, B00010011, B00010011, B00010001},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00011110, B00010001, B00010001, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010101, B00010011, B00001111},
{B00011110, B00010001, B00010001, B00011110, B00010100, B00010010, B00010001, B00010001},
{B00001110, B00010001, B00010000, B00001000, B00000110, B00000001, B00010001, B00001110},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010101, B00010101, B00001010},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00001010, B00010001, B00010001},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00011111, B00000001, B00000010, B00000100, B00001000, B00010000, B00010000, B00011111},
{B00001110, B00001000, B00001000, B00001000, B00001000, B00001000, B00001000, B00001110},
{B00010000, B00001000, B00001000, B00000100, B00000100, B00000010, B00000010, B00000001},
{B00001110, B00000010, B00000010, B00000010, B00000010, B00000010, B00000010, B00001110},
{B00000100, B00001010, B00010001, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00011111},
{B00001000, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00001110, B00010010, B00010010, B00010010, B00001111},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00011100},
{B00000000, B00000000, B00000000, B00001110, B00010000, B00010000, B00010000, B00001110},
{B00000000, B00000001, B00000001, B00000001, B00000111, B00001001, B00001001, B00000111},
{B00000000, B00000000, B00000000, B00011100, B00010010, B00011110, B00010000, B00001110},
{B00000000, B00000011, B00000100, B00000100, B00000110, B00000100, B00000100, B00000100},
{B00000000, B00001110, B00001010, B00001010, B00001110, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000100, B00000000, B00000100, B00000100, B00000100, B00000100},
{B00000000, B00000010, B00000000, B00000010, B00000010, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010100, B00011000, B00011000, B00010100, B00010000},
{B00000000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00001010, B00010101, B00010001, B00010001, B00010001},
{B00000000, B00000000, B00000000, B00010100, B00011010, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000000, B00001100, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00011100, B00010010, B00010010, B00011100, B00010000, B00010000, B00010000},
{B00000000, B00001110, B00010010, B00010010, B00001110, B00000010, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00001010, B00001100, B00001000, B00001000, B00001000},
{B00000000, B00000000, B00001110, B00010000, B00001000, B00000100, B00000010, B00011110},
{B00000000, B00010000, B00010000, B00011100, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00010010, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00010101, B00001010},
{B00000000, B00000000, B00000000, B00010001, B00001010, B00000100, B00001010, B00010001},
{B00000000, B00000000, B00010001, B00001010, B00000100, B00001000, B00001000, B00010000},
{B00000000, B00000000, B00000000, B00011111, B00000010, B00000100, B00001000, B00011111},
{B00000010, B00000100, B00000100, B00000100, B00001000, B00000100, B00000100, B00000010},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00001000, B00000100, B00000100, B00000100, B00000010, B00000100, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00001010, B00011110, B00010100, B00000000, B00000000}
};
```

```
void clearDisplay() {
  for (byte x=0; x<8; x++) {
  buffer[x] = B00000000;
  }
  screenUpdate();
}
void initMAX7219() {
      pinMode(DataPin, OUTPUT);
      pinMode(LoadPin, OUTPUT);
      pinMode(ClockPin, OUTPUT);
      clearDisplay();
      writeData(SCAN_LIMIT_REG, B00000111);  //предел сканирования установлен на 0:7
      writeData(DECODE_MODE_REG, B00000000); // режим декодирования выключен
      writeData(SHUTDOWN_REG, B00000001); // Устанавливаем регистр выключения в нормальный режим работы

      intensity(15);                         // Только значения от 0 до 15 (4 бита)
}
void intensity(int intensity) {
         writeData(INTENSITY_REG, intensity); //B0001010 - регистр интенсивности
}

void writeData(byte msb, byte lsb) {
   digitalWrite(LoadPin, LOW);     // устанавливаем loadpin на готовность к приему данных
   shiftOut(DataPin, ClockPin, MSBFIRST, (msb));
   shiftOut(DataPin, ClockPin, MSBFIRST, (lsb));
   digitalWrite(LoadPin, HIGH);              // фиксируем данные
}

void scroll(char myString[], int rate) {

byte firstChrRow, secondChrRow;
byte ledOutput;
byte chrIndex = 0;                // Инициализируем индекс позиции строки
byte Char1, Char2;
byte scrollBit = 0;
byte strLength = 0;

unsigned long time;
unsigned long counter;

    while (myString[strLength]) { // Увеличиваем счетчик, пока не дойдем до конца строки
    strLength++;}
     counter = millis();
  while (chrIndex < (strLength)) {
   time = millis();
  if (time > (counter + rate)) {
     Char1 = constrain(myString[chrIndex],32,126);
     Char2 = constrain(myString[chrIndex+1],32,126);
     for (byte y= 0; y<8; y++) {
         firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
        secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
```

```
      ledOutput = (firstChrRow << scrollBit)
                | (secondChrRow >> (8 - scrollBit) );
        buffer[y] = ledOutput;
      }
    scrollBit++;
    if (scrollBit > 6) {
    scrollBit = 0;
    chrIndex++;
    }
          counter = millis();
   }
  }
}

void screenUpdate() {
 for (byte row = 0; row < 8; row++) {
  writeData(row+1, buffer[row]);
 }
}

void setup() {
    initMAX7219();
    Timer1.initialize(10000); // инициализируем timer1 и устанавливаем период прерывания
    Timer1.attachInterrupt(screenUpdate);
    Serial.begin(9600);
}

void loop() {
  clearDisplay();
  scroll("   BEGINNING ARDUINO   ", 45);
  scroll("   ГЛАВА 7 - LСВЕТОДИОДНЫЕ
  ДИСПЛЕИ   ", 45);scroll("   HELLO
} WORLD!!!  :)   ", 45);
```

Когда вы загрузите код, вы увидите сообщение, прокручиваемое по дисплею. Конечно, вы можете изменить текст в функции цикла на свой собственный.

Проект 21 - Светодиодная точечная матрица - Прокручиваемое сообщение - Обзор устройства

Опять же, чтобы упростить понимание кода, вам нужно сначала узнать, как работает микросхема MAX7219.

MAX7219 работает очень похоже на регистры сдвига в том, что вы должны вводить данные последовательно, бит за битом. Одновременно в устройство должно быть загружено всего 16 бит. Чип прост в использовании и использует всего три вывода от Arduino. Цифровой вывод 2 Arduino подключается к выводу 1 на MAX, который является входом данных. Цифровой вывод 3 подключен к выводу 12 на MAX, что является НАГРУЗКОЙ, и, наконец, цифровой вывод 4 подключен к выводу 13 на MAX, который является тактовым. См. Рис. 7-5, на котором показана распиновка MAX7219.

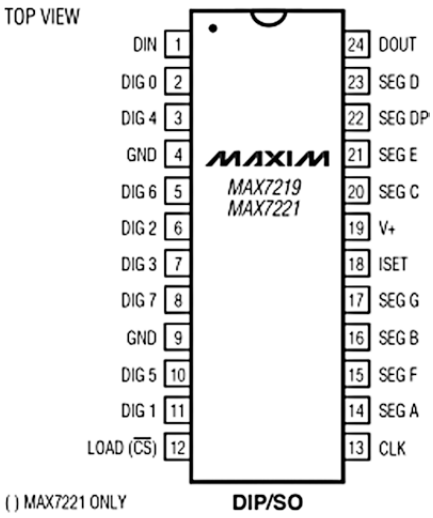


Рис. 7-5. *Распиновка MAX7219*

На выводе LOAD устанавливается низкий уровень, и первый бит данных устанавливается как HIGH или LOW на выводе DIN. Вывод CLK настроен на частоту между LOW и HIGH с помощью Arduino. По нарастающему фронту тактового импульса бит на выводе DIN сдвигается во внутренний регистр. Затем тактовый импульс падает до LOW, и следующий бит данных устанавливается на выводе DIN до того, как процесс повторится. После того, как все 16 бит данных были помещены в регистр по мере того, как тактовая частота изменяется 16 раз, вывод LOAD, наконец, устанавливается на HIGH, и это фиксирует данные в регистре. На рис. 7-6 представлена временная диаграмма из таблицы данных MAX7219, на которой показано, как нужно манипулировать тремя выводми, чтобы отправлять биты данных с D0 на D15 в устройство.

Вывод DOUT, который является выводом 24, в этом проекте не используется. Но если у вас было несколько последовательно соединенных между собой микросхем MAX7219, DOUT первой микросхемы подключается к DIN второй и так далее. Данные синхронизируются с вывода DOUT на заднем фронте тактового цикла.

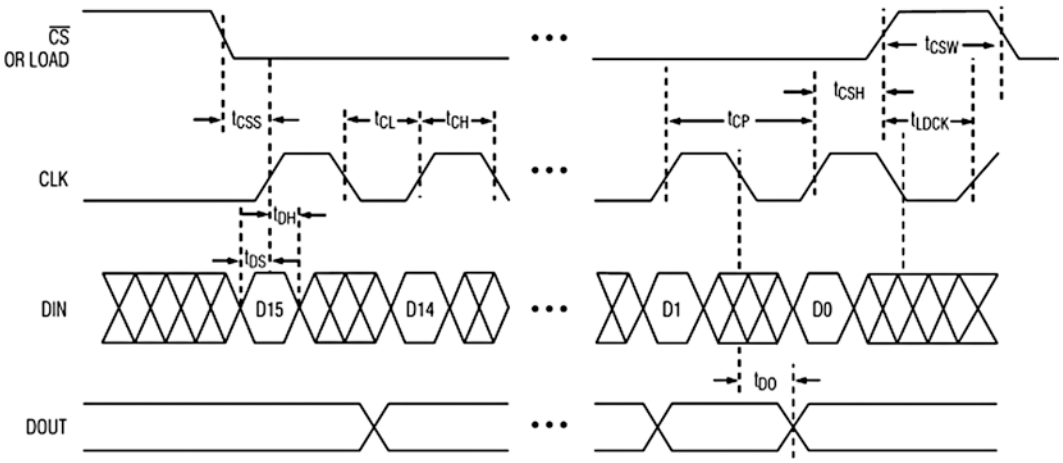


Рис. 7-6. *Временная диаграмма MAX7219*

Вам необходимо воссоздать эту временную последовательность в своем коде, чтобы иметь возможность отправлять соответствующие коды на чип.

Чип может обеспечивать ток до 100 мА, что более чем достаточно для большинства матричных дисплеев. Устройство принимает данные в виде 16-битных пакетов (или команд). D15 или старший бит (старший бит) отправляется первым, поэтому порядок идет от D15 до D0. Первые 4 бита - это биты «безразлично», то есть они не используются ИС, поэтому могут быть любыми. Следующие 4 бита составляют адрес регистра, а последние 8 бит составляют данные. В таблице 7-6 показан формат последовательных данных, а в таблице 7-7 показанакарта адресов регистров.

Таб. 7-6. *Формат последовательных данных (16 бит) MAX7219*

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB			DATA			LSB	
											X				
											X				
											X				
											X				
											X				
											X				
											X				

Таб. 7-7. *Карта адресов регистров MAX7219*

REGISTER	ADDRESS					HEX CODE
	D15-D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

Например, как вы можете видеть из карты адресов регистров в Таблице 7-7, адрес регистра интенсивности - 1010 двоичный. Регистр интенсивности устанавливает яркость дисплея со значениями от самого тусклого (0) до самого яркого (15) (от B000 до B1111). Чтобы установить интенсивность на 15 (максимум), вы должны отправить следующие 16 бит, причем старший бит (крайний левый) будет отправлен первым, а младший значащий бит (бит в крайнем правом углу) будет отправлен последним:

0000101000001111

Вторые 8 бит - это данные, отправляемые в регистр интенсивности. Первые 4 бита регистра интенсивности снова равны «безразлично», поэтому вы отправляете B0000. Следующие 4 бита определяют интенсивность. В этом случае вам нужна максимальная интенсивность, равная B1111. Отправляя эти 16 бит на устройство, вы устанавливаете максимальную яркость дисплея.

Полное 16-битное значение, которое вы хотите отправить, - это B0000101000001111, так как оно отправляется первым (старший значащий бит), а последним - LSB (младший бит).

Другой адрес, который вы будете использовать, - это ограничение на сканирование. Помните, что MAX7219 разработан для работы с 7-сегментными светодиодными дисплеями (см. Рисунок 7-7).

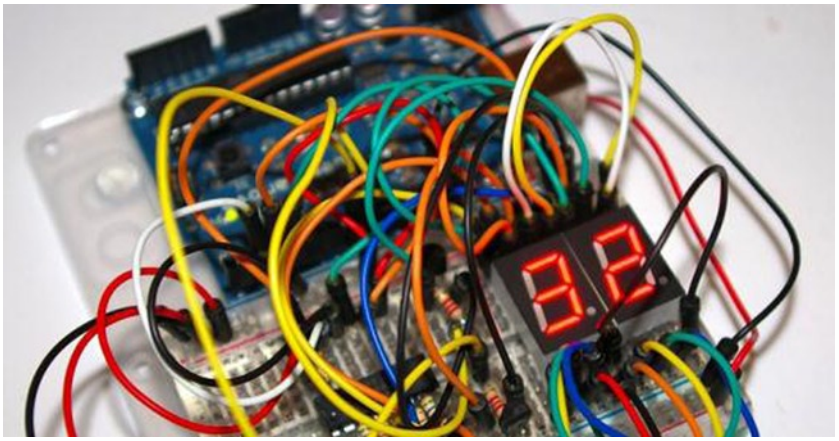


Рис. 7-7. *7-сегментный светодиодный дисплей*

Предел сканирования определяет, сколько из 8 цифр должно высветиться. В вашем случае мы используем не 7-сегментные дисплеи, а точечно-матричные дисплеи 8×8. Мы используем все 8-значные строки (0..7), управляющие нашим дисплеем, поэтому мы устанавливаем регистр ограничения сканирования на B00000111 (управляющие цифры 0..7).

Регистр режима декодирования имеет значение только при использовании 7-сегментных дисплеев, поэтому для отключения декодирования он будет установлен на B00000000.

Наконец, мы установим регистр выключения на B00000001, чтобы убедиться, что он находится в нормальном режиме работы, а не в режиме выключения. Если вы установите регистр выключения на B00000000, тогда все источники тока будут заземлены, что затем погасит дисплей.

Для получения дополнительной информации об микросхеме MAX7219 прочтите техническое описание. Просто прочтите те части таблицы, которые имеют отношение к вашему проекту, и вы увидите, что это намного проще для понимания, чем кажется на первый взгляд.

Теперь, когда вы (надеюсь) понимаете, как работает MAX7219, давайте взглянем на код и посмотрим, как заставить его отображать прокручиваемый текст.

Проект 21 - LED точечная матрица - Прокручиваемое сообщение - Обзор кода

Первое, что мы делаем в начале скетча, это загружаем две библиотеки, которые вы будете использовать в коде:

```
#include <avr/pgmspace.h>
#include <TimerOne.h>
```

Первая библиотека - это утилиты pgmspace или программное пространство. Функции этой библиотеки позволяют вашей программе получать доступ к данным, хранящимся в программном пространстве или флэш-памяти. Arduino с чипом ATmega328 имеет 32 КБ флеш-памяти (2 КБ из них используются загрузчиком, поэтому доступно 30 КБ). Arduino Mega имеет 128 КБ флеш-памяти,4 КБ из которых используется загрузчиком. Пространство программы - это именно то пространство, в котором будет храниться ваша программа. Вы можете использовать свободное неиспользуемое пространство во флэш-памяти с помощью утилит Program Space (программное пространство). Здесь вы будете хранить чрезвычайно большой 2D-массив, который будет содержать шрифт для ваших символов.

Вторая библиотека - это библиотека TimerOne, которая впервые была использована в Project 19. Затем объявляются три цифровых вывода, которые будут взаимодействовать с MAX7219:

```
int DataPin = 2;  // Вывод 1 на MAX
int LoadPin = 3;  // Вывод 12 на MAX
int ClockPin = 4; // Вывод 13 на MAX
```

Затем вы создаете массив типа buffer с 8 элементами:

```
byte buffer[8];
```

В этом массиве будет храниться набор битов, который будет определять, какие светодиоды включены или выключены, когда дисплей активен.Затем мы определяем 4 адреса в микросхеме MAX7219, которые мы будем использовать позже в коде.

```
#define SCAN_LIMIT_REG 0x0B
#define DECODE_MODE_REG 0x09
#define SHUTDOWN_REG 0x0C
#define INTENSITY_REG 0x0A
```

Далее идет большой 2D-массив типа byte:

```
static byte font[][8] PROGMEM = {
//Только печатаемые символы ASCII (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
..etc
```

В этом массиве хранится набор битов, составляющих шрифт, который вы будете использовать для отображения текста на дисплее. Это двумерный массив типа static byte. Мы также добавили после объявления массива команду PROGMEM.

Это атрибут из утилит Program Space, который сообщает компилятору хранить этот массив во флэш-памяти вместо SRAM (статическая память с произвольным доступом).

SRAM - это пространство памяти на микросхеме ATmega, которое обычно используется для хранения переменных и символьных строк, используемых в вашем скетче. При использовании они копируются из программного пространства и в SRAM. Однако массив, используемый для хранения шрифта, состоит из 96 символов, состоящих из восьми байтовкаждый. Массив состоит из 95 x 8 элементов, что составляет всего 760 элементов, и каждый элемент представляет собой байт (8 бит). Таким образом, шрифт занимает всего 760 байт. ATmega328 имеет всего 2 КБ,или примерно 2000 байт памяти для переменных. Как только вы добавите к этому другие переменные и текстовые строки, используемыев программе, вы рискуете быстро исчерпать память.

Arduino не может предупредить вас о нехватке памяти. Вместо этого он просто вылетает. Чтобы этого не произошло, мы сохраняем этот массив во флеш-памяти вместо SRAM, так как в нем гораздо больше места, с которым мы можете работать. Размер скетча составляет около 2800 байт, а размер массива чуть меньше 800 байт, поэтому мы использовали около 3,6 КБ из 30 КБ флеш-памяти, доступной вам.

Затем мы начинаем создавать различные функции, которые потребуются для программы. Первая просто очистит дисплей. Какие бы биты ни были сохранены в массиве буферов, они будут отображаться в матрице. Функция clearDisplay () просто циклически перебирает все восемь элементов массива и устанавливает их значения на ноль, чтобы светодиоды не горели, а дисплей оставался пустым. Затем она вызывает функцию screenUpdate (), которая отображает шаблон, хранящийся в массиве buffer [] на матрице. В этом случае, поскольку буфер не содержит ничего, кроме нулей, ничего не будет отображаться.

```
void clearDisplay() {
        for (byte x=0; x<8; x++) {
                buffer[x] = B00000000;
        }
        screenUpdate();
}
```

Следующая функция, initMAX7219 (), настраивает микросхему MAX7219, готовую к использованию. Сначала три вывода устанавливаются на ВЫХОД:

```
void initMAX7219() {
        pinMode(DataPin, OUTPUT);
        pinMode(LoadPin, OUTPUT);
        pinMode(ClockPin, OUTPUT);
```

Затем дисплей очищается:

```
clearDisplay();
```

Предел сканирования установлен на 7, режим декодирования выключен, а регистр выключения установлен на нормальную работу:

```
writeData(SCAN_LIMIT_REG, B00000111);        // предел сканирования установлен на 0:7
writeData(DECODE_MODE_REG, B00000000);       // режим декодирования выключен
writeData(SHUTDOWN_REG, B00000001); // Устанавливаем регистр выключения в нормальный режим работы
```

Затем интенсивность устанавливается на максимум, вызывая функцию intensity ():

```
intensity(15); // Только значения от 0 до 15 (4 бита)
```

Затем идет сама функция intensity (), которая просто берет переданное ей значение и записывает его в регистр интенсивности, вызывая функцию writeData:

```
void intensity(int intensity) {
         writeData(INTENSITY_REG, intensity); //B0001010 - регистр интенсивности
}
```

Следующая функция выполняет большую часть тяжелой работы. Его задача - записывать данные в MAX7219 по одному бит за раз, используя функцию shiftOut Arduino. Для функции требуются два параметра, оба байта, и они составляют старший байт (не бит) и младший байт 16-битного числа:

```
void writeData(byte MSB, byte LSB) {
```

Затем для loadPin устанавливается значение LOW. Это разблокирует данные в регистре ИС, готовые к приему новых данных:

```
digitalWrite(LoadPin, LOW);       // устанавливаем loadpin на готовность к приему данных
```

Затем мы используем функцию shiftOut () для сдвига двух байтов в MAX7219. Два байта - это старший байт msb(самый значительный байт) и lsb (наименьший значительный байт). MSB - это строка матрицы, а LSB - это точечный рисунок для соответствующего символа.

```
shiftOut(DataPin, ClockPin, MSBFIRST, (msb));
shiftOut(DataPin, ClockPin, MSBFIRST, (lsb));
```

Затем для loadPin устанавливается низкий уровень, чтобы скопировать содержимое регистров сдвига на выходные пины.

```
digitalWrite(LoadPin, HIGH);                    // зафиксировать данные
```

Далее идет функция scroll (), которая отображает на дисплее соответствующие символы текстовой строки. Функция принимает два параметра, первый - это текстовая строка, которую вы хотите отобразить, а второй - это скорость, с которой вы хотите, чтобы прокрутка выполнялась в миллисекундах между обновлениями:

```
void scroll(char myString[], int rate) {
```

Затем устанавливаются две переменные типа byte. В них будет храниться одна из восьми строк битовых шаблонов, составляющих конкретный отображаемый символ:

```
byte firstChrRow, secondChrRow;
```

Объявляется еще один байт, который называется ledOutput. Это сохранит результат вычисления первого битового шаблона и второго битового шаблона символов, и он определит, какие светодиоды включены или выключены (объяснение будет потом):

```
byte ledOutput;
```

Объявляется другая переменная типа byte, которая называется chrIndex и инициализируется нулем. chrIndex сохранит текущую позицию в отображаемой текстовой строке, начиная с нуля и увеличиваясь до длины строки:

```
byte chrIndex = 0;                              // Инициализируем указатель позиции строки
```

Объявлены еще два байта. Они будут содержать текущий и следующий символы в строке:

```
byte Char1, Char2;                    // два символа, которые будут отображаться
```

Они отличаются от firstChrRow и secondChrRow тем, что в них хранится значение ASCII (американский стандартный код для обмена информацией) символа, который будет отображаться, и следующего символа в строке. firstChrRow и secondChrRow хранят набор битов, из которых состоят отображаемые буквы.Все буквы, цифры, символы и т. д., которые могут отображаться на экране компьютера или отправляться через последовательный порт, имеют код ASCII. Это просто порядковый номер, указывающий, какой символ находится в таблице ASCII. Символы от 0 до 31 являются управляющими кодами, и мы не будем их использовать, так как они не могут отображаться на матричном дисплее. Мы будем использовать символы ASCII с 32 по 196, которые являются 95 печатными символами. Они начинаются с номера 32, который представляет собой пробел, и доходят до номера 126, который является символом тильды (~). Печатные символы ASCII перечислены в Таблице 7-8.

Таб. 7-8. *Печатные символы ASCII*

```
 !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

Объявляется еще один байт, который инициализируется нулем. Он будет показывать, сколько бит необходимо сдвинуть символьному шаблону текущего набора букв, чтобы создать впечатление прокрутки справа налево:

```
byte scrollBit = 0;
```

Другой байт будет содержать длину строки символов. Он инициализируется нулем:

```
byte strLength = 0;
```

Затем объявляются две переменные типа unsigned long, которые будут хранить текущее время в миллисекундах с момента загрузки или сброса микросхемы Arduino, а другая - для сохранения того же значения, но на этот раз через некоторое время будет запущена процедура. Вместе они гарантируют, что биты будут сдвинуты только через определенное время в миллисекундах, поэтому он будет прокручиваться с читаемой скоростью:

```
unsigned long time;
unsigned long counter;
```

Теперь вам нужно узнать, сколько символов в строке. Есть несколько способов сделать это, но в вашем случае мы просто настраиваем цикл while, который проверяет, есть ли данные в текущем индексе массива, который равен strLength (инициализирован нулем), и если да, увеличивает переменную strLength на единицу . Затем цикл повторяется до тех пор, пока условие myString [strLength] не станет ложным, т.е. в строке больше не будет символов, а strLength, которая увеличивается на единицу на каждой итерации, теперь будет содержать длину строки:

```
while (myString[strLength]) {strLength++;}
```

Затем вы устанавливаете значение counter на значение millis (). Вы встречали millis () в Project 4. Он хранит значение в миллисекундах с момента включения или сброса Arduino:

```
counter = millis();
```

Цикл while теперь будет выполняться при условии, что текущая позиция символа меньше, чем длина строки минус один:

```
while (chrPointer < (strLength-1)) {
```

Для переменной time установлено текущее значение millis ():

```
time = millis();
```

Затем оператор if проверяет, превышает ли текущее время последнее сохраненное время плюс значение в motion_interval, то есть 45 миллисекунд, и, если да, запускает блок кода в нем:

```
if (time > (counter + rate)) {
```

Char1 загружается с символьным значением ASCII символа в chrPointer в массиве myString, а Char2 - со значениемпосле этого:

```
Char1 = constrain(myString[chrIndex],32,126);
Char2 = constrain(myString[chrIndex+1],32,126);
```

Здесь мы использовали функцию constrain () Arduino, чтобы гарантировать, что печатаются только видимые символы ASCII.
Любые непечатаемые символы, такие как нулевой терминал или перевод строки, находятся за пределами допустимого диапазона, и мы не хотим пытаться их печатать. Следовательно, функция ограничения принимает три параметра: первый - это число, которое мы хотим ограничить, следующий - нижний диапазон, а третий - верхний диапазон. Т.о. любое число за пределами этого диапазона будет ограничено,т.е. любой символ с кодом меньше 32 преобразуется в 32, что является пробелом, а любой символ с кодом выше 126 заменяется на 126, что является ~.

Цикл for теперь перебирает каждую из восьми строк:

```
for (byte y= 0; y<8; y++) {
```

Теперь вы читаете массив шрифтов и помещаете битовый шаблон в текущей строке из восьми в firstChrRow, а второй - в secondChrRow. Помните, что массив шрифтов хранит битовые шаблоны, составляющие символы в таблице ASCII, но только печатаемые от 32 до 126. Первый индекс массива - это индекс, составленный из кода ASCII символа -32. (поскольку мы не используем первые 32 символа в таблице ASCII), а второй элемент массива хранит восемь строк битовых шаблонов, составляющих этот символ. Например, буквы A и Z представляют собой символы ASCII 65 и 90 соответственно. Вы вычитаете 32 из этих чисел, чтобы получить индекс массива.
Таким образом, битовый шаблон для буквы A, который представляет собой код ASCII 65, хранится в элементе массива 33 (65-32), а второе измерение массива в этом индексе хранит восьмибитовые шаблоны, составляющие букву. Буква Z - это код ASCII 90, который является индексом 58 в массиве.
Данные шрифтом [33] [0... 7], для буквы A,

```
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
```

Если вы поместите эти данные друг на друга, чтобы вы могли видеть их яснее, у вас есть

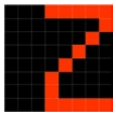
```
B00001110
B00010001
B00010001
B00010001
B00011111
B00010001
B00010001
B00010001
```

и если вы присмотритесь, вы увидите следующий узор, составляющий букву A:

Для буквы Z данные в массиве:

```
B00011111
B00000001
B00000010
B00000100
B00001000
B00010000
B00010000
B00011111
```

что соответствует битовому шаблону светодиода



Чтобы прочитать битовую комбинацию, вам нужно получить доступ к шрифту, который хранится в программном пространстве, а не в SRAM, как обычно.

Для этого вам нужно использовать одну из утилит из библиотеки pgmspace, pgm_read_byte:

```
firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
```

Когда вы обращаетесь к программному пространству, вы получаете данные, хранящиеся во флэш-памяти. . Для этого вам необходимо знать адрес в памяти, где хранятся данные. (Каждое место хранения в памяти имеет уникальный адресный номер.)

Для этого вы используете символ & перед переменной. Когда вы это сделаете, вы не читаете данные в этой переменной, вы вместо этого читаете адрес, по которому они хранятся. Команде pgm_read_byte необходимо знать адрес флэш-памяти данных, которые вы хотите получить, поэтому вы помещаете символ & перед шрифтом [Char1 - 32] [y], чтобы pgm_read_byte (& шрифт [Char1 - 32] [y]). Это просто означает, что вы читаете байт в программном пространстве, хранящийся по адресу font [Char1 - 32] [y].

Значение secondChrRow сдвигается влево на единицу просто для того, чтобы уменьшить промежуток между буквами, тем самым делая их более читаемыми на дисплее. Это связано с тем, что слева от всех символов для трех пробелов не используются биты.Вы можете сдвинуть его влево на два, чтобы приблизить их, но если вы это сделаете, его станет трудно читать.

Следующая строка загружает битовый шаблон для соответствующей строки в ledOutput:

```
ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit) );
```

Поскольку вы хотите, чтобы буквы прокручивались справа налево, вы сдвигаете первую букву влево на количество раз scrollBit, а вторую букву на 8 раз. Затем вы объединяете результаты логическим ИЛИ, чтобы объединить их в 8-битный шаблон, необходимый для отображения. Например, если отображаемые вами буквы были A и Z, то шаблоны для обоих были бы

```
B00001110 B00011111
B00010001 B00000001
B00010001 B00000010
B00010001 B00000100
B00011111 B00001000
B00010001 B00010000
B00010001 B00010000
B00010001 B00011111
```

Таким образом, вычисление выше в верхней строке, когда scrollBit установлено в 5, т.е. буквы прокручиваются на пять пикселей влево, будет

```
B110000000 B000000111
```

которая является верхней строкой A, сдвинутой влево 5 раз, и верхней строкой Z, сдвинутой вправо 3 раза (8-5). Вы можете видеть, что левый шаблон - это то, что вы получаете, когда букву A прокручиваете влево на пять пикселей, а правый шаблон - это то, что вы получаете, если буква Z была прокручена с правой стороны на 5 пикселей. Логическое ИЛИ, которое является | символ, имеет эффект слияния этих двух узоров вместе для создания

```
B11000111
```

это то, что вы получили бы, если бы буквы A и Z были рядом друг с другом и прокручивались влево на пять пикселей.Следующая строка загружает этот образец битов в соответствующую строку экранного буфера:

```
buffer[y] = ledOutput;
```

The scrollBit is increased by one:

```
scrollBit++;
```

Затем оператор if проверяет, достигло ли значение scrollBit 7. Если это так, он устанавливает его обратно в ноль и увеличивает chrPointer на единицу, чтобы при следующем вызове функции отображались следующие два набора символов:

```
if (scrollBit > 6) {
        scrollBit = 0;
        chrPointer++;
}
```

Наконец, значение counter обновляется до последнего значения millis ():

```
counter = millis();
```

Функция screenUpdate () просто берет восемь строк битовых шаблонов, которые вы загрузили в восьмиэлементный буферный массив, и записывает их в чип, который, в свою очередь, отображает их в матрице:

```
void screenUpdate() {
        for (byte row = 0; row < 8; row++) {
                writeData(row+1, buffer[row]);
        }
}
```

После настройки этих шести функций вы, наконец, дойдете до функций программы setup () и loop ().В setup () микросхема инициализируется вызовом initMax7219 (), создается таймер, устанавливается период обновления в 10000 микросекунд и присоединяется функция screenUpdate (). Как и прежде, это гарантирует, что функция screenUpdate () активируется каждые 10000 микросекунд, независимо от того, что еще происходит.

```
void setup() {
        initMAX7219();
        Timer1.initialize(10000); // инициализируем timer1 и устанавливаем период прерывания
        Timer1.attachInterrupt(screenUpdate);
}
```

Наконец, основной цикл программы состоит всего из четырех строк. Первый очищает дисплей, а затем следующие три вызывают процедуру прокрутки, чтобы отобразить три строки текста и настроить их прокрутку по дисплею.

```
void loop() {
        clearDisplay();
        scroll("   BEGINNING ARDUINO   ", 45);
        scroll("   ChapteГЛАВА - LСВЕТОДИОДНЫЕ
        ДИСПЛЕИ   ",
}       45);scroll("   HELLO WORLD!!!  :)   ", 45);
```

Вы, конечно, можете изменить текст в коде, чтобы на дисплее отображалось все, что угодно. Проект 21 был довольно сложным в плане кода. Как я сказал в начале, всей этой тяжелой работы можно было бы избежать, если бы вы просто использовали некоторые из уже существующих библиотек светодиодных

матриц, которые доступны в открытом доступе. Но при этом вы бы не узнали, как работает микросхема MAX7219. Идя по сложному пути, вы теперь должны хорошо разбираться в микросхеме MAX7219 и в том, как ею управлять. Эти навыки можно использовать для работы практически с любой другой внешней ИС, поскольку принципы в значительной степени те же.
В следующем проекте вы воспользуетесь этими библиотеками, чтобы увидеть, как они облегчат вам жизнь. Давайте позабавимся с помощью вашего дисплея .

Проект 22 - Светодиодный точечно-матричный дисплей - Игра в понг

Проект 21 был трудным и требовал большого внимания. Итак, для проекта 22 мы создадим простую игру с простым кодом, используя точечный матричный дисплей и потенциометр. На этот раз мы используем одну из многих доступных библиотек для управления матричными светодиодными дисплеями, чтобы увидеть, насколько проще это может облегчить вашу жизнь при кодировании.

Требуемые детали

Требуемые детали такие же, как и в Project 21, с добавлением линейного потенциометра 10 кОм (Таблица 7-8).

Таб. 7-8. *Дополнительные детали для проекта 22*

То же, что и Проект 21 плюс….
Линейный потенциометр 10 кОм 

Подключение

Оставьте схему такой же, как в Project 21, и добавьте линейный потенциометр.

Здесь используется линейный потенциометр, потому что нам нужна одинаковая пропорциональная величина движения на каждом конце хода потенциометра. Логарифмические потенциометры используются в аудиоприложениях, потому что некоторые настройки звука требуют применения нелинейного логарифмического управления. Затем элементу правления может быть помечен ступенями, имеющими одинаковый видимый размер. Если вы используете здесь неправильный тип, ваша ракетка-понг будет перемещаться по одной стороне экрана быстрее, чем по другой.

Левый и правый выводы идут на землю и + 5 В соответственно, а центральный вывод идет на аналоговый вывод 5.

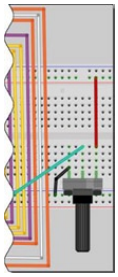


Рис. 7-8. *Добавьте потенциометр в схему Project 21. Средний вывод потенциометра идет на аналоговый вывод 5.*

Загрузите код

Перед загрузкой кода вам необходимо скачать, распаковать и установить библиотеку светодиодов. Перейдите к http://arduino.cc/playground/uploads/Main/LedControl.zip и загрузите файл, разархивируйте его, а затем поместите папку LedControl в папку ваших библиотек, как и раньше (см. http://playground.arduino.cc//Main / LedControl для получения дополнительной информации.). Затем загрузите код из Листинга 7-4. Когда программа запустится, мяч появляется из случайного места слева и направляется вправо. С помощью потенциометра управляйте ракеткой, чтобы мяч отскочил обратно к стене. Со временем скорость мяча будет увеличиваться все быстрее и быстрее, пока вы не сможете успевать за ним.

Листинг 7-4. Код для проекта 22

```
#include "LedControl.h"
LedControl myMatrix = LedControl(2, 4, 3, 1); // создать экземпляр матрицы

int column = 0, row = random(8);
int directionX = 1, directionY = 1;
int paddle1 = 5, paddle1Val;
int movement_interval = 300;
int counter = 0;

void setup()
{
  myMatrix.shutdown(0,false);
  /* Устанавливаем средние значения яркости */
  myMatrix.setIntensity(0,8);
  randomSeed(analogRead(0));
  oops();
}

void loop()
{
   paddle1Val = analogRead(paddle1);
   paddle1Val = map(paddle1Val, 0, 1024, 0,6);
   if (column == 6 && (paddle1Val == row ||
      paddle1Val+1 == row || paddle1Val+2 == row)) {directionX = -1;}
  if (column == 0) {directionX = 1;}
  if (row == 7) {directionY = -1;}
  if (row == 0) {directionY = 1;}
  if (column == 7) { oops();}
  column += directionX;
  row += directionY;
 displayDashAndDot();
 counter++;
}

void oops() {
  for (int x=0; x<3; x++) {
    myMatrix.clearDisplay(0);
    delay(250);
```

```
    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(150);
  }
  counter=0;
  movement_interval=300;
  column=0;
  row = random(8);
  displayDashAndDot();
}

void displayDashAndDot() {
  myMatrix.clearDisplay(0);
  myMatrix.setLed(0, column, row, HIGH);
  myMatrix.setLed(0, 7, paddle1Val, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
  if (!(counter % 10)) {movement_interval -= 5;}
  delay(movement_interval);
}
```

Когда мяч отскакивает от ракетки, экран мигает, и игра перезапускается. Посмотрите, сколько времени вы можете пройти, прежде чем игра перезагрузится.

■ **Примечание** помните, что проект не будет работать без загрузки библиотек, как описано выше.

Светодиодная матрица - Игра в пинг-понг. Обзор кода

Код для Project 22 действительно прост. В конце концов, вы отдыхаете от тяжелой работы, проделанной в Project 21.Сначала мы подключим внешнюю библиотеку. Это библиотека LedControl, которую вы, как и раньше, поместили в папку с библиотеками. Команда #include просто указывает среде IDE включить код из библиотеки LedControl в ваш код.

```
#include "LedControl.h"
```

Затем мы создаем экземпляр объекта LedControl следующим образом

```
LedControl myMatrix = LedControl(2, 4, 3, 1); // создаем экземпляр матрицы
```

Создаем объект LedControl с именем myMatrix. Для объекта LedControl требуется четыре параметра. Первые три - это номера пинов для MAX7219: Входящие данные, Такты и Нагрузка. Последнее число соответствует номеру микросхемы (в случае, если вы управляете более чем одним MAX7219 и дисплеем).

Затем мы решаем, в каком столбце и строке будет появляться мяч. Выбор строки определяется с использованием случайного числа.

```
int column = 0, row = random(8)+              // решаем, где появится мяч
```

Объявляем два целых числа, которые определяют направление, в котором будет двигаться мяч. Если число положительное, он будет двигаться слева направо и снизу вверх, соответственно, а если отрицательное, то наоборот.

```
int directionX = 1, directionY = 1; //  убедитесь, что он перемещается слева направо
```

Решаем, какой вывод потенциометра используется для управления ракеткой, и объявляем целое число для хранения значения, считанного с аналогового вывода:

```
int paddle1 = 5, paddle1Val;                    // Вывод потенциометра и значение
```

Интервал между движениями шара указывается в миллисекундах:

```
int movement_interval = 300;
```

Затем мы объявляем и инициализируем счетчик до нуля:

```
int counter = 0;
```

Функция setup () включает отображение, гарантируя, что для режима энергосбережения установлено значение false. Устанавливается средняя интенсивность, а затем игра инициализируется с помощью функции oops (), которую мы рассмотрим позже. Перед тем, как вы начнете, randomSeed устанавливается со случайным значением, считываемым с неиспользуемого аналогового вывода.

```
void setup()
{
        myMatrix.shutdown(0, false); // включить отображение
        myMatrix.setIntensity(0, 8); //  Устанавливаем среднюю яркость
        randomSeed(analogRead(0));
       oops();
}
```

В основном цикле мы начинаем с считывания аналогового значения с ракетки:

```
paddle1Val = analogRead(paddle1);
```

Затем эти значения отображаются в диапазоне от 1 до 6:

```
paddle1Val = map(paddle1Val, 0, 1024, 0, 6);
```

Для команды map требуется пять параметров. Первый - это номер, который нужно сопоставить. Далее следуют нижнее и верхнее значения числа, а также нижнее и высокое значения, с которыми вы хотите его сопоставить. В нашем случае мы берете значение в paddle1Val, которое представляет собой напряжение, считываемое с аналогового вывода 5. Это значение находится в диапазоне от 0 до 1023 (от 0 до 5 вольт) соответственно. Нам требуется, чтобы эти числа отображались только для чтения между 0 и 5, так как это строка, в которой будет отображаться ракетка при рисовании на дисплее.

Теперь нам нужно решить, ударился ли мяч о стену или ракетку, и, если да, отскочить назад (исключение составляет случай, когда он проходит мимо ракетки. Первый оператор if проверяет, попал ли мяч в ракетку. Он делает это путем решения если столбец, в котором находится мяч, находится в столбце 6 и попал ли он на ракетку:

```
if (column == 6 && (paddle1Val == row || paddle1Val+1 == row || paddle1Val+2 == row))
{directionX = -1;}
```

Для изменения направления мяча необходимо выполнить два условия. Во-первых, столбец равен 6, а во-вторых, мяч находится в том же ряду, что и любая из трех точек, составляющих ракетку. Это делается вложением в квадратные скобки набора команд или (логическое ИЛИ ||). Сначала проверяется результат этого вычисления, а затем результат добавляется к трем операторам && в первом наборе скобок.

Следующие три набора операторов if проверяют, попал ли мяч в верхнюю, нижнюю или левую боковые стенки, и если да, то меняем направление движения шара на противоположное:

```
if (column == 0) {directionX = 1;}
if (row == 7) {directionY = -1;}
if (row ==) {directionY = 1;}
```

Наконец, если мяч находится в столбце 7, очевидно, что он не попал в ракетку, а прошел мимо нее. В этом случае вызываем функцию oops (), чтобы замигать дисплеем и сбросить значения:

```
if (column == 7) { oops();}
```

Координаты столбца и строки увеличиваются на значения в directionX и directionY:

```
column += directionX;
row += directionY;
```

Затем вызывается функция, отображающая мяч и ракетку.

```
displayDashAndDot();
```

Давайте посмотрим на эту функцию. Мяч выбирается в столбце и строке. Это делается с помощью команды setLed библиотеки LedControl:

```
myMatrix.setLed(0, column, row, HIGH);
```

Для команды setLed требуется четыре параметра. Первое число - это адрес дисплея, затем координаты x и y (или столбца и строки) и, наконец, HIGH или LOW для включения и выключения. Они снова используются для рисования трех точек, составляющих ракетку, в столбце 7 и строке paddle1Val (два значения).

```
myMatrix.setLed(0, 7, paddle1Val, HIGH);
myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
```

Затем мы проверяем, является ли модуль счетчика% 10 НЕ (логическим НЕ!) истинным, и если да, то уменьшаем motion_interval на пять. По модулю - это остаток от деления одного целого числа на другое. В нашем случае мы делим счетчик на 10 и проверяем, является ли остаток целым числом или нет. Это в основном гарантирует, что motion_interval только увеличился через установленное время.

```
if (!(counter % 10)) {movement_interval -= 5;}
```

Задержка в миллисекундах значения movement_interval активируется, а затем значение счетчика увеличивается на единицу:

```
delay(movement_interval);
counter++;
}
```

Наконец, функция oops () заставляет цикл for внутри цикла for очищать дисплей, а затем повторно заполняет все строки с задержкой в 250 миллисекунд между ними. При этом все светодиоды загораются и выключаются, указывая на то, что мяч вышел из игры и игра скоро будет перезагружена. Затем все значения счетчика, движения_интервал и столбца устанавливаются в свои начальные позиции, а для строки выбирается новое случайное значение.

```
void oops() {
  for (int x=0; x<3; x++) {
  myMatrix.clearDisplay(0);
  delay(250);
                for (int y=0; y<8; y++) {
                        myMatrix.setRow(0, y, 255);
                }
                delay(250);
        }
        counter=0;       // сбрасываем все значения
        movement_interval=300;
        column=0;
  row = random(8) // выбираем новую начальную позицию
}
```

Команда setRow работает, передавая адрес дисплея, значение строки, а затем двоичный образец того, какие светодиоды включить или выключить. В этом случае мы хотим, чтобы все они были включены, это двоичный11111111 и десятичный 255.

Цель проекта 22 - показать, насколько проще управлять микросхемой драйвера светодиода, если использовать готовую библиотеку кода, разработанную для этой микросхемы. В Project 21 вы проделали сложный путь, кодируя все с нуля; в Проекте 22 вся тяжелая работа была сделана за вас за кулисами. Доступны и другие библиотеки Matrix, и раздел Playground на веб-сайте Arduino предоставит вам дополнительную информацию, или вы можете поискать информацию об управлении светодиодными матрицами на форуме Arduino, где вы найдете много полезной информации.

Вы можете использовать любую библиотеку, которая лучше всего соответствует вашим потребностям.

В следующей главе вы познакомитесь с другим типом точечно-матричного дисплея - ЖК-дисплеем.

упражнение

Возьмите концепции из проектов 21 и 22 и объедините их. Сделайте игру в понг, но пусть код будет вести счет в миллисекундах с момента запуска игры. Когда мяч выходит из игры, используйте функцию прокрутки текста.

Резюме

Глава 7 познакомила вас с некоторыми довольно сложными предметами, включая использование внешних микросхем. Вы еще даже не закончили работу над проектами и уже знаете, как управлять точечно-матричным дисплеем, используя как регистры сдвига, так и специальную микросхему драйвера светодиода. Кроме того, вы узнали, как кодировать вещи сложным способом, а затем как кодировать простым способом, включив готовую библиотеку, разработанную для вашей ИС драйвера светодиодов. Вы также узнали иногда сбивающую с толку концепцию мультиплексирования, навык, который очень пригодится для многих других вещей, а также для точечных матричных дисплеев.

Предметы и понятия, затронутые в главе 7:

- Как устроен точечный матричный дисплей
- Как установить внешнюю библиотеку
- Понятие мультиплексирования (или мультиплексирования)
- Как использовать мультиплексирование для индивидуального включения 64 светодиодов, используя всего 16 выходных контактов
- Основная концепция таймеро
- Как использовать библиотеку TimerOne для активации кода независимо от того, что еще происходит
- Как включить в код внешние библиотеки с помощью директивы #include
- Как использовать двоичные числа для хранения светодиодных изображений
- Как преобразовать двоичное число с помощью побитового НЕ ~
- Как воспользоваться постоянством зрения, чтобы обмануть глаз
- Как хранить кадры анимации в многомерных массивах
- Как объявить и инициализировать многомерный массив
- Доступ к данным в конкретном элементе двумерного массива
- Как выполнить побитовое вращение (также известное как круговой сдвиг)
- Как управлять светодиодными матричными дисплеями с помощью регистров сдвига
- Как управлять светодиодными матричными дисплеями с помощью микросхем MAX7219
- Как правильно синхронизировать импульсы для загрузки данных на внешние ИС и из них
- Как сохранить символьный шрифт в двумерном массиве данных
- Как читать временные диаграммы из таблиц данных
- Как использовать регистры в MAX7219
- Как обойти ограничения SRAM и сохранить данные в программном пространстве
- Как изменить порядок битов
- Как заставить текст и другие символы прокручиваться по матричному дисплею
- Понятие таблицы символов ASCII
- Выбор символа из его кода ASCII
- Как узнать длину текстовой строки
- Как писать и читать из области программы
- Как получить адрес в памяти переменной с помощью символа &
- Как использовать библиотеку LedControl.h для управления отдельными светодиодами и рядами светодиодов
- Как использовать логические операторы
- Как упростить жизнь и ускорить разработку кода с помощью библиотек кода

ГЛАВА 8

Жидкокристаллические дисплеи

Теперь мы собираемся перейти к другому популярному методу отображения текста и символов - ЖК-дисплею (жидкокристаллический дисплей). Это тип дисплеев, который обычно используется в калькуляторах или будильниках. Многие проекты Arduino используют их, поэтому важно, чтобы вы тоже знали, как их использовать. Для управления ЖК-дисплеями требуются микросхемы драйверов, которые встроены в дисплей. Самый популярный тип микросхемы драйвера - Hitachi HD44780 (или совместимый); они используются для управления большинством обычных LCDS.

Создавать проекты на основе ЖК-дисплеев легко благодаря целому набору доступных библиотек кодов ЖК-дисплеев. IDE Arduino поставляется с прекрасной библиотекой LiquidCrystal.h, которая имеет отличный список функций.

Поэтому мы будем использовать его в наших проектах.

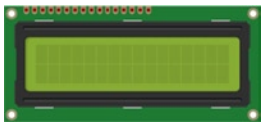
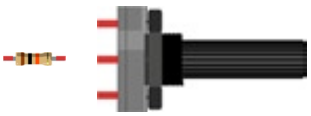
Проект 23 - Базовое управление ЖК-дисплеем

Для начала мы создадим демонстрационный проект, который продемонстрирует большинство функций, доступных в библиотеке LiquidCrystal.h. Мы будем использовать ЖК-дисплей 16x2 с подсветкой.

Требуемые детали

Вам потребуется ЖК-дисплей, использующий драйвер HD44780. Их много, и они бывают самых разных цветов. Как астроному-любителю, мне особенно нравятся дисплеи красного на черном, поскольку они сохраняют ваше ночное зрение. У них красный текст на черном фоне. Вы, конечно, можете выбрать любой доступный цвет текста и фона по вашему желанию. Ваш дисплей должен иметь подсветку и отображать шестнадцать столбцов и две строки символов. Их часто называют ЖК-дисплеями 16x2.

Таб. 8-1. *Требуемые детали для проекта 23*

ЖК-дисплей 16x2 с подсветкой	
Резистор (подсветка)	
Резистор настройки контрастности или потенциометр	

Подключение

Схема дляПроекта 23 довольно проста. Что нам понадобится, так это техническое описание используемого ЖК-дисплея. Следующие выводы (см. Таблицу 8-2) от Arduino, + 5V и Gnd, должны быть подключены к ЖК-дисплею.

Table 8-2. *распиновка LCD*

Arduino	Other	Matrix
Digital 11		Enable
Digital 12		RS (Register Select)
Digital 5		DB4 (Data Pin 4)
Digital 4		DB5 (Data Pin 5)
Digital 3		DB6 (Data Pin 6)
Digital 3		DB7 (Data Pin 7)
	Gnd	Vss (GND)
	Gnd	R/~WRead/Write)
	+5v	Vdd
	Gnd via resistor	Vo (Contrast)
	+5V via resistor	A/Vee (Power for LED)
	Gnd	Gnd for LED

Выводы данных с 0 по 3 не используются, так как мы будем использовать так называемый 4-битный режим. Для типичного ЖК-дисплея схема на Рис. 8-1 будет соответствующей.

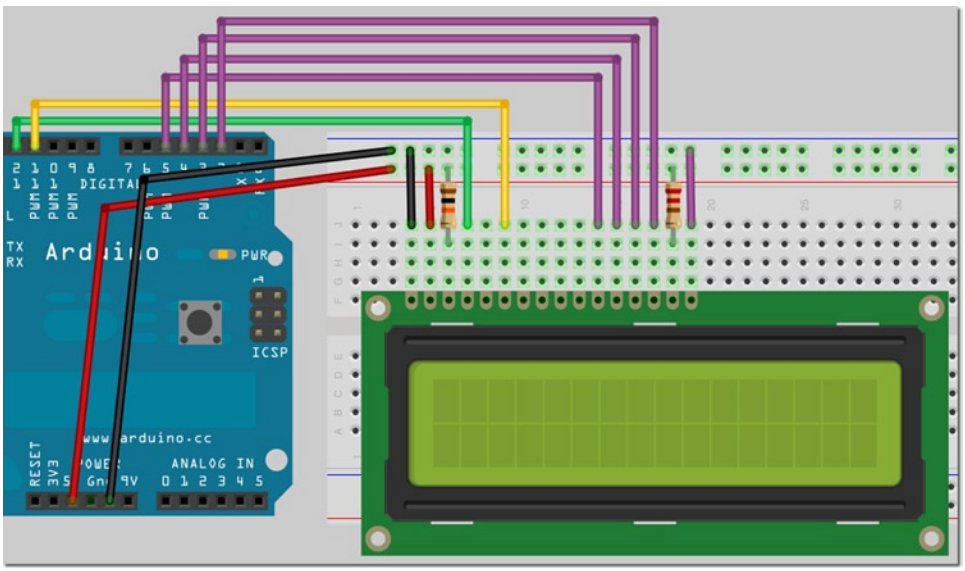


Рис. 8-1. *Схема для Проекта 23 - базовое управление ЖК-дисплеем*

Для регулировки контрастности на ЖК-дисплее подключите потенциометр со значением от 4 кОм до 10 кОм. Его крайние выводы с плюсом 5В и землей, а центральный вывод с контактом регулировки контрастности (контакт 3 на моем рисунке) ЖК-дисплея. Таким образом, вы можете осуществлять регулировку контрастности.
Для подсветки ЖК-дисплея, которую я использовал, требовалось 4,2 В, поэтому я добавил соответствующий токоограничивающий резистор между + 5 В и выводом источника питания светодиода (вывод 15 на моем ЖК-дисплее). Как только вы убедитесь, что ваша схема совпадает с моей с соответствующими контактами между Arduino, + 5V и землей (в соответствии с таблицей данных ЖК-дисплеев), вы можете ввести код.

Введите код

Проверьте свои соединения по рис. 8.1, затем загрузите код из листинга 8-1.

Listing 8-1. Код для проекта 23

```
#include <LiquidCrystal.h>

// инициализируем библиотеку номерами контактов интерфейсаЖК-дисплей LiquidCrystal (12, 11, 5, 4, 3, 2);
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // создаем ЖК-объект и назначаем контакты

void setup() {
       lcd.begin(16, 2);     // Устанавливаем отображение на 16 столбцов и 2 строки
}

void loop() {
       // запускаем 7 демонстрационных подпрограмм
       basicPrintDemo();
       displayOnOffDemo();
       setCursorDemo();
       scrollLeftDemo();
       scrollRightDemo();
       cursorDemo();
       createGlyphDemo();
}

void basicPrintDemo() {
       lcd.clear();                  // Очистить дисплей
       lcd.print("Basic Print");     // выводим текст
       delay(2000);
}

void displayOnOffDemo() {
       lcd.clear();                  // Очистить дисплей
       lcd.print("Display On/Off");  // выводим текст
       for(int x=0; x < 3; x++) {    // цикл 3 раза
              lcd.noDisplay();       // выключаем отображение
              delay(1000);
```

```
                lcd.display();              // снова включаем
                delay(1000);
        }
}

void setCursorDemo() {
        lcd.clear();                        // Очистить дисплей
        lcd.print("SetCursor Demo");        // выводим текст
        delay(1000);
        lcd.clear();                        // Очистить дисплей
        lcd.setCursor(5,0);                 // курсор в столбце 5 строка 0
        lcd.print("5,0");
        delay(2000);
        lcd.setCursor(10,1);                // курсор в столбце 10 строка 1
        lcd.print("10,1");
        delay(2000);
        lcd.setCursor(3,1);                 // курсор в столбце 3 строка 1
        lcd.print("3,1");
        delay(2000);
}

void scrollLeftDemo() {
        lcd.clear();                        // Очистить дисплей
        lcd.print("Scroll Left Demo");
        delay(1000);
        lcd.clear();                        // Очистить дисплей
        lcd.setCursor(7,0);
        lcd.print("Beginning");
        lcd.setCursor(9,1);
        lcd.print("Arduino");
        delay(1000);
        for(int x=0; x<16; x++) {
                lcd.scrollDisplayLeft();    // прокручиваем дисплей влево 16 раз
                delay(250);
        }
}

void scrollRightDemo() {
        lcd.clear();                        // Очистить дисплей
        lcd.print("Scroll Right");
        lcd.setCursor(0,1);
        lcd.print("Demo");
        delay(1000);
        lcd.clear();                        // Очистить дисплей
        lcd.print("Beginning");
        lcd.setCursor(0,1);
        lcd.print("Arduino");
        delay(1000);
        for(int x=0; x<16; x++) {
                lcd.scrollDisplayRight(); // прокручиваем дисплей вправо в 16 раз
                delay(250);
        }
}
```

```
void cursorDemo() {
        lcd.clear();                // Очистить дисплей
        lcd.cursor();               // Включение видимости курсора
        lcd.print("Cursor On");
        delay(3000);
        lcd.clear();                // Очистить дисплей
        lcd.noCursor();             // курсор невидим
        lcd.print("Cursor Off");
        delay(3000);
        lcd.clear();                // Очистить дисплей
        lcd.cursor();               // курсор виден
        lcd.blink();                // курсор мигает
        lcd.print("Cursor Blink On");
        delay(3000);
        lcd.noCursor();             // курсор невидим
        lcd.noBlink();              // моргаем
}

void createGlyphDemo() {
        lcd.clear();

        byte happy[8] = {           // создаем массив байтов со счастливым лицом
        B00000,
        B00000,
        B10001,
        B00000,
        B10001,
        B01110,
        B00000,
        B00000};

        byte sad[8] = {             // создаем массив байтов с грустным лицом
        B00000,
        B00000,
        B10001,
        B00000,
        B01110,
        B10001,
        B00000,
        B00000};

        lcd.createChar(0, happy);   // создать собственный символ 0
        lcd.createChar(1, sad);     // создать собственный персонаж 1

        for(int x=0; x<5; x++) {    // цикл анимации 5 раз
                lcd.setCursor(8,0);
                lcd.write((byte)0); // пишем произвольный символ 0
                delay(1000);
                lcd.setCursor(8,0);
                lcd.write(1);       // пишем произвольный символ 1
                delay(1000);
        }
}
```

Проект 23 - Базовое управление ЖК-дисплеем - Обзор кода

Сначала мы загружаем библиотеку, которую собираемся использовать для управления ЖК-дисплеем. IDE Arduino поставляется с библиотекой LiquidCrystal.h, которая проста в понимании и использовании. Есть много других библиотек и примеров кода, доступных для различных типов ЖК-дисплеев и вы можете найти их на площадке Arduino по адресу
http://www.arduino.cc/playground/Code/LCD

```
#include <LiquidCrystal.h>
```

Теперь нам нужно создать объект LiquidCrystal и установить соответствующие контакты.

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //создаем ЖК-объект и назначаем контакты
```

Итак, мы создали объект LiquidCrystal и назвали его lcd. Это действительно переменная, которая работает как дескриптор объекта, поэтому мы можем обращаться к нему по имени. Первые два параметра устанавливают контакты для RS (выбор регистра) и включения. Последние четыре параметра - это выводы данных с D4 по D7. Поскольку мы используем 4-битный режим, мы используем только четыре из восьми выводов данных на дисплее.
Разница между 4-битным и 8-битным режимами заключается в том, что в 8-битном режиме мы можем отправлять данные по одному байту за раз, тогда как в 4-битном режиме 8 бит должны быть разделены на 4-битные числа (известные как нибблы). Это делает код больше и сложнее. Однако, поскольку мы используем готовую библиотеку, мы не будем об этом беспокоиться. Однако, если вы пишете критичный по пространству или времени код, вы можете рассмотреть возможность записи непосредственно на ЖК-дисплей в 8-битном режиме. Преимущество использования 4-битного режима заключается в сохранении 4 контактов, что полезно, если мы хотим одновременно подключать другие устройства

В функции setup () мы инициализируем отображение до требуемого размера, в нашем случае 16 столбцов и 2 строки.

```
lcd.begin(16, 2); // Устанавливаем отображение на 16 столбцов и 2 строки
```

Основной цикл программы проходит через семь различных демонстрационных подпрограмм, одну за другой, перед перезапуском. Каждая демонстрационная подпрограмма показывает один набор связанных подпрограмм в библиотеке LiquidCrystal.h.

```
void loop() {
        // запускаем 7 демонстрационных подпрограмм
        basicPrintDemo();
        displayOnOffDemo();
        setCursorDemo();
        scrollLeftDemo();
        scrollRightDemo();
        cursorDemo();
        createGlyphDemo();
}
```

Первая демонстрация - это basicPrintDemo (). Она предназначена для демонстрации использования элемента print (). Эта демонстрация просто очищает дисплей с помощью lcd.clear (), а затем выводит на дисплей с помощью lcd.print (). Обратите внимание: если вы инициализировали свой объект LiquidCrystal и назвали его, например, LCD1602, то этими элементами будут соответственно LCD1602.clear () и LCD1602.print (). Другими словами, средство идет после имени объекта с точкой между ними.

Элемент print () распечатает все, что находится внутри скобок в текущем местоположении курсора. Местоположение начального курсора по умолчанию всегда - столбец 0 и строка 0, то есть верхний левый угол. Когда вы используете print (), введенный вами текст будет добавлен на дисплей, а курсор будет перемещен в конец нового текста. Любой дополнительный текст будет туда добавлен . После очистки дисплея курсор будет установлен в исходное положение или положение по умолчанию.

```
void basicPrintDemo() {
       lcd.clear();                 // Очистить дисплей
       lcd.print("Basic Print");    // выводим текст
       delay(2000);
}
```

Вторая демонстрация предназначена для демонстрации элементов display () и noDisplay (). Эти элементы просто включают или отключают отображение. Код выводит «Display On / Off», а затем трижды запускает цикл for (), чтобы выключить дисплей; подождать одну секунду, включить снова, подождать еще секунду, затем повторить. Всякий раз, когда вы выключаете дисплей, все, что было напечатано на экране до того, как оно исчезло, будет сохранено при повторном включении дисплея.

```
void displayOnOffDemo() {
       lcd.clear();                 // Очистить дисплей
       lcd.print("Display On/Off"); // выводим текст
       for(int x=0; x < 3; x++) {   // цикл 3 раза
              lcd.noDisplay();      // выключаем отображение
              delay(1000);
              lcd.display();        // снова включаем
              delay(1000);
       }
}
```

Следующая демонстрация показывает элемент setCursor (). Все, что делает этот метод, - это установка курсора в положение столбца и строки, указанное в скобках. Демонстрация устанавливает курсор в три места и печатает это место в тексте на дисплее. Элемент setCursor () полезен для управления макетом вашего текста и обеспечения того, чтобы ваш вывод попадал в соответствующую часть экрана дисплея.

```
void setCursorDemo() {
       lcd.clear();                 // Очистить дисплей
       lcd.print("SetCursor Demo"); // выводим текст
       delay(1000);
       lcd.clear();                 // Очистить дисплей
       lcd.setCursor(5,0);          // курсор в столбце 5 строка 0
       lcd.print("5,0");
       delay(2000);
       lcd.setCursor(10,1);         // курсор в столбце 10 строка 1
       lcd.print("10,1");
       delay(2000);
       lcd.setCursor(3,1);          // курсор в столбце 3 строка 1
       lcd.print("3,1");
       delay(2000);
}
```

В библиотеке есть два элемента прокрутки текста. Один - scrollDisplayLeft (), другой - scrollDisplayRight (). Это довольно очевидно из названий. Теперь работают две демонстрационные программы, демонстрирующие эти методы. Первый печатает «Beginning Arduino» на правой стороне дисплея и прокручивает его влево 16 раз ,пока он не выйдет за пределы экрана.

```
void scrollLeftDemo() {
       lcd.clear();                      // Очистить дисплей
       lcd.print("Scroll Left Demo");
       delay(1000);
       lcd.clear();                      // Очистить дисплей
       lcd.setCursor(7,0);
       lcd.print("Beginning");
       lcd.setCursor(9,1);
       lcd.print("Arduino");
       delay(1000);
       for(int x=0; x<16; x++) {
              lcd.scrollDisplayLeft();  // прокручиваем дисплей влево 16 раз
              delay(250);
       }
}
```

Следующая демонстрация делает то же самое, начиная с текста слева и прокручивая его вправо 16 раз, пока он не выйдет за пределы экрана.

```
void scrollRightDemo() {
       lcd.clear();                      // Очистить дисплей
       lcd.print("Scroll Right");
       lcd.setCursor(0,1);
       lcd.print("Demo");
       delay(1000);
       lcd.clear();                      // Очистить дисплей
       lcd.print("Beginning");
       lcd.setCursor(0,1);
       lcd.print("Arduino");
       delay(1000);
       for(int x=0; x<16; x++) {
              lcd.scrollDisplayRight(); // прокручиваем экран вправо 16 раз
              delay(250);
       }
}
```

Курсор пока не видно. Он всегда есть, но еще не замечен. Каждый раз, когда вы очищаете дисплей, курсор возвращается в верхний левый угол или в столбец 0 и строку 0. После печати некоторого текста курсор будет располагаться сразу после последнего напечатанного символа. Следующая демонстрация очищает дисплей, затем включает курсор с помощью курсора () и печатает текст.

Курсор будет виден сразу послеэтого текста в виде символа подчеркивания (_).

```
void cursorDemo() {
       lcd.clear();                      // Очистить дисплей
       lcd.cursor();                     // Включение видимости курсора
       lcd.print("Cursor On");
       delay(3000);
```

Дисплей снова очищается, и на этот раз курсор отключается, что является режимом по умолчанию, с использованием noCursor ().Теперь курсор снова скрыт.

```
lcd.clear();                    // Очистить дисплей
lcd.noCursor();                 // курсор невидим
lcd.print("Cursor Off");
delay(3000);
```

Затем активируется курсор и также включается режим мигания с помощью функции blink ().

```
lcd.clear();                    // Очистить дисплей
lcd.cursor();                   // курсор виден
lcd.blink();                    // курсор мигает
lcd.print("Cursor Blink On");
delay(3000);
```

На этот раз курсор не только будет виден, но и будет мигать. Этот режим полезен, если вы ждете ввода текста от пользователя. Мигающий курсор будет действовать как подсказка для ввода текста.

Наконец, курсор выключается и мигание также выключается, чтобы вернуть курсор в режим по умолчанию.

```
        lcd.noCursor();         // курсор невидим
        lcd.noBlink();          // моргаем
}
```

Последняя демонстрация под названием createGlyphDemo () создает настраиваемого персонажа. Большинство ЖК-дисплеев имеют возможность программировать собственные символы. Стандартный ЖК-дисплей 16x2 имеет место для хранения 8 пользовательских символов. Символы имеют ширину 5 пикселей и высоту 8 пикселей (пиксель - это элемент изображения, то есть отдельные точки, составляющие цифровой дисплей).

Дисплей очищается, а затем два массива байтов инициализируются двоичным шаблоном счастливого и грустного лица.

Двоичные шаблоны имеют ширину 5 бит.

```
void createGlyphDemo() {
        lcd.clear();
        byte happy[8] = {       // создаем массив байтов со счастливым лицом
        B00000,
        B00000,
        B10001,
        B00000,
        B10001,
        B01110,
        B00000,
        B00000};
        byte sad[8] = {         // создаем массив байтов с грустным лицом
        B00000,
        B00000,
        B10001,
        B00000,
        B01110,
        B10001,
        B00000,
        B00000};
```

Затем мы создаем два пользовательских символа с помощью элемента createChar (). Для этого требуются два параметра:

первый - это номер настраиваемого символа (от 0 до 7 в случае используемого мной ЖК-дисплея, который может хранить максимум 8 шаблонов), а второй - это имя массива, в котором хранится двоичный шаблон настраиваемого символа . Это создает и сохраняет этот шаблон в памяти ЖК-дисплея.

```
lcd.createChar(0, happy);          // создать собственный символ 0  happy -счастливчик
lcd.createChar(1, sad);            // создать собственный персонаж 1 sad-грустный
```

Цикл for теперь будет повторяться пять раз. На каждой итерации курсор устанавливается в столбец 8 и строку 0, и первый пользовательский символ записывается в это место с помощью элемента write (). Он записывает пользовательский символ в скобках (преобразованный как байт) в курсор location. Первый символ, являющийся счастливым лицом, записывается в то же место курсора, а после задержки в одну секунду второй символ, являющийся грустным лицом, записывается в то же место курсора. Это повторяется пять раз, чтобы получилась грубая анимация.

```
      for(int x=0; x<5; x++) {    // цикл анимации 5 раз
            lcd.setCursor(8,0);
            lcd.write((byte)0); // пишем произвольный символ 0
            delay(1000);
            lcd.setCursor(8,0);
            lcd.write(1);       // пишем произвольный символ 1
            delay(1000);
      }
}
```

Проект 23 охватывает большинство популярных методов в библиотеке LiquidCrystal.h. Есть и другие, которые стоит открыть, и вы можете прочитать о них в справочной библиотеке Arduino по адресу http://www.arduino.cc/en/Reference/LiquidCrystal.

Проект 23 - Базовое управление ЖК-дисплеем - Обзор схемы

Очевидно, что компонентом в этом проекте будет ЖК-дисплей. Типичный ЖК-дисплей 16 x 2 имеет 16-символьную сетку в два ряда. Как мы уже отмечали, каждая сетка символов имеет ширину 5 пикселей и высоту 8 пикселей. Если вы установите очень высокий контраст на вашем дисплее, станут видны 32 массива 5x8 пикселей.

Это действительно все, что вам нужно знать о структуре ЖК-дисплея.

Давайте теперь воспользуемся нашим ЖК-дисплеем и сделаем индикатор температуры.

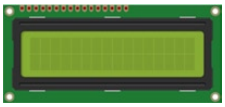
Проект 24 - ЖК-дисплей температуры

Этот проект будет простой демонстрацией использования ЖК-дисплея для представления полезной информации пользователю, в данном случае показания температуры с аналогового датчика температуры. Мы также добавим кнопку, позволяющую отображать температуру в градусах Цельсия или Фаренгейта, в зависимости от того, что вы предпочтете. Также во второй строке будут отображаться максимальная и минимальная температура.

Требуемые детали

Требуемые детали такие же, как и для Project 23, с добавлением кнопки и аналогового датчика температуры.
Убедитесь, что датчик температуры выдает только положительное напряжение.

Таб. 8-3. *Детали, необходимые для проекта 24*

ЖК-дисплей 16x2 с подсветкой	
Резистор подсветки	
Регулирующий контраст резистор	
Понижающий резистор	
Кнопка	
Аналоговый датчик температуры	

Подключение

Оставьте ту же схему, что и для Проекта 23. Затем добавьте кнопку и датчик температуры, как показано на рисунке 8-2.

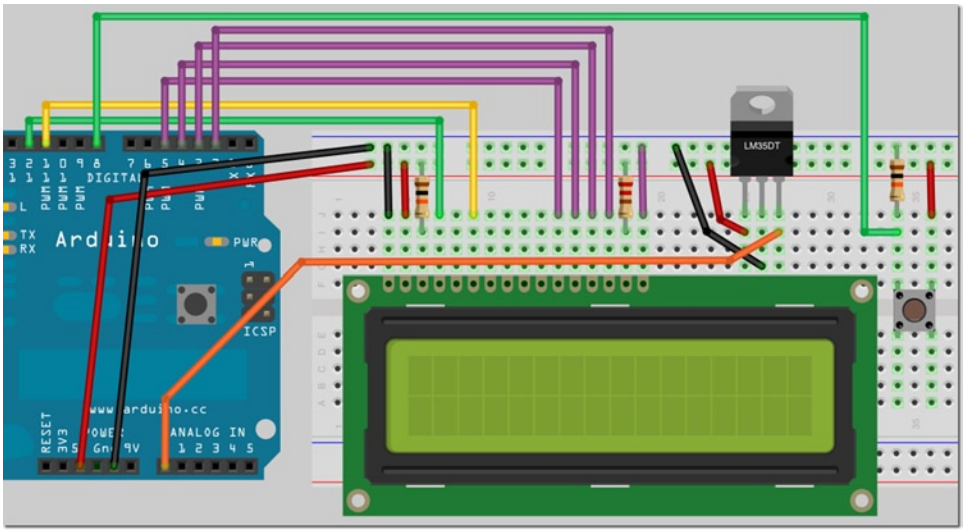


Рис. 8-2. *Схема для проекта 24 - ЖК-дисплей температуры*

Я использовал датчик температуры LM35DT, который имеет диапазон от 0°C до 100°C. Конечно, вы можете использовать любой аналоговый датчик температуры, какой захотите. LM35 имеет диапазон от -55°C до + 150°C. Вам нужно будет соответствующим образом скорректировать свой код (подробнее об этом позже).

Введите код

Проверьте соединения, затем загрузите код из листинга 8-2.

Листинг 8-2. Код для проекта24

```
#include <LiquidCrystal.h>

// инициализируем библиотеку номерами контактов интерфейса
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);          // создаем ЖК-объект и назначаем контакты
int maxC=0, minC=100, maxF=0, minF=212;
int scale = 1;
int buttonPin=8;

void setup() {
       lcd.begin(16, 2);              // Устанавливаем отображение на 16 столбцов и 2 строки
       analogReference(INTERNAL);
        analogReference(INTERNAL1V1); // Если у вас Arduino Mega
       pinMode(buttonPin, INPUT);
lcd.clear();
}

void loop() {
       lcd.setCursor(0,0);                      // установить курсор в исходную позицию
       int sensor = analogRead(0);              // считываем температуру с датчика
       int buttonState = digitalRead(buttonPin); // проверяем нажатие кнопки
       switch (buttonState) {                   // меняем состояние шкалы при нажатии
              case HIGH:
                     scale=-scale;              // инвертировать шкалу
                     lcd.clear();
       }

       switch (scale) {                         // решаем, C или F шкала
              case 1:
                     celsius(sensor);
                     break;
              case -1:
                     fahrenheit(sensor);
       }
       delay(250);
}

void celsius(int sensor) {
       lcd.setCursor(0,0);
       int temp = sensor * 0.1074188;           // конвертируем в C
       lcd.print(temp);
       lcd.write(B11011111);                    // символ градуса
```

```
        lcd.print("C ");
        if (temp>maxC) {maxC=temp;}
        if (temp<minC) {minC=temp;}
        lcd.setCursor(0,1);
        lcd.print("H=");
        lcd.print(maxC);
        lcd.write(B11011111);
        lcd.print("C L=");
        lcd.print(minC);
        lcd.write(B11011111);
        lcd.print("C ");
}

void fahrenheit(int sensor) {
        lcd.setCursor(0,0);
        float temp = ((sensor * 0.1074188) * 1.8)+32; // преобразовать в F
        lcd.print(int(temp));
        lcd.write(B11011111);                         // вывод символа градуса
        lcd.print("F ");
        if (temp>maxF) {maxF=temp;}
        if (temp<minF) {minF=temp;}
        lcd.setCursor(0,1);
        lcd.print("H=");
        lcd.print(maxF);
        lcd.write(B11011111);
        lcd.print("F L=");
        lcd.print(minF);
        lcd.write(B11011111);
        lcd.print("F ");
}
```

Когда вы запустите код, текущая температура будет отображаться на ЖК-дисплее в верхнем ряду. В нижнем ряду будут отображаться максимальные и минимальные температуры, зарегистрированные с момента включения Arduino или сброса программы. Нажав кнопку, вы можете изменить шкалу температуры между градусами Цельсия и Фаренгейта.

Проект 24 - ЖК-дисплей температуры

Как и раньше, в наш скетч загружена библиотека

```
LiquidCrystal.#include <LiquidCrystal.h>
```

Инициализируется объект `LiquidCrystal` с именем `lcd` и устанавливаются соответствующие контакты.

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // создаем ЖК-объект и назначаем контакты
```

Некоторые целочисленные переменные для хранения максимальной и минимальной температуры в градусах C и F объявляются и инициализируются невозможными максимальными и минимальными значениями. Они будут обновлены при первом запуске программы.

```
int maxC=0, minC=100, maxF=0, minF=212;
```

Переменная с именем scale типа int объявляется и инициализируется значением 1. Переменная scale решает, используем ли мы шкалу температуры по Цельсию или Фаренгейту. По умолчанию он установлен на 1, что соответствует Цельсию. Если хотите, можете изменить это значение на -1 для Фаренгейта.

```
int scale = 1;
```

Объявляется и инициализируется целое число для хранения пина, используемого для кнопки.

```
int buttonPin=8;
```

В функции setup () мы устанавливаем отображение на 16 столбцов и 2 строки.

```
lcd.begin(16, 2); // Устанавливаем отображение на 16 столбцов и 2 строки
```

Затем задание аналогового вывода устанавливается на INTERNAL - ВНУТРЕННИЙ.

```
analogReference(INTERNAL);
```

Если у вас Arduino Mega, это должно быть

```
analogReference(INTERNAL1V1);
```

Это дает нам лучший диапазон АЦП Arduino. Выходное напряжение LM35DT при 100 ° C составляет 1 В. Если бы мы использовали опорное напряжение по умолчанию 5 В, то при 50 ° C, что составляет половину диапазона датчика, показание АЦП было бы 0,5 В = (0,5 / 5) * 1023 = 102, что составляет всего около 10% от диапазона АЦП. При использовании внутреннего опорного напряжения 1,1 В значение на аналоговом выводе при 50 ° C теперь составляет 0,5 В = (0,5 / 1,1) * 1023 = 465.
Как видите, это почти половина всего диапазона значений, которые аналоговый вывод может считывать (от 0 до 1023).Таким образом, увеличилось разрешение и точность считывания, а также повысилась чувствительность схемы.
Однако точность зависит от точности внутреннего опорного напряжения посравнению с точностью напряжения питания 5 В.
Согласно паспорту Atmega328, внутреннее опорноенапряжение может составлять от 1,0 до 1,2 вольт, что составляет погрешность до 11%. Стабилизатор 5 В на Arduino Uno имеет допуск 1%.

Пин кнопки теперь установлен на вход, а ЖК-дисплей очищен.

```
pinMode(buttonPin, INPUT);
lcd.clear();
```

В основном цикле программа начинается с установки курсора в исходное положение.

```
lcd.setCursor(0,0);                    // установить курсор в исходную позицию
```

Затем мы считываем значение с датчика температуры на аналоговом выводе 0.

```
int sensor = analogRead(0);            // считываем температуру с датчика
```

Затем мы читаем состояние кнопки и сохраняем значение в buttonState.

```
int buttonState = digitalRead(buttonPin); // проверяем нажатие кнопки
```

Теперь нам нужно решить, была ли нажата кнопка или нет, и если да, то изменить шкалу с Цельсия на Фаренгейт или наоборот.

Это делается с помощью оператора switch / case.

```
switch (buttonState) {      // изменить состояние шкалы при нажатии
       case HIGH:
              scale=-scale; // инвертировать шкалу
              lcd.clear();
}
```

Это новое понятие, с которой мы раньше не сталкивались. Команда switch / case управляет потоком программы, указывая, какой код должен быть выполнен, если были выполнены разные условия. Оператор switch / case сравнивает значение переменной со значениями в операторах case и, если истинно, запускает код после этого оператора case.

Например, если у нас есть переменная с именем var, и мы хотим, чтобы что-то произошло, если ее значение было 1, 2 или 3, то мы могли бы решить, что делать для любого из этих значений, используя

```
switch (var) {
       case 1:
              // запускаем этот код здесь, если var равно 1
              break;
       case 2:
              // запускаем этот код здесь, если var равно 2
              break;
       case 3:
              // запускаем этот код здесь, если var равно 3
              break;
       default:
             // если ничего не подходит, запустите этот код здесь
       }
```

Итак, оператор switch / case проверит значение var. Если он равен 1, тогда он будет запускать код в блоке case 1 до команды break. Команда break используется для выхода из оператора switch / case. Без него код перешел бы к следующему блоку case и продолжал бы выполнение до тех пор, пока не будет достигнута команда break или не будет достигнут конец оператора switch / case. Если ни одно из проверенных значений не достигается, будет выполнен код в разделе по умолчанию. Обратите внимание, что раздел по умолчанию является необязательным и невостребованым.

В нашем случае мы проверяем только один случай, и это если buttonState имеет значение HIGH. Если это так, значение шкалы меняется (с C на F или наоборот), и дисплей очищается.

Далее небольшая задержка.

```
delay(250);
```

Затем еще один оператор switch / case, чтобы проверить, равно ли значение шкалы 1 для Цельсия или -1 для Фаренгейта, и если да, для запуска соответствующих функций.

```
       switch (scale) { // decide if C or F scale
              case 1:
                     celsius(sensor);
                     break;
              case -1:
                     fahrenheit(sensor);
       }
}
```

алее у нас есть две функции для отображения температуры на ЖК-дисплее. Один - если мы работаем в градусах Цельсия, а другой - в градусах Фаренгейта. Функции имеют один параметр. Мы передаем ему целое число, которое будет значением, считываемым с датчика температуры.

```
void celsius(int sensor) {
```

Курсор устанавливается в исходное положение.

```
lcd.setCursor(0,0);
```

Затем мы берем показание датчика и преобразуем его в градусы Цельсия, умножив на 0,1074188.

```
int temp = sensor * 0.1074188;                  // конвертируем в C
```

Этот коэффициент достигается путем деления 1,1, которое является опорным напряжением, на диапазон АЦП, равный 1024, а затем деление на чувствительность датчика, равную 0,01 В / ° C.

```
1.1/1024=0.0010742188
0.0010742188/0.01=0.10742188
```

Затем мы выводим это преобразованное значение на ЖК-дисплей вместе с символом с кодовым значением B11011111, который представляет собой символ градуса, за которым следует буква C, указывающая, что мы отображаем температуру в градусах Цельсия.

```
lcd.print(temp);
lcd.write(B11011111);                           // символ градуса
lcd.print("C  ");
```

Чтобы вести текущий счет максимальной и минимальной температуры, записанной с момента включения Arduino, мы используем функции Arduino max () и min (). Оба они принимают две переменные в качестве параметров и возвращают либо максимум из двух чисел, либо минимум. Мы сохраняем этот результат.

```
maxC = max(maxC,temp);
minC = min(minC, temp);
```

Затем во второй строке мы печатаем H (для HIGH) и значение max C, а затем L (для LOW), за которым следуют символ градуса и буква C.

```
lcd.setCursor(0,1);
lcd.print("H=");
lcd.print(maxC);
lcd.write(B11011111);
lcd.print("C L=");
lcd.print(minC);
lcd.write(B11011111);
lcd.print("C  ");
```

Функция по Фаренгейту делает то же самое, за исключением того, что преобразует температуру из Цельсия в Фаренгейт.умножив его на 1,8 и прибавив 32.

```
float temp = ((sensor * 0.10742188) * 1.8)+32; // конвертируем в F
```

Теперь, когда вы знаете, как использовать ЖК-дисплей для отображения полезной информации, вы можете создавать свои собственные проекты для отображения всего, что вам нравится на ЖК-дисплее, для отображения данных датчиков или для создания простого пользовательского интерфейса.

Резюме

В главе 8 вы узнали о наиболее часто используемых функциях библиотеки LiquidCrystal.h. Они включают в себя очистку дисплея, печать текста в определенных местах на экране, создание видимого или невидимого или мигающего курсора и даже то, как заставить текст прокручиваться влево или вправо. Проект 24 дал вам простое применение этих функций на реальном примере датчика температуры. Это дало вам представление о том, как ЖК-экран можно использовать в реальном проекте для отображения данных.

Предметы и понятия, затронутые в главе 8:

- Как загрузить библиотеку LiquidCrystal.h
- Как подключить ЖК-дисплей к Arduino
- Как отрегулировать яркость подсветки и контраст дисплея с помощью резисторов разного номинала
- Яркость подсветки можно регулировать с вывода ШИМ
- Как объявить и инициализировать объект LiquidCrystal
- Как установить правильное количество столбцов и строк на дисплее
- Как очистить ЖК-дисплей с помощью `clear ()`
- Как печатать в позицию курсора с помощью `print ()`
- Как включить и выключить дисплей с помощью `display ()` и `noDisplay ()`
- Как установить местоположение курсора с помощью `setCursor (x, y)`
- Как прокрутить дисплей влево с помощью `scrollDisplayLeft ()`
- Как прокрутить дисплей вправо с помощью `scrollDisplayRight ()`
- Как включить или отключить курсор с помощью `cursor ()` и `noCursor ()`
- Как заставить видимый курсор мигать с помощью `blink ()`
- Как создавать собственные символы с помощью createChar ()
- Как записать один символ в позицию курсора с помощью `write ()`
- Как считывать значения с аналогового датчика температуры
- Как увеличить разрешение АЦП с помощью внутреннего источника опорного напряжения
- Принятие решения с использованием оператора switch / case
- Как преобразовать значения АЦП в показания температуры в градусах Цельсия и Фаренгейта
- Как преобразовать код для чтения с разных датчиков температуры с разными диапазонами

ГЛАВА 9

Сервоприводы

В этой главе мы рассмотрим серводвигатели или сервомеханизмы. Сервопривод – это привод, вал которого может встать в заданное положение или поддерживать заданную скорость вращения. Другими словами, валом сервопривода можно управлять, например, задавая ему положение в градусах или определенную частоту вращения. Сервоприводы используются в самых разных областях, например, в робототехнике они помогают моделировать различные движения роботов.Точно так же они часто используются в качестве подвижных суставов в небольших манипуляторах роботов и для управления движением в аниматронике. Возможно, к концу этой главы вы будете вдохновлены поместить несколько сервоприводов внутрь плюшевого мишки или другой игрушки, чтобы заставить его двигаться. На рисунках 9-1 и 9-2 показаны другие способы использования сервоприводов.

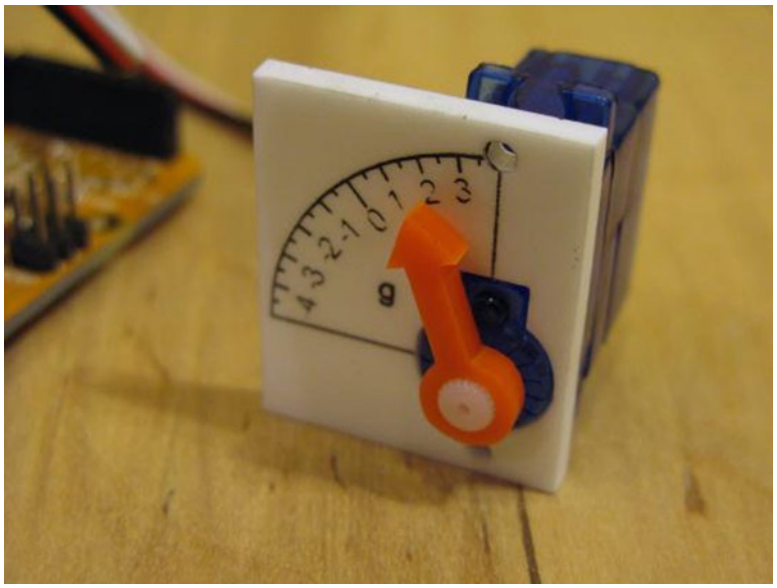


Рис. 9-1. *Сервопривод, используемый для управления измерителем*

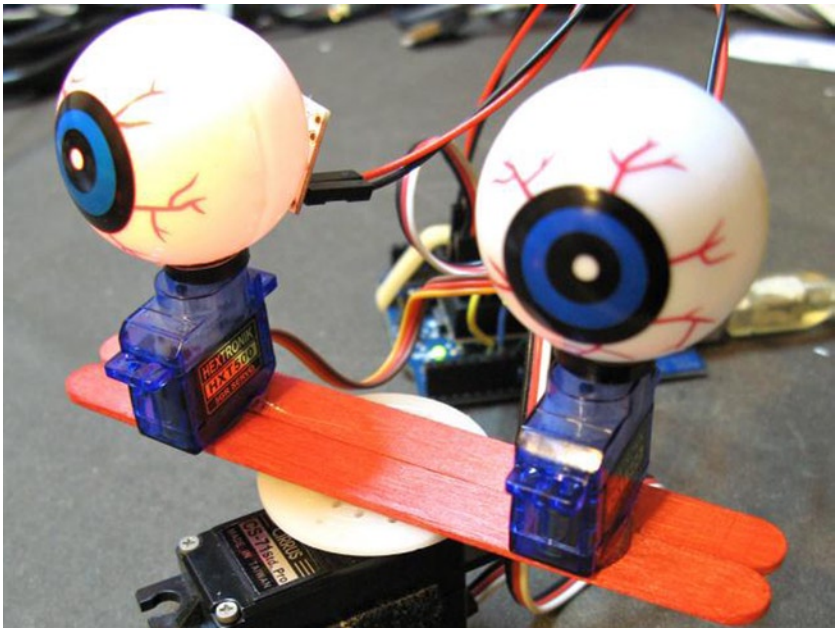


Рис. 9-2. *Три сервопривода, управляющие головой и глазными яблоками робота*

Сервоприводами действительно легко управлять благодаря серво-библиотеке, которая поставляется с Arduino IDE. Все три проекта в этой главе довольно просты и малы по сравнению с некоторыми другими проектами в книге, но при этом очень эффектны. Давайте начнем с очень простой программы для управления одним сервоприводом, затем перейдем к двум сервоприводам и закончим двумя сервоприводами, управляемыми джойстиком.

Проект 25 - Сервоуправление

В этом очень простом проекте вы будете управлять одним сервоприводом с помощью потенциометра.

Требуемые детали

Вам потребуется стандартный сервопривод RC; подойдет любой из малых или средних сервоприводов. Сервопривод должен питаться от собственного источника питания. Не подключайте его к источнику питания 5 В на Arduino, так как это вызовет чрезмерный нагрев, что может нарушить работу программы или, что еще хуже, повредить Arduino. Используйте внешний источник питания 5 В или Кроме того, добавьте резистор последовательно между выходом Arduino и управляющим входом сервопривода, чтобы ограничить ток, если Arduino включен, когда питание сервопривода отключено. Здесь подойдет резистор 220 Ом. Также вам понадобится потенциометр; Подойдет практически любой поворотный потенциометр. Для тестирования я использовал 4,7 кОм. Обратите внимание, что вы также можете подключить Arduino к внешнему источнику питания постоянного тока. См. Таблицу 9-1 для списка частей для этого проекта.

Таб. 9-1. *Детали, необходимые для проекта 25*

Стандартный RC сервопривод	
потенциометр	
Резистор 220 ом	

Подключение

Схема для Project 25 предельно проста. Произведите подключения , как показано на Рисунке 9-3.

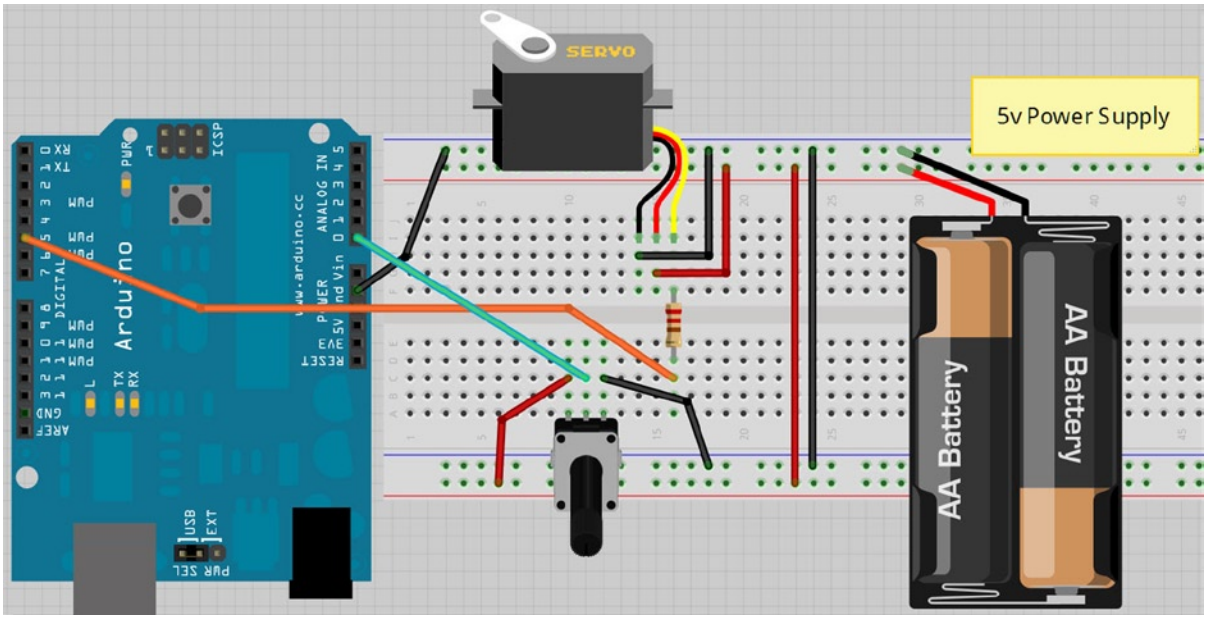


Рис. 9-3. *Схема для проекта 25 - Сервоуправление*

Не забудьте запитать сервопривод от внешнего источника 5 В, а не от Arduino. Сервопривод имеет три провода, идущих от него. Один будет красным и пойдет на + 5В. Один будет черным или коричневым и уйдет на землю. Третий будет белым, желтым или оранжевым и будет подключен к цифровому выводу 5 через резистор 220 Ом.

Поворотный потенциометр имеет крайние выводы, подключенные к + 5 В и земле, а средний вывод - к аналоговому контакту 0.

Как только все будет подключено должным образом, введите приведенный ниже код.

Введите код

А теперь об одной из самых коротких программ в книге см. Листинг 9-1!

Листинг 9-1. Код для проекта 25

```
// Project 25
#include <Servo.h>

Servo servo1;  // Создаем сервообъект

void setup()
{
servo1.attach(5);  // Присоединяет сервопривод к выводу  5  ШИМ


void loop()
{
        int angle = analogRead(0); // Считываем значение потенциометра
        angle=map(angle, 0, 1023, 0, 180); // Сопоставляем значения от 0 до 180 градусов
        servo1.write(angle); // Записываем угол сервопривода
delay(15); // Задержка 15 мс, чтобы сервопривод достиг положения
```

Проект 25 - Сервоуправление - Обзор кода

Во-первых, включена библиотека Servo.h:

```
#include <Servo.h>
```

Затем объявляется сервообъект с именем servo1:

```
Servo servo1;  // Создаем сервообъект
```

В цикле настройки вы присоединяете только что созданный сервопривод к выводу 5:

```
servo1.attach(5);  // Присоединяет сервопривод на выводе 5 к сервообъекту
```

Команда attach прикрепляет созданный сервообъект к назначенному выводу. Команда attach может принимать либо один параметр, как в нашем случае, либо три параметра. Если используются три параметра, первый параметр - это вывод, второй - минимальный (0 градусов) угол в ширине импульса в микросекундах (по умолчанию 544), а третий параметр - это максимальный угол (180 градусов) в ширине импульса в микросекундах (по умолчанию 2400). Это будет объяснено в обзоре устройства.
В большинстве случаев вы можете просто установить вывод и игнорировать необязательные второй и третий параметры.
Вы можете подключить до 12 сервоприводов к Arduino Duemilanove (или аналогу) и до 48 к Arduino Mega - идеально для приложений управления роботами!
Обратите внимание, что использование этой библиотеки отключает функцию analogWrite (PWM) на контактах 9 и 10. На Mega вы можете иметь до 12 двигателей, не мешая функциям PWM. Использование от 12 до 23 двигателей отключит функцию ШИМ на контактах 11 и 12.
В основном цикле считайте аналоговое значение с потенциометра, подключенного к аналоговому выводу 0:

```
int angle = analogRead(0); // Read the pot value
```

Затем это значение отображается так, что диапазон теперь составляет от 0 до 180, что будет соответствовать углу наклона сервопривода:

```
angle=map(angle, 0, 1023, 0, 180); // Сопоставляем значения от 0 до 180 градусов
```

Затем вы берете свой сервообъект и записываете под ним соответствующий угол в градусах (угол должен быть от 0 до 180 градусов):

```
servo1.write(angle); // Записываем угол сервопривода
```

Наконец, запрограммирована задержка 15 мс, чтобы дать сервоприводу время переместиться в нужное положение:

```
delay(15); // Delay of 15ms to allow servo to reach position
```

Вы также можете отсоединить сервопривод от 5 вывода командой detach(), что отключит его и позволит использовать вывод для чего-то еще.Кроме того, вы можете прочитать текущий угол с сервопривода командой write() (это последнее значение, переданное команде write ()).

Вы можете узнать больше о библиотеке сервоприводов на веб-сайте Arduino по адресу http://arduino.cc/en/Reference/Servo.

Проект 25 - Сервоуправление - Обзор схемы

Сервопривод - это небольшая коробка, которая содержит электродвигатель постоянного тока, набор шестерен между двигателем и выходным валом, механизм определения положения и схему управления. Механизм определения положения передает положение сервопривода обратно в схему управления, которая использует двигатель для регулировки рычага сервопривода в положение, в котором сервопривод должен находиться.

Сервоприводы бывают разных размеров, скоростей, сильных сторон и точности. Некоторые из них могут быть довольно дорогими. Чем мощнее или точнее сервопривод, тем выше цена.
Положение сервопривода управляется подачей набора импульсов. Это ШИМ, с которым вы сталкивались раньше.
Ширина импульсов измеряется в миллисекундах. Скорость отправки импульсов не имеет особого значения; для схемы управления имеет значение ширина импульса. Типичная частота следования импульсов составляет от 400 Гц до 50 Гц.
На стандартном сервоприводе центральное положение достигается за счет подачи импульсов шириной 1,5 миллисекунды, положение -45 градусов от центра диапазона за счет импульсов в 1 миллисекунду и положение +45 градусов за счет подачи импульсов в 2 миллисекунды. Вам нужно будет прочитать техническое описание сервопривода, чтобы определить ширину импульса, необходимую для различных углов. Однако для этого проекта вы используете библиотеку сервопривода, поэтому вам не о чем беспокоиться: библиотека предоставляет сервопривод требуемый сигнал ШИМ. Каждый раз, когда вы отправляете сервообъекту другое значение угла,код в библиотеке заботится о преобразовании угла в сигнал соответствующей ширины импульса сервопривода.
Некоторые сервоприводы обеспечивают непрерывное вращение. В качестве альтернативы вы можете относительно легко модифицировать стандартный сервопривод, чтобы обеспечить непрерывное вращение.
Сервопривод непрерывного вращения управляется таким же образом, обеспечивая угол от 0 до 180 градусов.Однако значение 0 обеспечит вращение на полной скорости в одном направлении, значение 90 будет неподвижным, а значение 180 обеспечит вращение на полной скорости в противоположном направлении. Значения между ними заставят сервопривод вращаться в одном или другом направлении и с разными скоростями. Сервоприводы с непрерывным вращением отлично подходят для создания небольших роботов (см. Рис. 9-4). Их можно соединить с колесами, чтобы обеспечить точное управление скоростью и направлением каждого колеса.

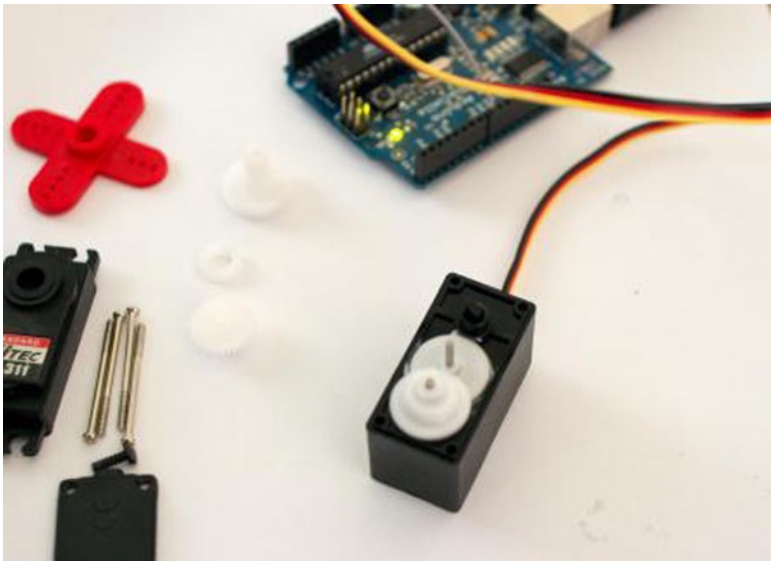


Рис. 9-4. *Модификация сервопривода для обеспечения непрерывного вращения*

Существует еще один вид сервопривода, известный как линейный привод, который вращает вал в желаемое положение, позволяя вам перемещать предметы, соединенные с концом вала. Они часто используются в телепрограмме «Разрушители мифов» их постоянным экспертом по робототехнике Грантом Имахарой.

Обратите внимание, что если сервопривод «рычит» на одном конце диапазона, Arduino пытается вывести сервопривод за пределы его диапазона, что приведет к высокому току. Этого можно избежать, указав второй и третий аргументы при вызове servo.attach () и увеличив минимальную ширину импульса с 544 микросекунды по умолчанию или уменьшив максимальную ширину импульса с 2400 микросекунд по умолчанию до тех пор, пока рычание не перестанет происходить, когда заданное положение не меняется.

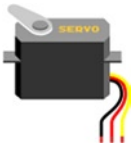
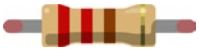
Проект 26 - Управление двумя сервоприводами

Теперь вы создадите еще один простой проект, но на этот раз вы будете управлять двумя сервоприводами с помощью команд монитора последовательного порта. Вы узнали о последовательном управлении в Project 10, когда меняли цвета на RGB-лампе с помощью последовательных команд. Итак, давайте возьмем код из проекта 10, чтобы создать этот.

Требуемые детали

Для этого проекта требуются два сервопривода. Потенциометр вам не понадобится. Детали перечислены в таб. 9-2.

Таб. 9-2. *Детали, необходимые для проекта 26*

Стандартный RC сервопривод	
2 резистора 220 Ом	

Подключение

Схема для проекта 26, опять же, предельно проста. Подключите его, как показано на рис. 9-5. По сути, вы удаляете потенциометр из последнего проекта и подключаете второй сервопривод к цифровому выводу 6 через резистор 220 Ом.

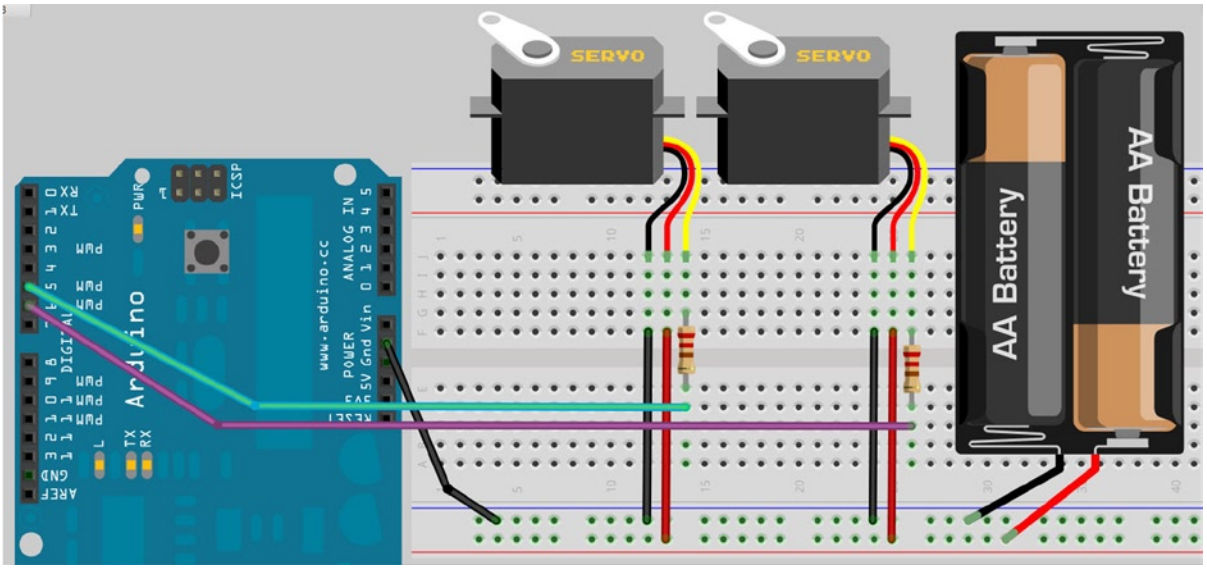


Рис. 9-5. *Схема для проекта 26 - Двойное сервоуправление*

Введите код

Введите код из Листинга 9-2.

Листинг 9-2. Код для проекта 26

```
// Проект 26
#include <Servo.h>

char buffer[11];
Servo servo1;  // Создаем сервообъект
Servo servo2;  // Создаем второй сервообъект
```

```
void setup()
{
        servo1.attach(5);  // Присоединяет сервопривод на выводе 5 к объекту
        servo1servo2.attach(6);  // Присоединяет сервопривод на выводе 6 к объекту
        servo2Serial.begin(9600);
        while(Serial.available())
                Serial.read();
        servo1.write(90);  // Поместим servo1 в исходное положение
        servo2.write(90);  // Поместите servo2 в исходное положение
        Serial.println
}       ("STARTING...");

void loop()
{
        if (Serial.available() > 0) { // Проверяем, введены ли данные
                int index=0;
                delay(100); // Подождите, пока буфер заполнится
                int numChar = Serial.available(); // Находим длину строки
                if (numChar>10) {
                numChar=10;
                }
                while (numChar--) {
                        // Заполняем буфер строкой
                        buffer[index++] = Serial.read();
                }
                buffer[index]='\0';
         splitString(buffer); // Запускаем функцию splitString
        }
}

void splitString(char* data) {
        Serial.print("Data entered: ");
        Serial.println(data);
        char* parameter;
        parameter = strtok (data, " ,"); //Строка для токена
        while (parameter != NULL) { // Если мы не достигли конца строки ...
                setServo(parameter); // ... запускаем функцию setServo
                parameter = strtok (NULL, " ,");
        }
        while(Serial.available())
                Serial.read();
}

void setServo(char* data) {
        if ((data[0] == 'L') || (data[0] == 'l')) {
                int firstVal = strtol(data+1, NULL, 10); // Строка в длинное целое число
                firstVal = constrain(firstVal,0,180); // Ограничение значений
                servo1.write(firstVal);
                Serial.print("Servo1 is set to: ");
                Serial.println(firstVal);
        }
```

```
        if ((data[0] == 'R') || (data[0] == 'r')) {
                int secondVal = strtol(data+1, NULL, 10); // Строка в длинное целое число
                secondVal = constrain(secondVal,0,255);   //Ограничиваем значения
                servo2.write(secondVal);
                Serial.print("Servo2 is set to: ");
                Serial.println(secondVal);
        }
}
```

Чтобы запустить код, откройте окно МПП. Arduino перезагрузится, и сервоприводы переместятся в свои центральные места. Теперь вы можете использовать МПП для отправки команд на Arduino.
Левый сервопривод управляется отправкой L, а затем числа от 0 до 180 для угла. Правый сервопривод управляется отправкой R и числа. Вы можете отправлять отдельные команды каждому сервоприводу или отправлять обе команды одновременно, разделяя команды пробелом или запятой, например:

```
L180
L45 R135
L180,R90
R77
R25 L175
```

Это простой пример того, как вы можете отправлять команды по проводам на руку робота, управляемую Arduino, или на игрушку-аниматрон. Обратите внимание, что последовательные команды не обязательно должны поступать от монитора последовательного порта Arduino. Вы можете использовать любую программу, способную обмениваться данными через последовательный порт, или написать свою собственную на Python или C ++.

Проект 26 - Управление двумя сервоприводами – Обзор кода

Код этого проекта в основном не отличается от кода проекта 10. Поэтому я не буду вдаваться в подробности каждой команды. Прочтите Project 10, чтобы узнать, как работают команды манипулирования строками.

Сначала включается заголовочный файл библиотеки Servo.h:

```
#include <Servo.h>
```

Затем создается массив типа char для хранения текстовой строки, которую вы вводите в качестве команды в МПП:

```
char buffer[11];
```

Создаются два сервообъекта:

```
Servo servo1;  // Создаем сервообъект
Servo servo2;  // Создаем второй сервообъект
```

В программе настройки прикрепите сервообъекты к выводам 5 и 6:

```
servo1.attach(5);  // Присоединяет сервопривод на выводе 5 к объекту servo1
servo2.attach(6);  // Присоединяет сервопривод на выводе 6 к объекту servo2
```

Затем начинаем последовательную связь, а затем выполняем некоторое время (Serial.available ()) Serial.read (); который считывает данные только тогда, когда они доступны.

```
Serial.begin(9600);
while(Serial.available())
Serial.read();
```

Оба сервопривода имеют значение 90, которое является центральной точкой, записанное для них, так что они устанавливаются в центральное положение:

```
servo1.write(90);  // Поместим servo1 в исходное положение
servo2.write(90);  // Поместите servo2 в исходное положение
```

Затем в окне монитора последовательного порта (МПП) отображается слово «STARTING .....», чтобы вы знали, что устройство готово к приему команд:

```
Serial.println("STARTING...");
```

В основном цикле проверим, были ли отправлены какие-либо данные по последовательной линии.

```
if (Serial.available() > 0) { // проверяем, введены ли данные
```

и если это так, заполняем буфер и получаем длину строки, гарантируя, что она не превысит максимум 10 символов. Как только буфер заполнится, вы вызываете процедуру splitString, отправляющую буферный массив в функцию:

```
int index=0;
delay(100); // Подождите, пока буфер заполнится
int numChar = Serial.available(); // Находим длину строки
if (numChar>10) {
        numChar=10;
}
while (numChar--) {
        // Заполняем буфер строкой
        buffer[index++] = Serial.read();
}
Buffer[index]="\0";
splitString(buffer); // Запускаем функцию splitString
```

Функция splitString принимает массив буферов, разбивает его на отдельные команды, если введено более одной, и вызывает подпрограмму setServo с параметром, удаленным из командной строки, полученной по последовательной линии:

```
void splitString(char* data) {
        Serial.print("Data entered: ");
        Serial.println(data);
        char* parameter;
        parameter = strtok (data, " ,"); //Строка для токена
        while (parameter != NULL) { // Если мы не достигли конца строки ...
                setServo(parameter); // ... запускаем функцию setServo
                parameter = strtok (NULL, " ,");
        }
        while(Serial.available())
                Serial.read();
}
```

Подпрограмма setServo получает меньшую строку, отправленную из функции splitString, и проверяет, введено ли L или R, и если да, перемещает вал левого или правого сервоприводов на величину, указанную встроке:

```
void setServo(char* data) {
        if ((data[0] == 'L') || (data[0] == 'l')) {
                int firstVal = strtol(data+1, NULL, 10);  // Строка в длинное целое число
                firstVal = constrain(firstVal,0,180);     // Ограничение значений
                servo1.write(firstVal);
                Serial.print("Servo1 is set to: ");
                Serial.println(firstVal);
        }
        if ((data[0] == 'R') || (data[0] == 'r')) {
                int secondVal = strtol(data+1, NULL, 10); // Строка в длинное целое число
                secondVal = constrain(secondVal,0,255);   // Ограничиваем значения
                servo2.write(secondVal);
                Serial.print("Servo2 is set to: ");
                Serial.println(secondVal);
```

Я пропустил описание этих двух последних функции, поскольку они почти идентичны функциям в Project 10.
Если вы не можете вспомнить, что было описано в Project 10, не стесняйтесь вернуться и перечитать это.

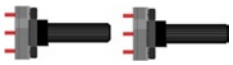
Проект 27 - Сервоуправление джойстиком

Для другого простого проекта давайте воспользуемся джойстиком для управления двумя сервоприводами. Вы разместите сервоприводы таким образом, чтобывы получаете наклонно-поворотную головку, которая используется для камер видеонаблюдения или для крепления камер или датчиков на роботах.

Требуемые детали

Оставьте схему, как это было для последнего проекта, и добавьте либо два потенциометра, либо джойстик с двухкоординатным потенциометром.
Список деталей см. В Таблице 9-3.

Таб. 9-3. *Детали, необходимые для проекта 27*

Стандартный RC сервопривод × 2	
2-осевой джойстик с потенциометром	 
2 резистора 220 Ом	

Подключение

Установите один сервопривод поверх другого Рис. 9-7.

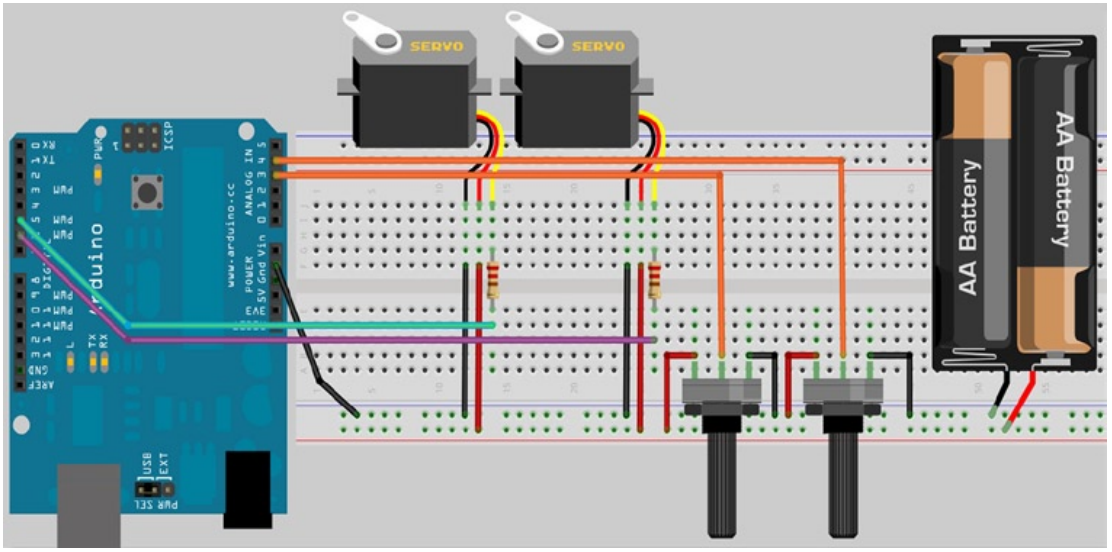


Рис. 9-6. *Схема для проекта 27 - сервоуправление джойстиком*

Джойстик с потенциометром - это просто джойстик, состоящий из двух потенциометров, расположенных под прямым углом друг к другу. Оси горшков соединены с рычагом, который поворачивается назад и вперед штырем и возвращается в свое центральное положение с помощью набора пружин.

Таким образом, подключение осуществляется легко: крайние выводы двух потенциометров подключаются к + 5 В и земле, а центральные выводы подключаются к аналоговым контактам 3 и 4 (вместо аналоговоговывода 1, используемого в Project 25) через резисторы. Если у вас нет джойстика, достаточно двух потенциометров, расположенных под углом 90 градусов друг к другу. Соедините два сервопривода так, чтобы один имел ось вертикально, а другой - горизонтально под углом 90 градусов к первому сервоприводу и был прикреплен к якорю первого сервопривода сбоку. См. Рисунок 9-7, где показано, как подключить сервоприводы. Для тестирования подойдет немного термоклея. Используйте более прочный клей для прочной фиксации.

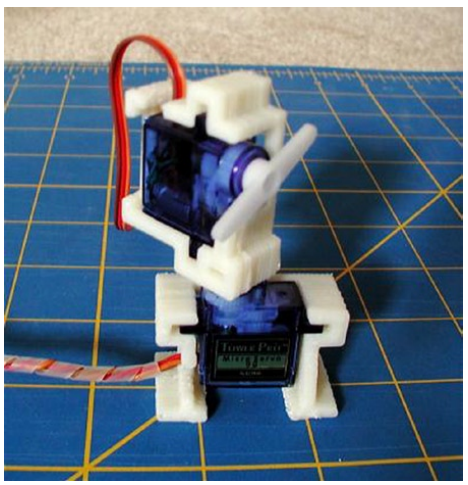


Рис. 9-7. *Установите один сервопривод поверх другого*

Как вариант, приобретите один из готовых сервоприводов панорамирования и наклона, которые вы можете купить для робототехники. Их можно дешево купить на eBay.
Когда нижний сервопривод движется, он заставляет вращаться верхний сервопривод, а когда верхний сервопривод движется, его рычаг раскачивается взад и вперед. Например, вы можете прикрепить к руке веб-камеру или ультразвуковой датчик.
Джойстик можно приобрести на eBay или у поставщика электроэнергии. Также можно было найти старый джойстик C64 или Atari.
Однако есть и дешевая альтернатива, называемая контроллером PS2: он содержит два джойстика с двухосевым потенциометром, а также набор вибромоторов и других кнопок. Их можно очень дешево купить на eBay, и их легко разобрать, чтобы получить доступ к внутренним частям (см. Рис. 9-8). Если вы не хотите разбирать контроллер, вы можете получить доступ к цифровому коду, идущему по кабелю контроллера PS2. Фактически, существуют библиотеки Arduino, которые позволят вам это сделать. Это даст вам полный доступ сразу ко всем джойстикам и кнопкам на устройстве.

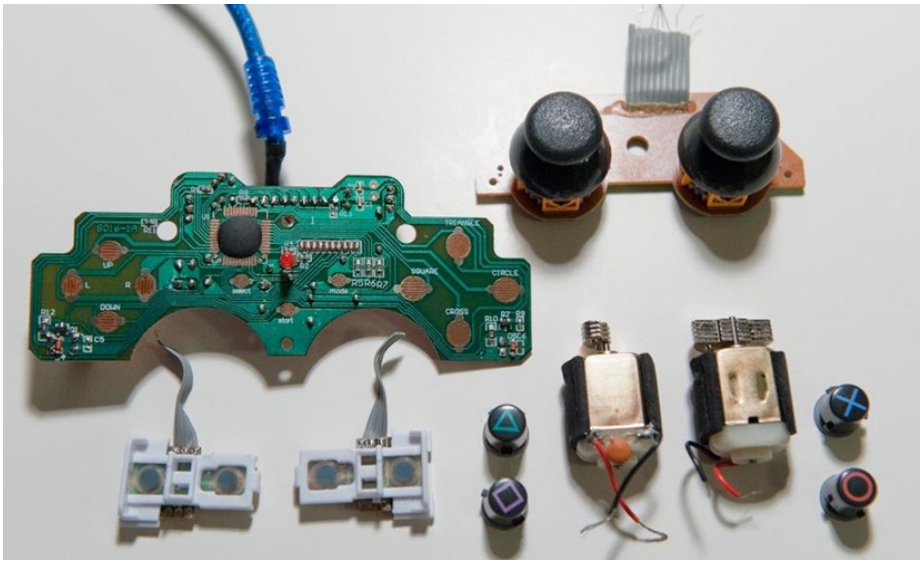


Рис. 9-8. *Все замечательные детали, доступные внутри контроллера PS2*

Введите код

Введите код из листинга 9-3.

Листинг 9-3. Код для проекта 27

```
// Project 27
#include <Servo.h>

Servo servo1;  // Создаем сервообъект
Servo servo2;  // Создаем второй сервообъект
int pot1, pot2;
```

```
void setup()
{
  servo1.attach(5);  // Присоединяет сервопривод на выводе 5 к объекту servo1
  servo2.attach(6);  // Присоединяет сервопривод на выводе 6 к объекту servo2

  servo1.write(90);  // Поместите servo1 в исходное положение
  servo2.write(90);  // Поместите servo2 в исходное положение

}

void loop()
{
  pot1 = analogRead(3); // Read the X-Axis
  pot2 = analogRead(4); // Read the Y-Axis
  pot1 = map(pot1,0,1023,0,180);
  pot2=map(pot2,0,1023,0,180);
  servo1.write(pot1);
  servo2.write(pot2);
  delay(15);
}
```

Когда вы запустите эту программу, вы сможете использовать сервоприводы в качестве поворотно-наклонной головки. Покачивание джойстика назад и вперед приведет к раскачиванию якоря верхнего сервопривода вперед и назад, а перемещение джойстика из стороны в сторону вызовет вращение нижнего сервопривода.Если вы обнаружите, что сервоприводы движутся в направлении, противоположном ожидаемому, то внешние выводы соответствующего сервопривода подключены неправильно. Просто поменяйте их местами.

Проект 27 - Сервоуправление джойстиком - Обзор кода

Опять же, это очень простой проект, но эффект от работы двух сервоприводов весьма убедителен.

Библиотека серво загружена:

```
#include <Servo.h>
```

Создаются два сервообъекта, и два набора целых чисел содержат значения, считываемые с двух потенциометров внутри джойстика:

```
Servo servo1;  // Создаем сервообъект
Servo servo2;  // Создаем второй сервообъект
int pot1, pot2;
```

Цикл настройки прикрепляет два сервообъекта к контактам 5 и 6 и перемещает сервоприводы в центральные положения:

```
servo1.attach(5);  // Присоединяет сервопривод на выводе 5 к объекту servo1
servo2.attach(6);  // Присоединяет сервопривод на выводе 6 к объекту servo2

servo1.write(90);  // Поместите servo1 в исходное положение
servo2.write(90);  // Поместите servo2 в исходное положение
```

В основном цикле аналоговые значения считываются с осей X и Y джойстика:

```
pot1 = analogRead(3); // Считываем ось X
pot2 = analogRead(4); // Считываем ось Y
```

Затем эти значения отображаются в диапазоне от 0 до 180 градусов.

```
pot1 = map(pot1,0,1023,0,180);
pot2 = map(pot2,0,1023,0,180);
```

и отправляются на два сервопривода

```
servo1.write(pot1);
servo2.write(pot2);
```

Диапазон движения, доступный для этой установки с панорамированием / наклоном, потрясающий, и вы можете заставить ее двигаться как человек. Такой тип сервосистемы часто используется для управления камерой при аэрофотосъемке (см. Рис.9-9).

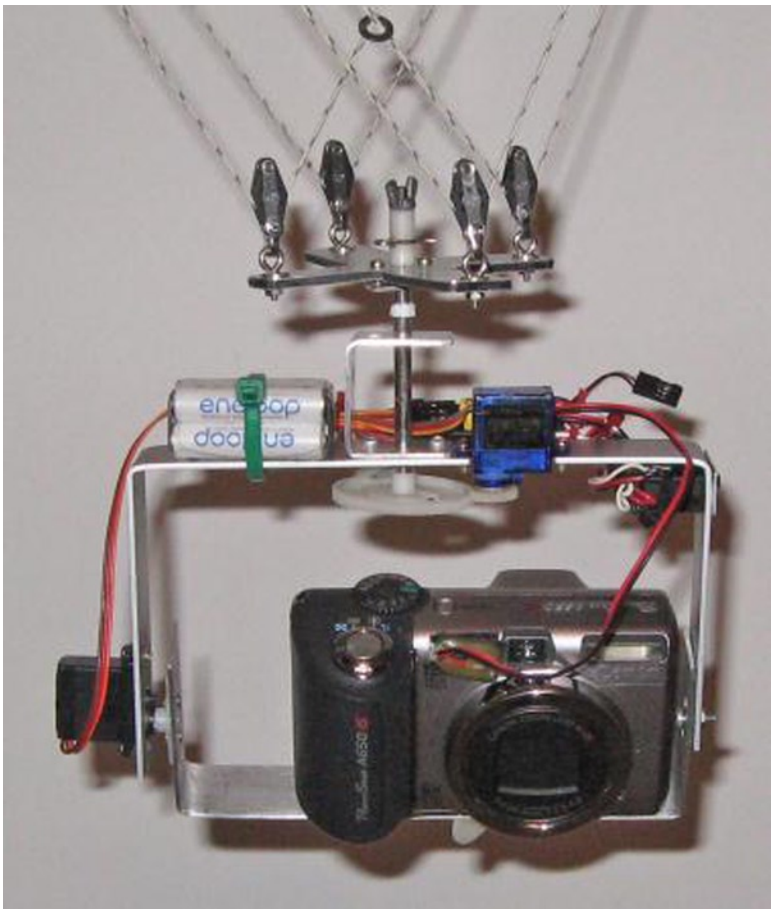


Рис. 9-9. *Устройство панорамирования / наклона, созданное для камеры с использованием двух сервоприводов (изображение Дэвида Митчелла)*

Резюме

В главе 9 вы проработали три очень простых проекта, показывающих, насколько легко сервоприводами можно управлять с помощью библиотеки servo.h. Вы можете легко изменить эти проекты, добавив до 12 сервоприводов, чтобы, например, игрушечный динозавр мог передвигаться реалистично.Сервоприводы отлично подходят для создания собственного радиоуправляемого автомобиля или для управления роботом. Кроме того, как показано на Рисунке 9-9, вы можете сделать установку панорамирования / наклона для управления камерой. Третий сервопривод можно даже использовать для нажатия кнопки спуска затвора на камере.

Предметы и понятия, затронутые в главе 9.

- Многочисленные возможности использования сервопривода
- Как использовать библиотеку сервоприводов для управления от 1 до 12 сервоприводов
- Как использовать потенциометр в качестве регулятора сервопривода
- Как работает сервопривод
- Как модифицировать сервопривод для обеспечения непрерывного вращения
- Как управлять набором сервоприводов с помощью последовательных команд
- Как использовать аналоговый джойстик для двухосного сервоуправления
- Как расположить два сервопривода для создания поворотно-наклонной головки
- Как контроллер PS2 может стать отличным источником для деталей джойстика и кнопок

ГЛАВА 10

Шаговые двигатели и роботы

Теперь вы познакомитесь с новым типом двигателя, называемым шаговым двигателем. Project 28 - это простой проект, который покажет вам, как управлять шаговым двигателем, заставлять его перемещаться на заданное расстояние и изменять его скорость и направление. Шаговые двигатели отличаются от стандартных двигателей тем, что их вращение разделено на несколько этапов.

Управляя синхронизацией и количеством шагов, выдаваемых двигателю, вы можете довольно точно управлять скоростью двигателя и углом поворота вала.

Шаговые двигатели бывают разных форм и размеров и имеют четыре, пять, шесть или восемь обмоток. Шаговые двигатели имеют множество применений; они используются в планшетных сканерах для позиционирования сканирующей головки и в струйных принтерах для управления расположением печатающей головки и бумаги.

В другом проекте в этой главе вы используете моторный шилд с редукторными двигателями ДС для управления шасси робота. В итоге вы заставите робота следовать по черной линии на полу!

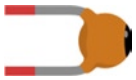
Проект 28 - Базовое управление шаговым двигателем

В этом очень простом проекте вы подключите шаговый двигатель, а затем заставите Arduino управлять им в разных направлениях и с разной скоростью в течение заданного количества шагов.

Требуемые детали

Вам понадобится биполярный или униполярный шаговый двигатель постоянного тока. При тестировании этого проекта я использовал униполярный шаговый двигатель Sanyo 103H546-0440.

Таб. 10-1. *Детали для проекта 28*

Шаговый двигатель	
L293D или SN754410 Микросхема драйвера двигателя	
*2 × 0,01 мкФ керамические конденсаторы	
*Токоограничивающий резистор	

Подключение

Подключите все, как показано на рис. 10-1. См. Вариант для биполярных двигателей.

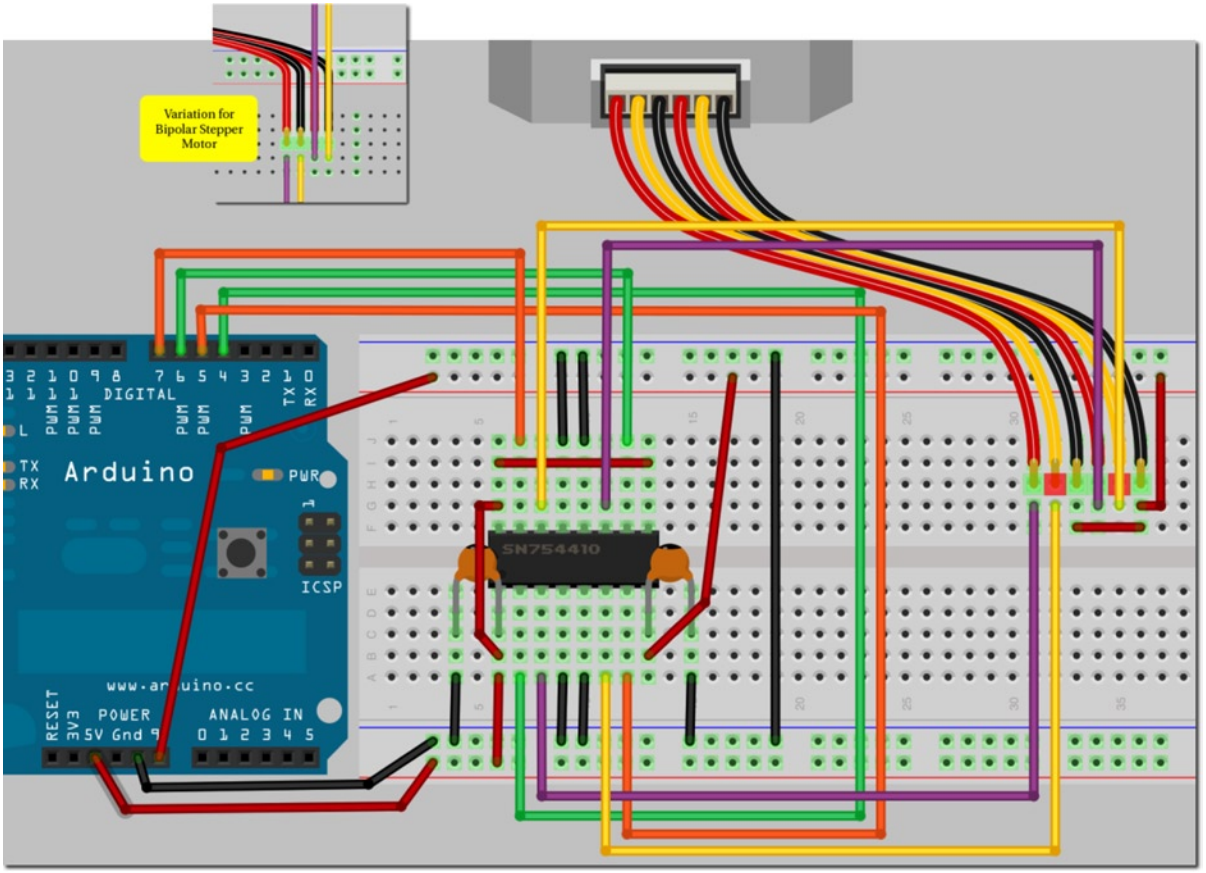


Рис. 10-1. *Схема для Проекта 28 - Базовое управление шаговым двигателем*

Убедитесь, что Arduino и L293 подключены к внешнему источнику питания постоянного тока, чтобы не перегружать Arduino. Arduino не нужно подключать к внешнему источнику питания, в отличие от L293 (или SN754410). Лучше всего подключить положительный вывод внешнего источника питания непосредственно к макетной плате и контакту 8 L293 и подключить к земле от Arduino на макетной плате. Если сделать это вместо того, чтобы подключать контакт 8 L293 к контакту Vin на разъеме платы Arduino, большие токи (которые могут создавать электрические помехи и нарушить работу) не будут проходитьчерез Arduino. (Этот совет верен всякий раз, когда вы управляете мощными устройствами, такими как двигатели). Вы также можете использовать электролитические конденсаторы с низким энергопотреблением, но убедитесь, что вы правильно подключили их (+ к питанию, - к земле), поскольку они полярные и могут быть повреждены или взорваны при переплюсовке! Я обнаружил, что без них схема не работает.

Вам также может потребоваться токоограничивающий резистор между источником питания и шиной питания, питающей микросхему SN754410 или L293D. Используйте соответствующий источник питания, который соответствует диапазону напряжения и тока вашего двигателя. Также убедитесь, что номинальная мощность любых токоограничивающих резисторов, которые вы используете, превышает ток, требуемый двигателем, иначе резистор нагреется и сгорит. Обратите внимание, что напряжение привода двигателя также будет ниже входного напряжения (даже без резистора) из-за внутренних падений напряжения в драйвере (которые в сумме составляют от 1,5 до 4 вольт, в зависимости от тока и других условий).

Цифровые выводы 4, 5, 6 и 7 на Arduino идут ко входам 1, 2, 3 и 4 на драйвере двигателя (см. Рисунок 10-2). Выходные выводы 1 и 2 драйвера двигателя проходят через катушку 1 двигателя, а выходные выводы 3 и 4 - на катушку 2. Вам нужно будет по техническому описанию вашего конкретного двигателя выяснить, какие цветные провода идут к катушке 1, а какие - к катушка 2. У униполярного двигателя также будет 5-й и / или 6-й провод, идущий на массу.

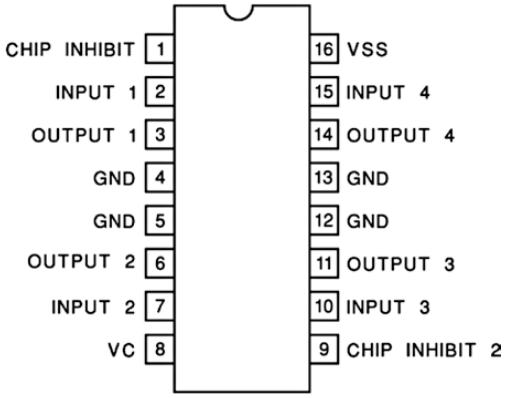


Рис. 10-2. *Цоколевка микросхемы драйвера двигателя L293D или SN754410*

Вывод Vin (входное напряжение) на Arduino подключается к выводу 8 микросхемы драйвера (VC). Все выводы 4, 5, 12 и 13 заземлены (см. Рисунок 10-2).

Убедившись, что все подключено должным образом, введите приведенный ниже код.

Введите код

Введите код из Листинга 10-1.

Листинг 10-1. Код для проекта 28

```
// Проект 28
#include <Stepper.h>

// значение шага - 360 / градусный угол мотора
#define STEPS 200

// создаем шаговый объект на контактах 4, 5, 6 и 7
Stepper stepper(STEPS, 4, 5, 6, 7);
```

```
void setup()
{
}

void loop()
{
    stepper.setSpeed(60);
    stepper.step(200);
    delay(100);
    stepper.setSpeed(20);
    stepper.step(-50);
    delay(100);
}
```

Перед запуском кода убедитесь, что ваш Arduino питается от внешнего источника постоянного тока. Когда скетч будет запущен, вы увидите, что шаговый двигатель совершает полный оборот, останавливается на короткое время, затем вращается назад на четверть оборота, останавливается на короткое время, затем все повторяется. В помощь можно приложить небольшой язычок из липкой ленты к шпинделю двигателя, чтобы вы могли видеть, как он вращается.

Проект 28 - Базовое управление шаговым двигателем - Обзор кода

Код этого проекта, опять же, простой, благодаря библиотеке stepper.h, которая делает за нас всю тяжелую работу.
Сначала вы включаете библиотеку в эскиз:

```
#include <Stepper.h>
```

Затем вам нужно определить, сколько шагов требуется двигателю для полного вращения на 360 градусов. Обычно шаговые двигатели бывают с углом поворота 7,5 или 1,8 градуса, но у вас может быть шаговый двигатель с другим углом шага. Чтобы проработать шаги, просто разделите 360 на угол шага. В случае с шаговым двигателем, который я использовал, угол шага составлял 1,8 градуса, то есть для полного вращения на 360 градусов требовалось 200 шагов:

```
#define STEPS 200
```

Затем вы создаете объект шагового двигателя, называете его шаговым и назначаете контакты, идущие по обе стороны от двух катушек:

```
Stepper stepper(STEPS, 4, 5, 6, 7);
```

Функция настройки вообще ничего не делает, но должна быть включена:

```
void setup()
{
}
```

В основном цикле вы сначала устанавливаете скорость двигателя в об / мин (обороты в минуту). Это делается спомощью команды .setSpeed, а скорость в об / мин указывается в скобках:

```
stepper.setSpeed(60);
```

Затем вы говорите степперу, сколько шагов он должен выполнить. Вы устанавливаете 200 шагов при 60 об / мин, что означаетсовершить полный оборот за одну секунду:

```
stepper.step(200);
```

В конце этого оборота выполняется задержка в 1 / секунды:

```
delay(100);
```

Затем скорость снижается до 20 об / мин:

```
stepper.setSpeed(20);
```

Затем двигатель заставляют вращаться в противоположном направлении (отсюда отрицательное значение) на 50 шагов, что для шагового двигателя 200 составляет четверть оборота:

```
stepper.step(-50);
```

Затем выполняется еще одна задержка в 100 миллисекунд, и цикл повторяется.

Как видите, управлять шаговым двигателем очень просто, но при этом вы можете контролировать его скорость, направление, в котором он движется, и то, насколько точно будет вращаться вал. Таким образом, вы можете точно контролировать, где и на сколько вращается элемент, прикрепленный к валу. Использование шагового двигателя для управления колесом робота даст очень точное управление, а вращение головы или камеры робота позволит вам расположить ее так, чтобы она указывала именно там, где вы этого хотите.

Проект 28 - Базовое управление шаговым двигателем - Обзор

Новое что вы используете в этом проекте - это шаговый двигатель. Шаговый двигатель - это двигатель постоянного тока, в котором вращение можно разделить на несколько более мелких шагов. Это достигается за счет прикрепления ротора в форме железной шестерни к валам внутри двигателя. Вокруг мотора находятся электромагниты с зубьями. Одна катушка находится под напряжением, в результате чего зубья железного ротора совпадают с зубьями электромагнита. Зубья следующего электромагнита немного смещены относительно первого; когда он находится под напряжением и первая катушка выключена, это заставляет вал немного вращаться в направлении следующего электромагнита. Этот процесс повторяется, несмотря на то, что внутри находится много электромагнитов, до тех пор, пока зубцы не будут почти выровнены с первым электромагнитом, и процесс повторяется.

Каждый раз, когда на электромагнит подается питание и ротор слегка перемещается, он выполняет один шаг. Изменяя последовательность электромагнитов, возбуждающих ротор, он поворачивается в противоположном направлении.

Задача Arduino - применить к катушкам соответствующие команды HIGH и LOW в правильной последовательности и скорости, чтобы вал мог вращаться. Это то, что происходит за кулисами библиотеки stepper.h.

У униполярного двигателя пять или шесть проводов, идущих к четырем катушкам. Центральные штыри катушек связаны вместе и идут к источнику питания. Биполярные двигатели обычно имеют четыре проводаи не имеют общих центральных соединений. На рис. 10-3 показаны различные типы и размеры шаговыхдвигателей.

Рис. 10-3. *Различные виды шаговых двигателей*

Последовательность шагов для шагового двигателя можно увидеть в Таб. 10-2.

Таб. 10-2. *Последовательность шагов для шагового двигателя*

Step	Wire 1	Wire 2	Wire 4	Wire 5
1	HIGH	LOW	HIGH	LOW
2	LOW	HIGH	HIGH	LOW
3	LOW	HIGH	LOW	HIGH
4	HIGH	LOW	LOW	HIGH

На рисунке 10-4 показана схема униполярного шагового двигателя. Как видите, катушек четыре (на самом деле катушек две, но они разделены центральными проводами на две катушки меньшего размера). Центральный провод идет к источнику питания, а два других провода идут к выходам на одной стороне ИС драйвера H-моста, а два других провода для второй катушки идут к последним двум выходам H-моста.

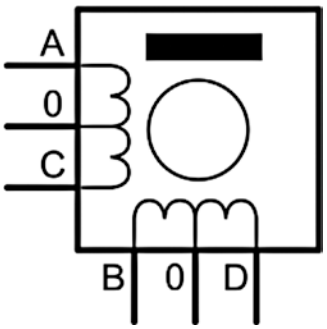


Рис. 10-4. *Принципиальная схема униполярного шагового двигателя*

На рисунке 10-5 показана схема биполярного шагового двигателя. На этот раз всего две катушки без отводов. Последовательность шагов для биполярного двигателя такая же, как для униполярного. Однако на этот раз вы меняете направление тока на катушки, т.е. ВЫСОКИЙ уровень соответствует положительному напряжению, а НИЗКИЙ - заземлению.

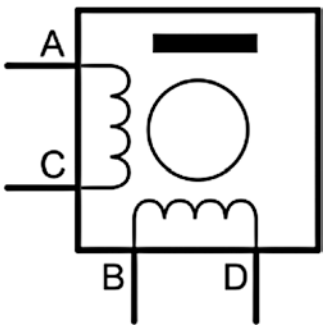


Рис. 10-5. *Принципиальная схема биполярного шагового двигателя*

Можно увеличить разрешение шагового двигателя, используя специальные методы полушага и микрошага. Это может повысить разрешающую способность двигателя, но за счет уменьшения крутящего момента и снижения надежности положения. Лучший способ увеличить разрешение - использовать зубчатую передачу.Разнообразные двигатели постоянного тока предлагают управление скоростью (с помощью модуляции напряжения или широтно-импульсной модуляции (ШИМ)) без управления положением. Серводвигатель обеспечивает управление положением, но не скоростью. Шаговый двигатель позволяет управлять скоростью и положением. Эти проекты были разработаны, чтобы дать простой обзор и введение в управление этими двигателями. Точность управления скоростью и / или положением зависит от соблюдения различных проектных характеристик соответствующих двигателей.
Чем больше вы увеличиваете скорость, ускорение или нагрузку на двигатель, тем больше вероятность того, что вы попадете в эти проектные ограничения и обнаружите, что двигатель больше не будет вести себя так, как вы ожидали. К счастью, есть много забавных и полезных вещей, которые можно выполнить, не перегружая доступные двигатели на пределе своих возможностей. Эта книга дает вам их обзор. Но если вы делаете что-то, что связано с безопасностью человека, важно, чтобы вы более подробно изучили двигатели и системы управления и тщательно проверили свои конструкции.

Теперь, когда вы знаете все об управлении шаговым двигателем, вы вернетесь к обычному управлению двигателем, но на этот раз вы будете использовать моторный шилд и управлять двумя двигателями, подключенными к колесу на шасси робота.

Проект 29 - Использование моторного шилда

Мы собираетесь использовать моторный шилд для раздельного управления двумя моторами. Вы узнаете, как управлять скоростью и направлением каждого двигателя, и примените эти знания для управления двухколесным роботом.

Требуемые детали

Вы можете запустить этот проект, используя всего два двигателя DC, если это все, что у вас есть. Так что либо получите базу для двухколесного робота, либо сделайте один из двух мотор-редукторов с колесами. Вращение на половинной скорости должно составлять около 20 см или 6 дюймов поступательного движения в секунду. Батарейный блок должен иметь достаточное напряжение для работы вашего Arduino, щита и двигателей. Я использовал 6 батареек AA в держателе с припаянным к проводу штекером для питания моей установки для тестирования этого проекта.

Таб. 10-3. *Детали, необходимые для проекта 29*

Мотор шилд	
2 × DC мотора ...	
... Двухколесное роботизированное шасси	
Аккумуляторная батарея	

Подключение

Поскольку для этого проекта не требуется ничего, кроме моторного шилда, подключенного к Arduino, и двух проводов ваших моторов, подключенных к четырем выходам от шилда, принципиальная схема не требуется. Вместо этого на рис. 10-6 вы видите красивую картинку моторного шилда, подключенного к Arduino. Подключите шилд и двигатель к внешнему источнику питания, а не к источнику питания от Arduino, чтобы обеспечить отсутствие помех и бесперебойную работу цепи.

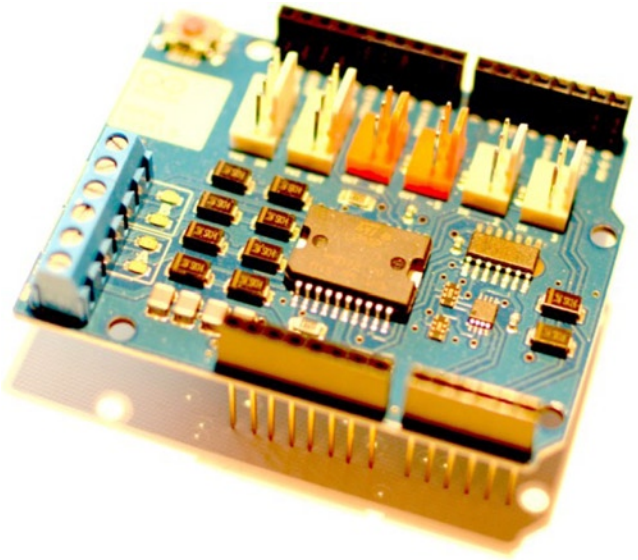


Рис. 10-6. *Моторный шилд Arduino*

Поскольку для этого проекта не требуется ничего, кроме моторного шилда, подключенного к Arduino, и двухпроводов ваших моторов, подключенных к четырем выходам от шилда, принципиальная схема не требуется. Вместо этого на рис. 10-6 вы видите красивую картинку моторного шилда, подключенного к Arduino. Подключите шилд и двигатель к внешнему источнику питания, а не к источнику питания от Arduino, чтобы обеспечить отсутствие помех и бесперебойную работу схемы.

Для тестирования этого проекта я использовал официальный Arduino Motor шилд,который предназначен для управления двумя двигателями. Также доступны шилды от Adafruit, которые позволят вам управлять четырьмя двигателями DC, двумя шаговыми двигателями или двумя сервоприводами. Шилд Adafruit также поставляется с отличной библиотекой AFMotor.h, которая упрощает управление двигателями, шаговыми двигателями или сервоприводами. Выбор остается за вами. Однако код этого проекта был разработан с учетом Arduino Motor Shield, поэтому, если вы используете другой шилд, вам нужно будет соответствующимобразом изменить код.

Подключите левый двигатель к клеммам двигателя A шилда двигателя, а правый двигатель - к клеммам двигателя B шилда. Я также использовал шасси двухколесного робота от DFRobot (см. Рис. 10-7). Это хорошая недорогая роботизированная шасси, которая поставляется с насадками и монтажными отверстиями для установки датчиков. Она идеально подходит для следующего проекта робота, следующего по линии. Однако для этого проекта подойдет любая роботизированное шасси; вы даже можете использоватьшасси четырехколесного робота, если вы просто не забываете управлять обоими наборами двигателей с каждой стороны, а не только одним. Многие аналогичные шасси для роботов доступны на eBay и Amazon. Конечно вы можете протестировать, просто используя для этого два двигателя постоянного тока.

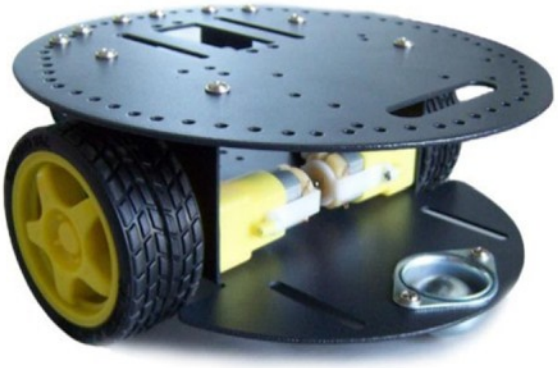


Рис. 10-7. *Шасси двухколесного робота*

Введите код

Введите код из Листинга 10-2 или из папки с кодами:

Листинг 10-2. Код для проекта 29

```
// Проект 29 - Использование моторного шилда
#define QUARTER_SPEED 64
#define HALF_SPEED 128
#define FULL_SPEED 255

// Устанавливаем пины для скорости и направления каждого двигателя
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;

void stopMotor() {
  // выключаем оба мотора
  analogWrite(speed1, 0);
  analogWrite(speed2, 0);
}

void setup()
{
  // устанавливаем все пины на выходы
  pinMode(speed1, OUTPUT);
  pinMode(speed2, OUTPUT);
  pinMode(direction1, OUTPUT);
  pinMode(direction2, OUTPUT);
}
```

```
void loop()
{
  // Оба мотора двигаются вперед со скоростью 50% в течение 2 секунд
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1, HALF_SPEED);
  analogWrite(speed2, HALF_SPEED);
  delay(2000);

  stopMotor(); delay(1000); // стоп

  // Левый поворот на 1 секунду
  digitalWrite(direction1, LOW);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1, HALF_SPEED);
  analogWrite(speed2, HALF_SPEED);
  delay(1000);

  stopMotor(); delay(1000); // стоп

  // Оба мотора двигаются вперед со скоростью 50% в течение 2 секунд
  digitalWrite(direction1, HIGH);
  digitalWrite(direction2, HIGH);
  analogWrite(speed1, HALF_SPEED);
  analogWrite(speed2, HALF_SPEED);
  delay(2000);

  stopMotor(); delay(1000); // стоп

  // вращаем вправо со скоростью 25%
  speeddigitalWrite(direction1, HIGH);
  digitalWrite(direction2, LOW);
  analogWrite(speed1, QUARTER_SPEED);
  analogWrite(speed2, QUARTER_SPEED);
  delay(2000);

  stopMotor(); delay(1000); // стоп

}
```

Проект 29 - Использование моторного шилда - Обзор кода

Сначала мы определяем три скорости, на которых мы хотим, чтобы двигатель работал. Вы можете выбрать любую из этих скоростей в своей программе.

```
#define QUARTER_SPEED 64
#define HALF_SPEED 128
#define FULL_SPEED 255
```

Затем назначьте, какие пины (контакты) управляют скоростью и направлением. На Arduino Motor Shield они закреплены на контактах 3, 11, 12 и 13:

```
int speed1 = 3;
int speed2 = 11;
int direction1 = 12;
int direction2 = 13;
```

Затем создайте функцию для выключения моторов. В коде мотор выключен четыре раза, поэтому есть смысл создать для этого функцию:

```
       void stopMotor() {
        // выключаем оба мотора
          analogWrite(speed1, 0);
          analogWrite(speed2, 0);
}
```

Чтобы выключить моторы, вам просто нужно установить скорость каждого мотора на ноль. Следовательно, вы записываете нулевое значение на контакты speed1 (левый двигатель) и speed2 (правый двигатель).

В цикле настройки четыре контакта установлены в режим вывода:

```
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

В основном цикле вы выполняете четыре отдельных режима движения. Сначала вы перемещаете робота вперед со скоростью 50 процентов в течение двух секунд:

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1, HALF_SPEED);
analogWrite(speed2, HALF_SPEED);
delay(2000);
```

Для этого сначала установите пины направления на HIGH - ВЫСОКИЙ, что соответствует прямому ходу (при условии, что ваши двигатели подключены правильно). Затем для контактов скорости обоих двигателей устанавливается значение HALF_SPEED, равное 128. Значения ШИМ варьируются от 0 до 255, так что 128 находится посередине между ними. Следовательно, рабочий цикл составляет 50 процентов, и двигателибудут работать с половинной скоростью. В каком бы направлении и скорости вы не установили двигатели, они будут продолжать работать. Следовательно, двухсекундная задержка гарантирует, что робот будет двигаться вперед со скоростью 50 процентов в течение двух секунд, что должно быть около метра.

Затем вы останавливаете моторы и ждете одну секунду:

```
stopMotor(); delay(1000); // stop
```

Следующая последовательность движений изменяет направление левого мотора на реверс, а правого мотора на вперед. Чтобы двигатель двигался вперед, установите его направляющий пин в положение HIGH; чтобы он пошел в обратном направлении, установите его на LOW - НИЗКИЙ. Скорость остается на уровне 50 процентов, поэтому значение 128 записывается на пины скорости обоих двигателей (линии для движения вперед на правом колесе и скорости не являются строго необходимыми, но я оставил их, чтобы вы могли изменить их, как вы хотите, чтобы получить разные движения).

Когда правое колесо движется вперед, а левое колесо движется назад, робот вращается против часовой стрелки (поворот влево). Это движение длится одну секунду, что должно повернуть его влево примерно на 90 градусов. После этой последовательности двигатель снова останавливается на одну секунду:

```
digitalWrite(direction1, LOW);
digitalWrite(direction2, HIGH);
analogWrite(speed1, HALF_SPEED);
analogWrite(speed2, HALF_SPEED);
delay(1000);
```

Следующая последовательность заставляет робота снова двигаться вперед со скоростью 50 процентов в течение двух секунд, после чего следует остановка на одну секунду:

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1, HALF_SPEED);
analogWrite(speed2, HALF_SPEED);
delay(2000);
```

Последняя последовательность движения изменяет направление колес таким образом, что левое колесо движется вперед, а правое колесо движется назад. Это заставит робота вращаться по часовой стрелке (повернуть направо). На этот раз скорость установлена на 64 или четверть скорости, что составляет 25 процентов рабочего цикла, что делает его вращаться медленнее, чем раньше. Вращение длится две секунды, чего должно хватить, чтобы робот развернулся на 180 градусов.

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, LOW);
analogWrite(speed1, QUARTER_SPEED);
analogWrite(speed2, QUARTER_SPEED);
```

Четыре последовательности вместе должны заставить вашего робота двигаться вперед на две секунды, остановиться, повернуть налево на 90 градусов, остановиться, снова двинуться вперед на две секунды, остановиться, затем повернуть с меньшей скоростью на 180 градусов и повторить сначала. Возможно, вам придется настроить время и скорость в соответствии с вашим собственным роботом, поскольку вы можете использовать другое напряжение, более быстрые или медленные мотор-редукторы и т. д. Поиграйте с последовательностями движений, чтобы получить робот мог двигаться быстрее или медленнее или поворачиваться с разной скоростью по вашему желанию.
Вся идея Project 29 состоит в том, чтобы вы привыкли к необходимым командам для перемещения шасси в разных направлениях и с разными скоростями. Если у вас есть только двигатели, но нет шасси робота, прикрепите ленту к шпинделю двигателей, чтобы вы могли видеть скорость и направление их вращения.

Проект 29 - Использование моторного шилда - Обзор оборудования

Новое оборудование, используемое в этом проекте, - моторный шилд. Шилд - это просто уже собранная печатная плата, размер которой удобен для размещения на Arduino. Шилд имеет контакты для подключения к контактам на Arduino; у них есть гнезда наверху, так что вы все еще можете получить доступ к контактам через шилд. Конечно, в зависимости от того, какой у вас тип шилда, некоторые из контактов Arduino будут использоваться шилдом и, следовательно, не будут использоваться в вашем коде. Например, шилд Ардумото использует цифровые выводы 10, 11, 12 и 13.
Преимущество шилдов в том, что они предоставляют дополнительную функциональность вашему Arduino, просто подключив его и загрузив необходимый код. Вы можете получить всевозможные шилды для расширения функций Arduino, включая такие вещи, как доступ к Ethernet, GPS, управление реле, доступ к SD и Micro SD картам, ЖК-дисплеи, возможность подключения к телевизору, беспроводная связь.
На рис. 10-8 показаны примеры различных видов экранов.

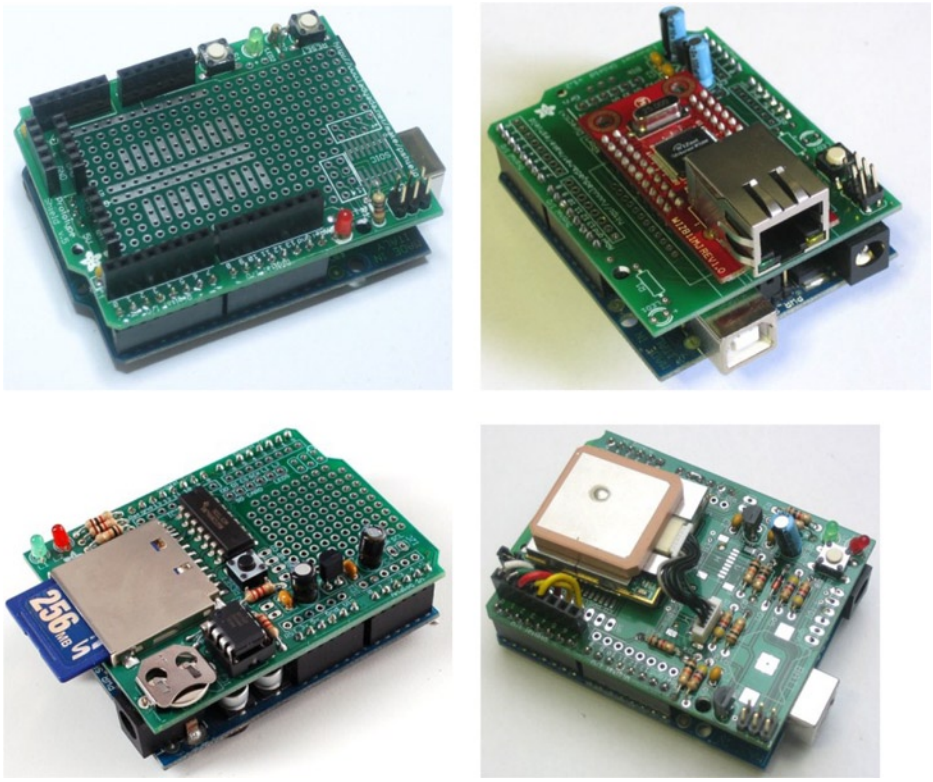


Рис. 10-8. *Прото шилд, ентернет шилд, шилд регистратора данных и шилд GPS*

Проект 30 - Робот-идущий по линии

Теперь, когда вы знаете, как управлять двумя двигателями или шасси робота с помощью моторного шилда, мы применим эти навыки с пользой! В этом проекте мы построим робота, который способен обнаруживать черную линию, проведенную на полу, и следовать по ней. Этот вид роботов до сих пор используется на заводах,где робот придерживается заданного пути через производственный цех. Они известны как автоматизированные транспортные средства (AGV).

Требуемые детали

Для этого проекта вам понадобится небольшая легкая шасси для робота. Их можно купить в готовом виде или в виде набора. Убедитесь, что колеса вращаются намного медленнее, чем двигатель, потому что робот должен иметь возможность двигаться довольно медленно, чтобы он мог перемещаться по линии. Поскольку робот будет автономным, ему потребуется собственный аккумулятор. Я обнаружил, что держателя для батареек размером шесть AA обеспечивает достаточно энергии для Arduino, датчиков, светодиодов, экрана и двигателей. Батареи большего размера продлят срок службы, но за счет большего веса, поэтому он будет зависеть от несущей способности вашего робота. Список деталей см. В Таб. 10-4.

Таб. 10-4. *Перечень компонентов для проекта 30*

Моторный шилд	
4 × токоограничивающие резисторы	
3 × Резисторы 1 кОм	
4 × белых светодиода	
3 × Светорезистора	
2 × двигателя DC или...	
... двухколесное шасси робота	
Аккумуляторная батарея	

Подключение

Подключите схему, как показано на рисунке 10-9.

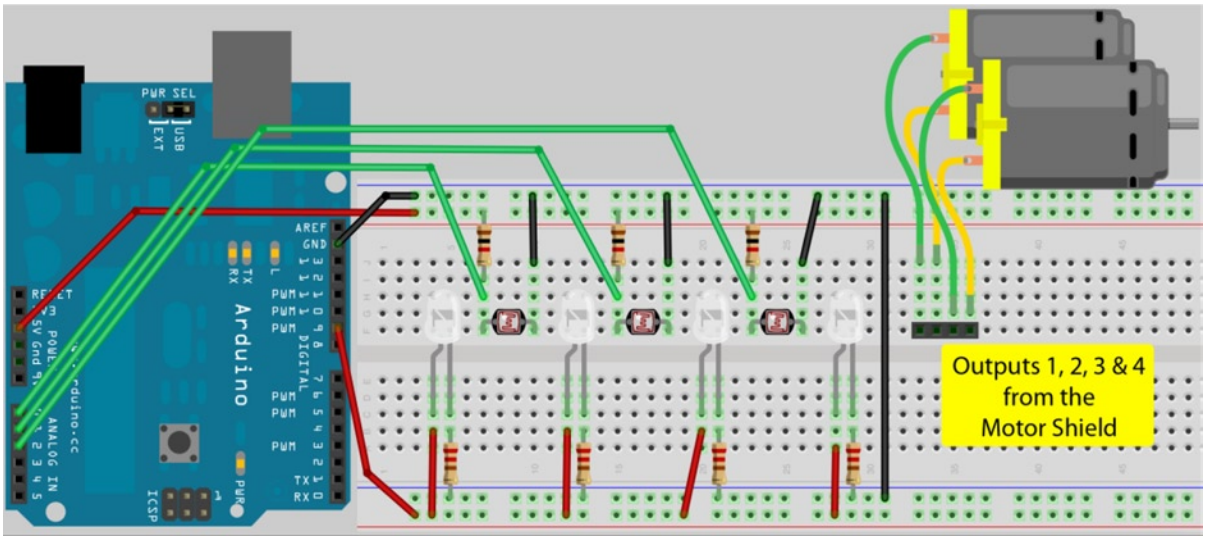


Рис. 10-9. *Схема для проекта 30 - Робот-следящий за линией*

Шилд не показан, но четыре выхода просто идут к мотору A и мотору B. Четыре светодиода могут быть любого цвета, если они имеют достаточную яркость, чтобы освещать пол и создавать контраст между полом и черной линией для обнаружения вашим датчиком. Белые светодиоды лучше всего, так как они обеспечивают максимальную яркость при минимальном токе. Четыре из них подключены параллельно контакту 9. Убедитесь, что вы используете подходящие токоограничивающие резисторы для ваших светодиодов.

Наконец, подключите три светорезистора к аналоговым контактам 0, 1 и 2. Одни выводы LDR идут на землю, а другие - на + 5 В через резистор 1 кОм. Провод от аналоговых выводов проходит между резистором и выводом LDR (вы использовали LDR в Project 14). Макетную плату выше можно использовать для тестирования. Однако для создания робота вам потребуется сделать сенсорную планку, на которой размещены четыре светодиода и три LDR. Они должны быть расположены так, чтобы все они были близко друг к другу и указывали в одном направлении.Я припаял светодиод и схему датчика на небольшой кусок картона и поместил три LDR между четырьмя светодиодами и чуть выше светодиодов, чтобы свет от них не падал на датчик. На рис. 10-10 показано, как я это сделал.

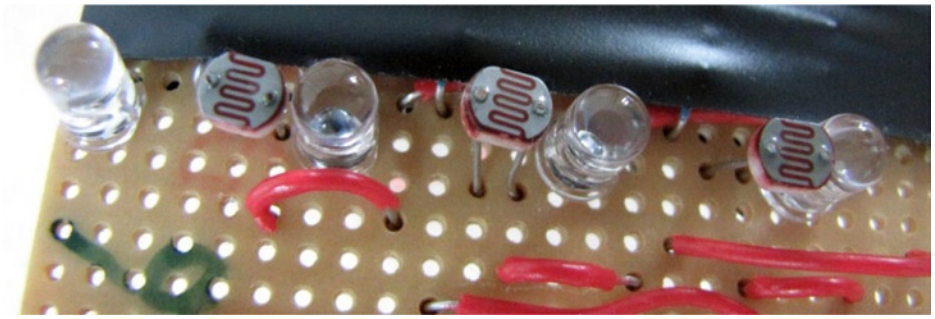


Рис. 10-10. *Сенсорная панель, состоящая из четырех светодиодов и трех LDR.*

Затем сенсорная планка прикрепляется к передней части вашего робота таким образом, чтобы светодиоды и LDR находились примерно на 1 см над полом (см. Рис. 10-11). Я просто приклеил ее изолентой к передней части своего робота, но для более надежного решения сенсорная планка должна быть прикручена к шасси.

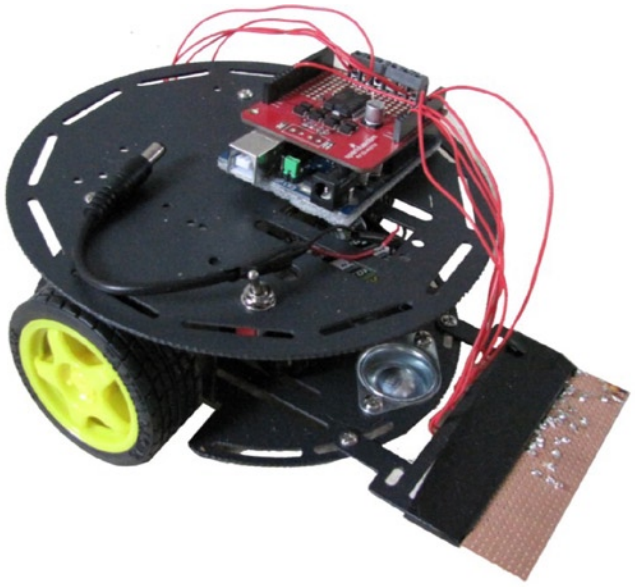


Рис. 10-11. *Шасси робота с сенсорной планкой*

Аккумулятор следует размещать по центру базы робота, чтобы не нарушать баланс нагрузки между колесами.
Как вы можете видеть на Рисунке 10-11, провода проходят между моторнымшилдом и двумя моторами, а также вниз к сенсорной планке. Один провод обеспечивает + 5 В, один для заземления, один - для + 5 В от цифрового контакта 9, а остальные три идут к аналоговым датчикам 0, 1 и 2, причем 0 является левым датчиком (когда вы смотрите на переднюю часть робот), где 1 - центральный датчик, а 2 - правый. На рис. 10-12 показано изображение моего робота в действии на полу моей кухни (на YouTube и Vimeo также есть видео сним в действии. Ищите «Arduino Controlled Line-Following Robot» - Робот,управляемый Arduino, следящий за линией - Майка МакРобертса).

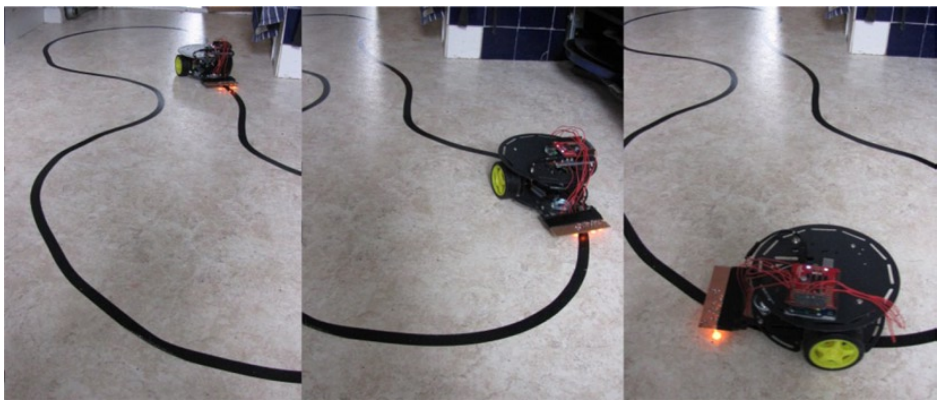


Рис. 10-12. *Мой собственный робот, следующий за линией, в действии*

Ввод кода

Когда вы построили своего робота и все проверили, введите код из Листинга 10-3 или из папки с кодами.

Листинг 10-3. Код для проекта 30

```
// Проект 30 - Робот, следящий за линией

int LDR1, LDR2, LDR3;               // значения датчиков

// калибровочные смещения
int leftOffset = 0, rightOffset = 0, center = 0;
// контакты для скорости и направления двигателя
int speed1 = 10, speed2 = 11, direction1 = 12, direction2 = 13;
// начальная скорость и смещение вращения
int startSpeed = 70, rotate = 30;
// порог датчика
int threshhold = 5;
int left = startSpeed, right = startSpeed;

int button = 2;                     // вывод кнопки (необязательно)
int motorsOff = true;               // mдвигатели изначально выключены, пока не будет нажата кнопка.
                                    // меняем начальное значение на false, если
                                    // на вашем роботе нет кнопки

// Процедура калибровки датчика
void calibrate() {

  for (int x=0; x<10; x++) {        // запускаем это 10 раз, чтобы получить среднее значение
  digitalWrite(9, HIGH);            // включенный свет
  delay(100);
  LDR1 = analogRead(0);             // считываем 3 датчика
  LDR2 = analogRead(1);
  LDR3 = analogRead(2);
  leftOffset = leftOffset + LDR1;   // добавляем значение левого датчика к итоговому значению
  center = center + LDR2;           // добавляем значение центрального датчика к итоговому значению
  rightOffset = rightOffset + LDR3; // добавляем значение правого датчика к сумме

  delay(100);
  digitalWrite(9, LOW);             // выключить свет
  delay(100);
  }
  // получаем среднее значение для каждого датчика
  leftOffset = leftOffset / 10;
  rightOffset = rightOffset / 10;
  center = center /10;
  // вычисляем смещения для левого и правого датчиков
  leftOffset = center - leftOffset;
  rightOffset = center - rightOffset;

}
```

```
void setup()
{
    pinMode(button, INPUT_PULLUP); //кнопка на выводе 2 (необязательно)
    // устанавливаем выводы двигателя на выходы
    pinMode(9, OUTPUT);
    pinMode(speed1, OUTPUT);
    pinMode(speed2, OUTPUT);
    pinMode(direction1, OUTPUT);pinMode(direction2,
    OUTPUT);
    // калибруем датчики
    calibrate();
    delay(3000);

    digitalWrite(9, HIGH);          // включить свет
    delay(100);

    // устанавливаем направление двигателя вперед
    digitalWrite(direction1, HIGH);
    digitalWrite(direction2, HIGH);
    // устанавливаем скорость обоих моторов
    analogWrite(speed1,left);analogWrite(speed2,right);

}

void loop() {
    if ( !digitalRead(button) ){
     // кнопка нажата. Включите или выключите моторы.
     motorsOff = !motorsOff;
     if (motorsOff) {
        analogWrite(speed1, 0);     // выключаем двигатели, устанавливая скорость на ноль
        analogWrite(speed2, 0);
     }
     delay(500);                    // даем время человеку отпустить кнопку,
                                      // и игнорировать любой дребезг
  }

  // сделать оба двигателя одинаковой скорости
  left = startSpeed;
  right = startSpeed;

  // считываем датчики и добавляем смещения
  LDR1 = analogRead(0) + leftOffset;
  LDR2 = analogRead(1);
  LDR3 = analogRead(2) + rightOffset;

  // если LDR1 больше, чем центральный датчик + порог повернуть направо
  if (LDR1 > (LDR2+threshhold)) {
    left = startSpeed + rotate;
    right = startSpeed - rotate;
  }
```

```
  если LDR3 больше, чем центральный датчик + порог, поверните налево
  if (LDR3 > (LDR2+threshhold)) {
    left = startSpeed - rotate;
    right = startSpeed + rotate;
  }

  // отправляем значения скорости двигателям
  if (!motorsOff) {
     analogWrite(speed1, left);
     analogWrite(speed2,right);
  }
}
```

Проложите курс для вашего робота на ровной поверхности. Я застилал пол на кухне линолеумом и сделал дорожку из черной изоленты. Включите робота, убедившись, что он не находится на дорожке. Будет запущена процедура калибровки, при которой светодиоды мигнут десять раз подряд. Как только это прекратится, у вас будет три секунды, чтобы поднять робота и поставить его на линию. Если все в порядке, робот теперь с радостью будет следовать по линии. Не делайте повороты слишком крутыми, так как этот быстрый переход не будет обнаружен тремя LDR. На рис. 10-12 показан пример курса, выполненного из черной изоленты. Код включен для дополнительной кнопки на контакте 2. Это позволит вам включать и выключать питание двигателей, чтобы вы могли настраивать значения в вашем коде для правильной работы робота.

Проект 30 - Робот, следящий за линией - Обзор кода

Сначала вы определяете контакт для источников света, а затем объявляете три целых числа, которые будут содержать значения, считанные с трех светорезисторов:

```
#define lights 9
int LDR1, LDR2, LDR3; // значения датчиков
```

Затем объявляются еще три целых числа, которые будут содержать значения смещения для трех датчиков, вычисленные в процедуре калибровки (это будет объяснено позже):

```
int leftOffset = 0, rightOffset = 0, center = 0;
```

Затем вы определяете пины, которые будут контролировать скорость и направление двух двигателей:

```
int speed1 = 3, speed2 = 11, direction1 = 12, direction2 = 13;
```

Затем создаются два целых числа для хранения начальной скорости двигателей и смещения скорости для каждого колеса, которое вы добавляете или вычитаете, чтобы заставить робота вращаться:

```
int startSpeed = 70, rotate = 30;
```

Скорость по умолчанию установлена на 70, что составляет около 27 процентов рабочего цикла. Возможно, вам придется отрегулировать это значение в соответствии с вашим собственным роботом. Слишком высокое значение приведет к тому, что робот выйдет за линию, а слишком низкое помешает двигателям вращаться достаточно быстро, чтобы повернуть колеса для преодоления трения.
Значение поворота - это то, насколько вы ускоряете или замедляете колеса, чтобы заставить робота повернуться. В моем случае необходимое значение - 30. Так, например, при повороте влево правое колесо вращается со скоростью 100, а левое колесо со скоростью 40 (70 + 30 и 70-30). Значение поворота - это еще один параметр, который вам может потребоваться настроить в соответствии с вашими настройками.

Another integer is created to hold the sensor threshold:

```
int threshhold = 5;
```

Это разница в значениях, которые требуются между центральным датчиком и левым или правым датчиками, прежде чем робот решит повернуть. В моем случае значение 5 работает хорошо. Это означает, что левый и правый датчики должны будут определить значение, превышающее значение, считываемое с центрального датчика, плюс пороговое значение, прежде чем будут предприняты действия. Другими словами, если центральный датчик считывает значение 600, а левый датчик считывает 603, то робот продолжит движение прямо. Однако значение левого датчика, равное612 (что выше, чем центральное значение плюс порог), означает, что левый датчик обнаруживает заднюю линию, указывая на то, что робот зашел слишком далеко влево. Таким образом, двигатели будут настраиваться, чтобы заставить робота повернуться вправо для компенсации.
Пороговое значение будет варьироваться в зависимости от контраста между вашим полом (или любой другой поверхностью, которую вы используете) и черной линией. Это может потребоватьсяотрегулировать, чтобы робот поворачивался только тогда, когда он обнаружил достаточно большую разницу между полом и линией, чтобы убедиться, что он переместился слишком далеко влево или вправо.Последний набор переменных сохранит значения скорости для левого и правого двигателей. Изначально для них установлено значение startSpeed:

```
int left = startSpeed, right = startSpeed;
```

Далее идут необязательные две строки кода, если вы включили кнопку в свою схему для входа в контакт 2. Если у вас нет кнопки в вашей цепи, измените motorsOff на false.

```
int button = 2; // вывод кнопки (необязательно)
int motorsOff = true;
```

После того, как все переменные объявлены и инициализированы, вы перейдете к своей первой и единственной функции - процедуре калибровки:

```
void calibrate() {
```

У этой процедуры двоякая цель. Во-первых, она получает среднее значение от каждого из трех датчиков за последовательность из десяти считываний. Во-вторых, она мигает 10 раз (при считывании значений с трех датчиков), чтобы показать вам, что робот калибруется и почти готов к работе. Датчики требуют калибровки, так как каждый из них будет считывать разные значения . Каждый LDR будет давать немного разные показания, и это будет зависеть от производственных допусков, допусков используемых резисторов делителя напряжения и сопротивления проводов. Вы хотите, чтобы все три датчика показывали (примерно) одно и то же значение, поэтому вы снимаете по десять показаний с каждого, усредняете их и вычисляете разницу между левым и правым датчиками от центрального датчика (который используется в качестве базовой линии).

Вы начинаете с создания цикла for, который выполняется десять раз:

```
for (int x=0; x<10; x++) {
```

Светодиоды, подключенные к выводу 9, включаются с задержкой в 100 миллисекунд:

```
digitalWrite(9, HIGH);
delay(100);
```

Затем значения со всех трех датчиков считываются и сохраняются в LDR1, LDR2 и LDR3:

```
LDR1 = analogRead(0);
LDR2 = analogRead(1);
LDR3 = analogRead(2);
```

Теперь вы берете эти значения и добавляете их к переменным leftOffset, center и rightOffset. Эти переменные начинаются с нуля, поэтому после десяти итераций они будут содержать промежуточную сумму всех десяти значений, считанных с датчиков.

```
leftOffset = leftOffset + LDR1;
center = center + LDR2;
rightOffset = rightOffset + LDR3;
```

Затем мы ждем 100 миллисекунд, выключаем свет, жде еще 100 миллисекунд, а затем повторяем процесс:

```
delay(100);
digitalWrite(9, LOW); // lights off
delay(100);
```

После того, как этот процесс повторится десять раз, мы выходим из цикла for и затем делим каждый промежуточный итог на десять.Это дает нам среднее показание датчика для каждого из трех LDR.

```
leftOffset = leftOffset / 10;
rightOffset = rightOffset / 10;
center = center /10;
```

Затем мы рассчитываем, каким будет смещение, вычитая значения левого и правого датчиков на центральный:

```
leftOffset = center - leftOffset;
rightOffset = center - rightOffset;
```

Эти значения будут добавлены к показаниям левого и правого датчиков, так что все три датчика будут давать примерно одинаковые показания. Это упростит определение разницы между тремя показаниями при обнаружении разницы между полом и черной линией.Затем у вас есть процедура настройки, которая начинается с установки контакта для кнопки, светодиодов и контактов для скорости и направления двигателя на ВЫХОД. Режим для кнопки - INPUT_PULLUP. Чип на Arduino имеет внутренние подтягивающие резисторы, которые вы можете использовать вместо внешних подтягивающих резисторов:

```
pinMode(button, INPUT_PULLUP);
pinMode(9, OUTPUT); // lights
pinMode(speed1, OUTPUT);
pinMode(speed2, OUTPUT);
pinMode(direction1, OUTPUT);
pinMode(direction2, OUTPUT);
```

Теперь выполняется процедура калибровки, чтобы гарантировать, что все три датчика выдают одинаковые показания, с задержкой в три секунды. После того, как светодиоды мигнут десять раз во время процедуры калибровки, у вас есть три секунды, чтобы поместить робота на линию, по которой он должен следовать:

```
calibrate();
delay(3000);
```

Включаются огни с небольшой задержкой:

```
digitalWrite(9, HIGH); // включенный свет
delay(100);
```

Направление обоих двигателей установлено вперед, а скорости установлены на значения, хранящиеся в переменных right и left (изначально установлены на значение, сохраненное в startSpeed):

```
digitalWrite(direction1, HIGH);
digitalWrite(direction2, HIGH);
analogWrite(speed1,left);
analogWrite(speed2,right);
```

Теперь перейдем к основному циклу программы. Он начинается с проверки, нажата ли дополнительная кнопка на контакте 2 включения или выключения двигателя соответственно. Кнопка позволит вам выключить робота, внести изменения в код и загрузить его.

```
if ( !digitalRead(button) ){
   // кнопка нажата. Включите или выключите моторы.
   motorsOff = !motorsOff;
   if (motorsOff) {
      analogWrite(speed1, 0);  // выключаем двигатели, устанавливая скорость на ноль
      analogWrite(speed2, 0);
   }
   delay(500);  // даем время человеку отпустить кнопку,
                    // и игнорировать любой ее дребезг .
}
```

Затем установим скорость левого и правого двигателей на значение startSpeed:

```
left = startSpeed;
right = startSpeed;
```

Скорость двигателя сбрасывается до этих значений в начале каждого цикла, так что робот всегда движется вперед, если значения не будут изменены позже в цикле, чтобы заставить робота повернуться.Теперь вы считываете значения датчиков из каждого из трех LDR и сохраняете значения в целых числах LDR1, LDR2 и LDR3. Смещения, рассчитанные для левого и правого датчиков, добавляются к значению, так что, когда все три датчика смотрят на одну и ту же поверхность, они будут показывать примерно одинаковые значения

```
LDR1 = analogRead(0) + leftOffset;
LDR2 = analogRead(1);
LDR3 = analogRead(2) + rightOffset;
```

Теперь вам нужно проверить значения этих датчиков и посмотреть, не сместилась ли черная линия слишком далеко от центра. Вы делаете это, проверяя левый и правый датчики и наблюдая, не превышают ли считанные значения значение, считываемое из центрального LDR, плюс смещение порога.

Если значение от левого датчика больше, чем смещение «показание плюс порог», значит, черная линия сместилась от центральной линии и находится слишком далеко вправо. Затем скорость двигателя регулируется так, чтобы левое колесо вращалось быстрее, чем правое, таким образом, поворачивая робота вправо, что вернет черную линию в центр.

```
if (LDR1 > (LDR2+threshhold)) {
    left = startSpeed + rotate;
    right = startSpeed - rotate;
  }
```

То же самое проделываем с правым датчиком, на этот раз поворачивая робота влево:

```
if (LDR3 > (LDR2+threshhold)) {
  left = startSpeed - rotate;
  right = startSpeed + rotate;
}
```

Скорости, которые могли или не могли быть отрегулированы, если линия была смещена от центра, затем отправляются на двигатели, если двигатель в настоящее время установлен на включение с помощью дополнительной кнопки:

```
if (!motorsOff) {
   analogWrite(speed1, left);
   analogWrite(speed2,right);

}
```

Весь этот процесс повторяется много раз в секунду и образует петлю обратной связи, которая заставляет робота следовать линии.Если ваш робот сбегает с линии, возможно, вам придется изменить скорость в startSpeed на более низкую. Если он не поворачивается быстро или недостаточно далеко, или если он поворачивается слишком сильно, то значение поворота необходимо соответствующим образом отрегулировать. Чувствительность LDR можно отрегулировать, изменив значение в пороговой переменной. Поиграйте с этими значениями, пока вы не заставите робота успешно следовать по линии.

Резюме

Глава 10 началась с простого введения в шаговые двигатели и способы управления их скоростью и направлением с помощью ИС привода двигателя. Вы также узнали о разнице между биполярным и униполярным шаговыми двигателями и о том, как они работают. Затем вы перешли к использованию простого моторного щита для независимого управления двумя двигателями и в итоге подключили этот щит к базе робота, чтобы создать двухколесного робота, следующего за линией.Вы также узнали о щитах и их различных применениях, а также нашли практическое применение в работе двигателя. Знание того, как управлять шаговыми двигателями и стандартными двигателями, а также сервоприводами (глава 9), означает, что вы на правильном пути к созданию собственного продвинутого робота или устройства аниматроники!

Предметы и понятия, затронутые в главе 10, включают:

- Разница между униполярными и биполярными шаговыми двигателями
- Как подключить шаговый двигатель к микросхеме драйвера двигателя
- Использование конденсаторов для сглаживания сигнала и уменьшения помех
- Как настроить шаговый объект в вашем коде
- Установка скорости и количества шагов двигателя
- Управление направлением двигателя с помощью положительных или отрицательных чисел
- Как работает шаговый двигатель
- Последовательность включения катушки, необходимая для привода шагового двигателя
- Это разрешение можно увеличить с помощью полушага и микрошага
- Как пользоваться моторным шилдом

- Использование моторного щита для управления скоростью и / или направлением нескольких двигателей
- Что такое шилд Arduino и какие типы доступны
- Как обнаружить вариации контрастности света с помощью нескольких LDR
- Как получить среднее показание датчика
- Использование значений смещения для балансировки показаний датчика
- Как сделать классного робота, следующего за линией

ГЛАВА 11

Датчики давления

Теперь вы познакомитесь с датчиками давления, в частности, с цифровым датчиком давления MPL3115A2 от Freescale. Это отличный датчик, который легко подключается к Arduino и обеспечивает точные показания давления и температуры. Устройство должно быть подключено к шине I2C Arduino для обмена данными. I2C - хотя в этой главе она не рассматривается подробно, вы узнаете основные понятия I2C и способы его использования для получения данных с датчика MPL3115A2.
Цифровые датчики давления идеально подходят для создания собственной метеостанции. В этой главе вы начнете с изучения того, как подключить MPL3115A2 к Arduino и считывать с него данные с помощью монитора последовательного порта. Затем вы будете использовать датчик и ЖК-дисплей для отображения графика давления за 24-часовой период. Попутно вы также узнаете, как управлять дисплеем GLCD (графический ЖК-дисплей).

Проект 31 - Цифровой датчик давления

В этом проекте мы узнаем, как использовать датчик давления MPL3115A2. Затем, в Проекте 32, мы будем использовать наши знания о том, как использовать этот датчик, чтобы создать график изменения давления во времени на графическом ЖК-дисплее.

Требуемые детали

Крошечный датчик MPL3115A2 можно приобрести у Sparkfun или у их дистрибьюторов, предварительно припаяв к коммутационной плате (см. Рис. 11-1), чтобы упростить взаимодействие с Arduino (или другим микроконтроллером),несколько выводов разъема к плате, если вы хотите вставить ее в макетную плату.
В противном случае припаяйте к нему несколько проводов, чтобы его можно было подключить к Arduino.

Рис. 11-1. MPL3115A2 на коммутационной плате Sparkfun

Таблица 11-1. Детали, необходимые для проекта 32

Arduino Mega или Uno	
Датчик давления MPL3115A2	

MPL3115A2 - отличный сенсор. Он может рассчитывать высоту с точностью до 30 см (1 фут), а также имеет на борту 12-битный датчик температуры.

Подключение

Подключите все, как показано на Рисунке 11-2. Я использовал Arduino Mega вместо Uno для этих проектов, так как у него есть дополнительные контакты для добавления дополнительных устройств.

Если вы действительно используете Arduino Uno для своего проекта, сигналы SDA и SCL находятся на 4-м и 5-м аналоговых входах соответственно в отличие от Arduino Mega, в котором вместо этого используются контакты 20 и 21.

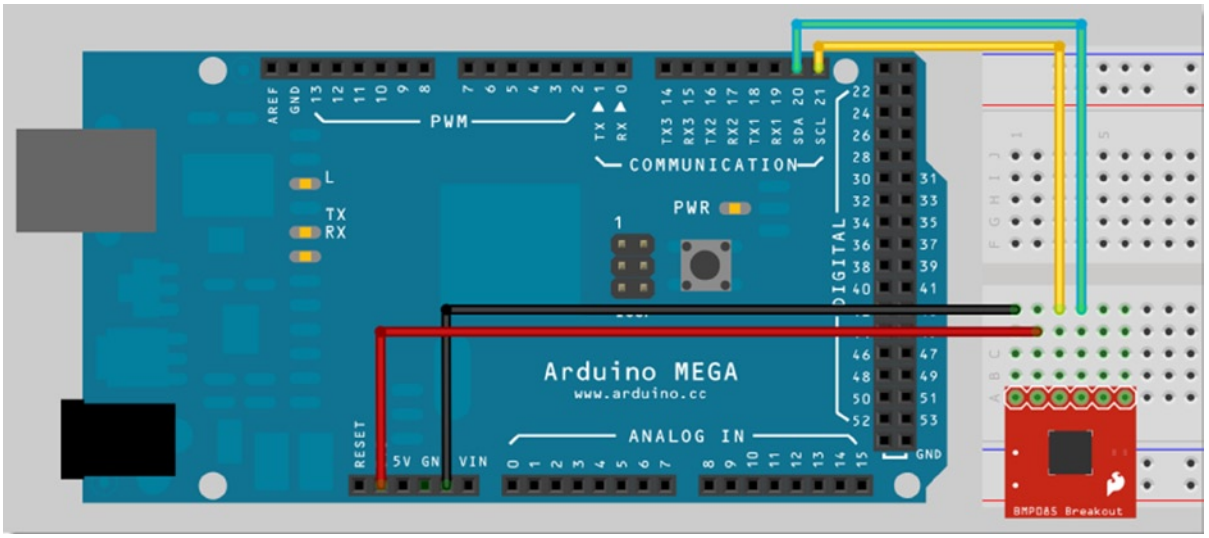


Рис. 11-2. Схема для проекта 31 - Цифровой датчик давления

Для этого проекта просто требуется Uno, MPL3115A2 на коммутационной плате и несколько перемычек.

Подключите контакт GND на датчике к контакту GND на Arduino. Подключите вывод VCC на датчике квыводу 3,3 В на Uno. Затем подключите выводы SDA (последовательные данные) и SCL (синхросигналы) на Uno к соответствующим выводам SDA и SCL на датчике. Имейте в виду, что библиотека включаетвнутренниеподтяжки до 5 В, а MPL3115A2 требует максимального напряжения 3,6 В на выводах SDA и SCL. На коммутационной плате Sparkfun есть подтягивающие резисторы, которые снижают напряжение до безопасного уровня 3,3 В. Если MPL3115A2 - единственное устройство на шине I2C, подтягивание до 3,3 В на коммутационной плате будет поддерживать уровни сигнала в безопасном диапазоне.

. Если вы добавите к шине I2C какие-либо другие устройства, которые работают от 5 В и имеют встроенные подтягивающие устройства, такие как коммутационная плата MPL3115A2, вы можете легко повредить MPL3115A2 и любые другие устройства I2C, работающие только на 3,3 В на шине. Поэтому не добавляйте в этот проект какие-либо другие устройства I2C без обеспечения некоторой защиты. Один из способов избежать этих проблем -зафиксировать линии SCL и SDA, используя стабилитрон малой на 3,6 В. Другой способ - использовать светодиоды с прямым напряжением 3,1–3,4В . (Большинство синих и белых светодиодов имеют прямое напряжение в этом диапазоне.)

Введите код

Введите код из Листинга 11-1.

Листинг 11-1. Код для проекта 31

```
// На основе примера One Shot Генри Лара
#include <Wire.h> // чтобы мы могли использовать I2C
#define MYALTITUDE 262 //определение высоты в вашем местоположении для расчета среднего давления на уровне моря в метрах
// Регистрируем адреса
const int SENSORADDRESS = 0x60; // Адрес MPL3115A1 из даташита
#define SENSOR_CONTROL_REG_1 0x26
#define SENSOR_DR_STATUS 0x00  // Адрес регистра состояния DataReady
#define SENSOR_OUT_P_MSB 0x01  // Начальный адрес регистров данных давления

float baroAltitudeCorrectionFactor = 1/(pow(1-MYALTITUDE/44330.77,5.255877));

byte I2Cdata[5] = {0,0,0,0,0}; //буфер для данных датчика

void setup(){
  Wire.begin(); // присоединяемся к шине i2c
  Serial.begin(9600); // запускаем последовательный порт для вывода со скоростью 9600 бод
  Serial.println("Setup");
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00000000); // переводим в режим ожидания
  // эти старшие биты регистра управления
  // можно изменить только в режиме ожидания
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00111000); // устанавливаем передискретизацию на 128
  Serial.println("Done.");
}

void loop(){
  float temperature, pressure, baroPressure;

  Read_Sensor_Data();
  temperature = Calc_Temperature();
  pressure = Calc_Pressure();
  baroPressure = pressure * baroAltitudeCorrectionFactor;
  Serial.print("Absolute pressure: ");
  Serial.print(pressure); //в Паскалях
  Serial.print(" Pa,    Barometer: ");
  Serial.print(baroPressure); // в Паскалях
  Serial.print(" Pa,    Temperature: ");
```

```
  Serial.print(temperature); // в градусах C
  Serial.println(" C");
  delay(1000);
}

  // Считываем показания давления и температуры с датчика
void Read_Sensor_Data(){

  // запрашиваем одно измерение от датчика
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00111010); //бит 1 - одноразовый режим

  // Дождемся завершения измерения.
  // Одноразовый бит очищается, когда это будет сделано.
  // Перечитываем текущий ( с управляемого датчика ) регистр
  // повторяем, пока датчик не сбросит бит OST
  do {
    Wire.requestFrom(SENSORADDRESS,1);
  } while ((Wire.read() & 0b00000010) != 0);

  I2C_ReadData(); //reads registers from the sensor
}
  // Эта функция собирает показания давления
  // из значений в буфере чтения
  // Два младших бита являются дробными, поэтому разделите их на 4
float Calc_Pressure(){
  unsigned long m_pressure = I2Cdata[0];
  unsigned long c_pressure = I2Cdata[1];
  float l_pressure = (float)(I2Cdata[2]>>4)/4;
  return((float)(m_pressure<<10 | c_pressure<<2)+l_pressure);
}

// Эта функция собирает показания температуры
// из значений в буфере чтения
float Calc_Temperature(){
  int m_temp;
  float l_temp;
  m_temp = I2Cdata[3]; //temperature in whole degrees C
  l_temp = (float)(I2Cdata[4]>>4)/16.0; //fractional portion of temperature
  return((float)(m_temp + l_temp));
}

// Считываем данные барометра и температуры (5 байтов)
void I2C_ReadData(){
  byte readUnsuccessful;
  do {
    byte i=0;
    byte dataStatus = 0;

    Wire.beginTransmission(SENSORADDRESS);
    Wire.write(SENSOR_OUT_P_MSB);
    Wire.endTransmission(false);
```

```
    // читаем 5 байтов. 3 для давления, 2 для температуры.
    Wire.requestFrom(SENSORADDRESS,5);
    while(Wire.available()) I2Cdata[i++] = Wire.read();

    // в некоторых режимах это возможно для датчика
    // обновить показания давления
    // пока мы читали его,
    // в этом случае наша копия - мусор
    // (части двух разных чтений)
    // Мы можем проверить биты в DR (данные готовы)//
    регистрируемся, чтобы узнать, произошло ли это.
    Wire.beginTransmission(SENSORADDRESS);
    Wire.write(SENSOR_DR_STATUS);
    Wire.endTransmission(false);

    Wire.requestFrom(SENSORADDRESS,1); //читаем 5 байтов. 3 для давления, 2 для температуры.
    dataStatus = Wire.read();
    readUnsuccessful = (dataStatus & 0x60) != 0;
    // This will be unsuccessful if overwrite happened
    // while we were reading the pressure or temp data.
    // So keep reading until we get a successful clean read
  } while (readUnsuccessful);
}

// Эта функция записывает один байт по I2C
void I2C_Write(byte regAddr, byte value){
  Wire.beginTransmission(SENSORADDRESS);
  Wire.write(regAddr);
  Wire.write(value);
  Wire.endTransmission(true);
}
```

После того, как вы загрузили код, откройте окно последовательного монитора и убедитесь, что ваша скорость передачи установлена на 9600. Вы увидите поток данных от датчика, показывающий давление в Па (паскалях) и температуру в градусах Цельсия. Паскали - это единица измерения давления, обычно используемая в прогнозах погоды.

Проект 31 - Цифровой датчик давления - Обзор кода

Сначала мы включаем библиотеку. Эта библиотека позволяет нам общаться с устройствами I2C / TWI.

```
#include <Wire.h>
```

Затем мы определяем MYALTITUDE как нашу высоту в метрах. Вам нужно будет узнать это с помощью GPS-навигатора или на местной метеостанции. Моя высота 262 метра, но измените ее в соответствии с вашим местоположением.

```
#define MYALTITUDE 262
```

Согласно таблице данных для MPL3115A2, адрес устройства I2C - 0x60 или 60 в шестнадцатеричной системе счисления, а в десятичной - 196. У каждого устройства I2C будет свой адрес.

```
const int SENSORADDRESS = 0x60;
```

Затем нам нужно определить три регистра в датчике, которые мы рассмотрим позже в коде.

```
#define SENSOR_CONTROL_REG_1 0x26
#define SENSOR_DR_STATUS 0x00
#define SENSOR_OUT_P_MSB 0x01
```

Чтобы иметь возможность рассчитать точные показания давления, откалиброванные для вашей высоты, нам нужнобудет сгенерировать поправочный коэффициент, который будет использоваться позже в коде для калибровкипоказаний. Итак, переменная с именем baroAltitudeCorrectionfactor создается здесь с использованием вашей высоты для расчета поправочного коэффициента. Чтобы рассчитать давление на уровне моря на основе нашей высоты, мы используем расчет в таблице данных.

$$P = P_0 \cdot \left(1 - \frac{h}{44330.77}\right)^{5.255876}$$

Где P0 - это среднее давление на уровне моря 101325 Па (Паскали), а h - наша высота в метрах. В коде это переводится на

```
float baroAltitudeCorrectionFactor = 1/(pow(1-MYALTITUDE/44330.77,5.255877));
```

Данные датчика давления занимают три байта, а температура - два байта. Поэтому мы создаем массив из пяти байтов для хранения показаний датчика давления и температуры.

```
byte I2Cdata[5] = {0,0,0,0,0};
```

Далее мы переходим в функцию настройки и начинаем общаться с датчиком. Однако, прежде чем мы это сделаем, давайте кратко рассмотрим шину I2C и как она работает.

I2C шина

I2C (иногда называемый I-квадрат-C) - это последовательная шина, изобретенная Philips Semiconductors и используемая для подключения низкоскоростных периферийных устройств к электроннымустройствам. I2C использует две двунаправленные линии, линию последовательных данных (SDA) ипоследовательные синхросигналы (SCL) с резисторами. Шина I2C имеет 7-битное или 10-битное (в зависимости от используемого устройства) адресное пространство. Устройства на шине I2C состоят из главного устройства и одного или нескольких подчиненных устройств. Все они подключены к тем же линиям данных и SCL, что и на рисунке 11-3.

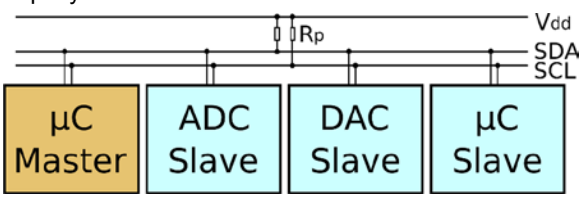


Рис. 11-3. Пример схемы I2C

Главный узел - это устройство, которое генерирует тактовый сигнал и инициирует связь с подчиненными устройствами. Подчиненный узел - это устройство, которое принимает тактовый сигнал и отвечает при обращении к ведущему. Возможно иметь несколько главных устройств. Кроме того, ведущее и ведомое устройства могут меняться ролями между сообщениями (после отправки команды STOP). Подчиненные устройства имеют 7-битный адрес, и каждое из них будет отличаться.

Внутри каждого устройства I2C есть регистры, которые можно читать или записывать, чтобы управлять устройством или изменять его конфигурацию. Представьте, что адрес устройства - это номер дома, а внутренние регистры - это номера квартир. Для чтения или записи в регистр вам понадобится адрес устройства (номер здания) и егорегистрационный адрес (номер квартиры). Далее идет функция setup ().

```
void setup(){
```

Теперь нам нужно инициировать связь с устройством I2C. Мы используем команду Wire.Begin (), чтобы присоединить ее к шине I2C. Устройства I2C могут быть ведущими или ведомыми. Ведущее устройство генерирует тактовый сигнал и обменивается данными с ведомыми устройствами. Подчиненное устройство получает тактовый сигнал от ведущего и отвечает на запросы ведущего. В нашем случае Arduino является ведущим, а MPL3115A2 - ведомым.

```
Wire.begin();
```

Мы будем использовать окно МПП для просмотра данных, выводимых с датчика, поэтому мы начнем последовательную связь со скоростью 9600 бод.

```
Serial.begin(9600);
```

Сообщить пользователю через МПП, что программа запущена и устройство настраивается.

```
Serial.println("Setup");
```

Затем нам нужно установить датчик в режим ожидания и установить частоту передискретизации 128x.

```
I2C_Write(SENSOR_CONTROL_REG_1, 0b00000000);
I2C_Write(SENSOR_CONTROL_REG_1, 0b00111000);
```

Датчик может либо выдавать нам показания данных, когда мы его запрашиваем, либо быть настроен на вывод данных с заданным интервалом от 1 секунды до 215 секунд (равный 32 768 секундам или чуть более 9 часов). Для наших целей легче читать данные, когда они нам нужны, поэтому мы будем использовать режим OST. Передискретизация - это процесс многократной выборки сигнала с последующим получением среднего значения этих сигналов. Это помогает снизить отношение сигнал / шум устройства и дает более точные показания. 128-кратная передискретизация - это максимум, который позволяет устройство, поэтому мы воспользуемся этим. Это означает, что снимаются и усредняются 128 показаний. Затем устройство дает нам это усредненное значение.Адрес 0x26 (SENSOR_CONTROL_REG_1) - это управляющий регистр 1 (CTRL-REG1) внутри устройства. Это просто адресное пространство, которое позволяет нам изменять настройки датчика, изменяя значение, хранящееся по этому адресу. Каждый отдельный бит CTRL-REG1 имеет различное назначение. В соответствии с таблицей данных биты показаны на рис. 12-4.

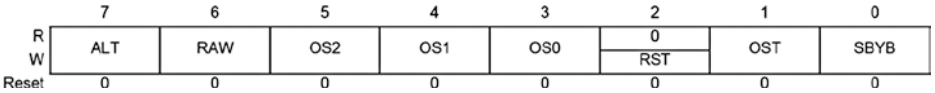


Рис. 12-4. *Отдельные биты CTRL-REG1*

Первый бит - это бит ожидания (SBYB). Если установлено значение 0, устройство переходит в режим ожидания. Если установлено значение 1, он находится в АКТИВНОМ режиме. Если вы посмотрите на код, первое значение, которое мы записываем в CTRL-REG1, имеет бит SBYB (крайний правый бит), установленный в 0, и, следовательно, мы переводим устройство в режим ожидания. Следующий бит - это бит режима One Shot (OST). Опять же, первое значение, которое мы отправляем в регистр, установит этот бит в 0. Это очищает бит режима One Shot Mode. Бит OST должен быть установлен в 0, чтобы очистить его, чтобы в следующий раз мы могли выполнить еще одно чтение. Второе значение, которое мы запишем в CTRL_REG1, очищает бит OST (устанавливает его в 0), как вы можете видеть ниже.

Следующий бит - это бит RST или программного сброса. Мы не используем его, поэтому оставьте его равным 0. Следующие 3 бита - это 3-битное число, которое устанавливаетчастота передискретизации. Частота передискретизации изменяется от 1 при установке на000 бит через 2, 4, 8, 16, 32, 64 и, наконец, 128, когда установлено значение 111. Следующие два бита не используются и поэтому остаются равными 0.Итак, с информацией, которая у нас есть выше, мы можем видеть, что сначала мы записываем значение в CTRL-REG1, которое устанавливает датчик в режим ожидания, а также очищает режим One Shot с частотой передискретизации, установленной на 128x.Затем мы уведомляем пользователя о завершении настройки устройства.

```
Serial.println("Done.");
```

Затем идет функция основного цикла нашего скетча, которая просто считывает с устройства три байта, составляющие показание давления, плюс два байта для показания температуры. Он преобразует эти байты в числа с плавающей запятой, а затем отображает значения с плавающей запятой в окне МПП. Сначала мы создаем три переменные для хранения температуры, давления и baroPressure (давление с поправкой на высоту).

```
float temperature, pressure, baroPressure;
```

Затем мы вызываем функцию Read_Sensor_Data (), чтобы получить показания с устройства.

```
Read_Sensor_Data();
```

Затем вызываем еще две функции, которые преобразуют три бита, в которых хранится давление, в десятичное число и два байта, составляющие показание температуры.

```
temperature = Calc_Temperature();
pressure = Calc_Pressure();
```

Затем мы снимаем показания давления, корректируем их в соответствии с нашей высотой и сохраняем их в переменной baroPressure.

```
baroPressure = pressure * baroAltitudeCorrectionFactor;
```

Затем мы используем серию команд Serial.print для отображения абсолютного давления, настроенного давления и температуры.

```
Serial.print("Absolute pressure: ");
Serial.print(pressure); //  в Паскале
Serial.print(" Pa,    Barometer: ");
Serial.print(baroPressure); //в Паскале
Serial.print(" Pa,    Temperature: ");

Serial.print(temperature); // в градусах C
Serial.println(" C");
```

Чтобы не перегружать монитор последовательного порта (МПП) показаниями, мы снимаем показания и отображаем их каждую секунду, а затем выполняем задержку в 1000 мс.

```
delay(1000);
```

Мы объявляем нашу следующую функцию для чтения данных датчика.

```
void Read_Sensor_Data(){
```

Затем мы устанавливаем бит 1 (режим One Shot Mode) на 1, который будет запрашивать одно показание давления и температуры от датчика.

```
I2C_Write(SENSOR_CONTROL_REG_1, 0b00111010);
```

Затем мы используем цикл do-while для запроса одного байта от устройства, используя функцию Wire.requestFrom (), предоставляющую адрес устройства и количество байтов для извлечения.

```
do {
  Wire.requestFrom(SENSORADDRESS,1);
```

Цикл будет продолжать читать байт, пока есть доступные данные и пока устройство не сбросит бит OST.

```
} while ((Wire.read() & 0b00000010) != 0);
```

Вышеупомянутое заполнит пять регистров внутри устройства, которые хранят давление (три байта) и температуру(два байта) текущими показаниями датчика. Теперь мы вызываем функцию I2C_ReadData (), которая считывает эти пять байтов с устройства и сохраняет их в нашем массиве I2Cdata []

```
I2C_ReadData(); //считывает регистры с датчика
```

Теперь давайте посмотрим на три функции, вызываемые функцией Read-Sensor_Data. Первая функция - это та, которая берет три байта, составляющие показание давления, и преобразует его в число с плавающей запятой.

```
float Calc_Pressure(){
```

Целочисленная (целая) часть показания давления хранится в 18 битах, причем два бита представляют собой дробную часть показания давления. Первые 16 битов хранятся в MSB и CSB (I2Cdata [0] и [1]). Затем биты 7 и 6 LSB (I2Cdata [2]) составляют последние два бита 18-битного числа. Биты 5 и 4 младшего разряда составляют дробную часть показания давления.

Старший байт показания давления хранится в первом элементе массива I2Cdata, поэтому мы сохраняем его в переменной m_pressure.

```
unsigned long m_pressure = I2Cdata[0];
```

Центральный байт из трех байтов, составляющих показание давления, хранится во втором элементе массива данных I2C, и мы сохраняем его в переменной c_pressure.

```
unsigned long c_pressure = I2Cdata[1];
```

Затем мы берем последние два бита числа, сдвигая младший бит на четыре разряда вправо, оставляя нам два бита, которые составляют последний из 18 бит целой части числа, и два бита, которые составляют дробная часть. Поскольку два бита, составляющие дробную часть, могут хранить только четыре числа (каждый бит равен 1/4 или 0,25 Паскаля), это наше разрешение. Разделите это на 4.

```
float l_pressure = (float)(I2Cdata[2]>>4)/4;
```

Наконец, мы берем все три байта и объединяем их, чтобы получить окончательное значение давления. Мы делаем это, создавая побитовое ИЛИ для цифр в MSB и CSB, а затем добавляя к нему цифры в LSB, чтобы получить окончательное значение давления. Цифры в m_pressure сдвинуты влево на 10 разрядов, а в c_pressure - на два разряда.

Поскольку целая часть показания давления занимает 18 бит, это перемещает биты в их правильное положение, оставляя место для добавления двух последних битов, составляющих оставшуюся часть целочисленной части, поскольку мы привели число к float, а также два бита, составляющие дробную часть числа. Затем мы возвращаем это число как давление из нашей функции.

```
return((float)(m_pressure<<10 | c_pressure<<2)+l_pressure);
```

Следующая функция делает то же самое, что и выше, но для двух байтов, составляющих показание температуры.

```
float Calc_Temperature(){
  int m_temp;
  float l_temp;
  m_temp = I2Cdata[3]; //temperature in whole degrees C
  l_temp = (float)(I2Cdata[4]>>4)/16.0; //fractional portion of temperature
  return((float)(m_temp + l_temp));
}
```

Следующая функция просто считывает с датчика пять байтов, составляющих показание давления (три байта) и показание температуры (два байта), и сохраняет их в массиве I2Cdata [].

```
void I2C_ReadData(){
```

Сначала мы создаем переменную типа byte, которая будет использоваться для чтения регистра dataStatus, чтобы определить, закончили ли мы чтение байтов с датчика или нет.

```
byte readUnsuccessful;
```

Остальная часть функции находится внутри цикла do-while, который будет повторяться, пока данные могут быть успешно прочитаны с устройства.

```
do {
```

Затем мы создаем индекс для нашего массива. Начнем с элемента 0.

```
byte i=0;
```

Затем создается переменная типа byte с именем dataStatus. Это будет использоваться позже, чтобы определить, были ли данные успешно прочитаны с устройства или нет.

```
byte dataStatus = 0;
```

Затем мы начинаем передачу на ведомое устройство I2C с помощью функции beginTransmission в библиотеке Wire.Параметр функции - это адрес устройства, который хранится в SENSORADDRESS.

```
Wire.beginTransmission(SENSORADDRESS);
```

Следующая функция ставит в очередь байты от ведомого устройства, начиная с адреса, указанного в параметре. В нашем случае первый байт данных давления начинается с адреса 0x01 (SENSOR_OUT_P_MSB), поэтому мы говорим устройству начать помещать байты в очередь с этого адреса, готовые к чтению.

```
Wire.write(SENSOR_OUT_P_MSB);
```

Затем мы используем функцию endTransmission, чтобы завершить передачу на ведомое устройство, начатую функцией beginTransmission, и передать байты, поставленные в очередь функцией write ().
Функция endTransmission принимает логический аргумент true или false. Если true, endTransmission () отправляет сообщение остановки после передачи, освобождая шину I2C. Если false, endTransmission () отправляет сообщение о перезапуске после передачи. Шина не будет освобождена, что препятствует передаче другим ведущим устройством между сообщениями. Это позволяет одному ведущему устройству отправлять несколько передач, находясь под управлением.

```
Wire.endTransmission(false);
```

Функция requestFrom () используется ведущим устройством для запроса байтов от ведомого устройства. Затем байты извлекаются с помощью функций available () и read (). Мы передаем функции два аргумента, первый из которых является адресом устройства, а второй - количеством запрошенных байтов. Нам нужны три байта для чтения давления итри байта для показаний температуры, которые начинаются с адреса 0x01 в устройстве и поэтому запрашивают пять байтов.

```
Wire.requestFrom(SENSORADDRESS,5);
```

Затем мы используем оператор while, чтобы определить, есть ли байты, доступные для чтения с помощью функции available (). Эта функция возвращает количество байтов, доступных для чтения. В нашем случае это число будет пять, так как это количество байтов, которое мы запросили у устройства. Итак, если возвращаемое значение больше нуля, команда после функции available () будет запущена. Внашем случае мы используем функцию read () для чтения одного байта с устройства и сохранения его в массиве I2Cdata []. Значение в i одновременно увеличивается с i ++. Пока данные доступны для чтения, оператор while будет продолжать читать по байтам и сохранять их в массиве I2Cdata []. Через пять итераций функция available () вернет 0, поскольку мы запросили только пять байтов. Затем программа выйдет из цикла while.

```
while(Wire.available()) I2Cdata[i++] = Wire.read();
```

В некоторых режимах датчик может обновлять показания давления в середине считывания, в результате чего получается мусорная копия (части двух разных показаний). Следовательно, мы можем проверить биты в регистре DR (данные готовы), чтобы узнать, произошло ли это. Сначала мы запрашиваем один байт из регистра DR.

```
Wire.beginTransmission(SENSORADDRESS);
Wire.write(SENSOR_DR_STATUS); // Адрес регистра состояния DataReady в датчике
Wire.endTransmission(false);
```

Затем сохраните его в dataStatus ()

```
Wire.requestFrom(SENSORADDRESS,1);
dataStatus = Wire.read();
```

Теперь нам нужно проверить, возвращается ли статус данных как успешный или нет. Биты 5 и 6 регистра DR_STATUS устанавливаются в 1 всякий раз, когда собираются новые данные до завершения поиска предыдущего набора. Таким образом, если байт, возвращаемый из регистра DR_STATUS, логически соединен операцией AND с 0x60, результатом будет ненулевое значение, если чтение было неудачным. Мы сохраняем этот результат как readUnsuccessful.

```
    readUnsuccessful = (dataStatus & 0x60) != 0;
```

Цикл while будет выполняться до тех пор, пока мы не получим чистое чтение.

```
    } while (readUnsuccessful);   // продолжаем читать, пока не получим успешное чистое чтение
}
```

Следующая функция позволяет нам записать один байт в один адрес внутри устройства. Мы использовали это ранее в программе, когда записывали наши настройки в CTRL_REG1, а также данные калибровки в устройство. Функция принимает два аргумента: адрес регистра, в который мы хотим записать, и значение, которое мы хотим записать по этому адресу.

```
void I2C_Write(byte regAddr, byte value){
```

Снова начинаем передачу на устройство с его адреса.

```
Wire.beginTransmission(SENSORADDRESS);
```

Затем мы передаем адрес, на который хотим записать, а затем значение, которое мы хотим записать на этот адрес, с помощью функции write ().

```
Wire.write(regAddr);
Wire.write(value);
```

Наконец, мы завершаем передачу и на этот раз передаем истинный аргумент функции, чтобы отправить сообщение остановки и освободить шину I2C.

```
Wire.endTransmission(true);
```

Теперь вы знаете основы взаимодействия с устройством I2C с помощью библиотеки Wire. Однако некоторые датчики давления не используют шину I2C, а вместо этого используют SPI. SPI означает последовательный интерфейс периферийных устройств. На всякий случай, если вы встретите аналогичный датчик давления, который использует SPI вместо I2C, позвольте нам сделать небольшой обходной путь, чтобы взглянуть на SPI и на то, как он работает.

SPI – Последовательный интерфейс периферийных устройств

SPI может быть сложной темой и намного сложнее, чем использование гораздо более простой шины I2C, поэтому я не собираюсь вдаваться в подробности. В конце концов, это книга для начинающих. Вместо этого я собираюсь дать вам достаточно знаний, чтобы вы поняли, что такое SPI, как он работает и что может быть актуально для вас, если вы будете использовать датчик SPI вместо датчика с шиной I2C. После этого вы сможете понять, как взаимодействовать с другими устройствами, которые также используют SPI.

SPI - это способ обмена информацией между двумя устройствами. У него есть преимущества в том, что для него требуется всего четыре контакта от Arduino, и он быстрый. SPI - это синхронный протокол, который позволяет главному устройству обмениваться данными с подчиненнымустройством. Данные куправляются тактовым сигналом (CLK), который определяет, когда данные могутизмениться и когда они пригодны для чтения. Тактовая частота может меняться, в отличие от некоторыхдругих протоколов, в которых тактовый сигнал должен быть синхронизирован очень точно.Это делает его идеальным для микроконтроллеров, которые не работают особенно быстро или чей внутренний генератор не работает точно.

SPI - это протокол ведущий-ведомый (см. Рисунок 11-5), что означает, что ведущее устройство управляет тактовым сигналом. Никакие данные не могут быть переданы, если такты не работают в импульсном режиме. SPI также является протоколом обмена данными. Это означает, что по мере того, как данные синхронизируются, синхронизируются новые данные.

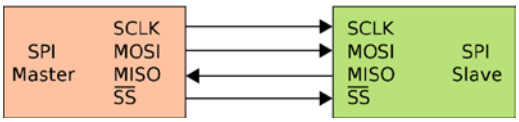


Рис. 11-5. *Шина SPU: один ведущий и один ведомый*

Контакт выбора подчиненного устройства будет определять, когда к устройству можно получить доступ, если к главному устройству подключено более одного подчиненного устройства (см. Рисунок 11-6). Когда есть только одно подчиненное устройство, SS (иногда обозначаемый как CSB на некоторых устройствах) является необязательным. Однако, как правило, его следует использовать независимо, поскольку он также используется как сброс для ведомого устройства, чтобы он был готов к приему следующего байта. Сигнал выбора ведомого отправляется ведущим, чтобы сообщить ведомому, что он желает начать обмен данными SPI.
Этот сигнал активен, когда НИЗКИЙ, поэтому, когда удерживается ВЫСОКИЙ, подчиненное устройство не выбирается.

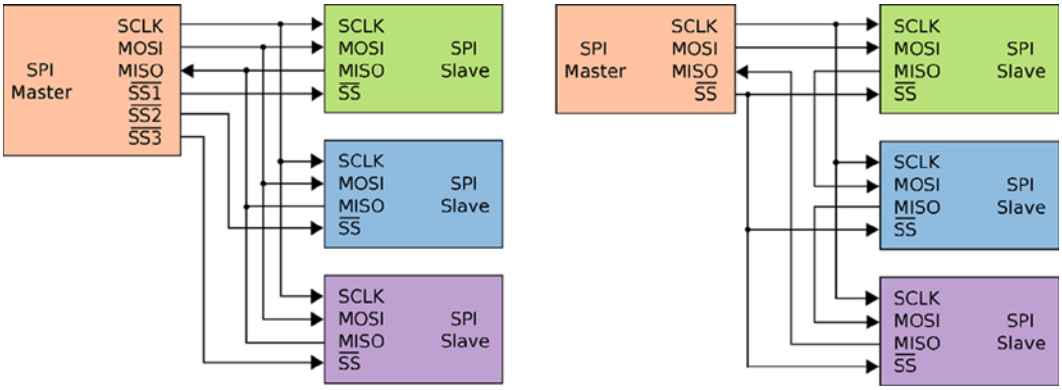


***Рис. 11-6.** Слева: мастер с тремя независимыми подчиненными. Справа - с зависимыми*

Данные выводятся только во время нарастания или спада тактового сигнала на SCK. Данные фиксируются на противоположном фронте SCK. Полярность тактового сигнала устанавливается мастером с помощью одного из флагов, установленных в регистре SPCR.
Две линии данных известны как MOSI (главный выход, подчиненный вход) и MISO (главный вход, подчиненный выход). Таким образом, если устройство настроено на отправку данных от ведущего по переднему фронту тактового импульса, данные будут отправлены обратно от ведомого по заднему фронту тактового импульса. Таким образом, данные отправляются (MOSI) и вводятся (MISO) от ведущего устройства в течение одного тактового импульса.
Помните, что даже если вы хотите только читать данные с устройства (как вы это делаете в большинстве случаев с датчиком давления), вам все равно нужно отправлять данные в обоих направлениях во время одного обмена.

Шина SPI использует три регистра:

- SPCR – регистр управления SPI
- SPDR – регистр данных SPI
- SPSR – регистр статуса SPI

Регистр управления имеет восемь битов, и каждый бит управляет определенной настройкой SPI. Эти биты перечислены в Таб. 11-2.

***Таб. 11-2.** Настройки регистра управления SPI*

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

- SPIE – Разрешение прерывания SPI - включает прерывание SPI, если 1.
- SPE – SPI Enable - SPI включен, если установлено значение 1.
- DORD – Порядок данных - LSB передается первым, если 1, и MSB, если 0
- MSTR – Master / Slave Select - устанавливает Arduino в главный режим, если 1, в подчиненный режим, когда 0.

- CPOL – Clock Polarity – Бит CPOL устанавливает полярность тактового сигнала в состоянии ожидания.
- CPHA – Clock Phase – Определяет, выбираются ли данные по переднему или заднему фронту тактового сигнала.
- SPR1/0 – Выбор тактовой частоты SPI 1 и 0 - Эти два бита управляют тактовой частотой ведущего устройства

Причина, по которой вы можете изменить эти настройки, заключается в том, что разные устройства SPI ожидают, что полярность тактового сигнала, фаза тактового сигнала, порядок данных, скорость и т. Д. Будут разными. В основном это связано с тем, что не существует стандартных производителей, создающих устройства с незначительными отличиями. В коде вы можете установить SPCR следующим образом:

```
SPCR = B01010011;
```

Проект 32 - Цифровой барограф

Теперь, когда вы можете подключить датчик давления MPL3115A2 и получать от него данные, вы собираетесь найти ему какое-то применение. Этот проект заключается в создании цифрового барографа, который представляет собой график изменения давления во времени. Для отображения графика мы собираемся использовать GLCD (графический ЖК-дисплей). Вы узнаете, как отображать не только текст, но и графику.

Требуемые детали

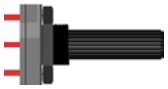
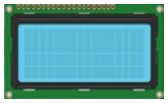
Список деталей идентичен Project 31, с добавлением 128 x 64 GLCD, дополнительного резистора и потенциометра. Мы используем библиотеку GLCD, поэтому GLCD должен иметь микросхему драйвера KS0108 (или эквивалентную). Перед покупкой проверьте техническое описание.

Чтобы библиотека GLCD работала с Uno:
In glcd/config/ks0108_Arduino.h

```
Change #define glcdEN   18
To     #define glcdEN   12
```

Эта модификация указывает библиотеке использовать цифровой вывод 12 для сигнала включения вместо аналогового вывода 5, который I2C использует для сигнала SDA.

Table 11-3. *Требуемые детали for Project 31*

Arduino Uno или Mega	
MPL3115A2 Pressure Sensor	
Резистор 150 Ом	
10 кОм Potentiometer	
128x64 GLCD	

Подключение

Подключите все, как показано на рисунке 11-7. Эта схема показывает, как соединить компоненты на Arduino Mega. Вам нужно будет изменить некоторые соединения, если вы используете другую плату Arduino.

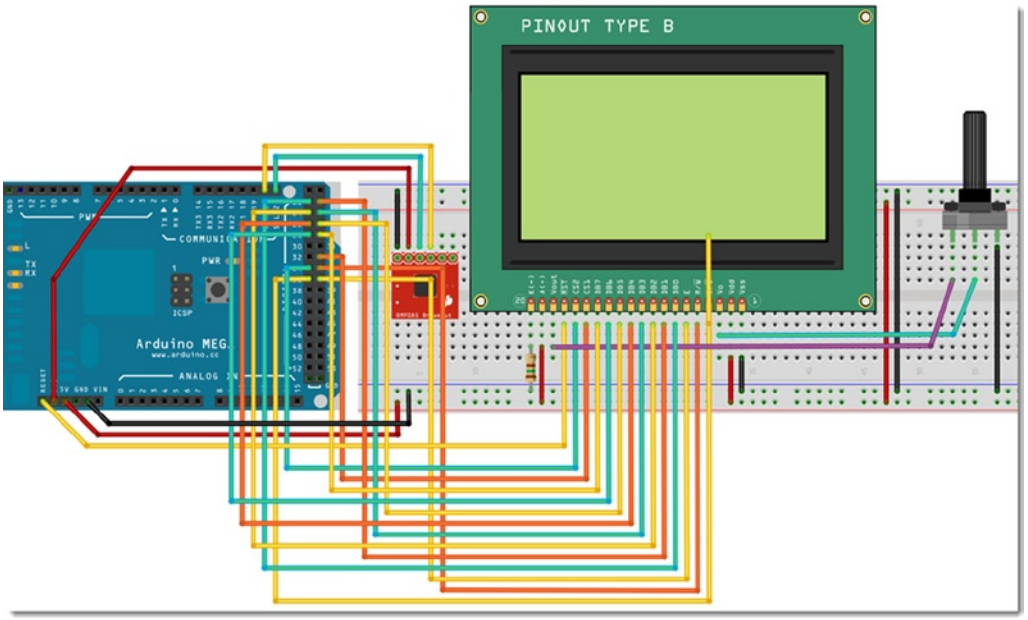


Рис. 11-7. *Схема для проекта 32 - цифровой барограф*

Часть схемы MPL3115A2 не изменилась по сравнению с проектом 31. Вы просто добавляете некоторые дополнительные компоненты в эту схему. Распиновка используемого вами GLCD может отличаться от распиновки в этом проекте. Прочтите техническое описание и убедитесь, что контакты соответствуют таковым в Таблице 11-4.

Таб.11-4. *Распиновка Uno или Mega и GLCD*

Uno	Mega	GLCD	Other
GND	GND	GND	
+5v	+5v	+5v	
		LCD контраст.	центр. вывод потенциометра
8	36	D/I	
9	35	R/W	
10	37	Enable	
11	22	DB0	
4	23	DB1	
5	24	DB2	
6	25	DB3	
7	26	DB4	
A0	27	DB5	
A1	28	DB6	
A2	29	DB7	
A3	33	CS1	
12	34	CS2	
Reset	Reset	Reset	
		VEE	Positive side of pot
		Backlight +5v	+5В
		Backlight +0v	GND через резистор 150 Ом

Обратите внимание, что распиновка для этих ЖК-дисплеев не стандартизирована - существует как минимум четыре общих варианта. Проверьте распиновку вашей конкретной модели ЖК-дисплея. Некоторые ЖК-дисплеи имеют питание на контакте 1 и заземление на контакте 2, у других - наоборот. Если вы перепутаете питание вы повредите ЖК-дисплей. Поэтому не следует слепо следовать приведенной выше схеме, не проверив предварительно распиновку для своего ЖК-дисплея. Не включайте схему до тех пор, пока все не будет подключено и вы дважды не проверите соединения. В частности, убедитесь, что один вывод потенциометра заземлен, при этом центральный вывод идет к контакту регулировки контрастности ЖК-дисплея, а другой вывод - к VEE (напряжение ЖК-дисплея). Резистор 15 Ом предназначен для ограничения тока, идущего к подсветке ЖК-дисплея; вам может потребоваться подобрать значение, чтобы получить соответствующую яркость, но убедитесь, что вы не превышаете напряжение, указанное в таблице данных для вашего ЖК-дисплея. Потенциометр используется для регулировки контрастности дисплея.

Также помните, что питание, поступающее на GLCD, должно быть + 5 В, а питание датчика давления должно быть 3,3 В. Не перепутайте их и не подключайте датчик давления к источнику питания 5 В, иначе вы повредите датчик. Перед включением схемы внимательно проверьте все.Убедившись, что все подключено правильно, включите Arduino. Вы должны увидеть, что GLCD загорится, готовый к приему данных. Теперь введите код.

Ввод кода

Перед тем, как ввести код, вам нужно будет загрузить и установить библиотеку GLCD. Вы можете найти ее на http://playground.arduino.cc/Code/GLCDks0108. Эта замечательная библиотека, написанная Майклом Марголисом, в настоящее время находится в третьей версии. Он поставляется сподробным документом, который показывает, как подключить различные типы GLCD, и полными инструкциями для всех команд в библиотеке. После того, как вы загрузите библиотеку, разархивируйте ее и поместите всю папку в папку«библиотеки» внутри вашей Arduino IDE. В следующий раз, когда вы запустите Arduino IDE, она будет загружена и готова к использованию.

Введите код из Листинга 11-2. или загрузите его из папки с кодами

Listing 11-2. Code for Project 32

```
// Project 32
#include <glcd.h>
#include "fonts/allFonts.h"
#include <Wire.h> // чтобы мы могли использовать I2C

// Высота в вашем местоположении необходима для расчета среднего значения моря
// уровень давления в метрах
#define MYALTITUDE 262
#define SENSORADDRESS 0x60 // Адрес MPL3115A1
#define SENSOR_CONTROL_REG_1 0x26
#define SENSOR_DR_STATUS 0x00  // Адрес регистра состояния
#define SENSOR_OUT_P_MSB 0x01  // Начальный адрес регистров данных давления

  // Временной интервал в секундах (приблизительно) на график
// Масштабирование для вычитания из показаний в гПа, чтобы соответствовать графику в 40 пикселей
990-1030
#define INTERVAL 900

float baroAltitudeCorrectionFactor = 1/(pow(1-MYALTITUDE/44330.77,5.255877));

byte I2Cdata[5] = {0,0,0,0,0}; //буфер для данных датчика давление поплавка,
температура, буфер [30], baroPressure;
int dots[124], dotCursor = 0, counter = 0, index = 0, numReadingsBuffered = 0;

void setup()
{
  Wire.begin(); // присоединяемся к шине i2c
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00000000); // переводим в режим ожидания
  // эти старшие биты регистра управления
  // можно изменить только в режиме ожидания

// Устанавливаем передискретизацию на 128. Затем запускаем первое чтение
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00111010);
```

```
  GLCD.Init();   // инициализируем библиотеку
  GLCD.ClearScreen();
  GLCD.SelectFont(System5x7, BLACK); // загружаем шрифт

  GLCD.DrawRect(1,1,125,44); // Рисуем прямоугольник
  for (int x=0; x<46; x+=11) { // Рисуем вертикальную шкалу
    GLCD.SetDot(0,1+x, BLACK);
    GLCD.SetDot(127,1+x, BLACK);
  }
  for (int x=0; x<128; x+=5) { // Рисуем горизонтальную шкалу
    GLCD.SetDot(1+x,0, BLACK);
  }

// Очищаем массив до стандартного уровня моря
  for (byte x=0; x<124; x++) {dots[x]=1013;}

  Read_Sensor_Data();
  drawPoints(dotCursor);
}

void loop()
{
  Read_Sensor_Data();

  GLCD.CursorToXY(0, 49); // печать давления
  GLCD.print("hPa:");
  GLCD.CursorToXY(24,49);
  GLCD.print(baroPressure/100);
  GLCD.print("   ");  // стираем любое старое значение длиннее нового

  float tempF = (temperature*1.8) + 32;

  GLCD.CursorToXY(0,57); // печать температуры
  GLCD.print("Temp:");
  GLCD.CursorToXY(28, 57);
//  GLCD.print(temperature); // change to temperature for Centigrade
  GLCD.print(tempF);
  GLCD.print("   ");  // erase any old value longer than new value

  delay(1000);

  GLCD.CursorToXY(84,49); // выводим тренд (тенденцию)
  GLCD.print("TREND:");
  GLCD.CursorToXY(84,57);
  printTrend();

  counter++;
  if (counter==INTERVAL) {
    drawPoints(dotCursor);
    counter = 0;
  }
}
```

```
void drawPoints(int position) {
  dots[dotCursor] = int(baroPressure/100);
  int midscale = dots[dotCursor];  // центральная шкала графика при текущих показаниях
  GLCD.FillRect(2, 2, 123, 40, WHITE); // очищаем область графика
  for (int x=0; x<124; x++) {
    // ограничиваем границей графа
    int y = constrain(22-((dots[position]- midscale)), 0,44);
    GLCD.SetDot(125-x,y, BLACK);
    position--;
    if (position<0) {position=123;}
  }
  dotCursor++;
  if (dotCursor>123) {dotCursor=0;}
}

// вычисляем тренд с момента последней точки данных и печатаем
void printTrend() {
  int dotCursor2=dotCursor-1;
  if (dotCursor2<0) {dotCursor2=123;}
  int val1=dots[dotCursor2];
  int dotCursor3=dotCursor2-1;
  if (dotCursor3<0) {dotCursor3=123;}
  int val2=dots[dotCursor3];
  if (val1>val2) {GLCD.print("RISING ");}
  if (val1==val2) {GLCD.print("STEADY ");}
  if (val1<val2) {GLCD.print("FALLING");}
}

  // Эта функция запускает чтение, ожидает завершения сбора и
  //  считывает показания давления и температуры с датчика
void Read_Sensor_Data(){
float tempPressure;

  // запрашиваем одно измерение от датчика
  I2C_Write(SENSOR_CONTROL_REG_1, 0b00111010); //бит 1 - одноразовый режим
  // ждем завершения измерения.
  // Одноразовый бит очищается, когда это будет сделано.
  do {
   // читаем текущий (сенсорный) регистр
   // затем повторяем, пока датчик не сбросит бит OST
    Wire.requestFrom(SENSORADDRESS,1);
  } while ((Wire.read() & 0b00000010) != 0);

  I2C_ReadData(); //считывает регистры с датчика
  temperature = Calc_Temperature();
  tempPressure = Calc_Pressure();
  buffer[index++]=tempPressure;
  if (index > 29)  index = 0; // в конце буфера, чтобы начать.
  numReadingsBuffered++;
  if (numReadingsBuffered > 30)   numReadingsBuffered = 30;
```

```
  float mean = 0;
  for (int i=0; i<numReadingsBuffered; i++) {
    mean = mean+buffer[i];
  }
  pressure = mean/numReadingsBuffered;
  baroPressure = pressure*baroAltitudeCorrectionFactor;
}

// Эта функция принимает значения из буфера чтения
// и преобразует их в единицы давления
float Calc_Pressure(){
  unsigned long m_pressure = I2Cdata[0];
  unsigned long c_pressure = I2Cdata[1];
  float l_pressure = (float)(I2Cdata[2]>>6)/4;
  return((float)(m_pressure<<10 | c_pressure<<2)+l_pressure);
}

// Эта функция собирает показания температуры
// из значений в буфере чтения
float Calc_Temperature(){
  int m_temp;
  float l_temp;
  m_temp = I2Cdata[3]; //температура в градусах Цельсия
  l_temp = (float)(I2Cdata[4]>>4)/16.0; //дробная часть
  return((float)(m_temp + l_temp));
}
void I2C_ReadData(){  //Считываем данные барометра и температуры (5 байтов)
  byte readUnsuccessful;
  do {
    byte i=0;
    byte dataStatus = 0;

    Wire.beginTransmission(SENSORADDRESS);
    Wire.write(SENSOR_OUT_P_MSB);
    Wire.endTransmission(false);

    //читаем 5 байтов. 3 для давления, 2 для температуры.
    Wire.requestFrom(SENSORADDRESS,5);
    while(Wire.available()) I2Cdata[i++] = Wire.read();

    // в некоторых режимах датчик может
    // обновлять показания давления, пока мы были в
    // в середине чтения, и в этом случае наша копия - мусор
    // потому что у него есть части двух разных чтений.
    // Мы можем проверить некоторые статусные биты в DR (данные готовы)//
    регистрируемся, чтобы узнать, произошло ли это.

    Wire.beginTransmission(SENSORADDRESS);
    Wire.write(SENSOR_DR_STATUS);
    Wire.endTransmission(false);
```

```
    //читаем 1 байт, чтобы получить значение статуса
    Wire.requestFrom(SENSORADDRESS,1);
    dataStatus = Wire.read();

    readUnsuccessful = (dataStatus & 0x60) != 0;
    // неудачно, если перезапись произошла, пока мы
    // читали данные о давлении или температуре
       // продолжаем читать, пока не получим успешное чистое чтение
  } while (readUnsuccessful);
}

// Эта функция записывает один байт по I2C
void I2C_Write(byte regAddr, byte value)
{Wire.beginTransmission(SENSORADDRESS);
  Wire.write(regAddr);
  Wire.write(value);
  Wire.endTransmission(true);
}
```

Когда вы запустите код, вы увидите график изменения давления во времени, а также текущее давление и температуру. Поначалу это не очень интересно, так как требуется более 24 часов, чтобы показать изменения давления. Каждая точка представляет 15 минут времени, а горизонтальная шкала - часы. Если вы хотите ускорить процесс и быстрее заполнить график, измените значение в определении ИНТЕРВАЛА на меньшее. 900 в ИНТЕРВАЛЕ - это секунды (900 секунд - 15 минут), поэтому измените его на меньшее значение интервала, чтобы увидеть ускоренную версию графика.Однако, чтобы график был полезен для прогнозирования погоды, оставьте его на 900 секунд и оставьте схему работать более 24 часов.

Проект 32 - Цифровой барограф - Обзор кода

Большая часть кода для Проекта 32 идентична Проекту 31. Однако есть новые разделы для управления GLCD,а части для отправки данных на последовательный монитор были удалены. Поэтому я исключу код, который повторяется из Проекта 31, и сконцентрируюсь на дополнениях. Во-первых, вам нужно включить в свой код файл заголовка библиотеки GLCD.h, а также шрифты, которые вы собираетесь использовать. Мы также включаем библиотеку проводов (wire), чтобы мы могли общаться с нашим датчиком I2C:

```
#include <glcd.h>
#include "fonts/allFonts.h"
#include <Wire.h>
```

Как и в Проекте 31, теперь мы устанавливаем значение нашей высоты в метрах, а также адрес датчика. На этот раз мы используем оператор #define, который более эффективен, поскольку не требует памяти для хранения значений.

Вместо этого компилятор просто заменяет каждое слово MYALTITUDE на 83 (или любую другую высоту), а SENSORADDRESS на 0x60. То же самое касается последних трех операторов define, которые предназначены для внутренних регистров датчика.

```
#define MYALTITUDE 262
#define SENSORADDRESS 0x60
#define SENSOR_CONTROL_REG_1 0x26
#define SENSOR_DR_STATUS 0x00
#define SENSOR_OUT_P_MSB 0x01
```

В разделе адресов есть новое определение для INTERVAL - ИНТЕРВАЛА. Оно определяет интервал в секундах между точками данных, сохраняемыми и отображаемыми на графике. Интервал составляет 900 секунд, чтосоставляет 15 минут между точками.

```
#define INTERVAL 900  // Временной интервал в секундах (приблизительно)
```

Мы также определяем масштаб для нашего графика.

```
#define SCALE 990
```

На графике показан диапазон 40 гПа. Поэтому вы хотите, чтобы ваш график находился где-то в центре графика. Используйте Проект 31, чтобы определить ваше текущее показание в гПа (гПа - это просто давление в Па, деленное на 100), а затем вычтите 22 из этого показания, чтобы получить масштабный коэффициент, например если ваше показание давления составляет 1028, тогда масштаб будет 1008, что приведет к началу вашего графика в центре графика в 44 пикселя.

Показания датчика давления необходимо откорректировать в соответствии с вашей высотой. Поэтому нам нужен поправочный коэффициент, и поэтому создайте переменную типа float с именем baroAltitudeCorrectionFactor, используя вычисление в таблице данных MPL3115A2.

```
float baroAltitudeCorrectionFactor = 1/(pow(1-MYALTITUDE/44330.77,5.255877));
```

Затем мы создаем наш массив, который будет хранить пять байтов показаний давления и температуры.

```
byte I2Cdata[5] = {0,0,0,0,0};
```

Мы также создаем переменные давления и температуры. В Project 31 они находились внутри функции Read-Sensor_Data (). Однако нам нужна эта функция, чтобы возвращать показания температуры и давления за пределы функции, чтобы ее можно было использовать в другом месте программы. Поскольку невозможно легко вернуть два отдельных значения из функции (это можно сделать, но усложняет ситуацию для новичков), мы создаем две переменные вне всех других функций, что дает им глобальную область действия. Это означает, что переменные доступны из любой функции в программе. На этот раз мы увеличиваем буферный массив с 10 до 30 элементов. Мы будем использовать этот массив в Project 32 для хранения показаний давления, снятых каждую секунду в течение 30 секунд, а затем их усреднение для получения более точных показаний давления. Вы могли заметить, что показания в Project 31 время от времени сильно менялись. Это произошло из-за шума или других факторов в нашей схеме, которые иногда приводили к неточным показаниям. Усредняя показания с течением времени, мы получим гораздо более точные показания. Однако имейте в виду, что этот подход требует много памяти. Каждый элемент представляет собой число с плавающей запятой, занимающее четыре байта каждый, а у нас есть 30 элементов. Это означает, что один только этот массив занимает 120 байт в памяти. Это может показаться не слишком большим в мире ПК, где Гб памяти является обычным явлением. Однако на нашем скромном маленьком Arduino у нас ограниченный объем памяти.

```
float pressure, temperature, buffer[30], baroPressure;
```

Объявлены и инициализированы пять новых целых чисел. Один из них - массив точек [], в котором измеряются значения давления, считываемые с пятнадцатиминутными интервалами. Следующей идет переменная dotCursor, в которой хранится индекс массива, в котором вы сейчас находитесь. Затем переменная счетчика будет увеличиваться каждый раз, когда основной цикл повторяется, и будет использоваться, чтобы узнать, сколько секунд (примерно) прошло с момента сохранения последней точки данных. Переменная index запомнит текущее место в буферном массиве, в котором мы храним данные датчика. Наконец, numReadingsBuffered используется для хранения того, сколько показаний мы сохранилив массиве buffer [] с течением времени. Каждые 900 секунд это будет увеличиваться на единицу, пока мы не достигнем максимума 30 показаний. Мы усредним наши показания позже, используя numReadingsBuffered.

```
int dots[124], dotCursor = 0, counter = 0, index = 0, numReadingsBuffered = 0;
```

В процедуре настройки вы сначала инициализируете связь I2C

```
Wire.begin();
```

Затем, как и в Проекте 31, датчик переводится в режим ожидания, а частота передискретизации устанавливается на 128x.

```
I2C_Write(SENSOR_CONTROL_REG_1, 0b00000000);
I2C_Write(SENSOR_CONTROL_REG_1, 0b00111010);
```

Затем инициализируйте дисплей GLCD:

```
GLCD.Init();
```

Все функции для управления GLCD идут после точки, например следующая команда для очистки экрана:

```
GLCD.ClearScreen();
```

Затем вы выбираете, какой шрифт вы собираетесь использовать, и будет ли он отображаться в виде черных или белых пикселей:

```
GLCD.SelectFont(System5x7, BLACK); // загрузить шрифт
```

Есть много разных типов и размеров шрифтов, которые можно использовать. Прочтите инструкции, прилагаемые к библиотеке GLCD, и попробуйте разные шрифты. Кроме того, библиотека включает классное бесплатную программу под названием FontCreator2, которое можно использовать для создания файла заголовка шрифта для включения в ваш скетч. Эта программа может конвертировать шрифты ПК для использования с библиотекой. Затем отображается поле для графика. Это делается с помощью команды DrawRect:

```
GLCD.DrawRect(1,1,125,44);
```

Система координат для дисплея 128x64 имеет ширину 128 пикселей и высоту 64 пикселя, при этом пиксель 0 для обеих осей находится в верхнем левом углу. Координата X затем простирается от 0 до 127. Координата Y изменяется от 0 до 63 внизу. DrawRect рисует прямоугольник по координатам X и Y, которые образуют первые два параметра. Это верхний левый угол коробки. Следующие два параметра - это высота и ширина, простирающиеся от оси X и Y ординаты. Ваше окно идет от точки 1,1 и растягивается на 125 пикселей в ширину и 44 пикселей в высоту.

Вы также можете создать заполненный прямоугольник с помощью команды FillRect (), поэтому

```
GLCD.FillRect(1,1,125,44, BLACK);
```

создаст сплошной черный прямоугольник тех же размеров. Далее вам понадобится вертикальный и горизонтальный масштаб. Вы используете цикл for для создания точек размером 0 пикселей и 127 пикселей по горизонтали и с интервалами в 11 пикселей, начиная с 1 пикселя вниз:

```
for (int x=0; x<46; x+=11) {
  GLCD.SetDot(0,1+x, BLACK);
  GLCD.SetDot(127,1+x, BLACK);
}
```

Вы делаете это с помощью команды SetDot, которая просто помещает один пиксель в координаты X и Y в ЧЕРНЫЙ или БЕЛЫЙ цвет. Белые пиксели просто сотрут любые уже отображаемые черные пиксели.

Затем вертикальная шкала (часы) рисуется с интервалом в пять пикселей от пикселя 1 до правой стороны:

```
  for (int x=0; x<128; x+=5) {
  GLCD.SetDot(1+x,0, BLACK);
}
```

Теперь мы перебираем массив dots [] и помещаем значение 1,013, которое представляет собой давление на уровне моря, в каждый элемент в качестве отправной точки для графика.

```
for (byte x=0; x<124; x++) {dots[x]=1013;} // очищаем массив до стандартного уровня моря
```

Затем мы вызываем две функции, которые считывают данные с датчика, а затем наносим их на график.

```
Read_Sensor_Data();
drawPoints(dotCursor);
```

Теперь запускается функция основного цикла программы.

```
void loop()
{
```

Сначала вызывается функция чтения данных датчика.

```
Read_Sensor_Data();
```

Затем мы используем функцию GLCD.CursorToXY (), чтобы переместить указатель на указанные координаты X и Y, а затем функцию GLCD.print () для печати давления и температуры.

```
  GLCD.CursorToXY(0, 49); // печать давления
  GLCD.print("hPa:");
  GLCD.CursorToXY(24,49);
  GLCD.print(baroPressure/100);
  GLCD.print("   ");  // стираем любое старое значение длиннее нового

  float tempF = (temperature*1.8) + 32;

  GLCD.CursorToXY(0,57); // печатаем температуру
  GLCD.print("Temp:");
  GLCD.CursorToXY(28, 57);
//  GLCD.print(temperature); // меняем на температуру по Цельсию
  GLCD.print(tempF);
  GLCD.print("   ");  // стираем любое старое значение длиннее нового
```

Подождем одну секунду.

```
  delay(1000);
```

Затем печатаем тренд (тенденцию) с помощью функции printTrend (). Тренд показывает, растет ли давление, стабильно или падает.

```
GLCD.CursorToXY(84,49); // печатаем trend
GLCD.print("TREND:");
GLCD.CursorToXY(84,57);
printTrend();
```

Далее счетчик увеличивается. Как только счетчик достиг того же значения, что и наш интервал (в нашем случае 900 секунд), функция drawPoints () используется для построения графика.

```
  counter++;
  if (counter==INTERVAL) {
    drawPoints(dotCursor);
    counter = 0;
  }
}
```

Затем вы добавили две новые функции для построения графика и печати текущего тренда давления. Первая - это функция drawPoints (). Вы передаете ему значение dotCursor в качестве параметра:

```
void drawPoints(int position) {
```

Текущее показание давления сохраняется в массиве точек [] в текущей позиции. Поскольку вас интересует толькоточка на дисплее с низким разрешением, вам не нужны числа после десятичной точки, поэтому преобразуйте hPa в целое число. Это также экономит память, поскольку массив целочисленных значений занимает меньше места, чем массив чисел с плавающей запятой.

```
dots[dotCursor] = int(baroPressure/100);
```

Затем график должен быть центрирован по текущему показанию.

```
int midscale = dots[dotCursor];
```

Теперь вам нужно очистить график от новых данных. Это делается с помощью команды FillRect, создавая белый прямоугольник внутри границ графического поля:

```
GLCD.FillRect(2, 2, 123, 40, WHITE); // clear graph area
```

Теперь вы перебираете 124 элемента массива с помощью цикла for:

```
for (int x=0; x<124; x++) {
```

Затем мы вычисляем Y-позицию графика. Это давление, хранящееся в массиве точек []. Мы берем 22 (высоту области графика, деленную на 2) и вычитаем позицию точки, из которой был вычтен коэффициент среднего масштаба.Весь этот расчет выполняется внутри команды ограничения, которая гарантирует, что результат расчета не будет ниже 0 или выше 44, или он будет распечатан за пределами границы области графика.

```
int y = constrain(22-((dots[position]-midscale)), 0,44);
```

Затем поместите точку в соответствующее место на графике с помощью команды SetDot ():

```
GLCD.SetDot(125-x,y, BLACK);
```

Вы хотите, чтобы график рисовался справа налево, чтобы самые актуальные показания находились справа, а график прокручивался влево. Итак, вы начинаете с рисования первой точки в координате X 125-x, которая перемещает точки влево по мере увеличения значения X. Координата Y рассчитывается, как указано выше. Используйте Проект 31, чтобы определить текущее значение гПа и вычтите из него 22, чтобы получить номер шкалы.
Это типичные значения гПа для большинства мест. Если вы живете врайоне с обычно более высоким или болеенизким давлением, вы можете соответствующим образомотрегулировать значение средней шкалы. Затем вы вычитаете это число из 22, чтобы получитьположение точки Y для этого конкретного значения давления в массиве.

Значение позиции, которое изначально было установлено как текущее значение dotCursor, затем уменьшается.

```
position--;
```

и, если он опускается ниже нуля, возвращается к 123.

```
if (position<0) {position=123;}
```

После того, как все 124 точки нарисованы, значение dotCursor увеличивается, и оно готово к сохранению следующего нажатия,измерение в массиве

```
dotCursor++;
```

и, если он превышает значение 123 (максимальный элемент массива), возвращается к нулю

```
if (dotCursor>123) {dotCursor=0;}
```

Следующая новая функция - это printTrend (). Задача этой функции состоит в том, чтобы просто определить, является ли текущее сохраненное измерение давления выше, ниже или таким же, как последнее сохраненное значение, и распечатать RISING, STEADY или FALLING соответственно.

Вы начинаете с сохранения последней позиции dotCursor в dotCursor2. Вы вычитаете единицу из его значения, поскольку оно увеличивалось после сохранения измерения в функции drawPoints ().

```
int dotCursor2=dotCursor-1;
```

Вы проверяете, меньше ли это значение нуля, и если да, то снова устанавливаете его на 123:

```
if (dotCursor2<0) {dotCursor2=123;}
```

dotCursor2 теперь сохраняет позицию в массиве последнего выполненного измерения. Теперь вы объявляете целое число с именем val1 и сохраняете в нем последнее измерение давления.

```
int val1=dots[dotCursor2];
```

Теперь вы хотите, чтобы измерение было выполнено BEFORE (ПЕРЕД) последним, поэтому вы создаете другую переменную с именем dotCursor3, которая будет хранить позицию в массиве перед последней выполненной:

```
int dotCursor3=dotCursor2-1;
if (dotCursor3<0) {dotCursor3=123;}
int val2=dots[dotCursor3];
```

Теперь у вас есть val1 с последним показанием давления и val2 с показанием давления перед этим показанием. Все, что остается сделать, это решить, является ли последнее измеренное значение давления выше, ниже или таким же, как и предыдущее, и соответствующим образом распечатать соответствующий тренд.

```
if (val1>val2) {GLCD.print("RISING ");}
if (val1==val2) {GLCD.print("STEADY ");}
if (val1<val2) {GLCD.print("FALLING");}
```

Наконец, есть пять функций, которые мы использовали в Project 31: Read_Sensor_Data (), Calc_Pressure (), Calc_Temperature (),I2C_ReadData () и I2C_Write ().

Когда вы запустите эту программу, вы получите экран, аналогичный показанному на рис. 11-8. Если давление значительно изменилось за последние 24 часа, вы увидите сильно изменяющуюся линию.

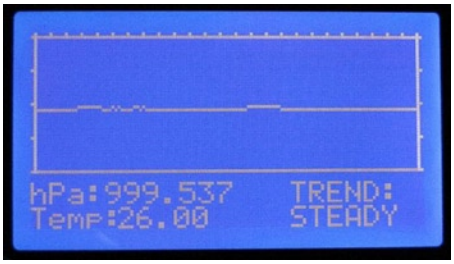


Рис. 11-8. *Дисплей для Project 32 - Цифровой барограф*

Основная цель проекта 32 состояла в том, чтобы показать вам практическое использование датчика MPL31125A2 и то, как использовать дисплей GLCD. Представленные команды для GLCD - это просто пример того, что вы можете делать с GLCD. Вы можете отображать прямоугольники, круги и линии, устанавливать точки и даже рисовать растровые изображения. Экран можно разделить на текстовую и графическую области, и вы можете получить к ним доступ и распечатать их независимо. В библиотеке очень хорошая документация авторства Майкла Марголиса, и им очень легко пользоваться. Внимательно прочтите документацию, и вы увидите, что с ней можно многое сделать. В библиотеке также есть целый набор примеров скетчей, в том числе «Игра жизни» Джона Хортона Конвея и отличная маленькая ракетная игра.

Резюме

В главе 11 показано, как использовать цифровой датчик давления и связываться с ним по шине I2C. Вы также познакомились с основными концепциями SPI и принципами его работы. Теперь, когда вы примерно знаете, как работает SPI, вы можете использовать отличную библиотеку SPI, поставляемую вместе с библиотекой Arduino. Это сделает за вас всю тяжелую работу при общении с любыми другими устройствами SPI. Вы узнали, как использовать I2C для чтения данных с датчика давления MPL3115A2. Если вы хотите создать собственную метеостанцию, этот недорогой датчик - идеальный выбор. Я выбрал этот датчик для любительской попытки создать проект на воздушном шаре на большой высоте, так как он маленький, точный и простой в использовании.Вы также узнали, как подключить графический ЖК-дисплей к Arduino и как легко печатать на нем текст и базовую графику с помощью библиотеки GLCD. Прочитав документацию дальше, вы сможете узнать, как делать интересные вещи, например отображать растровые изображения или создавать свои собственные игры. Arduino с GLCD можно легко поместить в небольшую коробку, чтобы сделать вашу собственную портативную игровую консоль.

Предметы и понятия, затронутые в главе 11

- Как подключить датчик давления MPL3115A2 к Arduino
- Как использовать #define для выполнения побитовых операций с набором чисел
- Как создавать числа с большей битовой длиной, комбинируя вместе меньшие числа битовой длины
- Как работает шина I2C
- Как обратиться к устройству I2C
- Как читать данные с устройства I2C
- Как записать данные на устройство I2C

- Как использовать функции библиотеки Wire для инициирования и завершения передачи
- Как использовать функции библиотеки Wire для запроса байтов от ведомого устройства
- Как работает интерфейс SPI
- Устройствами SPI можно управлять отдельно или в гирляндную цепочку
- Устройство SPI состоит из главного и подчиненного устройства
- Как данные поступают и отправляются из устройства SPI с помощью тактового импульса
- Назначение и использование трех регистров шины SPI
- Преобразование давления в паскалях в гектопаскалях и атмосферах
- Использование побитовых операторов для проверки, установлен ли один бит или нет
- Как подключить графический ЖК-дисплей к Arduino
- Использование основных команд библиотеки GLCD для рисования линий, точек и печати текста

В следующей главе вы узнаете, как использовать сенсорный экран.

ГЛАВА 12

Сенсорные дисплеи

Теперь вы познакомитесь с классным гаджетом, который можно легко использовать с Arduino - сенсорным дисплеем. С появлением смартфонов и портативных игровых консолей сенсорные дисплей стали недорогими и легкодоступными.

Сенсорный дисплей позволяет создать простой сенсорный интерфейс для устройства или может быть наложен на ЖК- дисплей для создания сенсорного интерфейса. Такие компании, как Sparkfun, упрощают взаимодействие с этими устройствами, предоставляя разъемы и коммутационные блоки.

Коммутационный блок позволяет вставить модуль в макетную плату. В этом проекте мы используем доступный сенсорный дисплей Nintendo DS с коммутационным блоком от Sparkfun.

Вы начнете с простого проекта, который покажет, как получать показания с сенсорного экрана, прежде чем использовать его.

Проект 33 - Базовый сенсорный дисплей

Для этого проекта вам понадобится сенсорный дисплей Nintendo DS, а также модуль коммутации. Последний важен, так как данные с сенсорного дисплея выводятся через очень тонкий и хрупкий ленточный разъем, и без дополнительных компонентов будет невозможно подключиться к Arduino.

Требуемые детали

Сенсорные дисплеи Nintendo DS недороги и их можно приобрести у многих поставщиков. Версия XL примерно в два раза больше стандартной версии; это рекомендуемая единица, если вы можете ее получить. Модуль коммутации от Sparkfun или их дистрибьюторов.

Таб. 12-1. *Детали, необходимые для проекта 33*

Сенсорный дисплей Nintendo DS	
Модуль коммутации сенсорного дисплея	
Резисторы 47K x 2	

Подключение

Подключите, как показано на рис. 12-1.

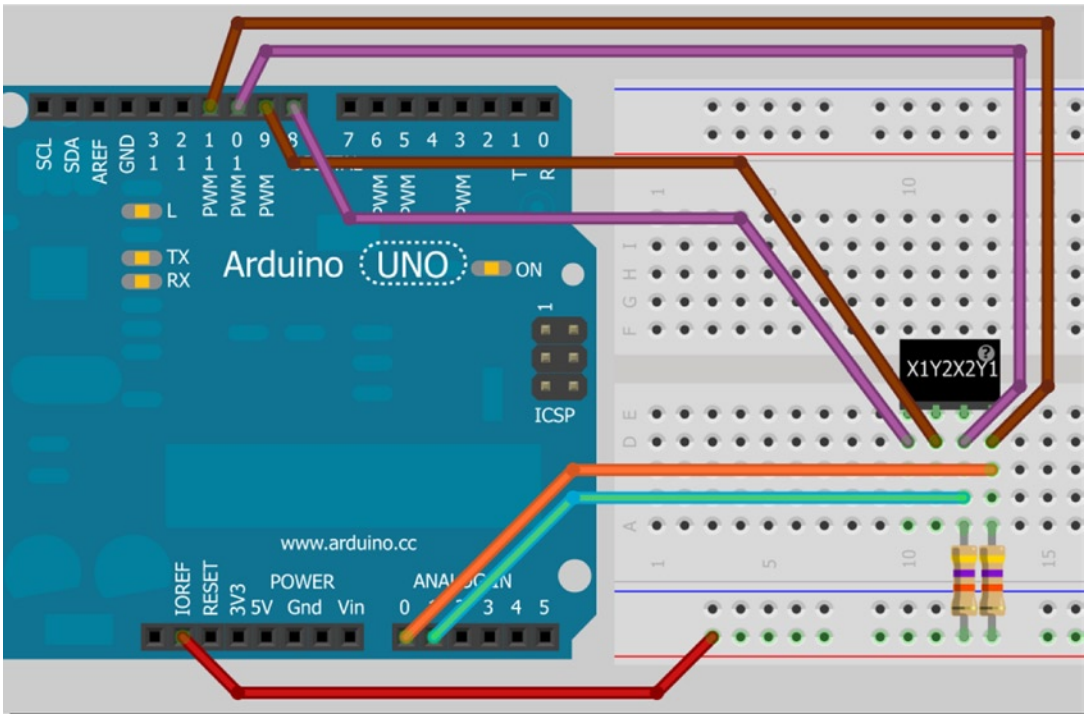


Рис. 12-1. *Схема для проекта 33 - Базовый сенсорный дисплей*

На коммутационном модуле контакты обозначены как X1, Y2, X2 и Y1. Соедините штыри, как описано в Таблице 12-2. Вам понадобится подключение вывода IOREF (или вывода 5V, если у вас более старая Arduino без вывода IOREF) к макетной плате, а затем резисторы подходящего номинала (около 50 кОм) между выводами X2 и Y1 и линией IOREF на макетной плате. .

Таб. 12-2. *Штыревые соединения для проекта 33*

Arduino	Разъем
Цифровой пин 8	X1
Цифровой пин 9	Y2
Цифровой пин 10	X2
Цифровой пин 11	Y1
Аналоговый пин 0	Y1
Аналоговый пин 1	X2
IOREF (или 5 В) через резистор Y1 47 кОм	Y1
IOREF (или 5 В) через резистор Y1 47 кОм	X2

Вам нужно будет припаять несколько контактов разъема к коммутационному блоку. Контакты припаяны таким образом, чтобы логотип Sparkfun был обращен вверх. Экран подключается к коммутационному блоку через небольшой разъем. Оттяните язычок и вставьте крошечный ленточный кабель в разъем, затем надавите на язычок, чтобы зафиксировать его на месте. Отныне будьте очень осторожны с конструкцией: оно очень хрупкое и легко ломается! При тестировании сломал три дисплея и два разъема. Если вы можете найти способ закрепить макетную плату, разъем и сенсорный дисплей на какой-нибудь площадке, чтобы предотвратить перемещение всех частей, сделайте это.

Введите код

Введите код из Листинга 12-1.

Листинг 12-1. Код для проекта 33

```
// Проект 33

Цифровые соединения
#define Left 8                  // Лево (X1) на цифровой пин 8
#define Bottom 9                // Вниз (Y2) на цифровой пин 9
#define Right 10                // вправо (X2) на цифровой пин 10
#define Top 11                  // Вверх (Y1) на цифровой пин 11

Аналоговые соединения
#define topInput 0              // Верх (Y1) к аналоговому выводу 0
#define rightInput 1            // Вправо (X2) на аналоговый вывод 1

int coordX = 0, coordY = 0;

void setup()
{
        Serial.begin(38400);
}

void loop()
{
                            if (touch())            // Если  коснулись, вывести координаты
        {
                Serial.print(coordX);
                Serial.print("  ");
                Serial.println(coordY);
                delay(250);
        }
}

// возвращаем TRUE (ИСТИНА) при прикосновении и устанавливаем координаты touchX и touchY
boolean touch()
{
        boolean touch = false;

                                // получаем горизонтальные координаты
```

```
        pinMode(Top, INPUT);     // Сверху и снизу до высокого импеданса (сопротивления)
        pinMode(Bottom, INPUT);

        pinMode(Left, OUTPUT);
        digitalWrite(Left, LOW); // Устанавливаем слева на низкое

        pinMode(Right, OUTPUT);  // Устанавливаем вправо на + 5В
        digitalWrite(Right, HIGH);

        delay(3);
        coordX = analogRead(topInput);

                                 // получаем вертикальные координаты

        pinMode(Right, INPUT);   // lслева и справа до высокого импеданса
        pinMode(Left, INPUT);

        pinMode(Bottom, OUTPUT); // устанавливаем низ  на землю
         digitalWrite(Bottom, LOW);

        pinMode(Top, OUTPUT);    // устанавливаем Top на + 5v
        digitalWrite(Top, HIGH);

        delay(3);
        coordY = analogRead(rightInput);

                        // если считанные координаты меньше 1000 и больше 24
                        // затем прикоснулись к экрану
        if(coordX < 1000 && coordX > 24 && coordY < 1000 && coordY > 24) {touch = true;}

          return touch;
}
```

Введите код и загрузите его в свой Arduino. После запуска откройте МПП и коснитесь сенсорного экрана. При каждом прикосновении к экрану координаты вашего пальца будут отображаться на МПП. Координаты: x в горизонтальной плоскости слева направо и y в вертикальной плоскости сверху вниз.

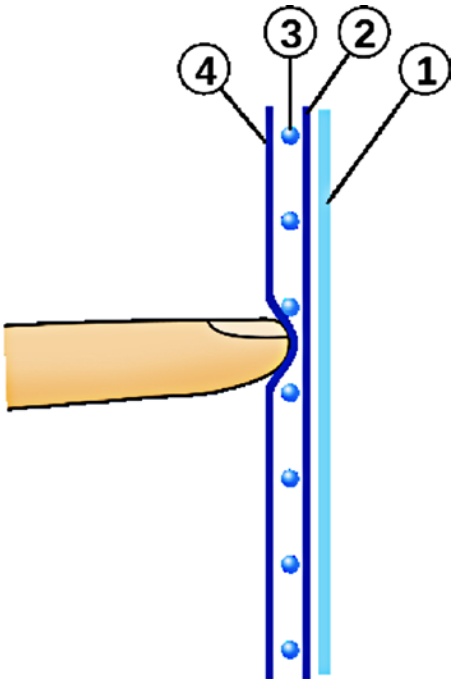
Прежде чем изучать код, будет полезно знать, как работает сенсорный экран. Поэтому рассмотрим работу дисплея, прежде чем изучать код.

Проект 33 - Базовый сенсорный дисплей - Обзор устройства

Дисплей Nintendo DS известен как резистивный сенсорный экран,состоящий из двух прозрачных панелей. Одна панель - стеклянная (находится со стороны дисплея), вторая панель - из эластичного пластика (находится с внешней стороны, именно на неё мы и нажимаем), она выполняет роль мембраны. На внутренние поверхности обеих панелей (которые «смотрят» друг на друга) нанесён тонкий токопроводящий слой, имеющий однородное сопротивление. С противоположных сторон панелей присутствуют выводы, по два на каждую панель. Между панелями существует небольшой зазор заполненный микро-изоляторами (диэлектрическими микро-шариками), гарантирующими, что панели электрически не соединены. При касании мембраны, токопроводящие слои на мембране и на стекле соприкасаются друг с другом, электрически соединяясь. Сопротивление между выводами обеих панелей зависит от координат точки их соприкосновения.

Если вы рассмотрите сенсорный дисплей, вы увидите четыре разъема на ленточном кабеле, которые ведут к четырем металлическим полоскам по краям экрана. Две металлические полосы находятся вверху и внизу экрана, и если вы перевернете экран, вы увидите две другие на втором слое, а также на левой и правой сторонах экрана.

Когда палец или стилус прижимается к верхнему гибкому слою 4, слой изгибается, чтобы коснуться жесткого слоя 1 (см. Рисунок 12-2).



1: Жесткий слой.

2: Слой оксида металла.

3: Изолирующие точки.

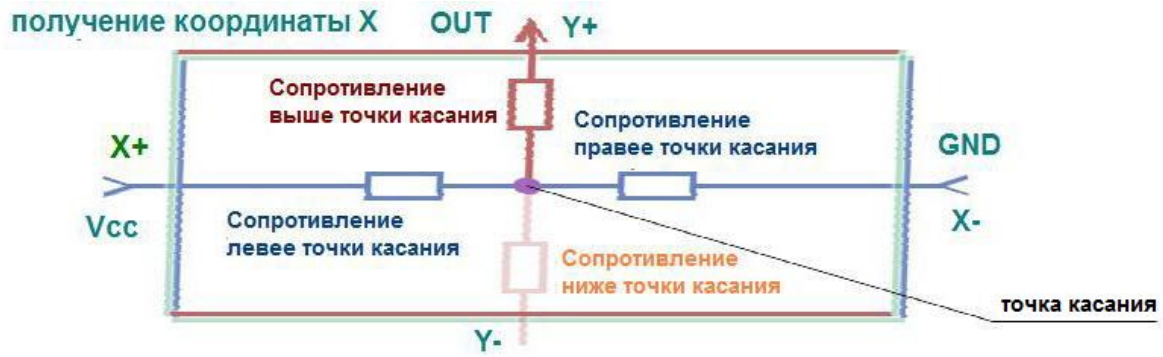
4: Гибкий слой с пленкой оксида металла

Рис. 12-2. *Рисунок 12-2. Как работает сенсорный экран*

Получение координат касания:

Для получения координат, нужно сконфигурировать два вывода Arduino подключённые к одной панели в режим входа, а два вывода Arduino подключённые к другой панели, в режим выхода. На выходы, подать разные логические уровни (на один LOW, на другой HIGH), а с одного из входов, считать уровень напряжения.

Получение координаты по оси X:



- Выводы Arduino к которым подключены выводы X+ и X- (с напыления на стекле) переводятся в режим выхода.
- На вывод X+ подаётся уровень HIGH, а на вывод X- подаётся уровень LOW.
- Выводы Arduino к которым подключены выводы Y+ и Y- (с напыления на мембране) переводятся в режим входа.
- Данные полученные функцией analogRead, со входа Y+ будут обратно пропорциональны координате касания по оси X.
- Вывод Y- находится в состоянии высокого импеданса, так как мы перевели его в режим входа, но данные с него читать не будем.

Напряжение снимаемое с выхода Y+ зависит от горизонтального делителя (R левее и R правее точки касания).
Вертикальный делитель не влияет на уровень снимаемого напряжения (R выше точки касания лишь ограничивает ток, а R ниже точки касания вообще не участвует в схеме, т.к. вывод Y- находится в состоянии высокого импеданса («висит» как не подключённый).

Получение координаты по оси Y:

- Выводы Arduino к которым подключены выводы Y+ и Y- (с напыления на мембране) переводятся в режим выхода.
- На вывод Y+ (YP) подаётся уровень HIGH, а на вывод Y- (YM) подаётся уровень LOW.
- Выводы Arduino к которым подключены выводы X+ и X- (с напыления на стекле) переводятся в режим входа.
- Данные полученные функцией analogRead, со входа X- (XM) будут обратно пропорциональны координате касания по оси Y.
- Вывод X+ находится в состоянии высокого импеданса, так как мы перевели его в режим входа,но данные с него читать не будем.

Напряжение снимаемое с выхода X- (XM) зависит от вертикального делителя (R выше и R ниже точки касания).
Горизонтальный делитель не влияет на уровень снимаемого напряжения (R правее точки касания лишь ограничивает ток, а R левее точки касания вообще не участвует в схеме, т.к. вывод X+ находится в состоянии высокого импеданса («висит» как не подключённый).

Если на выводы пластины, к которой подаётся напряжение, вместо LOW подать HIGH, а вместо HIGH подать LOW (поменять местами уровни выводов), то считанное значение будет не обратно, а прямо пропорционально координате касания.

Читать данные можно с любого вывода той пластины, на которую не подавалось напряжение (уровни LOW и HIGH), но так как обычно один из выводов соединён с аналоговым выводом Arduino, а другой с цифровым, то и читать данные приходится только с того вывода, который подключён к аналоговому выводу Arduino.

После чтения координат, все выводы нужно сконфигурировать в тот режим, который был установлен до чтения, для нормальной работы с дисплеем.

Проект 33 - Базовый сенсорный дисплей - Обзор кода

Код для чтения сенсорного экрана на самом деле очень прост. Вы начинаете с определения четырех цифровых выводов, которые вы будете использовать для подачи питания на слои, и двух аналоговых выводов, которые вы будете использовать для измерения напряжений:

```
// Соединения с цифровыми выводами
#define Left 8       // влево (X1) на цифровой вывод 8
#define Bottom 9     // вниз (Y2) к цифровому выводу 9
#define Right 10     // вправо (X2) на цифровой вывод 10
#define Top 11       // вверх (Y1) к цифровому выводу 11

// Соединения с аналоговыми выводами
#define topInput 0   // Верх (Y1) к аналоговому выводу 0
#define rightInput 1 // Вправо (X2) на аналоговый вывод 1
```

Затем вы объявляете и инициализируете целые числа X и Y, которые будут содержать координаты, которые изначально равны нулю:

```
int coordX = 0, coordY = 0;
```

Поскольку вы собираетесь считывать координаты с помощью МПП, в процедуре настройки все, что вам нужно сделать, это установить последовательную связь и установить скорость передачи. В этом случае вы будете использовать 38400 бод:

```
Serial.begin(38400);
```

Основной цикл программы содержит не что иное, как оператор if для определения прикосновения к экрану или отсутствие касания:

```
if (touch())  // Если касание есть, вывести координаты
```

Если прикоснулись к экрану, вы просто распечатаете координаты X и Y на мониторе последовательного порта (МПП) с пробелом между ними, используя команды Serial.print:

```
Serial.print(coordX);
Serial.print("  ");
Serial.println(coordY);
```

После того, как вы напечатали координаты, вы ждете четверть секунды, чтобы координаты можно было прочитать, если вы удерживаете палец на экране:

```
delay(250);
```

Далее идет функция, которая выполняет всю тяжелую работу. Функция будет возвращать логическое значение true (истина) или false (ложь), поэтому это логический тип данных. Вы не передаете функции никаких параметров, поэтому список параметров пуст.

```
boolean touch()
```

Вы объявляете переменную типа boolean и инициализируете ее значением false. Это будет содержать истинное или ложное значение в зависимости от прикосновения к экрану или нет.

```
boolean touch = false;
```

Затем 11 (Top) и 9 (Bottom) цифровые выводы устанавливаются на INPUT, чтобы они стали высокоимпедансными (высокоомными) ,т.е. на них нет ни LOW ни HIGH состояния:

```
pinMode(Top, INPUT);
pinMode(Bottom, INPUT);
```

Затем нужно подать напряжение на 8 и 10 цифровые выводы и считываем напряжение,используя аналоговый вывод 0. Для этого устанавливаем 8 и 10 выводы как выходы и 8 вывод в LOW состояние,а 10 вывод - HIGH состояние,чтобы на нем было 5В:

```
pinMode(Left, OUTPUT);
digitalWrite(Left, LOW); // Устанавливаем Left в Gnd (цифровой вывод 8)

pinMode(Right, OUTPUT);  // Устанавливаем вправо на +5В (цифровой вывод 10)
digitalWrite(Right, HIGH);
```

Затем после задержки, чтобы произошли указанные выше изменения состояния,считываем аналоговое значение с верхнего входного вывода (аналоговый вывод 0). Это значение затем сохраняется в координате X.

```
delay(3);
coordX = analogRead(topInput);
```

Теперь у нас есть координата X. Для определения координаты Y делаем то же самое,только имеем дело с 9 и 11 цифровыми выводами. Для вывода пинов 8 и 10 из игры,устанавливаем их как входы.т.е. в высокоимпедансный режим,когда на них нет ни HIGH ни LOW состояния и считываем напряжение с аналогового вывода 1.

```
pinMode(Right, INPUT);   // 8 И 10 выводы в высокоимпедансное состояние
pinMode(Left, INPUT);

pinMode(Bottom, OUTPUT); // вывод 9 устанавливаем как выход и  подтягиваем к земле
digitalWrite(Bottom, LOW);

pinMode(Top, OUTPUT);    // вывод 11 устанавливаем как выход и  подтягиваем к +5В
digitalWrite(Top, HIGH);

delay(3);
coordY = analogRead(rightInput);
```

Вы устанавливаете для логической переменной touch значение true, только если считанные значения больше 24 и меньше тысячи.Это необходимо для того, чтобы вы возвращали истинное значение только в том случае, если показания находятся в допустимых пределах.

```
if(coordX < 1000 && coordX > 24 && coordY < 1000 && coordY > 24) {touch = true;}
```

Вы обнаружите, что значения варьируются от примерно 100 на самом низком уровне до примерно 900 на верхнем. Наконец, вы возвращаете значение touch, которое будет false, если экран не нажимается, и true, если это так:

```
return touch;
```

Как видите, считывание значений с сенсорного экрана очень просто и позволяет использовать его в любых целях. Вы можете поместить изображение или диаграмму за экраном, относящуюся к кнопкам или другим элементам управления, или наложить экран на ЖК-дисплей, как в Nintendo DS, что позволит вам при необходимости изменить пользовательский интерфейс под экранами.Вы просто продемонстрируете это, распечатав клавиатуру, которую можно разместить под сенсорным экраном, и прочитав соответствующие значения, чтобы определить, какая клавиша была нажата.

Проект 34 - сенсорная клавиатура

Теперь вы разместите пользовательский интерфейс под сенсорным экраном в виде печатной клавиатуры и определите по точкам касания, какая клавиша была нажата. Как только вы поймете основы этого, вы можете перейти к замене печатной клавиатуры на клавиатуру, отображаемую на ЖК-дисплее или OLED-дисплее.Вы выведете нажатую клавишу на ЖК-дисплей, поэтому вам нужно будет добавить ее в список деталей для этого проекта.

Требуемые детали

Мы используем те же детали и схемы, что и в Project 33, с добавлением ЖК-дисплея 16 × 2.

Таб. 12-3. *Детали, необходимые для проекта 34*

Сенсорный экран Nintendo DS	
Модуль разъема	
ЖК-дисплей 16 × 2	
Резисторы 47K x 2	

Другое отличие состоит в том, что вы создадите и распечатаете клавиатуру для размещения под сенсорным экраном. Стандартный сенсорный экран DS имеет размер 70 мм × 55 мм (2,75 × 2,16 дюйма), поэтому вам нужно будет создать шаблон такого размера с помощью графического или текстового редактора, а затем поместить набор равномерно расположенных клавиш на прямоугольник, чтобы он напоминал клавиатуру телефона. На рис. 12-3 показана созданная мной клавиатура. Не стесняйтесь использовать это.

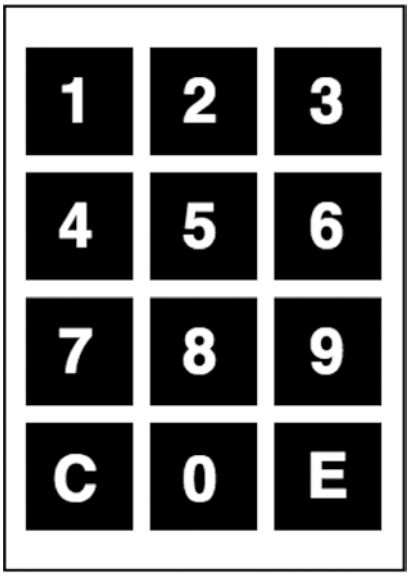


Рис. 12-3. *Схема клавиатуры для Project 34*

Подключение

Подключите как показано на рисунке 12-4.

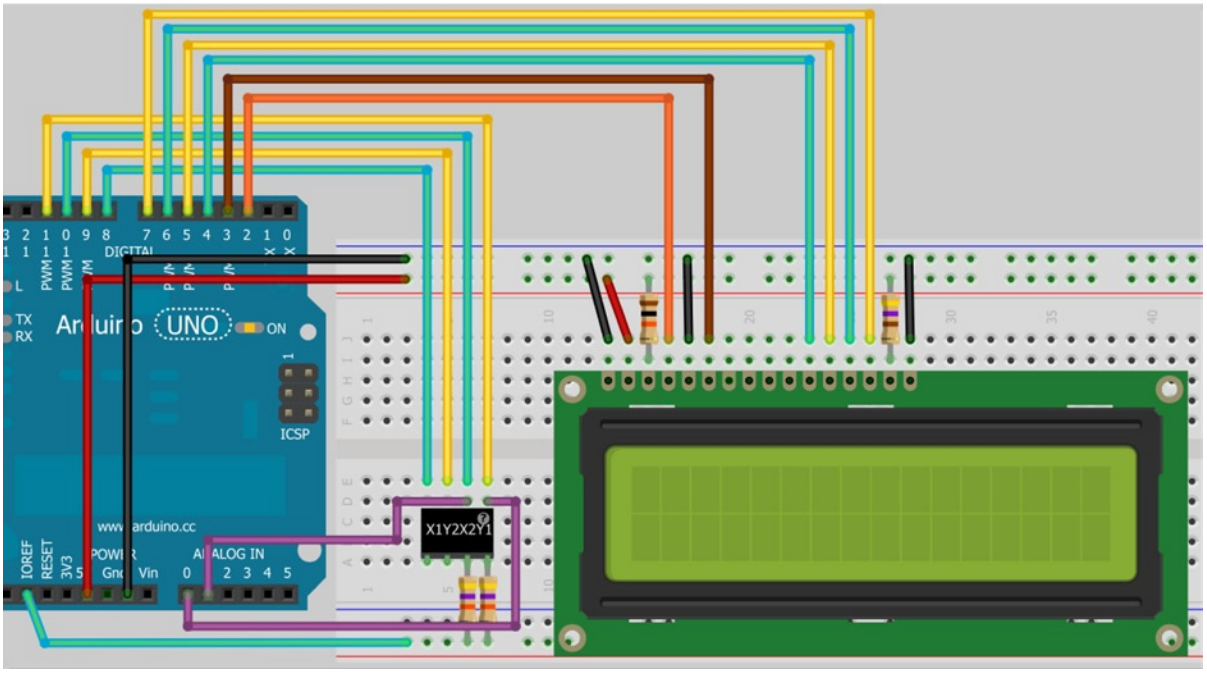


Рис. 12-4. *Схема для Project 34 - Touch Screen Keypad*

Обратитесь к Таблице 12-4 для получения информации о выводах для ЖК-дисплея.

Таб. 12-4. *Распиновка ЖК-дисплея в Project 34*

Arduino	Other	Matrix
Digital 2		RS (Register Select
Digital 3		Enable
Digital 4		DB4 (Data Pin 4)
Digital 5		DB5 (Data Pin 5)
Digital 6		DB6 (Data Pin 6)
Digital 7		DB7 (Data Pin 7)
	Gnd	Vss (GND)
	Gnd	R/W (Read/Write)
	+5V	Vdd
	+5V via resistor	Vo (Contrast)
	+5V via resistor	A/Vee (Power for LED)
	Gnd	Gnd for LED

Введите код

Введите код из листинга 12-2.

Листинг 12-2. Код для проекта 34

```
// Проект 34

#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // создать объект lcd и назначить пины

// Power connections
#define Left 8                      // влево (X1) на цифровой пин 8
#define Bottom 9                    // вниз (Y2) на цифровой пин 9
#define Right 10                    // вправо (X2) на цифровой пин 10
#define Top 11                      // верх (Y1) на цифровой пин 11

// Analog connections
#define topInput 0                  // Верх (Y1) на аналоговый пин 0
#define rightInput 1                // вправо (X2) на аналоговый пин 1

int coordX = 0, coordY = 0;
char buffer[16];
```

```
void setup()
{
  lcd.begin(16, 2);        // Устанавливаем отображение на 16 столбцов и 2 строки
  lcd.clear();
}

void loop()
{
  if (touch())
  {
    if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
    if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
    if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
    if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
    if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
    if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
    if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
    if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
    if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
    if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
    if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
    if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
    delay(250);
  }
}

// возвращаем TRUE-ИСТИНА при прикосновении и устанавливаем координаты touchX и touchY

{
        boolean touch = false;

                                    // получаем горизонтальные координаты

        pinMode(Top, INPUT);        // 9,11 пины в высокоимпедансное состояние
        pinMode(Bottom, INPUT);

        pinMode(Left, OUTPUT);
        digitalWrite(Left, LOW);    // вывод 8 устанавливаем как выход и  подтягиваем к земле

        pinMode(Right, OUTPUT);     // вывод 10 устанавливаем как выход и подтягиваем к +5В
        digitalWrite(Right, HIGH);

        delay(3);
        coordX = analogRead(topInput);

                                    // получаем вертикальные координаты

        pinMode(Right, INPUT);      // 8,10 пины переводим в высокоимпедансное состояние
        pinMode(Left, INPUT);

        pinMode(Bottom, OUTPUT);    // 9 пин подтягиваем к земле
        digitalWrite(Bottom, LOW);
```

```
        pinMode(Top, OUTPUT);        // 11 пин подтягиваем к +5В
        digitalWrite(Top, HIGH);


        delay(3);
        coordY = analogRead(rightInput);
                                     // если считанные координаты меньше 1000 и больше 24
                                     // значит прикосновение к экрану
        if(coordX < 1000 && coordX > 24 && coordY < 1000 && coordY > 24) {touch = true;}

          return touch;
}

void scrollLCD() {
  for (int scrollNum=0; scrollNum<16; scrollNum++) {
    lcd.scrollDisplayLeft();
    delay(100);
  }
  lcd.clear();
}
```

Введите код и загрузите его в свой Arduino. Вставьте шаблон клавиатуры под клавиатуру так, чтобы ленточный кабель находился справа внизу (рядом с буквой E). Теперь вы можете нажимать клавиши на сенсорном экране, и то, что вы нажимаете, отображается на ЖК-дисплее. Когда вы нажимаете кнопку C (для очистки), дисплей очищается. Когда вы нажимаете клавишу E (для ввода), отображаемые числа будут прокручиваться влево, пока не исчезнут.

Вы уже знаете, как работают ЖК-экран и сенсорный экран, поэтому я пропущу обзор оборудования в этом проекте и просто взгляну на код.

Проект 34 - Сенсорная клавиатура - Обзор кода

Вы начинаете с включения библиотеки LiquidCrystal и создания объекта lcd:

```
#include <LiquidCrystal.h>


LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // создаем ЖК-объект и назначаем контакты
```

На этот раз вы используете контакты 2 и 3 для RS и активируете их на ЖК-дисплее, а контакты с 4 по 7 - для линий данных. Далее определены контакты для сенсорного дисплея и инициализированы переменные X и Y:

```
// Соединения с цифровыми выводами
#define Left 8                       // влево (X1) на цифровой вывод 8
#define Bottom 9                     // вниз (Y2) к цифровому выводу 9
#define Right 10                     // вправо (X2) на цифровой вывод 10
#define Top 11                       // вверх (Y1) к цифровому выводу 11

// Соединения с аналоговыми выводами
#define topInput 0                   // Верх (Y1) к аналоговому выводу 0
#define rightInput 1                 // Вправо (X2) на аналоговый вывод 1

int coordX = 0, coordY = 0;
```

В процедуре настройки вы запускаете объект LCD и устанавливаете его на 16 столбцов, 2 строки, затем очищаете дисплей, чтобы вы были готовы начать:

```
lcd.begin(16, 2); // Устанавливаем отображение на 16 столбцов и 2 строки
lcd.clear();      //Очистка дисплея с установкой курсора в положение 0,0.
```

В основном цикле у нас есть оператор if, как и в проекте 33, но на этот раз вам нужно убедиться, что затронутые координаты находятся внутри прямоугольника, определяющего границу каждой кнопки. Если координата находится в пределах соответствующей границы кнопки, соответствующий номер отображается на ЖК-дисплее. Если нажата кнопка C, дисплей очищен; если кнопка E нажата, вызывается функция scrollLCD.

```
if (touch())
  {
    if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
    if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
    if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
    if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
    if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
    if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
    if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
    if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
    if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
    if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
    if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
    if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
    delay(250);
  }
```

Каждый оператор if представляет собой набор условных и логических операторов. Если вы посмотрите на заявление для кнопки три

```
if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
```

вы можете видеть, что первое логическое условие И проверяет, что позиция касания находится в пределах позиций 110 и 300 слева, а второе - в пределах позиций 170 и 360 сверху. Для нажатия кнопки должны быть выполнены все условия, поэтому используются логические операторы И (&&).Чтобы узнать координаты кнопки, просто осторожно нажмите острым стилусом на левую и правую стороны кнопки, чтобы получить координаты X. Затем повторите с верхней и нижней сторонами, чтобы получить координаты Y. Если вы используете Project 33 для печати координат на последовательном мониторе, вы можете использовать его для определения точных координат расположения ваших кнопок, если вам нужно скорректировать код или если вы хотите создать свой собственный макет кнопок.Далее идет сенсорная функция; вы уже знаете, как это работает. Наконец, есть функция scrollLCD, которая не возвращает никаких данных и не принимает никаких параметров, поэтому имеет тип void:

```
void scrollLCD() {
```

Затем у вас есть цикл for, который повторяется 16 раз, что является максимальным количеством символов, которые можно ввести и отобразить:

```
for (int scrollNum=0; scrollNum<16; scrollNum++) {
```

Внутри цикла for вы используете функцию scrollDisplayLeft () из библиотеки LiquidCrystal для прокрутки отображаемых символов на один пробел влево. После этого следует задержка в 100 миллисекунд.

```
lcd.scrollDisplayLeft();
delay(100);
```

После того, как это будет сделано 16 раз, введенные числа будут сдвигаться влево. Вы можете написать свои собственные процедуры, чтобы делать все, что вы хотите, с введенными данными.

Наконец, вы очищаете дисплей, чтобы убедиться, что он готов для новых данных, прежде чем выходить из функции обратно в основной цикл:

```
lcd.clear();
```

Этот проект дает вам представление о том, как отключить части сенсорного экрана, чтобы вы могли выбирать области для кнопок и т. Д.Бумагу можно заменить графическим ЖК-дисплеем или OLED-дисплеем, на котором вы можете рисовать кнопки. Преимущество этого заключается в том, что в зависимости от того, что выбрал пользователь, можно рисовать разные меню и разные кнопки. Используя эту технику, вы можете создать для своего проекта по-настоящему необычный пользовательский интерфейс с сенсорным экраном.Теперь мы перейдем к управлению светодиодом RGB,используя изображение ползунка вместо нажатия кнопок для управления цветами.

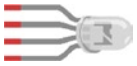
Project 35 - Контроллер цвета с сенсорным экраном

В этом проекте вы будете использовать сенсорный экран для включения и выключения светодиодной лампы RGB, а также для управления цветом светодиода

Требуемые детали

Вы будете использовать те же детали и схемы, что и в Project 33, с добавлением светодиода RGB. Светодиод RGB должен быть с общим катодом.

Таб. 12-5. *Детали, необходимые для проекта 35*

Сенсорный экран Nintendo DS	
Модуль разъема	
RGB LED (с общим катодом)	
Токоограничивающие резисторы x 3	
Резисторы 47k x 2	

Вам также понадобится шаблон клавиатуры, как в Project 34. На этот раз в нем должны быть области для ползунков цвета и кнопок включения / выключения. Не стесняйтесь использовать изображение на Рисунке 12-5.

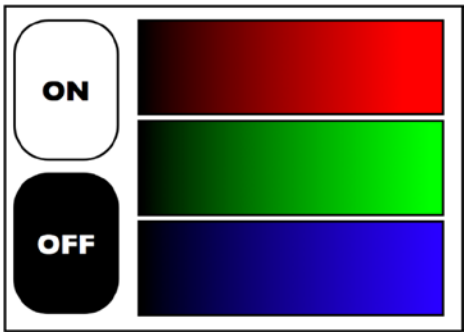


Рис. 12-5. *Схема клавиатуры для Project 35*

Подключение

Подключите все, как показано на Рисунке 12-6.

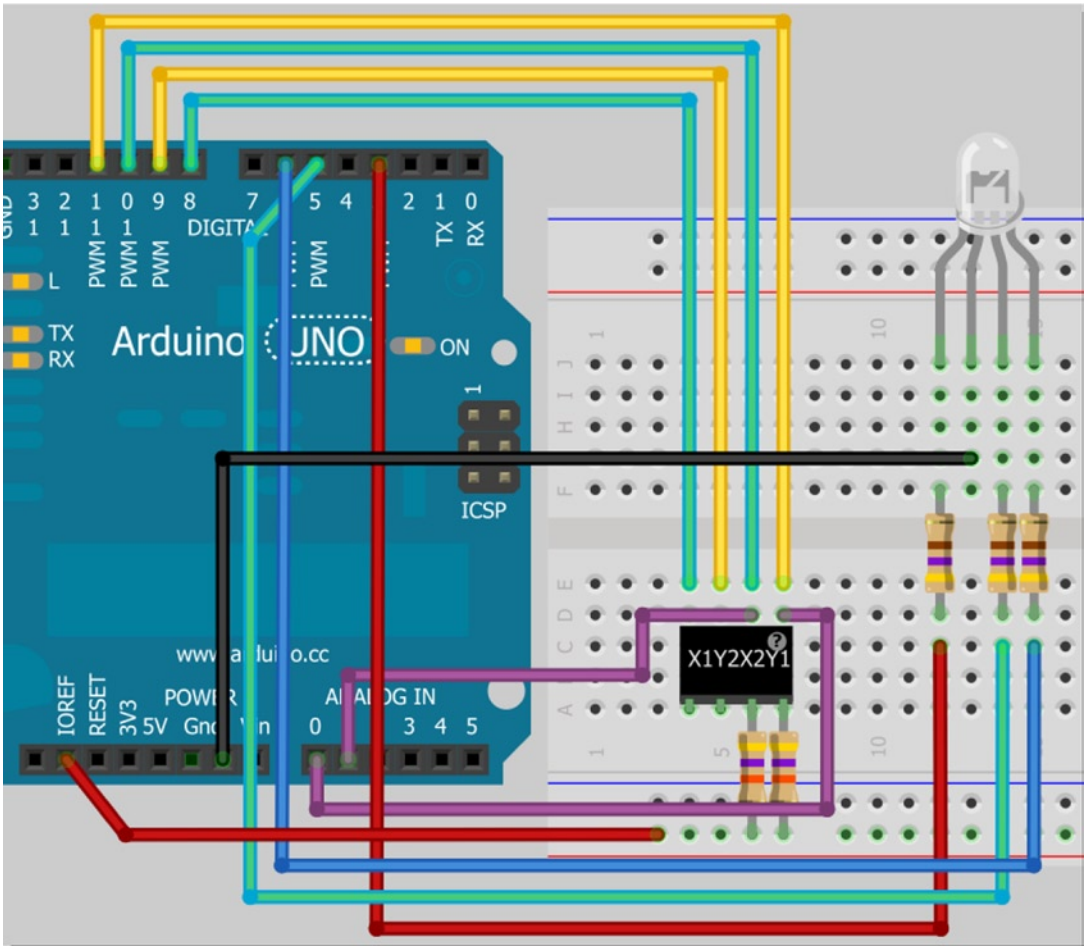


Рис. 12-6. *Схема для Project 35 - Touch Screen для управления цветом*

Земляной провод идет к катоду светодиода. Вывод 3 ШИМ подключается к красномуаноду, вывод 5 ШИМ - к зеленому аноду, а вывод 6 ШИМ - к синему аноду. Убедитесь, что к анодамподключены токоограничивающие резисторы.

Введите код

Введите код in Listing 12-3.

Listing 12-3. Code for Project 35

```
// Project 35

// Соединения с цифровыми выводами
#define Left 8                  // влево (X1) на цифровой вывод 8
#define Bottom 9                // вниз (Y2) к цифровому выводу 9
#define Right 10                // вправо (X2) на цифровой вывод 10
#define Top 11                  // вверх (Y1) к цифровому выводу 11

// Соединения с аналоговыми выводами
#define topInput 0              // TВерх (Y1) к аналоговому выводу 0
#define rightInput 1            // Вправо (X2) на аналоговый вывод 1

// пины RGB
#define pinR 3
#define pinG 5
#define pinB 6

int coordX = 0, coordY = 0;
boolean ledState = true;
int red = 100, green = 100, blue = 100;

void setup()
{
  pinMode(pinR, OUTPUT);
  pinMode(pinG, OUTPUT);
  pinMode(pinB, OUTPUT);
}

void loop()
{
        if (touch()) {
                if ((coordX>0 && coordX<270) && (coordY>0 && coordY<460))
                {
                    ledState = true; delay(50);
                }

                if ((coordX>0 && coordX<270) && (coordY>510 && coordY< 880))
                {
                    ledState = false; delay(50);
                }
```

```
                if ((coordX>380 && coordX<930) && (coordY>0 && coordY<300))
                {
                   red=map(coordX, 380, 930, 0, 255);
                }

                if ((coordX>380 && coordX<930) && (coordY>350 && coordY<590))
                {
                   green=map(coordX, 380, 930, 0, 255);
                }

                if ((coordX>380 && coordX<930) && (coordY>640 && coordY<880))
                {
                   blue=map(coordX, 380, 930, 0, 255);
                }

                delay(10);
}

        if (ledState) {
                analogWrite(pinR, red);
                analogWrite(pinG, green);
                analogWrite(pinB, blue);
        }
        else {
                analogWrite(pinR, 0);
                analogWrite(pinG, 0);
                analogWrite(pinB, 0);
        }
}

// вернуть TRUE-ИСТИНА при прикосновении и установить координаты на touchX и touchY

{
        boolean touch = false;

                                // получаем горизонтальные координаты

        pinMode(Top, INPUT);     // 9,11 пины в высокоимпедансное состояние
        pinMode(Bottom, INPUT);

        pinMode(Left, OUTPUT);
        digitalWrite(Left, LOW); // вывод 8 устанавливаем как выход и подтягиваем к земле

        pinMode(Right, OUTPUT);  // вывод 10 устанавливаем как выход и подтягиваем к +5В
        digitalWrite(Right, HIGH);


        delay(3);
        coordX = analogRead(topInput);

                                // получаем вертикальные координаты
```

```
        pinMode(Right, INPUT);   // 8,10 пины в высокоимпедансное состояние
        pinMode(Left, INPUT);

        pinMode(Bottom, OUTPUT); //  вывод 9 устанавливаем как выход и подтягиваем к земле
        digitalWrite(Bottom, LOW);

        pinMode(Top, OUTPUT);    // вывод 11 устанавливаем как выход и подтягиваемк +5В
        digitalWrite(Top, HIGH);


        delay(3);
        coordY = analogRead(rightInput);

                            // если считанные координаты меньше 1000 и больше 24
                            // значит прикосновение к экрану
        if(coordX < 1000 && coordX > 24 && coordY < 1000 && coordY > 24) {touch = true;}

          return touch;
}
```

Проект 35 - Контроллер сенсорного экрана - Обзор кода

Начальные определения такие же, как в проектах 33 и 34, с добавлением набора определений для трех выводов ШИМ, используемых для управления компонентами R, G и B светодиода RGB:

```
//пины RGB
#define pinR 3
#define pinG 5
#define pinB 6
```

Вы добавляете логическое значение ledState и устанавливаете для него значение true. Это логическое значение будет содержать состояние светодиодов, то есть true = on и false = off.

```
boolean ledState = true;
```

Объявляется набор из трех целых чисел, каждое из которых инициализируется значением 100:

```
int red = 100, green = 100, blue = 100;
```

Эти три целых числа будут содержать отдельные значения цвета светодиода. Они будут соответствовать выходным значениям ШИМ с контактов 3, 5 и 6.
В основной процедуре настройки все три вывода светодиода, которые вы определили, настроены на выходы:

```
pinMode(pinR, OUTPUT);
pinMode(pinG, OUTPUT);
pinMode(pinB, OUTPUT);
```

В основном цикле у вас снова есть оператор if, чтобы проверить, истинно ли значение, возвращаемое функцией touch (). Внутри есть больше операторов if, чтобы решить, какие части сенсорного экрана были нажаты. Первые два определяют границы кнопок ON и OFF и изменяют ledState на true, если касание обнаружено в пределах границы кнопки ON, и на false, если оно находится в пределах кнопки OFF. После этого включена небольшая задержка, чтобы предотвратить ложные показания с кнопок.

```
if ((coordX>0 && coordX<270)
        && (coordY>0 && coordY<460))
                {ledState = true; delay(50);}

if ((coordX>0 && coordX<270)
        && (coordY>510 && coordY< 880))
                {ledState = false; delay(50);}
```

Затем вы проверяете, было ли обнаружено касание в границах областей ползунка для красного, зеленого и синего элементов управления. Если касание было обнаружено, значение красного, зеленого или синего целого числа изменяется в соответствии с тем, какая часть ползунка была затронута.

```
if ((coordX>380 && coordX<930)
        && (coordY>0 && coordY<300))
                {red=map(coordX, 380, 930, 0, 255);}

if ((coordX>380 && coordX<930)
        && (coordY>350 && coordY<590))
                {green=map(coordX, 380, 930, 0, 255);}

if ((coordX>380 && coordX<930)
        && (coordY>640 && coordY<880))
                {blue=map(coordX, 380, 930, 0, 255);}
```

Это можно сделать с помощью функции map (), которая принимает пять параметров. Первая - это проверяемая вами переменная, за которой следуют верхний и нижний диапазоны переменной (все остальные игнорируются). Последние два параметра - это верхний и нижний диапазоны, которым вы хотите сопоставить значения. Другими словами, вы берете координаты X в области ползунка и сопоставляете это значение от 0 в крайнем левом углу ползунка до 255 в крайнем правом углу. Двигая пальцем слева направо, вы можете изменить соответствующий цветовой компонент с 0 при его самой тусклой яркости (будет как выключение), до 255 при максимальной яркости.

Наконец, у вас есть еще один оператор if для установки значений ШИМ контактов R, G и B на соответствующие значения, хранящиеся в красном, зеленом и синем цветах, но только если ledState истинно. Оператор else устанавливает все значения PWM на 0 или отключает, если ledState имеет значение false.

```
if (ledState) {
        analogWrite(pinR, red);
        analogWrite(pinG, green);
        analogWrite(pinB, blue);
}
else {
        analogWrite(pinR, 0);
        analogWrite(pinG, 0);
        analogWrite(pinB, 0);
}
```

Остальная часть программы - это уже описанная функция touch ().

Резюме

В Project 35 представлены понятия кнопок и ползунков, управляющих сенсорным экраном. Опять же, использование дисплея GLCD или OLED даст вам больший контроль над системой освещения. Project 35 относительно легко можно расширить для управления настенным RGB-освещением вокруг дома, заменив стандартные выключатели на цветные OLED-дисплеи и СЕНСОРНЫЕ ДИСПЛЕИ для универсального управления освещением.

Глава 12 показала, что резистивные СЕНСОРНЫЕ ДИСПЛЕИ очень легко взаимодействовать с Arduino и
использовать их. С помощью всего лишь короткой и простой программы сенсорный экран и Arduino могут обеспечить гибкость управления пользователем. В сочетании с графическими дисплеями сенсорный экран становится очень полезным инструментом для управления системами.

- Как использовать модуль коммутации, чтобы упростить взаимодействие с нестандартными разъемами
- Как работает резистивный сенсорный экран
- Правильный цикл измерения мощности и напряжения для получения координат X и Y
- *Значение высокого импеданса*
- Эти СЕНСОРНЫЕ ДИСПЛЕИ можно накладывать на графические дисплеи для создания интерактивных кнопок
- Как определить область кнопки с помощью координат и логических операторов И
- Как области сенсорного экрана можно разделить на кнопки или ползунки

ГЛАВА 13

Датчики температуры

Два проекта в этой главе продемонстрируют, как подключить аналоговые и цифровые датчики температуры к Arduino и как получать с них показания. Датчики температуры часто используются в проектах Arduino, от метеостанций до пивоварения и проектов на высотных воздушных шарах. Мы используем два датчика - аналоговый датчик LM335 и цифровой DS18B20.

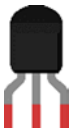
Проект 36 - Последовательный датчик температуры

В этом проекте используется аналоговый датчик температуры LM335. Этот датчик является частью линейки датчиков LM135 от National Semiconductors. Он имеет диапазон от -40 ° C до + 100 ° C (от -40 ° F до + 212 ° F) и поэтому идеально подходит, например, для использования на метеостанции.

Требуемые детали

Схема и код разработаны для датчика LM335, но вы можете так же легко заменить LM135 или LM235, если хотите. Вам нужно будет настроить свой код в соответствии с соответствующим датчиком. Подстроечный потенциометр 4,7 кОм можно заменить стандартным поворотным потенциометром аналогичного значения. Подстроечный потенциометр - это просто небольшой потенциометр, предназначенный для регулировки или подстройки части схемы. Подойдет любой подстроечный резистор или потенциометр со значением от 4,7 кОм до 10 кОм.

Таб. 13-1. *Детали, необходимые для проекта 36*

Датчик температуры LM335	
Подстроечный потенциометр 4,7К	
Резистор 2,2 кОм	

Подключение

Подключите как показано на Рисунке 13-1.

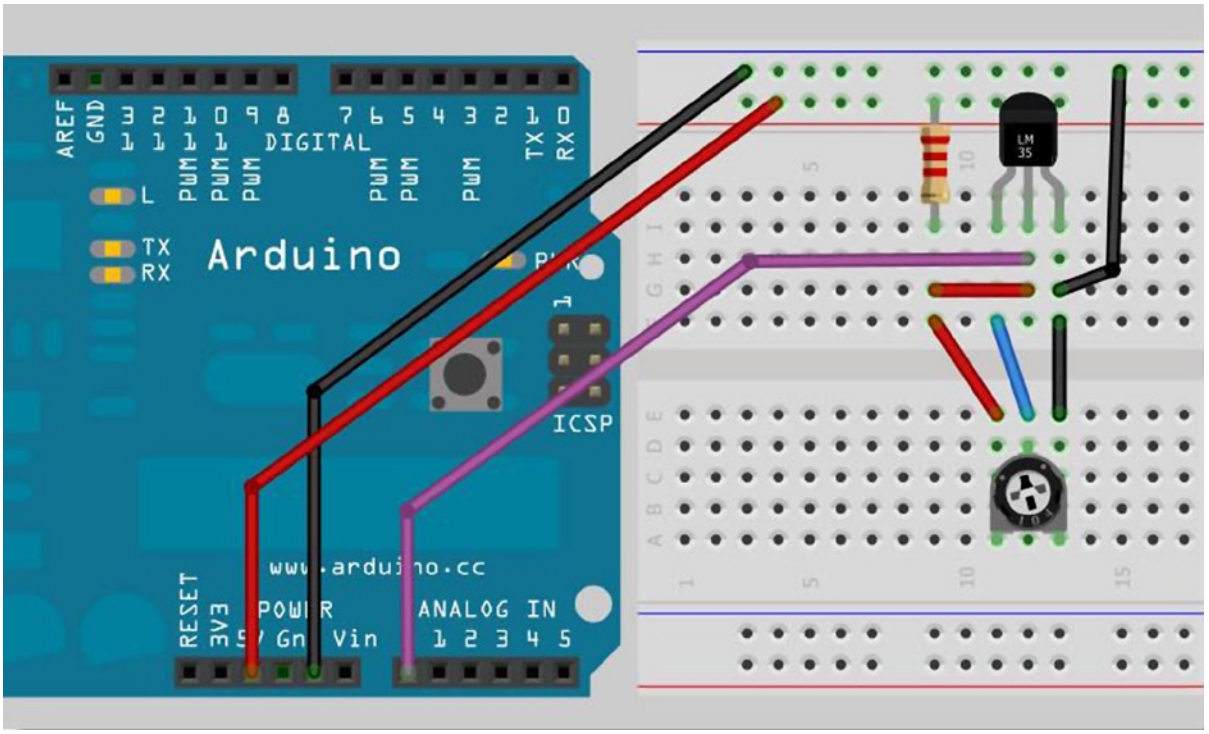


Рис. 13-1. *TCхема для проекта 36 - Последовательный датчик температуры*

См. Рисунок 13-2, где представлен чертеж из таблицы данных National Semiconductor. Центральный вывод идет к аналоговому выводу 0 на Arduino. ADJ - вывод управления; + и - питание

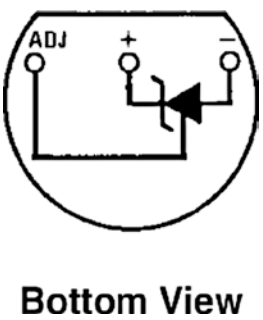


Рис. 13-2. *Распиновка датчика температуры LM335*

Введите код

Введите код из листинга 13-1.

Listing 13-1. Код для проекта 36

```
// Проект 36

#define sensorPin 0

float Celsius, Fahrenheit, Kelvin;
int sensorValue;

void setup() {
Serial.begin(9600);
Serial.println("Initialising.....");
}

void loop() {

  GetTemp();
  Serial.print("Celsius: ");
  Serial.println(Celsius);
  Serial.print("Fahrenheit: ");
  Serial.println(Fahrenheit);
  Serial.println();
  delay(2000);
}
void GetTemp()
{
  sensorValue = analogRead(sensorPin);                // читаем датчик
  Kelvin = (((float(sensorValue) / 1023) * 5) * 100); // конвертируем в Кельвин
  Celsius = Kelvin - 273.15;                          // конвертируем в градусы Цельсия
  Fahrenheit = (Celsius * 1.8) +32;                   // конвертируем в градусы Фаренгейта
}
```

Введите код и загрузите его в свой Arduino. Как только код будет запущен, откройте МПП и убедитесь, что ваша скорость передачи установлена на 9600. Вы увидите температуру, отображаемую как в градусах Фаренгейта, так и в градусах Цельсия. Температура может показаться вам неправильной. Вот тут-то и пригодится триммер; сначала необходимо откалибровать датчик. Для правильной калибровки вы должны использовать смесь льда и воды, которая успела стабилизироваться при температуре, при которой лед тает.

Поместите колотый или колотый лед в чашку из пенополистирола (чтобы ограничить влияние внешних факторов) и либо дайте ему оттаять, пока он частично не растает (в холодильнике), либо добавьте немного чистой (в идеале дистиллированной) воды. В смеси должно быть не менее 50 процентов льда.

Перемешайте и подождите не менее нескольких минут, чтобы вода успела остыть до точки замерзания. Затем поместитедатчик (защищенный тонким полиэтиленовым пакетом с минимальным количеством воздуха) в суспензию из водяного льда и подождите, пока показания перестанут меняться.

Затем настройте подстроечником на 0 ° C. Теперь поворачивайте ползунок подстроечника, пока показания на МПП не покажут правильную температуру. Теперь ваш датчик откалиброван. Хорошо бы использовать термоусадочную трубку для гидроизоляции датчика.

Вы можете удалить триммерную часть схемы, и она будет работать нормально. Однако температура будет приблизительно равна 1 ° C. Принцип работы датчика довольно таки сложен,поэтому просто рассмотрим код для этого проекта.

Проект 36 - Последовательный датчик температуры - Обзор кода

Код этого проекта короткий и простой. Вы начинаете с определения пина для подключения датчика. В этом случае вы используете аналоговый вывод 0.

```
#define sensorPin 0
```

Затем вам понадобятся некоторые переменные для хранения значений температуры в градусах Цельсия, Фаренгейта и Кельвина. Поскольку вы хотите иметь возможность представлять доли градуса, вы используете переменные типа float.

```
float Celsius, Fahrenheit, Kelvin;
```

Затем вы создаете целое число для хранения значения, считанного с аналогового вывода:

```
int sensorValue;
```

Функция настройки начинает последовательную связь со скоростью 9600 бод:

```
Serial.begin(9600);
```

Затем появляется надпись «Initializing.....Инициализация », чтобы показать, что программа вот-вот запустится:

```
Serial.println("Initializing.....");
```

В основном цикле программы вы вызываете функцию GetTemp (), которая считывает температуру с датчика и преобразует ее в градусы Цельсия и Фаренгейта. Затем он распечатывает температуру в окне МПП:

```
GetTemp();
Serial.print("Celsius: ");
Serial.println(Celsius);
Serial.print("Fahrenheit: ");
Serial.println(Fahrenheit);
Serial.println();
```

Теперь мы создаем функцию GetTemp ():

```
void GetTemp()
```

Сначала считывается датчик и значение сохраняется в sensorValue:

```
    sensorValue = analogRead(sensorPin); // читаем датчик
```

Выходной сигнал датчика выражается в градусах Кельвина, где каждые 10 мВ равны одному К. Кельвин начинается с нуля градусов Кельвина, когда температура равна абсолютному нулю или самой низкой возможной температуре во Вселенной. Таким образом, при абсолютном нуле датчик будет выдавать 0 вольт. Согласно техническому описанию датчик можно откалибровать, убедившись, что напряжение на датчике составляет 2,98 В при 25 ° C. Чтобы преобразовать градусы Кельвина в градусы Цельсия, мы просто вычитаем 273,15 из температуры Кельвина.чтобы получить температуру по Цельсию. Итак, 25 ° C по Кельвину составляет 298,15, и если каждый градус равен 10 мВ, то вы просто перемещаете десятичную запятую на две позиции влево, чтобы получить напряжение при этой температуре, которое на самом деле составляет 2,98 вольт.

Итак, чтобы получить температуру в Кельвинах, вы считываете значение с датчика, которое будет в диапазоне от 0 до 1023, а затем делите его на 1023 и умножаете результат на 5. Это эффективно отображает диапазон от 0 до 5 вольт. . Поскольку каждая степень K равна 10 мВ, вы затем умножаете этот результат на 100, чтобы получить градусы K.

```
Kelvin = (((float(sensorValue) / 1023) * 5) * 100); // конвертируем в Кельвин
```

Значение датчика является целым числом, поэтому оно преобразуется в число с плавающей запятой, чтобы гарантировать, что результат тоже будет с плавающей точкой.Теперь, когда у вас есть значение в K, его легко преобразовать в градусы Цельсия и Фаренгейта. Чтобы преобразовать в градусы Цельсия, вычтите 273,15 из температуры в K:

```
Celsius = Kelvin - 273.15;                    // конвертируем в градусы Цельсия
```

And to convert to Fahrenheit, multiply the Celsius value by 1.8 and add 32:

```
Fahrenheit = (Celsius * 1.8) +32;             // конвертируем в градусы Фаренгейта
```

Датчики серии LM135 хороши тем, что их можно легко откалибровать, поэтому вы можете каждый раз получать точные показания. Они также дешевы, поэтому вы можете купить их целую кучу и получать показания из разных частей вашего дома или внутреннюю и внешнюю температуру с помощью проекта высотного воздушного шара.Могут использоваться другие аналоговые датчики. Вы можете обнаружить, что третий вывод adj на некоторых датчиках является регулирующим выводом в LM335 для снятия температуры. Следовательно, вы должны использовать этот третий вывод для считывания температуры вместо вывода напряжения питания. Калибровку датчиков такого типа можно легко выполнить с помощью программы.

Далее вы рассмотрим цифровой датчик температуры. Безусловно, наиболее популярным из этих типов является DS18B20 от Dallas Semiconductor (Maxim).

Проект 37 - Цифровой датчик температуры однопроводной

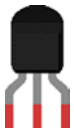
. Эти датчики температуры DS18B20 отправляют температуру в виде последовательного потока данных по одному проводу, поэтому протокол называется однопроводным. Каждый датчик также имеет уникальный серийный номер, что позволяет вам запрашивать разные датчики, используя их идентификационный номер. В результате вы можете подключить несколько датчиков к одной линии передачи данных. Это делает их очень популярными для использования с Arduino, потому что практически неограниченное количество датчиков температуры может быть последовательно соединено вместе и все подключено только к одному выводу на Arduino. Температурный диапазон также широк - от -55 ° C до + 125 ° C.

В этом проекте вы будете использовать два датчика, чтобы продемонстрировать не только, как подключать и использовать этот тип датчика, но и как последовательно соединять два или более датчика.

Требуемые детали

Вам понадобятся два датчика DS18B20 в формате TO-92 (см. таб. 13.2). Некоторые из них имеют маркировку DS18B20 +, что означает, что они не содержат свинца.

Таб. 13-2. *Детали, необходимые для проекта 37*

2 × датчик температуры DS18B20	
Резистор 4,7 кОм	

Подключение

Подключите как показано на рисунке 13-3.

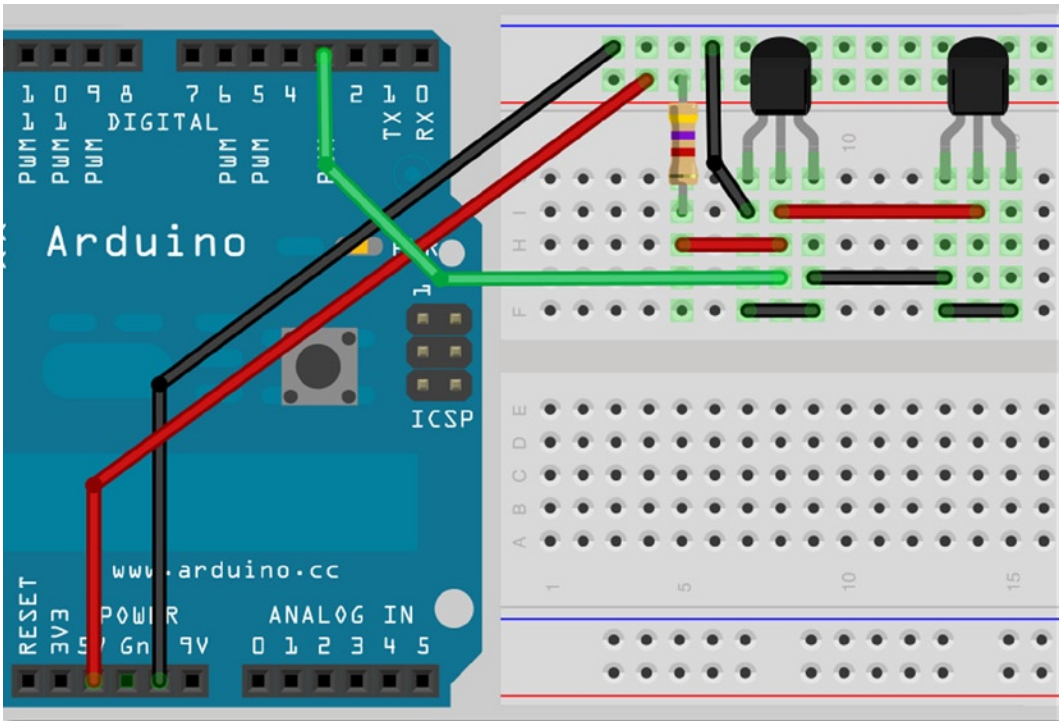


Рис. 13-3. *Схема для проекта 37 - однопроводной цифровой датчик температуры*

Я собираюсь разделить код на две части. Первая часть узнает адреса двух датчиков. Как только вы узнаете эти адреса, вы перейдете ко второй части, в которой адреса будут использоваться для получения значений температуры непосредственно от датчиков.

На рис. 13-4 показана распиновка датчика DS18B20. Устройство можно запитать двумя разными способами. Вы можете либо подать питание на вывод V_{DD} в диапазоне от 3,3 до 5 В, либо использовать так называемый «паразитный режим» и «украсть» питание от однопроводной шины через вывод DQ, когда на шине высокий уровень. В этом случае вывод VDD подключен к земле.

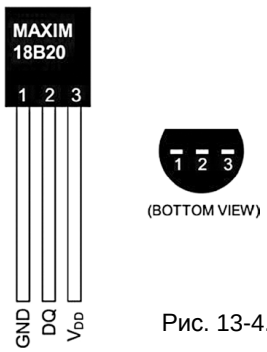


Рис. 13-4. Распиновка DS18B20

Одним из недостатков однопроводной шины является то, что она зависит от точной синхронизации. Поэтому жизненно важно, чтобы ваш процессор обменивался данными с устройством one-Wire только во время связи и не выполнял другие задачи одновременно (например, через прерывания).

Введите код

Перед тем, как ввести код, вам необходимо скачать и установить две библиотеки. Первая - это библиотека OneWire. Эта библиотека может использоваться для связи с любым однопроводным устройством. Поместите его в папку «библиотеки» вашей установки Arduino.

Затем загрузите и установите библиотеку DallasTemperature с http://milesburton.com/index.php? title = Dallas_Temperature_Control_Library и снова установите его в папку «библиотеки». После распаковки каталога в имени будут пробелы. Arduino не примет это, поэтому измените имя папки на DallasTemperature, прежде чем помещать ее в папку с библиотеками. Эта библиотека является ответвлением библиотеки OneWire и была разработана Майлзом Бертоном, с последующими улучшениями, внесенными несколькими другими участниками проекта. Этот проект основан на коде из примеров, включенных в эту библиотеку.

После того, как вы установили обе библиотеки, перезапустите IDE Arduino и затем введите код из программы в листинге 13-2.

Листинг 13-2. Код проекта 37 (часть 1)

```
// Проект 37 - Часть 1

#include <OneWire.h>
#include <DallasTemperature.h>
```

```
// Линия данных идет на цифровой вывод 3
#define ONE_WIRE_BUS 3

// SНастройте экземпляр oneWire для связи с любыми
// устройствами OneWire (а не только с температурными ИС Maxim / Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств
DeviceAddress insideThermometer, outsideThermometer;

void setup()
{
  // запускаем последовательный порт
  Serial.begin(9600);

  // Запускаем библиотеку
  sensors.begin();

  // находим устройства на шине
  Serial.print("Locating devices...");
  Serial.print("Found ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" devices.");

  if (!sensors.getAddress(insideThermometer, 0))
 {
     Serial.println("Unable to find address for Device 0");
  }

  if (!sensors.getAddress(outsideThermometer, 1))
 {
     Serial.println("Unable to find address for Device 1");
  }

  // выводим адреса обоих устройств
  Serial.print("Device 0 Address: ");
  printAddress(insideThermometer);
  Serial.println();

  Serial.print("Device 1 Address: ");
  printAddress(outsideThermometer);
  Serial.println();
  Serial.println();
}
```

```
// функция для печати адреса устройства
void printAddress(DeviceAddress deviceAddress)
{
  for (int i = 0; i < 8; i++)
  {
    // обнулить адрес, если необходимо
    if (deviceAddress[i] < 16) Serial.print("0");
    Serial.print(deviceAddress[i], HEX);
  }
}

// функция для печати температуры для устройства
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print("Temp C: ");
  Serial.print(tempC);
  Serial.print(" Temp F: ");
  Serial.print(DallasTemperature::toFahrenheit(tempC));
}

// основная функция для вывода информации об устройстве
void printData(DeviceAddress deviceAddress)
{
  Serial.print("Device Address: ");
  printAddress(deviceAddress);
  Serial.print(" ");
  printTemperature(deviceAddress);
  Serial.println();
}

void loop()
{
  // вызываем sensor.requestTemperatures () для выдачи глобальной температуры
  // запрос ко всем устройствам на шине
  Serial.print("Requesting temperatures...");
   sensors.requestTemperatures();
  Serial.println("DONE");
  delay(750);

  // выводим информацию об устройстве
  printData(insideThermometer);printData
  (outsideThermometer);Serial.println();

  delay(1000);
}
```

После загрузки кода откройте монитор последовательного порта. У вас будет экран, похожий на этот:

```
Locating devices...Found 2 devices.

Device 0 Address: 28CA90C202000088
Device 1 Address: 283B40C202000093

Requesting temperatures...DONE
Device Address: 28CA90C202000088 Temp C: 31.00 Temp F: 87.80
Device Address: 283B40C202000093 Temp C: 25.31 Temp F: 77.56
```

Программа дает вам два уникальных идентификационных номера используемых вами датчиков DS18B20. Вы можете узнать, где какой датчик, нагревая рукой одного из них. Я держался за правый датчик в течение нескольких секунд, и, как видите, температура на нем повысилась. Это говорит мне, что правильный датчик имеет адрес28CA90C202000088 а у левого адрес283B40C202000093. Адреса ваших датчиков, очевидно, будут отличаться. Запишите их или скопируйте и вставьте в текстовый редактор.Теперь, когда вы знаете идентификационные номера двух устройств, можно переходить ко второй части.

Введите код из Листинга 13-3.

Листинг 13-3. Код проекта 37 (часть 2)

```
// Проект 37 - Часть 2

#include <OneWire.h>
#include <DallasTemperature.h>

Провод данных подключен к выводу 3 на Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

// Настройте экземпляр oneWire для связи с любыми устройствами OneWire
// (а не только с температурными ИС Maxim / Dallas)
OneWire oneWire(ONE_WIRE_BUS);

// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств - замените адресами ваших датчиков
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };

void setup()
{
  // запускаем последовательный порт
  Serial.begin(9600);

  // Запускаем библиотеку
  sensors.begin();

  Serial.println("Initialising...");
  Serial.println();
```

```
// устанавливаем разрешение
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

// функция  печати температуры для устройства
void printTemperature(DeviceAddress deviceAddress)
{
  float tempC = sensors.getTempC(deviceAddress);
  Serial.print(" Temp C: ");
  Serial.print(tempC);
  Serial.print("  Temp F: ");
  Serial.println(DallasTemperature::toFahrenheit(tempC));
}

  // Измеряем и затем выводим значения температуры на последовательный порт
выводаvoid loop()
{
  sensors.requestTemperatures();
  delay(750);
  Serial.print("Inside Temp:");
  printTemperature(insideThermometer);
  Serial.print("Outside Temp:");
  printTemperature(outsideThermometer);
  Serial.println();
  delay(3000);
}
```

Замените два адреса датчика на те, которые вы обнаружили с помощью кода из первой части, а затем загрузите этоткод. Откройте монитор последовательного порта (МПП), и вы увидите следующее:

```
Initialising...

Inside Temp: Temp C: 24.25  Temp F: 75.65
Outside Temp: Temp C: 19.50  Temp F: 67.10

Inside Temp: Temp C: 24.37  Temp F: 75.87
Outside Temp: Temp C: 19.44  Temp F: 66.99

Inside Temp: Temp C: 24.44  Temp F: 75.99
Outside Temp: Temp C: 19.37  Temp F: 66.87
```

Если вы припаяете датчик наружной температуры к длинному сдвоенному проводу (припаяйте контакты 1 и 3 вместе для одного провода и контакт 2 для второго провода), а затем сделаете его водонепроницаемым, запечатав его в термоусадочную трубку, его можно разместить снаружи, чтобы измерять наружную температуру. Другой датчик может измерять температуру в помещении.

Проект 37 - Цифровой датчик температуры 1-Wire - Обзор кода

Сначала включим две библиотеки:

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Затем определяется цифровой контакт, который вы будете использовать для чтения данных с датчиков.

```
#define ONE_WIRE_BUS 3
```

за которым следует определение требуемой точности в битах

```
#define TEMPERATURE_PRECISION 12
```

Точность может быть установлена в пределах от 9 до 12 бит. Это соответствует шагам 0,5 ° C, 0,25 ° C, 0,125 ° C и0,0625 ° C соответственно. Разрешение по умолчанию - 12 бит. Максимальное разрешение 12 бит дает наименьшееприращение температуры, но за счет скорости. При максимальном разрешении датчику требуется 750 мс дляпреобразования температуры. При 11 битах это вдвое меньше, чем при 385 мс; 10 бит - это снова половина при 187,5 мс; и, наконец, 9 бит - это половина снова на 93,75 мс. 750 мс достаточно быстро для большинства целей. Однако, если по какой-либо причине вам необходимо снимать несколько показаний температуры в секунду, то разрешение в 9 бит даст самое быстрое время преобразования.

Затем вы создаете экземпляр объекта `OneWire` и называете его `oneWire`:

```
OneWire oneWire(ONE_WIRE_BUS);
```

Вы также создаете экземпляр объекта DallasTemperature, называете его датчиками и передаете ему ссылку на объект с именем oneWire:

```
DallasTemperature sensors(&oneWire);
```

Далее вам нужно создать массивы, которые будут содержать адреса датчиков. Библиотека DallasTemperature определяет переменные типа DeviceAddress (которые представляют собой простобайтовые массивы из восьми элементов). Мы создаем две переменные типа DeviceAddress, вызываем их для внутреннего и наружного термометров и назначаем адреса, найденные в первой части, массивам.

Просто возьмите адреса, которые вы нашли в первой части, разделите их на единицы из двух шестнадцатеричныхцифр и добавьте 0x (чтобы сообщить компилятору, что это шестнадцатеричное число, а не стандартное десятичное), и разделите их запятыми.
Адрес будет разбит на восемь блоков по две цифры в каждой.

```
DeviceAddress insideThermometer = { 0x28, 0xCA, 0x90, 0xC2, 0x2, 0x00, 0x00, 0x88 };
DeviceAddress outsideThermometer = { 0x28, 0x3B, 0x40, 0xC2, 0x02, 0x00, 0x00, 0x93 };
```

В функции настройки вы начинаете последовательную связь со скоростью 9600 бод:

```
Serial.begin(9600);
```

Далее запускается связь с объектом датчиков с помощью команды begin ():

```
sensors.begin();
```

Вы печатаете « Initialising - Инициализация. . . ”, чтобы показать, что программа запущена, после чего следует пустая строка:

```
Serial.println("Initialising...");
Serial.println();
```

Затем вы устанавливаете разрешение каждого датчика с помощью команды setResolution. Для этой команды требуются два параметра, первый из которых - адрес устройства, а второй - разрешение. Вы уже установили разрешение в начале программы на 12 бит.

```
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Затем вы создаете функцию с именем printTemperature (), которая будет распечатывать температуру в градусах Цельсия и Фаренгейта из адреса датчика, установленного в его единственном параметре:

```
void printTemperature(DeviceAddress deviceAddress)
```

Затем вы используете команду getTempC (), чтобы получить температуру в градусах Цельсия с указанного адреса устройства.

Вы сохраняете результат в float под названием tempC.

```
float tempC = sensors.getTempC(deviceAddress);
```

Затем вы распечатываете эту температуру

```
Serial.print(" Temp C: ");
Serial.print(tempC);
```

затем температура в градусах Фаренгейта

```
Serial.print("  Temp F: ");
Serial.println(DallasTemperature::toFahrenheit(tempC));
```

Вы используете :: для доступа к функции toFahrenheit, которая находится внутри библиотеки DallasTempera,которая преобразует значение tempC в градусы Фаренгейта.
В основном цикле вы просто запрашиваете температуру с помощью функции sensor.requestTemperatures () с последующей задержкой в 750 мс, чтобы датчик мог считывать температуру и передавать ее в Arduino. Затем можно безопасно вызвать функцию printTemperature () дважды, передавая адрес внутреннего, а затем внешнего датчика каждый раз с трехсекундной задержкой. Без трехсекундных задержек Arduino не будет успевать обрабатывать данные с датчиков:

```
sensors.requestTemperatures();
delay(750);

Serial.print("Inside Temp:");
printTemperature(insideThermometer);
Serial.print("Outside Temp:");
printTemperature(outsideThermometer);
Serial.println();
delay(3000);
```

Я рекомендую вам попробовать различные примеры, которые поставляются с библиотекой DallasTemperature, поскольку они дадут лучшее понимание различных функций, доступных в библиотеке. Я также рекомендую ознакомиться с даташитом на DS18B20. В этом датчике также могут быть установлены сигналы тревоги, которые срабатывают при соблюдении определенных температурных условий, которые могут быть полезны для определения слишком высоких или холодных условий.

DS18B20 - очень универсальный датчик, который имеет широкий диапазон измерения температурыи имеет преимущество перед аналоговым датчиком в том, что многие из них могут быть последовательно подключены к одной и той же линии передачи данных; таким образом,нужен только один вывод независимо от того, сколько у вас датчиков.

Далее вы познакомитесь с датчиком совершенно другого типа, который использует звуковые волны.

Резюме

В итоговой главе вы проработали два простых проекта, которые показали вам, как подключить аналоговые и цифровые датчики температуры к вашему Arduino. Проекты показали вам основы чтения данных с каждого датчика и отображения их на последовательном мониторе. Если вы знаете, как это сделать, сравнительно легко отобразить эти данные на жидкокристаллическом или точечно-матричном дисплее.
Знание того, как получать показания температуры с датчиков, открывает для энтузиастов Arduino целый ряд новых проектов. Вы вернетесь к датчикам температуры позже в книге, когда они будут применены на практике в главе 17.

Предметы и понятия, затронутые в главе 13

- Как подключить аналоговый датчик температуры к Arduino
- Как использовать триммер для калибровки датчика серии LM135
- Как преобразовать напряжение с датчика в Кельвин
- Как преобразовать Кельвин в Цельсий и Цельсий в Фаренгейт
- Как гидроизолировать датчики с помощью термоусадочной трубки
- Как подключить однопроводной датчик температуры к Arduino
- Однопроводные устройства могут быть подключены гирляндой
- Эти однопроводные устройства имеют уникальные идентификационные номера
- Как установить разрешение датчика DS18B20
- Чем выше разрешение, тем меньше скорость преобразования

ГЛАВА 14

Ультразвуковые дальномеры

Мы рассмотрим датчик другого типа, который часто используется в робототехнике и промышленных приложениях. Ультразвуковой дальномер предназначен для определения расстояния до объекта, посылая до него ультразвуковой импульс и отслеживая время, необходимое для его возвращения. Вы используем популярный тип ультразвукового дальномера из линейки датчиков Maxbotix LV-MaxSonar, но понятия, изученные в этой главе, могут быть применены к ультразвуковому дальномеру любой другой марки. Сначала мы изучим основы подключения датчика к Arduino, а затем перейдете к использованию датчика.

Проект 38 - Простой ультразвуковой дальномер

Ультразвуковой дальномер LV-MaxSonar выпускается в моделях EZ1, EZ2, EZ3 и EZ4. Характеристики датчиков серии MaxBotix LV MaxSonar EZ выглядят одинаково, и все они тоже похожи. Большая разница заключается в форме и способностях обнаружения ультразвукового луча.

EZ0 имеет самую широкую ширину луча и хорошо подходит для обнаружения препятствий, таких как люди или небольшие объекты, для предотвращения столкновений, но широкий луч означает, что вы не можете с большой точностью определить, где находится объект.

EZ4 имеет самый узкий и наименее чувствительный луч, поэтому он лучше подходит для более точного обнаружения таких функций, как местоположение двери, через которую нужно пройти, или местоположение других объектов для картирования.

EZ1, EZ2 и EZ3 находятся между этими крайностями и предлагают хороший компромисс функций для обнаружения или определения местоположения объекта.

При создании этой главы я использовал EZ3.

Требуемые детали

Таб. 14-1. *Требуемые детали для проекта 38*

LV-MaxSonar EZ3*	
Электролитический конденсатор 100 мФ	
Резистор 100 Ом	

** или любой из диапазона LV*

Подключение

Подключите все, как показано на рис. 14-1.

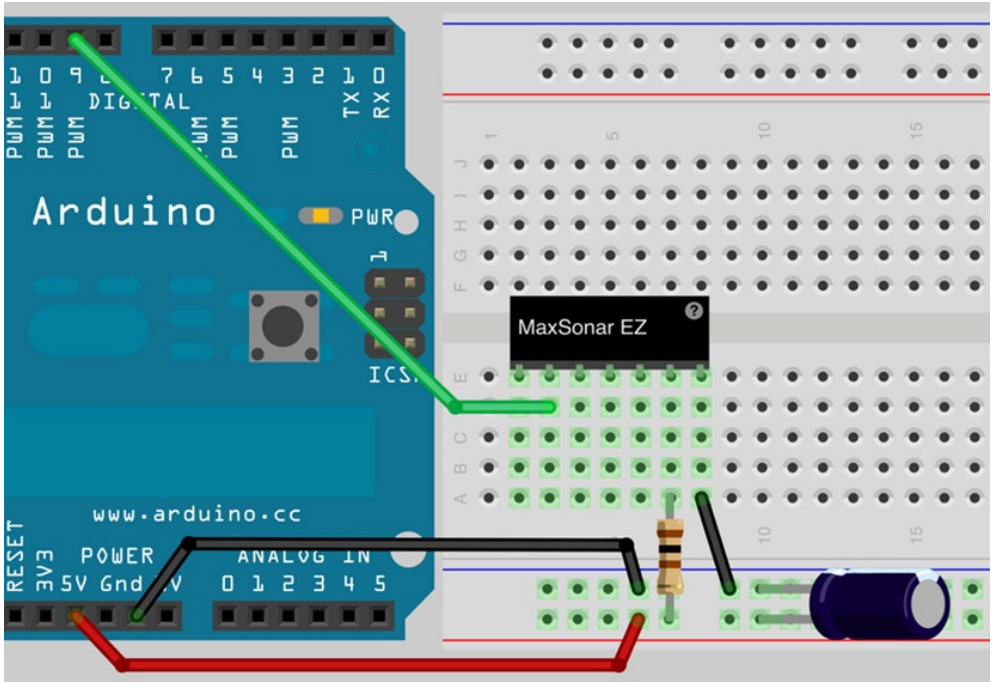


Рис. 14-1. *Схема для проекта 38 - простой ультразвуковой дальномер*

Поскольку Fritzing (программа, используемая для создания макетных схем в этой книге) не имеет LV-MaxSonar в своей библиотеке деталей, я использовал «загадочную деталь» в качестве замены. Подключите + 5В и землю (GND) к двум шинам питания на макетной плате. Поместите электролитический конденсатор на 100 мФ на шины питания, убедившись, что вы подключили более длинную ногу к + 5 В, а более короткую (также с белой полосой и знаками минус поперек нее) к шине GND. Затем подключите перемычку между землей и контактом Gnd на датчике. Обязательно соблюдайте полярность, так как при неправильном подключении он может взорваться! Затем подключите резистор 100 Ом между шиной + 5 В и контактом + 5 В на датчике. Наконец, подключите провод между контактом PW на датчике и цифровым контактом 9.

Введите код

После того, как вы проверили правильность подключения, введите код из Листинга 14-1 и загрузите его в свой Arduino.

Листинг 14-1. Код для проекта 38

```
// Project 38

#define sensorPin 9

long pwmRange, inch, cm;

void setup() {
  // Запуск последовательной связи
  Serial.begin(115200);
  pinMode(sensorPin, INPUT);
}
```

```
void loop() {
pwmRange = pulseIn(sensorPin, HIGH);

  // 147 микросекунд на дюйм в соответствии с таблицей данных
  inch = pwmRange / 147;
  // преобразовать дюйм в см
  cm = inch * 2.54;

  Serial.print(inch);
  Serial.print(" inches    ");
  Serial.print(cm);
  Serial.println(" cm");
}
```

После того, как вы загрузили код, выключите Arduino на секунду. Затем убедитесь, что ваш ультразвуковой датчик неподвижен и указывает на что-то, что не движется. Лучше всего положить его на стол и направить на потолок. При резервном питании Arduino убедитесь, что рядом с датчиком ничего нет. При первом включении устройства оно выполняет процедуру калибровки для первого цикла чтения. Убедитесь, что в его луче ничего не движется, иначе вы получите неточные показания. Эта информация затем используется для определения диапазона объектов в зоне прямой видимости датчика. Измерьте расстояние между датчиком и потолком, и это расстояние (примерно) будет выводиться на МПП, когда вы его откроете. Если расстояние неточно, выключите и снова включите Arduino, чтобы устройство могло откалибровать без препятствий. Перемещая датчик или поднимая и опуская руку над датчиком, расстояниедо объекта, находящегося на его пути, будет отображаться на МПП.

Проект 38 - Простой ультразвуковой дальномер - Обзор кода

Опять же, у вас есть короткий и простой фрагмент кода для использования с этим датчиком. Во-первых, вы начинаете с определения вывода, к которому подключен датчик. Вы используете цифровой вывод 9:

```
#define sensorPin 9
```

Затем объявляются три переменные типа long:

```
long pwmRange, inch, cm;
```

Они будут использоваться для сохранения диапазона, считанного датчиком, который будет преобразован в дюймы, а затем в сантиметры, соответственно.В программе настройки вы просто начинаете последовательную связь со скоростью 115200 бод и устанавливаете вывод с датчиком на вход:

```
Serial.begin(115200);
pinMode(sensorPin, INPUT);
```

В основном цикле вы начинаете со считывания импульса с вывода датчика и сохранения его в pwmRange:

```
pwmRange = pulseIn(sensorPin, HIGH);
```

Для этого мы используем специально разработаную новую команду pulseIn. Она предназначена для измерения длины импульса в микросекундах на выводе. Вывод PW датчика устанавливает HIGH уровень, когда ультразвуковой импульс отправляется с датчика, а затем LOW, когда этот импульс получен обратно. Время между переключениями уровней вывода PW даст вам расстояние после преобразования. Для команды pulseIn требуются два параметра.

Первый - это вывод, который вы хотите слушать, а второй - уровни, чтобы определить, в каком состоянии команда pulseIn начнет отсчет импульса. В вашем случае у вас установлено значение HIGH, поэтому, как только вывод датчика переходит в HIGH, команда pulseIn запускает отсчет времени; как только он станет НИЗКИМ, она остановит отсчет времени, а затем вернет время в микросекундах.
В соответствии с таблицей данных датчиков серии LV-MaxSonar, устройство обнаруживает расстояния от 0 до 254 дюймов (6,45 метра), при этом расстояния ниже 6 дюймов выводятся как 6 дюймов. Каждые 147 мс (микросекунд) равны одному дюйму. Итак, чтобы преобразовать значение, возвращаемое командой pulseIn, в дюймы, вам просто нужно разделить его на 147.

Затем это значение сохраняется в дюймах.

```
inch = pwmRange / 147;
```

Затем это значение умножается на 2,54, чтобы получить расстояние в сантиметрах:

```
cm = inch * 2.54;
```

Наконец, значения в дюймах и сантиметрах выводятся на монитор последовательного порта (МПП):

```
Serial.print(inch);
Serial.print(" inches
");Serial.print(cm);
Serial.println(" cm");
```

Проект 38 - Простой ультразвуковой дальномер - Обзор устройства

Новый компонент, представленный в этом проекте, - ультразвуковой дальномер. В этом устройстве используется ультразвук, который представляет собой звук с очень высокой частотой, превышающей верхний предел человеческого слуха. В случае MaxSonar он посылает импульс с частотой 42 кГц (средний человек имеет верхний предел слуха около 20 кГц). Импульс посылается устройством от датчика, а затем снова улавливается тем же датчиком, когда он отражается от объекта. Вычислив время, необходимое для возврата импульса, вы можете определить расстояние до отраженного объекта (см. Рисунок 14-2).
Звуковые волны распространяются со скоростью звука, которая в сухом воздухе при 20°C (68°F) составляет 343 метра в секунду, или 1125 футов в секунду. Зная это, вы можете рассчитать скорость в микросекундах, с которой звуковая волна возвращается к датчику. Как это часто бывает, в таблице данных указано, что для возврата импульса на каждый дюйм требуется 147 мс. Таким образом, если взять время в микросекундах и разделить его на 147, мы получим расстояние в дюймах, а затем вы можете преобразовать его в сантиметры, если необходимо.

Рис. 14-2. *Принцип измерения расстояния с помощью сонара или радара*

Этот принцип также называется SONAR (звуковая навигация и определение расстояния) и используется на подводных лодках для определения расстояния до других морских судов или близлежащих опасностей. Он также используется летучими мышами для обнаружения своей добычи.

У устройств MaxSonar есть три способа считывания данных с датчика. Один - аналоговый вход, второй - вход ШИМ и последний - последовательный интерфейс. Вход ШИМ, вероятно, самый простой в использовании с самыми надежными данными, поэтому я использовал его здесь. Не стесняйтесь исследовать и использовать два других контакта, если хотите, хотя от этого не будет никакой реальной пользы, если вам специально не нужен аналоговый или последовательный поток данных.

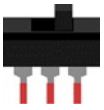
Теперь, когда вы знаете, как работает датчик, давайте применим его на практике и создадим ультразвуковую рулетку или индикатор расстояния.

Проект 39 - Ультразвуковой дальномер

В этом проекте мы используем ультразвуковой датчик для создания (достаточно) точного отображения расстояния с микросхемой драйвера светодиодов MAX7219, которая использовалась в главе 7 для отображения измеренного расстояния. Однако вместо точечно-матричных индикаторов мы используем набор 7-сегментных светодиодных индикаторов, для чего и был разработан MAX7219.

Требуемые детали

Table 14-2. *Требуемые детали for Project 39*

LV-MaxSonar EZ3*	
Электролитический конденсатор 100 мФ	
Резистор 100 Ом	
2 резистора 10 кОм	
выключатель	
5 × 7-сегментных светодиодных дисплеев (Общий катод)	
MAX7219 LED Driver IC	

**или любой из диапазона LV*

Переключатель должен быть одноходового типа (SPST). Однополюсный переключатель имеет только две клеммы, которые замыкаются в одном положении и размыкаются в другом. Вы будете использовать одно из этих положений для переключения дисплея между дюймами и сантиметрами. 7-сегментные светодиодные индикаторы должны быть с общим катодом. Обязательно получите техническое описание приобретаемого вами типа, чтобы узнать, как его подключить, поскольку он может отличаться от моего.

Подключение

Подключите как показано на рисунке 14-3.

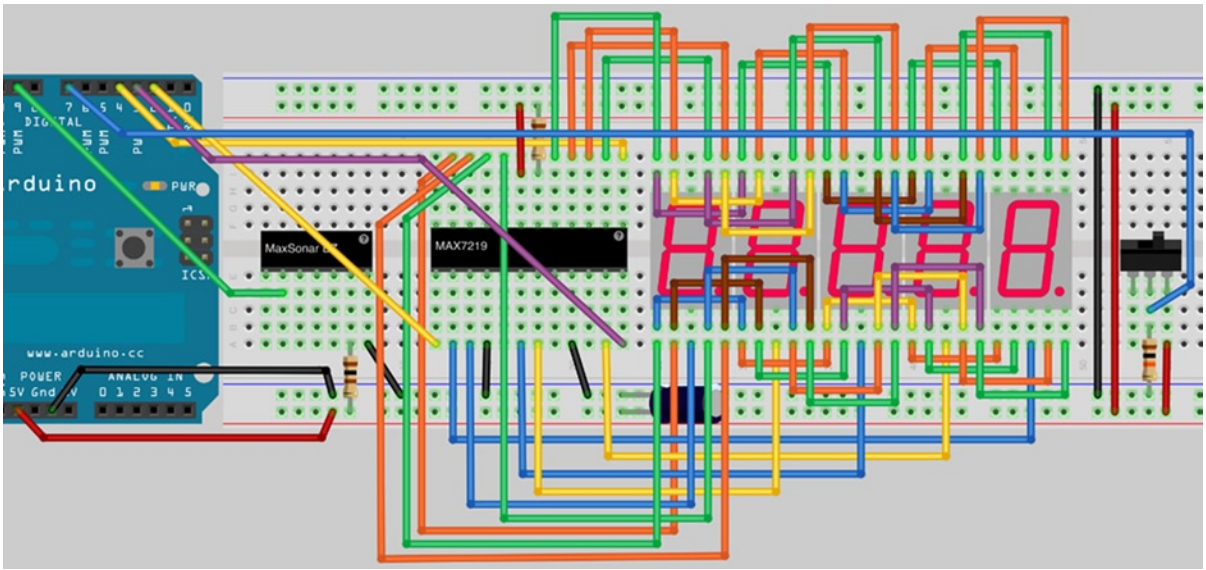


Рис. 14-3. *Схема для Проекта 39 - Ультразвуковой индикатор расстояния*

Эта схема довольно сложна, поэтому ниже я также предоставил таблицу контактов (Таблица 14-3) для Arduino, Max7219 и 7-сегментного дисплея, чтобы вы могли сопоставить их со схемой. У дисплеев, которые я использовал, был код 5101AB, но подойдет любой 7-сегментный дисплей с ОК.

Table 14-3. *Pin Outs Required for Project 39*

Arduino	MaxSonar	MAX7219	7-Segment	Other
Digital Pin 2		Pin 1 (DIN)		
Digital Pin 3		Pin 12 (LOAD)		
Digital Pin 4		Pin 13 (CLK)		
Digital Pin 7				Выключатель
Digital Pin 9	PW	Pin 4 (Gnd)		Gnd
		Pin 9 (Gnd)		Gnd
		Pin 18 (ISET)		+5В через резистор 10К
		Pin 19 (VDD)		+5В
		Pin 2 (DIG 0)	Земля для дисплея 0	
		Pin 11 (DIG 1)	Земля для дисплея 1	

(продолжение)

Таб. 14-3. (*продолжение*)

Arduino	MaxSonar	MAX7219	7-Segment
		Pin 6 (DIG 2)	Общий на дисплее 2
		Pin 7 (DIG 3)	Пин 7 (DIG 3) общий на дисплее 3
		Pin 3 (DIG 4)	Пин 3 (DIG 4) общий на дисплее 4
		Pin 14	
		Pin 16	SEG B
		Pin 20	SEG C
		Pin 23	SEG D
		Pin 21	SEG E
		Pin 15	SEG F
		Pin 17	SEG G
		Pin 22	SEG DP

После того, как вы подключили MAX7219 к контактам SEG AG и DP первого 7-сегментного дисплея, то есть ближайшего к микросхеме (см. Рисунок 14-4), подключите контакты SEG на первом дисплее ко второму, а затем от второго к третьему и так далее. Все выводы SEG связаны вместе на каждом дисплее, при этом выводы земли разделены и идут к соответствующим выводам DIG на MAX7219. Обязательно ознакомьтесь с таблицей данных для своего 7-сегментного дисплея, так как его контакты могут отличаться от моих.

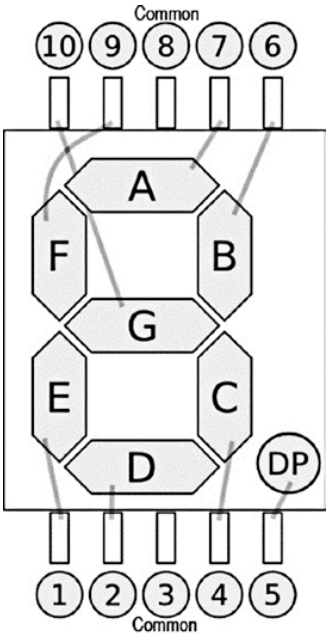


Рис. 14-4. *Типичный 7-сегментный светодиодный дисплей с общим катодом и с распиновкой*

MaxSonar подключается так же, как и раньше, за исключением того, что вывод PW подключается к цифровому выводу 9 вместо 3.
Обратите внимание, что вам может потребоваться использовать внешний источник питания для этого проекта, если вы обнаружите, что он работает нестабильно - он может потреблять слишком много энергии из порта USB.

Введите код

После того, как вы проверили правильность подключения, включите Arduino и введите код из Листинга 14-2, затем загрузите его на свой Arduino. Убедитесь, что у вас есть LedControl.h в папке с библиотеками (инструкции см. В главе 7).

Листинг 14-2. Код для проекта 39

```
// Проект 39

#include "LedControl.h"

#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0

float pwmRange, averageReading, inch, cm;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);

void setup() {
        // Пробуждаем MAX7219
        lc.shutdown(0,false);
        // Устанавливаем среднюю яркость
        lc.setIntensity(0,8);
        // очищаем дисплей
        lc.clearDisplay(0);
        pinMode(sensorPin, INPUT);
        pinMode(switchPin, INPUT);
}

void loop() {
        averageReading = 0;
        for (int i = 0; i<samples; i++) {
                pwmRange = pulseIn(sensorPin, HIGH);
                averageReading += pwmRange;
        }

        averageReading /= samples;
        // 147uS per inch according to datasheet
        inch = averageReading / 147;
        // конвертируем дюйм в см
        cm = inch * 2.54;
```

```
        if (digitalRead(switchPin)) {
                displayDigit(inch);
        }
        else {
                displayDigit(cm);
        }
}

void displayDigit(float value) {
        int number = value*100;
        lc.setDigit(0,4,number/10000,false);        // цифра 100
        lc.setDigit(0,3,(number%10000)/1000,false); // цифра 10
        lc.setDigit(0,2,(number%1000)/100,true);    // первая цифра при включенной десят. точке
        lc.setDigit(0,1,(number%100)/10,false);     // цифра 10
        lc.setDigit(0,0,number%10,false);           // цифра 100
}
```

Проект 39 - Ультразвуковой индикатор расстояния - Обзор кода

Проект начинается с включения библиотеки LedControl.h:

```
#include "LedControl.h"
```

Затем мы определяем выводы, которые потребуются для датчика и микросхемы MAX7219:

```
#define sensorPin 9
#define switchPin 7
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
```

Показания датчика сглаживаются с помощью простого алгоритма скользящего среднего, поэтому вам нужно определить, сколько образцов вы сделаете для этого:

```
#define samples 5.0
```

Позже вы будете использовать это число с числами с плавающей запятой (floats), поэтому, чтобы избежать ошибок, число определено как 5.0, а не 5, чтобы убедиться, что оно принудительно используется как число с плавающей запятой, а не как int.

Затем объявляются floats для датчика, как в Проекте 38, но с добавлением averageReading, которое вы будете использовать позже в программе:

```
float pwmRange, averageReading, inch, cm;
```

Вы создаете объект LedControl и устанавливаете используемые выводы и количество чипов:

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

Как и в Проекте 21, вы должны убедиться, что дисплей включен, интенсивность установлена на среднюю, адисплей очищен и готов к использованию:

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
```

Контакты датчика и кнопки установлены на ВХОД:

```
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
```

Затем вы попадаете в основной цикл. Сначала переменная averageReading устанавливается в ноль:

```
averageReading = 0;
```

Затем запускается цикл for для сбора выборок (замеров) с датчика. Значение датчика, как и раньше, считывается в pwmRange, но затем добавляется к averageReading при каждом запуске цикла. Цикл forбудет повторять количество раз, определенное в выборках в начале программы.

```
for (int i = 0; i<samples; i++) {
        pwmRange = pulseIn(sensorPin, HIGH);
        averageReading += pwmRange;
}
```

Затем вы берете значение в averageReading и делите его на число в выборках. В вашем случае номер выборки установлен на 5, поэтому берется пять выборок, добавляется к averageReading, которая изначально равна нулю, а затем делится на пять, чтобы получить среднее значение. Это обеспечивает более точное считывание за счет усреднения некоторых шумов в показаниях, вызванных различными причинами.

```
averageReading /= samples;
```

Как и раньше, время импульса преобразуется в дюймы и сантиметры:

```
inch = averageReading / 147;
cm = inch * 2.54;
```

Затем вы используете оператор if, чтобы проверить, установлен ли переключатель HIGH или LOW. Если он HIGH, то запускается функция displayDigit () (объясненная вкратце), и ей передается значение в дюймах. Если переключатель установлен в LOW, инструкция else запускает функцию, но вместо этого использует сантиметры.

```
if (digitalRead(switchPin)) {
        displayDigit(inch);
}
else {
        displayDigit(cm);
}
```

Этот оператор if-else обеспечивает отображение дюймов или сантиметров в зависимости от положения переключателя.

Наконец, мы определяем функцию displayDigit (). Эта функция просто печатает переданное ей число на 7-сегментном индикаторе. Число с плавающей запятой необходимо передать функции в качестве параметра. Это будут дюймы или сантиметры.

```
void displayDigit(float value) {
```

Число, переданное в эту функцию, является числом с плавающей запятой и будет иметь цифры после десятичной запятой. Нас интересуют только первые две цифры после десятичной точки, поэтому она умножается на 100, чтобы сдвинуть эти две цифры на два места влево:

```
int number = value*100;
```

Это связано с тем, что мы будем использовать оператор по модулю%, который требует целых чисел, и поэтому мы должны преобразовать число с плавающей запятой в целое число. Умножение его на 100 гарантирует, что две цифры после десятичной точки сохранятся, а все остальное будет потеряно. Теперь у нас есть исходное число, но без десятичной точки. Это не имеет значения, поскольку мы знаем, что после десятичной точки идут две цифры.

Затем нам нужно взять это число и отображать его по одной цифре на 7-сегментных дисплеях.

Каждая цифра отображается с помощью команды setDigit, для которой требуется четыре параметра:

```
setDigit(int addr, int digit, byte value, boolean dp);
```

где addr является адресом микросхемы MAX7219. У нас всего один чип, поэтому это значение равно нулю. Если был добавлен второй чип, его адрес был бы 1 и так далее. Цифра - это индекс управляемого 7-сегментного дисплея. В нашем случае на правом дисплее отображается цифра 0, слева - 1 и так далее. Значение - это фактическая цифра от 0 до 9, которую мы хотим отобразить на 7-сегментном индикаторе. Наконец, логическое значение false или true определяет, включена или выключена десятичная точка на этом дисплее.

Итак, используя команду setDigit, мы берем значение, хранящееся в целом числе с именем number, и выполняем над ним операции деления и по модулю, чтобы получить каждую цифру отдельно, а затем отображаем их на светодиодном индикаторе:

```
lc.setDigit(0,4,number/10000,false);        // цифра 100
lc.setDigit(0,3,(number%10000)/1000,false); // цифра 10
lc.setDigit(0,2,(number%1000)/100,true);    // первая цифра при включенном DP-десятичн. точка
lc.setDigit(0,1,(number%100)/10,false);     // цифра 10
lc.setDigit(0,0,number%10,false);           // цифра 100
```

У цифры 2 включена десятичная точка, так как мы хотим две цифры после десятичной точки, чтобы флаг DP был истинным.

Мы можем увидеть, как это работает, на следующем примере. Допустим, отобразится число 543,21. Помните, что это число умножается на 100, и получается 54 321. Для цифры 0 вы берете число и выполняете над ним операцию по модулю 10. Это оставляет вам первую цифру (крайнюю справа), которая равна 1.

543.21 * 100 = 54321
54321 % 10 = 1

Помните, что оператор по модулю % делит целое число на число после него, но оставляет вам только остаток. 54321 разделить на 10 будет 5432,1, а остаток равен 1. Это даст нам первую цифру (крайнюю справа), которая будет отображаться.

Вторая цифра (столбец 10) складывается по модулю 100, а затем делится на 10, чтобы получить вторую цифру.

54321 % 100 = 21
21/10 = 2 (помните, что это целочисленная арифметика, поэтому все, что находится после десятичной точки, теряется) и так далее.

Если вы выполните вычисления, используя 543,21 в качестве исходного числа, вы увидите, что набор операций по модулю и делению оставляет вам каждую отдельную цифру исходного числа. Добавление десятичной точки к цифре 2 (третья справа) гарантирует, что число будет отображаться с двумя цифрами после десятичной точки.

В итоге мы получаем довольно точную ультразвуковую рулетку, отображаемую с разрешением до сотых долей дюйма или сантиметра. Имейте в виду, что результаты могут быть неточными, поскольку звуковые волны будут двигаться быстрее или медленнее из-за разности температуры или давления воздуха. Кроме того, звуковые волны по-разному отражаются от разных поверхностей. Идеально ровная поверхность, перпендикулярная плоскости датчика, будет хорошо отражать звук и давать наиболее точные показания. Поверхность с неровностями, звукопоглощающая или наклонная поверхность дает неточные показания. Поэкспериментируйте с разными поверхностями и сравните показания с настоящей рулеткой.

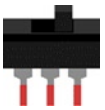
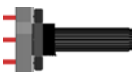
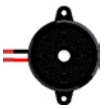
Давайте теперь использовать ультразвуковой датчик для чего-нибудь другого.

Проект 40 - Ультразвуковая сигнализация

Мы используем схему из последнего проекта и превратим ее в систему сигнализации.

Требуемые детали

Таб. 14-4. *Требуемые деталидля проекта 40*

LV-MaxSonar EZ3*	
Электролитический конденсатор 100 мФ	
2 × 100 Ом резистор	
Резистор 3 × 10 кОм	
Выключатель	
Кнопка	
7-сегментный индикатор (Общий катод)	
Микросхема драйвера светодиодов MAX7219	
Потенциометр 5-10 кОм	
Пьезозуммер	

**или любой из диапазона LV*

Подключение

Подключите все, как показано на Рис. 14-5.

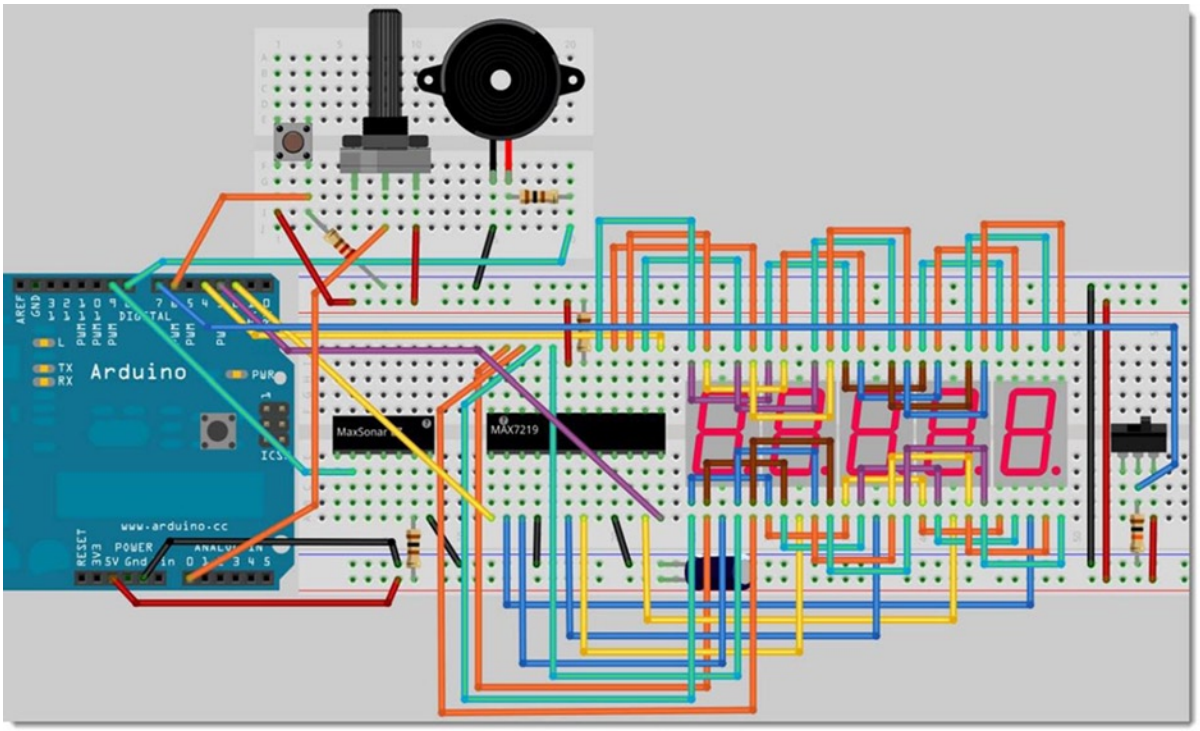


Рис. 14-5. *Схема для Проекта 40 - Ультразвуковая сигнализация*

Схема такая же, как и в Project 39, но с добавлением кнопки, потенциометра и пьезозуммера. На кнопке оба вывода подключены к + 5В и к земле,где Gnd подключен через резистор 10 кОм.
Провод идет от этого же контакта к цифровому выводу 6. Крайные выводы потенциометра подключены к +5В и к GND,а центральный вывод идет на аналоговый вывод 0.
Отрицательный вывод динамика подключен к заземлению, а положительный вывод через резистор 100 Ом - к цифровому выводу 8. Потенциометр будет использоваться для регулировки диапазона датчика аварийной сигнализации,а кнопка сбросит систему после срабатывания сигнализации.
Пьезозумер при срабатывании сигнализации подаст сигнал тревоги.

Введите код

После проверки правильности подключения включите Arduino и загрузите код из Листинга 14-3.

Listing 14-3. Код для проекта 40

```
// Проект 40

#include "LedControl.h"

#define sensorPin 9
#define switchPin 7
#define buttonPin 6
```

```
#define potPin 0
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0

float pwmRange, averageReading, inch, cm, alarmRange;
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);

void setup() {

 // Wakeup the MAX7219
  lc.shutdown(0,false);
  // Set it to medium brightness
  lc.setIntensity(0,8);
  // clear the display
  lc.clearDisplay(0);
  pinMode(sensorPin, INPUT);
  pinMode(switchPin, INPUT);
}

void loop() {
  readPot();
  averageReading = 0;
  for (int i = 0; i<samples; i++) {
   pwmRange = pulseIn(sensorPin, HIGH);
  averageReading += pwmRange;
  }

  averageReading /= samples;
  // 147uS per inch according to datasheet
  inch = averageReading / 147;
  // convert inch to cm
  cm = inch * 2.54;

  if (digitalRead(switchPin)) {
  displayDigit(inch);
  }
  else {
    displayDigit(cm);
  }

        // если текущий диапазон меньше, чем alarmRange, отключить тревогу
        if (inch<=alarmRange) {startAlarm();}
}

void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false);        // цифра 100
    lc.setDigit(0,3,(number%10000)/1000,false); //  цифра 10
```

```
    lc.setDigit(0,2,(number%1000)/100,true);    // первая цифра
    lc.setDigit(0,1,(number%100)/10,false);     // 10-я цифра
    lc.setDigit(0,0,number%10,false);           // 100-я цифра
}

// считываем потенциометр
float readPot()  {
  float potValue = analogRead(potPin);
  alarmRange = 254 * (potValue/1024);
  return alarmRange;
}

// set off the alarm sound till reset pressed
void startAlarm() {
  while(1) {
    for (int freq=800; freq<2500;freq++) {
      tone(8, freq);
      if (digitalRead(buttonPin)) {
        noTone(8);
        return;
      }
    }
  }
}
```

После ввода кода загрузите его в Arduino, а затем выключите устройство. Включите питание, убедившись, что датчик может правильно откалиброваться. Теперь вы можете повернуть ручку потенциометра, чтобы отрегулировать диапазон сигнала тревоги. Поместите руку в область излучения и постепенно приближайтесь, пока не сработает сигнализация. Показания после срабатывания будильника останутся неизменными и покажут вам последнее измеренное расстояние. Это ваш диапазон будильника. Нажмите кнопку сброса, чтобы отключить сигнал тревоги, перезагрузите систему, а затем продолжайте регулировать потенциометр, пока не получите диапазон, который вас устраивает. Теперь ваша сигнализация готова защитить все, что находится рядом. Все, что находится в пределах диапазона установленного вами датчика, будет активировать сигнал тревоги до тех пор, пока он не будет сброшен.

Проект 40 - Ультразвуковая сигнализация - Обзор кода

Большая часть кода такая же, как в Проекте 39, поэтому я пропущу объяснение этих разделов. Библиотека LedControl загружается в:

```
#include "LedControl.h"
```

Затем определяются используемые выводы, а также количество микросхем и выборок, как и раньше:

```
#define sensorPin 9
#define switchPin 7
#define buttonPin 6
#define potPin 0
#define DataIn 2
#define CLK 4
#define LOAD 3
#define NumChips 1
#define samples 5.0
```

Мы добавим определение для buttonPin и potPin. Объявлены переменные, включая новую переменную с именем alarmRange, которая будет удерживать пороговое значение расстояния, после которого будет звучать сигнал тревоги, если человек приблизится к установленному диапазону:

```
float pwmRange, averageReading, inch, cm, alarmRange;
```

Мы создадим объект LedControl с именем lc и определим выводы:

```
LedControl lc=LedControl(DataIn,CLK,LOAD,NumChips);
```

Цикл setup () такой же, как и раньше, с добавлением установки pinMode кнопки buttonPin на INPUT:

```
lc.shutdown(0,false);
lc.setIntensity(0,8);
lc.clearDisplay(0);
pinMode(sensorPin, INPUT);
pinMode(switchPin, INPUT);
pinMode(buttonPin, INPUT);
```

Основной цикл начинается с вызова новой функции с именем readPot (). Эта функция считывает значение потенциометра, которое мы будете использовать для настройки диапазона срабатывания сигнализации (обсуждается позже):

```
readPot();
```

Остальная часть основного цикла такая же, как в проекте 39.

```
averageReading = 0;
for (int i = 0; i<samples; i++) {
        pwmRange = pulseIn(sensorPin, HIGH);
        averageReading += pwmRange;
 }

averageReading /= samples;
inch = averageReading / 147;
cm = inch * 2.54;

if (digitalRead(switchPin)) {
        displayDigit(inch);
}
else {
        displayDigit(cm);
}
```

пока мы не дойдем до следующего оператора if

```
if (inch<=alarmRange) {startAlarm();}
```

который просто проверяет, меньше ли текущее измерение датчика или равно значению в alarmRange, которое был установлен пользователем, и если это так, вызывает функцию startAlarm ().

Функция displayDigit () такая же, как в Project 39:

```
void displayDigit(float value) {
    int number = value*100;
    lc.setDigit(0,4,number/10000,false);        // цифра 100
    lc.setDigit(0,3,(number%10000)/1000,false); // цифра 10
    lc.setDigit(0,2,(number%1000)/100,true);    // первая цифра
    lc.setDigit(0,1,(number%100)/10,false);     // 10-я цифра
    lc.setDigit(0,0,number%10,false);           // 100-я цифра
}
```

Далее идет первая из двух новых функций. Он предназначен для считывания показаний потенциометра и преобразования его значения в дюймы для установки диапазона срабатывания сигнализации. Функция не имеет параметров, но имеет тип float, так как она будет возвращать значение с плавающей запятой в alarmRange.

```
float readPot()
```

Затем вы считываете аналоговое значение из potPin и сохраняете его в potValue:

```
float potValue = analogRead(potPin);
```

Затем вы выполняете расчет этого значения, чтобы преобразовать значения от 0 до 1023, считываемые с потенциометра, и преобразовать их в максимальный и минимальный диапазон датчика, то есть от 0 до 254 дюймов.

```
alarmRange = 254 * (potValue/1024);
```

Затем вы возвращаете это значение в точку, в которой была вызвана функция:

```
return alarmRange;
```

Следующая функция отвечает за включение звукового сигнала будильника. Это функция startAlarm ():

```
void startAlarm() {
```

Затем у нас есть цикл while. Вы встречали цикл while в главе 3. Цикл будет выполняться, пока выражение в скобках истинно. Параметр для цикла while равен 1. Это просто означает, что пока проверяемое значение истинно, цикл будет выполняться. В этом случае проверяемое значение является постоянным значением1, поэтому цикл всегда выполняется бесконечно.

Вы будете использовать команду возврата для выхода из цикла.

```
while(1) {
```

Теперь у нас есть цикл for, который прокручивает частоты от 800 до 2500 Гц:

```
for (int freq=800; freq<2500;freq++) {
```

Мы воспроизводим тональный сигнал с вывода 8, где находится пьезозуммер, и воспроизводите частоту, сохраненную в freq:

```
tone(8, freq);
```

Теперь мы проверяем buttonPin с помощью digitalRead, чтобы узнать, была ли нажата кнопка или нет:

```
if (digitalRead(buttonPin)) {
```

Если кнопка была нажата, запускается код в скобках. Это начинается с команды noTone (), чтобы отключить звуковой сигнал, а затем возврат для выхода из функции и возврата к основному циклу программы:

```
noTone(8);
return;
```

В следующем проекте мы сохраним ту же схему, но загрузим немного измененый код, чтобы использовать датчик для другой цели.

Проект 41 - Ультразвуковой терменвокс

Для этого проекта мы используем ту же схему.
Мы не будем демонтировать потенциометр, переключатель или кнопку сброса в этом проекте, чтобы вы могли опять позже вернуться к использованию Проекту 40 .

Мы используем датчик для создания терменвокса, который использует ультразвуковой датчик вместо электрического поля, которое использует настоящий терменвокс. Если вы не знаете, что такое терменвокс, поищите описание в Википедии. По сути, это электронный инструмент, на котором вы можете играть, не касаясь его, поместив руку в электрическое поле и перемещая руку внутри этого поля. Устройство обнаруживает изменения в поле и воспроизводит ноту, относящуюся к расстоянию до катушки. Это сложно объяснить, поэтому посмотрите несколько видеороликов о его использовании на YouTube. Поскольку схема такая же, я сразу перейду к коду.

Введите код

Введите код из Листинга 14-4.

Листинг 14-4. Код для проекта 41

```
// Проект 41

#define sensorPin 9

#define lowerFreq 123  // C3
#define upperFreq 2093 // C7
#define playHeight 36

float pwmRange, inch, cm, note;

void setup() {
        pinMode(sensorPin, INPUT);
}

void loop() {
        pwmRange = pulseIn(sensorPin, HIGH);

        inch = pwmRange / 147;
        // конвертируем дюйм в см
        cm = inch * 2.54;
```

```
        // сопоставляем диапазон playHeight с верхними и нижними частотами
        note = map(inch, 0, playHeight, lowerFreq, upperFreq);
        if (inch<playHeight) {tone(8, note); }
        else {noTone(8);}
}
```

После того, как вы загрузите его в Arduino, вы можете поместить свою руку в область датчика, и терменвокс воспроизведет ноту, привязанную к этой высоте от датчика. Перемещайте руку вверх и вниз , и воспроизводимые тона также будут перемещаться вверх и вниз по шкале. Вы можете настроить верхний и нижний частотные диапазоны в коде, если хотите.

Проект 41 - Ультразвуковой терменвокс - Обзор кода

Этот код представляет собой урезанную версию Project 40 с некоторым кодом для преобразования диапазона датчика в тон, который будет воспроизводиться на пьезозуммере. Мы начинаем с определения пина с датчиком, как и раньше.

```
#define sensorPin 9
```

Затем у нас есть несколько новых определений для воспроизведения верхних и нижних нот и playHeight в дюймах.PlayHeight - это расстояние между датчиком и расстоянием, до которого может дотянуться ваша рука во время игры на инструменте.Вы можете настроить этот диапазон на что-то большее,или меньшее, если хотите.

```
#define lowerFreq 123  // C3
#define upperFreq 2093 // C7
#define playHeight 36
```

Переменные объявлены с одной для ноты,которая будет нотой, воспроизводимой через пьезозуммер:

```
float pwmRange, inch, cm, note;
```

Процедура настройки просто устанавливает вывод с датчиком как вход:

```
pinMode(sensorPin, INPUT);
```

В основном цикле код - это только самое необходимое. Значение с датчика считывается и конвертируется в дюймы:

```
pwmRange = pulseIn(sensorPin, HIGH);
inch = pwmRange / 147;
```

Затем значения в дюймах от нуля до значения, хранящегося в playHeight, сопоставляются с верхней и нижней частотами, определенными в начале программы:

```
note = map(inch, 0, playHeight, lowerFreq, upperFreq);
```

Вам нужно, чтобы звук воспроизводился только тогда, когда ваша рука находится в области луча, поэтому вы проверяете, меньше ли значение датчика или равно высоте люфта. Если это так, рука должна находиться в пределах игровой зоны, а затем проигрывается тон.

```
if (inch<=playHeight) {tone(8, note); }
```

Если рука не загораживает луч или удалена от него, звук прекращается:

```
else {noTone(8);}
```

Поиграйте со значениями playHeight, upperFreq и lowerFreq, чтобы получить желаемый звук.

Резюме

В этой главе вы узнали, как подключить ультразвуковой датчик. Я также представил несколько вариантов использования датчика, чтобы вы почувствовали, как его можно использовать в ваших собственных проектах. Подобные датчики часто используются в проектах по любительской робототехнике, в которых робот определяет, находится ли он рядом со стеной или другим препятствием. Я также видел, как они использовались в проектах автожиров, чтобы корабль не врезался в стены или людей. Другое распространенное использование - определение высоты жидкости внутри резервуара или трубки. Я уверен, что вы подумаете о других замечательных применениях таких датчиков.

Предметы и понятия, затронутые в главе 14:

- Как работает ультразвуковой датчик
- Как прочитать вывод PW с устройств MaxSonar
- Использование конденсатора для сглаживания пульсации
- Как использовать команду pulseIn для измерения ширины импульса
- Различные варианты использования ультразвукового дальномера
- Как использовать MAX7219 для управления 7-сегментными дисплеями
- Как подключить 7-сегментный дисплей с общим катодом
- Использование алгоритма скользящего среднего для сглаживания показаний данных
- Как использовать команду setDigit () для отображения цифр на 7-сегментных светодиодах
- Использование операторов деления и по модулю для выделения цифр из длинного числа
- Как создать бесконечный цикл с помощью команды while (1)

ГЛАВА 15

Чтение и запись на SD-карту

Теперь вы собираетесь изучить основы записи и чтения с SD-карты. SD-карты - это небольшой и дешевый метод хранения данных, и Arduino может относительно легко обмениваться данными с ними, используя свой интерфейс SPI. Вы узнаете достаточно, чтобы иметь возможность создавать новый файл, добавлять к существующему файлу, ставить метку времени для файла и записывать данные в этот файл. Это позволит вам использовать SD-карту и Arduino в качестве устройства регистрации данных для хранения любых данных, которые вы хотите. Как только вы усвоите основы, вы примените эти знания для создания регистратора данных температуры с отметкой времени.

Проект 42 - Простая SD-карта / чтение и запись

В этом проекте мы подключим SD-карту к Arduino и, используя библиотеку SD.h для доступа к карте и создадим новый файл на карте, напишем текст в этот файл, распечатаем содержимое этого файла, затем удалим файл. Это научит вас основным понятиям доступа к SD-карте, чтения и записи файлов на нее. Вам понадобится SD-карта и способ ее подключения к Arduino. Самый простой способ - получить плату подключения для карт SD / MMC от различных поставщиков электроники. Я использовал один от Sparkfun.

Требуемые детали

Таб. 15-1. *Детали, необходимые для проекта 42*

SD-карта с разъемом	
3 резистора по 3,3 кОм	
3 резистора по 1,8 кОм	

Резисторы образуют делитель напряжения и понижают логические уровни 5 В до 3,3 В. (Обратите внимание, что более безопасным способом было бы использовать специальный преобразователь логического уровня). Никогда не подключайте выходные контакты Arduino напрямую к SD-карте, не понизив предварительно напряжение на них с 5 В до 3,3 В, иначе вы повредите SD-карту.

Подключение

Подключите как показано на рисунке 15-1.

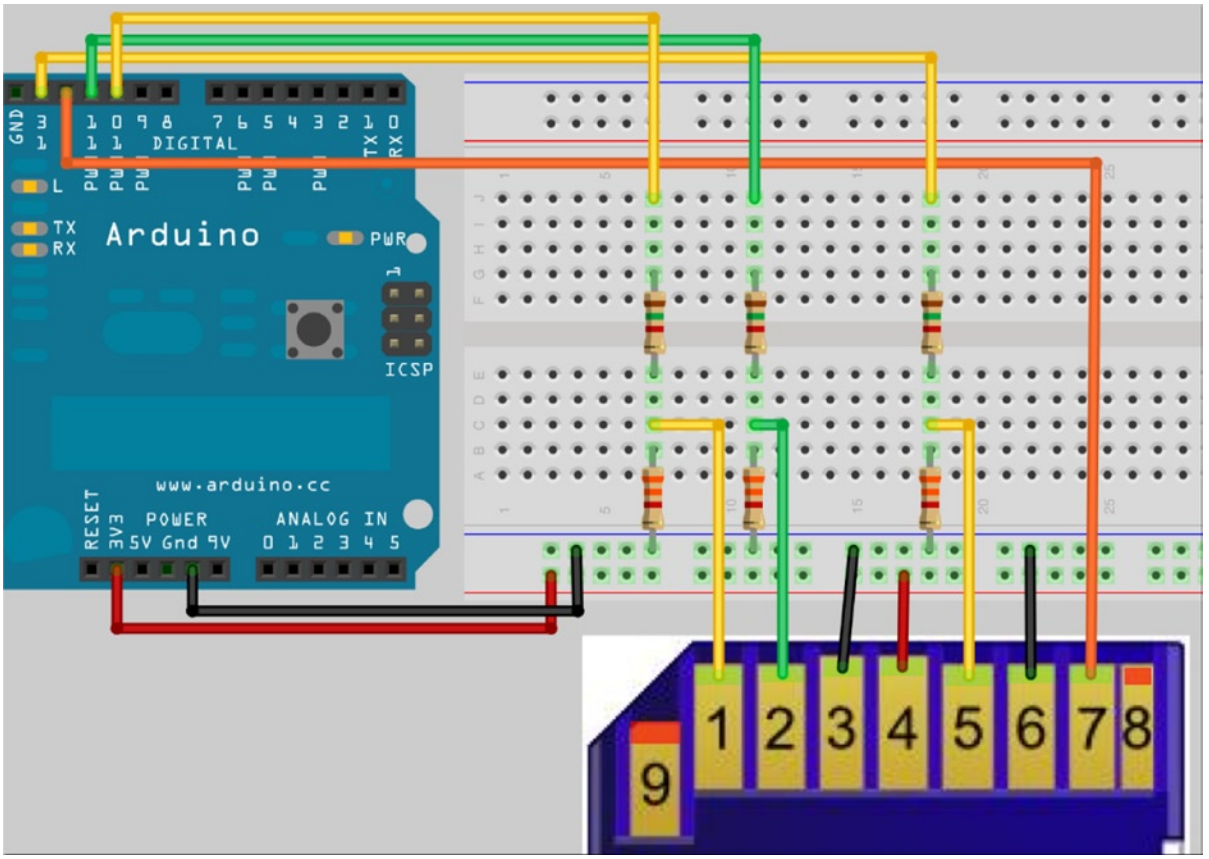


Рис. 15-1. *Схема для Project 42 - Простое чтение / запись SD-карты*

Таб. 15-2. *Таблица соединений между Arduino и SD-картой*

Arduino	SD Card
+3.3V	Pin 4 (VCC)
Gnd	Pins 3 & 6 (GND)
Digital Pin 13 (SCK)	Pin 5 (CLK)
Digital Pin 12 (MISO)	Pin 7 (DO)
Digital Pin 11 (MOSI)	Pin 2 (DI)
Digital Pin 10 (SS)	Pin 1 (CS)

Введите код

Этот код будет использовать библиотеку SD.h, включенную в ваш Arduino.Вы можете найти библиотеку по адресу https://code.google.com/p/sdfatlib/downloads/list. Скачайте последнюю версию.

Используйте отформатированную карту (Fat16 или Fat32). Убедившись, что соединения правильны, введите код из Листинга 15-1.

Листинг 15-1. Код для проекта 42

```
// Проект 42

#include <SD.h>

File File1;

void setup()
{
  Serial.begin(9600);
        while (!Serial) { } // ждем подключения последовательного порта..
                        // Требуется только для Леонардо
  Serial.println("Initializing the SD Card...");

  if (!SD.begin()) {
    Serial.println("Initialization Failed!");
    return;
  }
  Serial.println("Initialization Complete.\n");

  Serial.println("Looking for file 'testfile.txt'...\n");

  if (SD.exists("testfile.txt")) {
    Serial.println("testfile.txt already exists.\n");
  }
  else {
    Serial.println("testfile.txt doesn't exist.");
              Serial.println("Creating file testfile.txt...\n");
  }

  File1 = SD.open("testfile.txt", FILE_WRITE);
  File1.close();

  Serial.println("Writing text to file.....");
  String dataString;
  File1 = SD.open("testfile.txt", FILE_WRITE);
  if (File1) {
      for (int i=1; i<11; i++) {
        dataString = "Test Line ";
        dataString += i;
        File1.println(dataString);
      }
```

```
      Serial.println("Text written to file.....\n");
  }
  File1.close();

  Serial.println("Reading СОДЕРЖАНИЕ of
  textfile.txt...");File1 = SD.open("testfile.txt");

   if (File1) {
    while (File1.available()) {
      Serial.write(File1.read());
    }
    File1.close();
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening testfile.txt");
  }

  // delete the file:
  Serial.println("\nDeleting testfile.txt...\n");
  SD.remove("testfile.txt");

  if (SD.exists("testfile.txt")){
    Serial.println("testfile.txt still exists.");
  }
  else {
    Serial.println("testfile.txt has been deleted.");
  }
}

void loop()
{
  //Здесь ничего нет
}
```

Убедитесь, что ваша SD-карта была недавно отформатирована в формате FAT или FAT32. На Mac убедитесь, что для схемы разделов установлено значение «Основная загрузочная запись». Запустите программу и откройте монитор последовательного порта. Теперь программа попытается записать файл на SD-карту, записать текст в этот файл, прочитать содержимое файла и, наконец, удалить файл.Все это будет отображаться в окне МПП. Если все пойдет хорошо, вы получите следующую информацию:

```
Initializing the SD Card...     Инициализация SD-карты ...
Initialization Complete.        Инициализация завершена.

Looking for file 'testfile.txt'...   Ищем файл testfile.txt ...

testfile.txt doesn't exist.          testfile.txt не существует.
Creating file testfile.txt...        Создание файла testfile.txt ...

Writing text to file.....            Запись текста в файл .....
Text written to file.....            Текст, записанный в файл .....
```

```
Чтение содержимого textfile.txt ...

Test Line 1
Test Line 2
Test Line 3
Test Line 4
Test Line 5
Test Line 6
Test Line 7
Test Line 8
Test Line 9
Test Line 10

Deleting testfile.txt...          Удаление testfile.txt ...

testfile.txt has been deleted.    testfile.txt был удален.
```

Убедитесь, что карта имеет формат SD или SDHC и отформатирована с разделами «Master Boot Record» или MBR.

Посмотрим, как работает код.

Проект 42 - Простое чтение / запись SD-карты - Обзор кода

Сначала нам нужно включить в наш код библиотеку SD.h. Это дает нам доступ ко всем функциям карт и файлов в библиотеке, которые позволяют нам получать доступ к SD-карте, читать и записывать на нее файлы и т. д.

```
#include <SD.h>
```

Затем мы используем команду File, которая является частью библиотеки SD.h, для создания экземпляра объекта File. Нам нужен объект File для доступа, создания, закрытия и удаления файлов. Назовем наш объект File File1.

```
File File1;
```

Поскольку мы хотим, чтобы наша программа запускалась только один раз, мы поместим весь наш код в функцию setup ().

```
void setup()
```

Мы будем выводить некоторые данные обратной связи для пользователя с окном МПП , поэтому нам нужно начать последовательную связь. Мы оставляем скорость передачи данных по умолчанию.

```
Serial.begin(9600);
```

Следующая строка требуется только в том случае, если у вас есть Leonardo, так как для подключения к последовательному порту может потребоваться больше времени. Оператор While проверяет, готов ли последовательный порт, и пока он НЕ (!) Готов, он ждет, поскольку оператор While завершится только тогда, когда последовательный порт будет готов.

```
while (!Serial) { } // ждем подключения последовательного порта.
                    // Требуется только для Леонардо
```

Сообщаем пользователю, что программа пытается инициализировать (начать обмен данными) SD-карту.

```
Serial.println("Initializing the SD Card...");
```

The SD.Begin command initializes the SD card and the SD library. It returns a true if successful and false if it fails. We therefore need to inform the user if the initialization process failed by checking if !SD.Begin (NOT SD.Begin) or if the statement return a false. If so, inform the user and return. The return statement will exit the setup() function.

```
if (!SD.begin()) {
  Serial.println("Initialization Failed!");
  return;
}
```

Если процесс инициализации прошел успешно, сообщите пользователю «Initialization Complete - Инициализация завершена».

```
Serial.println("Initialization Complete.\n");
```

Затем мы собираемся создать текстовый файл на SD-карте с именем testfile.txt. Сначала мы проверяем, существует ли этот файл уже или нет, поэтому мы начинаем с информирования пользователя о том, что ищем файл.

```
Serial.println("Looking for file 'testfile.txt'...\n");
```

Затем функция SD.exists вернет истину, если файл или каталог, который мы ищем, существует, и ложь, если нет. Параметр функции - это имя файла, который мы хотим проверить. Следующий оператор if проверяет это и сообщает пользователю, существует файл или нет. Если это не так, пользователь получает сообщение о том, что программа собирается создать файл.

```
if (SD.exists("testfile.txt")) {
  Serial.println("testfile.txt already exists.\n");
}
else {
  Serial.println("testfile.txt doesn't exist.");
  Serial.println("Creating file testfile.txt...\n");
}
```

Затем мы используем функцию SD.open (), чтобы открыть файл. Если файл, который он пытается открыть, не существует, он создаст его. Функция требует одного или двух параметров. Первый параметр - это имя файла, который мы хотим открыть, а вторым необязательным параметром является либо FILE_READ, чтобы открыть файл для чтения, начиная с начала файла, либо FILE_WRITE, чтобы открыть файл для чтения и записи, начиная с конца файла. Мы назначаем этот открытый или созданный файл объекту File, который мы ранее назвали File1.

```
File1 = SD.open("testfile.txt", FILE_WRITE);
```

После того, как файл был открыт или создан, используется команда file.close, чтобы закрыть файл и убедиться, что он и все записанные в него данные сохранены на SD-карту.

```
File1.close();
```

Затем пользователю сообщают, что программа собирается записать текст в файл.

```
Serial.println("Writing text to file.....");        Serial.println ("Запись текста в файл .....");
```

Нам нужна строка символов для записи в текстовый файл, поэтому используется тип данных String (массив символов), и мы инициализируем эту переменную с именем dataString.

```
String dataString;
```

Затем мы снова открываем файл для записи и назначаем его объекту File File1.

```
File1 = SD.open("testfile.txt", FILE_WRITE);
```

Затем мы используем оператор if, чтобы проверить, что файл был успешно открыт, и если да, то используем оператор for для циклического перебора цифр от 1 до 10, которые присоединяются (добавляются к) к концу строки «Test Line»; затем мы печатаем эту строку в файл так же, как мы бы записывали строки в окно последовательного монитора, используя команду File1.println (dataString) . File1.println () добавит перевод строки и маркер новой строки в конец строки данных. Если вы не хотели этого делать, используйте команду File1.print () вместо File1.println().

```
if (File1) {
    for (int i=1; i<11; i++) {
      dataString = "Test Line ";
      dataString += i;
      File1.println(dataString);
    }
```

Если файл был открыт и записан успешно, мы информируем пользователя (это внутри оператора if, поэтому будет выполняться только в случае истины).

```
    Serial.println("Text written to file.....\n");
}
```

Наконец, файл закрывается. Данные записываются только после закрытия. Если вы хотите записать в файл, не закрывая его, вы можете использовать вместо этого функцию .flush ().

```
File1.close();
```

Теперь, чтобы подтвердить, что данные были записаны в текстовый файл, программа распечатает данные внутри файла, чтобы пользователь мог их увидеть. Мы начинаем с того, что информируем пользователя о том, что мы это делаем.

```
Serial.println("Reading СОДЕРЖАНИЕ of
textfile.txt...");
```

Затем файл снова открывается.

```
File1 = SD.open("testfile.txt");
```

Если открытие файла прошло успешно

```
if (File1) {
```

Убедитесь, что есть несколько байтов, доступных для чтения из файла (т. е. он не пустой), и пока есть доступные байты.

```
while (File1.available()) {
```

Считайте следующий доступный символ, используя функцию .read (), и запишите его в окно МПП

```
  Serial.write(File1.read());
}
```

Если открытие файла прошло успешно, и после того, как данные были прочитаны, закройте файл.

```
  File1.close();
}
```

Если файл не открылся, сообщите об этом пользователю.

```
else {
  Serial.println("error opening testfile.txt");
}
```

Теперь, когда мы создали файл и написали и прочитали из него, файл будет удален. Сначала пользователь информируется.

```
Serial.println("\nDeleting testfile.txt...\n");
```

Затем мы используем функцию SD.remove (), параметром которой является имя файла, который мы хотим удалить.

```
SD.remove("testfile.txt");
```

Затем, наконец, программа проверит, существует ли файл или нет, и проинформирует пользователя.

```
  if (SD.exists("testfile.txt")){
    Serial.println("testfile.txt still exists.");       Serial.println ("testfile.txt все еще существует.");
  }
  else {
    Serial.println("testfile.txt has been deleted."); Serial.println ("testfile.txt удален.");
  }
}
```

Поскольку мы хотели, чтобы приведенные выше команды выполнялись один раз и только один раз, весь код был помещен в функцию setup (). Поэтому функция main loop () остается пустой.

```
void loop()
{
  // Nothing to see here
}
```

В приведенном выше примере показан основной метод создания файла, записи некоторых строк в файл, чтения файла, закрытия файла и последующего его удаления. Теперь вы расширите эти знания и примените их на практике, используя SD-карту для регистрации некоторых данных с датчика температуры.

Проект 43 - Регистратор данных температуры SD

Теперь мы добавим в схему несколько датчиков температуры DS18B20 вместе с микросхемой DS1307 RTC (часы реального времени).

Показания датчиков температуры будут записаны на SD-карту, и вы будете использовать чип RTC для считывания даты и времени, чтобы показания датчиков и изменения файла могли иметь отметку времени. Мы используем два датчика температуры, один из которых предназначен для установки внутри вашего дома, а другой - с помощью длинного кабеля, размещенного снаружи. Однако для целей тестирования вы можете оставить их оба на макетной плате. Как только схема и код заработают, вы можете припаять длинный комплект кабелей ко второму датчику и разместить его снаружи, чтобы измерить внешнюю температуру.

Требуемые детали

Таб. 15-3. *Детали, необходимые для проекта 43*

SD-карта и разъем	
3 резистора по 3,3 кОм	
3 резистора по 1,8 кОм	
Резистор 4,7 кОм	
2 резистора по 1 кОм	
DS1307 RTC IC	
Кварц 32,768 кГц	
2 × DS18B20 Темп. датчики	
Выключатель	
*Держатель батарейки**	

Батарейка не является необходимым компонентом. Просто ее наличие позволит вам сохранять время и дату в часах реального времени, даже когда вы выключите свой проект. Вы также можете заменить датчики DS18S20 на DS18B20.

Единственное отличие состоит в том, что DS18S20 имеет разрешение по температуре 9 бит, тогда как DS18B20 имеет разрешение от 9 до 12 бит. Любой из них достаточно точен для наших целей.

Выключатель позволяет безопасно извлекать и вставлять карту в считывающее устройство. Программа отобразит через МПП, когда отключение питания безопасно , а затем отсоединит карту. При повторной установке карты вставьте карту в слот, прежде чем снова включить ее с помощью выключателя.

Подключение

Подключите все, как показано на рис. 15-2.

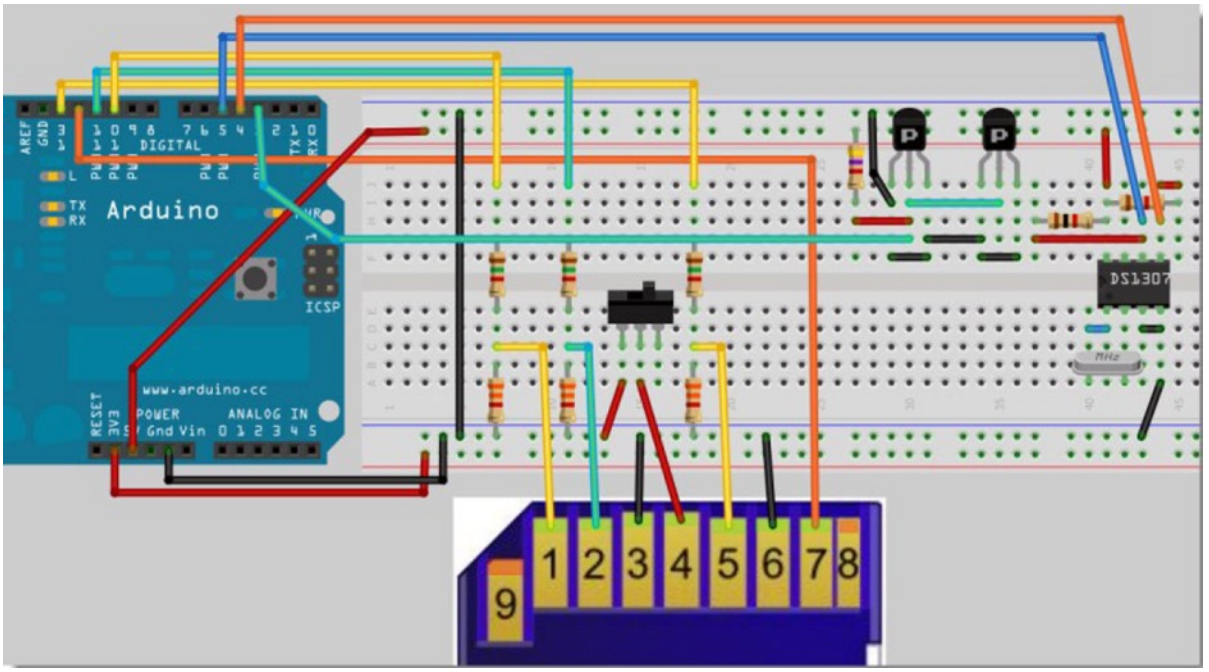


Рис. 15-2. *Схема для проекта 43 - Регистратор данных температуры SD*

Обратитесь к Таблице 15-4 для правильных выводов контактов. Подключите DS18B20s точно так же, как в Project 37. Если вы используете резервную батарею типа «таблетка», не прокладывайте провод между контактами 3 и 4 на RTC, как показано на схеме. Вместо этого подключите положительный полюс аккумулятора к контакту 3 микросхемы, а отрицательный - к контакту 4.

Таб. 15-4. *Соединения между Arduino, SD-картой и RTCRTC для SD-карты Arduino*

Arduino	SD Card	RTC
+5V	-	Pin 8
+3.3V	Pin 4 (VCC)	
Gnd	Pins 3 & 6 (GND)	Pins 3 & 4
Digital Pin 13 (SCK)	Pin 5 (CLK)	
Digital Pin 12 (MISO)	Pin 7 (DO)	
Digital Pin 11 (MOSI)	Pin 2 (DI)	
Digital Pin 10 (SS)	Pin 1 (CS)	
Digital Pin 4 (SDA)		Pin 5
Digital Pin 5 (SCL)		Pin 6

Поместите резисторы 1 кОм между контактом 8 и контактами 5 и 6 RTC.

Введите код

Убедитесь, что библиотеки OneWire.h и DallasTempera.h, используемые в Project 37, установлены для этого проекта:www.pjrc.com/teensy/arduino_libraries/OneWire.zip и http://download.milesburton.com/Arduino/MaximTemperature / DallasTemperature_372Beta.zip. Нам также понадобится библиотека SD.h. Мы также будем использовать библиотеку DS1307.h Хеннинга Карлсена с сайта henningkarlsen.com для управления микросхемой DS1307: http://www.henningkarlsen.com/electronics/ Убедитесь, что имя папки библиотеки соответствует заголовкам и файлам кода в них.

Listing 15-2. Код для проекта 43

```
// Проект 43
// Библиотека DS1307 Хеннинга Карлсена

#include <SD.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <DS1307.h> // написано Хеннингом Карлсеном

#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

File File1;

// Настраиваем экземпляр oneWire для связи с любыми устройствами OneWire
OneWire oneWire(ONE_WIRE_BUS);
// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств
DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };
DeviceAddress outsideThermometer = { 0x28, 0xA5, 0x02, 0xC2, 0x03, 0x00, 0x00, 0xF0 };

  float tempC, tempF;

  // Запускаем DS1307
DS1307 rtc(4, 5);

void setup() {
 Serial.println("Initializing the SD Card...");

  if (!SD.begin()) {
    Serial.println("Initialization Failed!");
    return;
  }
  Serial.println("Initialization Complete.\n");

  // Устанавливаем часы в рабочий режим
  rtc.halt(false);
```

```
  Serial.begin(9600);
  Serial.println("Type any character to start");
  while (!Serial.available());
  Serial.println();

  // Запустите библиотеку датчиков
  sensors.begin();
  Serial.println("Initialising Sensors.\n");

  // устанавливаем разрешение
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  delay(100);

  // Устанавливаем время на RTC.
  // Закомментируйте этот раздел,если вы уже установили время и имеете резервную батарею
  // Следующие строки закомментировать,чтобы использовать значения,уже сохраненные в DS1307
  rtc.setDOW(TUESDAY);         // Set Day-of-Week to TUESDAY
  rtc.setTime(9, 27, 00);      // Set the time HH,MM,SS
  rtc.setDate(30, 04, 2013);   // Set the date DD,MM,YYYY
}

void getTemperature(DeviceAddress deviceAddress)
{
  sensors.requestTemperatures();
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}

void loop() {
    File1 = SD.open("TEMPLOG.txt", FILE_WRITE);
    Serial.println("File Opened.");
    if (File1) {
          File1.print(rtc.getDateStr());
          Serial.print(rtc.getDateStr());
          File1.print(", ");
          Serial.print(", ");
          File1.print(rtc.getTimeStr());
          Serial.print(rtc.getTimeStr());
          File1.print(":  Inside: ");
          Serial.print(":  Inside: ");
          getTemperature(insideThermometer);
          File1.print(tempC);
          Serial.print(tempC);
          File1.print("C  Outside: ");
          Serial.print("C  Outside: ");
          getTemperature(outsideThermometer);
          File1.print(tempC);
          Serial.print(tempC);
          File1.println(" C");
```

```
            Serial.println(" C");
            Serial.println("Data written to file.");
        }
        File1.close();
        Serial.println("File Closed.\n");
        Serial.println("Safe to disconnect card");
        delay(10000);
        Serial.println("Card in use, do not disconnect!!");
}
```

Откройте МПП, и программа попросит вас ввести символ для запуска кода. После этого вы получите примерно такой результат:

```
Type any character to start    Введите любой символ,'чтобы начать
```

Инициализация датчиков.

```
File Opened.
30.04.2013, 10:11:27:  Inside: 25.94C  Outside: 26.00 C
Data written to file.
File Closed.

File Opened.
30.04.2013, 10:11:39:  Inside: 25.94C  Outside: 26.00 C
Data written to file.
File Closed.
```

Если вы дождетесь появления сообщения «Файл закрыт», выключите Arduino и извлеките SD-карту, а затем вставьте ее вна вашем ПК или Mac вы увидите файл TEMPLOG.TXT. Откройте этот файл в текстовом редакторе, и вы увидите показания датчиков с отметками времени, которые выглядят так:

```
30.04.2013, 09:49:06:  Inside: 25.56C  Outside: 25.56 C
30.04.2013, 09:49:18:  Inside: 25.56C  Outside: 25.62 C
30.04.2013, 09:49:29:  Inside: 25.62C  Outside: 25.62 C
30.04.2013, 09:49:41:  Inside: 25.62C  Outside: 25.69 C
30.04.2013, 10:11:27:  Inside: 25.94C  Outside: 26.00 C
30.04.2013, 10:11:39:  Inside: 25.94C  Outside: 26.00 C
30.04.2013, 10:11:50:  Inside: 26.81C  Outside: 27.00 C
30.04.2013, 10:12:02:  Inside: 28.00C  Outside: 27.94 C
```

Посмотрим, как работает программа.

Проект 43 - Регистратор данных температуры SD - Обзор кода

Поскольку большая часть этого кода описана в проектах 37 и 42, я сконцентрируюсь на новых битах.Во-первых, включены соответствующие библиотеки:

```
#include <SD.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <DS1307.h> // written by Henningh Karlsen
```

Новым здесь является библиотека DS1307.h Хеннинга Карлсена. Эта библиотека позволяет нам читать и записывать дату и время в микросхему DS1307 RTC. Нам это понадобится, чтобы сначала установить дату и время на микросхеме. После установки чип будет отсчитывать каждую секунду и соответственно обновлять внутренние настройки времени и даты. Затем мы можем прочитать эти данные с чипа, чтобы получить точную временную метку для нашего журнала температуры.Затем мы определяем, какой вывод мы используем для шины One Wire; этот вывод используется для связи с датчиками температуры. Затем мы определяем, с какой точностью должна быть установлена температура в датчике DS18B20 (не действует для DS18S20 с фиксированной точностью).

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Затем мы создаем экземпляр файла, который хотим записать на SD-карту, и называем его File1.

```
File File1;
```

Затем создается экземпляр объекта oneWire, который называется OneWire. Это позволяет нам связываться с устройствами OneWire, в данном случае с датчиками температуры.

```
OneWire oneWire(ONE_WIRE_BUS);
```

Затем ссылка на шину oneWire передается в библиотеку температуры Далласа, чтобы она могла получить доступ к датчикам.

```
DallasTemperature sensors(&oneWire);
```

Адреса двух датчиков температуры, которые были обнаружены ранее в Проекте 37, Часть 1, затем сохраняются в переменных DeviceAddress. DeviceAddress - это тип данных, созданный в библиотеке OneWire для хранения адресов устройств One Wire.

```
DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };
DeviceAddress outsideThermometer = { 0x28, 0xA5, 0x02, 0xC2, 0x03, 0x00, 0x00, 0xF0 };
```

Нам нужно сохранять значения температуры от датчиков, поэтому мы создаем переменные типа float для хранения этой температуры в градусах Цельсия и Фаренгейта.

```
float tempC, tempF;
```

Нам нужно сообщить библиотеке DS1307, какие контакты используются для связи с микросхемой RTC. Это выводы SDA (последовательные данные) и SCL (такты) на DS1307. Это цифровые выводы 4 и 5.

```
DS1307 rtc(4, 5);
```

Теперь в функции настройки мы подготовили датчики температуры и микросхему RTC. Мы начинаем с того, что через МПП информируем пользователя о том, что программа собирается инициализировать SD-карту.

```
void setup() {
 Serial.println("Initializing the SD Card...");
```

Если SD-карта не инициализируется успешно, сообщите об этом пользователю и вернитесь из функции, которая выйдет из программы.

```
if (!SD.begin()) {
  Serial.println("Initialization Failed!");
  return;
}
```

В противном случае сообщите пользователю, что инициализация SD-карты прошла успешно.

```
Serial.println("Initialization Complete.\n");
```

Микросхема DS1307 имеет флаг CH (Clock Halt), который необходимо установить либо в TRUE, чтобы часы не тикали, либо в FALSE, чтобы начать их отсчет. Поскольку нам нужно убедиться, что устройство весело тикает, флаг установлен в FALSE.

```
rtc.halt(false);
```

Начато последовательное соединение, что позволяет нам записывать данные на МПП

```
Serial.begin(9600);
```

Затем пользователя просят ввести любой символ, чтобы начать.

```
Serial.println("Type any character to start");
```

Затем мы проверяем, доступны ли какие-либо данные в строке последовательных данных, то есть пользователь ввел символ. В противном случае функция while будет продолжать проверку, пока что то не появится.

```
while (!Serial.available());
```

Распечатайте возврат строки.

```
Serial.println();
```

Библиотека датчиков инициализируется функцией Sensors.begin ().

```
sensors.begin();
```

и пользователь проинформирован.

```
Serial.println("Initialising Sensors.\n");
```

Датчик DS18B20 может иметь разрешение от 9 до 12 бит. Это невозможно с датчиком DS18S20 с фиксированным битовым разрешением. Мы используем функцию .setResolution () библиотеки температуры Далласа, чтобы установить разрешение, передав функции адрес датчика и разрешение в битах, а затем введем небольшую задержку, чтобы данные были записаны.

```
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
delay(100);
```

День недели, время и дату нужно выставить на микросхеме DS1307. Мы будем использовать функцию setDOW (), чтобы установить день недели. Введите день заглавными буквами в скобки параметров функций.

```
rtc.setDOW(TUESDAY);
```

Далее функция setTime () используется для установки времени в формате ЧЧ, ММ, СС.

```
rtc.setTime(9, 27, 00);
```

Затем установите дату в формате ДД, ММ, ГГГГ с помощью функции setDate ().

```
  rtc.setDate(30, 04, 2013);
}
```

Теперь мы создаем функцию для получения от датчиков температуры в градусах Цельсия и Фаренгейта. Функция требует адреса устройства.

```
void getTemperature(DeviceAddress deviceAddress)
{
```

Также мы используем функцию requestTemperatures () библиотеки Dallas Temperature для получения температуры.

```
sensors.requestTemperatures();
```

Мы сохраняем температуру в tempC с помощью функции getTempC (), которая извлекает температуру в градусахЦельсия.

```
tempC = sensors.getTempC(deviceAddress);
```

а затем используем функцию toFahrenheit () библиотеки температуры Далласа, чтобы преобразовать эту температуру в градусы Фаренгейта и сохранить ее в tempF.

```
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

Теперь у нас есть основной цикл программы.

```
void loop() {
```

Мы начинаем с открытия или создания файла на SD-карте, который мы назовем TEMPLOG.txt.

```
File1 = SD.open("TEMPLOG.txt", FILE_WRITE);
```

Сообщаем пользователю, что файл был открыт.

```
Serial.println("File Opened.");
```

Если файл был открыт или успешно создан

```
if (File1) {
```

то мы записываем соответствующие строки в текстовый файл. Начнем с получения строки даты с помощью функции rtc.getDateStr () библиотеки DS1307 и записи ее в файл.

```
File1.print(rtc.getDateStr());
```

Те же данные распечатываются на МПП, чтобы пользователь мог видеть, что происходит.

```
Serial.print(rtc.getDateStr());
```

За этой датой следует запятая и пробел.

```
File1.print(", ");
Serial.print(", ");
```

Затем мы получаем строку времени и записываем ее в файл.

```
File1.print(rtc.getTimeStr());
Serial.print(rtc.getTimeStr());
```

Затем идет внутренняя температура, поэтому мы даем ей метку.

```
File1.print(":  Inside: ");
Serial.print(":  Inside: ");
```

Получите последнюю информацию о температуре от внутреннего датчика температуры.

```
getTemperature(insideThermometer);
```

и запишите его в файл.

```
File1.print(tempC);
Serial.print(tempC);
```

Если вы хотите, чтобы ваша температура регистрировалась в градусах Фаренгейта, просто измените tempC на tempF. Затем мы печатаем C (или, возможно, F), а затем маркируем наружную температуру.

```
File1.print("C  Outside: ");
  Serial.print("C  Outside: ");
```

Затем мы получаем показания внешнего датчика и записываем его в файл.

```
getTemperature(outsideThermometer);
File1.print(tempC);
Serial.print(tempC);
File1.println(" C");
Serial.println(" C");
```

Все вышеперечисленные команды печати вводят строку сразу после последней. Когда мы используем println, то в конце строки вводится команда новой строки, чтобы гарантировать, что следующий набор данных будет на следующей строке.Наконец, пользователю сообщается, что данные были записаны в файл.

```
Serial.println("Data written to file.");
    }
```

Файл закрывается, что приводит к записи всех вышеперечисленных данных в файл, и пользователь информируется.

```
File1.close();
Serial.println("File Closed.\n");
```

И, наконец, программа будет ждать 10 секунд перед следующим чтением. Очевидно, мы можем изменить это значение на любой желаемый интервал.

```
    delay(10000);
}
```

Теперь у нас есть основное представление о том, как записывать данные датчика или другие данные на SD-карту. Для получения дополнительных сведений см. Документацию, поставляемую с библиотекой SD.h. Теперь мы посмотрим на микросхему RTC, прежде чем перейти к следующей главе.

Проект 43 - Регистратор данных температуры SD - Обзор устройства

В Проекте 43 мы познакомились с микросхемой часов реального времени DS1307. Это отличная маленькая ИС, которая позволяет легко добавлять часы в свои проекты. С добавлением резервного аккумулятора типа «таблетка» мы можете отключить Arduino от источника питания, и чип автоматически переключится на резервный аккумулятор и будет обновлять свои данные и время, используя аккумулятор. Благодаря кварцу хорошего качества устройство будет показывать достаточно точное время. Устройство даже настраивается на високосные годы и на месяцы с числом дней меньше 31. Его также можно настроить на работу в 24-часовом или 12-часовом режиме. Связь сустройством осуществляется через интерфейс I2C, о котором я вскоре расскажу. Чип интересен тем, чтоон также имеет прямоугольный сигнал на выводе 7. Это может быть 1 Гц , 4,096 кГц, 8,192 кГц или 32,768 кГц. Поэтому мы также можете использовать его в качестве генератора звука или других целей, требующих импульсного сигнала. Вы можете легко добавить светодиод к этому выводу, чтобы он мигал с частотой 1 Гц.

Связь с микросхемой осуществляется по протоколу I 2 C. До сих пор мы сталкивались с однопроводной связью и SPI, но это новый протокол. Чтобы использовать протокол I 2 C, нам необходимо включить в свой код библиотеку Wire.h. Однако DS1307.h был написан для использования протокола без библиотеки Wire.h, поэтому в этом случае мы не будем его использовать.

Протокол I2C (иногда также называемый TWI или двухпроводным интерфейсом) был разработан Philips Semiconductors (теперь известный как NXP) для создания простой двунаправленной шины, использующей всего два провода для управления между ИС. В протоколе используется всего два провода: линия последовательной передачи данных (SDA) и линия последовательной синхронизации (SCL). Однако библиотека DS1307 использует собственный набор определяемых пользователем контактов. Единственно необходимый внешний компонент - это подтягивающий резистор на каждой линии шины. Вы можете подключить до 128 устройств (или узлов) I2C к одним и тем же двум проводам. Некоторые устройства I2C используют + 5 В, а другие + 3,3 В, так что при их использовании следует соблюдать осторожность. Убедитесь, что вы прочитали техническое описание и используете соответствующее напряжение перед подключением устройства I2C. Если вы когда-нибудь хотели, чтобы два Arduino "разговаривали" друг с другом, то I2C будет хорошим протоколом для использования. Протокол I2C похож на SPI в том, что есть ведущие и ведомые устройства. Arduino является ведущим, а устройство I2C - ведомым. Каждое устройство имеет свой собственный адрес I2C. Каждый бит данных отправляется с каждым тактовым импульсом.

Связь начинается, когда мастер выдает на шину условие START, и прекращается, когда мастер выдает условие STOP. Чтобы начать коммуникацию I 2 C на Arduino, мы вводим команду Wire.begin ().

Она инициализирует библиотеку Wire и Arduino как главное устройство I2C. Она также перенастраивает аналоговые выводы 4 и 5 на контакты I2C. Чтобы инициировать связь в качестве ведомого устройства (т.е. для двух Arduino, подключенных через I2C), адрес ведомого устройства должен быть включен в скобки. Другими словами,

Wire.begin(5);

заставит Arduino присоединиться к шине I 2 C в качестве ведомого устройства по адресу 5. Байт может бытьполучен от устройства I2C, используя

Int x = Wire. Receive();

Перед этим мы должны запросить количество байтов с помощью Wire.requestFrom (адрес, количество), поэтому

```
Wire.requestFrom(5,10);
```

будет запрашивать 10 байт с устройства 5. Отправка на устройство также проста с

```
Wire.beginTransmission(5);
```

который устанавливает устройство для передачи на устройство номер 5, а затем

```
Wire.send(x);
```

отправить один байт или

```
Wire.send("Wire test.");
```

отправить 10 байт.

Вы можете узнать больше о библиотеке Wire на www.arduino.cc/en/Reference/Wire и о I2C из Википедии. или прочитав превосходное объяснение I2C в таблицах данных Atmega на веб-сайте Atmel. Библиотека DS1307 от Хеннинга Карлсена использует собственный внутренний набор функций для связи через One Wire, поэтому эти функции не требуются.

Резюме

В главе 15 вы узнали основы чтения и записи на SD-карту. Вы можете узнать больше об использовании SD-карты с библиотекой SD.h, прочитав прилагаемую к ней документацию. Вы только что прикоснулись к проектам в этойглаве! Попутно вы познакомились с протоколом I 2 C. Вы также узнали, как подключить микросхему часовреального времени DS1307, которая будет очень полезна для ваших собственных проектов на основе часов в будущем.Еще одинотличный способ получить очень точный сигнал времени - использовать дешевое устройствоGPS споследовательным выходом; вы также можете использовать модули приемников времени, такие как приемники времени MSF (Великобритания), WWVB (США) или DCF (Европа), для получения абсолютно точного времени.
Умение читать и записывать на SD-карту - жизненно важная часть знаний для создания регистраторов данных, особенно для удаленных устройств с батарейным питанием. Многие из проектов High Altitude Balloon (HAB), основанных на чипах Arduino или AVR, используют SD-карты для регистрации данных GPS и данных датчиков для извлечения, когда воздушный шар оказывается на земле.

Предметы и понятия, затронутые в главе 15

- Как подключить SD-карту к Arduino
- Использование схем делителя напряжения с резисторами для понижения уровня напряжения с + 5В до + 3,3В
- Как использовать библиотеку SD.h
- Запись строк и чисел в файлы
- Открытие и закрытие файлов
- Именование файлов
- Создание меток времени файлов
- Выявление файловых ошибок
- Понятия цикла do-while
- Как подключить микросхему часов реального времени DS1307 к Arduino
- Как использовать библиотеку DS1307.h для установки и получения времени и даты
- Использование резервного аккумулятора на DS1307 для сохранения данных после отключения питания
- Введение в протокол I2C

ГЛАВА 16

Создание считывателя RFID

Считыватели RFID (радиочастотной идентификации) сегодня довольно распространены. Они являются предпочтительным методом контроля доступа в офисные блоки и для систем входа в общественный транспорт. Маленькие RFID-метки вводятся животным для идентификации в случае их утери. На транспортных средствах есть бирки для взимания платы за проезд. Они даже используются в больницах для маркировки одноразового оборудования в операционных, чтобы гарантировать, что внутри пациентов не останется посторонних предметов. Стоимость технологии за последние годы резко снизилась до такой степени, что теперь считыватели можно приобрести менее чем за 10 долларов.Их легко подключить к Arduino и легко использовать. В результате из них можно создавать всевозможные крутые проекты.
Мы будем использовать легко доступный и дешевый считыватель ID-12 от Innovations. Эти считыватели используют технологию 125 кГц для считывания меток и карт на расстоянии до 180 мм от считывателя. В том же диапазоне есть и другие считыватели, которые дают больший диапазон, и с добавлением внешней антенны вы можете еще больше увеличить диапазон.
Мы начнем с того, что подключите один считыватель и узнаете, как легко получить от него последовательные данные. Затем мы создадим простую систему контроля доступа.

Проект 44 - Простой считыватель RFID

Контакты на считывателях ID12 имеют нестандартное расстояние, поэтому они не поместятся в макетную плату. Вам нужно будет получить коммутационную плату у такого поставщика, как Sparkfun. Карты или бирки можно купить из любых источников, и они очень дешевы; Я купил на eBay сумку с маленькими брелками в стиле цепочки для ключей за несколько долларов. Убедитесь, что метка или карта изготовлены по технологии 125 кГц, иначе они не будут работать с этим считывателем.

Требуемые детали

Таб. 16-1. *Детали, необходимые для проекта 44*

Считыватель RFID ID-12	
Коммутационная плата ID12 *	

(*продолжение*)

Таб. 16-1. (продолжение)

Токоограничивающий резистор	
мм светодиод	
RFID-метки или карты 125 кГц * (Минимум 4)	

Подключение

Подключите все, как показано на рис. 16-1.

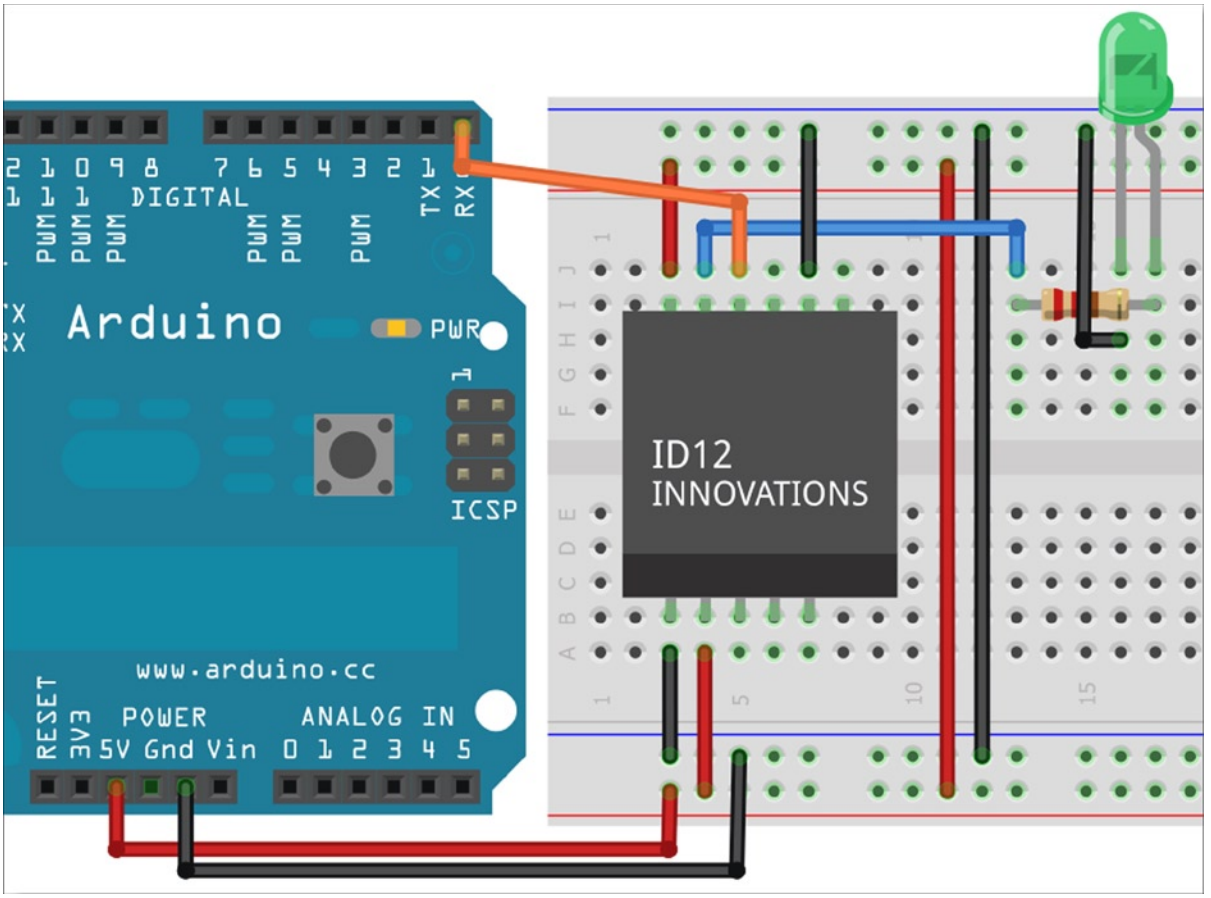


Рис. 16-1. *Схема для проекта 44 - Простой считыватель RFID*

Подключите светодиод через токоограничивающий резистор к контакту 10 (BZ) считывателя.

Таб. 16-2. *Контактные соединения от Arduino к ID12*

Arduino	ID12
5v	Pin 2 (RST)
5v	Pin 11 (V)
Gnd	Pin 1 (GND)
Gnd	Pin 7 (FS)
RX	Pin 9 (DO)

Также подключите светодиод через резистор 1 кОм к контакту 10 (BZ) в ID12.

Введите код

Введите код из листинга 16-1.

Листинг 16-1. Код для проекта 44.

```
char val = 0; // чтение значения для последовательного порта

void setup() {
    Serial.begin(9600);
}

void loop () {

  if(Serial.available() > 0) {
  val = Serial.read(); // читаем из последовательного порта
  Serial.write(val);   // и выводим на монитор
  }
}
```

Перед загрузкой кода отключите провод RX. Подключите его перед запуском кода. Запустите код, откройте МПП и поднесите RFID-метку или карту к считывателю. Светодиод будет мигать, показывая, что карта считана, а в окне МПП отобразится 12-значный номер, который составляет уникальный идентификатор карты. Запишите идентификаторы ваших тегов, поскольку они понадобятся вам в следующем проекте.

Проект 44 - Простой считыватель RFID - Обзор кода

Давайте посмотрим, как работает эта простая программа. Нам нужно будет где-то хранить символы, считанные из считывателя ID12, поэтому мы создаем переменную типа char для ее хранения.

```
char val = 0; // чтение значения для последовательного порта
```

Функция настройки просто подготавливает последовательную связь со скоростью 9600 бод.

```
void setup() {
    Serial.begin(9600);
}
```

In the loop function, we check if data is available to be read with the Serial.available command.

```
void loop () {

  if(Serial.available() > 0) {
```

Если да, сохраняем прочитанный байт в val.

```
  val = Serial.read(); // читаем из последовательного порта
```

Затем распечатайте это в окне МПП

```
  Serial.write(val); // и выводим на монитор
  }
}
```

Эта программа очень проста. Он считывает один байт из считывателя ID12 (когда карта удерживается в считывателе), а затем выводит это значение в окно последовательного монитора (МПП).

Проект 44 - Простой считыватель RFID - Обзор устройства

RFID есть везде, от вашего автобусного билета до дверей, через которые вы попадете в офис или колледж. Бирки или карточки бывают самых разных форм и размеров (см. Рис. 16-2), и их можно сделать настолько маленькими, что ученые даже прикрепили RFID-метки к муравьям, чтобы отслеживать их перемещения. Это простые устройства, которые ничего не делают, кроме как передают уникальный серийный код по радиоволнам. В большинстве случаев карты или метки пассивны, то есть в них нет батареи и требуется питание от внешнего источника. Другие варианты включают пассивную RFID, которая имеет собственный источник питания, и пассивную систему с батарейным питанием (BAP), которая ожидает, пока внешний источник разбудит его, а затем использует свой собственный источник питания для передачи, обеспечивая больший диапазон.

Рис. 16-2. *RFID-метки и карты бывают всех форм и размеров*

Мы используем пассивный тип без батареи. Они получают ток от магнитного поля, передаваемого считывателем. Когда метка попадает в магнитное поле, оно индуцирует ток в обмотке внутри метки. Этот ток используется для пробуждения крошечной микросхемы, которая передает серийный номер метки. Затем считыватель отправляет эти данные в последовательном формате на подключенный к нему ПК или микроконтроллер. Формат данных, отправляемых считывателем ID12, следующий:

STX (02h)	DATA (10 ASCII)	CHECKSUM (2 ASCII)	CR	LF	ETX (03H)

Сначала отправляется STX или символ начала передачи (ASCII 02), за которым следуют 10 байтов, которые составляют отдельные HEX (шестнадцатеричные) цифры числа. Следующие две шестнадцатеричные цифры - это контрольная сумма числа (это будет объяснено в следующем проекте), затем идет возврат каретки (CR) и перевод строки (LF), за которыми следует ETX или код конца передачи. На МПП будут отображаться только 12 цифр ASCII, так как остальные символы не печатаются. В следующем проекте мы будем использовать контрольную сумму, чтобы убедиться, что полученная строка верна, и код STX, чтобы сообщить нам, что строка отправляется.

Проект 45 - Система контроля доступа

Теперь мы создадим систему контроля доступа. Мы будем читать теги с помощью считывателя RFID и проверять выбранные теги, чтобы они могли открыть дверь. Затем Arduino через транзистор активирует электрический замок.

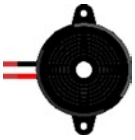
Требуемые детали

Таб. 16-3. *Детали, необходимые для проекта 45*

Считыватель RFID ID12	
Коммутационная плата ID12	
Токоограничивающий резистор	
Резистор 2,2 кОм	
5mm LED	

(*продолжение*)

Таб. 16-3. (*продолжение*)

RFID-метки или карты 125 кГц * (Минимум 4)	
1N4001 Диод	
Транзистор TIP-120 NPN	
Разъем питания 2,1 мм	
Источник питания 12 В постоянного тока	
Пьезозуммер	
Электрозамок 12 В	

Подключение

Подключите все, как показано на рисунке 16-3.

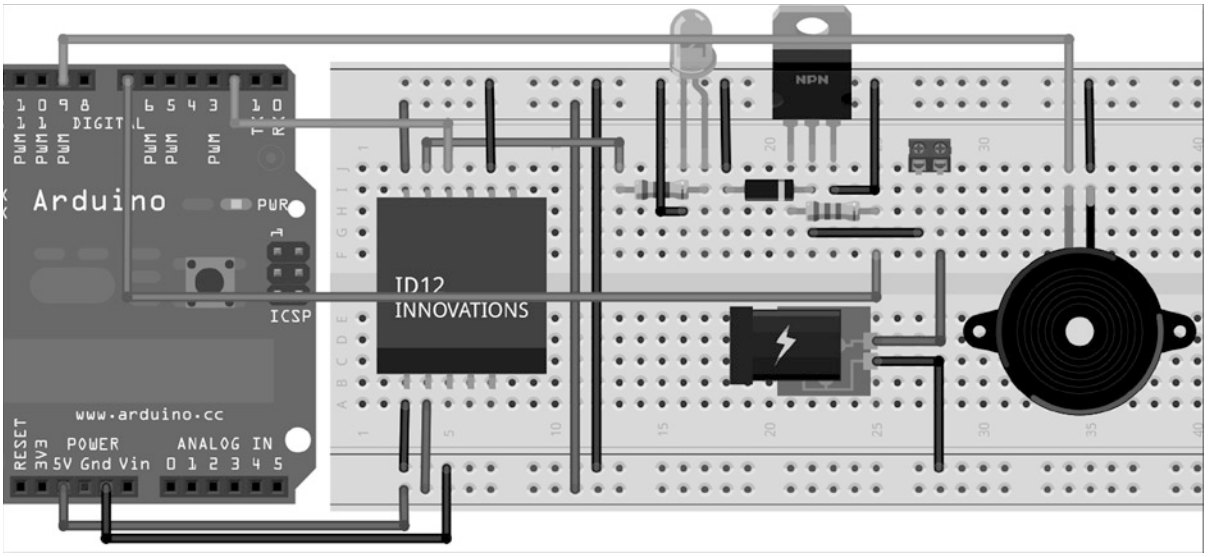


Рис. 16-3. *Схема для проекта 45 - Система контроля доступа*

Слева направо на TIP120 находятся база, коллектор и эмиттер. База подключается к цифровому выводу 7 через резистор 2,2 кОм, коллектор подключается к земле через диод, а также к отрицательной клемме пьезозуммера.
Параллельно обмотке соленоида электрозамка подключен диод 1N4001 катодом к +12В. Он необходим для защиты транзистора от пробоя напряжнием противоЭДС,возникающим при отключении питания 12В.

Электропитание замка должно поступать от внешнего источника постоянного тока 12 В с номиналом не менее 500 мА.
Подключите провода от внешнего источника питания к электрозамку. Источник питания 12 В подается на разъем 12 В на замке, а провод земли идет через средний вывод TIP-120 к контакту земли на замке.

Введите код

Введите код с листинга 16-2.

Листинг 16-2. Код для проекта 45

```
// Проект 45

#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000

#include <SoftwareSerial.h>

SoftwareSerial rfidReader = SoftwareSerial(rx, tx);

int users = 3;

char* cards[] = { // действительные карты
  "3D00768B53",
  "3D00251C27",
  "3D0029E6BF",
};

char* names[] = { // имена держателей карт
  "Tom Smith",
  "Dick Jones",
  "Harry Roberts"
};

void setup() {
  pinMode (lockPin, OUTPUT);
  pinMode (speakerPin, OUTPUT);
  digitalWrite(lockPin, LOW);
  Serial.begin(9600);
  rfidReader.begin(9600);
}
```

```
void loop() {
  char cardNum[10];  // массив для хранения номера карты
  byte cardBytes[6]; // байтовая версия номера карты + контрольная сумма
  int index=0;       // текущая цифра
  byte byteIn=0;     // чтение байта из RFID
  byte lastByte=0;   // прочитанный последний байт
  byte checksum = 0; // результат контрольной суммы хранится здесь

  if (rfidReader.read()==2) {         // читать считыватель RFID
      while(index<12) {               // 12 цифр в уникальном серийном номере
          byteIn = rfidReader.read(); // сохраняем значение в byteIn
          if ((byteIn==1) || (byteIn==2) || (byteIn==10) || (byteIn==13)) {return;}
           //если STX, ETX, CR или LF прерываются
          if (index<10) {cardNum[index]=byteIn;} //сохраняем только первые 10 шестнадцатеричных цифр
          (последние 2 - контрольная сумма))
          // преобразование шестнадцатеричного ascii в целое шестнадцатеричное значение
          if ((byteIn>='0') && (byteIn<='9')) {byteIn -= '0';
          }
          else if ((byteIn>='A') && (byteIn<='F')) {
          byteIn = (byteIn+10)-'A';
          }
          if ((index & 1) == 1) { // если нечетное число объединяет 2 4-битные цифры в 8-битный байт
                 // перемещаем последнюю цифру на 4 бита влево и добавляем новую цифру
                 cardBytes[index/2]= (byteIn | (lastByte<<4));
                 if (index<10) {checksum ^= cardBytes[index/2];} // суммируем значение контрольной суммы
          }
          lastByte=byteIn; // сохраняем последний прочитанный байт
          index++; // увеличиваем индекс
          // если мы дошли до конца всех цифр, добавляем нулевой терминатор
          if (index==12) {cardNum[10] = '\0';}
    }

   Serial.println(cardNum);                      // выводим номер карты
   int cardIndex =checkCard(cardNum);  // проверяем действительность карты и возвращаем порядковый номер
   if(cardIndex>=0 && (cardBytes[5]==checksum)) { // если номер карты и контрольная сумма действительны
        Serial.println("Card Validated");
        Serial.print("User: ");
        Serial.println(names[cardIndex]);         // выводим соответствующее имя
        unlock();                                 // отпираем дверь
        Serial.println();
    }
    else {
        Serial.println("Card INVALID");
        tone(speakerPin, 250, 250);
        delay(250);
        tone(speakerPin, 150, 250);
        Serial.println();
    }
  }
}
```

```
// Проверяем обнаруженную карту по всем известным номерам действительных карт
//Возвращаем порядковый номер массива совпадающего номера карты
// или отрицательное значение для обозначения несовпадающего номера карты
int checkCard(char cardNum[10])
{
   for (int x=0; x<=users; x++)
   {
      if(strcmp(cardNum, cards[x])==0)
      {
         return (x);
      }
   }
   return (-1);
}

void unlock() {
    tone(speakerPin, 1000, 500);
    digitalWrite(lockPin, HIGH);
    delay(unlockLength);
    digitalWrite(lockPin, LOW);
}
```

Убедитесь, что номера кодов для трех ваших тегов введены в массив карт в начале программы. При необходимости используйте Project 44, чтобы узнать номера кодов (запустите код и откройте монитор последовательного порта). А теперь представьте читателю свои четыре карты. Считыватель мигнет своим светодиодом, показывая, что он прочитал карту, и вы получите результат, подобный этому:

```
3D00251C27
Card Validated
User: Dick Jones

3D0029E6BF
Card Validated
User: Harry Roberts

3D002A7C6C
Card INVALID

3D00768B53
Card Validated
User: Tom Smith
```

Отобразится номер карты, за которым следует либо «Card INVALID», либо «Card Validated», за которым следует имя пользователя. Если карта действительна, прозвучит высокий сигнал и замок откроется на две секунды. Если карта недействительна, раздастся низкий двухтональный звуковой сигнал, и дверь не откроется. Электрозащелка на 12 В питается от транзистора, обеспечивающего более высокое напряжение и ток, чем может обеспечить Arduino. Чтобы предотвратить перегорание TIP120, мы добавили диод для защиты от скачков напряжения, возникающих при отключении питания 12В. Мы использовали тот же принцип в главе 5, когда управляли двигателем постоянного тока. Посмотрим, как работает этот проект.

Проект 45 - Система контроля доступа - Обзор кода

Во-первых, у нас есть несколько определений контактов, которые будут использоваться для замка и динамика. Кроме того, мы используете библиотеку SoftwareSerial вместо обычных последовательных выводов на цифровых выводах 0 и 1 и поэтому мы должны определить выводы rx и tx.

У нас также есть время в микросекундах, на которое замок откроется.

```
#define lockPin 7
#define speakerPin 9
#define tx 3
#define rx 2
#define unlockLength 2000
```

Для удобства мы используем библиотеку SoftwareSerial (теперь это основная часть Arduino IDE). Если бы мы использовали стандартные выводы rx и tx, нам пришлось бы отключать все, что было подключено к этим выводам, каждый раз, когда мы хотели бы загрузить какой-либо код в Arduino. Используя библиотеку SoftwareSerial, мы можем использовать любой вывод, какой хотите.

```
#include <SoftwareSerial.h>
```

Затем вы создаете экземпляр объекта SoftwareSerial и называете его rfidReader. Мы передаем ему контакты rx и tx, которые мы определили.

```
SoftwareSerial rfidReader = SoftwareSerial(rx, tx);
```

Далее идет переменная для хранения количества пользователей в базе данных:

```
int users = 3;
```

Далее идут два массива для хранения идентификационных номеров карт и имен держателей карт. Измените номера карт на свои (только первые 10 цифр).

```
char* cards[] = { // действительные карты
  "3D00768B53",
  "3D00251C27",
  "3D0029E6BF",
};

char* names[] = { // имена держателей карт
  "Tom Smith",
  "Dick Jones",
  "Harry Roberts"
};
```

Программа настройки устанавливает контакты замка и динамика в качестве выходов.

```
pinMode (lockPin, OUTPUT);
pinMode (speakerPin, OUTPUT);
```

Затем программа установки устанавливает lock (стопорный) пин на НИЗКИЙ, чтобы не было разблокировки при старте.

```
digitalWrite(lockPin, LOW);
```

Затем мы связываемся через последовательный порт и порт SoftwareSerial:

```
Serial.begin(9600);
rfidReader.begin(9600);
```

Далее идет основной цикл. Мы начинаем с определения переменных, которые будут использоваться в цикле:

```
char cardNum[10];  // массив для хранения номера карты
byte cardBytes[6]; // байтовая версия номера карты + контрольная сумма
int index=0;       // текущая цифра
byte byteIn=0;     // чтение байта из RFID
byte lastByte=0;   // прочитанный последний байт
byte checksum = 0; // результат контрольной суммы хранится здесь
```

Затем мы проверяем, поступают ли какие-либо данные в последовательный порт считывателей RFID. Если это так, и первый символ - это ASCII-символ 2, который является кодом начала передачи, значит, мы знаем, что строка идентификатора собирается передать, и можно начать считывание цифр.

```
if (rfidReader.read()==2) {
```

Затем следует целый цикл, который будет выполняться, пока индекс меньше 12:

```
while(index<12) {
```

Переменная index будет содержать значение вашего текущего места в цифре, которую мы читам. Поскольку вы читаете цифру длиной 12 символов, мы будете считывать только первые 12 цифр.

Затем значение из последовательного порта считывается и сохраняется в byteIn:

```
byteIn = rfidReader.read();
```

На всякий случай, если по какой-либо причине были пропущены некоторые символы, мы снова проверяем наличие кодов начала, конца передачи, возврата каретки или перевода строки. Если они обнаружены, цикл завершается.

```
if ((byteIn==1) || (byteIn==2) || (byteIn==10) || (byteIn==13)) {return;} // if STX, ETX, CR or LF break
```

Последние две цифры 12-значной строки - это контрольная сумма. Мы не хотим хранить это в массиве cardNum, поэтому мы сохраняем только первые 10 цифр:

```
if (index<10) {cardNum[index]=byteIn;}
```

Затем нам нужно преобразовать символы ASCII, которые мы читаем, в их эквиваленты шестнадцатеричных чисел, поэтому мы запускаем оператор if-else, чтобы определить, находятся ли символы между 0 и 9 и A и Z. Если да, они преобразуются в их шестнадцатеричные числа - эквиваленты.

```
if ((byteIn>='0') && (byteIn<='9')) {
        byteIn -= '0';
}
else if ((byteIn>='A') && (byteIn<='F')) {
        byteIn = (byteIn+10)-'A';
}
```

Логические операторы И (&&) используются для обеспечения того, чтобы символы попадали между 0 и 9 или между A и Z. Затем вы конвертируете две последние шестнадцатеричные цифры в байт. Шестнадцатеричная цифра - это шестнадцать. Обычно мы используем десятичную систему счисления с цифрами в диапазоне от 0 до 9. В шестнадцатеричном формате цифры идут от 0 до 15. Буквы от A до F используются для чисел от 10 до 15. Итак, число FF в шестнадцатеричном формате равно 255 в десятичной системе счисления:

$$F = 15$$
$$(F * 16) + F = (15 * 16) + 15 = 255$$

Поэтому нам необходимо преобразовать две последние цифры ASCII в один байт и десятичный эквивалент. Мы уже объявили переменную с именем lastByte, в которой хранится последняя цифра, которую мы обработали при последнем запуске цикла while.

Первоначально он установлен на ноль. Это нужно делать только для каждой второй цифры, поскольку каждая из двух шестнадцатеричных цифр составляет один байт. Итак, мы проверяем, является ли индекс нечетным числом, выполнив побитовую операцию И (&) над значением, хранящимся в индексе с 1, и проверив, равен ли результат 1. Помните, что индекс начинается с нуля, поэтому вторая цифра имеет индекс 1.

```
if ((index & 1) == 1) {
```

Любое нечетное число, соединенное И (AND) с 1, приведет к 1, а любое четное число приведет к 0:

```
       12 (even) & 1 = 0
       00001100
       00000001 &
=      00000000

       11 (odd) & 1 = 1
       00001011
       00000001 &
=      00000001
```

Если результат определяет, что мы находимся на второй, четвертой, шестой, восьмой или двенадцатой цифре, то мы сохраняем в cardBytes результат следующего вычисления:

```
cardBytes[index/2]= (byteIn | (lastByte<<4)); // перемещаем последнюю цифру на 4 бита влево и добавляем новую цифру
```

Мы используем index / 2 для определения номера индекса. Поскольку индекс является целым числом, будет сохранено только значение до десятичной точки. Таким образом, для каждых двух цифр, для которых индекс увеличивает индекс, массив cardBytes будет увеличиваться на единицу.
Вычисление берет последнее значение байта и сдвигает его на четыре позиции влево. Затем оно берет это число и выполняет с ним операцию поразрядного ИЛИ (|) с текущим считанным значением. Это приводит к тому, что берется первое шестнадцатеричное значение, которое составляет первые четыре бита числа, и сдвигается на четыре позиции влево. Затем он объединяет это число со второй шестнадцатеричной цифрой, чтобы получить полный байт. Итак, если первая цифра была 9, а вторая - E, вычисление сделало бы следующее:

```
Lastbyte = 9 =  00001001
00001001 << 4 = 10010000
E = 14 =        00001110
                10010000 OR
=               10011110
```

Контрольная сумма - это число, которое вы используете, чтобы убедиться, что вся строка была прочитана правильно. Контрольные суммы часто используются при передаче данных; они просто результат совмещения каждого байта всей строки данных с помощью операции XOR (исключающее ИЛИ).

```
if (index<10) {checksum ^= cardBytes[index/2];}
```

Идентификационный номер вашей первой карты:

```
3D00768B53
```

Следовательно, его контрольная сумма будет:

```
3D XOR 00 XOR 76 XOR 8B XOR 53

3D = 00111101
00 = 00000000
76 = 01110110
8B = 10001011
53 = 01010011
```

Если мы выполним XOR для каждой из этих цифр друг с другом, мы получим 93. Итак, 93 - это контрольная сумма для этого идентификационного номера. Если какая-либо из цифр была передана неправильно из-за помех, контрольная сумма будет отличаться от 93 и мы будем знать, что карта была прочитана неправильно и мы ее сбрасываем.

Вне этого цикла мы устанавливаем lastByte на текущее значение, чтобы в следующий раз у нас была его копия:

```
lastByte=byteIn;
```

Номер индекса увеличивается:

```
index++;
```

При достижении конца строки мы должны убедиться, что десятая цифра в массиве cardNum установлена на код ASCII для \ 0 или нулевой терминатор. Это необходимо в дальнейшем, когда нам нужно будет определить, достигнут ли конец строки или нет. Нулевой терминатор показывает, что мы находимся в конце строки.

```
if (index==12) {cardNum[10] = '\0';}
```

Затем вы распечатываете номер карты, который считали с RFID-считывателя:

```
Serial.println(cardNum); // выводим номер карты
```

Затем создается целое число с именем cardIndex, которому присваивается значение, возвращаемое функцией checkCard () (поясняется вкратце). Функция checkCard () вернет положительное значение, если номер карты действителен из базы данных, и отрицательное число, если это не так.

```
int cardIndex =checkCard(cardNum); // проверяем действительность карты и возвращаем порядковый номер
```

Затем мы проверяем, что возвращенное число положительное, а также правильность контрольной суммы. Если это так, карта была прочитана правильно и действительна, поэтому вы можете открыть дверь.

```
if(cardIndex>=0 && (cardBytes[5]==checksum)) { // если номер карты и контрольная сумма действительны
        Serial.println("Card Validated");
        Serial.print("User: ");
        Serial.println(names[cardIndex]);      // выводим соответствующее имя
        unlock();                              // отпираем дверь
        Serial.println();
    }
```

Если карта недействительна или контрольная сумма неверна, карта считается недействительной, и пользователю сообщается:

```
else {
        Serial.println("Card INVALID");
        tone(speakerPin, 250, 250);
        delay(250);
        tone(speakerPin, 150, 250);
        Serial.println();
    }
```

Далее идет функция checkCard (). Он будет возвращать целое число, так что это его тип, а его параметр - номер карты, который мы ему передадим.

```
int checkCard(char cardNum[10]) {
```

Затем мы просматриваем каждую карту в базе данных, чтобы увидеть, соответствует ли она номеру карты, которую мы прочитали:

```
for (int x=0; x<=users; x++) { // check all valid cards
```

Мы используем функцию strcmp или String Compare, чтобы проверить, совпадают ли номер карты, переданный в функцию checkCard (), и номер карты в текущем расположении базы данных. Вот почему нам нужен нуль символ в конце номера карты, поскольку этого требует функция strcmp.

```
if(strcmp(cardNum, cards[x])==0) { // сравнить с номером последней прочитанной карты
```

Функция strcmp требует два параметра. Это две строки, которые мы хотим сравнить. Число, возвращаемое функцией, равно нулю, если обе строки идентичны. Ненулевое число означает, что они не совпадают. Если они совпадают, мы возвращаем значение x, которое будет индексом в карточке и базе данных имен действующей карточки.

```
return (x); // возвращаем индекс номера карты
```

Если карты не совпадают, возвращается -1.

```
return (-1); // отрицательное значение означает отсутствие совпадения
}
```

Последняя функция - unlock (), которая воспроизводит высокий тон, открывает дверь, ждет в течение заданного времени, а затем снова запирает дверь:

```
void unlock() {
    tone(speakerPin, 1000, 500);
    digitalWrite(lockPin, HIGH);
    delay(unlockLength);
    digitalWrite(lockPin, LOW);
}
```

Следующим шагом в рамках этого проекта будет добавление дополнительных считывающих устройств и замков, чтобы обезопасить весь ваш дом.

Авторизованные пользователи будут иметь при себе карточку или бирку, чтобы позволить им войти в соответствующие комнаты. Каждому пользователю могут быть предоставлены индивидуальные права доступа, чтобы у них был разный доступ к разным частям здания.

Теперь перейдем к последней главе этой книги, где мы подключим Arduino к Интернету!

Резюме

В этой главе вы увидели, как легко считывать данные с RFID-карты или метки, а затем использовать эти данные для разблокировки электрического замка или для выполнения другого действия. Я видел проекты, в которых цепочка ключей RFID прикреплялась к связке ключей. Считыватель RFID находится в корзинке, и когда пользователь возвращается домой, он бросает в него свои ключи. Дом реагирует на возвращение этого человека, например, устанавливает выбранную температуру и уровень освещенности, играет его любимую музыку, включает душ и т. д. Когда дело доходит до использования считывателя RFID, вы ограничены только вашим собственным воображением.

Предметы и понятия, затронутые в главе 16

- Как работает технология RFID
- Как подключить считыватель RFID ID12 к Arduino
- Чтение последовательных данных с RFID-считывателя
- Использование транзистора для управления высокомощным устройством
- Использование библиотеки SoftwareSerial
- Преобразование ASCII в шестнадцатеричные значения
- Использование побитового И для определения четности или нечетности числа
- Объединение двух шестнадцатеричных цифр с использованием битовых сдвигов и побитового ИЛИ длясоздания байта
- Создание контрольных сумм с использованием XOR (Исключающее ИЛИ)
- Использование strcmp для сравнения двух строк

ГЛАВА 17

Связь через интернет

В этой последней главе вы узнаете, как подключить Arduino к роутеру, чтобы данные можно было отправлять по кабелю Ethernet. Сделав это, вы можете прочитать их из любого места в вашей сети. Вы также можете отправлять данные в Интернет, чтобы их можно было просматривать в веб-браузере. Для этого мы будем использовать официальный Arduino Ethernet Shield или Arduino Ethernet. Возможность подключить ваш Arduino к сети или Интернету открывает целый новый список потенциальных проектов.

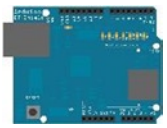
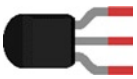
Мы можем отправлять данные на веб-сайты, например размещать обновления Twitter. Мы также можем управлять Arduino через Интернет или использовать Arduino в качестве веб-сервера для обслуживания простых веб-страниц, содержащих данные датчиков, и т. д. Эта глава даст вам базовые знания для создания ваших собственных проектов на основе Ethernet или Internet Arduino.

Проект 46 - Ethernet-шилд

Мы используем Ethernet Shield или Arduino Ethernet и пару датчиков температуры, чтобы продемонстрировать доступ к Arduino через Ethernet.

Требуемые детали

Table 17-1. *Детали, необходимые для проекта 46*

Шилд Arduino Ethernet	
2 × Датчик температуры DS18B20	
Резистор 4,7 кОм	

Подключение

Вставьте Ethernet Shield в верхнюю часть Arduino, затем подключите все, как показано на рисунке 17-1, с проводами, идущими в Ethernet Shield в том же месте, что и на Arduino.

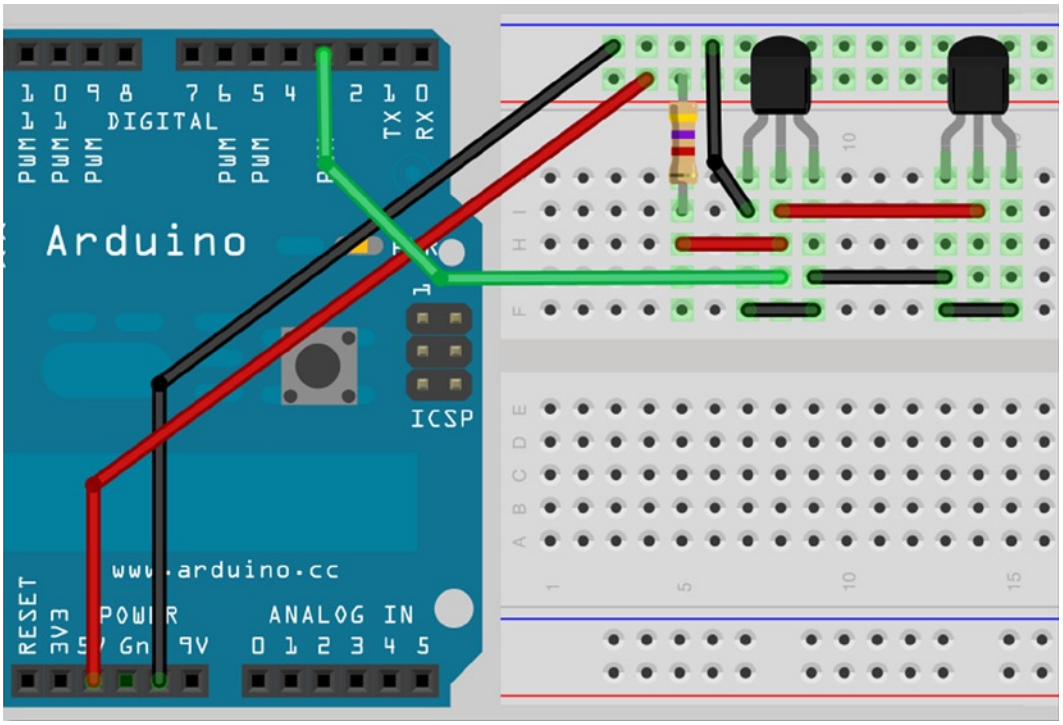


Рис. 17-1. *Схема для проекта 46 - Ethernet Shield*

Введите код

Введите код из Листинга 17-1.

Листинг 17-1. Код для проекта 46

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Провод данных подключен к контакту 3 на Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;

// Настраиваем экземпляр oneWire для связи с любыми устройствами OneWire
temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);
//Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);
```

```
массивы для хранения адресов устройств
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};

byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192,168,0, 104 };

// Запускаем сервер на 80-м порту
EthernetServer server(80);

void setup()
{
  // Запускаем Ethernet и сервер
  Ethernet.begin(mac, ip);server.begin();

    // Запускаем библиотеку датчиков
  sensors.begin();
    // устанавливаем разрешение
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
}

// функция для получения температуры устройства
void getTemperature(DeviceAddress deviceAddress)
{
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
void loop()
{
  sensors.requestTemperatures();

  // слушаем входящих клиентов
  EthernetClient client = server.available();
  if (client) {
    // http-запрос заканчивается пустой строкой
    boolean BlankLine = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();

        // Если строка пуста, а конец строки - символ новой строки '\ n' = конец HTTP-запроса
        if (c == '\n' && BlankLine) {
          getTemperature(insideThermometer);
          getTemperature(outsideThermometer);
          // Display internal temp
          client.println("HTTP/1.1 200 OK\n"); // Стандартный ответ HTTP
          client.println("Content-Type: text/html\n");
          client.println("\n");
          client.println("<html>");
          client.println("<head>");
          client.println("<META HTTP-EQUIV=\"refresh\" CONTENT=\"5\">");
```

```
          client.println("<title>Arduino Web Server</title>");
          client.println("</head>");
          client.println("<body>");
          client.println("<h1>Arduino Web Server</h1>");
          client.println("<h3>Internal Temperature</h3>");
          client.println("Temp C:");
          client.println(tempC);
          client.println("<br/>");
          client.println("Temp F:");
          client.println(tempF);
          client.println("<br/>");
          // Display external temp
          client.println("<h3>External Temperature</h3>");
          client.println("Temp C:");
          client.println(tempC);
          client.println("<br/>");
          client.println("Temp F:");
          client.println(tempF);
          client.println("<br/>");
          client.println("</body>");
          client.println("</html>");

          break;
      }
        if (c == '\n') {
          // Запуск новой строки
          BlankLine = true;
        }
        elseif (c != '\r') {
          //В текущей строке есть символ
          BlankLine = false;
        }
      }
    }
    // Даем время браузеру получить данные
    delay(10);
    // Закрываем соединение
    client.stop();
  }
}
```

Вам нужно будет ввести два адресных номера датчиков температуры (см. Проект 37) в этой строке:

```
byte ip[] = { 192,168,0, 104 };
```

Вам также потребуется изменить IP-адрес на свой собственный. Для этого вам нужно будет узнать у вашего роутера, какой диапазон IP-адресов выделен для устройств на вашем компьютере. Обычно адрес начинается как 192.168.0 или 192.168.1 - затем вы просто добавляете еще одно число выше 100, чтобы убедиться, что оно не мешает.с существующими устройствами. Вам также может потребоваться войти в настройки вашего маршрутизатора, чтобы убедиться, что любые HTTP-запросы к порту 80 перенаправляются на IP-адрес Ethernet Shield. Посмотрите раздел «Переадресация портов» в руководстве к маршрутизатору. Также может потребоваться открыть порт 80 в вашем брандмауэре.

Теперь откройте свой веб-браузер и введите IP-адрес и порт, например

```
192.168.0.104:80
```

Если все работает правильно, в вашем браузере откроется веб-страница, показанная на рис. 17-2.

Рис. 17-2. *Вывод веб-браузера с веб-сервера Arduino*

Страница будет автоматически обновляться каждые пять секунд, показывая любые изменения температуры. Если мы правильно настроили переадресацию портов и брандмауэр в своем роутере, мы также сможем получить доступ к странице из любого места, где есть доступ в Интернет. Нам необходимо знать IP-адрес маршрутизатора, который можно найти на странице администрирования маршрутизатора. Введите его, а затем номер порта в любой веб-браузер, например

```
95.121.118.204:80
```

Вышеупомянутая веб-страница теперь появится в браузере, и вы сможете проверить показания температуры из любого места, где есть доступ в Интернет.

Что нужно знать о сети

DHCP (протокол динамической конфигурации хоста) - это протокол автоконфигурации, используемый в IP-сетях. Это позволяет Ethernet Shield автоматически назначать IP-адрес из того, который доступен на маршрутизаторе. Ранее вы устанавливали IP-адрес вручную; на этот раз вы используете библиотеку Ethernet, чтобы автоматически назначить один за вас. Компромисс в том, что ваш код намного длиннее.
Адрес MAC (Media Access Control) - это уникальный идентификатор для сетевых интерфейсов. Сетевая карта на вашем ПК или Mac будет иметь MAC-адрес, установленный производителем. В вашем случае вы решаете, какой MAC-адрес. Это просто 48-битное число, поэтому просто введите в адрес любые шесть шестнадцатеричных цифр, хотя можно оставить его таким, какой он есть в коде. IP-адрес необходимо будет установить вручную, и он должен быть одним из диапазона, разрешенного вашим маршрутизатором.

Проект 46 - Ethernet Shield - Обзор кода

Некоторые части этого кода повторяются из Project 37, поэтому я замалчиваю эти разделы и вместо этого сосредоточусь на частях, относящихся к Ethernet Shield. Сначала мы загрузим библиотеки и убедимся, что у нас есть библиотеки для датчиков температуры в папке с библиотеками (см. Проект 37). Обратите внимание, что начиная с Arduino IDE версии 0019 было необходимо включить библиотеку SPI.h в любой проект, для которого требуется библиотека Ethernet.h.

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Далее устанавливается пин и точность датчиков.

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

вместе с двумя числами с плавающей запятой, которые мы будем использовать для хранения температуры в градусах Цельсия иФаренгейта.

```
float tempC, tempF;
```

Создается экземпляр объекта oneWire, и мы передаем ссылку на библиотеку Dallas Temperature:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Устанавливаются адреса для двух датчиков температуры. Не забудьте выяснить, что они используют в коде Project 37, если это необходимо.

```
DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
```

Далее нам необходимо определить MAC и IP-адрес устройства:

```
byte mac[] = { 0x48, 0xC2, 0xA1, 0xF3, 0x8D, 0xB7 };
byte ip[] = { 192,168,0, 104 };
```

Затем создается экземпляр объекта сервера Ethernet вместе с номером порта для устройства:

```
EthernetServer server(80);
```

Сервер будет прослушивать входящие соединения на указанном порту. Номер порта - это просто путь для данных.У нас есть только один кабель Ethernet, подключенный к нашему устройству, но номер порта решает, куда будут идти эти данные. Представьте, что MAC-адрес - это адрес большого многоквартирного дома, а номер порта - это индивидуальный номер квартиры.
Далее идет процедура настройки. Мы начинаем с инициализации связи Ethernet и передачи MAC и IP-адреса устройства экземпляру:

```
Ethernet.begin(mac, ip);
```

Теперь вам нужно указать серверу начать прослушивание входящих соединений с помощью команды begin ():

```
server.begin();
```

Также нам нужно запустить свои датчики и установить их разрешение:

```
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Затем мы создаем функцию для получения температуры от датчика (как это было сделано в Project 37):

```
void getTemperature(DeviceAddress deviceAddress)
{
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

Далее идет основной цикл программы. Сначала мы запрашиваем температуру у двух датчиков:

```
sensors.requestTemperatures();
```

Нам необходимо прислушиваться к любым входящим клиентам, то есть веб-страницам, запрашивающим просмотр веб-страницы, обслуживаемой Arduino.Для этого мы создаем экземпляр типа EthernetClient и используете его для проверки наличия данных, доступных для чтения с сервера. Клиент - это веб-браузер, который будет подключаться к Arduino.Сервер - Ардуино.

```
EthernetClient client = server.available();
```

Затем мы проверяем, подключился ли клиент и доступны ли для него какие-либо данные. Если true, выполняется код, принадлежащий оператору if ().

```
if (client) {
```

Сначала оператор if создает логическую переменную с именем BlankLine и устанавливает для нее значение true:

```
boolean BlankLine = true;
```

HTTP-запрос от клиента заканчивается пустой строкой, завершающейся символом новой строки. Таким образом, мы используем переменную BlankLine, чтобы определить, достигли ли мы конца данных или нет.Затем мы проверяем, подключен ли клиент по-прежнему или нет, и если да, запускаем код в цикле while:

```
while (client.connected()) {
```

Далее мы проверяем, доступны ли данные для клиента или нет. Если данные доступны, выполняется код следующего оператора if. Команда available () возвращает количество байтов, записанных клиенту сервером, к которому он подключен. Если это значение больше нуля, выполняется оператор if.

```
if (client.available()) {
```

Затем создается переменная типа char для хранения следующего байта, полученного от сервера. Используйте команду client.read (), чтобы получить байт.

```
char c = client.read();
```

Если прочитанный символ является символом новой строки (‘\ n’), нам также необходимо проверить, истинно ли значение BlankLine или нет. Если это так, значит, мы достигли конца HTTP-запроса и можем передать HTML-код клиенту (веб-браузеру пользователя).

```
if (c == '\n' && BlankLine) {
```

Next comes the data you will send out from your server. You start by obtaining the temperature from the internal and external sensors.

```
getTemperature(insideThermometer);
getTemperature(outsideThermometer);
```

Затем идет HTML-код, который мы должны передать клиенту. Каждая страница состоит из кода, который называется HTML (или язык разметки гипертекста). Объяснение HTML выходит за рамки этой книги, поэтому я дам лишь базовую информацию.

Если вы хотите узнать больше о HTML, ознакомьтесь с записью HTML в Википедии по адресу http://en.wikipedia.org/wiki/HTML.

В Интернете также есть множество учебных пособий по HTML. Вы используете команду client.println () для передачи данных клиенту. По сути, мы отправляем код для создания веб-страницы. Если вы щелкните правой кнопкой мыши на веб-странице в большинстве браузеров, вам будет предоставлена возможность просмотреть исходный код. Попробуйте это, и вы увидите HTML-код, который составляет веб-страницу, которую вы только что просматривали. Код сообщает браузеру, что отображать и как это отображать.

Во-первых, мы сообщаем клиенту, что используем HTTP версии 1.1, который является стандартным протоколом, используемым для выдачи веб-страниц, и что контент, который мы собираемся отправить, является HTML:

```
client.println("HTTP/1.1 200 OK\n"); // Standard HTTP response
client.println("Content-Type: text/html\n\n");
client.println("\n");
```

Затем у нас есть тег HTML, чтобы сказать, что отныне все будет HTML-кодом и тегом заголовка HTML-кода. В заголовке перед основной частью кода содержатся любые команды, которые мы хотим передать браузеру, скрипты, которые вы хотите запустить, и т. д. Первая команда сообщает браузеру, что мы хотим, чтобы страница автоматически обновлялась каждые пять секунд.

```
client.println("<html>");
client.println("<head>");
client.println("<META HTTP-EQUIV=\"refresh\" CONTENT=\"5\">");
```

Затем вы даете странице название. Он появится в верхней части браузера и на любых вкладках для этой страницы.

```
client.println("<title>Arduino Web Server</title>");
client.println("</head>");
```

Мы завершаем раздел заголовка, вставляя тег </head>. Далее идет тело HTML. Это часть, которая будет видна пользователю.

```
client.println("<body>");
```

Мы отобразим заголовок <h1> с надписью «Веб-сервер Arduino». H1 - самый крупный заголовок, за ним идут H2, H3 и т. д.

```
client.println("<h1>Arduino Web Server</h1>");
```

Здесь код имеет заголовок следующего раздела, который называется «Внутренняя температура» в виде заголовка h3.

```
client.println("<h3>Internal Temperature</h3>");
```

Затем мы печатаем температуру в градусах Цельсия и F с разрывами строк
:

```
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");

client.println("<h3>External Temperature</h3>");
client.println("Temp C:");
client.println(tempC);
client.println("<br/>");
client.println("Temp F:");
client.println(tempF);
client.println("<br/>");
client.println("</body>");
client.println("</html>");
```

Затем цикл while завершается командой break:

```
break;
```

Теперь мы устанавливаем для BlankLine значение true, если читается символ \ n (новая строка), и false, если это не \ r (возврат каретки), т.е. есть еще символы, которые нужно прочитать с сервера.

```
if (c == '\n')
{
   // Starting a new line
   BlankLine = true;
}
elseif (c != '\r')
{
   // Current line has a character in it
   BlankLine = false;
}
```

Мы ждем небольшую задержку, чтобы дать браузеру время для получения данных, а затем останавливаем клиента с помощью команды stop (). Это отключает клиента от сервера.

```
delay(10);
client.stop();
```

Этот проект представляет собой базовое введение в обслуживание веб-страницы со встроенными в нее данными датчиков через Ethernet Shield.

Существуют более удобные способы представления данных через Интернет, и мы рассмотрим один из этих методов далее.

Проект 47 - Прогноз погоды в Интернете

Теперь вы собираетесь использовать те же детали и схему, но на этот раз вы отправите данные о температуре для двух датчиков в Xively (формально Cosm, а затем формально Pachube). Xively - это онлайн-служба базы данных, которая позволяет пользователям подключать данные датчиков к Интернету (см. Рис. 17-3). Данные могут быть отправлены в различных форматах и отображаются в ленте на веб-сайте в виде графика и данных в реальном времени. Графики отображаются в реальном времени и могут быть встроены в веб-сайт. Ты можешь
также просматривать исторические данные, полученные из канала, а также отправлять оповещения для управления скриптами, устройствами и т. д. На веб-сайте, посвященном Arduino, есть серия руководств.

Рис. 17-3. *Сайт Xively*

Чтобы использовать сервис, вы должны создать аккаунт разработчика Xively. Это идеальный способ отображения данных с датчиков, домашних мониторов энергоснабжения, систем мониторинга зданий и т. д. Услуга полностью бесплатна для аккаунта разработчика и до пяти подключенных устройств. При коммерческом использовании взимается плата за обслуживание.

Перед загрузкой данных необходимо создать аккаунт разработчика, поэтому для начала перейдите на веб-сайт www.xively.com и нажмите кнопку «ПОДПИСАТЬСЯ» в правом верхнем углу. Появится поле для ввода вашего имени пользователя, электронной почты и пароля. Затем появляется окно с надписью «Расскажите немного о себе».

После успешной регистрации вы войдете в систему и перейдете на страницу «Development Devices - Устройства для разработки». Теперь нам нужно создать устройство и ленту. Нажмите кнопку + Add Device - Добавить устройство. Затем вам будет представлено окно с запросом имени устройства и описания устройства; затем вы выбираете в разделе конфиденциальности, хотите ли вы, чтобы устройство и его данные должны быть частными или общедоступными (см. рис. 17-4). Назовите свое устройство «Arduino» и введите описание, если хотите.

Затем выберите, хотите ли вы, чтобы ваши данные были частными или общедоступными.

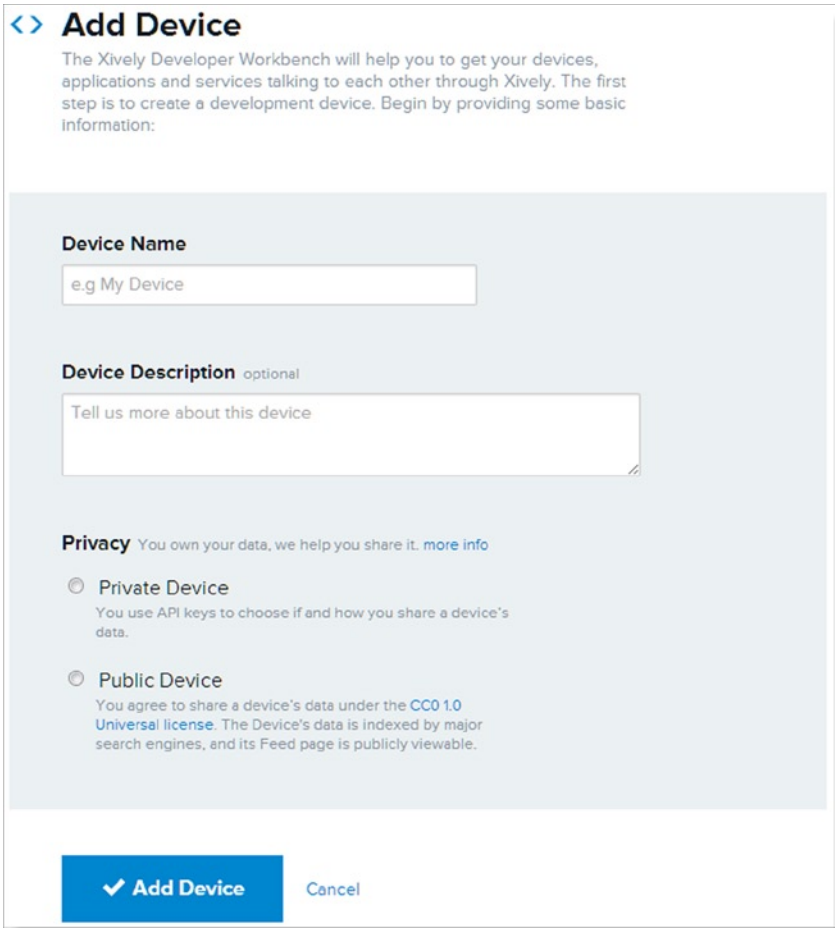


Рис. 17-4. *Выберите тип вашего устройства*

После того, как мы создали устройство, нам нужно создать канал. Канал - это просто поток данных с устройства. В случае этого проекта мы создадим четыре канала для внешней температуры в градусах Цельсия, также в градусах Фаренгейта, и еще два для внутренней температуры в градусах Цельсия и Фаренгейта. Дайте каналу имя, а затем добавьте единицы измерения (Цельсия или Фаренгейта), за которыми следует символ C или F. Оставьте поле текущего значения пустым.
(См. Рисунок 17-5).

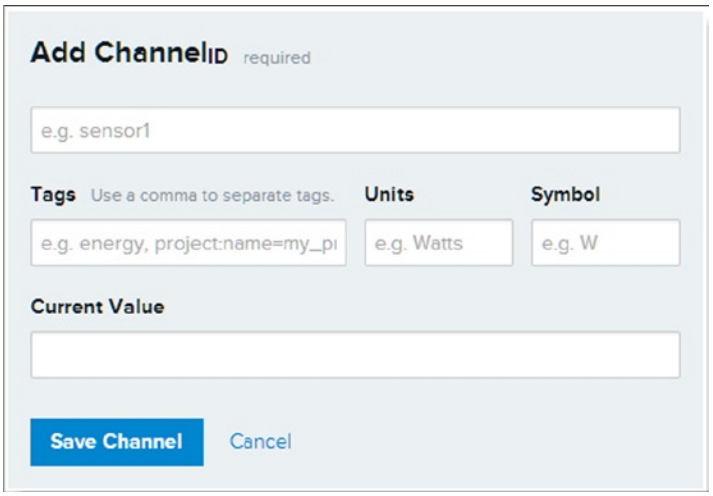


Рис. 17-5. *The screen seen after successfully creating your channel*

После того, как мы создали свой канал, он готов получать данные от нашего Arduino. Сначала запишем свой идентификатор канала, а также ключ API (интерфейс прикладного программирования). Мы можем получить идентификатор канала, щелкнув имя канала в верхней части Центра разработчиков. Ключ API находится в правой части страницы. На странице «Ключи API» будут перечислены ключи для всех созданных нами устройств (см. Рис. 17-6). Скопируйте и вставьте ключ в Блокнот или другой текстовый редактор и сохраните его для дальнейшего использования.

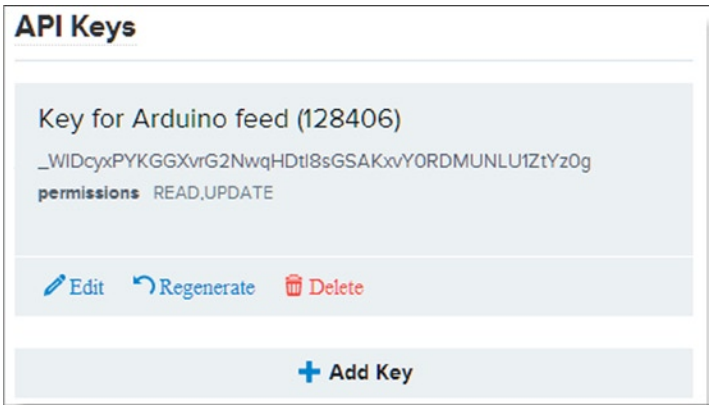


Рис. 17-6. *Ключ API*

Теперь, когда мы создали свой аккаунт Xively, сгенерировали устройство, скопировали свой идентификатор канала и ключ API, мы готовы ввести код.

Введите код

Введите код из Листинга 17-2. Нам понадобится библиотека Dallas Temperature, а также библиотеки HttpClient и Xively, которые можно загрузить с веб-сайта Xively по адресу https://github.com/xively/xively_arduino.
В инструкциях показано, как установить библиотеку HttpClient и библиотеку Xively.

Листинг 17-2. Код для проекта 47

```
// Project 47 - на основе примеров Xively Arduino

#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <HttpClient.h>
#include <Xively.h>

#define SHARE_FEED_ID 128497//  это ваш идентификатор xively фида, которым вы хотите поделиться
#define xivelyKey  "_WlvrG2NwORxPYKGGDMUNLqHDtl8sDcyXGSAKxvYU1ZtYzOg" // заполните свой ключ API

char sensorId1[] = "IntTempC";
char sensorId2[] = "IntTempF";
char sensorId3[] = "ExtTempC";
char sensorId4[] = "ExtTempF";
XivelyDatastream datastreams[] = {
  XivelyDatastream(sensorId1, strlen(sensorId1), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId2, strlen(sensorId2), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId3, strlen(sensorId3), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId4, strlen(sensorId4), DATASTREAM_FLOAT)
};
// Наконец, оборачиваем потоки данных в фид
XivelyFeed feed(SHARE_FEED_ID, datastreams, 4 /* количество потоков данных */);

// Провод данных подключен к контакту 3 на Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

// Настраиваем экземпляр oneWire для связи с любыми устройствами OneWire (не только с
Maxim / Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств
DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };
DeviceAddress outsideThermometer = { 0x28, 0xA5, 0x02, 0xC2, 0x03, 0x00, 0x00, 0xF0 };
```

```
unsigned int interval;
float itempC, itempF, etempC, etempF;

byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 }; // make sure this is unique on your network

EthernetClient localClient;
XivelyClient xivelyclient(localClient);

void setup()
{
  Serial.begin(9600);
  Serial.println("Starting 4 stream data upload to Xively...");
  Serial.println();
  while (Ethernet.begin(mac) != 1)
  {
    Serial.println("Error getting IP address via DHCP, trying again...");
    delay(15000);
  }
      // Запускаем библиотеку датчиков
  sensors.begin();
    // set the resolution
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  delay(100);
}

void xively_in_out() {
  getTemperatures();
  datastreams[0].setFloat(itempC);
  datastreams[1].setFloat(itempF);
  datastreams[2].setFloat(etempC);
  datastreams[3].setFloat(etempF);
  xivelyclient.put(feed, xivelyKey);
  Serial.println("Read sensor value ");
  Serial.println(datastreams[0].getFloat());
  Serial.println(datastreams[1].getFloat());
  Serial.println(datastreams[2].getFloat());
  Serial.println(datastreams[3].getFloat());
  delay(10000);
}

// функция для получения температуры устройства
void getTemperatures()
{
  sensors.requestTemperatures();
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}
```

```
void loop()
{
  xively_in_out();
}
```

Загрузите код и откройте монитор последовательного порта (МПП). Если все работает правильно и мы успешно отправляем данные в Xively, результат будет выглядеть примерно так:

```
Starting 4 stream data upload to Xively...   Запуск 4 потоковой загрузки данных в Xively ...

Read sensor value    Считать значение датчика
27.00
80.60
27.06
80.71
Read sensor value    Считать значение датчика
26.94
80.49
27.00
80.60
```

Теперь откройте свой веб-браузер и перейдите на www.Xively.com. Перейдите к своей ленте в Центре разработчиков и щелкните имя устройства. Теперь вам будет показана страница устройства. Показания температуры с датчиков будут отображаться с течением времени. Щелкнув символ часов под каждым потоком данных, вы можете изменить период времени с 6 часов по умолчанию на любой, какой захотите. Дата и время последнего обновления канала показаны справа от каналов в журнале запросов (см. Рис. 17-7). Графики также должны показывать изменения температуры с течением времени в нижней части графика. Если вы оставите это на длительное время, вы должны увидеть изменения температуры в течение дня.

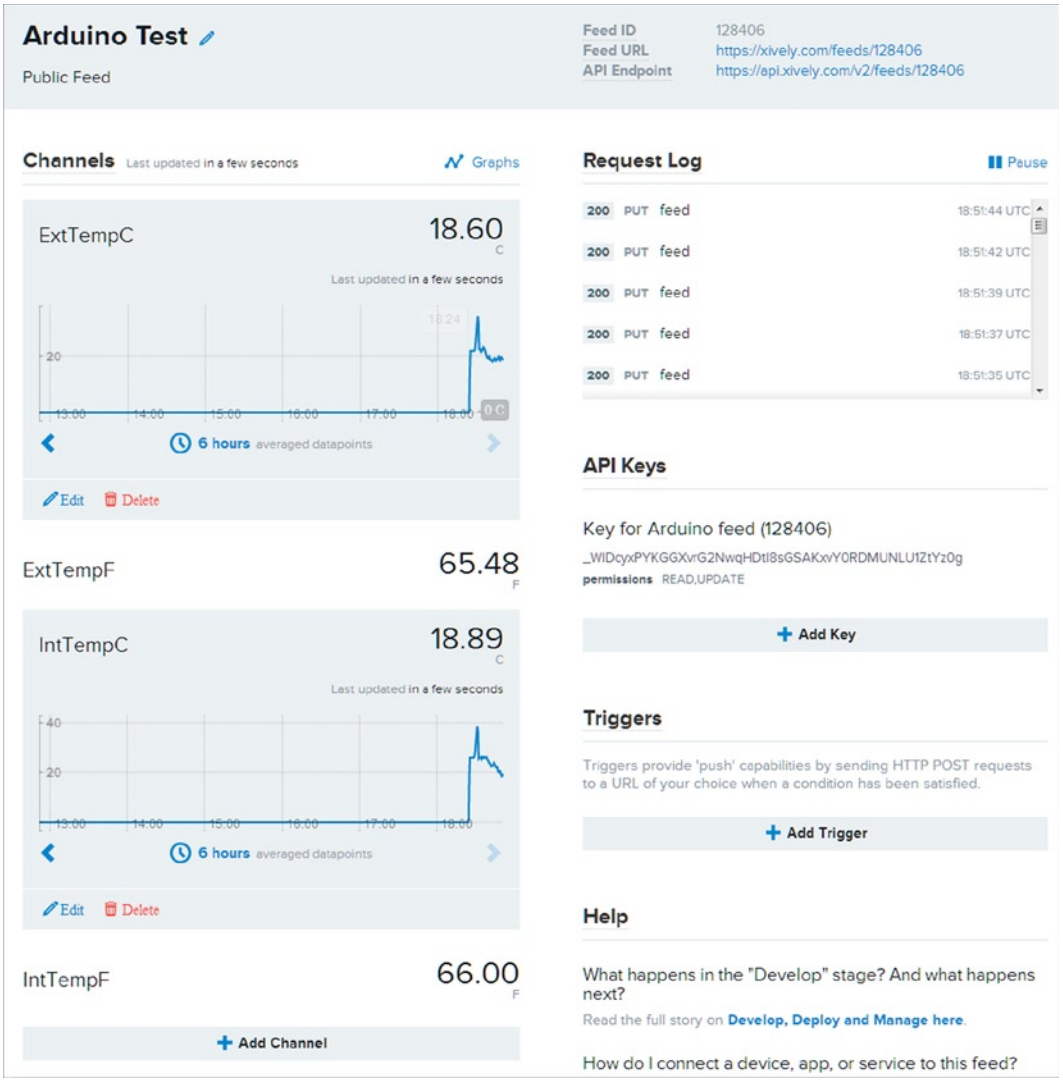


Рис. 17-7. *Живая страница канала Xively*

Щелкнув значок часов на каждом графике, мы сможем выбрать просмотр необработанных данных за 5 минут, 30 минут или 1 час. Мы также можем выбрать усредненные точки данных за 6 часов, 1 день, 7 дней, 1 месяц и 3 месяца.

Мы можем изменить код, чтобы добавить другие датчики температуры, такие как датчик давления, который мы использовали в Проекте 31, датчики света для измерения окружающего света, датчики влажности и датчики скорости ветра, чтобы создать полноценную метеостанцию с нашими данными, зарегистрированными и доступными черезИнтернет на Xively.

Теперь давайте посмотрим на код этого проекта, чтобы увидеть, как он работает.

Проект 47 - Прогноз погоды в Интернете - Обзор кода

Код начинается с включения Ethernet Shield и однопроводных датчиков температуры:

```
#include <SPI.h>
#include <Ethernet.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Также требуются новые библиотеки для использования Xively. Это клиентская библиотека Http и библиотека Xively.
Библиотека Xively находится по адресу https://github.com/xively/xively_arduino. Нажмите кнопку "Загрузить ZIP" справа.
Библиотеку клиента HTTP можно найти по адресу https://github.com/amcewen/HttpClient. Снова нажмите кнопку«Загрузить ZIP». Разархивируйте библиотеки и поместите их по-прежнему в соответствующие папки в папку с библиотеками Arduino. На страницах также есть инструкции по использованию библиотек.

```
#include <HttpClient.h>
#include <Xively.h>
```

Затем мы используем четыре оператора define для установки некоторых констант манифеста; первый предназначен для хранения идентификационного номера фида, а второй - ключа API для фида. Убедитесь, что вы ввели собственный идентификатор фида и ключ API. Их можно найти в настройках Xively и на странице Feed.

```
#define SHARE_FEED_ID   128406
#define xivelyKey  "_WlDcyxPYKGGXvrG2NwqHDtl8sGSAKxvYORDMUNLU1ZtYzOg"
```

Затем нам нужно создать поток данных для загрузки в Xively. У нас будет четыре потока данных для внутренней и внешней температуры в градусах Цельсия и Фаренгейта. Мы создаем некоторые переменные типа char для хранения массива символов, которые будут составлять имена потоков каналов. Это метки для каждого датчика, которые отображаются на графике данных на странице канала Xively.

```
char sensorId1[] = "IntTempC";
char sensorId2[] = "IntTempF";
char sensorId3[] = "ExtTempC";
char sensorId4[] = "ExtTempF";
```

Затем мы создаем массив типа XivelyDatastream для хранения потоков. Этот тип данных актуален только для библиотеки Xively. У каждого потока должно быть три параметра: имя потока, длина имени и тип загружаемых данных. В нашем случае мы загружаем данные о температуре в виде числа с плавающей запятой. Мы также могли бы отправить значение как int, используя DATASTREAM_INT, или как строку, используя DATASTREAM_STRING, и как буфер символов, используяDATASTREAM_BUFFER. Обратитесь к файлу readme для библиотеки Xively для получения дополнительной информации.

```
XivelyDatastream datastreams[] = {
  XivelyDatastream(sensorId1, strlen(sensorId1), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId2, strlen(sensorId2), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId3, strlen(sensorId3), DATASTREAM_FLOAT),
  XivelyDatastream(sensorId4, strlen(sensorId4), DATASTREAM_FLOAT)
};
```

Затем нам нужно иметь возможность отправлять эти данные в удаленную систему, поэтому мы помещаем эти потоки в фид с помощью XivelyFeed. Мы даем каналу имя, и его три параметра - это номер идентификатора канала, имя массива потока данных и количество потоков данных в массиве. Функция XivelyDatastream собирает данные для загрузки в Xively, а функция XivelyFeed создает канал, в который мы будем отправлять поток.

```
XivelyFeed feed(SHARE_FEED_ID, datastreams, 4 /* number of datastreams */);
```

Затем, как и в главе 15, мы определяем контакт, который мы будем использовать для однопроводной шины для подключения к датчикам температуры, и используемую точность.

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Мы создаем экземпляр One Wire, чтобы мы могли общаться через однопроводную шину Arduino.

```
OneWire oneWire(ONE_WIRE_BUS);
```

Затем мы отправляем ссылку на этот экземпляр One Wire в библиотеку Dallas Temperature.

```
DallasTemperature sensors(&oneWire);
```

Затем мы создаем два адреса устройства для внутреннего и внешнего датчиков температуры.

```
DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };
DeviceAddress outsideThermometer = { 0x28, 0xA5, 0x02, 0xC2, 0x03, 0x00, 0x00, 0xF0 };
```

Четыре переменных типа float созданы для хранения внутренней и внешней температуры в градусах Цельсия и F.

```
float itempC, itempF, etempC, etempF;
```

Чтобы использовать библиотеку Ethernet, нам понадобится MAC-адрес (Media Access Control) устройства Ethernet, которое мы собираемся использовать, в данном случае наш Ethernet Shield. MAC-адрес - это уникальный идентификатор, назначаемый сетевым устройствам.На современных экранах Ethernet Shield адрес указан на наклейке на нижней стороне устройства. Если у устройства нет заранее определенного MAC-адреса, просто придумайте его. Адрес состоит из 12 шестнадцатеричных символов.

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 };
```

Затем мы используем функцию EthernetClient для создания клиента, который может подключаться к определенному IP-адресу и порту в Интернете (обычно определяется с помощью функции client.connect(), но библиотека Xively позаботится об этом за нас) и предоставить клиенту название.

```
EthernetClient localClient;
```

Затем мы создаем экземпляр клиента Xively и передаем ему имя экземпляра EthernetClient.

```
XivelyClient xivelyclient(localClient);
```

Теперь мы подошли к функции setup ().

```
void setup()
{
```

Мы будем использовать МПП для обратной связи, поэтому начнем последовательную связь и установим скорость передачи.

```
Serial.begin(9600);
```

Затем сообщим пользователю, что мы собираемся делать.

```
Serial.println("Starting 4 stream data upload to Xively...");
Serial.println();
```

Функция Ethernet.begin () инициализирует библиотеку Ethernet и сетевые настройки. В случае успеха он вернет истинный или ложный флаг. Мы используем функцию While, чтобы проверить, успешна ли связь через Ethernet, и соответствующим образом проинформировать пользователя.

```
while (Ethernet.begin(mac) != 1)
{
  Serial.println("Error getting IP address via DHCP, trying again...");
  delay(15000);
}
```

Как и в главе 15, мы используем библиотеку температуры Далласа, чтобы установить связь с датчиками и установить их разрешение.

```
  sensors.begin();
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  delay(100);
}
```

После функции setup () у нас есть собственная функция под названием xively_in_out, которая будет отправлять наши потоки данных в Xively.

```
void xively_in_out() {
```

Сначала вызывается функция getTemperatures () для чтения и сохранения показаний температуры.

```
getTemperatures();
```

Затем в первом элементе переменной массива потока данных мы сохраняем первую температуру и убеждаемся, что для нее установлено число с плавающей запятой, используя функцию setFloat () библиотеки Xively. Мы также можем установить его в буфер типа int, string или char, используя setInt (), setString () или setBuffer () соответственно.

```
datastreams[0].setFloat(itempC);
datastreams[1].setFloat(itempF);
datastreams[2].setFloat(etempC);
datastreams[3].setFloat(etempF);
```

Теперь поток готов к отправке в Xively. Мы используем команду put для отправки потока. Команда put добавляется в конец имени клиента Xively с расширением. обозначение, за которым следуют два обязательных параметра или идентификатор фида и ключ API.

```
xivelyclient.put(feed, xivelkyKey);
```

Наконец, пользователь информируется о том, какие данные были загружены, с помощью функции getFloat () с задержкой в 10 секунд.

```
  Serial.println("Read sensor value ");
  Serial.println(datastreams[0].getFloat());
  Serial.println(datastreams[1].getFloat());
  Serial.println(datastreams[2].getFloat());
  Serial.println(datastreams[3].getFloat());
  delay(10000);
}
```

Следующая функция такая же, как в главе 15, которая считывает данные с датчиков и присваивает значения соответствующим переменным.

```
void getTemperatures()
{
  sensors.requestTemperatures();
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}
```

Наконец, наша основная программная функция просто многократно вызывает функцию xively_in_out ().

```
void loop()
{
  xively_in_out();
}
```

Теперь, когда мы знаем основные методы отправки данных датчиков в Xively для хранения ипросмотра, нам будет относительно легко изменить код для добавления дополнительных датчиковили других данных.
До сих пор вы узнали, как отправлять данные через Ethernet в веб-браузер в сети, в веб-браузер через Интернет и в службу хранения данных и построения графиков Xively. Затем вы узнаете, как заставить Arduino отправлять электронное письмо, чтобы предупреждать вас, когда температура становится слишком высокой или слишком низкой.

Проект 48 - Система оповещения по электронной почте

Теперь мы рассмотрим другой метод отправки данных. В этом проекте мы получим Arduino с Ethernet Shield для отправки электронной почты, когда температура либо слишком низкая, либо слишком высокая. Этот проект разработан, чтобы показать нам основы отправки электронной почты через Ethernet Shield. Мы будете использовать ту же схему, но только с одним из датчиков температуры.

Введите код

Введите код из Листинга 17-3. Нам нужно будет получить IP-адрес нашего почтового SMTP-сервера. Для этого откроем окно терминала (окно команд, консоль, все, что мы знаете в вашей системе) и введем ping, а затем веб-адрес, IP-адрес которого вы хотите получить. Другими словами, если мы хотим узнать IP-адрес SMTP-сервера на smtp.livehotgmail.com, мы должны ввести

```
ping smtp.livehotgmail.com
```

и вы получите ответ, похожий на (примечание: это не настоящий веб-адрес)

```
PING smtp.hot.glbdns.livehotgmail.com (10.75.161.202): 56 data bytes
```

Он показывает, что IP-адрес - 10.75.161.202. Сделайте это для SMTP-сервера вашей почтовой службы и введите это в серверную часть кода. Если ваш SMTP-сервер требует аутентификации, нам необходимо получить версию Base-64 вашего имени пользователя и пароля. Есть много веб-сайтов, которые сделают это за нас, например

www.motobit.com/util/base64-decoder-encoder.asp

Enter your username and encrypt it to Base-64 and then do the same with your password. Copy and paste the results into the relevant section in the code. Also, change the FROM and TO sections of the code to your own e-mail address and the e-mail address of the recipient.

Листинг 17-3. Код для проекта 48

```
// Проект 48 - Система оповещений по электронной почте

#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define time 1000
#define emailInterval 60
#define HighThreshold 28
#define LowThreshold 10

// Провод данных подключен к контакту 3 на Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float tempC, tempF;
char message1[35], message2[35];
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
unsigned long lastMessage;

// Настраиваем экземпляр oneWire для связи с любыми устройствами OneWire (не только с
Maxim / Dallas ИС)
OneWire oneWire(ONE_WIRE_BUS);

// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств
DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };

byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 };
byte ip[] = { 192,168,0, 104 };
byte server[] = { 10, 254, 30, 60 };  // Mail server address. Change this to your own mail servers IP.
```

```
EthernetClient client;

void sendEmail(char subject[], char message1[], char message2[], float temp) {
  Serial.println("connecting...");

 if (client.connect(server, 25)) {
   Serial.println("connected");
   client.println("EHLO MYSERVER");  delay(time);
   client.println("AUTH LOGIN"); delay(time);
   client.println("lcbWNybbWl2JttnRzLmNvrZSbQ==");  delay(time);
   client.println("GVOyVGbjLlnZ2VEw");  delay(time);
   client.println("MAIL FROM:<sales@earthshineelectronics.com>");      delay(time);
   client.println("RCPT TO:<fred@crunchytoad.com>");      delay(time);
   client.println("DATA");       delay(time);
   client.println("From: < sales@earthshineelectronics.com >");       delay(time);
   client.println("To: < fred@crunchytoad.com >");       delay(time);
   client.print("SUBJECT: ");
   client.println(subject);       delay(time);
   client.println();      delay(time);
   client.println(message1);      delay(time);
   client.println(message2);      delay(time);
   client.print("Temperature: ");
   client.println(temp);   delay(time);
   client.println(".");      delay(time);
   client.println("QUIT");      delay(time);
   Serial.println("Email sent.");
   lastMessage=millis();
 } else {
   Serial.println("connection failed");
 }

}

void checkEmail() {
  while (client.available()) {
   char c = client.read();
   Serial.print(c);
 }

 if (!client.connected()) {
   Serial.println();
   Serial.println("disconnecting.");
   client.stop();
 }
}

// функция для получения температуры устройства
void getTemperature(DeviceAddress deviceAddress)
{
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

```
void setup()
{
 lastMessage = 0;
 Ethernet.begin(mac, ip);
 Serial.begin(9600);

     // Запускаем библиотеку датчиков
  sensors.begin();
    // устанавливаем разрешение
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
 delay(1000);
}
void loop()
{
  sensors.requestTemperatures();
  getTemperature(insideThermometer);
  Serial.println(tempC);
  if (tempC >= HighThreshold && (millis()>(lastMessage+(emailInterval*1000)))) {
    Serial.println("High Threshhold Exceeded");
    char message1[] = "Temperature Sensor\0";
    char message2[] = "High Threshold Exceeded\0";
    sendEmail(subject, message1, message2, tempC);
  }
  else if (tempC<= LowThreshold && (millis()>(lastMessage+(emailInterval*1000)))) {
    Serial.println("Low Threshhold Exceeded");
    char message1[] = "Temperature Sensor\0";
    char message2[] = "Low Threshold Exceeded\0";
    sendEmail(subject, message1, message2, tempC);
  }
    if (client.available()) {checkEmail();}
}
```

Загрузите код, а затем откройте окно монитора последовательного порта (МПП). МПП будет повторно отображатьтемпературу с первого датчика. Если температура упадет ниже значения нижнего порога, МПП отобразит«Low Threshold Exceeded -Превышен нижний порог», а затем отправит соответствующее предупреждение по электронной почте.
Если температура превысит верхний порог, отобразится сообщение «High Threshold Exceeded - Превышен верхний порог» и будет отправлено соответствующее предупреждение.

Мы можем проверить это, установив верхний порог чуть выше температуры окружающей среды, а затем подогревая датчик, пока температура не поднимется выше порогового значения. Это приведет в действие систему оповещения.
Обратите внимание, что в течение первых 60 секунд система игнорирует любые ситуации с предупреждением о температуре. Она начнет отправлять оповещения только по прошествии 60 секунд. Если пороговые значения были превышены, система предупреждений будет продолжать отправлять электронные письма до тех пор, пока температура не упадет до приемлемого уровня. Электронные письма будут отправляться каждый интервал электронной почты в секундах, когда превышены пороговые значения. Мы можете настроить этот интервал по своему усмотрению.
После того, как электронное письмо будет отправлено, система будет ждать, пока от клиента не будет получена успешная квитанция, а затем отобразит ответ. Вы можете использовать эти данные для отладки системы, если что-то пойдет не так, как планировалось.

Проект 48 - Система оповещений по электронной почте - Обзор кода

Во-первых, библиотеки подключены:

```
#include <Ethernet.h>
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```

Затем мы обеспечиваем задержку в миллисекундах при отправке данных на сервер.

```
#define time 1000
```

За этим определением следует время в секундах между отправкой каждого электронного письма.

```
#define emailInterval 60
```

Затем нам нужно установить высокий и низкий уровни температуры, которые вызовут предупреждение:

```
#define HighThreshold 40 // Максимально допустимая температура
#define LowThreshold 10  // Самая низкая температура
```

Затем вы устанавливаете пин и точность для датчиков.

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Объявляем числа с плавающей запятой для хранения температуры,

```
float tempC, tempF;
```

затем пара массивов символов, которые сохранят сообщение в электронной почте

```
char message1[35], message2[35];
```

Создайте еще один массив символов для хранения темы электронного письма. Это объявлено и инициализировано:

```
char subject[] = "ARDUINO: TEMPERATURE ALERT!!\0";
```

Поскольку вы не хотите бомбардировать пользователя сообщениями электронной почты после превышения пороговых значений, нам необходимо сохранить время, когда было отправлено последнее электронное письмо. Это будет сохранено в беззнаковом целом числе с именем lastMessage:

```
unsigned long lastMessage;
```

Экземпляры для датчика настраиваются вместе с адресом датчика:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
```

Определены MAC и IP-адрес Ethernet Shield. Обратите внимание, что MAC-адрес должен быть уникальным номером, чтобы отличать устройство от миллиардов других подключенных к Интернету устройств в мире. Обычно достаточно генерации случайного адреса. Обратите внимание, что невозможно подключиться к устройству из-за пределов своей домашней сети без дополнительных настроек сети:

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
byte ip[] = { 192,168,0, 105 };
```

Затем мы устанавливаем IP-адрес своего почтового SMTP-сервера. Его необходимо изменить на наш собственный IP-адрес, иначе код не будет работать.

```
byte server[] = { 10, 234, 219, 95 };
```

Создается экземпляр клиента, которому присваивается имя.

```
EthernetClient client
```

Затем идет первая из наших собственных функций. Она выполняет работу по отправке электронной почты на сервер. Для функции требуются четыре параметра: тема электронного письма, первая строка сообщения, вторая строка сообщения и, наконец, температура.

```
void sendEmail(char subject[], char message1[], char message2[], float temp) {
```

Пользователю сообщается, что мы пытаемся подключиться:

```
Serial.println("connecting...");
```

Затем мы проверяем, подключился ли клиент. Если это так, выполняется код в блоке if. Для функции connect () также требуются IP-адрес и порт сервера. Это адрес и порт нашего почтового SMTP-сервера. Изменим IP-адрес и порт в соответствии с нашими требованиями.

```
if (client.connect(server, 25)) {
```

Сначала пользователю сообщается, что вы подключились к клиенту. В данном случае клиентом является SMTP-сервер вашей электронной почты.

```
Serial.println("connected");
```

Теперь вы отправляете команды на сервер почти так же, как в Project 46. Во-первых, вы должны представиться SMTP-серверу. Это делается с помощью команды EHLO и сведений о сервере. После каждой команды вы должны подождать некоторое время, чтобы команда могла быть обработана. Я обнаружил, что для моего сервера требуется 1000 миллисекунд; вам может потребоваться увеличить или уменьшить это число.

```
client.println("EHLO MYSERVER");  delay(time); // авторизоваться
```

Это похоже на процедуру «рукопожатия» между сервером и клиентом, в которой они представляются друг другу. Далее нам необходимо авторизовать соединение. Если наш SMTP-сервер не требует авторизации, мы можем закомментировать эту строку, а также строки имени пользователя и пароля.

```
client.println("AUTH LOGIN"); delay(time); // разрешить
```

Иногда серверу требуется незашифрованный логин, и в этом случае мы должны отправить AUTH PLAIN, а также имя пользователя и пароль в виде обычного текста.

Затем на сервер должны быть отправлены зашифрованные Base-64 имя пользователя и пароль:

```
client.println("lcbWNybbWl2JttnRzLmNvrZSbQ==");  delay(time);
client.println("GVOyVGbjLlnZ2VEw");  delay(time);
```

Затем нам нужно сообщить серверу, от кого идет почта и кому она будет отправляться:

```
client.println("MAIL FROM:<sales@earthshineelectronics.com>");      delay(time);
client.println("RCPT TO:<fred@crunchytoad.com>");      delay(time);
```

Их необходимо изменить на наш собственный адрес электронной почты и адрес получателя. Большинство серверов SMTP позволяют отправлять электронную почту только с адреса электронной почты изсобственного домена (например, вы не можете отправлять электронную почту из учетной записи Hotmail спомощью сервера Yahoo).

Затем идет команда DATA, которая сообщает серверу, что дальше идут данные электронной почты, то есть материал, который будет виден получателю.

```
client.println("DATA");       delay(time);
```

Вы хотите, чтобы получатель видел, кому и от кого отправлено электронное письмо, чтобы они снова были включены для удобства получателя.

```
client.println("From: < sales@earthshineelectronics.com >");       delay(time);
client.println("To: < fred@crunchytoad.com >");       delay(time);
```

Затем мы отправляем тему электронного письма. Это слово «SUBJECT:», за которым следует тема, переданная в функцию:

```
client.print("SUBJECT: ");
client.println(subject);       delay(time);
```

Перед телом сообщения электронной почты необходимо отправить пустую строку, чтобы отделить ее от заголовков.

```
client.println();      delay(time);
```

Далее следуют две строки сообщения, переданного функции.

```
client.println(message1);      delay(time);
client.println(message2);      delay(time);
```

Затем мы включаем температуру:

```
client.print("Temperature: ");
client.println(temp);   delay(time);
```

Все электронные письма должны заканчиваться точкой в отдельной строке, чтобы сообщить серверу, что мы закончили:

```
client.println(".");      delay(time);
```

Затем мы отправляем команду QUIT для отключения от сервера:

```
client.println("QUIT");      delay(time);
```

Наконец, пользователю сообщается, что электронное письмо (e-mail) было отправлено:

```
Serial.println("Email sent.");
```

Затем мы сохраняем текущее значение millis () в lastMessage, так как вы будете использовать его позже, чтобы увидеть, прошел ли указанный интервал между отправками сообщений.

```
lastMessage=millis();
```

Если соединение с клиентом не было успешным, электронное письмо не отправляется, а пользователь сообщает:

```
 } else {
   Serial.println("connection failed");
 }
```

Затем идет функция для чтения ответа от клиента:

```
void checkEmail() { // смотрим, доступны ли какие-либо данные от клиента
```

Пока данные доступны для считывания с клиента

```
while (client.available()) {
```

Код сохраняет этот байт в переменной c

```
char c = client.read();
```

затем выводит содержимое переменной c в окно МПП

```
Serial.print(c);
```

Если клиент НЕ подключен:

```
if (!client.connected()) {
```

пользователь информируется, система отключается, а подключенный клиент останавливается.

```
Serial.println();
Serial.println("disconnecting.");
client.stop();
```

Далее следует функция, которую вы использовали ранее для получения температуры от однопроводного датчика.

```
void getTemperature(DeviceAddress deviceAddress)
{
  tempC = sensors.getTempC(deviceAddress);
  tempF = DallasTemperature::toFahrenheit(tempC);
}
```

затем следует процедура настройки, которая просто настраивает Ethernet и датчики.

```
void setup()
{
Ethernet.begin(mac, ip);
 Serial.begin(9600);

     // Запускаем библиотеку датчиков
  sensors.begin();

    // устанавливаем разрешение
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);

 delay(1000);
}
```

Наконец, главный цикл программы:

```
void loop()
```

Вы начинаете с запроса температуры из библиотеки DallasTemperature.

```
sensors.requestTemperatures();
```

затем вызовите функцию getTemperature, передав ей адрес датчика

```
getTemperature(insideThermometer);
```

который затем отображается в окне МПП.

```
Serial.println(tempC);
```

Затем вы проверяете эту температуру, чтобы увидеть, достигла ли она вашего верхнего порога или превышает ее. Если да, то будет отправлено соответствующее электронное письмо. Однако вы хотите отправлять только одно электронное письмо каждые (emailInterval * 1000) секунд, поэтому проверьте также, что значение millis () больше, чем время последней отправки сообщения электронной почты (lastMessage) плюс интервал времени.Если true, код выполняется.

```
if (tempC >= HighThreshold && (millis()>(lastMessage+(emailInterval*1000)))) {
```

Пользователь информируется, а затем отправляются две строки, составляющие сообщение электронной почты:

```
Serial.println("High Threshhold Exceeded");
char message1[] = "Temperature Sensor\0";
char message2[] = "High Threshold Exceeded\0";
```

Затем вызывается функция sendEmail, передающая ей параметры, составляющие тему, первую и вторую строку сообщения и текущую температуру:

```
sendEmail(subject, message1, message2, tempC);
```

Если порог высокой температуры не был достигнут, мы проверяем, не упала ли она ниже порога низкой температуры. Если да, выполняем ту же процедуру с соответствующим сообщением.

```
else if (tempC<= LowThreshold && (millis()>(lastMessage+(emailInterval*1000))))
    Serial.println("Low Threshhold Exceeded");
    char message1[] = "Temperature Sensor\0";
    char message2[] = "Low Threshold Exceeded\0";
    sendEmail(subject, message1, message2, tempC);
  }
```

Наконец, мы проверяем, есть ли какие-либо данные, готовые к получению от клиента (после отправки электронного письма), и отображаем результаты:

```
if (client.available()) {checkEmail();}
```

Эти данные полезны для целей отладки.

Этот проект дал вам базовые знания по отправке электронной почты из Arduino с Ethernet Shield. Вы можете использовать это для отправки предупреждений или отчетов, когда произошло какое-либо действие, например, было обнаружено, что человек входит в комнату или был открыт ящик. Система также может выполнять другие действия, например, открывать окно, если температура в комнате становится слишком высокой, или пополнять аквариум, если уровень воды падает слишком низко.

Далее вы узнаете, как отправлять данные из Arduino в Twitter.

Проект 49 - Twitterbot

Мы снова будем использовать схему с двумя датчиками температуры. На этот раз мы будем регулярно отправлять обновления о состоянии двух датчиков в Twitter. Это даст нам простую систему для проверки состояния любых датчиков, которые мы подключили к Arduino.

Twitter - это служба микроблогов, которая позволяет отправлять миниатюрные сообщения в блоге или «твиты» длиной до 140 символов. Твиты общедоступны для всех, кто выполняет поиск, или для тех, кто подписался на ваш блог. Twitter невероятно популярен, и к нему можно получить доступ из любого веб-браузера или из одного из множества доступных клиентов Twitter, включая приложения для мобильных телефонов. Это делает его идеальным для отправки простых коротких фрагментов информации, которые вы можете проверить в пути.

Вам нужно будет перейти на Twitter.com и создать новую учетную запись. Я рекомендую создать учетную запись только для твитов с вашего Arduino.

По состоянию на 31 августа 2010 г. Twitter изменил свою политику в отношении сторонних приложений, получающих доступ к веб-сайту. Сейчас используется метод аутентификации, известный как OAuth, что очень затрудняет отправку твитов непосредственно с Arduino; до этого изменения это был простой процесс. На данный момент твитнуть можно только через третье лицо. Другими словами, вы отправляете твит на веб-сайт или прокси-сервер, который будет твитнуть от вашего имени, используя токен OAuth (код авторизации). Текущий Твиттербиблиотека использует этот метод.

После того, как вы настроите свою учетную запись Twitter (или воспользуетесь существующей), введите приведенный ниже код.

Введите код

Перед загрузкой кода нам понадобится токен для учетной записи Twitter. Используемая нами библиотека была создана NeoCat и использует его веб-сайт в качестве прокси для отправки твита. Это означает, что мы должны сначала получить токен, который представляет собой зашифрованную версию нашего имени пользователя и пароля, чтобы получить доступ к веб-сайту Twitter. Для этого посетите веб-сайт NeoCat по адресуhttp://arduino-tweet.appspot.com и щелкнем ссылку «Шаг 1», чтобы получить токен. Скопируйте и вставьте это в раздел токенов кода.

Обратите внимание, что, поскольку вы используете прокси-сервер и должны указать свое имя пользователя и пароль в Твиттере для получения токена, рекомендуется создать новую учетную запись в Твиттере и сохранить ее анонимность (т. е. не добавлять имена или адреса электронной почты в профиль Twitter этой учетной записи). Я считаю, что при желании можно безопасно использовать библиотеку со своей учетной записью, но лучше перестраховаться.

Затем щелкните ссылку «Шаг 2» и получите два набора библиотек, на которых основан код: <SPI.h> и <Ethernet.h>.Установите их в папку с библиотеками IDE Arduino. Вам нужно будет перезапустить среду IDE, прежде чем вы сможете их использовать. В библиотеке Twitter также есть несколько примеров, которые вы можете попробовать. Если вы хотите узнать больше о библиотеке Twitter, вы можете найти ее на игровой площадке Arduino по адресу www.arduino.cc/playground/Code/TwitterLibrary.

После установки вашего токена и библиотек введите и загрузите код из Листинга 17-4.

Listing 17-4. Код для проекта 49

```
#include <Ethernet.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SPI.h>

// Провод данных подключен к контакту 3 на Arduino
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12

float itempC, itempF, etempC, etempF;
boolean firstTweet = true;

// Настраиваем экземпляр oneWire для связи с любыми устройствами OneWire

OneWire oneWire(ONE_WIRE_BUS);

// Передаем нашу ссылку oneWire в Dallas Temperature.
DallasTemperature sensors(&oneWire);

// массивы для хранения адресов устройств

DeviceAddress insideThermometer = { 0x28, 0x44, 0x12, 0xC2, 0x03, 0x00, 0x00, 0x92 };
DeviceAddress outsideThermometer = { 0x28, 0xA5, 0x02, 0xC2, 0x03, 0x00, 0x00, 0xF0 };byte

mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 };

// Ваш токен для твита (получите его на http://arduino-tweet.appspot.com/)
Twitter twitter("19735326-neITsUBnLTZHgN9UGaRkcGvAe9vYuaRP7E55K26J"); // DuinoBot

unsigned long interval = 600000; // 10 минут
unsigned long lastTime; // время с момента последнего твита

// Сообщение для публикации
char message[140], serialString[60];
```

```
// функция для получения температуры для устройства
void getTemperatures()
{
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}

void tweet(char msg[]) {
   Serial.println("connecting ...");
  if (twitter.post(msg)) {
    int status = twitter.wait();
    if (status == 200) {
      Serial.println("OK. Tweet sent.");
      Serial.println();
      lastTime = millis();
      firstTweet = false;
    } else {
      Serial.print("failed : code ");
      Serial.println(status);
    }
  } else {
    Serial.println("connection failed.");
  }
}

void setup()
{
  Ethernet.begin(mac);
  Serial.begin(9600);
   sensors.begin();
    // set the resolution
  sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
  sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
  sensors.requestTemperatures();

  getTemperatures();

  while (firstTweet) {
    sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from Arduino. %ld",
int(itempC), int(itempF), int(etempC), int(etempF), millis());
    tweet(message);
  }
}

void loop()
{
  Ethernet.maintain();
  sensors.requestTemperatures();
  sprintf(serialString, "Internal Temp: %d C  %d F. External Temp: %d C %d F", int(itempC),
```

```
int(itempF), int(etempC), int(etempF));
  delay(500);
  Serial.println(serialString);
  Serial.println();

  if (millis() >= (lastTime + interval)) {
    sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from Arduino. %ld",
int(itempC), int(itempF), int(etempC), int(etempF), millis());
       delay(500);
      tweet(message);
    }

  delay(10000); // 10 seconds
}
```

После того, как вы загрузили код в Arduino, откройте окно МПП. Arduino попытается подключиться к Twitter (на самом деле веб-сайту NeoCat) и отправить твит. Если первый твит будет успешным, вывод в окне МПП будет примерно таким:

```
connecting ...
OK. Tweet sent.

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F

Internal Temp: 26 C  79 F. External Temp: 26 C 79 F
```

Когда программа запускается в первый раз, она измеряет температуру, а затем продолжает попытки подключиться к Twitter в процедуре настройки, прежде чем она перейдет в основной цикл. Она не остановится, пока не подключится успешно. Если программе не удается подключиться, вы получите сообщение «сбой: код 403» или «сбой подключения». Если твит будет успешным, он не будет твитать снова, пока не пройдет период интервала. По умолчанию это значение составляет 10 минут, но мы можем его изменить. Твиттер ограничивает нас 350 запросами в час, так что не переусердствуйте. Теперь мы можем получить доступ к веб-сайту Twitter и просмотреть учетную запись из любого места, чтобы проверить показания температуры.

Посмотрим, как работает этот код.

Проект 49 - Twitterbot - Обзор кода

Программа начинается с включения соответствующих библиотек:

```
#include <Ethernet.h>
#include <Twitter.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SPI.h>
```

Далее устанавливаются определения для датчиков:

```
#define ONE_WIRE_BUS 3
#define TEMPERATURE_PRECISION 12
```

Вы создаете четыре floats значения для температур, на этот раз для внутренних и внешних температур в C и F:

```
float itempC, itempF, etempC, etempF;
```

В первый раз, когда программа пытается создать твит, мы хотим, чтобы она продолжала попытки, пока она успешно не подключится и не отправит сообщение. Таким образом, создается логическое значение, для которого устанавливается значение true, чтобы мы знали, сделали ли мы еще свой первый твит или нет:

```
boolean firstTweet = true;
```

Как и раньше, вы создаете экземпляры для однопроводных датчиков и датчиков температуры, а также адреса для двух датчиков:

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

DeviceAddress insideThermometer = { 0x10, 0x7A, 0x3B, 0xA9, 0x01, 0x08, 0x00, 0xBF };
DeviceAddress outsideThermometer = { 0x10, 0xCD, 0x39, 0xA9, 0x01, 0x08, 0x00, 0xBE};
```

Мы даем шилду Ethernet MAC-адрес:

```
byte mac[] = { 0x64, 0xB9, 0xE8, 0xC3, 0xC7, 0xE2 };
```

Затем мы создаем экземпляр библиотеки Twitter и передаем ему токен для своей учетной записи:

```
Twitter twitter("19735326-neITsUBnLTZHgN9UGaRkcGvAe9vYuaRP7E55K26J");
```

Установлен интервал между твитами.

```
unsigned long interval = 600000; // 10 минут
```

как переменная для хранения времени последнего твита.

```
unsigned long lastTime; // время с момента последнего твита
```

Создаются два символьных массива. Они сохранят сообщение для твита и сообщение, которое мы выведем в окно МПП.

```
char message[140], serialString[60];
```

Теперь мы создаем несколько функций. Первая - это функция получения значений температуры от двух датчиков и их сохранения в ваших переменных.

```
void getTemperatures()
{
  itempC = sensors.getTempC(insideThermometer);
  itempF = DallasTemperature::toFahrenheit(itempC);
  etempC = sensors.getTempC(outsideThermometer);
  etempF = DallasTemperature::toFahrenheit(etempC);
}
```

Далее идет функция, которая будет писать за нас твиты. Для этого требуется один параметр, который представляет собой массив символов, в котором находится наше сообщение.

```
void tweet(char msg[]) {
```

Пользователь проинформирован о том, что мы пытаемся подключиться:

```
Serial.println("connecting ...");
```

Затем мы используем метод post () объекта Twitter для отправки сообщения. Если публикация прошла успешно, функция возвращает true. Если не удается подключиться, возвращается false.

```
  if (twitter.post(msg)) {
```

Если мы успешно подключаемся, проверяется статус сообщения с помощью метода wait (). Он возвращает HTTP код статуса в ответе Twitter.

```
int status = twitter.wait();
```

Если код состояния 200, это способ HTTP-кода сказать, что все в порядке. Другими словами, если твит был успешно отправлен, код в блоке будет выполнен.

```
if (status == 200) {
```

В случае успеха мы сообщаем пользователю:

```
Serial.println("OK. Tweet sent.");
Serial.println();
```

Затем установим lastTime на текущее значение в millis (). Это нужно для того, чтобы мы могли определить, сколько времени прошло с момента последнего твита.

```
lastTime = millis();
```

В первый раз, когда мы выполняем успешный твит, мы хотим, чтобы программа выскочила из цикла while в процедуре настройки и перешла в основной цикл, поэтому мы устанавливаем флаг firstTweet в значение false.

```
firstTweet = false;
```

Если статус не 200, то есть публикация не удалась, то пользователь информируется, и код передается обратно для целей отладки.

```
} else {
      Serial.print("failed : code ");
      Serial.println(status);
}
```

и если вы вообще не смогли подключиться, пользователь будет проинформирован об этом.

```
} else {
    Serial.println("connection failed.");
}
```

Пользователь выполняет свои функции, теперь мы перейдем к процедуре настройки:

```
void setup()
```

Сначала мы запускаем библиотеку Ethernet и передаем ей MAC-адрес:

```
Ethernet.begin(mac);
```

Затем мы начинаем последовательную связь со скоростью 9600 бод и настраиваем датчики, как раньше:

```
Serial.begin(9600);
sensors.begin();
sensors.setResolution(insideThermometer, TEMPERATURE_PRECISION);
sensors.setResolution(outsideThermometer, TEMPERATURE_PRECISION);
```

Температуры запрашиваются, поскольку мы собираемся их использовать:

```
sensors.requestTemperatures()
getTemperatures();
```

Теперь вы пытаетесь отправить свой первый твит. Цикл while для этого будет продолжать работать, пока для firstTweet установлено значение true:

```
while (firstTweet) {
```

Затем мы используете команду sprintf для компиляции твита в массив message []. Вы передаете ему четыре набора температур, а также значение millis (). Поскольку millis является длинным числом без знака, вы используете спецификатор % ld в sprintf для вывода длинного целого числа. Команда sprintf (форматирование строки для печати) - отличный способ упаковать множество разных битов информации в одну строку.

```
sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from Arduino. %ld",
int(itempC), int(itempF), int(etempC), int(etempF), millis());
```

Функция sprintf () принимает три параметра: переменную, в которой вы будете хранить отформатированные данные (в данном случае сообщение для твита), содержимое строки со спецификаторами, а затем переменные. При этом первая переменная вставляется в строку, где появляется первый % d, вторую переменную, где появляется следующий % d, и так далее.
Четыре спецификатора разделены запятыми. Поэтому в последней строке числа будут разделены запятыми.
Итак, если бы значения переменных были

```
itempC          25
itempF          77
etempC          14
tempF           52
millis()        4576
```

тогда, после запуска команды sprintf, содержимое сообщения будет

```
"Int. Temp: 25 C (77 F) Ext. Temp: 14 C (52 F). Tweeted from Arduino. 4576"
```

Как видите, команда sprintf - это мощный инструмент для преобразования длинных комбинаций строк и чисел в одну строку.

Причина, по которой вы добавляете значение millis () в конец твита, заключается в том, что Twitter не будет публиковать сообщение, которое совпадает с последним отправленным. Если температура не изменилась с момента последнего твита, сообщение будет таким же, и вместо этого Twitter вернет код ошибки. Поскольку вы хотите регулярно обновлять каждый интервал, добавив значение millis () в конец, вы убедитесь, что сообщение отличается от последнего отправленного. Убедитесь, что длина вашего твита не превышает 140 символов; в противном случае на вашей временной шкале Twitter появятся странные сообщения.

Теперь, когда мы скомпилировали свое сообщение, мы передаем его функции tweet ():

```
tweet(message);
```

Далее следует основной цикл, в который мы войдем только в том случае, если первый твит в программе настройки будет успешным:

```
void loop()
```

Сначала вы запускаете команду обслуживания в библиотеке Ethernet. Это сохраняет автоматически назначенный IP-адрес действующим и реальным.

```
Ethernet.maintain();
```

Температуры обновлены.

```
sensors.requestTemperatures();
```

Затем мы используем команду sprintf для компиляции вывода для последовательного монитора. Это удобнее, чем целый список команд Serial.print (), поэтому мы можем использовать его, хотя он увеличивает размер вашего кода.

```
sprintf(serialString, "Internal Temp: %d C  %d F. External Temp: %d C %d F", int(itempC),
int(itempF), int(etempC), int(etempF));
```

Затем после небольшой задержки строка выводится на МПП:

```
delay(500);
Serial.println(serialString);
Serial.println();
```

Затем мы проверяем, прошел ли интервал с момента последнего твита, и, если да, отправляем еще один. Мы вычисляем значение lastTime + interval и смотрим, больше ли текущее значение в millis (), чем оно (то есть период интервала прошел с момента последнего твита). Если это так, мы компилируем новое сообщение и снова пишем в Твиттере.

```
if (millis() >= (lastTime + interval)) {
          sprintf(message, "Int. Temp: %d C (%d F) Ext. Temp: %d C (%d F). Tweeted from
          Arduino. %ld", int(itempC), int(itempF), int(etempC), int(etempF), millis());
          tweet(message);
}
```

Наконец, у нас есть 10-секундная задержка между обновлениями МПП, чтобы мы не засыпали пользователя информацией:

```
delay(10000); // 10 секунд
```

Теперь, когда вы знаете, как отправлять твиты со своего Arduino, вы можете использовать его для любых целей. Как насчет комнатного растения, которое пишет в Твиттере, чтобы сообщить вам, что его нужно поливать? Или датчики вокруг дома, чтобы твитнуть, когда кто-то входит в комнату, дверной звонок, который твитнет, когда кто-то находится у двери, или кошачий клапан, который сообщает вам, когда ваша кошка ушла или вошла в дом? Возможности безграничны.
Теперь вы достигли последнего проекта в своем путешествии. В этом последнем проекте вы будете использовать Ethernet Shield для чтения некоторых данных из Интернета вместо отправки данных.

Project 50 - RSS-программа прогноза погоды

В последнем проекте этой книги снова будет использоваться Ethernet Shield, но вместо передачи данных в веб-службу мы будем использовать Arduino и Ethernet Shield для извлечения данных из Интернета и последующего отображения их в окне МПП. Данные, которые мы собираемся использовать, представляют собой канал RSS (Really Simple Syndication) с веб-сайта www.weather.gov для получения данных о погоде для выбранной вами области в Соединенных Штатах. Этот код легко адаптируется для чтения RSS-ленты погоды из любого другого источника, если вы находитесь за пределами США.
RSS - это веб-формат для публикации часто обновляемой информации, такой как погода, новости и т. Д. Данные представлены в формате XML (Extensible Markup Language), который представляет собой набор правил для кодирования документов в машиночитаемой форме. XML - это простой формат, и совсем не обязательно понимать, как он работает. Arduino просто ищет теги в XML-коде, где хранятся данные о температуре, влажности и давлении, и удаляет эту информацию для отображения.

Вы будете использовать XML-канал для базы BBC Эдвардс в Калифорнии. Если вы хотите использовать другой канал, перейдите в http://www.weather.gov/xml/current_obs/ и выберите свой регион, затем найдите полный адрес XML-данных дляэтого канала. Измените код соответствующим образом, чтобы отображать погоду для этой области.

Что касается устройства, на этот раз вы используете не что иное, как Ethernet Shield, подключенный к Arduino.

Введите код

Подключите шилд Ethernet к Arduino (если его еще нет) и введите код из Листинга 17-5. Спасибо Бобу С. (Xtalker) с форумов Arduino за код.

Листинг 17-5. Код для проекта 50

Получить текущие наблюдения за погодой для авиабазы Эдвардс с сайта `weather.gov` в формате `XML`

```
// Включаем файлы описания для других используемых библиотек (если есть)
//#include <string.h>
#include <Ethernet.h>
#include <SPI.h>

Максимальная длина строки может быть изменена в зависимостиот данных, которые необходимо извлечь
#define MAX_STRING_LEN  20

// Setup vars
char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";
char endTag[3] = {'<', '/', '\0'};
int len;
```

```
// Флаги для отличия тегов XML от элементов документа (т.е. данных)
boolean tagFlag = false;
boolean dataFlag = false;

// Ethernet vars
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 };
byte ip[] = {192, 168, 0, 35};
byte server[] = { 140, 90, 113, 200 }; // www.weather.gov

// Запускаем Ethernet-клиент
EthernetClient client;

void setup()
{
  Serial.begin(9600);
  Serial.println("Starting Weather RSS Reader");
  Serial.println("connecting...");
  Ethernet.begin(mac, ip);
  delay(1000);

  if (client.connect(server, 80)) {
    Serial.println("connected");
    Serial.println("Current weather at Edwards AFB:");
    client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
    client.println();
    delay(2000);
  } else {
    Serial.println("connection failed");
  }
}

void loop() {

  // Чтение последовательных данных из сети:
  while (client.available()) {
    serialEvent();
  }

  if (!client.connected()) {

    client.stop();

      for (int t=0; t<15; t++) {
      delay(60000); // 1 minute
      }

    if (client.connect(server, 80)) {
      client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
      client.println();
      delay(2000);
```

```
    } else {
      Serial.println("Reconnection failed");
    }
  }
}

// Обработка каждого символа из Интернета
void serialEvent() {
  //Читаем символ
        char inChar = client.read();

   if (inChar == '<') {
      addChar(inChar, tmpStr);
      tagFlag = true;
      dataFlag = false;

   } else if (inChar == '>') {
      addChar(inChar, tmpStr);

      if (tagFlag) {
         strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
      }

      // Очистить tmp
      clearStr(tmpStr);

      tagFlag = false;
      dataFlag = true;

   } else if (inChar != 10) {
      if (tagFlag) {
         // Add tag char to string
         addChar(inChar, tmpStr);

         // Добавляем символ тега в строку
         if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
            clearStr(tmpStr);
            tagFlag = false;
            dataFlag = false;
         }
      }

      if (dataFlag) {
         // Добавляем данные в строку
         addChar(inChar, dataStr);
      }
   }
```

```
   // Если LF, обрабатываем строку
   if (inChar == 10 ) {

      // Находим определенные теги и распечатываем данные
      if (matchTag("<temp_f>")) {
             Serial.print("TempF: ");
         Serial.print(dataStr);
      }
          if (matchTag("<temp_c>")) {
             Serial.print(", TempC: ");
         Serial.print(dataStr);
      }
      if (matchTag("<relative_humidity>")) {
             Serial.print(", Humidity: ");
         Serial.print(dataStr);
      }
      if (matchTag("<pressure_in>")) {
             Serial.print(", Pressure: ");
         Serial.print(dataStr);
         Serial.println("");
      }

      // Очистить все строки
      clearStr(tmpStr);
      clearStr(tagStr);
      clearStr(dataStr);

      // Очистить флаги
      tagFlag = false;
      dataFlag = false;
   }
}

// Функция для очистки строки
void clearStr (char* str) {
   int len = strlen(str);
   for (int c = 0; c < len; c++) {
      str[c] = 0;
   }
}

//Функция для добавления символа в строку и проверки ее длины
void addChar (char ch, char* str) {
   char *tagMsg  = "<TRUNCATED_TAG>";
   char *dataMsg = "-TRUNCATED_DATA-";

   // Проверяем максимальный размер строки, чтобы убедиться, что она не становится слишком большой.
   //   Если строка превышает MAX_STRING_LEN, предположите, что она неважна
   // и замените ее предупреждающим сообщением.
```

```
   if (strlen(str) > MAX_STRING_LEN - 2) {
      if (tagFlag) {
         clearStr(tagStr);
         strcpy(tagStr,tagMsg);
      }
      if (dataFlag) {
         clearStr(dataStr);
         strcpy(dataStr,dataMsg);
      }

      // Очищаем временный буфер и флаги, чтобы остановить текущую обработку
      clearStr(tmpStr);
      tagFlag = false;
      dataFlag = false;

   } else {
      // Добавляем символ в строку
      str[strlen(str)] = ch;
   }
}

// Функция для проверки текущего тега на наличие определенной строки
boolean matchTag (char* searchTag) {
   if ( strcmp(tagStr, searchTag) == 0 ) {
      return true;
   } else {
      return false;
   }
}
```

Загрузите код и откройте серийный монитор. Если все работает правильно, вы получите следующий результат:

```
Starting Weather RSS Reader
connecting...
connected
Current weather from Edwards AFB:
TempF: 60.0, TempC: 15.4, Humidity: 100, Pressure: 29.96
```

Каждые шестьдесят секунд дисплей снова обновляется с последними данными. Посмотрим, как работает этот код.

Project 50 - RSS-программа для прогноза погоды - Обзор кода

Программа начинается с включения необходимых библиотек Ethernet:

```
#include <Ethernet.h>
#include <SPI.h>
```

Затем мы определяем максимальную длину строки данных:

```
#define MAX_STRING_LEN  20
```

Вам может потребоваться увеличить это значение, если вы запрашиваете дополнительную информацию из канала. Затем Мы создаем три массива, в которых будут храниться различные строки, которые мы будете обрабатывать (все они имеют определенную длину).

```
char tagStr[MAX_STRING_LEN] = "";
char dataStr[MAX_STRING_LEN] = "";
char tmpStr[MAX_STRING_LEN] = "";
```

Затем мы создаем еще один массив для хранения возможных закрывающих тегов, с которыми вы столкнетесь в XML-фиде:

```
char endTag[3] = {'<', '/', '\0'};
```

Затем мы создаем переменную, которая будет хранить длину обрабатываемой строки в соответствующем разделе кода:

```
int len;
```

Затем мы создаем два флага. Они будут использоваться для различения тегов XML и информации после тегов, которую мы хотим удалить из кода XML.

```
boolean tagFlag = false;
boolean dataFlag = false;
```

Затем мы настраиваем MAC и IP-адрес Ethernet Shield:

```
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xF7, 0x38 };
byte ip[] = {192, 168, 0, 35};
```

Затем IP-адрес веб-сайта www.weather.gov:

```
byte server[] = { 140, 90, 113, 200 }; // www.weather.gov
```

Если вы используете другой веб-сайт для своей ленты новостей, измените этот IP-адрес на URL-адрес, который вы используете.
Затем мы создаем клиентский объект и называем его:

```
EthernetClient client();
```

Далее идет процедура настройки

```
void setup()
```

который начинается с установления последовательной связи на скорости 9600 бод, чтобы вы могли распечатать данныена последовательном мониторе.

```
Serial.begin(9600);
```

Мы сообщаем пользователю название программы и пытаемся подключиться:

```
Serial.println("Starting Weather RSS Reader");
Serial.println("connecting...");
```

Начинается связь по сети Ethernet, передается MAC- и IP-адрес нашего устройства, после чего следует небольшая задержка для подключения:

```
Ethernet.begin(mac, ip);
delay(1000);
```

Затем вы проверяете, успешно ли вы подключились к своему клиенту (веб-сайт www.weather.gov). Мы используем функцию connect () и передаем ей IP-адрес сервера и порт:

```
if (client.connect(server, 80)) {
```

Если да, мы информируем пользователя

```
Serial.println("connected");
Serial.println("Current weather at Edwards AFB:");
```

Затем выполните команду HTML GET для доступа к данным XML из подкаталога, в котором хранится соответствующий канал, после чего следует задержка для обеспечения успешного обмена данными.

```
client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
client.println();
delay(2000);
```

Если соединение не было установлено, вы информируете пользователя о неудачном соединении:

```
  } else {
    Serial.println("connection failed");
  }
}
```

Далее идет основной цикл:

```
void loop() {
```

Поскольку мы выполнили команду GET в цикле настройки, последовательный буфер должен содержать содержимое XML-канала, возвращенного сервером. Итак, пока у нас есть данные

```
while (client.available()) {
```

the `serialEvent()` function is called.

```
serialEvent();
```

вызывается функция serialEvent ().

```
if (!client.connected()) {
```

Эта функция будет вскоре объяснена. Если соединение не было установлено

```
client.stop();
```

Затем вы ждете 15 минут, прежде чем пытаться установить другое соединение. Канал данных обновляется не чаще одного раза в 15 минут, поэтому обновлять информацию раньше этого бессмысленно:

```
if (int t=0; t<15; t++) { // фид обновляется каждые 15 минут
delay(60000); // 1 минута
      }
```

Если мы успешно подключились к клиенту

```
if (client.connect(server, 80)) {
```

затем мы выполняем другую команду GET, чтобы получить последние данные канала XML

```
client.println("GET /xml/current_obs/KEDW.xml HTTP/1.0");
client.println();
delay(2000);
```

и если соединение не удается, пользователь информируется.

```
} else {
      Serial.println("Reconnection failed");
    }
```

Далее идет функция serialEvent (). Цель этой функции - считывать данные из XML-канала и обрабатывать их в соответствии с тем, что он находит.

```
void serialEvent() {
```

Функция начинается с чтения первого символа и сохранения его в inChar:

```
    char inChar = client.read();
```

Теперь нам нужно взглянуть на этот символ и решить, является ли это тегом или данными. Если это тег, мы устанавливаем для tagFlag значение true. Если это данные, мы устанавливаем для параметра dataFlag значение true. Другой флаг каждый раз устанавливается в значение false.

The raw data for the feed looks like:

```
<current_observation version="1.0"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="http://www.weather.gov/view/current_observation.xsd">
        <credit>NOAA's National Weather Service</credit>
        <credit_URL>http://weather.gov/</credit_URL>
        <image><url>http://weather.gov/images/xml_logo.gif</url><title>NOAA's National
Weather Service</title><link>http://weather.gov</link></image>
        <suggested_pickup>15 minutes after the hour</suggested_pickup>
        <suggested_pickup_period>60</suggested_pickup_period>
        <location>Edwards AFB, CA</location>
        <station_id>KEDW</station_id>
        <latitude>34.91</latitude>
        <longitude>-117.87</longitude>
        <observation_time>Last Updated on Apr 30 2013, 9:55 am PDT</observation_time>
        <observation_time_rfc822>Tue, 30 Apr 2013 09:55:00 -0700</observation_time_rfc822>
```

```
<weather>Mostly Cloudy</weather>
<temperature_string>62.0 F (16.4 C)</temperature_string>
<temp_f>62.0</temp_f>
<temp_c>16.4</temp_c>
<relative_humidity>100</relative_humidity>
```

Как видите, каждая часть информации встроена в тег. Например, температура в градусах Фаренгейта имеет тег <temp_f> для начала и </temp_f> в конце. Все, что находится между тегами, - это данные.

Сначала мы проверяем, является ли этот символ символом <. Если да, то это начало тега.

```
if (inChar == '<') {
```

Если это так, м вызываем функцию addChar, которая проверяет, находится ли длина строки в пределах MAX_STRING_LEN, и если да, добавляем символ в нашу строку tmpStr. Мы изучим эту функцию позже.

```
addChar(inChar, tmpStr);
```

Как только вы нашли тег, для tagFlag установлено значение true, а для параметра dataFlag установлено значение false:

```
tagFlag = true;
dataFlag = false;
```

Если вы дойдете до конца тега, найдя символ>

```
} else if (inChar == '>') {
```

затем символ добавляется в строку tmpStr.

```
addChar(inChar, tmpStr);
```

Если вы в настоящее время обрабатываете тег и достигли конца тега, вы можете скопировать весь тег из tmpStr (временная строка) в строку тега (tgrStr). Для этого вы используете команду strncpy.

```
if (tagFlag) {
        strncpy(tagStr, tmpStr, strlen(tmpStr)+1);
}
```

Команда strncpy копирует часть одной строки в другую строку. Для этого требуются три параметра: строка, в которую мы копируем данные, строка, из которой мы копируем данные, и количество символов для копирования. Например, если у нас

```
strncpy(firstString, secondString, 10);
```

затем первые 10 символов secondString копируются в firstString. В этом случае мы копируем все содержимое, определяя длину временной строки (tmpStr) +1 и копируя это количество символов в строку тега.

После того, как временная строка будет скопирована, нам нужно очистить ее, чтобы она была готовадля следующего фрагмента данных. Для этого мы вызываем функцию clearStr и передаем ей строку, которую хотим очистить.

```
clearStr(tmpStr);
```

Для двух флагов установлено значение false, готовое для следующей части информации:

```
tagFlag = false;
dataFlag = true;
```

Если читаемый символ является переводом строки (ASCII 10)

```
} else if (inChar != 10) {
```

затем мы добавляем символ в строку, если в настоящее время обрабатывается тег.

```
if (tagFlag) {
       addChar(inChar, tmpStr);
```

Мы хотим игнорировать конечные теги, поэтому мы проверяем, обрабатываем ли мы тег в данный момент и достигли ли конца тег (путем сравнения с символами endTag).

```
if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) {
```

затем тег игнорируется, строка очищается, а теги устанавливаются в значения по умолчанию.

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

Команда strcmp сравнивает две строки. В нашем случае она сравнивает временную строку (tmpStr) с символами в массиве endTag:

```
strcmp(tmpStr, endTag)
```

Результатом будет 0, если строки совпадают, и другое значение, если нет. Сравнивая его с массивом endTag, мы проверяем наличие любого из трех символов конечного тега.

Если текущая строка - это данные

```
if (dataFlag) {
```

затем мы добавляем текущий символ в строку данных (dataStr).

```
addChar(inChar, dataStr);
```

Приведенный выше код в основном определяет, обрабатываете ли вы тег, и если да, то сохраняет символы в строке тега (tagStr); если это данные, он сохраняет их в строке данных (dataStr). В итоге мы получим тег и данные, хранящиеся отдельно.

Если мы достигли символа перевода строки, значит, мы находимся в конце текущей строки. Итак, теперь нам нужно проверить теги, чтобы узнать, являются ли они данными о температуре, влажности или давлении, которые м ы хотим получить.

```
if (inChar == 10 ) {
```

Для этого мы используем функцию matchTag (к которой мы скоро вернемся), которая проверяет, находится ли указанный тег в строке тега, и возвращает истинное значение. Мы начинаем с поиска температуры по шкале Фаренгейта.

```
<temp_f>

if (matchTag("<temp_f>")) {
```

и, если он найден, распечатывает строку данных, которая, если тег <temp_f>, будет содержать температуру в градусах Фаренгейта.

```
Serial.print("Temp: ");
Serial.print(dataStr);
```

Затем вы проверяете температуру в градусах Цельсия,

```
if (matchTag("<temp_c>")) {
        Serial.print(", TempC: ");
        Serial.print(dataStr);
}
```

влажность,

```
if (matchTag("<relative_humidity>")) {
        Serial.print(", Humidity: ");
        Serial.print(dataStr);
      }
```

и давление.

```
if (matchTag("<pressure_in>")) {
         Serial.print(", Pressure: ");
         Serial.print(dataStr);
         Serial.println("");
}
```

Затем все строки очищаются и готовы к переходу к следующей строке.

```
clearStr(tmpStr);
clearStr(tagStr);
clearStr(dataStr);
```

и теги тоже очищаются.

```
tagFlag = false;
dataFlag = false;
```

У нас есть свои пользовательские функции, начиная с функции очистки строки (clearStr).

```
void clearStr (char* str) {
```

который просто находит длину строки, переданной функции, с помощью команды strLen ()

```
int len = strlen(str);
```

затем использует цикл for для заполнения каждого элемента массива символом ASCII 0 (нулями).

```
for (int c = 0; c < len; c++) {
      str[c] = 0;
}
```

Следующая функция - это функция addChar. Мы передаем ему читаемый в данный момент символ и текущую строку в качестве параметров.

```
void addChar (char ch, char* str) {
```

Мы определяем два новых массива символов и сохраняем в них сообщения об ошибках:

```
char *tagMsg  = "<TRUNCATED_TAG>";
char *dataMsg = "-TRUNCATED_DATA-";
```

Если мы обнаружим, что длина строк превышает MAX_STRING_LEN, мы заменим их этими сообщениями об ошибках.

Теперь мы проверяем длину строки, чтобы увидеть, достигла ли она максимальной длины:

```
if (strlen(str) > MAX_STRING_LEN - 2) {
```

Если это так, мы в настоящее время обрабатываем тег

```
if (tagFlag) {
```

затем строка тега очищается, и мы копируем сообщение об ошибке в строку тега.

```
clearStr(tagStr);
strcpy(tagStr,tagMsg);
```

Если вы обрабатываете данные, строка данных очищается, и вы копируете сообщение об ошибке данных в строку данных.

```
if (dataFlag) {
         clearStr(dataStr);
         strcpy(dataStr,dataMsg);
}
```

Временная строка и теги очищаются

```
clearStr(tmpStr);
tagFlag = false;
dataFlag = false;
```

и если длина строки не превышает максимальную длину, мы добавляем текущий считанный символ в строку. Мы используем длину строки, чтобы узнать последний символ, то есть следующее место, куда мы можем добавить символ.

```
} else {
      // Add char to string
      str[strlen(str)] = ch;
}
```

Наконец, вы попадаете в функцию matchTag, которая используется для проверки того, что тег поиска, переданный ему в качестве параметра, был найден или нет, и если да, возвращает true или false соответственно:

```
boolean matchTag (char* searchTag) {
```

Функция имеет логический тип, поскольку она возвращает логическое значение и требует в качестве параметра символьный массив:

```
if ( strcmp(tagStr, searchTag) == 0 ) {
      return true;
   } else {
      return false;
   }
}
```

Изменив URL-адрес XML-канала и теги, найденные в этом канале, вы можете использовать этот код для поиска фрагментов данных в любом RSS-канале по вашему желанию. Например, вы можете использовать погодные каналы Yahoo на http://weather.yahoo.com, перейти в регион, который вы хотите просмотреть, и нажать кнопку RSS. Затем URL-адрес этого канала можно ввести в код. Вы можете просмотреть необработанный источник канала, щелкнув правой кнопкой мыши и выбрав пункт меню правой кнопки мыши, чтобы просмотреть источник. Затем вы можете просмотреть теги и изменить код, чтобы найти соответствующую информацию.

Этот последний проект показал вам, как использовать Ethernet Shield для получения информации из Интернета.

Раньше вы отправляли данные из Shield во внешние источники. В этом проекте вместо этого вы читаете данные из Интернета. Вместо того, чтобы отображать данные о погоде в окне последовательного монитора, вы можете использовать навыки, полученные в предыдущих проектах, для отображения их на ЖК-экране или на светодиодном точечно-матричном дисплее.

Резюме

В этой последней главе было показано, как подключить Arduino к Интернету либо с целью отправки данных в виде обслуживаемой веб-страницы, твита в Twitter, электронной почты, данных датчиков, отправленных в Xively, либо для запроса веб-страницы и удаление данных с нее для вашего собственного использования. Возможность подключения Arduino к локальной сети или Интернету открывает целый новый список потенциальных проектов. Данные могут быть отправлены в любую точку вашего дома или офиса, где доступен порт Ethernet, или данные могут быть прочитаны из Интернета для Arduino для обработки, отображения или действий.

Например, вы можете использовать текущую погодную ленту, чтобы определить, скоро ли идет дождь, и предупредить вас, чтобы вы принесли белье с бельевой веревки или закрыли окно на крыше. То, что вы делаете со своим Arduino после его подключения, ограничивается только вашим воображением.

Я надеюсь, что вам понравились 50 проектов, представленных вам в Основы Arduino, и что вам очень нравится использовать полученные знания для создания собственных фантастических творений Arduino. Свяжитесь с нами в Twitter или G +, если вам понадобится дополнительная помощь. Всего наилучшего, Майк.

Предметы и понятия, затронутые в главе 17:

- Как вручную назначить устройству MAC- и IP-адрес
- Понятие клиента и сервера
- Как прослушивать клиентские соединения с помощью команды client.connected ()

- Как прослушивать клиентские соединения с помощью команды client.connected ()
- Отправка HTML-кода с помощью печати клиенту
- Использование Arduino в качестве веб-сервера
- Подключение к Arduino из веб-браузера
- Проверка данных доступна с помощью команды client.available ().
- Чтение данных с помощью команды client.read ()
- Отправка данных в Xively, просмотр данных в виде графиков и т. д.
- Отправка твитов на Twitter.com с Arduino через прокси-сервер.
- Отправка электронной почты из Arduino
- Получение данных из RSS-канала и их анализ для отображения
- Копирование строк с помощью команд strcpy и strncpy
- Сравнение строк с помощью команды strcmp
- Заполнение ячеек памяти командой memset
- Определение размера массивов с помощью команды sizeof
- Обработка строк с помощью команды sprintf
- Определение длины строк с помощью команды strlen
- Поиск подстрок с помощью команды strstr
- Поиск IP-адресов с помощью команды ping
- Проверка подключения клиента с помощью команды client.connect ()
- Создание соединения Ethernet с помощью команды Ethernet.begin (mac, ip)
- Шифрование имен пользователей и паролей до Base-64 с помощью утилит веб-сайта
- Использование библиотеки Ethernet.h для автоматического назначения IP-адреса
- Использование сообщений библиотеки Twitter и команд ожидания
- Поиск тегов в XML-канале RSS