

Community Experience Distilled

Освоение Arduino

Проектный подход к электронике, схемам и программированию

Джон Хофман

Arduino

Проектный подход к электронике, схемам и
программированию

Авторы

Джон Хоффман имеет более чем 25-летний опыт работы в области информационных технологий. Все эти годы Джон работал в области системного администрирования, сетевого администрирования, сетевой безопасности, разработки приложений и архитектуры. В настоящее время Джон работает Enterprise Software Manager для Syntech Systems. Джон активно занимается разработкой для платформы iOS с 2008 года. Это включает в себя несколько приложений, которые он опубликовал в App Store, приложения, которые он написал для третьих лиц, и многочисленные корпоративные приложения. Что действительно движет Джоном, так это проблемы в области информационных технологий, и для него нет ничего более захватывающего, чем преодоление трудностей.

Некоторые другие интересы Джона - бейсбол и баскетбол. Джону также очень нравится тхэквондо, где он и его старшая дочь вместе заработали свои черные пояса в начале 2014 года. Ким (его жена) получила свой черный пояс в конце 2014 года.

Оглавление

	1
1: Arduino	7
Arduino	8
Что такое Ардуино?	10
Arduino UNO R3	11
Питание Arduino	12
Использование выводов Vin / GND для питания Arduino	13
Использование внешнего источника питания	13
Использование разъема USB для питания Arduino	14
Arduino	15
Arduino	17
Цифровые выводы	18
Аналоговые входные выводы	18
ШИМ выводы	18
Выводы питания	19
Выводы монитора последовательного порта	20
SPI выводы	20
Arduino	20
Arduino Micro	21
Arduino Mega 2560	21
Lilypad	22
Arduino Nano	23
Стандартные платы	23
	26
2:	27
	28
	29
Dupont (перемычки)провода	34
Прототипирование	35
Четыре базовых блока электронного проекта	36
Создание схемы	37
Первый прототип	38
Резюме	41
Глава 3: Arduino IDE	42
Скетч Arduino	42
Arduino IDE	43
Изучение IDE	44
Настройка Arduino в среде IDE	45
Веб-редактор Arduino	47

Изучение	49
Настройка Arduino в среде IDE	50
Примеры	50
Библиотеки Arduino	54
	58
Hello World	61
	62
	64
Глава 4: Программирование Arduino - Основы	65
Фигурные скобки	66
Точка с запятой	66
Комментарии	66
Переменные	67
Типы данных	68
Логический	68
Байт	68
Целое число	69
Длинная	69
Двойной и плавающий	70
Символ	70
Массивы	70
Массивы символов	72
Константы	73
Арифметические функции	74
Операторы сравнения	75
Логические операторы	75
	76
Принятие решений	76
Зацикливание	79
Функции	81
	83
Глава 5: Программирование Arduino - помимо основ	84
Установка режима цифрового вывода	85
Цифровая запись	86
Цифровое чтение	87
	88
Аналоговое чтение	90
Структуры	91
Союзы	93
Добавление вкладок	94
Работа с вкладками	98
Объектно-ориентированное программирование	100

Библиотека строк	103
	105
Глава 6: Датчик движения	106
	106
Необходимые компоненты	109
Принципиальные схемы	110
Код	112
Запуск проекта	113
	114
	114
Глава 7: Датчики окружающей среды	115
	115
Необходимые компоненты	118
Принципиальные схемы	118
Код	119
	127
	128
	128
Глава 8: Объезд препятствий и обнаружение столкновений	129
	129
Датчик столкновения	130
Датчик предотвращения препятствий	131
Ультразвуковой дальномер	132
Необходимые компоненты	133
Принципиальные схемы	134
Код	135
Запуск проекта	137
	138
	139
Глава 9: Развлечение со светом	140
	140
Необходимые компоненты	143
Принципиальные схемы	143
Код	144
RGB LED	144
NeoPixel шилд	146
Запуск проекта	149
	149
	150
Глава 10: Развлечения со звуком	151

	151
Необходимые компоненты	153
Принципиальные схемы	154
Код	154
Использование функции тона	155
Воспроизведение рингтона в формате RTTTL	158
	162
	162
Глава 11: Использование ЖК-дисплеев	163
	163
Необходимые компоненты	165
Принципиальные схемы	166
Код	167
Рисование линии	168
Отображение текста	169
Вращающийся текст	171
Основные формы	171
Заполнение формы	172
Прямоугольник	173
Закрашенный прямоугольник	173
Прямоугольник с закругленными углами	174
Закругленный прямоугольник с заливкой	175
	176
	176
Глава 12: Распознавание речи и синтез голоса	177
	177
Необходимые компоненты	179
Принципиальные схемы	180
Код	180
Запуск проекта	183
	183
	184
Глава 13: Двигатели постоянного тока и контроллеры двигателей	185
	185
Необходимые компоненты	189
Принципиальные схемы	190
Код	192
Запуск проекта	193
	193
	194
14:	195

	195
Необходимые компоненты	197
Принципиальные схемы	198
Код	199
	201
	201
Глава 15: Использование реле	202
	202
Необходимые компоненты	206
Принципиальные схемы	206
Код	208
	209
	209
Глава 16: Удаленное управление Arduino	210
	210
Необходимые компоненты	213
Принципиальные схемы	214
Код	216
	220
	221
Глава 17: Создание робота	222
	222
Шасси и движение	223
Двигатели и мощность	227
	229
Удаленное управление роботом	233
Отзывы пользователей	233
Заставляем вещи вращаться	234
Не робототехнические проекты	235
Метеостанция	235
Умный термостат	235
Датчик приближения	236
	236
	236
Глава 18: Bluetooth LE	238
	238
Радио Bluetooth LE	240
Топология сети	242
Bluetooth LE вещание	242
Соединения Bluetooth LE	243
Профили Bluetooth LE	244

Общий профиль доступа (GAP)	244
Профиль универсального атрибута (GATT)	246
Модуль Bluetooth HM-10	250
Необходимые компоненты	251
Принципиальные схемы	252
Проект 1 - последовательная связь	253
Тестовая команда	256
Запрос версии программного обеспечения	256
Восстановить заводские настройки по умолчанию	257
Модуль перезапуска	257
Запросить MAC-адрес (управление доступом к среде)	257
Имя набора	257
Имя запроса	257
Установите рекламный интервал	258
Запросить рекламный интервал	258
Установить тип рекламы	259
Тип рекламы запроса	259
Установить скорость передачи	259
Скорость передачи запроса	260
Установить идентификатор характеристики	260
Установить идентификатор службы	260
Идентификатор службы запросов	260
Установить роль	260
Роль запроса	261
Очистить последнее подключенное устройство	261
Попробуйте подключиться к последнему подключенному устройству	261
Попробуйте подключиться к адресу	262
Установить пин-код	262
Запросить пин-код	262
Установить мощность модуля	262
Query module power	263
Установить режим связи	263
Режим связи запроса	263
Установить информацию для уведомлений	263
Информация для уведомления о запросе	263
Проект 2 - управляющий светодиод	270
Project 3 – environmental sensor	272
Что нового в Bluetooth 4.1, 4.2 и 5.0?	276
Bluetooth 4.1	276
Bluetooth 4.2	276
Bluetooth 5.0	276
Сетка Bluetooth	277
	277
	277
Глава 19: Классический Bluetooth	278

	278
Bluetooth-радио	280
Топология сети	282
	283
Принципиальные схемы	283
Проект 1 - настройка модулей Bluetooth	289
Тестовая команда	289
Команда сброса	289
Запросить прошивку	289
Восстановить настройки по-умолчанию	289
Адрес модуля запроса	289
Установить / запросить режим модуля	289
Установить / запросить параметры UART	290
Установить / запросить режим подключения	290
Установить / запросить адрес привязки	290
Проект 2 - последовательное соединение, отправка данных	293
Проект 3 - пульт дистанционного управления джойстиком	296
	301

Предисловие

Arduino - это комплексное руководство по максимально эффективному использованию Arduino. Это практическое и серьезное руководство научит вас всем навыкам в области программирования, которые необходимы вам для создания сложных проектов Arduino. Эта книга наполнена реальными проектами, над которыми вы можете попрактиковаться, объединяя все знания, изложенные в книге, и давая вам навыки создания собственного робота на основе примеров из этой книги. В последних двух главах обсуждаются беспроводные технологии и их использование в ваших проектах.

Книга начинается с IDE Arduino и показывая вам, как подключить Arduino к компьютеру и запускать простые проекты на вашем Arduino.

После того, как основы будут изучены, следующие 10 глав книги будут посвящены небольшим проектам, сосредоточенным на определенных компонентах, таких как ЖК-дисплеи, шаговые двигатели или синтезаторы голоса. Каждая из этих глав познакомит вас с задействованной технологией, с тем, как с ее помощью строить, как ее программировать и как ее можно использовать в ваших собственных проектах.

Arduino для всех, кто хочет экспериментировать с платой Arduino и создавать простые проекты.

Ардуино

Вы когда-нибудь смотрели на гаджет и задавались вопросом, как он работает? Вы хотите создать свой крутой и увлекательный проект в области электроники, но не знаете, с чего начать? Ваше решение начать читать эту книгу - отличный первый шаг.

В этой книге мы научим вас всему, что вам нужно для начала работы с Arduino: от прототипирования до настройки среды разработки и программирования Arduino. В этой книге также есть множество примеров проектов, демонстрирующих, как использовать эти знания на реальных примерах. Прежде чем мы перейдем ко всем этим забавным вещам, давайте взглянем на саму Arduino и познакомимся с ней.

В этой главе вы узнаете:

- Что такое платы Arduino
- Как запитать платы Arduino
- Что такое шилды Arduino
- Выводы на платах Arduino
- Общие и совместимые платы Arduino

Arduino - это компания, советы по разработке, сообщество и образ мышления. Как вы скоро узнаете, Arduino - это также название бара в северной Италии. Хотя мы могли бы начать эту книгу, написав несколько глав обо всем, что означает название Arduino, эта книга не об этом. Эта книга научит вас использовать плату Arduino для создания увлекательных и захватывающих проектов. Везде в этой книге, если не указано иное, когда мы говорим об Arduino, мы будем иметь в виду платы для разработки Arduino. Тем не менее, мы считаем, что для того, чтобы по-настоящему понять плату Arduino, вы должны хотя бы иметь базовое представление об ее истории; поэтому мы начнем с краткой истории платы и ее предшественников.

Arduino

В 2003 году Эрнандо Барраган начал работать над проектом под названием Wiring для своей магистерской диссертации в Институте дизайна взаимодействия Ивреа (IDII) в Италии. В то время студенты использовали плату микроконтроллера, которая стоила 100 долларов США, и для этого требовалось дополнительное оборудование и программное обеспечение. Массимо Банци и Кейси Реас, известные своими работами над языком обработки, были руководителями его диссертации. Название было Wiring: Prototyping Physical Interaction Design.



Вы можете просмотреть диссертацию здесь:
http://people.interactionivrea.org/h.barragan/thesis/thesis_low_res.pdf.

Целью диссертации было создание недорогого и простого в использовании инструмента, чтобы люди, не являющиеся инженерами, могли создавать цифровые проекты. Для этого Эрнандо хотел абстрагироваться от сложных деталей электроники, чтобы позволить пользователю сосредоточиться на своем проекте. Это означало, что нужно было просто подключить устройство к компьютеру и иметь простой в использовании интерфейс для его программирования.

Первый прототип использовал микроконтроллер Parallax Javelin Stamp, который использовал подмножество языка программирования Java. Это решение требовало проприетарных инструментов Parallax для компиляции, связывания и загрузки проектов в микроконтроллер; следовательно, он не отвечал требованиям проекта в отношении открытого исходного кода.

Во втором прототипе использовался микроконтроллер 91R40008 на базе ARM Atmel. Hernandoo добился лучших результатов с этим новым микроконтроллером; однако он определил, что микроконтроллер слишком сложен, и его почти невозможно припаять вручную к печатной плате.

В третьем прототипе использовался микроконтроллер Atmel ATmega128 с платой микроконтроллера MAVRIC. Эрнандо добился большого успеха в использовании этого микроконтроллера. Он использовал инструмент Avrdude, написанный Брайаном Дэном, чтобы легко загружать новые программы на доску.



Avrdude используется до сих пор, и его можно найти здесь: <http://www.nongnu.org/avrdude/>.

Микросхема FTDI была выбрана для USB для последовательной связи, потому что у нее были доступные драйверы для платформ Linux, Windows и macOS. Это позволило проекту Wiring быть совместимым со всеми тремя основными платформами.

В 2004 году IDII заказал и оплатил 25 монтажных плат. Эти платы были произведены и доставлены он-лайн. Они включали микроконтроллер ATmega128, FTDI USB для последовательного подключения, встроенный светодиод, и светодиоды последовательного порта RX / TX. Тесты на удобство использования были выполнены с использованием этих плат, и результаты были отличными.

После окончания с отличием в 2004 году Эрнандо вернулся в свою родную Колумбию, чтобы преподавать в Universidad de Los Andes, где он продолжает работать в области электромонтажа. В мае 2005 года Эрнандо заказал 200 монтажных плат и начал сборку первых монтажных плат за пределами IDII. Он продал эти платы примерно за 60 долларов США. К концу 2005 года плата использовалась в разных частях света.

Также в 2005 году была создана первая плата Arduino. Плата Arduino использует недорогой микроконтроллер ATmega128 для снижения стоимости. Команда Arduino разделила Wiringcode и добавила поддержку этой платы.

Первоначальная основная команда Arduino состояла из Массимо Банци, Дэвида Куартиэльеса, Тома Иго, Джанлуки Мартино и Дэвида Меллиса. Эрнандо не был приглашен для участия в этом проекте. Есть несколько рассказов разных людей о том, почему его не пригласили.



Я не знаю из первых рук, какие из этих историй правдивы, а какие - ложны; поэтому для этой книги я оставляю это на известном уровне: Эрнандо не был приглашен для участия в проекте Arduino.

Команда Arduino твердо верила в оборудование и программное обеспечение с открытым исходным кодом. Они считали, что, открыв платформу, гораздо больше людей получают доступ к ней и будут вовлечены в нее. Другой причиной для открытия платформы было то, что IDII израсходовал свое финансирование и собирался закрыть проект. Открыв исходный код платформы, они знали, что она выживет и не будет использоваться другими пользователями в своих коммерческих целях.

Первоначально команда определила цену платы в 30 долларов США. Они полагали, что это сделает его легко доступным как для студентов, так и для отдельных лиц. Они также решили сделать плату синей, что отличалось от большинства других плат того времени, которые были зелеными. Еще одно дизайнерское решение, которое помогло повысить популярность платы, заключалось в предоставлении ей большого количества входных и выходных контактов. Большинство плат в то время ограничивали количество операций ввода-вывода для снижения затрат.

Изначально команда заказала 300 печатных плат для проведения теста на удобство использования. Они раздали эти платы студентам IDII с тремя простыми инструкциями: найдите инструкции по сборке в Интернете, соберите свою плату и используйте ее для создания чего-нибудь. У них был большой успех с этим тестом, потому что студенты смогли собрать платы и создать с их помощью множество проектов.

Вскоре после этого теста люди начали слышать об этой плате и захотели такую для себя. Проект начал набирать обороты; однако имя по-прежнему отсутствовало. Обсуждая название, команда пила в местном баре, который часто посещал Массимо Банци. Бар назывался Bar Di Re Arduino, а новая плата стала известна как Arduino.

?

В основе Arduino лежит микроконтроллер. Микроконтроллер - это автономная однокристалльная интегральная схема, которая содержит ЦП, постоянную память, оперативную память и различные шины ввода-вывода. Большинство плат Arduino используют 8-битный микроконтроллер AVR Atmel.

Arduino UNO R3, основная плата, используемая в этой книге, использует микросхему ATmega328. Этот чип представляет собой 8-битный микроконтроллер на основе RISC, который имеет 32 КБ флэш-памяти с возможностью чтения и записи, 1 Кбайт EEPROM, 2 Кбайта SRAM, 23 линии ввода-вывода общего назначения и 32 регистра общего назначения. Не беспокойтесь, если вы не понимаете всех этих спецификаций, потому что мы будем взаимодействовать с микроконтроллером, используя интерфейс, который предоставляет нам плата Arduino. Эти спецификации полезно знать, когда вы начинаете разрабатывать более сложные приложения.

Все аппаратное и программное обеспечение, составляющее платформу Arduino, распространяется как с открытым исходным кодом и лицензируется на условиях GNU Lesser General Public License (LGPL) или GNU General Public License (GPL). Это позволяет производить и распространять платы Arduino кем угодно и привело к появлению множества универсальных недорогих плат, совместимых с Arduino.

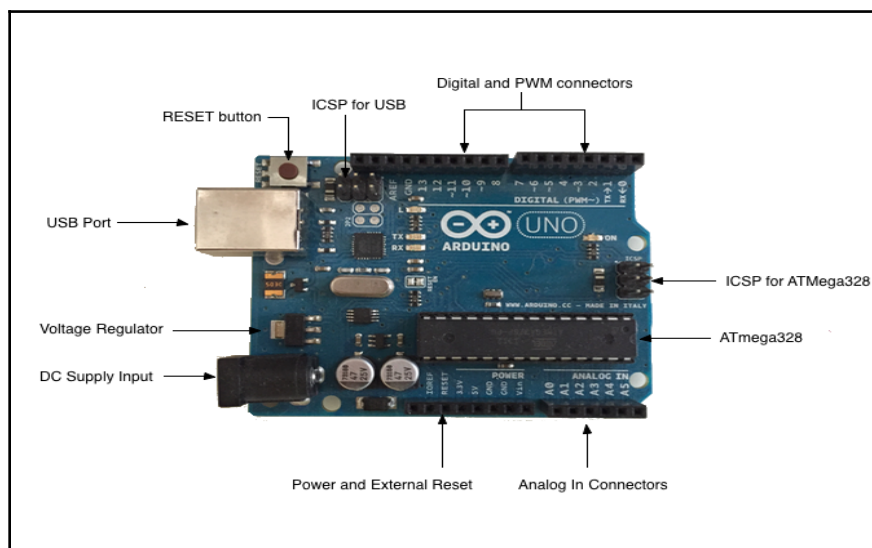


Вы можете найти дополнительную информацию о лицензии и платах Arduino на веб-сайте Arduino здесь: <https://www.arduino.cc>.

Arduino UNO R3

Arduino - это аппаратная и программная платформа с открытым исходным кодом, которая невероятно проста в использовании. Вы можете посмотреть и скачать код из любого репозитория Arduino на GitHub здесь:

<https://github.com/arduino>. Эта платформа захватила воображение энтузиастов электроники и сообщества производителей во всем мире. Это позволяет людям недорого поэкспериментировать с электронными прототипами и увидеть, как их проекты воплощаются в жизнь. Эти проекты могут варьироваться от простого мигания светодиода или регистрации температуры до управления 3D-принтерами или создания роботов. Хотя существует множество моделей Arduino, в этой книге мы в первую очередь будем использовать очень популярную плату Arduino UNO R3. На следующей фотографии показана компоновка платы ArduinoUno с обозначенными основными разъемами:



В этой книги всякий раз, когда мы ссылаемся на плату Arduino Uno, мы имеем в виду плату Arduino Uno R3, изображенную на предыдущей фотографии.

Как мы видим, в современной Arduino Uno по-прежнему используется синий цвет, который первоначальные дизайнеры Arduino выбрали, чтобы помочь своим платам выделиться. Ниже приводится список основных компонентов Arduino Uno:

- **Разъем внешнего питания:** может использоваться с адаптером питания переменного тока в или аккумулятором; может иметь максимальное входное напряжение 20 вольт; однако не рекомендуется использовать напряжение более 12 В.
- **Стабилизатор напряжения:** создает напряжение 5В, поступающее на плату.
- **Порт USB:** порт USB можно использовать для питания и программирования платы.
- **Кнопка RESET:** при нажатии этой кнопки выполняется сброс платы.
- **ICSP для USB:** внутрисхемные выводы последовательного программирования используются для загрузки скетчей
- **ICSP для ATmega328:** выводы последовательного программирования в используются для прошивки микроконтроллера ATmega.
- **Цифровые выводы:** : эти выводы, обозначенные от 0 до 13, могут использоваться как цифровые входные или выходные выводы. Выводы, помеченные тильдой (~), также могут использоваться для вывода ШИМ.
- **Разъемы аналогового входа:** ... с маркировкой от A0 до A5 используются для считывания выходного сигнала с аналоговых датчиков.
- **Выводы питания:** : эти выводы данного разъема обеспечивают питание внешних устройств и датчиков от Arduino. Arduino также может получать питание через эти выводы. Также имеется вывод сброса, который можно использовать для внешнего сброса Arduino.
- **ATmega328:** микроконтроллер для платы Arduino Uno.

Arduino

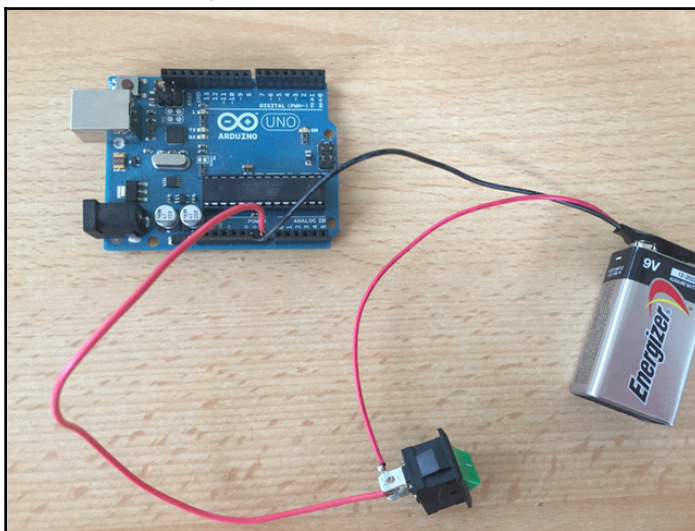
Arduino можно запитать одним из трех способов: через контакты Vin / GND, через разъем внешнего питания или порт USB.

VIN / GND

Arduino

Выводы Vin и GND в разъеме питания и внешнего сброса могут использоваться для питания Arduino от внешней батареи. Такое питание Arduino в основном используется, когда мы хотим последовательно подключить батарею с переключателем для включения и выключения питания Arduino.

Следующая фотография иллюстрирует это:



Не рекомендуется включать Arduino таким образом, если мы не ищем самый дорогой и недолговечный способ питания Arduino. Мы могли бы использовать шесть батареек AA последовательно, что позволит обеспечивать такое же напряжение, как и батарея 9 В на предыдущей фотографии, но дает напримерно в четыре раза большую емкость. По-прежнему не рекомендуется использовать такой способ питания Arduino, так как это будет довольно дорого.

Если нет особой необходимости использовать батарею для питания Arduino, я бы не стал их использовать.

Использование входа источника постоянного тока для питания Arduino

Входной разъем источника постоянного тока может использоваться с адаптером питания переменного тока в постоянный или аккумулятором для питания Arduino. Разъем имеет центрально-положительный штекер диаметром 2,1 мм. Пока Arduino работает от 5 вольт, можно использовать максимум 20 вольт; однако, как указывалось ранее, не рекомендуется использовать напряжение более 12 В.

Мы можем использовать регулируемый адаптер питания переменного тока в постоянный, подобный показанному на следующей фотографии, для питания Arduino с помощью входного разъема источника постоянного тока:

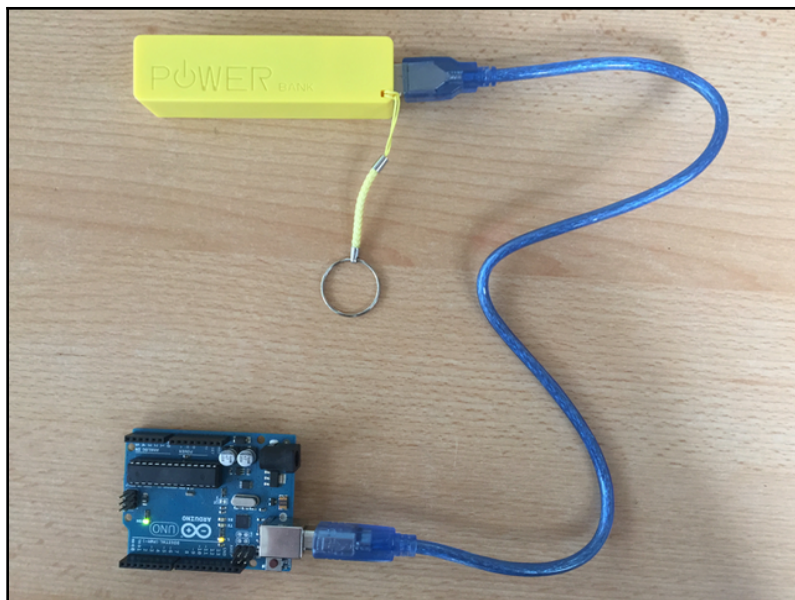


С помощью этого адаптера вы можете отрегулировать выходную мощность до желаемого напряжения. Подобные источники питания можно найти в Интернете или в большинстве магазинов, торгующих электроникой.

USB

Arduino

Обычно я использую USB-разъем для питания Arduino. Это, безусловно, самый простой и безопасный способ запитать Arduino и наименее затратный. Вы можете запитать Arduino напрямую от USB-порта вашего компьютера или от USB-аккумулятора, как показано на следующей фотографии.



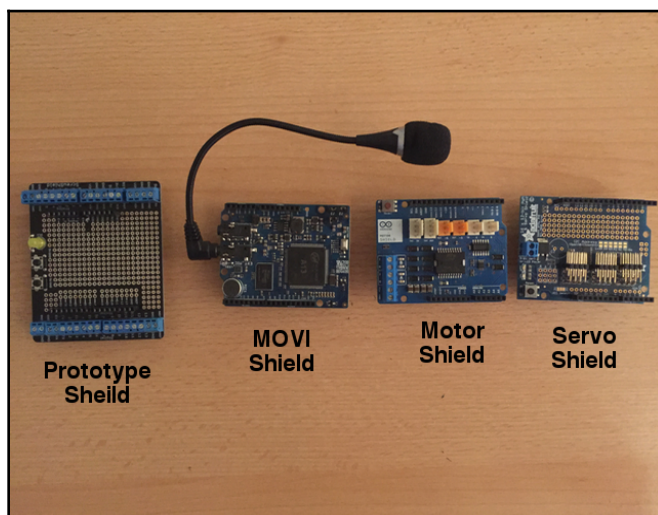
Это очень доступный и простой способ запитать Arduino. Его также можно использовать для роботизированных или аналогичных проектов, которым требуется мобильность для передвижения; однако нам нужно быть осторожными, когда мы подключаем шилды или другие аксессуары к Arduino, чтобы USB-разъем мог потреблять достаточно энергии. В качестве примера ниже в этой книге мы рассмотрим шилд синтезатора речи и распознавания голоса MOVI, который потребляет слишком много энергии для питания Arduino от USB-разъема при подключенном шилде.

Теперь, когда мы упомянули шилды Arduino, давайте посмотрим, что они собой представляют, и посмотрим, какие функциональные возможности они могут предоставить.

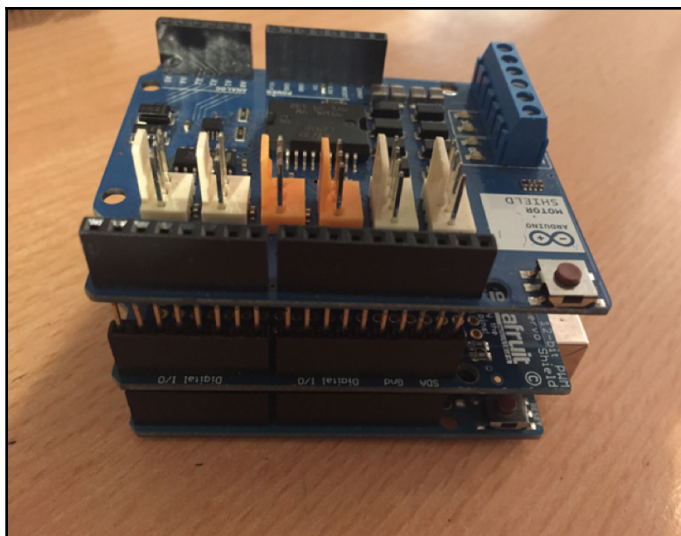
Arduino

Шилд Arduino - это модульная печатная плата, которая подключается непосредственно к контактным разъемам платы Arduino. Эти шилды добавляют дополнительную функциональность плате Arduino. Если мы хотим подключиться к Интернету, выполнить распознавание речи, управлять двигателями постоянного тока или добавить другие функции к Arduino, вероятно, есть шилд, который может нам помочь. Хотя мы не обязаны использовать шилды, они очень упрощают добавление дополнительных функций к нашим платам Arduino.

На следующей фотографии показаны примеры нескольких шилдов. Мы будем использовать шилды в некоторых из наших примеров проектов позже в этой книге:



Шилд устанавливается поверх Arduino, вставляясь непосредственно в разъемы. Мы также можем ставить один шилд поверх другого, если они не используют одни и те же ресурсы. Вот как выглядит Arduino с двумя прикрепленными шилдами:



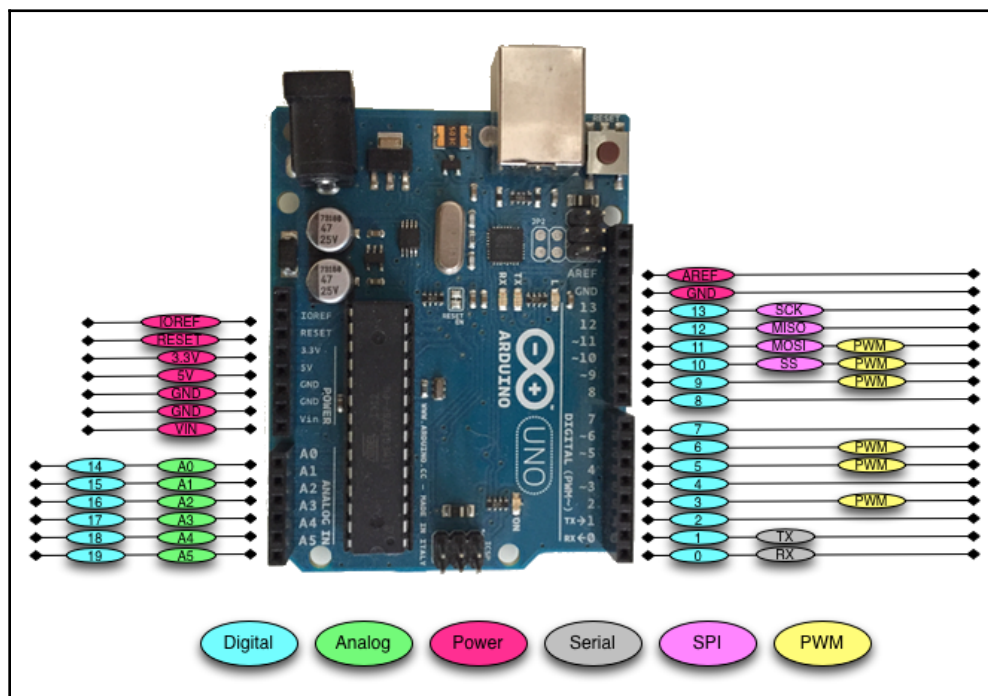
Шилд Arduino позволяет невероятно легко добавить функциональность к Arduino Uno. У большинства шилдов также обычно есть отличная документация, что также упрощает их программирование. Недостатком шилдов является то, что они обычно стоят дороже, чем покупка компонентов и их подключение к Arduino с помощью макета.

Некоторые шилды, такие как шилд синтеза речи и распознавания голоса MOVI и шилд радиомодуля Sparkfun Xbee, добавляют функциональность, которую нельзя просто добавить как отдельный компонент. Для такой функциональности потребуется шилд или внешняя печатная плата.

Давайте подробнее рассмотрим разъемы выводов для Arduino Uno R3.

Arduino

Всего в разъемах Arduino Uno 31 вывод. Большинство этих выводов можно настроить для выполнения различных функций. На следующей схеме показано, для чего можно использовать различные выводы:



Давайте посмотрим, что делают разные выводы.

Цифровые выводы на Arduino - это те, которые чаще всего используются при подключении внешних датчиков. Эти выводы могут быть настроены как для ввода, так и для вывода и по умолчанию находятся в состоянии ввода; поэтому, когда мы используем вывод для ввода, нам не нужно явно объявлять их как выводы ввода; тем не менее, это хорошая практика, потому что это поможет кому-то, читающему наш код, понять, для чего используется вывод.

Цифровые выводы будут иметь одно из двух значений: HIGH (1), равное 5V, или LOW (0), которое равно 0V.

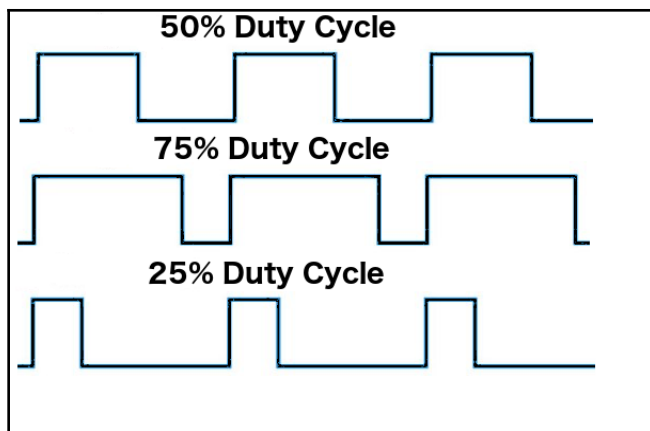
Как только мы начнем программировать Arduino, мы увидим, как использовать эти выводы в режиме ввода или вывода.

Arduino Uno содержит встроенный аналого-цифровой (АЦП) преобразователь с шестью каналами, который дает нам шесть аналоговых входных контактов. АЦП преобразует аналоговый сигнал в цифровое значение. В то время как цифровые выводы имеют два значения В 1 или 0, выводы аналогового входа имеют значения от 0 до 1023 относительно эталонного значения Arduino. Arduino Uno имеет эталонное значение 5 В.

Выводы аналогового разъема используются для считывания показаний аналоговых датчиков, таких как датчики температуры. Шесть аналоговых выводов также можно настроить как цифровые выводы, если в нашем проекте закончились цифровые выводы.

Если выводы аналогового входа предназначены для считывания аналоговых датчиков (вход), выводы ШИМ предназначены для вывода. ШИМ - это метод получения аналоговых результатов с цифровым выходом.

Поскольку цифровой выход имеет импульсный характер, для получения аналогового выхода цифровой выход быстро переключается между HIGH(1) и LOW(0). Процент времени, в течение которого сигнал имеет высокий уровень, называется рабочим циклом. Следующая диаграмма иллюстрирует эту концепцию.



У нас есть возможность установить частоту, с которой сигнал может переключаться между HIGH и LOW уровнями. Эта частота измеряется в Герцах и устанавливает, сколько раз сигнал может переключаться в секунду. Например, если мы установим частоту 500 Гц, это будет означать, что сигнал может переключаться 500 раз в секунду.

Мы будем использовать выводы ШИМ для нескольких примеров в этой книге и рассмотрим их подробнее, когда узнаем, как программировать Arduino.

Arduino имеет несколько выводов питания. Вот они:

- **Vin**: этот вывод используется, когда мы запитываем плату Arduino от внешнего источника питания. Этот вывод используется в разделе Использование выводов VIN / GND для питания Arduino в этой главе.
- **GND**: это выводы заземления.
- **+5V**: это выход 5V, который используется для питания большинства датчиков.
- **+3,3 V**: это выход 3,3 V, который может использоваться для питания датчиков питаемых от 3,3 V.
- **Reset**: этот вывод можно использовать для сброса платы Arduino от внешнего источника.
- **ioref**: Это опорное напряжение для платы. Для Arduino это будет 5 V.

Выводы последовательной связи

Эти выводы можно использовать для последовательной связи. RX (цифровой вывод 0) используется для приема, в то время как TX (цифровой вывод 1) используется для передачи. Эти выводы подключаются напрямую к микросхеме последовательного интерфейса USB-to-TTL. Одно замечание: вы не должны подключать эти выводы напрямую к последовательному порту RS-232, потому что вы повредите свою плату.

SPI

Выводы **Serial Peripheral Interface** (SPI) используются для протокола синхронных последовательных данных, который используется микроконтроллерами для связи с периферийными устройствами. Этот протокол всегда имеет одно ведущее устройство с одним или несколькими ведомыми устройствами. выводы бывают:

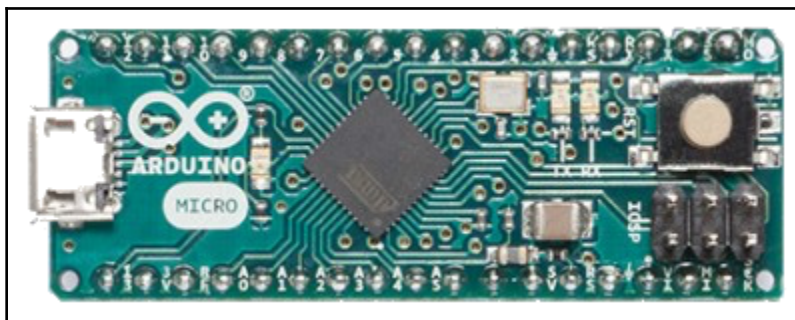
- **MISO**: вывод **Master in Slave out** используется для отправки данных с ведомого на ведущее устройство.
- **MOSI**: **Master out Slave in** используется для отправки данных от ведущего к ведомому устройству.
- **SCK**: **serial clock** синхронизация передачи данных и генерируются мастером.
- **SS**: Пин выбора ведомого сообщает ведомому, что нужно перейти в активное состояние или перейти в режим сна. Это используется для выбора ведомого устройства, которое должно получать передачу от ведущего.

Теперь, когда мы быстро рассмотрели контакты на Arduino Uno R3, давайте посмотрим на некоторые из различных плат Arduino.

Arduino

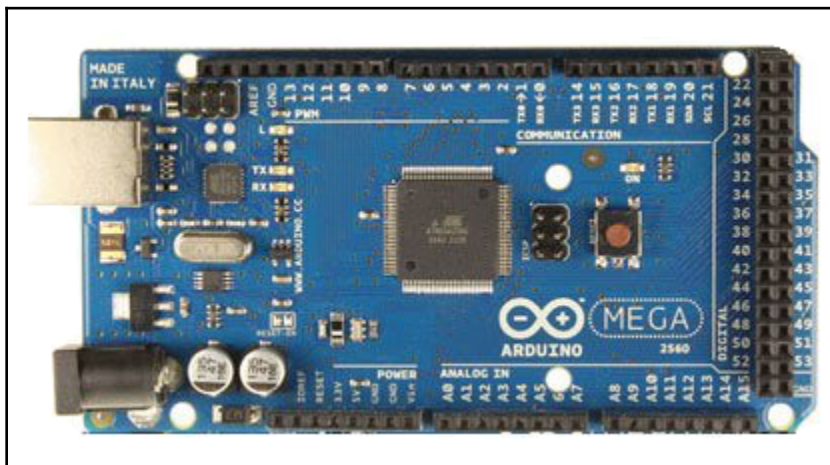
Существует ряд различных официальных плат и модулей Arduino, которые можно использовать для различных целей. Чтобы увидеть все различные платы, вы можете перейти на страницу продукта Arduino. (<https://www.arduino.cc/en/Main/Products>), где перечислены все официальные платы Arduino. Хотя Arduino Uno R3 является самой популярной платой Arduino в сообществе производителей, ниже перечислены некоторые другие популярные платы:

Arduino Micro



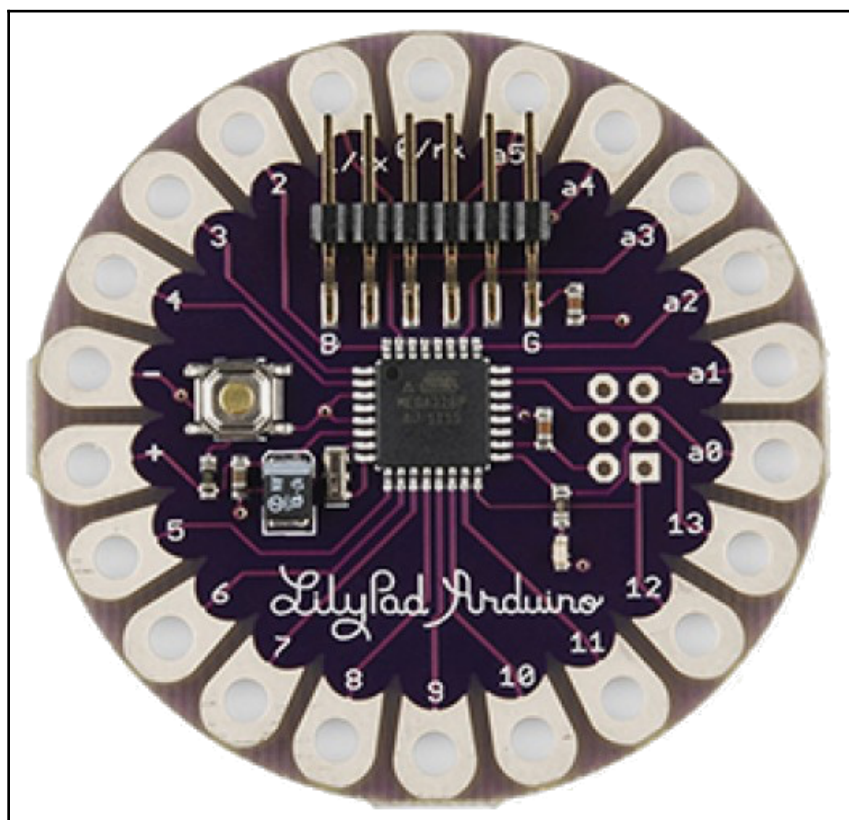
Arduino Micro - самая маленькая плата в семействе Arduino. Он основан на микроконтроллере ATmega32U4. Эта плата имеет 20 цифровых входов / выходов, из которых 7 можно использовать для вывода ШИМ, а 12 можно использовать как аналоговый вход. Micro и Nano (которые мы увидим немного позже) можно использовать для проекта, для которого Arduino Uno может быть слишком велик.

Arduino Mega 2560



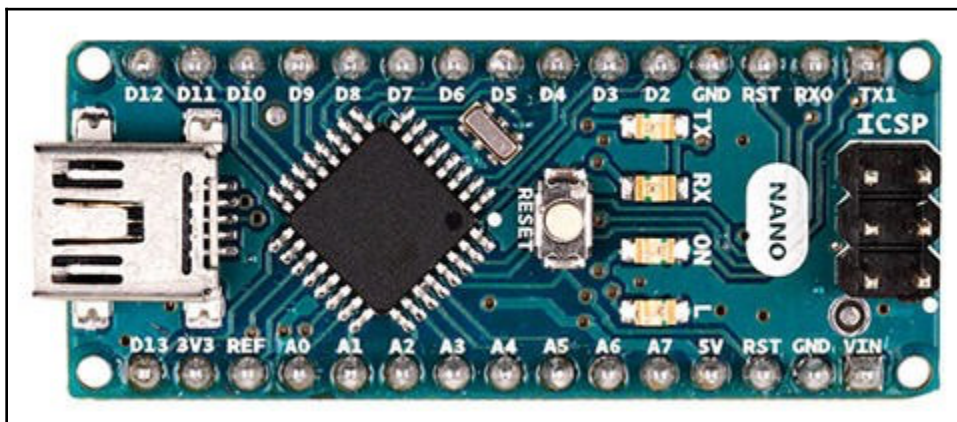
Arduino Mega 2560 предназначена для самых сложных проектов. Он имеет 53 цифровых входа / выхода, 16 аналоговых входных контактов и 15 выходных контактов ШИМ. Он также имеет 4 последовательных порта UART для последовательного подключения. Если вы хотите создать сложный проект, например, робота, Mega - это плата, которой более всего подойдет для этого.

Lilypad



Arduino Lilypad разработан для носимых проектов. Его можно вшивать в ткань и использовать источники питания и датчики, которые также вшиваются в ткань. Lilypad основан на ATmega168V или ATmega328V (версии с низким энергопотреблением). Эта плата имеет 16 цифровых входов / выходов, 6 аналоговых входов и 6 выходов ШИМ.

Arduino Nano



Между Nano и Micro много общего. Micro был выпущен в 2012 году, а Nano - в 2008 году. Nano имеет 14 цифровых контактов ввода / вывода, 8 аналоговых входных контактов и 6 выходных контактов PWM. С этими характеристиками вы можете подумать, что вам следует использовать плату Micro вместо Nano, однако, если вы посмотрите на большинство онлайн-магазинов, таких как Amazon или eBay, вы можете найти Nano примерно за половину цены Micro.

Вы также обнаружите, что Nano легче получить, чем Micro, потому что существует множество обычных плат Nano. Мы также будем использовать Nano для некоторых проектов в этой книге.

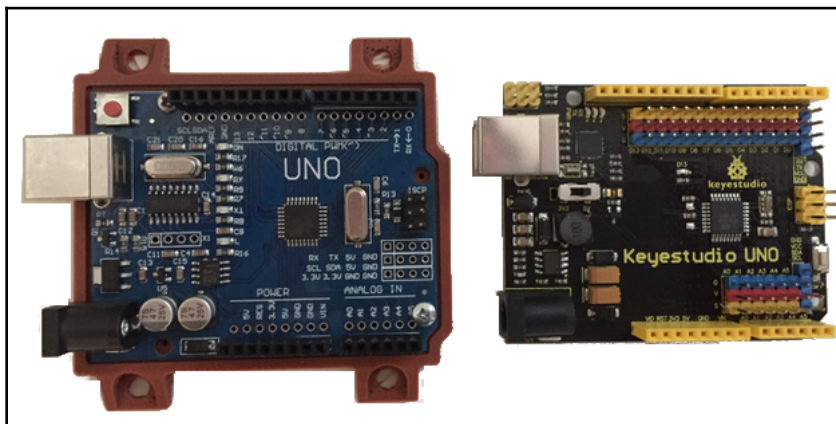
В начале этой книги мы отметили, что Arduino - это аппаратная и программная платформа с открытым исходным кодом. Все файлы оригинального дизайна оборудования выпущены под лицензией Creative Commons Attribution.

Лицензия Share-Alike. Эта лицензия позволяет использовать как личные, так и коммерческие производные от всех плат Arduino, если они используют Arduino и выпускают свой дизайн под той же лицензией. Это привело к появлению многих универсальных плат по более низкой цене.

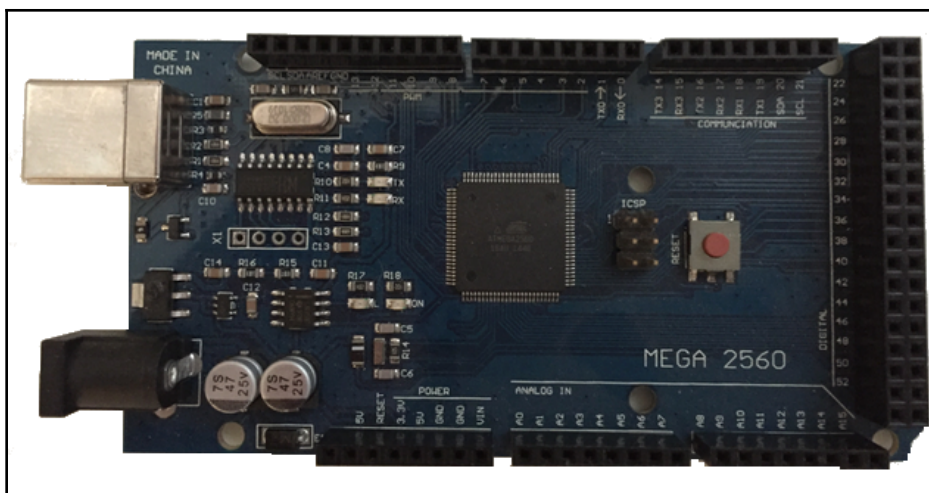
Если вы ищете плату Arduino на большинстве сайтов онлайн-магазинов, большинство плат на самом деле не будут настоящими платами Arduino. Если вы посмотрите на плату Arduino Uno на следующей фотографии, вы заметите знак бесконечности с плюсом (+) и минусом (-) в нем. Это официальный логотип Arduino, и любая плата с этим логотипом является настоящей платой Arduino.



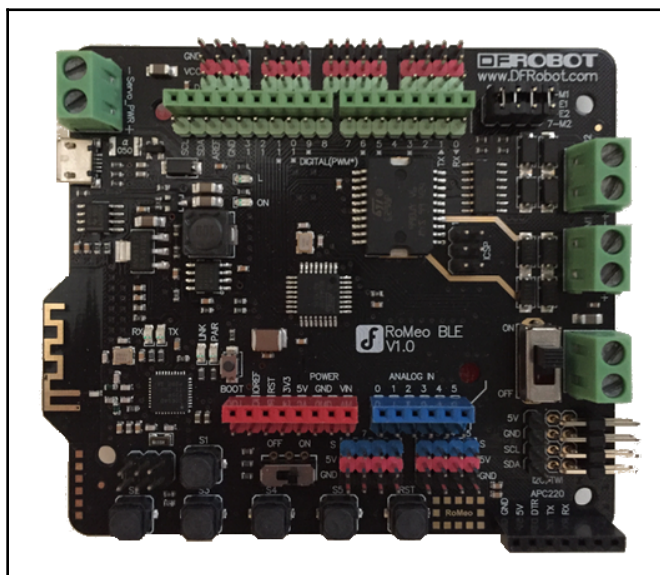
В этой книге мы будем использовать в основном стандартные платы Arduino, поскольку они дешевле и обычно их легче достать. На следующей фотографии показано, как выглядят некоторые стандартные платы Arduino. На первой фотографии показаны две стандартные платы Arduino Uno:



На следующей фотографии показана стандартная плата Arduino Mega 2560:



Вы заметите, что эти общие платы не содержат логотипа Arduino; тем не менее, они все еще содержат название официальной платы. Некоторые производители выбрали эталонную конструкцию Arduino и добавили в свои платы дополнительные функции. Плата на следующей фотографии является примером этого:



Плата DFRobot RoMeo BLE - это Arduino-совместимая плата управления роботом с Bluetooth LE 4.0. Эта плата использует плату Arduino Uno и добавляет ряд дополнительных функций, таких как встроенный Bluetooth и встроенный двусторонний драйвер двигателя постоянного тока.

Независимо от того, какой у вас проект, вы, вероятно, сможете найти либо настоящую плату Arduino, либо общую / совместимую, которая будет соответствовать вашим потребностям.

В этой главе мы кратко рассказали об истории Arduino, которая включала ее развитие от магистерской диссертации до полноценного коммерческого проекта. Мы также совершили экскурсию по разным платам Arduino и показали различные способы питания плат Arduino и дали краткое объяснение различных типов контактов.

В следующей главе мы дадим вам краткое введение в электронику и часто используемые компоненты.

2

В прототипировании воплощаются все наши идеи. Когда я начал работать с микроконтроллерами, мне очень хотелось спроектировать и построить весь свой проект и посмотреть, как он волшебным образом работает. Я понял, что все, что я делал, это подавлял себя, и быстро научился разбивать большие проекты на гораздо более мелкие. Тогда я мог бы создать прототипы для каждого из этих небольших проектов, чтобы убедиться, что они работают, прежде чем включать их в более крупный проект.

В этой главе мы узнаем:

- Где настроить рабочую зону
- Все о макетных платах
- Какие бывают провода Dupont (перемычки)
- Как создать прототип проекта

Когда вы впервые начинаете создавать проекты на основе Arduino, важно организовать рабочее место, где создавать свои проекты и хранить детали и прототипы. Когда я впервые начал использовать Arduino, я использовал стол в гостиной, где я смотрел бейсбол по телевизору, в качестве рабочего места, пока не понял, что гостиная - не идеальное место для работы, потому что все необходимые инструменты и компоненты проектов быстро заняли все помещение. В этом разделе мы рассмотрим что делает рабочее место удобным местом для своих занятий.

Первое, что нам нужно учитывать при выборе рабочего места, - это статическое электричество. Хотя статическое электричество не является такой большой проблемой, как многие люди представляют, это то, что мы должны учитывать при создании рабочего места. Я лично не потерял ни одного электронного компонента из-за статического электричества за последние пятнадцать-двадцать лет; Тем не менее, я также не ношу пушистые свитера и не тру ноги о ворсистый ковер, пока работаю с электронными компонентами. Как правило, мы должны стараться избегать участков, подверженных статическому электричеству, таких как участки с толстым ковром.

Несколько советов, как избежать статического электричества при работе с Arduino или другими электронными компонентами:

- Прикоснитесь к чему-нибудь металлическому, чтобы снять статическое электричество, накопившееся до прикосновения к электронному компоненту.
- По возможности избегайте ковров на рабочем месте и обуви на пластиковой подошве.
- Будьте осторожны с материалом вашей одежды. Шерстяные свитера (носки) могут вызвать накопление статического электричества, поэтому используйте вместо этого хлопок.
- Избегайте гладить пушистых домашних животных во время работы.
- Если ваше рабочее место сухое, используйте увлажнитель, чтобы увлажнить воздух.

Еще одна проблема при выборе рабочей зоны - наличие достаточно большого стола или скамейки, чтобы вместить ваши проекты, и, желательно, такого стола, на котором вы можете хранить проекты в течение длительных периодов времени. Когда я использовал стол в гостиной в качестве рабочей зоны, я быстро понял, насколько неудобно убирать все это каждую ночь.

Вы также должны убедиться, что у вас много света в рабочей зоне. Даже если ваша рабочая зона имеет достаточное освещение, я бы порекомендовал приобрести настольный светильник, который можно прикрепить к краю стола, когда вам нужно дополнительное освещение. Поверьте, будут моменты, когда вам понадобится дополнительное освещение, чтобы рассмотреть мелкий шрифт на электронных компонентах.

Последнее, что вам нужно организовать на рабочем месте, - это место для хранения компонентов. Чем больше прототипов вы сделаете, тем больше деталей вы приобретете, и вам понадобится место, чтобы их хранить. Когда я впервые начал работать с микроконтроллерами, я хранил большую часть своих небольших детали в паре небольших пластиковых ящиков для инструментов. Вначале это не напрягало, но со временем я купил несколько контейнеров для мелких деталей и более крупные пластиковые ящики для хранения крупных компонентов.

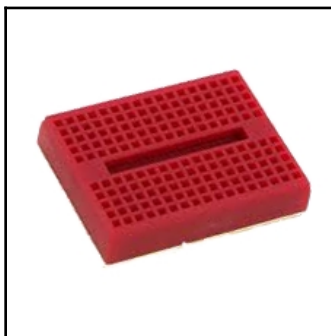
Прежде чем вы начнете работать с Arduino, стоит потратить время на создание надлежащего рабочего места. Я не делал этого вначале, и это значительно усложнило создание моих прототипов. Теперь давайте рассмотрим два наиболее важных элемента после микроконтроллера, которые вы будете использовать при создании прототипа. Это два элемента: макетная плата и кабели Dupont.

Использование беспаячной макетной платы

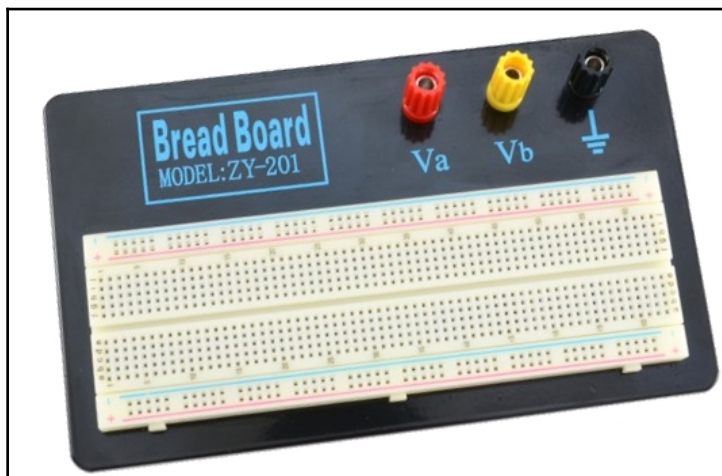
Рекомендуется избегать подключения светодиодов, резисторов и других электронных компонентов непосредственно к Arduino, потому что вы можете легко повредить разъемы на Arduino, и схема быстро превратится в неорганизованный беспорядок. При создании прототипа гораздо проще соединить компоненты вместе с помощью беспаячной макетной платы.

Макетная плата без пайки позволяет нам соединять электронные компоненты вместе без пайки. Хороший беспаячный макет MB-102 можно купить менее чем за десять долларов США. Их также можно использовать повторно, что делает их идеальными для создания прототипов и экспериментов со схемотехникой.

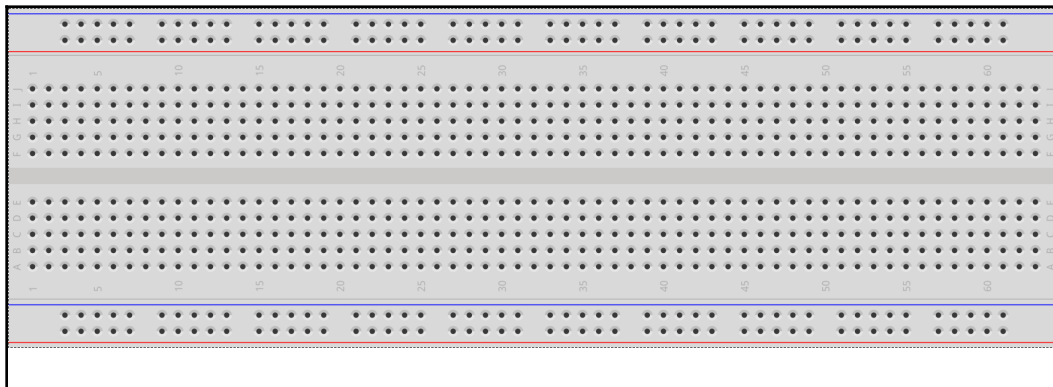
Я бы порекомендовал не слишком экономить на ваших макетных платах, потому что вы будете их очень часто использовать. Вы можете найти очень дешевые и маленькие макеты, подобные изображенному на следующей фотографии; однако гораздо сложнее организовать схемотехнику с помощью этих меньших макетов. Я использую их для быстрого и небольшого прототипирования, но я бы не рекомендовал использовать их в качестве основных макетов.



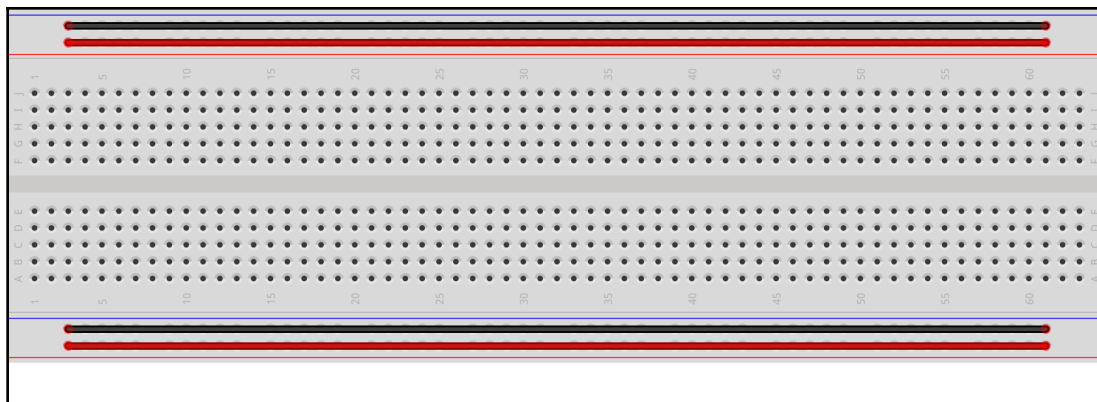
Для прототипирования с Arduino также не обязательно покупать макеты со встроенными разъемами питания, как показано на следующей фотографии:



Я бы порекомендовал приобрести макет MB 102, подобный изображенному на следующей фотографии:

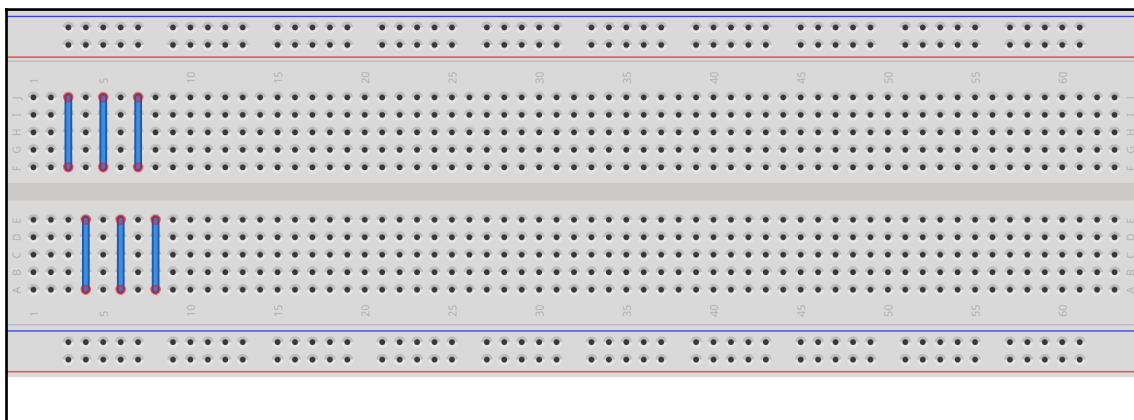


Макетная плата MB-102 содержит две положительные и две отрицательные клеммы питания, что позволяет легко подключить выводы питания и заземления от Arduino к электронным компонентам, необходимым для прототипа. На следующей фотографии мы выделили четыре шины питания, где шины, выделенные красным, являются положительными, а направляющие, выделенные черным, - отрицательными. Положительные шины обычно помечаются на макетной плате знаком +, а отрицательные шины знаком -:



Шины питания проходят горизонтально через макетную плату, где все разъемы вдоль горизонтальной шины обычно соединяются вместе. Это означает, например, что если мы подключим один из выводов напряжения на Arduino к любому из выводов вдоль положительной шины, то все разъемы вдоль этой шины должны быть подключены к выводу напряжения на Arduino.

Штыри посередине доски соединены между собой вертикально; однако соединение не пересекает центр макета. На следующей фотографии показано, как соединяются контакты в середине платы:

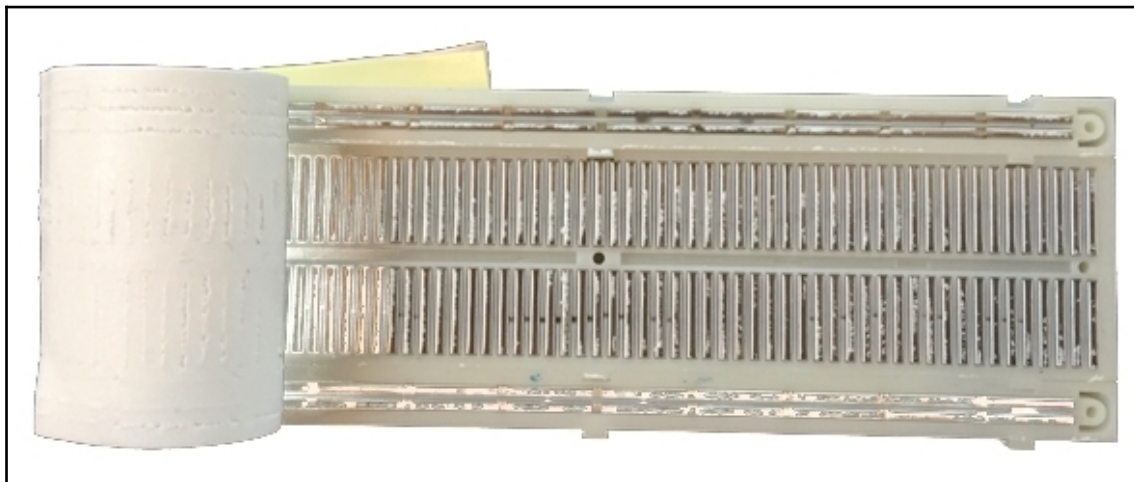


Хотя на предыдущей фотографии показано только соединение шести вертикальных рядов, все вертикальные ряды соединены таким же образом. Следует обратить внимание на то, как мы описали макетную плату МВ-102, как сконфигурировано большинство из них; однако не все из них построены таким образом.

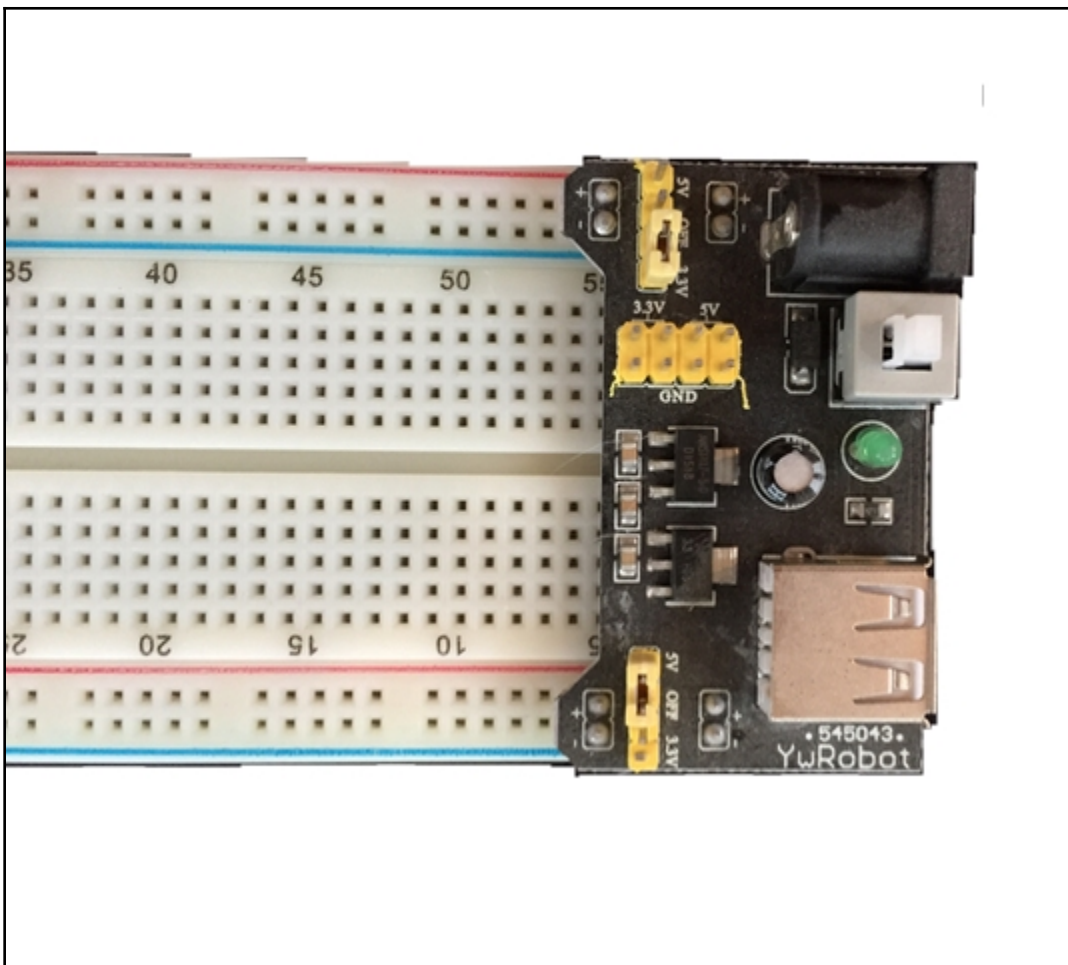


Всегда читайте спецификации при покупке компонентов, таких как макетная плата, чтобы убедиться, что вы знаете, есть ли что-то отличное от того, который вы покупаете. Это сэкономит вам время на устранение неполадок.

Если мы снимем заднюю часть макета, мы сможем лучше увидеть, как соединены контакты. На следующей фотографии показано, как будет выглядеть задняя часть макета, если снять прокладку:



При работе с Arduino мы обычно можем подавать питание и заземление непосредственно от Arduino к шинам питания на макетной плате; однако бывают случаи, когда нам может потребоваться внешнее питание. На этот раз они действительно делают внешние адаптеры питания, которые могут подключаться непосредственно к макетной плате и позволяют нам использовать USB, адаптеры переменного тока и другие источники питания для питания проектов. На следующей фотографии показан один такой адаптер, а также то, как мы могли бы подключить эти адаптеры питания к макетной плате:



Теперь вопрос в том, как соединить компоненты вместе с помощью макета, и ответ - провода Dupont (перемычки).

() Dupont

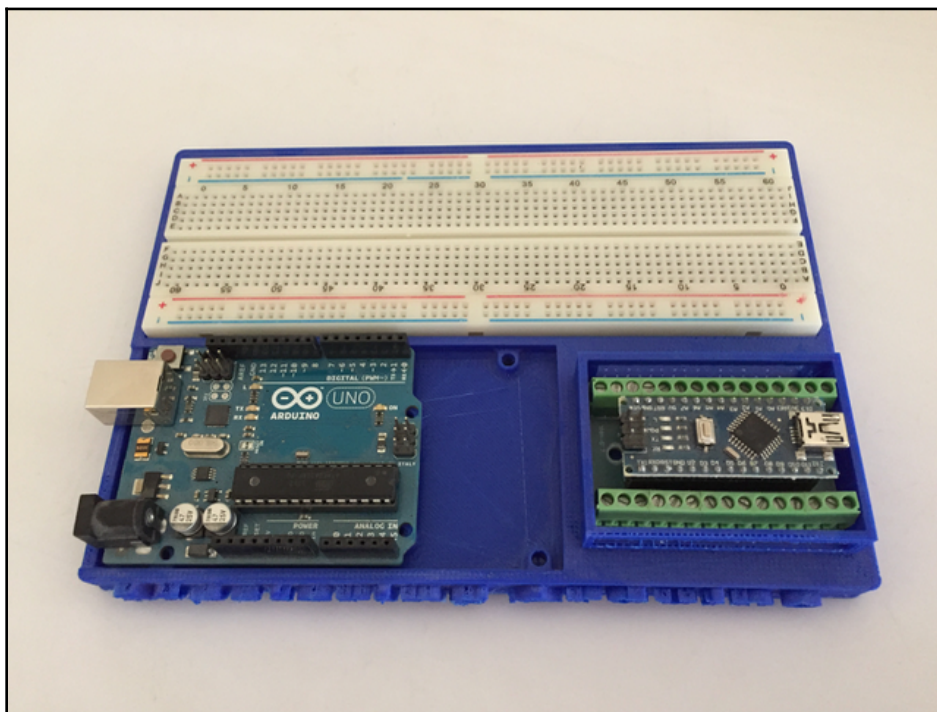
Провода Dupont, также известные как перемычки, используются для соединения компонентов вместе при использовании макетной платы без пайки. Эти кабели бывают трех типов: мужчина-мужчина, мужчина-женщина и женщина-мама. При использовании этих перемычек с макетной платой без пайки и с Arduino мы обычно используем кабели с разъемами «папа» на обоих концах; однако есть некоторые компоненты, на которых уже есть штекерные разъемы, поэтому стоит также иметь несколько штыревых и женских кабелей. На следующей фотографии показано, как выглядят штекерные разъемы на концах перемычек Dupont:



Готовые перемычки Dupont довольно дешевы, но если вы хотите иметь перемычки разной длины, вы можете сделать их самостоятельно, купив щипцы для обжима и несколько концов. Эти перемычки не так уж и сложно сделать, но я бы рекомендовал начать с покупки готовых.

Прототипы используются для подтверждения спецификаций рабочей концепции или процесса. Прототип, как мы его называем в этой книге, - это модель для проверки концепции или процесса. Для простых концепций или процессов мы можем создать прототип для всей системы, но для моих сложных систем мы захотим разбить систему на отдельные компоненты и создать прототип для каждого компонента.

Прототипы с Arduino обычно состоят из одного микроконтроллера Arduino с одной макетной платой, используемой для подключения компонентов к Arduino. Для этих прототипов я использую держатель, который я спроектировал и распечатал на своем 3D-принтере. На следующей фотографии показан держатель с Arduino Uno, Nano и макетной платой:



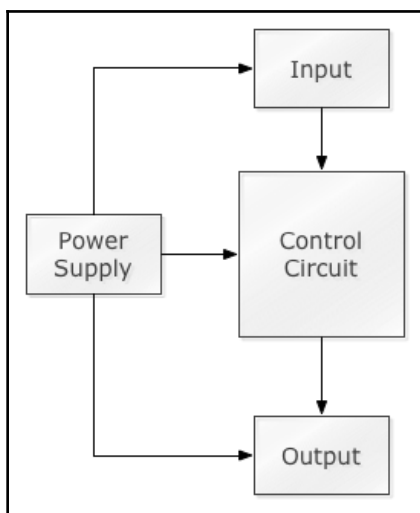
Держатель предназначен для размещения макета MB-102, Arduino Nano, а также Arduino Uno или Mega. Если у вас есть доступ к 3D-принтеру, я включил файл STL для этого держателя в загружаемый код для этой книги, чтобы вы могли распечатать его для себя. Такие держатели особенно полезны, когда нам нужно переместить прототип, потому что он держит все вместе и организовано.

Прежде чем мы сможем приступить к созданию прототипа, нам нужно иметь представление о том, что мы хотим построить.

Это может быть что-то простое, например мигание светодиода, или сложное - создание автономного робота. Идея - это всегда первый шаг при запуске проекта.

Второй шаг - разбить идею на различные строительные блоки, которые обсуждались в главе 2 «Базовая электроника». После того, как мы разбили проект на различные блоки, мы должны были схематизировать схемы, которые необходимо прототипировать. После того, как схемы будут составлены, мы можем приступить к созданию прототипов. Давайте рассмотрим эти шаги немного подробнее, начав с четырех строительных блоков электронной схемы.

В главе 2 «Базовая электроника» мы обсудили четыре базовых блока электронного проекта. На следующем рисунке показаны эти четыре блока:



Очень легко разбить простой прототип, например схему включения и выключения светодиода, на отдельные блоки, но для более сложных проектов это становится труднее из-за различных компонентов.

Разделив компоненты на разные блоки, становится легче увидеть, как разбить более крупный проект на отдельные прототипы. Например, если бы мы хотели создать автономного робота, мы могли бы увидеть, что один из входов - это звуковой датчик, который будет обнаруживать препятствия перед роботом, а выходом, на который будет влиять вход из датчика, будут двигатели, которые переместят робота. В этом примере мы, вероятно, захотим изменить направление двигателей, если датчики обнаружат препятствие впереди. Затем мы могли бы создать прототип, чтобы проверить, как работает эта конкретная система.

На этом этапе мы определяем, что такое входы и выходы, и какие выходы запускаются на основе этих входов. Здесь же мы определяем большую часть логики проекта, чтобы мы могли настроить прототипы.

Когда вы впервые начнете создавать эти проекты, вам нужно будет создать диаграмму для каждого проекта. Как только вы наберетесь опыта, для большинства небольших и средних проектов вам не нужно будет ничего записывать на этом этапе. Этот шаг превратится просто в разбивку проекта в ваших головах и выяснение его входов и выходов. Для более крупных проектов мы можем захотеть создать блок-схемы и даже диаграммы, которые показывают, как мы хотим, чтобы все работало вместе. Как только мы перейдем к проектам в этой книге, мы увидим разные способы разделения компоненты в их отдельные блоки и как определить логику для входов и выходов.

После того, как мы разбили наш проект на отдельные прототипы с отдельными входными и выходными цепями, мы хотели бы создать принципиальные схемы для каждого из этих прототипов.

После того, как мы определили входы и выходы схемы и логику проекта, следующим шагом будет построение схем цепей. Мы хотим убедиться, что мы составили рисунки всех схем, которые мы создаем, даже самых простых. Это поможет нам определить необходимые резисторы и наглядно представить, как мы хотим организовать и подключить компонент.

При создании принципиальных схем я бы рекомендовал использовать программу Fritzing, описанное в главе 3 «Принципиальные схемы». Теперь давайте посмотрим на последний шаг, построение прототипа.

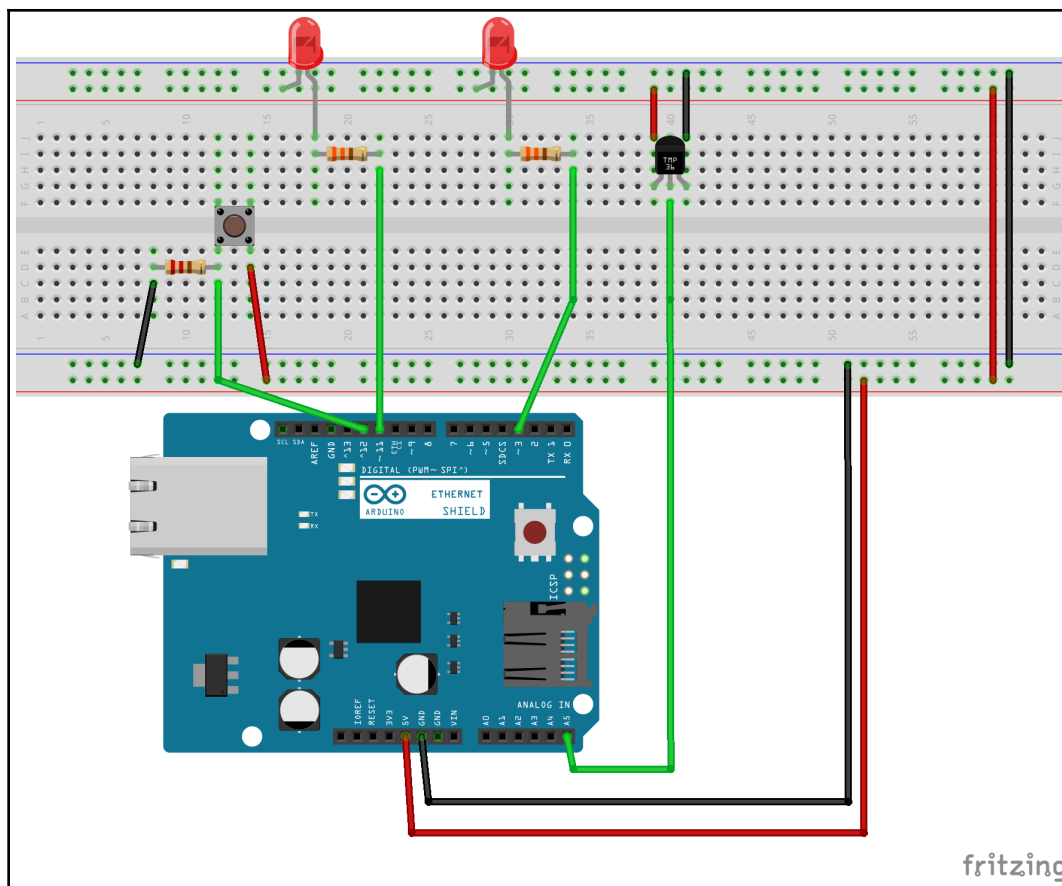
Последний шаг - создание рабочего прототипа. Здесь объединяется вся тяжелая работа первых трех шагов, и вы можете увидеть, работает ли прототип так, как ожидалось.

Все мы предпочли бы пропустить первые несколько шагов и сразу перейти к созданию прототипа; однако вы добьетесь большего успеха и повредите меньше деталей, если не торопитесь и пройдете каждый из этих шагов при создании своих прототипов.

Теперь давайте посмотрим, как мы пройдем эти шаги и создадим наш первый прототип. Вы захотите продолжить и построить этот прототип самостоятельно, потому что мы будем использовать его в следующих нескольких главах, когда мы научимся программировать Arduino.

Первый прототип, который мы создадим, довольно прост и предназначен для использования в следующих нескольких главах, где мы научимся программировать Arduino. У этого прототипа будет одна кнопка, которая будет использоваться для включения или выключения светодиода, еще один светодиод, который мы можем включить или выключить, и датчик температуры TMP36.

В этом прототипе у нас будет два входа (кнопка и датчик температуры) и два выхода (два светодиода). Мы будем использовать +5 В от Arduino для питания компонентов. Вот схема Фритцинга этого прототипа:



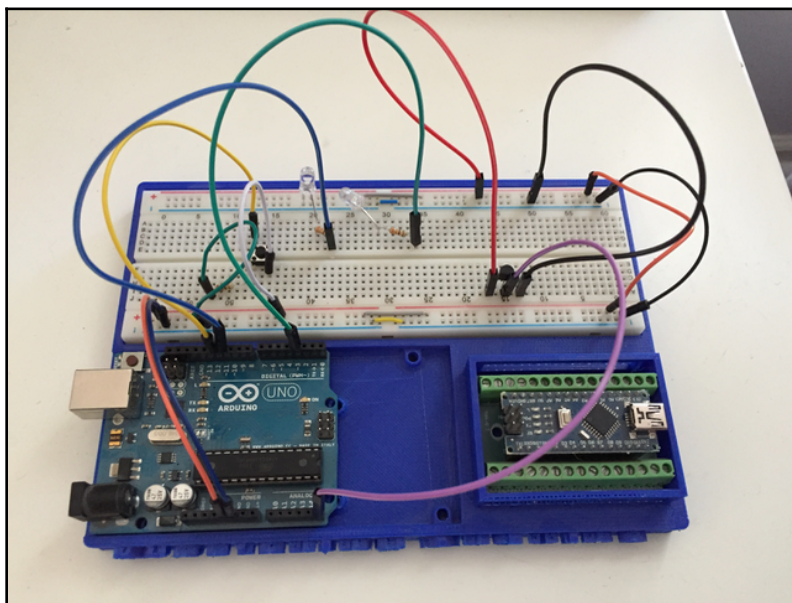
На этой диаграмме слева направо мы видим кнопку, два светодиода и датчик температуры TMP36. Каждый из светодиодов имеет тот же резистор на 330 Ом, который использовался в главе 3 «Принципиальная схема». Поскольку питание от Arduino составляет всего 5 вольт, мы можем снизить сопротивление резистора до 100 Ом, но резисторы на 330 Ом также будут работать нормально.

Резистор, который мы используем в кнопке, называется понижающим резистором, потому что один конец подключен к земле. Цифровая логическая схема действительно может иметь три состояния уровня: высокий, низкий и плавающий. Плавающее состояние возникает, когда вывод не подтягивается ни высоко, ни низко, а остается плавающим. В этом плавающем состоянии микроконтроллер может непредсказуемо интерпретировать это состояние как высокое или низкое. Чтобы решить эту проблему, используется понижающий резистор, который переводит плавающее состояние в низкий уровень т.е. к земле.

Если резистор был подключен к источнику напряжения, а не к земле, он считается подтягивающим резистором. Подтягивающий резистор подтягивает вывод к +5В. Мы будем использовать подтягивающие и понижающие резисторы в нескольких проектах этой книги.

Кнопка подключена к выводу 12 Arduino; следовательно, мы сможем прочитать состояние кнопки, проверив состояние вывода 12. Светодиоды подключены к контактам 11 и 3; следовательно, мы сможем включить их, подав цифровой сигнал на эти выводы, или выключить их, послав цифровой сигнал низкого уровня. Наконец, выходной контакт датчика температуры TMP36 (средний контакт) подключен к аналоговому 5-му контакту, так как выход датчика является аналоговым. Контакты напряжения и заземления на датчике температуры TMP36 подключены к шинам +5В и земли на макетной плате.

Диаграмма Фритцинга делает прототип таким красивым и организованным; однако большинство прототипов так не выглядят. Вот как выглядел прототип, когда я создавал его для этой книги:



Приятно иметь ваш прототип аккуратным и хорошо выглядящим, но в этом нет необходимости, если вы можете с ним работать. Особенно сложно содержать прототип хорошо выглядящим, когда у вас подключено от 10 и больше перемычек Dupont .

Вы также заметите короткие провода, проходящие через шины питания. На этой макетной плате шины питания не подключены полностью по горизонтали. Это пример макета, который настроен немного иначе. Вы можете сказать, что шины питания не проходят полностью, потому что красные и синие линии на макетной плате имеют промежуток на полпути; поэтому мы использовали небольшие кабели для соединения двух сторон. Бывают случаи, когда нам нужны разные источники питания, поэтому такое разделение шин питания может быть очень полезным, но для этого прототипа, поскольку мы используем один и тот же источник питания для всех компонентов, мы соединили силовые шины вместе.

В этой главе мы рассмотрели основы прототипирования и то, какие шаги мы должны предпринять при создании прототипов. Образец прототипа, который мы создали в конце этой главы, был довольно простым, но мы все же прошли каждый из шагов, чтобы убедиться, что все сделано правильно. Мы рассмотрим каждый из этих шагов в главах, посвященных проектам, далее в этой книге.

Теперь, когда мы создали наш первый прототип, нам нужно научиться программировать Arduino. Следующие три главы написаны, чтобы научить вас программировать Arduino. Мы начнем с того, что покажем вам, как загрузить и установить IDE Arduino.

Arduino IDE

Я занимался программированием в качестве хобби или профессионально уже более 37 лет. В то время я использовал множество различных интегрированных сред разработки (IDE) и текстовых редакторов для написания кода. Я написал свою первую программу Hello World на телетайпе, который не использовал ни IDE, ни текстовый редактор. Когда я купил свой первый компьютер, которым был Commodore VIC-20, я использовал язык программирования BASIC для написания своих программ. В режиме программирования вы вводили код построчно, и каждая строка попадала в память по мере того, как вы ее вводили, хорошего редактора или IDE не было. Я не использовал настоящую IDE, пока не научился программировать на языке C на IBM PCjr.

В этой главе вы узнаете:

- Что такое эскиз Arduino
- Что такое Arduino IDE
- Что такое веб-редактор Arduino
- Как написать свой первый скетч

Прежде чем мы рассмотрим IDE Arduino и веб-редактор, давайте посмотрим, что такое Arduino Sketch, чтобы помочь нам понять, зачем нам эти инструменты.

Arduino

Когда мы программируем Arduino, код помещается в проект. Эти проекты называются скетчами, и скетч сохраняется в Sketchbook (папка с скетчами). Скетч разработан так, чтобы быть максимально простым и понятным, за счет абстрагирования многих технических аспектов программирования Arduino с использованием предварительно созданных функций.

Язык кодирования, используемый для программирования Arduino, очень похож на язык программирования C. Мы рассмотрим, как программировать Arduino, в главе 4 «Программирование Arduino - основы» и в главе 6 «Программирование Arduino - помимо основ». В этой главе вы познакомитесь с скетчами и познакомитесь с инструментами, которые мы можем использовать.

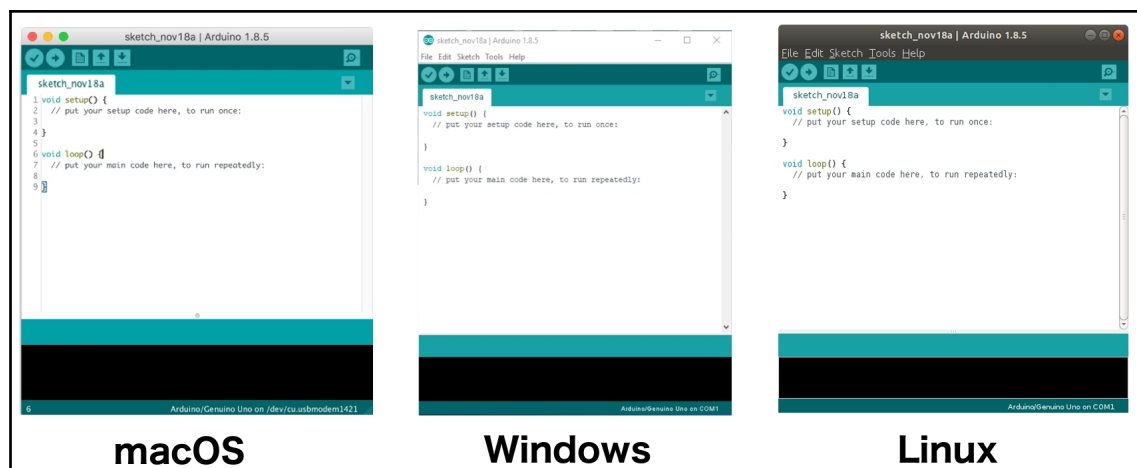
Прежде чем скетч можно будет загрузить в Arduino, Arduino IDE или веб-редактор должны выполнить несколько шагов для создания приложения. Первый шаг к созданию скетча - выполнить некоторую предварительную обработку, которая превращает скетч в программу C++ (см. Плюс-плюс), которая передается компилятору, чтобы превратить этот читаемый человеком код C++ в машиночитаемые инструкции (объектные файлы). Затем эти объектные файлы связываются со стандартными библиотеками Arduino, которые обеспечивают базовую функциональность Arduino. Результатом этого связывания является один шестнадцатеричный файл, который можно загрузить в Arduino и запустить. Приятно то, что инструменты Arduino выполняют все это автоматически, когда мы говорим им загрузить скетч на плату Arduino.

Если бы мы дали определение тому, что такое Arduino Sketch, мы бы сказали, что это проект, который содержит читаемый человеком код, который можно создать и загрузить в Arduino. Теперь давайте рассмотрим два инструмента, которые мы можем использовать, чтобы помочь нам написать и построить эти скетчи, начиная с Arduino IDE.

Arduino IDE

Arduino IDE - это интегрированная среда разработки, которую можно установить локально на компьютеры под управлением Windows, macOS и Linux. IDE можно загрузить со страницы программного обеспечения Arduino по этому URL-адресу: <https://www.Arduino.cc/en/Main/Software>. На момент написания этой книги последней стабильной версией IDE была 1.8.5.

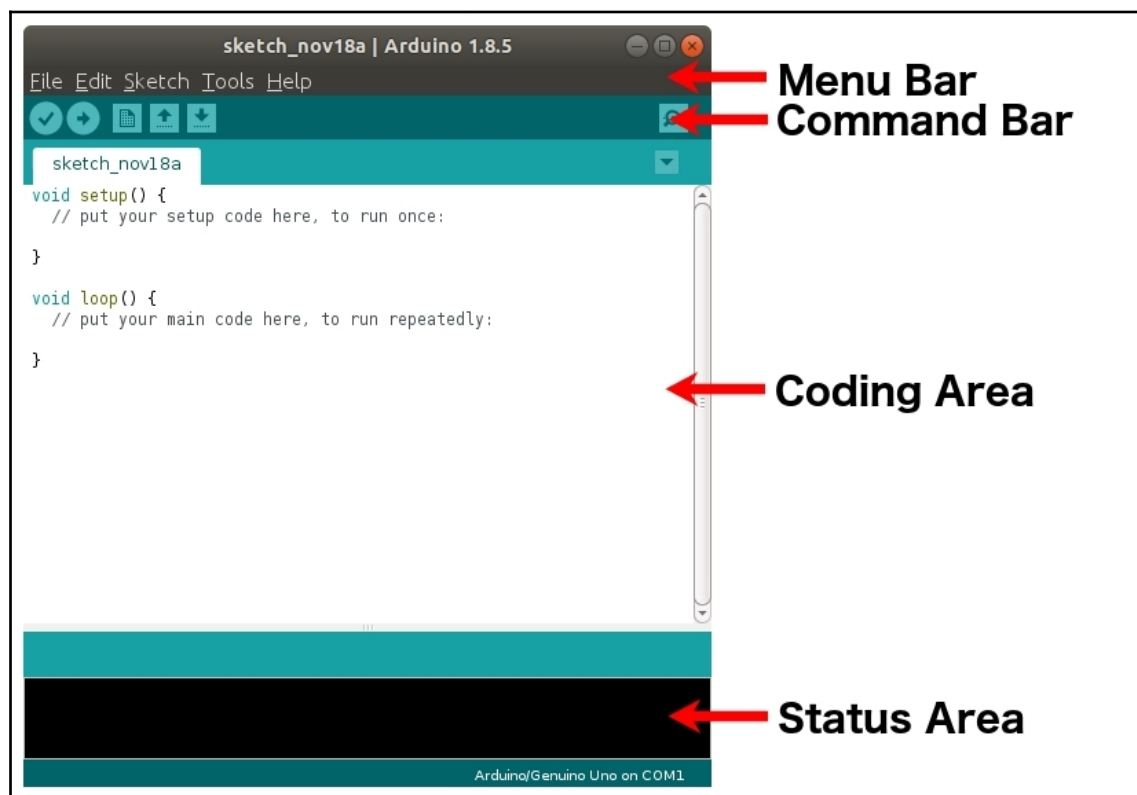
Следующие изображения показывают, как IDE будет выглядеть при первом запуске:



Мы начнем использовать IDE в конце этой главы, где мы создадим наш первый скетч. Мы будем широко использовать его, а также веб-редактор на протяжении всей оставшейся части книги. А пока давайте кратко исследуем среду IDE, чтобы мы могли ознакомиться с некоторыми его основными функциями

IDE

Основное окно разработки IDE состоит из четырех областей, которые показаны на следующем снимке экрана :



Строка меню для IDE функционирует как строки меню в других приложениях, где вы щелкаете по одному из параметров, и появляется подменю с дополнительными параметрами. По ходу книги мы рассмотрим некоторые из наиболее часто используемых параметров меню.

Панель команд обеспечивает быстрый доступ к пяти наиболее часто используемым командам. Эти команды слева направо: проверить, загрузка, новый, открыть и сохранить. Команда проверить попытается скомпилировать скетч, чтобы убедиться, что с кодом все в порядке. Команда загрузка попытается построить и загрузить скетч в подключенный Arduino. Команда новый создаст новый скетч. Команда открыть откроет эскиз. Наконец, команда сохранить сохранит эскиз.



. Чтобы загрузить скетч, у вас должна быть Arduino, подключенная к компьютеру, и настроенная в среде IDE, в противном случае вы получите сообщение об ошибке. Мы рассмотрим, как это сделать, в разделе «Настройка Arduino в среде IDE».

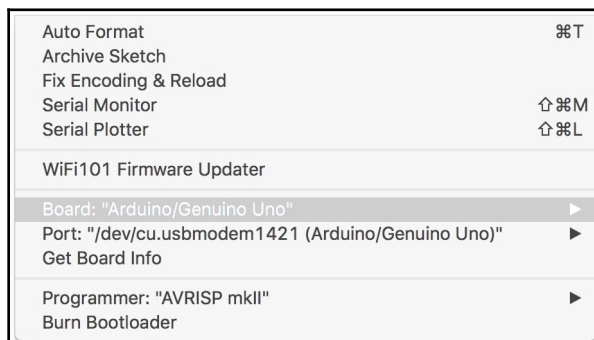
В области кодирования мы пишем код для Arduino. Вы заметите, что когда мы запускаем новый скетч, на главной вкладке автоматически создаются две функции ((`setup`(настройка) и `loop`(цикл)). В этой книге мы будем много работать с этими функциями. Мы рассмотрим, что делают эти две функции, в конце этой главы, когда мы создадим наш первый набросок.

Область состояния используется IDE, чтобы сообщить нам, что происходит, когда IDE выполняет что-то вроде компиляции, загрузки или проверки скетча.

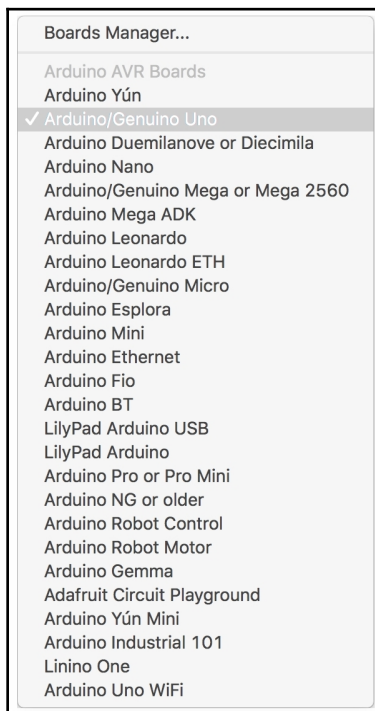
Чтобы загрузить эскиз в Arduino, нам нужно подключить Arduino к компьютеру, на котором работает IDE, с помощью кабеля USB и настроить его в IDE. Для настройки Arduino в среде IDE нам необходимо указать, какой тип Arduino мы используем и на каком порту он находится. Посмотрим, как это сделать.

Arduino IDE

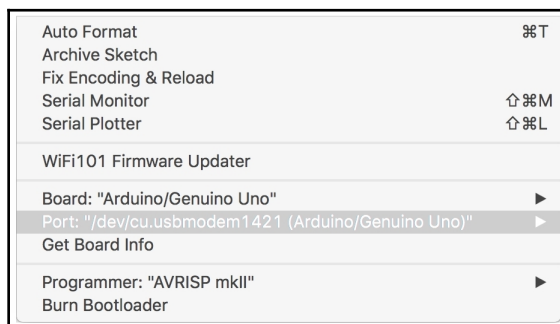
Чтобы подключить Arduino к IDE, первое, что нужно знать IDE, - это какой тип Arduino используется. Для этого мы нажимаем на опцию Инструменты в строке меню и выбираем опцию Платы, как показано на следующем снимке экрана:



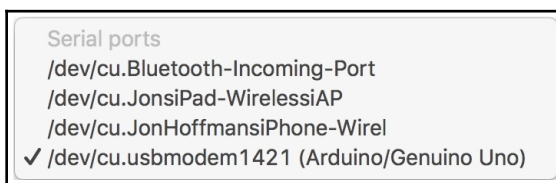
Когда выбрана опция Платы, IDE отображает список совместимых плат. Этот список будет похож на следующий снимок экрана:



Из этого списка выберите плату, которую вы используете для своего проекта. После выбора платы следующее, что нужно знать IDE, - это порт, к которому также подключена Arduino. Чтобы выбрать порт, щелкните пункт меню «Инструменты» в строке меню, а затем выберите параметр «Порт», как показано на следующем снимке экрана:



Когда в этом меню выбран параметр «Порт», в среде IDE отображается список известных ей портов. Этот список должен выглядеть примерно так:



У вас, вероятно, будет другой список портов; однако обычно очевидно, какой порт следует выбрать, потому что он отображает имя платы, подключенной к порту, если она видит плату. В большинстве случаев IDE автоматически выбирает соответствующий порт для вас. После выбора платы и порта среда IDE готова загрузить скомпилированный скетч на плату. Теперь, когда мы увидели, как использовать IDE Arduino, давайте посмотрим, как настроить и использовать веб-редактор Arduino.

- Arduino

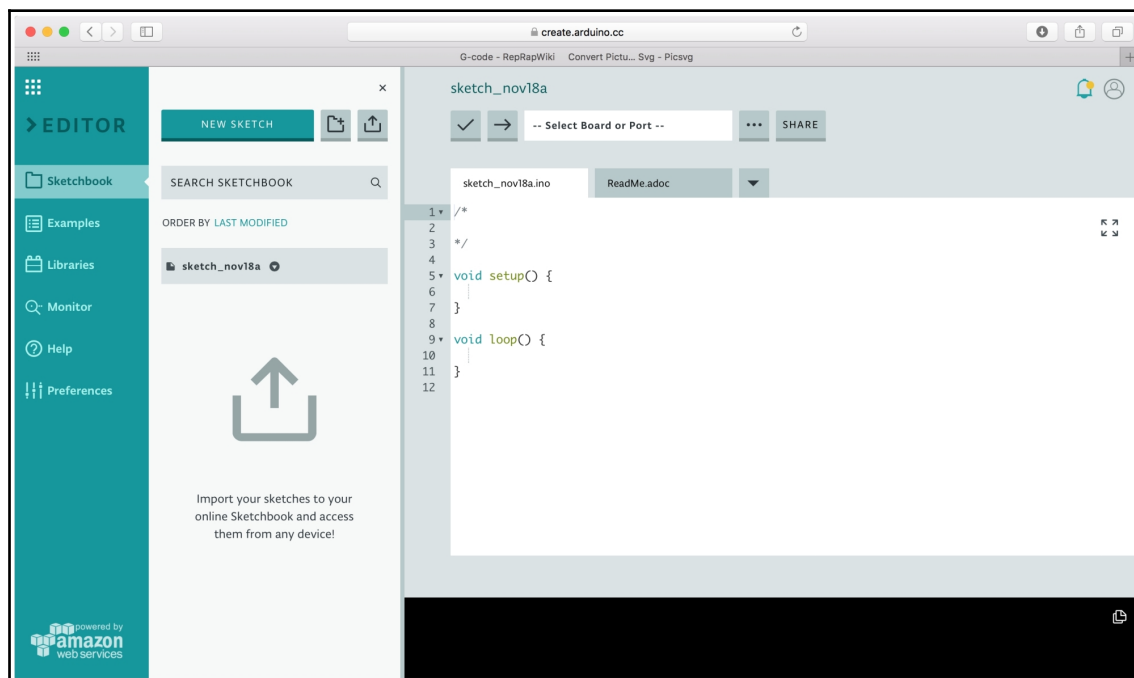
Веб-редактор Arduino позволяет нам создавать и загружать эскизы в большинстве веб-браузеров. Официально веб-редактор поддерживается браузерами Chrome, Firefox, Safari и Edge с установкой плагина.



Веб-редактор является частью проекта Arduino Create, и к нему можно получить доступ здесь: <https://create.arduino.cc>

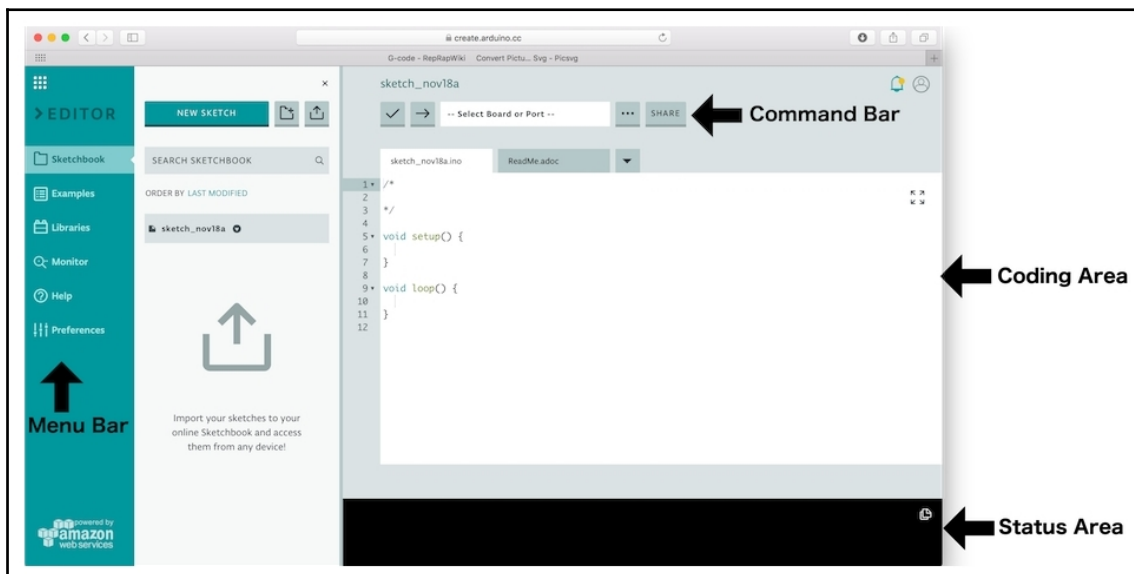
Прежде чем вы сможете установить плагин и использовать веб-редактор, вам необходимо создать бесплатный аккаунт Arduino. Как только мы войдем в наш аккаунт, сайт проведет вас через установку плагина.

После установки плагина вы должны увидеть страницу, подобную этой:



Давайте изучим веб-редактор, чтобы узнать, как им пользоваться.

Четыре основных области веб-редактора показаны на следующем снимке экрана:



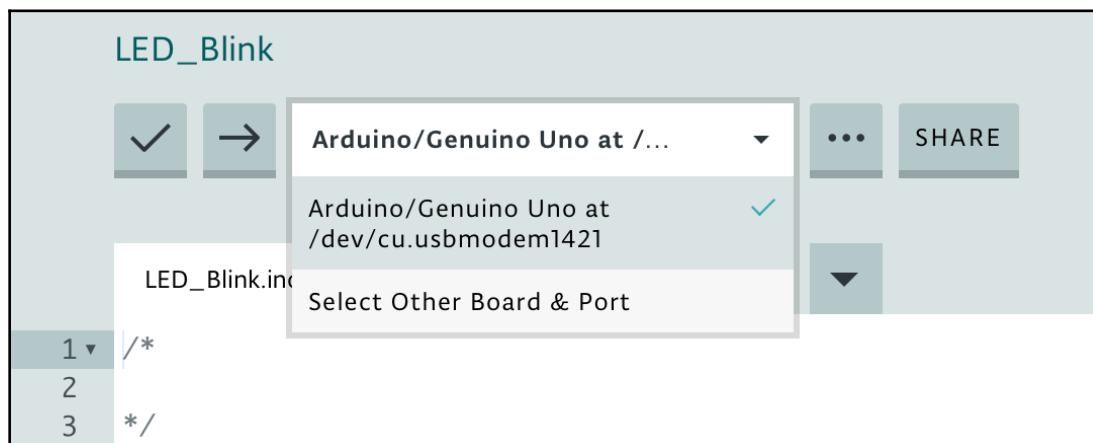
Строка меню веб-редактора позволяет нам быстро получить доступ к определенным элементам, таким как примеры, библиотеки и монитор последовательного порта. Мы рассмотрим эти элементы позже в этой главе.

Панель команд обеспечивает быстрый доступ к часто используемым командам и дает нам возможность выбрать используемую плату. Значок с галочкой будет проверять эскиз, в то время как значок со стрелкой компилирует и загружает эскиз в Arduino. Значок с тремя точками открывает меню, которое позволяет нам сохранять, переименовывать, загружать и удалять текущий эскиз.

В области кодирования мы пишем код для Arduino. Как и в случае с Arduino IDE, вы заметите, что функции `setup()` и `loop()` были автоматически созданы при запуске нового скетча. Область состояния используется IDE, чтобы сообщить нам, что происходит, когда IDE выполняет что-то вроде компиляции, загрузки или проверки скетча. Давайте посмотрим, как настроить Arduino в веб-редакторе, чтобы мы могли загружать в него скетчи.

Arduino IDE

Чтобы загрузить эскиз в Arduino, нам нужно подключить Arduino к компьютеру, на котором работает веб-редактор, с помощью кабеля USB. После подключения Arduino к компьютеру мы можем выбрать Arduino и порт в веб-редакторе, щелкнув раздел «Выбрать плату или порт». Если веб-редактор распознает плату Arduino, вы должны увидеть плату и порт Arduino, указанные в раскрывающемся меню. Список будет выглядеть так: снимок экрана:

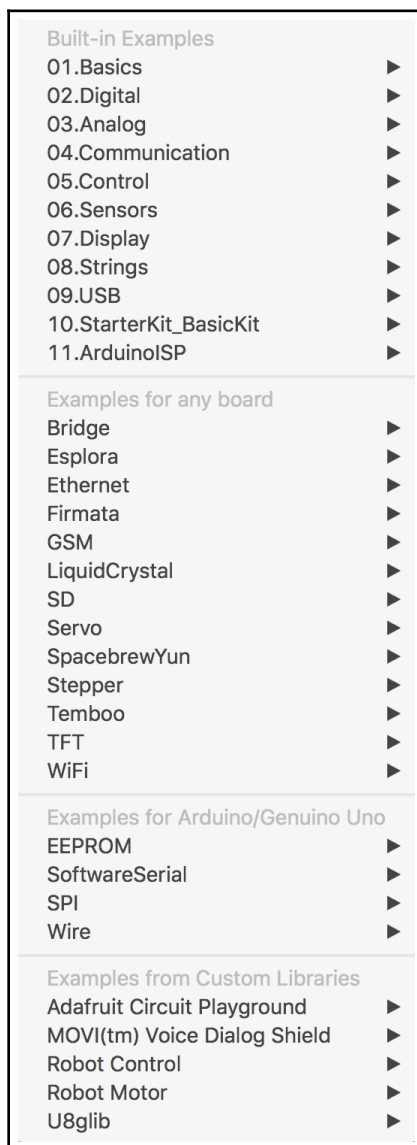


Если вы видите Arduino, выберите его, и тогда вы сможете загрузить его скомпилированные скетчи. Лучший способ изучить Arduino IDE или веб-редактор - это использовать его, и мы будем использовать оба варианта в этой книге.

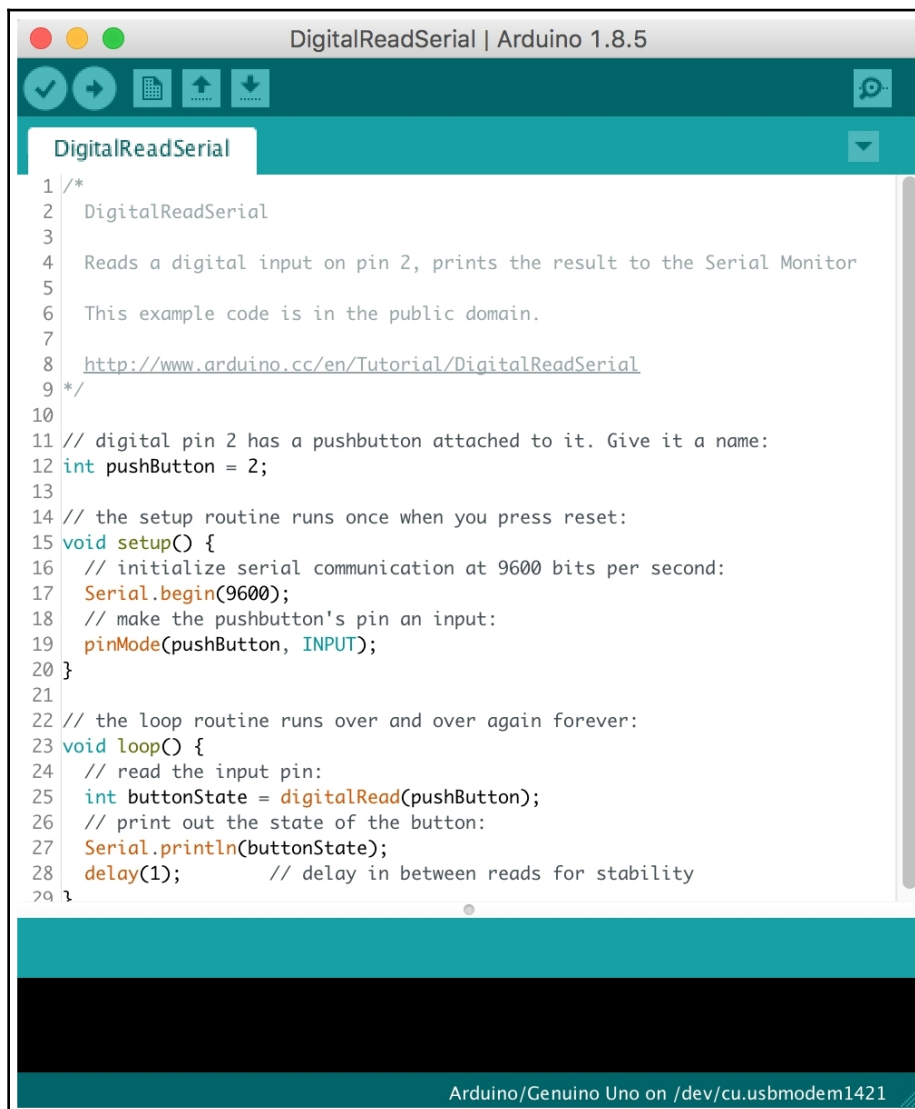
Давайте рассмотрим некоторые функции Arduino IDE и веб-редактора, начиная с включенных примеров.

В состав Arduino IDE и веб-редактора включены многочисленные примеры, которые представляют собой простые наброски, демонстрирующие команды Arduino и способы их использования. Эти примеры варьируются от самых простых набросков, демонстрирующих, как читать и писать цифровой ввод / вывод, до более сложных эскизов, показывающих, как использовать датчики. Хотя эти примеры предназначены для демонстрации команд Arduino, их также можно использовать в качестве примеров того, как писать хороший код для Arduino.

Чтобы получить доступ к примерам в среде Arduino IDE, щелкните параметр «Файл» в строке меню и перейдите к параметру «Примеры». Вы увидите список категорий для примеров, которые выглядят как на следующем снимке экрана:



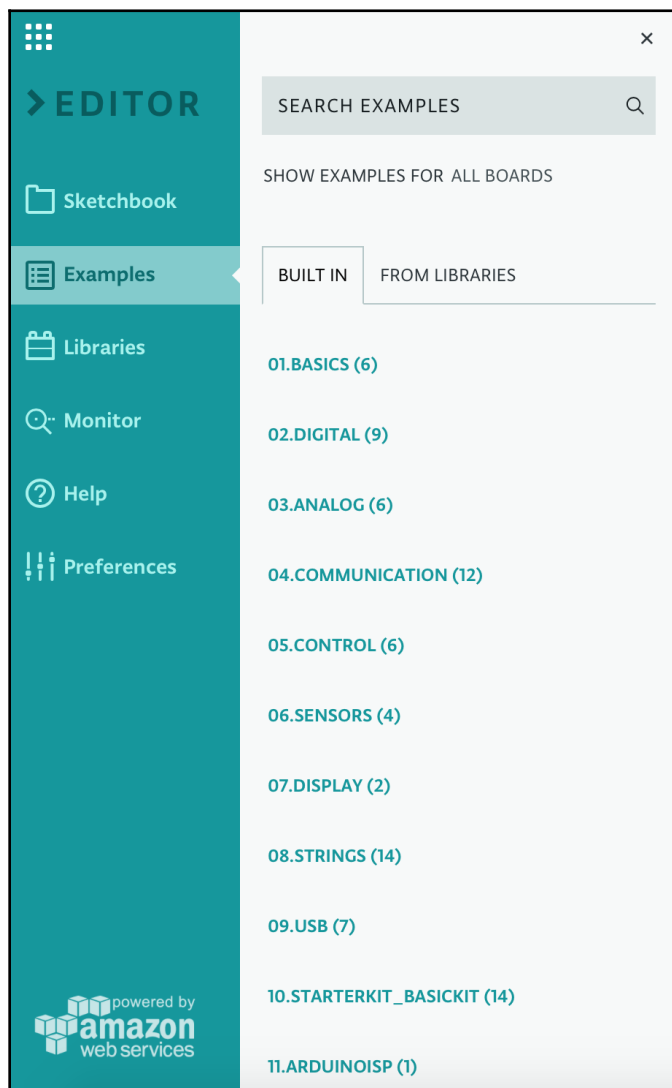
Если вы перейдете к любой из категорий, вы увидите список примеров для этой категории. Если вы выберете пример, такой как пример `DigitalReadSerial` в категории «Основные», код этого примера загрузится в новом окне, и эскиз будет выглядеть так, как показано на следующем снимке экрана:



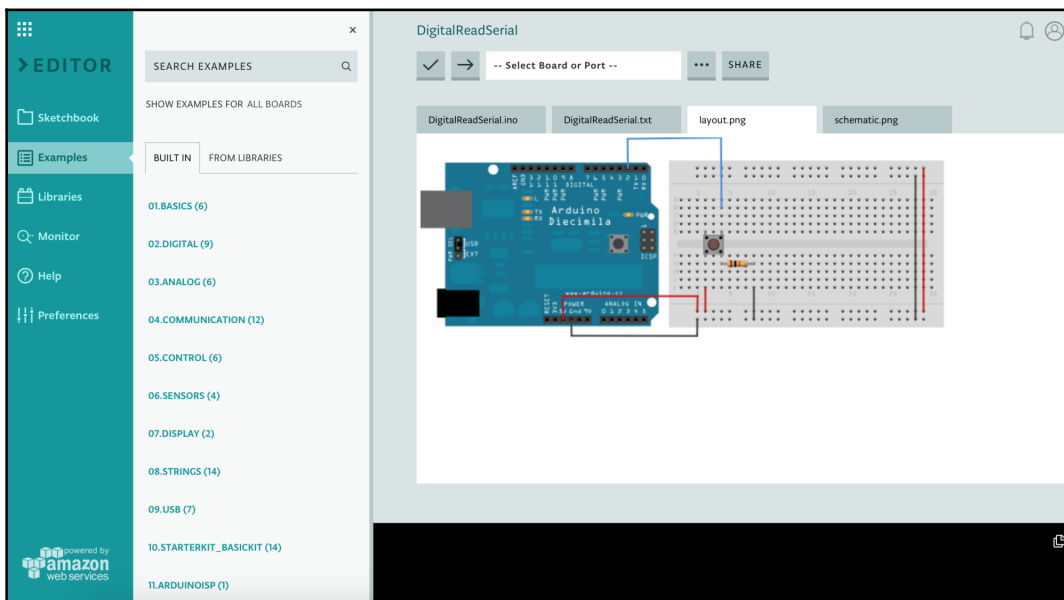
```
1 /*
2  DigitalReadSerial
3
4  Reads a digital input on pin 2, prints the result to the Serial Monitor
5
6  This example code is in the public domain.
7
8  http://www.arduino.cc/en/Tutorial/DigitalReadSerial
9  */
10
11 // digital pin 2 has a pushbutton attached to it. Give it a name:
12 int pushButton = 2;
13
14 // the setup routine runs once when you press reset:
15 void setup() {
16   // initialize serial communication at 9600 bits per second:
17   Serial.begin(9600);
18   // make the pushbutton's pin an input:
19   pinMode(pushButton, INPUT);
20 }
21
22 // the loop routine runs over and over again forever:
23 void loop() {
24   // read the input pin:
25   int buttonState = digitalRead(pushButton);
26   // print out the state of the button:
27   Serial.println(buttonState);
28   delay(1);          // delay in between reads for stability
29 }
```

Arduino/Genuino Uno on /dev/cu.usbmodem1421

В веб-редакторе, чтобы загрузить пример, выберите опцию Примеры в строке меню. Как и в случае с Arduino IDE, мы увидим тот же список категорий примеров. Этот список будет похож на следующий снимок экрана:



Затем мы можем выбрать любую из категорий, чтобы увидеть список примеров. Что отличает примеры в веб-редакторе от примеров в среде Arduino IDE, так это то, что большинство примеров в веб-редакторе также включают макет (Fritzing) и схематические рисунки, показывающие, как создать схему, которую можно использовать с примером. Например, если мы выберем тот же пример `DigitalReadSerial`, который мы выбрали в Arduino IDE, мы не только увидим код для скетча, но также увидим макет, как показано на следующем снимке экрана:



Рисунки, входящие в состав веб-редактора, отлично подходят для новичков, потому что они показывают, как построить схему без полного понимания кода примера.

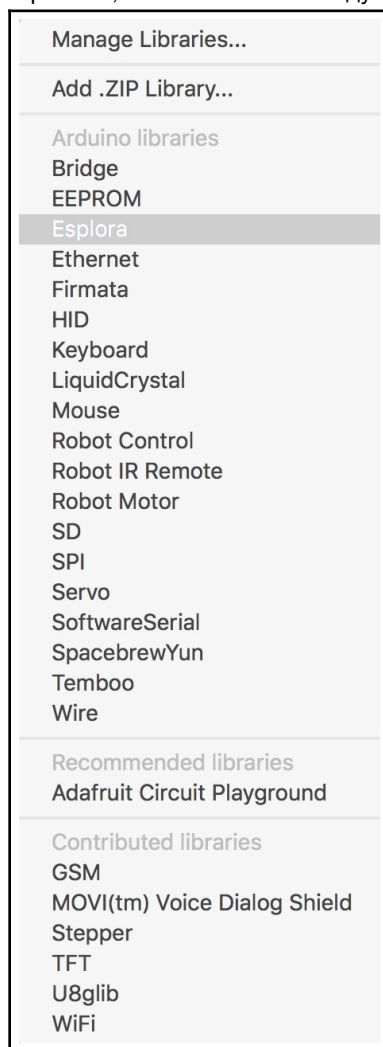
Существует множество внешних библиотек, которые можно использовать с Arduino. Давайте посмотрим, что такое библиотеки.

Arduino

Как и большинство платформ разработки, среда Arduino может быть расширена библиотеками. Эти библиотеки предоставляют дополнительные функции, которые мы можем использовать в наших набросках, такие как предоставление доступа к определенному оборудованию, манипулирование данными и добавление дополнительных функций, таких как планировщик задач (библиотека `Arduino Cron`). В среду IDE и веб-редактор встроено множество библиотек, но мы также можем загрузить другие библиотеки или создать свои собственные.

Чтобы получить доступ к библиотекам в Arduino IDE, мы выбираем опцию Sketch в меню, а затем выбираем опцию Include Library. Появится другое меню, которое позволяет вам загружать библиотеку или управлять библиотеками.

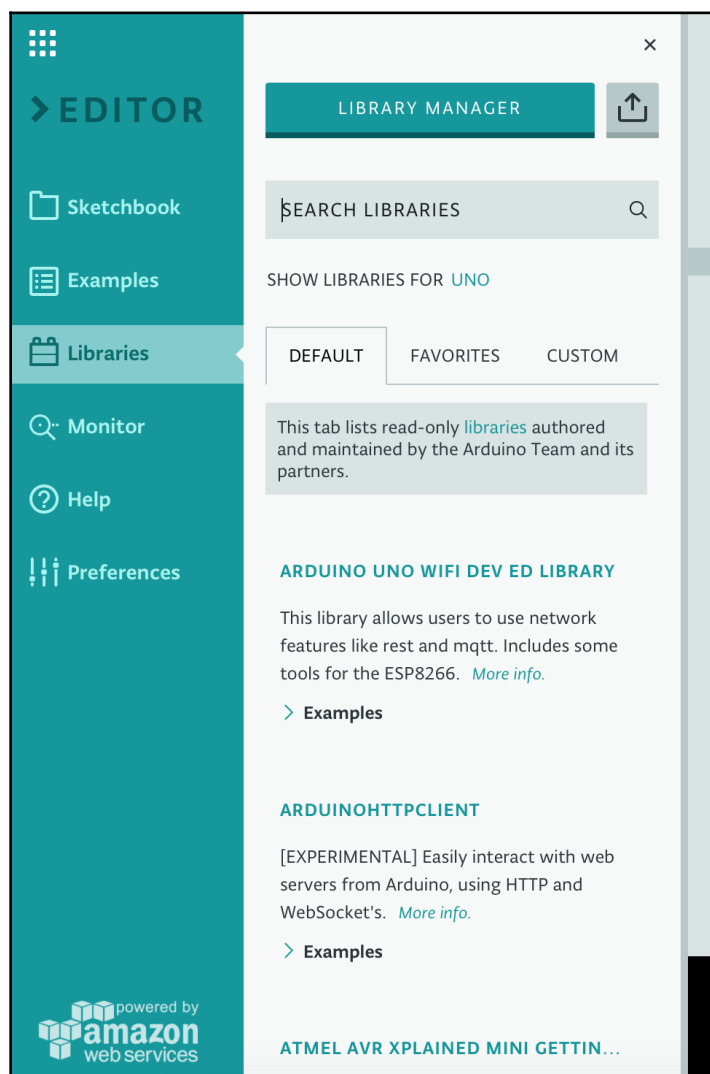
Это меню должно выглядеть примерно так, как показано на следующем снимке экрана:



Если вы выберете любую из встроенных библиотек, файлы заголовков будут автоматически включены в ваш эскиз. Мы узнаем больше о файлах заголовков в главе 6, Программирование Arduino - основы и главе 7, Программирование Arduino - помимо основ.

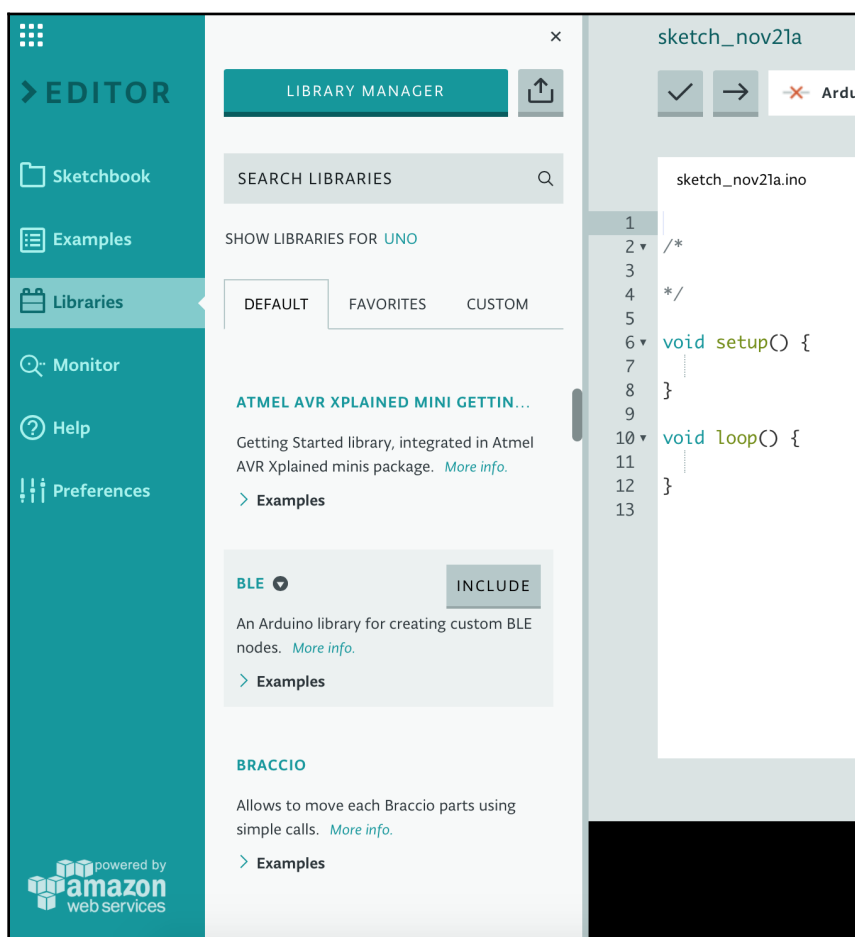
Выбор опции «Управление библиотеками» позволяет нам загружать и устанавливать другие библиотеки, которые не входят в стандартную установку Arduino IDE. После загрузки и установки библиотеки она появится в быстром списке библиотек Arduino, и ее можно будет использовать так же, как встроенные библиотеки. Некоторые из этих библиотек также устанавливают пример кода, доступ к которому можно получить в разделе примеров среды IDE.

Чтобы получить доступ к библиотекам в веб-редакторе, выберите опцию «Библиотека» в строке меню, и справа от строки меню появится список доступных библиотек с полосой поиска. Интерфейс будет выглядеть как следующий снимок экрана:



В веб-редактор включены сотни библиотек. Это упрощает доступ к библиотекам по сравнению с Arduino IDE, потому что нам не нужно их устанавливать. Веб-редактор также упрощает обмен скетчами, для которых требуются библиотеки. При совместном использовании скетча, созданного с помощью Arduino IDE, человек, который получает скетч, должен установить правильные версии необходимых библиотек. Иногда это может стать сложным и запутанным. С помощью веб-редактора, когда мы публикуем скетч, веб-редактор гарантирует, что при компиляции скетча используются правильные библиотеки.

Чтобы добавить библиотеку в скетч, выполните поиск библиотеки в строке поиска, и когда библиотека появится в списке, наведите на нее указатель мыши, и на следующем снимке экрана появится кнопка ВКЛЮЧИТЬ:



Нажмите кнопку INCLUDE - ВКЛЮЧИТЬ, и необходимые заголовки появятся в коде, а библиотека будет включена в эскиз.

Прежде чем мы создадим наш первый набросок, давайте посмотрим, что такое монитор последовательного порта.

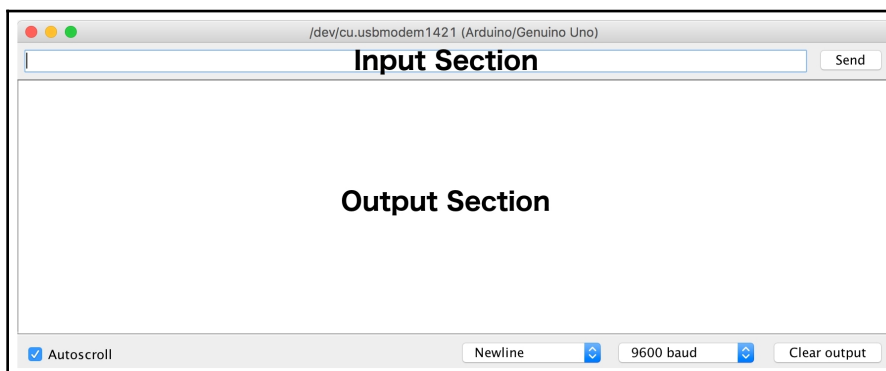
Монитор последовательного порта отправляет и / или принимает текст, обычно через USB-кабель. Это позволяет нам получать отладочные сообщения или отправлять текст с клавиатуры в веб-редакторе или Arduino IDE. Посмотрим как сделать и то, и другое, когда мы создадим наши первые скетчи в конце этой главы.

Чтобы использовать последовательный монитор с IDE Arduino или с веб-редактором, вы должны сначала подключить Arduino к компьютеру и установить связь между Arduino и IDE или редактором.

Чтобы начать использовать монитор последовательного порта в среде Arduino IDE, щелкните значок монитора последовательного порта в верхнем правом углу IDE. На следующем снимке экрана выделен значок серийного монитора:

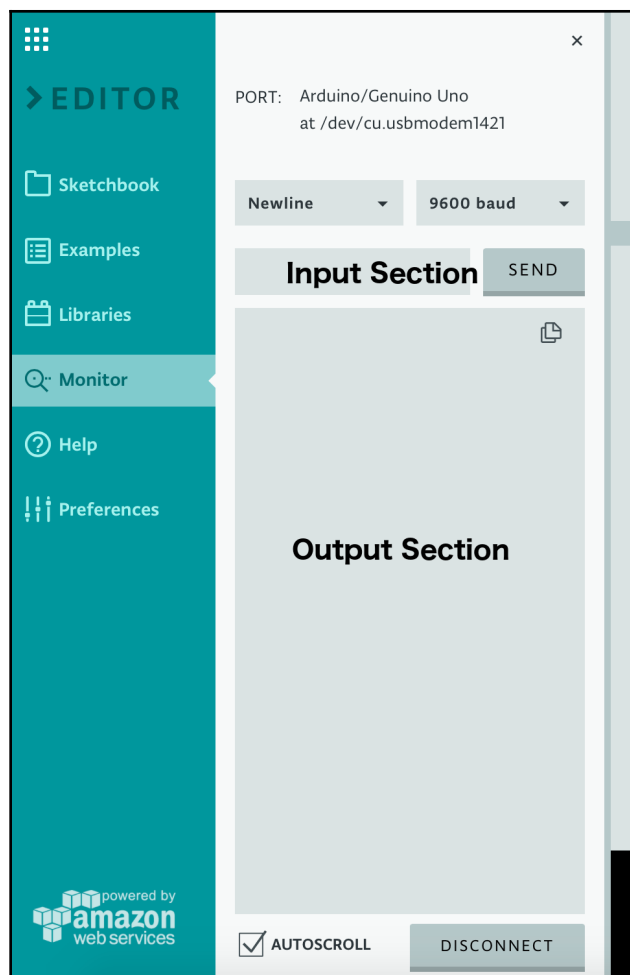


Монитор последовательного порта откроется в отдельном окне, как показано на следующем снимке экрана:



Раздел ввода позволяет нам отправлять текст в Arduino. Для этого введите текст в поле ввода и нажмите кнопку «Отправить» или нажмите «Ввод», чтобы отправить его. Текст из Arduino появится в разделе вывода.

Чтобы использовать монитор последовательного порта с веб-редактором, щелкните опцию «Монитор» в меню, и монитор последовательного порта появится справа от строки меню. На следующих снимках экрана показан монитор последовательного порта в веб-редакторе:



Как и в случае с монитором последовательного порта в среде Arduino IDE, чтобы отправить текст в Arduino, введите его в поле ввода в разделе ввода, а затем нажмите кнопку «ОТПРАВИТЬ» или нажмите «Ввод», чтобы отправить его. Выходные данные Arduino появятся в разделе выходных данных.

Теперь, когда у нас есть базовое представление о том, как работают IDE Arduino и веб-редактор, давайте создадим несколько набросков.

Hello World

Для нашего первого Sketch мы создадим традиционное приложение Hello World с помощью Arduino. Это приложение будет выводить слова «Hello World» на монитор последовательного порта; однако, прежде чем мы создадим это приложению, нам нужно понять, что делают функции `setup ()` и `loop ()`.

Функция `setup ()` запускается один раз и только один раз при первом запуске приложения. Эта функция позволяет нам инициировать любые переменные или оборудование при первом запуске приложения.

После завершения, функция `loop ()` вызывается в первый раз. Когда функция `loop ()` завершается, она будет вызвана снова и будет продолжать цикл до тех пор, пока Arduino не выключится.

Продemonстрируем, как работают эти функции. Нам нужно будет начать с создания нового скетча либо в Arduino IDE, либо в веб-редакторе. Чтобы создать новый скетч с помощью Arduino IDE, мы можем использовать значок «Новый» на панель команд или выберите Файл | Новое из меню. Чтобы создать новый скетч с помощью веб-IDE, щелкните опцию Sketchbook (папка с скетчами) в строке меню, а затем нажмите кнопку NEW SKETCH.

После создания нового скетча добавьте в функцию `setup ()` следующий код:

```
Serial.begin(9600);  
Serial.println("Hello World");
```

Затем нам нужно будет подключить Arduino к компьютеру и установить соединение между Arduino и IDE или веб-редактором, как описано ранее в этой главе. Затем мы можем запустить эскиз, используя кнопку загрузки на панели команд для Arduino IDE и Веб-редактор. После того, как код скомпилирован и загружен в Arduino, вы должны увидеть, что слова Hello World выводятся один раз на монитор последовательного порта.

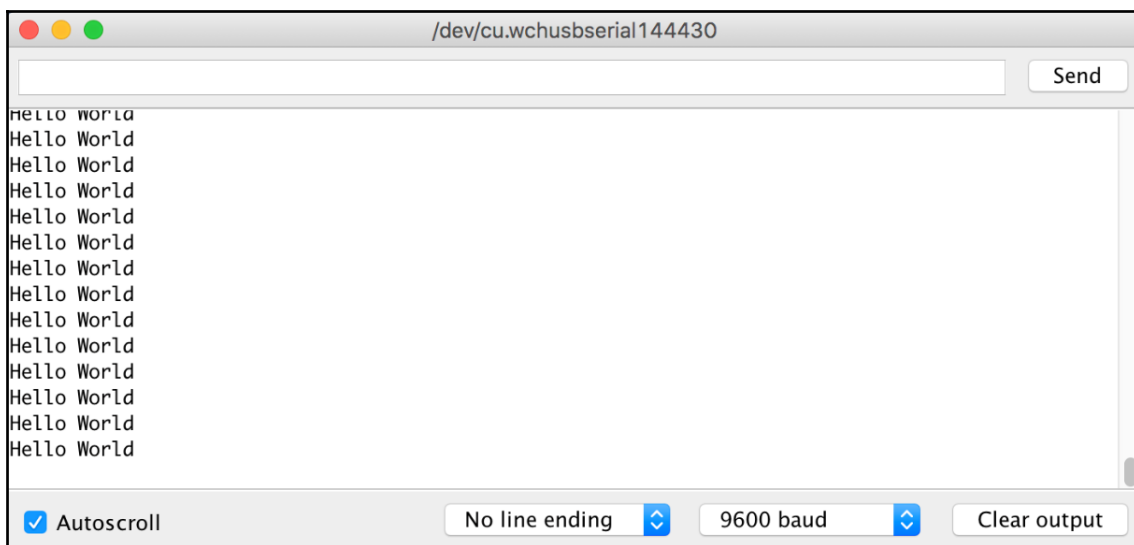
Теперь давайте удалим `Serial.println ("Hello World");` строку из функции `setup ()` и поместите ее в функцию `loop ()`, чтобы наш код выглядел так:

```
void setup() {  
  Serial.begin(9600);  
}  
void loop() {  
  Serial.println("Hello World");  
}
```

Затем мы можем загрузить скетч, и мы должны увидеть, как Hello World снова и снова печатаются на последовательном мониторе. Текст будет продолжать печататься, пока мы не отключим Arduino от компьютера.

В последних двух примерах мы использовали функцию `Serial.println()` для вывода текста на монитор пробной версии. Эта функция выведет текст, а затем добавит новую строку в конце. Мы также могли бы использовать функцию `Serial.print()`, которая выводит текст, но не добавляет новую строку в конце.

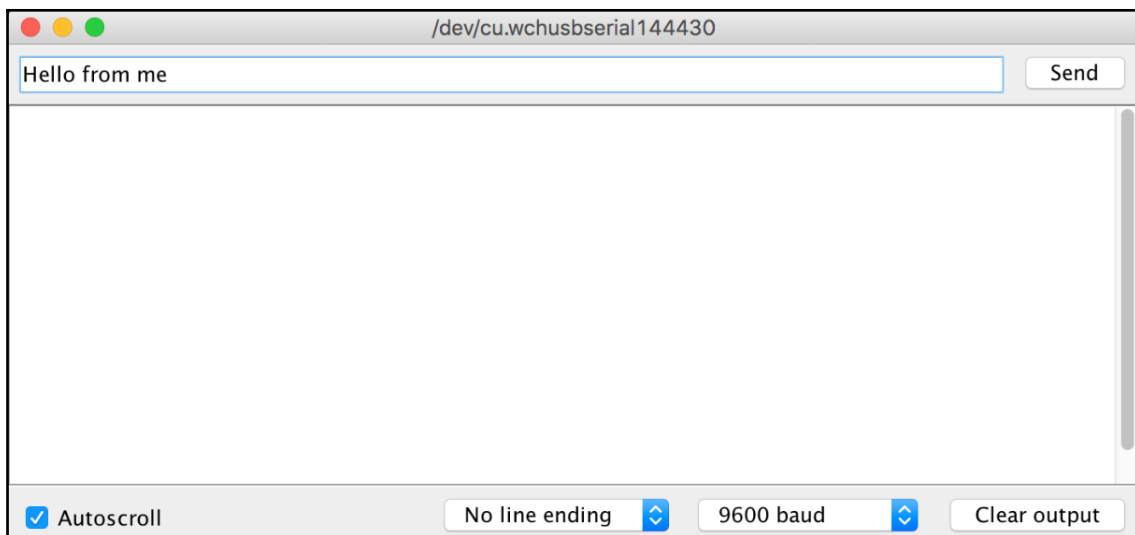
Вывод на монитор последовательного порта должен выглядеть примерно так:



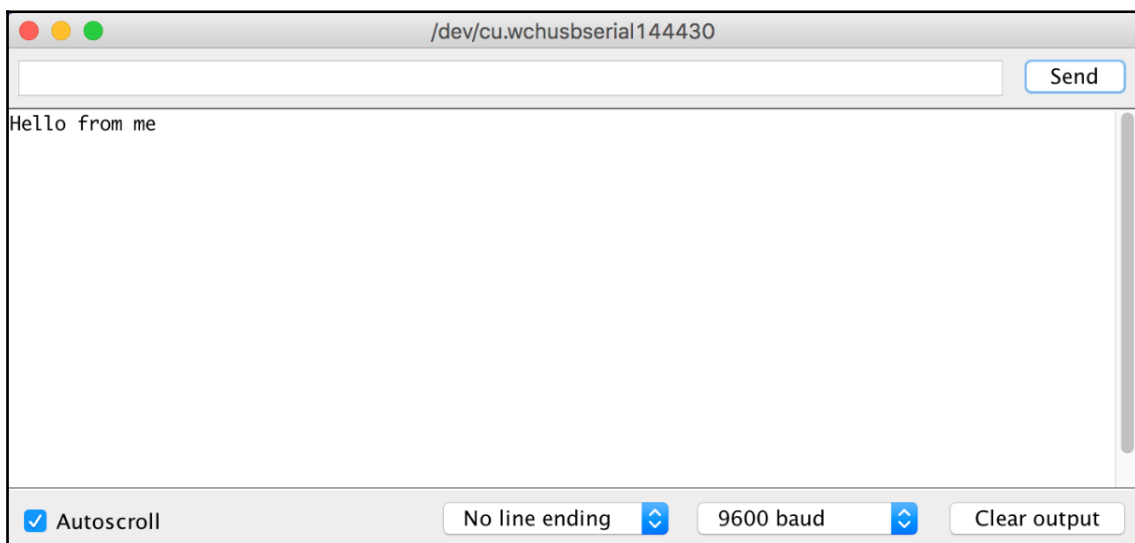
Теперь, когда мы увидели, как выводить текст из Arduino на последовательную консоль, давайте посмотрим, как Arduino может получать текст из последовательного монитора, создавая эхо-приложение.

Приложение (Echo) эхо считывает текст из последовательного монитора и затем выводит его обратно.

Текст будет введен в поле ввода, как показано на следующем снимке экрана:



И текст будет отображаться эхом, как показано на следующем снимке экрана:



Мы начнем с создания нового скетча и добавим в него следующий код:

```
byte bytesIn;

void setup() {
  Serial.begin(9600);
```

```
    }  
  
    void loop() {  
        if (Serial.available()) {  
            bytesIn = Serial.read();  
            Serial.write(bytesIn);  
        }  
    }  
}
```

В этом коде мы начинаем с определения переменной с именем `bytesIn` байтового типа. Затем с помощью функции `setup()` скорость последовательной передачи данных устанавливается на 9600 бод.

В функции `loop()` мы используем функцию `Serial.available()`, чтобы узнать, хранятся ли какие-либо данные в последовательной очереди. Функция `Serial.available()` возвращает количество байтов, доступных для чтения в последовательном приемном буфере. Если есть байты, доступные для чтения, код затем использует функцию `read()` для чтения байтов, а затем использует функцию `write()` для записи байтов обратно в последовательный монитор.

Разница между функцией `write()`, используемой в этом коде, и функцией `println()`, используемой в предыдущих примерах, заключается в том, что функция `println()` будет печатать данные в виде текста ASCII, удобочитаемого человеком, в то время как функция `write()` будет записывать данные в байтах. В этом примере, если мы использовали функцию `println()`, мы увидим ASCII-эквивалент введенных символов, а не самих символов.

В этой главе мы увидели, как настроить IDE Arduino и веб-редактор. Мы также изучили их базовые функции. В конце этой главы мы увидели, как использовать Serial Monitor - монитор последовательного порта для отправки и получения данных на Arduino и обратно.

В следующей главе мы начнем узнавать, как программировать Arduino.

Программирование Arduino - основы

Сколько себя помню, я программировал все, от телетайпов до персональных компьютеров и встроенных устройств. Я программировал игры, бизнес-приложения, веб-сайты и мобильные приложения, но могу честно сказать, что мне больше всего нравится программировать платы микроконтроллеров, такие как Arduino.

Причина этого в том, что с помощью микроконтроллеров мои программы могут взаимодействовать с внешним миром через различные датчики и двигатели. С микроконтроллерами мы ограничены только нашим воображением и изобретательностью. Однако, прежде чем мы сможем начать завоевывать мир, мы должны сначала изучить основы языка программирования Arduino. В этой главе вы узнаете:

- Что такое переменные и константы и как их использовать
- Какие математические функции предлагает язык программирования Arduino
- Как добавить комментарии к нашему коду
- Как принимать решения с помощью языка программирования Arduino
- Как создавать циклы для повторения блоков кода

В главе 5, Arduino IDE, мы узнали, как использовать Arduino IDE и веб-редактор. Мы также изучили функции `setup ()` и `loop ()` и узнали, как их использовать. В этой главе и главе 7, Программирование Arduino - помимо основ, мы узнаем о языке программирования Arduino и о том, как использовать этот язык для разработки приложений для Arduino. Начнем с фигурных скобок.

Левая фигурная скобка (`{`) определяет, где начинается блок кода, а правая фигурная скобка (`}`) определяет, где он заканчивается. Мы видели эти скобки, когда рассматривали функции `setup ()` и `loop ()`; однако фигурные скобки не ограничиваются определением кода внутри функции, они также используются для определения других блоков кода. Мы увидим примеры этого в разделах «Decision making - Принятие решений» и «Looping - Зацикливание» этой главы.

Когда есть левая фигурная скобка, должна быть и правая фигурная скобка. Мы говорим, что фигурные скобки сбалансированы, когда у нас есть равное количество левой и правой фигурных скобок. Несбалансированные фигурные скобки могут привести к ошибкам компилятора. Если вы получаете очень сложные и трудные для понимания ошибки компилятора, вы можете начать поиск и устранение неисправностей, убедившись, что фигурные скобки сбалансированы.

Теперь посмотрим на точки с запятой.

Точка с запятой используется в конце каждого оператора для отделения одного оператора от следующего. Если оператор не заканчивается точкой с запятой, это приведет к ошибке времени компиляции. Текст ошибки при отсутствии точки с запятой довольно очевиден и будет включать номер строки утверждения, в котором оно отсутствует.

Точки с запятой также используются в цикле `for` для разделения различных элементов. Мы рассмотрим цикл `for` в разделе «Looping - Зацикливание» этой главы. Теперь давайте посмотрим, как мы можем добавить комментарии к нашему коду.

Есть два типа комментариев, которые можно использовать в нашем коде Arduino. Это блочные комментарии и строковые комментарии. Блочные комментарии используются, когда текст комментария занимает несколько строк и обычно используется перед вызовом функции, чтобы читатель знал, что делает функция.

Строчные комментарии используются, когда требуется короткий однострочный комментарий, и обычно используются в функциональных блоках, чтобы читатель знал, что делает конкретная строка кода.

Комментарий блока начинается с `/*` и заканчивается `*/`. Следующий код показывает, как будет выглядеть блок-комментарий:

```
/* This is a block comment
   This comment can span multiple lines
   This type of comment is usually found outside function calls
*/
```

Комментарий строки начинается с `//` и продолжается до конца строки. Комментарий к строке может начинаться с начала строки или после окончания оператора. В следующих примерах показано, как будет выглядеть строковый комментарий:

```
// This is a single line comment
Serial.println("Hello World"); // comment after statement
```

Всегда полезно добавлять комментарии к вашему коду, чтобы читатель знал, что делают определенные блоки кода. Теперь давайте посмотрим, что такое переменные.

Переменная используется для хранения информации, на которую можно ссылаться или которой можно управлять в коде. Переменной присваивается уникальное имя, которое затем можно использовать для доступа к информации. Имя переменной должно соответствовать тому, что описывает, что это за переменная, чтобы любой, кто использует код, понял, для чего эта переменная используется. При именовании переменной следует использовать Camelcase.

Camelcase - Верблюжий регистр стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово внутри фразы пишется с прописной буквы. Стиль получил название CamelCase, поскольку прописные буквы внутри слова напоминают горбы верблюда. Некоторыми примерами CamelCase являются `ledOne`, `myVariable` и `redLedOnRightSide`.

Когда переменная объявляется, обычно рекомендуется присвоить ей начальное значение. Это помогает избежать случайного доступа к переменной до ее инициализации. Чтобы объявить переменную, мы определяем тип переменной, за которым следует имя переменной, а затем, если мы собираемся дать ей начальное значение, мы добавляем знак равенства, за которым следует начальное значение. Следующий код показывает, как это сделать:

```
int myInt = 0;
```

В предыдущей строке кода мы объявили переменную `myInt` целочисленного (`int`) типа с начальным значением 0. Давайте посмотрим на некоторые из наиболее популярных встроенных типов данных, которые мы можем использовать с языком Arduino.

В языке программирования Arduino существует множество встроенных типов данных. В этом разделе мы рассмотрим наиболее часто используемые. Начнем с логического типа.

Логический

Тип логических (Boolean) данных может содержать одно из двух возможных значений: true (истина) или false (ложь). В следующем примере показано, как объявить переменную логического типа:

```
boolean myBool = true;
```

В предыдущем коде объявляется переменная с именем myBool логического типа и устанавливается начальное значение true. Логические типы часто используются в стандартной программе Arduino, и все операции сравнения, как мы увидим позже в этой главе, возвращают логическое значение.

Байт

Тип данных байта - это 8-битное числовое значение в диапазоне от 0 до 255. Ниже показано, как мы можем объявить переменную байтового типа:

```
byte myByte = 128;
```

В предыдущем коде объявляется переменная myByte байтового типа с начальным значением 128.

Целое число - это основной тип данных, используемый для хранения числовых данных, когда десятичное значение не требуется. Переменная целочисленного типа может содержать числа от -32 768 до 32 768. Целое число определяется используя ключевое слово int.

Мы можем объявить целое число беззнаковым с помощью ключевого слова unsigned. Целое число без знака может находиться в диапазоне от 0 до 65 535, тогда как нормальное целое число находится в диапазоне от -32 768 до 32 768. В следующем коде показано, как мы определяем обычное целое число, так и целое число без знака:

```
int mySignedInt = 25;  
unsigned int myUnsignedInt = 15;
```

В предыдущем коде мы объявили переменную `mySignedInt` целочисленного типа с начальным значением 25. Мы также объявили вторую переменную `myUnsignedInt` целочисленного типа беззнака с начальным значением 15.



На некоторых платах Arduino, таких как Due или SAMD, целое число может хранить значения больше 32 768 и меньше -32 768. Поскольку большинство плат имеют целочисленный диапазон от -32 768 до 32 768, я бы рекомендовал всегда предполагать, что это диапазон, который вы можете использовать.

Long

Тип данных `long` может хранить целые числа от -2 147 483 648 до 2 147 483 647. В следующем коде показано, как определить переменную `long`:

```
long myLong = 123,456,789;
```

В предыдущем коде мы объявили переменную с именем `myLong` типа `long` и присвоили ей значение 123 456 789. Рекомендуется избегать использования длинного типа данных, если нет необходимости хранить большие числа, поскольку он использует больше памяти, чем целочисленный тип.

Double float

Типы данных `Double` и `float` - это числа с плавающей запятой, что означает, что они могут содержать десятичную точку. Оба типа `double` и `float` могут содержать значения в диапазоне от -3,4028235E + 38 до 3,4028235E + 38.

На большинстве платформ тип данных `float` имеет точность до шести или семи десятичных цифр, в то время как тип данных `double` обычно состоит из пятнадцати цифр; однако на платформе Arduino это не так. На платформе Arduino типы `double` и `float` абсолютно одинаковы, поэтому оба они имеют точность до шести или семи десятичных цифр.

Есть две очень веские причины не использовать значения типа `double` или `float`, если вам абсолютно не нужно десятичное число. Первая причина заключается в неточности: например, 6.0, деленное на 3.0, не всегда может равняться 2. Вы можете получить что-то вроде 1.9999999999. Вторая причина в том, что математика с плавающей запятой намного медленнее, чем целочисленная.

В следующем коде показано, как определить переменные типа `double` и `float`:

```
double myDouble = 1.25;  
float myFloat = 1.5;
```

В предыдущем коде мы объявляем переменную `myDouble` типа `double` со значением 1,25. Мы также объявляем переменную `myFloat` типа `float` со значением 1,5.

Character

Тип данных `char` обычно описывается как тип данных, в котором хранится символ, однако это технически неверно. Тип данных `char` хранит символ как числовое значение на основе таблицы ASCII.

Когда переменная типа `char` определена, она может быть определена либо с помощью числа, представляющего символ, либо с помощью самого символа, как показано в следующем коде:

```
char myChar = 'A';  
char myChar = 65;
```

В предыдущем коде в обеих строках объявляется переменная с именем `myChar` типа `char`, в качестве значения которой используется заглавная буква А. Полезно иметь тип, который может содержать только один символ, но было бы более полезно, если бы мы могли хранить целые слова или предложения. Позже в этой главе мы увидим, как мы можем хранить слова или предложения, используя массив символов.

Arrays ()

Массив - это упорядоченный набор переменных одного типа. Каждая переменная в массиве называется элементом, и к этим элементам можно получить доступ по местоположению (индексу) в массиве. Когда массив определен, мы должны объявить тип переменных, которые будут в нем храниться. Существует несколько способов определения массива. В следующих примерах показаны некоторые из основных способов определения массива:

```
int myInts[10];  
int myInts[] = {1, 2, 3, 4};  
int myInts[8] = {2, 4, 6, 8, 10};
```

Каждый из этих примеров определяет массив целых чисел. В первом примере определяется неинициализированный массив из десяти целых чисел. Будьте осторожны при определении неинициализированных массивов, потому что ячейки памяти никогда не инициализируются, что может привести к очень неожиданным результатам.

Во втором примере определяется массив из четырех целых чисел, все элементы которого инициализированы значениями. Размер этого массива автоматически устанавливается в соответствии с количеством элементов в массиве инициализации.

В последнем примере определяется массив из восьми целых чисел, в котором первые пять элементов инициализируются значениями, но последние три элемента не инициализируются. Еще раз, я бы рекомендовал не определять такой массив, потому что последние три элемента не инициализированы. Через мгновение мы увидим, что происходит, когда мы пытаемся получить доступ к элементу в массиве, значение которого не было инициализировано, но сначала нам нужно увидеть, как мы будем получать доступ к элементам в массиве.

Мы получаем доступ к элементу в массиве по индексу. Мы помещаем индекс элемента, который хотим получить, между двумя квадратными скобками, как показано в следующем коде:

```
int myInts[] = {1, 2, 3, 4};  
int myInt = myInts[1];
```

В предыдущем коде мы начинаем с определения массива из четырех целых чисел и инициализируем все четыре значения. В следующей строке мы получаем элемент с индексом 1 и помещаем значение в переменную myInt.

Было бы неправильно думать, что переменная myInt содержит число 1, потому что массивы имеют нулевой индекс, что означает, что первое значение будет с индексом 0, поэтому переменная myInt содержит число 2.

Следующий код показывает, как это работает:

```
int myInts[] = {1, 2, 3, 4};  
int myInt0 = myInts[0]; // содержит 1  
int myInt1 = myInts[1]; // содержит 2  
int myInt2 = myInts[2]; // содержит 3  
int myInt3 = myInts[3]; // содержит 4
```

Этот код показывает, что когда мы объявили массив из четырех целых чисел, действительные индексы для этого массива начинаются с 0 и заканчиваются на 3. Теперь, когда мы знаем, как получить доступ к массиву, давайте посмотрим, что произойдет, когда мы получим доступ к неинициализированным элементам. Добавьте следующий код в функцию setup () скетча и запустите его:

```
int myInts[5];  
Serial.println(myInts[0]);  
Serial.println(myInts[1]);  
Serial.println(myInts[2]);  
Serial.println(myInts[3]);  
Serial.println(myInts[4]);
```

В последовательном мониторе вы увидите пять распечатанных значений, но они могут быть любым допустимым целочисленным значением, поскольку элементы никогда не инициализировались. Присвоение нового значения элементу в массиве точно так же, как присвоение значения любой переменной. Следующий код показывает это:

```
int myInts[2];  
myInts[0] = 0;  
myInts[1] = 1;
```

В предыдущем коде мы определили массив из двух целых чисел, а затем присвоили значение 0 первому элементу и значение 1 - второму элементу.

Мы также можем создавать многомерные массивы, которые в основном представляют собой массивы массивов. В следующем коде показаны два способа определения массива целых чисел 3 × 4:

```
int myInts[3][4];  
int myInts[][] = { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} };
```

К элементам многомерного массива обращаются индексы так же, как и к одномерному массиву. Следующий код показывает, как это сделать:

```
int myInt = myInts[1,2]; // The value would be 6
```

Теперь, когда мы увидели, как использовать массивы, давайте посмотрим, как мы можем использовать символьные массивы для хранения слов и предложений.

Character arrays -

Ранее в этой главе мы видели, что мы можем использовать тип символа (char) для хранения одного символа; однако что, если мы хотим хранить целые слова или предложения? Для этого мы можем использовать массив символов. Массивы символов могут быть инициализированы точно так же, как и другие массивы, как показано в следующем коде:

```
char myStr[10];  
char myStr[8] = {'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
```

Обычно массивы символов называются строками. В предыдущем коде мы определяем неинициализированную строку, которая может содержать до десяти символов, а также массив символов, содержащий слово Arduino.

Вы можете заметить, что в конце строки Arduino стоит символ \0. Этот символ представляет собой ноль. При определении строки мы всегда должны завершать строку нулевым символом, это называется нулевым завершением. Завершая строку нулевым символом, такие функции, как функция serial.println (), знают, где в памяти заканчивается строка.

Без нулевых символов эти функции будут продолжать читать память до тех пор, пока не встретится нулевой символ, что приведет к появлению большого количества мусора в консоли.

Есть более простые способы объявить строку, как показано в следующем коде:

```
char myStr[] = "Arduino";  
char myStr[10] = "Arduino";
```

В предыдущем коде первая строка инициализирует строку, содержащую слово Arduino, и размер массива автоматически изменяется с добавлением нулевого терминатора в конце. Во второй строке мы инициализируем строку, которая содержит слово Arduino и содержит дополнительный пробел.

Нулевой терминатор добавлен в конце работы Arduino.

Язык Arduino действительно содержит отдельный строковый объект; однако вы обнаружите, что символьные массивы часто используются в примере кода. Мы рассмотрим строковый объект в главе 7 «Программирование Arduino - помимо основ».

Теперь, когда мы увидели, как использовать переменные и массивы, давайте посмотрим, как определять константу.

Константа - это значение, которое никогда не меняется. В языке программирования Arduino у нас есть два способа объявления констант. Мы можем использовать ключевое слово `const` или компонент `#define`.

Компонент `#define` позволяет нам дать имя постоянному значению до компиляции приложения. Компилятор заменит все ссылки на эти константы присвоенным значением до того, как приложение будет скомпилировано. Это означает, что определенные константы не занимают места в памяти программы, что может быть преимуществом, если вы пытаетесь втиснуть большую программу в Arduino Nano.

Компонент `#define` имеет некоторые недостатки, самый большой недостаток которых заключается в том, что имя, определенное для константы, также включается в какую-либо другую константу или имя переменной, тогда это имя будет заменено значением, определенным в компоненте `#define`. По этой причине, когда я использую `#define` для определения константы, я обычно использую все заглавные буквы для имени.

В следующем коде показано, как использовать компонент `#define`. В следующем коде вы заметите, что в компоненте `#define` в конце строки нет точки с запятой. При использовании такой директивы, как `#define`, вам не нужно использовать точку с запятой:

```
#define LED_PIN 8
```

Второй способ объявить константу - использовать ключевое слово `const`. Ключевое слово `const` - это квалификатор переменной, который изменяет поведение переменной, делая ее доступной только для чтения. Это позволит нам использовать переменную точно так же, как и любую другую переменную, за исключением того, что мы не можем изменить значение переменной. Если мы попытаемся изменить значение, мы получим ошибку во время компиляции.

В следующем коде показано, как использовать ключевое слово `const`:

```
const float pi = 3.14;
```

Ключевое слово `const` обычно предпочтительнее компонента `#define`; однако с устройствами с ограниченным объемом памяти можно использовать `#define`. Теперь давайте посмотрим, как мы можем выполнять математические функции на языке программирования Arduino.

Язык программирования Arduino включает в себя операторы, которые позволяют нам вычислять сумму, разницу, произведение и частное двух операндов. Чтобы использовать эти операторы, два операнда должны быть одного типа. Это означает, например, что у нас есть возможность вычислить сумму двух целочисленных переменных; однако мы не можем вычислить сумму переменной с плавающей запятой и целочисленной переменной без преобразования одной из переменных, заставляя их быть одного типа.

Мы рассмотрим подбор немного позже в этой главе.

В следующем примере показано, как мы вычисляем сумму, разницу, произведение и частное двух переменных:

```
z = x + y; // вычисляет сумму x и y
z = x - y; // calculates the difference of x and y
z = x * y; // calculates the product of x and y
z = x / y; // calculates the quotient of x and y
```

Когда мы выполняем операцию деления, бывают случаи, когда нам нужен только остаток. Для этого мы можем использовать оператор по модулю. Если мы разделим 5 на 2, результат будет 2,5, следовательно, с учетом модуля оператор результат будет 5, так как это остаток. В следующем примере кода показано, как использовать оператор по модулю:

```
z = x % y // z будет содержать остаток от x, деленного на y
```

Язык программирования Arduino также включает составные операторы присваивания, которые позволяют нам комбинировать арифметические операции и операции присваивания переменных. Это позволяет нам выполнить арифметическую операцию и присвоить результат исходной переменной. В следующем коде показаны составные операторы на языке программирования Arduino:

```
x++; // увеличивает x на 1 и присваивает результат x
x--; // уменьшает x на 1 и присваивает результат x
x += y; // увеличивает x на y и присваивает результат x
x -= y; // уменьшает x на y и присваивает результат x
x *= y; //умножает x и y и присваивает результат x
x /= y; //делит x и y и присваивает результат x
```

Также существует множество математических функций. В следующем коде показаны некоторые из наиболее распространенных функций:

```
abs(x) // возвращает абсолютное значение x
max(x, y) // возвращает большее из двух значений
min(x, y) // возвращает меньшее из двух значений
pow(x, y) // возвращает значение x в степени y
ysq(x) // возвращает значение x в квадрате
sqrt(x) // возвращает квадратный корень из значения
```

Теперь, когда мы увидели арифметические операторы и функции, которые предоставляет язык программирования Arduino, давайте посмотрим на операторы сравнения.

Язык программирования Arduino включает операторы сравнения, которые позволяют сравнивать значения двух операндов. Операторы сравнения возвращают логическое значение, указывающее, было ли сравнение истинным или ложным. Следующий код показывает, как мы будем использовать эти операторы:

```
x == y // возвращает истину, если x равно y
x != y // возвращает истину, если x не равно y
x > y // возвращает истину, если x больше y
x < y // возвращает истину, если x меньше y
x >= y // возвращает истину, если x больше или равно y
x <= y // возвращает истину, если x меньше или равно y
```

Теперь, когда мы увидели операторы сравнения, которые предоставляет язык программирования Arduino, давайте посмотрим на логические операторы.

В язык программирования Arduino включено несколько логических операторов. Эти операторы являются операторами AND(И), OR(ИЛИ) и NOT (НЕ). Оператор NOT позволяет нам отменить операцию сравнения. Операторы AND и OR позволяют объединить несколько операторов сравнения в один шаг. Вследующем коде показано, как использовать логические операторы:

```
(x > 5 && x < 10) // истина, если x больше 5 и меньше 10
(x > 5 || x < 1) // истина, если x больше 5 или меньше 1
!(x == y) // возвращает истину, если x не равно y
```

Теперь давайте посмотрим, как мы можем преобразовать переменную.

Оператор приведения преобразует тип переменной в другой тип. Это позволит нам выполнять операции, такие как арифметические операции, с переменными разных типов. Например, если мы хотим добавить две переменные, одна из которых имеет тип float, а другая - целое число type, то нам нужно будет преобразовать одну из них, чтобы две переменные были одного типа.

Следует отметить, что когда мы приводим значение с плавающей запятой к целочисленному значению, значение усекается, а не округляется. Это означает, что если переменная с плавающей запятой содержит значение 2,9 и мы приводим ее к целому числу, значение будет 2. Помня об этом, мы обычно хотим преобразовывать целочисленные значения в значения с плавающей запятой, а не значения с плавающей запятой в целочисленные значения, даже если они означают, что операция продлится дольше.

В следующем коде показано, как можно преобразовать целочисленную переменную в переменную с плавающей запятой для выполнения арифметических вычислений:

```
int x = 5;
float y = 3.14;
float z = (float)x + y;
```

Мы можем написать очень мало полезных приложений, в которых нет какой-то логики. Эта логика обычно выполняется путем принятия решения о том, что делать, на основе некоторого ввода. Это требует, чтобы наши приложения принимали решения. Давайте посмотрим, как мы можем сделать это с помощью языка программирования Arduino.

В языке программирования Arduino мы принимаем решения с помощью оператора if. Оператор if проверяет, является ли условие истинным, и если да, то выполняет блок кода в фигурных скобках.

Ниже показан синтаксис оператора if:

```
if (condition) {
    // Code to execute
}
```

Мы можем использовать оператор else после оператора if для выполнения блока кода, если условие не истинно.

Ниже показан синтаксис оператора if / else:

```
if (condition) {  
    // Код для выполнения, если условие истинно  
} else {  
    // Код для выполнения, если условие ложно  
}
```

Условие в операторе if может быть любым логическим значением или операцией, возвращающей логический результат. Вы обнаружите, что большинство операторов if в вашем коде будут содержать операции сравнения. Давайте посмотрим на код, который это проиллюстрирует:

```
if (varA > varB) {  
    Serial.println("varA is greater than varB");  
} else {  
    Serial.println("varB is greater or equal to varA");  
}
```

В предыдущем коде мы использовали оператор сравнения «больше» (>), чтобы узнать, больше ли varA, чем varB. Если операция сравнения вернула истину, тогда код отправляет на консоль сообщение varA больше, чем varB. Если операция сравнения вернула false, тогда varB больше или равно varA, сообщение отправляется на консоль.

Мы также можем связать операторы if вместе, используя оператор if с оператором else. Следующий код иллюстрирует это:

```
if (varA == varB) {  
    Serial.println("varA is equal to varB");  
} else if (varA > varB) {  
    Serial.println("varA is greater than varB");  
} else {  
    Serial.println("varB is greater than varA");  
}
```

В предыдущем коде мы использовали оператор сравнения equal (==), чтобы увидеть, равняется ли varA varB, и если да, мы отправляем сообщение varA is equal to varB на консоль. Если они не были равны, мы использовали оператор сравнения больше (>), чтобы увидеть, больше ли varA чем varB, и если это так, мы отправляем на консоль сообщение varA больше, чем varB. Если ни одна из двух операций сравнения не увенчалась успехом, мы отправляем на консоль сообщение varA is equal to varB.

При совместном использовании операторов else и if код выполнит первый блок кода, который возвращает истинное условие, а затем проигнорирует оставшуюся часть операторов else.

Использование операторов `if` и `else` - наиболее распространенный способ выполнения логики в приложении; однако код может стать очень запутанным, если нам нужно проверить более двух или трех условий. Только представьте, есть ли в последнем примере `if / else` десять различных условий, которые нам нужно проверить. Если необходимо проверить более двух или трех условий, мы можем использовать операторы `switch / case`.

Оператор `switch / case` принимает значение, сравнивает его с несколькими возможными совпадениями и выполняет соответствующий блок кода на основе первого успешного совпадения. Оператор `switch` является альтернативой использованию нескольких операторов `else-if`, когда может быть несколько возможных совпадений. Оператор `switch` предпочтительнее операторов `else-if`, если существует три или более возможных совпадений. Оператор `switch` имеет следующий формат:

```
switch (var) {  
  case match1:  
    // Код для выполнения, если условие совпадает с  
    регистром break;  
  case match2:  
    // Код для выполнения, если условие совпадает с  
    регистром break;  
  case match3:  
    // Код для выполнения, если условие совпадает с  
    регистром break;  
  default:  
    // Код для выполнения, если условие соответствует регистру  
}
```

Предыдущий код начинается с оператора `switch`, а в скобках оператора `switch` есть переменная с именем `var`. Код будет пытаться сопоставить значение переменной `var` с каждым оператором `case`, начиная с первого, и как только он найдет `match` он выполнит код.

Код в каждом операторе `case` должен заканчиваться оператором `break`. Оператор `break` необходим, потому что, как только оператор `switch` соответствует случаю, он будет выполнять не только код в этом операторе `case`, но также код в каждом последующем операторе `case`. Это означает, что если мы не включили операторы `break` и переменная `var` соответствует значению в случае `match2`, код в случае `match2`, `case match3` и значение по умолчанию будут выполняться. Код встречает оператор `break`, он немедленно выходит из оператора `switch`, предотвращая выполнение кода в других операторах `case`.

Теперь, когда мы увидели, как принимать решения на языке программирования Arduino, давайте посмотрим, как выполнять цикл.

Looping -

Цикл `for` используется для многократного выполнения блока кода. Цикл `for` обычно используется для выполнения блока кода определенное количество раз или для доступа к элементам в массиве. Оператор `for` состоит из трех частей. Этими частями являются инициализация, условие и приращение.

В части инициализации оператора `for` мы инициализируем все переменные, которые необходимо инициализировать. Может быть несколько инициализаций, разделенных запятыми, но я бы рекомендовал избегать здесь любой инициализации, которая напрямую не связана с циклом `for`.

В части условия оператора `for` ожидается оператор, который вернет либо истинное, либо ложное значение, и обычно он содержит условный оператор. Эта часть цикла определяет, когда цикл закончится.

Пока условный оператор возвращает `true` - истину, цикл `for` продолжит выполнение блока кода. Как только условный оператор вернет `false` - ложь, цикл завершится.

Часть приращения оператора `for` используется для изменения значения переменной. Это изменение выполняется каждый раз при выполнении цикла. В следующем коде показан синтаксис оператора `for`:

```
for (initialization; condition; change) { }
```

Чтобы увидеть, как это будет выглядеть с реальным кодом, ниже показано, как мы могли бы создать оператор `for`, который будет повторяться десять раз:

```
for (int i = 0; i < 10; i++) {  
    // Code to execute  
}
```

В предыдущем коде оператор `for` инициализирует переменную `i` нулем в части инициализации. В части условия операторы `for` проверяют, меньше ли значение переменной `i` десяти, и если это так, код продолжит цикл. В части изменения цикл `for` увеличивает переменную `i` на единицу при каждом выполнении цикла. В этом примере цикл `for` сначала присваивает значение 0 переменной `i`, а затем увеличивает его в каждом цикле, пока переменная `i` не станет равной 9.

Следующий цикл, который мы рассмотрим, - это цикл `while`. Цикл `while` будет многократно выполнять блок кода до тех пор, пока условие, определенное в операторе `while`, не вернет `false`. Это может быть опасным циклом, потому что, если условие никогда не возвращает `false`, цикл будет продолжаться бесконечно. Оператор `while` имеет следующий синтаксис:

```
while (condition) {  
    // код для выполнения  
}
```

Условие в операторе `while` должно возвращать либо `true` (истина), либо `false` (ложь). Это условие обычно является сравнением. В следующем коде показан пример оператора `while`:

```
int x = 0;  
while (x < 200) {  
    // code to execute  
    x++;  
}
```

В предыдущем коде блок кода выполняется, пока переменная `x` меньше 200. В конце блока кода переменная `x` увеличивается на 1. Если бы мы забыли поместить строку, увеличивающую `x` в коде `block`, тогда цикл `while` будет повторяться бесконечно.

Очень важно убедиться, что вы поместили оператор изменения в блок кода, иначе цикл никогда не завершится.

В цикле `while` условие проверяется до выполнения блока кода.

Это означает, что если условия возвращают `false` при первом вызове оператора `while`, блок кода никогда не будет выполнен. Если мы требуем, чтобы блок кода выполнялся один раз перед проверкой условия, мы можем использовать цикл `do / while`.

Цикл `do / while` точно такой же, как цикл `while`, за исключением того, что условие проверяется после выполнения блока кода, а не до него. В следующем коде показан синтаксис этого цикла:

```
do {  
    // code to execute  
} while (condition);
```

Как и в случае с циклом `while`, условие в операторе `while` должно возвращать либо `true` - истину, либо `false` -ложь и обычно является оператором сравнения. Следующие коды показывают примероператора `do / while`:

```
int x = 0;  
do {  
    // код для выполнения
```

```
x++;  
} while (x < 200);
```

Предыдущий код выполнит блок кода 200 раз, точно так же, как код в предыдущем цикле while. Единственная разница в том, что в цикле while условие проверяется до выполнения блока кода, а в цикле do / while условие проверяется после.

Функция - это именованный блок кода, который выполняет определенную задачу. Когда создается новый скетч, IDE или веб-редактор автоматически создает для нас две функции, как мы видели в предыдущей главе; однако мы не ограничены только этими двумя функциями, у нас также есть возможность самостоятельно объявлять пользовательские функции.

В следующем коде показан синтаксис для создания функции:

```
type name (parameters) { }
```

Чтобы объявить функцию, нам нужно объявить, к какому типу относится функция. Тип функции - это значение, возвращаемое функцией. Если функция не собирается возвращать значение, как в случае с функциями setup () и loop (), тогда тип функции будет недействительным.

После объявления типа функции мы определяем имя функции. Имя функции должно описывать то, что она делает. Например, если мы создаем эскиз, который будет включать или выключать светодиод, тогда у нас могут быть функции с именами ledOff () и ledOn (). Хорошей практикой является использование верблюжьего регистра при именовании функций как переменных.

После имени функции мы помещаем параметры функции в круглые скобки. Параметры - это данные, которые передаются функции вызывающим ее кодом. Функция обычно полагается на данные для выполнения необходимой логики. Вы можете указать несколько параметров в круглых скобках, разделив их запятыми.

Мы используем фигурные скобки, чтобы определить начало и конец блока кода для функции. Левая фигурная скобка указывает на начало функции, а правая фигурная скобка указывает на конец функции.

В следующих примерах показаны различные примеры функций:

```
void myFunction() {  
    // Function code  
}  
  
void myFunction(int param)  
{  
    // Function code  
}
```



```
int myFunction() {  
    // Function code  
}  
  
int myFunction(int param) {  
    // Function Code  
}
```

Первая функция имеет тип возвращаемого значения `void`, что означает, что она не возвращает никакого значения. У него также нет никаких параметров. Этот тип функции будет использоваться для выполнения задачи, для которой не требуется возвращать какую-либо информацию обратно в код, который ее вызвал, и не требуется никакой дополнительной информации для выполнения требуемой задачи.

Вторая функция также имеет тип возвращаемого значения `void`, но принимает один параметр. Этот тип функции может использоваться, если функции требуется некоторая информация из кода, который ее вызвал, для выполнения своей задачи. Первая часть параметра - это тип параметра. В этом примере типом является `int`, что означает, что данные будут целочисленного типа. Вторая часть параметра - это имя параметра. Это будет означать, что параметр в этом примере называется `param` и имеет целочисленный тип. Чтобы объявить несколько параметров, мы должны разделить их запятыми следующим образом: `(int param1, int param2, float param3)`.

Третья функция имеет тип возврата `int`, что означает, что она должна возвращать целое число; однако он не принимает никаких параметров. Этот тип функции может использоваться, если мы хотим передать информацию от функции обратно в код, который ее вызвал.

Четвертая функция возвращает целое число и принимает параметр. Этот тип функции мог бы использоваться, если бы мы хотели передать информацию обратно в код, который ее вызвал, и ему нужна информация из этого кода для выполнения своей задачи.

Мы используем оператор `return` для возврата значения из функции. В следующем коде показано, как это сделать:

```
int myFunction() {  
    var x = 1;  
    var y = 2;  
    return x + y;  
}
```

Когда переменная создается внутри функции, как мы видели в последнем примере, переменная доступна только внутри этой функции. Следующий код иллюстрирует это:

```
int g = 1;  
void function myFunction1() {  
    int x1 = 2;
```

```
}  
void function myFunction2() {  
    int x2 = 3;  
}
```

В предыдущем коде переменная `g`, поскольку она объявлена вне функций, доступна для любой из функций. Когда вы объявляете переменную вне функций, она считается глобальной переменной. Переменная `x1` доступна только в функции `myFunction1 ()`, а переменная `x2` доступна только в функции `myFunction2 ()`.

В этой главе мы рассмотрели основы языка программирования Arduino. Материал в этой главе закладывает основу для всего остального, что описано в этой книге, поэтому важно понять элементы, представленные здесь.

В следующей главе мы рассмотрим некоторые более продвинутые функции языка программирования Arduino и среды разработки Arduino.

5

Программирование Arduino - помимо основ

Одна из вещей, которую я усвоил в начале своей карьеры разработчика, это то, что я могу писать довольно удивительные приложения, даже если я знаю только основы языка программирования, который использую; однако это обычно затрудняет обслуживание и чтение кода, а также увеличивает время разработки проекта. Я всегда говорю людям, которые изучают язык, не торопитесь, чтобы понять некоторые из более сложных функций языка, который они изучают, прежде чем использовать его в серьезных проектах.

В этой главе мы узнаем:

- Как установить режим вывода на цифровом выводе Arduino
- Как получить и установить значения цифрового вывода Arduino
- Как получить и установить значения аналогового вывода Arduino
- Как использовать структуры и союзы
- Как использовать дополнительные вкладки
- Как использовать классы и объекты

В предыдущей главе мы рассмотрели основы языка программирования Arduino. В этой главе мы собираемся выйти за рамки самого языка. Мы начнем с того, что посмотрим, как мы можем взаимодействовать с цифровыми выводами на Arduino.



В качестве примеров в этой главе мы будем использовать прототип, который мы создали в конце главы 4 «Базовое прототипирование».

В главе 1, Arduino, мы увидели, что Arduino имеет несколько цифровых выводов, к которым мы можем подключать внешние датчики и другие устройства. Прежде чем использовать эти выводы, мы должны настроить их либо для ввода, либо для вывода, в зависимости от того, для чего мы их используем. Для этого мы используем функцию `pinMode()`, встроенную в язык программирования Arduino. Обычно для небольших скетчей мы вызываем функцию `pinMode()` внутри функции `setup()`; однако это не требуется. В следующем коде показан синтаксис функции `pinMode()`:

```
pinMode(pin, mode);
```

Эта функция вызывается с двумя параметрами. Первый - это номер устанавливаемого вывода, а второй - это режим вывода. Режим вывода может быть либо `INPUT`, чтобы считывать значение с вывода (внешний датчик записывает значение на вывод), либо `OUTPUT`, чтобы установить значение для вывода. В следующем коде показано, как использовать эту команду для установки режима вывода для двух выводов:

```
pinMode( 11 , INPUT);  
pinMode( 12 , OUTPUT);
```

В предыдущем коде мы установили вывод 11 на вход и вывод 12 на вывод. Следовательно, мы будем записывать значения на вывод 11 и считывать значения с вывода 12.

Рекомендуется никогда не использовать сами номера выводов, как показано в последнем примере, для доступа к выводам на Arduino. Вместо использования таких номеров выводов, мы должны установить переменную или константу с номером вывода, а затем использовать эту переменную или константу при доступе к выводу. Это предотвратит ввод неправильного числа в коде.



Лично я предпочитаю использовать `#define` для определения номеров выводов, которые я использую, если номер вывода не изменится. Это позволяет мне отделить определения выводов от других констант в моем скетче.

Если вы хотите использовать константы вместо `#define`, это вполне приемлемо, и даже может быть предпочтительнее.

Следующий код показывает, как мы должны использовать функцию `pin Mode()` в скетче:

```
#define BUTTON_ONE 12  
#define LED_ONE 11  
  
void setup() {  
    pinMode(BUTTON_ONE, INPUT);  
    pinMode(LED_ONE, OUTPUT);  
}
```

В предыдущем коде мы определили константы, представляющие два вывода. Первая строка определяет `BUTTON_ONE` на номер (вывод) 12, а вторая строка определяет `LED_ONE` на номер (вывод) 11. Затем мы устанавливаем вывод `BUTTON_ONE` в режим ввода, а вывод `LED_ONE` в режим вывода. в функции `setup()`.

Функцию `pinMode()` также можно использовать для настройки внутреннего подтягивающего резистора, установив режим вывода на `INPUT_PULLUP`. Это изменит поведение вывода, когда он находится в режиме ввода.

Эти цифровые выводы могут иметь одно из двух значений: `HIGH` или `LOW`. Давайте посмотрим, как мы можем установить значение цифрового пина.

Чтобы установить значение цифрового вывода на языке программирования Arduino, мы используем функцию `digitalWrite()`. Эта функция имеет следующий синтаксис:

```
digitalWrite(pin, value);
```

Функция `digitalWrite()` принимает два параметра, где первый - это номер вывода, а второй - значение, которое нужно установить. Мы должны использовать `HIGH` или `LOW` при установке значения цифрового вывода.

Следующий код показывает, как это сделать:

```
digitalWrite(LED_ONE, HIGH);  
delay(500);  
digitalWrite(LED_ONE, LOW);  
delay(500);
```

В предыдущем коде мы устанавливаем вывод, определенный константой `LED_ONE`, в 1, а затем делаем паузу на полсекунды. Функция `delay()` в языке программирования Arduino приостанавливает выполнение скетча на определенное время. Время для этой функции в миллисекундах. После функции `delay()` мы затем устанавливаем вывод, определенный константой `LED_ONE`, в 0 и ждем еще полсекунды, прежде чем вернуться к началу цикла.

Предыдущий код можно использовать в функции `loop()` для мигания светодиода; однако перед этим нам нужно определить константу `LED_ONE`, а также установить режим вывода. Давайте посмотрим на полный эскиз, необходимый для мигания светодиода.

```
#define LED_ONE 11  
  
void setup() {  
    pinMode(LED_ONE, OUTPUT);  
}
```

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

Этот код начинается с определения константы LED_ONE и установки значения 11. Режим вывода для вывода LED_ONE затем устанавливается в функции setup (). Наконец, к функции loop () добавляется код, который заставит мигать светодиод. Если вы подключите прототип, который мы разработали в главе 2, «Базовое прототипирование», и запустите этот код, вы должны увидеть, как мигает один из светодиодов.

Теперь, когда мы знаем, как писать на цифровой вывод, давайте посмотрим, как мы можем прочитать значение одного вывода.

Чтобы прочитать значение цифрового вывода на языке программирования Arduino, мы используем функцию digitalRead (). Эта функция имеет следующий синтаксис:

```
digitalRead(pin);
```

Функция digitalRead () принимает один параметр, который представляет собой номер считываемого цифрового вывода, и возвращает целочисленное значение. Следующий код показывает, как мы можем использовать функцию digitalRead () для чтения одного из цифровых выводов Arduino:

```
int val = digitalRead(BUTTON_ONE);
```

С помощью этого кода функция digitalRead () вернет значение вывода, определенное константой BUTTON_ONE, и поместит это значение в переменную с именем val. Переменная val определяется как целое число. Однако функция digitalRead () вернет только 0 или 1. Мы можем использовать те же самые константы HIGH и LOW, которые мы видели в разделе цифровой записи, чтобы увидеть, высокий или низкий уровень вывода. Использование этих констант является предпочтительным и делает ваш код более читабельным.

Теперь давайте посмотрим, как мы можем использовать функцию digitalRead () для чтения статуса кнопки. Следующий код будет считывать статус кнопки из прототипа, который мы создали в главе 4, Базовое прототипирование:

```
#define BUTTON_ONE 12  
  
void setup() {  
    Serial.begin(9600);
```

```
pinMode(BUTTON_ONE, INPUT);  
}  
  
void loop() {  
  int val = digitalRead(BUTTON_ONE);  
  if (val == HIGH) {  
    Serial.println("Button HIGH");  
  } else {  
    Serial.println("Button LOW");  
  }  
}
```

Этот код начинается с определения константы `BUTTON_ONE` и установки для нее значения 12. Монитор последовательного порта и режим вывода для вывода, к которому подключена кнопка, настраиваются в функции `setup()`. Внутри кнопки цикла функция `digitalRead()` используется для чтения вывода, а оператор `if` используется для сравнения возвращенного значения с константой `HIGH`. Если они равны, то на последовательный монитор отправляется сообщение `Button HIGH`, в противном случае отправляется сообщение `Button LOW`.

Если этот код запущен на прототипе, который был создан в главе 4 «Базовое прототипирование», то вы должны увидеть одно из двух сообщений, выводимых на последовательный монитор, в зависимости от того, нажата кнопка или нет.

Теперь давайте посмотрим, как мы можем писать на аналоговый вывод на Arduino.

Аналоговые значения записываются в Arduino с помощью выводов ШИМ (PWM). В главе 1, Arduino, мы рассмотрели, что такое ШИМ и как она работает. На большинстве плат Arduino выводы шим сконфигурированы для контактов 3, 5, 6, 9, 10 и 11; тем не менее, Arduino Mega имеет значительно больше выводов, доступных для работы с ШИМ.

Для выполнения аналоговой записи мы используем функцию `analogWrite()`, которая принимает следующий синтаксис:

```
analogWrite(pin, value);
```

Функция `analogWrite()` принимает два параметра, где первый - это номер контакта, а второй - значение, которое необходимо установить. Значение функции `analogWrite()` может находиться в диапазоне от 0 до 255.

Давайте посмотрим на образец скетча, чтобы увидеть, как мы можем использовать функцию `analogWrite ()` для постепенного появления и исчезновения свечения светодиода:

```
#define LED_ONE 11

int val = 0;
int change = 5;

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

void loop()
{
    val += change;
    if (val > 250 || val < 5) {
        change *= -1;
    }
    analogWrite(LED_ONE, val);
    delay(100);
}
```

Этот код начинается с определения константы `LED_ONE` со значением 11. Это будет вывод, к которому подключен светодиод. Также определены две глобальные переменные целочисленного типа с именами `val` и `change`. Целое число `val` будет хранить текущее значение аналогового вывода, а целое число `change` будет хранить, насколько целое число `val` должно изменять каждый цикл.

Вывод, определенный константой `LED_ONE`, устанавливается в режим вывода в функции `setup ()`. Это позволит нам записать на вывод и изменить яркость светодиода, подключенного к выводу.

Функция `loop ()` начинается с добавления переменной `change` к переменной `val`, а результат сохраняется в переменной `val`. Если значение переменной `val` больше 250 или меньше 5, мы умножаем переменную `change` на -1. Это заставляет переменную `change` вращаться между 5 и -5, что приводит к увеличению или уменьшению переменной `val` в каждом цикле. Наконец, значение переменной `val` записывается на вывод, определенный константой `LED_ONE`, и затем происходит небольшая задержка перед обратным циклом.

Если этот код запускается на прототипе, который был создан в главе 2, «Базовое прототипирование», вы должны увидеть, как светодиод постепенно загорается и гаснет. Теперь давайте посмотрим, как мы можем прочитать аналоговый вывод.

Мы считываем значение с аналогового вывода с помощью функции `analogRead()`. Эта функция вернет значение от 0 до 1023. Это означает, что если датчик возвращает полное напряжение 5 В, то функция `analogRead()` вернет значение 1023, что приведет к значению 0,0049 В на единицу (мы будем использовать это номер в образце кода). В следующем коде показан синтаксис функции `analogRead()`:

```
analogRead(pin);
```

Функция `analogRead()` принимает один параметр, который является выводом для чтения. Следующий код использует функцию `analogRead()` с датчиком температуры `tpr36` для определения текущей температуры:

```
#define TEMP_PIN 5

void setup() {
  Serial.begin(9600);
}

void loop() {
  int pinValue = analogRead(TEMP_PIN);
  double voltage = pinValue * 0.0049;
  double tempC = (voltage - .5) * 100.0;
  double tempF = (tempC * 1.8) + 32;
  Serial.print(tempC);
  Serial.print(" - ");
  Serial.println(tempF);
  delay(2000);
}
```

Предыдущий код начинается с определения вывода, к которому подключен датчик температуры, аналогового вывода 5. Функция `setup()` настраивает монитор последовательного порта, чтобы приложение могло распечатать на нем температуру.

Функция `loop()` начинается с чтения аналогового вывода и сохранения значения в переменной значения вывода. Чтобы преобразовать это значение в фактическое напряжение, мы умножаем его на значение 0,0049 В, которое мы видели ранее в этом разделе. Если мы посмотрим на техническое описание датчика температуры `tpr36`, мы определим, что $(\text{напряжение} - 0,5) * 100,0$ является соответствующей формулой для расчета температуры в градусах Цельсия. Затем мы можем использовать стандартную формулу $(\text{температура Цельсия} * 1,8) + 32$, чтобы определить температуру в градусах Фаренгейта. Наконец, мы выводим эти значения на монитор последовательного порта и задерживаем на две секунды перед повторным запуском цикла.

We will be using the `digitalRead()`, `digitalWrite()`, `analogRead()` and `analogWrite()` functions a lot in this book so you will be getting familiar with them.

Now let's look at structures.

Структура - это определяемый пользователем составной тип данных, который используется для группировки нескольких переменных вместе. Переменные в структуре могут быть разных типов, что позволяет нам хранить связанные данные разных типов вместе. В следующем коде показан синтаксис определения структуры:

```
struct name {  
    variable list  
    .  
    .  
};
```

Когда структура определена, используется ключевое слово `struct`, за которым следует имя структуры. Список переменных затем определяется между фигурными скобками. Давайте посмотрим, как мы можем создать и использовать структуру, изменив предыдущий скетч, который использовал функцию `analogRead()` для чтения температуры TMP36, чтобы использовать структуру.

Первое, что нам нужно сделать, это определить структуру, которая будет хранить информацию о температуре от датчика. Мы назовем эту структуру `tmp36_reading`, и она будет содержать три переменные типа `double`. Следующий код показывает, как определить эту структуру:

```
struct tmp36_reading {  
    double voltage;  
    double tempC;  
    double tempF;  
};
```

В предыдущем коде определяется структура с именем `tmp36_reading`, которая содержит три переменные типа `double`. Имейте в виду, что переменные в структуре не обязательно должны быть одного типа, просто выяснилось, что все отдельные переменные в этой структуре были двойного типа.

В следующем коде показано, как создать переменную типа `tmp36_reading`:

```
struct tmp36_reading temp;
```

В предыдущем коде создается переменная с именем `temp` типа `tmp36_reading`. Затем мы можем присвоить или получить значения, используя синтаксис с точкой, как показано в следующем коде:

```
temp.voltage = pinValue * 0.0049;
temp.tempC = (temp.voltage - .5) * 100.0;
temp.tempF = (temp.tempC * 1.8) + 32;
```

В предыдущем коде мы присваиваем значения переменным напряжения, `tempC` и `tempF` структуры `tmp36_reading`. Теперь давайте посмотрим, как мы можем интегрировать этот код в эскиз, который считывает датчик температуры TMP36. Ниже приведен полный код нового скетча:

```
#define TEMP_PIN 5

struct tmp36_reading {
    double voltage;
    double tempC;
    double tempF;
};

void setup() {
    Serial.begin(9600);
}

void loop() {
    struct tmp36_reading temp;
    int pinValue = analogRead(TEMP_PIN);
    temp.voltage = pinValue * 0.0049;
    temp.tempC = (temp.voltage - .5) * 100.0;
    temp.tempF = (temp.tempC * 1.8) + 32;

    showTemp(temp);
    delay(2000);
}

void showTemp(struct tmp36_reading temp) {
    Serial.print(temp.tempC);
    Serial.print(" - ");
    Serial.println(temp.tempF);
}
```

Этот скетч работает точно так же, как предыдущий, который считывает датчик температуры TMP36, за исключением того, что теперь мы используем структуру для хранения значений с датчика, а не переменных.

Если у вас есть несколько значений, которые можно сгруппировать таким образом, рекомендуется использовать структуру, а не переменные, потому что все значения сгруппированы в одну структуру.

Теперь давайте посмотрим на другой специальный тип данных, который может быть похож на структуру; однако функциональность существенно отличается.

()

Союзы - это особый тип данных, который позволяет нам хранить различные типы данных в одном определении, аналогичном структуре; однако только один из членов может содержать данные одновременно. Ниже показан синтаксис определения объединения:

```
union name {  
    variable list  
    .  
    .  
};
```

Если синтаксис очень похож на синтаксис структуры. Фактически, это тот же синтаксис, за исключением ключевых слов `struct` / `union`.

Посмотрим, как мы будем использовать союз. Следующий код определяет новое объединение:

```
union some_data {  
    int i;  
    double d;  
    char s[20];  
};
```

Предыдущий код определяет объединение с именем `some_data`, которое может содержать целое число, двойное число или строку символов. Ключевое слово в последнем предложении - `или`. В отличие от структуры, которая может хранить несколько разных значений, объединение может хранить только одно значение за раз. Следующий код проиллюстрирует это:

```
union some_data {  
    int i;  
    double d;  
    char s[20];  
};  
  
void setup() {  
    Serial.begin(9600);  
    union some_data myData;  
    myData.i = 42;
```

```
myData.d = 3.14;
strcpy( myData.s, "Arduino");
Serial.println(myData.s);
Serial.println(myData.d);
Serial.println(myData.i);
}
```

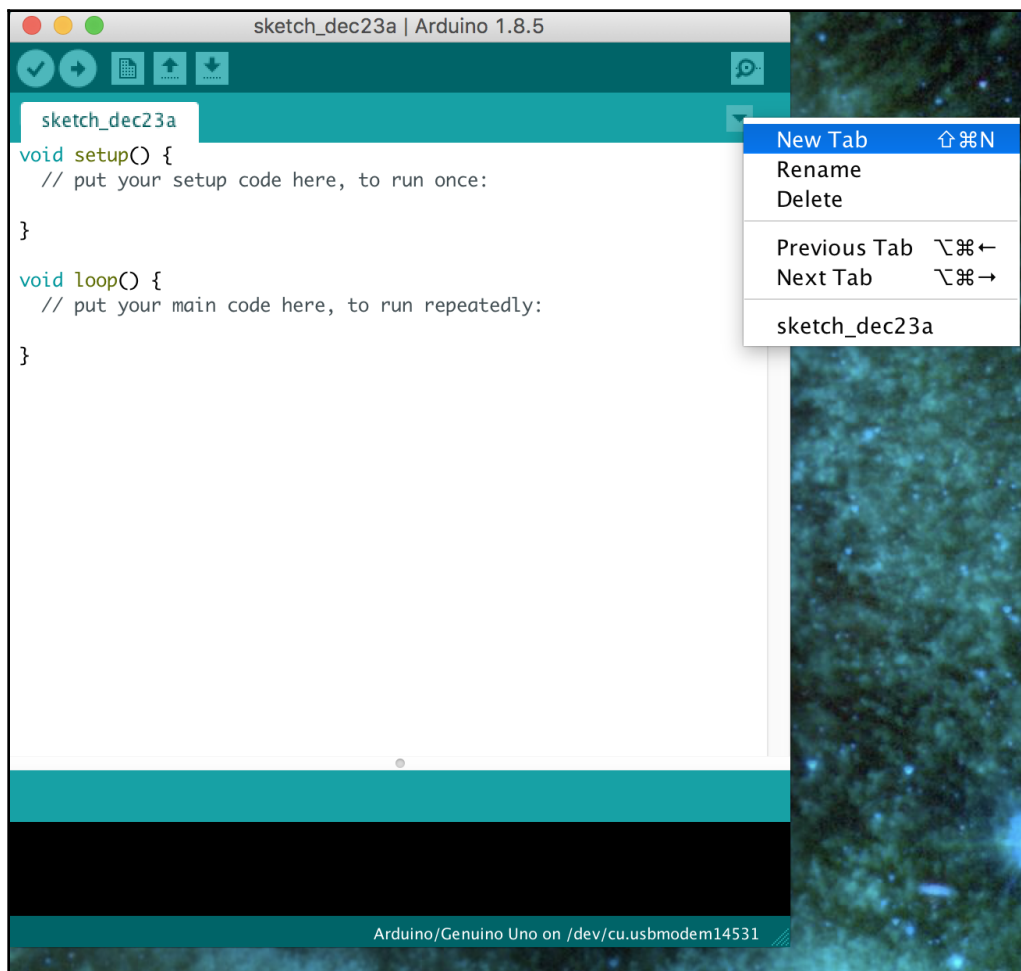
В предыдущем коде мы определяем объединение с именем `some_data`. Затем в функции `setup()` мы создаем экземпляр объединенного типа `some_data` с именем `myData`. Затем мы присваиваем значения каждому члену типа объединения. Целочисленный член установлен на 42, двойной член установлен на 3,14, а символьная строка установлена на `Arduino`. Когда этот код будет запущен, мы увидим, что символьная строка `Arduino` правильно напечатана на последовательном мониторе; однако при выводе на монитор последовательного порта целочисленных и двойных элементов информация неверна.

В предыдущем примере, когда член `some_data.i` имеет значение 42, объединение `some_data` будет содержать целое число 42. Затем, когда мы устанавливаем член `some_data.d` равным 3,14, целочисленное значение 42 перезаписывается, и теперь объединение `some_data` будет содержать 3.14. Наконец, когда мы устанавливаем член `some_data.s` на `Arduino`, он перезаписывает член `some_data.d`, поэтому объединение `some_data` теперь содержит строку `Arduino`.

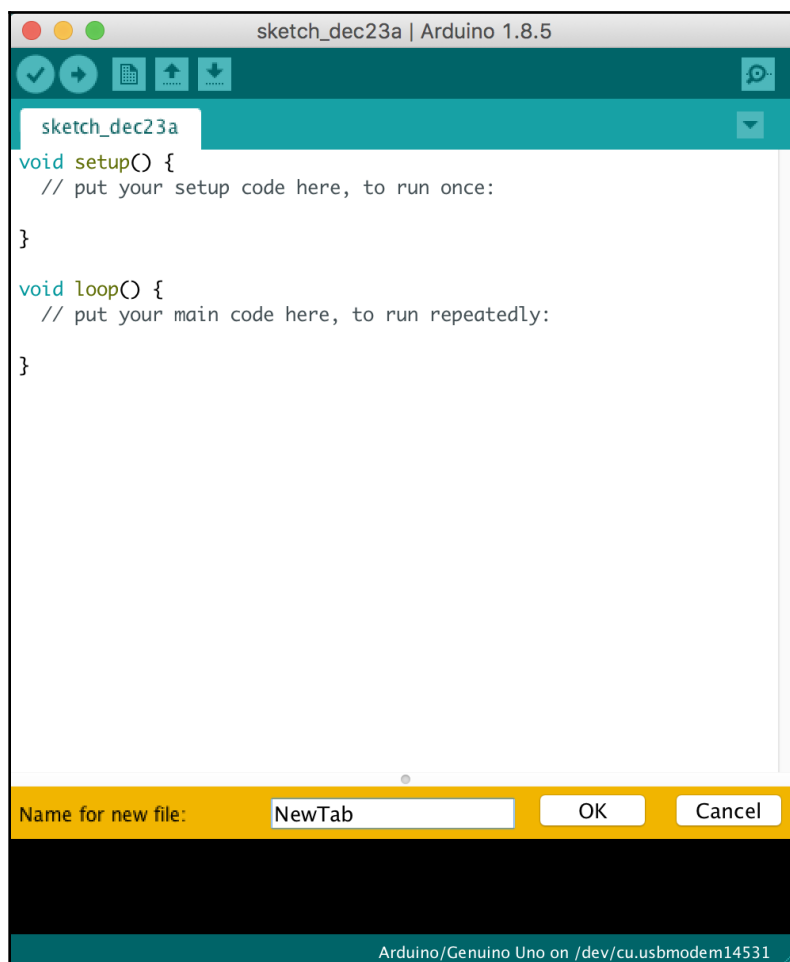
Прежде чем мы рассмотрим другие функции языка программирования Arduino, давайте рассмотрим еще одну функцию Arduino IDE и веб-редактора.

Когда вы начинаете работать с более крупными и сложными проектами, очень важным становится разделить код на отдельные рабочие области, поскольку это упрощает управление кодом. Для этого как в Arduino IDE, так и в веб-редакторе мы можем добавлять новые вкладки в эскиз.

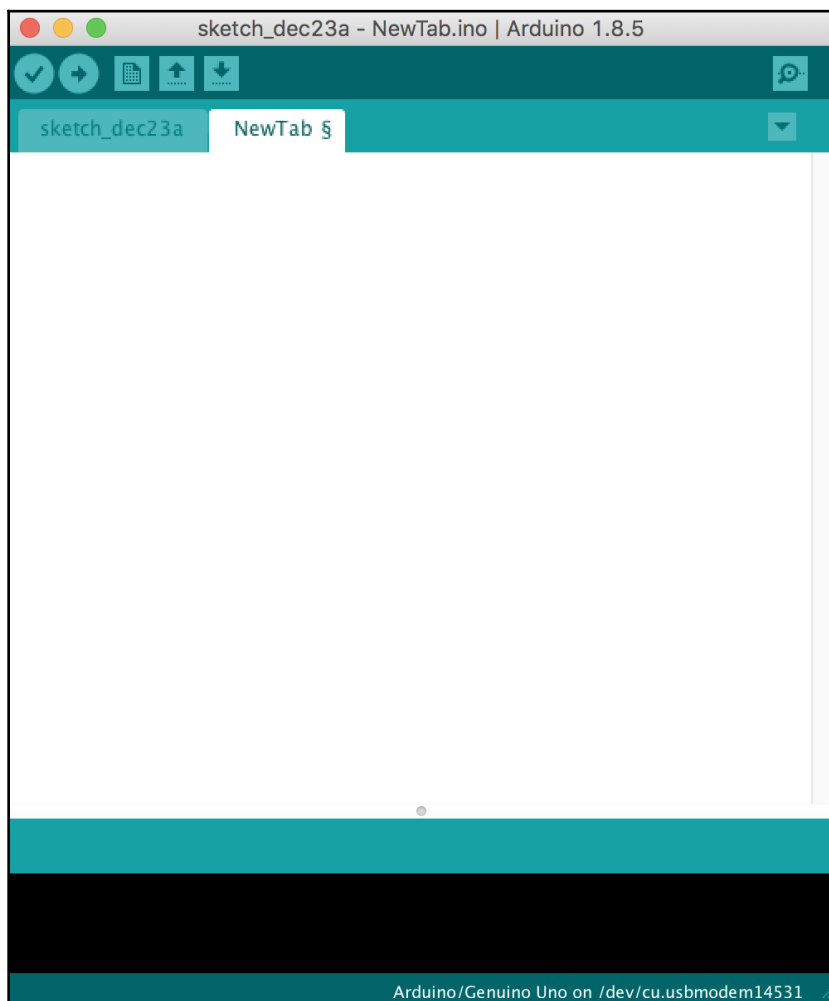
Чтобы добавить новую вкладку в IDE Arduino, нажмите кнопку с перевернутым треугольником в ней, которая расположена в верхней правой части окна IDE, как показано на следующем снимке экрана:



Во всплывающем окне нажмите кнопку «Новая вкладка», и вы увидите оранжевую полосу под разделом кода в окнах IDE Arduino. В этой оранжевой полосе вы можете назвать новую вкладку, а затем нажать кнопку ОК, чтобы создать вкладку. На следующем снимке экрана показано, как назвать новую вкладку:



После того, как вы нажмете ОК, будет создана новая вкладка с именем, которое вы ей дали, как показано на следующем снимке экрана:



Мы можем создать новую вкладку в веб-редакторе точно так же, как в Arduino IDE. В веб-редакторе есть аналогичная кнопка с перевернутым треугольником. Когда эта кнопка будет нажата, появится меню, и вы сможете выбрать опцию **New Tab**. После того как вы присвоите имя новой вкладке, она появится в веб-редакторе.

Прежде чем мы начнем добавлять вкладки в наши проекты, нам нужно иметь план того, как мы хотим разделить код. Я считаю, что для больших проектов рекомендуется использовать только функции `setup ()` и `loop ()` на главной вкладке. Затем я создаю вкладку для каждой функциональной области проекта.

Например, если бы я сделал метеостанцию с обоими датчиками температуры и дождя, у меня была бы основная вкладка с функциями `setup ()` и `loop ()`, а затем две дополнительные вкладки; один для функции датчика температуры и один для функции датчика дождя.

Помимо использования дополнительных вкладок для кода, для более крупных проектов и библиотек рекомендуется иметь вкладки, определяющие константы, которые необходимо использовать на нескольких вкладках. Эти константы обычно помещаются в файлы заголовков. Заголовочный файл должен иметь расширение `.h`.

Теперь посмотрим, как работать с вкладками.

При создании новой вкладки первое, что нам нужно решить, - это то, что будет происходить на вкладке. Например, в этом разделе мы создадим две новые вкладки. Один будет называться `led.h`, а другой `led`. Файл `led.h` будет содержать определение константы, а светодиодный файл будет содержать код.

Когда мы создаем вкладку с расширением `.h`, который мы создаем, то, что известно в C language, является заголовочным файлом. Заголовочный файл - это файл, содержащий объявления и макроопределения. Затем эти вкладки можно включить в обычные вкладки кода. В следующем разделе мы увидим другой тип вкладки - вкладка `cpp`.

После создания новых вкладок добавьте на вкладку `led.h` следующий код:

```
#ifndef LED_H
#define LED_H

#define LED_ONE 3
#define LED_TWO 11
#endif
```

Этот код будет определять две константы, которые представляют собой номера заголовков выводов для двух светодиодов на прототипе, который мы построили в главе 4, «Базовое прототипирование». `#ifndef` и `#endif` гарантируют, что файл заголовка импортируется только один раз на любой вкладке. `#ifndef` проверяет, определена ли константа `LED_H`, а если нет, то включает код между `#ifndef` и `#endif`.

Теперь во вкладке светодиода добавьте следующий код:

```
void blink_led(int led) {  
    digitalWrite(led, HIGH);  
    delay(500);  
    digitalWrite(led, LOW);  
    delay(500);  
}
```

Функция `blink_led ()` содержит единственный параметр, который будет выводом светодиода, который мы хотим мигать. Сама функция включит светодиод на 1/2 секунды, а затем погаснет. Теперь на главной вкладке нам нужно будет включить оператор `#include` в верхней части вкладки, чтобы включить файл заголовка `led.h`. Следующий код показывает, как это сделать:

```
#include "led.h"
```

Оператор `#include` берет файл заголовка и включает его во вкладку, что позволяет нам использовать определения в нашем коде. Если бы мы попытались использовать одну из констант в нашем коде, но забыли включить `#include`, мы получили бы ошибку о том, что константа не была объявлена в этой области видимости, что означает, что компилятор не смог найти объявления для константы.

Если мы добавляем заголовочный файл из скетча, с которым мы работаем, имя заголовочного файла заключено в двойные кавычки. Если мы включим файл заголовка из отдельной библиотеки, то имя будет окружено знаками «меньше» и «больше». Мы увидим это позже в этой книге, поскольку будем использовать сторонние библиотеки.

В функции `loop ()` мы хотим вызвать функцию `blink_led ()` из вкладки `led`. Следует отметить, что нам нужно только включить оператор `#include` для файла заголовка, а не для вкладки, содержащей код. Ниже показан код основной вкладки:

```
#include "led.h"  
void setup() {  
    // put your setup code here, to run once:  
    pinMode(LED_ONE, OUTPUT);  
    pinMode(LED_TWO, OUTPUT);  
}
```

```
}

void loop() {
  // put your main code here, to run repeatedly:
  blink_led(LED_ONE);
  delay(1000);
  blink_led(LED_TWO);
}
```

Теперь, если вы подключите прототип, который мы создали в главе 2 «Основы прототипирования», вы увидите, что светодиоды мигают один за другим. Разделение кода между отдельными вкладками - отличный способ организовать его при работе с более крупными проектами. Это значительно упрощает обслуживание и организацию вашего кода. Классы обычно используются при создании библиотек для Arduino. Хотя создание библиотек выходит за рамки этой книги, полезно знать, что такое классы и как их использовать, поскольку мы будем использовать библиотеки в определенных разделах этой книги.

-

-

()

- это парадигма программирования, которая помогает нам разделить наш код на повторно используемые компоненты с использованием классов и объектов. Объект предназначен для того, чтобы что-то моделировать. Например, мы можем создать светодиодный объект, который будет содержать свойства и функциональность, которые мы хотим для светодиода; однако, прежде чем мы сможем создать объект, нам нужно иметь для него план. Этот план называется классом. Давайте посмотрим, как это работает, создав классы, которые помогут нам управлять светодиодом.

Мы начнем с создания двух новых вкладок с именами led.cpp и led.h. Файл led.h будет содержать определение класса, а файл led.cpp будет содержать код. Начнем с добавления следующего кода в файл led.h:

```
#ifndef LED_H
#define LED_H

#define LED_ONE 3
#define LED_TWO 11

class Led{
  int ledPin;
  long onTime;
  long offTime;
public:
  Led(int pin, long on, long off);
```

```
void blinkLed();  
void turnOn();  
void turnOff();  
};  
  
#endif
```

Код аналогичен файлу led.h в разделе работы с вкладками, за исключением того, что добавлено определение класса Led. Определение класса Led определяет три свойства (переменные) для класса: ledPin, onTime и offTime. До этого примера все используемые нами переменные были либо глобальными, либо определены в функции. Свойства класса - это переменные, которые определены внутри класса и обычно что-то определяют в объекте. В этом примере свойство ledPin определяет, к какому выводу подключен светодиод; свойство onTime определяет количество времени, в течение которого светодиод остается включенным, а свойство offTime определяет, как долго светодиод остается выключенным.

После свойств определяется конструктор класса. Конструктор используется для создания экземпляра класса, и мы увидим, как его использовать позже в этом разделе.

После конструктора три метода (функции) для класса. Метод класса - это просто функция, которая является частью класса и обычно определяет функциональность объекта.

Если вкладка led.h содержит определение класса Led, вкладка led.cpp содержит код этого класса. Добавим на вкладку led.cpp следующий код:

```
#include "led.h"  
#include "Arduino.h"  
  
Led::Led(int pin, long on, long off) {  
    ledPin = pin;  
    pinMode(ledPin, OUTPUT);  
    onTime = on;  
    offTime = off;  
}  
  
void Led::turnOn() {  
    digitalWrite(ledPin, HIGH);  
}  
  
void Led::turnOff(){  
    digitalWrite(ledPin, LOW);  
}  
  
void Led::blinkLed() {  
    this->turnOn();  
    delay(onTime);  
    this->turnOff();  
}
```

```
    delay(offTime);  
}
```

Этот код начинается с импорта двух файлов заголовков. Первый файл заголовка - это только что созданный файл `led.h`, а второй - файл заголовка `Arduino.h`. Заголовочный файл `Arduino.h` содержит определения всех пользовательских функций Arduino. Он автоматически добавляется на главную вкладку; однако, если вы хотите использовать пользовательские функции Arduino на других вкладках, как это необходимо здесь, нам необходимо импортировать этот файл.

После импорта следует реализация конструктора для класса `Led`, который был определен на вкладке `led.h`.

Когда мы реализуем конструктор или метод для класса, мы ставим перед его именем префикс имени класса, за которым следует два двоеточия (`::`). Имя конструктора класса должно совпадать с именем класса. Следовательно, реализация конструктора - `Led :: Led`. В конструкторе мы устанавливаем свойства класса и режим вывода для вывода, к которому также подключен светодиод.

Следующие два метода класса, `Led :: turnOn` и `Led :: turnOff`, используют метод `digitalWrite()` для включения или выключения светодиода. Обратите внимание, как эти два метода используют свойство `ledPin` в методе `digitalWrite()`. Это свойство устанавливается в конструкторе при создании класса.

Наконец, определяется реализация метода `Led :: blinkLed()`. В этом методе используются методы `Led :: turnOn` и `Led :: turnOff`, определенные ранее, для включения и выключения светодиода. Когда мы вызываем метод класса, мы используем вместе знаки тире / больше (`->`), как показано в методе `blinkLed()`.

Ключевое слово `this` используется для ссылки на текущий экземпляр.

Теперь посмотрим, как мы будем использовать класс `Led`. На главной вкладке первое, что нам нужно сделать, это включить файл `led.h`. Добавьте следующую строку в верхнюю часть вкладки:

```
#include "led.h"
```

Затем нам нужно создать глобальный экземпляр класса `Led` и присвоить ему имя `led`. Для этого мы используем конструктор, который мы создали для класса. Следующий код создаст экземпляр класса `Led`:

```
Led led(LED_ONE, 1000, 500);
```

В классе `LED` конструктор определяется следующим образом:

```
Led::Led(int pin, long on, long off)
```

Обратите внимание, что определение класса `Led` имеет три параметра (`pin`, `on` и `off`). Установленные параметры `hree` соответствуют трем значениям, которые мы передаем в конструктор при создании экземпляра класса `Led`.

Теперь мы можем использовать класс, чтобы светодиод мигал, вызвав метод `blinkLed()` класса. Следующий код показывает, как это сделать:

```
led.blinkLed();
```

Следующий код показывает код на главной вкладке, который будет использовать класс `Led` для мигания светодиода:

```
#include "led.h"
Led led(LED_ONE, 1000, 500);
void setup() {
}
void loop() {
    led.blinkLed();
}
```

Если вы запустите этот код на прототипе, который мы создали в главе 2 «Базовое прототипирование», вы увидите, как мигает один из светодиодов. В этом разделе мы только очень кратко познакомились с ООП, чтобы вы могли понять, как создаются самые профессиональные библиотеки Arduino и как их использовать. Об ООП написаны целые книги, и если вы хотите создавать библиотеки для Arduino, я бы рекомендовал прочитать больше об объектно-ориентированном проектировании в целом и ООП для Arduino. Теперь давайте посмотрим, как мы можем использовать встроенную библиотеку `String` для Ардуино.

Библиотека строк

Библиотека `String`, которая является частью основных библиотек Arduino, позволяет нам использовать и управлять текстом проще, чем это делают массивы символов. Для использования библиотеки `String` требуется больше памяти, чем использовать символьные массивы, но проще использовать библиотеку `String`. Существует множество способов создать экземпляр типа `String`. Давайте посмотрим здесь на несколько примеров:

```
String str1 = "Arduino";
String str2 = String("Arduino");
String str3 = String('B');
String str4 = String(str2 + " is Cool");
```

Обе первые две строки создают простую строку со словом Arduino. В третьей строке новый экземпляр String создается из одного постоянного символа. Обратите внимание, что в этой строке используется одинарная кавычка.

Последний пример объединяет две строки. Есть еще несколько конструкторов, которые позволяют нам создавать экземпляры класса String из числа. Вот несколько примеров:

```
String strNum1 = String(42);  
String strNum2 = String(42, HEX);  
String strNum3 = String(42, BIN);
```

В предыдущем коде экземпляр strNum1 String будет содержать текст 42, который является десятичной версией числа 42. Экземпляр strNum2 String будет содержать текст 2a, который является шестнадцатеричной версией числа 42. Экземпляр strNum3 String будет содержать текст 101010, который является двоичной версией числа 42.

Также существует множество методов, которые можно использовать в экземплярах класса String. Вот некоторые из этих методов:

- `concat(string)`: объединить одну строку с концом исходной строки.
- `endsWith(string)`: возвращает истину, если исходная строка заканчивается символом другой строки.
- `equals ()`: сравнивает две строки и возвращает истину, если строки содержат одинаковый текст. При сравнении строк этот метод чувствителен к регистру.
- `equalsIgnoreCase ()`: сравнивает две строки и возвращает истину, если строки содержат один и тот же текст. При сравнении строк этот метод нечувствителен к регистру.
- `length ()`: возвращает длину строк. Длина не будет включать завершающий нулевой символ.
- `replace (substring1, substring2)`: этот метод заменяет все экземпляры одной подстроки другой подстрокой.
- `startsWith (string)`: возвращает true - истину, если исходная строка начинается с символов другой строки.
- `toLowerCase ()`: возвращает версию исходной строки в нижнем регистре.
- `toUpperCase ()`: возвращает версию исходной строки в верхнем регистре.

Библиотеку String можно использовать как замену символьному массиву; однако вы обнаружите, что в большинстве примеров кода в Интернете используются массивы символов, главным образом потому, что они занимают меньше памяти и выполняются быстрее, чем библиотека String.

На этом заканчивается введение в язык программирования Arduino. Вы можете обратиться к страницам краткого справочника Arduino для получения дополнительной информации о языке программирования Arduino.

Вы можете найти справочные страницы здесь: <https://www.arduino.cc/reference/en/>. На этой странице вы найдете ссылки на информацию о встроенных функциях и переменных Arduino, а также на информацию об операторах и других элементах языка Arduino.

Не волнуйтесь, если вам неудобно писать собственные программы для Arduino прямо сейчас, потому что мы будем писать много кода в оставшихся главах этой книги, и, в конце концов, вы почувствуете себя комфортно при написании собственных приложений Arduino.

Датчик движения

В этой главе мы рассмотрим, как использовать датчик движения HC-SR501. Этот датчик очень легко подключить к Arduino, поэтому обычно это один из первых датчиков, с которым люди экспериментируют, когда начинают работать с микроконтроллерами. Кроме того, он недорогой и часто идет в комплекте с большинством стартовых наборов.

В этой главе вы узнаете:

- Как подключить датчик движения HC-SR501 к Arduino
- Как прочитывать выходной сигнал датчика движения HC-SR501
- Прочитать схему Фритцинга проекта датчика движения

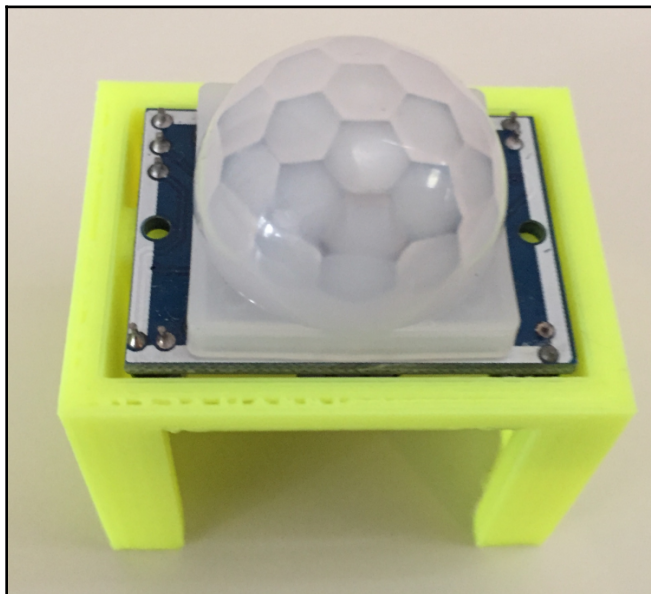
Датчики PIR, также известные как пассивные инфракрасные датчики, используются микроконтроллером для обнаружения движения, как правило, человека, но они обнаруживают любое движение в пределах диапазона датчика. Эти датчики небольшие, недорогие, маломощные и простые в использовании, что делает их идеальными для начинающих экспериментировать, но промышленные версии этих датчиков также можно найти во многих потребительских и военных товарах.

Датчики PIR состоят из пироэлектрических датчиков, которые могут определять уровни инфракрасного излучения. Каждый объект, имеющий температуру выше абсолютного нуля, испускает небольшое инфракрасное излучение низкого уровня, которое может обнаружить пироэлектрический датчик. Пассивная часть названия означает, что сенсоры не генерируют и не излучают энергию, которая может быть обнаружена другими устройствами. Вместо этого он работает, обнаруживая инфракрасное излучение, испускаемое другими объектами.

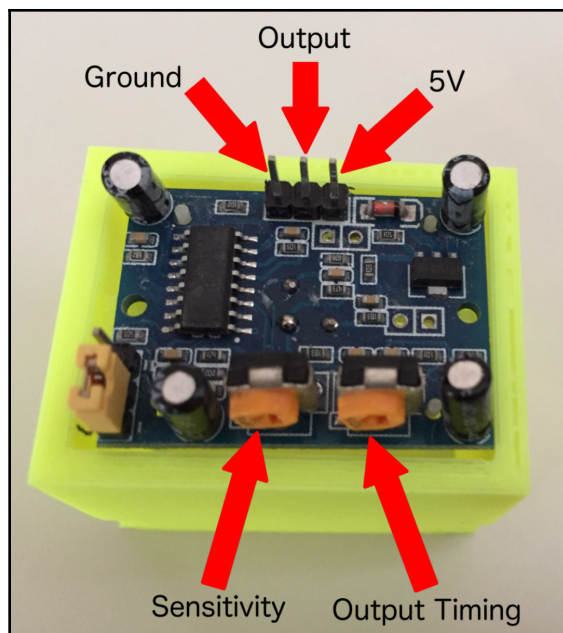
Пироэлектрический датчик в датчике движения обычно разделен на две части, что позволяет датчику движения обнаруживать изменения уровней инфракрасного излучения. Когда датчик не обнаруживает никакого движения, обе части обнаруживают одинаковое количество инфракрасного излучения и нейтрализуют друг друга. Когда что-то начинает двигаться в пределах досягаемости датчика, одна половина датчика улавливает больше инфракрасного излучения, чем другая половина, в результате чего датчик запускает сигнал движения.

Датчики PIR бывают разных размеров и характеристик. Эти датчики используются во многих коммерческих продуктах, таких как охранная сигнализация, автоматическое освещение и праздничные украшения, которые загораются или загораются при приближении к ним.

В этой главе мы будем использовать датчик движения HC-SR501, который показан на следующей фотографии, с подставкой, которую я спроектировал и распечатал для него с помощью 3D принтера. Загружаемый код для этой книги включает файл STL, который вы можете использовать для распечатки подставки для собственного использования:



На следующем изображении показаны разъемы и подстроечные резисторы в нижней части датчика движения HC-SR501:



Регулировка чувствительности (**Sensitivity**) изменяет диапазон обнаружения движения датчика. Дальность обнаружения может быть установлена от 3 до 7 метров. Поворот движка подстроечника чувствительности по часовой стрелке снижает чувствительность датчика.

Регулировка времени вывода (**Output Timing**) устанавливает, как долго вывод будет оставаться на высоком уровне после обнаружения движения.

Время вывода может составлять от 5 секунд до 5 минут. Поворот движка регулировки времени вывода по часовой стрелке увеличит задержку времени.

Вывод земли (**Ground**) должен быть подключен к шине земли на макетной плате или непосредственно к выводу земли на Arduino. Вывод **5V** должен быть подключен к + питания на макетной плате или напрямую к +5V на Arduino. Наконец, средний вывод - это выходной сигнал датчика. Если датчик обнаруживает движение, этот вывод находится на высоком уровне в течение времени, определенного **Output Timing**.



. Некоторые совместимые датчики могут иметь конфигурацию выводов, отличную от показанной в этой книге; Пожалуйста, проверьте конфигурацию выводов вашего датчика, прежде чем выполнять какие-либо подключения.

Прежде чем вы посмотрите на раздел принципиальной схемы в этой главе, подумайте, как бы вы подключили датчик движения HC-SR5012 к Arduino. Один намек, вам ничего не понадобится, кроме датчика движения, Arduino и перемычки. Использование макетной платы в этом конкретном проекте не является обязательным.

Теперь давайте посмотрим на компоненты, необходимые для проекта этой главы.

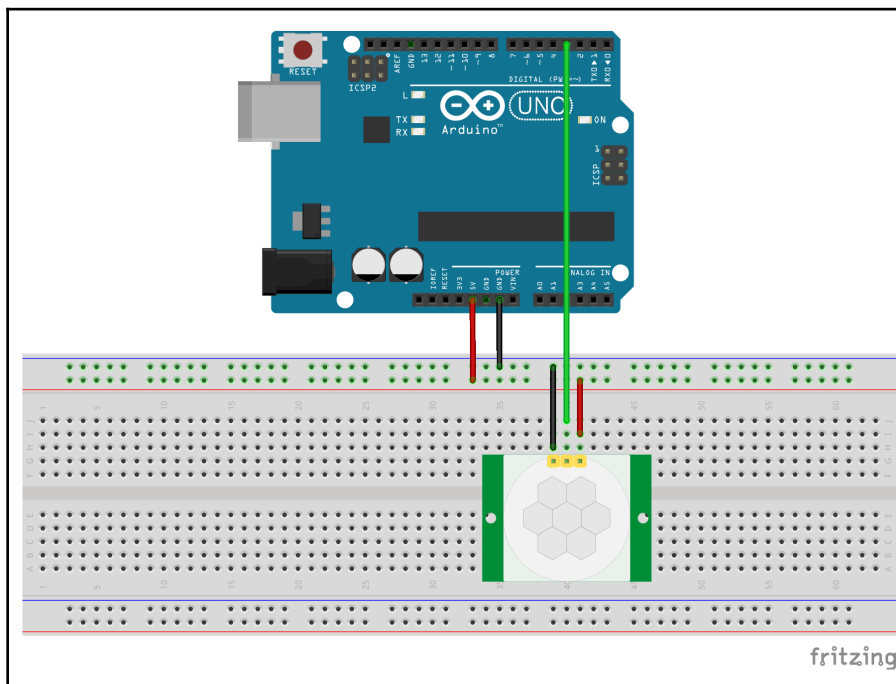
Для проекта этой главы нам потребуются следующие компоненты:

- Arduino Uno или совместимая плата
- Датчик движения HC-SR501
- Перемычки
- Для задания вам понадобится светодиод
- Макетная плата



Макетная плата не является обязательной, поскольку вы можете подключить датчик движения HC-SR501 непосредственно к Arduino. Вам понадобится макетная плата, чтобы выполнить задание в этой главе.

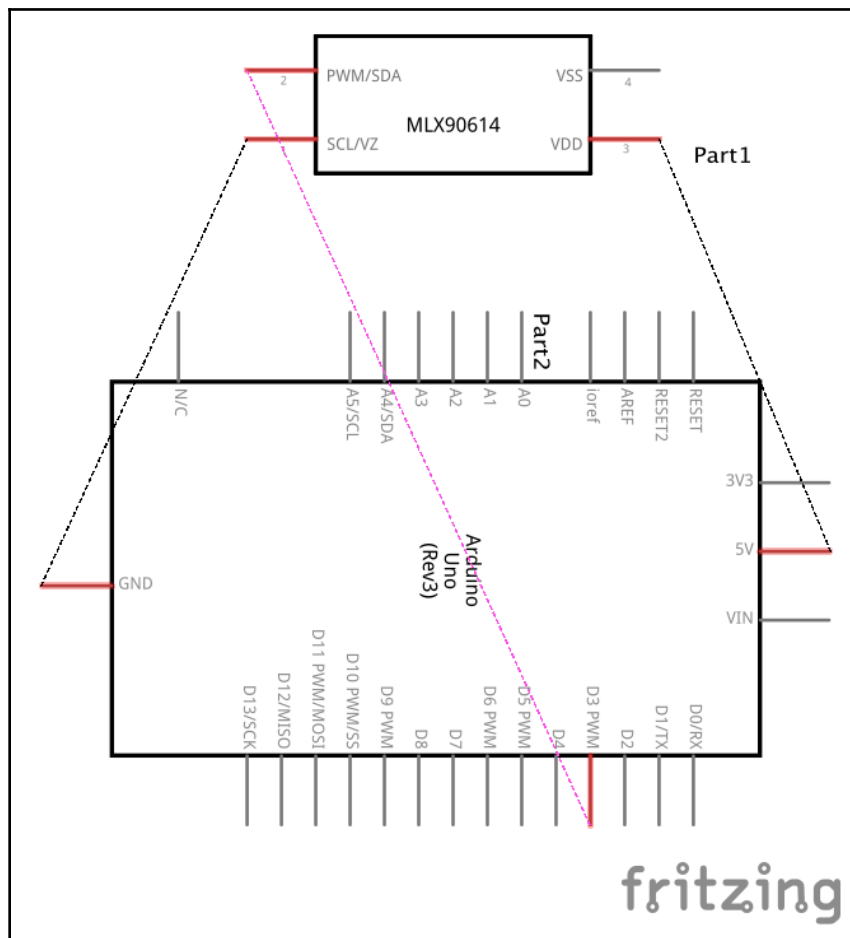
На следующей рисунке показана схема Фритцинга для этого проекта:



На схеме мы видим, что вывод земли датчика движения HC-SR501 подключен к шине земли на макетной плате, а +5 В на датчике движения подключен к + шине питания на макетной плате. Шины питания и земли макета подключены к выводам питания 5 В и земли на Arduino.

Вывод выходного сигнала датчика движения является цифровым, поэтому мы можем подключить его напрямую к любому из цифровых выводов на Arduino. В этом случае мы подключаем вывод выходного сигнала датчика к третьему выводу на Arduino.

Вот принципиальная схема той же схемы:



Чтобы использовать датчик движения HC-SR501, все, что нам нужно сделать, это прочитать цифровой выходной сигнал датчика. Если выходной сигнал ВЫСОКИЙ, то датчик обнаружил движение, а если НИЗКИЙ, то нет. Выходной сигнал датчика будет оставаться ВЫСОКИМ в течение времени, определяемого временем вывода (Output Timing). Я обычно устанавливаю небольшое время вывода, обычно пару секунд.

Для этого проекта мы выведем состояние датчика на последовательную консоль. Результат станет немного интереснее в разделе испытаний.

Ниже приведен код для чтения датчика движения HC-SR501:

```
#define MOTION_SENSOR 3

void setup() {
  pinMode(MOTION_SENSOR, INPUT);
  Serial.begin(9600);
}

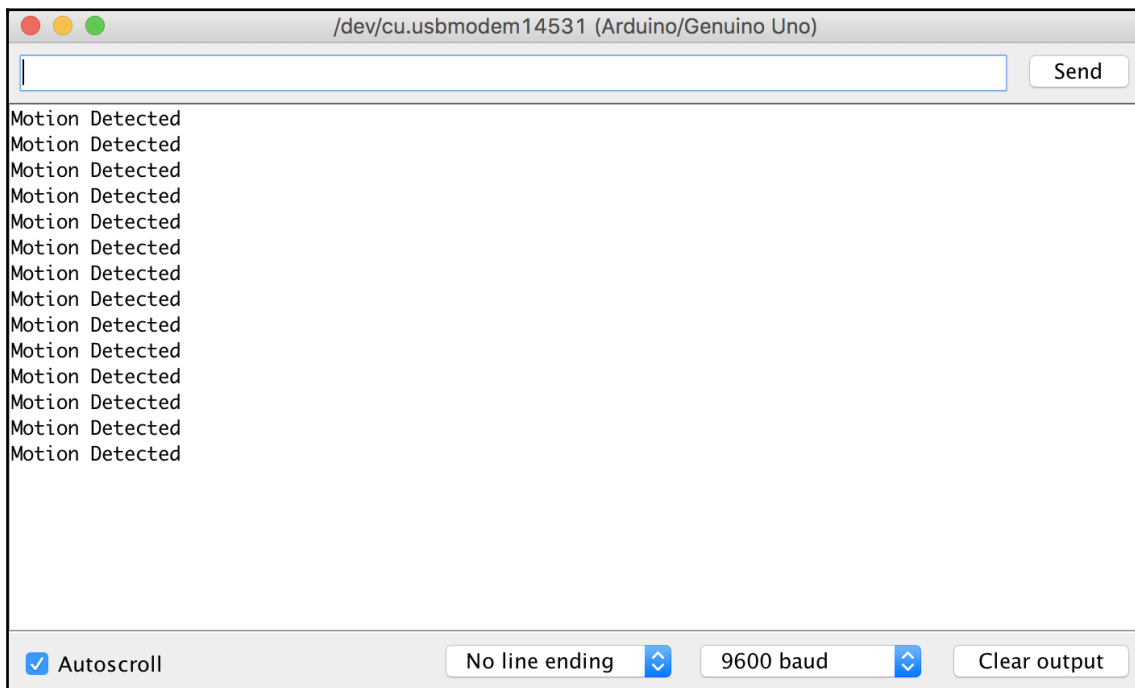
void loop() {
  int sensorValue = digitalRead(MOTION_SENSOR);
  if (sensorValue == HIGH) {
    Serial.println("Motion Detected");
  }
  delay(500);
}
```

Этот код начинается с использования директивы `#define` для создания макроса `MOTION_SENSOR` и установки для него значения 3. В функции `setup ()` мы устанавливаем сигнальный вывод как вход. Мы также иницилируем последовательную консоль в функции `setup ()`.

Функция `loop ()` начинается с вызова функции `digitalRead ()` для чтения выходных данных датчика движения и присвоения значения переменной `sensorValue`. Если переменная `sensorValue` равна `HIGH`, мы отправляем сообщение «Motion Detected - Обнаружено движение» на последовательную консоль. Если движения не обнаружено, мы ничего не печатаем. В конце функции `loop ()` есть задержка в полсекунды перед повторным циклом.

Теперь загрузим и запустим код.

Нам нужно будет открыть последовательную консоль, когда мы запустим код, чтобы увидеть результат. После того, как код будет загружен и запущен, помашите рукой рядом с датчиком, и вы должны увидеть, что сообщение «Motion Detected - Обнаружено движение» выводится на консоль, как показано на следующем снимке экрана:



Теперь перейдем к разделу, посвященному задаче этой главы.

Задача состоит в том, чтобы добавить в этот проект светодиод, чтобы он загорался, когда датчик движения обнаруживает движение. В первых нескольких главах мы дадим либо код, либо принципиальную схему, чтобы упростить поиск и устранение любых проблем, если проект не работает при первоначальной сборке.

Для этой задачи мы дадим код и позволим вам выяснить, как подключить светодиод к проекту. Подсказка, не забудьте про резистор для светодиода.

Ниже приведен код, который зажжет светодиод, подключенным к выводу 5 Arduino, когда датчик движения обнаруживает близлежащее движение:

```
#define MOTION_SENSOR 3
#define LED 5

void setup() {
  pinMode(MOTION_SENSOR, INPUT);
  pinMode(LED, OUTPUT);

  digitalWrite(LED, LOW);
  Serial.begin(9600);
}

void loop() {
  int sensorValue = digitalRead(MOTION_SENSOR);
  if (sensorValue == HIGH) {
    Serial.println("Motion Detected");
  }

  digitalWrite(LED, sensorValue);
  delay(500);
}
```

Теперь вам предстоит выполнить это задание.

Summary

В этой главе мы узнали о датчике движения HC-SR501 и о том, как он работает. Мы увидели, как подключить его к Arduino, где Arduino обеспечивает питание датчика, а также считывает состояние вывода, к которому подключен датчик.

В следующей главе мы увидим, как мы можем чувствовать погоду вокруг нас.

Датчики окружающей среды

В этой главе мы рассмотрим создание действительно простой метеостанции с использованием датчика температуры / влажности DHT11 и датчика дождевых капель. В то время как в предыдущей главе использовался базовый цифровой вход, датчик температуры DHT11 даст нам возможность использовать стороннюю библиотеку, а датчик капли дождя будет использовать аналоговый вывод. Мы также представим несколько удобных функций, которые мы можем использовать.

В этой главе вы узнаете:

- Как добавить в скетч сторонние библиотеки
- Как использовать функцию `isnan()`
- Как использовать функцию `themap()`
- Как использовать датчик температуры и влажности DHT11
- Как использовать датчик дождя

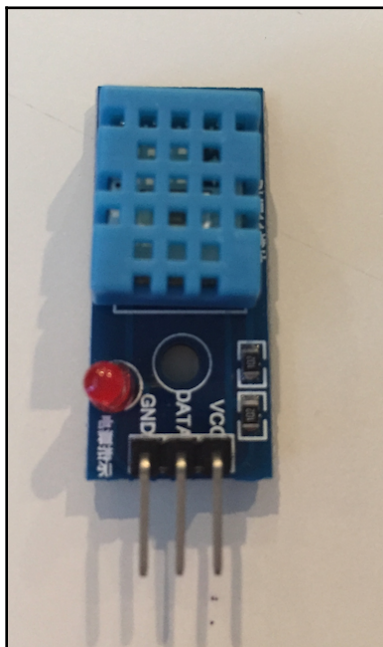
DHT11 - недорогой датчик температуры и влажности. Этот датчик использует термистор для измерения температуры.

Термин термистор представляет собой комбинацию термического (температура) и резистора, потому что это тип резистора, сопротивление которого очень чувствительно к температуре даже в большей степени, чем обычный резистор. Текущая температура может быть определена на основе выходного напряжения термистора.

При работе с термистором первое, что нам нужно сделать, это определить, как рассчитать температуру на основе выходного напряжения.

С датчиком температуры TMP36, который мы использовали с прототипом, созданным в главе 4, Базовое прототипирование, мы могли легко рассчитать температуру на основе выходного напряжения датчика по базовой формуле (напряжение - 0,5) * 100,0, поскольку он использует твердотельный метод определения температуры. С термистором дело обстоит иначе. Хотя линейное приближение, подобное тому, как мы рассчитали температуру с помощью датчика TMP36, может работать для небольшого диапазона температур, для получения точного измерения температуры с помощью термистора нам необходимо определить кривую сопротивления / температуры для устройства.

К счастью, для Arduino есть несколько библиотек, которые помогут нам получить точную температуру с помощью датчика температуры и влажности DHT11. В этой главе мы будем использовать библиотеку Adafruit. Датчик DHT11 будет выглядеть примерно так, как на следующей фотографии:



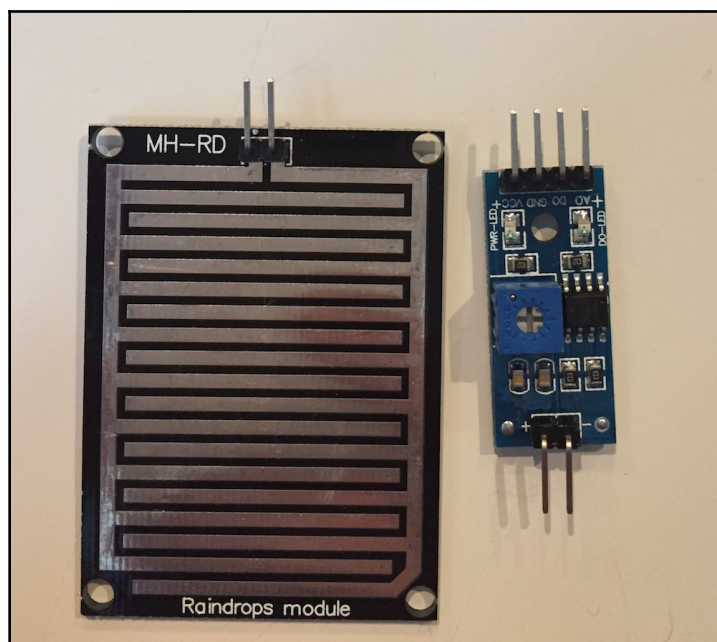
У большинства датчиков DHT11 выводы четко обозначены, как показано на предыдущей фотографии. Вывод **VCC** будет подключаться к шине питания на макетной плате, которая должна быть подключена к выводу 5V на Arduino. Вывод **GND** будет подключаться к заземляющей шине на макетной плате, которая должна быть подключена к выводу заземления на Arduino. Вывод **DATA** будет подключаться к одному из цифровых выводов на Arduino.



Некоторые датчики температуры DHT имеют встроенный подтягивающий резистор, в то время как другим требуется внешний резистор. Пожалуйста, посмотрите документацию на свой датчик, чтобы проверить, нужно ли вам добавлять внешний подтягивающий резистор или нет. В проекте для этой главы мы покажем внешний подтягивающий резистор.

Для проекта в этой главе мы также будем использовать обычный датчик дождевых капель. Этот датчик состоит из двух частей. Первая часть - это плата датчика дождя, которая обнаруживает дождь, когда вода замыкает цепи на печатных выводах платы. Эта сенсорная плата действует как переменный резистор, сопротивление которого уменьшается по мере увлажнения платы. Вторая часть датчика дождевых капель - это электронная печатная плата, которая будет определять количество воды на основе тока от платы датчика.

На следующей фотографии показано, как выглядит датчик капли дождя:

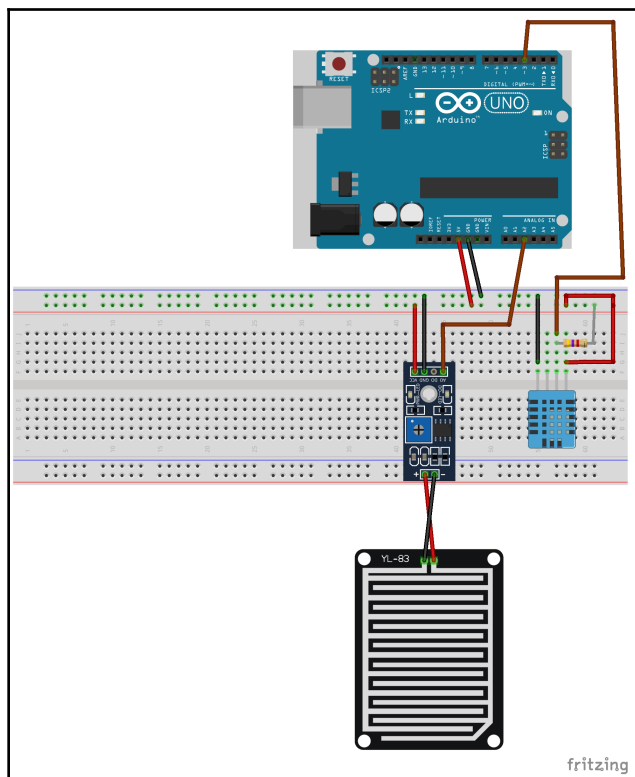


Выводы +/- на печатной плате подключаются к выводам на плате датчика дождя. На противоположной стороне печатной платы четыре вывода. Выводы VCC и GND будут подключаться к шине питания и заземлению макета соответственно. В проекте, описанном в этой главе, мы будем использовать вывод аналогового выхода A0 в качестве выхода для датчика. Вывод A0 подключается напрямую к одному из аналоговых выводов на Arduino.

Для проекта этой главы нам потребуются следующие компоненты:

- Arduino Uno или совместимая плата
- Датчик температуры / влажности DHT11
- Датчик капель дождя MH-RD
- Один резистор 4,7 кОм
- переключки
- макетная плата

На следующем рисунке показана схема Фритцинга для этого проекта:



На этой схеме мы видим, что выводы VCC и земли на обоих датчиках подключены к шинам питания и заземления на макетной плате. Шины питания и земли на макетной плате подключены к выходу 5 В и выводам земли на Arduino.

Изображение датчика DHT11, которое мы показали ранее в этой главе, показывает датчик DHT11 с тремя выводами; однако датчик в библиотеке Fritzing имеет четыре вывода. Дополнительный вывод на диаграмме Фритцинга можно смело игнорировать.

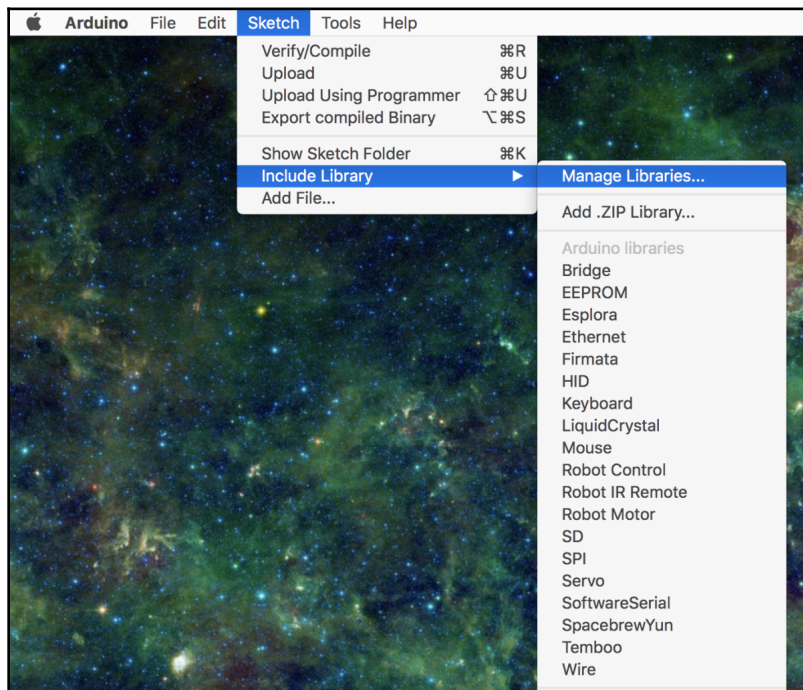
На этой схеме показано, что вывод данных на датчике DHT11 подключен к цифровому выводу 3 на Arduino, а также имеет подтягивающий резистор 4,7 кОм. Если датчик DHT11, который вы используете, не имеет встроенного подтягивающего резистора, вам нужно будет добавить этот внешний резистор, который показан на этой схеме. Аналоговый выход на датчике дождя подключен к аналоговому входу 2 на Arduino.

Прежде чем мы сможем начать писать код, нам нужно будет загрузить библиотеку DHT11 Adafruit, которую мы будем использовать для считывания показаний температуры и влажности. Вы можете найти исходный код этой библиотеки на странице Adafruit GitHub здесь: <https://github.com/adafruit/DHTsensor-library>.

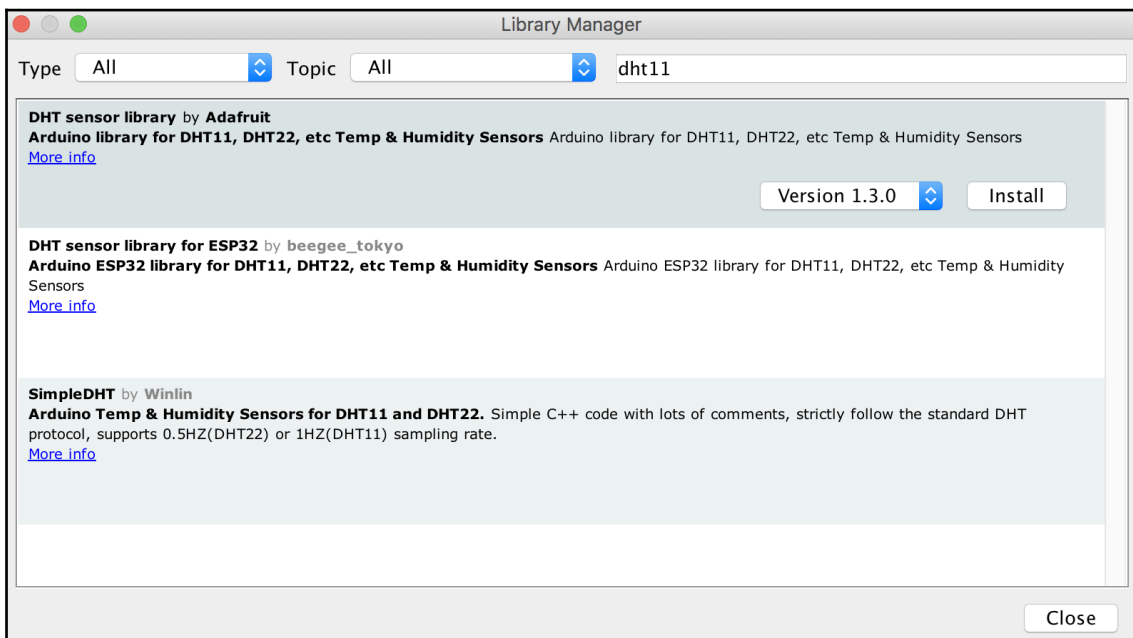


Примечание: вам нужно будет обратиться к этому коду в разделе задач в этой главе.

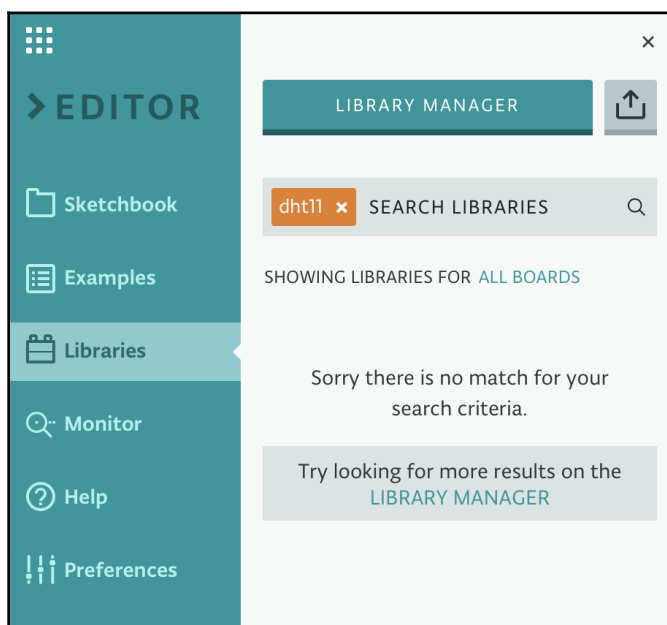
Чтобы установить библиотеку, если вы используете Arduino IDE, выберите опцию **Sketch** в строке меню и выберите « Include Library - Включить библиотеку», а затем «Manage Libraries - Управление библиотеками», как показано на следующем снимке экрана:



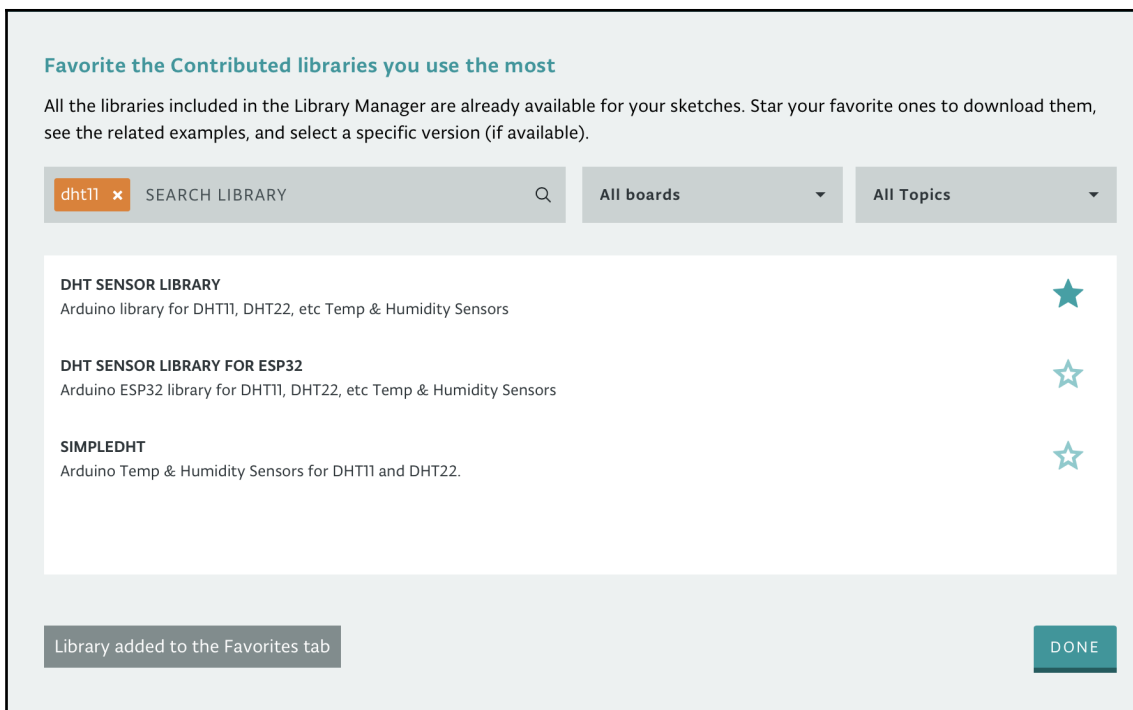
В открывшемся окне введите dht11 в строке поиска, и вы должны увидеть несколько разных библиотек датчиков DHT11. В этой главе мы будем использовать продукт от Adafruit (в следующем списке - первый результат). Нажмите на эту библиотеку, и вы увидите кнопку до конца вправо, чтобы установить ее, как показано на следующем снимке экрана:



В веб-редакторе Arduino выберите опцию « Libraries - Библиотеки », а затем введите dht11 в строке поиска, как показано на следующем снимке экрана:



Вероятно, это не даст никаких результатов; поэтому нам нужно будет щелкнуть ссылку LIBRARYMANAGER, которая вызовет диспетчер библиотеки с результатами поиска DHT11, показанными на следующем изображении:



Как говорят нам инструкции в верхней части этого окна, нам нужно щелкнуть звездочку библиотеки, которую мы хотим включить, а затем щелкнуть кнопку DONE - ГОТОВО. К сожалению, менеджер библиотеки в веб-редакторе не сообщает нам, кто создал библиотеку; однако, если вы заметили, имена из трех библиотек совпадают с именами библиотек, которые мы видели в среде разработки Arduino.

Таким образом, мы можем определить, какая библиотека является библиотекой Adafruit по названию. Теперь, когда у нас установлена библиотека, пора приступить к написанию кода. Первое, что нам нужно сделать, это включить файл заголовка для библиотеки датчиков DHT. Мы можем сделать это, добавив следующий оператор include вверху скетча:

```
#include "DHT.h"
```

Далее нам нужно определить некоторые макросы. Мы начнем с определения выводов Arduino, к которым подключены DHT11 и датчик дождевых капель:

```
#define DHT_PIN 3  
#define RAIN_PIN A2
```

Этот код сообщает нам, что датчик DHT11 подключен к цифровому выводу 3, а датчик дождя подключен к аналоговому выводу 2. Библиотека датчиков Adafruit DHT может считывать как датчики DHT11, так и DHT22. Следовательно, нам нужно будет сообщить библиотеке, какой тип датчика мы используем, и мы должны создать макрос, содержащий этот тип. Следующий код определяет тип датчика DHT:

```
#define DHT_TYPE DHT11
```

Наконец, нам нужно будет создать четыре макроса, которые помогут нам понять показания датчика дождя. Если вы помните, выходы аналогового входа будут отображать входное напряжение в целочисленные значения от 0 до 1023. Когда мы считываем входные данные от датчика дождя, значение 1023 означает, что дождя нет, а значение 0 означает наводнение. Это имеет смысл с чисто электронной точки зрения; однако с логической точки зрения это должно быть наоборот, когда датчик дождя должен сообщать более высокое значение, когда идет большой дождь.

Мы воспользуемся функцией `Arduinomap()`, чтобы изменить это за нас. Следовательно, нам нужно будет определить минимальные / максимальные значения для аналоговых показаний и минимальные / максимальные значения, в которые мы хотим преобразовать аналоговые значения. Мы объясним это еще немного, когда посмотрим на код функции `map()`; а пока вот макросы:

```
#define RAIN_SENSOR_MAX 1023
#define RAIN_SENSOR_MIN 0
#define RAIN_OUT_MAX 20
#define RAIN_OUT_MIN 0
```

Теперь нам нужно создать экземпляр класса DHT, используя макросы `DHT_PIN` и `DHT_TYPE`, которые мы только что определили. Следующий код создаст экземпляр класса `DHTclass`:

```
DHT dht(DHT_PIN, DHT_TYPE);
```

Теперь, когда мы определили необходимые макросы и создали глобальный экземпляр класса DHT, нам нужно создать функцию `setup()`. В функции `setup()` нам нужно будет инициализировать последовательный монитор и класс DHT. Метод `begin()` класса DHT используется для инициализации экземпляра класса. Следующий код показывает функцию `setup()`:

```
void setup() {
    Serial.begin(9600);
    dht.begin();
}
```

Теперь, когда у нас есть функция `setup()`, давайте посмотрим, как читать DHT и датчики дождя. Оставшаяся часть кода в этой главе перейдет в функцию `loop()` нашего скетча. Следующий код будет считывать влажность и температуру с датчика DHT с использованием библиотеки Adafruit:

```
float humidity = dht.readHumidity();
float celsius = dht.readTemperature();
float fahrenheit = dht.readTemperature(true);

if (isnan(humidity) || isnan(celsius) || isnan(fahreheit)) {
    Serial.println("Read Failed");
    return;
}

Serial.print("Humidity: ");
Serial.println(humidity);
Serial.print("Temperature: ");
Serial.print(celsius);
Serial.println(" *C ");
Serial.print(fahreheit);
Serial.println(" *F");

delay(3000);
```

Этот код начинается с вызова метода `readHumidity()` класса DHT для считывания влажности с датчика DHT.

Затем метод `readTemperature()` вызывается дважды: один раз для чтения температуры в градусах Цельсия и один раз для чтения температуры в градусах Фаренгейта. Обратите внимание, что когда метод `readTemperature()` вызывается без параметра, мы получаем температуру в градусах Цельсия, а когда мы передаем логический параметр, равный `true`, мы получаем температуру в градусах Фаренгейта. Мы также можем передать логический параметр `false` для получения температуры в градусах Цельсия.

После того, как датчик считывает температуру и влажность, рекомендуется убедиться, что считывание прошло успешно. Для этого мы используем функцию `isnan()`. Функция `isnan()` вернет истину, если переданное значение не является числом, поэтому строка `if (isnan(влажность) || isnan(по Цельсию) || isnan(по Фаренгейту))` читается как «если влажность не является числом, или Цельсий не является числом, или Фаренгейт не является числом, тогда выполните блок кода». Этот блок кода выведет на консоль сообщение об ошибке, а затем выполнит оператор `return` для выхода из этого цикла.

Если все переменные являются числами, мы выводим влажность и температуру на последовательную консоль, а затем ждем 3 секунды, прежде чем выйти из этого цикла и снова запустить функцию цикла.

Теперь давайте посмотрим, как мы будем считывать датчик дождя. Поместите следующий код между окончательным оператором `Serial.println()` и вызовом функции `delay()` кода датчика ДНТ:

```
int rain = analogRead(RAIN_PIN);
if (isnan(rain)) {
    Serial.println("Read Failed");
    return;
}
int range = map(rain, RAIN_SENSOR_MIN, RAIN_SENSOR_MAX, RAIN_OUT_MAX,
RAIN_OUT_MIN);

Serial.print("Rain: ");
Serial.println(range);
Serial.println("-----");
```

В этом коде мы вызываем функцию `analogRead()` для чтения аналогового вывода, к которому подключен датчик дождя, и используем функцию `isnan()`, чтобы убедиться, что чтение было успешным.

Убедившись, что функция `analogRead()` была успешно выполнена, мы вызываем функцию `map()`. Функция `map()` повторно отобразит значение из одного диапазона чисел в новый диапазон чисел. Эта функция имеет пять параметров, а именно:

- `value`: значение для сопоставления
- `fromLow`: нижний предел текущего диапазона значения.
- `fromHigh`: верхний предел текущего диапазона значения.
- `toLow`: нижний предел нового диапазона значения.
- `toHigh`: верхний предел нового диапазона значения.

Если мы возьмем вызов функции `map()` в предыдущем коде и заменим макросы на фактические значения, функция `map()` будет выглядеть так:

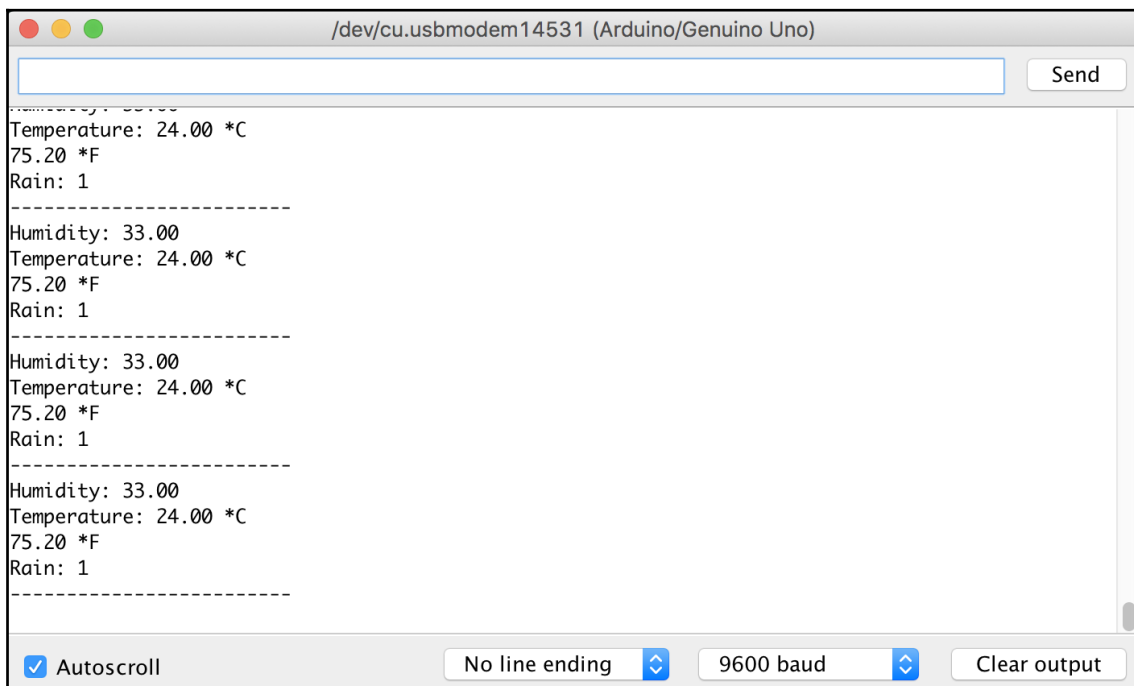
```
int range = map(rain, 0, 1023, 20, 0);
```

Значение переменной `rain` поступает из функции `analogRead()`, которая, как мы знаем, будет иметь значение от 0 до 1023. Поэтому мы устанавливаем текущий диапазон так, чтобы нижний предел был равен 0, а верхний предел - 1023. Напомним, что значение 1023 означает, что дождя нет, а значение 0 означает, что есть наводнение. Мы хотим изменить это положение с помощью нового диапазона, где более высокое значение будет означать большой дождь, а меньшее значение - маленький дождь. Поэтому мы устанавливаем нижний предел нового диапазона на 20 и верхний предел на 0. Это сопоставит значение 1023 из старого диапазона со значением 0 в новом диапазоне и значение 0 в старом диапазоне с значением 20 в новом диапазоне.

В этом новом диапазоне высокое значение 20 означает, что у нас наводнение, а низкое значение 0 означает, что дождя нет. Среднее значение из старого диапазона (511 или 512) будет отображаться на среднее значение в новом диапазоне (10). Функция `map()` очень полезна, когда мы хотим изменить масштаб и / или изменить порядок, как мы видим в этом примере.

После вызова функции `map()` мы выводим результаты на последовательную консоль. Теперь посмотрим, что произойдет, когда мы запустим этот проект.

Когда мы запустим этот проект, мы должны увидеть результат, подобный следующему снимку экрана:



Теперь попробуйте сбрызнуть плату датчика дождя (часть датчика каплей дождя, которая воспринимает дождь) водой и посмотрите, как это изменит выходные данные платы дождя.



: Всегда соблюдайте осторожность при использовании воды для электронных проектов. Если вы намочите Arduino или другой электронный компонент, вы их повредите. При работе от сети переменного тока с реле вы также рискуете получить удар электрическим током.

Для решения этой задачи используйте библиотеку DHT для вычисления теплового индекса. Индекс жары - это дискомфорт, который ощущается в результате сочетания температуры и влажности. В классе DHT есть методы, которые сделают это за вас.



В начале этой главы мы дали ссылку на репозиторий GitHub, который содержал код библиотеки датчиков DHT. Посмотрите файл DHT.h, чтобы узнать, какие методы входят в класс DHT.

В этой главе мы впервые использовали стороннюю библиотеку. Это была библиотека датчиков Adafruit DHT. Мы также увидели две новые функции, которые раньше не использовали. Это были функции `isnan()` и `map()`.

В следующей главе мы рассмотрим датчики дальности и обнаружения столкновений.

8

Обход препятствий и обнаружение столкновений

Если вы создаете автономного робота, который должен избегать препятствий, автомобиль с дистанционным управлением, который должен обнаруживать, когда он сталкивается с чем-то или даже 3D-принтер, который должен знать, когда печатающие головки достигли пределов области печати, вам нужно будет включить некоторые виды системы предотвращения препятствий или обнаружения столкновений в вашем проекте. В этой главе мы рассмотрим несколько датчиков, которые можно использовать для систем предотвращения препятствий и обнаружения столкновений.

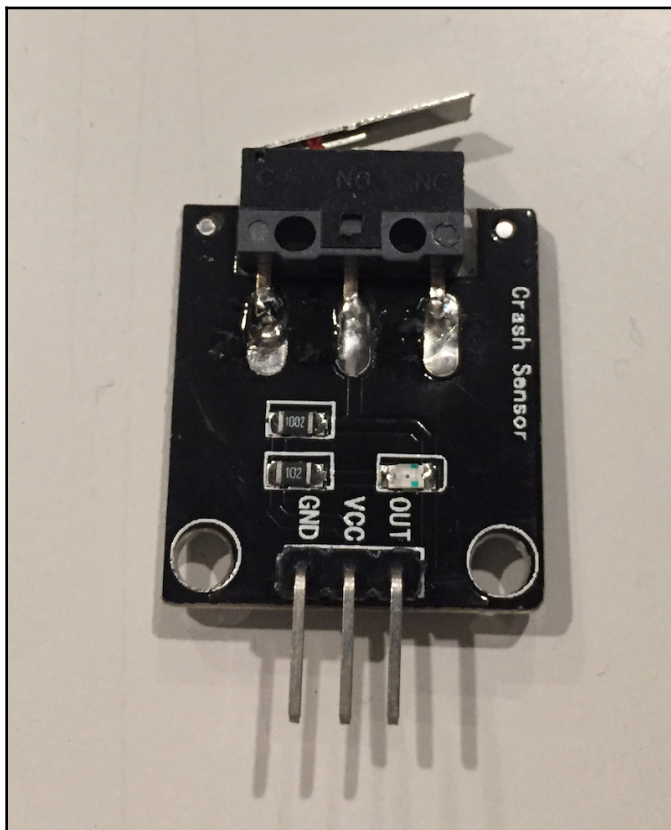
В этой главе вы узнаете:

- Как использовать датчик столкновения
- Как использовать инфракрасный датчик объезда препятствий
- Как пользоваться ультразвуковым дальномером

В этой главе мы рассмотрим три датчика предотвращения препятствий и / или обнаружения столкновений. Эти датчики:

- : используется для обнаружения аварии, а также в качестве концевых выключателей для 3D-принтеров.
- : используется для предотвращения препятствий в робототехнике.
- **Ультразвуковой дальномер**: используется для обхода препятствий в робототехнике и имеет множество других коммерческих и военных целей.

Датчик столкновения - это, по сути, простой переключатель, на котором есть какой-то удлинитель, который дает ему большую площадь для обнаружения аварии. На следующей фотографии показано, как будет выглядеть базовый датчик столкновения:

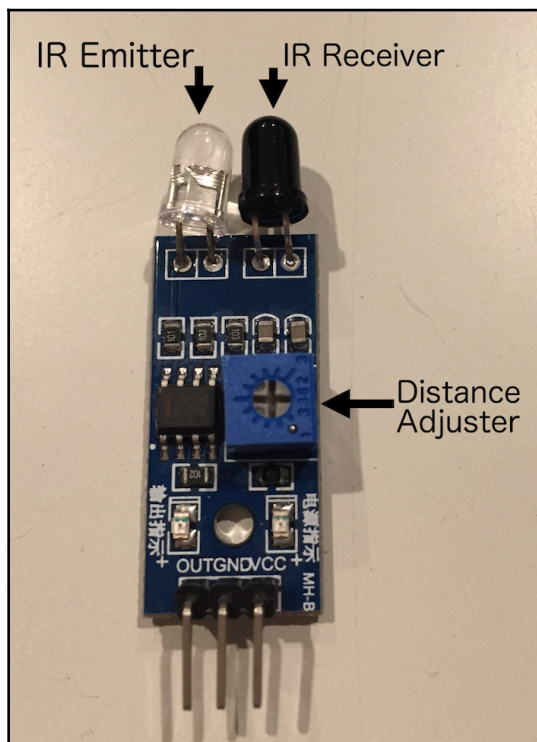


Датчик столкновения, показанный на предыдущей фотографии, использует простой механический переключатель, подобный типам, используемым для концевых упоров на 3D-принтерах. Это упрощает монтаж шасси робота или других поверхностей. Концепция датчика столкновения заключается в том, что при срабатывании переключателя датчик во что-то врезался.

Датчик столкновения имеет три контакта, которые четко обозначены как GND, VCC и OUT. Контакт GND подключается к шине заземления, а VCC подключается к шине питания на макетной плате. Вывод OUT подключается напрямую к цифровому выводу на Arduino с подтягивающим резистором 4,7 кОм.

Инфракрасный датчик предотвращения препятствий состоит из инфракрасного передатчика, инфракрасного приемника и потенциометра, который регулирует расстояние, на котором датчик обнаруживает препятствия. На следующих фотографиях показан датчик предотвращения препятствий, который используется для проекта в этой главе.

Излучатель (emitter) на инфракрасном датчике объезда препятствий испускает инфракрасное излучение, и если препятствие находится перед датчиком, часть излучения отражается назад и улавливается приемником (receiver). Если перед датчиком нет объектов, излучение рассеивается, и приемник ничего не получает обратно.

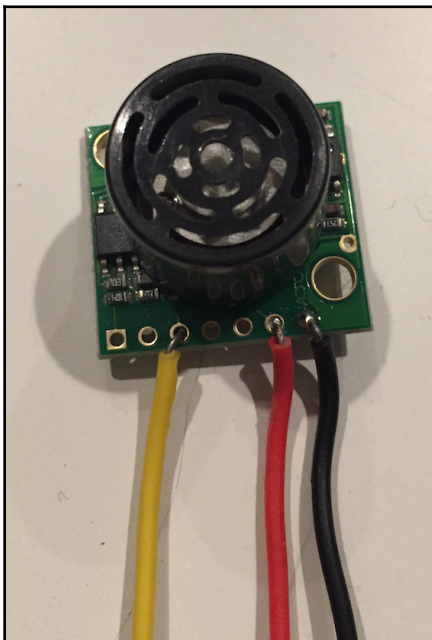


Выводы датчика имеют четкую маркировку OUT, GND и VCC слева направо.

Вывод GND подключен к шине заземления, а вывод VCC подключен к шине питания макета. Вывод OUT подключается непосредственно к цифровому выводу на Arduino. Если сигнал с вывода OUT имеет низкий (LOW) уровень, значит, объект был обнаружен. Если выход ВЫСОКИЙ, значит, объект не обнаружен.

Регулятор расстояния (Distance adjuster) будет регулировать расстояние, на котором датчик обнаруживает объекты. Если регулятор повернуть против часовой стрелки, то расстояние будет уменьшено, а если повернуть его по часовой стрелке, расстояние увеличится. Датчик обнаружит объекты от 2 до 30 см.

Третий датчик, который мы будем использовать, - это ультразвуковой дальномер MaxSonar EZ1. Этот датчик - один из моих любимых датчиков. Я использовал его почти в каждом автономном роботе, который я построил, чтобы определять расстояние до ближайших объектов. Ниже приведено изображение ультразвукового дальномера EZ1:



Например, в этой главе мы будем использовать выводы 3, 6 и 7 на датчике. Контакт 3 используется для аналогового выхода, вывод 6 - для VCC, а вывод 7 - для заземления. Выводы 4 и 5 предназначены для последовательного подключения RX / TX, а вывод 2 - для широтно-импульсного выхода, однако мы не будем использовать эти выходы в примере для этой главы.

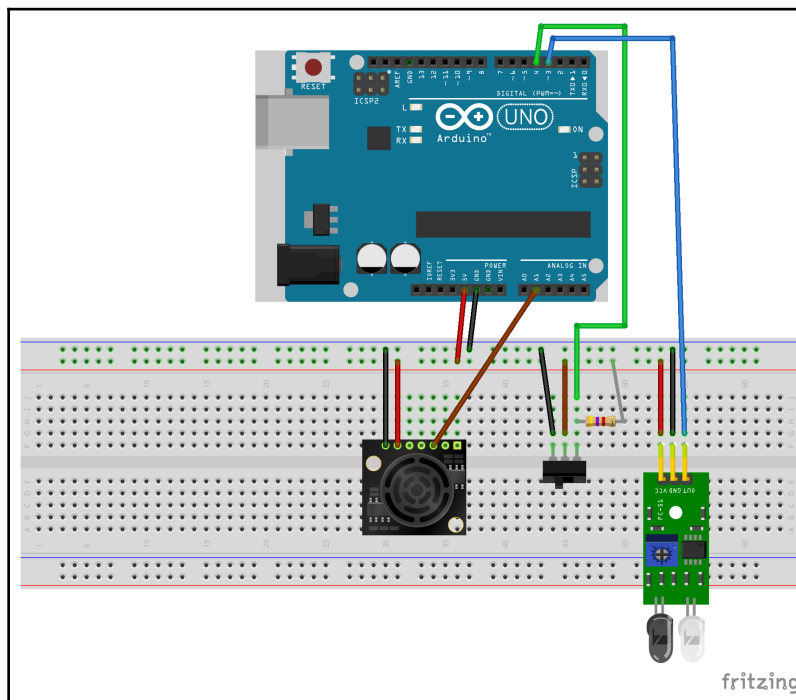
Ультразвуковой дальномер работает, посылая ультразвуковой импульс в определенном направлении. Если на пути импульса есть объект, тогда он отражается обратно в виде эха. Датчик определяет расстояние до объекта, измеряя время, необходимое для получения эхо-сигнала.

Ультразвуковой датчик EZ1 может обнаруживать и измерять расстояние до объекта от 0 до 6,45 метра (254 дюйма). Этот датчик практически не имеет мертвой зоны и обнаруживает объекты вплоть до передней поверхности датчика.

Для проекта этой главы нам потребуются следующие компоненты:

- Arduino Uno или совместимая плата
- Датчик столкновения
- Датчик предотвращения препятствий
- EZ1 Ультразвуковой датчик
- Резистор 4,7 кОм
- Перемычки
- Макетная плата

На следующем рисунке показана диаграмма Фритцинга для этого проекта:



Средний датчик, показанный на схеме, представляет собой датчик столкновения, поскольку для датчика столкновения нет детали Fritzing. Переключатель на схеме имеет такое же расположение контактов, что и датчик столкновения, показанный ранее в этой главе.

На схеме мы можем видеть, что все выводы заземления на датчиках подключены к шине заземления макета, а все выводы VCC на датчиках подключены к шине питания на макетной плате.

Аналоговый выход ультразвукового датчика дальномера EZ1 подключен к аналоговому выводу A1 на Arduino, датчик столкновения подключен к цифровому выводу 3, а инфракрасный датчик подключен к цифровому выводу 2.

Датчик столкновения также имеет подтягивающий резистор 4,7 кОм. Теперь, когда у нас есть датчики, подключенные к Arduino, давайте посмотрим на код этого проекта.

Мы начнем код с трех макросов, которые определяют выводы, к которым подключены три датчика. Макросы будут выглядеть так:

```
#define COLLISION_SWITCH 4
#define IR_SENSOR 3
#define RANGE_SENSOR A1
```

Эти макросы показывают, что датчик столкновения подключен к цифровому выводу 4, инфракрасный датчик подключен к цифровому выводу 3, а ультразвуковой датчик подключен к аналоговому выводу 1. Теперь нам нужно установить режим для двух цифровых выводов, которые мы используем, а также запустить последовательный мониторинг. Мы можем сделать это, добавив следующий код в функцию `setup ()`:

```
Serial.begin(9600);
pinMode(COLLISION_SWITCH, INPUT);
pinMode(IR_SENSOR, INPUT);
```

Программа начинается с запуска последовательного монитора, а затем настраивает выводы датчика аварийного сигнала и инфракрасного датчика для ввода, чтобы мы могли считывать значения. Теперь нам нужно добавить код к функции `loop ()`, которая будет считывать данные с датчиков. Давайте начнем с того, как мы будем читать и прерывать работу датчика удара:

```
int collisionValue = digitalRead(COLLISION_SWITCH);
if (isnan(collisionValue)) {
    Serial.println(" Failed to read collision sensor");
    return;
}
if (collisionValue == LOW) {
    Serial.println("Collision Detected");
}
```

Этот код начинается с использования функции `digitalRead ()` для считывания вывода, к которому подключен датчик столкновения, а затем с помощью функции `isnan ()` для проверки того, что функция `digitalRead ()` вернула правильное значение. Если значение, возвращаемое функцией, недействительно (не число), то на последовательную консоль выводится сообщение об ошибке, вызывается оператор `return` для выхода из этого цикла.

Если значение, возвращаемое функцией `digitalRead ()`, является допустимым, мы проверяем, является ли значение `LOW`, и если да, то было обнаружено препятствие, и на последовательную консоль выводится сообщение. Теперь добавим код инфракрасного датчика:

```
int irValue = digitalRead(IR_SENSOR);
if (isnan(irValue)) {
    Serial.println(" Failed to read infrared sensor");
}
```

```
    return;
}
if (irValue == LOW) {
    Serial.println("IR Detected");
}
```

Этот код в точности совпадает с кодом датчика столкновения, за исключением того, что мы считываем контакт инфракрасного датчика и проверяем это значение. Теперь добавим код ультразвукового датчика:

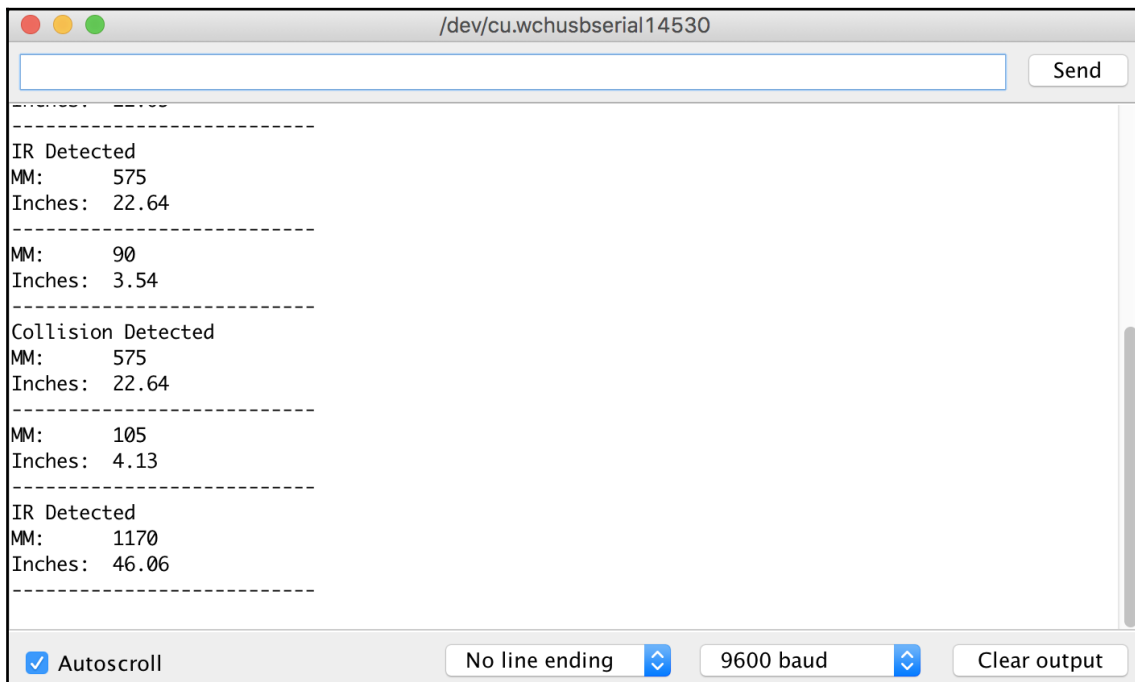
```
int anVolt = analogRead(RANGE_SENSOR);
if (isnan(anVolt)) {
    Serial.println(" Failed to read range sensor");
    return;
}
int mm = anVolt*5;
float inch = mm/25.4;
Serial.println(mm);
Serial.print("MM:   ");
Serial.println(mm);
Serial.print("Inches: ");
Serial.println(inch);
Serial.println("-----");
delay(1000);
```

Этот код начинается с использования функции `analogRead()` для считывания вывода, к которому также подключен ультразвуковой датчик. Затем мы используем функцию `isnan()`, чтобы убедиться, что было возвращено правильное значение.

Затем расстояние до объекта рассчитывается как в миллиметрах, так и в дюймах. Числа, использованные в расчетах, указаны в таблице данных датчика и могут отличаться в зависимости от модели, которую вы используете. Теперь мы хотим поставить короткую задержку в конце функции `loop()`, чтобы приостановить выполнение

Теперь запустим проект.

Когда мы запускаем этот проект, результат должен выглядеть примерно так, как на следующем снимке экрана:



На этом снимке экрана показано, что объект дважды сработал с инфракрасным датчиком, причем ИК-сигнал «Detected - Обнаружено» распечатывается на последовательной консоли. Датчик столкновения сработал один раз, а сообщение «Collision Detected - Обнаружено столкновение» выводится на последовательную консоль. Снимок экрана также показывает самое близкое расстояние до объекта, которое измерил дальномер.

Эта задача будет немного отличаться от большинства других. На самом деле нет никакого проекта, вместо этого это проблема мышления. Задача состоит в том, чтобы подумать о том, как все три этих датчика могут работать вместе, чтобы создать автономного робота. Для этого подумайте, как работают все три датчика:

1. : цифровой датчик, который срабатывает, когда датчик сталкивается с чем-либо.
2. : цифровой датчик, который срабатывает, когда что-то приближается
3. : аналоговый датчик, используемый для определения расстояния до объекта от датчика.

Вот ответы:

Ультразвуковые дальнометры на сегодняшний день являются самыми дорогими, поэтому я обычно использую только два из этих датчиков, обращенных наружу от передней части робота. Робот использует их для обхода препятствий. С возможностью определить, как далеко что-то находится от передней части робота, мы можем дать роботу логику, которая ему нужна, чтобы решить, когда повернуть, а также, с двумя ультразвуковыми датчиками, логику, чтобы решить, в какую сторону повернуть. Мы также можем использовать ультразвуковые датчики для отображения комнаты.

Инфракрасные датчики дешевы, их можно использовать по бокам и сзади, чтобы убедиться, что робот не наткнется ни на что при повороте или движении задним ходом. Поскольку они намного дешевле ультразвуковых датчиков, мы можем использовать несколько инфракрасных датчиков, чтобы обеспечить покрытие всей площади вокруг робота. Мы также можем использовать инфракрасные датчики, повернутые вниз, чтобы убедиться, что робот не съезжает с уступа.

Датчики столкновения также очень недороги и могут использоваться повсюду для робота, чтобы определить, врежется ли робот во что-нибудь, что пропустили ультразвуковые или инфракрасные датчики. Самая большая проблема с ультразвуковыми и инфракрасными датчиками - это их высота над роботом. Если они будут слишком высокими, они могут пропустить препятствия, расположенные низко над землей. Для их обнаружения можно использовать датчик столкновения.

В этой главе мы увидели, как использовать три датчика, которые можно использовать для предотвращения препятствий и обнаружения столкновений. Датчик столкновения - это цифровой датчик, который можно использовать, чтобы определить, когда датчик что-то задевает. Инфракрасный датчик предотвращения препятствий также является цифровым датчиком, который может определить, находится ли датчик на определенном расстоянии от препятствия. Ультразвуковой дальномер - это аналоговый датчик, который можно использовать для определения расстояния до препятствия.

В следующей главе мы рассмотрим несколько различных типов светодиодов и посмотрим, как мы можем использовать их в наших проектах.

Развлечения со светом

Большинство крупных проектов, которые мы создаем, будут использовать один или несколько светодиодов в качестве индикаторов. Эти светодиоды могут указывать на такие вещи, как питание, получение данных, предупреждения или что-то еще, для чего нам может потребоваться визуальная обратная связь. Мы уже видели, как использовать простой одноцветный светодиод, но что, если нам нужно несколько светодиодов или даже многоцветные светодиоды? В этой главе мы рассмотрим другие способы добавления светодиодов в ваш проект.

В этой главе вы узнаете:

- Что такое NeoPixels
- Как работает светодиод RGB
- Как использовать NeoPixels в ваших проектах
- Как использовать светодиод RGB в ваших проектах

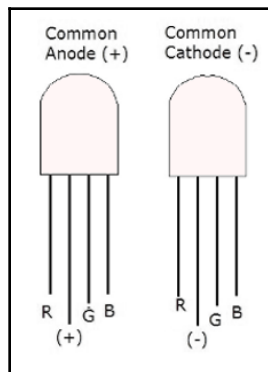
В этой главе мы рассмотрим, как использовать светодиоды RGB и экран WS2812 40 RGB LED PixelArduino.

Начнем с изучения светодиода RGB.

Многоцветный или RGB-светодиод - это на самом деле не один светодиод, который может менять цвет, на самом деле это три светодиода. Светодиод RGB состоит из трех светодиодов: красного, зеленого и синего цвета. Различные цвета светодиода - это сочетание цветов, производимых этими тремя светодиодами.

Есть два типа светодиодов RGB - с общим анодом и общим катодом. В светодиоде с общим катодом три светодиода имеют общий источник заземления, а в светодиоде RGB с общим анодом три светодиода имеют общий источник питания.

Светодиод RGB имеет четыре вывода, по одному для каждого цвета, а четвертый - общий катод (OK) или общий анод (OA). На следующей схеме показаны выводы для RGB-светодиода с общим катодом (Common Cathode -) и общим анодом (Common Anode +):



Чтобы получить различные цвета, мы можем отрегулировать интенсивность трех разных светодиодов с помощью выводов PWM на Arduino. Затем свет будет смешиваться вместе, потому что светодиоды расположены так близко, что дает нам нужный цвет. Теперь давайте посмотрим, что такое интегрированный источник света WS2812 или, как их называют на сайте Adafruit, NeoPixel. На протяжении большей части этой главы мы будем называть интегрированный источник света WS2812 NeoPixel, потому что он короче и звучит круто.

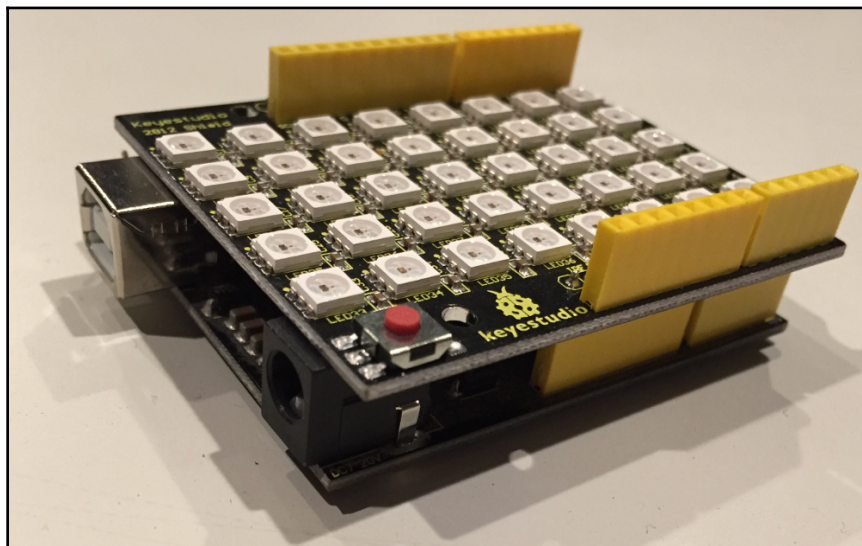
Как вы понимаете, если бы мы хотели включить в проект 10 светодиодов RGB, где для каждого светодиода требовалось три входных контакта, проект очень быстро превратился бы в клубок проводов. Не говоря уже о том, что у нас не хватит выводов на Arduino. Один из способов решить эту проблему - использовать NeoPixel. NeoPixel объединяет красный, зеленый и синий светодиоды вместе с микросхемой драйвера в крошечном корпусе для поверхностного монтажа. Этим пакетом можно управлять по одному проводу и использовать индивидуально или в группе. NeoPixel выпускается во многих форм-факторах, включая полосы, кольца, экраны Arduino и даже украшения.

Одна хорошая вещь в NeoPixel заключается в том, что нет внутреннего ограничения на количество NeoPixels, которые могут быть связаны вместе. Однако существуют некоторые практические ограничения, основанные на ограничениях оперативной памяти и мощности используемого вами контроллера.

В этой главе мы будем использовать экран NeoPixel. Если вы используете отдельные NeoPixels, вам нужно помнить о нескольких вещах:

- Перед подключением NeoPixels к источнику питания вам нужно добавить конденсатор на 1000 мкФ, 6,3 В или выше.
- Вы надо добавить резистор 470 Ом между выходом данных Arduino и входной линией на первом NeoPixel.
- По возможности избегайте подключения / отключения NeoPixels, когда схема активна. Если вы должны подключить их к схеме под напряжением, всегда сначала подключайте заземление. Если вам необходимо отключить их от схемы, находящейся под напряжением, всегда сначала отключайте питание 5 В.
- NeoPixels всегда следует запитывать от источника питания 5 В.

В этой главе мы будем использовать шилд Keyestudio 40 RGB LED 2812 Pixel Matrix. Этот шилд уже содержит конденсатор и резистор, поэтому все, что нам нужно сделать, это поместить шилд поверх Arduino Uno, и все готово. Шилд Keyestudio прикрепляется к Arduino, как показано на следующей фотографии:

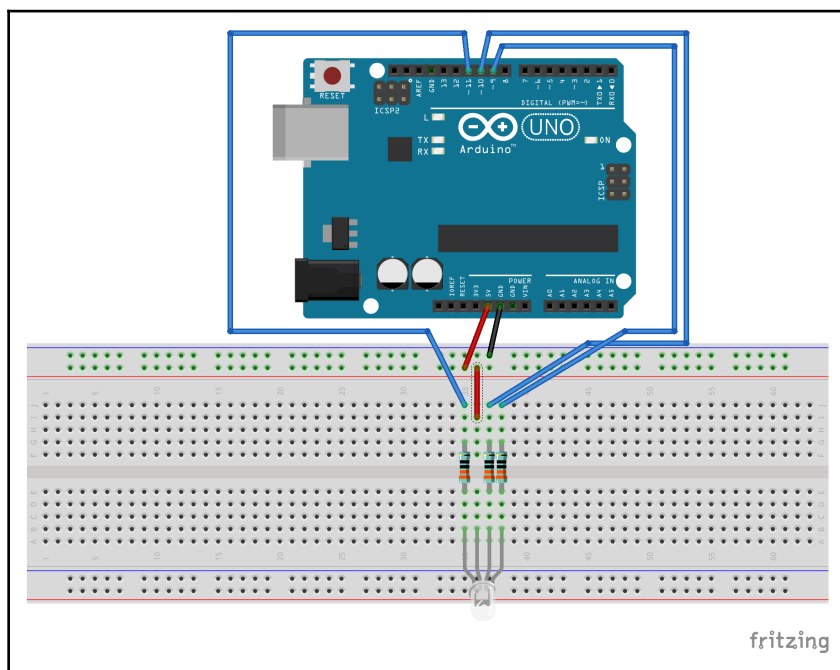


При использовании других форм-факторов NeoPixel всегда читайте паспорт производителя, прежде чем подключать его к Arduino. Повредить NeoPixels действительно легко, поэтому обязательно следуйте рекомендациям производителя.

Для проекта этой главы нам потребуются следующие компоненты:

- Arduino Uno или совместимая плата
- Один светодиод RGB с общим катодом (ОК) или общим анодом (ОА)
- Три резистора 330 Ом
- Keystudio 40 RGB LED 2812 Pixel Matrix экран
- Перемычки
- макетная плата

На следующей схеме показано, как подключить обычный светодиод RGB к Arduino:



На этой схеме мы показываем, как подключить обычный RGB-светодиод ОА, где общий контакт подключен к + шине питания на макетной плате. Если светодиод RGB, который вы используете, является светодиодом с общим катодом, подключите общий вывод светодиодов к шине заземления, а не шине питания. Каждый из контактов RGB подключен к контактам Arduino PWM с помощью резистора 330 Ом.

Мы не показываем принципиальную схему для экрана NeoPixel, потому что нам нужно только прикрепить шилд к Arduino. Теперь посмотрим на код.

Начнем с рассмотрения кода светодиода RGB.

RGB LED

Мы начнем с определения того, какие выводы Arduino подключены к выводам RGB светодиода:

```
#define REDPIN 11
#define BLUEPIN 10
#define GREENPIN 9
```

Этот код показывает, что красный контакт подключен к контакту Arduino 11 PWM, синий контакт подключен к контакту Arduino 10 PWM, а зеленый контакт подключен к контакту Arduino 9 PWM. Мы собираемся определить пустой макрос, который позволит коду приложения распознать, какой светодиод RGB подключен в схеме - с ОА или ОК. Следующий код сделает это:

```
#define COMMON_ANODE
```

Если вы используете обычный светодиод RGB с ОК, закомментируйте или удалите эту строку из своего кода. Мы увидим, как это использовать, когда посмотрим на функцию, которая устанавливает цвета светодиода. Теперь посмотрим на функцию `setup()`.

```
void setup() {
  pinMode(REDPIN, OUTPUT);
  pinMode(GREENPIN, OUTPUT);
  pinMode(BLUEPIN, OUTPUT);
}
```

Функция `setup()` установит режим вывода для выводов, которые подключены к выводам RGB светодиода. Это позволит нам использовать выводы ШИМ для установки интенсивности света трехцветных светодиодов, составляющих светодиод RGB. Далее нам нужно будет создать функцию, которая будет устанавливать эти цвета. Мы назовем эту функцию `setColor()`, и она будет принимать три параметра, которые будут определять интенсивность каждого светодиода RGB и содержать следующий код:

```
void setColor(int red, int green, int blue) {
    #ifdef COMMON_ANODE
        red = 255 - red;
        green = 255 - green;
        blue = 255 - blue;
    #endif
    analogWrite(REDPIN, red);
    analogWrite(GREENPIN, green);
    analogWrite(BLUEPIN, blue);
}
```

Код в этой функции начинается с оператора `#ifdef`. Этот оператор говорит, что если определен макрос `COMMON_ANODE`, то выполнить код между операторами `#ifdef` и `#endif`; в противном случае пропустить этот код. Следовательно, если мы определим макрос `COMMON_ANODE` в начале кода, то мы вычтем каждый параметр из 255, чтобы получить правильную интенсивность. Затем мы используем функцию `analogWrite()` для записи значений в выводы RGB.

В начале этой главы мы объяснили, что светодиод RGB работает, регулируя интенсивность каждого из трех светодиодов RGB, находящихся внутри светодиода RGB. Если мы запишем значение 255 на светодиод с общим катодом, то светодиод будет максимально ярким. Для светодиода с общим анодом нам нужно будет записать значение 0 для максимальной яркости светодиода. Вот почему мы вычитали значение каждого параметра на 255, если определен макрос `COMMON_ANODE`.

В функции `loop()` мы перебираем пару цветов, чтобы продемонстрировать, как светодиод отображает разные цвета. Ниже показан код функции `loop()`:

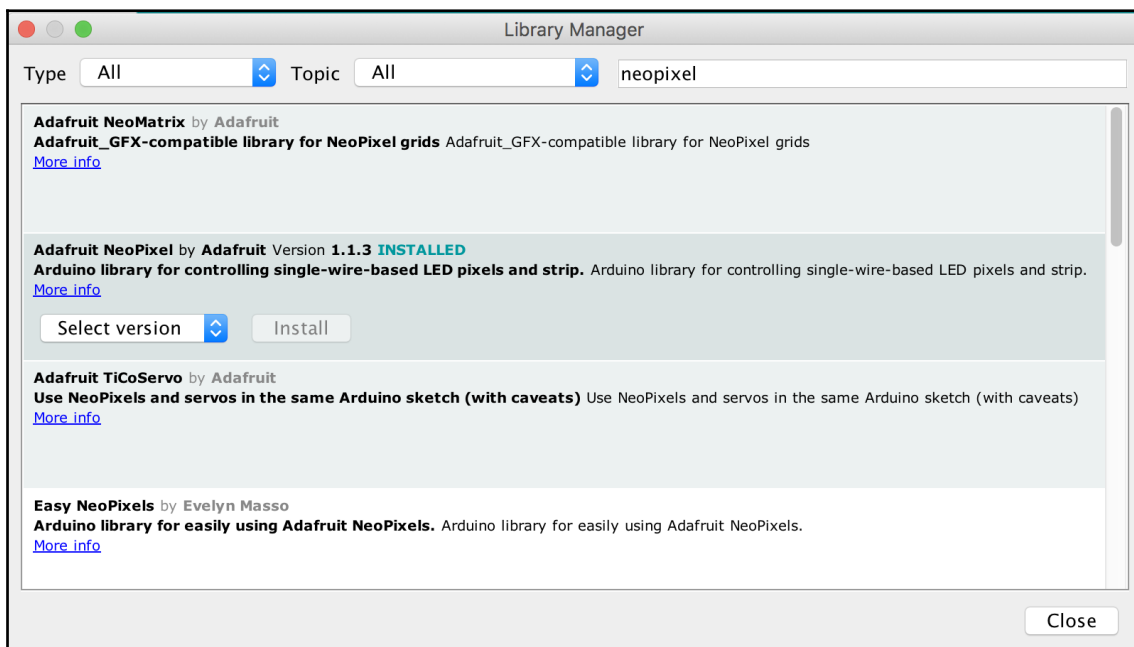
```
void loop() {
    setColor(255, 0, 0); // Red
    delay(1000);
    setColor(0, 255, 0); // Green
    delay(1000);
    setColor(0, 0, 255); // Blue
    delay(1000);
    setColor(255, 255, 255); // White
    delay(1000);
    setColor(255, 0, 255); // Purple
    delay(1000);
}
```


В функции `loop ()` мы пять раз вызываем функцию `setColor ()`, чтобы изменить цвет светодиода. Мы отображаем следующие цвета: красный, зеленый, синий, белый и фиолетовый. Каждый раз, когда цвет меняется, перед отображением следующего цвета будет пауза в одну секунду. Пауза от функции `delay ()`.

То, как мы отображаем цвета в светодиоде RGB, очень похоже на то, как мы зажигаем обычный светодиод, за исключением того, что мы определяем интенсивность света (яркость) для трех цветов. Теперь посмотрим на код шилда NeoPixel.

NeoPixel

Прежде чем мы начнем кодировать, нам нужно будет установить библиотеку Adafruit NeoPixel. На следующем снимке экрана показана библиотека, которую должен установить менеджер библиотек. Если вы не помните шаги по установке библиотеки, вернитесь к главе 9, «Датчики окружающей среды»:



Датчики, где мы устанавливаем библиотеку для датчика температуры и влажности DHT11.

После установки библиотеки нам нужно будет включить ее, поместив следующую строку вверху нашего кода:

```
#include <Adafruit_NeoPixel.h>
```

Когда мы используем библиотеку Adafruit NeoPixel, нам нужно сообщить ей, к какому выводу подключены NeoPixels и сколько подключено NeoPixel. Поэтому мы определим макросы, содержащие эти значения:

```
#define SHIELD_PIN 13  
#define MAX_PIXELS 40
```

Согласно таблице данных для шилда Keyestudio, экран подключается к выводу 13 на Arduino, а шилд содержит 40 NeoPixels; поэтому мы определяем эти значения в макросах. Теперь мы будем использовать эти значения для запуска экземпляра класса Adafruit_NeoPixelclass, как показано в следующем коде:

```
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(MAX_PIXELS, SHIELD_PIN,  
NEO_GRB + NEO_KHZ800);
```

Первый параметр - это количество пикселей в экране, а второй параметр - это точка, к которой подключены NeoPixels. Последний параметр - это флаг типа пикселя. Значения, показанные в этом примере, являются наиболее распространенными. Возможные значения:

- NEO_KHZ800: 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
- NEO_KHZ400: 400 KHz (classic v1 (not v2) FLORA pixels, WS2811 drivers)
- NEO_GRB: Pixels are wired for GRB bitstream (most NeoPixel products)
- NEO_RGB: Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)

В этом примере мы будем преобразовывать каждый пиксель, один за другим, в определенный цвет. Следовательно, нам потребуется глобальная переменная, указывающая на пиксель, на котором мы находимся, и еще одна глобальная переменная, чтобы определить, какой цвет использовать. Мы в этом примере будем использовать два цвета и переключаться между ними. Следующий код определяет эту глобальную переменную:

```
int num = 0;  
boolean color = 0;
```

В функции setup () нам нужно будет запустить NeoPixels. Следующий код показывает функцию setup () с кодом для запуска NeoPixels:

```
void setup() {  
  pixels.begin();  
  pixels.show();  
  pixels.setBrightness(50);  
}
```

Функция `begin()` подготавливает вывод данных на Arduino для вывода в NeoPixels. Функция `show()` выталкивает данные в NeoPixels и здесь не является абсолютно необходимой; Я считаю хорошей практикой включать функцию каждый раз, когда мы пишем что-либо в NeoPixels для полноты. Третья функция регулирует яркость пикселей. Я обычно устанавливаю 50%, потому что NeoPixels очень яркие.

Теперь давайте посмотрим на функцию `loop()`, которая установит цвет для каждого пикселя один за другим.

```
void loop() {
  num++;
  if (num > (MAX_PIXELS - 1)) {
    num = 0;
    color = !color;
  }
  if (color) {
    pixels.setPixelColor(num, 170, 255, 10);
  } else {
    pixels.setPixelColor(num, 10, 255, 170);
  }
  pixels.show();
  delay(500);
}
```

В функции `loop()` мы начинаем с увеличения переменной `num` на единицу, а затем проверяем, достигли ли мы последнего пикселя. Если мы достигли последнего пикселя, мы устанавливаем значение переменной `num` обратно в ноль и меняем местами переменную цвета. В строке `color = !Color` символ `!` - это оператор НЕ, который заставляет переменную цвета переключаться между истиной (`true`) и ложью (`false`). Это работает, потому что оператор НЕ возвращает значение, противоположное текущему значению цветовой переменной. Следовательно, если, например, цветовая переменная в данный момент была `false`, то операция `!Color` вернет `true`.

Затем мы используем функцию `setPixelColor()`, чтобы установить для текущего пикселя один из двухцветов в зависимости от того, является ли цветовая переменная истинной или ложной. Функция `setPixelColor()` существует в двух версиях. Версия, которую мы видим здесь, использует первый параметр как номер пикселя, который мы устанавливаем, а затем следующие три числа определяют интенсивность красного, зеленого и синего цветов, составляющих нужный нам цвет. Если мы использовали RGBWNeoPixel, нам также потребуется определить белый цвет. Следовательно, эта функция добавит дополнительный параметр, например:

```
setPixelColor(n, red, green, blue, white);
```

Второй способ вызвать функцию `setPixelColor()` - передать два аргумента, где первый - это номер пикселя, а второй - 32-битное число, которое объединяет значения красного, зеленого и синего цветов. Эта версия функции выглядит так:

```
setPixelColor(n, color);
```

Значение цвета может находиться в диапазоне от 0 до 16 777 216.

После того, как мы установили цвет пикселя, мы затем вызываем функцию `show()`, чтобы передать значения пикселям, а затем использовать функцию задержки, чтобы вставить полсекундную паузу в коде.

Если мы запустим скетч для светодиода RGB, мы увидим, что светодиод медленно переключается между пятью цветами. Код для NeoPixels будет переключать пиксели один за другим между двумя цветами.

Это будет одна из самых сложных задач в книге. Шилд NeoPixel фирмы Keyestudio обрабатывает несколько столбцов пикселей, где каждый столбец содержит пять пикселей, пронумерованные следующим образом:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40

Для этого задания установите каждый столбец на свой цвет и пусть цвета будут вращаться слева направо по кругу. Вот несколько советов для начала. Первая - это библиотека Adafruit NeoPixel, которая имеет функцию с именем `Color()`, которая возвращает 32-битный цвет на основе трех значений красного, зеленого и синего. Следовательно, вы можете использовать следующий код для преобразования 8-битного числа в 32-битный цвет.

```
uint32_t colorNum(int color) {  
    colorPos = 255 - colorPos;  
    if(colorPos < 85) {  
        return pixels.Color(255 - colorPos * 3, 0, colorPos * 3);  
    }  
}
```

```
if(colorPos < 170) {  
    colorPos -= 85;  
    return pixels.Color(0, colorPos * 3, 255 - colorPos * 3);  
}  
colorPos -= 170;  
return pixels.Color(colorPos * 3, 255 - colorPos * 3, 0);  
}
```

Затем мы могли бы использовать следующий код, который установит для всех пикселей в столбце их цвет:

```
for (int j=0; j<5; j++) {  
    int pixNum = (j*8) + i;  
    pixels.setPixelColor(pixNum, colorNum((tmpColorMode * 30) & 255));  
}
```

Переменная tmpColorMode - это число от 1 до 8, которое будет использоваться для выбора цвета для этого столбца.

Это должно дать вам основы, чтобы начать эту задачу. Ответ находится в загружаемом коде книги.

В этой главе мы узнали, как работают светодиоды RGB, как их использовать, и рассмотрели различия между светодиодами RGB с общим анодом и общим катодом. Мы также узнали, как работает WS2812 (NeoPixel) и как его использовать. NeoPixels бывают разных форм-факторов и могут использоваться практически везде, где требуется большое количество светодиодов RGB.

В следующей главе мы рассмотрим, как использовать небольшой зуммер с Arduino для воспроизведения звука.

Развлечения со звуком

Добавление звука в ваш роботизированный проект может стать отличием хорошего робота от классного. Только подумайте, каким милым был бы R2-D2 из фильма «Звездные войны», если бы он не издавал ни звука. Мы можем использовать звук не только для роботов. Например, мы можем добавить громкую сигнализацию, если датчик движения обнаруживает движение, или, может быть, мы просто хотим сыграть мелодию, когда температура у нас дома как на улице.

В этой главе вы узнаете:

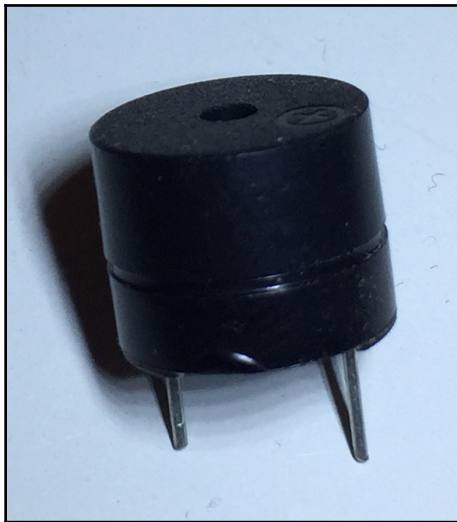
- Как подключить пьезозуммер к Arduino
- Как подключить динамик к Arduino
- Как использовать функцию `tone()` для создания звука
- Как играть музыку с Arduino

В этой главе мы выполним несколько проектов, в которых можно использовать пьезозуммер или небольшой 8-омный динамик. Используя и зуммер, и динамик, вы сможете услышать разницу между ними, чтобы определить, что подходит для нашего проекта.

Пьезозуммер компактный, надежный и очень недорогой. Их легче установить и использовать в большинстве электронных проектов, чем обычный динамик. Эти зуммеры могут издавать самые разные звуки, от тихого мычания до громких сигналов.

Пьезозуммер, иногда называемый пьезо-динамиком, издает звуки, немного отличные от обычного динамика. Рабочий компонент этих зуммеров представляет собой тонкий диск из пьезоэлектрического материала, обычно прикрепленный к металлической диафрагме. При приложении напряжения к пьезоэлектрическому материалу он деформируется. Это приводит к изгибу металлической диафрагмы вперед или назад. Эта деформация происходит очень быстро, заставляя керамический / металлический изгибающийся элемент вибрировать с частотой приложенного напряжения, что создает слышимый звук.

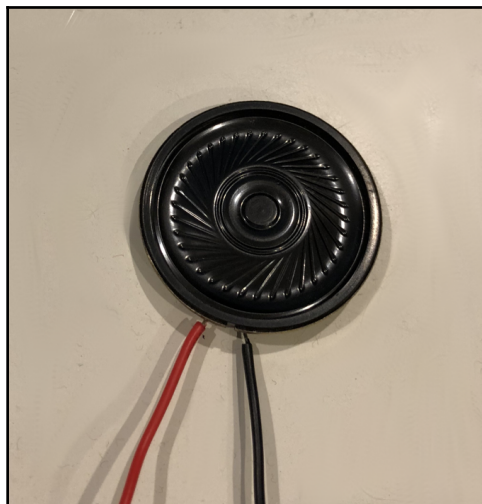
На следующей фотографии показано, как выглядит пьезозуммер:



Более короткий вывод должен быть подключен к земле, а более длинный вывод должен быть подключен к источнику питания.

Динамик на 8 Ом - это типичный динамик, который содержит электромагнит, который представляет собой металлическую катушку, которая создает магнитное поле при приложении электричества. При изменении направления катушки полюса магнита меняются местами. Этот электромагнит расположен перед обычным магнитом, полюса которого нельзя поменять местами. Направление тока, подаваемого на электромагнит, быстро меняется, в результате чего магниты притягиваются и отталкиваются друг от друга, создавая звук из конуса, подключенного к электромагниту.

На следующей схеме показано, как может выглядеть динамик на 8 Ом:

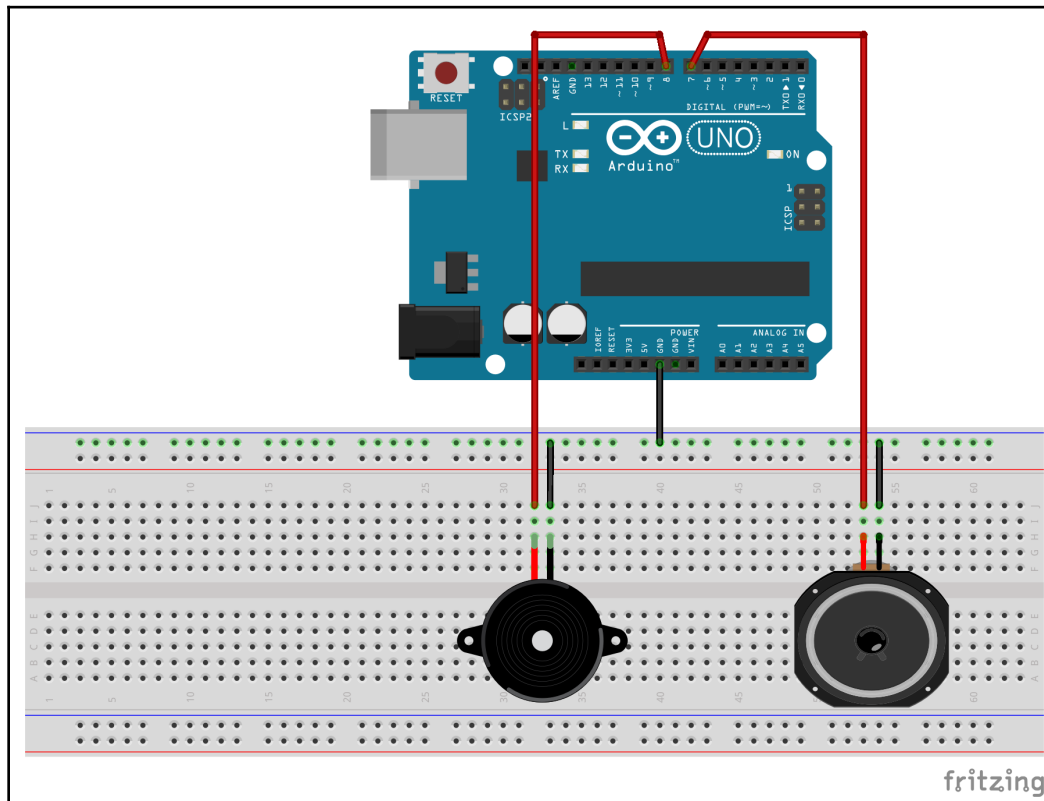


Теперь давайте посмотрим на компоненты, необходимые для этого проекта.

Для проекта этой главы нам потребуются следующие компоненты:

- Arduino Uno или совместимая плата
- пьезо-зуммер
- Динамик 8 Ом
- Перемычки
- макетная плата

Вот принципиальная схема, которую мы будем использовать для всех примеров кода в этой главе:



На этой схеме показано, что выводы земли динамика и пьезозуммера подключены к земляной шине на макетной плате. Плюс питания пьезозуммера подключается к контакту 8 на Arduino, а плюс питания динамика подключается к контакту 7 на Arduino.

Начнем с использования функции `tone()`.

Для первых нескольких примеров в этой главе мы будем использовать функцию Arduino `tone()`. Эта функция бывает двух разновидностей. Первая разновидность принимает два аргумента, где первый - это номер вывода, зуммер или динамик, а второй - это частота в герцах, на которой воспроизводится звук. Функция выглядит так:

```
tone(pinNumber, frequency);
```

Когда эта функция используется только с двумя параметрами, звук воспроизводится бесконечно. Следующий код показывает, как мы могли бы использовать эту функцию для воспроизведения ноты, используя предыдущую принципиальную схему:

```
#define PIEZOPIN 7
#define SPEAKERPIN 8

int soundPin = PIEZOPIN;

void setup() {
    tone(soundPin, 1000);
}
```

В этом коде функция `tone()` используется в функции `setup()` для воспроизведения на частоте 1000 Гц. Мы можем установить вывод звука либо на пьезозуммер, либо на вывод динамика, в зависимости от того, какой из них вы хотите использовать. Мы бы использовали эту версию функции `tone()`, если бы мы хотели воспроизводить звук непрерывно, пока не произойдет какое-либо взаимодействие с пользователем. Примером этого может быть воспроизведение звукового сигнала до тех пор, пока пользователь не отключит его.

Вторая разновидность этой функции принимает третий аргумент, который представляет собой продолжительность воспроизведения звука в миллисекундах. Эта функция выглядит так:

```
tone(pinNumber, frequency, duration);
```

Эту версию функции `tone()` можно использовать так:

```
#define PIEZOPIN 7
#define SPEAKERPIN 8

int soundPin = PIEZOPIN;

void setup() {
    tone(soundPin, 1000, 1000);
}
```

Этот код точно такой же, как и предыдущий, за исключением того, что звук воспроизводится только в течение одной секунды. Мы бы использовали эту версию функции тона, если бы мы хотели играть короткие ноты с определенной продолжительностью. Примером этого может быть воспроизведение песни, которую мы увидим в следующем примере.

Прежде чем мы сможем воспроизвести песню с помощью Arduino, нам нужно определить, на какой частоте воспроизводить разные ноты. Список частот достаточно большой и его можно скачать с кодом для скачивания этой книги. Файл с частотами называется `pitch.h`, а частоты определяются следующим образом:

```
#define NOTE_G6 1568
#define NOTE_GS6 1661
#define NOTE_A6 1760
#define NOTE_AS6 1865
#define NOTE_B6 1976
#define NOTE_C7 2093
```

Теперь давайте посмотрим, как мы можем использовать эти частоты для воспроизведения песни. Первое, что нам нужно сделать, это создать вкладку заголовков питчей с именем `pitches.h`, которая будет содержать частоты, а затем включать его на главной вкладке со следующей строкой:

```
#include "pitches.h"
```

Теперь нам нужно определить ноты или мелодию, из которых состоит песня. Эти ноты будут помещены в массив с именем `melody`:

```
int melody[] = {
  NOTE_E5, NOTE_E5, NOTE_E5, NOTE_E5, NOTE_E5, NOTE_E5,
  NOTE_E5,
  NOTE_G5,
  NOTE_C5,
  NOTE_D5,
  NOTE_E5,
  NOTE_F5, NOTE_F5, NOTE_F5, NOTE_F5, NOTE_F5,
  NOTE_E5, NOTE_E5, NOTE_E5, NOTE_E5, NOTE_E5,
  NOTE_D5, NOTE_D5,
  NOTE_E5,
  NOTE_D5,
  NOTE_G5
};
```

Каждая нота в песне должна воспроизводиться определенное время. Мы можем создать еще один массив, содержащий продолжительность каждой ноты, и назовем этот массив `tempo`:

```
int tempo[] = {
    4, 4, 2, 4, 4, 2,
    4,
    4,
    4,
    4,
    1,
    4, 4, 4, 4, 4,
    4, 4, 8, 8, 4,
    4, 4,
    4,
    2,
    2
};
```

Мы будем использовать функцию `tone ()` для создания заметок. С этой функцией нам не нужно ничего настраивать в функции `setup ()`. Следующий код можно поместить в функцию `loop ()` для воспроизведения песни, определенной массивами `melody` и `tempo` (темпа):

```
// Get the number of notes in the song
int songSize = sizeof(melody) / sizeof(melody[0]);

//Loop through each note
for (int note = 0; note < songSize; note++) {

    //Calculate how long to play the note
    int noteDuration = 1000 / tempo[note];

    //Play the note
    tone(soundPin, melody[note], noteDuration);

    //Calculate how long to pause before playing next note
    int pauseBetweenNotes = noteDuration * 1.20;
    delay(pauseBetweenNotes);
}
delay(3000);
```

Этот код начинается с вычисления количества нот в массиве `melody` путем деления размера массива `melody` на размер первого элемента в массиве. Мы используем эту логику для вычисления количества элементов в массиве, потому что код `sizeof (melody)` возвращает количество байтов, занятых массивом, а `sizeof (melody [0])` возвращает количество байтов, занятых первым элементом в массиве. Для хранения одного целого числа требуется два байта, а в массиве `melody` 26 нот. Следовательно, размер кода (мелодии) вернет 52, а код `sizeof (мелодия[0])` вернет 2.

Цикл `for` используется для циклического перебора массивов `melody` и временного интервала. Внутри цикла `for` длительность ноты рассчитывается путем деления одной секунды на тип ноты (элементы в массиве темпа), где четвертная нота равна 1000, деленной на 4, а восьмая нота равна 1000, деленной на 8.

Функция `tone` используется для воспроизведения ноты из массива `melody` в течение рассчитанной длительности. Функция `tone` не заставляет приложение приостанавливаться во время воспроизведения ноты. Поэтому нам нужно создать собственную паузу. Мы также организуем паузу немного дольше, чем продолжительность ноты, чтобы сделать небольшую паузу между нотами. Для этого мы умножаем длительность ноты на 1,2, а затем используем функцию `delay ()`. После завершения цикла `for` есть еще одна задержка на три секунды перед тем, как начать заново.

Этот последний пример показывает, как мы можем воспроизвести песню с помощью функции `tone ()` с двумя массивами, один для нот и один для темпа. Теперь давайте посмотрим, как мы можем использовать библиотеку, которая позволит нам воспроизводить музыку в формате **RTTTL (Ring Tone Text Transfer Language)**. Формат RTTTL был разработан Nokia для передачи рингтонов на мобильные телефоны.

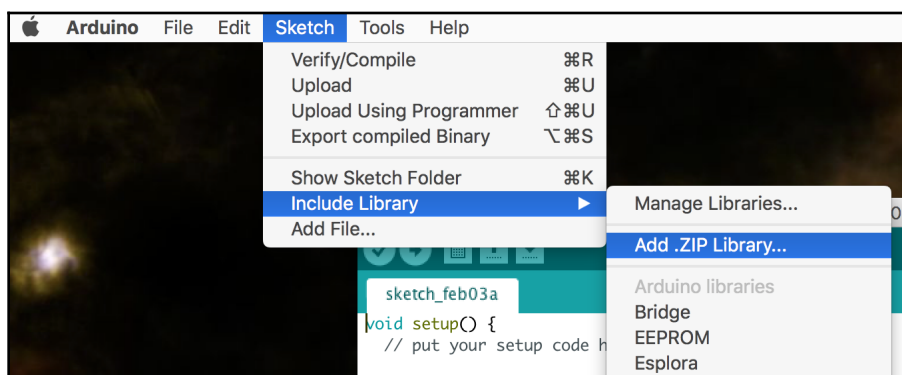
RTTTL

В диспетчере библиотек Arduino нет библиотеки, которую мы можем загрузить для воспроизведения файлов RTTTL в настоящее время. Поэтому нам нужно будет скачать и вручную установить библиотеку. Мы будем использовать `Arduino-rtttl-player`, который можно скачать здесь:

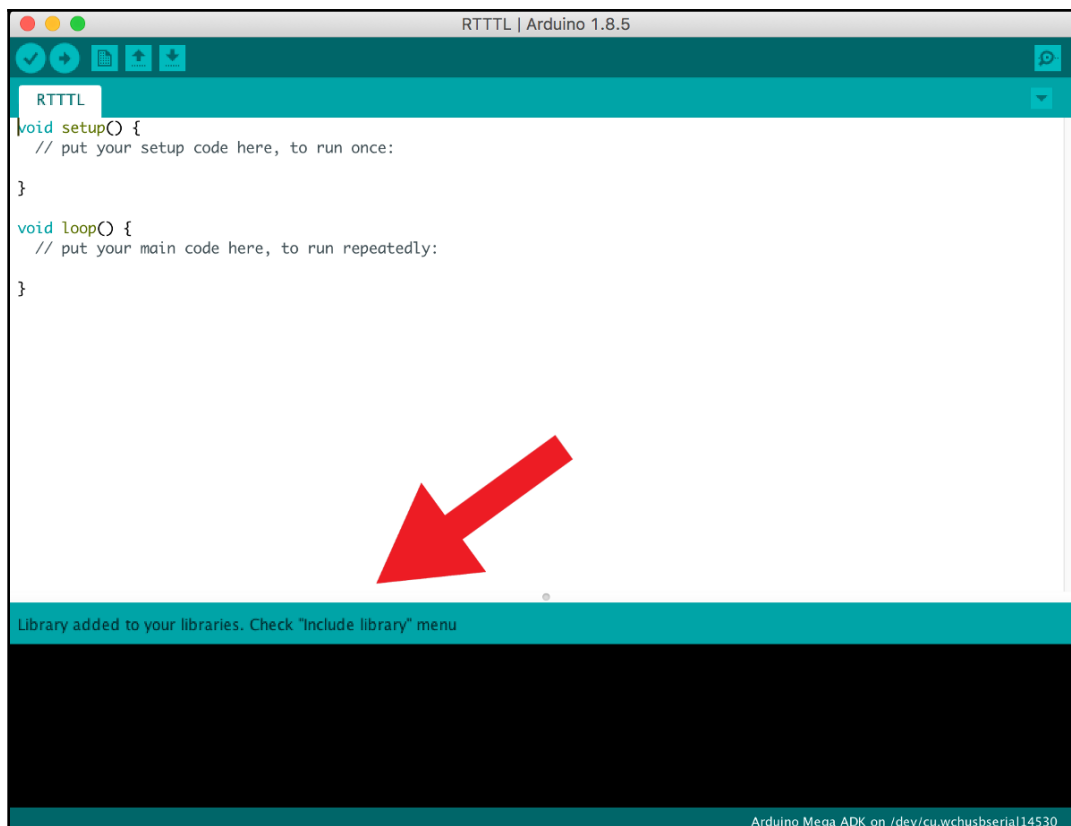
<https://github.com/ponty/arduino-rtttl-player>. Нам нужно будет создать ZIP-файл библиотеки, чтобы загрузить его в IDE. Если у вас нет доступа к утилите, которая может заархивировать файлы, загружаемый код для этой книги содержит уже заархивированную библиотеку.

Когда мы создаем ZIP-файл для загрузки в Arduino IDE, мы не хотим заархивировать все, что загружается из репозитория GitHub, потому что Arduino IDE не распознает ZIP-файл как файл библиотеки. Мы хотим заархивировать только каталог, содержащий код для библиотеки, а в случае библиотеки Arduino-rtttl-player это будет папка rtttl.

После того, как мы загрузим библиотеку и создадим ZIP-файл, содержащий папку rtttl из библиотеки, мы загрузим библиотеку в IDE Arduino. Для этого нам нужно выбрать Sketch | Include Library | Add .ZIP Library ... из главного меню, как показано на следующем снимке экрана:



После того, как вы выберете опцию «Add ZIP Library - Добавить ZIP-библиотеку», вам будет предложено средство выбора файлов, в котором вы можете найти местоположение созданного вами ZIP-файла и выбрать его. Если библиотека была успешно импортирована, вы увидите сообщение на панели сообщений, как показано на следующем снимке экрана:



Теперь мы готовы сыграть мелодию RTTTL. Первое, что нам нужно сделать, это включить библиотеку в проект, добавив к скетчу следующий оператор `include`:

```
#include <rtttl.h>
```

Мы включим пьезозуммер и динамик, как мы это делали в предыдущих проектах, со следующим кодом:

```
#define PIEZOPIN 7
#define SPEAKERPIN 8

int soundPin = PIEZOPIN;
```

Нам нужно будет определить песню для воспроизведения. В Интернете есть множество кодов RTTTL. Чтобы найти их, выполните поиск `rtttl songs` (песен `rtttl`), и вы увидите множество кодов RTTTL для самых разных песен. Для этого примера мы будем играть в тему «Звездных войн». Следующий код содержит для этого код RTTTL:

```
char *song = "Star Wars:d=8,o=5,b=180:f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,d,c,2a#.,4f.,d#,d,d#,2c,4p,f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,d,c,2a#.,4f.,d#,d,d#,2c";
```

Чтобы воспроизвести эту мелодию, добавьте следующий код в функцию `setup()`:

```
Rtttl player;
player.begin(soundPin);
player.play(song, 0);
```

Мы используем функцию `begin` из библиотеки `Arduino-rtttl-player`, чтобы запустить библиотеку и определить, к какому выводу подключен динамик, а затем функцию воспроизведения для воспроизведения песни. Второй параметр в функции воспроизведения - октава. Чем выше октава, тем выше высота звука, на котором будет играть песня. Я обычно оставляю это значение на нуле.

Когда этот код будет запущен, вы должны распознать тему «Звездных войн».

В этой задаче мы остановимся на теме «Звездных войн». Допустим, мы хотели создать робота, похожего на R2-D2 из «Звездных войн». Одна из функций, которые мы бы добавили, - сделать так, чтобы он звучал как R2-D2. Как бы вы сделали робота похожим на R2-D2?

В этой главе мы увидели, как подключить динамик и пьезозуммер к Arduino. Затем мы узнали, как использовать функцию `tone ()` для создания звуков, а также для воспроизведения музыки. Мы также увидели, как можно установить и использовать стороннюю библиотеку, чтобы мы могли воспроизводить файлы RTTTL.

В следующей главе мы рассмотрим, как можно использовать ЖК-дисплеи для отображения сообщений.

Использование ЖК-дисплеев

11

Бывают случаи, когда мы хотели бы иметь возможность отображать данные с Arduino пользователю. Для этого мы можем использовать ЖК-дисплей. Существует множество типов ЖК-дисплеев, которые мы можем использовать и наверное, самым популярным является дисплей 1602. Поскольку он очень популярен, вы можете найти множество руководств по его использованию в Интернете и вы ограничены в том, что вы можете с ним делать.

В этой главе мы рассмотрим дисплей, с которым мы можем сделать гораздо больше. Это ЖК-дисплей Nokia 5110.

В этой главе вы узнаете:

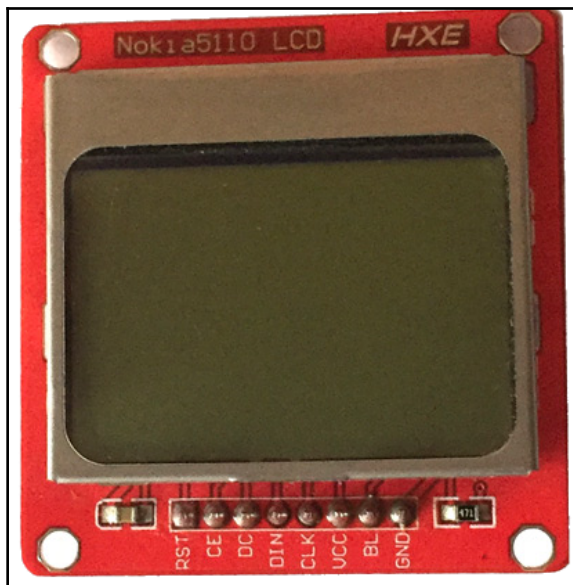
- Как подключить ЖК-дисплей Nokia 5110 к Arduino
- Как печатать текст на ЖК-дисплее
- Как рисовать окружности на ЖК-дисплее
- Как рисовать прямоугольники на ЖК-дисплее
- Как рисовать прямоугольники со скругленными углами на ЖК-дисплее

ЖК-дисплей Nokia 5110 - это простой ЖК-экран с монохромной графикой, который можно использовать во многих проектах. Nokia изначально разработала этот дисплей для использования в сотовых телефонах в конце 1990-х годов. В этом дисплее используется контроллер / драйвер ЖК-дисплея PCD8544.

Наличие ЖК-дисплея значительно улучшает пользовательский интерфейс любого проекта, позволяет отображать как текст, так и графику.

ЖК-дисплей 5110 имеет площадь дисплея примерно 4,2 см с разрешением 84 × 48 отдельных пикселей. Дисплей недорогой и очень простой в использовании с библиотекой ЖК-дисплея Adafruit 5110, которую мы будем использовать в этой главе.

ЖК-дисплей 5110, который мы будем использовать в этой главе, выглядит следующим образом:



ЖК-дисплей 5110 устанавливается на печатной плате и имеет восемь контактов, которые используются для питания дисплея и как интерфейс. Эти пины слева направо:

1. **RST**: сброс - активный низкий уровень
2. **CE**: Chip Select - Активный низкий уровень
3. **DC**: выбор режима (данные / инструкция) - выбор между командным режимом (ноль) или режимом данных (1).
4. **DIN**: последовательные данные в строке
5. **CLK**: Serial Clock Line - Последовательная линия синхронизации
6. **VCC**: входное напряжение 3,3 В
7. **BL**: Управление светодиодной подсветкой - 3,3 В
8. **GND**: земля

- Arduino Uno или совместимая плата
- Nokia 5110 ЖК-дисплей
- Четыре резистора 10 кОм
- Один резистор на 1 кОм
- Перемычки
- макетная плата

[166]

В следующей таблице показано, какие контакты ЖК-модуля 5110 подключаются к Arduino:

5110	Arduino
RST	3
CE	4
DC	5
DIN	11
CLK	13
VCC	3.3V out
BL	GND
GND	GND

Подсветка установлена на массу, чтобы выключить ее. Если вы хотите использовать подсветку, вы можете подключить ее на +3,3 В.

Теперь давайте посмотрим, как мы можем отображать элементы на ЖК-дисплее.

Нам нужно будет начать с установки двух библиотек Adafruit. Это **Adafruit GFXLibrary** и **Adafruit PCD8544 Nokia 5110 LCD**, а также библиотеку **SPI library**. Мы можем сделать это, добавив следующие операторы включения в начало скетча:

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
```

Мы иницилируем экземпляр типа Adafruit_PCD8544, используя следующий код:

```
Adafruit_PCD8544 display = Adafruit_PCD8544(13, 11, 5, 4, 3);
```

Параметры - это номера выводов Arduino, к которым соответственно подключены выводы CLK, DIN, DC, CE и RST.

В функции setup () мы добавим следующий код для настройки экземпляра Adafruit_PCD8544 :

```
Serial.begin(9600);

display.begin();
```

```
display.setContrast(40);
```

Теперь остальной код можно передать в функцию `setup()` для тестовых целей или в функцию `loop()`. Давайте начнем с того, что посмотрим, как осветить один пиксель на дисплее. Это можно сделать с помощью функции `drawPixel()`, как показано в следующем коде:

```
display.clearDisplay();  
display.drawPixel(10, 10, BLACK);  
display.display();
```

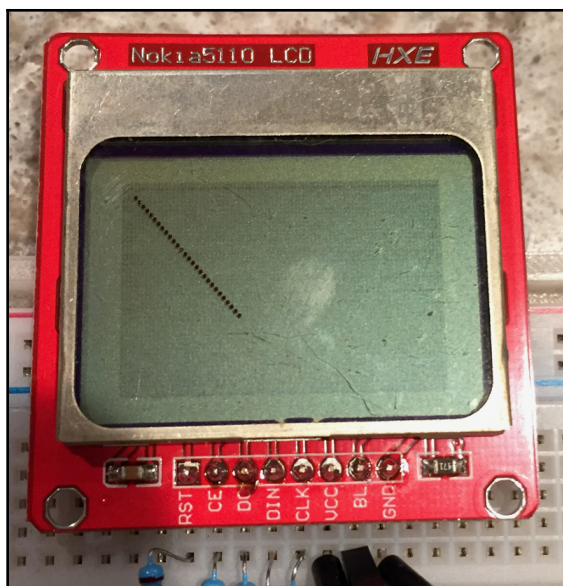
Прежде чем рисовать что-либо на экране, нам нужно очистить дисплей и буфер. Мы делаем это с помощью функции `clearDisplay()`. Затем мы используем функцию `drawPixel()`, чтобы осветить один пиксель, расположенный в координате X 10 и координате Y 10. Прежде чем что-либо отобразится на ЖК-дисплее, нам нужно запустить функцию `display()`, как показано в предыдущем коде. Важно помнить, что нужно запускать функцию `clearDisplay()` до того, как мы что-либо отрисовываем на ЖК-дисплее, и мы запускаем функцию `display()` после того, как отрисовываем все на экране для отображения.

Мы могли бы объединить несколько вызовов функции `drawPixel()`, чтобы нарисовать линию, но было бы намного проще использовать функцию `drawLine()`, как показано в следующем коде:

```
// draw a line  
display.drawLine(3,3,30,30, BLACK);  
display.display();
```

Функция `drawLine()` принимает пять параметров. Первые два параметра - это координаты X / Y начальной точки линии. Следующие два параметра - это координаты X / Y для конечной точки линии, а последний параметр - это цвет для рисования линии. Поскольку ЖК-дисплей Nokia 5110 является монохромным, здесь можно выбрать только ЧЕРНЫЙ или БЕЛЫЙ цвет.

Если бы мы запустили этот код, мы бы увидели на дисплее строку, подобную той, что показана на следующей фотографии:



Библиотека Adafruit также упрощает отображение текста на ЖК-дисплее Nokia 5110. Следующий код показывает, как мы можем отображать текст:

```
// Display text
display.setTextSize(1);
display.setTextColor(BLACK);
display.setCursor(0,0);
display.println("Hello, world!");

// Display Reverse Text
display.setTextColor(WHITE, BLACK);
display.println(3.14);

// Display Larger Text
display.setTextSize(2);
display.setTextColor(BLACK);
display.print("This is larger text");
display.display();
```


Функция `setTextSize ()` устанавливает размер текста. В первом примере размер текста установлен на 1. Функция `setTextColor ()` устанавливает цвет текста. Еще раз, поскольку ЖК-дисплей Nokia 5110 является монохромным, есть два варианта - ЧЕРНЫЙ или БЕЛЫЙ. Функция `setCursor ()` устанавливает положение курсора в положение на экране для написания текста.

В этом случае курсор устанавливается в левый верхний угол экрана. Наконец, функция `println ()` используется для вывода сообщения Hello World!

В следующем примере мы используем функцию `setTextColor ()`, чтобы установить цвет переднего плана на БЕЛЫЙ, а цвет фона на ЧЕРНЫЙ, чтобы перевернуть текст, а затем с помощью функции `println ()` вывести значение PI на экран. Поскольку мы не вызывали функцию `setTextSize ()`, размер текста остается ранее определенного размера, равного 1.

В последнем примере размер текста установлен на 2, а цвет текста снова установлен на черный. На следующем изображении показано, что будет отображаться на экране при запуске этого кода:

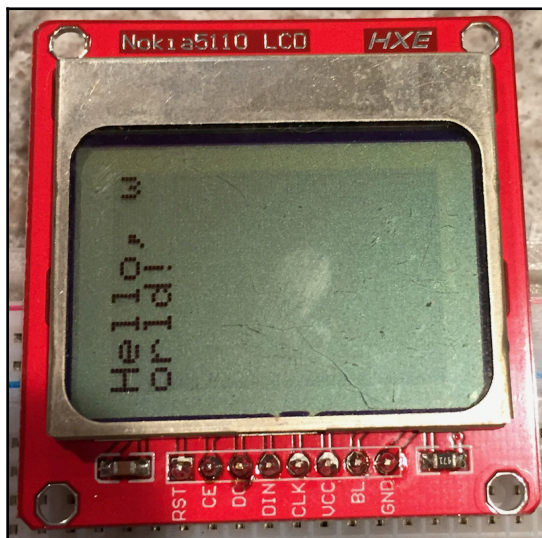


Мы также можем вращать текст. Следующий код показывает, как это сделать:

```
display.setRotation(1);  
display.setTextSize(1);  
display.setTextColor(BLACK);  
display.setCursor(0,0);  
display.println("Hello, world!");  
display.display();
```

Функция `setRotation()` будет вращать текст против часовой стрелки. Значение 1 поворачивает текст на 90 градусов против часовой стрелки. Значения 2 и 3 также можно использовать для поворота текста на 180 и 270 градусов.

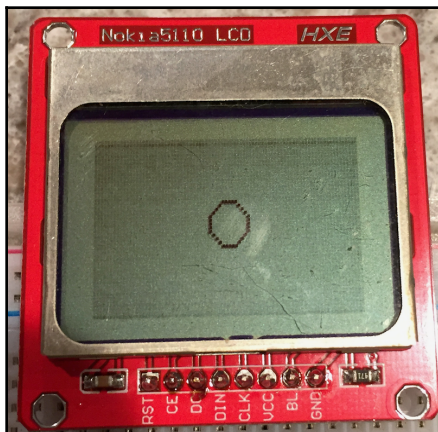
На следующей фотографии показано, как будет выглядеть текст при запуске этого кода:



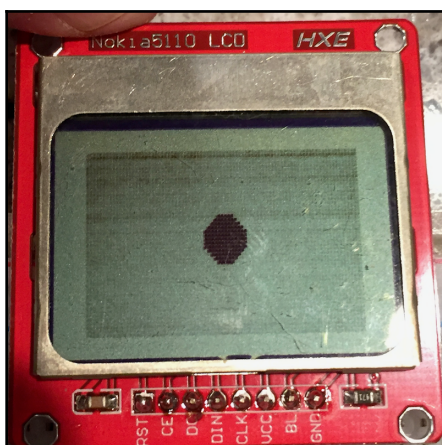
Обратите внимание, что текст будет перенесен на новую строку, если он длиннее, чем может отображаться в одной строке.

Библиотека Adafruit также позволяет нам создавать основные формы на ЖК-дисплее. К ним относятся круги, прямоугольники и прямоугольники с закругленными углами. Существуют также функции, которые позволяют нам создавать эти формы и заполнять их. Следующий код и снимки экрана показывают, как использовать функции круга:

```
display.drawCircle(display.width()/2, display.height()/2, 6, BLACK);
```



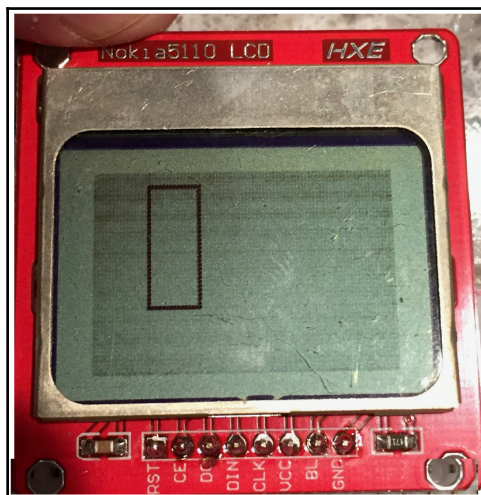
```
display.fillCircle(display.width()/2, display.height()/2, 6, BLACK);
```



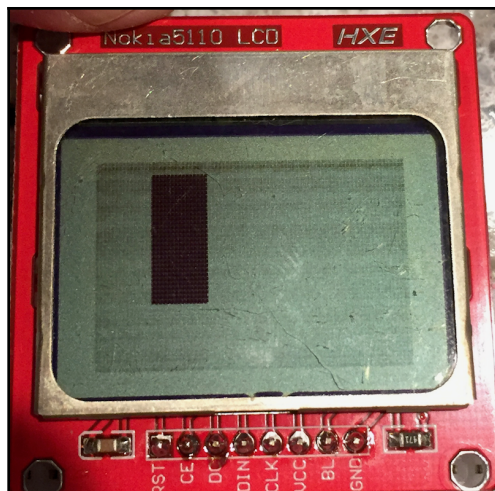
Функции круга принимают четыре параметра. Первые два параметра - это координаты X / Y центра круга. В этих двух примерах центр круга является центром экрана. Третий параметр - это радиус круга, а последний параметр - это цвет круга, а также цвет заливки круга в случае функции fillCircle().

В следующих двух примерах показано, как рисовать прямоугольник, а также прямоугольник с заливкой:

```
display.drawRect(15,15,30,15,BLACK);
```



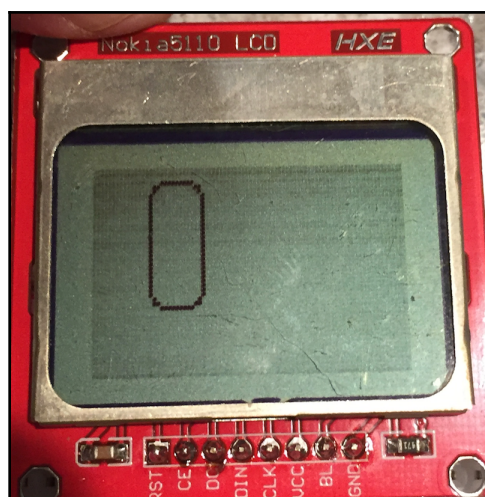
```
display.fillRect(15,15,30,15,BLACK);
```



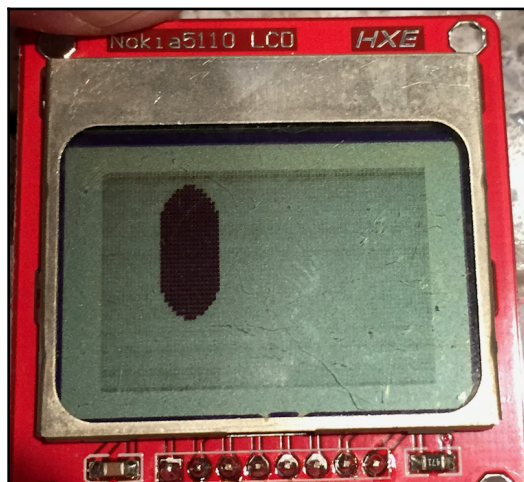
Функции прямоугольника принимают пять параметров. Первые два - это координаты X / Y верхнего левого угла прямоугольника. Следующие два параметра - это координаты X / Y нижнего правого угла прямоугольника, а последний параметр - это цвет для рисования прямоугольника и цвет для заполнения прямоугольника для функции `fillRect()`.

В следующих двух примерах показано, как рисовать прямоугольники с закругленными углами с помощью библиотеки Adafruit:

```
display.drawRoundRect(15,15,30,15,4,BLACK);
```



```
display.fillRoundRect(15, 15, 30, 15, 8, BLACK);
```



Первые четыре параметра для функций круглого прямоугольника такие же, как и функции обычного прямоугольника, которые являются координатами левого верхнего и правого нижнего углов прямоугольника. Следующий параметр - это степень закругления углов, а последний параметр - цвет для рисования прямоугольника с закругленными углами и цвет для его заливки.

Как видно из примеров в этой главе, с ЖК-дисплеем Nokia 5110 мы можем сделать гораздо больше, чем просто текст, а библиотека Adafruit делает его очень простым в использовании.

Для испытания возьмите любой из предыдущих проектов в этой книге и добавьте к нему ЖК-дисплей Nokia 5110. Затем вместо вывода на последовательную консоль отобразите вывод на ЖК-дисплей. Примером может служить добавление ЖК-дисплея к проекту дальномера из главы 10 «Избегание препятствий и обнаружение столкновений» и использование ЖК-дисплея для отображения расстояния.

В этой главе мы увидели, как добавить в наши проекты монохромный ЖК-дисплей Nokia 5110. Эти дисплеи могут значительно улучшить взаимодействие с пользователем практически любого проекта, потому что мы можем сообщить пользователям, что происходит и что не так, если есть проблема.

В следующей главе мы увидим, как добавить в наши проекты синтезатор голоса и распознавание голоса.

12

Любой, кто использовал Amazon Echo, динамик Google Home или даже Siri от Apple, знает, насколько мощным и удобным может быть распознавание речи и синтез голоса. А теперь представьте, можно ли добавить эти функции в наши умные устройства в меньшем масштабе? Если бы мы могли, у нас была бы возможность напрямую поговорить с нашим кофейником и сказать ему, чтобы он начал варить кофе утром, или командовать роботами, которых мы будем создавать.

В этой главе мы рассмотрим, как можно добавить распознавание и синтез голоса в любые проекты Arduino с помощью шилда MOVI. В этой главе мы узнаем:

- Как использовать шилд MOVI для распознавания речи
- Как использовать шилд MOVI для синтеза голоса
- Как создать термометр с голосовым управлением

Название MOVI расшифровывается как «**My Own Voice Interface** - Мой собственный голосовой интерфейс». Шилд MOVI Arduino от Audeme - чрезвычайно простой в использовании шилд распознавания речи и синтеза голоса. Этот шилд будет работать напрямую с Ардуино Uno R3, Duemilanove, Mega 2560 или Leonardo. **Однако вы не должны подавать питание на плату через разъем USB, пока подключен шилд MOVI.** Для шилда MOVI требуется минимум 7 В. Следовательно, вы можете повредить MOVI и / или Arduino, если попытаетесь включить его через USB-соединение.



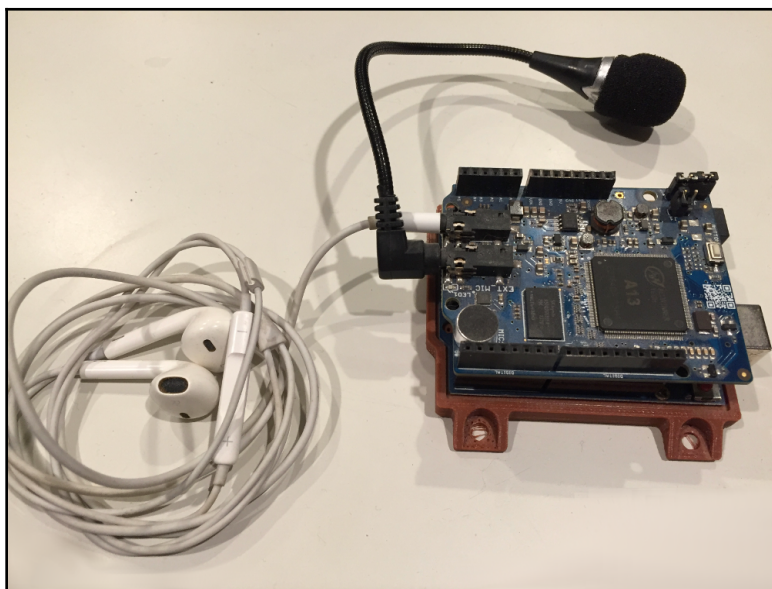
Вы можете прочитать о MOVI Shield и загрузить руководство пользователя с веб-сайта Audeme здесь: <https://www.audeme.com>.

При включении и программировании Arduino с подключенным MOVI вам понадобится питание Arduino через входной разъем источника постоянного тока, используя вход 9 В. После того, как платы включены, вы можете подключить кабель USB между Arduino и вашим компьютером для программирования. Поскольку плата изначально питалась от входного разъема источника постоянного тока, она будет продолжать получать питание от этого источника, а не через USB-соединение.



Очень важно: не подавайте питание на Arduino, когда к нему подключен шилд MOVI, через разъем USB.

При программировании MOVI рекомендуется подключить внешний микрофон для лучшего распознавания голоса и наушники, чтобы вы могли слышать ответы MOVI. На следующей фотографии показан MOVI, подключенный к Arduino с внешним микрофоном и наушниками, подключенными к MOVI:



На плате MOVI имеется встроенный микрофон, который можно использовать вместо внешнего. Однако встроенного динамика нет. Для получения звуковой обратной связи, включающей ошибки и системные сообщения, необходимо подключить к шилду MOVI наушники или внешний динамик. Сопротивление динамика должно составлять 32 Ом, что является стандартом для наушников. Не следует подключать динамик на 4 или 8 Ом к разъему для внешнего динамика.

MOVI можно использовать как замену кнопкам, пультам дистанционного управления или любому другому управляющему входу. Как мы увидим в примере проекта для этой главы, мы можем использовать MOVI для выдачи голосовых команд, на которые Arduino может реагировать.

Одной из лучших особенностей шилда MOVI Shield является отсутствие необходимости в подключении к Интернету, что снимает любые проблемы с конфиденциальностью, которые обычно связаны с другими устройствами голосового управления, такими как Amazon для внешних серверов.

На шилде MOVI имеется светодиод, который указывает его состояние. В следующем списке показаны различные состояния, в которых может находиться шилд MOVI, и связанное с ним состояние светодиода:

- **LED** : означает, что экран выключен или для работы MOVI недостаточно энергии.
- **LED** : MOVI загружается.
- **LED** : MOVI записывается на SD-карту.
- **LED** : возможно, проблема с SD-картой.
- **LED горит**: означает, что MOVI включен и готов к работе.

Шилд MOVI - одна из самых интересных и забавных плат, которые вы можете использовать с Arduino. Если вас интересуют более сложные вещи, которые вы можете с ним делать, вам следует взглянуть на примеры, поставляемые с библиотекой MOVI.

В этой главе мы создадим термометр с голосовым управлением, используя датчик температуры DHT-11, который мы использовали в главе 9 «Датчики окружающей среды» и шилд MOVI. Чтобы подключить датчик температуры и шилд MOVI к Arduino, нам нужно сначала прикрепить шилд MOVI к Arduino, а затем подключить датчик температуры DHT-11 к контактным разъемам на шилде MOVI.

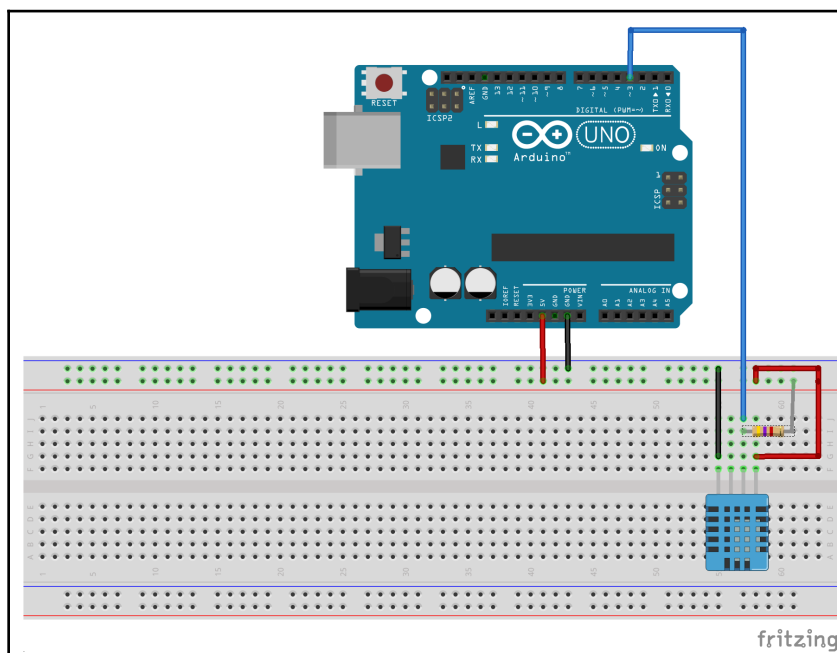
Давайте посмотрим на компоненты, которые нам понадобятся для этого проекта.

В этой главе вам потребуются следующие компоненты.

- Arduino Uno или совместимая плата
- Источник питания 9 В, например сетевой адаптер.
- Шилд MOVI
- Датчик температуры / влажности DHT-11

- Один резистор 4,7 кОм
- Перемычки
- Макетная плата

На следующей схеме показано, как подключить датчик температуры DHT-11 для этого проекта. Не забудьте прикрепить шилд **MOVI** к **Arduino** перед подключением датчика температуры. Резистор, показанный на следующей схеме, представляет собой резистор 4,7 кОм:



Теперь давайте посмотрим на код нашего голосового датчика температуры.

Код

У вас уже должна быть загружена библиотека DHT-11 из примера в главе 9 «Датчики окружающей среды», но вам нужно будет загрузить библиотеку **MOVI**. Если вы перейдете к менеджеру библиотеки и выполните поиск **Moví**, вы найдете несколько библиотек, соответствующих этому термину. Найдите библиотеку **MOVI Voice Dialog Shield** от **Audeme LLC** и загрузите ее.

Мы начнем скетч с включения библиотек MOVI и DHT. Следующий код показывает, как их включить:

```
#include <DHT.h>
#include <MOVIShield.h>
```

Затем мы определим вывод / тип DHT и создадим экземпляр типа DHT, как мы это делали в главе 9, Датчики окружающей среды:

```
#define DHT_PIN 3
#define DHT_TYPE DHT11
DHT dht(DHT_PIN, DHT_TYPE);
```

Теперь нам нужно создать экземпляр типа MOVI, как показано в следующей строке кода. Логическое значение false указывает, что мы не хотим включать последовательную отладку:

```
MOVI movi(false);
```

Наконец, нам понадобится массив символов, который будет использоваться для создания предложения, содержащего текущую температуру, чтобы шилд MOVI мог сообщить нам температуру, когда мы ее запросим.

```
char answer[21];
```

В функции setup () нам нужно будет инициализировать датчик температуры DHT и шилд MOVI. Следующий код показывает функцию setup ():

```
void setup() {
  dht.begin();
  movi.init();
  movi.callSign("buddy");
  movi.addSentence("temp");
  movi.train();
}
```

Эта функция начинается с инициализации датчика температуры DHT путем вызова функции begin () из типа dht. Затем мы иницилируем тип movi, вызывая функцию init (). Эта функция должна быть вызвана первой для инициализации типа movi.

Большинство устройств с голосовой активацией, таких как Amazon Echo, активируются позывным. Для устройств Amazon позывной является «Alexa - Алекса». В нашем примере шилд MOVI также будет использовать позывной, который его активирует. Позывной можно настроить с помощью метода callSign (), где используемый позывной передается в виде строки. В этом примере позывной будет «buddy - друг».

Затем мы захотим добавить предложение или слова, которым будет соответствовать MOVI. Мы делаем это с помощью функции `addSentence ()`. В этом примере мы попытаемся сопоставить слово «temp».

У Wedo есть возможность тренировать шилд MOVI с помощью полных предложений или слов. Если вы хотите, чтобы шилд MOVI распознал предложение, рекомендуется добавить полное предложение, даже если вам нужно добавить несколько версий одного и того же предложения. Добавляя полное предложение, алгоритм MOVI может быть использован для идентификации предложения. Это дает большую точность. Также рекомендуется, чтобы все обученные предложения были близки к одному и тому же размеру. Одно длинное предложение будет предпочтительнее гораздо меньшего, если произносится много слов.

Наконец, вызывается метод `train ()`, чтобы сообщить шилду MOVI, что мы добавили все эти предложения и позывной. При первом добавлении предложения или позывного шилда MOVI потребуется некоторое время для тренировки, но если позывной и предложение остаются неизменными между сборками вашего приложения, то шилд MOVI запускается очень быстро.

Теперь, когда функция `setup ()` завершена, давайте посмотрим на функцию `loop ()`. Следующий код показывает функцию `loop ()`:

```
void loop() {
  signed int res=movi.poll();
  if (res == 1) {
    float fahreheit = dht.readTemperature(true);
    int tmp = (int)fahreheit;
    sprintf(answer, "The temperature is %02d", tmp);
    movi.say(answer);
  }
}
```

В первой строке мы используем функцию `poll ()` из экземпляра `movi`. Эта функция будет опрашивать любые совпадения с обученными предложениями. Эта функция вернет ноль (0), если событие не произошло, или положительное число, если оно соответствует предложению. Возвращаемое число - это номер совпавшего предложения. В нашем примере у нас только одно предложение, поэтому единственное возможное совпадение - это предложение номер 1.

Если совпадение с предложением найдено, текущая температура считывается с датчика температуры DHT-11, а затем преобразуется из значения с плавающей запятой в целочисленное значение путем приведения типа.

Чтобы создать строку, которую мы хотим, чтобы шилд MOVI сказал, используется функция `sprintf ()`. Эта функция может использоваться для создания массива символов. В этом примере мы начинаем с предложения `temperature is`, а затем добавляем значение температуры в формате `%02d`. Это указывает функции `sprintf ()` добавить к строке двузначное целое число. Созданный массив символов сохраняется в массиве ответов, который был создан в начале этого скетча.

Мы используем функцию `say ()` из экземпляра `movi`, чтобы шилд `MOVI` сообщал нам текущую температуру через подключенные наушники или динамик. Теперь запустим проект.

При первом запуске проекта на обучение шилда `MOVI` потребуется немного времени. Подождите, пока шилд скажет, что он готов, а затем используйте позывной, чтобы активировать его. Как только вы произнесете позывной, вы услышите звуковой сигнал, если шилд `MOVI` распознает его. Если `MOVI` распознает позывной, произнесите слово «темп.» Шилд `MOVI` должен ответить, сообщив вам текущую температуру, которая была считана датчиком температуры `DHT-11`.

Этот пример затрагивает только самые основы того, что вы можете делать с щитом `MOVI`, и дает вам достаточно, чтобы начать с ним работать.

Есть и другие очень полезные функции, которые мы можем использовать с экземпляром `MOVI`. Вот несколько дополнительных функций, которые вы можете попробовать добавить в проект:

- **`isReady ()`**: вернет логическое значение `true`, если `MOVI` готов, или `false`, если он не готов .
- **`setVolume (int volume)`**: устанавливает громкость вывода `MOVI` от 0 (без звука) до 100 (полная громкость).
- **`setVoiceGender (bool female)`**: установит пол для голоса `MOVI`. Истинное значение установит его на женский голос, а ложное значение - на мужской.
- **`setThreshold (int threshold)`**: устанавливает порог шума для распознавателя речи. Значения могут находиться в диапазоне от 2 до 95. Значение 15 подходит для шумной среды, а значение 30 подходит для очень шумной среды.
- **`beeps (bool on)`**: включает или выключает звуковой сигнал распознавания.
- **`welcomeMessage (bool on)`**: включает или выключает приветственное сообщение `MOVI`.
- **`ask()` and `ask(string question)`**: слушает напрямую, не дожидаясь позывного. Если передана строка, `MOVI` задаст вопрос перед прослушиванием.

Задача состоит в том, чтобы попытаться добавить некоторые из этих функций в пример проекта и посмотреть, что можно делать с помощью шилда `MOVI`.

Также попробуйте добавить дополнительные предложения для `MOVI` `tolisten` .

В этой главе мы увидели, как можно использовать экран MOVI для распознавания речи, а также для синтеза голоса. Мы использовали распознавание речи для прослушивания определенной команды и синтезирование голоса для ответа на команду.

В следующей главе мы рассмотрим, как можно использовать двигатели постоянного тока и контроллеры двигателей.

13

Двигатели постоянного тока (DC) и контроллеры двигателей

До сих пор в этой книге все проекты были стационарными. Под стационарными проектами я имею в виду, что проекты не могли двигаться самостоятельно. В этой главе мы рассмотрим, как добавить двигатели постоянного тока (DC) в любой проект, дав ему возможность двигаться самостоятельно. При использовании двигателей DC я бы рекомендовал использовать контроллер двигателя для управления ими. Контроллеры двигателей позволяют нам очень легко подключать к двигателю внешний источник питания и управлять направлением и скоростью двигателя.

В этой главе вы узнаете:

- Как работает щеточный двигатель DC
- Как работает H-мост
- Как использовать контроллеры мотора L298 и L293D

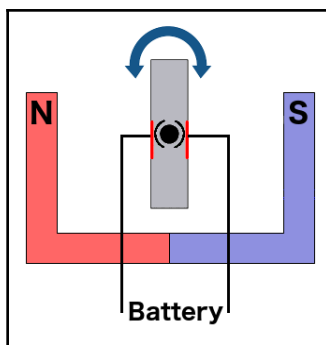
Двигатель DC - это класс вращающихся электрических устройств, которые преобразуют электрическую энергию в физическое движение. Существует множество типов двигателей DC; однако в этой главе мы рассмотрим один конкретный тип, которым является щеточный двигатель DC.

Щеточные двигатели DC используются в самых разных областях, от игрушек и робототехники до окон с электроприводом и электроинструментов. Некоторыми преимуществами щеточных двигателей DC являются их первоначальная низкая стоимость, простота управление и крутящий момент на низкой скорости. Недостатками этих двигателей являются их высокая стоимость обслуживания и небольшой срок службы в высокоинтенсивных средах. Для прототипов и робототехнических проектов, которые мы обычно делаем с Arduino, недостатки щеточных двигателей DC обычно не вызывают беспокойства.

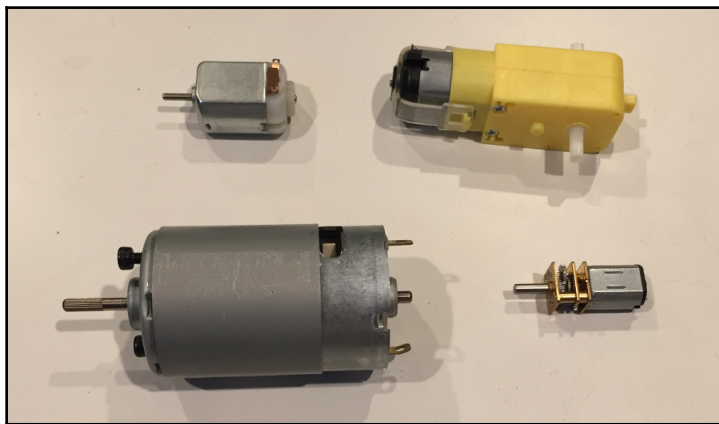
В центре щеточного двигателя находится вращающийся якорь, который содержит электромагнит. С внешней стороны вращающегося якоря находится постоянный неподвижный магнит. Когда на электромагнит в ядре подается питание, создается магнитное поле, которое притягивает и отталкивает постоянные неподвижные магниты. Это приводит к тому, что якорь начинает вращаться.

Чтобы якорь продолжал вращаться, необходимо поменять полярность электромагнита. Для этого используется сегментированная медная втулка, называемая коммутатором, которая находится на оси двигателя. Когда двигатель вращается, щетки скользят по коммутатору, вступая в контакт с различными частями коммутатора, вызывая переключение полярности магнита.

На следующей схеме показаны части щеточного двигателя постоянного тока:

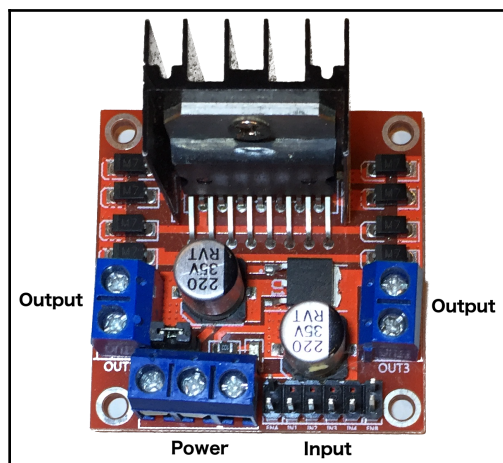


Щеточные двигатели постоянного тока бывают разных форм и размеров. Некоторые из этих двигателей имеют встроенные коробки передач, которые могут изменять крутящий момент и скорость вращения двигателя. На следующей фотографии показаны некоторые примеры щеточных двигателей постоянного тока:



Включение двигателя постоянного тока непосредственно от Arduino для чего-либо, кроме демонстрационных целей, обычно не является хорошей идеей, потому что напряжение и ток, подаваемые с контактных разъемов, довольно ограничены.

Мы можем использовать контроллер двигателя для управления направлением и скоростью двигателя постоянного тока от Arduino, при этом обеспечивая внешний источник питания для его питания. В этой главе мы рассмотрим, как мы можем использовать драйвер двигателя L298 с двойным H-мостом, показанный на рисунке на следующей фотографии, а также как использовать микросхему L293D:



Драйвер двигателя L298 позволяет нам управлять направлением и скоростью двух двигателей. Этот драйвер позволяет нам управлять двигателями от 5 В до 35 В с максимальным током 2 А. Если напряжение питания 12 В или меньше, мы также можем использовать выход 5 В для питания Arduino. Драйвер двигателя L298 имеет несколько помеченных входов, выходов и разъемов питания. Эти входы слева направо:

- **ENA**: включает двигатель A и регулирует скорость двигателя.
- **IN1 IN2**: управляет направлением двигателя A
- **IN3 IN4**: управляет направлением двигателя B.
- **ENB**: включает двигатель B и управляет скоростью двигателя.

ENA и ENB обычно имеют переключки на контактах. Чтобы управлять щеточными двигателями DC, нам нужно будет удалить эти переключки и подключить контакт к порту ШИМ. Выходы:

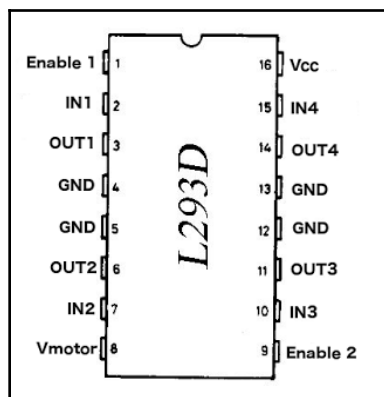
- **OUT1 OUT2:** выходная мощность на двигатель A
- **OUT3 OUT4:** выходная мощность на двигатель B

Входы питания слева направо:

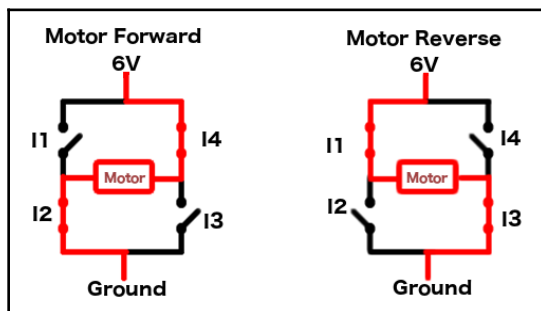
- **Vmotor:** питание от внешнего источника, который будет испол. для питания двигателей.
- **GND:** земля
- **Vout:** выход 5 В, который можно использовать для питания Arduino

Если мы создаем проект с ограниченным пространством, вместо того, чтобы использовать контроллер двигателя, такой как драйвер двигателя с двойным H-мостом L298, мы можем использовать интегрированный чип, такой как ИС L293D H-bridge.

Микросхема L293D может приводить в действие два двигателя, аналогично мотору L298, и может приводить в действие двигатели до 35 В при постоянном токе 600 мА с максимальным током 1,2 А. На следующей схеме показана распиновка микросхемы L293D:



И контроллер мотора L298, и микросхема L293D являются H-мостами. Давайте посмотрим, как работает H-мост. H-мост - это электрическая схема, которая позволяет нам подавать напряжение на наши двигатели в любом направлении, позволяющее двигателю вращаться вперед или назад. Термин H-мост происходит от типичного графического представления схемы, которое выглядит как заглавная буква H. На следующей диаграмме показано, как работает H-мост:



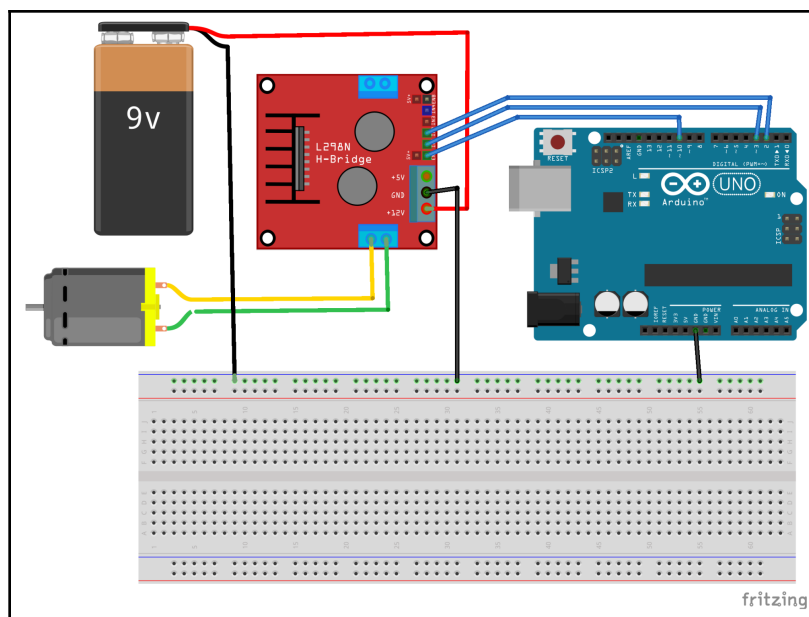
H-мост обычно состоит из четырех твердотельных переключателей. Как мы видим на предыдущем изображении, когда переключатели 1 и 3 (I1 и I3) разомкнуты, а переключатели 2 и 4 (I2 и I4) замкнуты, правая сторона двигателя подключена к источнику питания, а левая сторона отключена. подключен к земле, вращая мотор в одном направлении. Если переключатели 1 и 3 (I1 и I3) замкнуты, а переключатели 2 и 4 (I2 и I4) разомкнуты, то левая сторона двигателя подключена к источнику питания, а правая сторона подключена к земле, вращая двигатель. в другом направлении.

Давайте посмотрим какие компоненты нам понадобятся для наших проектов в этой главе.

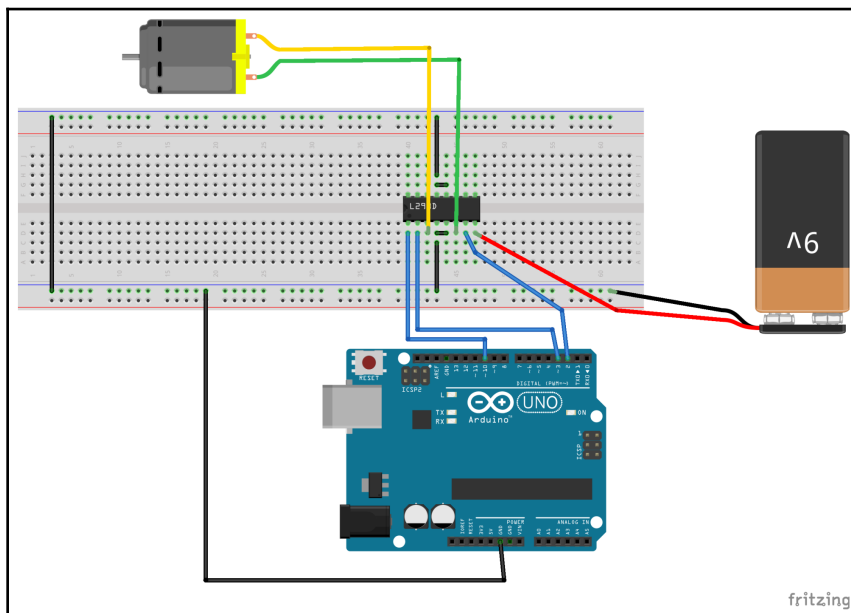
В этой главе вам потребуются следующие компоненты.

- Arduino Uno или совместимая плата
- Драйвер двигателя L298
- L293D H-мостовой чип
- Два щеточных двигателя постоянного тока
- внешняя батарея 9-12 В
- Перемычки
- Макетная плата

В этой главе мы создадим два проекта. В первом проекте будет использоваться драйвер двигателя L298 для управления одним двигателем, а во втором проекте будет использоваться микросхема L293D для управления одним двигателем. Вот принципиальная схема проекта драйвера двигателя L298:



Прежде чем объяснять эту схему, давайте также рассмотрим принципиальную схему микросхемы L293D, потому что между этими двумя схемами есть много общего:



Первое, на что следует обратить внимание на этих двух схемах, - это то, что цепи содержат общую землю, а это означает, что разъемы заземления на Arduino, батарее и контроллерах мотора (как L298, так и L293D) соединены вместе. В подобных проектах, которые включают в себя несколько источников питания, мы должны иметь общую землю между всеми устройствами и источниками питания.

В обеих схемах ШИМ из 10 контактов на Arduino подключен к разрешающему контакту на контроллере мотора. Кроме того, в обеих схемах цифровые контакты 2 и 3 подключены к контактам IN1 и IN2 на контроллерах двигателей. Это позволяет нам использовать один и тот же код для обоих проектов.

Теперь посмотрим на код этих проектов.

Код для управления двигателями должен использовать только стандартные функции `digitalWrite ()` и `analogWrite ()` из стандартной библиотеки Arduino, поэтому для этого кода не требуются никакие внешние библиотеки. Поэтому наш код начнется с определения выводов на Arduino, которые подключены к контроллерам двигателей. Следующий код делает это:

```
#define MC_IN_1 3
#define MC_IN_2 2
#define MC_ENABLE 10
```

Теперь нам нужно настроить выводы для вывода в функции `setup ()`, как показано в следующем коде:

```
void setup() {
  pinMode(MC_ENABLE, OUTPUT);
  pinMode(MC_IN_1, OUTPUT);
  pinMode(MC_IN_2, OUTPUT);
}
```

Теперь мы готовы включить моторы. Поместим следующий код в функцию `loop ()`:

```
void loop() {
  digitalWrite(MC_IN_1, HIGH);
  digitalWrite(MC_IN_2, LOW);
  analogWrite(MC_ENABLE, 250);
  delay(2000);
  analogWrite(MC_ENABLE, 0);
  delay(1000);
  digitalWrite(MC_IN_1, LOW);
  digitalWrite(MC_IN_2, HIGH);
  analogWrite(MC_ENABLE, 125);
  delay(2000);
  analogWrite(MC_ENABLE, 0);
  delay(1000);
}
```

Функция `loop ()` начинается с использования функции `digitalWrite ()` для установки входа 1 на контроллере двигателя на ВЫСОКИЙ уровень, а для входа 2 на НИЗКИЙ. Затем функция `analogWrite ()` используется для создания рабочего цикла 250 для разрешающего вывода на контроллере мотора. Помните, что максимальный рабочий цикл PWM-контактов равен 255; поэтому после выполнения функции `analogWrite ()` двигатель должен начать вращаться почти с полной скоростью.

Для вращения щеточного двигателя DC один вход должен иметь ВЫСОКИЙ уровень, а другой - НИЗКИЙ. Если на обоих входах ВЫСОКИЙ, НИЗКИЙ или рабочий цикл на разрешающем контакте равен 0, двигатель не будет вращаться. Следующая таблица показывает это:

IN1	IN2	Включить рабочий цикл	Результат
HIGH	LOW	>0	Мотор вращается по направлению
LOW	HIGH	>0	Мотор вращается в другом направлении
HIGH	HIGH		Мотор остановился
LOW	LOW		Мотор остановился
		0	Мотор остановился

После вызова функции `analogWrite ()` функция задержки используется для приостановки выполнения приложения на две секунды, чтобы дать двигателю поработать. Затем снова вызывается функция `analogWrite ()`, чтобы установить рабочий цикл на 0, что остановит вращение двигателя и задержит на одну секунду, чтобы дать двигателю возможность остановиться.

Затем функции `digitalWrite ()` используются для установки на 1 выводе LOW уровня и на 2 выводе HIGH уровня, что противоположно тому, как они были изначально установлены - двигатель вращается в противоположном направлении.

Затем вызывается функция `analogWrite ()`, чтобы установить рабочий цикл на 125, что запустит двигатель на половинной скорости. Затем функция задержки используется для приостановки выполнения приложения на две секунды, а затем снова останавливаем мотор.

При запуске этого кода двигатель должен вращаться в одном направлении в течение двух секунд, останавливаться на одну секунду, вращаться в другом направлении в течение двух секунд, останавливаться на одну секунду и затем запускаться заново.

Для задачи этой главы попробуйте добавить вторые двигатели в оба проекта, а затем измените код, чтобы оба двигателя вращались одновременно. Вы также можете попробовать подключить двигатели, чтобы они вращались в одном направлении, если вы примените ВЫСОКОЕ значение к контактам IN1 и IN3, а НИЗКОЕ - к контактам IN2 и IN4.

В этой главе мы узнали основы того, как работает щеточный двигатель постоянного тока и как мы можем использовать драйвер двигателя L298 и микросхему L293D для управления щеточным двигателем. Мы также узнали, как работает H-мост.

В следующей главе мы рассмотрим другой тип двигателя постоянного тока. Этот двигатель называется серводвигателем и используется в проектах, где требуется точное позиционирование, например, с роботизированными манипуляторами.

Сервомоторы

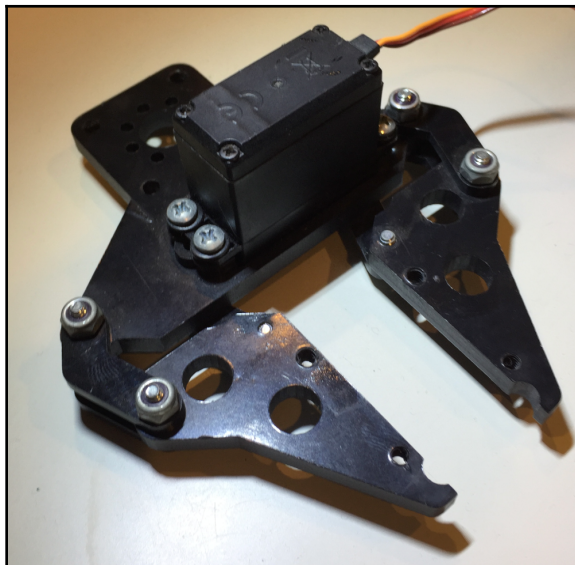
Когда питание подается на щеточный двигатель DC, он начинает непрерывно вращаться, пока не будет отключено питание. Это делает щеточные электродвигатели очень хорошими для таких вещей, как вращение колес робота или лопастей вентилятора. Бывают случаи, когда нам нужно более точное управление скоростью вращения двигателя. Например, для управления роботизированной рукой нам нужно, чтобы двигатели вращались на определенную величину, чтобы поместить руку туда, где она должна быть. Для таких приложений мы можем использовать серводвигатель.

В этой главе вы узнаете:

- Как управлять серводвигателем
- Как использовать сервобиблиотеку Arduino
- Как запитать серводвигатель

Типы серводвигателей, которые мы будем использовать с Arduino, довольно малы, но большинство из них имеют довольно высокий крутящий момент и очень энергоэффективны. Это позволяет нам использовать эти двигатели для промышленных применений, таких как роботизированные манипуляторы, конвейерные ленты, автофокусные линзы в камерах и даже системы слежения за солнечным светом для солнечных панелей.

Серводвигатель состоит из двигателя постоянного тока, который выполняет фактическую работу; потенциометра, который управляет количеством мощности, поступающей на двигатель; схемы управления, которая управляет движением двигателя и шестерен. На следующей фотографии показан серводвигатель, подключенный к роботизированной лапе:



Серводвигатель содержит три провода для управляющего сигнала, питания и заземления. Сигнальный провод обычно оранжевый или желтый. Питание обычное красное, а провод заземления обычно коричневый или черный.

Некоторые серводвигатели меньшего размера могут использовать выход 5 В на Arduino; однако в этой главе я буду использовать высокомоментный двигатель MG996R, который может выдерживать напряжение до 7,2 В и имеет рабочий ток от 500 мА до 900 мА. Поэтому мы будем подключать его к внешнему аккумулятору на 6 В, который содержит 4 батарейки АА. Я бы рекомендовал обратиться к техническому описанию вашего серводвигателя, чтобы определить правильную потребляемую мощность для вашего серводвигателя.



ПРИМЕЧАНИЕ. Хотя некоторые из двигателей меньшего размера могут питаться от выхода 5 В, я бы рекомендовал использовать внешний источник питания .

Заземляющий провод должен быть подключен к общему заземлению, которое используется совместно с аккумуляторным блоком, а также с землей Arduino. Сигнальный провод должен быть подключен к выходу PWM на Arduino.

Рабочий цикл от вывода ШИМ определяет положение вала сервопривода. Когда вал серводвигателя находится в желаемом положении, мощность, подаваемая на двигатель, отключается. Скорость вращения двигателя пропорциональна разнице между фактическим положением и желаемым положением, что означает, что чем дальше от фактического положения находится желаемое положение, тем быстрее будет вращаться двигатель. Это делает серводвигатель очень эффективным, поскольку он работает ровно настолько, насколько это необходимо.

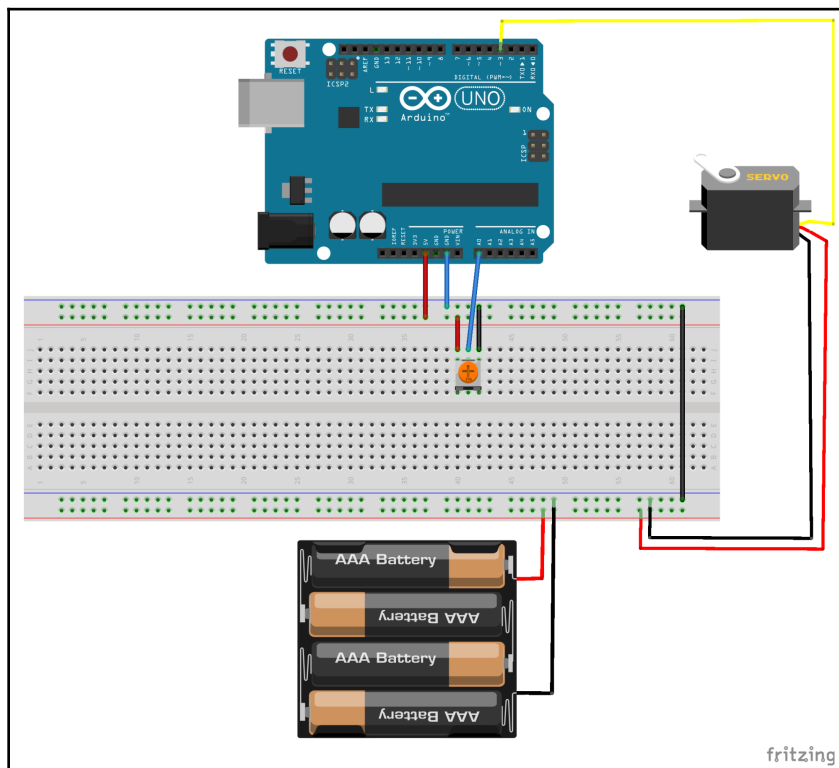
Разные серводвигатели имеют разные максимальные радиусы поворота. Радиус поворота большинства серводвигателей составляет 120 градусов (60 градусов в каждом направлении) или 180 градусов (90 градусов в каждом направлении). Серводвигатель MG996R, который я буду использовать в этой главе, имеет максимальный радиус поворота 120 градусов. Как только серводвигатель вращается в желаемое положение, он будет пытаться удерживать это положение и будет сопротивляться любой попытке вытолкнуть его из этого положения.

Давайте посмотрим на компоненты, которые нам понадобятся для проекта этой главы.

В этой главе вам понадобятся следующие компоненты:

- Arduino Uno или совместимая плата
- серводвигатель (код был протестирован с сервоприводом MG996R. Однако любой стандартный сервопривод должен работать)
- Потенциометр
- Один батарейный отсек 4 AA с батареями для питания серводвигателя
- Перемычки
- Макетная плата

На следующей схеме показано, как подключить серводвигатель к Arduino:



В этом проекте мы будем использовать потенциометр для управления положением серводвигателя. Обратите внимание, что потенциометр использует источник питания 5 В от Arduino, в то время как серводвигатель использует 6 В ($4 \times 1,5$ В) от батарей; однако два источника питания имеют общую землю.

Теперь посмотрим на код для управления серводвигателем.

И IDE Arduino, и веб-редактор поставляются с серво-библиотекой, которую мы можем использовать, просто включив файл заголовка. Следующий код сделает это:

```
#include <Servo.h>
```

Затем нам нужно определить вывод, к которому подключены серводвигатель и потенциометр. Следующий код подключит сигнальный провод к цифровому контакту 3, а потенциометр - к аналоговому контакту 0 на Arduino:

```
#define SERVO0_POT 0  
#define SERVO0_OUT 3
```

Теперь нам нужно определить экземпляр сервотипа, как показано в следующей строке:

```
Servo servo0;
```

В функции настройки нам нужно вызвать метод `attach()` из экземпляра сервопривода, чтобы инициализировать экземпляр и сообщить ему, к какому выводу прикреплен сервопривод. Следующие коды показывают это:

```
void setup() {  
    servo0.attach(SERVO0_OUT);  
}
```

Мы захотим определить функцию, которая будет считывать показания потенциометра и устанавливать положение сервопривода в зависимости от того, насколько повернут потенциометр.

```
void setServo(int pot, Servo out) {  
    int servo = analogRead(pot);  
    long int servo_val = map(servo, 0, 1023, 0, 120);  
    out.write(servo_val);  
}
```

Эта функция принимает целое число, то есть вывод, к которому подключен потенциометр. Функция `analogRead ()` вызывается для чтения вывода, к которому подключен потенциометр. Мы используем функцию `map ()` для сопоставления считанного значения от аналогового вывода (значения от 0 до 1023) на 120 градусов, на которые может перемещаться серводвигатель. Затем функция `write ()` из типа сервопривода используется для записи этого значения в сервопривод, в результате чего сервопривод корректирует свое положение.

Затем из функции `loop ()` вызывается функция `setServo ()` для считывания показаний потенциометра и установки сервопривода, как показано в следующем коде:

```
setServo(SERVO0_POT, servo0);  
delay(15);
```

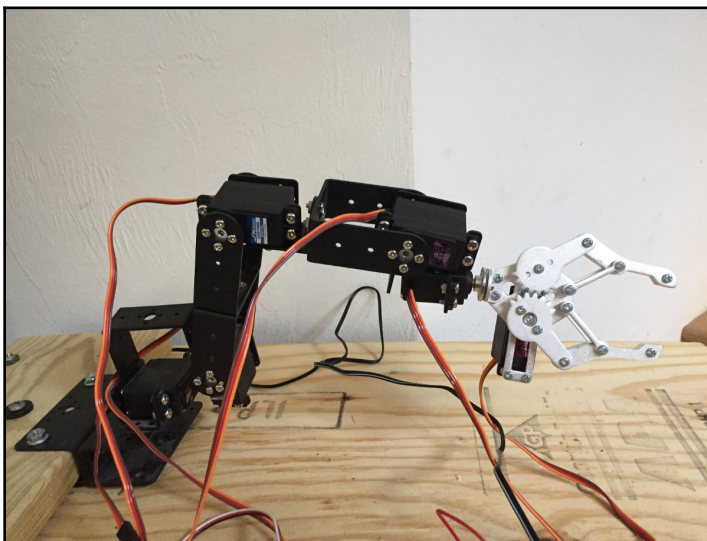
Причина, по которой мы создали функцию `setServo ()` вместо того, чтобы помещать этот код непосредственно в функцию `loop ()`, заключается в том, что она значительно упрощает добавление нескольких серводвигателей. Например, если мы хотим создать роботизированную руку с пятью сервоприводами, мы могли бы очень легко сделать это, настроив сервоприводы, как мы сделали с первым, а затем используя следующий код в функции `setup ()`:

```
setServo(SERVO0_POT, servo0);  
setServo(SERVO1_POT, servo1);  
setServo(SERVO2_POT, servo2);  
setServo(SERVO3_POT, servo3);  
setServo(SERVO4_POT, servo4);  
delay(15);
```

Если у нас есть код, например код в функции `setServo ()`, который можно использовать несколько раз, всегда полезно поместить этот код в отдельную функцию, подобную этой.

Если этот проект запущен, положение сервопривода изменится при повороте потенциометра.

Для выполнения задания в этой главе вам понадобится роботизированная рука с 6 степенями свободы, которая показана на следующем рисунке :



Для этой задачи вам нужно будет выяснить, как подключить оставшиеся серводвигатели к Arduino и выбрать правильную конфигурацию питания. Вы можете заказать 6 наборов роботов-манипуляторов DOF на Amazon или eBay. Зайдите на их сайт и выполните поиск робота-манипулятора с 6 степенями свободы. Цены на эти комплекты сильно различаются в зависимости от размера и мощности руки / когтя. Вы можете получить роботизированные манипуляторы в готовом виде или в виде набора, который вам нужно собрать самостоятельно.

В этой главе мы узнали, как работает серводвигатель и как мы можем управлять им с помощью Arduino. Мы также увидели, из каких компонентов состоит серводвигатель.

В следующей главе мы увидим, как использовать релейную плату.

Использование реле

Бывают случаи, когда мы хотим управлять предметами, питающиеся от сети 220В, такими как свет, вентилятор или любой другой бытовой прибор. Однако Arduino и все проекты в этой книге используют постоянный ток (DC), в то время как ваши бытовые приборы используют переменный ток (AC). Между переменным и постоянным током есть существенные различия. В этой главе мы рассмотрим, как мы можем использовать реле с Arduino для управления лампой, работающей от переменного тока.

В этой главе вы узнаете:

- Что такое реле
- Как использовать реле для управления устройством с питанием 220В переменного тока
- Как использовать реле для управления устройством с питанием от постоянного тока
- Как изолировать схемы с помощью реле

: В этой главе мы будем использовать 120 или 240 В переменного тока в зависимости от страны, в которой вы живете, что значительно опасней, чем все остальное, что мы использовали в этой книге.



Неправильное обращение, или неправильное использование реле или силового кабеля может привести к серьезным травмам и даже смерти. Прежде чем приступить к реализации проекта, описанного в этой главе, убедитесь, что вы прочитали и поняли, как работает ваша релейная плата, напряжение и ток, на которые она рассчитана, а также риски, связанные с использованием сети переменного тока.

Не бойтесь обращаться за профессиональной помощью, если вы в чем-то не уверены. Питание переменного тока значительно опаснее, чем питание постоянного тока, которое мы использовали ранее в этой книге.

Если вам неудобно использовать питание переменного тока в проекте или вы недостаточно знакомы с ним, для этой главы будут представлены две принципиальные схемы. На одной схеме показано, как подключить лампу с питанием от переменного тока к реле, а на другой показано, как подключить двигатель и источник питания 9 В к реле. Код для этой главы будет работать с любым проектом.



При работе с устройством, работающим от сети переменного тока, всегда убедитесь, что устройство отключено от сети, прежде чем выполнять какие-либо работы с ним. Поражение электрическим током от розетки может привести к серьезным травмам и даже смерти.

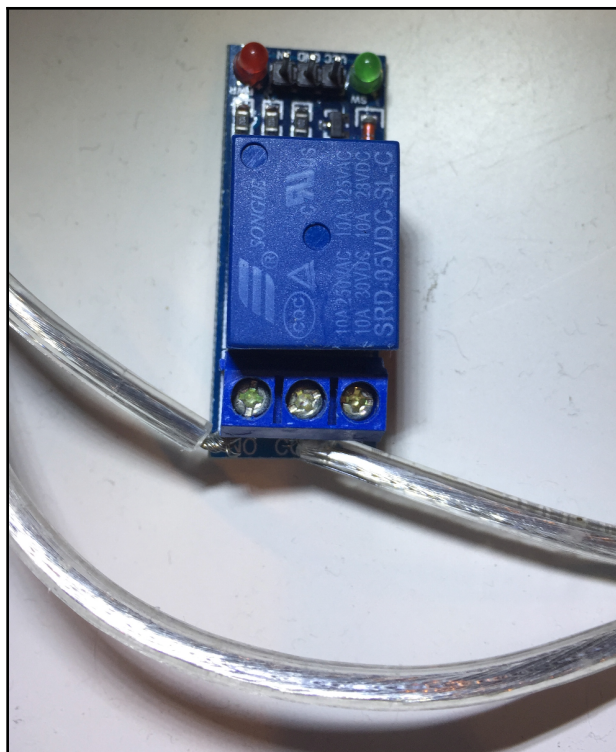
Реле можно рассматривать как электрический выключатель. Многие реле используют электромагниты для механического управления переключателем. Однако есть и другие способы управления реле. Примером этого является твердотельное реле, в котором не используются механические детали. Большинство реле, которые мы будем использовать с Arduino, используют электромагниты для управления переключателем.

Реле используются, когда необходимо изолировать две или более цепей друг от друга, в то же время имея возможность для компонента одной из цепей (Arduino) управлять компонентом в других цепях.

Когда вы используете питание постоянного тока для всех компонентов в проекте, обычно нет необходимости изолировать цепи; однако, если вы хотите управлять устройством с питанием от переменного тока, например настольной лампой, вам потребуется устройство, подобное реле.

Как мы упоминали ранее, если вы не знакомы с работой с устройствами с питанием от переменного тока, используйте проект двигателя постоянного тока, а не проект лампы. В обоих проектах используются одни и те же концепции; однако один питается от переменного тока, а другой - от постоянного тока.

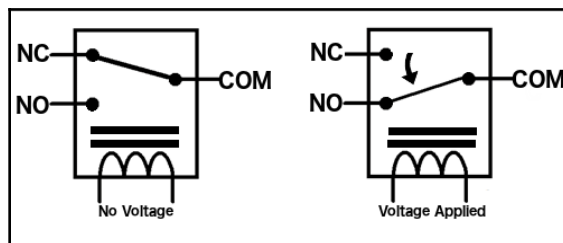
Вилка для устройств с питанием от переменного тока содержит два провода. Чтобы подключить это устройство к реле, нам нужно перерезать один из проводов и подключить один конец этого провода к СОМ-соединению на реле, а другой - к NO-соединению на реле. Следующая фотография иллюстрирует это:



Метка NO на реле обозначает нормально разомкнутый, а метка NC - нормально замкнутый. COM-соединение, которое обычно находится между NO и NC соединениями, является общим соединением. Когда реле выключено, питание проходит через соединение NC, что означает, что если устройство с питанием от переменного тока должно быть включено, когда реле выключено, то оно должно быть подключено к соединениям COM и NC. Если устройства с питанием от переменного тока должны быть включены только тогда, когда реле включено, то они должны быть подключены к соединениям COM и NO, как мы делаем здесь.

Реле обычно рассчитаны на максимальное напряжение от 230 до 250 В переменного тока или 30 В постоянного тока при 10 А. Вам нужно будет убедиться, что реле в вашем проекте может выдерживать напряжение и ток, которые вы используете.

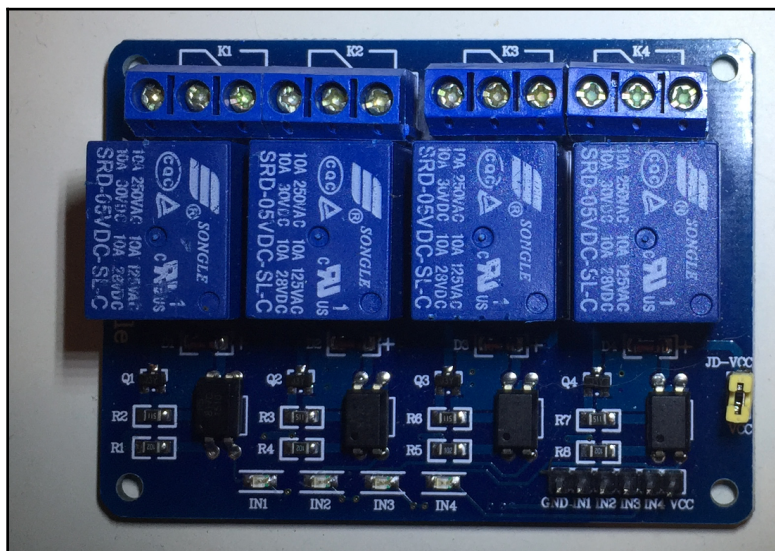
На следующей схеме показано, как работает реле:



На предыдущих изображениях мы видим, что, когда на реле не подается напряжение (изображение слева), контакт NC подключается к клемме COM, замыкая цепь. При подаче напряжения якорь подтягивается к контакту NO, соединяющему его с контактом COM, замыкая эту цепь.

На предыдущей фотографии использовалась плата с одним реле. Однако есть платы, содержащие несколько реле, которые дают нам возможность управлять компонентами в нескольких схемах.

На следующей фотографии показана плата с четырьмя реле:

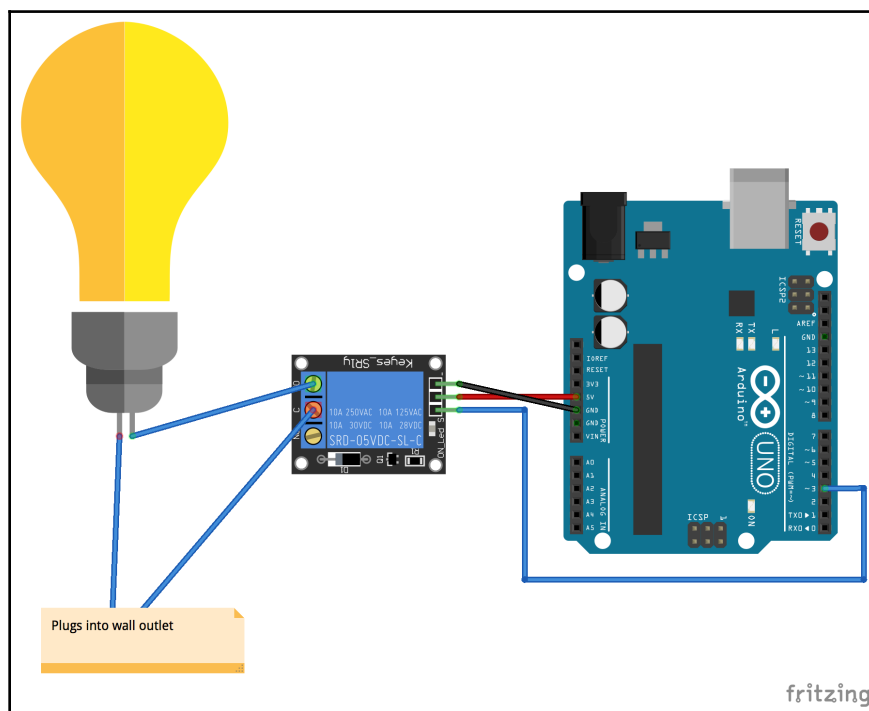


Реле, которые были показаны в этой главе, рассчитаны как на цепи переменного, так и на постоянный ток, поэтому их можно использовать для любого проекта в этой главе. Вы даже можете попытаться выполнить оба проекта, если хотите.

Давайте рассмотрим компоненты, которые нам понадобятся для проектов этой главы.

- Arduino Uno или совместимая плата
- плата реле
- Перемычки
- Устройство с питанием от переменного тока, которым вы хотели бы управлять с помощью Arduino, например, настольная лампа, или если вы хотите использовать устройство с питанием от постоянного тока, а не с питанием от переменного тока.
- Адаптер аккумулятора 9 В с аккумулятором с двигателем постоянного тока

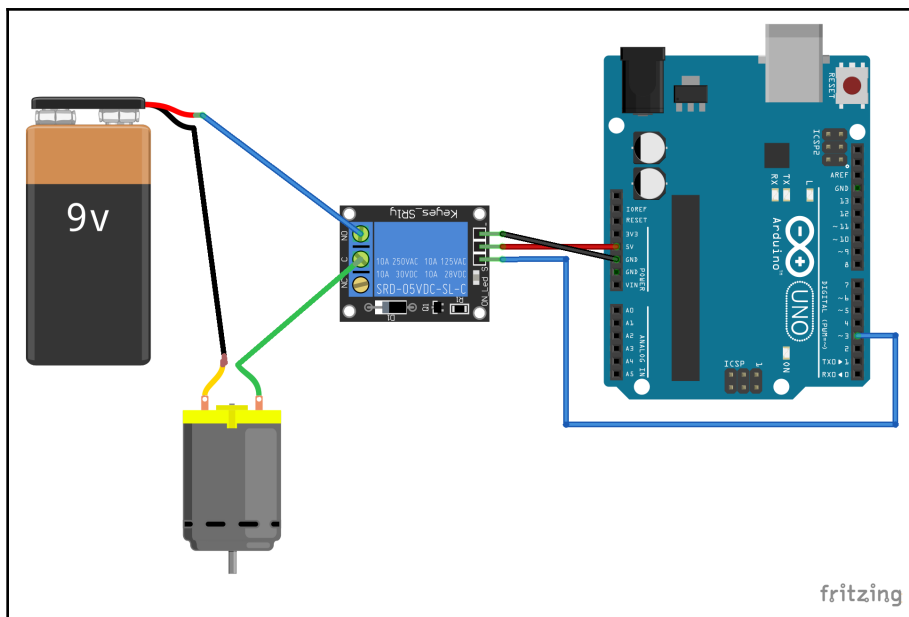
На следующих схемах показано, как подключить устройство с питанием от переменного тока и Arduino к реле:



Устройство с питанием от переменного тока подключается к реле, как описано в разделе «Введение». Контакт VCC на реле подключен к выходу 5 В на Arduino, а контакт GND на реле подключен к выводу GND на Arduino. Мы подключаем цифровой 3-контактный разъем Arduino к контакту с маркировкой IN на реле.

Цифровой 3-й вывод будет использоваться для управления реле.

На следующей диаграмме показано, как мы будем использовать реле для управления двигателем постоянного тока и источником питания 9 В с Arduino:



В предыдущих схемах мы подчеркивали необходимость иметь общую землю между различными компонентами:

однако в этой схеме вы заметите, что между Arduino и цепью двигателя / 9-вольтовой батареи нет общего заземления. При использовании реле цепи на противоположных сторонах реле изолированы друг от друга; Если вы хотите иметь общее заземление между двумя схемами, в реле нет необходимости, потому что реле используется для изоляции двух схем.

Теперь посмотрим на код этих схем.

Ниже приводится код для проектов в этой главе:

```
#define RELAY 3

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

void loop() {
  digitalWrite(RELAY, HIGH);
  delay(3000);
  digitalWrite(RELAY, LOW);
  delay(3000);
}
```

Этот код начинается с подключения реле к цифровому 3-му выводу на Arduino. В функции `setup()` мы активируем вывод реле для использования функции `digitalWrite()` для включения и выключения реле.

В функции `loop()` мы используем функцию `digitalWrite()`, чтобы установить вывод реле на высокий уровень, сделать паузу в три секунды, снова использовать функцию `digitalWrite()`, чтобы установить вывод реле на низкий уровень и, наконец, снова сделать паузу на три секунды. Это будет включать и выключать компоненты, подключенные к реле, каждые три секунды. Этот код будет работать либо со схемой переменного тока, либо со схемой постоянного тока, показанной ранее в этой главе.

Для решения задачи этой главы используйте плату с четырьмя реле и попробуйте подключить компонент к каждому реле, которым может управлять Arduino. Имейте в виду, что каждой схеме нужен собственный изолированный источник питания.

В этой главе мы увидели, как можно использовать реле для управления компонентами переменного и постоянного тока. Мы также увидели, что цепи по обе стороны от реле нуждаются в собственном изолированном источнике питания.

В следующей главе мы увидим, как мы можем использовать радиочастоты для удаленного управления Arduino.

16

Удаленное управление Arduino

Когда я был ребенком, мои родители использовали меня и мою сестру в качестве пульта дистанционного управления для телевизора, потому что тогда телевизоры не поставлялись с пультами дистанционного управления (ПДУ). К счастью, Юджин Полли, инженер из Zenith, придумал управлять телевизором с помощью ПДУ, избавляя миллионы детей от необходимости переключать каналы для своих родителей. ПДУ значительно улучшил наше взаимодействие с телевизором и может сделать то же самое для вашего проекта Arduino.

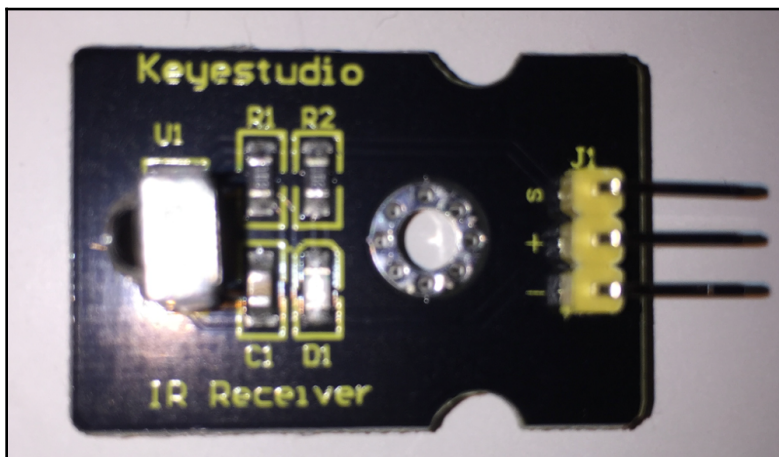
В этой главе вы узнаете:

- Как подключить радиочастотный пульт дистанционного управления к Arduino
- Как определить, какая кнопка нажата на радиочастотном пульте
- Как подключить инфракрасный пульт дистанционного управления к Arduino
- Как определить, какая кнопка нажата на инфракрасном пульте дистанционного управления

В этой главе мы рассмотрим несколько способов дистанционного управления нашим проектом Arduino. Для первого проекта мы будем использовать ИК (инфракрасный) приемник фирмы Keystudio, в котором используется инфракрасный модуль управления HX1838. Модуль инфракрасного управления HX1838 используется во множестве ИК-приемников, которые могут использоваться Arduino. Поэтому можно использовать инфракрасные приемники других производителей.

Инфракрасный передатчик имеет светодиод, излучающий инфракрасное излучение, которое улавливается инфракрасным приемником. При нажатии кнопки на пульте дистанционного управления светодиод на передатчике будет очень быстро мигать в течение доли секунды, а приемник будет считывать характер мигания и интерпретировать его.

ИК-приемник Keyestudio, который мы будем использовать в этой главе:



Вывод, помеченный буквой **S**, является сигнальным и должен быть подключен к одному из цифровых выводов на Arduino. Вывод, отмеченный знаком +, должен быть подключен к 5 В, а вывод со знаком - должен быть подключен к земле.

Одна из действительно хороших вещей в использовании ИК-приемника в качестве ПДУ для вашего проекта - это то, что вы можете использовать практически любой ИК-пульт дистанционного управления в качестве передатчика. Например, я могу использовать этот пульт, который идет с одним из моих ИК-приемников.



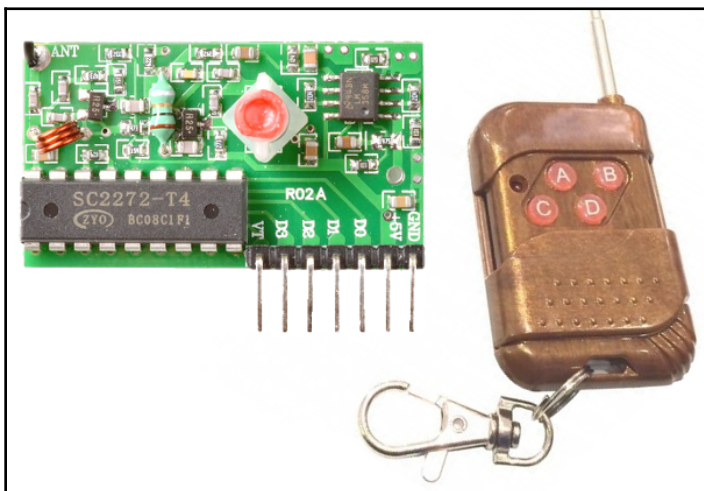
Или можно использовать любой из пультов дистанционного управления от телевизора, которые используют инфракрасный порт, например, тот, который показан на следующей фотографии:



Однако некоторые пульты дистанционного управления не используют инфракрасные технологии, например пульт Apple TV, который использует Bluetooth 4.0. Поэтому их нельзя использовать с инфракрасным приемником.

У инфракрасных пультов есть пара недостатков, самый большой из которых заключается в том, что они должны иметь прямую видимость для связи с приемником. Это означает, что передатчик должен быть направлен прямо на приемник. Еще одним недостатком инфракрасных пультов ДУ является то, что они работают только на расстоянии до 10 метров.

Вместо использования инфракрасного передатчика / приемника мы могли бы использовать радиочастотный (RF) передатчик и приемник. В этой главе мы рассмотрим, как использовать базовый радиочастотный передатчик с четырьмя кнопками и приемник, подобный показанному на следующей фотографии:



Два самых больших преимущества использования РЧ-передатчика и приемника заключаются в том, что РЧ-сигнал может распространяться дальше, и они могут проходить через обычные стены, что означает отсутствие необходимости в прямой видимости. В отличие от инфракрасного приемник, который может работать практически с любым инфракрасным передатчиком, если РЧ-передатчик и приемник не предназначены для совместной работы и настроены на одинаковые частоты, они не смогут обмениваться данными. РЧ передатчики, подобные показанному на предыдущей фотографии, также имеют гораздо меньше кнопок, чем инфракрасные пульты дистанционного управления.

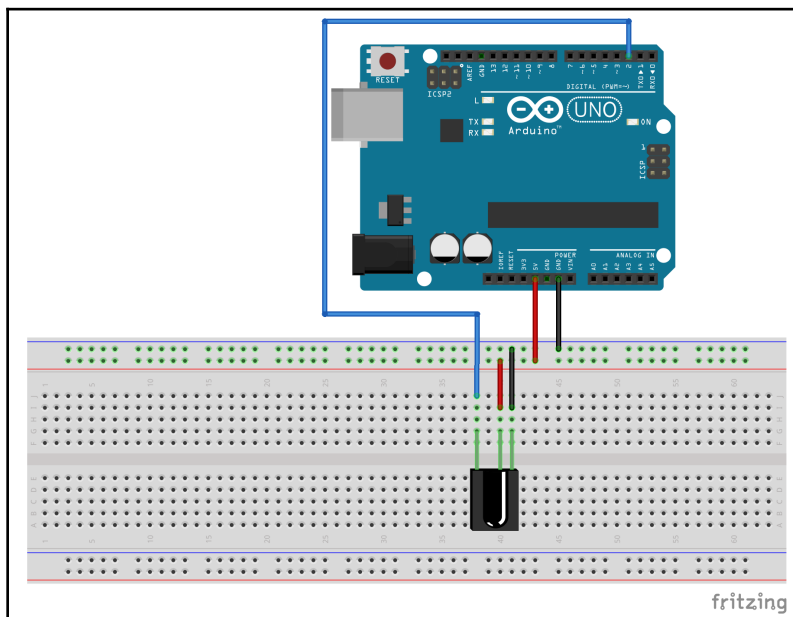
Давайте посмотрим, какие детали нам понадобятся для этих проектов.

Необходимые компоненты

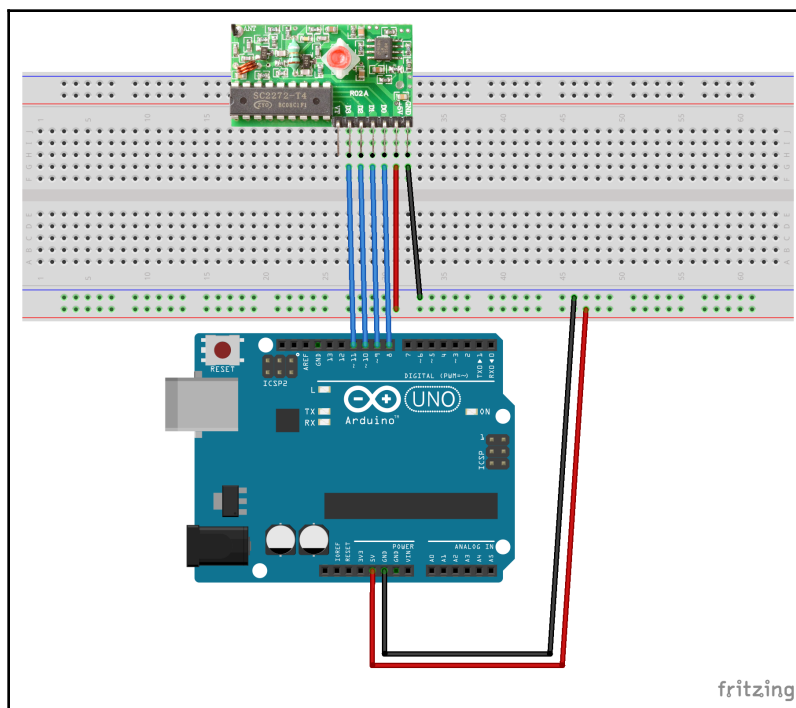
Для этих проектов вам потребуются следующие предметы:

- Arduino Uno или совместимая плата
- Один инфракрасный приемник
- Один или несколько инфракрасных передатчиков
- Одна пара РЧ-передатчика и приемника
- переключки
- Макетная плата

На следующей схеме показано подключение инфракрасного приемника к Arduino:



Вход 5 В и выводы заземления на ИК-приемнике подключены к соответствующим шинам на макетной плате. Сигнальный контакт подключен к цифровому 2-му выводу на Arduino. Теперь подключим радиоприемник к Arduino:

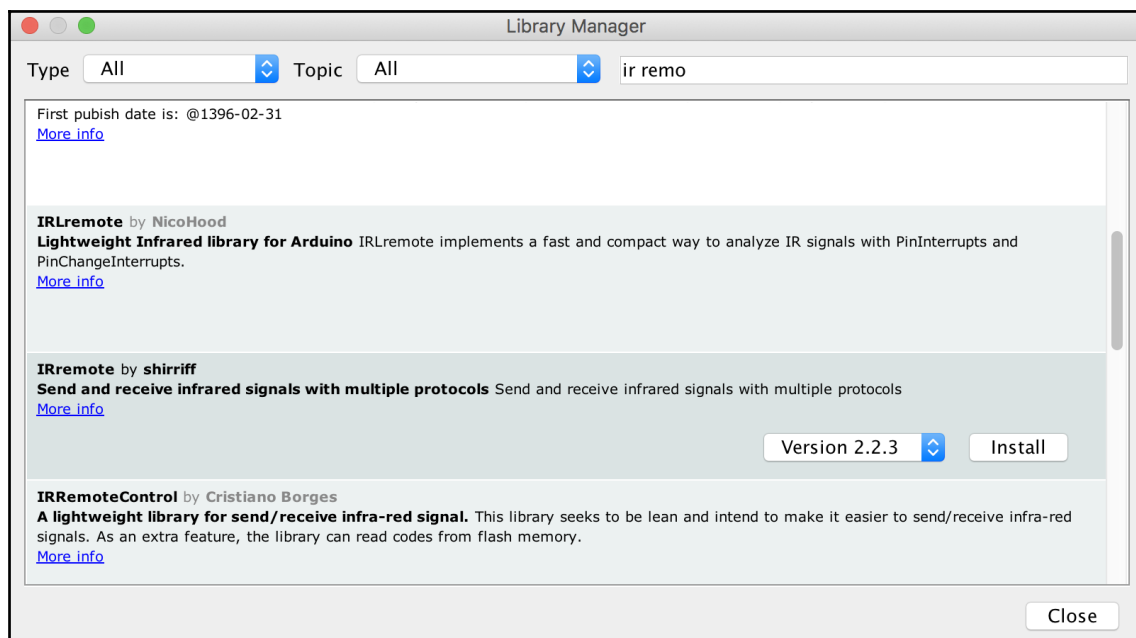


Вход 5 В и контакты заземления на РЧ-приемнике подключены к соответствующим шинам на макетной плате.

Четыре выходных вывода на приемнике подключены к 8, 9, 10 и 11 цифровым контактам на Arduino. При нажатии кнопки на передатчике соответствующий выход приемника получает **ВЫСОКИЙ** уровень.

Теперь посмотрим на код наших проектов.

Прежде чем мы сможем начать писать код, который будет считывать ввод с инфракрасного приемника, нам нужно будет загрузить библиотеку **IRremote** с помощью shirriff. На следующем снимке экрана показаны библиотека и версия, которые мы будем использовать:



После загрузки библиотеки нам нужно будет начать с импорта файла заголовка для библиотеки IRremote и создания глобальных переменных и директив. В следующем коде показано, как это сделать:

```
#include <IRremote.h>

#define IR_PIN 2
IRrecv ir(IR_PIN);
decode_results irCode;
```

В предыдущем коде мы начинаем с включения файла заголовка IRremote.h в наш проект. Затем мы определяем, что инфракрасный приемник подключен к контакту 2 на Arduino. Затем мы создаем экземпляр типа IRrecv, который используется для чтения ввода с ИК-приемника. Наконец, мы создаем экземпляр типа decode_results, который используется для хранения значений от ИК-приемника.

Теперь нам нужно добавить код инициализации в функцию setup (). Следующие коды показывают функцию setup () для этого примера:

```
void setup()
{
  Serial.begin(9600);
  ir.enableIRIn();
}
```

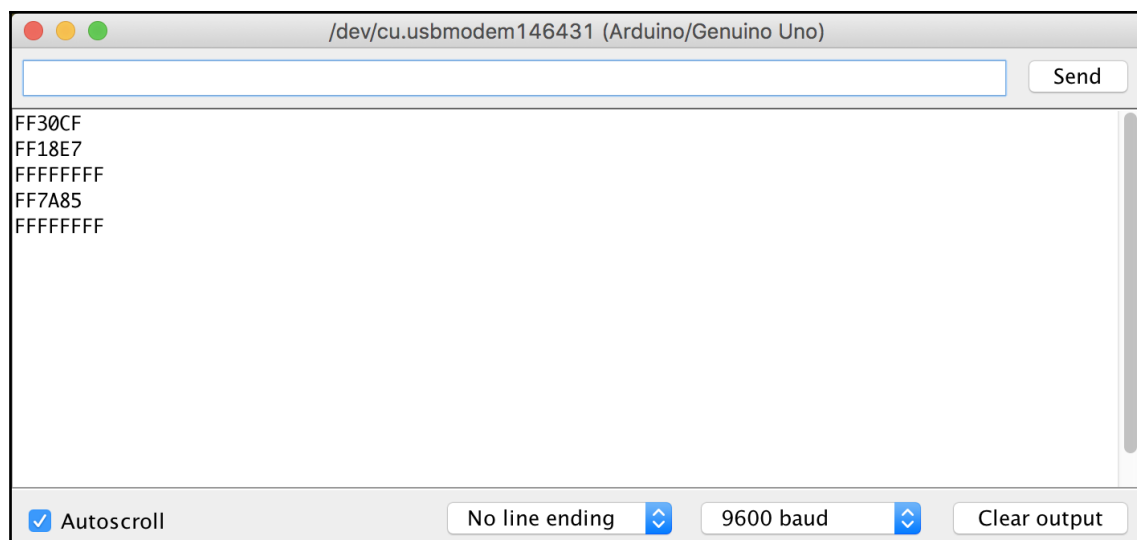
В этом примере мы начинаем с инициализации последовательного монитора, чтобы мы могли распечатать результаты. Затем мы вызываем функцию enableIRIn () из экземпляра типа IRrecv, которая подготовит Arduino к считыванию входных данных с ИК-приемника.

В функции loop () мы ищем вход от ИК-приемника и распечатываем коды для кнопок, нажатых на пульте дистанционного управления. Следующий код показывает, как будет выглядеть функция loop ():

```
void loop()
{
  if (ir.decode(&irCode))
  {
    Serial.println(irCode.value, HEX);
    ir.resume();
  }
  delay(100);
}
```


В функции `loop ()` мы используем функцию `decode ()`, передавая экземпляр типа `decode_results`, чтобы прочитать код нажатой кнопки. После получения кода функция `Serial.println ()` используется для вывода кода на последовательную консоль. Мы откладываем выполнение приложения на 100 миллисекунд, чтобы дать пользователю возможность отпустить кнопку до того, как будет отправлен повторный код. Наконец, вызывается функция `resume ()`, чтобы снова начать прослушивание результатов.

Результат кода должен выглядеть примерно так, как на следующем снимке экрана:



Автомобильный тр3-пульт, показанный ранее в этой главе, использовался для получения результатов, показанных на предыдущем снимке экрана. FF30CF - это код для кнопки номер 1, код FF18E7 - это кнопка с номером 2, а код FF7A85 - это кнопка с номером 3. Результат FFFFFFFF означает, что кнопка удерживается нажатой; поэтому снова следует использовать последний действительный код.

Вы, вероятно, захотите где-нибудь сохранить этот проект, потому что он очень полезен для получения действительных кодов для кнопок на ваших пультах дистанционного управления. Когда у вас есть коды, вы можете затем использовать библиотеку **IRremote** в других своих проектах и выполнять все необходимые действия в зависимости от кодов, возвращаемых приемником.

РЧ-приемник немного легче читать, потому что нам не нужна внешняя библиотека для его чтения, потому что на передатчике есть один вывод на каждую кнопку. Когда пользователь нажимает кнопку, соответствующий вывод на приемнике принимает высокий уровень.

Мы начнем код с определения того, какие выводы на Arduino подключены к выводам на РЧ-приемнике. Если вы подключили РЧ-приемник, как показано на диаграмме Фритцинга, кнопки будут определены следующим образом:

```
#define BUTTON_A 10
#define BUTTON_B 8
#define BUTTON_C 11
#define BUTTON_D 9
```

Затем нам нужно настроить эти выводы для ввода в функции `setup()`, а также инициализировать последовательный монитор. В следующем коде показана функция `setup()` для этого проекта:

```
void setup() {
  pinMode(BUTTON_A, INPUT);
  pinMode(BUTTON_B, INPUT);
  pinMode(BUTTON_C, INPUT);
  pinMode(BUTTON_D, INPUT);
  Serial.begin(9600);
}
```

В функции `loop()` нам нужно прочитать каждый из выводов на наличие ВЫСОКОГО уровня, и при подтверждении этого выполнить ту функцию, которая необходима в вашем проекте. Для этого проекта мы просто распечатываем информацию о том, что кнопка была нажата. Ниже показана функция `loop()` для этого проекта:

```
void loop() {
  int valA = digitalRead(BUTTON_A);
  if (valA == HIGH) {
    Serial.println("Button A");
  }

  int valB = digitalRead(BUTTON_B);
  if (valB == HIGH) {
    Serial.println("Button B");
  }

  int valC = digitalRead(BUTTON_C);
  if (valC == HIGH) {
    Serial.println("Button C");
  }

  int valD = digitalRead(BUTTON_D);
  if (valD == HIGH) {
    Serial.println("Button D");
  }
  delay(100);
}
```

Когда этот код будет запущен, вы сможете увидеть, какие кнопки были нажаты в последовательном мониторе.

Добавление пульта дистанционного управления к вашему проекту может показаться просто приятным занятием; тем не менее, это может действительно повысить удобство использования вашего проекта, а также избавить вас от необходимости постоянно вставать, чтобы взаимодействовать с ним. Теперь давайте посмотрим на нашу задачу в этой главе.

В этой главе мы рассмотрели два типа устройств дистанционного управления. Первым был ИК-пульт, который требует прямой видимости проекта и может иметь множество различных кнопок. Радиочастотный пульт дистанционного управления хорош, когда пульт дистанционного управления должен работать на больших расстояниях от устройства или даже в другой комнате.

Существует множество других способов создания пультов дистанционного управления с использованием беспроводных сигналов, таких как радио Zigbee или даже Wi-Fi; Однако, решая эту задачу, мы хотим, чтобы вы мыслили нестандартно и начали расширять свои горизонты. Задача этой главы - продумать способы удаленного управления вашим устройством без использования беспроводного сигнала.

Возможно, вы сейчас качаете головой, задаваясь вопросом, что мы подразумеваем под удаленным управлением проектом без использования беспроводного сигнала. Одним из примеров этого может быть трещотка. Клаппер - это электрический выключатель, активируемый звуком. Вы хлопаете один раз, и переключатель включается, вы снова хлопаете, и переключатель выключается. Другой пример - датчик движения, который управляет внешним освещением. Если датчик движения обнаруживает движение, он включает свет. Теперь попробуйте мыслить нестандартно и придумайте другие способы управления устройством без использования беспроводного сигнала.

В этой главе мы увидели, как использовать ИК-пульт и ВЧ-пульт с Arduino. Вам также было предложено мыслить нестандартно и думать о других способах удаленного управления своим проектом без использования беспроводного сигнала. Причина, по которой эта задача была в этой последней главе проекта, заключалась в том, чтобы заставить вас начать мыслить нестандартно при разработке своих проектов, потому что нестандартное мышление и создание новых и улучшенных способов что-то делать - вот что вдохновляет людей в проектах такого типа.

Это также может принести вам много денег, если вы сможете монополизировать свой проект. В следующей главе мы поговорим о том, как можно использовать знания, полученные из предыдущих глав проекта, для создания простого робота. Мы не будем писать код или проектировать схемы за вас. Вместо этого мы покажем вам, как соединить части, которые вы изучили в книге, чтобы вы могли спроектировать своего собственного робота или создать другие проекты.

Создание робота

Когда я впервые начал работать с платами Arduino, моей первоначальной целью было построить робота. Однако я понятия не имел, с чего начать. У меня было так много вопросов? Какие моторы мне использовать? Как мне запитать робота? Нужен ли мне отдельный источник питания для двигателей? Как работает объезд объектов? Эта глава написана, чтобы помочь вам ответить на многие из этих вопросов и показать вам, что после прочтения предыдущих глав этой книги у вас теперь достаточно знаний, чтобы спроектировать и построить своего собственного робота.

В этой главе мы обсудим:

- Как использовать знания, полученные в книге, для создания полностью рабочего робота
- Ряд дополнительных советов и подсказок, которые помогут вам в ваших проектах
- Какие еще проекты мы можем построить с Arduino помимо этой книги?

Мы закончим главу, предложив вам создать свой собственный проект, а затем поделиться им с нами.

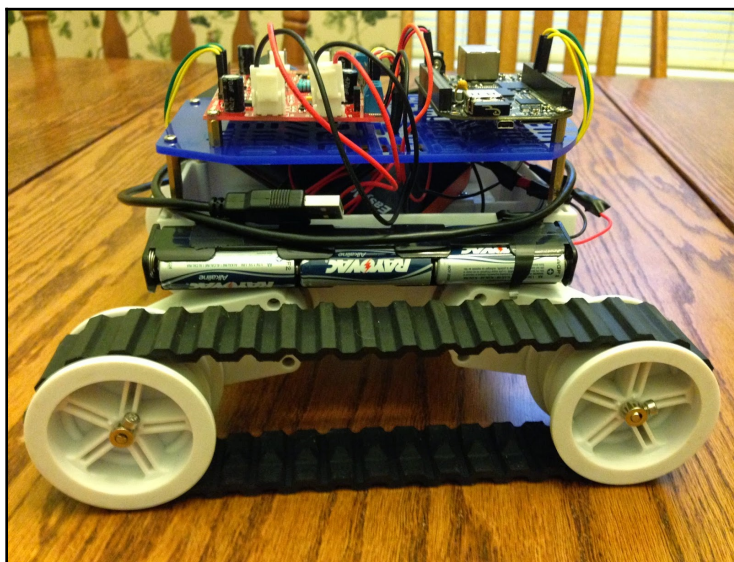
В этой главе, в отличие от предыдущих глав о проектах, мы не будем проектировать и строить какой-то конкретный проект.

Вместо этого мы покажем вам, как вы можете взять знания, полученные в предыдущих главах, и использовать их для создания собственного робота. Мы также дадим вам различные советы и подсказки, которые помогут избежать некоторых ошибок, которые люди совершают, когда впервые начинают создавать роботов.

Когда мы начали проектировать нашего первого робота, первое, что мы сделали, - это решили, что он должен выполнять максимальное количество действий. Это было БОЛЬШОЙ ошибкой, потому что список обязанностей, которые должен выполнять робот, был обширным. Это должно было быть нечто среднее между R2-D2, Wall-E и Commander Data (упоминаются роботы из фантастических фильмов). Это должен был быть самый крутой робот нашего проекта, а затем, как вы, наверное, догадались, мы пытались объять необъятное.

При первоначальном проектировании ваших первых роботов вы должны определиться с проектирования / покупки шасси (корпуса) робота.

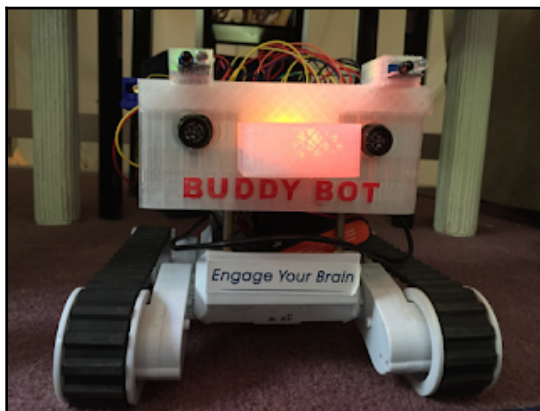
Один из самых простых способов начать сборку робота - купить готовое шасси танка. Некоторые из этих шасси даже имеют встроенные двигатели и контроллеры двигателей или предназначены для совместной работы, что упрощает начало работы. Первый робот, который я построил, использовал шасси Rover 5 Tank Chassis и плату контроллера двигателя Rover 5, разработанную и изготовленную Dagu Electronics. Вот фотография моего первого робота:



Были куплены все детали для этого робота, включая верхнюю пластину синего цвета, к которой крепятся все платы с электронной начинкой.

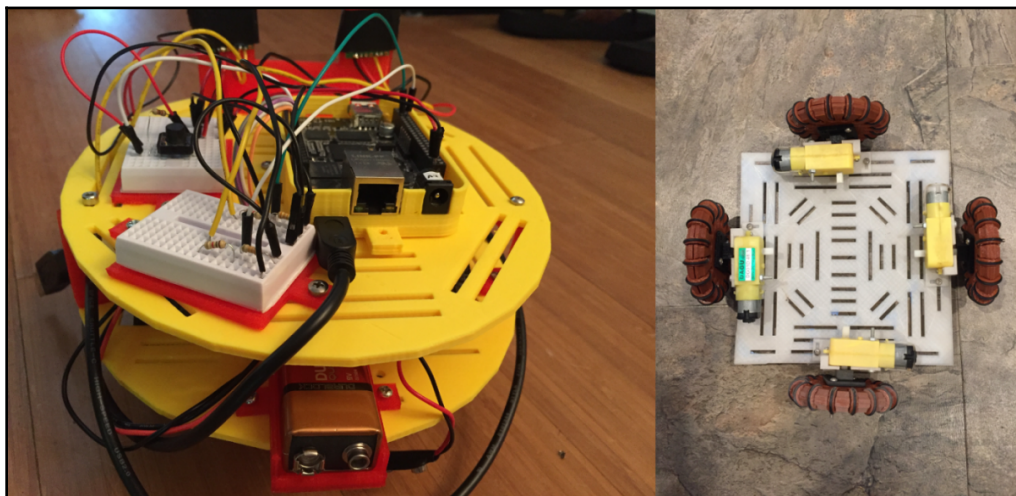
Покупка всех этих деталей может обойтись дорого, особенно когда вы начнете добавлять к роботу дополнительные датчики и оборудование.

Если вы планируете построить множество роботов или даже создать множество других прототипов с помощью Arduino, следует использовать 3D-принтер. Покупка 3D-принтера стоит денег, потому что вместо того, чтобы покупать готовые детали для шасси, вы можете спроектировать и напечатать свои собственные детали. В качестве примера, совсем недавно я снял шасси Rover 5 и создал другого робота - Inamed BuddyBot. Он был создан с использованием деталей, которые я спроектировал и распечатал с помощью 3D-принтера, что позволило мне придать BuddyBot тот вид, который я хотел. На следующей фотографии показана передняя часть BuddyBot:



Вы даже можете распечатать все шасси, если хотите. На следующей фотографии показано шасси для двух роботов, которые я спроектировал и напечатал на своем 3D-принтере.

Шасси изображенное справа, - экспериментальное шасси, которое я проектирую. Я также распечатал колеса Omni для шасси справа:



Шасси бывают всех форм и размеров. Поиск или создание идеального робота обычно является одним из самых простых этапов создания вашего робота. Главное - убедиться, что у вас есть шасси, которое допускает расширения. Если вы заметили, на предыдущей фотографии пластины имеют длинные и узкие прямоугольные бороздки. Они разработаны таким образом, чтобы мы могли легко удалять и добавлять новые детали. Вы заметите, что маленькие макетные платы на левом роботе на предыдущей фотографии можно легко отвинтить и снять или переместить в другую часть робота.

Создавая модульного робота, мы можем очень быстро расширить его функциональные возможности. Вы захотите избежать шасси с ограниченной возможностью расширения, потому что, если вы не проектируете робота для конкретной функции, вы всегда будете думать о новых компонентах и функциях, которые вы хотели бы добавить к нему.

До сих пор мы много говорили о шасси, но как насчет определения того, как будет двигаться робот? В шасси Rover 5, которое я показал ранее в этой главе, для движения используются гусеницы танка. Желтый робот, который был на предыдущей фотографии, использует стандартные и очень дешевые автомобильные мотор и колеса Arduino (которые показаны на следующей фотографии), в то время как экспериментальное шасси использует колеса Omni, которые я распечатал.



Использование танковых гусениц для передвижения вашего робота дает вам возможность перемещаться практически по любой местности, однако они дороже стандартных колес / моторов, и шасси также должно быть адаптировано для них. Вам также могут понадобиться двигатели с более высоким крутящим моментом, но мы поговорим о двигателях в следующем разделе.

Использование автомобильного двигателя и колеса Arduino - гораздо более дешевый вариант, чем гусеницы танка, однако вы ограничиваете использование своего робота в помещении, если у вас нет высококлассного шасси, такого как Bogie Runt Rover, показанный на следующей фотографии:



Bogie Runt Rover - очень хорошее шасси для работы, но я определенно рекомендовал бы иметь 3D-принтер, чтобы вы могли распечатать для него свои собственные детали расширения.

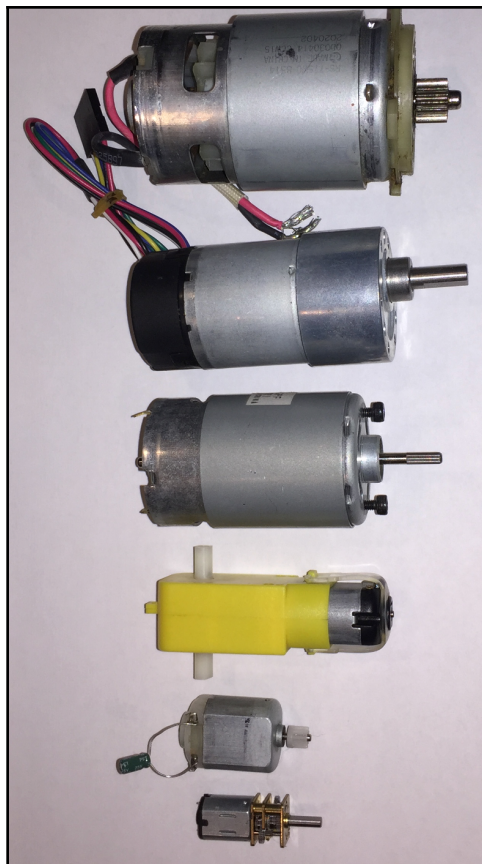
Омни-колеса - это особый тип колес, у которых вокруг колеса есть маленькие диски, которые будут вращаться, что снижает трение. Если вы спросите людей, которые использовали колеса Omni, их мнение о них, вы получите противоречивые ответы на весь спектр этих колес.

Вы также можете сделать роботов, которые ходят. Эти типы роботов выходят за рамки нашей книги, и я бы порекомендовал для вашей первой пары роботов использовать гусеницы или колеса танка.

Теперь давайте посмотрим на двигатели и на то, как мы будем их приводить в действие.

Двигатели и мощность

Решение, какой двигатель использовать и как их приводить в действие, может быть одним из самых сложных решений, которые вы принимаете, когда начинаете конструировать роботов, потому что есть очень много вариантов. На следующей фотографии показаны некоторые из двигателей, которые я использовал в своих проектах:



Вы можете сэкономить много денег на приобретении подержанных двигателей, и есть несколько способов их приобрести. Я купил множество подержанных игрушечных автомобилей с дистанционным управлением и другую электронику, такую как старые DVD-плееры, и снял с них моторы. Вот откуда взялись два самых маленьких мотора, показанных на предыдущей фотографии; однако, если вам нужны более мощные двигатели, попробуйте снять их с дрели. Самый большой двигатель на предыдущей фотографии был от старой дрели Ryobi 12 В.

Когда вы впервые начинаете создавать роботов, я бы рекомендовал начать с двигателя, используемых в игрушечных автомобилях. Они очень просты в использовании и могут работать с напряжением от 3 В до 12 В. Рекомендуемый диапазон напряжения от 6 до 8 В. Однако я часто использую аккумулятор на 12 В с максимальным током 1,3 А для питания этих двигателей. При покупке двигателей внимательно ознакомьтесь со спецификациями, чтобы убедиться, что двигатель рассчитан на источник питания, который вы используете в своем проекте.

Для управления моторами вам понадобится контроллер мотора. Контроллер двигателя L298, который мы показали в главе 15 «Двигатели постоянного тока и контроллеры двигателей», является идеальным контроллером двигателя для начала. Он работает в широком диапазоне напряжений от 5 В до 46 В и тока до 2 А. Вы можете вернуться к главе 15 «Двигатели постоянного тока и контроллеры двигателей», чтобы узнать, как использовать этот контроллер.

Поначалу питание двигателей может сбивать с толку, особенно при покупке батарей. Начнем с того, что лучше не запитывать двигатели от источника питания 5V Arduino. Для небольших робототехнических проектов можно использовать батарейки типа АА или батарею на 9 В. Для шасси, которое может работать с большей батареей, я бы порекомендовал приобрести небольшую батарею 12 В, такую как Duracell Ultra 12V1.3Ah, показанная на следующей фотографии:



Одним из важнейших факторов при выборе аккумулятора являются характеристики двигателей и их количество. Когда вы подключаете двигатели через контроллеры двигателей к батарее, вы будете подключать их параллельно друг другу. Если вы вспомните сравнение последовательных и параллельных цепей в главе 3 «Принципиальные схемы», вы вспомните, что в параллельной цепи каждая ветвь будет иметь максимальное напряжение, которое выводится из источника питания. Однако ток будет разделен между ветвями. Это означает, что если наша батарея может выдавать 12 В и 1,2 А, и мы пытаемся запитать 6 двигателей, то каждый двигатель будет иметь 12 В и может потреблять 200 мА (1,2 А, разделенные на 6), если каждый двигатель вращается с одинаковой скоростью. Таким образом, ранее показанный Duracell Ultra подойдет для питания шести автомобильных двигателей Arduino, но если вы планируете использовать более мощные двигатели, вам может потребоваться аккумулятор большей емкости.

Вы могли заметить, что батарея Duracell была рассчитана на 12 В и 1,3 Ач. Не путайте А-рейтинг с Ампер. Батарея оценивается по емкости, и 1,3 Ач означает, что она может непрерывно подавать 1,3 А в течение 1 часа, прежде чем вам потребуется ее подзарядить.

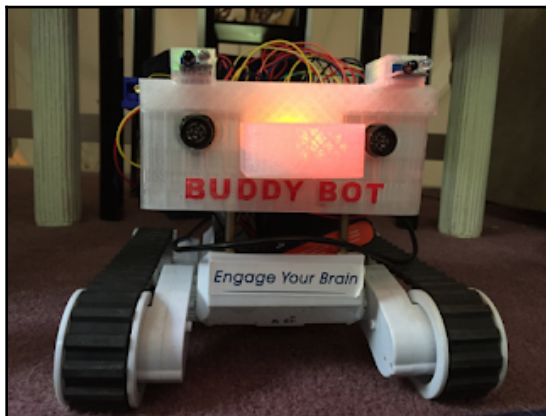
Одно хорошее правило, которое следует использовать при подборе аккумулятора, который будет питать ваш роботизированный проект: лучше иметь больше мощности, чем меньше. Это правило отлично работает, если вы не доведете его до крайности и не попытаетесь использовать аккумулятор 12 В для питания маленькой радиоуправляемой машины с двумя маленькими моторами.

С помощью контроллера мотора L298 мы можем управлять скоростью и направлением колес, что позволяет нам управлять роботами с помощью колес или гусениц танка. Робот повернется, если колеса или гусеница с одной стороны робота вращаются быстрее, чем колеса или гусеница с другой стороны. Чем выше разница в скорости между двумя сторонами, тем быстрее будет поворачиваться робот. Вы также можете вращать робота на месте, если колеса или гусеницы с одной стороны робота вращаются в прямом направлении, в то время как колеса или гусеница с другой стороны вращаются в обратном направлении.

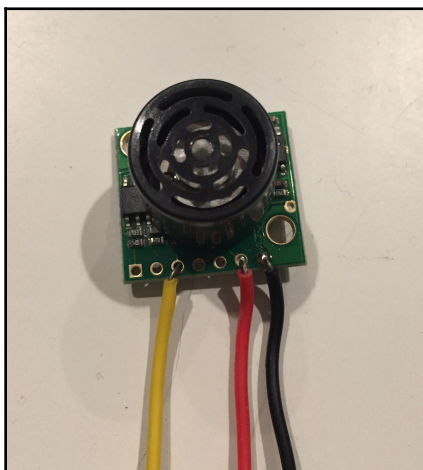
—

Если вы хотите построить автономного робота, вам понадобится некоторая форма предотвращения препятствий и обнаружения столкновений с логикой, которая сообщает роботу, как перемещаться между препятствиями. Мы показали, как использовать несколько датчиков предотвращения препятствий и обнаружения столкновений в главе 10 «Избегание препятствий и обнаружение столкновений», но вопрос может заключаться в том, как разработать логику для обхода или избегания обнаруженных объектов.

Прежде чем мы начнем обсуждать логику избегания препятствий, давайте еще раз взглянем на BuddyBotrobot:



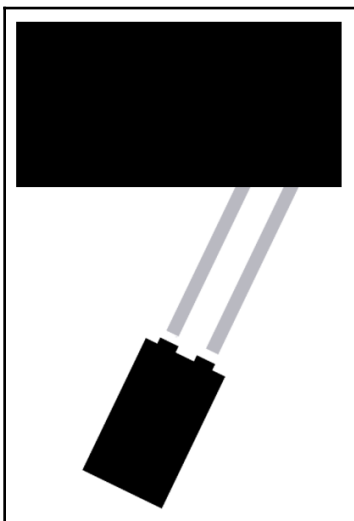
Если вы внимательно посмотрите на «глаза» на BuddyBot, вы можете узнать в них ультразвуковой дальномер MaxSonarEZ1, который обсуждался в главе 10 «Избегание препятствий и обнаружение столкновений». Чтобы освежить вашу память, MaxSonar EZ1 отправляет ультразвуковой импульс в определенном направлении. Если на пути импульса есть объект, он отражается обратно в виде эха. Датчик определяет расстояние до объекта, измеряя время, необходимое для получения эхо-сигнала. На следующей фотографии показано, как выглядит MaxSonar EZ1:



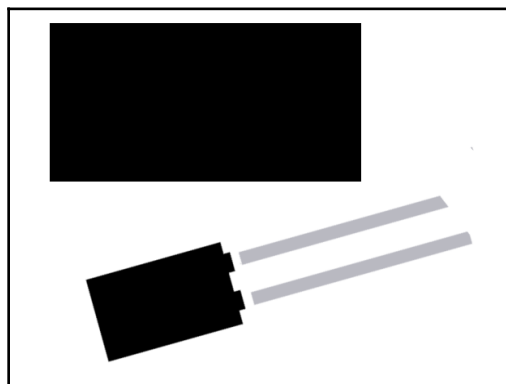
BuddyBot использует два датчика MaxSonar, чтобы помочь с логикой уклонения от препятствий. Давайте посмотрим, как это работает. Первое, что нам нужно сделать, это определить, на каком расстоянии мы хотим запустить логику уклонения от препятствий. Например, нам, вероятно, не нужно беспокоиться об избегании препятствий на расстоянии пяти футов, если мы строим маленького робота, который перемещается по дому. Если у нас есть большой робот, который перемещается по фабрике, возможно, пять футов - это разумное расстояние, с которого можно начать логику уклонения от препятствий.

Среда, в которой будет перемещаться робот, играет большую роль в разработке логики предотвращения препятствий.

Как только правильное расстояние для запуска логики уклонения от препятствий определено, мы можем приступить к построению логики. Используя два датчика, мы можем определить, под каким углом робот приближается к объекту. Следующая диаграмма показывает, как робот приближается к объекту под небольшим углом:



Когда робот приближается к объекту под таким углом, мы можем сравнить расстояние, сообщаемое обоими датчиками, и определить, что объект находится ближе к левой стороне робота. Определив, что левая сторона робота ближе к объекту, мы можем использовать некоторую фундаментальную логику, чтобы сказать роботу повернуть направо, пока объект не выйдет за пределы диапазона, как показано на следующей диаграмме:



Как только объект окажется вне зоны досягаемости, мы снова сможем двигаться вперед. Это очень простой способ избежать препятствий и требует постоянного опроса датчика. Для более дешевого варианта, поскольку датчики MaxSonar довольно дороги по сравнению с другими датчиками, мы могли бы использовать инфракрасный датчик избегания препятствий, который также описан в главе 10 «Предотвращение препятствий и обнаружение столкновений». Причина, по которой я предпочитаю датчик MaxSonar, заключается в том, что луч инфракрасного излучения, излучаемый ИК-датчиком, намного более узкий, чем звуковая волна, излучаемая датчиком MaxSonar. С более широким лучом датчика маловероятно, что мы пропустим объекты, расположенные немного левее / правее или выше / ниже датчика.

Вы также захотите установить вокруг робота датчики столкновения; их можно использовать для обнаружения, когда робот на что-то наталкивается. Они очень хорошо работают на задней части робота, чтобы обнаружить, если робот на что-то наткнется, когда он движется задним ходом. Избегание препятствий может быть очень сложной задачей, с которой легко начать, как показано здесь, но может быть потому, что она очень сложна.

Теперь давайте посмотрим, как мы можем управлять роботом удаленно.

Радиочастотный пульт, который мы видели в главе 18 «Дистанционное управление Arduino», является гораздо лучшим выбором для удаленного управления роботом, чем инфракрасный пульт, потому что радиочастотный пульт не требует прямой видимости для сигнала. Единственная проблема с радиочастотным пультом дистанционного управления заключается в том, что обычно не хватает кнопок для всего, что мы хотим от наших роботов. Это отсутствие кнопок можно преодолеть, превратив робота в автономного робота, в котором он может использовать избегание объектов, чтобы перемещаться самостоятельно, а затем использовать пульт дистанционного управления, чтобы сообщить роботу о выполнении определенной задачи. Эти задачи могут включать в себя такие вещи, как приказ роботу начать работу. / перестать двигаться, включить музыку через динамик или принести вам напиток из холодильника.

В конце главы 18, Удаленное управление Arduino, вам было предложено продумать нестандартные способы удаленного управления проектом, кроме использования беспроводного сигнала. Подумайте о своих ответах на этот вызов и посмотрите, сможете ли вы использовать их для управления устройством. робот.

Один из моих любимых способов управления роботом - распознавание голоса с помощью MOVIshield, которое мы видели в главе 14 «Распознавание речи и синтезатор голоса». С помощью MOVIshield мы можем запрограммировать такие команды, как поворот направо, поворот налево, остановка или любые другие действия, которые должен делать ваш робот.

Другой датчик, который мы можем использовать, - это звуковые датчики, которые мы помещаем три или четыре по кругу или квадрату вокруг робота, чтобы он мог определять направление, откуда исходит звук, и затем двигаться в этом направлении. Мы также могли бы использовать одиозвуковой датчик, чтобы запускать или останавливать робота, когда мы хлопаем в ладоши или издаем другой громкий звук. Давайте посмотрим, как мы можем предоставить обратную связь пользователю робота.

Мы всегда хотим предоставить какой-либо способ обратной связи, чтобы сообщить нам, что происходит с роботом.

Это очень удобно, когда мы программируем робота для отладки. Если вы помните изображение BuddyBot из ранее в этой главе, нос подсвечивался разноцветными светодиодами. Цвет светодиода указывал, что робот должен был делать, и обнаружил ли он препятствия с левой или правой стороны. Увидев, какого цвета светодиод, я знал, что должен делать робот, а если он этого

не делал, я знал, что что-то не так с программным обеспечением или оборудованием.

Использование многоцветных светодиодов - один из самых простых и быстрых способов добавить отзыв от нашего робота. Мы можем очень быстро установить разные цвета для обозначения различных действий. Если нам нужно указать несколько действий одновременно, мы могли бы добавить несколько светодиодов без больших затрат. Я предпочитаю многоцветные светодиоды одноцветным, потому что мы можем использовать разные цвета для обозначения разных вещей, в то время как одноцветные светодиоды либо включены, либо выключены, и поэтому они могут указывать только на один элемент.

Еще один способ дать обратную связь - использовать звуки. В главе 12 «Развлечения со звуком» мы увидели, как Arduino может воспроизводить звук с помощью различных динамиков. Если вы когда-нибудь использовали Roomba или другой автономный робот-пылесос, вы знаете, что когда что-то не так, например, вакуум застрял на чем-то, он издает звук, чтобы сообщить владельцу, что он застрял. Воспроизведение звука - еще один простой способ добавить обратной связи от робота.

В главе 12 «Использование ЖК-дисплеев» мы говорим, как можно использовать ЖК-дисплеи для передачи сообщений из нашего проекта. Добавление ЖК-дисплея позволяет вашему проекту предоставлять пользователю точную информацию. Они могут быть в форме слов или изображений.

Отзывы пользователей должны быть одной из первых вещей, которые вы вкладываете в свой проект, потому что их можно использовать для устранения неполадок при разработке проекта. Теперь давайте поговорим о том, как заставить вещи вращаться.

В главе 16 «Сервомоторы» мы увидели, как можно использовать серводвигатель для открытия и закрытия клешни робота; однако серводвигатель может делать гораздо больше. С серводвигателем, как мы видели с роботизированной клешней, мы можем повернуть двигатель на определенный угол. Раньше я прикреплял датчик MaxSonar к серводвигателю и направлял датчик прямо вперед. Затем, когда датчик обнаруживал объект перед роботом, сервопривод поворачивал датчик в разные стороны, чтобы он мог определить наилучшее направление для движения. Это позволило мне создать автономного робота с возможностью обхода препятствий, используя только один датчик.

Мы также могли бы прикрепить источник света к серводвигателю, чтобы сделать вращающийся прожектор, который позволил бы вам видеть, где находится робот в темноте, или, в равной степени, это просто действительно крутое дополнение к вашему роботу, не имеющее никакой функциональной цели.

Важный совет, который я рекомендую, заключается в том, что когда вы впервые начинаете создавать роботов, избегайте попыток прикрепить роботизированную руку к вашему роботу. Роботизированные манипуляторы обычно весят слишком много для большинства небольших и средних корпусов роботов, и требуется сложное программирование, чтобы заставить их двигаться именно туда, куда вы хотите. Я не говорю, что вам не следует думать о добавлении роботизированной руки к вашему роботу. Тем не менее, это действительно продвинутый проект, на доведение которого потребуется много времени.

Если вы не увлекаетесь роботами, есть еще много других проектов, которыми вы можете заниматься с Arduino.
Давайте посмотрим на некоторые из них.

В главе 9 «Датчики окружающей среды» мы увидели, как использовать многочисленные датчики окружающей среды, такие как датчик температуры / влажности DHT-11 и датчик дождя. Мы могли бы использовать эти датчики с любыми дополнительными, например датчиком скорости ветра, для создания метеостанции. Просто помните, что вам нужно будет поместить Arduino и другие электронные компоненты в водонепроницаемый контейнер.

В главе 9 «Датчики окружающей среды» мы увидели, как Arduino может считывать температуру и влажность с помощью датчика температуры / влажности DHT-11. Если мы подключим оконный или переносной кондиционер или увлажнитель к плате реле, как описано в главе 17, «Использование реле», мы можем автоматически включить или выключить кондиционер и / или увлажнитель.

Фактически, используя релейную плату, вы можете включать или выключать практически любое устройство с питанием от сети переменного тока в зависимости от использования различных датчиков. Например, мы могли бы очень легко создать собственный клон Clapper, подключив звуковой датчик к Arduino и использование Arduino для включения или выключения шилда каждый раз, когда он обнаруживает громкий звук.

В главе 10 «Избегание препятствий и обнаружение столкновений» мы увидели, как использовать датчик Max Sonar.

Если мы прикрепим датчик к сервомотору, который мы видели в главе 16 «Сервомоторы», мы могли бы создать датчик приближения, который мог бы вращаться на 180 градусов для наблюдения за большей частью комнаты. При первом запуске необходимо выполнить начальный цикл, чтобы сопоставить, где находятся объекты, а затем отслеживать, не изменится ли что-нибудь после первоначального запуска. Если датчик приближения обнаруживает, что что-то находится ближе, чем должно быть, он может включить сигнал тревоги через динамик, как описано в главе 12 «Развлечения со звуком».

Это лишь некоторые из проектов, которые вы можете создать с помощью Arduino. А теперь последнее испытание.

Я начал учиться использовать макетные платы, такие как Arduino, чтобы создавать роботов. Возможно, вы захотите построить другие проекты, такие как метеостанция, использующая датчики температуры, влажности и дождя, или, возможно, систему безопасности, использующую датчик движения, описанный в главе 8, Датчик движения.

Что бы вы ни интересовали, я призываю вас сделать несколько супер крутых проектов с Arduino. Как только вы закончите, я хотел бы увидеть фотографии с описанием вашего проекта и даже опубликую некоторые из них в своем блоге, указав как авторские проекты. Если у вас есть видео вашего проекта на YouTube, я бы тоже хотел его увидеть.

Вы можете отправлять свои изображения и описания по адресу: mastering.arduino@gmail.com.

В первой части этой книги мы узнали об Arduino и базовой электронике. Эти главы были разработаны, чтобы дать вам, читателю, общее представление о том, как работает Arduino и как применять электронные компоненты к Arduino.

Затем мы узнали об инструментах разработки, которые мы можем использовать с Arduino, и о том, как запрограммировать Arduino. Эти главы дали вам общее представление об инструментах разработки, а также о языке программирования Arduino.

В последней главе мы объединили то, что узнали ранее в книге, и показали, как подключать различные компоненты к Arduino. Эти главы были разработаны, чтобы показать вам широкий спектр различных компонентов, которые по-разному взаимодействуют с Arduino. Мы надеемся, что это даст вам достаточно разнообразия, чтобы при покупке различных датчиков для своего собственного проекта вы понимали, как они взаимодействуют с Arduino, хотя они не были подробно описаны в этой книге.

В следующих двух главах мы рассмотрим радиомодули Bluetooth, чтобы увидеть, как мы можем реализовать двустороннюю связь в наших проектах.

Bluetooth LE

В этой книге до сих пор все внешние связи с нашими проектами Arduino осуществлялись в закрытой среде. Под закрытой средой мы подразумеваем, что наш проект просто получал информацию или указание с пульта дистанционного управления, и никакая информация не передавалась из проекта. Существует множество вариантов использования, когда нам нужно передать информацию из нашего проекта Arduino на внешнее устройство, такое как смартфон или другое устройство IoT. Когда возникает такая необходимость, одной из первых технологий, которая поднимается, является **Bluetooth Low Energy** - Bluetooth с низким энергопотреблением, также известный как Bluetooth LE или Bluetooth Smart.

В этой главе вы узнаете:

- Что такое Bluetooth LE
- Как работает радио Bluetooth LE
- Что такое профиль GAP
- Что такое профиль GATT
- Как использовать радиомодуль HM-10 Bluetooth LE с Arduino

Одно из наиболее частых недоразумений людей, незнакомых с технологией Bluetooth LE, заключается в том, что Bluetooth LE представляет собой облегченное подмножество Bluetooth Classic. Это неправда, поскольку Bluetooth Classic и Bluetooth LE - это два принципиально разных протокола с разными целями проектирования.

Большинство беспроводных технологий, таких как Wi-Fi и Bluetooth Classic, были разработаны для решения широкого круга задач; однако конструкция Bluetooth LE немного отличается. Первоначально созданный Nokia и известный как Wibree, основной целью разработки Bluetooth LE было создание стандарта радиосвязи с минимально возможным энергопотреблением, оптимизированного для низкой стоимости, низкой сложности и низкой пропускной способности.

Спецификации Bluetooth LE были выпущены как часть спецификаций Bluetooth 4.0 Core в июне 2010 года. Основные характеристики Bluetooth контролируются и обновляются Special Interest Group (SIG) - специальной группой по интересам Bluetooth .



Вы можете найти информацию о Bluetooth и загрузить спецификации с их сайта <https://www.bluetooth.com>; однако, если у вас более 2500 страниц, я бы рекомендовал вам прочитать эту главу вместо спецификаций Bluetooth, если вам не нужна помощь при бессоннице.

Скорость внедрения Bluetooth LE была намного выше, чем у большинства других беспроводных технологий. Причина этого - принятие стандарта Bluetooth LE в мобильной индустрии, где Apple и Google приложили значительные усилия для включения надежных стеков Bluetooth LE в операционные системы iOS и Android и разработки простых в использовании и понятных API-интерфейсов Bluetooth LE для разработчиков. Это упрощает разработчикам создание и взаимодействие с устройствами, имеющими радиомодули Bluetooth LE.

Причина, по которой мобильная индустрия настаивает на внедрении Bluetooth LE, заключается в том, что устройства, которые подключаются с помощью Bluetooth LE, потребляют гораздо меньше энергии, отсюда и название Bluetooth Low Energy, по сравнению с другими беспроводными технологиями, такими как Bluetooth Classic и Wi-Fi. Это приводит к увеличению времени автономной работы их телефонов, что делает клиентов более счастливыми.

Устройства Bluetooth бывают трех типов, каждый из которых поддерживает либо Bluetooth Classic, либо Bluetooth LE, либо и то, и другое. В следующей таблице показано, что поддерживает каждый тип:

Device Type	Bluetooth Classic Support	Bluetooth LE Support
Pre-4.0 Bluetooth	Yes	No
Single-Mode	No	Yes
Dual-Mode	Yes	Yes

Хотя спецификации Bluetooth 5.0 были выпущены в июне 2016 года, на момент написания этой книги было очень мало модулей Bluetooth для Arduino, поддерживающих эту новую спецификацию. Фактически, в настоящее время также очень мало поддерживается спецификации Bluetooth 4.1 или 4.2; поэтому в этой книге мы сосредоточимся на спецификациях Bluetooth 4.0, зная, что Bluetooth 5.0, 4.2 и 4.1 обратно совместимы с этим стандартом.

Чтобы разрабатывать устройства IoT, использующие Bluetooth LE, нам действительно нужно понимать технологию, чтобы мы знали, когда на самом деле ее использовать. Поэтому мы углубимся в эту технологию намного дальше, чем с другими технологиями в этой книге. Мы начнем с изучения характеристик радиостанции.

Bluetooth LE

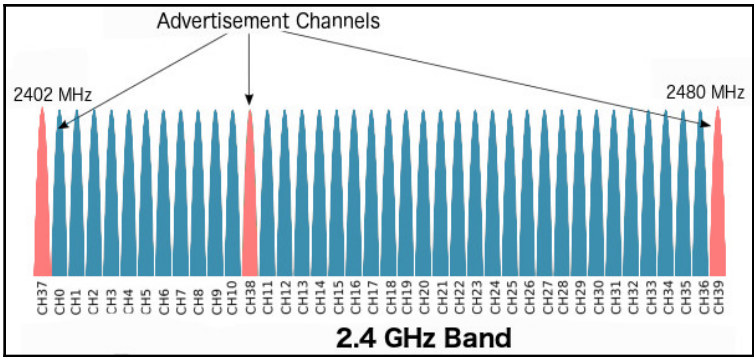
Поскольку мы будем использовать Bluetooth LE 4.0 для всех проектов в этой книге, следующие спецификации относятся к этому стандарту:

Диапазон	До 100 метров
Радиочастота	2.402 - 2.481 GHz
Радиоканалы	40 (37 данных и 3 рекламных)
Максимальная скорость передачи данных ОТА	1 Mbit/s
Пропускная способность данных приложений	0.125 Mbit/s
Сетевые топологии	Точка-точка
Сетевой стандарт	IEEE 802.15.1

Bluetooth LE имеет максимальную дальность действия 100 метров, но это очень зависит от окружающей обстановки. Когда подключенные устройства находятся в помещении, дальность действия будет значительно сокращена из-за стен и других препятствий, через которые должен проходить радиосигнал. Как правило, диапазон около 100 метров ограничен, если только мы не находимся на открытом воздухе. Даже в этом случае редко удастся достичь 100-метровой дальности.

Радиомодуль Bluetooth LE работает на более чем 40 каналах в диапазоне от 2,402 ГГц до 2,481 ГГц. Из этих каналов 37 зарезервированы для передачи данных и три - для рекламы. Причина множественности каналов - Bluetooth LE использует скачкообразную перестройку частоты для уменьшения помех. Три рекламных канала используются для обнаружения устройств. После обнаружения устройства один и тот же канал используется для обмена начальными параметрами соединения. После обмена параметрами соединения для связи используются обычные каналы данных.

На следующем рисунке показаны каналы, используемые Bluetooth LE:



Bluetooth LE разработан для работы с низким энергопотреблением, и лучший способ избежать потребления энергии - это выключать радио как можно чаще и на как можно дольше. Для Bluetooth LE это достигается путем отправки коротких пакетов на определенной частоте, а в промежутках между этими пакетами радио отключается. Это часто называют скачком в режим ожидания, поскольку радиоприемник по существу отправляет информацию так быстро, как только может, а затем отключается на короткий период времени.

Самая большая жертва, которую мы приносим в пользу низкого энергопотребления Bluetooth LE, - это пропускная способность приложений. На предыдущем графике мы видели, что у радиомодуля Bluetooth LE максимальная скорость передачи данных по воздуху составляет 1 Мбит / с. Однако мы также указали, что пропускная способность приложения составляет всего 0,125 Мбит / с. Это означает, что радио теоретически может передавать 1 Мбит / с; однако с учетом ограничений, наложенных на радиостанцию по экономии энергии, максимальная скорость передачи данных составляет 0,125 Мбит / с. В реальном приложении мы фактически никогда не увидим ничего близкого к этой скорости передачи данных.

Давайте рассмотрим гонку за простоями и ограничения, налагаемые стандартами Bluetooth LE, чтобы понять, почему скорость передачи данных такая низкая. Для начала, спецификации Bluetooth LE определяют, что интервал соединения, который представляет собой интервал времени между двумя последовательными событиями соединения (когда два устройства обмениваются данными), должен составлять от 7,5 мс до 4 с. Это означает, что если мы установим интервал подключения на минимально возможное время (7,5 мс), у нас будет максимум 133 события подключения в секунду.

Радиостанция может передавать до шести пакетов данных на одно событие соединения, причем каждый пакет данных может содержать максимум 20 байтов пользовательских данных. Это дает максимум 120 байтов на каждое событие соединения.

Если мы сложим всю информацию вместе, мы получим следующую формулу:

$$133 \text{ connection events per second} * 120 \text{ bytes per event} = 15960 \text{ bytes/second} \\ \text{or} - .125 \text{ Mbit/second}$$

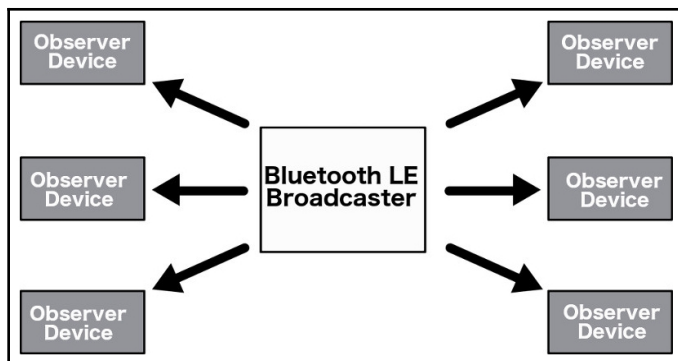
Это показывает, что максимальная пропускная способность данных будет 0,125 Мбит / с; однако, как мы упоминали ранее, даже это число никогда не будет достигнуто, потому что обычно мы никогда не исчерпываем максимальное количество байтов на пакет или не имеем 133 событий соединения за одну секунду. Сами устройства могут добавлять дополнительные ограничения на интервал соединения и количество пакетов данных на каждое соединение. В лучшем случае мы обычно видим около 5-10 Кбайт в секунду для пропускной способности данных. Это означает, что мы обычно хотим использовать технологию Bluetooth LE только при обмене короткими пакетами данных и избегать этого, когда мы хотим обмениваться большими объемами или даже передавать данные в потоке.

Теперь давайте посмотрим на топологию сети для соединений Bluetooth LE.

Устройства Bluetooth LE могут связываться с другими устройствами Bluetooth LE посредством вещания или установленного соединения. У каждого из этих методов есть свои достоинства и недостатки. Мы начнем с рассмотрения топологии сети, когда устройства общаются посредством широковещательной передачи. В проектах в этой главе мы сосредоточимся на обмене данными по установленным соединениям, но хорошо знать, как можно обмениваться данными с помощью широковещательной передачи. Поэтому мы рассмотрим это в этом вводном разделе.

Bluetooth LE

На следующей схеме показана топология сети для широковещательной сети:



При трансляции определены две роли:

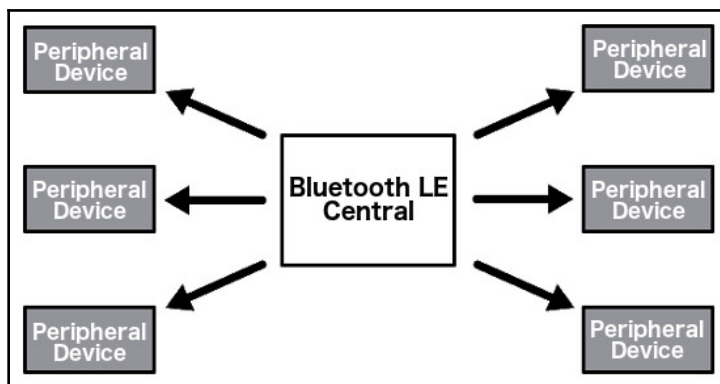
- **Broadcaster**: это устройство отправляет неподключаемые рекламные пакеты через заданные интервалы времени на любое другое устройство, которое прослушивает
- **Observer**: это устройство сканирует рекламные частоты, чтобы получить неподключаемые рекламные пакеты, которые рассылает вещательная компания.

Широковещательная передача данных - единственный способ, которым устройство может отправлять данные на несколько устройств. Стандартный широковещательный пакет может иметь полезную нагрузку из 31 байта данных, которые обычно используются для описания широковещательной передачи и ее возможностей. Однако он также может включать любую пользовательскую информацию, которую мы хотим транслировать на другие устройства. Bluetooth LE также поддерживает дополнительную вторую рекламную полезную нагрузку, называемую **scan response** - откликом на сканирование, которая может включать дополнительный 31 байт данных.

Широковещательная передача выполняется быстро и легко, если мы хотим передавать небольшие объемы данных на несколько устройств; тем не менее, данные не являются безопасными или конфиденциальными. Безопасность обычно является главной причиной избегать использования широковещательных пакетов. Однако еще одна важная причина избегать использования широковещательных пакетов заключается в том, что наблюдатель не имеет возможности отправлять какие-либо данные обратно в широковещательную компанию. Теперь давайте посмотрим на соединения Bluetooth LE.

Bluetooth LE

На следующей схеме показано, как работают соединения Bluetooth:



Как и в случае топологии вещания Bluetooth LE, топология подключения также определяет два элемента:

- **Central:** центральным обычно является такое устройство, как ноутбук, планшет или телефон. Эти устройства будут сканировать рекламные каналы и прослушивать подключаемые рекламные пакеты. Когда устройство найдено, центральная станция может попытаться установить соединение с устройством. После того, как соединение установлено, центральное устройство управляет синхронизацией и инициирует обмен данными. Центральное устройство может подключать более одного периферийного устройства.
- **Peripheral:** периферийным устройством обычно является устройство, такое как умные часы, метеостанция или медицинское устройство. Эти устройства периодически отправляют пакеты с возможностью подключения и принимают входящие соединения. После того, как соединение установлено, периферийное устройство обычно будет следовать синхронизация центра и обмен данными по запросу центра. Периферийное устройство может подключаться только к одному центральному устройству.

Периферийное устройство обычно будет рекламировать, пока центральное устройство не обнаружит его и не запросит соединение. Как только соединение будет установлено, периферийное устройство прекратит рекламу, и тогда два устройства смогут обмениваться данными. Обмен данными в этой топологии может происходить в обоих направлениях, при этом периферийное и центральное устройства могут как отправлять, так и получать данные.

Когда центральное и периферийное устройства устанавливают соединение, передаваемые и получаемые данные объединяются в блоки, называемые **services and characteristics** - услугами и характеристиками. Мы рассмотрим это подробнее, когда рассмотрим **Generic Attribute Profile** - Общий профиль атрибутов (GATT) чуть позже в этой главе. Теперь необходимо понять, что периферийное устройство Bluetooth LE может иметь несколько характеристик, которые используются для отправки и получения данных. Эти характеристики организованы или сгруппированы в службы.

Самым большим преимуществом установления соединения Bluetooth LE является то, что вы можете иметь несколько характеристик для организации данных, и каждая из этих характеристик может иметь собственные права доступа и описательные метаданные. Еще одно преимущество - возможность устанавливать безопасные зашифрованные соединения.

С Bluetooth 4.0 устройство может действовать как центральное или периферийное, но не как то и другое одновременно. Начиная с Bluetooth 4.1, это ограничение было снято, и в более новых версиях Bluetooth LE устройство может действовать как периферийное, центральное или и то, и другое.

Теперь посмотрим на профиль Bluetooth LE.

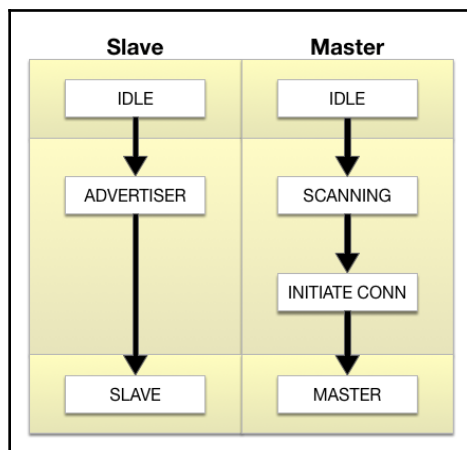
Bluetooth LE

Bluetooth LE определяет два типа профилей. Это профили, которые определяют базовый режим работы, необходимый всем устройствам Bluetooth LE для обеспечения взаимодействия (Generic Access Profile и GATT), или профили, которые используются для конкретных случаев использования (профиль устройства работоспособности и профиль близости). В этой главе мы не будем вдаваться в конкретные варианты использования этих профилей; тем не менее, мы рассмотрим как Generic Access Profile (GAP), так и Generic Attribute Profile (GATT). Мы начнем с рассмотрения GAP.

Generic access profile (GAP) -

GAP определяет, как устройства взаимодействуют друг с другом, чтобы гарантировать совместимость устройств. Он определяет, как устройства Bluetooth LE обнаруживают друг друга, устанавливают безопасные соединения, завершают соединения, транслируют данные и конфигурируют устройства. Это самый низкий уровень стека Bluetooth LE, который мы рассмотрим в этой главе.

Ранее в этой главе мы видели, что устройство Bluetooth LE может находиться в одном из двух состояний. В топологии вещания устройство может быть либо вещателем (подчиненным), либо наблюдателем (ведущим). Если соединение между двумя устройствами установлено, они становятся либо центральными (ведущими), либо периферийными (ведомыми). Мы ввели здесь термины ведущий и ведомый, чтобы проиллюстрировать состояния, в которых могут находиться устройства. На следующем рисунке показаны различные состояния:



Оба типа устройств запускаются в состоянии простоя или ожидания. Это начальное состояние при сбросе устройства. Подчиненное устройство затем станет рекламодателем, где оно рекламирует определенные данные, позволяя любому главному устройству знать, что это подключаемое устройство и какие услуги оно предлагает. После состояния ожидания ведущее устройство начнет сканирование ведомых устройств, которые передают рекламу. Когда ведущее устройство получает рекламу, оно отправляет рекламодателю запрос на сканирование, а ведомое устройство отвечает ответом на сканирование. Это процесс обнаружения устройства.

После процесса обнаружения устройства, если мастер желает подключиться к рекламному устройству, он инициирует соединение. При инициировании соединения мастер укажет параметры соединения. Как только соединение будет установлено, устройства примут на себя свои роли как ведущие и ведомые. С Bluetooth LE 4.0 ведомое устройство может иметь только одно главное устройство. Кроме того, с Bluetooth LE 4.0 устройства могут действовать как ведущие или ведомые, но не оба одновременно. В более поздних версиях спецификаций Bluetooth эти ограничения были сняты. Я знаю, что мы уже несколько раз упоминали об этом в этой книге, но об этом важно помнить при разработке устройств.

Мы упоминали, что мастер указывает количество параметров соединения при инициации соединения. Вот некоторые из этих параметров:

- **Connection Interval** - Интервал соединения: с Bluetooth LE использует схему скачкообразной перестройки частоты, два устройства, которые обмениваются данными, знают, какой канал для передачи / приема включен, когда переключать каналы и когда устанавливать соединение. Интервал времени между попытками соединения известен как интервал соединения.
- **Slave Latency**: Задержка ведомого дает ведомому устройству возможность пропустить определенное количество событий соединения. Подчиненное устройство должно пропускать больше, чем количество событий соединения, определенное этим параметром.
- **Supervision Time-out**: Тайм-аут супервизии - это максимальное время между двумя успешными событиями соединения. Если это время будет превышено, устройство завершит соединение, а ведомое устройство вернется в неподключенное состояние.

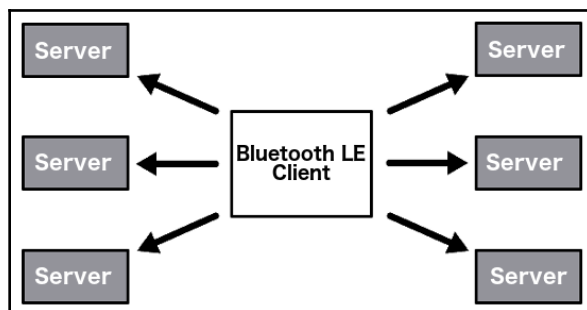
При принятии решения о настройке этих параметров необходимо учитывать ряд факторов. Основное внимание уделяется энергопотреблению и скорости передачи данных. По мере увеличения пропускной способности устройство будет потреблять больше энергии. Например, если мы уменьшим интервал подключения, тем самым увеличив количество попыток подключения в секунду, энергопотребление устройства увеличится, потому что радио будет работать большую часть времени. Уменьшая задержку ведомого, радио снова станет активнее. Следовательно, это также увеличит потребление энергии. При работе с радиомодулем Bluetooth LE вам необходимо найти баланс между потребляемой мощностью и пропускной способностью, необходимой для вашего проекта. Не существует магического соотношения, подходящего для всех типов устройств; это то, что вам нужно будет рассмотреть для каждого проекта.

В примерах проектов для этой главы мы покажем, как с помощью AT-команд настраивать различные параметры модуля Bluetooth LE. Теперь посмотрим на профиль GATT.

(GATT)

В то время как профиль GAP определяет низкоуровневые взаимодействия устройств Bluetooth LE (реклама и соединение), профиль GATT определяет детали того, как устройства обмениваются данными. GATT также является эталонной структурой для всех профилей на основе атрибутов, которые определяют конкретные варианты использования, такие как профили частоты сердечных сокращений и артериального давления.

Как и профиль GAP, профиль GATT определяет две роли. Эти роли - клиент и сервер. Когда вы посмотрите на схему, показывающую, как это работает, роли сначала могут показаться немного странными; однако, как только мы увидим, как устройства Bluetooth LE обмениваются данными, это станет более понятным. Роль клиента в профиле GATT соответствует главной роли в профиле GAP, а роль сервера в профиле GATT соответствует подчиненной роли. Следующая диаграмма иллюстрирует это:



На этой диаграмме мы видим, что у одного клиента может быть несколько серверов; однако у каждого сервера может быть только один клиент. В профиле GATT клиент (центральная роль в GAP) запрашивает информацию у сервера (периферийное устройство в GAP). Хотя мы показываем взаимосвязь между ролями GATT и GAP, стоит отметить, что роли GATT и GAP фактически независимы друг от друга, и в более поздних версиях спецификаций Bluetooth LE устройство может действовать как центральное, так и периферийное устройство.

Наименьший объект данных, определенный профилем GATT, является атрибутом. Атрибут - это адресуемая часть информации, расположенная на сервере, которая может быть доступна и может быть изменена клиентом. Каждый атрибут уникально идентифицируется UUID (универсальный уникальный идентификатор), который может быть 16-битным или 128-битным числом. Этот идентификатор известен как дескриптор.

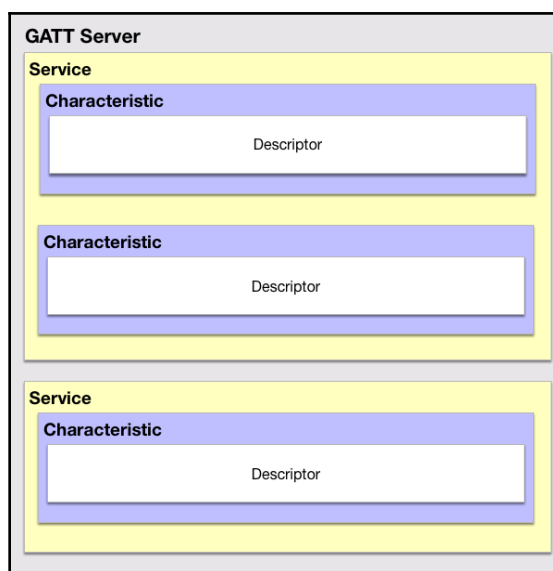
Профиль GATT определяет набор разрешений, связанных со всеми атрибутами. Разрешения определяют, какие операции могут выполняться с каждым атрибутом. Разрешения:

- **Access Permissions:** разрешения доступа определяют, какие действия могут быть выполнены с атрибутом. Каждый атрибут будет иметь одно из следующих разрешений:
 - **None:** атрибут не может быть прочитан или записан клиентом.
 - **Readable:** атрибут может быть прочитан клиентом
 - **Writable** (Возможность записи): атрибут может быть записан клиентом.
 - **Readable and Writable** (Доступно для чтения и записи): атрибут может быть прочитан и записан клиентом.

- **Encryption** (Шифрование): разрешение на шифрование определяет уровень шифрования, необходимый клиенту для доступа к атрибуту.
 - **No Encryption** (режим безопасности 1, уровень 1): шифрование не требуется
 - **Unauthenticated Encryption** (режим безопасности 1, уровень 2): соединение должно быть зашифровано; однако ключи шифрования не нуждаются в аутентификации
 - **Authenticated Encryption** (режим безопасности 2, уровень 2): соединение должно быть зашифровано, а ключи шифрования должны быть аутентифицированы.
- **Authorization**: разрешение авторизации определяет, должен ли пользователь быть авторизован для доступа к атрибуту.
 - **No Authorization**: для доступа к атрибуту авторизация не требуется.
 - **Authorization Required**: для доступа к атрибуту требуется авторизация.

GATT определяет строгую иерархию, которая упорядочивает атрибуты. Атрибуты сгруппированы в службы, каждая из которых может содержать ноль или более характеристик. Эти характеристики могут включать ноль или более дескрипторов. Услуги, характеристики и дескрипторы - все это атрибуты на сервере GATT.

На следующей рисунке показана иерархия:



Службы используются для группировки связанных атрибутов в общую сущность. Каждая служба идентифицируется уникальным UUID, который может быть либо 16-битным для официально принятых типов сервисов, либо 128-битным для пользовательских типов сервисов.



Вы можете увидеть список официально принятых сервисов на сайте Bluetooth SIG здесь:

<https://www.bluetooth.com/api/silentlogin/login?return=http%3a%2f%2fwww.bluetooth.com%2fspecifications%2fgatt%2fservices>.

Если вы посмотрите на сервис Heart Rate, вы увидите, что этот сервис содержит три характеристики.

Характеристики - это контейнеры для данных, где каждая характеристика инкапсулирует отдельную точку данных. Как и в случае со службами, характеристика идентифицируется 16-битным или 128-битным UUID.

Характеристики - это основная точка входа, с которой клиент Bluetooth LE взаимодействует с сервером.



Вы можете найти список официально принятых характеристик на сайте Bluetooth SIG здесь:

<https://www.bluetooth.com/specifications/gatt/characteristics>.

Разрешения на доступ для каждой характеристики должны быть доступны только для чтения или только для записи. Очень редко характеристика имеет разрешение на чтение и запись. В качестве примера, если бы мы хотели создать простой последовательный интерфейс для нашего устройства Bluetooth LE, мы бы создали характеристику TX для передачи данных с разрешением только для чтения для клиента и характеристику RX для получения данных с разрешением только на запись для клиента. Мы не хотели бы создавать единую характеристику, у которой были бы права на чтение и запись, потому что, когда клиент записывает в нее данные, сервер может их перезаписать.

Дескрипторы используются для предоставления клиентским устройствам дополнительной информации о характеристиках и их значениях.



Вы можете найти список официально принятых дескрипторов на сайте Bluetooth SIG здесь:

<https://www.bluetooth.com/specifications/gatt/descriptors>.

Обычно сервер просто отвечает на запрос клиента о данных из характеристики; однако сервер может инициировать обмен данными с помощью обновлений, инициированных сервером. Существует два типа обновлений, инициируемых сервером, а именно:

- : Уведомления об изменении значения характеристики используются, когда сервер настроен для уведомления клиента об изменении значения характеристики, но не ожидает, что клиент подтвердит уведомление. Уведомление включено для всех проектов в этой главе; однако оно используется только в первом и третьем проектах.
- : Индикация изменения значения характеристики используется, когда сервер настроен для указания клиенту, что значение характеристики изменилось, и ожидает, что клиент подтвердит, что он получил индикацию.

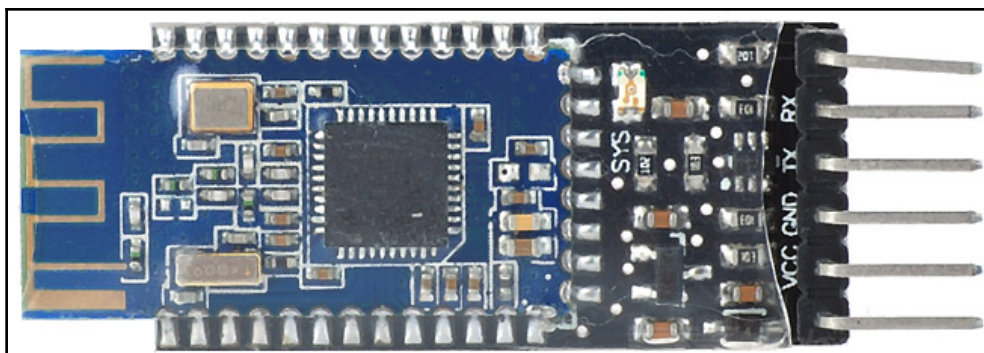
Теперь, когда у нас есть базовое представление о Bluetooth LE и о том, как он работает, давайте рассмотрим модуль Bluetooth HM-10, который мы будем использовать в этой главе.

Bluetooth HM-10

HM-10 - это модуль Bluetooth 4.0, основанный на TI CC2530 или CC2541 Bluetooth **SOC** (**System-on-Chip**). HM-10 - очень популярный модуль Bluetooth 4 для Arduino, в основном из-за его низкой стоимости и простоты использования. HM-10 обеспечивает стандартное последовательное соединение с уровнем Bluetooth.

Это обеспечивает очень простой интерфейс; однако он скрывает фактический уровень Bluetooth LE. В главе 21, Bluetooth Classic, когда мы рассмотрим модуль Bluetooth HC-05, вы заметите, что интерфейс между HC-05 и HM-10 использует один и тот же последовательный интерфейс; однако понимание разницы между технологиями Bluetooth LE и Bluetooth Classic поможет вам решить, что использовать в своем проекте.

Мы можем управлять модулем с помощью AT-команд, и мы рассмотрим, как это сделать, в разделе «Проекты» этой главы. На следующем рисунке показано, как выглядит модуль Bluetooth HM-10:



HM-10 имеет шесть выводов. Однако нас интересуют только средние четыре, а именно:

- **VCC**: подключен к выходу питания 3,3 В на Arduino
- **GND**: подключен к земле на Arduino
- **TX**: вывод передачи, подключенный к одному из цифровых выводов на Arduino.
- **RX**: приемный вывод, подключенный к одному из цифровых выводов на Arduino.

Теперь давайте посмотрим на все компоненты, которые нам понадобятся для проектов, которые мы будем реализовывать в этой главе.

For these, projects you will need the following items:

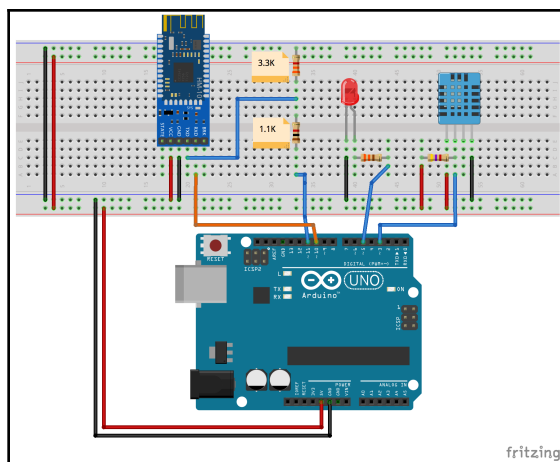
- Arduino Uno или совместимая плата
- Модуль HM-10 Bluetooth 4.0
- Датчик температуры DHT-11
- Один светодиод
- Один резистор 4,7 кОм
- Один резистор 3,3 кОм
- Один резистор 1,1 кОм
- Один резистор 330 кОм
- Перемычки
- Макетная плата

Вам понадобится приложение Bluetooth LE для вашего телефона / планшета или компьютера. Я использую приложение BTCommander - Serial port HM10

(<https://itunes.apple.com/us/app/btcommanderserial-port-hm10/id1312640906?mt=8>) на своем телефоне. Существует множество других приложений, например приложение nRF connect для Android (https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcpnl=en_US) и

iOS (<https://itunes.apple.com/us/app/nrf-connect/id1054362403?mt=8>).

В этой главе мы будем выполнять три проекта. Первым проектом будет простой проект последовательной связи, который будет отправлять текст на Arduino и обратно через радиомодуль Bluetooth. Мы также покажем, как настроить радио Bluetooth в первом проекте. Во втором проекте мы покажем, как дистанционно включать и выключать светодиод. Для финального проекта мы построим мини-метеостанцию, которая позволит нам удаленно считывать температуру через Bluetooth-радио. Каждый проект будет иметь свою собственную схему подключения, включенную в него; однако, если вы хотите подключить все оборудование одновременно, на следующих схемах показано, как все подключено:



Эта схема подключения может изначально выглядеть сложной по сравнению с предыдущими схемами; однако, если мы разделим ее на три части, это действительно не так сложно. Первая часть - это датчик температуры DHT-11, который расположен с правой стороны макета. Вторая часть - это светодиод, который находится в центре макетной платы. Третья и последняя часть - это модуль Bluetooth HM-10, который расположен на левой стороне макета.

Мы уже рассмотрели подключение датчика температуры DHT-11 в главе 9 «Датчики окружающей среды» и светодиода в главе 4, «Базовое прототипирование». Поэтому мы повторим объяснение здесь.

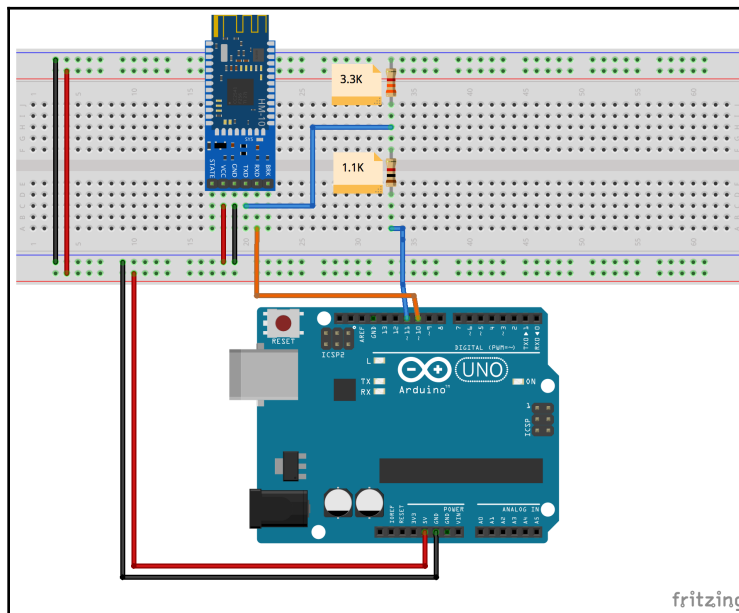
Чтобы подключить модуль Bluetooth HM-10 к Arduino, вывод VCC подключается к шине питания на макетной плате, которая подключена к выходу питания 5 В на Arduino. Контакт GND на модуле Bluetooth подключен к шине заземления на макетной плате, которая подключена к контакту заземления на Arduino. Вывод **RX** на модуле Bluetooth подключается непосредственно к цифровому выводу **10** на Arduino.

Подключение вывода **TX** на модуле Bluetooth к Arduino немного отличается. Для этого мы хотим убедиться, что напряжение не превышает 3,3 В. Поэтому мы используем простой делитель напряжения. Делитель напряжения - это простая схема, которая преобразует большее напряжение в меньшее. Для этого мы используем два резистора: 1,1 кОм и 3,3 кОм. Эти два резистора подключены последовательно, где один конец резистора 3,3 кОм подключен к земле, а один конец резистора 1,1 кОм подключен к цифровому выводу 11 на Arduino. Вывод TX подключен между двумя резисторами.

1 -

В этом первом проекте мы будем использовать только модуль Bluetooth HM-10 и Arduino. Вам нужно будет подключить модуль Bluetooth к Arduino, как показано на предыдущей принципиальной схеме.

Следующая рисунки показывает это:



Теперь нам нужно написать код для доступа к модулю Bluetooth. Мы будем использовать библиотеку SoftwareSerial для взаимодействия с модулем HM-10 Bluetooth LE. Эта библиотека была разработана для обеспечения последовательной связи на цифровых выводах, отличных от выводов 0 и 1. Если вы используете плату, отличную от Uno, для этой библиотеки могут быть ограничения. Вы можете обратиться к документации (<https://www.arduino.cc/en/Reference/softwareSerial>), чтобы узнать, есть ли у вашей платы какие-либо ограничения.

Код нужно будет начать с включения файла заголовка SoftwareSerial, а затем инициировать экземпляр типа SoftwareSerial. Мы также хотим добавить новую строку, когда от монитора последовательного порта выдается новая команда. Поэтому мы также определим логическую переменную, которая будет иметь значение true (истина) всякий раз, когда поступает новая команда (это упростит чтение ответов в мониторе последовательного порта). Следующий код сделает это:

```
#include <SoftwareSerial.h>
SoftwareSerial HM10(10, 11); // RX | TX
bool addNewLine = false;
```

При создании экземпляра типа SoftwareSerial необходимо определить, какие контакты использовать для приема (**RX**) и передачи (**TX**) данных. Первое значение - это вывод RX, а второе значение - вывод TX.

Затем нам нужно инициализировать как последовательный монитор, так и экземпляр SoftwareSerial. Мы сделаем это с помощью функции setup (). Мы также хотим, чтобы пользователь знал, когда приложение готово принимать команды или соединения. В следующем коде показан код функции setup ():

```
void setup()
{
  Serial.begin(9600);
  HM10.begin(9600);
  Serial.println("Connected to HM-10. Try connecting from any device or
  issue AT commands");
}
```

Когда мы запускаем последовательный монитор и интерфейс SoftwareSerial, нам нужно определить, какая будет скорость передачи данных. И модуль Bluetooth HM-10, и монитор последовательного порта обмениваются данными со скоростью 9600 бод. После того, как все запущено, на монитор последовательного порта выводится сообщение, позволяющее пользователю узнать, что все в порядке.

В функции loop () нам нужно будет записать все, что пользователь вводит в последовательный монитор, в модуль Bluetooth и записать все, что поступает из модуля Bluetooth в последовательный монитор. В следующем коде показана функция loop ():

```
void loop()
{
  if (Serial.available()) {
    HM10.write(Serial.read());
    addNewLine = true;
  }

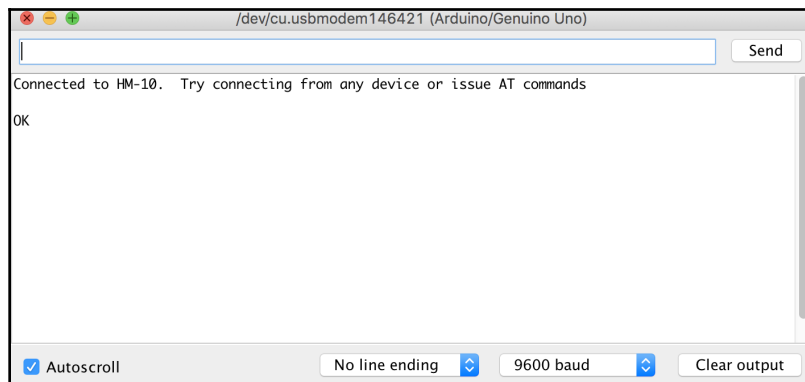
  if (HM10.available()) {
    if (addNewLine) {
```

```
    Serial.write("\r\n");  
    addNewLine = false;  
  }  
  Serial.write(HM10.read());  
}  
}
```

В этой функции мы используем функцию `available ()` как на последовательном мониторе, так и на экземпляре типа `SoftwareSerial`, чтобы проверить, есть ли на каком-либо устройстве данные для чтения. Если да, мы читаем данные и записываем их на другое устройство. В разделе, который читает последовательные мониторы и записывает в экземпляр `SoftwareSerial`, мы устанавливаем логическую переменную `addNewLine` в значение `true` - истина, поэтому в следующий раз, когда мы будем писать в последовательный монитор, мы напишем возврат каретки и новую строку. В разделе, который читает из модуля Bluetooth и записывает в последовательный монитор, мы проверяем, является ли логическая переменная `addNewLine` истинной, и если да, мы записываем возврат каретки и новую строку в последовательный монитор перед установкой переменной `addNewLine` в `false` - ложь.

Есть два способа использования этого приложения. Первый - ввести команды AT (ATtention) в монитор последовательного порта, что позволит вам получить / установить параметры конфигурации на модуле Bluetooth, а также управлять модулем. Второй - использовать приложение Bluetooth LE на вашем телефоне для чтения и записи значений в модули Bluetooth. Давайте сначала посмотрим на команды AT.

Чтобы отправить AT-команду на модуль Bluetooth LE, запустите предыдущие блоки кода, а затем откройте монитор последовательного порта, который является частью Arduino IDE. Как только все будет запущено, вы увидите `Connected` - Подключено к HM-10. Попробуйте подключиться с любого устройства или выдайте сообщение AT-команд, отображаемое на мониторе. Это означает, что модуль готов и все запущено. Как только вы увидите сообщение, введите в поле ввода и нажмите кнопку « Send - Отправить » или нажмите клавишу « Enter - Ввод ». Вы должны увидеть ответ OK от модуля Bluetooth. Результат должен выглядеть так:



Чтобы отправить AT-команду, вы должны использовать следующий формат:

```
Set item: AT+{command}{new setting}
Query item: AT+{command}?
```

Чтобы установить элемент, вы вводите буквы AT, за которыми следует знак плюса (+), команду и новую настройку без пробелов. Например, чтобы установить имя, которое модуль Bluetooth будет рекламировать как «Buddy - Приятель», мы должны выполнить следующую команду:

```
at+nameBuddy
```



Примечание: AT-команды нечувствительны к регистру.

Чтобы запросить элемент, мы должны ввести буквы AT, за которыми следует знак плюса (+), команда, а затем вопросительный знак (?). Например, чтобы запросить имя, которое рекламирует модуль Bluetooth, мы могли бы использовать следующую команду:

```
at+name?
```

Мы можем использовать приложение, которое мы только что написали, чтобы установить конфигурацию вручную из пробного монитора, или мы можем установить конфигурации из приложения, используя функцию `print()` из библиотеки `SoftwareSerial` следующим образом:

```
HM10.print("AT+Name?\r\n");
```

Давайте посмотрим на некоторые из наиболее часто используемых команд. Любую из них можно использовать из пробного монитора или в коде, как мы только что показали.

Команда	Ответ	Параметры	Описание
AT	OK	None	Тестовая команда для проверки соединения с модулем Bluetooth

Команда	Ответ	Параметры	Описание
AT+VERR	Номер версии	None	Эта команда вернет номер версии микропрограммы для модуля.

Команда	Ответ	Параметры	Описание
AT+RENEW	OK+RENEW	None	Восс. заводские настройки по умолчанию.

Команда	Ответ	Параметры	Описание
AT+RESET	OK+RESET	None	Перезагрузить модуль Bluetooth.

MAC- ()

Команда	Ответ	Параметры	Описание
AT+ADDR?	OK+ADDR:{MAC Address}	None	Команда для запроса MAC-адрес Bluetooth-радио

Команда	Ответ	Параметры	Описание
AT+NAME{parameter}	OK+set{parameter}	None	Эта команда задает имя для модуля.

Команда	Ответ	Параметры	Описание
AT+NAME?	OK+NAME{parameter}	None	Эта команда вернет имя модуля.

Установите рекламный интервал

Команда	Ответ	Параметры	Описание
AT+ADVI{parameter}	OK:Set:{parameter}	0: 100ms 1: 152.5 ms 2: 211.25 ms 3: 318.75 ms 4: 417.5 ms 5: 546.25 ms 6: 760 ms 7: 852.5 ms 8: 1022.5 ms 9: 1285 ms A: 2000 ms B: 3000 ms C: 4000 ms D: 5000 ms E: 6000 ms F: 7000 ms	Интервал рекламы для модуля Bluetooth LE. Параметр должен быть 0-F

Команда	Ответ	Параметры	Описание
AT+ADVI?	OK+get:{parameter}	None	Эта команда вернет текущую рекламную интервал и вернет параметр 0-f.

Команда	Ответ	Параметры	Описание
AT+ADTY{parameter}	OK+set:{parameter}	0: реклама ScanResponse, возможность подключения 1: только последнее устройство подключается через 1,28 секунды 2. Разрешить только рекламу и ScanResponse 3. Разрешить только рекламу	Эта команда устанавливает тип рекламы.

Команда	Ответ	Параметры	Описание
AT+ADTY?	OK+get:{parameter}	None	Эта команда получит текущий тип рекламы и вернет параметр 0–3.

Команда	Ответ	Параметры	Описание
AT+BAUD{parameter}	OK+set:{parameter}	0: 9600 1: 19200 2: 38400 3: 57600 4: 115200 5: 4800 6: 2400 7: 1200 8: 230400	Эта команда устанавливает скорость передачи данных для последовательного интерфейса модуля Bluetooth.

Команда	Ответ	Параметры	Описание
AT+BAUD?	OK+get:{parameter}	None	Эта команда получит текущую скорость передачи и вернет параметр 0-8.

(ID)

Команда	Ответ	Параметры	Описание
AT+CHAR{parameter}	OK+set:{parameter}	0x0001 -> 0xFFFe	команда устанавливает идентификатор характеристики.

(ID)

Команда	Ответ	Параметры	Описание
AT+UUID{parameter}	OK+set:{parameter}	0x0001 -> 0xFFFe	команда установит идентификатор службы

(ID)

Команда	Ответ	Параметры	Описание
AT+UUID?	OK+get:{parameter}	None	Эта команда получит текущий идентификатор службы.

Команда	Ответ	Параметры	Описание
AT+ROLE{parameter}	OK+set:{parameter}	0: Peripheral 1: Central	команда установит роль модуля Bluetooth

Команда	Ответ	Параметры	Описание
AT+ROLE?	OK+get:{parameter}	None	команда вернет роль модуля Bluetooth

Команда	Ответ	Parameters	Описание
AT+CLEAR	OK+CLEAR	None	Удаляет адрес последнего подключенного устройства.



ПРИМЕЧАНИЕ. Команда at + clear используется только тогда, когда устройство находится в центральном режиме.

Команда	Ответ	Параметры	Описание
AT+CONNL	OK+CONN{parameter}	L: Подключение E: Ошибка подкл. F: Ошибка подкл. N: без адреса	Эта команда попытается подключиться к устройству, которое было успешно подключено к нему последним.



ПРИМЕЧАНИЕ. Команда at + connl используется только тогда, когда устройство находится в центральном режиме.

Команда	Ответ	Параметры	Описание
AT+CON{parameter}	OK+CONN{parameter}	A: Подключение E: Ошибка подкл. F: Ошибка подкл.	Эта команда попытается подключиться к устройству с указанным адресом.



ПРИМЕЧАНИЕ. Команда `at + con` используется только тогда, когда устройство находится в центральном режиме.

-

Команда	Ответ	Параметры	Описание
AT+PASS{parameter}	OK+set:{parameter}	000000 -> 999999	Устанавливает PIN-код для подключения.

-

Команда	Ответ	Параметры	Описание
AT+PASS?	OK+get:{parameter}	None	Эта команда вернет текущий пин-код.

Установить мощность модуля

Команда	Ответ	Параметры	Описание
AT+POWE{parameter}	OK+set:{parameter}	0: -23db 1: -6db 2: 0db 3: 6db	Устанавливает мощность модуля.

Команда	Ответ	Параметры	Описание
AT+POWE?	OK+get:{parameter}	None	Эта команда вернет текущую мощность модуля.

Установить режим связи

Команда	Ответ	Параметры	Описание
AT+TYPE{parameter}	OK+set:{parameter}	0: PIN code not needed 1: Auth without PIN code 2: Auth and PIN 3: Auth and bond	Эта команда устанавливает аутентификацию, необходимую при подключении к этому устройству.

Режим связи запроса

Команда	Ответ	Параметры	Описание
AT+TYPE?	OK+get:{parameter}	None	команда вернет текущую аутентификацию необходимо для подключения к этому устройству

Установить информацию для уведомлений

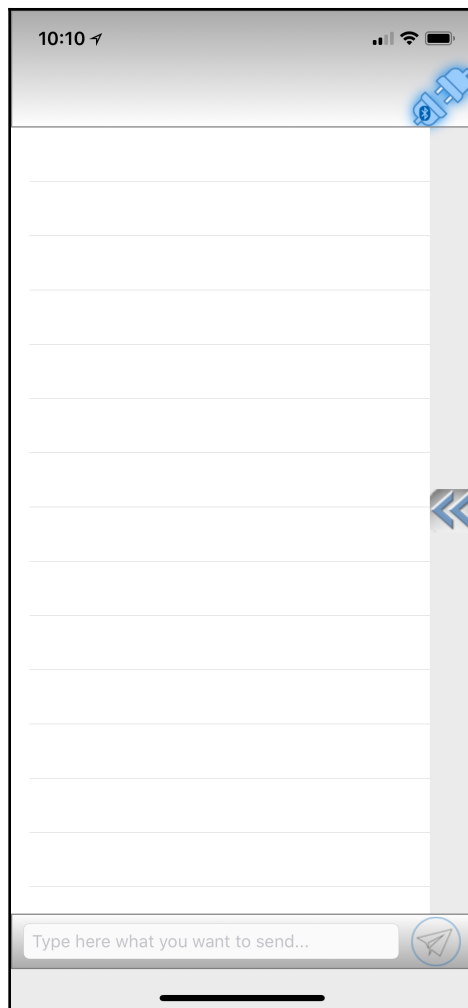
Команда	Ответ	Параметры	Описание
AT+NOTI{parameter}	OK+set:{parameter}	0: Don't Notify 1: Notify	команда включает или отключает уведомление, при подключении и отключении устройства.

Информация для уведомления о запросе

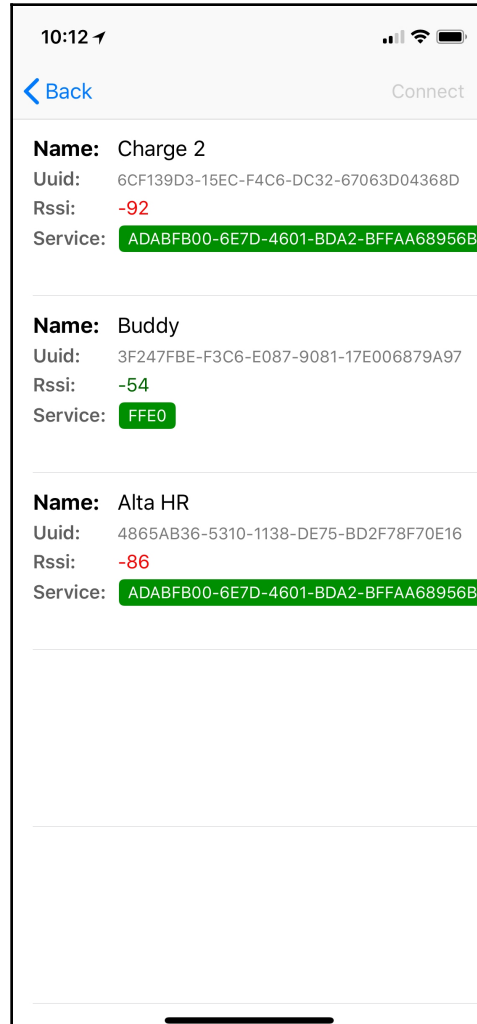
Команда	Ответ	Параметры	Описание
AT+NOTI?	OK+get:{parameter}	None	Эта команда вернется, если устройство отправит уведомление при подключении или отключении устройства.

Мы можем отправлять AT-команды модулю Bluetooth только тогда, когда к нему не подключено другое устройство. Как только устройство подключено к приложению, код, который мы написали, вступает во владение, и данные, которые вводятся в последовательную консоль, отправляются на подключенное устройство. Посмотрим, что происходит, когда мы подключаемся к модулю Bluetooth с другого устройства. Я буду использовать приложение BTCommander - Serial Port HM10, чтобы показать, как это работает.

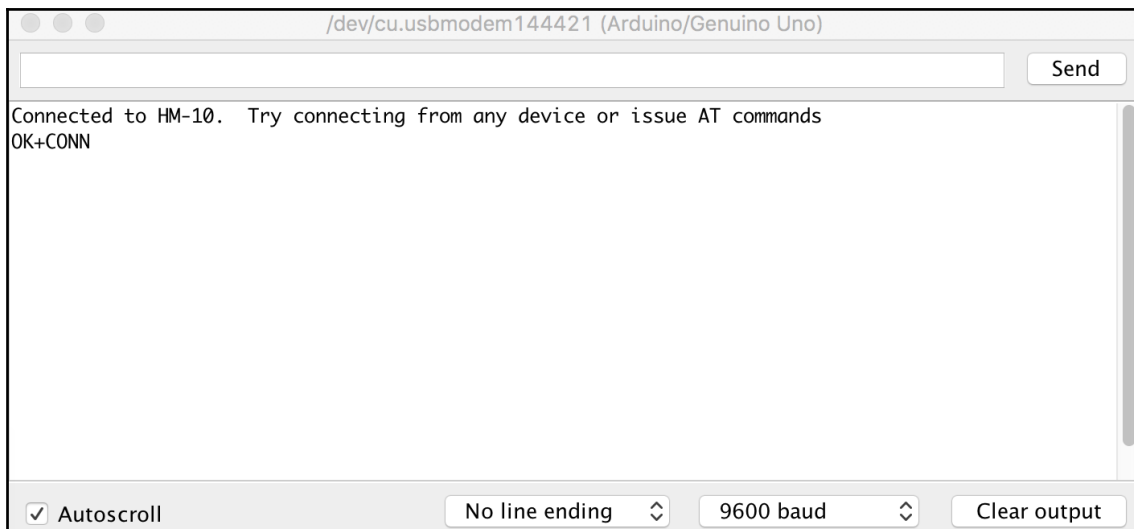
Запустив приложение на Arduino, запустите приложение Bluetooth на своем телефоне / планшете или компьютере. Приложение BTCommander будет выглядеть так:



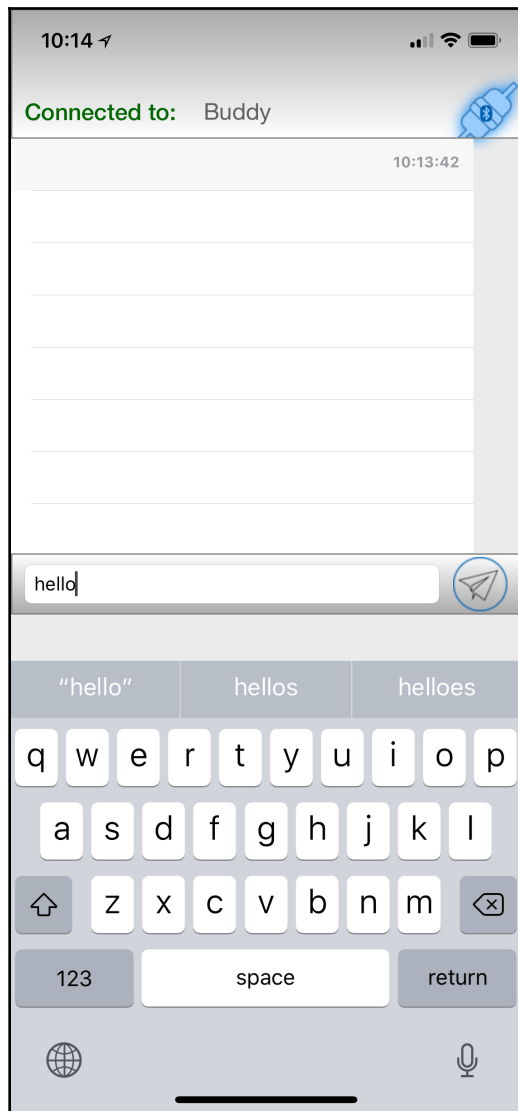
Чтобы подключиться к устройству, нажмите синюю кнопку подключения, которая выглядит как вилка розетки, расположенную в правом верхнем углу приложения. После того, как вы нажмете кнопку, вы должны увидеть список устройств, к которым приложение может подключиться. Этот экран выглядит так:



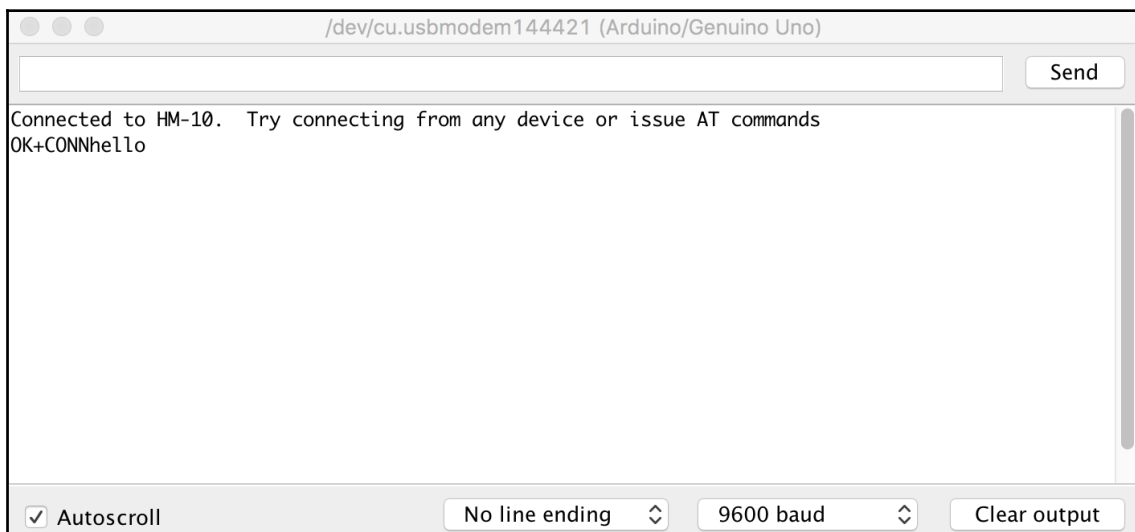
На этом экране показаны все устройства, которые рекламируются и находятся достаточно близко для подключения. Ранее в этой главе, когда мы запускали команду AT + name Buddy, мы переименовали наше устройство в Buddy. Таким образом, мы знаем, что это устройство, к которому мы хотим подключиться. Если мы коснемся этого устройства, а затем нажмем кнопку «Подключить» в правом верхнем углу приложения, приложение попытается подключиться. Если попытка подключения успешна и в настройке AT + NOTI на модуле Bluetooth включены уведомления, мы должны увидеть OK + CONN на последовательной консоли, как показано на следующем снимке экрана:



После успешного подключения приложение вернется на главный экран. Теперь введите сообщение в поле ввода внизу экрана. Например, здесь мы напишем простое приветственное сообщение, как показано на следующем снимке экрана:

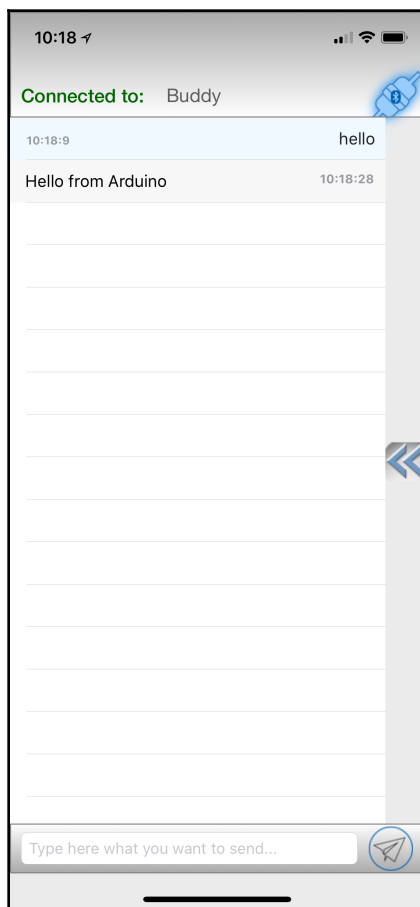


После ввода сообщения нажмите кнопку рядом с полем ввода, которое выглядит как самолет, чтобы отправить сообщение. Если сообщение было успешно отправлено, мы увидим его в пробной консоли, как показано на следующем снимке экрана:



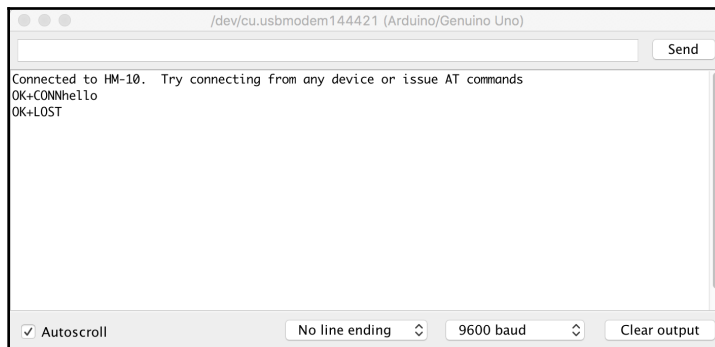
Здесь происходит то, что сообщение, введенное в приложение, передается из приложения телефона в модуль Bluetooth HM-10, поэтому наше приложение может его прочитать. Сообщение отправляется по одному символу за раз.

Чтобы отправить сообщение обратно, введите сообщение в поле ввода последовательной консоли и нажмите кнопку отправки. Если сообщение было отправлено успешно, мы должны увидеть его в приложении, как показано на следующем снимке экрана:



На этом снимке экрана мы видим, что сообщение приветствия было отправлено из приложения, а сообщение Hello от Arduino было получено от подключенного устройства. Когда сообщение отправляется из модуля Bluetooth HM-10 обратно в приложение телефона, приложение записывает сообщение в характеристику (по одному символу за раз), а модуль Bluetooth использует уведомление, чтобы уведомить клиента (приложение телефона) о появлении новых данных. .

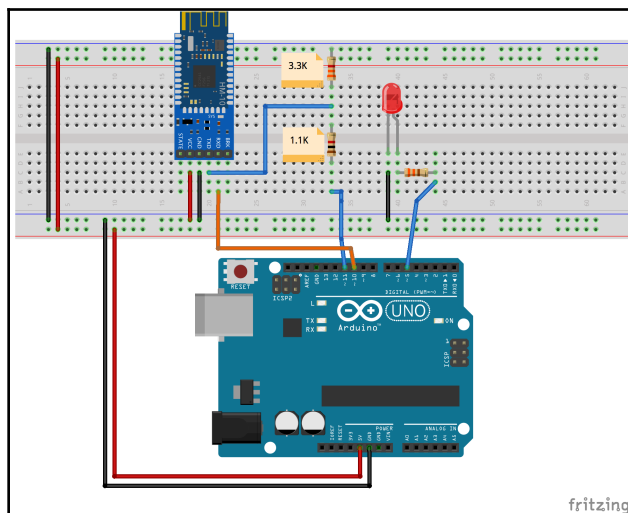
Если мы снова нажмем кнопку подключения в приложении BTCommander для отключения, а конфигурация AT + NOTI настроена на уведомление, мы увидим сообщение OK + LOST в последовательной консоли, как показано на следующем снимке экрана:



Теперь давайте посмотрим, как мы можем использовать модуль Bluetooth для управления светодиодом с нашего телефона.

2 -

В этом проекте мы включим или выключим светодиод, подключенный к Arduino, в зависимости от входа с телефона. Первое, что нам нужно сделать, это добавить светодиод в нашу схему. На следующей схеме показана новая схема:



Светодиод подключен к 5 цифровому выводу на Arduino через резистор на 330 Ом. Теперь нам нужно написать код для управления светодиодом. Мы начнем с настройки библиотеки SoftwareSerial для модуля Bluetooth и определения вывода, к которому подключен светодиод. Следующий код сделает это:

```
#include <SoftwareSerial.h>
#define LED_PIN 5
SoftwareSerial HM10(10, 11);
```

Мы видим, что модуль Bluetooth подключен к тем же выводам, что и в предыдущем примере, а светодиод подключен к цифровому контакту 5 на Arduino. В функции setup () нам нужно будет настроить экземпляр SoftwareSerial и режим вывода, к которому подключен светодиод. В следующем коде показана функция setup () для этого примера:

```
void setup()
{
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  Serial.begin(9600);
  HM10.begin(9600);
  Serial.println("Connected to HM-10");
}
```

Этот код начинается с определения режима вывода, к которому подключен светодиод. Затем он настраивает последовательный порт для последовательного монитора и экземпляра SoftwareSerial. Мы, наконец, распечатываем сообщение в последовательная консоль, позволяющая пользователю узнать, что все настроено и готово к работе.

В нашей функции loop () нам нужно будет проверить экземпляр SoftwareSerial, и если от подключенного устройства будет получена 1, он включит светодиод, а если получен 0, он выключит светодиод. Если не получено ни 1, ни 0, то ввод игнорируется. Вот код функции loop ():

```
void loop()
{
  if (HM10.available()) {
    char val = HM10.read();
    if(val == '1') {
      digitalWrite(LED_PIN, HIGH);
    } else if(val == '0') {
      digitalWrite(LED_PIN, LOW);
    }
  }
}
```

В этом коде мы проверяем, доступно ли значение из модуля Bluetooth, и, если так, мы читаем устройство, чтобы получить символ, который был получен. Если символ равен 1 (символ 1, а не цифра 1), устанавливаем на выводе к которому подключен LED высокий уровень, чтобы включить его. Если символ равен 0 (символ 0, а не цифра 0), устанавливаем на выводе низкий уровень, чтобы выключить светодиод.

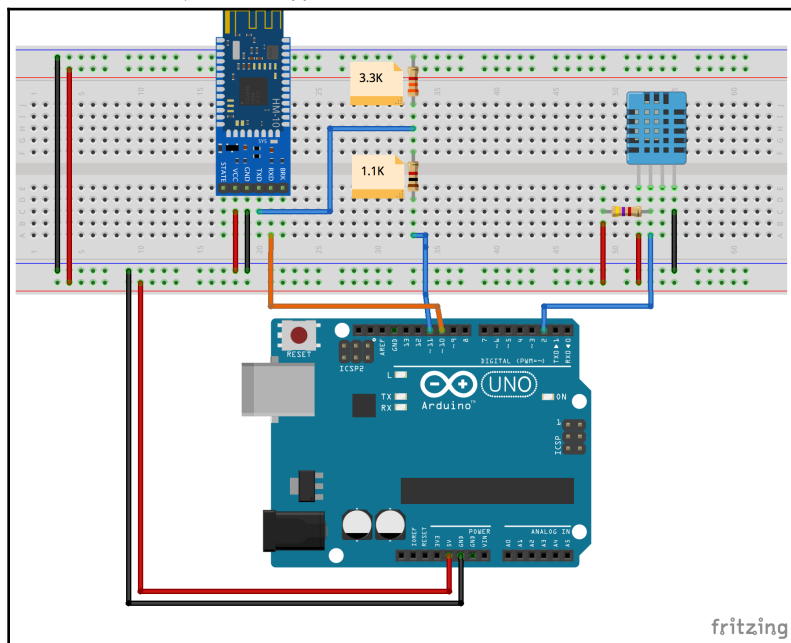
Теперь давайте запустим это приложение и воспользуемся приложением VTCommander для подключения к нему. Из приложения VTCommander, если мы отправим 1, светодиод загорится, или если мы отправим 0, светодиод погаснет. Этот тип примера можно использовать, когда мы хотим иметь телефонное приложение для управления чем-то, что подключено к Arduino, например светодиодом, двигателем постоянного тока или каким-либо датчиком.

Теперь давайте посмотрим, как мы можем получить данные о температуре и влажности с датчика DHT-11 через Bluetooth LE.

3 -

В этом проекте мы будем запрашивать по телефону, чтобы Arduino отправлял информацию о температуре или влажности в зависимости от отправленного символа. Нам нужно будет добавить в нашу схему датчик DHT-11.

На следующей схеме показано, как это сделать:



Датчик температуры DHT-11 подключается к Arduino точно так же, как мы делали это в главе 9 «Датчики окружающей среды». Если вы не знаете, как подключить этот датчик к Arduino, вернитесь к этой главе. Теперь нам нужно будет написать код, чтобы мы могли получить доступ к данным датчика с помощью службы Bluetooth LE. Начнем с настройки библиотеки SoftwareSerial для модуля Bluetooth и датчика температуры DHT-11. Следующий код сделает это:

```
#include <DHT.h>
#include <SoftwareSerial.h>

#define DHT_PIN 3
#define DHT_TYPE DHT11
DHT dht(DHT_PIN, DHT_TYPE);

SoftwareSerial HM10(10, 11); // RX | TX
```

Этот код включает библиотеки для датчика температуры DHT, а также устройство SoftwareSerial. Затем он определяет контакт, к которому подключен датчик температуры, и тип датчика. Наконец, он создает экземпляры типов DHT и SoftwareSerial.

В функции setup () нам нужно будет настроить экземпляр SoftwareSerial и режим вывода, к которому подключен светодиод. В следующем коде показана функция setup () для нашего примера:

```
void setup()
{
  Serial.begin(9600);
  HM10.begin(9600);
  Serial.println("Connected to HM-10");
}
```

Эта функция настраивает экземпляр SoftwareSerial и выводит сообщение на последовательную консоль, когда все запущено и готово к работе. В функции loop () будет считан ввод от устройства, подключенного к службе, а затем дан ответ соответствующей информацией.

Следующая таблица показывает ввод и, что нужно вернуть:

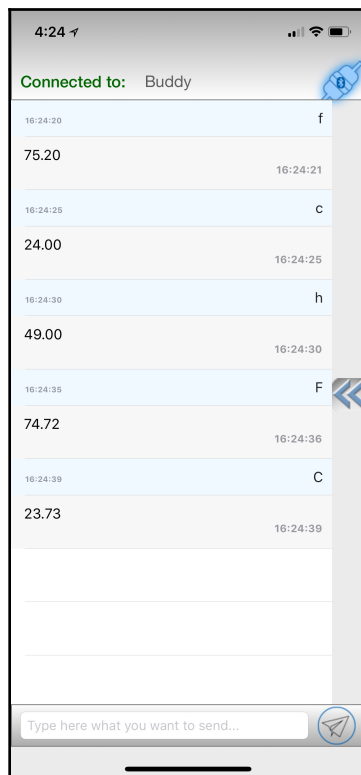
Input - вход	Возвращенное свойство
f	Температура в градусах Фаренгейта
c	Температура в градусах Цельсия
h	Влажность
F	Индекс тепла по Фаренгейту
C	Индекс тепла в градусах Цельсия

Из таблицы мы видим, что у нас будет пять входов, и каждый будет возвращать различную информацию обратно на удаленное устройство. Давайте посмотрим на код, который прочитает ввод и вернет запрошенную информацию:

```
void loop()
{
  if (HM10.available()) {
    char val = HM10.read();
    if(val == 'f') {
      float fahrenheit = dht.readTemperature(true);
      HM10.println(fahrenheit);
    } else if(val == 'c') {
      float celsius = dht.readTemperature();
      HM10.println(celsius);
    } else if(val == 'h') {
      float humidity = dht.readHumidity();
      HM10.println(humidity);
    } else if(val == 'F') {
      float fahrenheit = dht.readTemperature(true);
      float humidity = dht.readHumidity();
      float hif = dht.computeHeatIndex(fahrenheit, humidity);
      HM10.println(hif);
    } else if(val == 'C') {
      float celsius = dht.readTemperature();
      float humidity = dht.readHumidity();
      float hic = dht.computeHeatIndex(celsius, humidity, false);
      HM10.println(hic);
    }
  }
}
```

Этот код начинается с проверки адаптера Bluetooth, чтобы увидеть, есть ли какие-либо входные данные, и если да, он считывает символ. Если входной символ является одним из символов, перечисленных в предыдущей таблице, код получает соответствующее значение от датчика DHT-11 и возвращает значение обратно на подключенное устройство.

Вывод кода в приложении BTCommander для iOS будет выглядеть примерно так:



Все, что мы здесь показали, совместимо с Bluetooth LE 4.0 и выше. На момент написания этой книги действительно не так много недорогих модулей Bluetooth LE для Arduino, совместимых с новыми стандартами Bluetooth LE 4.2 и 5.0, поэтому мы остановились на 4.0. Хорошая новость заключается в том, что все новые стандарты обратно совместимы со стандартом 4.0, поэтому все, о чем мы говорили в этой главе, будет работать по мере выпуска новых модулей Bluetooth, совместимых с новыми стандартами. Давайте посмотрим, какие функции доступны в новом Bluetooth. стандарты.

Bluetooth 4.1, 4.2 5.0?

В этом разделе мы рассмотрим, что нового в Bluetooth 4.1, 4.2 и 5.0. Хотя эти функции несовместимы с модулем Bluetooth LE 4.0, который мы использовали в этой главе, со временем будут выпущены модули Bluetooth для Arduino, совместимые с этими стандартами; поэтому полезно знать, какие функции они предлагают.

Bluetooth 4.1

Bluetooth 4.1 в основном предлагает обновления для удобства использования. Одно из самых важных обновлений - позволить радиомодулям Bluetooth LE и LTE лучше сосуществовать. Это обновление позволяет радиостанциям координировать передачи, чтобы снизить вероятность помех. Это также делает передачу данных более эффективной и позволяет лучше восстанавливать соединение после потери соединения.

Большое изменение, связанное с отсутствием удобства использования с Bluetooth 4.1, позволило устройству быть одновременно и периферийным, и центральным.

Bluetooth 4.2

Bluetooth 4.2 предлагает множество новых функций для Интернета вещей, безопасности, более высокой скорости и большей емкости.

Для Интернета вещей в Bluetooth 4.2 добавлены Интернет-шлюзы Bluetooth Smart, которые позволяют устройствам Bluetooth 4.2 подключаться к Интернету. Вместе с интернет-шлюзами в Bluetooth 4.2 также добавлен протокол IPv6 / 6LoWPAN, который обеспечивает поддержку IPv6 через соединение Bluetooth.

Bluetooth 4.2 также добавил дополнительную безопасность с LE Privacy 1.2. Стандарты шифрования с Bluetooth 4.2 соответствуют Федеральным стандартам обработки информации (FIPS), который является стандартом компьютерной безопасности правительства США.

Размер пакета передачи также был увеличен в десять раз. Это обеспечивает более быструю и надежную передачу данных.

Bluetooth 5.0

Bluetooth предлагает ряд усовершенствований, в четыре раза увеличивая диапазон и удваивая скорость. Уловка в том, что если производитель устройства увеличивает радиус действия своего устройства, скорость снижается; и аналогично, если скорость увеличивается, дальность действия уменьшается.

Bluetooth

Одна из последних и, на мой взгляд, самых захватывающих технологий Bluetooth - это сетка Bluetooth. Все предыдущие технологии Bluetooth основывались на подключении «один-к-одному» или «один-ко-многим», где всегда было одно главное / центральное устройство. Технология Mesh позволяет устройствам Bluetooth устанавливать соединения "многие ко многим", что позволяет создавать крупномасштабные сети устройств, не зависящие от центрального контроллера.

На момент написания этой книги ячеистая технология Bluetooth все еще находится в зачаточном состоянии. Тем не менее, я считаю, что это будущее Bluetooth, и это технология, на которую стоит обратить внимание.

Следует отметить, что со стандартами Bluetooth выше 4.0 большинство новых функций являются необязательными и не требуют полной реализации. Например, производитель может сказать, что его устройство соответствует требованиям

Стандарт Bluetooth 4.2. Однако IPv6 / 6LoWPAN не может быть реализован в устройстве.

Хороший тому пример - iPhone. Мой iPhone X поддерживает Bluetooth 5.0; однако он не может работать с сеткой Bluetooth или IPv6 / 6LoWPAN.

Для задания возьмите ЖК-дисплей Nokia 5110, который мы использовали в главе 13 «Использование ЖК-дисплеев», и код последовательной связи, который мы использовали в первом проекте этой главы, и распечатайте любое сообщение, отправленное из приложения для телефона на ЖК-дисплей. Это потребует некоторой модификации кода в этой главе, чтобы использовать ЖК-экран, а не последовательную консоль.

В этой главе мы многое узнали о Bluetooth LE, начав с краткого введения о том, как работает радио, и о топологии сети для соединений Bluetooth LE. Мы узнали, как GAP использует Устройства Bluetooth LE для обнаружения и подключения к другим устройствам. Мы также видели, как GATT использует атрибуты (службы, характеристики и дескрипторы), чтобы два устройства Bluetooth LE могли взаимодействовать друг с другом. В конце этой главы мы наконец продемонстрировали, как Bluetooth LE работает с тремя проектами.

Bluetooth LE - это технология, которая лучше всего подходит, когда мы хотим использовать внешнее устройство, например телефон, для управления устройством, которое мы создаем, потому что почти все смартфоны имеют встроенный Bluetooth LE, и эта технология проста в использовании. Это также хорошая технология, когда мы хотим отправлять короткие пакеты данных с одного устройства на другое. Если вы хотите создать отдельное устройство, например пульт дистанционного управления, для управления вашим основным устройством или потоковой передачи большого количества данных, я бы порекомендовал другую технологию Bluetooth, известную как Bluetooth Classic или Legacy, которую мы увидим в следующей главе. .

Классический Bluetooth

Bluetooth LE, который мы рассматривали в главе 20, является отличным выбором, когда два устройства должны обмениваться данными по беспроводной сети короткими пакетами данных и когда потребление энергии является проблемой. В Bluetooth LE версии 4.2 и 5.0 произошли изменения. Это делает его более привлекательным для устройств, которым необходимо передавать большие объемы данных или даже передавать их в потоке. Тем не менее, есть еще одна технология Bluetooth, которая очень успешно работает в течение многих лет, эта технология известна как Bluetooth Classic. Хотя название может означать, что эта технология устарела, не позволяйте названию вводить вас в заблуждение, так как Bluetooth Classic используется во многих устройствах Bluetooth, и до тех пор, пока для Arduino не появится больше модулей Bluetooth 5.0, которые также поддерживают некоторые из новых функций, Bluetooth Classic останется отличным выбором, когда нам нужно передать большие объемы данных между двумя устройствами.

В этой главе вы узнаете:

- Что означают номера версий Bluetooth Classic
- Как работает Bluetooth-радио
- Сетевая топология сети Bluetooth
- Как использовать модуль Bluetooth HC-05

Bluetooth - это стандарт беспроводной технологии, который используется двумя устройствами для передачи или приема данных на короткие расстояния с использованием беспроводного соединения 2,4 ГГц. В то время как целью проектирования Bluetooth LE было создание беспроводного протокола с низким энергопотреблением, у Bluetooth Classic были другие цели создания. Bluetooth Classic был создан инженерами компании Ericsson Mobile в Лунде, Швеция, в качестве беспроводной альтернативы последовательным (RS232) кабелям. Это означало, что этот новый протокол потребует для передачи больших объемов данных или даже потоковой передачи данных на короткие расстояния.

Спецификации Bluetooth Classic управляются **Bluetooth Special Interest Group** - специальной группой по интересам Bluetooth (Bluetooth SIG) как часть основных спецификаций Bluetooth. Как мы упоминали в главе 20, Bluetooth LE, вы можете найти информацию как о Bluetooth LE, так и о Bluetooth Classic, загрузив форму спецификаций с сайта Bluetooth SIG по адресу <https://www.bluetooth.com>.

Поначалу может показаться странным рассматривать новую технологию (Bluetooth LE 4.0) до использования старой технологии (Bluetooth Classic). Причина, по которой Bluetooth LE был описан в первую очередь, заключается в том, что вы обнаружите, что его уместно использовать в более значительном большинстве проектов, которые вы будете создавать с помощью Arduino, поскольку большинство проектов работают с короткими пакетами данных, что и является приоритетом Bluetooth LE. Bluetooth LE также легче интегрировать со смартфонами, использующими Bluetooth LE, потому что каждая ОС смартфона имеет простой в использовании и хорошо документированный API Bluetooth LE, чего нельзя сказать о Bluetooth Classic. В тех случаях, когда вы хотите передавать данные в потоковом режиме или обмениваться большими объемами данных между двумя пользовательскими устройствами, Bluetooth Classic может быть более подходящим.

Приобретая модуль Bluetooth Classic для своего проекта, у вас будет выбор из трех различных версий Bluetooth. Эти версии:

- **Bluetooth 2.0 + EDR**: основные спецификации для этой версии были выпущены в 2004 году. Это обновление основных спецификаций Bluetooth содержало ряд незначительных улучшений стандарта Bluetooth. Единственным значительным улучшением был EDR (Enhanced Data Rate), который увеличил скорость передачи данных до 3 Мбит / с с 1 Мбит / с. Название стандарта читается как Bluetooth 2.0+ EDR, что означает, что функция EDR не является обязательной. Модуль Bluetooth HC-05, который мы будем использовать в этой главе, - это модуль Bluetooth 2.0, совместимый, что означает, что он не включает функцию EDR. Для подавляющего большинства проектов, которые вы создадите с помощью Arduino, подойдут модули, совместимые с Bluetooth 2.0, и на самом деле они предпочтительны, поскольку мы можем избежать функции безопасного сопряжения, которая была введена с Bluetooth 2.1. Хотя новая функция сопряжения может называться Simple Secure Pairing, она обычно требует участия человека для процесса сопряжения, чего мы, возможно, захотим избежать, поскольку многие проекты Arduino не имеют возможности ввода для этого.
- **Bluetooth 2.1 + EDR**: основные спецификации для этой версии были выпущены в 2007 году. Эта версия основных спецификаций Bluetooth также предлагала ряд улучшений по сравнению с предыдущей версией, при этом основным улучшением было введение **SSP (Simple Secure Pairing)**. SSP переработал процесс сопряжения, сделав его простым и безопасным.

- **Bluetooth 3.0 + HS:** основные спецификации для этой версии были выпущены в 2009 году. HS в названии спецификации означает High Speed. Bluetooth 3.0 + HS может иметь теоретическую скорость передачи данных 24 Мбит / сек, однако данные не передаются через соединение Bluetooth. В высокоскоростном режиме данные фактически передаются через соединение 801.11 (Wi-Fi). Связь Bluetooth используется только для согласования и установления соединения Wi-Fi. Как и в случае со спецификацией Bluetooth 2.X + EDR, функция HS является дополнительной, и вы увидите устройства, которые соответствуют только стандарту Bluetooth 3.0.

Как и в случае с Bluetooth LE, чтобы действительно понять, когда использовать Bluetooth Classic, нам нужно понять саму технологию, поэтому давайте углубимся в нее немного подробнее.

Bluetooth-

Радиус действия Bluetooth-радио зависит от класса. В приведенной таблице показан диапазон радиомодулей Bluetooth по классам:

Класс	Мощность (мВт)	Мощность (dBm)	Диапазон в метрах
1	100	20	~100
2	2.5	4	~10
3	1	0	~1

Как и в случае с любой другой радиотехникой, область вокруг радиоприемника существенно влияет на дальность его действия. Диапазон, указанный в предыдущем графике, является теоретическим максимальным диапазоном при идеальных условиях. Типичный диапазон обычно меньше этого теоретического максимального диапазона.

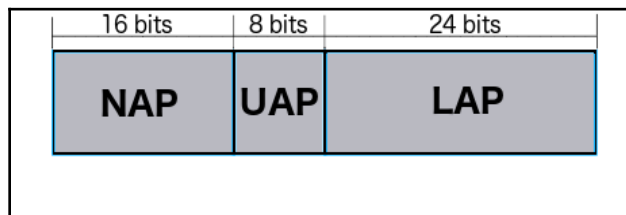
Если радиомодуль Bluetooth LE работает в диапазоне от 2402 МГц до 2480 МГц, причем каждый канал находится на расстоянии 2 МГц, то радиостанция Bluetooth Classic использует 79 каналов от 2402 МГц до 2480 МГц, причем каждый канал находится на расстоянии 1 МГц. Как и Bluetooth LE, Bluetooth Classicradio использует скачкообразную перестройку частоты, при которой радио меняет каналы 1600 раз в секунду, чтобы уменьшить помехи.

С Bluetooth LE радиостанция постоянно отключается для снижения энергопотребления, Bluetooth Classic этого не делает. Это делает радиотехнологию Bluetooth LE лучше при коротких пакетах данных с низким энергопотреблением, в то время как радиомодуль Bluetooth Classic лучше передает большие объемы данных или потоковую передачу данных, поскольку радиомодуль постоянно включен.

Все устройства Bluetooth имеют уникальный 48-битный адрес, присвоенный радиомодулю Bluetooth производителем. Верхняя половина адреса (24 старших бита) известна как Organizationally Unique - уникальный идентификатор организации и состоит из двух частей. Этими частями являются Non-Significant Address - Незначительный адрес (NAP) и Upper Address Part - Верхняя часть адреса (UAP).

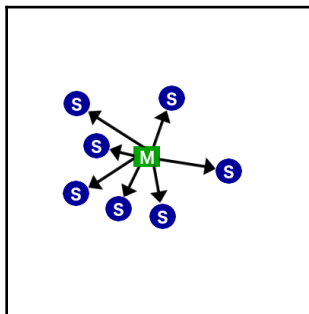
NAP - это первые 16 бит адреса, которые используются для синхронизации скачкообразной перестройки частоты. UAP - это следующие 8 битов, которые назначаются производителю радиостанции организацией IEEE.

Последние 24 бита адреса известны как нижняя часть адреса (LAP). LAP назначается производителем для однозначной идентификации радио. На следующей диаграмме показан составленный адрес Bluetooth:



Теперь посмотрим топологию сети для Bluetooth Classic.

Топология пикосети Bluetooth Classic очень похожа на топологию сети Bluetooth LE, где одно устройство действует как ведущее, а другие устройства действуют как ведомые устройства. В классической пикосети Bluetooth у одного ведущего устройства может быть до семи ведомых устройств, всего восемь устройств в пикосети. На следующей схеме показана классическая пикосеть Bluetooth:



Пикосети могут взаимодействовать с другими пикосетями, образуя так называемые рассеянные сети. Скаттернет - это где мастер одной пикосети действует как подчиненный в другой пикосети. Это позволяет устройствам в одной пикосети обмениваться данными с устройствами в других пикосетях; однако это требует сложной синхронизации и разделения полосы пропускания, что делает эти сети более сложными и менее эффективными. Приятно знать, что мы можем создавать рассеянные сети, но, по моему опыту, это редко используется.

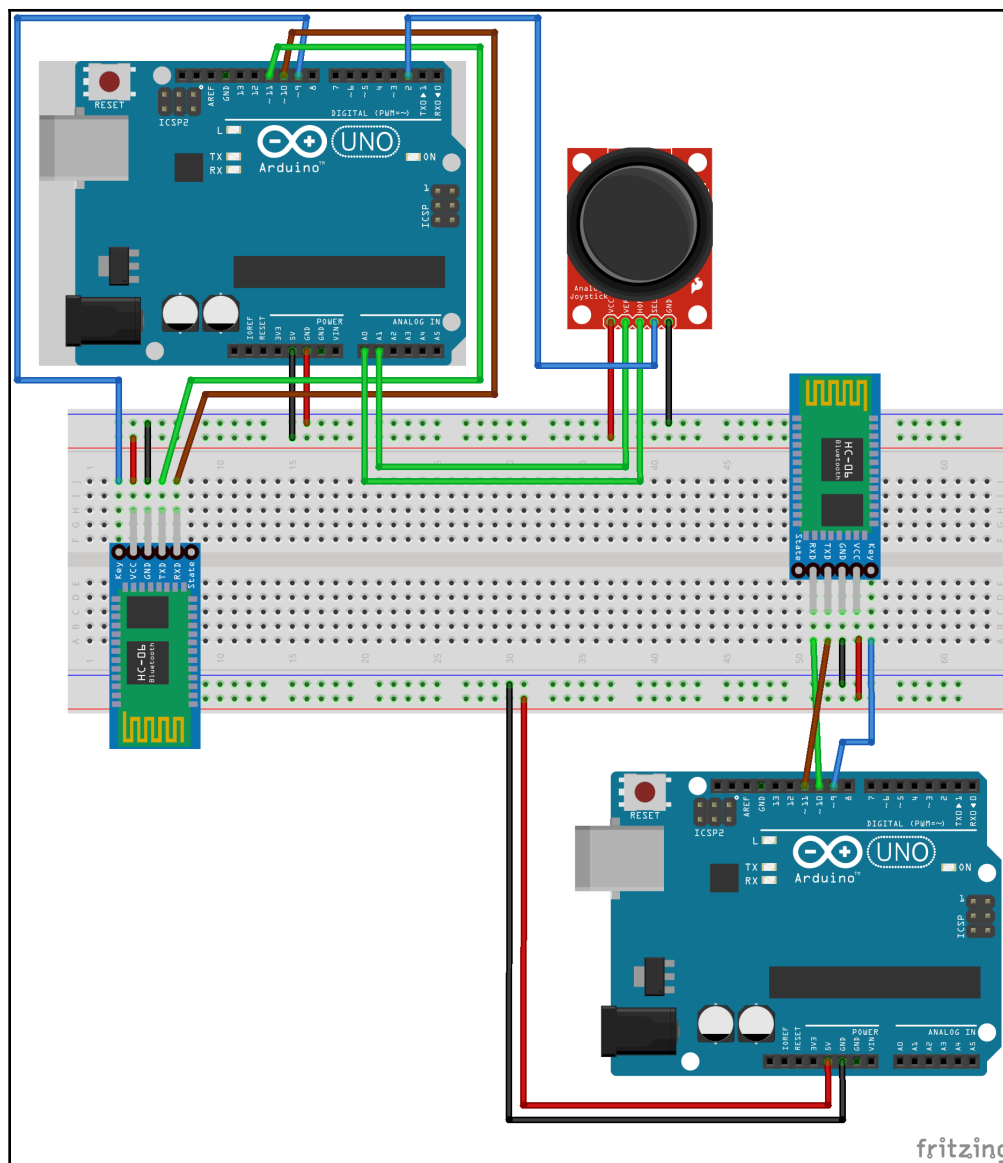
Bluetooth - это гораздо больше, чем описано здесь; однако в подавляющем большинстве случаев вы захотите использовать Bluetooth LE, как описано в главе 20, Bluetooth LE. С Arduino мы будем использовать Bluetooth Classic, когда мы хотим соединить два устройства и передавать данные между ними. Давайте посмотрим, как бы мы это сделали с тремя проектами. В первом проекте мы настроим модули Bluetooth, во втором проекте мы узнаем, как отправлять и получать данные из модуля Bluetooth, а в третьем проекте мы увидим, как мы можем передавать данные с одного радиомодуля на другой. Мы начнем с рассмотрения компонентов, которые нам понадобятся для этих проектов.

Для этих проектов вам понадобятся следующие компоненты:

- Две Arduino Uno или совместимые платы
- Два модуля Bluetooth HC-05
- Один модуль джойстика для Arduino
- Перемычки
- Макетная плата

Теперь посмотрим на принципиальную схему нашего проекта.

В этой главе мы напишем код для трех проектов. В первом проекте мы будем настраивать модули Bluetooth, во втором проекте мы создадим приложение, которое будет отправлять данные в байтовом формате от одного радиомодуля Bluetooth к другому, а в последнем проекте мы подключим джойстик к одному из Arduino и будем передавать его положение другому Arduino через соединение Bluetooth. Ниже показана принципиальная схема наших проектов:



Две схемы Arduino полностью изолированы друг от друга, поэтому им не нужна общая земля. Оба модуля Bluetooth HC-06 подключаются к Arduino таким же образом, где вывод VCC на модуле Bluetooth HC-06 подключен к выводу 5 В, а вывод GND подключен к земле на Arduino. Ключевой вывод на модуле Bluetooth подключен к цифровому выводу 9 на Arduino, вывод RX подключен к цифровому выводу 10, а вывод TX подключен к цифровому выводу 11.

Мы подключим модуль джойстика к одному из Arduino. Для этого нам нужно подключить вывод VCC на коммутационной плате к +5 В Arduino, а вывод GND - к минусу Arduino. Мы подключим вывод SEK или SW, в зависимости от вашего модуля джойстика, к цифровому выводу 2 на Arduino. Наконец, мы подключим вывод оси HOR или x на коммутационной плате к выводу Analog 0 на Arduino, а вывод оси VER или y к выводу Analog 1.

Теперь начнем с наших проектов.

1 -

Bluetooth

Для связи с модулем Bluetooth HC-05 мы будем использовать ту же библиотеку SoftwareSerial, которую мы использовали в главе 20, Bluetooth LE. Код, который используется для связи, очень похож на HM-10 (Bluetooth LE) и HC-05 (Bluetooth Classic). Способ передачи и приема данных двумя радиостанциями сильно различается, поэтому понимание того, как радиостанции работают и для чего их следует использовать, определит, когда использовать разные технологии.

Для этого первого проекта мы напишем приложение, которое позволит нам настраивать модули Bluetooth. Этот код начнется точно так же, как мы сделали с кодом Bluetooth LE, включив библиотеку SoftwareSerial и создав экземпляр типа SoftwareSerial.

Следующий код показывает, как это сделать:

```
#include <SoftwareSerial.h>
SoftwareSerial HC05(10, 11);
bool addNewLine = false;
```

Первая строка включает библиотеку SoftwareSerial, а вторая строка создает экземпляр типа. Логическая переменная в последней строке будет использоваться, чтобы сообщить приложению, когда нужно добавить новую строку в последовательную консоль.

Теперь нам нужно добавить код в функцию setup (), которая будет настраивать последовательную консоль и экземпляр SoftwareSerial. Следующий код показывает функцию setup () для этого первого проекта:

```
void setup()
{
  Serial.begin(9600);
  pinMode(9, OUTPUT);
  digitalWrite(9, HIGH);
  HC05.begin(38400);
}
```

```
    Serial.println("Connected to HC-05. Try connecting from any device or  
    issue AT commands");  
}
```

Этот код начинается с настройки последовательной консоли со скоростью 9600 бод. Затем определяется, что цифровой 9-контактный вывод будет выходным, и устанавливается на высокий уровень. Цифровой 9-контактный разъем подключен к ключевому выводу на HC-05. На этом выводе установим 1, чтобы включить модуль Bluetooth. Затем мы настраиваем экземпляр HC05 типа SoftwareSerial со скоростью передачи 38400 и выводим сообщение на последовательную консоль, позволяющее пользователю узнать, что все настроено и готово к работе.

Вы заметите, что в этом первом проекте мы установили скорость передачи экземпляра SoftwareSerial равной 38400, потому что мы настраиваем модуль Bluetooth. В следующих двух проектах мы устанавливаем скорость передачи данных 9600, потому что мы будем отправлять и получать данные в / из модуля Bluetooth.

В функции loop (), как и в коде Bluetooth LE, мы будем принимать любой ввод от модуля Bluetooth и выводить его на последовательную консоль, а любой ввод от последовательной консоли мы будем отправлять через модуль Bluetooth. Следующий код делает это:

```
void loop()  
{  
    if (HC05.available())  
    {  
        if (addNewLine) {  
            Serial.write("\r\n");  
            addNewLine = false;  
        }  
        Serial.write(HC05.read());  
    }  
  
    if (Serial.available())  
    {  
        HC05.write(Serial.read());  
        addNewLine = true;  
    }  
}
```

В этой функции первое, что мы делаем, это проверяем, есть ли какие-либо данные, доступные из экземпляра HC05 SoftwareSerial (модуль Bluetooth), с помощью функции available (). Если есть данные, мы проверяем, нужно ли добавлять новую строку в последовательную консоль, проверяя логическую переменную addNewLine. Если нам нужно добавить новую строку, мы записываем возврат каретки и перевод строки в последовательную консоль, а затем устанавливаем для логической переменной addNewLine значение false. Затем мы записываем данные, полученные от модуля Bluetooth, в последовательную консоль.

Затем мы проверяем, есть ли какие-либо данные, доступные с последовательной консоли, также используя функцию `available()`, и если да, мы записываем эти данные в модуль Bluetooth, который затем передается на подключенное устройство. Мы также устанавливаем логическую переменную `addNewLine` в значение `true`, чтобы в следующий раз, когда данные будут получены от подключенного устройства, мы добавим возврат каретки и перевод строки в последовательную консоль.

Прежде чем мы подключим Arduino и запустим этот код, нам нужно будет установить модуль Bluetooth HC-05 в режим конфигурации. Для этого нам нужно будет нажать и удерживать кнопку на модуле Bluetooth, а затем подключить Arduino к компьютеру, подавая питание на модуль Bluetooth. Буквально через пару секунд индикатор на модуле Bluetooth начнет очень медленно мигать; он будет гореть две секунды, а затем погаснет на две секунды. Как только индикатор начнет мигать, мы можем отпустить кнопку, и модуль Bluetooth будет готов к настройке.

Для настройки модуля Bluetooth мы выдаем AT-команды аналогично тому, как мы это делали с модулем Bluetooth LE. Чтобы отправить AT-команду, вы должны использовать следующий формат:

```
Set item: AT+{command}{new setting}
Query item: AT+{command}?
```

Чтобы установить элемент, вы вводите буквы AT, за которыми следует знак плюса, команду и новую настройку без пробелов. Например, чтобы установить роль модуля Bluetooth в качестве вспомогательной роли, мы должны выполнить следующую команду:

```
at+role0
```



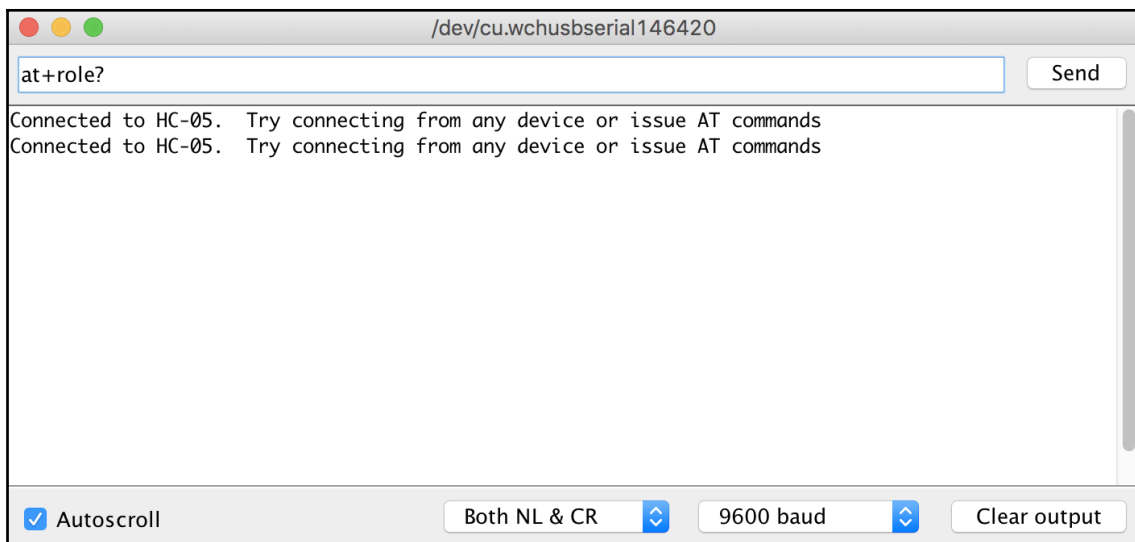
: AT-команды нечувствительны к регистру.

Например, чтобы запросить роль модуля Bluetooth, мы могли бы использовать следующую команду:

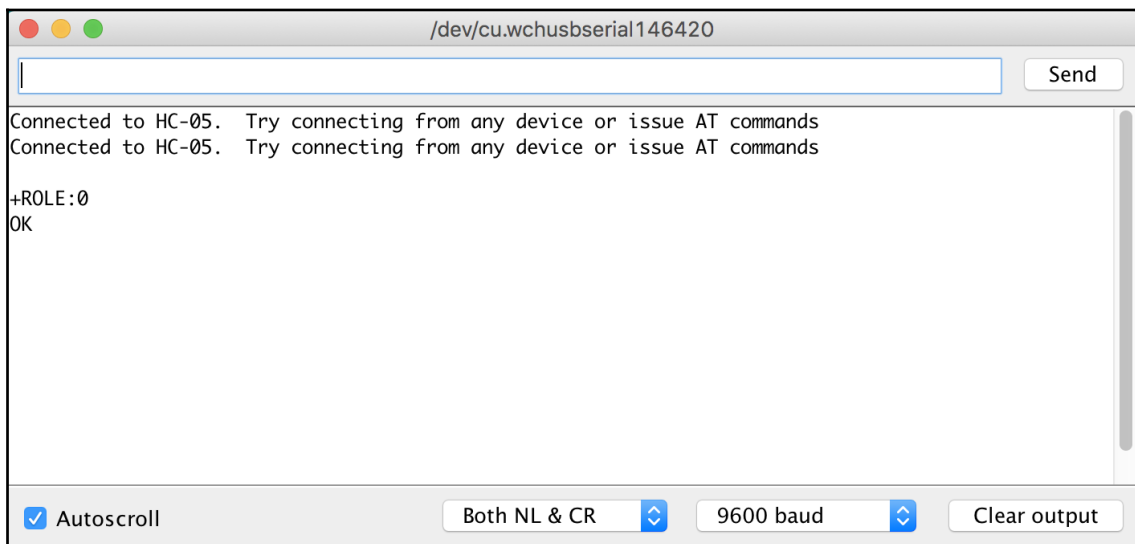
```
at+role?
```

Чтобы выполнить команду, мы вводим ее в поле ввода на последовательной консоли и нажимаем Enter. Нам нужно будет настроить последовательную консоль для добавления как NL (новая строка), так и CR (возврата каретки).

На следующем снимке экрана показано, как выполнить команду AT:



После того, как мы введем команду `at+role?`, нажимаем клавишу Enter или кнопку Send (Отправить), чтобы отправить команду в модуль Bluetooth. Модуль Bluetooth ответит результатами запроса, как показано на следующем снимке экрана:



Прежде чем настраивать модули, давайте рассмотрим некоторые команды, которые мы можем отдать модулю Bluetooth HC-05.

Тестовая команда

	Ответ	Параметры	
AT	OK	None	Это тестовая команда, которую можно использовать для проверки подключения к модулю Bluetooth.

Команда сброса

Команда	Ответ	Параметры	Описание
AT+RESET	OK	None	Эта команда сбросит модуль Bluetooth.

Запросить прошивку

Команда	Ответ	Параметры	Описание
AT+VERSION?	+VERSION:<Param>	None	Возвращает версию прошивки модуля Bluetooth HC-05.

Восстановить настройки по-умолчанию

Команда	Ответ	Параметры	Описание
AT+ORGL	OK	None	Восстанавливает для модуля HC-05Bluetooth настройки по умолчанию.

Адрес модуля запроса

Команда	Ответ	Параметры	Описание
AT+ADDR?	+ADDR:<Param>	None	Возвращает адрес модуля Bluetooth HC-05.

Установить / запросить режим модуля

Команда	Ответ	Параметры	Описание
AT+ROLE?	+ROLE:<Param>	0 Подчинен. 1 Ведомый	Запрашивает роль модуля Bluetooth HC-05.

Команда	Ответ	Параметры	Описание
AT+ROLE=<Param>	OK	0 Подчинен. 1 Ведомый	Устанавливает роль модуля Bluetooth HC-05.

Установить / запросить параметры UART

Команда	Ответ	Параметры	Описание
AT+UART?	+UART:<Param1>, <Param2>, <Param3>	Param1 = Baud Rate Param2 = Stop Bit Param3 = Parity	Запрашивает параметры UART.

Команда	Response	Параметры	Описание
AT+UART=<Param1>, <Param2>, <Param3>	OK	Param1 = скорость передач Param2 = стоповый бит Param3 = Четность	Устанавливает параметры UART.

Установить / запросить режим подключения

Команда	Ответ	Параметры	Описание
AT+CMODE?	+UART:<Param>	0 Подключиться к фиксированному адресу 1 Подключиться к любому адресу 2 ведомые циклы	Запрашивает режим подключения Bluetooth-модуля HC-05.

Команда	Ответ	Параметры	Описание
AT+CMODE=<Param>	OK	0 Подключиться к фиксированному адресу 1 Подключиться к любому адресу 2 ведомые петли	Устанавливает режим подключения для модуля Bluetooth HC-05.

Установить / запросить адрес привязки

Команда	Ответ	Параметры	Описание
AT+BIND?	+BIND:<Param>	None	Заголовок запроса, к которому настроен модуль.

Команда	Ответ	Параметры	Описание
AT+BIND=<Param>	OK	Фиксир. адрес	Устанавливает адрес для привязки.

Теперь, когда мы увидели большинство AT-команд, давайте настроим два модуля Bluetooth. Нам нужно будет настроить один из модулей Bluetooth как ведущий, а другой как ведомый. Для следующих двух проектов я настроил модуль Bluetooth, который подключен к тому же Arduino, что и джойстик, в качестве подчиненного устройства. Однако в этом нет необходимости, и любой модуль может быть ведущим или ведомым.

Начнем с настройки ведомого устройства. Для этого подключите один из Arduino к компьютеру, запустите приложение, которое мы написали в начале этого раздела, а затем выполните команду, которую мы опишем в следующих нескольких абзацах.

Первое, что нам нужно сделать, это подать тестовую AT-команду модулю Bluetooth. Модуль должен ответить сообщением OK. Если вы не получили ответ, убедитесь, что последовательная консоль настроена для отправки как NL, так и CR. Если вы получили ответ об ошибке, попробуйте снова ввести AT-команду.

Теперь, когда мы уверены, что последовательный монитор и модуль Bluetooth обмениваются данными, нам нужно посмотреть, какие настройки UART в настоящее время установлены для этого модуля. Для этого отправьте команду AT + UART?. Для примеров в этой главе мы предполагаем, что настройки UART - 9600 бод, 0 стоповых битов и 0 четности. Если ваш модуль настроен не так, выполните следующую команду:

AT+UART=9600,0,0

Следующее, что мы хотим сделать, это установить роль устройства как ведомую. Для этого мы выдаем следующую команду:

AT+ROLE=0

Наконец, нам нужно получить адрес этого модуля Bluetooth. Следующая команда получит адрес:

AT+ADDR?

Убедитесь, что адрес записан, потому что мы будем использовать его при настройке главного устройства.

Для настройки подчиненного модуля мы выполнили следующие команды:

Команда	Ответ
AT	OK
AT+UART?	+ UART: 9600,0,0 (если нет, установите по этим данным)
AT+ROLE=0	OK
AT+ADDR?	+ADDR:{address}

Теперь настроим мастера. Для этого подключите другой Arduino к компьютеру (не забудьте нажать и удерживать кнопку при включении модуля), запустите код, написанный в начале этого раздела, и выполните команды, которые мы рассмотрим в следующих нескольких абзацах. .

Как и в случае с ведомым устройством, первое, что нам нужно сделать, это подать AT-команду модулю Bluetooth. Модуль должен ответить сообщением OK. Если вы не получили ответ, убедитесь, что последовательная консоль настроена для отправки как NL, так и CR. Если вы получили ответ об ошибке, попробуйте снова ввести AT-команду.

Теперь нам нужно посмотреть, каковы настройки UART для модуля. Для этого отправьте команду AT + UART? . Для примеров в этой главе мы будем предполагать, что настройки UART - 9600 бод, 0 стоповых битов и 0 четности. Если ваш модуль настроен не так, введите следующую команду:

```
AT+UART=9600,0,0
```

Следующее, что мы хотим сделать, это установить роль устройства как ведущую. Для этого введите следующую команду:

```
AT+ROLE=1
```

Теперь нам нужно установить режим подключения для подключения к фиксированному адресу (режим 0). Для этого введите следующую команду:

```
AT+CMODE=0
```

Поскольку мы говорим модулю Bluetooth подключиться к фиксированному адресу, нам нужно указать ему адрес ведомого устройства, к которому оно нужно подключиться.

Для этого введите следующую команду:

```
AT+BIND=????, ??, ???????
```

Знак вопроса - это адрес ведомого устройства. Когда мы запросили адрес ведомого устройства, адрес был возвращен, разделенный двоеточиями, например, 98d3: 31: 300e42. При вводе адреса в команде BIND, адрес должен быть разделен запятыми, например, 98d3,31,300e42.

Команды, которые мы использовали для настройки главного устройства:

Команда	Ответ
AT	OK

AT+UART?	+ UART: 9600,0,0 (если нет, установите по этим данным)
AT+ROLE=1	OK
AT+CMODE=0	OK
AT+BIND=????, ??, ?????? (question marks are the address of the slave device)	OK

Теперь, если мы перезагрузим оба устройства, повторно включив питание, два модуля Bluetooth должны подключиться. Начните с повторного включения питания ведомого устройства, и вы увидите, что светодиод быстро мигает. Затем повторно включите питание ведущего устройства, и как только два устройства подключатся, светодиод на обоих устройствах быстро мигнет дважды, затем погаснет на две секунды и затем повторите. Эта последовательность индикаторов указывает на то, что два устройства подключены.

Если устройства не подключаются, наиболее распространенной ошибкой является ввод неправильного адреса в команде AT + BIND. Я бы начал с проверки, запустив команду AT + BIND? и проверки правильности адреса. Если это верно, проверьте правильность выполнения команд AT + CMODE и AT + ROLE, запустив команды AT + CMODE? а AT + РОЛЬ? . Теперь, когда у нас есть два подключенных модуля Bluetooth, перейдем ко второму проекту.

2 -

,

Для этого проекта, чтобы видеть данные, передаваемые с одного устройства на другое, вам понадобятся два компьютера.

Один подключен к главному устройству, а другой - к ведомому. Если у вас нет двух компьютеров, все равно стоит прочитать этот раздел, чтобы понять протокол, который мы создаем. потому что мы будем использовать тот же протокол и для третьего проекта.

Когда мы осуществляем потоковую передачу данных или отправляем большие объемы данных переменной длины, нам нужен способ сообщить принимающему устройству, где начинается и где заканчивается новое сообщение. К счастью для нас, есть встроенные коды ASCII, которые позволяют это. Коды 0x01 SOH (Start Of Heading - начало заголовка) и 0x04 EOT (End Of Transmission - конец передачи) могут использоваться, чтобы сообщить принимающему устройству, когда сообщение начинается и когда оно заканчивается.

В этом и следующем проектах мы определим протокол: когда принимающее устройство получает символ ASCII 0x01, оно будет знать, что началось новое сообщение. Когда он получает символ ASCII 0x04, он будет знать, что сообщение закончилось, и все, что находится между символами 0x01 и 0x04, является самим сообщением.


```
Serial.begin(9600);  
pinMode(9, OUTPUT);  
digitalWrite(9, HIGH);  
HC05.begin(9600);  
Serial.println("Connected to HC-05. ");  
}
```

В этом коде мы начинаем с настройки последовательной консоли на скорость передачи 9600 бод. Затем мы определяем, что цифровой 9-контактный вывод будет выходным и устанавливаем его на высокий уровень. Цифровой 9-контактный разъем подключается к ключевому выводу на модуле Bluetooth HC-05. Мы устанавливаем на нем высокий уровень, чтобы включить его. Мы настраиваем экземпляр HC05 типа SoftwareSerial на скорость передачи 9600 бод и затем выводим сообщение на последовательную консоль, позволяющее пользователю узнать, что все настроено и готово к работе.

Функция loop () должна будет контролировать как последовательную консоль, так и экземпляр HC05SoftwareSerial на предмет поступления новых данных. Если она получает новые данные от последовательной консоли, ей необходимо передать их через модуль Bluetooth, а если она получит новые данные от модуль Bluetooth, он должен будет отображать данные в последовательной консоли. Следующий код делает это:

```
void loop()  
{  
  if (HC05.available())  
  {  
    byte val = HC05.read();  
    Serial.write(val);  
    if (val == 0x04)  
    {  
      Serial.write("\r\n");  
    }  
  }  
  if (Serial.available())  
  {  
    if (newMessage)  
    {  
      HC05.write(0x01);  
      newMessage = false;  
    }  
    char val = Serial.read();  
    if (val == '~')  
    {  
      HC05.write(0x04);  
      newMessage = true;  
    }  
    else  
    {  
      HC05.write(val);  
    }  
  }  
}
```

```
    }  
  }  
}
```

В этой функции мы проверяем, доступны ли какие-либо данные из экземпляра HC05SoftwareSerial, и если да, они считываются в переменную val. Затем переменная val записывается в последовательную консоль. Затем мы видим если переменная val равна 0x04 и если мы напишем возврат каретки и перевод строки в последовательную консоль, потому что это конкретное сообщение закончилось.

Теперь мы проверяем, есть ли какие-либо данные, доступные из последовательной консоли, и если да, мы проверяем, запускаем ли мы новое сообщение, проверяя, равна ли переменная newMessage значению true. Если переменная newMessage равна true, мы записываем символ 0x01 в экземпляр HC05 SoftwareSerial, а затем устанавливаем для переменной newMessage значение false. Затем мы прочитаем символ из последовательной консоли и посмотрим, равен ли он символу тильды (~). Мы собираемся использовать символ тильды, чтобы указать, что сообщение закончилось повторно. Когда пользователь вводит тильду, мы напишем 0x04 в экземпляр HC05 SoftwareSerial и установим для переменной newMessage значение true, потому что это конкретное сообщение закончилось. Если символ не равен тильде, мы записываем символ в экземпляр SoftwareSerial.

Теперь, если мы запустим этот код как на главном, так и на подчиненном устройстве, все, что мы введем в пробную консоль на одном устройстве, будет передано на другое устройство через модули Bluetooth. Сообщение будет продолжать печататься в одной строке последовательной консоли до тех пор, пока пользователь не введет тильду, обозначающую конец сообщения.

Передача текста вперед и назад - это хорошо, но мы можем сделать это с помощью Bluetooth LE, как мы видели в главе 20, Bluetooth LE. Давайте сделаем что-нибудь более полезное, посмотрев, как это можно использовать в качестве пульта дистанционного управления, подключив модуль джойстика Arduino к одному из устройств и передав информацию о положении джойстика на другое устройство.

Project 3 -

Если вы не подключили модуль переключения джойстика к одному из Arduino, вам нужно будет сделать это до того, как вы начнете этот проект. Как только модуль переключения джойстика будет подключен к Arduino, мы напишем код, который будет считывать положение джойстика и передавать его на другой Arduino через модули Bluetooth HC-05; однако, прежде чем мы это сделаем, нам нужно выяснить протокол, который мы собираемся использовать.

В этом примере мы будем использовать тот же протокол, который мы использовали в предыдущем проекте, где сообщение будет начинаться с байта 0x01 и заканчиваться байтом 0x04, а все, что находится между ними, является самим сообщением.

Само сообщение будет содержать два байта, один из которых указывает положение джойстика по оси x, а другой - положение по оси y. Следовательно, полная передача будет содержать в общей сложности четыре байта, например:

```
0x01 - Start of header
0xDD - X position (221 decimal)
0xDD - Y position (221 decimal)
0x04 - End of transmission
```

Теперь, когда у нас есть протокол, который будет использоваться для передачи положения джойстика от одного Arduino к другому, давайте начнем с написания кода, который будет работать на Arduino, к которому подключен модуль переключения джойстика. Положение джойстика считывается через два подключенных к нему аналоговых контакта. Нам также нужно вывести на вывод SEL, который подключен к цифровому 2-выводу, высокий уровень.

Первое, что нам нужно сделать в коде, - это включить библиотеку SoftwareSerial для модуля Bluetooth, создать экземпляр типа SoftwareSerial и определить контакты, к которым подключен модуль джойстика. Следующий код сделает это:

```
#include <SoftwareSerial.h>
#define SW_PIN 2 // digital pin Joystick
#define BT_PIN 9 // digital pin Bluetooth
#define X_PIN 0 // analog pin
#define Y_PIN 1 // analog pin

SoftwareSerial HC05(10, 11);
```

В этом коде мы определяем вывод SEL для джойстика как цифровой 2-контактный, ключевой вывод на модуле Bluetooth как цифровой 9-контактный, а оси x / y как аналоговые выводы 0 и 1.

В функции setup () нам потребуется установить высокий уровень для SW_PIN и BT_PIN и инициализировать как последовательную консоль, так и экземпляр экземпляра SoftwareSerial. Вот код функции setup ():

```
void setup()
{
  pinMode(BT_PIN, OUTPUT);
  digitalWrite(BT_PIN, HIGH);
  pinMode(SW_PIN, OUTPUT);
  digitalWrite(SW_PIN, HIGH);
  HC05.begin(9600);
  Serial.begin(9600);
  Serial.println("Connected to HC05.");
}
```

К настоящему моменту этот код должен выглядеть хорошо знакомым. Первые четыре строки инициализируют цифровые выходы и устанавливают на них 1. Следующие две строки инициализируют экземпляр SoftwareSerial и последовательную консоль со скоростью 9600 бод. Наконец, на последовательную консоль выводится сообщение, информирующее пользователя о том, что все готово.

В нашей функции loop () нам нужно будет прочитать положение джойстика, а затем записать сообщение в модуль Bluetooth. Последующий

```
void loop()
{
  int xpos = analogRead(X_PIN) / 4;
  int ypos = analogRead(Y_PIN) / 4;
  HC05.write(0x01);
  HC05.write(xpos);
  HC05.write(ypos);
  HC05.write(0x04);
  delay(500);
}
```

Первые две строки читают оси x и y модуля джойстика. При чтении аналогового пинта возвращаемые значения находятся в диапазоне от 0 до 1024; однако мы хотим отправить только один байт, чтобы представить положение джойстика. Один байт может иметь диапазон от 0 до 255, поэтому мы делим значение аналогового считывания на 4.

После того, как мы получили значения для осей x и y джойстика, нам нужно отправить сообщение через модуль Bluetooth с этими значениями. Следующие четыре строки кода записывают 0x01 (SOH), значение оси x, значение оси y и, наконец, 0x04 (EOT). После отправки сообщения мы делаем паузу на 500 миллисекунд, а затем возвращаемся назад.

Теперь, когда у нас есть код, который будет работать на Arduino, к которому подключен джойстик, нам нужно написать код, который будет запускаться на Arduino, который будет получать данные. Этот код необходимо начать с включения библиотеки SoftwareSerial для модуля Bluetooth и создания экземпляра типа SoftwareSerial. Нам также нужно будет определить буфер, который будет использоваться для хранения данных, поступающих через модуль Bluetooth.

Следующий код сделает это:

```
#include <SoftwareSerial.h>
#define MAXBUF 255
#define BT_PIN 9 // digital pin Bluetooth
SoftwareSerial HC05(10, 11);
byte buf[MAXBUF];
```


Этот код включает библиотеку SoftwareSerial, а затем определяет максимальный размер входного буфера, равный 255. Хотя мы можем ограничить размер буфера до четырех, поскольку мы знаем, что каждое сообщение будет иметь размер четыре байта, мы всегда хотим иметь дополнительное место в буфере, особенно при беспроводной связи, на случай, если сообщение будет неправильно передано. Если бы это была производственная система, я бы, вероятно, ограничил размер буфера 12 или 16 байтами.

Мы определяем, что ключевой вывод на модуле Bluetooth подключен к цифровому 9 выводу на Arduino. Затем мы создаем экземпляр типа SoftwareSerial и байтовый массив для входного буфера.

В функции setup () мы инициализируем последовательную консоль и экземпляр SoftwareSerial. Нам также нужно будет выставить на ключевом выводе **единицу** для этого модуля Bluetooth.

Следующий код делает это:

```
void setup()
{
  Serial.begin(9600);
  pinMode(BT_PIN, OUTPUT);
  digitalWrite(BT_PIN, HIGH);
  HC05.begin(9600);
  Serial.println("Connected to HC05");
}
```

Теперь в функции loop () нам нужно будет постоянно читать входные данные из модуля Bluetooth, пока мы не получим байт EOT (0x04). Когда мы считываем данные, они будут помещены в массив байтов, и как только будет считан байт 0x04, мы распечатаем сообщение, а затем вернемся в цикл. Вот код функции loop ():

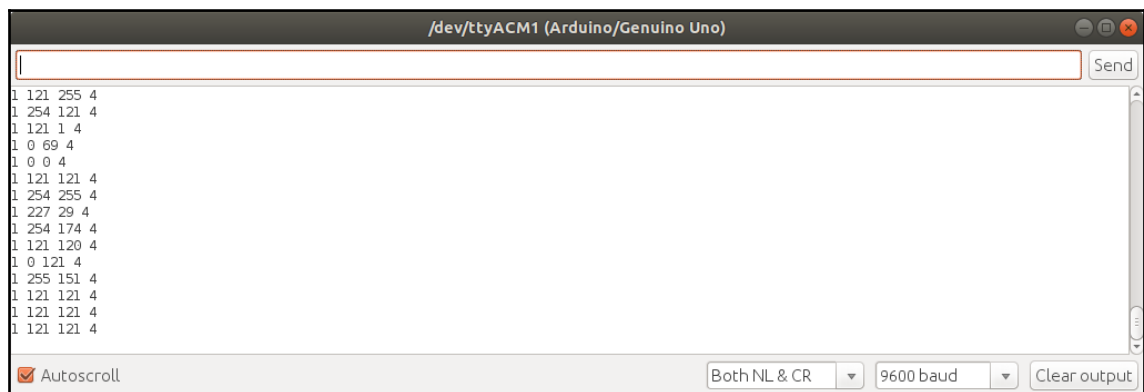
```
void loop()
{
  memset(buf, 0, MAXBUF);
  int counter = 0;
  while (counter < MAXBUF)
  {
    if (HC05.available())
    {
      byte val = HC05.read();
      buf[counter] = val;
      counter++;
      if (val == 0x04)
      {
        break;
      }
    }
  }
}
```

```
for(int i=0; i<counter; i++)
{
    Serial.print(buf[i]);
    Serial.print(" ");
}
Serial.println(" ");
}
```

Эта функция начинается с использования функции `memset()` для инициализации буфера всеми нулями. Затем мы создаем целочисленную переменную, которая будет подсчитывать, сколько байтов прочитано.

Цикл `while` используется для непрерывного цикла до тех пор, пока не будет прочитано максимальное количество байтов. В цикле `while` мы используем функцию `available()` из экземпляра `HC05 SoftwareSerial`, чтобы проверить, есть ли любые значения для чтения из модуля Bluetooth. Если есть значение для чтения, мы используем функцию `read()` для чтения значения, сохранения его в массиве байтов `buf` и увеличения счетчика. Затем мы проверяем, равно ли считанное значение `0x04`, и если да, мы используем оператор **break** для выхода из цикла `while`.

Наконец, мы создаем цикл `for`, который будет перебирать значения в буфере и выводить их на последовательную консоль. Если мы выполним код на обоих Arduino и переместим джойстик, мы увидим результат, аналогичный следующему снимку экрана:



Как видно из вывода, каждое сообщение начинается с байта `0x01` и заканчивается байтом `0x04`. Между этими двумя байтами находится положение джойстика по оси `x` и оси `y`.

Мы знаем, что пакеты должны иметь длину четыре байта. В производственной среде мы хотели бы отбрасывать любые сообщения длиной менее четырех байтов, потому что мы знаем, что если длина сообщения меньше четырех байтов, то при передаче сообщение было ошибочным.

Мы также можем использовать контрольную сумму, чтобы убедиться, что сообщение было получено правильно. Контрольная сумма - это некоторое значение, которое рассчитывается с использованием отправленных данных. Один из самых простых способов сгенерировать контрольную сумму - это сложить все байты данных, сохранив значение в байте, что приведет к переносу значения, когда оно больше 255. Вот пример функции, которая генерирует контрольную сумму:

```
byte checksum(byte *bytes, int buf_size)
{
    byte checksum = 0;
    for (int i=0; i< buf_size; i++)
    {
        checksum += bytes[i];
    }
    return checksum;
}
```

Эта функция принимает указатель на массив байтов и размер массива в качестве параметров. Затем он перебирает массив и добавляет каждый байт к контрольной сумме, а затем возвращает значение. Байт может иметь максимум значение 255, поэтому, как только значение превысит 255, значение будет повторяться. Например, если байт контрольной суммы имел значение 252 и к нему было добавлено значение 10, тогда значение контрольной суммы было бы 7. Затем мы могли бы отправить контрольную сумму перед значением 0x04, и устройство, получившее сообщение, могло бы проверить целостность сообщения. путем вычисления контрольной суммы на принимающей стороне и проверки совпадения двух значений.

В этой главе мы многое узнали о Bluetooth Classic, начав с краткого введения о том, как работает радио, и о топологии сети для подключений Bluetooth Classic. Мы продемонстрировали, как можно настроить модуль Bluetooth HC-05 Bluetooth как в качестве подчиненного, так и в качестве главного. Мы также увидели, как можно настроить модули Bluetooth для автоматического подключения друг к другу при запуске. Наконец, мы увидели, как можно передавать данные с одного устройства на другое, используя классический Bluetooth.

В главе 20 «Bluetooth LE» и в этой главе мы рассмотрели две разные технологии Bluetooth, но все еще может возникнуть вопрос, когда какую из них использовать. Когда у нас есть сценарий использования, который определяет, что мы хотим, чтобы одно устройство периодически запрашивало информацию у другого устройства, например, метеостанция, мы обычно хотим использовать Bluetooth LE. Когда мы хотим передавать данные с одного устройства на другое, не дожидаясь, пока принимающее устройство запросит об этом, мы обычно хотим использовать Bluetooth Classic.

В течение этой книги мы рассмотрели множество различных элементов, от микроконтроллера до датчиков и от двигателей до модулей беспроводной связи. Идея заключалась в том, чтобы познакомить вас с рядом различных предметов, чтобы, надеюсь, дать вам идеи для ваших собственных проектов. Самое лучшее в Arduino - это то, что проекты, которые вы делаете, ограничены только вашим воображением, поэтому начните представлять, какие супер-крутые проекты вы можете делать, а затем создавайте их.