Нил Кэмерон

Электронные проекты на основе ESP8266 и ESP32

Создание приложений и устройств с поддержкой Wi-Fi

ДМК Пресс
издательство

Нил Кэмерон

Электронные проекты на основе ESP8266 и ESP32

Создание приложений и устройств с поддержкой Wi-Fi

Electronics Projects with the ESP8266 and ESP32

Building Web Pages, Applications, and WiFi Enabled Devices

Neil Cameron

Apress®

Электронные проекты на основе ESP8266 и ESP32

Создание приложений и устройств с поддержкой Wi-Fi

Нил Кэмерон

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2022

УДК 621.3
ББК 32.85
К98

Нил Кэмерон

К98 Электронные проекты на основе ESP8266 и ESP32: Создание приложений и устройств с поддержкой Wi-Fi / пер. с англ. Ю. В. Ревича. – М.: ДМК Пресс, 2022. – 456 с.: ил.

ISBN 978-5-93700-141-2

Микроконтроллеры ESP8266 и ESP32 необычайно популярны во всем мире как основа для построения интернета вещей и систем умного дома. Они сочетают простоту применения и дешевизну с достаточно высокими возможностями, характерными для 32-разрядных платформ. Популярность их в значительной мере обусловлена наличием легкодоступного и бесплатного ПО, совместимого с уже ставшей стандартом в любительских кругах средой разработки Arduino IDE. В книге делается акцент на практических проектах – начиная от создания мобильных приложений для удаленного управления устройствами с распознаванием речи до GPS-трекинга с использованием Google Maps.

Книга адресована всем любителям DIY, умеющим работать с Arduino и заинтересованным в создании настоящих IoT-устройств и интеграции их в систему умного дома.

Дизайн обложки разработан с использованием ресурса freepik.com.

УДК 621.3
ББК 32.85

ISBN (анг.) 978-1-48426-335-8
ISBN (рус.) 978-5-93700-141-2

Оглавление

Предисловие от издательства 10

Об авторе 11

О техническом рецензенте 12

Предисловие 13

Глава 1. Интернет-радио 15

Выбор и отображение станции 20
Простейшее интернет-радио 28
Итоги 29
Перечень компонентов 29

Глава 2. Сетевая фотокамера 30

Загрузка изображений на веб-страницу 36
Потоковая передача изображений на веб-страницу 39
Потоковая передача изображений на веб-страницу по сигналу PIR-датчика 41
Итоги 45
Перечень компонентов 45

Глава 3. Международная метеостанция 46

Сенсорный дисплей ILI9341 SPI TFT LCD 46
Калибровка сенсорного экрана 49
Рисование на экране 51
Особенности ESP8266 при калибровке сенсорного экрана и рисовании 52
Данные о погоде для различных городов 56
Итоги 65
Перечень компонентов 65

Глава 4. Интернет-часы 66

Светодиодная RGB-лента WS2812, управляемая звуком 69
ESP8266 и мультиплексор 72
Часы на светодиодных кольцах 75
Протокол NTP (Network Time Protocol) 79
Интернет-часы и ESP32 81
Итоги 82
Перечень компонентов 82

Глава 5. MP3-плеер 83

Команды управления для MP3-плеера 84
Управление MP3-плеером с помощью микроконтроллера 85

Инфракрасный пульт дистанционного управления MP3-плеером ... 91
Создание треков и две системы сигнализации ... 94
Сигнализация с обнаружением перемещения ... 98
Говорящие часы ... 100
Диктофон ... 104
Итоги ... 106
Перечень компонентов ... 106

Глава 6. Bluetooth-динамик ... 107

Итоги ... 111
Перечень компонентов ... 111

Глава 7. Беспроводная локальная сеть ... 112

HTTP-запрос ... 114
HTML-код ... 118
XML HTTP-запросы, JavaScript и AJAX ... 120
Итоги ... 125
Перечень компонентов ... 125

Глава 8. Обновление веб-страницы ... 126

XML HTTP-запросы, JavaScript и AJAX ... 130
JSON ... 132
Доступ к данным WWW ... 135
MQTT-брокер и IFTTT ... 139
Парсинг текста ... 148
Ведение логов консоли ... 149
Подключение к Wi-Fi ... 150
Файл с информацией о доступе ... 151
Итоги ... 152
Перечень компонентов ... 152

Глава 9. WebSocket ... 153

Дистанционное управление и связь через WebSocket ... 156
WebSocket и AJAX ... 161
Доступ к изображениям, времени и показаниям датчиков через интернет ... 165
Итоги ... 173
Перечень компонентов ... 173

Глава 10. Создаем мобильное приложение ... 174

Приложение для управления с обратной связью ... 175
Установка приложения ... 184
Приложение для управления сервороботом ... 185
Приложение для распознавания речи ... 191
Итоги ... 195
Перечень компонентов ... 195

Глава 11. Приложение базы данных и Google Maps 196

База данных MIT App Inventor 196
MIT App Inventor и Google Maps 201
Итоги 207
Перечень компонентов 207

Глава 12. Приложение для GPS-трекинга с использованием Google Maps 208

Передача GPS-данных о местоположении 215
Получение GPS-данных о местоположении 219
Проверка передачи GPS-данных о местоположении 220
Улучшение GPS-сигнала 227
Итоги 232
Перечень компонентов 233

Глава 13. Связь через USB OTG 234

Приложение для приема данных 235
Приложение для передачи данных 239
Приложение для приема и передачи данных 243
Итоги 244
Перечень компонентов 245

Глава 14. Обмен данными через ESP-NOW и LoRa 246

ESP-NOW 246
LoRa 256
Итоги 265
Перечень компонентов 265

Глава 15. Радиочастотная связь 266

Передача и прием текста 269
Декодирование сигналов дистанционного управления 273
Управление сервоприводами поворота и наклона с помощью RF-связи 277
Управление реле по RF-связи 282
Реле 285
Твердотельное реле 288
Итоги 289
Перечень компонентов 290

Глава 16. Генерация сигналов 291

Генерация колебаний 294
Цифроаналоговый преобразователь 296
Генерация колебаний 300
8-разрядный ЦАП ESP32 305
12-разрядный ЦАП 305

Итоги 309
Перечень компонентов 310

Глава 17. Генерация сигнала с помощью микросхемы таймера 555 311

Микросхема таймера 555 311
Моностабильный режим 314
Бистабильный режим 316
Режим генерации 317
Переменный коэффициент заполнения 320
50%-ный коэффициент заполнения 322
Режим ШИМ 325
Функциональный генератор 326
Преобразование прямоугольного колебания в синусоидальное 330
Биполярный транзистор в качестве ключа 332
Приложение с MP3-плеером и PIR-датчиком 334
Итоги 337
Перечень компонентов 338

Глава 18. Электрические измерения 339

Делитель напряжения 339
Аналого-цифровой преобразователь 341
Измеритель напряжения 342
Измеритель напряжения с нагрузкой 345
Измеритель сопротивления (омметр) 348
Измеритель емкости 350
Измеритель тока (амперметр) 353
Датчик тока 358
Датчик тока и напряжения 360
Измеритель для солнечной панели с аккумулятором 362
Измеритель индуктивности 369
Итоги 373
Перечень компонентов 373

Глава 19. Поворотный энкодер 375

Устранение дребезга контактов 378
Прерывания 378
Подсчет состояний 380
Переключение состояний 385
Увеличение значения 386
Итоги 389
Перечень компонентов 390

Глава 20. OTA и сохранение данных в EEPROM, SPIFFS и Microsoft Excel 391

OTA-обновление 391
Сохранение данных 394

Сохранение в EEPROM....395
Сохранение в SPIFFS....398
Загрузка файлов из SPIFFS....402
Сохранение данных в Excel....404
Итоги....407
Перечень компонентов....407

Глава 21. Микроконтроллеры....408

Arduino Uno....412
Arduino Nano....412
Arduino Pro Micro....413
Модули ESP8266....414
Аналоговый вход ESP8266....417
Прерывания ESP8266....417
Сторожевой таймер ESP8266....419
Модули ESP32....419
Цифровой вход ESP32....422
Аналоговый вход ESP32....422
Широтно-импульсная модуляция в ESP32....423
Вход последовательного порта ESP32....424
Связь по Wi-Fi и веб-сервер....424
Прерывания ESP8266 и ESP32....425
ESP8266, ESP32 и OLED-экран....425
ESP32 и сервопривод....425
Итоги....426
Перечень компонентов....426

Глава 22. Особенности микроконтроллера ESP32....427

Процессор и память....427
Ядра ESP32....428
Связь по Bluetooth....434
Связь Bluetooth Low Energy....436
Таймеры....445
RTC и спящий режим....447
Цифроаналоговый преобразователь....449
Емкостный сенсорный датчик....449
Датчик Холла....450
Итоги....451
Перечень компонентов....451

Приложение....452

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Нил Кэмерон – опытный аналитик и программист с глубоким пониманием работы электронных устройств. Нил – автор книги *Arduino Applied: Comprehensive Projects for Everyday Electronics* (изд-во «Апресс»). Работал научным сотрудником и преподавал в Эдинбургском и Корнелльском университетах.

О техническом рецензенте

Майк МакРобертс – автор книги *Beginning Arduino* (изд-во «Апресс»). Лауреат Pi Wars 2018 и член Medway Makers. Энтузиаст Arduino и Raspberry Pi.

C/C++, Arduino, Python, Processing, JS, Node-Red, NodeJS, Lua.

Предисловие

Никогда еще не было так просто получать доступ к информации через интернет: разрабатывать веб-страницы для обновления информации с датчиков, создавать мобильные приложения для удаленного управления устройствами с распознаванием речи или интегрировать Google Maps в приложение для GPS-трекинга. Сочетание беспроводного доступа по Wi-Fi, высокой вычислительной мощности и низкой стоимости микроконтроллеров ESP8266 и ESP32 расширяет спектр возможностей. Связь с устройствами и доступ к информации через интернет с помощью микроконтроллеров ESP8266 и ESP32 – в центре внимания книги «Электронные проекты на основе ESP8266 и ESP32».

Главы с 1 по 6 демонстрируют мощь и легкость использования микроконтроллеров ESP8266 и ESP32 для получения и отображения интернет-информации. Представленные проекты включают в себя создание интернет-радио, интернет-часов и международной погодной станции, а также проект с камерой ESP32-CAM для загрузки фотографий на веб-страницы.

Главы с 7 по 9 посвящены проектам дизайна и обновления в режиме реального времени веб-страниц с информацией от датчиков, представленной в графической форме, или проектам управления удаленным устройством через веб-страницу. Вы узнаете об AJAX (асинхронный JavaScript и XML), который сочетает в себе расширяемый язык разметки XML и протокол передачи гипертекста HTTP, о запросах на обновление веб-страницы с помощью JavaScript, запросах JSON[1] для объединения информации, передаваемой сервером клиенту, о двухсторонней быстрой связи через протокол WebSocket, MQTT-брокеры и IFTTT[2] для связи между устройствами в разных сетях. Практические проекты включают в себя загрузку информации в интернет и управление устройствами из любой точки мира с помощью микроконтроллеров ESP8266 и ESP32.

Мобильные приложения теперь повсеместны, что делает представленные в главах с 10 по 13 проекты очень актуальными. Приложение для управления дистанционно расположенными сервоприводами, подключенными к плате с ESP8266 или ESP32, имитирует робототехнику, используемую в автомобильной промышленности; приложение с распознаванием речи управляет устройствами; приложение для GPS-трекинга, включающее Google Maps, отображает текущую позицию и информацию о маршруте. Каждый проект с микроконтроллерами ESP8266 и ESP32 полностью описан, причем от читателя не требуется предыдущего опыта проектирования и сборки мобильных приложений.

---

[1] JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript. Позволяет сократить объем кода и выполнять запросы быстрее, чем обычные XML HTTP-запросы. – *Прим. перев.*

[2] Расшифровывается как If This, Then That – *если это, то то-то*; подробнее см. главу 8. – *Прим. перев.*

Связь между микроконтроллерами ESP8266 и ESP32 описана главах с 14 по 18. Встроенная система связи ESP-NOW, связь LoRa (дальнего радиуса действия) и радиочастотная RF-связь применяются для управления удаленно расположенными устройствами с помощью информации, обновляемой на веб-странице микроконтроллерами ESP8266 и ESP32. Коммуникационные протоколы расширены для приложений генерации сигналов с помощью ESP8266 и ESP32, передающих теперь не только буквенно-цифровой текст, но и звуковые музыкальные сигналы. Генерация сигналов без микроконтроллеров показана на примерах проектов электронного пианино, управления сервоприводом и системы сигнализации, включающей MP3-плеер и детектор движения. Эти главы охватывают встроенный протокол связи микроконтроллеров ESP8266 и ESP32 с применением базовой электроники. Глава об электрических измерениях с помощью микроконтроллеров ESP8266 или ESP32, применяемых в проекте солнечной панели, распространяет электронную тему на измерения, чтобы понять методологию, лежащую в основе построения различных датчиков.

Более производительный, чем ESP8266, микроконтроллер ESP32 среди прочего включает связь Bluetooth и ее специальный вариант с низким энергопотреблением Bluetooth Low Energy (BLE). Главы 21 и 22 – о практических различиях между микроконтроллерами ESP8266 и ESP32 и отдельных особенностях ESP32.

На протяжении всей книги описаны все различия в библиотеках или инструкциях для микроконтроллеров ESP8266 и ESP32, так что каждый проект совместим с обоими микроконтроллерами.

Все главы книги являются самостоятельными, поэтому вы можете углубиться в любую главу, а не начинать с начала. Несколько глав основываются на информации из предыдущих глав. Например, глава 12 («Приложение для GPS-трекинга с использованием Google Maps») включает в себя дизайн мобильного приложения, связь Bluetooth, получение информации из интернета и обновление страницы в интернете. Предполагается некоторый опыт программирования в среде Arduino IDE, хотя все скетчи описаны полностью и исчерпывающе прокомментированы. Для знакомства с микроконтроллерами, начиная от мигания светодиода и заканчивая созданием автомобиля-робота, рекомендуется книга *Arduino Applied: Comprehensive Projects for Everyday Electronics*[3]. Принципиальные схемы[4] были созданы с помощью *программного обеспечения Fritzing (www.fritzing.org)* с акцентом на максимальной наглядности размещения компонентов и минимизации пересекающихся соединений. Авторы библиотек, использованных в книге, указаны в каждой главе, а сведения о библиотеках включены в *приложение*. Все скетчи Arduino IDE и исходный код MIT App Inventor для приложений доступны для загрузки на GitHub (github.com/Apress/ESP8266-and-ESP32). Среда программирования Arduino и библиотеки постоянно обновляются, поэтому информация о последних обновлениях также доступна на сайте GitHub.

[3] Для русскоязычных читателей можно порекомендовать такие книги: *Ревич Ю.* Азбука электроники. Изучаем Arduino. М.: АСТ, 2017; *Петин В. В., Биняковский А. А.* Практическая энциклопедия Arduino. 2-е изд. М.: ДМК Пресс, 2019. – *Прим. перев.*

[4] Строго говоря, представленные в книге схемы соединений компонентов принципиальными схемами не являются; принципиальные схемы представлены только в отдельных случаях для лучшей иллюстрации отдельных положений. – *Прим. перев.*

Глава 1

Интернет-радио

Интернет-радио – это непрерывная потоковая передача цифрового аудио через интернет. Цифровой звук в формате MP3 принимается микроконтроллером ESP8266 или ESP32 через соединение Wi-Fi. Микроконтроллер ESP8266 или ESP32 взаимодействует с аудиодекодером VS1053 с помощью SPI[5], принятые данные декодируются 18-разрядным цифроаналоговым преобразователем (ЦАП) и превращаются опять в аудиосигнал, который усиливается для вывода на динамик. Микроконтроллеры ESP8266 и ESP32 оборудованы интерфейсом Wi-Fi и имеют достаточную скорость работы процессора для интернет-радио. Для подключения к беспроводной локальной сети (WLAN) требуется SSID[6] сети Wi-Fi и пароль.

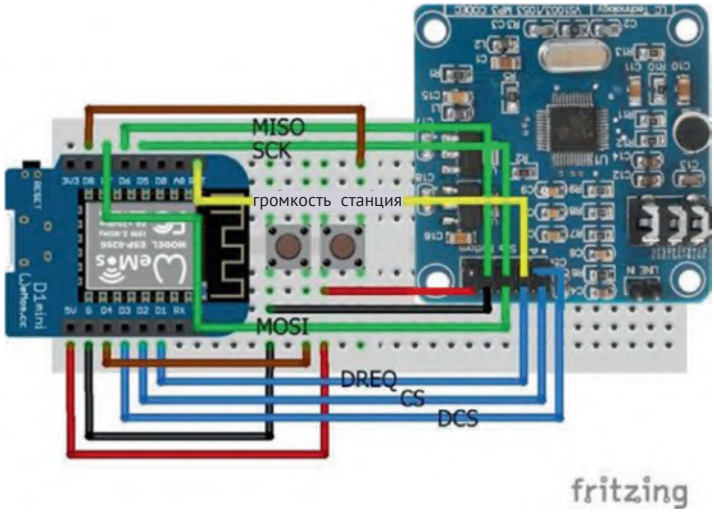


Рисунок 1-1. Интернет-радио с переключателями громкости и станций на основе платы LOLIN (WeMos) D1 mini

[5] SPI (Serial Peripheral Interface, последовательный периферийный интерфейс) – стандарт последовательной синхронной передачи данных для высокоскоростного соединения микроконтроллеров и периферии в масштабах устройства. – *Прим. перев.*

[6] SSID (Service Set Identifier, идентификатор набора услуг) – символьное наименование сети Wi-Fi, связанной с конкретной точкой доступа. Обычно точки доступа непрерывно передают в эфир наименование своей SSID, благодаря чему воспринимающее устройство «видит» все доступные сети в данном месте и может выбрать какую-либо из них для подключения. Существует и скрытый режим, при котором воспринимающее устройство обязано заранее «знать» SSID сети, чтобы подключиться к ней. – *Прим. перев.*

Соединения платы ESP8266 и аудиодекодера VS1053 показаны на рис. 1-1, подробно описаны на рис. 1-2 и перечислены в табл. 1-1. Соединения для связи по интерфейсу SPI обозначены зеленым цветом, а соединения для передачи данных – синим. Две кнопки (работающие на замыкание), подключенные к прерываниям, управляют громкостью (*volume*) и выбором интернет-радиостанции (*station*). Для платы ESP8266 кнопки переключения громкости и выбора станции на выводах D4 и D8 подключены к GND (черный провод) и 5V (красный провод), так как контакты D4 и D8 подключены к внутренним подтягивающему и заземляющему резисторам соответственно[7]. Соединения для платы на основе ESP32 (а не ESP8266) также приведены в табл. 1-1. При использовании платы ESP32 кнопки громкости и станции обе подключены к GND.

Усилитель и динамик, или мини-колонки, подобные используемым с мобильным телефоном, подключаются к аудиодекодеру VS1053 через обычный аудиоразъем типа «джек»[8].

Рисунок 1-2. Соединения аудиоплаты VS1053 (плата перевернута в сравнении с рис. 1-1)

Таблица 1-1. Соединения интернет-радио и кнопок

Компонент	Подключено к ESP8266	Подключено к ESP32
VS1053 5V	5V	VIN или V5
VS1053 DGND	GND	GND
VS1053 MOSI	(MOSI) D7	(MOSI) GPIO 23
VS1053 DREQ (запрос данных)	D1	GPIO 4
VS1053 XCS (*chip select*, выбор кристалла)	D2	GPIO 0
VS1053 MISO	(MISO) D6	(MISO) GPIO 19
VS1053 SCK	(SCK) D5	(CLK) GPIO 18

[7] Контакт D4 платы LOLIN D1 mini соответствует выводу GPIO 2 контроллера ESP8266, а D8 – выводу GPIO16 (см. рис. 21-1 на стр. 621), единственному, имеющему в ESP8266 заземляющий, а не подтягивающий встроенный резистор. Отсюда такой разнобой при установке внешних прерываний для контроллеров ESP8266 и ESP32 (см. далее). – *Прим. перев.*

[8] Разъем типа «джек» (audio jack socket) – круглый разъем с тремя или четырьмя контактами на одной оси. В настоящее время обычно «джеком» называют разъем диаметром 3,5 мм, хотя изначально он имел диаметр 1/4" (6,35 мм), а уменьшенный вариант 3,5 мм именовался «мини-джек». На рис. 1-1 гнездо для «джека» видно над правым нижним крепежным отверстием платы аудиодекодера. – *Прим. перев.*

Окончание табл. 1-1

Компонент	Подключено к ESP8266	Подключено к ESP32
VS1053 XRST (*reset*, сброс)	RST	GPIO EN
VS1053 XDCS (*data chip select*, выбор кристалла данных)	D3	GPIO 2
Левый вывод кнопки громкости (*volume*)	GND	GND
Правый вывод кнопки громкости (*volume*)	D4	GPIO 26
Левый вывод кнопки выбора станций (*station*)	5V	GND
Правый вывод кнопки выбора станций (*station*)	D8	GPIO 27

URL-адрес или веб-адрес интернет-радиостанции можно получить с веб-сайта www.radio.de. Найдите нужную станцию, нажмите кнопку воспроизведения и выберите в меню браузера *View Page Source* (*Исходный код страницы*)[9]. В отображаемом HTML-коде веб-страницы выполните поиск потока (*stream*), который содержится в URL-адресе радиостанции[10]. URL-адрес отформатирован как *хост:порт/путь*. Например, радиостанция Великобритании 1940-х годов имеет URL-адрес *1940sradio1.co.uk:8100/stream/1/*; хост здесь представлен текстом перед первой обратной косой чертой или двоеточием (*1940sradio1.co.uk*), а путь равен оставшемуся тексту (*stream/1/*). Если порт не равен значению 80, которое является портом для просмотра веб-страниц по умолчанию, то его значение следует за двоеточием после хоста, в данном случае, например, 8100.

В скетче интернет-радио с платой ESP8266 (см. листинг 1-1) используется библиотека *VS1053* Эда Смолленбурга (Ed Smallenburg) и Джеймса Колиза (James Coliz), которую можно загрузить в виде zip-файла по адресу *github.com/baldram/ESP_VS1053_Library*. Первый раздел скетча определяет количество интернет-радиостанций и их URL-адреса, инициализирует аудиодекодер, устанавливает соединение Wi-Fi и определяет прерывания. Переменные `newStation` и `newVolume` представлены типом `volatile`, поскольку к ним обращается как основной код программы, так и прерывания. На плате ESP32 вывод кнопки смены станции устанавливается в высокий уровень (*HIGH*) с помощью внутреннего подтягивающего резистора (инструкция `pinMode(statPin, INPUT_PULLUP;)`, а прерывание, подключенное к кнопке станции, устанавливается на срабатывание по падающему фронту (`FALLING`), а не возрастающему (`RISING`), как для платы ESP8266. Микроконтроллеры ESP8266 и ESP32 будут хранить скомпилированный код во внутренней оперативной памяти (IRAM), а не в более медленной флеш-памяти, если добавить к коду атрибут `IRAM_ATTR`. Поэтому обработчик прерывания (ISR[11]) определяется как `IRAM_ATTR void ISR()`, а не просто `void ISR()`.

---

[9] В большинстве браузеров этот пункт размещается в главном меню (клавиша **F10** или три точки в правом верхнем углу) примерно по следующему пути: *Инструменты (Дополнительные инструменты)* > *Инструменты веб-разработки* > *Исходный код страницы*. – *Прим. перев.*

[10] URL (Uniform Resource Locator, единый локатор ресурсов) – утвержденная форма представления интернет-адреса в виде буквенной строки, понятной для человека. – *Прим. перев.*

[11] Interrupt Service Routine, букв. *процедура обслуживания прерываний*. – *Прим перев.*

В функции `loop()` устанавливается соединение с веб-сайтом интернет-радиостанции, и аудиодекодер VS1053 обрабатывает данные в 32-байтовых пакетах. Две процедуры обслуживания прерываний, `chan` и `vol`, осуществляют переход к следующей радиостанции и увеличение громкости соответственно. Шкала громкости составляет от 0 до 100 %. Библиотека *VS1053* ссылается на библиотеку SPI, и инструкция `#include <SPI.h>` не требуется.

Подключение к серверу интернет-радиостанции с помощью команды `connect (host[station], port[station])` сопровождается HTTP-запросом. Библиотека VS1053 использует протокол HTTP для связи между клиентом и сервером интернет-радиостанции. Клиент отправляет HTTP-запрос на сервер для получения аудиоданных, и сервер отправляет клиенту ответ с требуемыми данными. За инструкциями HTTP-запроса "`GET pathname HTTP/1.1`" и "`Host: hostname`" следует инструкция по закрытию подключения "`Connection: close`". На примере радиостанции Великобритании 1940-х годов инструкции запроса будут следующими:

```
GET stream/1/HTTP/1.1
Host: 1940sradio1.co.uk
Connection: close
<\r\n>
```

Обратите внимание, что требуется четвертая команда – возврата каретки `\r` и новой строки `\n`, что эквивалентно инструкции `println()`.

Листинг 1-1. Интернет-радио с переключателями громкости и станций для платы ESP8266

```
#include <VS1053.h>                          // include VS1053 library
#include <ESP8266WiFi.h>                     // include ESP8266WiFi library
int CS = D2;
int DCS = D3;                                // define VS1053 decoder pins
int DREQ = D1;
VS1053 decoder(CS, DCS, DREQ);               // associate decoder with VS1053
int statPin = D8;                            // define switch pins for
int volPin = D4;                             // station and volume
WiFiClient client;                           // associate client and library
char ssid[] = "xxxx";                        // change xxxx to Wi-Fi ssid
char password[] = "xxxx";                    // change xxxx to Wi-Fi password
const int maxStat = 4;                       // number of radio stations
String stationName[] = {"1940 UK", "Bayern3", "ClassicFM", "BBC4"};
char * host[maxStat] = {"1940sradio1.co.uk", // station host
                        "streams.br.de",
                        "media-ice.musicradio.com",
                        "bbcmedia.ic.llnwd.net"};
char * path[maxStat] = {"/stream/1/",        // station path
                        "/bayern3_2.m3u",
                        "/ClassicFMMP3",
                        "/stream/bbcmedia_radio4fm_mf_q"};
int port[] = {8100,80,80,80};                // default station port is 80
unsigned char mp3buff[32];                   // VS1053 loads data in 32 bytes
int station = 0;
int volume = 0;                              // volume level 0-100
volatile int newStation = 2;                 // station number at start up
volatile int newVolume = 80;                 // volume at start up
void setup ()
```

```
{
   Serial.begin(115200);              // Serial Monitor baud rate
   SPI.begin();                       // initialise SPI bus
   decoder.begin();                   // initialise VS1053 decoder
   decoder.switchToMp3Mode();         // MP3 format mode
   decoder.setVolume(volume);         // set decoder volume
   WiFi.begin(ssid, password);        // initialise Wi-Fi
   while (WiFi.status() != WL_CONNECTED) delay(500);
   Serial.println("WiFi connected");  // wait for Wi-Fi connection
   pinMode(volPin, INPUT_PULLUP);     // switch pin uses internal
                                      // pull-up resistor
   attachInterrupt(digitalPinToInterrupt(statPin), chan, RISING);
   attachInterrupt(digitalPinToInterrupt(volPin), vol, FALLING);
}                            // define interrupts for changing station and volume
void loop()
{
    if(station != newStation)         // new station selected
    {
       station = newStation;          // display updated station name
       Serial.print("connecting to CH"); Serial.print(station);
       Serial.print(" ");Serial.println(stationName[station]);
       if(client.connect(host[station], port[station]))
       {                              // connect to radio station URL
          client.println(String("GET ")+ path[station] + " HTTP/1.1");
          client.println(String("Host: ") + host[station]);
          client.println("Connection: close");
          client.println();           // new line is required
       }
    }
    if(volume != newVolume)           // change volume selected
    {
       volume = newVolume;            // display updated volume
       Serial.print("volume ");Serial.println(volume);
       decoder.setVolume(volume);     // set decoder volume
    }
    if(client.available() > 0)        // when audio data available
    {                                 // decode data 32 bytes at a time
       uint8_t bytesread = client.read(mp3buff, 32);
       decoder.playChunk(mp3buff, bytesread);
    }
    }
IRAM_ATTR void chan()                 // ISR to increment station number
{
   newStation++;
   if(newStation > maxStat-1) newStation = 0;
}                                     // stations numbered 0, 1, 2...
IRAM_ATTR void vol()                  // ISR to increase volume
{
   newVolume = newVolume + 5;
   if(newVolume > 101) newVolume = 50;
}                                     // maximum volume is 100
```

В случае использования платы ESP32 ее подключения к аудиодекодеру VS1053 показаны на рис. 1-3 и 1-2 соответственно, а также приведены в табл. 1-1. Оба контакта переключателя подключены к внутренним под-

тягивающим резисторам, поэтому оба прерывания активируются падающим фронтом (FALLING). Единственные изменения в листинге 1-1, помимо определения выводов декодера, кнопок выбора станции и регулятора громкости, – используется библиотека *WiFi*, а не библиотека *ESP8266WiFi*, и инструкция `pinMode (statPin, INPUT_PULLUP)` для смены условия прерывания на выводе переключателя станции вместо возрастающего фронта (RISING) на падающий (FALLING).

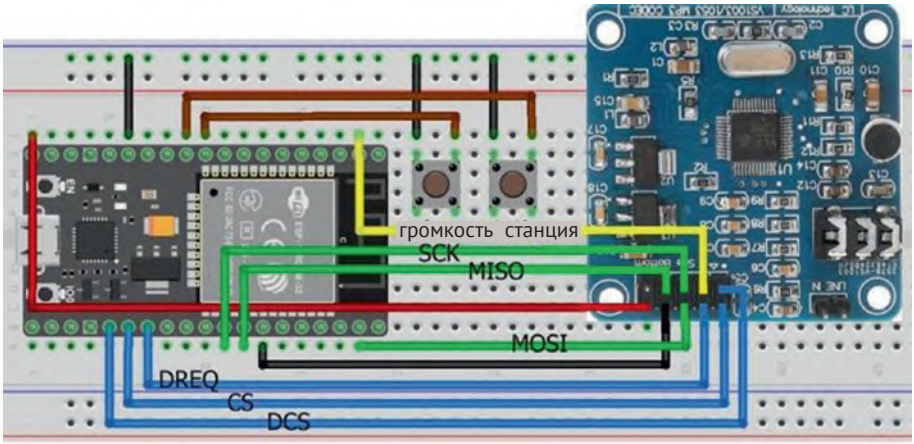


Рисунок 1-3. Интернет-радио с переключателями громкости и станций на основе платы ESP32

Выбор и отображение станции

В листинге 1-1 выбор станции и регулировка громкости активируются кнопками, при этом информация о станции и громкости отображается через монитор последовательного порта среды Arduino. Для портативного интернет-радио информация о станции и громкости отображается на TFT ЖК-дисплее ST7735, и выбор станции или регулировка громкости осуществляется с помощью поворотного энкодера (см. рис. 1-4 и 1-5 и подключения в табл. 1-2). Обратите внимание, что как поворотный энкодер, так и ЖК-экран ST7735 подключены к напряжению 3,3 В, только аудиодекодер VS1053 подключен к выводу 5V. Микроконтроллер ESP32 взаимодействует как с аудиодекодером VS1053, так и с ЖК-экраном ST7735 с помощью SPI, поэтому микроконтроллер имеет одни и те же подключения MOSI и SCK к аудиодекодеру и экрану, но соединения CS зависят от конкретного устройства[12].

[12] Расшифровка названий соединений SPI: MOSI – Master Output, Slave Input (выход ведущего, вход ведомого); SCK – Serial Clock (тактовый сигнал); MISO – Master Input, Slave Output (вход ведущего, выход ведомого); CS – Chip Select (выбор кристалла, т. е. микросхемы). Вывод CS обычно имеет отрицательную логику (активируется низким уровнем), поэтому часто указывается с надстрочной чертой. – *Прим. перев.*

Рисунок 1-4. Скриншоты дисплея интернет-радио

В скетче используется библиотека *ESP32 vs1053_ext* от Wolle, которая загружается в виде zip-файла с *github.com/schreibfaul1/ESP32-vs1053_ext*. Библиотека *ESP32 vs1053_ext* предназначена для микроконтроллера ESP32, в то время как библиотека *VS1053* Эда Смолленбурга (Ed Smallenburg) и Джеймса Колиза (James Coliz) совместима как с микроконтроллерами ESP8266, так и с ESP32. Библиотека *ESP32 vs1053_ext* предоставляет информацию о станции и треке, в том числе название трека. Инструкция для подключения к серверу интернет-радиостанции выглядит как `connecttohost("host:port/stream")`, например `connecttohost("1940sradio1.co.uk:8100/stream/1/")`. Номер порта требуется только в том случае, если он не равен значению по умолчанию 80. Функции `vs1053_showstation`, `vs1053_icyurl`, `vs1053_bitrate` и `vs1053_ showstreamtitle` содержат название интернет-радиостанции, URL-адрес домашней страницы, скорость передачи данных и название трека. При передаче нового трека автоматически обновляется функция `vs1053_showstreamtitle`. Переменная громкости `volume` имеет 22 уровня 0, 50, 60, 65, 70, 75, 80, 82…90, 91…100 %; при этом значению 10 соответствует уровень громкости 88 %, тогда как значение 0 соответствует громкости 0 %.

В листинге 1-2 демонстрируются выходные данные функций библиотеки *ESP32 vs1053_ext*, которые используются в листинге 1-3 для отображения информации об интернет-радиостанции и треке.

Листинг 1-2. Библиотечные функции ESP32 vs1053_ext

```
#include <vs1053_ext.h>                 // include ESP32 VS1053_ext lib
#include <WiFi.h>                       // include Wi-Fi library
int CS = 0;
int DCS = 2;                            // define VS1053 decoder pins
int DREQ = 4;
VS1053 decoder(CS, DCS, DREQ);          // associate decoder with VS1053
char ssid[] = "xxxx";                   // change xxxx to Wi-Fi ssid
char password[] = "xxxx";               // change xxxx to Wi-Fi password
int volume = 10;                        // volume level

void setup()
{
   Serial.begin(115200);                // Serial Monitor baud rate
```

```
    SPI.begin();                        // initialise SPI bus
    WiFi.begin(ssid, password);         // initialise Wi-Fi
    while (WiFi.status() != WL_CONNECTED) delay(500);
    decoder.begin();                    // initialise VS0153 decoder
    decoder.setVolume(volume);          // set decoder volume level
    decoder.connecttohost
    ("media-ice.musicradio.com:80/ClassicFMMP3");
}

void loop()
{
    decoder.loop();
}
void vs1053_showstation(const char * info)
{                                       // display radio station name
    Serial.print("Station: ");
    Serial.println(info);
}
void vs1053_bitrate(const char * info)
{                                       // display streaming bit rate
    Serial.print("Bit rate: ");
    Serial.println(String(info)+"kBit/s");
}
void vs1053_icyurl(const char * info)
{                                       // display radio station URL
    Serial.print("Homepage: ");
    Serial.println(info);
}
void vs1053_showstreamtitle(const char * info)
{                                       // title of streamed track
    Serial.print("Stream title: ");
    Serial.println(info);
}
```

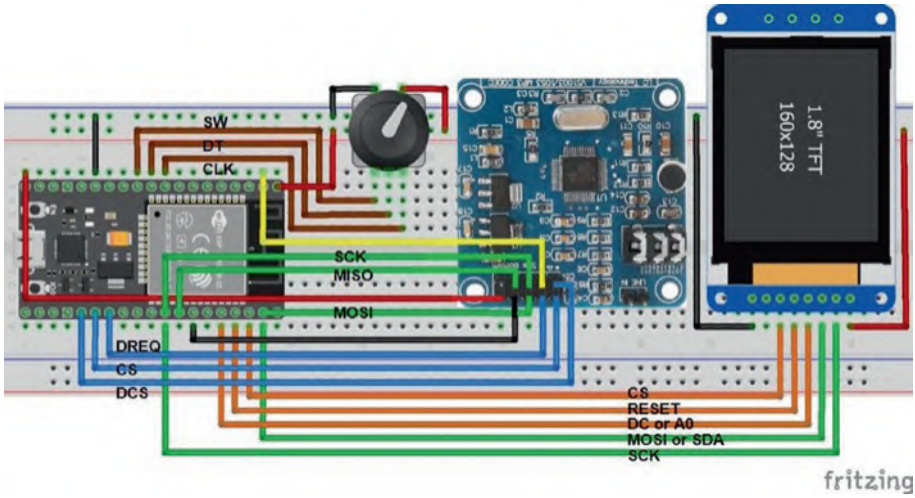


Рисунок 1-5. Интернет-радио с экраном, поворотным энкодером и платой ESP32

Таблица 1-2. Интернет-радио с экраном, поворотным энкодером и платой ESP32

Компонент	Подключено к EPS32
VS1053 audio decoder	См. табл. 1-1
Поворотный энкодер CLK	GPIO 25
Поворотный энкодер DT	GPIO 26
Поворотный энкодер SW	GPIO 27
Поворотный энкодер VCC	3V3
Поворотный энкодер GND	GND
ST7735 TFT LCD GND	GND
ST7735 TFT LCD CS	GPIO 22
ST7735 TFT LCD RESET	GPIO 1
ST7735 TFT LCD DC или A0	GPIO 3
ST7735 TFT LCD SDA	GPIO 23
ST7735 TFT LCD SCK	GPIO 18
ST7735 TFT LCD LED	3V3

Скетч для портативного интернет-радио приведен в листинге 1-3. При однократном нажатии переключателя, связанного с валом поворотного энкодера, отображается меню доступных радиостанций с регулировкой громкости в качестве первого пункта меню. Поворот энкодера перемещает меню радиостанций вверх или вниз по экрану ST7735. Станция в середине экрана, выделенная красным цветом, выбирается повторным нажатием на вал; и HTTP-запрос отправляется на сервер интернет-радиостанции для получения аудиоданных. Когда в меню выбирается *Volume*, отображается текущий уровень громкости, и поворот энкодера уменьшает или увеличивает этот уровень, который затем фиксируется нажатием на вал. На ЖК-экране ST7735 обновляется информация о текущей радиостанции и отображается обновленный уровень громкости, но меню радиостанции по-прежнему показывает текущую радиостанцию.

Скетч в листинге 1-3 состоит из нескольких функций для разделения инструкций. Длинный первый раздел скетча определяет библиотеки, URL-адреса интернет-радиостанций, номера выводов для аудиодекодера VS1053, ЖК-экрана ST7735, а также параметры поворотного энкодера, включая начальные значения для станции и уровня громкости. Библиотека *Adafruit ST7735* доступна в среде Arduino IDE. Библиотеки *ESP32 vs1053_ext* и *Adafruit ST7735* ссылаются на библиотеки *SPI* и *Adafruit GFX*, поэтому инструкции `#include <SPI. h>` и `#include <Adafruit_GFX.h>` не требуются. Функция `setup` устанавливает соединение Wi-Fi, инициализирует аудиодекодер VS1053 и ЖК-экран ST7735, подключает внутренние подтягивающие резисторы к контактам энкодера и определяет прерывания для них. Направление и количество оборотов поворотного энкодера определяются прерыванием по изменению уровня, как описано в главе 19 («Поворотный энкодер»).

При нажатии на вал замыкаются контакты переключателя энкодера, функция `loop` вызывает функцию `screen` для отображения меню громкости и радиостанции, функцию `readMenu` для определения выбранной радиостанции или функцию `readValue` для получения нового уровня громкости, а затем функцию `radio`. Функция `radio` либо подключает устройство к серверу выбранной радиостанции, либо изменяет громкость на аудиодекодере VS1053. Функции `readMenu` и `readValue` определяют номер выбранной строки меню, представляющего собой список станций, и выбранный уровень громкости, когда энкодер поворачивается. Функция `vs1053_icyurl` получает строку, начинающуюся с *https://*, за которой следует URL-адрес станции, и извлекает подстроку, начинающуюся через две позиции после расположения первой обратной косой черты. Функции `vs1053_showstation` и `vs1053_showstreamtitle` получают название радиостанции и название трека, а затем вызывают функцию `showStation`, которая отображает название станции, название трека, значение громкости и информацию об URL-адресе станции на ЖК-дисплее ST7735. Некоторый текст, такой как название станции или заголовок транслируемого трека, окажется длиннее ширины экрана ST7735, поэтому функция `lines` разбивает название или заголовок станции на подстроки размером с экран. Функции `encoder` и `swPress` подсчитывают направление и количество оборотов поворотного энкодера, а также количество нажатий переключателя энкодера.

Листинг 1-3. Интернет-радио с экраном и поворотным энкодером и платой ESP32

```
#include <vs1053_ext.h>              // include ESP32 VS1053_ext,
#include <WiFi.h>                    // WiFi and
#include <Adafruit_ST7735.h> // Adafruit_ST7735 libraries
int CS = 0;
int DCS = 2;                         // define VS1053 decoder pins
int DREQ = 4;
VS1053 decoder(CS, DCS, DREQ);       // associate decoder with VS1053
char ssid[] = "xxxx";                // change xxxx to Wi-Fi ssid
char password[] = "xxxx";            // change xxxx to Wi-Fi password
const int maxStation = 11;           // number of radio stations
String stationName[] = {"Volume",    // first item on menu
"1940 UK", "Berlin", "Bayern3", "Classic", "BBC4",
"Vermont", "Ketchikan", "Kathmandu", "Ithaca", "Trondeim",
"Virgin"};
char * URL[maxStation] = {           // radio station URLs
"1940sradio1.co.uk:8100/1",
"streambbr.ir-media-tec.com/berlin/mp3-128/vtuner_web_mp3/",
"streams.br.de/bayern3_2.m3u",
"media-ice.musicradio.com:80/ClassicFMMP3",
"bbcmedia.ic.llnwd.net/stream/bbcmedia_radio4fm_mf_q",
"vpr.streamguys.net/vpr64.mp3",
"96.31.83.94:8082/stream",
"streaming.softnep.net:8037/stream.nsv",
"17993.live.streamtheworld.com/WITHFM.mp3",
"stream.radiometro.no/metro128.mp3",
"radio.virginradio.co.uk/stream"
};
int TFT_CS = 22;
int DCpin = 3;                       // define ST7735 TFT screen pins
```

```
int RSTpin = 1;              // associate tft with Adafruit ST7735
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, DCpin, RSTpin);
int CLKpin = 25;
int DTpin = 26;
int SWpin = 27;              // define rotary encoder pins
int oldRow = 0;
int newRow = 1;
int menuItem, val, upLimit;
int displayVol[] =           // define volume values for 22 levels
{0,50,60,65,70,75,80,82,84,86,88,90,91,92,93,94,95,96,97,98,
99,100};
int volume = 0;
int newVolume = 10;          // volume level at start up
int station = 0;
int newStation = 3;          // station level at start up
int textlen, textrows;
String showstatn, showtitle, showurl, text, text1, text2;
volatile int change = 0;     // rotary encoder variables
volatile int pressed = 0;
volatile int vals[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,0,-1,0};
volatile int score = 0;
volatile int oldState = 0;
volatile int turn;
void setup()
{
  SPI.begin();                        // initialise SPI bus
  WiFi.begin(ssid, password);         // initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  decoder.begin();                    // initialise VS0153 decoder
  decoder.setVolume(volume);          // set decoder volume level
  tft.initR(INITR_BLACKTAB);          // initialise screen
  tft.fillScreen(ST7735_BLACK);       // clear screen
  tft.setRotation(1);                 // orientate ST7735 screen
  tft.setTextSize(2);                 // set screen text size
  tft.drawRect(0,0,158,126,ST7735_WHITE); // draw white frame line
  tft.drawRect(2,2,154,122,ST7735_RED); // and second frame line
  pinMode(CLKpin, INPUT_PULLUP);
  pinMode(DTpin, INPUT_PULLUP);       // rotary encoder uses
  pinMode(SWpin, INPUT_PULLUP);       // internal pull-up resistors
  attachInterrupt(CLKpin, encoder, CHANGE);
  attachInterrupt(DTpin, encoder, CHANGE); // attach rotary encoder interrupts
  attachInterrupt(SWpin, swPress, CHANGE);
}
void loop()
{
  if(pressed == 1)                    // switch pin pressed first time
  {                                   // to change station or volume
    clearScreen();                    // call clearScreen function
    screen();                         // call screen function
    menuItem = readMenu(maxStation); // selected row in menu
  }
  else if (pressed == 2)              // switch pin pressed second time
  {                                   // to select station
    if(menuItem > 0)                  // station selected
    {
      newStation = menuItem-1;        // selected station in menu
```

```
        clearScreen();                // call clearScreen function
        showStation(volume, showstatn, showtitle);
                                      // call showStation function
        if(newStation == station) showStation(volume, showstatn,
        showtitle);
      }                               // volume change selected
      else if(menuItem== 0) newVolume = readValue("volume: ",
      volume, 21, 1);
      pressed = 0;                    // reset variable
    }
    else if(pressed > 2) pressed = 0;  // volume changed
    radio();                          // call radio function
}
void radio()                          // function to connect to
{                                     // selected radio station server
    if(station != newStation)         // new station selected
    {
      clearScreen();                  // call clearScreen function
      station = newStation;
      showurl = "";
      decoder.connecttohost(URL[station]);
    }                                 // connect to radio station server
    if(volume != newVolume)           // new volume level selected
    {
      volume = newVolume;
      newRow = station+1;             // retain station number on menu
      decoder.setVolume(volume);      // update VS1053 volume
      clearScreen();                  // call clearScreen function
      showStation(volume, showstatn, showtitle);
    }                                 // call showStation function
    decoder.loop();
}
    int readMenu (int rows)           // function to obtain station
    {                                 // number on menu
      while(pressed < 2)              // while station not selected
      {
        if(change != 0)               // rotary encoder turned
        {
          newRow = oldRow + change;   // retain row number on menu
          newRow = constrain(newRow, 0, rows);
          clearScreen();              // call clearScreen function
          screen();                   // call screen function
          oldRow = newRow;
          change = 0;
        }
        delay(10);
      }
      return newRow;                  // return row number on menu
}
                                      // function to obtain volume level
int readValue(String text, int current, int upLimit, int gain)
{
    val = current;                    // current volume level
    clearScreen();                    // call clearScreen function
    tft.setTextColor(ST7735_WHITE);
    tft.setTextSize(2);
```

```
   tft.setCursor(10, 50);
   tft.print(text);                    // display text and
   tft.print(displayVol[val]);         // current volume value
   while(pressed < 3)                  // while switch pin is not pressed
   {
      if(change != 0)                  // rotary encoded turned
      {
         val = val + change * gain; // increment volume level
         val = constrain(val, 0, upLimit);
                                       // constrain volume level
         clearScreen();                // call clearScreen function
         tft.setCursor(10, 50);
         tft.print(text);              // display text and
         tft.print(displayVol[val]);   // new volume value
         change = 0;
      }
      delay(10);
   }
   return val;                         // return new volume level
}
void vs1053_showstation(const char * info)
{                                      // function to obtain station name
   showstatn = String(info);           // station name
   showtitle = "";
   if(showstatn == "No Name") showstatn = stationName[station+1];
   clearScreen();
   showStation(volume, showstatn, showtitle);
}                                      // call showStation function
void vs1053_showstreamtitle(const char * info)
{                                      // function to obtain streamed title
   showtitle = String(info);
   clearScreen();
   showStation(volume, showstatn, showtitle);
}
void vs1053_icyurl(const char * info)
{                                      // function to obtain station URL
   showurl = String(info);
   int i = showurl.indexOf("/");       // position of first / in string
   showurl = showurl.substring(i+2);  // station URL as substring
   clearScreen();
   showStation(volume, showstatn, showtitle);
}
void showStation(int volume, String showstatn, String showtitle)
{                             // function to display station name, streamed title
                              // and station URL on screen
   tft.setTextColor(ST7735_GREEN);
   tft.setTextSize(1);
   lines(showstatn, 10);      // lines function to display station
   tft.setTextColor(ST7735_YELLOW);
   lines(showtitle, 40);      // lines function to display title
   tft.setTextColor(ST7735_GREEN);
   tft.setCursor(80, 100);    // display volume value
   tft.print("volume: ");tft.print(displayVol[volume]);
   tft.setCursor(5, 110);
   tft.print(showurl);        // display URL
}
void lines(String text, int line)
```

```
{                    // function to split string into screen sized substrings
  textlen = text.length();           // get string length
  textrows = 1+textlen/23;           // required number of screen rows
  for(int i=0; i<textrows; i++)
  {
    tft.setCursor(10, line + i*10); // move cursor to next row
    tft.println(text.substring(i*23, (i+1)*23));
  }                                  // display substring
}
void screen()                        // function to display station menu
{
  tft.setTextSize(2);
  tft.setTextColor(ST7735_RED);      // selected station in RED
  tft.setCursor(20, 55);
  tft.print
  (stationName[newRow]);             // display station name
  tft.setTextSize(1);
  tft.setTextColor(ST7735_WHITE);    // all other stations in WHITE
  for (int i=1; i<4; i++)            // display other station names
  {
    tft.setCursor(30, 50 - i*12);    // above selected station
    if(newRow-i >=0) tft.print(stationName[newRow-i]);
    tft.setCursor(30, 65 + i*12);    // below selected station
    if(newRow+i < maxStation+1) tft.print(stationName
    [newRow+i]);
  }
}
void clearScreen()                   // function to clear screen
{                                    // by displaying a BLACK rectangle
  tft.fillRect(3,3,152,120,ST7735_BLACK);
}
IRAM_ATTR void encoder()             // function to count rotary
{ // encoder turns
  int newState = (oldState<<2)+(digitalRead(CLKpin)<<1)
  +digitalRead(DTpin);
  score = score + vals[newState];    // allocate score from array
  oldState = newState % 4;           // remainder to leave new CLK and DT
  if(score == 2 || score == -2)      // 2 steps for complete rotation
  {
    change = score/2;                // unit change per two steps
    score = 0;                       // reset score
  }
}
IRAM_ATTR void swPress()             // function to count switch presses
{                                    // pressed = 1, 2, 3 to change station,
                                     // station selected, volume changed
  if(digitalRead(SWpin) == HIGH) pressed = pressed + 1;
}
```

Простейшее интернет-радио

Скетч в листинге 1-3, включающий ЖК-экран ST7735 для отображения названия и URL-адреса радиостанции, названия потокового трека и уровня громкости с поворотным кодером для выбора станции и регулировки громкости, состоит из 250 строк кода. Напротив, в листинге 1-4 для минимальной настройки

интернет-радио на одну радиостанцию с одним значением громкости содержится всего 21 строка кода. Просто измените URL-адрес интернет-радиостанции в скетче на требуемый URL-адрес!

Листинг 1-4. Простейшее интернет-радио

```
#include <vs1053_ext.h>                // include ESP32 VS1053_ext
#include <WiFi.h>                      // and WiFi libraries
int CS = 0;
int DCS = 2;                           // define VS1053 decoder pins
int DREQ = 4;
VS1053 decoder(CS, DCS, DREQ);         // associate decoder with VS1053
char ssid[] = "xxxx";                  // change xxxx to Wi-Fi ssid
char password[] = "xxxx";              // change xxxx to Wi-Fi password
void setup()
{
  SPI.begin();                         // initialise SPI bus
  WiFi.begin(ssid, password);          // initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  decoder.begin();                     // initialise VS0153 decoder
  decoder.setVolume(10);               // pre-set decoder volume level
  decoder.connecttohost
  ("media-ice.musicradio.com:80/ClassicFMMP3");
}                                     // connect to pre-set radio station server
void loop()
{
  decoder.loop();
}
```

Итоги

Интернет-радио было построено с использованием аудиодекодера VS1053 и микроконтроллера ESP8266 или ESP32, с возможностью выбора радиостанции и регулировки громкости с помощью кнопок. Портативное интернет-радио состояло из аудиодекодера VS1053, платы ESP32 и ЖК-дисплея ST7735 для отображения сведений о радиостанции, названия транслируемого трека и уровня громкости с поворотным энкодером для управления выбором станции и громкостью. Скетч для минимального интернет-радио состоял всего из 21 строки кода.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN D1 mini (WeMos) или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU
- Модуль аудиодекодера VS1053
- Миниатюрные аудиоколонки
- Тактовые кнопки: 2 шт.
- Поворотный энкодер KY-040
- Дисплей TFT LCD ST7735, 1,8 дюйма

Глава 2

Сетевая фотокамера

Модуль ESP32-CAM основан на микроконтроллере ESP32-S и включает в себя 2-мегапиксельную камеру OV2640 и слот для карт памяти формата micro-SD. Файлы изображений в формате JPEG сохраняются на карте micro-SD, загружаются на веб-страницу или передаются потоком на веб-страницу, отображаемую на компьютере, планшете Android или мобильном телефоне.

Модуль ESP32-CAM (см. рис. 2-1) содержит выводы последовательного порта TX и RX, шесть контактов, связанных с картой micro-SD, COB-светодиод[13], который кратковременно включается при съемке, и красный сигнальный светодиод, который включается при подаче низкого уровня. Доступ к модулю осуществляется с помощью выводов GPIO[14] 4 и 33 соответственно. COB-светодиод включает в себя множество светодиодных кристаллов, соединенных непосредственно с подложкой, образующих единый модуль. Модуль ESP32-CAM имеет три вывода GND, напряжение 3,3 В и входной вывод 5V, а вывод VCC выводит 3,3 В или 5 В в зависимости от установленной перемычки. Вывод GPIO 0 определяет режим ESP32-CAM микроконтроллера; при загрузке скетча (*flashing mode*) этот вывод, имеющий встроенный подтягивающий резистор, подключается к GND. Выводы GPIO 2, 4, 12, 13, 14 и 15 связаны с функциональностью карты micro-SD. Когда карта micro-SD не используется, выводы GPIO доступны в качестве выходных контактов. Расположение выводов модуля ESP32-CAM показано на рис. 2-1, где *Rup* указывает на вывод встроенного подтягивающего резистора.

---

[13] COB-светодиод – мощный осветительный светодиод, изготовленный по технологии Chip On Board (букв. *кристалл на плате*). COB-светодиоды состоят из множества мелких светодиодов, установленных на одну теплоотводящую подложку. – *Прим. перев.*

[14] Часто упоминаемое в тексте сокращение GPIO означает General-Purpose Input-Output (универсальный вход-выход) – так обозначаются выводы микроконтроллера общего назначения, не привязанные к конкретной функции. – *Прим. перев.*

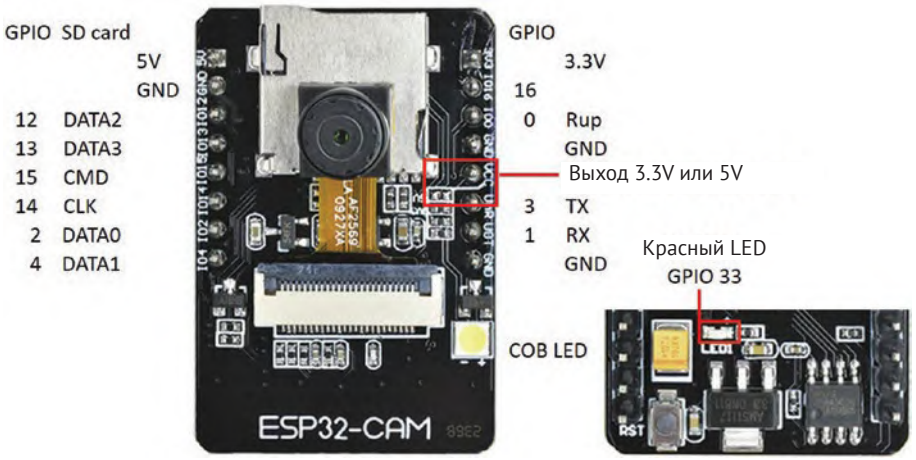
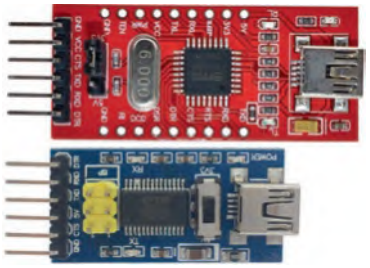


Рисунок 2-1. Выводы модуля ESP32-CAM



Модуль ESP32-CAM не имеет USB-разъема, и модуль подключается к компьютеру или ноутбуку с помощью интерфейса USB-to-UART, например такого, как модуль преобразователя последовательного интерфейса FTDI FT232RL. Напряжение коммуникационных линий интерфейса USB для преобразования в напряжение UART должно быть установлено на уровне 3,3 В, при этом выводы RX и TX модуля преобразователя USB-UART подключены к выводам TX и RX модуля ESP32-CAM соответственно[15] (см. рис. 2-2 и соединения в табл. 2-1). Вывод модуля USB-UART 5V подключается к выводу 5V модуля ESP32-CAM. Подробная информация об установке драйвера CP210x модуля USB-UART Bridge для микроконтроллера ESP32 с дополнительными URL-адресами менеджера плат и библиотеками для ESP32 приведена в главе 21 («Микроконтроллеры»). Для подключения модуля камеры к модулю ESP32-CAM необходимо поднять черный язычок на модуле ESP32-CAM, вставить модуль камеры в разъем и опустить черный язычок.

[15] Следует подчеркнуть, что сигнальные выводы интерфейса UART (Universal asynchronous receiver/transmitter, *универсальный асинхронный приемопередатчик*) всегда соединяются перекрестно: RX (RxD) – TX (TxD) и TX (TxD) – RX (RxD). В этом его отличие от других последовательных интерфейсов, например SPI и I2C, где одноименные выводы соединяются между собой. – *Прим. перев.*

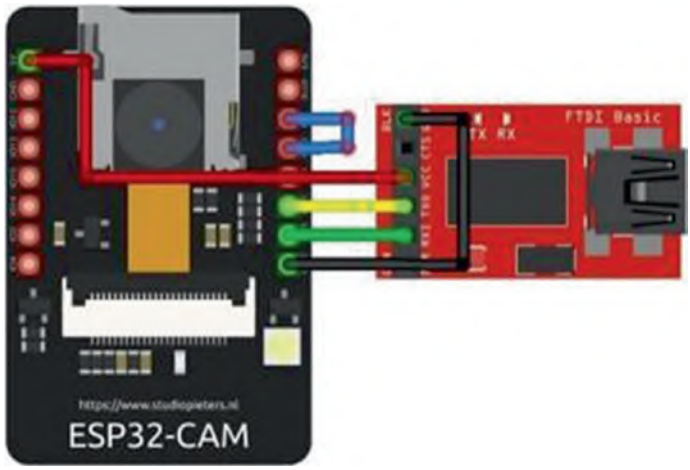


Рисунок 2-2. Подключение модуля USB-UART к модулю ESP32-CAM

Таблица 2-1. Подключение модуля USB-UART к модулю ESP32-CAM

Компонент	Модуль ESP32-CAM
USB-UART RxD	UOT (вывод TX)
USB-UART TxD	UOR (вывод RX)
USB-UART VCC	5V
USB-UART GND	GND (а также GPIO 0, см. в тексте)

В среде IDE Arduino в раскрывающемся списке *Инструменты* ➤ *Плата* (*Tools* ➤ *Board*) выберите *ESP32 Wrover Module*; в разделе *Инструменты* ➤ *Схема разделов* (*Tools* ➤ *Partition Scheme*) выберите *Huge APP (3MB no OTA/1MB SPIFFS)*; в меню *Инструменты* ➤ *Порт* (*Tools* ➤ *Port*) выберите соответствующий COM-порт. Перед загрузкой скетча в модуль ESP32-CAM вывод GPIO 0 подключается к выводу GND модуля[16], а затем на модуле нажимается кнопка *Reset*. После загрузки скетча вывод GPIO 0 модуля ESP32-CAM отсоединяется от вывода GND модуля, а затем опять нажимается кнопка *Reset* модуля.

С помощью скетча, приведенного в листинге 2-1, модуль ESP32-CAM делает снимок каждые две секунды, и результирующий файл JPEG сохраняется на карте micro-SD. Количество сделанных фотографий хранится в EEPROM для последовательной нумерации имен файлов JPEG как *pictureN.jpg* (где N – номер фотографии). При повторном запуске скетча нумерация файлов изображений производится с последнего сохраненного файла JPEG, а не с */picture0.jpg*, что перезаписало бы существующие файлы, уже хранящиеся на карте micro-SD. Сохранение данных в EEPROM описано в главе 20 («OTA и сохранение данных в EEPROM, SPIFFS и Microsoft Excel»). Усилие при нажатии кнопки *Reset* модуля ESP32-CAM после загрузки скетча вызывает дрожание камеры, поэтому двух-

[16] Вывод GPIO 0 на плате ESP32-CAM обозначен как IO0. Временное соединение его с GND показано на рис. 2-2 синим цветом. – *Прим. перев.*

секундная задержка дает модулю камеры время на стабилизацию. Количество фотографий, которые нужно сделать, вводится через монитор последовательного порта среды Arduino, и после инициализации камеры и карты micro-SD камера делает необходимое количество снимков. Встроенная библиотека SD-MMC Arduino IDE использует более быструю аппаратную шину ESP32 SDMMC вместо SPI, как это делается в стандартной библиотеке SD. Обратите внимание, что модуль ESP32-CAM поддерживает скорость передачи данных 115 200 бод[17].

Файлы JPEG в формате UXGA с разрешением 1200×1600 пикселей имеют средний размер 100 кБ; а карта micro-SD объемом 4 ГБ в формате FAT32 хранит тысячи изображений. В этой главе использовалась карта micro-SD объемом 16 ГБ. Замедленная съемка возможна с помощью модуля ESP32-CAM путем сохранения изображений в формате JPEG на карте micro-SD с интервалами 2–30 с между фотографиями. В листинге 2-1 установка переменной `maxPhoto` равной 3000 позволит генерировать количество изображений, достаточное для двухминутного видео с частотой кадров 25 кадров в секунду, для которого требуется всего 300 МБ памяти на карте micro-SD.

Инструкции по настройке камеры ESP32-CAM включены во вкладку *config_pins.h*, а не в основной скетч, чтобы упростить анализ программы. Дополнительная вкладка создается в IDE Arduino, если щелкнуть треугольник под кнопкой монитора последовательного порта в правой части окна IDE и выбрать из выпадающего меню пункт *Новая вкладка* (*New Tab*). Новая вкладка озаглавлена *config_pins.h*.

Скетч в листинге 2-1 загружает библиотеки для ESP32-CAM на вкладке *config_pins.h* (см. листинг 2-2), включая инструкции по настройке камеры ESP32-CAM с помощью функции `configCamera`. Карта micro-SD инициализируется с помощью функции `initSDcard`, которая определяет тип SD-карты. По истечении требуемого интервала времени между фотографиями вызывается функция `takePhoto`. Файл изображения в формате JPEG сохраняется на карту micro-SD, имя файла увеличивается после каждой фотографии, а номер изображения записывается в EEPROM. Функция `takePhoto` использует символ амперсанда (&) и звездочки (*) для указания на адрес переменной в памяти. В главе 14 («Обмен данными через ESP-NOW и LoRa») в листинге 14-3 показано использование указателя адреса памяти.

Листинг 2-1. Фотосъемка и сохранение на карту micro-SD

```
#include <esp_camera.h>       // include esp_camera library
#include <SD_MMC.h>           // include SD_MMC library
#include <EEPROM.h>           // include EEPROM
#include "config_pins.h"      // configure instructions tab
uint8_t SDtype;
```

[17] Следует отметить, что указание скорости последовательного порта UART и его производных (RS-232, COM-порт) в *бодах* (посылках, или символах в секунду) – принятая в среде Arduino, но ошибочная практика. Скорость обмена по UART измеряется в битах в секунду. В данном случае эти единицы измерения совпадают (одна посылка равна одному биту), но необходимо иметь в виду, что не для всех устройств последовательной связи это справедливо: так, для модемов с различными видами модуляции одна посылка (символ) может содержать несколько (до 16) битов информации. – *Прим. перев.*

```
int SDpics;                   // number of pictures on SD card
int maxPhoto = 0;             // maximum number of photos
int Nphoto = 0;               // number of photos taken
int photoTime = 2000;         // delay (ms) between photos
String filename;
unsigned long nowTime, lastTime = 0;
void setup()
{
  Serial.begin(115200);       // baud rate for Serial Monitor
  Serial.println("\n\nenter number of required photos");
  Serial.println("\n\nsettling down for 2s");
                              // time to settle vibration
  delay(2000);
  Serial.println("initialising camera, then take photos");
  configCamera();             // functions to configure camera
  initSDcard();               // and to initialise micro-SD card
  EEPROM.begin(1);            // EEPROM with one record
  SDpics = EEPROM.read(0);    // number of saved pictures
}
void loop()
{
  while (Serial.available()>0)
  {                           // maximum photo number
    maxPhoto = Serial.parseInt(); // parsed from Serial buffer
    Nphoto = 0;
  }                           // if photo number < maximum
                              // photo number
  nowTime = millis();         // take photo after photoTime ms
  if((nowTime - lastTime > photoTime) && (Nphoto < maxPhoto))
  {
    Nphoto++;                 // increment photo number
    takePhoto();              // call function to take photo
    lastTime = millis();      // update time of photo
  }
}
void initSDcard()             // function to initialise SD card
{
  if(!SD_MMC.begin())         // check SD card in position
  {
    Serial.println("error loading SD card");
    return;
  }
  SDtype = SD_MMC.cardType(); // obtain SD card type
  if(SDtype == CARD_NONE)
  {
    Serial.println("insert SD Card");
    return;
  }
  Serial.print("SD card type: ");
  if(SDtype == CARD_MMC) Serial.println("MMC");
  else if(SDtype == CARD_SD) Serial.println("SDSC");
  else if(SDtype == CARD_SDHC) Serial.println("SDHC");
  else Serial.println("UNKNOWN");
}
```

```
void takePhoto()                        // function to take and save photo
{
   camera_fb_t * frame = NULL;          // associate fb with esp_camera
   frame = esp_camera_fb_get();         // take photo with camera
   if(!frame)
   {
      Serial.println("photo capture error");
      return;
   }
   SDpics ++;                           // increase picture number
   filename = "/picture" + String(SDpics) +".jpg";
                                        // generate JPEG filename
   fs::FS & fs = SD_MMC;
   File file = fs.open(filename.c_str(), FILE_WRITE);
                                        // access SD card
   if(!file) Serial.println("file save error");
   else
   {
      file.write(frame->buf, frame->len);       // save file to SD card
      Serial.print("Picture filename: ");
      Serial.println(filename);
      EEPROM.write(0, SDpics);                  // update EEPROM
      EEPROM.commit();                          // with picture number
   }
   file.close();                        // close file on SD card
   esp_camera_fb_return(frame);         // return frame buffer to driver for
}                                       // reuse
```

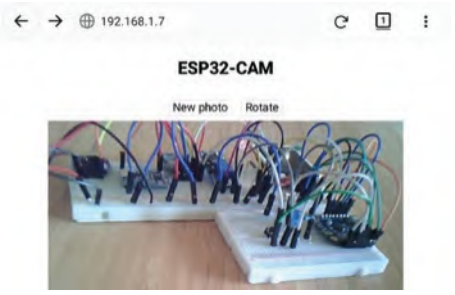
Инструкции по настройке камеры ESP32-CAM включены во вкладку *config_pins.h*, а не в основной скетч (см. листинг 2-2). Формат пикселей JPEG выбирается из доступных опций YUV422, Grayscale, RGB565 и JPEG. Если микроконтроллер не поддерживает псевдостатическую оперативную память PSRAM, представляющую собой динамическую оперативную память (RAM), ведущую себя как статическая память (SRAM), будет необходимо установить меньший размер кадра изображения, более низкое качество JPEG со значением от 0 до 63 и меньшее количество кадров.

Листинг 2-2. Вкладка с инструкциями по настройке камеры

```
camera_config_t config;                 // store camera configuration parameters
void configCamera()
{
   config.ledc_channel = LEDC_CHANNEL_0;
   config.ledc_timer = LEDC_TIMER_0;
   config.pin_d0 = 5;
   config.pin_d1 = 18;
   config.pin_d2 = 19;                  // GPIO pin numbers
   config.pin_d3 = 21;
   config.pin_d4 = 36;
   config.pin_d5 = 39;
   config.pin_d6 = 34;
   config.pin_d7 = 35;
   config.pin_xclk = 0;
   config.pin_pclk = 22;
   config.pin_vsync = 25;
```

```
    config.pin_href = 23;
    config.pin_sscb_sda = 26;
    config.pin_sscb_scl = 27;
    config.pin_pwdn = 32;
    config.pin_reset = -1;
    config.xclk_freq_hz = 20000000;             // clock speed of 20MHz
    config.pixel_format = PIXFORMAT_JPEG;       // JPEG file format
    config.frame_size = FRAMESIZE_SVGA;         // 800x600 pixels
    config.jpeg_quality = 10;                   // image quality index
    config.fb_count = 1;                        // frame buffer count
    esp_err_t err = esp_camera_init(&config);   // initialize camera
    if (err != ESP_OK)
    {
       Serial.print("Camera initialise failed with error");
       Serial.println(err);
       return;
    }
}
```

Загрузка изображений на веб-страницу



Фотография, сделанная модулем ESP32-CAM, загружается на веб-страницу с помощью HTTP-запроса. Как только будет установлено соединение Wi-Fi и через сеть (WLAN) загружена веб-страница, нажатие кнопки *New photo* вызывает функцию *newPhoto*, которая инициирует HTTP-запрос клиента с URL-адресом */photoURL* серверной камеры, чтобы она сделала снимок и отправила JPEG-изображение клиенту. Веб-страница перезагружается с помощью инструкции `location.reload()` обновления веб-страницы с новой фотографией. Щелчок по кнопке *Rotate* поворачивает изображение на веб-странице на 90°. Повернутое изображение закрывает кнопки, поэтому положение кнопок переопределяется, когда меняется ориентация картинки с альбомной на портретную (поворот на 90° или 270°). Загрузка изображения ESP32-CAM непосредственно на веб-страницу основана на методе Нино Сантоса (Nuno Santos, techtutorialsx.com).

Скетч в листинге 2-3 включает HTTP-запросы GET с соответствующими URL-адресами и ссылается на вкладку *buildpage.h*, содержащую HTML-код для веб-страницы (см. листинг 2-4). Добавление *P* через подстрочную черту в запросе `request->send_P` указывает на то, что изображение в формате JPG хранится в PROGMEM[18], поскольку программная флеш-память микроконтроллера имеет бóльший объем, чем оперативная память.

Размер фотографии отображается в мониторе последовательного порта только для информации. Требуются библиотеки *ESPAsyncWebServer* и *AsyncTCP* Хрис-

[18] PROGMEM – инструкция компилятору в среде Arduino для загрузки данных во флеш-память программ вместо оперативной памяти SRAM. – *Прим. перев.*

то Гочкова (Hristo Gochkov). Zip-файлы, содержащие библиотеки, загружаются с github.com/me-no-dev/ESPAsyncWebServer и github.com/me-no-dev/AsyncTCP соответственно. Библиотека *ESPAsyncWebServer* ссылается на библиотеки *AsyncTCP* и *Wi-Fi*, поэтому инструкции *#include <AsyncTCP.h>* и *#include <WiFi.h>* не требуются. Библиотека Wi-Fi включается в Arduino IDE при установке драйвера ESP32. В содержимом вкладки *config_pins.h* (листинг 2-2) нет никаких изменений.

Листинг 2-3. Фотографирование и загрузка на веб-страницу

```
#include <esp_camera.h>              // include esp_camera,
#include <ESPAsyncWebServer.h>       // ESPAsyncWebServer libraries
AsyncWebServer server(80);           // associate server with library
#include "config_pins.h"             // configure instructions tab
#include "buildpage.h"               // HTML code for webpage
char ssid[] = "xxxx";                // change xxxx to Wi-Fi ssid
char password[] = "xxxx";            // change xxxx to Wi-Fi password
String pSize;                        // photo size (bytes)
void setup()
{
   Serial.begin(115200);             // Serial Monitor baud rate
   Serial.println("\n\nsettling down for 2s");
                                     // time to settle vibration
   delay(2000);
   Serial.println("initialising camera, then take photos");
   configCamera();                   // function to configure camera
   WiFi.begin(ssid, password);       // initialise Wi-Fi
   while (WiFi.status() != WL_CONNECTED) delay(500);
   Serial.print("IP Address: ");
   Chapter 2 Intranet camera
   Serial.println(WiFi.localIP());   // display WLAN IP address
   server.begin();                   // initialise server
   server.on("/", HTTP_GET, [](AsyncWebServerRequest * request)
   { request->send_P(200, "text/html", page);});
   server.on("/photoURL", HTTP_GET, [](AsyncWebServerRequest *
   request)
   {
      camera_fb_t * frame = NULL;
      frame = esp_camera_fb_get();   // take photo as JPEG
      request->send_P(200, "image/jpeg", //send JPEG image to client
      (const uint8_t *)frame->buf, frame->len);
      esp_camera_fb_return(frame);   // clear photo buffer
      pSize = String(frame->len);    // display photo size
      Serial.print("pSize ");Serial.println(pSize);
   });
}
void loop()                          // nothing in loop function
{}
```

HTML-код веб-страницы, сохраненный в виде строковой константы, содержится на вкладке *buildpage.h* (см. листинг 2-4). HTML-инструкции для XML HTTP-запросов описаны в главе 8 («Обновление веб-страницы»). Инструкция `location.reload()` перезагружает веб-страницу точно так же, как кнопка перезагрузки в браузере. Изображение поворачивается на *N* градусов с помощью команды `rotate(Ndeg)` с атрибутом трансформации.

HTML-код отображает две кнопки в строке таблицы и распределяет функции для каждой кнопки. В коде AJAX функция `newPhoto` делает HTTP-запрос XML с URL-адресом `/photoURL` для инициализации фотосъемки и последующей перезагрузки веб-страницы.

Листинг 2-4. AJAX-код для веб-страницы с фотографией ESP32-CAM

```
char page[] PROGMEM = R"(
<!DOCTYPE HTML><html><head>
<title>ESP32-CAM</title>
<style>
body {text-align:center; font-size: 25px;}
.vert {margin-bottom: 10%}
.hori {margin-bottom: 0%}
.btn {background-color:White; font-size: 25px}
table {margin: auto}
td {padding: 10px}
</style></head>
<body>
<h2>ESP32-CAM</h2>
<div id='buttons'>
<table><tr>
<td><button onclick='newPhoto()' class='btn'>New photo
</button></td>
<td><button onclick='turn()' class='btn'>Rotate</button></td>
</tr></table></div>
<img src='/photoURL' id='photo' width='80%'>
<script>
function newPhoto()
{
   var xhr = new XMLHttpRequest();
   xhr.open('GET', '/photoURL', true);
   xhr.send();
   location.reload();
}
var deg = 0;
function turn()
   {
   deg = deg + 90;
   var img = document.getElementById('photo');
   img.style.transform = 'rotate(' + deg + 'deg)';
   if((deg/90)%2 == 1)
   document.getElementById('buttons').className = 'vert';
   else document.getElementById('buttons').className = 'hori';
}
</script>
</body></html>
)";
```

Учитывая инструкцию `var rpt = setInterval(Newphoto, 5000)` в разделе `<script>`, в листинге 2-4 каждые пять секунд вызывается функция `newPhoto`, представляющая собой очень простой вариант потоковой передачи изображений. В следующем разделе изображения передаются на веб-страницу.

Потоковая передача изображений на веб-страницу

Модуль ESP32-CAM передает изображения на веб-страницу, а частота кадров потоковой передачи колеблется от 3 FPS[19] для формата UXGA (1600×1200 пикселей) до 30 FPS в формате QQVGA (160×120 пикселей). Несколько форматов изображений показаны в табл. 2-2. Чем меньше размер изображения, тем выше частота кадров. Скетч *CameraWebServer*, доступный в среде IDE Arduino по адресу *File* ➤ *Examples* ➤ *ESP32* ➤ *Camera*, включает функции распознавания и определения лиц с возможностью изменения многочисленных характеристик изображения.

Таблица 2-2. Размеры изображения в пикселах

Кадр	UXGA	SXGA	XGA	SVGA	VGA	CIF	QVGA	HQVGA	QQVGA
Ширина	1600	1280	1024	800	640	400	320	240	160
Высота	1200	1024	768	600	480	296	240	176	120

Скетч в листинге 2-5 управляет потоковой передачей изображений на веб-страницу. Программа загружает необходимые библиотеки, которые доступны в ESP32 Arduino IDE, устанавливает соединение Wi-Fi, настраивает камеру на вкладке *config_pins.h* (см. листинг 2-2) и вызывает функцию `startCameraServer`, которая обращается к функции `stream_handler` (см. листинг 2-6) на вкладке *stream_handler*. Команда `delay()` в функции `loop` может потребоваться для предотвращения запуска программного сброса сторожевого таймера (см. главу 21 «Микроконтроллеры»).

Листинг 2-5. Просмотр веб-страницы в режиме реального времени

```
#include <esp_http_server.h>                  // include esp http_server,
#include <esp_camera.h>                       // camera and Wi-Fi libraries
#include <WiFi.h>
#include "config_pins.h"                      // configure instructions tab
#include "stream_handler.h"                   // code to stream images
char ssid[] = "xxxx";                         // change xxxx to Wi-Fi ssid
char password[] = "xxxx";                     // change xxxx to Wi-Fi password
void setup()
{
  Serial.begin(115200);                       // Serial Monitor baud rate
  Serial.setDebugOutput(false);               // no debug information
  WiFi.begin(ssid, password);                 // initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());             // display WLAN IP address
  configCamera();
  sensor_t * s = esp_camera_sensor_get();     // reduce frame size
  s->set_framesize(s, FRAMESIZE_VGA);         // to 640x480 pixels
  startCameraServer();
}
void startCameraServer()                      // function to start camera server
```

[19] Frames per second, кадров в секунду. – *Прим перев.*

```
{
  httpd_handle_t stream_httpd = NULL;
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;
  httpd_uri_t index_uri = {.uri="/", .method=HTTP_GET,
                           .handler=stream_handler,
                           .user_ctx=NULL};
  if (httpd_start(&stream_httpd, &config) == ESP_OK)
      httpd_register_uri_handler(stream_httpd, &index_uri);
}
void loop()                              // nothing in loop function
{}
```

Листинг 2-6 составлен на основе разделов скетча *CameraWebServer*, управляющих потоковой передачей изображений на веб-страницу. Инструкция `httpd_resp_send_chunk()` возвращает буфер JPEG в виде 64-разрядных разделов в ответ на HTTP-запрос клиента, при этом инструкция `snprintf((char *) part_buf, 64,...)` генерирует 64-разрядные разделы буфера JPEG. Служебное слово `static` создает переменную, специфичную для функции, и значение переменной сохраняется между повторными вызовами функции. Несколько инструкций в листинге 2-6 идентичны для используемых в функции `takePhoto` в листинге 2-1, как указано в комментариях к листингу 2-6.

Листинг 2-6. Потоковая передача изображений в реальном времени

```
#define Boundary "1234567890000000000000987654321"
static const char* ContentType =
"multipart/x-mixed-replace;boundary=" Boundary;
static const char* StreamBound = "\r\n--" Boundary "\r\n";
static const char* StreamContent =
"Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
static esp_err_t stream_handler(httpd_req_t *req)
{
  camera_fb_t * frame = NULL;        // as in Listing 2-1
  esp_err_t res = ESP_OK;            // error status
  uint8_t * jpgBuffer = NULL;        // JPEG buffer
  size_t jpgLength = 0;              // length of JPEG buffer
  char * part_buf[64];
  res = httpd_resp_set_type(req, ContentType);
  if(res != ESP_OK) return res;
  while(true)
  {
    frame = esp_camera_fb_get();      // as in Listing 2-1
    if (!frame) // as in Listing 2-1
    {
      Serial.println("Camera capture failed");
                                     // as in Listing 2-1
      res = ESP_FAIL;
    } else {
      if(frame->width > 400)
      {
      jpgLength = frame->len;        // set JPEG buffer length
      jpgBuffer = frame->buf;        // set JPEG buffer
      }
    }
```

```
      if(res == ESP_OK)                        // no error, stream image
      {
        size_t hlen = snprintf((char *)part_buf, 64,
        StreamContent, jpgLength);
        res = httpd_resp_send_chunk(req, (const char *)
        part_buf, hlen);
      }
      if(res == ESP_OK) res =
      httpd_resp_send_chunk(req, (const char *)jpgBuffer,
      jpgLength);
      if(res == ESP_OK) res =
      httpd_resp_send_chunk(req, StreamBound,
      strlen (StreamBound));
      if(frame)
      {
        esp_camera_fb_return(frame);           // as in Listing 2-1
        frame = NULL;
        jpgBuffer = NULL;                      // reset to NULL value
      } else
      if(jpgBuffer)
      {
        free(jpgBuffer);                       // reset to NULL value
        jpgBuffer = NULL;
      }
      if(res != ESP_OK) break;
    }
    return res;
}
```

Потоковая передача изображений на веб-страницу по сигналу PIR-датчика

Модулю ESP32-CAM требуется ток 130 мА для потоковой передачи изображений на веб-страницу, что быстро разряжает батарею. Можно оставить питание от батареи, если модуль ESP32-CAM установить в спящий режим, когда нет потоковой передачи изображений, из которого для начала фотосъемки и передачи изображения на веб-страницу его выводит PIR-датчик[20] на модуле HC-SR501 (см. рис. 2-3 и подключения в табл. 2-3). Спящий режим ESP32 описан в главе 22 («Функции микроконтроллера ESP32») – сигнал высокого или низкого уровня на специальном выводе GPIO выводит микроконтроллер из спящего режима, что определяется инструкцией `esp_sleep_enable_ext0_wakeup(pin, state)`,

[20] PIR-датчик (*passive infrared sensor*, дословно «пассивный инфракрасный детектор»), или, как его чаще всего называют, *пироэлектрический датчик движения*, – датчик на основе пироэлектрического эффекта, появления зарядов на поверхности некоторых кристаллических материалов при изменении температуры. Типовой PIR-датчик может реагировать на возмущения температурного поля в миллионные доли градуса, причем работа датчика не зависит от температуры окружающего воздуха. Проходящий мимо человек даже в шубе вызовет срабатывание датчика на расстоянии до 7 м, а движение руки датчик регистрирует на расстоянии 2–3 м. Читатель наверняка сталкивался с применениями PIR-датчиков в быту: их очень часто используют для управления освещением подъездов и автоматическими дверями в универсамах и офисах. – *Прим. перев.*

где состояние `state` вывода `pin` задает условие выхода. Когда PIR-датчик не обнаруживает движения, значение `state` равно нулю, так как вывод PIR-датчика подключен к GND с помощью заземляющего резистора. Встроенные заземляющие резисторы микроконтроллера ESP32 отключены во время сна, поэтому должна присутствовать команда `rtc_gpio_pulldown_en(pin)`, включающая заземляющий резистор на выводе GPIO, подключенном к выводу PIR-датчика. Когда PIR-датчик активируется инфракрасным излучением, например когда человек движется в зоне действия, сигнальный вывод PIR-модуля устанавливается в высокий уровень, что выводит микроконтроллер из спящего режима. После того как модуль ESP32-CAM транслирует изображения на веб-страницу в течение требуемого времени, микроконтроллер опять переводится в спящий режим с помощью команды esp_deep_sleep_start().

Модулю PIR HC-SR501 или HC-SR505 требуется до 50 с для стабилизации, особенно модулю HC-SR505[21]. Если PIR-датчик срабатывает без движения после периода стабилизации, то организуйте ему питание, независимое от ESP32-CAM, но с объединенным общим проводом GND.

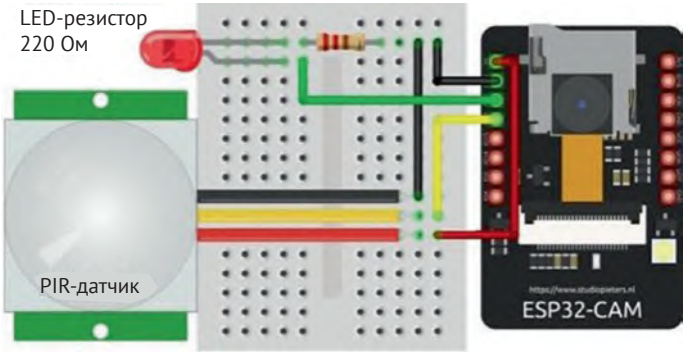


Рисунок 2-3. Подключение PIR-датчика HC-SR501 для запуска потоковой передачи изображений ESP32-CAM

Таблица 2-3. Подключение PIR-датчика HC-SR501 для запуска потоковой передачи изображений ESP32-CAM

Компонент	Подключено к	Также к
ESP32-CAM 5V	HC-SR501 5V	
ESP32-CAM GND	HC-SR501 GND	

[21] Выходной сигнал собственно PIR-сенсора – короткий импульс, но в PIR-модулях, подобных рекомендуемым здесь HC-SR501/HC-SR505, к нему добавляют одновибратор, который выдает однократный положительный импульс регулируемой длительности. Так что утверждение автора о «периоде стабилизации» 50 с не соответствует истине – для датчика HC-SR501 длительность выходного импульса регулируется от 5 до 200 с, для датчика HC-SR505 она постоянна и равна 8 с. В сочетании с небольшой (1–2 с) задержкой перед повторным срабатыванием такой режим служит достаточной защитой от несанкционированного многократного срабатывания. А вот обычные для цифровых устройств помехи по питанию (см. следующую фразу) могут, конечно, служить причиной ложных срабатываний. – *Прим. перев.*

Окончание табл. 2-3

Компонент	Подключено к	Также к
ESP32-CAM GPIO 12	LED длинный вывод	
ESP32-CAM GPIO 13	HC-SR501 OUT	
LED короткий вывод	Резистор 220 Ом	GND

В скетче в листинге 2-7 ESP32-CAM запускается датчиком PIR, светодиод мигает, указывая на обнаруженное движение, установлено соединение Wi-Fi и настроена камера ESP32-CAM. Индикатор снова мигает, указывая на начало потоковой передачи изображений, изображения передаются на веб-страницу в течение нескольких секунд, и индикатор снова мигает, указывая на окончание потоковой передачи изображений. Скетч в листинге 2-7 основан на листинге 2-5, только с добавлением библиотеки ввода-вывода подсистемы RTC ESP32[22] (`rtc_io`), функции `flash` мигания для светодиода, инструкции `esp_sleep` и функций в цикле `loop`, которые определяют прошедшее время с момента начала потоковой передачи изображений. Содержимое вкладок *config_pins.h* и *stream_handler.h* не изменилось. В листинге 2-7 приведены только дополнительные инструкции к листингу 2-5, чтобы подчеркнуть несколько изменений, необходимых для этого скетча.

Листинг 2-7. Подключение PIR-датчика HC-SR501 для запуска передачи изображений ESP32-CAM на веб-страницу

```
#include <esp_http_server.h>
#include <esp_camera.h>
#include <WiFi.h>
#include "config_pins.h"
#include "stream_handler.h"
#include <driver/rtc_io.h>           // include rtc input-output library
int PIRpin = 13;                     // define PIR and LED pins
int LEDpin = 12;
unsigned long startTime,
lastTime = 0;                        // timer variables
int camTime = 10;                    // define image streaming time (s)
int count = 0;                       // counter for steaming time
char ssid[] = "xxxx";
char password[] = "xxxx";
void setup()
{
  pinMode(LEDpin, OUTPUT);           // LED pin as output
  flash();                           // call function to flash LEDs
  Serial.begin(115200);
  Serial.setDebugOutput(false);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  configCamera();
```

[22] О подсистеме RTC микроконтроллера ESP32 см. раздел «RTC и спящий режим» в главе 22 («Особенности микроконтроллера ESP32»). – *Прим. перев.*

```
    sensor_t * s = esp_camera_sensor_get();
    s->set_framesize(s, FRAMESIZE_VGA);
    startCameraServer();
    rtc_gpio_pulldown_en
    ((gpio_num_t)PIRpin);              // pull-down PIR pin
    esp_sleep_enable_ext0_wakeup((gpio_num_t)PIRpin, 1);
}                                      // wakeup on PIR pin with state 1
void startCameraServer()
{
    httpd_handle_t stream_httpd = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;
    httpd_uri_t index_uri = {.uri="/", .method=HTTP_GET,
                             .handler=stream_handler,
                             .user_ctx=NULL};
    if (httpd_start(&stream_httpd, &config) == ESP_OK)
      httpd_register_uri_handler(stream_httpd, &index_uri);
}
void loop()
{
    if(count < 1) flash();             // call function to flash LEDs
    startTime = millis();              // start of image streaming time
    if(startTime - lastTime > 1000 && count < camTime)
    { // display seconds elapsed
      Serial.print("camera ");Serial.println(count);
      count++; // update counter
      lastTime = startTime;            // reset image streaming time
    }
    if(count == camTime)               // defined streaming time elapsed
      {
        flash();
        Serial.print("sleep mode on PIR pin ");
        Serial.println(PIRpin);
        esp_deep_sleep_start();        // ESP32 in sleep mode
      }
    }
void flash()                           // function to flash LEDs
{
    for (int i=0; i<3; i++)            // flash LED three times
    {
      digitalWrite(LEDpin, HIGH);      // turn on LED
      delay(200);
      digitalWrite(LEDpin, LOW);       // turn off LED
      delay(100);
    }
}
```

Если объединить скетчи в листингах 2-1 и 2-7, ESP32-CAM будет запускаться PIR-датчиком для фотосъемки и сохранения файлов изображений в формате JPEG на карте micro-SD.

Итоги

Модуль ESP32-CAM включает в себя микроконтроллер ESP32-S, 2-мегапиксельную камеру OV2640 и слот для карт micro-SD. Изображения сохраняются на карте micro-SD, загружаются на веб-страницу или передаются потоком на веб-страницу на компьютере, планшете Android или мобильном телефоне. Изображения сохраняются на карте micro-SD через определенные промежутки времени для использования при покадровой съемке.

Изображения удаленно загружаются на веб-страницу на планшете или мобильном телефоне Android, где изображение может поворачиваться, а также сохраняться. Изображения в реальном времени передаются потоком с ESP32-CAM-модуля на веб-страницу с частотой кадров до 30 кадров в секунду. PIR-датчик используется для вывода микроконтроллера из спящего режима, который затем передает изображения на веб-страницу в течение определенного периода времени. Затем модуль ESP32-CAM возвращается в энергосберегающий спящий режим.

Перечень компонентов

- Модуль ESP32-CAM
- Преобразователь USB-UART: модуль FT232RL FTDI USB-TTL
- Пироэлектрический датчик (PIR-датчик) HC-SR501 или HC-SR505
- Светодиод с гибкими выводами
- Резистор 220 Ом

Глава 3

Международная метеостанция

Международную информацию о погоде можно отобразить на сенсорном экране, где пользователь выбирает информацию для разных городов и выбирает между двумя экранами отображаемой информации. Первоначально в этой главе описывается отображение текста и фигур на экране, калибровка сенсорной функции экрана и создание изображений нажатием на сенсорный экран. Как только разработаны скетчи для отображения информации и использования сенсорной функции экрана, внимание переключается на доступ к международным данным о погоде из OpenWeatherMap.org и переформатирование этих данных для отображения.

Для отображения текста и фигур на экране или создания изображений нажатием на сенсорный экран не требуется доступ в интернет. Arduino Uno или Nano достаточны для запуска таких задач, однако требуется преобразователь логического уровня для снижения напряжения для сенсорного экрана до 3,3 В, так как Arduino Uno и Nano работают при напряжении 5 В. Для доступа к данным OpenWeatherMap требуется подключение к локальной сети Wi-Fi, которая обеспечивается микроконтроллером ESP8266 или ESP32.

Сенсорный дисплей ILI9341 SPI TFT LCD

2,4-дюймовый сенсорный дисплей ILI9341 SPI TFT LCD с разрешением 240×320 пикселей предназначен для отображения текста и рисования фигур на экране разными цветами. Аббревиатуры SPI, TFT и LCD в названии расшифровываются как Serial Peripheral Interface (последовательный периферийный интерфейс), Thin-Film Transistor (тонкопленочный транзистор) и Liquid Crystal Display (жидкокристаллический дисплей) соответственно. Дисплей ILI9341 и микроконтроллеры ESP8266 и ESP32 работают при напряжении 3,3 В, поэтому преобразователь логического уровня не требуется. Соединения дисплея ILI9341 с платами ESP8266 и ESP32 показаны на рис. 3-1 и 3-2 и приведены в табл. 3-1. Обратите внимание, что выводы интерфейса SPI микроконтроллера MOSI, MISO и SCL подключены как к дисплею, так и к выводам сенсорной функции ЖК-экрана ILI9341.

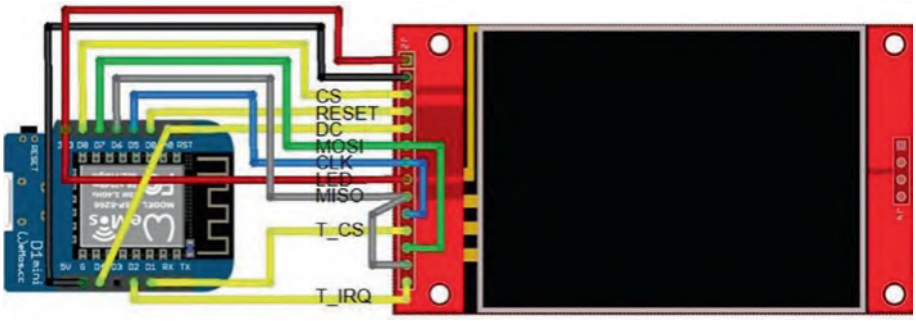


Рисунок 3-1. Соединения дисплея ILI9341 SPI TFT LCD и платы LOLIN (WeMos) D1 mini (ESP8266)

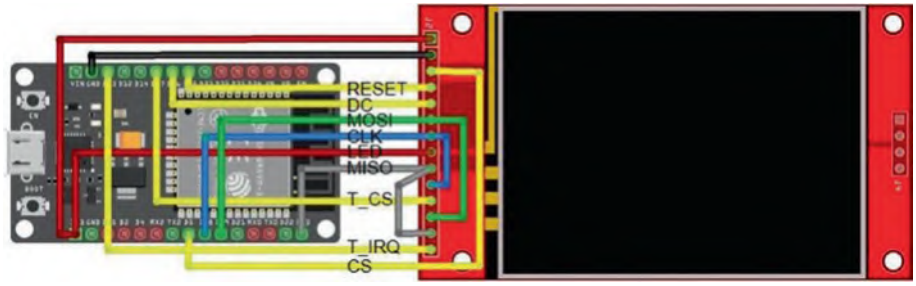


Рисунок 3-2. Соединения дисплея ILI9341 SPI TFT LCD и платы ESP32 DEVKIT DOIT

Таблица 3-1. Соединения дисплея ILI9341 SPI TFT с платами ESP8266 и ESP32

Компонент	Назначение вывода	Вывод ESP8266	Вывод ESP32
ILI9341 VCC 3.3 V		3V3	3V3
ILI9341 GND		GND	GND
ILI9341 CS	Выбор кристалла	D8	GPIO 5
ILI9341 RESET		D0	GPIO 25
ILI9341 D/C	Данные/команда	D4	GPIO 26
ILI9341 SDA (MOSI)	SPI вход данных	D7	GPIO 23
ILI9341 SCL (CLK)	SPI такты	D5	GPIO 18
ILI9341 LED		3V3	3V3
ILI9341 SDO (MISO)	SPI выход данных	D6	GPIO 19
	Сенсорная часть («touch»)		
ILI9341 T_CLK	SPI такты	D5	GPIO 18
ILI9341 T_CS	Выбор кристалла	D1	GPIO 27
ILI9341 T_DIN	SPI вход данных	D7	GPIO 23
ILI9341 T_DO	SPI выход данных	D6	GPIO 19
ILI9341 T_IRQ	Прерывание	D2	GPIO 13

В листинге 3-1 показано отображение текста и фигур для рисования на экране ILI9341. Библиотеки *Adafruit ILI9341* и *Adafruit GFX* установлены в среде Arduino IDE. На библиотеки *Adafruit GFX* и *SPI* ссылается библиотека *Adafruit ILI9341*, поэтому они явно не включены в скетч. Цветовые коды доступны в библиотеке *Adafruit ILI9341*, поэтому шестнадцатеричные коды для цветов в скетче не определяются. Ориентация экрана устанавливается как портретная или альбомная с помощью команды `setRotation(N)` со значением 0 или 1 соответственно или значением 2 или 3 для поворота изображения на экране на 180° для портретной или альбомной ориентации соответственно. Например, портретная ориентация, когда соединения платы экрана ILI9341 обращены вверх, устанавливается с помощью `setRotation(2)`. Размер шрифта по умолчанию составляет 5×8 пикселей на символ; он может быть увеличен до 5*N*×8*N* пикселей с помощью инструкции `setTextSize(N)`.

Листинг 3-1. Отображение текста и фигур

```
#include <Adafruit_ILI9341.h>         // include ILI9341 library
int tftCS = D8;                       // screen chip select pin
int tftDC = D4;                       // data command select pin
int tftRST = D0;                      // reset pin
                                      // associate tft with ILI9341 lib
Adafruit_ILI9341 tft = Adafruit_ILI9341(tftCS, tftDC, tftRST);
String texts[] =                      // color names
  {"BLUE","RED","GREEN","CYAN","MAGENTA","YELLOW","WHITE","GREY"};
unsigned int colors[ ] =              // color codes
  {ILI9341_BLUE, ILI9341_RED, ILI9341_GREEN, ILI9341_CYAN,
   ILI9341_MAGENTA, ILI9341_YELLOW, ILI9341_WHITE,
   ILI9341_LIGHTGREY};
String text;
unsigned int color, chkTime;
void setup()
{
   tft.begin();                       // initialise screen
   tft.setRotation(2);                // portrait, connections at top
   tft.fillScreen(ILI9341_BLACK);     // fill screen in black
   tft.drawRect(0,0,239,319,ILI9341_WHITE); // draw white frame line
   tft.drawRect(1,1,237,317,ILI9341_WHITE); // and second frame line
   tft.setTextSize(4);                // set text size
}
void loop()
{                                     // clear screen apart from frame
   tft.fillRect(2,2,235,314,ILI9341_ BLACK);
   for (int i=0; i<8; i++)            // for each color
   {
      color = colors[i];              // set color
      text = texts[i];                // set text for color
      tft.setTextColor(color);        // set text color
      tft.setCursor(20,40*i+2);       // position cursor
      tft.print(text);                // print color text (name)
      delay(250);                     // delay 250ms between colors
   }
   for (int i=0; i<8; i++)            // for each color
   {
      color = colors[i];
```

```
      text = texts[i];
      tft.fillRect(2,2,235,314,ILI9341_BLACK);
      tft.setCursor(20,25);                    // cursor to position (20, 25)
      tft.setTextColor(color);
      tft.print(text);                         // draw filled-in
      triangle
      if ((i+1)%3==0) tft.fillTriangle(20,134,64,55,107,134,color);
                                               // draw open rectangle
      else if ((i+1)%2==0) tft.drawRect(20,55,88,80,color);
      else tft.fillCircle(64,95,39,color);     // draw filled-in circle
      delay(250);
    }
    tft.fillRect(2,2,235,314,ILI9341_BLACK);
    tft.drawLine(2,158,236,158,ILI9341_RED );
                                               // draw horizontal RED line
    delay(250);
  }
```

В листинге 3-1 определены контакты SPI для платы ESP8266, которые необходимо изменить для платы ESP32. В качестве альтернативы инструкции, приведенные в листинге 3-2, могут быть включены в начало скетча. Загляните в главу 21 («Микроконтроллеры») для получения дополнительной информации.

Листинг 3-2. Определения контактов для плат ESP8266 и ESP32

```
#ifdef ESP32
   int tftCS = 5;     // screen chip select pin
   int tftDC = 26;    // data command select pin
   int tftRST = 25;   // reset pin
#elif ESP8266
   int tftCS = D8;
   int tftDC = D4;
   int tftRST = D0;
#else                 // Arduino IDE error message
   #error "ESP8266 or ESP32 microcontroller only"
#endif
```

Калибровка сенсорного экрана

Библиотека функций сенсорного экрана для ESP8266 и ESP32 микроконтроллеров, *TFT_eSPI* по Бодмер (Bodmer), доступна в Arduino IDE. Перед использованием библиотеки *TFT_eSPI* в скетчах драйвер сенсорного экрана, вывод соединения с платами ESP8266 или ESP32 и частота SPI должны быть определены в *User_Setup.h*, который находится в папке библиотеки *TFT_eSPI*. В листинге 3-3 приведены настройки, используемые в этой главе для микроконтроллеров ESP8266 и ESP32 (настройки для ESP32 закомментированы). Обратите внимание, что для ESP8266 SPI подключения выводов MOSI, MISO и CLK по умолчанию приводить не требуется, но номерам выводов для платы ESP8266 предшествует `PIN_`, в то время как для микроконтроллера ESP32 достаточно просто номера GPIO.

Листинг 3-3. Настройки User_Setup библиотеки TFT-ESPI для плат ESP8266 и ESP32

```
#define TFT_CS PIN_D8          // ESP8266 SPI and touch screen
#define TFT_DC PIN_D4
#define TFT_RST PIN_D0
#define TOUCH_CS PIN_D1
/*                             // lines between /* and */ are commented out
#define TFT_MISO 19            // ESP32 SPI and touch screen
#define TFT_MOSI 23
#define TFT_SCLK 18
#define TFT_CS 5
#define TFT_DC 26
#define TFT_RST 25
#define TOUCH_CS 27
*/
#define ILI9341_DRIVER         // ILI9341 SPI TFT LCD screen
#define TFT_RGB_ORDER TFT_BGR  // color order Blue-Green-Red
#define LOAD_GLCD              // font 1: Adafruit 8-pixel high
#define LOAD_FONT2             // font 2: small 16-pixel high
#define LOAD_FONT4             // font 4: medium 26-pixel high
#define SPI_FREQUENCY 40000000                 // SPI 40 MHz
#define SPI_READ_FREQUENCY 20000000    // SPI read 20 MHz
#define SPI_TOUCH_FREQUENCY 2500000    // SPI touch 2.5 MHz
```

Перед использованием сенсорных функций экран ILI9341 необходимо откалибровать (см. листинг 3-4). Стрелки отображаются на экране ILI9341, когда пользователь нажимает на него экранным пером (стилусом). На экране ILI9341 отображаются пять параметров калибровки, включенных в массив `calData` последующих скетчей. На библиотеку *SPI* ссылается библиотека *TFT_eSPI*, поэтому она явно не включена в скетч. Листинг 3-4 представляет собой адаптированный скетч *Touch_calibrate*, расположенный по адресу *TFT_eSPI* ➤ *Generic*.

Листинг *3-4.* Калибровка экрана ILI9341 SPI TFT LCD

```
#include <TFT_eSPI.h>          // include TFT_eSPI library
TFT_eSPI tft = TFT_eSPI();     // associate tft with TFT-eSPI lib
uint16_t calData[5];           // calibration parameters
String str;
void setup()
{
  tft.init();                  // initialise ILI9341 TFT screen
  tft.setRotation(1);          // landscape, connections on right
  tft.setTextFont(1);          // set text font and size
  tft.setTextSize(1);
  calibrate();                 // call calibration function
}
void calibrate()               // function to calibrate ILI9341 TFT screen
{
  tft.fillScreen(TFT_BLACK);   // fill screen in black
  tft.setTextColor(TFT_WHITE, TFT_B LACK);
                                       // set text color, white on black
  tft.setCursor(30, 0);                // move cursor to position (0, 30)
  tft.println("Touch corners as indicated");
  tft.calibrateTouch(calData, TFT_RED, TFT_ BLACK, 15);
                                       // calibrate screen
```

```
  tft.fillScreen(TFT_BLACK);
  tft.setCursor(0, 50);
  tft.setTextSize(2);
  tft.print("Calibration parameters");
  str = "";                          // display calibration parameters
  for (int i=0; i<4; i++) str = str + String(calData[i])+",";
  str = str + String(calData[4]);
  tft.setCursor(0, 90);
  tft.print(str);
}
void loop()                          // nothing in loop function
{}
```

Например, параметры калибровки для дисплея ILI9341, используемые далее в этой главе, 450, 3400, 390, 3320 и 3, копируются в массив `calData` инструкцией `uint16_t calData[] = {450, 3400, 390, 3320, 3}` из последующих скетчей. Однако параметры калибровки для вашего дисплея ILI9341 будут включены в эти скетчи.

Рисование на экране

Скетч в листинге 3-5 рисует на экране ILI9341 SPI TFT изображения цветами, выбранными из цветовой палитры при нажатии на экран экранным пером. В первом разделе скетча устанавливаются библиотеки, задаются выводы управления экраном и размер кисти. Функция `setup` задает ориентацию сенсорного экрана и включает параметры калибровки экрана, установленные в листинге 3-4. При обнаружении нажатия на экран определяется положение касания. Если координата *x* меньше 20, то экран был нажат на цветовой палитре, и цвет, выбранный в соответствии с координатой *y*, впоследствии используется для рисования на экране. Коды цветов доступны в файле *TFT_eSPI.h* библиотеки *TFT_eSPI*. Функция очистки сбрасывает экран, отображает заголовок и перерисовывает цветовую палитру. Получайте удовольствие от рисования!

Листинг 3-5. Окно для рисования с библиотекой TFT-eSPI

```
#include <TFT_eSPI.h>                // include TFT-eSPI library
TFT_eSPI tft = TFT_eSPI();           // associate tft with TFT-eSPI lib
uint16_t calData[] = {450, 3400, 390, 3320, 3};
                                    // calibration parameters
uint16_t x = 0, y = 0;
int radius = 2;                      // define paintbrush radius
unsigned int color;
void setup()
{
  tft.init();                        // initialise ILI9341 TFT screen
  tft.setRotation(1);                // landscape, connections on right
  tft.setTouch(calData);             // include calibration parameters
  clear();                           // call function to reset screen
}
```

```
void loop()
{
   if (tft.getTouch(&x, &y)>0)                // if screen pressed
   {
      if(x>20) tft.fillCircle(x, y, radius, color); // draw point
      if(x>0 && x<20)                         // select color from color palette
      {
            if(y>75 && y<95) color = TFT_RED;
        else if(y>100 && y<120) color = TFT_YELLOW;
        else if(y>125 && y<145) color = TFT_GREEN;
        else if(y>150 && y<170) color = TFT_BLUE;
        else if(y>175 && y<195) color = TFT_WHITE;
                                              // display selected color
        if(y>75 && y<195) tft.fillCircle(10, 50, 10, color);
        else if(y>215) clear();               // clear screen
      }
   }
}
void clear()                                  // function to reset screen
{
   tft.fillScreen(TFT_BLACK);                 // fill screen
   tft.setTextColor(TFT_GREEN);               // set text color
   tft.setTextSize(2);                        // set text size
   tft.setCursor(110,5);                      // position cursor
   tft.print("Paintpot");                     // screen title
   tft.fillRect(0,75,20,20, TFT_RED);
   tft.fillRect(0,100,20,20,TFT_YELLOW);
   tft.fillRect(0,125,20,20,TFT_GREEN);       // build color palette
   tft.fillRect(0,150,20,20, TFT_BLUE);
   tft.fillRect(0,175,20,20, TFT_WHITE);
   tft.drawCircle(10,225,10, TFT_WHITE);      // select to clear screen
   tft.setCursor(25,217);
   tft.setTextColor(TFT_WHITE);
   tft.print("clear");
   color = TFT_WHITE;
}
```

Особенности ESP8266 при калибровке сенсорного экрана и рисовании

Преимущество библиотеки *TFT-eSPI* заключается в том, что одна библиотека включает в себя функции отображения экрана и сенсорного управления, что в сочетании с дисплеем ILI9341 применимо как к микроконтроллерам ESP8266, так и ESP32. Библиотека *Adafruit_ILI9341esp*, адаптированная специально для микроконтроллера ESP8266, содержит лучшую функцию рисования на сенсорном экране. Библиотека *Adafruit_ILI9341esp* от Nailbuster Software содержится в архиве *tft28esp.zip*, который можно загрузить с nailbuster.com/?page_id=341. Для библиотеки *Adafruit_ILI9341esp* требуется библиотека *XPT2046* от Спироса Пападимитриу (Spiros Papadimitriou), которую можно найти на github.com/spapadim/XPT2046.

Калибровка экрана ILI9341 с помощью библиотеки *XPT2046* отличается от калибровки с помощью библиотеки *TFT-eSPI*. В скетче калибровки (см. листинг 3-6) на экране ILI9341 отображаются два креста с отступами в 20 пикселей

от краев экрана, на которые пользователь должен нажать экранным пером. Затем на экране отображаются четыре параметра калибровки, которые включены в инструкцию `touch.setCalibration` последующих скетчей.

Функция `getpoints` определяет положение сенсорного экрана, при этом параметр & ссылается на указатель на весь массив, а не на указатель на первый элемент массива. Положение касания `(p, q)` сопоставляется с положением `(np, nq)` на экране ILI9341 с разрешением 240×320 пикселей. Для экрана, используемого в этой главе, были определены уравнения регрессии[23] $np = (150 - p) \times 240/145$ и $nq = (115 - q) \times 320/100$, выведенные из серии отмеченных позиций на экране ILI9341 и соответствующих позиций касания, определенных библиотекой *XPT2046*.

Листинг 3-6. Калибровка экрана ILI9341 с помощью библиотеки XPT2046

```
#include <Adafruit_ILI9341esp.h>      // include ILI9341esp and
#include <XPT2046.h>                  // XPT2046 libraries
int tftCS = D8;
int tftDC = D4;
int tftRST = D0;                      // define screen and touch pins
int touchCS = D1;
int touchIRQ = D2;                    // associate tft with ILI9341 lib
Adafruit_ILI9341 tft = Adafruit_ILI9341(tftCS, tftDC, tftRST);
XPT2046 touch(touchCS, touchIRQ);     // associate touch with XPT2046
uint16_t p,q, np, nq;                 // co-ordinates of touched point
String str;
void setup()
{
   tft.begin();                       // initialise ILI9341 screen
   tft.setRotation(0);                // portrait connections at bottom
   touch.begin(tft.height(), tft.width()) ;
                                      // XPT2046 orientated y, x
   touch.setRotation(XPT2046::ROT0);  // no screen rotation
   tft.fillScreen(ILI9341_BLACK);     // fill screen
   tft.setTextColor(ILI9341_WHITE);   // set text color
   tft.setTextSize(1);                // set text size
   tft.setCursor(0, 100);             // position cursor
   str = "width: "+String(tft.width())+", height:"
   +String(tft.height());
   tft.print(str);
   calibrate();                       // calibrate touch screen
}
void calibrate()                      // function to calibrate screen
{
   uint16_t x1,y1,x2,y2,i1,j1,i2,j2; // uint16_t is unsigned integer
   tft.setCursor(0, 50);              // position cursor
   tft.print("press screen on crosses");
   touch.getCalibrationPoints(x1, y1, x2, y2);
                                      // values pre-set in library at 20
   getPoints(x1, y1, i1, j1);         // function to get touch position
```

[23] Уравнение регрессии – уравнение, устанавливающее по экспериментальным данным связь между выходной величиной (в данном случае – теоретической координатой экрана *np* или *nq*) и входной независимой переменной (в данном случае – определенной экспериментально координатой точки касания *p* или *q*). – *Прим. перев.*

```
    delay(500);
    getPoints(x2, y2, i2, j2);                  // get second touch position
    touch.setCalibration(i1, j1, i2, j2);       // string with parameters
    str = String(i1)+","+String(j1)+","+String(i2)+","+String(j2);
    tft.setTextColor(ILI9341_WHITE);
    tft.setCursor(0, 175);
    tft.print("calibration parameters");
    tft.setCursor(0, 200);
    tft.setTextSize(2);                         // reset text size
    tft.print(str);                             // display calibration parameters
}
void getPoints(uint16_t x, uint16_t y, uint16_t &i, uint16_t &j)
{
    marker(y, x, ILI9341_WHITE);                // draw white cross on screen
    while (!touch.isTouching()>0) delay(10);    // wait for screen touch
    touch.getRaw(i, j);
    marker(y, x, ILI9341_BLACK);                // over-write cross
    touch.getPosition(p, q);                    // get position of screen touch
    np = (150.0-p)*240.0/145.0;                 // transform from touch to tft
    nq = (115.0-q)*320.0/100.0;
    tft.fillCircle(np, nq, 2, ILI9341_ GREEN);
}                                               // indicated touch position
void marker(unsigned short x, uint16_t y, int col)
{
    tft.setTextColor(col);                      // set marker color
    tft.drawLine(x-8, y, x+8, y, col);          // draw horizontal line
    tft.drawLine(x, y-8, x, y+8, col);          // draw vertical line
}
void loop()                                     // nothing in loop function
{}
```

Скетч в листинге 3-7 имеет ту же структуру, что и в листинге 3-5, в котором используется библиотека *TFT_eSPI*. Различия между скетчами заключаются в том, что выводы экрана, сенсорного модуля и параметры поворота определяются в скетче, а не во внешнем файле с библиотекой *TFT_eSPI*, и что цвета ссылаются на библиотеки *Adafruit_ILI9341* и *TFT_eSPI*. Функция рисования с библиотекой *XPT2046* более чувствительна, чем с библиотекой *TFT_eSPI*, которая больше подходит просто для указателя экрана. Скетч в листинге 3-7 работает при любом повороте экрана, но рекомендуется использовать альбомную ориентацию.

Листинг 3-7. Окно для рисования с библиотекой XPT2046

```
#include <Adafruit_ILI9341esp.h>      // include ILI9341esp and
#include <XPT2046.h>                  // XPT2046 libraries
int tftCS = D8;
int tftDC = D4;
int tftRST = D0;                      // define screen and touch pins
int touchCS = D1;
int touchIRQ = D2;                    // associate tft with ILI9341 lib
Adafruit_ILI9341 tft = Adafruit_ILI9341(tftCS, tftDC, tftRST);
XPT2046 touch(touchCS, touchIRQ); // associate touch with XPT2046
String str;
uint16_t x, y;
int radius = 2;                       // define paintbrush radius
unsigned int color;                   // rotation: 0, 1, 2 or 3 refers to
```

```
int rotate = 1;                        // rotation of 0, 90, 180 or 270°
void setup()
{
  tft.begin();                         // initialise ILI9341 screen
  setRotation();                       // function for rotation parameters
  touch.setCalibration(1850,1800,3 20,300);
                                       // calibration parameters
clear();                               // call function to reset screen
}
void loop()
{
  if (touch.isTouching()>0)            // if screen pressed
  {
    touch.getPosition(x, y);
    if(x>20) tft.fillCircle(x, y, radius, color); // draw point
    if(x>0 && x<20)                    // select color from color palette
    {
          if(y>75 && y<95) color = ILI9341_RED;
    else if(y>100 && y<120) color = ILI9341_YELLOW;
    else if(y>125 && y<145) color = ILI9341_GREEN;
    else if(y>150 && y<170) color = ILI9341_BLUE;
    else if(y>175 && y<195) color = ILI9341_WHITE;
                                       // display selected color
    if(y>75 && y<195) tft.fillCircle(10, 50, 10, color);
    else if(y>215) clear();            // clear screen
    }
  }
}
void clear()                           // function to reset screen
{
  tft.fillScreen(ILI9341_BLACK);       // fill screen
  tft.setTextColor(ILI9341_GREEN);     // set text color
  tft.setTextSize(2);                  // set text size
  tft.setCursor(110,5);                // position cursor
  tft.print("Paintpot");               // screen title
  tft.fillRect(0,75,20,20, ILI9341_RED);
  tft.fillRect(0,100,20,20,ILI9341_YELLOW);
  tft.fillRect(0,125,20,20,ILI9341_GREEN);    // build color palette
  tft.fillRect(0,150,20,20, ILI9341_BLUE);
  tft.fillRect(0,175,20,20, ILI9341_WHITE);
  tft.drawCircle(10,225,10, ILI9341_WHITE);  //select to clear screen
  tft.setCursor(25,217);
  tft.setTextColor(ILI9341_WHITE);
  tft.print("clear");
  color = ILI9341_WHITE;
}
void setRotation()                     // function to set rotation parameters
{
  tft.setRotation(rotate);
  switch (rotate)
  {
    case 0:                            // no rotation
      touch.begin(tft.width(), tft.height());       // portrait
      touch.setRotation(XPT2046::ROT0);             // connections at bottom
      break;
    case 1:                            // rotation through 90°
```

```
        touch.begin(tft.height(), tft.width());       // landscape
        touch.setRotation(XPT2046::ROT90);            // connections on right
        break;
      case 2:                                         // rotation through 180°
        touch.begin(tft.width(), tft.height());       // portrait
        touch.setRotation(XPT2046 ::ROT180);          // connections at top
        break;
      case 3:                                         // rotation through 270°
        touch.begin(tft.height(), tft.width());       // landscape
        touch.setRotation(XPT2046::ROT270);           // connections on left
        break;
  }
}
```

Данные о погоде для различных городов

Другой пример использования сенсорного экрана ILI9341 отображает подробную информацию о погоде для различных городов, полученную с OpenWeatherMap.org. Данные *OpenWeatherMap* доступны бесплатно в пределах, определенных на ее веб-сайте. Для получения данных *OpenWeatherMap* требуется имя пользователя, пароль, ключ интерфейса прикладного программирования (API) и идентификационный код города (ID). Подробная информация о создании учетной записи и получении ключа API для *OpenWeatherMap* доступна по адресу openweathermap.org/appid. Ключ API идентифицирует клиента для веб-сервера. Идентификатор города может быть получен с веб-сайта OpenWeatherMap.org, если ввести название города в поле поиска. Выберите соответствующий город, а идентификатор города – это номер в конце URL-адреса. Например, берлинский URL-адрес выглядит как https://openweathermap.org/city/2950159.

Библиотеки *ESP8266WiFi* или *Wi-Fi*, *ArduinoJson* и *Time* требуются для связи по Wi-Fi, для интерпретатора JSON, для форматирования данных и вычисления даты и времени. Как библиотека *ArduinoJson* Бенуа Бланшона (Benoît Blanchon), так и библиотека *Timelib* Майкла Марголиса (Michael Margolis) доступны в Arduino IDE, причем последняя указана в разделе *Time*. Некоторые поставщики данных представляют дату и время как количество секунд с 1 января 1970 года (формат времени Unix). Библиотека *Time* преобразует время Unix в соответствующую минуту, час, день, месяц и т. д.

Пример скетча (см. листинг 3-8) получает и отображает ультрафиолетовый (УФ) индекс[24] для Эдинбурга из данных OpenWeatherMap. Если возникают проблемы с подключением Wi-Fi, HTTP-запросом или получением данных, то отображается соответствующее сообщение. Сообщения об ошибке являются необязательными и не включены в функцию подключения в листинге 3-9 да-

[24] Об УФ-индексе (*UV-index*) см. раздел «Связь Bluetooth Low Energy» в главе 22 («Особенности микроконтроллера ESP32»). – *Прим. перев.*

лее, но инструкции клиента, выделенные жирным шрифтом, должны быть сохранены. Если инструкция

```
if (!client.find("\r\n\r\n"))
{
  Serial.println("Received data not complete");
  return;
}
```

с сообщением об ошибке удаляется, то ее необходимо заменить инструкцией `client.find("\r\n\r\n")`.

URL-адрес – здесь это, например, *api.openweathermap.org/data/2.5/uvi?lat=55.95 & lon=-3.19&appid="+APIkey*, который отображает следующие данные:

```
lat          55.95                  // широта (latitude)
lon          - 3.19                 // долгота (longitude)
date_iso     "2020-06-13T12:00:00Z" // дата
date         1592049600             // время Unix
value               6.94            // УФ-индекс
```

Данные в формате JSON состоят из пар `name` (имя) и `value` (значение), таких как `lat` и `55,95`, формирующих документ JSON, который в скетче определяется как массив символов `jsonDoc[]`. Значение пары `name` и `value` идентифицируется по имени и извлекается с помощью инструкции `jsonDoc['name'].as<x>()`, где `x` означает `float`, `long` или `char*` для действительного числа, целого числа или строки соответственно. Например, широта = `jsonDoc['lat'].as<float>()`.

В листинге 3-8 устанавливаются библиотеки, стабилизируется соединение Wi-Fi и отправляется HTTP-запрос на сервер OpenWeatherMap для индекса Edinburgh UV. Широта, долгота, дата и УФ-индекс извлекаются из документа JSON, полученного в ответ на HTTP-запрос. Выполняется полный набор проверок ошибок, касающихся установления соединения Wi-Fi, HTTP-запроса, ответа на HTTP-запрос, получения документа JSON в HTTP-отклике и его извлечения.

Листинг 3-8. Пример установок для данных OpenWeatherMap

```
#include <ESP8266WiFi.h>      // library to connect to Wi-Fi network
#include <ArduinoJson.h>
WiFiClient client;            // create client to connect to IP address
char ssid[] = "xxxx";         // change xxxx to your Wi-Fi SSID
char password[] = "xxxx";     // change xxxx to your Wi-Fi password
String APIkey = "xxxx";       // and xxx to openweathermap API key
char server[]="api.openweathermap.org";
String output;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  WiFi.begin(ssid, password);         // initialise Wi-Fi and wait
  while (WiFi.status() != WL_CONNECTED) delay(500);
                                      // for Wi-Fi connection
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // Wi-Fi network IP address
  Serial.println("Connecting...");
```

```
    client.connect(server, 80);        // connect to server on port 80
    if (!client.connect(server, 80))   // connection error message
    {
      Serial.println("Connection failed");
      return;
    }
    Serial.println("Connected!");
    client.println("GET /data/2.5/uvi?lat=55.95&lon=-3.19&"
    "appid="+APIkey+" HTTP/1.1");      // send HTTP request
    client.println("Host: api.openweathermap.org");
    client.println("User-Agent: ESP8266/0.1");
    client.println("Connection: close");
    client.println();
    if (client.println() == 0)         // HTTP request error message
    {
      Serial.println("HTTP request failed");
      return;
    }
    char status[32] = {0};
    client.readBytesUntil('\r', status, sizeof(status));
    if (strcmp(status, "HTTP/1.1 200 OK") != 0)
    {                                  // HTTP status error message
      Serial.print("Response not valid: ");
      Serial.println(status);
      return;
    }
    else Serial.println("HTTP status OK");
    if (!client.find("\r\n\r\n"))      // received data error request
    {
      Serial.println("Received data not complete");
      return;
    }
    // client.find("\r\n\r\n");
    DynamicJsonDocument jsonDoc(1024);
    DeserializationError error =
    deserializeJson(jsonDoc, client);
    if (error)
    {                                  // JSON error message
      Serial.print("deserializeJson() failed: ");
      Serial.println(error.c_str());
      return;
    }
    serializeJson(jsonDoc, output);    // display all data
    Serial.print("data length: ");Serial.println(output.length());
    Serial.println(output);
    Serial.println("extracted text");
    Serial.println(jsonDoc["lon"].as<float>(), 2);
                                       // display specific data
    Serial.println(jsonDoc["lat"].as<float>(), 2);
    Serial.println(jsonDoc["date_iso"].as<char*>());
    Serial.println(jsonDoc["date"].as<long>());
    Serial.print("UV ");
    Serial.println(jsonDoc["value"].as<float>(), 2);
}
void loop()                            // nothing in loop function
{}
```

В листинге 3-9 отображаются данные OpenWeatherMap для выбранного города с информацией о погоде, отображаемой на разных экранах (см. скриншоты в начале раздела). На первом экране отображаются текущая минимальная и максимальная температура и прогнозируемая погода с текущей влажностью и давлением. На втором экране отображается облачность, скорость и направление ветра, а также более подробная информация о прогнозируемой погоде с указанием времени восхода и захода солнца. Город выбирается из аббревиатур городов в правой части экрана, и отображение на экране чередуется нажатием символа ⊕ слева вверху экрана. Скетч получился объемным из-за обработки полных данных о погоде и разделен на функции, чтобы его было легче интерпретировать. Объем информации о погоде можно уменьшить за счет удаления функции `SecondScreen` и ссылки на нее в функции `getWeather`.

Первый раздел скетча включает библиотеки и определяет экранные и сенсорные контакты, ключ API *OpenWeatherMap* и идентификационные коды городов. Первый (нулевой) элемент массивов `days` и `mths` является пустым, так что `days[1]` и `mths[1]` соответствуют воскресенью и январю соответственно. Функция настройки устанавливает соединение Wi-Fi, инициализирует сенсорный экран и включает в себя параметры калибровки сенсорного экрана, полученные в листинге 3-4. Функция `loop` вызывает функцию `checkTime`, чтобы определить, прошло ли требуемое время с момента последнего обновления экрана и выбора города. Функция `connect` отправляет HTTP-запрос GET на сервер *OpenWeatherMap*, включающий ключ API, и загружает данные в формате JSON для отображения на дисплее ILI9341. Вывод данных в формате JSON из *OpenWeatherMap* сопровождается следующими инструкциями:

```
String output
serializeJson(jsonDoc, output)
Serial.println(output.length())
Serial.println(output)
```

Пример данных в формате OpenWeatherMap в формате JSON:

```
{"coord":{"lon":-3.2,"lat":55.95},
"weather":[{"id":803,"main":"Clouds", "description":"broken
clouds", "icon":"04d"}],
"base":"stations",
"main":{ "temp":14.38,"feels_like":7.99,"temp_min":12.78,
         "temp_max":15.56, "pressure":1024,"humidity":62},
"visibility":10000,"wind":{"speed":8.2,"deg":80},"clouds":{"all":68},"
dt":1591882710,
"sys":{"type":1,"id":1442,"country":"GB", "sunrise":1591846048,
"sunset":1591909066},
"timezone":3600,"id":2650225,"name":"Edinburgh", "cod":200}
```

На значения ссылаются в соответствии с классом вне фигурных скобок, {}, категорией в фигурных скобках и именем переменной, причем каждый параметр заключен в квадратные скобки [ ]. Например, значение влажности, равное 62, указывается как `["main"]["humidity"]`. Класс погоды может иметь два уровня, обозначенных как [0] и [1]. Значения времени приходят в формате времени Unix. Например, значение времени восхода солнца (sunrise), равное

1591846048, обозначается как ["sys"] ["sunrise"] и равно 04:27 по Гринвичу 11 июня 2020 года.

Форматы JSON-данных могут иметь типы string, float или unsigned long, например:

```
String weather = jsonDoc["weather"][0]["main"]
float temp = jsonDoc["main"]["temp"]
unsigned long srise = jsonDoc["sys"]["sunrise"]
```

Данные отображаются инструкцией вывода через последовательный порт, например Serial.println(weather). Данные в формате JSON отображаются непосредственно путем включения ссылки на данные и выходного формата в инструкцию вывода, например

```
Serial.println(jsonDoc["weather"][0]["main"].as<char*>())
Serial.println(jsonDoc["main"]["temp"].as<float>(), 2) // for 2 DP
Serial.println(jsonDoc ["sys"]["sunrise"].as<long>())
```

Структуры функций firstScreen и secondScreen одинаковы (см. рис. 3-3). Функция citySquare очищает экран; отображает список сокращений городов, используя функцию screen, которая позиционирует и отображает строки; и рисует прямоугольник вокруг выбранной аббревиатуры города. Время Unix для обновления погоды преобразуется в день, дату и время и отображается на экране. Данные о погоде извлекаются из документа JSON, а функция screen позиционирует и отображает строку в требуемом цвете. Чтобы сохранить позиционирование текста, текст заполняется пробелами с помощью функции addb. Чтобы сделать функции в листинге 3-9 более понятными для интерпретации, первая часть инструкций screen("variable" выделена жирным шрифтом.

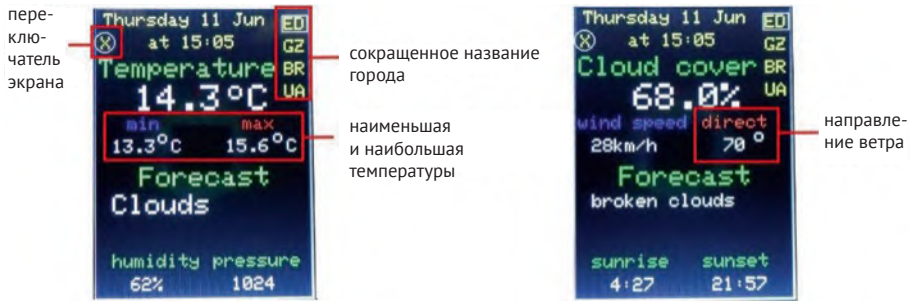


Рисунок 3-3. Информация OpenWeatherMap на экране

Листинг 3-9. . Вывод информации OpenWeatherMap

```
#include <ESP8266WiFi.h>              // include ESP8266WiFi library
#include <ArduinoJson.h>              // include JSON library
#include <TimeLib.h>                  // include TimeLib library
#include <TFT_eSPI.h>                 // include TFT_eSPI library
TFT_eSPI tft = TFT_eSPI();            // associate tft with TFT-eSPI lib
uint16_t calData[] = {450, 34 00, 390, 3320, 3};
                                     // calibration parameters
uint16_t x = 0, y = 0;
```

```
WiFiClient client;                    // create client to connect to IP address
char ssid[] = "xxxx";                 // change xxxx to your Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to your Wi-Fi password
String APIkey = "xxxx";               // change xxxx to weathermap API key
String city[] = {"ED", "GZ","BR","UA"};
                                      // Edinburgh, Günzburg, Brisbane, Ushuaia
                                      // openweathermap city identity codes
String cityID[] = {"2650225","2913555","2174003","3833367"};
int Ncity = 4;                        // number of cities
int cityNow = 0;                      // current city, initially set at 0
int count = 99;                       // run getWeather function at start
char server[] = "api.openweathermap.org";
int screenFlag = 0;                   // flag for first or second screen
int touchFlag = 0;                    // to indicate screen pressed
String days[] = {"",
"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday",
"Saturday"};
String mths[] = {"",
"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct",
"Nov","Dec"};
String wkdy, mth, addb, text;
int sunriseh, sunrisem, sunseth, sunsetm, dy, hr, mn;
unsigned int chkTime;
DynamicJsonDocument jsonDoc(1024);    // JSON formatted data
void setup()
{
  WiFi.begin(ssid, password);         // initialise Wi-Fi and wait
  while (WiFi.status() != WL_CONNECTED) delay(500);
                                      // for WiFi connection
  tft.init();                         // initialise screen for graphics
  tft.setTouch(calData);              // calibration parameters
  tft.setRotation(2);                 // portrait, connections on top
  tft.fillScreen(TFT_BLACK);          // fill screen in black
  tft.drawRect(0,0,239,319,TFT_WHITE); // draw white frame line
  tft.drawRect(1,1,237,317,TFT_WHITE); // and second frame
}
void loop()
{
  checkTime();                                  // check if time to refresh screen
  if(tft.getTouch(&x, &y)>0 && touch Flag == 0)
                                                // if screen pressed
  {
    touchFlag = 1;
    if(y > 290)                                 // city to be selected
  {
    screenFlag = 0;                             // start with first screen
    if(x > 215) cityNow = 0;
    else if(x > 195 && x<216) cityNow = 1;
    else if(x > 175 && x<196) cityNow = 2;      // select city
    else if(x > 155 && x<176) cityNow = 3;
  }
  if(y < 20) screenFlag = 1-screenFlag;         // change screen
  count = 99;                                   // run getWeather function immediately
  }
}
```

```
void checkTime()              // check for screen refresh
{
  if(count > 4)               // update screen every 5 minutes
  {
    getWeather();             // call weather report function
    count = 0;                // reset counter
  }
  else if(millis()-chkTime>60000)    // increment counter after 60s
  {
    chkTime = millis();               // reset timer
    count++;                          // increment counter
  }
}
void getWeather()                     // function to get weather data
{
  connect();                          // call Wi-Fi
  connection function
  if(screenFlag == 0) firstScreen();
                                      // select screen to be displayed
  else secondScreen();
  touchFlag = 0;                      // reset touch indicator
}
void connect()                        // function for Wi-Fi connection
{
  client.connect(server, 80);         // connect to server on port 80
  client.println("GET /data/2.5/weather?id="+cityID[cityNow]+
  "&units=metric&appid="+APIkey+" HTTP/1.1");        // send HTTP
                                                     // request
  client.println("Host: api.openweathermap.org");
  client.println("User-Agent: ESP8266/0.1");
  client.println("Connection: close");
  client.println();
  client.find("\r\n\r\n");            // essential instruction
  DeserializationError error = deserializeJson(jsonDoc, client);
}
void firstScreen()                    // weather data for first screen
{
  citySquare();                       // call function to display header
  String weather = jsonDoc["weather"][0]["main"];
  String weather2 = jsonDoc["weather"][1]["main"];
  String id1 = jsonDoc["weather"][1]["id"];
  float temp = jsonDoc["main"]["temp"];
  float pres = jsonDoc["main"] ["pressure"];
                                      // convert JavaScript objects
  float humid = jsonDoc["main" ]["humidity"];
                                      // to strings or real numbers
  float tempMin = jsonDoc["main"]["temp_min"];
  float tempMax = jsonDoc["main"]["temp_max"];
  if(id1.length()<1) weather2 = " ";
  screen("Temperature",TFT_GREEN,5, 55,3);
                                      // display weather variable name
  text = String(temp,1);              // convert value to string
  if(temp<9.95) text = " "+text;      // add space if less than 10
  screen(text,TFT_WHITE,45,85,4);     // display string on screen
  screen("o", TFT_WHITE,148,80,3);    // add °symbol
  screen("C", TFT_WHITE,170,85,4);    // add C for Celsius
```

```
    screen("min",TFT_BLUE,37,120,2);          // minimum temperature
    text = String(tempMin,1);
    if(tempMin<10) text = " "+text;
    screen(text,TFT_WHITE,20,145,2);
    screen("o",TFT_WHITE,70,135,2);
    screen("C",TFT_WHITE,85,145,2);
    screen("max",TFT_RED,163,120,2);          // maximum temperature
    text = String(tempMax,1);
    if(tempMax<10) text = " "+text;
    screen(text,TFT_WHITE,145,145,2);
    screen("o",TFT_WHITE,197,135,2);
    screen("C",TFT_WHITE,212,145,2);
    screen("Forecast",TFT_GREEN,50,175,3);    // forecast weather
    addb = blank(weather);                    // add spaces after text
    weather=weather+addb;
    screen(weather,TFT_WHITE,20,205,3);
    if(weather2 == "null") weather2="";
    addb = blank(weather2);
    weather2=weather2+addb;
    screen(weather2,TFT_WHITE,20,235,3);      // forecast weather detail
    screen("humidity",TFT_GREEN,20,270,2);    // humidity
    text = String(humid,0)+"% ";
    screen(text,TFT_WHITE,40,295,2);
    screen("pressure",TFT_GREEN,130,270,2);   // pressure
    text = String(pres,0);
    if(pres<1000) text = " "+text;
    screen(text,TFT_WHITE,150,295,2);
}
void secondScreen()                           // weather data for second screen
{
    citySquare();                             // call function to display header
    String desc = jsonDoc["weather"][0]["description"];
    String desc2 = jsonDoc["weather"][1]["description"];
    String id1 = jsonDoc["weath er"][1]["i d"];
                                              // convert JavaScript objects
    float windspd = jsonDoc["wi nd"]["spee d"];
                                              // to strings or real numbers
    float winddeg = jsonDoc["wind"]["deg"];
    float cloud = jsonDoc["clouds"]["all"];
    unsigned long srise = jsonDoc["sys"]["sunrise"];
    unsigned long sset = jsonDoc["sys"]["sunset"];
    if(id1.length()<1) desc2 = " ";
    screen("Cloud cover",TFT_GREEN,5,55,3);   // cloud cover
    text = String(cloud,1)+"%";
    screen(text,TFT_WHITE,65,85,4);
    screen("wind speed",TFT_BLUE,5,120,2);    // wind speed
    windspd = windspd*3.6;                    // convert m/s to km/h
    text = String(windspd,0)+"km/h";
    screen(text,TFT_WHITE,20,145,2);
    screen("direct",TFT_RED,140,120,2);       // wind direction
    text = String(winddeg,0);
    if(winddeg<10) text = " "+text;
    if(winddeg<100) text = " "+text;
    screen(text,TFT_WHITE,152,145,2);
    screen("o",TFT_WHITE,197,135,2);
    screen("Forecast",TFT_GREEN,50,175,3);    // weather forecast (2)
    addb = blank(desc);
```

```
    desc=desc+addb;
    if(desc.length()<13) screen(desc,TFT_WHITE,20,205,3);
    else screen(desc,TFT_WHITE,20, 205,2);
                                              // font size depends on text length
    if(desc2 == "null") desc2="";
    addb = blank(desc2);
    desc2=desc2+addb;
    if(desc.length()<13 && desc2.length()<13)
    screen(desc2,TFT_WHITE,5,235,3);
    else screen(desc2,TFT_WHITE,5,235,2);
    screen("sunrise",TFT_GREEN,20,270,2);      // sunrise time
    text = String(minute(srise));
    if(minute(srise)<10) text = "0"+text;
    text = String(hour(srise)+1)+":"+text;
    screen(text,TFT_WHITE,40,295,2);
    screen("sunset",TFT_GREEN,140,270,2);      // sunset time
    text = String(minute(sset));
    if(minute(sset)<10) text = "0"+text;
    text = String(hour(sset)+1)+":"+text;
    screen(text,TFT_WHITE,150,295,2);
}
void citySquare()                     // display header and city abbreviations
{
    tft.fillRect(2,2,235,315,TFT_BLACK );
                                      // clear screen apart from frame
                                      // display city abbreviations
    for (int i=0; i<Ncity; i++) screen(city[i],
    TFT_YELLOW,210,10+i*25,2);
    for (int i=0; i<Ncity; i++) tft.drawRect(208,8+i*25,29,19,
    TFT_BLACK);
                                      // draw rectangle for selected city
    tft.drawRect(208,8+cityNow*25,29,19,TFT_WHITE);
    screen("X",TFT_YELLOW,7,31,2);     // draw X with circle
    tft.drawCircle(11,37,11,TFT_WHITE);
    unsigned long stime = jsonDoc["dt"];
                                      // time in secs since 1 Jan 1970
    hr = hour(stime)+1;
    mn = minute(stime);
    dy = day(stime);                  // convert time
    wkdy = days[weekday(stime)];
    mth = mths[month(stime)];
    text = wkdy+" "+String(dy)+" "+mth; // display day, date and time
    screen(text,TFT_YELLOW,10,5,2);
    text = "at "+String(hr)+":"+String(mn);
    if(mn<10) text = "at "+String(hr)+":0"+String(mn);
    screen(text,TFT_YELLOW,60,30,2);
}
                                      // function to position and display strings
void screen(String text, unsigned int color, int x, int y,
int size)
{
    tft.setCursor(x, y);              // position cursor
    tft.setTextColor(color,TFT_BLACK); // background color: black
    tft.setTextSize(size);
    tft.print(text);
}
```

```
String blank(String txt)              // function to add spaces to text
{
   String addb = "";
   int len = 12-txt.length();         // add up to 11 spaces
   for (int i=0;i<len;i++) addb=addb+" ";
   return addb;
}
```

Итоги

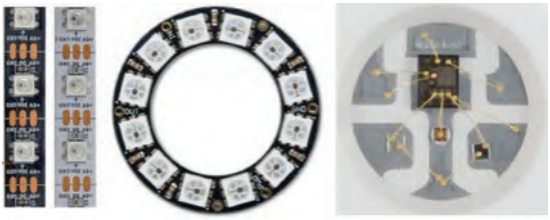
Wi-Fi-функции микроконтроллеров ESP8266 и ESP32 обеспечивают доступ в интернет к данным о погоде с *OpenWeatherMap*. Сенсорные функции экрана ILI9341 позволяют выбрать город из меню сенсорного экрана. Полученные данные о погоде в формате JSON отображаются на TFT ЖК-экране ILI9341. Использование библиотек *TFT-eSPI* и *XPT2046* иллюстрирует два процесса калибровки экрана и различия в функциях рисования на экране ILI9341.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (wemos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата DEVKIT DOIT или NodeMCU
- Сенсорный экран ILI9341 SPI TFT LCD, 2,4 дюйма, 240×320 пикселей

Глава 4

Интернет-часы



Каждый светодиод (LED) в RGB[25] светодиодной ленте или кольце адресуется индивидуально, через общий для всех светодиодов вывод микроконтроллера. Название WS2812 5050 RGB LED относится к ленте или кольцу, состоящему из RGB-светодиодов с контроллером WS2812, встроенным в каждый светодиод, а 5050 относится к размерам светодиода 5,0×5,0 мм. *NeoPixel* – это торговая марка Adafruit для индивидуально адресуемых RGB-светодиодов.

Библиотека *Adafruit NeoPixel* применима к светодиодным лентам и кольцам WS2812 RGB, а также к другим продуктам *NeoPixel*. Количество RGB-светодиодов в ленте или кольце (`LEDnumber`) и управляющий вывод микроконтроллера (`LEDpin`) задаются с помощью инструкции

```
Adafruit_NeoPixel strip(LEDnumber, LEDpin, NEO_GRB + NEO_KHZ800)
```

которая связывает светодиодную ленту `strip` с библиотекой *Adafruit NeoPixel* и подключает ее к битовому потоку GRB (зеленый–красный–синий) частотой 800 кГц. Для других случаев могут потребоваться настройки `NEO_RGB + NEO_KHZ800`. За инструкциями RGB-дисплея следует инструкция `show()`, активирующая их выполнение.

Яркость RGB-светодиода определяется в функции установок `setup`, а не в функции основного цикла `loop`, инструкцией `brightness(N)` с параметром *N* в диапазоне от 1 до 255 от минимальной до полной яркости. Инструкция `unsigned long col = strip.Color(R, G, B)` преобразует три 8-битных красных (R), зеленых (G) и синие (B) компоненты цвета в 32-разрядное число, как того требует библиотека *Adafruit NeoPixel*, со значениями RGB для каждого цвета, хранящимися в отдельных массивах. Цвет RGB рассчитывается как $16^4R + 16^2G + B$ с помощью инструкции `pow(16,4)*R + pow(16,2)*G + B`. RGB-светодиоды в ленте или кольце нумеруются с нуля, и инструкция `strip.setPixelColor(n, col)` задает цвет (n + 1)-го RGB-светодиода. Цвет некоторого количества `number` последовательно расположенных RGB-светодиодов устанавливается с помощью инструкции `strip.fill(col, n, number)`, начиная с (n + 1)-го светодиода. Если количество светоди-

[25] Red-Green-Blue, красный–зеленый–синий. – *Прим. перев.*

одов `number` не указано в инструкции по заполнению, то устанавливаются все RGB-светодиоды от (n + 1)-го светодиода до конца ленты или кольца. Аналогично, если номер начального светодиода опущен, то устанавливается вся лента или кольцо целиком. Инструкция `strip.clear()` отключает все светодиоды в ленте или кольце.

Один RGB-светодиод потребляет до 60 мА при полной яркости всех трех составляющих. Когда плата ESP8266 питается от USB, вывод 5V выдает 400 мА[26], поэтому абсолютный максимум при питании от вывода 5V платы ESP8266 – шесть светодиодов RGB с полной яркостью. Поэтому светодиодная лента должна питаться от внешнего источника питания 5 В, как показано на рис. 4-1 с соединениями, указанными в табл. 4-1. Для защиты светодиодов RGB от скачков тока при включении источника питания на выводах источника питания 5 В и GND установлен конденсатор емкостью 100 мкФ. Установка резистора 470 Ом между выводом *data in* светодиодной ленты или кольца и выводом *data* платы ESP8266 предотвращает скачки напряжения на линии передачи данных, которые могут повредить первый RGB-светодиод на ленте или кольце. Подключения к светодиодной ленте следует проверять мультиметром с помощью функции «прозвонки».

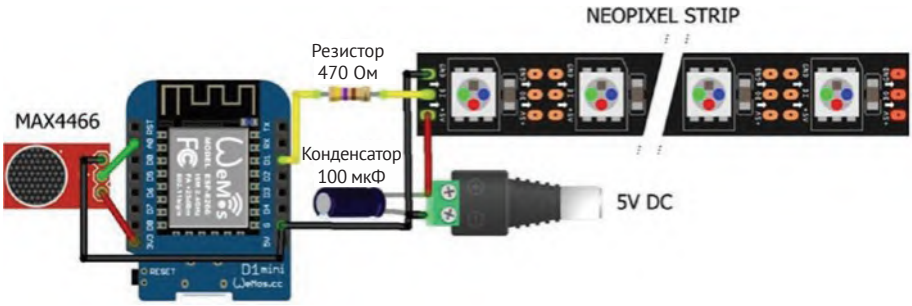


Рисунок 4-1. Светодиодная лента WS2812 RGB и микрофон MAX4466[27]

Таблица 4-1. Соединения светодиодной ленты WS2812 RGB и микрофона MAX4466 с платой ESP8266

Компонент	Подключено к	Также к	Также к
LED-лента VCC	Внешнее питание 5V		Конденсатор 100 мкФ положит. вывод
LED-лента GND	Внешнее питание GND	ESP8266 GND	Конденсатор 100 мкФ отрицат. вывод

[26] Вывод питания в стандартах USB 1.x и 2.0 поддерживает ток в нагрузке до 500 мА, однако автор, очевидно, учитывает собственное потребление ESP8266 и Wi-Fi-модуля. Более современный стандарт USB 3.x поддерживает ток до 900 мА, то есть формально потянет до 13–14 RGB-светодиодов, однако рассчитывать на это не следует – автор далее совершенно справедливо говорит о необходимости питания такой мощной нагрузки от отдельного источника. – *Прим. перев.*

[27] О подключении микрофона рассказывается в следующем разделе этой главы. – *Прим. перев.*

Окончание табл. 4.1

Компонент	Подключено к	Также к	Также к
LED-лента DI (data in)	Резистор 470 Ом	ESP8266 D1	
MAX4466 VCC	ESP8266 3V3		
MAX4466 OUT	ESP8266 A0		
MAX4466 GND	ESP8266 GND		

Единственным изменением в листинге 4-1 для микроконтроллера ESP32, в отличие от микроконтроллера ESP8266, является установка для вывода *data* платы `int LEDpin = x`.

Скетч в листинге 4-1 демонстрирует распространение цветов вдоль светодиодной ленты.

Листинг 4-1. Распространение цветов вдоль светодиодной RGB-ленты

```
#include <Adafruit_NeoPixel.h>        // include Adafruit NeoPixel library
int LEDpin = D1;                      // define data pin
int LEDnumber = 30;                   // number of LEDS in strip
                                      // associate strip with NeoPixel library
Adafruit_NeoPixel strip(LEDnumber, LEDpin, NEO_GRB + NEO_KHZ800);
                     // color white, red, lime, blue, yellow, cyan, magenta,
                     // grey, maroon, olive, green, purple, teal, navy
int R[ ] = {255,255, 0, 0,255, 0,255,128,128,128, 0,128, 0, 0};
int G[ ] = {255, 0,255, 0,255,255, 0,128, 0,128,128, 0,128, 0};
int B[ ] = {255, 0, 0,255, 0,255,255,128, 0, 0, 0,128,128,128};
uint32_t color;                       // color is 32-bit or unsigned long
void setup()
{
  strip.begin();                      // initialise LED strip
  strip.setBrightness(10);            // define LED brightness (1 to 255)
  strip.show();                       // sets all pixels to “off” as no color set
}
void loop()
{
  for (int i=0; i<14; i++)            // cycle through the RGB colors
  {                                   // convert RGB values to 32-bit number
    color = strip.Color(R[i],G[i],B[i]);
    sweep(color, 40);                 // sweep color through the LED strip
  }
  rainbow(3, 10);                     // rainbow colors for three cycles
}                                     // with a 10ms time lag for each color
void sweep(uint32_t color, int lag)   // color sweep function
{
  for (int i=0; i<strip.numPixels(); i++)    // for each LED in strip
  {
     strip.setPixelColor(i, color);           // set the LED color
//   strip.setPixelColor
//   (strip.numPixels()-i-1, color);          // reverse direction
     strip.show();                            // update LED strip
     delay(lag);                              // time lag between color changes
  }
}
```

```
void rainbow(int cycle, int lag)
{                                    // from Adafruit NeoPixel>strandtest
   for (long Pixel1Hue = 0; Pixel1Hue < cycle*65536;
   Pixel1Hue += 256)
   {
      for (int i=0; i<strip.numPixels(); i++)
      {
         int pixelHue = Pixel1Hue + (i * 65536L / strip.numPixels());
         strip.setPixelColor(i,
         strip.gamma32(strip.ColorHSV(pixelHue)));
      }
      strip.show();                  // update LED states and colors
      delay(lag);
   }
}
```

Светодиодная RGB-лента WS2812, управляемая звуком

Управление светодиодной RGB-лентой WS2812 с помощью модуля усилителя электретного микрофона MAX4466 позволяет создать световой дисплей, реагирующий на звуки, такие как речь или музыка. Уровень звука, определяемый микрофоном, определяет количество включаемых светодиодов, цвет которых зависит от уровня звука (см. рис. 4-1). Модуль усилителя электретного микрофона MAX4466 питается от 3,3 В, а не 5 В. Уровни звука от пика до пика для модуля усилителя электретного микрофона MAX4466 определяются как разница между максимальными и минимальными уровнями звука, записанными во время выборки.

Модуль усилителя электретного микрофона MAX4466, используемый в этой главе, выдает сигнал, искаженный шумом, особенно если скетч включает инструкции по управлению светодиодной лентой RGB. В левой части рис. 4-2 показан звук минимальной громкости, обнаруживаемой модулем усилителя электретного микрофона MAX4466. Правая часть рис. 4-2 отражает воспроизводимую музыку. В обоих случаях шум очевиден, и требуется медианный фильтр для его исключения из выборки звука. Медианный фильтр определяет медианное значение выборки, в отличие от обычного циклического буфера, который оперирует средним значением. Например, если три выборочные последовательности с пятью значениями каждое являются 3, 4, 5, 6, 80; 4, 5, 6, 80, 7; и 5, 6, 80, 7, 8, то три медианных значения равны 5, 6 и 7 соответственно[28]. Медианная фильтрация может отставать от фактической последовательности выборки, но экстремальные значения исключаются из рассмотрения. Рисунок 4-2 иллюстрирует, что медианная фильтрация эффективно устраняет шум как при низкой, так и при средней громкости звука. Медианная фильтрация используется в листинге 4-2 с помощью библиотеки *RunningMedian* Роба Тиллаарта (Rob Tillaart), которая доступна в среде Arduino IDE.

[28] Медианное значение выборки данных – число, для которого половина элементов выборки не меньше его, а другая половина не больше. Медианное значение меньше зависит от случайных пиковых выбросов, в отличие от обычного среднего арифметического, то есть в данном случае служит эффективным методом фильтрации шума. В числовом примере автора медианные значения равны 5, 6 и 7, а средние (19.6, 20.4 и 21.2) искажены единственным случайным выбросом большой амплитуды. – *Прим. перев.*

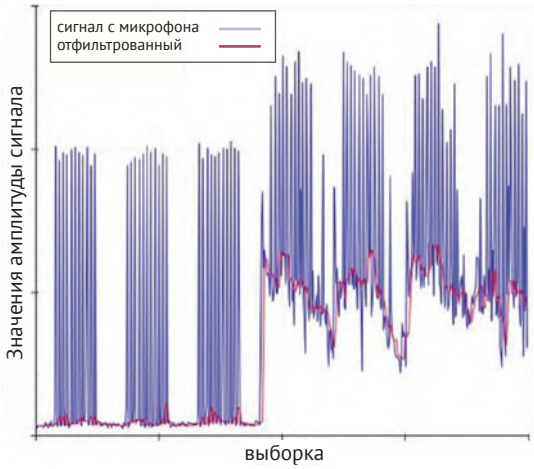


Рисунок 4-2. Медианный фильтр значений амплитуды от пика до пика

В скетче по листингу 4-2 модуль усилителя электретного микрофона MAX4466 производит выборку звука в течение периода времени 50 мс, соответствующего частоте дискретизации 20 Гц, для определения количества RGB-светодиодов, которые будут включены. Медианный фильтр с объемом выборки в семь отсчетов удаляет шум. Чувствительность электретного микрофона регулируется с помощью переменной `adjustVol` для увеличения или уменьшения количества включаемых RGB-светодиодов. Изменение чувствительности приводит к изменению количества постоянно включенных RGB-светодиодов, которые настраиваются на ноль в соответствии с переменной `baseline`. Переменные `adjustVol` и `baseline` выводятся в монитор последовательного порта Arduino IDE вместе со строкой последовательного буфера, преобразованной в два целых числа с помощью инструкции `parseInt()`. Для ленты, содержащей 30 RGB-светодиодов, было выбрано семь цветов по четыре светодиода для каждого цвета. Значения RGB для каждого цвета хранятся в массивах с цветом RGB-светодиодов, заданным инструкциями `strip.color()` и `strip.setPixelColor()`.

В листинге 4-2 определены данные платы ESP32 и выводы аналогового ввода с помощью инструкций `int ledPin = D1` и `sound = analogRead(A0)`. Микроконтроллер ESP32 имеет 12-разрядный аналого-цифровой преобразователь (АЦП), в отличие от 10-разрядного АЦП микроконтроллера ESP8266, поэтому для микроконтроллера ESP32 инструкции `Soundmin = 1024` и `peak2peak = SoundMAX - Soundmin` должны быть изменены на `Soundmin = 4096` и `peak2peak = (SoundMAX - soundmin)/4`.

Шум в звуковом сигнале, возникающий при работе с микроконтроллером ESP8266, не был заметен при работе с микроконтроллером ESP32, поэтому медианная фильтрация не потребовалась. Количество светодиодов для включения или выключения определяется непосредственно из переменной `peak2peak` с помощью команды `LEDs = adjustVol*(peak2peak/1024.0)*LEDnumber-baseline`.

Листинг 4-2. Светодиодная RGB-лента и звук

```
#include <Adafruit_NeoPixel.h>        // include Adafruit NeoPixel lib
int LEDpin = D1;                      // define data pin
int LEDnumber = 30;                   // number of LEDS in strip
                                      // associate strip with NeoPixel lib
Adafruit_NeoPixel strip(LEDnumber, LEDpin, NEO_GRB + NEO_KHZ800);
                      // colors red, orange, yellow, green, blue, indigo, violet
int R[ ] = {255, 255, 255, 0, 0, 75, 238};
int G[ ] = { 0, 102, 153, 255, 0, 0, 130};
int B[ ] = { 0, 0, 0, 0, 255, 130, 238};
uint32_t color;                       // color is 32-bit
or unsigned long
#include <RunningMedian.h>            // include Running Median lib
RunningMedian samples = RunningM edian(7);
                                      // median filter sample size of 7
int sound, soundMax, soundMin, peak2peak, median, LEDs, val;
int adjustVol = 1;                    // initial volume and baseline
int baseline = 0;
int soundTime = 50;                   // sample sound time (ms)
unsigned long startTime;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  Serial.println("\nenter volume adjustment , baseline");
  strip.begin();                      // initialise LED strip
  strip.setBrightness(10);            // define LED brightness (1 - 255)
  strip.show();                       // sets all pixels to "off" as no color set
}
void loop()
{
  while(Serial.available()>0)         // adjust volume and baseline
  {
    adjustVol = Serial.parseInt();    // convert Serial buffer to integers
    baseline = Serial.parseInt();
    Serial.print("\nVolume ");Serial.print(adjustVol);
    Serial.print("\tBaseline ");Serial.println(baseline);
  }                                   // get new peak to peak value
  getSound();                         // number of LEDs to turn on
  LEDs = adjustVol*(median/1024.0)*LEDnumber-baseline;
  strip.clear();                      // turn all LEDs off
  delay(10);                          // allow time to switch off LEDs
  for (int i=0; i<LEDs; i++)
  {
    val = i/4;                        // four LEDs have the same color
    color = strip.Color(R[val], G[val],B[val]);
                                      // convert RGB values to color
    strip.setPixelColor(i, color);    // set the LED color
  }
  strip.show();                       // update LED states and colors
}
void getSound()                       // function for peak to peak value
{
  soundMax = 0;                       // initial values for minimum and
  soundMin = 1024;                    // maximum sound values
```

```
    startTime = millis();              // start of sampling period
    while(millis() - startTime < soundTime)
                                       // during sampling period
    {                                  // determine minimum and
       sound = analogRead(A0);         // maximum sound values
       if(sound > soundMax) soundMax = sound;
       else if(sound < soundMin) soundMin = sound;
    }
    peak2peak = soundMax - soundMin;   // peak to peak value
    samples.add(peak2peak);
    median = samples.getMedian();      // median peak to peak value
}
```

Для портативного дисплея на светодиодной ленте значения переменных `adjustVol` и `baseline` определяются выходным напряжением от двух потенциометров вместо ввода этих переменных через монитор последовательного порта. Микроконтроллер ESP32 имеет несколько аналоговых входов, но микроконтроллер ESP8266 имеет только один аналоговый вход. Эта проблема решается путем подключения мультиплексора.

ESP8266 и мультиплексор

Микроконтроллер ESP8266 имеет один аналоговый вход, который ограничивает количество устройств с аналоговым выходом, одновременно подключенных к плате ESP8266. Восьмиканальный аналоговый мультиплексор 74HC4051 позволяет подключать к плате ESP8266 до восьми аналоговых устройств. Сигналы от них выводятся через выходной канал мультиплексора, переключающийся тремя управляющими выводами мультиплексора, состояния которых задаются микроконтроллером ESP8266. Для управления восемью каналами требуется три вывода переключателя, так как $8 = 2^3$.

Выводы мультиплексора 74HC4051 пронумерованы от 1 до 16, а вырез или точка на краю корпуса мультиплексора указывает на местоположение выводов 1 и 16. Подключения к мультиплексору 74HC4051 показаны в табл. 4-2, при этом черта над символом $\bar{E}$ (разрешение входа) указывает на то, что вывод активен при низком уровне. Вывод 5V платы разработки ESP8266 питает мультиплексор 74HC4051, который имеет рабочее напряжение 2–10 В. Напряжение сигналов входного канала должно находиться между VCC и VEE, причем последний подключен к GND. Устройства с аналоговым выходом, подключенные к мультиплексору 74HC4051, не могут питаться от 5 В, так как контакты микроконтроллера ESP8266 не допускают подачи напряжения 5 В. На рис. 4-3 потенциометры питаются от вывода 3,3 В платы ESP8266, поэтому максимальное напряжение на выводе аналогового входа микроконтроллера также будет составлять 3,3 В.

Таблица 4-2 . Мультиплексор 74HC4051 и плата ESP8266

Компонент	Подключено к
74HC4051 выв. 3 (выход)	ESP8266 A0
74HC4051 выв. 6 $\bar{E}$ (разрешение входа)	
74HC4051 выв. 7 VEE	ESP8266 GND
74HC4051 выв. 8 GND	
74HC4051 выв. 9 S2 (упр. 2)	ESP8266 D2
74HC4051 выв. 10 S1 (упр. 1)	ESP8266 D1
74HC4051 выв. 11 S0 (упр. 0)	ESP8266 D0
74HC4051 выв. 16 VCC	ESP8266 5V
Переменные резисторы (левые выводы)	ESP8266 GND
Переменные резисторы (сигнальные выводы)	74HC4051 выв. 13, 14, 15
Переменные резисторы (правые выводы)	ESP8266 3V3

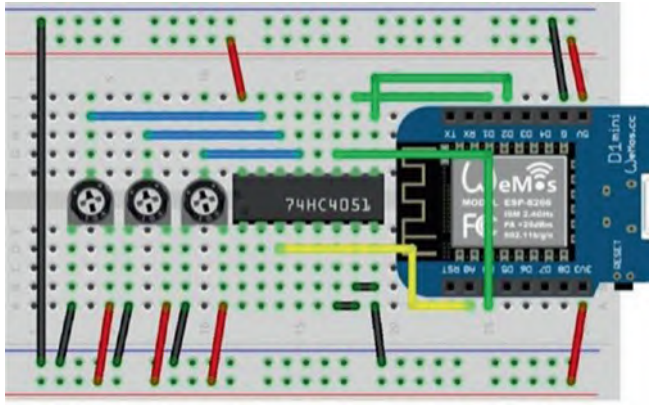


Рисунок 4-3. Мультиплексор 74HC4051 и плата LOLIN (WEMOS) D1 mini

Три управляющих вывода 11, 10 и 9 позволяют направлять до 8 аналоговых сигналов через выходной канал мультиплексора. Если к мультиплексору подключены только два аналоговых устройства, то управляющие выводы 10 и 9 подключаются к GND. Аналогично если к мультиплексору подключено не более четырех устройств, то управляющий вывод 9 подключается к GND. Шум от входных каналов мультиплексора, которые не подключены к какому-либо устройству, значительно снижается за счет задержки в 10 мс между установкой уровней на управляющих контактах и считыванием выходного канала мультиплексора.

Входной канал мультиплексора перенаправляется в выходной канал мультиплексора, когда уровни на управляющих входах равны двоичному представлению номера входного канала мультиплексора. Например, аналоговый входной сигнал на входе 3 мультиплексора выводится мультиплексором с уровнями на управляющих входах 011. Входные каналы 0–7 соответствуют выводам микросхемы мультиплексора 13, 14, 15, 12, 1, 5, 2 и 4 соответственно.

Подключение трех потенциометров к плате ESP8266 демонстрирует использование мультиплексора 74HC4051 (см. рис. 4-3). Скетч в листинге 4-3 устанавливает на управляющих выводах мультиплексора соответствующее двоичное число для подключения к выходному каналу мультиплексора. Команда `bitRead(number, j)` считывает *j*-й бит числа, начиная с младшего значащего бита (LSB), который равен нулевому биту. Значение бита, равное единице, соответствует высокому уровню (*HIGH*) на выводе. Мультиплексор 74HC4051 позволяет микроконтроллеру ESP8266 считывать «одновременно» три аналоговых значения потенциометра.

Листинг 4-3. Мультиплексор 74HC4051 и плата ESP8266

```
int Spin[] = {D0, D1, D2};            // multiplexer S0, S1 and S2
                                      // switch pins
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  for (int i=0; i<3; i++)
  {
    pinMode(Spin[i], OUTPUT);         // multiplexer pins as OUTPUT
    digitalWrite(Spin[i], LOW);       // set multiplexer pins to LOW
  }
}
void loop()
{
  for (int i=0; i<8; i++)             // 8 multiplexer pin combinations
  {                                   // set pins from bit sequence
    for (int j=0; j<3; j++) digitalWrite(Spin[j], bitRead(i, j));
    delay(10); // display readings
    Serial.print(analogRead(A0));S erial.print("\t");
    for (int j=0; j<3; j++) digitalWrite(Spin[j], LOW);
  }                                   // reset pins
  Serial.println();
  delay(1000);                        // delay between readings
}
```

Для управления светодиодной лентой звуковым сигналом значения переменных `adjustVol` и `baseline` определяются выходными напряжениями от двух потенциометров. Модуль усилителя электретного микрофона MAX4466 и потенциометры подключаются к микроконтроллеру ESP8266 через мультиплексор 74HC4051 (см. рис. 4-4). Вместо ввода переменных `adjustVol` и `baseline` через последовательный порт, как в листинге 4-2, светодиодной лентой управляют выходные напряжения потенциометров. Например, потенциометр регулировки напряжения, подключенный на вывод 13 мультиплексора, который является входным каналом мультиплексора 0, считывается инструкциями

```
for (int j=0; j<3; j++) digitalWrite(Spin[j], bitRead(0, j));
adjustVol = analogRead(A0)/50
```

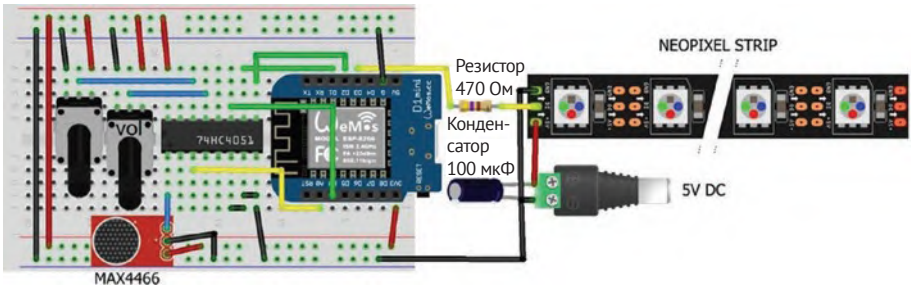


Рисунок 4-4. Управление светодиодной лентой потенциометрами и микрофоном

Потенциометр `adjustVol` приведет к значениям регулировки напряжения между 0 и 20. Аналогично значения `baseline` получаются с потенциометра `baseline`, подключенного к выводу 14 мультиплексора, который является входным каналом 1.

Часы на светодиодных кольцах

Два светодиодных кольца из 12 RGB-светодиодов каждый отображают часы и минуты, например на рисунке они показывают 16:40. Часы переключаются на следующий светодиод каждые пять минут. В момент переключения пьезопреобразователь издает мелодию звонка из двух звуков, а на минутном дисплее отображается радуга. Каждый час часовые и минутные кольца переливаются разными цветами. Часы со светодиодными кольцами питаются от вывода 5V платы ESP8266 с подключенным конденсатором емкостью 100 мкФ и резистором 470 Ом в соответствии с рекомендациями (см. рис. 4-5 и подключения в табл. 4-3).

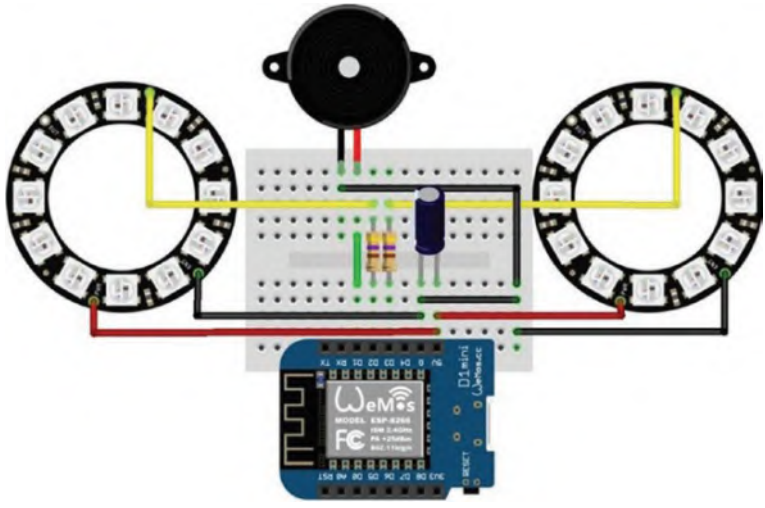


Рисунок 4-5. Часы на светодиодных кольцах

Таблица 4-3. Часы на светодиодных кольцах

Компонент	Подключено к	Также к
LED-кольца VCC	ESP8266 5V	Конденсатор 100 мкФ положит. вывод
LED-кольца GND	ESP8266 GND	Конденсатор 100 мкФ отрицат. вывод
LED-кольца IN	Резистор 470 Ом	ESP8266 D2, D3
Пьезопреобразователь VCC	ESP8266 D1	
Пьезопреобразователь GND	ESP8266 GND	

Если красный, зеленый и синий светодиоды включены на уровне яркости 1, 40 или 100, при шкале 1–255, то текущее потребление 12 светодиодов кольца составляет 15, 75 или 160 мА соответственно. Поэтому два светодиодных кольца с максимальной яркостью 100 могут безопасно питаться от 5V-контакта платы ESP8266.

Скетч в листинге 4-4 вычисляет количество часов и минут для включения светодиодов, отталкиваясь от времени, которое изначально задано с помощью времени Unix, представляющего собой количество секунд с 1 января 1970 года. Время Unix определяется с сайта www.epochconverter.com с опцией GMT[29], если требуется, и для его получения компиляция и загрузка программы задерживаются на 15 с. Например, время Unix, соответствующее 31 мая 2020 года, 17:20:51 по Гринвичу, составляло 1 590 945 651 с. Библиотека *Time* Майкла Марголиса (Michael Margolis) вычисляет количество часов и минут, основываясь на полученном времени Unix. Количество RGB-светодиодов, которое необходимо включить для отображения часов и минут, равно остатку, когда часы библиотеки времени делятся на 12 (поскольку наши часы не являются 24-часовыми часами), а минуты библиотеки времени делятся на 5, поскольку часы включают следующий светодиод каждые пять минут. Произвольная задержка в 22 с включается после обновления минутных светодиодов. Отображение часов требует 13 с для завершения, так что до 35 с пройдет, пока часы и минуты обновятся. Переменная *display* предотвращает повторение отображения часов в течение пятиминутного периода после очередного часа.

В первом разделе листинга 4-4 задается время Unix, включаются библиотеки и устанавливаются выводы для подключения светодиодного кольца. Требуются два экземпляра библиотеки *Adafruit NeoPixel*, по одному для каждого из двух светодиодных RGB-колец. Функция `setup` вызывает функцию библиотеки *Time* `setTime` для преобразования времени Unix в часы и минуты и функцию `getTime` для преобразования часов и минут в количество включенных функциями `LEDhours` и `LEDminutes` RGB-светодиодов. Обратите внимание, что инструкции для колец активируются с помощью команды `show()`. В функции `loop` вызыва-

[29] GMT (Greenwich Mean Time) – среднее географическое (астрономическое) время по Гринвичу, устаревшее представление точки отсчета для часовых поясов Земли. В настоящее время заменено на всемирное время UTC (Coordinated Universal Time, всемирное координированное время), определяемое по атомным часам. На практике GMT с точностью до долей секунды совпадает с UTC и с UT1, современным более точным эквивалентом GMT. – *Прим. перев.*

ется функция `getTime`, и при обновлении часа вызывается функция `phonering` с функцией `playTone`, генерирующей мелодию звонка. Мелодия звонка состоит из импульса 400 мс, паузы 200 мс, импульса 400 мс и двухсекундной паузы. Импульсы содержат по пять периодов прямоугольного колебания с частотой, чередующейся между 300 Гц и 350 Гц в течение 40 мс каждая. Функция Adafruit `rainbow` в течение двух периодов отображает различные цвета на минутном светодиодном кольце.

Для функции `playTone` команда `analogWriteFreq()` задает частоту ШИМ-модуляции[30] по умолчанию 1 кГц, которая может меняться при задании требуемой частоты в качестве параметра этой команды. 50%-ный коэффициент заполнения прямоугольного колебания, используемого для генерации звука, получается при установке переменной `value` в инструкции `analogWrite(pin, value)` равной 512, что составляет половину от 1023 ($2^{10} - 1$), поскольку цифроаналоговый преобразователь микроконтроллера ESP8266 имеет 10-битное разрешение.

Листинг 4-4. Светодиодные кольцевые часы

```
#include <TimeLib.h>                  // include Time library
unsigned long pctime = 1590945651;    // set Unix epoch time
#include <Adafruit_NeoPixel.h>        // include NeoPixel library
int LEDpinM = D3;                     // ring to display minutes
int LEDpinH = D2;                     // ring to display hours
int piezoPin = D1;                    // Piezo transducer pin
int LEDnumber = 12;                   // number of LEDs on ring
unsigned long interval = 1000;        // one sec time interval
int color = 0;                        // color flag for hour LED ring
int display = 0;                      // indicator hour display completed
String text;
int minutes, hours;
                     // associate ringM and ringH with Neopixel library
Adafruit_NeoPixel ringM(LEDnumber, LEDpinM, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel ringH(LEDnumber, LEDpinH, NEO_GRB + NEO_KHZ800);
void setup()
{
   setTime(pctime);                   // set time to Unix epoch time
   getTime();                         // get time parameters
   ringH.begin();                     // initialise hours LED ring
   ringH.setBrightness(1);            // set LED brightness (1 to 255)
   LEDhours();                        // function to display hours
   ringH.show();                      // update hours LED ring
   ringM.begin();
   ringM.setBrightness(1);
   LEDminutes();
   ringM.show();                      // update minutes LED ring
}
```

[30] ШИМ (PWM, *pulse-width modulation*) – простейший способ цифроаналогового преобразования, позволяющий управлять уровнем среднего напряжения за счет изменения коэффициента заполнения, т. е. ширины импульса относительно длительности периода колебания прямоугольной формы. Широко используется для управления светодиодными источниками и двигателями постоянного тока, может, как в данном случае, служить для формирования колебания заданной частоты и т. п. – *Прим. перев.*

```
void loop()
{
  getTime();                          // calculate hours and minutes
  if(minutes == 0 && display ==0)     // on the hour, run the display
  {
    display = 1;                      // flag to prevent repeat hour display
    phonering();                      // sound of phone ringtones
    rainbow();                        // rainbow of LED colors
    ringM.clear();                    // clear minutes LED ring
    ringM.show();
    color = 1 - color;                // change color flag
    ringH.clear();
    ringH.show();
    LEDhours();                       // update hours LED ring
  }
  if(minutes > 0) LEDminutes();       // update minutes LEDs
  for(int i=0; i<22; i++) delay(interval); // 22s delay
}
void getTime()                        // function to calculate hours and minutes
{
  hours = hour() % 12;                // convert 24hr time to 12hr time
  minutes= int(minute()/5);           // number of 5min intervals
  if(minutes > 0 ) display = 0;       // reset hour display flag
}
void LEDminutes()                     // function to turn on minute LEDs
{
  for(int i=0; i<minutes; i++)
  {                                   // set LED RGB values
    ringM.setPixelColor(i, ringM.Color(255*color,
    255*(1-color), 0));
  }
  ringM.show();
}
void LEDhours()                       // function to turn on hour LEDs
{
  for(int i=0; i<hours; i++)
  {
    ringH.setPixelColor(i, ringH.Color(255*(1-color),
    255*color, 0));
  }
  ringH.show();
}
void rainbow()                        // Adafruit rainbow function
{
  int cycle = 2;                      // two cycles of colors
  for(long Pixel1Hue=0; Pixel1Hue<cycle*65536; Pixel1Hue += 256)
  {
    for(int i=0; i<ringM.numPixels(); i++)
    {
      int pixelHue = Pixel1Hue + (i * 65536L / ringM.numPixels());
      ringM.setPixelColor(i, ringM.gamma32(ringM.
      ColorHSV(pixelHue)));
    }
    ringM.show();
    delay(10);
  }
}
```

```
void phonering()                        // function for phone ringtone
{
   for(int k=0; k<2; k++)               // two cycles
      {
      for(int j=0; j<2; j++)            // of two rings
         {
         for(int i=0; i<5; i++)         // with five repeats
         {
            playTone(300, 40);          // frequency 300Hz for 40ms
            playTone(350, 40);          // frequency 350Hz for 40ms
         }
         delay(200);                    // 200ms delay between rings
      }
      delay(2000);                      // 2s delay between cycles
   }
}
void playTone(int freq, int duration)
{
   analogWriteFreq(freq);               // frequency and duty cycle
   analogWrite(piezoPin, 512);          // to generate square wave
   delay(duration);                     // for duration (ms)
   analogWrite(piezoPin, 0);            // disable PWM on pin
}
```

Протокол NTP (Network Time Protocol)

Использование внутренних тактовых сигналов микроконтроллера ESP8266 или ESP32 для измерения времени удобно, но такие внутренние часы не всегда точны и имеют дрейф, зависящий от конкретной микросхемы и температуры окружающей среды. Внутренние часы микроконтроллера можно обновлять с помощью протокола NTP (Network Time Protocol) произвольно в 08:30 и 20:30, с информацией о времени, полученной из локального пула серверов. Подробная информация о пулах серверов доступна по адресу www.pool.ntp.org, в программе требуется указывать конкретный IP-адрес. Доступ к данным NTP осуществляется с помощью библиотеки *NTPtimeESP* Андреаса Шписса (Andreas Spiess), которую можно загрузить в виде zip-файла с github.com/Sensorslot/NTPtimeESP. Плата LOLIN (WeMos) D1 mini меньше по размерам, чем плата ESP32, но плату ESP32 также можно использовать (см. окончание этой главы).

Листинг 4-5 содержит дополнительные инструкции к первому разделу листинга 4-4 для подключения к сети Wi-Fi и доступа к локальному NTP с помощью библиотеки *NTPtimeESP*. Комментируются только дополнительные инструкции или инструкции, подлежащие замене, чтобы подчеркнуть несколько изменений, необходимых для этого скетча. В функции `setup` инструкция `setTime(pctime)` заменяется вызовом функции `getEpoch`, которая также вызывается в функции `loop` с помощью инструкции `if(hours == 8 && minutes == 30) getEpoch()`.

Функция `getEpoch` устанавливает соединение Wi-Fi, получает доступ к службе NTP для времени Unix, отключает соединение Wi-Fi и сбрасывает внутренние часы микроконтроллера. Коды подключения Wi-Fi и их значения

```
WL_CONNECTED 3
WL_CONNECT_FAILED 4
```

```
WL_CONNECTION_LOST 5
WL_DISCONNECTED 6
```

доступны через инструкцию `WiFi.status()`.

Листинг 4-5. Часы на светодиодных кольцах с обновлением времени через NTP для платы ESP8266

```
#include <Adafruit_NeoPixel.h>
int LEDpinL = D2;             // left ring to display hours
int LEDpinR = D3;             // right ring to display minutes
int piezoPin = D1;            // Piezo transducer pin
int LEDnumber = 12;
unsigned long interval = 1000;
int color = 0;
String text;
Adafruit_NeoPixel ringM(LEDnumber, LEDpinR, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel ringH(LEDnumber, LEDpinL, NEO_GRB + NEO_KHZ800);
#include <ESP8266WiFi.h>      // library to connect to Wi-Fi network
#include <ESP8266WebServer.h>
                              // library for web server functionality
ESP8266WebServer server;
                              // associate server with ESP8266WebServer library
char ssid[] = "xxxx";         // replace xxxx with Wi-Fi ssid
char password[] = "xxxx";     // replace xxxx with Wi-Fi password
#include <NTPtimeESP.h>       // include NTPtime library
                              // associate NTP with NTPtime library
NTPtime NTP("uk.pool.ntp.org"); // UK server pool for NTPtime
strDateTime dateTime;
unsigned long epoch;          // Unix epoch time
#include <TimeLib.h>          // include Time library
int minutes, hours;
int display = 0;
void setup()
{
  pinMode(piezoPin, OUTPUT); // Piezo transducer pin as output
  getEpoch();                // get Epoch time from NTP
  getTime();
  ringH.begin();
  ringH.setBrightness(1);
  LEDhours();
  ringH.show();
  ringM.begin();
  ringM.setBrightness(1);
  LEDminutes();
  ringM.show();
}
void loop()
{
  getTime();
  if(minutes == 0 && display ==0)
  {
    display = 1;
    phonering();
    rainbow();
    ringM.clear();
    ringM.show();
```

```
      color = 1 - color;
      ringH.clear();
      ringH.show();
      LEDhours();
   }
   if(minutes > 0) LEDminutes();
   if(hours == 8 && minutes == 30) getEpoch(); // update Epoch time
   delay(interval);                    // delay between time calculations
}
void getTime()                         // as in Listing 4-4
void getEpoch()                        // function to get NTP time
{
   WiFi.begin(ssid, password);         // wait for Wi-Fi connect
   while (WiFi.status() != WL_CONNECTED) delay(500);
   epoch = 0;
   for (int i=0; i<5; i++)             // five attempts to access NTP
   {
      delay(500);                      // delay between Wi-Fi connect
      dateTime = NTP.getNTPtime(0, 1); // and sourcing NTP data
      if(dateTime.valid)
      {
         epoch = dateTime.epochTime;   // NTP Epoch time obtained
         i = 5;                        // stop attempting connect to NTP
      }
   }
   WiFi.disconnect(true);              // disconnect Wi-Fi connection
   WiFi.mode(WIFI_OFF);                // switch off Wi-Fi connection
   setTime(epoch);                     // set internal clock to Epoch time
}
void LEDminutes()                      // as in Listing 4-4
void LEDhours()                        // as in Listing 4-4
void rainbow()                         // as in Listing 4-4
void phonering()                       // as in Listing 4-4
void playTone(int freq, int duration) // as in Listing 4-4
```

Интернет-часы и ESP32

В листинге 4-5 приведена программа интернет-часов, управляемых микроконтроллером ESP8266. Для микроконтроллера ESP32 инструкции по библиотеке Wi-Fi и веб-серверу

```
#include <ESP8266WiFi.h>               // library to connect to Wi-Fi network
#include <ESP8266WebServer.h>          // library for web server functionality
ESP8266WebServer server;               // associate server with library
```

заменяются на

```
#include <WiFi.h>                      // library to connect to Wi-Fi network
#include <WebServer.h>                 // library for web server functionality
WebServer server (80);                 // associate server with WebServer lib
```

Генерация звука с помощью пьезопреобразователя отличается для микроконтроллеров ESP8266 и ESP32, поэтому функция `playTone` для микроконтроллера ESP8266 заменена функцией `playToneESP32`:

```
void playToneESP32(int freq, int duration)
{
  ledcSetup(channel, freq, 10);      // 10-bit resolution
  ledcWrite(channel, 512);           // square wave with 50% duty cycle
  delay(duration);                   // for duration (ms)
  ledcWrite(channel, 0);
}
```

Третьим параметром инструкции `ledcSetup` является уровень разрешения ШИМ 8, 10, 12 или 15 бит. Для соответствия листингу 4-5 функция `playToneESP32` использует 10-разрядное разрешение с 1024 значениями и 50%-ным рабочим циклом, получаемым при значении 512. Вывод пьезопреобразователя сопоставляется с переменной `channel` с помощью команды `ledcAttachPin(piezoPin, channel)` в функции `setup`, при этом канал определяется значением `int channel = 0`. Номера контактов подключения пьезопреобразователя и светодиодного кольца D1, D2 и D3 должны быть изменены на контакты платы ESP32 25, 26 и 27 соответственно.

Итоги

Цвет каждого светодиода на светодиодной ленте WS2812 5050 RGB определялся индивидуально, причем диапазон RGB-цветов охватывал диапазон светодиодной ленты. Модуль усилителя электретного микрофона измерял уровни звука, чтобы определить количество и цвет RGB-светодиодов для включения в светодиодной ленте. Ограничение одного аналогового входного вывода микроконтроллера ESP8266 было устранено с помощью включения мультиплексора 74HC4051 для доступа к нескольким аналоговым устройствам. Микроконтроллер ESP8266 «одновременно» получал доступ к напряжениям от двух потенциометров для управления настройками на светодиодной ленте и модуля усилителя электретного микрофона.

Интернет-часы построены на двух светодиодных кольцах для отображения часов и минут. При наступлении очередного часа пьезопреобразователь издает мелодию звонка из двух тонов, за которой следует демонстрация радуги на одном из светодиодных колец. Wi-Fi-функциональность микроконтроллеров ESP8266 и ESP32 обеспечивала доступ к сервису протокола сетевого времени (NTP) дважды за каждые 24 часа для обновления внутренних часов микроконтроллера и поддержания точности LED-часов.

Перечень компонентов

- Платы микроконтроллера ESP8266 LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: платы DEVKIT DOIT или NodeMCU
- RGB LED-кольцо: 2 шт.
- RGB LED-лента
- Резистор: 470 Ом 2 шт.
- Конденсатор: 100 мкФ
- Пьезопреобразователь
- Модуль усилителя электретного микрофона: MAX4466
- Мультиплексор: 74HC4051

Глава 5

MP3-плеер

Мини-MP3-плеер DFPlayer со встроенным модулем карты micro-SD либо используется в качестве автономного модуля с питанием от батареи и кнопочным управлением, либо подключается к микроконтроллеру для более полного управления функциями. MP3-плеер имеет каналы для подачи звука на наушники или на вход усилителя (цифроаналоговые преобразователи R и L) с выводом на динамики (SPK1 и SPK2). DFPlayer оборудован модулем последовательной связи (RX и TX), а также четырьмя контрольными выводами (IO1, IO2, ADKEY1 и ADKEY2) с активным низким уровнем (см. рис. 5-1). Вывод *BUSY* устанавливается в низкий уровень (LOW) при воспроизведении аудиофайла, в остальное время находится в высоком уровне (HIGH). В MP3-плеере есть музыкальный эквалайзер с шестью фиксированными настройками.

Рисунок 5-1. Мини-MP3-плеер DFPlayer

Когда MP3-плеер работает автономно, короткое нажатие на тактовые кнопки, подключенные к выводам управления *IO1* или *IO2*, воспроизводит предыдущий или следующий трек соответственно, в то время как длительное нажатие уменьшает или увеличивает громкость. Нажатие на кнопку, подключенную к управляющим выводам *ADKEY1* или *ADKEY2* воспроизводит первую или пятую дорожку соответственно.

Для этой главы *mp3*-аудиофайлы хранятся в папке с именем *mp3* на карте micro-SD, отформатированной в формате FAT32 с объемом памяти до 32 ГБ. Имена 01–99 разрешены для папок, содержащих аудиофайлы с именами от *001.mp3* до *255.mp3*.

Рисунок 5-2 иллюстрирует кнопочное управление автономным MP3-плеером с динамиком мощностью менее 3 Вт, подключенным к выводам *SPK1* и *SPK2* через аудио- и GND-контакты аудиоразъема (соединения приведены в табл. 5-1). Вместо управления MP3-плеером с помощью тактовых кнопок соответствующий управляющий вывод напрямую подключается к выводу GND.

Обратите внимание, что при подаче питания на MP3-плеер красный индикатор загорается после подключения вывода *IO1* или *IO2* к GND и выключается по окончании воспроизведения трека.

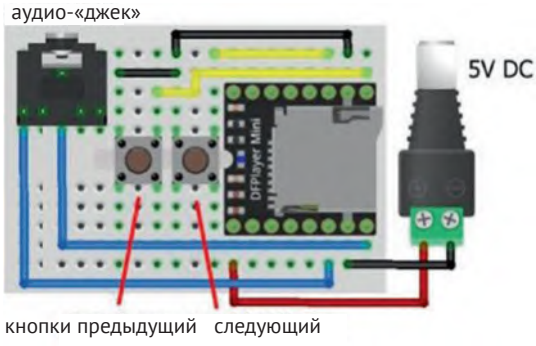


Рисунок 5-2. Автономный MP3-плеер

Таблица 5-1. Автономный MP3-плеер

Компонент	Подключено к
MP3 player VCC	Внешний источник 5V
MP3 player GND	Внешний источник GND
MP3 player SPK1	Аудиовыход
MP3 player SPK2	Аудиовыход
MP3 player IO1 (предыдущий трек или уменьшение громкости)	Правый вывод кнопки
MP3 player IO2 (следующий трек или увеличение громкости)	Правый вывод кнопки
Левые выводы кнопок	GND

Команды управления для MP3-плеера

MP3-плеер управляется командами управления, передаваемыми с помощью последовательного порта UART. Десять компонентов управляющей команды – это начальный бит (*start bit*), версия (*version*), количество байтов (*number of bytes*), команда (*command*), обратная связь (*feedback*), два бита параметра (*parameter*[1,2]), два бита контрольной суммы (*checksum*[1,2]) и конечный бит (*finish bit*). Начальный и конечный биты в шестнадцатеричном формате равны 0x7E и 0xEF, количество байтов равно 0x06, обратная связь равна 0x00, а версия равна 0xFF. Управляющая команда имеет два параметра, но обычно значение *parameter*[1] равно нулю. Контрольная сумма – это отрицательная сумма всех компонентов без начального и конечного битов. Контрольная сумма форматируется как старший и младший байты. Когда управляющая команда передается на MP3-плеер, буфер ответа содержит десять компонентов, из которых *parameter*[2] содержит данные.

Например, управляющая команда для воспроизведения четвертого трека равна 0x03 с *parameter*[2], равным 0x04 (см. табл. 5-2). Контрольная сумма равна –(0xFF + 0x06 + 0x03 + 0x00+ 0x00 + 0x04) = –(255 + 6 + 3 + 0 + 0 + 4) = –268,

что соответствует 2^{16} – 268 = 65268 = 0xFEF4, со старшим и младшим байтами 0xFE и 0xF4 соответственно. Компоненты управляющей команды в шестнадцатеричном формате – это стартовый бит (*start bit*) = 0x7E; версия (*version*) = 0xFF; длина (*number of bytes*) = 0x06; команда (*command*) = 0x03; обратная связь (*feedback*) = 0x00; первый параметр (*parameter*[1]) = 0x00; второй параметр (*parameter*[2]) = 0x04; контрольная сумма *checksum*[1] = 0xFE и *checksum*[2] = 0xF4; бит завершения (*finish bit*) = 0xEF.

Перечень команд управления MP3-плеером приведен в табл. 5-2 из руководства, доступного по адресу usermanual.wiki/Pdf/DFPlayer20Mini20Manual.1647715389/pdf. Команда 0x03 воспроизводит *N*-й трек из треков, упорядоченных по времени их загрузки на карту micro-SD. Команда 0x12 воспроизводит дорожку с именем файла *000N XXX* независимо от временного порядка загрузки аудиофайла на карту. Имя аудиофайла не обязательно должно отражать порядок загрузки аудиофайлов на карту. Например, на micro-SD были загружены четыре аудиофайла карты в порядке *0014 ABC*, *0012 AAA*, *0011 XYZ* и *0013 PQR*, которые MP3-плеер размещает как треки 1, 2, 3 и 4 соответственно. Команды *(0x12, 13)* и *(0x03, 4)* будут воспроизводить аудиофайл *0013 PQR*, который для MP3-плеера является треком номер 4.

Таблица 5-2. Команды управления MP3-плеером

Команда	Действие
0x01	Следующий трек
0x02	Предыдущий трек
0x03	Воспроизвести трек номер *N*
0x04	Увеличение громкости на один уровень
0x05	Уменьшение громкости на один уровень
0x06	Установите уровень громкости на уровень *N* в диапазоне от 0 до 30
0x07	Эквалайзер (обычный, поп, рок, джаз, классика, базовый)
0x0D	Воспроизвести текущий трек
0x0E	Текущий трек пауза
0x12	Воспроизвести трек номер *N*
0x18	Воспроизвести в случайном порядке, начиная с дорожки 1
0x43	Получить уровень громкости
0x46	Получить номер версии программного обеспечения
0x48	Получить количество файлов на SD-карте
0x46	Получить номер трека

Управление MP3-плеером с помощью микроконтроллера

Подключение MP3-плеера к микроконтроллеру ESP8266 или ESP32 обеспечивает значительно бóльшую функциональность, чем автономный MP3-плеер. MP3-плеер работает при напряжении 3,3–5 В. Если MP3-плеер питается от вы-

вода 3V3 платы ESP8266 или ESP32, то выводы передачи (TX) и приема (RX) порта UART функционируют при напряжении 3,3 В. Если MP3-плеер питается от 5 В, то для сопряжения вывода TX плеера с 3,3-вольтовым выводом RX микроконтроллера ESP8266 или ESP32 потребуется преобразователь логического уровня для снижения напряжения 5 В. Напряжение также можно снизить с помощью делителя напряжения, состоящего из резисторов 5 кОм и 10 кОм, как описано в главе 16 («Генерация сигнала»).

Контакты RX и TX платы ESP8266 необходимы для связи с монитором последовательного порта Arduino IDE, поскольку на нем отображается информация, гарантирующая, что скетч работает должным образом. Поэтому микроконтроллер ESP8266 взаимодействует с MP3-плеером с помощью программного последовательного порта, с использованием встроенной в Arduino IDE библиотеки *SoftwareSerial*. Преимущества компактной платы LOLIN (WeMos) D1 mini с мощным процессором и функциональностью Wi-Fi в некоторой степени компенсируются ограничением пригодных для управления контактов, поскольку вывод D8 имеет заземляющий резистор, а вывод D0 не имеет функции прерывания. Контакты D4, D3, D2 и D8 платы ESP8266 определены как управляющие выводы для воспроизведения следующей дорожки, увеличения громкости, уменьшения громкости и управления музыкальным эквалайзером соответственно (см. рис. 5-3 и соединения в табл. 5-3). Управляющие контакты D4, D3 и D2 подключены к GND, в то время как вывод D8, который имеет встроенный заземляющий резистор, для управления музыкальным эквалайзером подключен к 3V3.

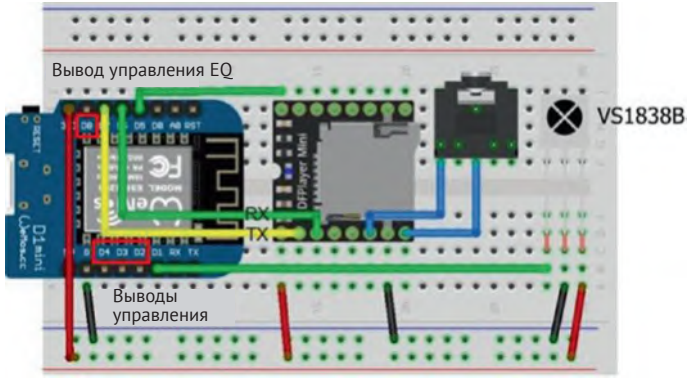


Рисунок 5-3. MP3-плеер и плата LOLIN (WeMos) D1 mini

Таблица 5-3. Подключения MP3-плеера к платам ESP8266 и ESP32

Компонент	ESP8266	ESP32
MP3-плеер VCC	3V3	3V3
MP3-плеер RX	D7	TX2 (GPIO 17)
MP3-плеер TX	D6	RX2 (GPIO 16)
MP3-плеер SPK1	Аудиовыход	Аудиовыход
MP3-плеер GND	GND	GND

Окончание табл. 5-3

Компонент	ESP8266	ESP32
MP3-плеер SPK2	Аудиовыход	Аудиовыход
MP3-плеер BUSY	D5	GPIO 34
IR-приемник OUT	D1	GPIO 23
IR-приемник GND	GND	GND
IR-приемник VCC	3V3	3V3

В отличие от ESP8266, плата ESP32 имеет множество выводов GPIO, при этом платы ESP32 DEVKIT DOIT с 30 выводами и NodeMCU с 36 выводами имеют два и три порта UART соответственно (см. главу 21 «Микроконтроллеры»). MP3-плеер с платой ESP32 DEVKIT DOIT показан на рис. 5-4, а подключения приведены в табл. 5-3.

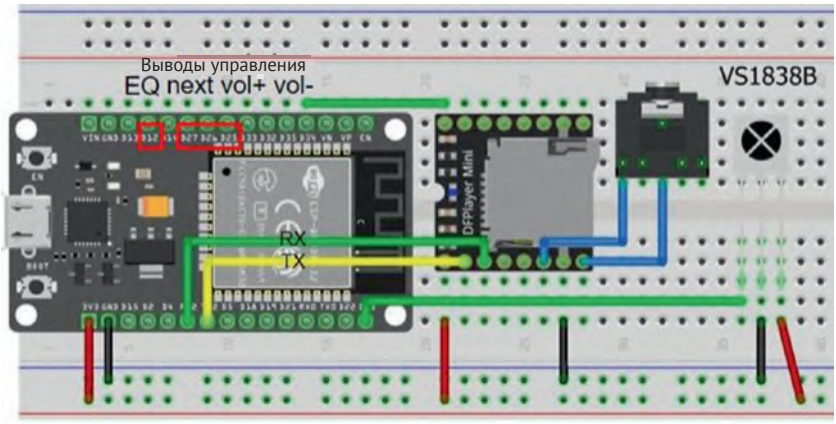


Рисунок 5-4. MP3-плеер с платой ESP32 DEVKIT DOIT

Скетч в листинге 5-1 является основой для последующего скетча (листинг 5-3), использующего для управления MP3-плеером инфракрасный пульт. В функции `setup` скетча определяются минимальный и максимальный номера файлов. Например, если файлы с названиями *0009 abc.mp3* и *0013 xyz.mp3* имеют наименьший и наибольший номера файлов, то переменным `fileMin` и `fileMax` присваиваются значения 9 и 13 соответственно.

В первом разделе скетча определяются управляющие выводы, имена состояний музыкального эквалайзера, шаблон команды управления и имена аудиофайлов. Функция `setup` подключает внутренние подтягивающие резисторы к управляющим выводам и определяет прерывание, активируемое выводом *BUZY* MP3-плеера, при его изменении с низкого уровня (`LOW`) на высокий (`HIGH`), когда трек заканчивается.

Процедура обработки прерывания (ISR) устанавливает для переменной `finish` значение, равное единице, что указывает на завершение воспроизведения аудиофайла. Чтобы определить, вставлена ли в MP3-плеер карта micro-SD, количество файлов на карте подсчитывается с помощью команды 0x48; и если аудио-

файлы отсутствуют, то через монитор последовательного порта выводится сообщение. Устанавливается значение громкости, воспроизводится первый файл, и таймер сбрасывается для измерения времени воспроизведения аудиофайла.

Функция `loop` отслеживает активность управляющих выводов и вызывает функцию `cmd` (команда). Четыре управляющих вывода используют команды 0x12 для воспроизведения следующего аудиофайла и 0x4C для получения соответствующего номера трека, команду 0x04 или 0x05 для увеличения или уменьшения уровня громкости, команду 0x43 для получения значения громкости и команду 0x07 для изменения состояния музыкального эквалайзера.

Следующий аудиофайл воспроизводится, когда `nextPin` оказывается подключен к GND или текущий файл завершается с помощью инструкции `if(digitalRead(nextPin) == LOW || finish == 1)`. Аудиофайлы проверяются при воспроизведения каждого аудиофайла в течение фиксированного периода времени, например 5 с, с помощью замены первой инструкции функции `loop` на `if(digitalRead(nextPin) == LOW || millis()- timed > 5000)`.

Функция `cmd` строит контрольную сумму на основе значений *command* и *parameter[2]*, поскольку *parameter[1]* равен нулю, и разбивает контрольную сумму на старший байт и младший байт для передачи на MP3-плеер. Функции `highByte` и `lowByte` – это функции в среде Arduino IDE. Ответ от MP3-плеера хранится в массиве `buffer[ ]`, причем элемент `buffer[6]` содержит значение данных. При вызове команды 0x4C для получения соответствующего номера дорожки отображается как номер дорожки, так и имя аудиофайла.

В листинге 5-1 приведена программа для микроконтроллера ESP8266. Последовательные порты платы ESP32 обеспечивают связь с несколькими устройствами без необходимости использования библиотек для обеспечения их функциональности. Команда `Serial2.begin(baud, SERIAL_8N1, RXD2, TXD2)` устанавливает связь на втором последовательном порту с выводами *RXD2* и *TXD2* со скоростью передачи данных, определяемой параметром `baud`.

При использовании микроконтроллера ESP32 следующие инструкции для микроконтроллера ESP8266

```
#include <SoftwareSerial.h>          // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);      // define SoftSer RX, TX pins
SoftSer.begin(9600);                 // SoftwareSerial baud rate
for (int i=0; i<10; i++) SoftSer.wr ite(serialCom[i]);
                                     // transmit to or receive from MP3
for (int i=0; i<10; i++) buffer[i] = SoftSer.read();
```

заменяются инструкциями

```
Serial2.begin(9600, SERIAL_8N1, 16, 17);   // define RX2, TX2
for (int i=0; i<10; i++) Serial2.write(serialCom[i]);
                                     // transmit to or receive from MP3
for (int i=0; i<10; i++) buffer[i] = Serial2 .read();
```

и инструкция `pinMode(busyPin, INPUT)` включается в функцию `setup`. Выводы *BUZY*, D5 и управляющие выводы D4, D3 и D2 заменяются на GPIO 34 и на три других подходящих вывода GPIO, таких как GPIO 27, GPIO 26 и GPIO 25. Для соответствия скетчу микроконтроллера ESP8266 вывод музыкального эквалайзера D8 заменен на GPIO 12, который имеет встроенный заземляющий резистор.

Листинг 5-1. MP3-плеер

```
#include<SoftwareSerial.h>           // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);      // define SoftSer RX, TX pins
int nextPin = D4;
int volUp = D3;
int volDown = D2;                    // define control pins
int EQpin = D8;
int busyPin = D5;
int EQstate = 0;                     // equaliser settings
String EQ[] = {"Normal","Pop","Rock","Jazz","Classic","Bass"};
unsigned long timed = 0;
unsigned int checksum;
byte highChk, lowChk;                // control command template
byte serialCom[10] = {0x7E,0xFF,0x06,0x00,0x00,0x00,0x00,0x00,
0x00,0xEF};
// start version length CMD feedback para[1, 2] checksum[high, low] end
byte buffer[10];
int fileMin = 9;                     // lowest and highest file number
int fileMax = 13;                    // on micro SD card
int file = fileMin;
String fileName[] = {                // file names in order loaded on SD card
"0012 Nina Simone - My baby just cares for me",
"0011 Reamonn - Supergirl",
"0013 Spider Murphy Gang - Ich Grüsse Alle Und Den Rest Der Welt",
"0009 Proclaimers - I'm Gonna Be (500 Miles)",
"0010 Railroad Earth - The Good Life"
};
volatile int finish;                 // variable in loop and ISR functions
int track;
void setup()
{
  Serial.begin(115200);              // Serial Monitor baud rate
  SoftSer.begin(9600);               // software Serial baud rate
  pinMode(nextPin, INPUT_PULLUP);    // control pins use internal
  pinMode(volUp, INPUT_PULLUP);      // pull-up resistors
  pinMode(volDown, INPUT_PULLUP);
                                     // interrupt finished, BUSY pin HIGH
  attachInterrupt(digitalPinToInterrupt(busyPin), finished,
  RISING);
  cmd(0x48, 0);                      // get number of files on SD card
  cmd(0x06, 10);                     // set volume to 10 (range 0 - 30)
  cmd(0x43, 0);                      // get volume value
  cmd(0x12, file);                   // play first audio file
  finish = 0;                        // set finish variable
  cmd(0x4C, 0);                      // get track number
  timed = millis();                  // start timer
}
void loop()
{                                    // next file selected or current file ended
  if(digitalRead(nextPin) == LOW || finish == 1)
  {
    file = file+1;                   // increment file name
    if(file > fileMax) file = f ileMin;
                                     // constrain file name <= fileMax
    cmd(0x12, file);                 // play next audio file
```

```
      finish = 0;                       // set finish variable
      cmd(0x4C, 0);                     // get track number
    }
    else if(digitalRead(volUp) == LOW) // increase volume is selected
    {
      cmd(0x04, 0);                     // increase volume
      cmd(0x43, 0);                     // get volume value
    }
    else if(digitalRead(volDown) == LOW)
    {                                   // decrease volume is selected
      cmd(0x05, 0);                     // decrease volume
      cmd(0x43, 0);
    }
    else if(digitalRead(EQpin) == HIGH)
                                        //change equaliser is selected
    {                                   // when pin state is HIGH
      EQstate = EQstate+1;              // increment equaliser
      if(EQstate > 5) EQstate = 0;      // constrain equaliser value
      Serial.println(EQ[EQstate]);
      cmd(0x07, EQstate);               // change equaliser setting
    }
  }
  void cmd(byte CMD, byte param2)       // command function
  {
    delay(100);                         // delay to debounce button
    checksum = -(0xFF + 0x06 + CMD + 0x0 0 + 0x00 + param2);
                                        // build checksum
    highChk = highByte(checksum);       // split checksum into
    lowChk = lowByte(checksum);         // high byte and low bytes
    serialCom[3] = CMD;
    serialCom[6] = param2;              // command components
    serialCom[7] = highChk;
    serialCom[8] = lowChk;              // transmit command to MP3
    for (int i=0; i<10; i++) SoftSer.write(serialCom[i]);
    delay(100);                         // receive command from MP3
    for (int i=0; i<10; i++) buffer[i] = SoftSer.read();
    delay(100);
    if(CMD == 0x12)                     // play next audio file
    {
      Serial.print("finished track ");Serial.print(track);
      Serial.print("\ttime");
      Serial.print((millis() - timed)/1000);
      Serial.println("s");              // display audio play time
      timed= millis();                  // reset timer
    }
    else if(CMD == 0x43)                // get volume
    {
      Serial.print("volume ");          // display volume value
      Serial.println(buffer[6]);
    }
    else if(CMD == 0x48)                // get number of files on SD card
    {
      if(buffer[6]<2)                   // no audio files present
      {
        Serial.println("SD card not inserted ");
        Serial.println("insert SD card and reset microcontroller");
```

```
        for(;;) delay(1000);  // do nothing
      }
    }
    else if(CMD == 0x4C)       // get track number
    {
      track = buffer[6];       // display track number of file
      Serial.print("playing track ");Serial.print(track);
      Serial.print("\t\t");Serial.println(fileName[track-1]);
    }                          // array starts at [0], but track starts at [1]
}
IRAM_ATTR void finished()      // ISR
{
  finish = 1;                  // set finish variable
}
```

Инфракрасный пульт дистанционного управления MP3-плеером

MP3-плеер управляется инфракрасным пультом дистанционного управления (IR Remote) и инфракрасным приемником (IR-приемником) VS1838B вместо подключения управляющих контактов к GND, как было показано в листинге 5-1. Соединения для IR-приемника VS1838B приведены в табл. 5-3 и показаны на рис. 5-3 и 5-4. Для ESP8266 рекомендуется библиотека *IRremoteESP8266* Дэвида Конрана (David Conran), Себастьена Варина (Sebastien Warin), Марка Сабо (Mark Szabo) и Кена Ширриффа (Ken Shirriff), доступная в среде Arduino IDE. Для микроконтроллера ESP32 рекомендуется использовать библиотеку *IRremote* от Кена Ширриффа (Ken Shirriff). ZIP-файл, содержащий библиотеку *IRremote*, загружается с github.com/z3t0/Arduino-IRremote. Библиотека *IRremote*, доступная в Arduino IDE, не всегда является последней версией. При нажатии кнопки дистанционного управления инфракрасный приемник получает сигнал, который декодируется и сопоставляется с соответствующим управляющим выводом. Например, декодированный 32-разрядный инфракрасный сигнал 0xFF18E7 для кнопки 2 инфракрасного пульта дистанционного управления показан на рис. 5-5.

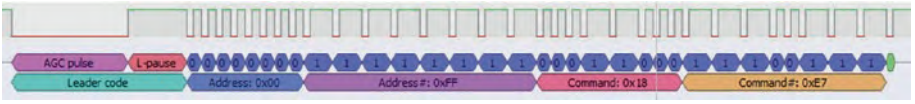


Рисунок 5-5. Сигнал ИК-пульта дистанционного управления

Листинги 5-2 и 5-3 декодируют инфракрасные сигналы и отображают принятые сигналы в шестнадцатеричном формате для микроконтроллеров ESP8266 и ESP32 соответственно. В листинг 5-3 включены только типы декодирования NEC и Sony, но файл библиотеки *IRremote.h* содержит полный список типов декодирования для включения в скетч. В листинге 5-4 используются шестнад-

цатеричные коды, соответствующие нажатию следующего аудиофайла, увеличению громкости, уменьшению громкости и кнопки «стоп» на инфракрасном пульте дистанционного управления Sony: 0x8D1, 0x491, 0xc91 и 0x1D1 соответственно. Аналогично шестнадцатеричные коды для кнопок дистанционного управления 1–5 были 0x011, 0x811, 0x411, 0xC11 и 0x211; они используются командой воспроизведения дорожек с номерами от 1 до 5.

Листинг 5-2. Декодирование инфракрасных сигналов для платы ESP8266

```
#include <IRutils.h>                  // include IRutils library
int IRpin = D1;                       // IR receiver pin
int BufferSize = 1024;                // longer signal length
int Timeout = 50;                     // block repeat signals
IRrecv irrecv(IRpin, BufferSize, Timeout, true);
decode_results reading;               // IRremote reading
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  irrecv.enableIRIn();                // initialise the IR receiver
}
void loop()
{
  if (irrecv.decode(&reading))        // read pulsed signal
  Serial.print(resultToHumanReadableBasic(&reading));
}                                     // display signal information
```

Листинг 5-3. Декодирование инфракрасных сигналов для платы ESP32

```
#include <IRremote.h>                 // include IRremote library
int IRpin = 23;                       // IR receiver pin
IRrecv irrecv(IRpin);
decode_results reading;
void setup()
{
  Serial.begin(115200);
  irrecv.enableIRIn();
}
void loop()
{
  if (irrecv.decode(&reading))
  {
            if(reading.decode_type == NEC) Serial.print("NEC: ");
    else if(reading.decode_type == SONY) Serial.print("Sony: ");
    else Serial.print("Other: ");
    Serial.print(reading.value, HEX);
    Serial.print("\tBits: ");        // display signal HEX code
    Serial.println(reading.bits);    // and bit number
    delay(200);                      // delay before next IR signal
    irrecv.resume();                 // receive the next value
  }
}
```

Инфракрасные сигналы сопоставляются с функциями управляющего вывода с использованием инструкций, связанных с управляющим выводом в листинге 5-1. Например, инструкция по увеличению громкости `if(digitalRead(volUp) ==`

LOW) заменяется инструкцией if(reading.value == 0x491). Сопоставление кнопки дистанционного управления с воспроизведением определенной дорожки использует инструкции switch ... case, а не инструкции if ... else, при этом каждая инструкция case отображает инфракрасный сигнал на номер дорожки. Инструкции по замене команд управляющего вывода в листинге 5-1 сигналами дистанционного управления приведены в листинге 5-4.

В первом разделе скетча для микроконтроллера ESP8266 замените

Листинг 5-4. Инфракрасный пульт дистанционного управления MP3-плеером

```
int nextPin = D4;
int volUp = D3;
int volDown = D2;                    // define control pins
int EQpin = D8;
```

на следующие инструкции:

```
#include <IRutils.h>                // include IR library
int IRpin = D1;                     // IR receiver pin
int BufferSize = 1024;              // longer signal length
int Timeout = 50;                   // block repeat signals
IRrecv irrecv(IRpin, BufferSize, Timeout, true);
decode_results reading;             // IRremote reading
```

Для микроконтроллера ESP32 новые инструкции будут такими:

```
#include <IRremote.h>
int IRpin = 23;
IRrecv irrecv(IRpin);
decode_results reading;
```

Для обоих микроконтроллеров замените int track на int track, oldTrack.

В функции setup замените

```
pinMode(nextPin, INPUT_PULLUP);     // control pins use internal
pinMode(volUp, INPUT_PULLUP);       // pull-up resistors
pinMode(volDown, INPUT_PULLUP);
```

на

```
irrecv.enableIRIn();                // initialise the IR receiver
```

Функция loop целиком заменяется следующими инструкциями. Заметим, что инструкция irrecv.resume() требуется только для микроконтроллера ESP32:

```
void loop()
{
  if(finish == 1)                   // current audio file ended
  {
    cmd(0x01, 0);                   // play next track
    finish = 0;                     // set finish variable
    cmd(0x4C, 0);                   // get track number
  }
  if(irrecv.decode(&reading))       // read pulsed signal
  {
    if(reading.value == 0x8D1)      // next audio file is selected
```

```
      {
        file = file+1;                  // increment file name
        if(file > fileMax) file = fi leMin;
                                        // constrain file name < fileMax
        cmd(0x12, file);                // play next audio file
        finish = 0;                     // set finish variable
        cmd(0x4C, 0);                   // get track number
      }
      else if(reading.value == 0x491) // increase volume is selected
      {
        cmd(0x04, 0);                   // increase volume
        cmd(0x43, 0);                   // get volume value
      }
      else if(reading.value == 0xC91) // decrease volume is selected
      {
        cmd(0x05, 0);                   // decrease volume
        cmd(0x43, 0);                   // get volume value
      }
      else if(reading.value == 0x1D1) //change equaliser is selected
      {
        EQstate = EQstate+1;            // increment equaliser
        if(EQstate > 5) EQstate = 0;  // constrain equaliser value
        Serial.println(EQ[EQstate]);
      cmd(0x07, EQstate);               // change equaliser setting
      }
      else
      {
        switch(reading.value)           // switch case for selected track
        {                               // map remote signal to play track
          case 0x011: track = 1; break;
          case 0x811: track = 2; break;
          case 0x411: track = 3; break;
          case 0xC11: track = 4; break;
          case 0x211: track = 5; break;
        }
        cmd(0x03, track);               // play track
        finish = 0;                     // set finish variable
        cmd(0x4C, 0);                   // get track number
      }
//      irrecv.resume(); // for ESP32, receive next value
      }
   delay(100);
}
```

В функции `cmd` команда `if(CMD == 0x12)` для воспроизведения следующего аудиофайла заменяется инструкцией `if(CMD == 0x01 ||CMD == 0x03 || CMD == 0x12)` для воспроизведения следующего трека или выбранного трека / выбранного аудиофайла. Две инструкции далее, инструкция `Serial.print(track)` заменяется на `Serial.print(oldTrack)`. В группе инструкций для получения номера трека после инструкции `oldTrack = track[6]` добавляется инструкция `oldTrack = track`.

Создание треков и две системы сигнализации

Веб-сайт www.fromtexttospeech.com создает MP3-файлы для речи, соответствующей тексту, введенному в текстовое поле онлайн, с различными голосами на

нескольких доступных языках. Имена аудиофайлов обозначаются комбинацией цифр и текста, например *"0001 alarm on"* («тревога включена»), так что доступ к аудиофайлам осуществляется независимо от порядка загрузки аудиофайлов на карту micro-SD. Приложением создаваемых пользователем треков является система сигнализации с ультразвуковым датчиком расстояния HC-SR04 или пассивным инфракрасным PIR-датчиком для обнаружения движения с конкретными объявлениями с помощью MP3-плеера при срабатывании датчика (см. рис. 5-6 и 5-7).

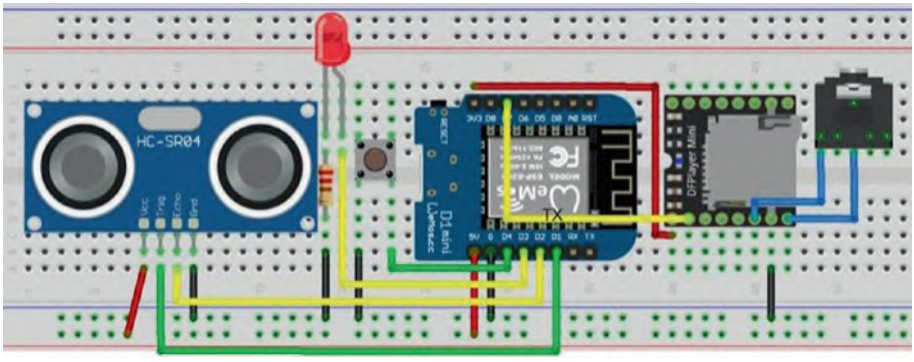


Рисунок 5-6. Сигнализация на MP3-плеере с платой LOLIN (WeMos) D1 mini

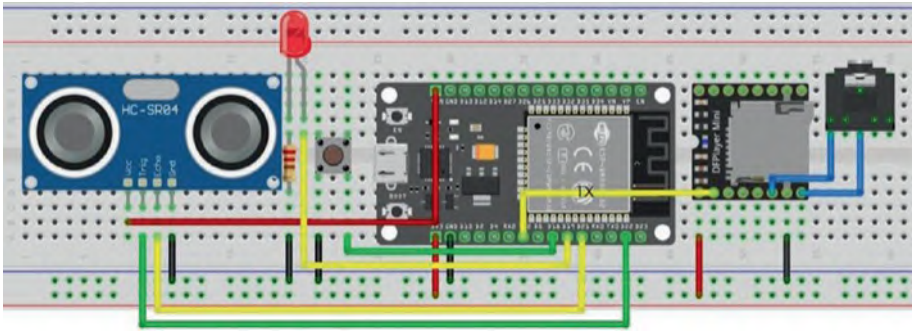


Рисунок 5-7. Сигнализация на MP3-плеере с платой ESP32 DEVKIT DOIT

Скетч в листинге 5-5 включает ультразвуковой датчик расстояния HC-SR04 для обнаружения изменения расстояния, например при открытии двери, и если включена сигнализация, то MP3-плеер выдает объявление. Расстояние в сантиметрах между датчиком и объектом составляет половину времени эха, измеряемого в микросекундах, умноженного на 0,0343, при условии что скорость звука составляет 343 м/с[31]. Первый раздел скетча определяет ультразвуковой датчик расстояния HC-SR04 и контакты для кнопки активации прерывания `alarmISR`, которая включает или выключает сигнал тревоги и светодиодный индикатор, при этом MP3-плеер воспроизводит соответствующий аудиофайл. В функции `loop` ультразвуковой датчик расстояния HC-SR04 измеряет расстo-

[31] Что соответствует температуре около 20 °C. Скорость звука в воздухе растет примерно на 0,59 м/с на каждые 10° увеличения температуры. – *Прим. перев.*

яние каждые две секунды. Когда измеренное расстояние меньше порогового значения, функция `play` активируется, запускается объявление MP3-плеера, и сигнал тревоги отключается. Существует интервал времени между воспроизведением аудиофайлов, чтобы исключить последовательное воспроизведение треков без перерыва, когда сигнал на выводе *BUZY* MP3-плеера изменяется в состояние высокого уровня (`HIGH`).

Подключения для сигнализации на MP3-плеере приведены в табл. 5-4. Нет соединения *TX* или *BUZY* MP3-плеера с микроконтроллером ESP8266 или ESP32, поскольку MP3-плеер не передает сигнал. В листинге 5-5 приведена программа для микроконтроллера ESP8266, а библиотека *NewPing8266* загружается с github.com/jshaw/NewPingESP8266. Для микроконтроллера ESP32 библиотека *NewPing* Тима Экеля (Tim Eckel) доступна в Arduino IDE. Основное внимание в макетах на рис. 5-6 и 5-7 уделяется минимизации перекрывающихся соединений, поскольку на практике требуется доступ MP3-плеера и держателя карты micro-SD к платам ESP8266 или ESP32 для обеспечения питания.

Таблица 5-4. Сигнализация на MP3-плеере с платами ESP8266 или ESP32

Компонент	ESP8266	ESP32
MP3-плеер VCC	3V3	3V3
MP3-плеер RX	D7	TX2 (GPIO 17)
MP3-плеер SPK1	Аудиовыход	Аудиовыход
MP3-плеер GND	GND	GND
MP3-плеер SPK2	Аудиовыход	Аудиовыход
HC-SR04 VCC	5V	VIN
HC-SR04 Trig	D1	GPIO 22
HC-SR04 Echo	D2	GPIO 21
HC-SR04 GND	GND	GND
LED длинный вывод	D3	GPIO 19
LED короткий вывод	Резистор 220 Ом к GND	
Кнопка правый	D4	GPIO 18
Кнопка левый	GND	GND

В листинге 5-5 приведена программа для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 инструкции ESP8266

```
#include<SoftwareSerial.h>            // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);       // define SoftSer TX pin
#include <NewPingESP8266.h>           // include NewPingESP8266 lib
NewPingESP8266 sonar (trigPin, echoPin, maxdist);
SoftSer.begin(9600);                  // software Serial baud rate
for (int i=0; i<10; i++) SoftSer.write(serialCom[i]);
```

заменяются на

```
#include <NewPing.h>                          // include NewPing library
NewPing sonar (trigPin, echoPin, maxdist);
Serial2.begin(9600, SERIAL_8N1, 16, 17);      // Serial2 TX2 on GPIO 17
for (int i=0; i<10; i++) Serial2.write(serialCom[i]);
```

и определения выводов *trigPin*, *echoPin*, *LEDpin* и *alarmPin* изменены на GPIO 22, GPIO 21, GPIO 19 и GPIO 18 соответственно.

Листинг 5-5. Сигнализация на MP3-плеере

```
#include<SoftwareSerial.h>                    // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);               // define SoftSer TX pin
#include <NewPingESP8266.h>                   // include NewPingESP8266 lib
int trigPin = D1;                             // HC-SR04 trigger pin
int echoPin = D2;                             // HC-SR04 echo pin
int maxdist = 200;                            // set max scan distance (cm)
int echoTime;
float distance;                               // scanned distance (cm)
                                              // associate sonar with NewPing
NewPingESP8266 sonar (trigPin, echoPin, maxdist);
int LEDpin = D3;                              // define LED pin
int alarmPin = D4;                            // define alarm switch pin
unsigned int checksum;
byte highChk, lowChk;                         // control command template
byte serialCom[10] = {0x7E,0xFF,0x06,0x00,0x00,
0x00,0x00,0x00,0x00,0xEF};
byte buffer[10];
volatile int alarmSet = 0;                    // set alarm state
String fileName[] = {                         // file names in numerical order
"0001 alarm off",
"0002 alarm on",
"0003 someone entered the room",
"0004 close the door please",
"0005 press switch to reset the alarm"
};
void setup()
{
   Serial.begin(115200);                      // Serial Monitor baud rate
   SoftSer.begin(9600);                       // software Serial baud rate
   pinMode(trigPin, OUTPUT);                  // define trigger pin as output
   pinMode(LEDpin, OUTPUT);                   // define LEDpin as output
   pinMode(alarmPin, INPUT_PUL LUP);
                                              // alarm pin uses pull-up resistor
   attachInterrupt(digitalPinToInterrupt(alarmPin), alarmISR,
   FALLING);
}
void loop()
{
   echoTime = sonar.ping();                   // echo time (µs)
   distance = (echoTime/2.0)*0.0343;          // distance to target
   Serial.println(distance);                  // play audio files if
                                              // distance < 100 and alarm set
   if(distance < 100 && alarmSet == 1 ) play();
   delay(2000);                               // delay between readings
}
void play()
```

```
{
  cmd(0x06, 10);                    // volume to 10 (range 0 to 30)
  cmd(0x12, 3);                     // play audio file named 0003
  delay(2000);                      // interval between audio files
  cmd(0x12, 4);                     // play audio file named 0004
  delay(2000);
  cmd(0x12, 5);                     // play audio file named 0005
  delay(2500);
  alarmISR();                       // turn alarm off
}
void cmd(byte CMD, byte param2)
{                                   // build checksum
  checksum = -(0xFF + 0x06 + CMD + 0x00 + 0x00 + param2);
  highChk = highByte(checksum);     // split checksum into
  lowChk = lowByte(checksum);       // high byte and low bytes
  serialCom[3] = CMD;
  serialCom[6] = param2;            // components of command
  serialCom[7] = highChk;
  serialCom[8] = lowChk;            // transmit command to MP3
  for (int i=0; i<10; i++) SoftSer.write(serialCom[i]);
}
IRAM_ATTR void alarmISR()
{
  alarmSet = 1 - alarmSet;          // turn off (0) or on (1) alarm
  digitalWrite(LEDpin, alarmSet);   // turn off or on LED
  cmd(0x12, alarmSet+1);            // play audio file 0001 or 0002
}
```

Сигнализация с обнаружением перемещения

Сигнализация обнаружения движения состоит из пассивного теплового датчика (PIR-датчика), обнаруживающего движение, который запускает MP3-плеер для воспроизведения предупреждения и включения светодиода на десять секунд (см. рис. 5-8 и 5-9). Скетч в листинге 5-6 состоит всего из 20 строк кода, при этом ультразвуковой датчик расстояния заменен PIR-датчиком, но с использованием соединений, приведенных в табл. 5-5. MP3-плеер воспроизводит только третью дорожку, для чего требуется команда (0x12, 0x03). Контрольная сумма команды представляет собой отрицательное значение шестнадцатеричного представления компонентов сигнала (0xFF + 0x06 + 0x12 + 0x00 + 0x00 + 0x03), который равен –(255 + 6 + 18 + 3), или –282 в десятичной системе счисления. Шестнадцатеричное представление –282 равно 2^{16} – 282 = 65 256, или 0xFEE6, где старший и младший байты равны 0xFE и 0xE6 соответственно.

Таблица 5-5. Сигнализация на MP3-плеере с платами ESP8266 и ESP32 – упрощенная версия

Компонент	ESP8266	ESP32
PIR-датчик VCC	5V	VIN
PIR-датчик OUT	D2	GPIO 21
PIR-датчик GND	GND	GND

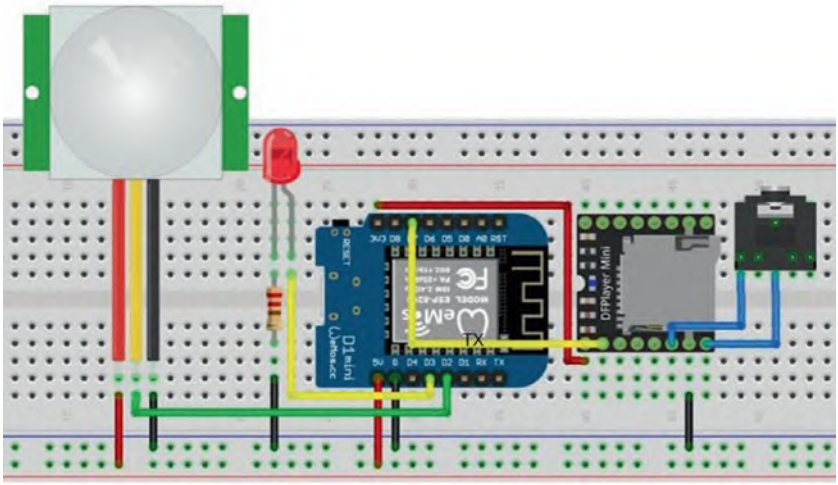


Рисунок 5-8. Сигнализация на MP3-плеере с платой LOLIN (WeMos) D1 mini

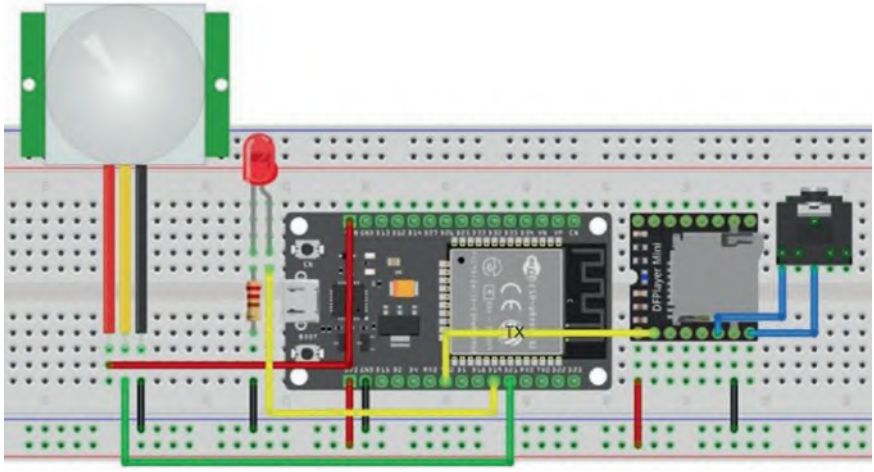


Рисунок 5-9. Сигнализация на MP3-плеере с платой ESP32 DEVKIT DOIT

В листинге 5-6 приведена программа для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 инструкции ESP8266

```
#include<SoftwareSerial.h>                    // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);               // define SoftSer TX pin
SoftSer.begin(9600);                          // software Serial baud rate
for (int i=0; i<10; i++) SoftSer.write(serialCom[i]);
```

заменить на инструкции

```
Serial2.begin(9600, SERIAL_8N1, 16, 17);      // Serial2 TX2 on GPIO 17
for (int i=0; i<10; i++) Serial2.write(serialCom[i]);
```

и определения выводов *PIRpin* и *LEDpin* изменены на GPIO 21 и GPIO 19 соответственно.

Листинг 5-6. Сигнализация на MP3-плеере – упрощенная версия

```
#include<SoftwareSerial.h>            // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);       // define SoftSer TX pin
int PIRpin = D2;                      // PIR sensor and LED pins
int LEDpin = D3;
byte serialCom[10] = {0x7E,0xFF,0x06,0x12,0x00,
0x00,0x03,0xFE,0xE6,0xEF};            // one control command
void setup()
{
   SoftSer.begin(9600);               // software Serial baud rate
   pinMode(LEDpin, OUTPUT);           // LED pin as OUTPUT
}
void loop()
{
   if(digitalRead(PIRpin) == HIGH)    // PIR sensor triggered
   {
      digitalWrite(LEDpin, HIGH);     // turn on LED and play sound
      for(int i=0; i<10; i++) SoftSer.write(serialCom[i]);
      delay(10000);
      digitalWrite(LEDpin, LOW);      // turn off LED after 10s
   }
}
```

Говорящие часы

Говорящие часы построены с использованием часов реального времени (RTC) DS3231 и MP3-плеера. Модуль DS3231 имеет встроенный датчик температуры, питается от 3,3 В или 5 В и оснащен литиевой батареей CR2032 для питания микросхемы RTC, когда часы не подключены к плате ESP8266 или ESP32. Модуль DS3231 использует интерфейс I2C, связь в котором осуществляется по двум двунаправленным линиям: линии данных (SDA) и линии тактового сигнала (SCL). Объявление времени и температуры говорящими часами активируется нажатием кнопки. Один из сценариев предназначен для людей с нарушениями зрения, которые могут найти большой кнопочный переключатель, чтобы иметь возможность слышать время и температуру, но не могут легко прочитать время на часах. Рисунки 5-10 и 5-11 иллюстрируют говорящие часы с помощью плат ESP8266 и ESP32.

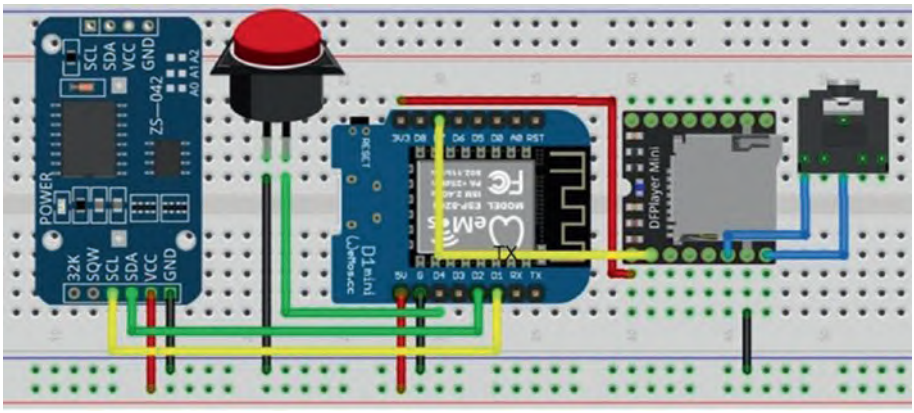


Рисунок 5-10. Говорящие часы с платой LOLIN (WeMos) D1 mini

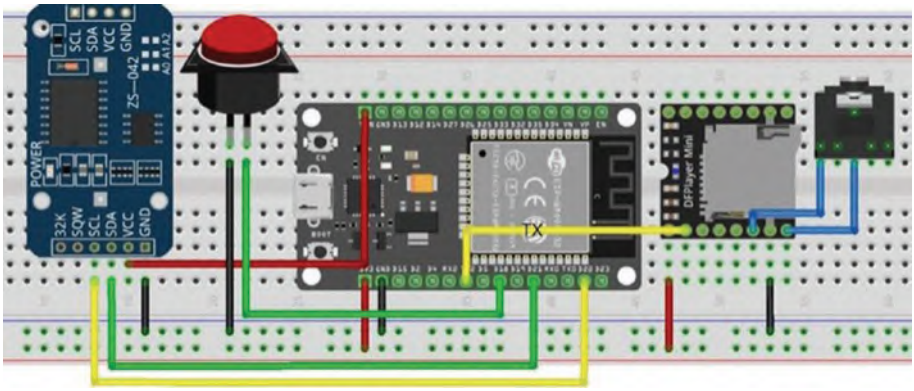


Рисунок 5-11. Говорящие часы с платой ESP32 DEVKIT DOIT

Соединения модулей DS3231 и MP3-плеера с платами ESP8266 и ESP32 приведены в табл. 5-6 и 5-3 соответственно. Нет соединений TX и *BUZY* MP3-плеера с микроконтроллером ESP8266 или ESP32, так как MP3-плеер не передает сигнал.

Таблица 5-6. Модуль часов реального времени с платами ESP8266 и ESP32

Компонент	ESP8266	ESP32
DS3231 GND	GND	GND
DS3231 VCC	5V	VIN
DS3231 SDA	D2	GPIO 21
DS3231 SCL	D1	GPIO 22
Кнопка правый	D4	GPIO 18
Кнопка левый	GND	GND

Рекомендуется использовать имеющуюся в Arduino IDE библиотеку *MD_DS3231* Марко Колли (Marco Colli) из-за простоты доступа к компонентам времени. Когда скетч сначала компилируется и загружается, в него включаются текущие дата и время[32]; а затем скетч необходимо немедленно компилировать и загрузить снова, но с закомментированными инструкциями по настройке даты и времени, как показано в листинге 5-7. При установке времени и даты используется 24-часовой формат времени без ведущих нулей. Компиляция и загрузка занимают около 30 с, поэтому перенесите время вперед на 30 с[33].

[32] То есть системное время компьютера, на котором проводится компиляция. – *Прим. перев.*

[33] Это время зависит от многих причин, учесть которые невозможно, а перестановка времени на компьютере – довольно хлопотная операция. Обратите внимание, что повторная компиляция без внесения изменений в текст скетча в среде Arduino IDE занимает намного меньше времени, чем компиляция в первый раз. Поэтому, чтобы не возиться с перестановкой часов, можно просто заранее выполнить один раз компиляцию с раскомментированными инструкциями по настройке (пункт «Проверить/Компилировать» Arduino IDE), а потом уже провести загрузку. В этом случае установка часов не будет отличаться от системного времени более, чем на несколько секунд. – *Прим. перев.*

В общей сложности для чисел времени и температуры требуется 28 звуковых файлов, на которые ссылаются файлы 0001 «*один*», 0002 «*два*» ... 0020 «*двадцать*», 0030 «*тридцать*», 0040 «*сорок*», 0050 «*пятьдесят*», 0060 «*ноль*», 0070 «*время*», 0080 «*градусов Цельсия*», 0090 «*и*» и 0100 «*часов*». Звуковые файлы 21–24 соответствуют числам 30, 40, 50 и 0; файлы 25–28 соответствуют времени, градусам Цельсия и часам. Веб-сайт www.fromtexttospeech.com создает голосовые mp3-файлы, соответствующие тексту, введенному в текстовое поле онлайн, с различными голосами на нескольких доступных языках.

Модуль DS3231 предоставляет информацию о времени, а затем MP3-плеер воспроизводит трек 25 со словами «*настоящее время*» ("the time is"), и функция *speak20* вызывается для воспроизведения треков, соответствующих часу (см. листинг 5-7). Если минут меньше десяти, то воспроизводится трек «*ноль*» ("zero") и от «*один*» до «*девять*»; а если минут меньше 21, то воспроизводится соответствующий трек «*десять*»...«*двадцать*». В противном случае воспроизводится трек «*тридцать*», «*сорок*» или «*пятьдесят*», соответствующий номерам треков 21, 22 или 23, вычисляемых как 18 плюс старший десятичный разряд минут (целая часть от результата деления числа минут на 10), за которыми следует трек для младшего разряда минут (остаток от деления минут на 10). Если минуты равны нулю, то воспроизводится трек 28 «*часов*» ("o'clock"). Функция *getTemp* для DS3231 считывает температуру, затем воспроизводится трек «*и*» ("and"), за которым следует функция *speak20*, вызываемая для воспроизведения треков цифр температуры, и, наконец, воспроизводится трек для градусов Цельсия ("degrees Celsius"). Треки короткие, поэтому нет необходимости в прерывании, чтобы определить, когда дорожка закончила воспроизведение, и состояние вывода *BUZY* не считывается.

В листинге 5-7 приведена программа для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 инструкции ESP8266

```
#include<SoftwareSerial.h>                    // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);               // define SoftSer TX pin
SoftSer.begin(9600);                          // software Serial baud rate
for (int i=0; i<10; i++) SoftSer.write(serialCom[i]);
```

заменяются инструкциями

```
Serial2.begin(9600, SERIAL_8N1, 16, 17);      // TX2 on GPIO 17
for (int i=0; i<10; i++) Serial2.write(serialCom[i]);
```

и определение `switchPin` должно быть изменено на GPIO 18.

Листинг 5-7. Говорящие часы

```
#include<SoftwareSerial.h>                    // include SoftwareSerial library
SoftwareSerial SoftSer(D6, D7);               // define SoftSer TX pin
#include <MD_DS3231.h>                        // include MD_DS3231 library
unsigned int checksum;
byte highChk, lowChk;
byte serialCom[10] ={0x7E,0xFF,0x06,0x00,0x00,
0x00,0x00,0x00,0x00,0xEF};
byte buffer[10];
int switchPin = D4;                           // define switch pin
int val, deg;
```

```
void setup()
{
   SoftSer.begin(9600);               // software Serial baud rate
   pinMode(switchPin, INPUT_PULLUP);
                                      // switch pin uses pull-up resistor
   cmd(0x06, 10);                     // set volume to 10 (range 0 - 30)
   RTC.control(DS3231_12H, DS3231_OFF);       // 24 hour clock
          // RTC date of Wednesday 7 September 2020 at 20:37:50
   /*                         // instructions to set time commented out
   RTC.yyyy = 2020;           // year
   RTC.mm = 9;                // month
   RTC.dd = 7;                // day
   RTC.h = 20;                // hour in 24 hour format
   RTC.m = 37;                // minutes
   RTC.s = 50;                // seconds, allow 30s to compile
   RTC.dow = 4;               // day of week, Sunday = 1
   RTC.writeTime();
   */
}
void loop()
{
   if(digitalRead(switchPin) == LOW)  // switch is pressed
   {
      speak(25);                      // MP3 play "the time is"
      RTC.readTime();                 // components of date and time
      speak20(RTC.h);                 // MP3 play the hour
      if(RTC.m == 0) speak(28);       // MP3 play "o'clock"
      else if(RTC.m <10)
      {
         speak(24);                   // MP3 play "zero"
         speak(RTC.m);                // MP3 play minute < 10
      }
      else if(RTC.m <21) speak(RTC.m); // MP3 play minute <21
      else
      {
         speak(RTC.m/10 + 18);        // MP3 play "30 40 or 50 mins"
         speak(RTC.m % 10);    // MP3 play minute < 10
      }                               // temperature measurement
      deg = round(RTC.readTempR egister());
      speak(27);                      // MP3 play "and"
      speak20(deg);                   // MP3 play the temperature
      speak(26);                      // MP3 play "degrees Celsius"
   }
}
void speak(int file)                  // function to play MP3 file
{
   if(file == 27) delay(200);         // delay before playing "and"
   cmd(0x03, file);
   delay(300);                        // time for short track to play
   if(file == 25 || file == 27 ) delay(300);
}                                     // delay for "the time is" or "and"
void speak20(int val)                 // function to play combination
{                                     // of "20" and units
   if(val < 21) speak(val);           // MP3 play number < 21
   else
   {
```

```
      speak(20);                        // MP3 play "20"
      speak(val % 20);                  // MP3 play track numbered
    }                             // remainder after dividing by 20
}
void cmd(byte CMD, byte param2)
{
   delay(500);                          // stop repeated button push
   checksum = -(0xFF + 0x06 + CMD + 0 x00 + 0x00 + param2);
                                        // build checksum
   highChk = highByte(checksum);        // split checksum into
   lowChk = lowByte(checksum);          // high byte and low bytes
   serialCom[3] = CMD;
   serialCom[6] = param2;               // components of command
   serialCom[7] = highChk;
   serialCom[8] = lowChk;               // transmit command to MP3
   for (int i = 0; i<10; i++) SoftSer.write(serialCom[i]);
   delay(20);                           // time to load command
}
```

Диктофон

Модуль записи и воспроизведения ISD1820 сохраняет записанные звуки длительностью до 10 с во внутренней флеш-памяти, хранящей информацию при выключенном питании. Запись активируется после появления высокого уровня (HIGH) сигнала на выводе *REC* или после нажатия кнопки *REC* на модуле. Светодиод модуля включается в течение 10 с записи. Опция воспроизведения *P-E* воспроизводит весь записанный звук после подачи высокого уровня (HIGH) сигнала на вывод *P-E* или при нажатии кнопки *PLAYE*. Опция воспроизведения *P-L* также запускается высоким уровнем (HIGH) сигнала, с воспроизведением записанного звука, пока вывод *P-L* находится на высоком уровне или нажата кнопка воспроизведения *PLAYL*. Модуль ISD1820 питается от 3,3 В и использует динамик 8 Ом мощностью 0,5 Вт (см. рис. 5-12 и соединения в табл. 5-7).

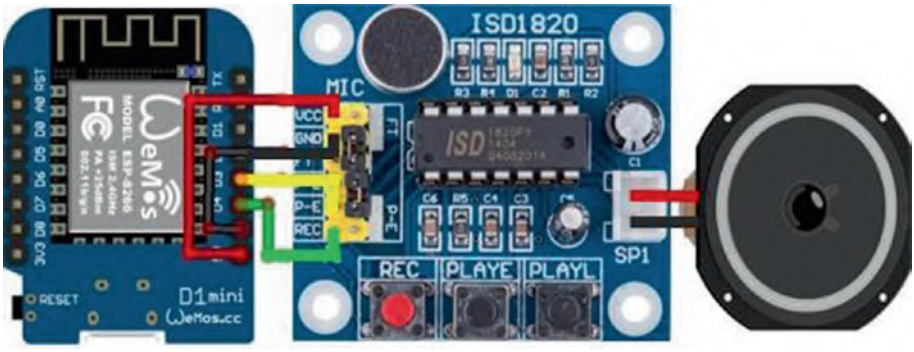


Рисунок 5-12. Модуль записи и воспроизведения ISD1820

В листинге 5-8 символ *r* или *p* вводится через монитор последовательного порта среды Arduino для начала записи в течение 10 с или для начала воспроизведения соответственно. Когда вызывается команда `pinMode(playPin, OUTPUT)`, вывод *PLAYE* автоматически устанавливается на низкий (LOW) уровень. Чтобы

быть уверенным, что вывод изначально установлен на низкий уровень, команда `digitalWrite(playPin, LOW)` предшествует инструкции `pinMode(playPin, OUTPUT)`[34]. Сигнал на инфракрасный датчик, подключенный к плате ESP8266 или ESP32, или обнаружение открытия двери PIR-датчиком может устанавливать высокий уровень (`HIGH`) сигнала для воспроизведения записанного сообщения, вместо ввода символа через монитор последовательного порта.

Таблица 5-7. Модуль записи и воспроизведения ISD1820

Компонент	Подключено к
ISD1820 VCC	ESP8266 3V3
ISD1820 GND	ESP8266 GND
ISD1820 P-E	ESP8266 D3
ISD1820 REC	ESP8266 D4

Листинг 5-8. Модуль записи и воспроизведения ISD1820

```
int playPin = D3;                    // define playback pin
int recPin = D4;                     // define record pin
char data;
void setup()
{
   Serial.begin(115200);             // Serial Monitor baud rate
   Serial.print("Enter r to record (10 seconds) or");
   Serial.println(" p to playback");
   digitalWrite(playPin, LOW);       // avoid playPin going HIGH
   pinMode(playPin, OUTPUT);         // define playPin and recPin
   pinMode(recPin, OUTPUT);          // as OUTPUT
}
void loop()
{
   while(Serial.available() > 0)     // if data available in Serial buffer
   {
      data = Serial.read();          // read Serial buffer
      if(data == 'r')
      {
         Serial.println("recording while light is on");
         digitalWrite(recPin, HIGH);   // HIGH to activate recording
         delay(10000); // recording time of 10s
         digitalWrite(recPin, LOW);    // reset to LOW signal
      }
      else if(data == 'p')
      {
         Serial.println("playback");
         digitalWrite(playPin, HIGH);  // HIGH to activate playback
         delay(10); // short delay of 10ms
         digitalWrite(playPin, LOW);   // reset to LOW signal
      }
   }
}
```

[34] Выводы микроконтроллера установлены в низкий (нулевой) уровень по умолчанию, но для надежности рекомендуется выполнять такую установку явно, отдельной командой. – *Прим. перев.*

Итоги

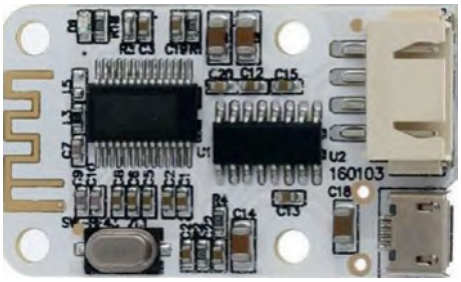
Мини-MP3-плеер DFPlayer работает и как автономный MP3-плеер, и при управлении микроконтроллером ESP8266 или ESP32 для воспроизведения следующего трека, увеличения или уменьшения громкости и регулировки с помощью музыкального эквалайзера. MP3-плеер управляется с помощью сигналов от инфракрасного пульта дистанционного управления, после сопоставления кнопок пульта с функциями MP3-плеера. Описана сигнализация с помощью ультразвукового датчика расстояния, запускающего воспроизведение объявлений, сделанных MP3-плеером. Сигнализация с обнаружением движения, требующая всего 20 строк кода, использует PIR-датчик для запуска объявления MP3-плеером и включения светодиодного индикатора. Нажатие кнопки запускает объявления MP3-плеера о текущем времени и температуре, информация о которых предоставляется модулем часов реального времени DS3231. Также описано управление модулем записи и воспроизведения ISD1820.

Перечень компонентов

- Платы микроконтроллера ESP8266: LOLIN (WeMos) D1 mini или NodeMCU
- Платы микроконтроллера ESP32: DEVKIT DOIT или NodeMCU
- MP3-плеер: DFPlayer Mini
- Тактовые кнопки: 2 шт.
- Динамик: менее, чем 3 Вт
- Аудиоразъем «джек» и мини-колонки
- Инфракрасный приемник: VS1838B
- Резистор: 220 Ом
- Светодиод
- Ультразвуковой датчик расстояния: HC-SR04
- Тепловой PIR-датчик: HR-SC501 или HR-SC505
- Модуль часов реального времени (RTC): DS3231
- Модуль записи/вопроизведения: ISD1820

Глава 6

Bluetooth-динамик



Bluetooth-динамик дополняет интернет-радио из главы 1, MP3-проигрыватель из главы 5 и светодиодную ленту WS2812 5050 RGB, реагирующую на звук из главы 4. Модуль Bluetooth-приемника стереозвука с аудиоусилителем PAM8403 класса D не требует микроконтроллера ESP8266 или ESP32, но своей увлекательностью и простотой проект заслуживает включения в книгу.

После того как модуль Bluetooth-приемника стереозвука связался с планшетом или мобильным телефоном Android с помощью bluetooth-связи, появляется объявление "*Bluetooth mode: the Bluetooth device is ready to pair*" («Режим bluetooth: устройство bluetooth готово к сопряжению»), после окончания подключения следует еще одно объявление "*The Bluetooth device is connected successfully*" («Устройство bluetooth подключено успешно»).

Модуль приемника стереозвука Bluetooth с аудиоусилителем PAM8403 класса D работает на динамики мощностью 3 Вт с питанием 5 В. Модуль Bluetooth-приемника и динамики питаются от литий-ионного аккумулятора 18650. Модуль повышающего преобразователя постоянного тока MT3608 увеличивает напряжение литий-ионной батареи 18650 3,7 В до требуемых 5 В. Повышающий преобразователь MT3608 работает при входном напряжении 2–24 В и обеспечивает выход до 28 В при 2 А. Выходное напряжение MT3608 регулируется вращением движка потенциометра, обозначенного стрелкой на рис. 6-1.

Рисунок 6-1. Повышающий преобразователь постоянного тока MT3608

Литий-ионные аккумуляторы не следует ни перезаряжать, ни разряжать до нуля; в противном случае литий-ионный аккумулятор 18650 может сильно нагреваться при попытке его зарядить. Модуль зарядки аккумулятора TP4056 представляет собой линейное зарядное устройство для одноэлементных литий-ионных аккумуляторов, работающее как в режиме постоянного тока, так и постоянного напряжения. Модуль TP4056 контролирует уровень напряжения литий-ионного аккумулятора во время зарядки и разрывает цепь питания, когда напряжение литий-ионного аккумулятора превышает 4,2 В. Свечение красного светодиода показывает, что модуль TP4056 находится в режиме зарядки, смена цвета на синий – что зарядка окончена. На рис. 6-2 показаны два модуля TP4056, с подключением к напряжению питания 4,5–8 В через предусмотренный на левой стороне разъем mini-USB или через контакты +/– (IN+/IN–). Вариант модуля TP4056, показанный на рис. 6-2 справа, используется только для зарядки литий-ионного аккумулятора и не может одновременно питать нагрузку.

Рисунок 6-2. Модули TP4056 для зарядки Li-ion-аккумуляторов

Модуль зарядки батареи TP4056 на рис. 6-2 слева включает в себя микросхему защиты батареи DW01A, управляющую двойным MOSFET-транзистором 8205A. Когда напряжение заряжаемой литий-ионной батареи достигает 4,2 В, модуль TP4056 переключается с постоянного тока зарядки 1 А на постоянное напряжение 4,2 В, и ток постепенно уменьшается до нуля. Если при отсутствии внешнего источника напряжение разряжающейся через нагрузку литий-ионной батареи падает до 2,4 В, MOSFET-транзистор отключается, отсоединяя батарею от нагрузки.

В модуле TP4056 выводы B+ и OUT+ соединены вместе, а выводы B– и OUT– соединены через MOSFET-транзистор. На рис. 6-3 показаны соединения модуля TP4056 с литий-ионной аккумуляторной батареей 18650 и модулем Bluetooth-приемника (см. табл. 6-1). Модуль зарядки аккумулятора TP4056, такой же, как на рис. 6-2 слева, имеет разъем mini-USB для подключения зарядного кабеля к разъему USB с напряжением 5 В.

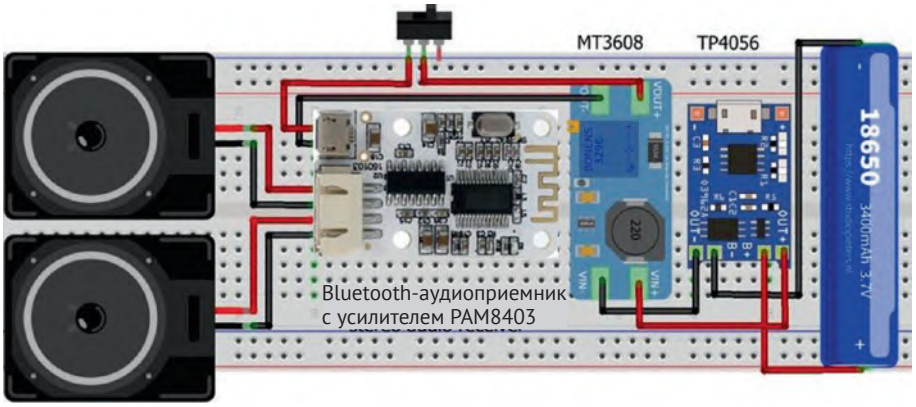


Рисунок 6-3. Модуль Bluetooth-приемника стереозвука с динамиками

Таблица 6-1. Соединения Bluetooth-приемника, модуля зарядки TP4056 и повышающего преобразователя MT3608

Компонент	Подключено к
TP4056 B+	Плюс аккумулятора 18650
TP4056 B–	Минус аккумулятора 18650
TP4056 OUT+	MT3608 VIN+
TP4056 OUT–	MT3608 VIN-
MT3608 VOUT+	Вход выключателя
Выход выключателя	Модули Bluetooth и PAM8403 вывод 5V
MT3608 VOUT–	Модули Bluetooth и PAM8403 вывод GND

На рис. 6-3 показана законченная схема соединений литий-ионной аккумуляторной батареи 18650, модуля зарядки аккумулятора TP4056, повышающего преобразователя MT3608, выключателя, модуля Bluetooth-стереоприемника с аудиоусилителем PAM8403 и, наконец, динамиков.

Альтернативой объединению модуля зарядки батареи TP4056 с модулем питания повышающего преобразователя постоянного тока MT3608 является использование USB-модуля зарядки литий-ионного аккумулятора, совмещенного с повышающим преобразователем, например такого, как модуль, входящий в состав блока питания на основе литий-ионного аккумулятора 18650 (см. рис. 6-4).

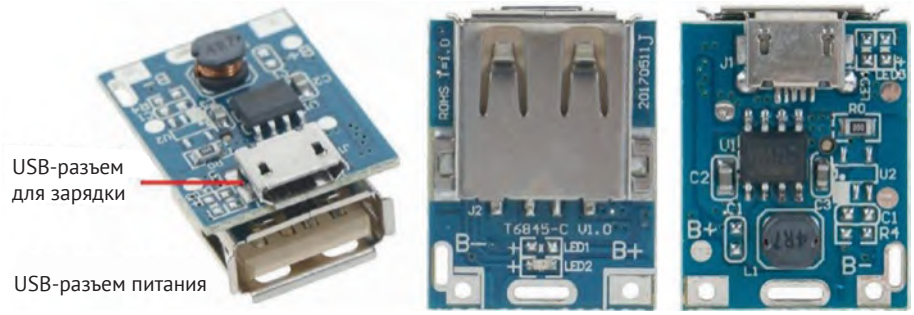


Рисунок 6-4. USB-модуль для зарядки литий-ионного аккумулятора со встроенным повышающим преобразователем T6845

Например, USB-модуль T6845 для зарядки литий-ионного аккумулятора с повышающим преобразователем повышает напряжение аккумулятора 3,7 В до требуемых 5 В при максимальном токе 1 А и включает защиту аккумулятора, при этом пороги отключения заряженной и разряженной батареи составляют 4,2 В и 2,9 В соответственно. Кабель для зарядки USB подключается к разъему mini-USB USB-модуля T6845. Выключатель установлен на положительном проводе линии, соединяющей USB-разъем питания модуля T6845 с разъемом mini-USB модуля Bluetooth-приемника (см. рис. 6-5 и подключения в табл. 6-2).

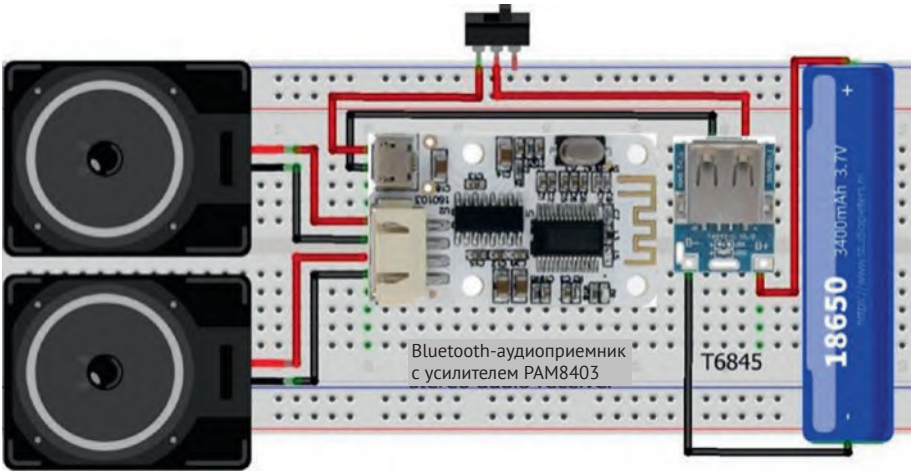


Рисунок 6-5. Bluetooth-приемник стереозвука и USB-модуль зарядки литий-ионного аккумулятора T6845 с повышающим преобразователем

Таблица 6-2. Подключения USB-модуля зарядки литий-ионного аккумулятора T6845 с повышающим преобразователем

Компонент	Подключено к
T6845 B+	Плюс аккумулятора 18650
T6845 B–	Минус аккумулятора 18650
T6845 USB-разъем питания +	Вход выключателя
Выход выключателя	Модули Bluetooth и PAM8403 вывод 5V
T6845 USB-разъем питания –	Модули Bluetooth и PAM8403 вывод GND

Литий-ионный аккумулятор 18650 и модуль T6845 зарядки литий-ионного аккумулятора USB с повышающим преобразователем, модуль Bluetooth-приемника со стереоаудиоусилителем PAM8403 и динамики легко помещаются в цилиндрический контейнер, например в 40-граммовую коробку из-под чипсов Pringles. Это не проект микроконтроллера ESP8266 или ESP32, и не нужно никакого программирования, однако потребуются некоторые навыки в пайке.

Итоги

Bluetooth-динамик был построен из модуля Bluetooth-приемника стереозвука с аудиоусилителем PAM8403 класса D, работающим от литий-ионного аккумулятора 18650, соединенного с повышающим преобразователем постоянного тока MT3608, с контролем зарядки и разрядки аккумулятора с помощью модуля зарядки TP4056. В качестве альтернативы комбинацию отдельного модуля зарядки аккумулятора TP4056 и повышающего преобразователя постоянного тока MT3608 можно заменить на модуль зарядки литий-ионного аккумулятора USB T6845 со встроенным повышающим преобразователем.

Перечень компонентов

- Модуль Bluetooth-приемника стереозвука с аудиоусилителем PAM8403
- Повышающий DC-DC преобразователь: MT3608
- Модуль зарядки и контроля Li-ion-аккумулятора: TP4056 с микросхемой защиты батареи DW01A
- Модуль зарядки Li-ion-аккумулятора со встроенным повышающим преобразователем: T6845
- Li-ion-аккумулятор: 18650
- Динамик: 3 Вт, 2 шт.

Глава 7

Беспроводная локальная сеть

Беспроводная локальная сеть (WLAN) устанавливается с помощью микроконтроллера ESP8266 или ESP32, при этом сетевые устройства запрашивают информацию у микроконтроллера (см. рис. 7-1). Такая беспроводная сеть не имеет доступа в интернет. Микроконтроллер является точкой доступа (*access point*, AP) для WLAN, включающей микроконтроллер и до четырех внешних устройств, и называется программной точкой доступа, или SoftAP[35]. Если микроконтроллер подключен к существующей сети Wi-Fi, то он находится в режиме станции (STA). В режиме точки доступа (AP) микроконтроллер определяет сеть Wi-Fi для подключения станций. Клиент WLAN – это браузер на ноутбуке, планшет на базе Android или мобильный телефон. Клиент подключается к беспроводной сети, выбирая имя беспроводной сети и пароль доступа с помощью браузера, открытого по URL-адресу на основе предопределенного IP-адреса беспроводной сети.

Рисунок 7-1. Беспроводная локальная сеть с микроконтроллером ESP8266 или ESP32

Чтобы проиллюстрировать создание беспроводной сети с микроконтроллером ESP8266 или ESP32, веб-страница WLAN управляет двумя светодиодами, которые подключены к плате ESP8266 или ESP32, и отображает состояния светодиодов и счетчик, который увеличивается при изменении состояния светодиода (см. рис. 7-2). Пример иллюстрирует использование клиента для удаленного управления устройствами, подключенными к плате ESP8266 или ESP32, действующей как сервер, а клиент получает информацию в виде HTML-кода, отображаемого на веб-странице WLAN.

[35] SoftAP (software enabled access point, букв. точка доступа с поддержкой программного обеспечения) – точка доступа (необязательно беспроводная) на основе снабженного необходимым программным обеспечением компьютерного устройства, которое не было специально создано для того, чтобы выполнять функции маршрутизатора. Также используется термин «виртуальный маршрутизатор» (virtual router). – *Прим. перев.*

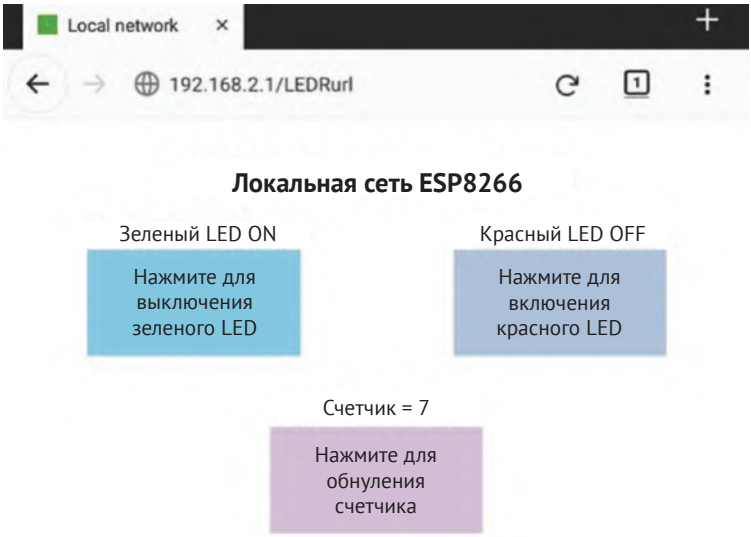


Рисунок 7-2. Веб-страница беспроводной локальной сети

Нажатие кнопки на веб-странице включает или выключает соответствующий индикатор или сбрасывает счетчик на ноль. Схемы микроконтроллеров ESP8266 и ESP32, функционирующих как сервер WLAN и подключенных к двум светодиодам, показаны на рис. 7-3, а соединения приведены в табл. 7-1.

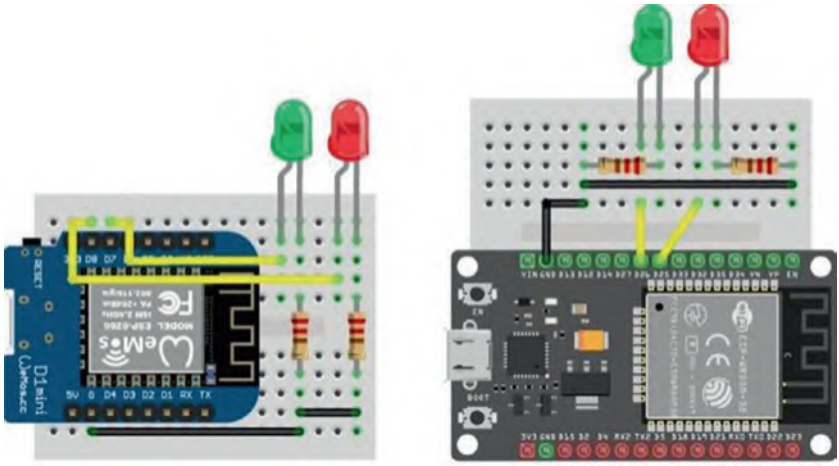


Рисунок 7-3. Платы LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT со светодиодами

Таблица 7-1. Платы ESP8266 и ESP32 и светодиоды

Компонент	Соединения ESP8266	Соединения ESP32
LED длинный вывод	ESP8266 D7 и D8	ESP32 GPIO 25 и GPIO 26
LED короткий вывод	Резистор 220 Ом → ESP8266 GND	Резистор 220 Ом → ESP32 GND

IP-адрес WLAN не генерируется заранее, а определяется в скетче вместе с IP-адресом шлюза (*gateway*) и маской IP-подсети. IP-адрес WLAN и адрес шлюза идентичны. Маска IP-подсети (255,255,255,0) для IP-адресов класса C, такого как 192.168.2.1, указывает, что первые три элемента IP-адреса, 192.168.2, идентифицируют WLAN, а последний элемент определяет идентификатор хоста. Пароль WLAN должен содержать не менее восьми буквенно-цифровых символов. Например, инструкции по установке для WLAN сетевого имени (SSID) *ESP8266*, пароля *12345678* и IP-адреса *192.168.2.1* следующие:

```
char ssidAP[] = "ESP8266";                  // WLAN SSID
char passwordAP[] = "12345678";             // and password
IPAddress local_ip(192,168,2,1);            // pre-defined IP address,
IPAddress gateway(192,168,2,1);             // gateway
IPAddress subnet(255,255,255,0);            // and subnet mask
```

В функции `setup` скетча беспроводная сеть инициализируется в режиме точки доступа (AP) с помощью команды `WiFi.mode(WIFI_AP)`. По умолчанию используется совмещенный режим точки доступа и станции с инструкцией `WiFi.mode(WIFI_AP_STA)`, а отдельный режим станции (STA) задается инструкцией `WiFi.mode(WIFI_STA)`. Сетевой IP-адрес определяется с помощью инструкций:

```
WiFi.mode(WIFI_AP);                         // WLAN in AP mode
WiFi.softAP(ssidAP, passwordAP);            // WLAN SSID, password
WiFi.softAPConfig(local_ip, gateway, subnet); // initialise WLAN
```

IP-адрес беспроводной сети или хоста отображается инструкциями:

```
IPAddress IP = WiFi.softAPIP();
Serial.println(IP);
```

HTTP-ЗАПРОС

Клиент делает HTTP-запрос на получение информации, отправляя URL-адрес на сервер. Сервер вызывает функцию, сопоставленную с URL-адресом, и отвечает клиенту HTML-кодом на информацию, предоставленную функцией. Например, URL-адрес сопоставляется с функцией, которая изменяет состояние светодиода, в результате состояние светодиода отображается на веб-странице WLAN. На рис. 7-4 нажатие кнопки на веб-странице WLAN отправляет HTTP-запрос, содержащий URL-адрес, связанный с кнопкой, на сервер, которым является микроконтроллер ESP8266 или ESP32. Сервер вызывает функцию, сопоставленную с URL-адресом, чтобы изменить состояние светодиода и обновить HTML-код по указанному URL. HTTP-отклик сервера заключается в отправке обновленного HTML-кода клиенту, который обновляет веб-страницу WLAN.

Рисунок 7-4. HTTP-запрос клиента и HTTP-отклик сервера

Скетч в листинге 7-1 демонстрирует, как на веб-странице WLAN отображаются состояния зеленого и красного светодиодов, а также значение счетчика, которое можно сбросить до нуля. Состояния светодиодов контролируются двумя функциями `LEDGfunct` и `LEDRfunct`, при этом функция `ZEROfunct` сбрасывает счетчик. Соответствующие три URL, */LEDGurl*, */LEDRurl* и */ZEROurl*, сопоставляются с функциями с помощью инструкции `server.on(URL, function)`.

HTTP-отклик сервера выполняется с помощью инструкции `server.send(status code, content type, content)`. Код состояния 200 или 404 указывает, соответственно, на успешный HTTP-запрос или на то, что запрошенный URL-адрес не найден. Тип содержимого – обычный текст, HTML-код или текст в формате JSON, описанный в главе 3 («Международная метеостанция») и главе 8 («Обновление веб-страницы»), которые обозначаются как `text/plain`, `text/html` и `text/json`. Когда тип содержимого `text/html`, HTML-код представлен в виде строки или функции. В листинге 7-1 HTML-код представлен в виде функции `webcode`, которая имеет три параметра: состояний зеленого светодиода, красного светодиода и значения счетчика. Функция `webcode` возвращает строку `page`, содержащую HTML-код для веб-страницы WLAN. Сервер отправляет HTML-код клиенту инструкцией `server.send(200, "text/html", webcode(LEDG, LEDR, counter))`.

В листинге 7-1 SSID и пароль WLAN определяются вместе с IP-адресом WLAN, URL-адреса сопоставляются с функциями, управляющими светодиодами и счетчиком, и определяется HTML-код для веб-страницы WLAN (см. листинг 7-3). Библиотека *ESP8266WebServer* ссылается на библиотеку *ESP8266WiFi*, поэтому библиотека *ESP8266WiFi* не включена в скетч явно. Аналогично для микроконтроллера ESP32 библиотека *Wi-Fi* ссылается на библиотеку *WebServer*. HTML-код содержится на вкладке *buildpage.h*, чтобы отделить HTML-код от основного скетча. Дополнительная вкладка создается в IDE Arduino, если щелкнуть треугольник под кнопкой монитора последовательного порта в правой части окна IDE и выбрать из выпадающего меню пункт «*Новая вкладка*» (“*New tab*”). Новую вкладку назовите *buildpage.h*. Обратите внимание, что функция `loop` включает только одну команду `server.handleClient()`.

Листинг 7-1. Функции WLAN и светодиодов

```
#include <ESP8266WebServer.h>          // include ESP8266WebServer lib
ESP8266WebServer server;               // associate server with library
char ssidAP[] = "ESP8266";             // WLAN SSID and password
char passwordAP[] = "12345678";
IPAddress local_ip(192,168,2,1);       // pre-defined IP address values
IPAddress gateway(192,168,2,1);
IPAddress subnet(255,255,255,0);
```

```
#include "buildpage.h"                // webpage HTML code
int LEDGpin = D7;                     // define LED pins
int LEDRpin = D8;
int LEDR = LOW;                       // default LED states
int LEDG = LOW;
int counter = 0;
void setup()
{
  WiFi.mode(WIFI_AP);                 // Wi-Fi AP mode
  delay(1000);                        // setup AP mode
  WiFi.softAP(ssidAP, passwordAP);    // initialise Wi-Fi with
  WiFi.softAPConfig(local_ip, gateway , subnet);
                                      // predefined IP address
  server.begin();                     // initialise server
  server.on("/", base);               // load default webpage
  server.on("/LEDGurl", LEDGfunct);   // map URLs to functions:
  server.on("/LEDRurl", LEDRfunct);   // LEDGfunct, LEDRfunct
  server.on("/zeroUrl", zeroFunct);   // and zeroFunct
  pinMode(LEDGpin, OUTPUT);           // define LED pins as output
  pinMode(LEDRpin, OUTPUT);
}
void base()                           // function to load default webpage
{                                     // and send HTML code to client
  server.send(200, "text.html", webcode(LEDG, LEDR, counter));
}
void LEDGfunct()                      // function to change green LED state,
{                                     // increment counter and
  LEDG = !LEDG;                       // send HTML code to client
  digitalWrite(LEDGpin, LEDG);
  counter++;
  server.send(200, "text/html", webcode(LEDG, LEDR, counter));
}
void LEDRfunct()                      // function to change red LED state,
{                                     // increment counter and
  LEDR = !LEDR;                       // send HTML code to client
  digitalWrite(LEDRpin, LEDR);
  counter++;
  server.send(200, "text/html", webcode(LEDG, LEDR, counter));
}
void zeroFunct()                      // function to zero counter
{                                     // and send HTML code to client
  counter = 0;
  server.send(200, "text/html", webcode(LEDG, LEDR, counter));
}
void loop()
{
  server.handleClient();              // manage HTTP requests
}
```

Когда клиент первоначально загружает веб-страницу, инструкция `server.on("/", base)` вызывает функцию `base`, которая загружает веб-страницу WLAN по умолчанию со значениями переменных `LEDG`, `LEDR` и `counter`, установленными на `LOW`, `LOW` и ноль соответственно. Функция `base` определяет настройки по умолчанию для веб-страницы WLAN.

Ответ сервера на HTTP-запрос клиента заключается в обновлении соответствующего состояния светодиода, значения счетчика и HTML-кода для веб-страницы. Например, если нажать кнопку для включения красного светодиода, то HTTP-запрос клиента содержит URL */LEDRurl*, который отображается в верхней части веб-страницы (см. рис. 7-2). Сервер вызывает соответствующую функцию `LEDRfunct`, и состояние красного светодиода изменяется с `LOW` (ноль) на `HIGH` (единица), а счетчик увеличивается. Затем функция `webcode` обновляет HTML-код, чтобы вставить строку

```
<td>Red LED is ON now<a href='/LEDRurl' class='btn on'>
Press to turn Red LED OFF</a></td>
```

Когда сервер отвечает клиенту, отправляя обновленный HTML-код, на веб-странице отображается текст "*Red LED is ON now"* («Красный светодиод сейчас включен»), а текст кнопки обновляется до "*Press to turn Red LED OFF"* («Нажмите, чтобы выключить красный светодиод»). В веб-браузере индикатор загрузки веб-страницы, расположенный рядом с заголовком веб-страницы, изменяется по мере ее загрузки, и отображается URL-адрес *192.168.2.1/LEDRurl*.

Нажатие кнопки *zero* отправляет URL */zeroUrl* на сервер, а значение счетчика сбрасывается на ноль с помощью связанной функции `zeroFunct`. Функция `webcode` обновляет HTML-код <p>"Counter is "+String(counter)+" now<a href='/zeroUrl'" со значением счетчика, равным нулю, а на веб-странице WLAN отображается текст "*Counter is 0 now"* («Счетчик равен 0»).

С микроконтроллером ESP8266 в качестве точки доступа WLAN инструкции библиотеки веб-сервера будут такими:

```
#include <ESP8266WebServer.h>
ESP8266WebServer server
```

а соответствующие инструкции для микроконтроллера ESP32 – такими:

```
#include <WebServer.h>
WebServer server(80);   // requires a port number
```

В качестве альтернативы в начале скетча можно разместить инструкции установки выводов светодиодов из листинга 7-2, приспособленные для обоих микроконтроллеров ESP8266 и ESP32.

Листинг 7-2. Определения выводов для плат ESP8266 и ESP32

```
#ifdef ESP8266
   #include <ESP8266WebServer.h>      // include ESP8266 library
   ESP8266WebServer server;
   int LEDGpin = D7;                  // define LED pins
   int LEDRpin = D8;
#elif ESP32
   #include <WebServer.h>             // include ESP32 library
   WebServer server (80);
   int LEDGpin = 26;                  // define LED pins
   int LEDRpin = 25;
#else                                 // Arduino IDE error message
   #error "ESP8266 or ESP32 microcontroller only"
#endif
```

HTML-код

Подробное описание языка гипертекстовой разметки HTML выходит за рамки книги. Учебник по адресу www.w3schools.com[36] рекомендуется для получения информации о HTML и каскадных таблицах стилей CSS, для создания и определения стиля веб-страниц.

Вкратце, HTML-страница состоит из раздела `<head>`, где определяются заголовок и стили веб-страницы, и раздела `<body>`, который содержит содержимое веб-страницы. Разделы заключены между открывающим и закрывающим тегами `<head>...</head>` и `<body>...</body>`. Стиль определяет типы и размеры шрифтов, заголовки и интервалы, заключенные между тегами `<style>...</style>`. Специфические элементы на веб-странице форматируются отдельно и заключаются между тегами `<span>...</span>`.

Например (см. листинг 7-3), в разделе `<head>` стиль `body` устанавливает верхнее поле на уровне 50 пикселей (при разрешении 96 пикселей на дюйм) и выравнивает по центру текст размером 20 пикселей шрифтом Arial. Стиль кнопки (`btn`) определяет ширину кнопки 220 пикселей, расстояние между кнопками в 30 пикселей и черный текст кнопки размером 30 пикселей. Параметр `display:block` разрешает обтекание кнопки текстом, `margin:auto` центрирует кнопку на веб-странице и `text-decoration:none` предотвращает подчеркивание HTML-ссылки. Стили `on`, `off`, и `zero` комбинируются со стилем кнопки для определения цвета фона кнопки. Подробности задания цветов доступны на www.w3schools.com/colors/colors_hex.asp.

На веб-странице расположена таблица, содержащая строки и ячейки, заключенные в теги `<tr>...</tr>` и `<td>...</td>` соответственно. Таблица используется для выравнивания объектов веб-страницы, таких как красная и зеленая светодиодные кнопки. Кнопка счетчика заключается в текстовый абзац, определяемый тегами `<p>...</p>`.

Листинг 7-3. HTML-код для веб-страницы WLAN

```
String webcode(int LEDG, int LEDR, int counter)
{
   String page = "<!DOCTYPE html><html><head>";
   page +="<title>Local network</title>";
   page +="<style>";
   page +="body {margin-top:50px; font-family:Arial;";
   page +="font-size:20px; text-align:center}";
   page +=".btn {display:block; width:220px;";
   page += "margin:auto; padding:30px}";
   page +=".btn {font-size:30px; color:black;";
   page += "text-decoration:none}";
   page +=".on {background-color:SkyBlue}";
   page +=".off {background-color:LightSteelBlue}";
   page +=".zero {background-color:Thistle}";
   page +="td {font-size:30px; margin-top:50px;";
   page += "margin-bottom:5px}";
   page +="p {font-size:30px; margin-top:50px;";
   page += "margin-bottom:5px}";
```

[36] На русском языке учебник HTML можно найти, например, на html5book.ru. – *Прим. перев.*

```
    page +="</style></head>";
    page +="<body>";
    page +="<h1>ESP8266 local area network</h1>";
    page +="<table style='width:100%'><tr>";
    if(LEDG>0)
    {
       page +="<td>Green LED is ON now";
       page +="<a href='/LEDGurl' class='btn on'>";
       page +="Press to turn Green LED OFF</a></td>";
    }
    else
    {
       page +="<td>Green LED is OFF now";
       page +="<a href='/LEDGurl' class='btn off'>";
       page +="Press to turn Green LED ON</a></td>";
    }
    if(LEDR>0)
    {
       page +="<td>Red LED is ON now";
       page +="<a href='/LEDRurl' class='btn on' >";
       page +="Press to turn Red LED OFF</a></td>";
    }
    else
    {
       page +="<td>Red LED is OFF now";
       page +="<a href='/LEDRurl' class='btn off'>";
       page +="Press to turn Red LED ON</a></td>";
    }
    page +="</tr></table>";
    page +="<p>Counter is "+String(counter);
    page +=" now<a href='/zeroUrl'";
    page +="class='btn zero'>Press to zero counter</a></p>";
    page +="</body></html>";
    return page;
}
```

HTML-код для веб-страницы может быть включен в основной скетч, но мы его включаем в качестве дополнительной вкладки, например *buildpage.h*, чтобы отделить основной скетч от HTML-кода и упростить интерпретацию программы. Дополнительная вкладка создается в среде IDE Arduino, если щелкнуть треугольник под кнопкой монитора последовательного порта в правой части окна IDE и выбрать «*Новая вкладка*» (“*New tab*”) из выпадающего меню. Назовите новую вкладку *buildpage.h*.

Функция `webcode` (см. листинг 7-3) содержится на этой вкладке, заданной инструкцией `#include "buildpage.h"` в первом разделе листинга 7-1. На вкладку *buildpage.h* ссылаются с помощью кавычек " ", а не с помощью угловых скобок < >, как это делается в случае библиотек. Функция `webcode` возвращает строку `page`, содержащую HTML-код веб-страницы. HTML-код строится строка за строкой в виде длинной строки и включает условные операторы для изменения HTML-кода в зависимости от значений переменных `LEDG`, `LEDR` и `counter`.

Вся веб-страница перезагружается, даже если изменилось только одно значение данных. Технология AJAX позволяет обновлять лишь отдельные значения, не перезагружая всю веб-страницу.

При запуске скетча в листингах 7-1 и 7-3 в браузере, таком как Google Chrome или Mozilla Firefox, на вашем ноутбуке, планшете Android или мобильном устройстве осуществляется подключение к серверу под названием *ESP8266*. URL-адрес для доступа к веб-странице равен 192.168.2.1, как это определено в листинге 7-1.

XML HTTP-запросы, JavaScript и AJAX

XML[37] HTTP-запрос обновляет определенную переменную на веб-странице, вместо того чтобы перезагружать всю веб-страницу целиком. Комбинация XML HTTP-запроса с командами JavaScript для управления HTTP-запросом – это AJAX (что расшифровывается как Asynchronous JavaScript and XML). Пример кода AJAX для XML HTTP-запроса приведен в листинге 7-4. Готовность ответа на HTTP-запрос указывается значением `readyState`, равным 4, то, что HTTP-запрос выполнен успешно, – с помощью значения `status`, равного 200 (см. пометку [1] в листинге 7-4). Затем объект XML HTTP-запроса отправляет запрос на сервер для обновления информации о переменной (пометка [2] в листинге 7-4) по заданному URL-адресу (пометка [3] в листинге 7-4). В веб-браузере индикатор загрузки веб-страницы, который был расположен рядом с заголовком веб-страницы, теперь отсутствует. Дополнительная информация о XML HTTP-запросах, JavaScript и AJAX доступна по адресу www.w3schools.com.

Листинг 7-4. AJAX-код для XML HTTP-запроса

```
<script>                                        // start of JavaScript
var xhr = new XMLHttpRequest();                 // XMLHttpRequest object
xhr.onreadystatechange = function()
{                                               // [1] if request successful
  if(this.readyState == 4 && this.status == 200)
  document.getElementById(variable).innerHTML =
  this.responseText;                            // [2] update variable
};
xhr.open('GET', URL, true);                     // [3] at URL
xhr.send();
</script>                                       // end of JavaScript
```

Поскольку в коде нет переменных, XML HTML-код представлен в виде одной длинной символьной строки, что позволяет избежать построчного создания HTML-кода, как это делалось в листинге 7-3. JavaScript и инструкции XML HTTP-запроса хранятся в виде строки символов в PROGMEM[38], находящейся между символами `R"(` и `)"`, или, иначе, `R"str(` и `)str"`, где `str` – это символ строки, не отображаемый в коде AJAX. Примером, представляющем символьную строку, является “===” в таких случаях, как `R"(===` или `===)"`. В JavaScript для обрамления строки применяются как одинарная кавычка ', так и двойные кавычки ", в листингах из этой книги предпочтительно используются одинарные кавычки.

[37] XML расшифровывается как eXtensible Markup Language, расширяемый язык разметки. – *Прим. перев.*

[38] См. листинг 7.6 далее. – *Прим. перев.*

Добавление XML HTTP-запросов требует внесения изменений в основной скетч и HTML-код. Для функций `LEDGfunct`, `LEDRfunct` и `zeroFunct` основного скетча в листинге 7-1 инструкция `server.send(200, "text/html", webcode(LEDG, LEDR, counter))` заменяется на `server.send(200, "text/plain", str)`. Только обновленная переменная, содержащаяся в строке `str`, отправляется сервером клиенту для обновления веб-страницы, а не HTML-код для всей веб-страницы. Например, изменения в функции `LEDRfunct` выделены жирным шрифтом, функция `LEDGfunct` обновлена аналогичным образом:

```
void LEDRfunct()                                  // function to change red LED state
{
  LEDR = !LEDR;                                   // change LED state
  digitalWrite(LEDRpin, LEDR);
  counter++;                                      // increment counter
  String str = "ON";
  if(LEDR == LOW) str = "OFF";                    // map string str to LED state
  server.send(200, "text/plain", str);            // send response to client
}
```

Функция `zeroFunct` также изменена, чтобы отправлять только обновленное значение счетчика:

```
void zeroFunct()                                 // function to zero counter
{
  counter = 0;                                   // reset counter
  String str = String(counter);                  // convert counter to a string
  server.send(200, "text/plain", str);           // send response to client
}
```

Функция `base` отправляет HTML-код веб-страницы по умолчанию клиенту с параметром `page`, без вызова функции `webcode`.

Код HTML и AJAX показан в листинге 7-6. Раздел `<style>` по содержанию идентичен разделу `<style>` в листинге 7-3. Раздел `<body>` теперь включает HTML-код для макета веб-страницы и AJAX-код для XML HTTP-запросов по обновлению веб-страницы. Скрипты JavaScript, заключенные в квадратные скобки `<script>` и `</script>`, располагаются перед HTML `</body>` кодом для повышения скорости отображения веб-страницы.

Различия между HTML-кодом в листинге 7-2 и кодом AJAX в листинге 7-6 можно проиллюстрировать кодом кнопки, управляющей зеленым светодиодом. В листинге 7-2 HTML-код ячейки таблицы, содержащей кнопку для зеленого светодиода в состоянии, когда зеленый светодиод горит, выглядит следующим образом:

```
<td>Green LED is ON now<a href='/LEDGurl' class='btn on'>
     Press to turn Green LED OFF</a></td>
```

HTML-код для состояния, когда зеленый светодиод выключен, выглядит следующим образом:

```
<td>Green LED is OFF now<a href='/LEDGurl' class='btn off'>
     Press to turn Green LED ON</a></td>
```

Для каждого состояния зеленого светодиода HTML-код определяет текст над кнопкой, URL-адрес, связанный с кнопкой `href='/LEDGurl'`, класс кнопки и текст на кнопке.

Приведеный в листинге 7-6 HTML-код обновления кнопки на веб-странице для управления зеленым светодиодом:

```
<td>Green LED is <span id='LEDG'>OFF</span> now</td>
<td><button class = 'btn off' id='Green LED'
     onclick = 'sendData(id)'>Press to turn Green LED ON
     </button></td>
```

HTML-код определяет текст над кнопкой с состоянием зеленого светодиода, удерживаемого переменной с идентификатором 'LEDG', которая имеет значение по умолчанию *OFF*, класс кнопки и текст на кнопке. Нажатие на кнопку с идентификатором, определенным `id='Green LED'` (см. листинг 7.5), вызывает функцию `sendData` с нужным идентификатором кнопки.

Код AJAX для функции `sendData`, относящийся к управлению зеленой светодиодной кнопкой, приведен в листинге 7-5. Функция обновляет состояние кнопки до *btn on* или *btn off* и текст кнопки *Press to turn Green LED ON* или *Press to turn Green LED OFF*. Функция `sendData` задает значения переменной `variable` для инструкции XML HTTP-запроса и URL-адрес:

```
document.getElementById(variable).innerHTML = this.responseText
xhr.open('GET', URL, true)
```

и, наконец, отправляет XML HTTP-отклик.

Листинг 7-5. AJAX-код для обновления переменной

```
function sendData(butn)
   {
   if(butn == 'Red LED' || butn == 'Green LED')
   {
      var state = document.getElementById(butn).className;
      state = (state == 'btn on' ? 'btn off' : 'btn on');
      text = (state == 'btn on' ? butn + ' OFF' : butn + ' ON');
      document.getElementById(butn).className = state;
      document.getElementById(butn).innerHTML = 'Press to turn '
      + text;
   }
   var URL, variab, text;
   else if(butn == 'Green LED')
   {
      URL = 'LEDGurl';
      variab = 'LEDG';
   }
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function(butn)
   {
      if (this.readyState == 4 && this.status == 200)
      document.getElementById(variab).innerHTML = this.
      responseText;
   };
   xhr.open('GET', URL, true);
   xhr.send();
}
```

Полный AJAX-код для управления XML HTTP-запросами и обновления веб-страницы при нажатии кнопки для управления красным или зеленым светодиодом или для обнуления счетчика приведен в листинге 7-6.

Листинг 7-6. AJAX-код для веб-страницы WLAN

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html><head>
<title>Local network</title>
<style>
body {margin-top:50px; font-family:Arial}
body {font-size:20px; text-align:center}
.btn {display:block; width:280px; margin:auto; padding:30px}
.btn {font-size:30px; color:black; text-decoration:none}
.on {background-color:SkyBlue}
.off {background-color:LightSteelBlue}
.zero {background-color:Thistle}
td {font-size:30px; margin-top:50px; margin-bottom:5px}
p {font-size:30px; margin-top:50px; margin-bottom:5px}
</style></head>
<body>
<h1>ESP8266 local area network</h1>
<table style='width:100%'><tr>
<td>Green LED is <span id='LEDG'>OFF</span> now</td>
<td>Red LED is <span id='LEDR'>OFF</span> now</td>
</tr></table>
<table style='width:100%'><tr>
<td><button class = 'btn off' id='Green LED'
      onclick = 'sendData(id)'>Press to turn Green LED ON
      </button></td>
<td><button class = 'btn off' id='Red LED'
      onclick = 'sendData(id)'>Press to turn Red LED ON
      </button></td>
</tr></table>
<p>Counter is <span id='counter'>0</span> now</p>
<button class = 'btn zero' id = 'zero'
      onclick = 'sendData(id)'>Press to zero counter</button>
<script>
function sendData(butn)
{
   var URL, variab, text;
   if(butn == 'Red LED')              // set URL and variab values
   { // for Red LED button
      URL = 'LEDRurl';
      variab = 'LEDR';
   }
   else if(butn == 'Green LED')       // or for Green LED button
   {
      URL = 'LEDGurl';
      variab = 'LEDG';
   }
   else if(butn == 'zero')            // or for the zero button
   {
   URL = 'zeroUrl';
   variab = 'counter';
   }
   if(butn == 'Red LED' || butn == 'Gree n L ED')
   {                                  // change button class and text
      var state = document.getElementById(butn).className;
      state = (state == 'btn on' ? 'btn off' : 'btn on');
      text = (state == 'btn on' ? butn + ' OFF' : butn + ' ON');
```

```
      document.getElementById(butn).className = state;
      document.getElementById(butn).innerHTML = 'Press to turn '
      + text;
   }
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function(butn)
   {
      if (this.readyState == 4 && this.status == 200)
      document.getElementById(variab).innerHTML = this.
      responseText;
   };
   xhr.open('GET', URL, true);
   xhr.send();
}
</script>
</body></html>
)";
```

При нажатии кнопок, управляющих красными и зелеными светодиодами, клиент отправляет XML HTTP-запрос для обновления значений состояний светодиодов, но счетчик автоматически не обновляется. Интервал между повторяющимися HTTP-запросами определяется с помощью инструкции `setInterval(reload, 1000)`, при этом функция `reload` создает XML HTTP-запрос каждые 1000 мс, как показано в листинге 7-7. В основном скетче URL-адрес */countUrl* сопоставляется функции `countFunct` с помощью команды `server.on("/countUrl", countFunct)`, включенной в функцию `setup`. Инструкции для функции `countFunct` следующие:

```
void countFunct()                          // function to update counter
{
   String str = String(counter);           // convert counter to a string
   server.send(200, "text/plain", str);     // send response to client
}
```

AJAX-код для периодического обновления счетчика в листинге 7-7 добавляется между инструкциями `<script>` и `</script>` в листинг 7-6.

Листинг 7-7. AJAX-код для периодического обновления счетчика

```
setInterval(reload, 1000);
function reload()
{
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      document.getElementById('counter').innerHTML = this.
      responseText;
   };
   xhr.open('GET', 'countUrl', true);
   xhr.send();
}
```

Два подхода, с использованием HTML-кода и кода AJAX, приводят к отображению одной и той же информации на веб-странице, но с HTML-кодом вся веб-страница перезагружается при каждом запросе клиента. В отличие от этого, при использовании кода AJAX на веб-странице обновляется только опреде-

ленная переменная. AJAX-код в два раза длиннее HTML-кода и может быть более сложным для понимания. Обновление только определенного элемента на веб-странице, а не перезагрузка всей веб-страницы, выгодно для веб-страниц, содержащих существенную информацию.

Итоги

Беспроводная локальная сеть создана с помощью микроконтроллера ESP8266 или ESP32 в качестве сервера WLAN. Браузер на планшете или мобильном телефоне Android, который является клиентом, получил доступ к беспроводной сети, с использованием IP-адреса и пароля, определенных скетчем. На веб-странице отображаются состояния двух светодиодов, подключенных к плате ESP8266 или ESP32, и значение счетчика. Состояния светодиодов управляются удаленно нажатием кнопок на веб-странице у клиента. HTML-код веб-страницы был построен построчно в виде длинной строки, и при обновлении информации всю веб-страницу приходилось перезагружать. AJAX-код, состоящий из XML HTTP-запросов и инструкций JavaScript, включенных в виде символьной строки, позволил обновлять только определенные переменные на веб-странице, а не перезагружать всю веб-страницу.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU
- Светодиод: 2 шт.
- Резистор: 220 Ом 2 шт.

Глава 8

Обновление веб-страницы

Устройства подключаются к беспроводной локальной сети (WLAN) с помощью Wi-Fi (см. главу 7 «Беспроводная локальная сеть»). Маршрутизатор, подключенный к интернет-провайдеру (Internet Service Provider, ISP) через телефонную линию с использованием технологии DSL (Digital Subscriber Line, *цифровая абонентская линия*), обеспечивает доступ в интернет (см. рис. 8-1). Интернет образован взаимосвязанными компьютерными сетями, использующими набор интернет-протоколов TCP/IP, состоящий из протокола управления передачей (TCP) и интернет-протокола (IP). Главная часть интернета, Всемирная паутина (WWW), определяет информационные ресурсы по URL-адресам. Клиент или веб-браузер, такой как Google Chrome или Mozilla Firefox, отправляет HTTP-запрос на серверное устройство, на котором размещен ресурс для получения информации по веб-адресу, определяемому URL. Сервер отвечает на HTTP-запрос клиента, и клиент отображает запрошенную информацию о веб-странице на планшете или мобильном телефоне Android. Клиент должен запрашивать информацию о веб-странице с сервера, так как сервер не может навязывать обновления клиенту. Исключением является WebSocket, как обсуждается в главе 9.

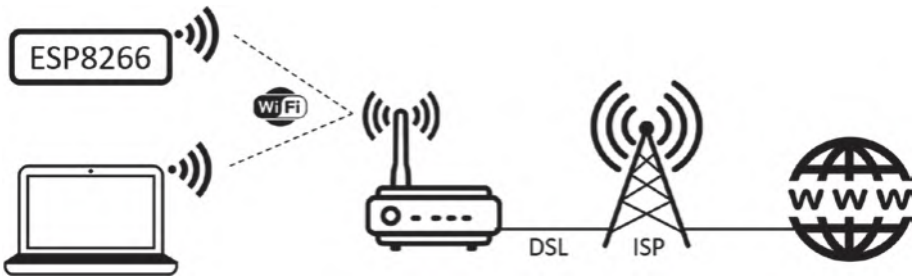


Рисунок 8-1. Клиент, интернет-провайдер и WWW

В этой главе сервером является микроконтроллер ESP8266 или ESP32. Инструкции по установке библиотеки веб-сервера для платы ESP8266 следующие:

```
#include <ESP8266WebServer.h>
ESP8266WebServer server
```

и для платы ESP32:

```
#include <WebServer.h>
WebServer server(80); // requires a port number
```

Чтобы продемонстрировать процесс запроса клиентом информации с сервера, на рисунке в листинге 8-1 приведены показания температуры BMP280[39], счетчик и состояние светодиода (см. рис. 8-2). Веб-страница обновляется автоматически, а время перезагрузки определяется HTML-кодом веб-страницы `<meta http-equiv='refresh' content='N'>` с обновлением каждые *N* с. Функция `BMP` обновляет показания температуры с датчика BMP280, увеличивает счетчик и изменяет состояние светодиода; затем сервер отправляет клиенту обновленный HTML-код веб-страницы. Синхронизацией функции `BMP` управляет библиотека *Ticker* с помощью команды `timer.attach(T, BMP)`, и функция `BMP` вызывается каждые *T* с. Если интервал обновления веб-страницы *N* существенно меньше интервала *T* вызова функции BMP, то клиент будет ждать (*T* – *N*) с, прежде чем сервер отправит обновленный HTML-код. В листинге 8-1 интервал вызова функции BMP равен *N* + 1 с, а не *N* с, поскольку сервер и клиент не синхронизированы.

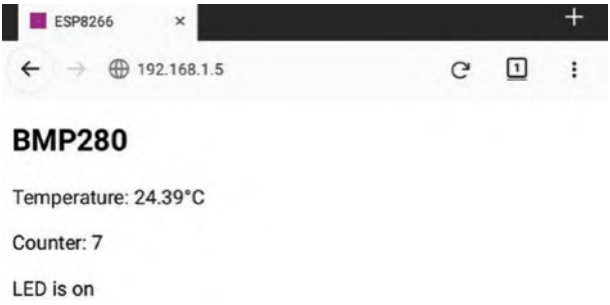


Рисунок 8-2. Веб-страница BMP280 и счетчика включения светодиода

На рис. 8-3 показаны датчик BMP280 и светодиод с платой ESP8266 и ESP32. Соединения приведены в табл. 8-1. Библиотека *Adafruit_BMP280* доступна в среде Arduino IDE.

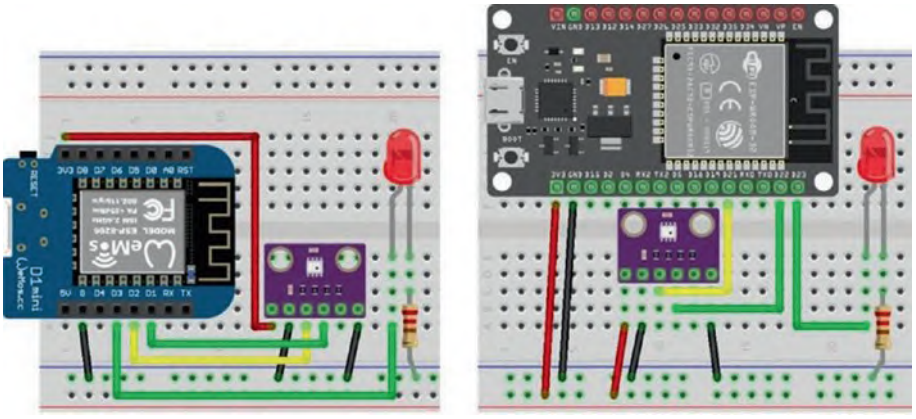


Рисунок 8-3. BMP280 и светодиод с платами LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT

[39] В данном проекте из показаний датчика атмосферного давления BMP280 используется только значение температуры. Однако далее в этой главе (см. раздел «MQTT-брокер и IFTTT») применяются и показания об атмосферном давлении тоже. – *Прим. перев.*

Таблица 8-1. BMP280 и светодиод (LED) с платами ESP8266 и ESP32

Компонент	ESP8266	ESP32
BMP280 VCC	3V3	3V3
BMP280 GND	GND	GND
BMP280 SDA	D2	GPIO 21
BMP280 SCK	D1	GPIO 22
BMP280 SDO	Не подключен	Не подключен
LED длинный вывод	D3	GPIO 23
LED короткий вывод	Резистор 220 Ом → GND	Резистор 220 Ом → GND

В функции `setup` из листинга 8-1 устанавливается соединение Wi-Fi, IP-адрес сервера отображается через монитор последовательного порта, веб-страница задается функцией `webcode` и определяется время выполнения функции `BMP`. На библиотеку *ESP8266WiFi* ссылается библиотека *ESP8266WebServer*, поэтому ее не нужно явно включать в скетч, аналогично с библиотеками *WebServer* и *WiFi* для микроконтроллера ESP32. Функция `webcode` возвращает строку *page*, содержащую HTML-код веб-страницы с обновленными значениями температуры, счетчика и состояния светодиода. В HTML-коде нет условных выражений, как в главе 7 («Беспроводная локальная сеть»), поэтому URL-адрес веб-страницы напрямую задается функцией `webcode`. Построчное построение HTML-кода включает переменные `temp`, `counter` и `LED`, которые не являются постоянными, поэтому HTML-код не может быть представлен в качестве единой символьной строки. Значения переменных в HTML-коде заключены в одинарные кавычки, как в инструкции `"<meta http-equiv='refresh' content='9'>"`, этот HTML-код заключен в двойные кавычки, как обычная строка.

Листинг 8-1. HTTP-запрос BMP280 и светодиода

```
#include <ESP8266WebServer.h>          // include ESP8266WebServer lib
ESP8266WebServer server;               // associate server with library
char ssid[] = "xxxx";                  // change xxxx to Wi-Fi SSID
char password[] = "xxxx";              // change xxxx to Wi-Fi password
#include <Adafruit_Sensor.h>           // include Unified Sensor
#include <Adafruit_BMP280.h>           // and BMP280 libraries
Adafruit_BMP280 bmp;                   // associate bmp with BMP280
int BMPaddress = 0x76;                 // I2C address of BMP280
#include <Ticker.h>                    // include Ticker library
Ticker timer;                          // associate timer with Ticker lib
int lag = 10;                          // set timer interval at 10s
int LEDpin = D3;                       // LED pin on D3
String LED = "off";                    // initial LED state
int count = 0;
String temp, counter;
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  WiFi.begin(ssid, password);          // initialise Wi-Fi
  while (WiFi.status() != WL_CON NECTED) delay(500);
                                       // wait for Wi-Fi connect
```

```
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());    // display server IP address
    server.begin();
    server.on("/", webcode);           // map URL to function
    bmp.begin(BMPaddress);             // initialise BMP280
    timer.attach(lag, BMP);            // BMP called every lag seconds
    pinMode(LEDpin, OUTPUT);
    digitalWrite(LEDpin, LOW);         // turn off LED
}
void BMP()                             // function to get readings
{
    temp = String(bmp.readTemperature());      // update BMP280 reading
    counter = String(count++);                 // increment counter
    digitalWrite(LEDpin, !digitalRead(LEDpi n));
                                               // turn on or off the LED
    if(LED == "on") LED = "off";               // update LED state
    else LED = "on";
    server.send (200, "text/html", webcode( ));
}                                              // send response to client
String webcode()                               // return HTML code
{
    String page;
    page = "<!DOCTYPE html><html><head>";      // refresh every 9s
    page += "<meta http-equiv='refresh' con tent= '9'>";
    page += "<title>ESP8266</title></head>";
    page += "<body>";
    page += "<h2>BMP280</h2>";                 // display temp
    page += "<p>Temperature: " + temp + " " + "& degC</p>";
    page += "<p>Counter: " + counter + "</p>"; // counter
    page += "<p>LED is " + LED + "<p>";        // LED state
    page += "</body></html>";
    return page;
}
void loop()
{
    server.handleClient();                     // manage HTTP requests
}
```

В результате HTTP-запроса от клиента сервер отправляет HTML-код для всей веб-страницы, а затем вся веб-страница перезагружается. Веб-страница в листинге 8-1 невелика и предназначена только для примера, но если бы веб-страница содержала больше информации (например, изображения), то время ее перезагрузки пришлось бы учитывать.

Листинг 8-1 предназначен для микроконтроллера ESP8266. Единственными изменениями в скетче для микроконтроллера ESP32 являются включение библиотеки *WebServer*, а не библиотеки *ESP8266WebServer*, и определение вывода светодиода. Альтернативой инструкциям скетча, специфичным для каждого конкретного микроконтроллера, является использование директивы компилятора, эквивалентной конструкции *if..then..else*, для условной компиляции скетча. Инструкции как для микроконтроллеров ESP8266, так и для ESP32 включаются в скетч. Если микроконтроллер не является ESP8266 или ESP32, то в Arduino IDE отображается сообщение об ошибке. Более подробная информация приведена в главе 21 («Микроконтроллеры»).

Листинг 8-2 содержит пример инструкций, которые следует включить в начало листинга 8-1.

Листинг 8-2. Определения выводов для плат ESP8266 и ESP32

```
#ifdef ESP32
   #include <WebServer.h>              // include ESP32 library
   WebServer server (80);              // and define LED pin
   int LEDpin = 23;
#elif ESP8266
    #include <ESP8266WebServer.h>      // include ESP8266 library
    ESP8266WebServer server;           // and define LED pin
    int LEDpin = D3;
#else                                  // Arduino IDE error message
   #error "ESP8266 or ESP32 microcontroller only"
#endif
```

XML HTTP-запросы, JavaScript и AJAX

В главе 7 («Беспроводная локальная сеть») описывается обновление определенной переменной на веб-странице с помощью XML HTTP-запроса вместо перезагрузки всей веб-страницы. Изменения в листинге 8-1 для преобразования в AJAX-код вместо HTML-кода не потребуют библиотеки *Ticker*, поэтому необходимо удалить следующие инструкции:

```
#include <Ticker.h>
Ticker timer
int lag = 10
timer.attach(lag, BMP)
```

поскольку время обновления веб-страницы управляется кодом AJAX. Функция `BMP` в листинге 8-1 больше не требуется и разделена на три функции: `tempFunct`, `countFunct` и `LEDfunct` для обновления показаний температуры с датчика BMP280, увеличения счетчика и изменения состояния светодиода. Каждая функция отправляет клиенту обновленную информацию для одной переменной. Функция `base` заменяет функцию `webcode` в листинге 8-1 и при первоначальной загрузке отправляет клиенту HTML-код веб-страницы по умолчанию. URL-адреса сопоставляются с четырьмя функциями инструкциями

```
server.on("/", base);
server.on("/tempUrl", tempFunct);
server.on("/countUrl", countFunct);
server.on("/LEDurl", LEDfunct);
```

В скетче из листинга 8-3 перечислены функции для получения исходных данных и указания серверу отправлять информацию клиенту. Обратите внимание, что параметры “`текст/html`” и `page` включены в функцию `base` для сервера, возвращающего HTML-код клиенту, в то время как функции `tempFunct`, `countFunct` и `LEDfunct` включают в себя параметр “`text/plain`” и имена переменных.

Листинг 8-3. XML HTTP-запросы для температуры BMP280, счетчика и состояния светодиода

```
void base()                   // function to load default webpage
{                             // and send HTML code to client
  server.send (200, "text/html", page);
}
void tempFunct()              // function to get temperature reading
{                             // and send value to client
  temp = String(bmp.readTemperature());
  server.send (200, "text/plai n", temp);
}                             // send plain text not HTML code
void countFunct()             // function to increment counter
{                             // and send value to client
  counter = String(count++);
  server.send (200, "text/plain", counter);
}
void LEDfunct()               // function to update LED
{                             // and send LED state to client
   digitalWrite(LEDpin, !digitalRead(LEDpin));
   if(LED == "on") LED = "off";
   else LED = "on";
   server.send (200, "text/plain", LED);
}
```

Листинг 8-4 содержит код AJAX для веб-страницы и XML HTTP-запросы, определенные как символьная строка. Код AJAX размещается на вкладке *buildpage.h* с помощью инструкции `#include "buildpage.h"`, чтобы отделить его от основного скетча. Дополнительная вкладка создается в IDE Arduino, если щелкнуть треугольник под кнопкой монитора последовательного порта в правой части окна IDE и выбрать пункт «*Новая вкладка*» (“*New tab*”) из выпадающего меню. Назовите новую вкладку *buildpage.h*. Обратите внимание, что функция `loop` включает только команду `server.handleClient()`.

Раздел `<body>` содержит HTML-код веб-страницы; а переменные `tempId`, `countId` и `LEDid` отвечают HTTP-запросам. Инструкция JavaScript `setInterval(function, time)` управляет временным интервалом в миллисекундах между HTTP-запросами, составляющим пять секунд для функции `reload` получения показаний температуры и одну секунду для состояния счетчика и светодиода. Сценарии JavaScript, заключенные между тегами `<script>...</script>`, располагаются перед секцией HTML-кода `</body>` для повышения скорости отображения веб-страницы.

Вся веб-страница больше не перезагружается, когда обновляется температура или состояние светодиода, так как обновляются только определенные переменные. В веб-браузере индикатор загрузки веб-страницы, расположенный рядом с заголовком веб-страницы, теперь отсутствует.

Листинг 8-4. AJAX-запрос для BMP280 и светодиода

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head><title>ESP8266</title></head>
<body>
<h2>BMP280</h2>
<p>Temperature: <span id = 'tempId'>0</span>&degC</p>
```

```
<p>Counter: <span id = 'countId'>0</span></p>
<p>LED is <span id = 'LEDid'> </span><p>
<script>
setInterval(reload, 5000);           // time in milliseconds
function reload()                    // reload function called every 5s
{                                    // to get tempId from tempUrl
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      document.getElementById('tempId').innerHTML =
      this.responseText;
   };
   xhr.open('GET', 'tempUrl', true);
   xhr.send();
}
setInterval(LEDreload, 1000);
function LEDreload()                 // LEDreload function called every 1s
{                                    // to get countId from countUrl
   var xhr = new XMLHttpRequest();   // and LEDid from LEDurl
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      document.getElementById('countId').innerHTML =
      this.responseText;
   };
   xhr.open('GET', 'countUrl', true);
   xhr.send();
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      document.getElementById('LEDid').innerHTML =
      this.responseText;
   };
   xhr.open('GET', 'LEDurl', true);
   xhr.send();
}
</script>
</body></html>
)";
```

Каждая переменная, отображаемая на веб-странице, связана с XML-файлом HTTP-запроса, URL-адресом, связанным с переменной, и функцией для получения значения переменной. Для согласованности и чтобы сделать скетч лучше интерпретируемым, XML-код веб-страницы и HTTP-запроса на переменную под именем *X* ссылается как *Xid*, а URL-адрес ссылается как *Xurl* в обоих функциях – `setup` основного скетча и HTTP-запроса, вместе с функциями поиска значения переменной, обозначенными как `Xfunct` в основном скетче.

JSON

В листинге 8-3 каждая переменная была связана с определенной функцией, прикрепленной к определенному URL-адресу; а в листинге 8-4 каждая пере-

менная обновлялась отдельным HTTP-запросом. Для данных, собираемых одновременно, более эффективно для получения информации иметь один HTTP-запрос для всех переменных и одну функцию, прикрепленную к одному URL-адресу. JSON объединяет несколько значений переменных в виде текста, который отправляется сервером клиенту. Клиент при обновлении веб-страницы выделяет из текста JSON отдельные переменные. Текст JSON состоит из пар «имя переменной – значение», каждая из которых заключена в двойные кавычки и разделена двоеточием, при этом пары разделены запятой, а текст JSON заключен в фигурные скобки {}. Примером текста в формате JSON с тремя парами имен переменных и их значений является `{"device": "LED", "state": "off", "pin": "15"}`.

В листинге 8-4 состояние счетчика и индикатора обновляется одновременно, так как объединяются два XML HTTP-запроса, два URL-адреса и две функции. Объединенный URL-адрес "*/countLEDurl*" и объединенная функция `countLEDfunct` определяются инструкцией `server.on("/countLEDurl",countLEDfunct)` в функции `setup` основного скетча. Функция `countLEDfunct` в основном скетче состоит из инструкций

```
void countLEDfunct()
{
  count++;                                    // increment count
  digitalWrite(LEDpin, !digitalRead(LEDpi n));
                                              // turn on or off the LED
  if(LED == "on") LED = "off";                // update LED state
  else LED = "on";
  JsonConvert(count, LED);                    // convert to JSON text
  server.send (200, "text/json", json);       // send JSON text to client
}
```

Строка `json` определена в основном скетче инструкцией `String json`. Обратите внимание, что инструкция `server.send()` указывает на отправку текста в формате JSON. Функция `JsonConvert` для объединения в тексте JSON целого числа `count` и строки `LED` состоит из инструкций

```
String JsonConvert(int val1, String val2)
{
  json = "{\"var1\": \"" + String(val1) + "\",";
  json += " \"var2\": \"" + val2 + "\"}";
  return json;
}
```

которые выдают текст JSON `{"var1": "123", "var2": "off"}`, если счетчик `var1` равен 123, а состояние светодиода `var2` выключенное. Пара символов \" подчеркнута, чтобы выделить, что символы находятся в паре, причем " интерпретируется как символ двойной кавычки, а не как индикатор конца строки. Символ \ (называемый обратным слешем) – признак управляющего символа (escape-символа[40]).

[40] Escape-символ (управляющий символ) – символ или группа символов, задающих определенные свойства последующего текста. Типичным примером могут служить символы \n (переход на другую строку) или \x (интерпретировать как шестнадцатеричное значение) в нотации языка C. – *Прим. перев.*

HTML-код для отображения состояния счетчика и светодиода изменен на

```
<p>Counter: <span id = 'var1'>0</span></p>
<p>LED is <span id = 'var2'> </span><p>
```

который ссылается на JSON-имена `var1` и `var2`. Код JavaScript для парсинга[41] текста, отправляемого сервером

```
document.getElementById('Xid').innerHTML = this.responseText
```

меняется на

```
var obj = JSON.parse(this.responseText);
document.getElementById('var1').innerHTML = obj.var1
document.getElementById('var2').innerHTML = obj.var2
```

где текст JSON разбирается на две пары имени-значения для счетчика `var1` и состояния индикатора `var2`.

Листинг 8-5 содержит обновленный код AJAX для веб-страницы путем объединения двух XML HTTP-запросов и двух URL-адресов для состояния счетчика и светодиода из листинга 8-4 и парсинга текста JSON, при этом обновления выделены жирным шрифтом.

Листинг 8-5. AJAX-код с разборкой текста JSON

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head><title>ESP8266</title></head>
<body>
<h2>BMP280</h2>
<p>Temperature: <span id = tempId>0</span>&degC</p>
<p>Counter: <span id = 'var1'>0</span></p>
<p>LED is <span id = 'var2'> </span><p>
<script>
setInterval(reload, 5000);    // time in milliseconds
function reload()             // update the temperature every 5s
{
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      {document.getElementById('tempId').innerHTML = this.
      responseText;}
   };
   xhr.open('GET', '/tempUrl', true);
   xhr.send();
}
setInterval(countLEDreload, 1000);
```

[41] Парсинг (от англ. *parsing*) – программный разбор, анализ какого-либо текста с целью извлечения определенных фрагментов или ключевых слов и их заданной интерпретации. Например, необходимым предварительным этапом компиляции является парсинг исходного текста программы, написанной на каком-либо языке программирования с целью выделения служебных слов и операторов. Другой пример парсинга – выделение и интерпретация управляющих элементов (тегов) в HTML-коде при отображении веб-страницы в браузере. Программу или часть программы, выполняющую парсинг, называют *парсером*. – *Прим. перев.*

```
function countLEDreload()     // update the counter and
{                             // LED state every second
   var xhr = new XMLHttpRequest();
   xhr.onreadystatechange = function()
   {
      if(this.readyState == 4 && this.status == 200)
      {                       // parse JSON text
         var obj = JSON.parse(this.responseText);
         document.getElementById('var1').innerHTML = obj.var1;
         document.getElementById('var2').innerHTML = obj.var2;
      }
   };
   xhr.open('GET', '/countLEDurl', true);
   xhr.send();
}
</script>
</body></html>
)";
```

Доступ к данным WWW

Отображение на веб-странице данных, предоставляемых микроконтроллером ESP8266 или ESP32, не требует доступа к интернету, поскольку микроконтроллер, выполняющий функции сервера, может устанавливать беспроводную сеть для соединения клиента с сервером, как описано в главе 7 («Беспроводная локальная сеть»). Для демонстрации отображения данных, доступных из Всемирной паутины, возьмем информацию о дате и времени, доступных с веб-сайтов www.calendardate.com/todays.htm и 24timezones.com/Edinburgh/time, а также о температуре и влажности для Эдинбурга, Шотландия, доступных с веб-сайта www.metoffice.gov.uk.

Информация получается с помощью интерфейса прикладного программирования API. Ключ для извлечения данных с помощью HTTP-запросов выдается *ThingSpeak* (www.thingspeak.com). В меню приложений *ThingSpeak* опция *ThingHTTP* генерирует ключ API для доступа к определенному элементу на заданной веб-странице. Например, ключ API для получения текущего времени веб-сайта 24timezones.com/Edinburgh/time создается, если щелкнуть правой кнопкой мыши по отображаемому времени и выбрать *Inspect Element* (Q). На отобразившейся *веб-консоли* щелкните три точки и выберите *Dock to Right*. HTML-код, соответствующий выбранному времени, выделяется синим цветом, и при наведении курсора на HTML-код создается поле, окружающее выбранный элемент на веб-странице. Щелкните правой кнопкой мыши выделенный HTML-код и выберите *Copy* (Копировать) и *XPath*. На веб-странице thingspeak.com/apps/thinghttp выберите *New ThingHTTP* и введите имя API и URL веб-страницы, содержащей данные, например 24timezones.com/Edinburgh/time, в поле *Parse String* вставьте скопированный XPath и нажмите *Save ThingHTTP*. Ключ API генерируется *ThingSpeak* для доступа к требуемой информации и проверяется с помощью загрузки веб-страницы с URL-адресом по образцу *https://api.thingspeak.com/apps/thinghttp/send_request?api_key=API key*.

Если *ThingSpeak* API-ключ возвращает сообщение “*Error parsing document, try a different parse string*” («Ошибка парсинга документа, попробуйте другую строку»), то требуется альтернативный источник данных или связанная веб-страница.

Информация, полученная с помощью *ThingSpeak* API-ключа, потребует разборки (парсинга). Например, сведения о дате и времени формируются как `<p>Today's Date is Monday June 15, 2020</p>` и `6:05:34 PM, Monday 15, June 2020` соответственно, в то время как температура и влажность представлены в формате HTML-кода, как `<div data-value="14.66">15°</div>` и `<span class="humidity" data-type="percentage" data-value="90">90%</span>` соответственно. Подстрока `date` формируется как текст, следующий за словом *is*. Подстрока `time` – это текст, предшествующий запятой. Обе подстроки температуры и влажности выделяются символами = и >. Функции `toInt` и `toFloat` извлекают из строки целое и действительное числа соответственно, при условии что первым символом строки является цифра.

Скетч в листинге 8-6 получает доступ к текущей дате, времени, температуре и влажности с помощью *ThingSpeak* API-ключей и преобразует информацию в строку JSON для включения в ответ сервера на HTTP-запрос клиента. Обратите внимание, что веб-страница обновляется с интервалом в 30 с, поэтому все начальные значения равны нулю.

Листинг 8-6. Парсинг данных, доступных с помощью ThingSpeak API-ключей

```
#include <ESP8266WebServer.h>         // include web server library
ESP8266WebServer server;              // associate server with library
WiFiClient client;                    // associate client with Wi-Fi library
#include "buildpage.h"                // webpage AJAX code
char ssid[] = "xxxx";                 // change xxxx to your Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to your Wi-Fi password
char APItime[] = "xxxx";
char APIdate[] = "xxxx";              // change xxxx to ThingSpeak API key
char APItemp[] = "xxxx";
char APIhumid[] = "xxxx";
char url[] = "/apps/thinghttp/send_request?api_key=";
char host[] = "api.thingspeak.com";
int indexS, indexF, chk, humid;
float temp;
String data, ndata, text, json, mdy, tim;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  WiFi.begin(ssid, password);         // connect and initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // display server IP address
  server.begin();                     // initialise server
  server.on("/", base);               // load default webpage
  server.on("/API", APIfunct);
}
void APIfunct()
{
  getData(APIdate, "date");           // call function to access date
  getData(APItime, "time");           // time
  getData(APItemp, "temp");           // temperature
  getData(APIhumid, "humid");         // humidity
  JsonConvert(mdy, tim, temp, humid); // convert to JSON text
  server.send (200, "text/json", json);
}
```

```
String JsonConvert(String val1, String val2, float val3, int val4)
{
   json = "{\"var1\": \"" + val1 + "\",";             // start with {
   json += " \"var2\": \"" + val2 + " \",";
                                                      // end with comma
   json += " \"var3\": \"" + String(val3) + "\",";
   json += " \"var4\": \"" + String(val4) + " \"}";   // end with }
   return json;
}
void getData(String APIkey, String text)      // function to access data
{
   for (int i=0; i<5; i++)                    // with up to five attempts
   {
      getVal(APIkey, text);           // call function to get information
      if(chk > 0) i = 5;              // data accessed successfully
   }
}
void getVal(String APIkey, Stri ng text)
{                                     // function to access information
   chk = 0;
   Serial.print("sourcing ");Serial.println(text);
   client.connect(host, 80);
   client.println(String("GET ") + url + APIkey);
   client.println(String("Host: ") + host);
   client.println("User-Agent: ESP8266/0.1");
   client.println("Connection: close");
   client.println();
   client.flush();
   delay(100);
   while(client.connected())          // while connected to ThingSpeak
   {
      if(client.available())          // if data is available
      { // read data till end of line
         data = client.readStringUnti l('\n');
         Serial.println(data);
         if(text == "humid")          // parse humidity data
         {
            indexS = data.lastIndexOf("=");
                                      // position of last "=" in string
            indexF = data.indexOf("%");
            ndata = data.substring(indexS+2, indexF-2);
            humid = ndata.toInt();
            chk = data.length();
         }
         else if(text == "temp")      // parse temperature data
         {
            indexS = data.indexOf("=");
                                      // position of first "=" in string
            ndata = data.substring(indexS+2);
            temp = ndata.toFloat();
            chk = data.length();
         }
         else if(text == "date")      // date: day month dd, yyyy
         {
            indexS = data.indexOf("is");
            mdy = data.substring(indexS+2);
```

```
            chk = data.length();
          }
          else if(text == "time")       // time: hh:mm:ss AM or PM
          {
            indexF = data.indexOf(",");
            tim = data.substring(0, indexF);
            chk = data.length();
          }
          client.stop();                // close connection after data collected
          delay(100);
        }
      }
    }
    void base()                         // function to return HTML code
    {
      server.send (200, "text/html", page);
    }
    void loop()
    {
      server.handleClient();            // handle HTTP requests
    }
```

Проанализированная информация отображается на веб-странице с интервалом в 30 с (см. листинг 8-7) только для примера, поскольку погода обычно меняется не так быстро. Для параметров даты, времени и погоды можно выбрать различные временные интервалы, определив отдельные функции перезагрузки с соответствующими временными интервалами.

Листинг 8-7. AJAX-код с JSON-парсингом

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head><title>ESP8266</title></head>
<body>
<h2>BMP280</h2>
<p>Date: <span id = 'var1'>00 000 0000</span></p>
<p>Time1: <span id = 'var2'>00:00:00</span></p>
<p>Temp is <span id = 'var3'>0</span>&degC<p>
<p>Humidity is <span id = 'var4'>0</span>%<p>
<script>
setInterval(APIreload, 30000);              // time in milliseconds
function APIreload()
{
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function()
  {
    if(this.readyState == 4 && this.status == 200)
    {
      var obj = JSON.parse(this.responseText);
      document.getElementById('var1').innerHTML = obj.var1;
      document.getElementById('var2').innerHTML = obj.var2;
      document.getElementById('var3').innerHTML = obj.var3;
      document.getElementById('var4').innerHTML = obj.var4;
    }
  };
  xhr.open('GET', 'API', true);
```

```
    xhr.send();
}
</script>
</body></html>
)";
```

MQTT-брокер и IFTTT

Для связи между устройствами в разных сетях Wi-Fi требуется иное решение, чем для связи между устройствами внутри одной сети Wi-Fi. Протокол MQTT[42] обеспечивает связь между устройствами через MQTT-брокер, который обменивается информацией между ним и одним устройством и между ним и вторым устройством, причем эти два устройства могут находиться в разных сетях Wi-Fi. MQTT-брокер обеспечивает передачу данных между устройствами без нарушения защитного сетевого экрана. Когда устройство в одной сети Wi-Fi запрашивает информацию у второго устройства в другой сети, передача информации разрешена через сетевой экран, так как запрос поступает из «своей» сети Wi-Fi. Предоставление информации MQTT-брокеру называется публикацией (*publish*), а получение доступа к информации от брокера называется подпиской (*subscribe*). Существует несколько MQTT-брокеров, в этой главе используется MQTT-брокер Cayenne.

Cayenne (см. mydevices.com/cayenne/features) предоставляет панель мониторинга для отображения информации с устройств, подключенных к микроконтроллеру ESP8266 или ESP32 (см. рис. 8-4). Панель управления *Cayenne* можно посмотреть локально или удаленно на cayenne.mydevices.com/cayenne/dashboard/start. Информация с устройств отображается в цифровом виде, в виде циферблата или графически, с двоичными переменными, отображаемыми как *ON/OFF*. Устройство включается или выключается с панели управления *Cayenne*, которая обеспечивает как локальный, так и удаленный доступ к устройству.

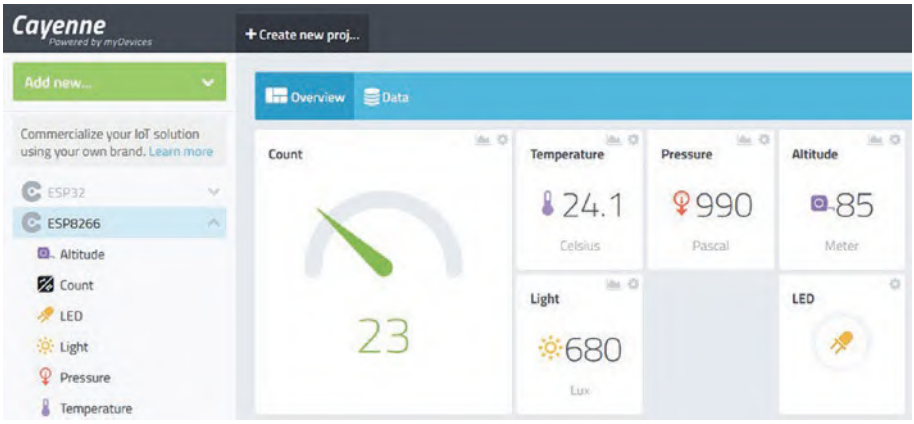


Рисунок 8-4. Панель управления Cayenne

[42] MQTT (Message Queuing Telemetry Transport, передача телеметрии с очередью сообщений) – протокол обмена сообщениями между «умными» устройствами различной природы. Связь осуществляется через «посредника» – MQTT-сервер, который принято называть MQTT-брокером. – *Прим. перев.*

Функция IFTTT[43] позволяет запускать события на основе датчиков, подключенных к плате ESP8266 или ESP32 и видимых на панели управления *Cayenne*. Например, если освещенность фоторезистора (LDR) увеличивается выше порогового значения из-за открытия двери или наступления светлого времени суток, то на Cayenne MQTT-брокер отправляется IFTTT-команда, вызывающая посылку электронного письма или текстового сообщения на адрес электронной почты или номер мобильного телефона, сохраненный в панели управления *Cayenne*.

Подробная информация о Cayenne доступна по адресу mydevices.com/cayenne/ docs/intro, библиотека *CayenneMQTT* доступна в Arduino IDE. Связь между микроконтроллером ESP8266 или ESP32 и Cayenne MQTT осуществляется по виртуальным каналам, которые имеют произвольные номера V0, V1, V2 и т. д. Инструкция по отправке данных в панель управления *Cayenne* – это `Cayenne.virtualWrite(virtual channel, variable, type code, unit code)`, где `type code` и `unit code` определяют атрибуты переменной. Некоторые примеры кодов `type` и `unit` приведены в табл. 8-2, полный список находится в файле библиотеки *CayenneMQTT>src>CayenneUtils>CayenneTypes.h*. Например, если переменная `light` является мерой освещенности (*luminosity*) в люксах (*lux*), то инструкция, отправляющая значение `light` на панель управления *Cayenne* по виртуальному каналу V3, будет `Cayenne.virtualWrite(V3, light, "lum", "lux")`.

Таблица 8-2. Имена и коды переменных type и unit

Описание	Имя type	Код type
Атмосферное давление	TYPE_BAROMETRIC_PRESSURE	“bp”
Расстояние	TYPE_PROXIMITY	“prox”
Освещенность	TYPE_LUMINOSITY	“lum”
Относительная влажность	TYPE_RELATIVE_HUMIDITY	“rel_hum”
Температура	TYPE_TEMPERAT URE	“temp”
Описание	Имя unit	Код unit
Гектопаскаль	UNIT_HECTOPASCAL	“hpa”
Метр	UNIT_METER	“m”
Люкс	UNIT_LUX	“lux”
Градус Фаренгейта	UNIT_FAHRENHEIT	“f”
Градус Цельсия	UNIT_CELSIUS	“c”

Включение кодов *type* и *unit* в инструкцию `Cayenne.virtualWrite()` автоматически настраивает панель управления *Cayenne*, добавляя описание переменной, соответствующий значок и единицу измерения. Обратите внимание, что обновление инструкций `Cayenne.virtualWrite()` ограничено частотой 60 в мину-

[43] IFTTT (расшифровывается как If This, Then That) – платформонезависимый сервис, позволяющий создавать правила для запуска различных действий по достижению определенных условий. Необходимо отметить, что с сентября 2020 года сервисы IFTTT распространяются на платной основе. – *Прим. перев.*

ту, поэтому в листинге 8-8 далее интервал между сообщениями MQTT составляет десять секунд.

Инструкция для чтения целочисленной переменной на виртуальном канале 3 на панели управления *Cayenne* выглядит следующим образом:

```
CAYENNE_IN(3) // define virtual channel number 3
{
   int variable = getValue.asInt(); //read value of integer variable
}
```

Функции `getValue.asDouble()` и `getValue.asString()` считывают действительное число и строку соответственно, с номером канала, которому не предшествует V, как в инструкции `Cayenne.virtualWrite()`.

Информация об объявлении устройств или переменных, таких как состояние светодиода или показания фотодиода, на приборной панели *Cayenne* доступны по адресу mydevices.com/cayenne/docs/features/#features-dashboard.

Микроконтроллер ESP8266 или ESP32 добавляется в качестве устройства Cayenne dashboard с помощью следующих действий:

1. Выберите *Add new* в верхней левой части панели инструментов.
2. Выберите *Device/Widget* и выберите *Microcontrollers* ➤ *Generic ESP8266* или *Bring Your Own Thing* для микроконтроллеров ESP8266 или ESP32 соответственно.

Соответствующие имя пользователя и пароль MQTT, идентификатор клиента, MQTT-сервер и сведения о порте генерируются Cayenne MQTT-брокером. Скопируйте имя пользователя и пароль MQTT и идентификатор клиента (Client ID) в скетч; затем скомпилируйте и загрузите его. Микроконтроллер ESP8266 или ESP32 затем подключится к Cayenne MQTT-брокеру. Добавление имени устройства в Cayenne различается для микроконтроллеров ESP8266 и ESP32.

Виджеты панели управления *Cayenne* настраиваются следующим образом:

1. Выберите *Add new* в верхней левой части панели инструментов.
2. Выберите *Device/Widget*, *Custom Widgets* и *Button (Controller widget)*.
3. Введите название выбранного виджета, например *LED*.
4. Введите имя устройства, например *ESP32-A* или *ESP8266-project1*.
5. Выберите *Data* ➤ *Digital Actuator* и *Unit* ➤ *Digital (0/1)*.
6. Выбор номера виртуального канала в соответствии со скетчем.
7. Выберите значок и *Add Widget*.

На рис. 8-5 показаны примеры определения виджета контроллера LED, связанного с виртуальным каналом 0, и форматирования переменной счетчика на виртуальном канале 6 для отображения в виде индикатора на панели управления *Cayenne*.

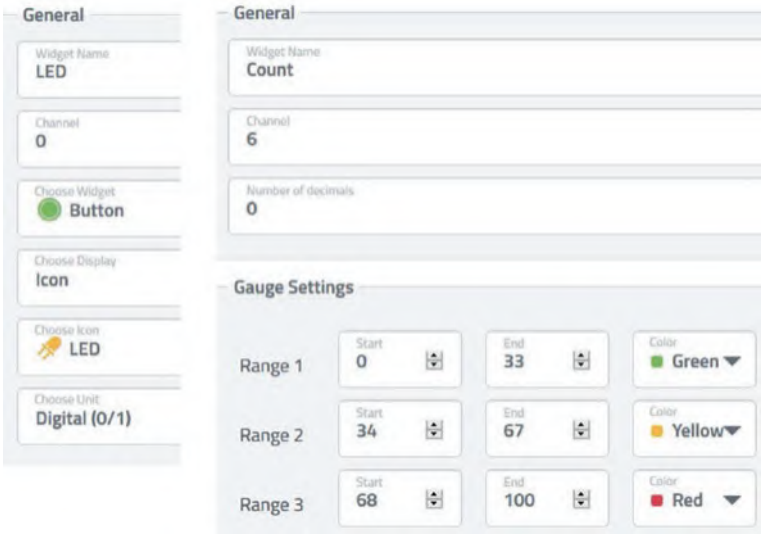


Рисунок 8-5. Переменные и устройства Cayenne

Листинг 8-8 отображает на панели управления или в приложении Cayenne (см. рис. 8-4) результаты измерения температуры и давления с помощью датчика BMP280, окружающего освещение с использованием фоторезистора и состояние счетчика. В серии инструкций `Cayenne.virtualWrite` виртуальный канал *V2* не используется, чтобы избежать путаницы с GPIO 2 для переменной `flashPin`. Заголовки переменных определяются с помощью параметра настройки переменных на панели управления Cayenne, и для каждой переменной также выбираются графические отображения. Виджет контроллера на виртуальном канале 0 создается на панели управления *Cayenne* с помощью виджета двоичного значения, получаемого с помощью функции `CAYENNE_IN(0)`, включающей или выключающей светодиод, подключенный к плате ESP8266.

Листинг 8-8. Cayenne и ESP8266 со светодиодом, фоторезистором и датчиком BMP820

```
#include <CayenneMQTTESP8266.h>       // Cayenne MQTT library
char ssid[] = "xxxx";                 // change xxxx to your Wi-Fi ssid
char password[] = "xxxx";             // change xxxx to Wi-Fi password
char username[] = "xxxx";             // change xxxx to Cayenne username
char mqttpass[] = "xxxx";             // change xxxx to Cayenne password
char clientID[] = "xxxx";             // change xxxx to Cayenne client identity
#include <Adafruit_Sensor.h>          // include Adafruit_Sensor library
#include <Adafruit_BMP280.h>          // include Adafruit_BMP280 library
Adafruit_BMP280 bmp;                  // bmp with BMP280 library
int LEDpin = D3;                      // LED pin
int LDRpin = A0;                      // light dependent resistor pin
int flashPin = D4;                    // flashing LED pin
unsigned long count = 0;
int interval = 10000;                 // 10s interval between MQTT messages
unsigned long lastTime = 0;
float temp, pressure, BasePressure, altitude;
int light;
```

```
void setup()
{
   bmp.begin(0x76);                              // initiate bmp with I2C address
                                                 // initiate Cayenne MQTT
   Cayenne.begin(username, mqttpass, clientID, ssid, password);
   pinMode(LEDpin, OUTPUT);                      // define LED pins as output
   digitalWrite(LEDpin, LOW);
   pinMode(flashPin, OUTPUT);
}
void loop()
{
   Cayenne.loop();                               // Cayenne loop function
   if(millis()-lastTime > interval)
   {
      temp = bmp.readTemperature();              // BMP280 temperature
      pressure = bmp.readPressure()/100.0;       // and pressure
      BasePressure = pressure + 10.0;            // assumed sea level pressure
      altitude = bmp.readAltitude(BasePress ure);
                                                 // predicted altitude (m)
      light = analogRead(LDRpin);                // ambient light intensity
      light = constrain(light, 0, 1023);         // constrain light reading
      count++;                                   // increment counter
      if(count>99) count = 0;
      digitalWrite(flashPin, LOW);               // turn on then off flashing LED
      delay(10);
      digitalWrite(flashPin, HIGH);
                            // send readings to Cayenne on virtual channels
      Cayenne.virtualWrite(V1, temp, "temp", "c");
                                                          // temperature reading
      Cayenne.virtualWrite(V3, pressure, "bp", "pa");  // pressure
      Cayenne.virtualWrite(V4, altitude, "prox", "m"); // altitude
      Cayenne.virtualWrite(V5, light, "lum", "lux");   // luminosity
      Cayenne.virtualWrite(V6, count,"prox","");       // counter
      lastTime=millis();                               // update time
   }
}
CAYENNE_IN(0)                                    // Cayenne virtual channel 0
{
   digitalWrite(LEDpin, getValue.asInt( ));      // turn on or off LED
}
```

В листинге 8-9 используется функциональность Cayenne MQTT для имитации системы сигнализации, которая срабатывает при изменении интенсивности света на фоторезисторе, например при открытии двери. Если интенсивность освещения превышает пороговое значение 500 и на приборной панели *Cayenne* включена (*ON*) сигнализация, отправляется электронное письмо и/или текстовое сообщение с уведомлением о том, что событие произошло. Если для настройки сигнализации установлено значение *OFF*, то реакция на изменение интенсивности света отсутствует. Светодиод на плате ESP8266 или ESP32 мигает каждые две секунды, указывая на то, что микроконтроллер включен.

Если настройка сигнализации установлена на значение *ON*, то MQTT-брокеру по виртуальному каналу 1 отправляются показания интенсивности света, или значение, равное нулю, если для сигнализации установлено зна-

чение *OFF*. Виджеты сигнализации и светодиодного контроллера на виртуальных каналах 3 и 0 создаются на приборной панели *Cayenne* для включения или выключения сигнализации и для отображения индикатора срабатывания сигнализации. К плате ESP8266 или ESP32 подключены синий (`alarmPin`) и красный (`LEDpin`) светодиоды, которые соответственно указывают на состояние сигнализации и на то, что сигнализация сработала. Два светодиода включаются или выключаются с помощью функций `CAYENNE_IN(3)` и `CAYENNE_IN(0)`. Сигналы для светодиодного виджета, а также для уведомлений по электронной почте и текстовых уведомлений, основанных на виджете сигнализации, определены в функции `IFTTT` панели управления *Cayenne*. Рисунок 8-6 показывает панель управления *Cayenne* с включенной сигнализацией и показателем интенсивности света 268, что ниже порогового значения для срабатывания сигнализации и включения виджета со светодиодным индикатором. На рис. 8-7 показана схема с подключениями из табл. 8-3.

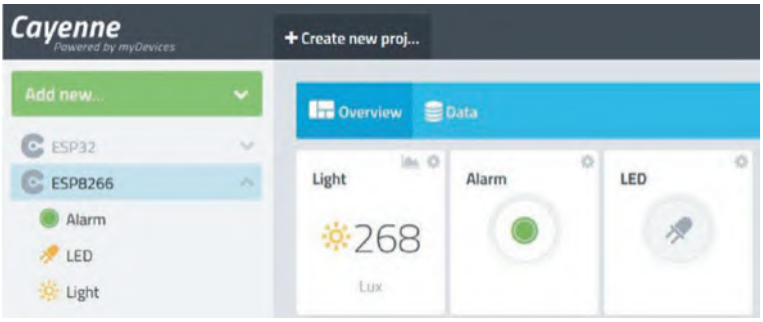


Рисунок 8-6. Сигнализация, светодиод и измерение освещенности

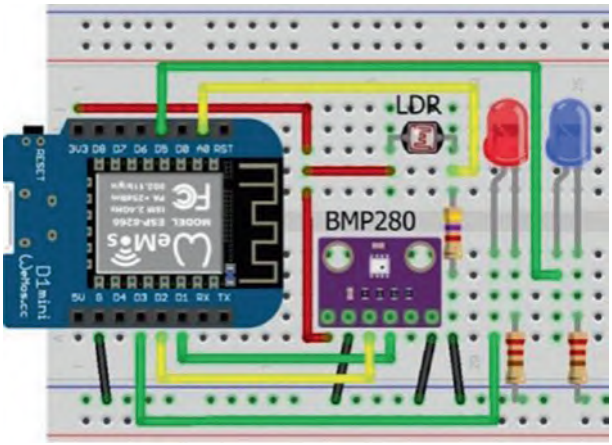


Рисунок 8-7. Сигнализация, светодиод и фоторезистор (LDR) с платой LOLIN (WeMos) D1 mini

Таблица 8-3. Сигнализация, светодиод и фоторезистор (LDR)

Компонент	Подключение к ESP8266	Подключение к ESP32
BMP280 VCC	3V3	3V3
BMP280 GND	GND	GND
BMP280 SDA	D2	GPIO 21
BMP280 SCK	D1	GPIO 22
BMP280 SDO	Не подключен	Не подключен
LDR левый	3V3	3V3
LDR правый	4,7 ком → GND	4,7 ком → GND
LDR правый	A0	GPIO 36
LED длинный вывод	D3, D5	GPIO 23
LED короткий вывод	Резистор 220 Ом → GND	Резистор 220 Ом → GND

Листинг 8-9. Сигнализация, светодиод и измерение освещенности

```
#include <CayenneMQTTESP8266.h>       // Cayenne MQTT library
char ssid[] = "xxxx";                 // change xxxx to your Wi-Fi ssid
char password[] = "xxxx";             // change xxxx to your Wi-Fi password
char username[] = "xxxx";             // change xxxx to Cayenne username
char mqttpass[] = "xxxx";             // change xxxx to Cayenne password
char clientID[] = "xxxx";             // change xxxx to Cayenne client identity
int LEDpin = D3;
int alarmPin = D5;                    // define LED, alarm and LDR pins
int LDRpin = A0;
int flashPin = D4;                    // flashing LED
int reading, alarm, alert;
int interval = 2000;                  // 2s interval between LDR readings
unsigned long LDRtime = 0;
void setup()
{
  Cayenne.begin(username, mqttpass, clientID, ssid, password);
  pinMode(LEDpin, OUTPUT);            // define LED pins as output
  pinMode(alarmPin, OUTPUT);
  pinMode(flashPin, OUTPUT);
  alarm = 0;                          // set alarm to OFF
}
void loop()
{
  Cayenne.loop();                     // Cayenne loop function
  if(millis() - LDRtime > interval)
  {
    LDRtime = millis();
    reading = analogRead(LDRpin);
    if (alarm == 1)
          Cayenne.virtualWrite(V1, reading, "lum", "lux");
    else Cayenne.virtualWrite(V1, 0, "lum", "lux");
    digitalWrite(flashPin, LOW);
    delay(10);                        // flash LED to indicate power on
    digitalWrite(flashPin, HIGH);
  }
}
```

```
CAYENNE_IN(0)                          // Cayenne virtual channel 0
{
  alert = getValue.asInt();            // get alarm triggered status
  digitalWrite(LEDpin, alert);         // update alarm triggered LED
}
CAYENNE_IN(3)                          // Cayenne virtual channel 3
{
  alarm = getValue.asInt();            // get alarm set state
  digitalWrite(alarmPin, alarm);       // update alarm set indicator LED
}
```

Функция IFTTT для запуска события на панели управления *Cayenne* определена в Cayenne, а не в скетче. Информация о функциях-триггерах IFTTT панели управления *Cayenne* доступна по адресу mydevices.com/cayenne/docs/features/#featurestriggers. Для системы сигнализации требуются три триггера IFTTT. Когда интенсивность света превышает пороговое значение 500, при включенном виджете сигнализации запускается электронное и текстовое уведомление о событии, второй триггер включает светодиодный виджет на виртуальном канале 0, чтобы указать, что сработал сигнал тревоги. Третий триггер IFTTT отключает виджет сигнализации на виртуальном канале 3, что указывает на выключенную сигнализацию. Светодиоды, подключенные к плате ESP8266 или ESP32, включаются или выключаются в зависимости от значений в виджетах будильника и светодиода на панели управления *Cayenne*.

Доступ к функциям Cayenne IFTTT осуществляется с помощью следующих действий:

1. Выберите *User Menu* ➤ *Triggers and Alerts* в правом верхнем углу панели мониторинга.
2. Выберите *Trigger* и присвойте ему имя, например LEDon.
3. Перетащите устройство *ESP8266* в поле *if*.
4. Выберите функцию, такую как освещенность (*light*), а также порог (*threshold*) и условие срабатывания – выше (*Sensor above*) или ниже (*Sensor below*) порогового значения с датчика.
5. Перетащите устройство *ESP8266* в поле *then*.
6. Выберите действие, например управление светодиодом (*LED* на рис. 8-8), и установите либо *On(1)*, либо *Off (0)*.
7. Выберите *Save*.

При отправке уведомления в виде текстового сообщения укажите номер мобильного телефона плюс код страны в поле *Add custom recipient*.

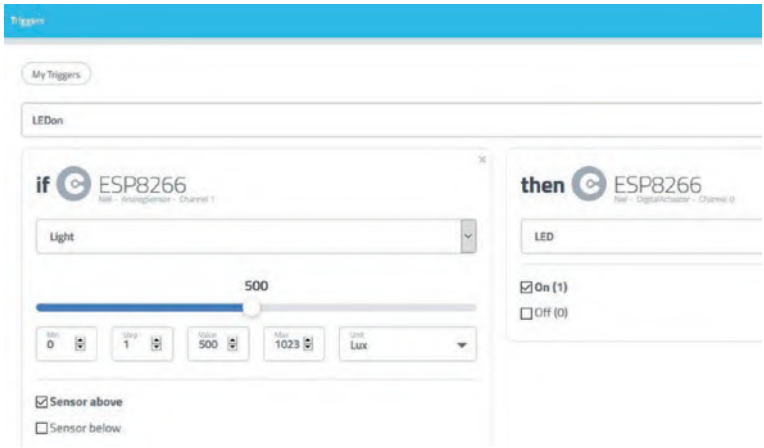


Рисунок 8-8. Cayenne IFTTT-триггер

На рис. 8-8 показана функция-триггер IFTTT для включения светодиодного виджета на виртуальном канале 0, когда интенсивность света на виртуальном канале 1 превышает пороговое значение 500. Второй и третий триггеры IFTTT отправляют электронное письмо или текстовое уведомление и выключают виджет сигнализации на виртуальном канале 3, когда включается светодиодный виджет. На рис. 8-9 показано соответствующее уведомление по электронной почте, посланное IFTTT.

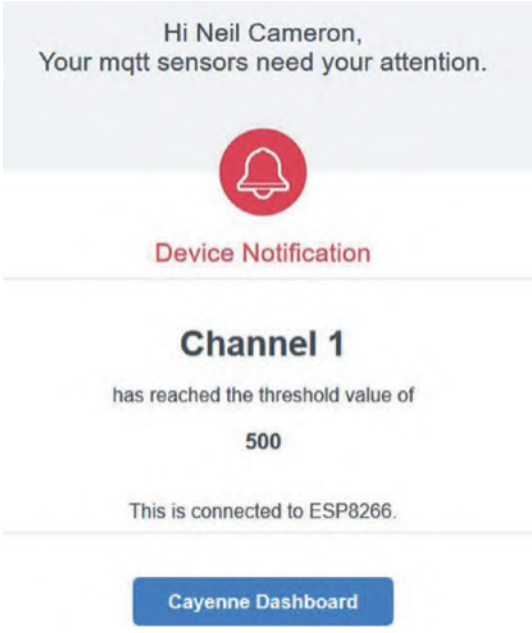


Рисунок 8-9. Сообщение Cayenne IFTTT

Листинги 8-8 и 8-9 используют библиотеку для микроконтроллера ESP8266. Доступ к библиотеке для микроконтроллера ESP32 осуществляется с помощью инструкции `#include <CayenneMQTTESP32.h>`, которая включена в библиотеку *CayenneMQTT*. Таким образом, в скетчах с микроконтроллером ESP32 не требуется никаких изменений, кроме других номеров контактов для подключенных датчиков.

Парсинг текста

Парсинг (разбор) текста, например данных из последовательного порта или данных, загруженных в результате HTTP-запроса, требуется очень часто. Инструкция `Serial.read()` считывает следующий доступный символ из буфера последовательного порта, в то время как инструкция `Serial.readStringUntil('\n')` считывает все данные из буфера до символа конца строки. Содержимое буфера сохраняется в виде строки `str` с помощью инструкции `str = Serial.readString()` и превращается в целое или действительное число с помощью команд парсинга `Serial.parseInt()` или `Serial.parseFloat()`. Например, если буфер последовательного порта содержит текст *abc25* или *abc3.14*, то в результате парсинга возвращается целое число 25 или действительное число 3.14 соответственно. Действие парсинга игнорирует начальные нецифровые символы, отличные от десятичной точки или знака минус, и останавливается, когда после последнего символа цифры считывается нецифровой символ. Инструкции парсинга повторяются для извлечения более одного числа из буфера. Например, если буфер последовательного порта содержит *abc-25de3.14*, то следующими инструкциями возвращается целое число –25 и действительное 3.14:

```
if(Serial.available()>0)
{
  x = Serial.parseInt();
  y = Serial.parseFloat();
}
```

Функции `parseInt` и `parseFloat` являются блокирующими функциями, которые не позволяют микроконтроллеру обрабатывать другие инструкции до завершения парсинга.

Строки преобразуются в целые или вещественные числа с помощью инструкций `toInt()` или `toFloat()` соответственно, при условии что первым символом строки является цифра, десятичная точка или знак минус. Например, если строка `str` равна *–25abc* или *3,14abc*, то инструкции `str.toInt()` или `str.toFloat()` возвращают целое число –25 или действительное число 3,14 соответственно. Парсинг останавливается, когда после символа цифры или десятичной точки считывается нецифровой символ. Обратите внимание, что действительное число при хранении занимает от шести или семи десятичных разрядов[44], поэтому при преобразовании строки 2.7182818284 в действительное число в результате получается значение 2,718282.

[44] Это справедливо для типа *float* в Arduino IDE, в стандартах для других микропроцессоров и языков программирования представление действительных чисел отличается. Отметим, что и в Arduino IDE, кроме *float*, существует тип *double*, который совпадает с *float* для 8-разрядных контроллеров ATmega, но может отличаться для 32-разрядных контроллеров. – *Прим. перев.*

Строка, которая не начинается с символа цифры, анализируется путем извлечения подстроки, которая начинается с символа цифры, десятичной точки или знака минус. Для строки `str` инструкция `str.substring(x, y)` создает подстроку между позициями *x* (включительно) и *y* (исключительно). Например, позиции запятой в строке str = "abc,def,gh" равны 4 и 8, таким образом, инструкция `str.substring(4+1, 8)` создает подстроку *def*, состоящую из символов 5, 6 и 7 исходной строки. Если последний параметр опущен, как в инструкции `str.substring(x)`, то подстрока продолжается до конца строки. Например, инструкция `str.substring(5)` создает подстроку *def,gh*.

Инструкции `str.indexOf("x")` и `str.indexOf("x", y)` определяют положение подстроки *x* внутри строки поиском от первого до последнего символа или от позиции *y* до последнего символа. Аналогично инструкции `str.lastIndexOf("x")` и `str.lastIndexOf("x", y)` определяют положение подстроки *x* в строке поиском от последнего до первого символа или от позиции *y* до первого символа. Функции `indexOf` и `lastIndexOf` объединяются с функцией `substring` для выделения из строки определенной подстроки, когда в строке имеется несколько отдельных подстрок.

Например, десятичное значение влажности содержится в строке `str = <span class="влажность" data-value="68.1">68%</span>`, находясь между символами «=» и «>», но в строке есть две группы цифровых символов. Функция `toFloat` не может извлечь значение влажности из полной строки, так как первый символ строки не является цифрой, поэтому должна быть создана подстрока для извлечения десятичного числа, причем не целого, а реального. Инструкции `indexS = str.lastIndexOf("=")` и `indexF = str.indexOf("%")` определяют позиции последнего символа «=» и символа «%», при этом нужная подстрока, содержащая символы «68.1">», будет определяться инструкцией `str.substring(indexS+2, indexF-2)`, из результата которой десятичное значение может быть извлечено с помощью функции `toFloat`.

Инструкция `str.length()` определяет длину строки, не включая нулевой завершающий символ `\n`, и полученная длина строки используется для обеспечения того, чтобы позиция подстроки ее не превышала. Чтобы проверить, начинается или заканчивается подстрокой `abc` строка `str`, выполните инструкции `str.startsWith("abc")` и `str.endsWith("abc")`, они возвращают значение один, если утверждение верно, или значение ноль, если ошибочно.

Ведение логов консоли

При отладке скетча или проверке хода его выполнения информация о переменной отображается на мониторе последовательного порта с помощью инструкции `Serial.print("text")`. Эквивалентом отображения информации через монитор последовательного порта для веб-браузера является ведение лога (журнала) консоли с использованием кода JavaScript для отображения значения переменной или текста, например "button pressed" (кнопка нажата). Доступ к логу консоли осуществляется в браузере нажатием на клавиатуре клавиши *F12* и выбором *Console* и *Logs* (см. рис. 8-10). Чтобы определить кодировку символов для журнала консоли, в раздел `<head>` HTML-кода веб-страницы должна быть включена инструкция `<meta charset='UTF-8'>`. Инструкция `console.`

`log(variable)` записывает информацию в лог, где `variable` может быть равна переменной, некоторому тексту или ответу сервера клиенту. Например, в листинге 8-5 инструкция `console.log(this.responseText)` вставлена после `document.getElementById('tempid').innerHTML = this.responseText`, а инструкции

```
console.log("LED updated");
console.log(this.responseText);
console.log(obj.var1);
var value = document.getElementById('var2').innerHTML;
console.log(value);
```

включаются после инструкции

```
document.getElementById('var2').innerHTML = obj.var2
```

Для получения повторных показаний температуры интервал функции `reload` был уменьшен до одной секунды, а интервал функции `countLEDreload` для счетчика и светодиода был увеличен до пяти секунд. Содержание лога показано на рис. 8-10: пять значений температуры, текст "LED updated," (*светодиод обновлен*), текст JSON, полученный клиентом, счетчик и состояние светодиода, определенное как новая переменная `value`.

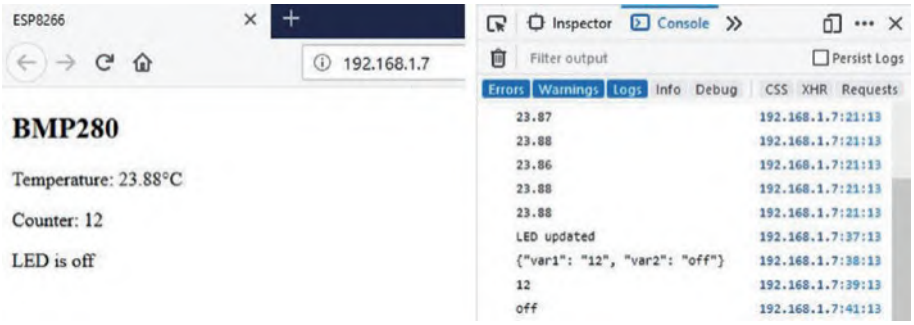


Рисунок 8-10. Лог консоли

Подключение к Wi-Fi

Библиотеки *ESP8266* и *ESP32* устанавливаются автоматически, когда вы устанавливаете *esp8266* от ESP8266 Community и *esp32* от Espressif Systems в менеджере плат Arduino IDE. В этой главе использовались библиотеки версий 2.7.4 и 1.0.4. Если сообщение об ошибке загрузки *http://downloads.arduino.cc/packages/packages_index.json* появляется в менеджере плат Arduino IDE, удалите все файлы *.tmp* в папке <*имя пользователя*> ➤ *AppData* ➤ *Local* ➤ *Arduino15* и перезапустите Arduino IDE.

Соединение Wi-Fi между микроконтроллером ESP8266 или ESP32 и маршрутизатором WLAN проверяется с помощью команды *ping* для заданного IP-адреса ESP8266 или ESP32. Либо щелкните правой кнопкой мыши логотип Windows в нижней левой части экрана и введите «*Выполнить*» (“*Execute*”), либо одновременно нажмите клавишу *Windows* и клавишу *R*, чтобы сразу открыть окно ввода команд. Введите *cmd* и нажмите кнопку *OK*. В отрывшемся окне командной строки введите *ping*, за которым должен следовать

IP-адрес ESP8266 или ESP32, как показано на рис. 8-11. Команда *ping* отправляет небольшие пакеты данных на IP-адрес ESP8266 или ESP32, которые передаются обратно отправителю. В приведенном примере было отправлено и получено четыре пакета данных, которые указывали на то, что канал передачи данных, соединение Wi-Fi и интернет-протокол функционировали правильно.

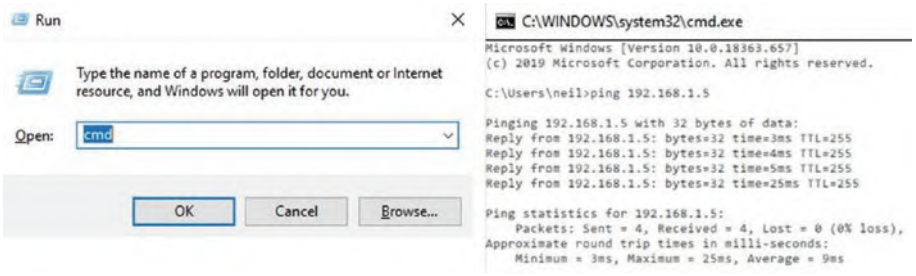


Рисунок 8-11. Проверка Wi-Fi-подключения

Файл с информацией о доступе

Для доступа к сети Wi-Fi, к данным ThingSpeak (см. листинг 8-6) или к MQTT-брокеру (см. листинг 8-8) требуется пароль, SSID, ключ API или ключи брокера MQTT. Вместо того чтобы хранить информацию о доступе в скетче с помощью инструкций типа `char mqttpass[] = "abcdef"`, информация хранится в библиотеке, на которую ссылается скетч. Для хранения информации о доступе создается текстовый файл с расширением *.h*, который помещается в папку *libraries* Arduino IDE. Чтобы определить местоположение папки библиотек *libraries*, выберите *Файл* ➤ *Настройки* (*File* ➤ *Preferences*) в среде IDE Arduino; и папка *libraries* будет находиться по пути, показанному в пункте *Размещение папки скетчей* (*Sketchbook location*), например *C:\Users\user\Documents\Arduino*[45]. На файл информации о доступе ссылается скетч с инструкцией `#include <access_info.h>`. Пример файла с информацией о доступе в листинге 8-10 содержит ключи доступа для Wi-Fi, ThingSpeak и Cayenne.

Листинг 8-10. Информация для доступа

```
char ssid[] = "PhoneNetwork12";       // Wi-Fi access
char password[] = "diff1cu1t";
char APItime[] = "efth1234";
char APIdate[] = "mhtd5678";          // ThingSpeak API keys
char APItemp[] = "plmf4567";
char APIhumid[] = "thkl6789";
char username[] = "ABC-234";          // Cayenne access
char mqttpass[] = "XYZ-567";
char clientID[] = "GHJ-876";
```

[45] Кроме папки *libraries*, размещенной в пользовательской папке со скетчами по указанному автором пути, имеется также папка *libraries* в основной папке Arduino. Подробнее см. сноску 140 в приложении, на стр. 452. – *Прим. перев.*

Итоги

На веб-странице отображались показания температуры модуля BMP280, счетчика и состояния светодиода, при этом микроконтроллер ESP8266 или ESP32 функционировал в качестве сервера. HTML-код веб-страницы построен построчно в виде строки, и вся веб-страница перезагружается для отображения обновленной информации. Код AJAX, состоящий из XML HTTP-запросов и инструкций JavaScript, включенных в виде символьной строки, позволил обновлять на веб-странице только определенные переменные, а не перезагружать ее всю целиком. Данные по нескольким переменным были объединены в виде текста JSON в ответе сервера на HTTP-запрос клиента, были описаны варианты разбора (парсинга) этого текста. Доступ к информации из Всемирной паутины осуществлялся с помощью ключей API и отображался на веб-странице с использованием как AJAX, так и JSON-кода. Доступ к MQTT-брокеру позволял загружать данные датчиков на веб-страницу, при этом значение датчика, превышающее пороговое значение, вызывало уведомление о событии по электронной почте или текстовом сообщении. В логе консоли для проверки сохраняются данные, отправленные сервером и полученные клиентом.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU
- Модуль BMP280
- Светодиод: 2 шт.
- Резистор: 220 Ом 2 шт.

Глава 9

WebSocket

Протокол WebSocket (www.websocket.org) позволяет осуществлять двусторонний диалог в режиме реального времени между веб-сервером и клиентом через стандартизированное соединение, позволяющее как серверу, так и клиенту отправлять данные в любое время. Клиент отправляет серверу запрос на переключение с протокола HTTP на протокол WebSocket; и если сервер может разместить протокол WebSocket, HTTP-соединение заменяется подключением WebSocket, но с использованием того же порта, что и HTTP.

Примером протокола WebSocket является передача и прием текста в диалоге между веб-сервером и клиентом (см. рис. 9-1). Для передачи на сервер текст вводится в поле *transmit text* на веб-странице и нажимается кнопка отправки текста. Затем переданный текст отображается на мониторе последовательного порта, подключенном к серверу. И наоборот, когда текст введен в монитор последовательного порта, за чем следует нажатие клавиши <*Enter*> компьютера или ноутбука, текст, полученный клиентом, отображается на веб-странице в поле *receive text*. Щелчок на поле *receive text* веб-страницы очищает текст, полученный с сервера. Как поле *transmit text*, так и поле *receive text* можно увеличить в размере, если потащить мышью правый нижний угол.

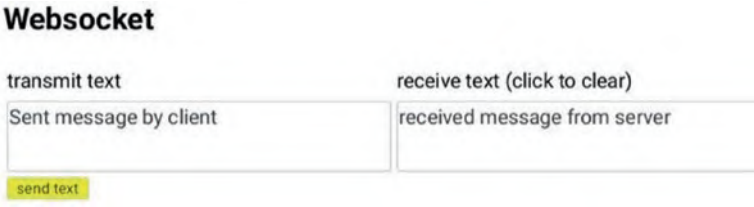


Рисунок 9-1. Веб-страница WebSocket

Библиотека *WebSocketsServer* Маркуса Саттлера (Markus Sattler) доступна в Arduino IDE в разделе *WebSockets*[46]. WebSocket подключен к порту 81, так как HTTP-порт по умолчанию равен 80, и функция `wsEvent` отображает полученное сообщение от клиента. Листинг 9-1 содержит скетч для примера передачи и приема текста. Инструкция `Serial.write()` посылает код ASCII побайтно, в то время как `Serial.print()` отображает принятый код ASCII. Функция `loop()` в листинге 9-1 по-прежнему включает инструкцию `server.handleClient()` по управле-

[46] См. пункт меню *Скетч* ➤ *Подключить библиотеку* (*Sketch* ➤ *Include library*). – *Прим. перев.*

нию HTTP-запросами, но когда текст передается сервером клиенту, инструкция `websocket.broadcastTXT (str.c_str(), str.length())` отправляет клиенту содержимое буфера последовательного порта. Текстовая строка преобразуется в строку в стиле C с завершающим нулевым символом с помощью команды `string.c_str()`. Функция `base` отправляет клиенту AJAX-код веб-страницы по умолчанию при первоначальной загрузке веб-страницы.

В этой главе сервером является микроконтроллер ESP8266 или ESP32. Инструкции библиотеки веб-сервера для микроконтроллера ESP8266 следующие:

```
#include <ESP8266WebServer.h>
ESP8266WebServer server
```

и для микроконтроллера ESP32:

```
#include <WebServer.h>
WebServer server(80);                 // requires a port number
```

Библиотека веб-сервера ссылается на библиотеку Wi-Fi, поэтому инструкции `#include <ESP8266WiFi.h>` или `#include <WiFi.h>` не требуются.

Листинг 9-1. Основной скетч WebSocket

```
#include <ESP8266WebServer.h>         // include web server library
ESP8266WebServer server;              // associate server with library
#include <WebSocketsServer.h>         // include WebSocket library
WebSocketsServer websocket = WebSoc ketsServer(81);
                                      // set WebSocket port 81
#include "buildpage.h"                // webpage AJAX code
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to Wi-Fi password
String str;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  WiFi.begin(ssid, password);         // connect and initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // display web server IP address
  server.begin();
  server.on("/", base);               // load default webpage
  websocket.begin();                  // initialise WebSocket
  websocket.onEvent(wsEvent);         // call wsEvent function
}                                     // on WebSocket event
void wsEvent(uint8_t num, WStype_t type, uint8_t * message,
size_t length)
{
  if(type == WStype_TEXT)             // when text received from client
  {                                   // display text on Serial Monitor
    for(int i=0; i<length; i++) Serial.write(message[i]);
    Serial.println();
  }
}
void loop()
{
  server.handleClient();              // manage HTTP requests
  websocket.loop();                   // handle WebSocket data
```

```
    if(Serial.available() > 0)
    {                                   // read text in Serial buffer
      str = Serial.readString();        // and send to client
      websocket.broadcastTXT(str.c_str(), str.length());
    }
}
void base()                             // function to return HTML code
{
  server.send (200, "text/html", page);
}
```

Реализация протокола WebSocket содержится в разделе JavaScript листинга 9-2, который включает AJAX-код веб-страницы. При загрузке веб-страницы создается таблица с заголовками для отображения принятого и переданного текста и вызывается функция `init` для открытия соединения WebSocket по адресу `ws://web server IP address:81/`. Когда клиент передает сообщение на сервер, команда `send(document.getElementById('txtext').value)` в функции `SendText` отправляет содержимое переменной `txText`, которая затем очищается. Когда клиент получает сообщение от сервера, содержимое сообщения сохраняется в переменной `rxText`, которая отображается на веб-странице.

Передаваемое сообщение `txText` содержится в HTML `textarea` внутри HTML `form` со свойством `action`, связанным с функцией JavaScript `SendText`. Переданное сообщение не отображается в поле HTML `input`, так как поле `input` не допускает разрыва текста. Если требуется поле HTML `input`, то функция `SendText` вызывается с помощью символа возврата каретки в конце отправленного сообщения, инструкцией `onkeydown='if(event.keyCode == 13) SendText()'`, где ASCII-код 13 соответствует возврату каретки. Полученное сообщение `rxText` вставляется в HTML `textarea`, которое позволяет обтекать текст.

Листинг 9-2. AJAX-код веб-страницы WebSocket

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head><title>ESP8266</title>
<style>
body {font-family:Arial}
td {vertical-align: top;}
textarea {font-family:Arial; width:300px; height:50px;}
input[type=submit] {background-color:yellow;}
</style></head>
<body id='initialise'>
<h2>WebSocket</h2>
<table><tr>
<td>transmit text</td>
<td>receive text (click to clear)</td>
</tr><tr>
<td><form action='javascript:sendText()'>
<textarea id='txText'></textarea><br>
<input type='submit' value="send text">
</form></td>
<td><textarea id='rxText'
onfocus='this.value=""'></textarea><br></td>
</tr></table>
```

```
<script>
var wskt;
document.getElementById('initialise').onload = function()
{init()};
function init()                          // open WebSocket
{
   wskt = new WebSocket('ws:             //' + window.location.hostname + ':81/');
   wskt.onmessage = function(rx)
   {                                     // client receive message
      var obj = rx.data;
      document.getElementById('rxText').value = obj;
   };
}
function sendText()                      // client transmit message
{
   wskt.send(document.getElementById('txText').value);
   document.getElementById('txText').value = "";
}
</script>
</body></html>
)";
```

Дистанционное управление и связь через WebSocket

Лазер, установленный на поворотно-наклонном кронштейне с сервоприводом, подключенным к плате ESP8266 или ESP32, управляется дистанционно путем перемещения ползунка на веб-странице клиента, при этом информация о положении ползунка передается на сервер на основе микроконтроллера ESP8266 или ESP32. Клиент получает информацию о положении сервопривода и состоянии лазера с сервера (см. рис. 9-2). Микроконтроллер перемещает сервопривод и включает или выключает лазер в соответствии с управляющей информацией, полученной от клиента. Информация на веб-странице постоянно обновляется, поскольку протокол WebSocket позволяет серверу передавать информацию клиенту без запроса от клиента.

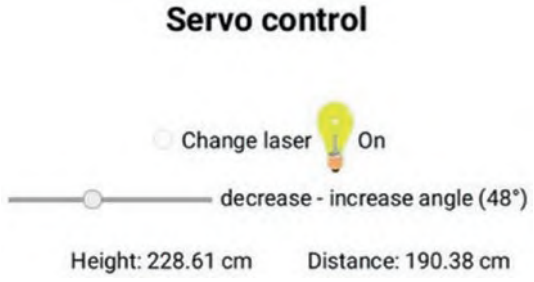


Рисунок 9-2. Веб-страница с указанием положения и состояния лазера

Применение дистанционно управляемого лазера, установленного на наклонном кронштейне, заключается в измерении расстояния по вертикали и горизонтали до точки, подсвеченной лазером. Расстояние по вертикали до точки определяется по углу наклона, а расстояние по горизонтали от объекта измеряется с помощью ультразвукового датчика расстояния, прикрепленного к передней части кронштейна наклона. На рис. 9-3 шарнир наклонного кронштейна находится на 5 см выше основания, расстояние между шарниром и передней частью ультразвукового датчика расстояния HC-SR04 составляет 8 см, а разница в высоте между шарниром и лазером 2 см. Расстояние (см) по вертикали от основания до точки, отмеченной лазером, равно `5 + (8 + d)tan(x) + 2/cos(x)`, где *x* – угол (в градусах, °) наклона кронштейна.

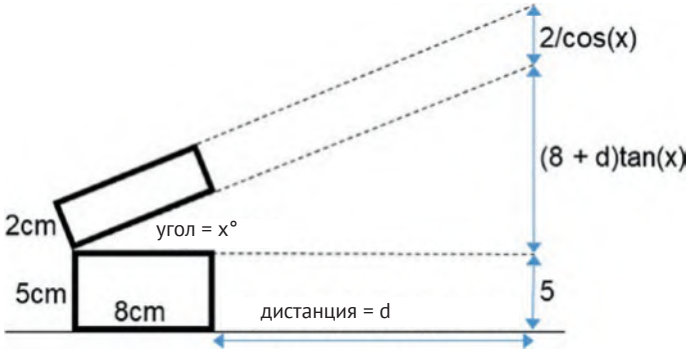


Рисунок 9-3. Вывод результатов измерения высоты и расстояния

Соединения для платы ESP8266 или ESP32 с ультразвуковым датчиком расстояния, лазерным модулем и сервоприводом приведены в табл. 9-1 и показаны на рис. 9-4. Ультразвуковой датчик расстояния HC-SR04 требует качественного питания 5 В, которое не обеспечивается USB-блоком питания 5 В. Сервоприводу требуется внешний источник питания, такой как батарея 5 В или батарея 9 В со стабилизатором напряжения L4940V5, поскольку двигатель может потреблять сотни миллиампер в течение нескольких миллисекунд[47], когда вращается ротор, что больше, чем могут выдать платы ESP8266 или ESP32 на выводах 5V или VIN. Батарея 9 В со стабилизатором напряжения L4940V5 и конденсаторами 100 нФ и 22 мкФ (см. рис. 9-3) питает ультразвуковой датчик расстояния и сервопривод. Лазер KY-008 работает на длине волны 650 нм из диапазона красного света (635–700 нм).

[47] Скорее, долей секунды и целых секунд, так как сервопривод потребляет не только в момент трогания с места, но и все время, пока происходит перемещение. – *Прим. перев.*

Таблица 9-1. Измерение высоты и расстояния с помощью микроконтроллеров ESP8266 и ESP32

Компонент	Подключено к	Также к
Ультразвуковой датчик расстояния VCC	Шина VCC	
Ультразвуковой датчик расстояния TRIG	ESP8266 D8 или ESP32 GPIO 13	
Ультразвуковой датчик расстояния ECHO	ESP8266 D7 или ESP32 GPIO 27	
Ультразвуковой датчик расстояния GND	Шина GND	
Лазерный модуль S	ESP8266 D6 или ESP32 GPIO 26	
Лазерный модуль	Шина GND	
Сигнальный вывод сервопривода (оранжевый или белый[48])	ESP8266 D5 или ESP32 GPIO 25	
Сервопривод VCC (красный)	Шина VCC	
Сервопривод GND (коричневый или черный)	Шина GND	
ESP8266 или ESP32 GND	Шина GND	
L4940V5 вход (input)	Плюс батареи 9 В	выв. 1 конденсатора 100 нФ
L4940V5 GND	Шина GND	
L4940V5 выход (output)	Шина VCC	плюс конденсатора 22 мкФ
Минус батареи 9 В	Шина GND	
Выв. 2 конденсатора 100 нФ	Шина GND	
Минус конденсатора 22 мкФ	Шина GND	

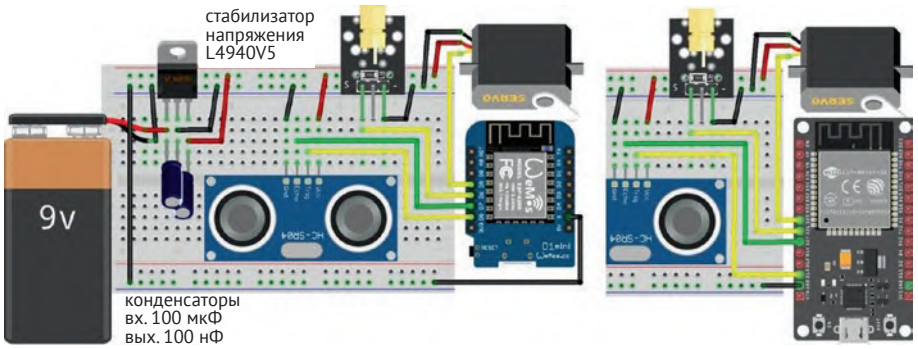


Рисунок 9-4. Измерение высоты и расстояния с помощью плат LOLIN (WeMos) D1 mini или ESP32 DEVKIT DOIT[49]

[48] Обычно вывод для подачи сигнала управления сервоприводом оранжевого или желтого цвета. – *Прим. перев.*

[49] На рисунке конденсатор 100 нФ ошибочно показан как электролитический, такой же, как и 22 мкФ. В стабилизаторах питания обычно ставят керамический многослойный (тип К10-17А или аналогичный). Керамические конденсаторы неполярные, и подключать относительно плюса питания их можно в любой ориентации. – *Прим. перев.*

Листинг 9-3 предназначен для микроконтроллера ESP8266 и библиотеки *NewPing8266*, которую можно загрузить с github.com/jshaw/NewPingESP8266. Для микроконтроллера ESP32 библиотеки *WebServer* и *NewPing* заменяют библиотеки *ESP8266WebServer* и *NewPing8266* библиотекой *NewPing*, доступной в Arduino IDE. Инструкция `ESP8266WebServer server` заменяется на `Webserver server (80)`. Для микроконтроллера ESP32 также требуется библиотека *Servo*, специфичная для ESP32, а не та *Servo*, которая встроена в Arduino IDE. Рекомендуется библиотека *ESP32Servo* Кевина Харрингтона (Kevin Harrington) и Джона К. Беннетта (John K. Bennett), также доступная в Arduino IDE. Инструкции микроконтроллера ESP8266 из библиотеки *Servo*

```
#include <Servo.h>    // include Servo library
servoFB.attach(FBpin) // initialise servo motor to FBpin
```

заменяются инструкциями библиотеки *ESP32Servo* для микроконтроллера ESP32:

```
#include <ESP32Servo.h>
servoFB.setPeriodHertz(F)            // define servo frequency (F)
servoFB.attach(FBpin, min, max)      // initialise servo motor to FBpin
```

Частота прямоугольного колебания *F* для управления сервоприводом задается командой `servofb.setPeriodHertz(F)` и обычно составляет 50 Гц. В инструкции `servoFB.attach(FBpin, min, max)` параметры `min` и `max` относятся к длительности импульса в микросекундах для перемещения сервопривода на 0° и 180° соответственно. Значения по умолчанию для параметров `min` и `max` составляют 1000 мкс и 2000 мкс, а для сервопривода Tower Pro SG90 – 500 мкс и 2500 мкс.

В следующих инструкциях изменений нет:

```
Servo servoFB                // associate servoFB with Servo lib
servoFB.writeMicroseconds(T) // move to position mapped to Tμs
servoFb.write(N)             // move to angle N°
```

В скетче из листинга 9-3 большинство инструкций относятся к включению библиотек, определению переменных, связанных с сервоприводом и ультразвуковым датчиком расстояния, установлению соединения Wi-Fi и загрузке веб-страницы по умолчанию путем вызова функции `base` для доступа к коду AJAX, содержащемуся в виде символьной строки `page`, расположенной на вкладке *buildpage*. Инструкции, связанные с *WebSocket*, включены в функцию *wsEvent*. Когда сервер получает от клиента сообщение, содержащее угол поворота сервопривода и состояние лазера, вызывается функция `wsEvent`, которая загружает полученное сообщение в строку. Из строки извлекаются угол сервопривода (servo angle) и состояние лазера (laser state) с помощью определения положения запятой, разделяющей два значения. Угол сервопривода заменяется на соответствующее количество микросекунд длительности импульса, необходимого для перемещения сервопривода на требуемый угол. Состояние лазера также обновляется. Расстояние по горизонтали измеряется ультразвуковым датчиком расстояния, а расстояние по вертикали рассчитывается на основе угла наклона сервопривода и расстояния по горизонтали. Два расстояния преобразуются в пары имя–значение в формате JSON с помощью функции `JsonConvert`, которая передает информацию клиенту. Расстояние в сантиметрах

между ультразвуковым датчиком расстояния и объектом составляет половину времени эха, измеряемого в микросекундах, умноженного на 0,0343, при условии что скорость звука составляет 343 м/с[50].

Листинг 9-3. Измерение высоты и расстояния

```
#include <ESP8266WebServer.h>         // include Webserver library
ESP8266WebServer server;              // associate server with library
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to Wi-Fi password
#include <WebSocketsServer.h>         // include Websocket library
WebSocketsServer websocket = WebSocketsServer(81);
                                      // set WebSocket port 81
#include "buildpage.h"                // webpage AJAX code
#include <Servo.h>
Servo servoFB;                        // associate servoFB with Servo lib
int FBpin = D7;                       // forward-backward servo pin
int laserPin = D8;
int minMicrosec = 450;                // minimum and maximum time
int maxMicrosec = 1150;               // for servo motor pulse length
#include <NewPingESP8266.h>           // include NewPing library
int trigPin = D5;                     // ultrasonic trigger and echo pins
int echoPin = D6;
float maxdist = 300;                  // ultrasound maximum distance
NewPingESP8266 sonar(trigPin, echoPin, maxdist);
float distance, height, temp, angle;
int microsec, laser, comma;
String text[2];                       // strings in JSON text
String str, json;
unsigned long timer = 0;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  WiFi.begin(ssid, password);         // connect and initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // display web server IP address
  server.begin();
  server.on("/", base);               // load default webpage
  websocket.begin();                  // initialise WebSocket
  websocket.onEvent(wsEvent);         // wsEvent on WebSocket event
  servoFB.attach(FBpin);              // initialise servo motor
  servoFB.writeMicroseconds(minMic rosec);
                                      // and move to initial position
  pinMode(laserPin, OUTPUT);          // define laser pin as output
}
                    // function called when message received from client
void wsEvent(uint8_t n, WStype_t type, uint8_t * message,
size_t length)
{
  if(type == WStype_TEXT)
  {
    str = "";                         // convert message to string
    for (int i=0; i<length; i++) str = str + char(message[i]);
```

[50] См. также сноску __ на стр. 132. – *Прим. перев.*

```
      comma = str.indexOf(",");               // location of comma
      text[0] = str.substring(0, comma);      // extract substrings
      text[1] = str.substring(comma+1);
      angle = text[0].toFloat();              // parse servo angle
      microsec = map(angle,20,90,minMicros ec,maxMicrosec);
                                              // map angle to µs
      servoFB.writeMicroseconds(microsec);    // move servo to angle
      delay(10);                              // time to move servo
      laser = text[1].toInt();                // parse laser state
      digitalWrite(laserPin, laser);          // turn on or off laser
      distance = (sonar.ping_median(10)/2. 0)*0.0343;
                                              // get horizontal distance
      angle = angle*PI/180.0;                 // convert angle to radians
      height = 5.0+(distance+8.0)*tan(angl e)+2.0/cos(angle);
                                              // vertical distance
      JsonConvert(height, distance);          // convert to JSON format
      websocket.broadcastTXT(json.c_str(), json.length());
   }                                          // send JSON text
}
                     // function converts variables to JSON name/value pairs
String JsonConvert(float val1, float val2)
{                                             // start with open bracket
   json = "{\"var1\": \"" + String(val1) + "\",";
                                              // partition with comma
   json += " \"var2\": \"" + String(val2) + "\"}";
                                              // end with close bracket
   return json;
}
void base()                                   // function to return HTML code
{
   server.send(200, "text/html", page);
}
void loop()
{
   server.handleClient();                     // manage HTTP requests
   websocket.loop();                          // handle WebSocket data
}
```

WebSocket и AJAX

На веб-странице (см. рис. 9-5) нажатие кнопки *Change laser* (*Изменить лазер*) включает или выключает лазер, и отображается соответствующее изображение. Перемещение ползунка изменяет угол наклона кронштейна, при этом угол отображается на веб-странице. Клиент передает угол сервопривода и состояние лазера на сервер, и сервер отвечает, отправляя клиенту измеренное горизонтальное расстояние и рассчитанную высоту, которые клиент отображает на веб-странице.

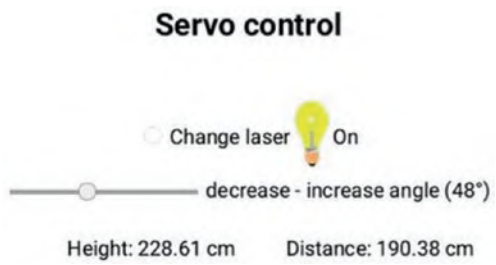


Рисунок 9-5. Измерение высоты и расстояния

AJAX-код для веб-страницы (см. листинг 9-4) включен в символьную строку `page`, заключенную между символами R"(и)", с переменными, обозначенными одиночными апострофами '. В HTML-коде раздел `<head>` включает две инструкции `<meta>`, которые необходимы для форматирования текста в логе консоли, в котором отображаются данные, полученные и переданные клиентом, например `console.log(FBVal)`. Раздел `<style>` центрирует текст на веб-странице и определяет ползунок (slider), размер изображения и высоту строки таблицы. Содержимое веб-страницы форматируется в виде таблицы с первой строкой, охватывающей два столбца. Она содержит кнопку *Change laser*, изображение лампы и текст `laserId`, описывающий состояние лазера *On* или *Off*.

Вторая строка таблицы содержит ползунок для выбора угла наклона в диапазоне от 20° до 90°, за которым следуют текст и значение угла. Ползунок задается с помощью инструкции

```
  <input autocomplete='on' type='range' min='20' max='90' value='20'
class='slider' id='FBSlider' oninput='sendFB()'>
```

которая устанавливает `autocomplete` (автозаполнение) в положение *On* с начальным положением ползунка, заданным в `value`, тогда как установка `autocomplete` в положение *Off* поместит ползунок в среднее положение по умолчанию. Инструкция связывает ползунок с функцией `sendFB`, которая вызывается при перемещении ползунка. В третьей строке таблицы отображаются рассчитанная высота и измеренное расстояние.

Когда веб-страница загружается, вызывается функция `init` для подключения к WebSocket по адресу `ws://web server IP address:81/`. Когда клиент получает сообщение, его содержимое сохраняется в переменной `rx.data`, из которой извлекаются переменные `vertical` и `horizontal` для отображения на веб-странице. Сообщение передается клиентом с помощью функции `sendFB`, которая объединяет установленный ползунком угол с состоянием лазера. При нажатии кнопки *Change laser* вызывается функция `changeLaser`, которая изменяет значение состояния лазера, `laserVal`, обновляет состояние лазера в коде веб-страницы и переключает изображение лампы, загружаемое с сайта www.w3schools.com. Местоположение изображения для загрузки с веб-сайта определяется щелчком правой кнопки мыши по изображению и выбором *View Image Info* (Просмотреть информацию об изображении) или *Copy Image Location* (Скопировать местоположение изображения) и вставкой местоположения изображения в код AJAX.

При изменении положения ползунка или нажатии кнопки *Change laser* угол, выбранный ползунком, и состояние лазера передаются клиентом на сервер инструкцией `wskt.send(FBVal +','+ laserVal)`. Для целей тестирования скетча инструкции `console.log(obj.var1)` и `console.log(FBVal)` отображают в логе значения установленного клиентом вертикального расстояния и переданного угла соответственно.

Листинг 9-4. AJAX-код для измерения высоты и расстояния

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head>
<meta name='viewport' content='width=device-width,
initial-scale=
1.0'>
<meta charset='UTF-8'>
<title>ESP8266</title>
<style>
html {text-align: center}
.slider {-webkit-appearance: none; height: 2px;
background: DarkGrey}
img {width:25px; height:50px}
td {height:50px}
</style>
<title>WebSocket</title>
</head>
<!-- initiate WebSocket when webpage loaded-->
<body id='initialise'>
<h2>Servo control</h2>
<table align='center'><tr>
<td colspan='2'><input type='radio' id='r1'
onclick='changeLaser()'> Change laser
<img id='bulb' src='
https://www.w3schools.com/jsref/
pic_bulboff.gif'>
<span id='laserId'>Off</span></td>
</tr>
<tr>
<!--autocomplete='off': returns slider to default mid-point position-->
<td><input autocomplete='on' type='range' min='20' max='90'
value='20' class='slider' id='FBSlider' oninput='sendFB()'></td>
<td><label id='FBId'>decrease - increase angle (20&deg)
</label></td>
</tr>
<tr>
<td style='width:200px'>Height: <span id='vertical'>0</span>
cm</td>
<td>Distance: <span id='horizontal'>0</span> cm</td>
</tr></table>
<script>
var FBVal = 20;
var laserVal = 0;
document.getElementById('initialise').onload = function()
{init()};
```

```
function init()
{
   wskt = new WebSocket('ws:          //' + window.location.hostname +':81/');
   wskt.onmessage = function(rx)
   {
      var obj = JSON.parse(rx.data);
      console.log(obj.var1);
      console.log(obj.var2);
      document.getElementById('vertical').innerHTML = obj.var1;
      document.getElementById('horizontal').innerHTML = obj.var2;
   };
}
function sendFB()
{
   FBVal = document.getElementById('FBSlider').value;
   document.getElementById('FBId').innerHTML = 'decrease -
   increase angle ('+FBVal.toString() + '&deg)';
   wskt.send(FBVal +','+ laserVal);
   console.log(FBVal);
   console.log(laserVal);
}
function changeLaser()
{
   laserVal = 1 - laserVal;
   if(laserVal == 1) {laserTag = 'On';}
   else {laserTag = 'Off';}
   document.getElementById('laserId').innerHTML = laserTag;
   document.getElementById('r1').checked=false;
   wskt.send(FBVal +',' + laserVal);
   var image = document.getElementById('bulb');
   if (image.src.match('bulboff')) {image.src =
                'https://www.w3schools.com/js/pic_bulbon.gif';}
   else {image.src = 'https://www.w3schools.com/js/
   pic_bulboff.gif';}
}
</script>
</body></html>
)";
```

Команда `servo.write(N)` библиотеки *Servo* перемещает сервопривод на угол *N*°. Альтернативной инструкцией является `servo.writeMicroseconds(microsec)`, где количество микросекунд определяет длину импульса. Стандартный сервопривод перемещается на угол 0° или 180°, когда длительность прямоугольного импульса составляет 500 мкс или 2500 мкс, в то время как длительность импульса для серводвигателей другого стандарта составляет 1000 мкс или 2000 мкс. Для двух стандартных случаев формула для количества микросекунд, необходимых для перемещения на заданный угол, равна 500 + *angle* × 200/18 и 1000 + *angle* × 100/18. Сервопривод, используемый в этой главе, был откалиброван в соответствии со скетчем в листинге 9-5 путем ввода различных значений микросекунд через монитор последовательного порта и измерения угла наклона сервопривода. Например, если для позиционирования сервопривода под углом 45° и 90° требовалось 700 мкс и 1150 мкс, получим уравнение *microseconds* = 250 + *angle* × 10, или, что то же самое, 250 + *angle* × 180/18.

Листинг 9-5. Калибровка сервопривода

```
#include <Servo.h>            // include Servo library
Servo servoFB;                // associate servoFB with Servo library
int FBpin = D7;               // servo pin
int microsec;
void setup()
{
  Serial.begin(115200);       // Serial Monitor baud rate
  servoFB.attach(FBpin);      // initialise servo motor
}
void loop()
{
  if(Serial.available() > 0) // text entered in Serial Monitor
  {                           // parse Serial buffer to integer
    microsec = Serial.parseInt();
    servoFB.writeMicroseconds(micros ec);
  }                           // move servo motor
}
```

Доступ к изображениям, времени и показаниям датчиков через интернет

Другим примером протокола WebSocket является графическое отображение данных на веб-странице с заданными свойствами графика, такими как минимальные и максимальные значения по оси Y и временной интервал по оси X, изменяемые пользователем в режиме реального времени. Изменение масштаба по оси Y графика влияет только на клиента, вносящего изменения, поэтому ось Y графика зависит от конкретного клиента. Напротив, когда временной интервал по оси X передается на сервер, графическое отображение обновляется одновременно для всех клиентов, подключенных к серверу. На рис. 9-6 приведена веб-страница, отображающая данные датчика температуры в режиме реального времени с возможностью изменения минимальных и максимальных значений оси Y и интервала времени для обновления данных. Отображаются текущая дата и время, а также некоторые изображения, загруженные из интернета.

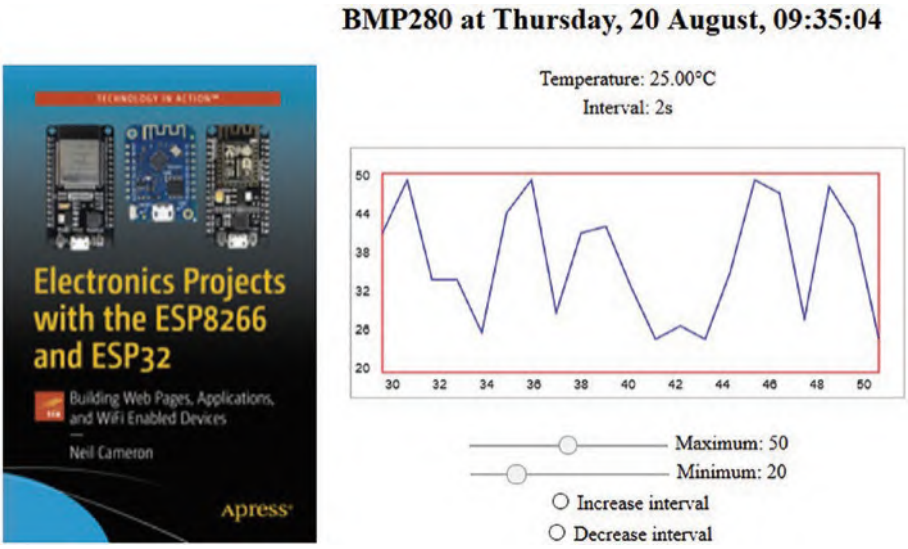


Рисунок 9-6. График показаний датчика в реальном времени

Датчик BMP280 измеряет температуру и давление, передает данные по I2C или SPI и работает при напряжении 3,3 В. I2C-адрес датчика BMP280 равен 0x76. Соединения между датчиком BMP280 и платой ESP8266 или ESP32 показаны на рис. 9-7 и приведены в табл. 9-2.

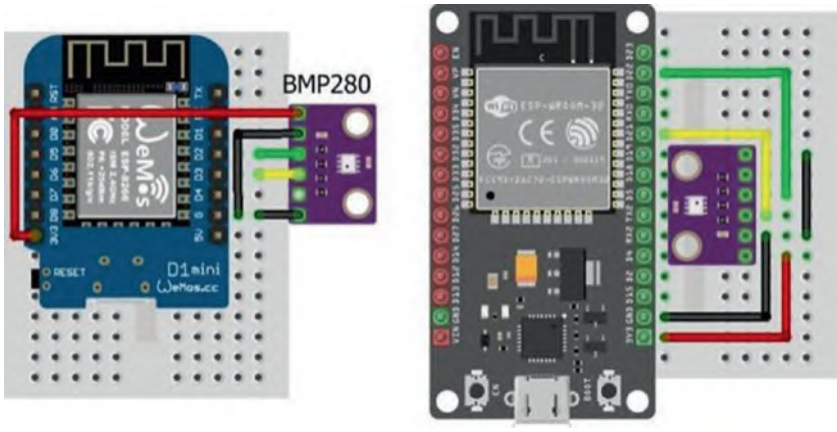


Рисунок 9-7. BMP280 с платами LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT51

[51] На рис. 9-7 ошибочно показано соединение вывода SDO с GND. При подключении по интерфейсу I2C выход SDO датчика BMP280 должен оставаться свободным. – *Прим. перев.*

Таблица 9-2. BMP280 с микроконтроллерами ESP8266 и ESP32

Компонент	ESP8266	ESP32
BMP280 VCC	3V3	3V3
BMP280 GND	GND	GND
BMP280 SDI	D2	GPIO 21
BMP280 SCK	D1	GPIO 22
BMP280 SDO	Не подключен	Не подключен

Первый раздел и функция `setup` скетча в листинге 9-6 по существу такие же, как и в листинге 9-3. Библиотека *Adafruit_sensor*, находящаяся в разделе *Adafruit_Unified_Sensor* в Arduino IDE, и библиотека *Adafruit_BMP280* устанавливаются для датчика BMP280, который инициализируется с помощью его адреса I2C в функции `setup`. URL-адрес `tempUrl` задается функцией `tempFunct`, которая подключена к библиотеке *Ticker* для синхронизации обновлений веб-страницы.

Когда сервер получает от клиента сообщение, содержащее временной интервал, вызывается функция WebSocket `wsEvent`, которая загружает полученное сообщение для парсинга. В листинге 9-6 сообщение содержит только переменную временного интервала `interval`, а строка сообщения преобразуется в целое число с помощью функции `atoi()` C++. Функция `tempFunct` обновляет переменную интервала для библиотеки *Ticker* для управления временем вызовов функции `tempFunct` и получает показания температуры от датчика BMP280. Преобразуются как температура, так и временной интервал для создания пар имя–значение в формате JSON с помощью функции `JsonConvert` и передаются клиенту. Если к серверу подключено несколько клиентов и один клиент изменяет временной интервал, то также обновляется временной интервал, отображаемый каждым клиентом. Функции `base` и `loop` в листинге 9-6 идентичны функциям в листинге 9-3. При тестировании скетча команда `JsonConvert(bmp.readTemperature(), interval)` заменяется на `JsonConvert(random(20, 50)*1.0, interval)` для генерации различных вариантов значений.

Листинги 9-1, 9-3 и 9-6 иллюстрируют три примера обработки клиентского сообщения WebSocket. Сообщение отображается на мониторе последовательного порта с помощью команды `Serial.write(message[i])`, преобразуется в строку командой `str = str + char(message[i])` или преобразуется в целое число с помощью `interval = atoi((char *) &message[0])`. Эквивалентом C++ функции `atoi` для действительного числа является функция `atof`.

Листинг 9-6. График показаний датчиков в реальном времени

```
#include <ESP8266WebServer.h>         // include WebServer library
ESP8266WebServer server;              // associate server with library
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to Wi-Fi password
#include <WebSocketsServer.h>         // include Websocket library
WebSocketsServer websocket = WebSo cketsServer(81);
                                      // set WebSocket port 81
#include "buildpage.h"                // webpage AJAX code
```

```
String json;
#include <Ticker.h>                    // include Ticker library
Ticker timer;                          // associate timer with Ticker lib
int interval = 1;
int oldInterval = 1;
#include <Adafruit_Sensor.h>           // include Adafruit Sensor
#include <Adafruit_BMP280.h>           // and BMP280 libraries
Adafruit_BMP280 bmp;                   // associate bmp with BMP280
int BMPaddress = 0x76;                 // BMP280 I2C address
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  WiFi.begin(ssid, password);          // connect and initialise Wi-Fi
  while(WiFi.status()!= WL_CONNECTED ) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());      // display web server IP address
  server.begin();
  server.on("/",base);                 // load default webpage
  server.on("/tempUrl", tempFunct) ;
                                       // map URL to tempFunct
  websocket.begin();                   // initialise WebSocket
  websocket.onEvent(wsEvent);          // wsEvent on WebSocket event
  bmp.begin(BMPaddress);               // initialise BMP280 sensor
  timer.attach(interval, tempFunct );
}                                      // attach timer to tempFunct
                     // function called when message received from client
void wsEvent(uint8_t n, WStype_t type, uint8_t * message,
size_t length)
{                                      // convert message to integer
  if(type == WStype_TEXT) interval = atoi((char *)&message[0]);
}
void tempFunct()
                     // function to transmit temperature and update interval
{                                      // convert to JSON format
  JsonConvert(bmp.readTemperature( ), interval);
  websocket.broadcastTXT(json.c_st r(), json.length());
                                       // send JSON text
  if(interval != oldInterval)
  {
    timer.detach();
    timer.attach(interval, tempFun ct);
    oldInterval = interval;            // update timer interval
  }
}
             // function converts variables to JSON name/value pairs
String JsonConvert(float val1, int val2)
{            // start with open bracket
  json = "{\"var1\": \"" + String (val1) + "\",";
                                       // partition with comma
  json += " \"var2\": \"" + String (val2) + "\"}";
                                       // end with close bracket
  return json;
}
void base()                            // function to return HTML code
{
  server.send(200, "text/html", page);
}
```

```
void loop()
{
  server.handleClient();      // manage HTTP requests
  websocket.loop();           // handle WebSocket data
}
```

AJAX-код для веб-страницы приведен в листинге 9-7. Содержимое веб-страницы отформатировано в виде таблицы со строкой заголовка, содержащей текущую дату и время браузера, с двумя столбцами, состоящими из семи строк. Первый столбец, который охватывает все семь строк, согласно инструкции `<td rowspan='7'>`, содержит изображение, которое загружается при инициализации веб-страницы. Первые три строки во втором столбце содержат обновленные значения температуры и временного интервала, передаваемые сервером, а также основу `<canvas>` для графика. Два ползунка для управления максимальным и минимальным значениями оси Y графика, а также кнопки для увеличения или уменьшения временного интервала по оси X управляются функциями `setMaxy`, `setMiny`, `sendadd` и `sendsub` соответственно.

Функция `graph` использует инструкцию `canvas.getContext()` для доступа к функциям рисования, подробная информация доступна по адресу www.w3schools.com/tags/ref_canvas.asp. Инструкции `clearRect` и `strokeRect` очищают прямоугольное пространство (445×200 пикселей) с координатами верхнего левого угла (0, 0), в котором очерчен прямоугольник графика (400×160 пикселей), начиная с позиции (25, 20). Расположение шести меток по оси Y определяется инструкцией

```
ctx.fillText(Math.round(maxy-i*(maxy-miny)/5), 3, 25+31*i)
```

которая вычисляет положения меток по максимальным и минимальным значениям оси Y. Метки по оси Y расположены рядами на расстоянии 31 пикселя друг от друга, начиная с позиции пикселя (3, 25). 11 меток по оси X постоянно обновляются на основе общего количества значений данных `Ndata` с помощью инструкции

```
ctx.fillText(String(Ndata+i-20), 27+19*i, 193)
```

Метки по оси X расположены в столбцах на расстоянии 19 пикселей друг от друга, начиная с позиции пикселя (27, 193). Инструкции по рисованию линии, соединяющей точки данных вместе, – это `beginPath` с начальной точкой данных `moveTo(x, y)` и последующими точками данных `lineTo(x,y)`, за которыми следует `stroke`[52]. График строится сериями по 21 точке, которые постоянно обновляются, причем 21-я точка является самым последним значением, согласно инструкциям:

```
Ndata++;                    // increment the number of data points
if(Ndata>maxVal) datay.shift( );
                            // remove the first element of datay[ ]
datay.push(obj.var1);       // add new data point to end of datay[ ]
```

Подробная информация о командах JavaScript для обработки массивов доступна по адресу www.w3schools.com/jsref/jsref_obj_array.asp.

[52] stroke() – команда JavaScript, означающая, что рисование закончено. – *Прим. перев.*

Максимальное и минимальное значения оси Y графика являются соответствующими значениями для ползунка, а функции `setMaxy` и `setMiny` преобразуют значение ползунка в строку для отображения на веб-странице справа от ползунка. Функции `sendadd` и `sendsub` увеличивают и уменьшают интервал времени между обновлениями веб-страницы на одну секунду, при этом обновленное значение отправляется на сервер.

Когда веб-страница загружается первый раз, функция `init` открывает подключение к WebSocket по адресу `ws://IP-адрес веб-сервера:81/`. Когда клиент получает сообщение, его содержимое, хранящееся в переменной `rx.data`, преобразуется в переменные `temp` и `interval` для отображения на веб-странице температуры и интервала времени между показаниями. Текущее время браузера будет получено и отформатировано с помощью инструкций:

```
var dt = new Date();
var tm = dt.toLocaleTimeString
     ('en-GB', {weekday: 'long', day: '2-digit', month: 'long'});
document.getElementById('timeNow').innerHTML = tm;
```

Результирующий формат времени: *Tuesday, 16 June, 10:11:47*, но если `innerHTML = dt`, то формат отображения будет *Tue Jun 16 2020 10:11:47 GMT+0100* (летнее британское время). Если требуется время в формате *чч:мм:сс*, то переменная `tm` определяется как `var tm = dt.toLocaleTimeString ('en-GB')`. Подробная информация о форматировании даты доступна по адресу www.w3schools.com/Jsref/jsref_obj_date.asp.

Листинг 9-7. AJAX-код графика показаний датчиков в реальном времени

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head>
<meta name='viewport' content='width=device-width, initial-scale=1.0'>
<meta charset='UTF-8'>
<!-- padding top right bottom left-->
<style>
html {text-align: center}
.slider {-webkit-appearance: none; height: 2px;
background: DarkGrey}
td {padding: 0px 0px 0px 25px}
img {width:253px; height:384px}
</style>
<title>ESP8266</title>
</head>
<body onload='javascript:init()'>
<table>
<tr><th></th><th><h2>BMP280 at <span id = 'timeNow'></span>
</h2></th></tr>
<tr>
<td rowspan='7'>
<img src=
'https://images.springer.com/sgw/books/medium/9781484263358.jpg'
     alt='book'></td>
<td>Temperature: <span id = 'temp'>0</span>&degC</td>
</tr>
<tr><td>Interval: <span id='interval'>1</span>s</td></tr>
<tr><td>
```

```
<canvas id = 'myCanvas' width = '445' height = '200'
style = 'border:1px solid DarkGrey'>
Your browser does not support the canvas element.
</canvas>
</td></tr>
<tr><td>
<input autocomplete='off' type='range' min='1' max='100'
value='25' class='slider' id='maxySlider' oninput='setMaxy()' >
<label id='maxyId'>Maximum: 25</label>
</td></tr>
<tr><td>
<input autocomplete='off' type='range' min='0' max='100'
value='15' class='slider' id='minySlider' oninput='setMiny()' >
<label id='minyId'>Minimum: 15</label>
</td></tr>
<tr><td>
<input type='radio' id='r1' oninput='sendadd()'> Increase
interval
</td></tr>
<tr><td>
<input type='radio' id='r2' oninput='sendsub()'> Decrease
interval
</td></tr>
</table>
<script>
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
ctx.strokeStyle = 'red';
ctx.strokeRect(25, 20, 400, 160);
ctx.lineWidth = 1;
var y = 0;
var miny = 15;
var maxy = 25;
var timeval = 1;
var datay = [0];
var Ndata = 0;
var maxVal = 20;
var dt = 0;
var tm = 0;
function init()
{
   webSocket = new WebSocket('ws://' + window.location.hostname + ':81/');
   webSocket.onmessage = function(rx)
   {
      var obj = JSON.parse(rx.data);
      document.getElementById('temp').innerHTML = obj.var1;
      document.getElementById('interval').innerHTML = obj.var2;
      Ndata++;
      if(Ndata>maxVal) datay.shift();
      datay.push(obj.var1);
      dt = new Date();
      tm = dt.toLocaleTimeString
      ('en-GB', {weekday: 'long', day: '2-digit', month:'long'});
      document.getElementById('timeNow').innerHTML = tm;
      graph()
   };
}
```

```
function graph()
{
   ctx.clearRect(0, 0, 445, 200);
   ctx.strokeStyle = 'red';
   ctx.strokeRect(25, 20, 400, 160);
   for (i=0; i<6; i++)
   ctx.fillText(Math.round(maxy-i*(maxy-miny)/5), 3, 25+31*i);
   if(Ndata<21) {for (i=0; i<21; i=i+2)
   ctx.fillText(String(i), 27+19*i, 193);}
   if(Ndata>20) {for (i=0; i<21; i=i+2)
   ctx.fillText(String(Ndata+i-20), 27+19*i, 193);}
   ctx.beginPath();
   y = 20+160*(maxy-datay[0])/(maxy-miny);
   if(y<20) y=20;
   if(y>180) y=180;
   ctx.moveTo(25, y);
   for(i=1; i<21; i++)
   {
      y = 20+160*(maxy-datay[i])/(maxy-miny);
      if(y<20) y=20;
      if(y>180) y=180;
      ctx.strokeStyle = 'blue';
      ctx.lineTo(25+20*i, y);
   }
   ctx.stroke();
}
function setMaxy()
{
   maxy = document.getElementById('maxySlider').value;
   document.getElementById('maxyId').innerHTML =
     'Maximum: ' + maxy.toString();
   }
   function setMiny()
   {
   miny = document.getElementById('minySlider').value;
   document.getElementById('minyId').innerHTML =
   'Minimum: ' + miny.toString();
}
function sendadd()
{
   timeval = parseInt(document.getElementById('interval').
   innerHTML) + 1;
   document.getElementById('interval').innerHTML = timeval;
   document.getElementById('r1').checked=false;
   webSocket.send(timeval);
}
function sendsub()
{
   timeval = parseInt(document.getElementById('interval').
   innerHTML) - 1;
   if(timeval<1) timeval = 1;
   document.getElementById('interval').innerHTML = timeval;
   document.getElementById('r2').checked=false;
   webSocket.send(timeval);
}
</script>
</body></html>
)";
```

Итоги

Три примера иллюстрируют преимущество использования протокола WebSocket для двустороннего обмена данными в режиме реального времени между клиентом и веб-сервером. В первом примере веб-страница позволяла клиенту отправлять текст на сервер и получать текст с сервера в режиме реального времени. Во втором примере клиент использовал ползунок веб-страницы и кнопку для дистанционного управления положением и состоянием лазера, прикрепленного к сервоприводу на наклонном кронштейне. Сервер отвечал высотой и расстоянием до объекта, идентифицированного лазером, с горизонтальным расстоянием, измеренным ультразвуковым датчиком. Третий пример продемонстрировал графическое отображение в реальном времени на веб-странице данных датчика температуры, передаваемых сервером, со свойствами графика и интервалом между измерениями температуры, управляемыми удаленно клиентом с помощью ползунка и кнопок на веб-странице.

Перечень компонентов

- Микроконтроллер ESP8266: LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: DEVKIT DOIT или NodeMCU
- Датчик температуры: модуль BMP280
- Ультразвуковой датчик расстояния: HC-SR04
- Лазерный модуль: KY-008
- Сервопривод: SG90
- Поворотно-наклонный сервокронштейн
- Конденсаторы: 100 нФ, 22 мкФ
- Стабилизатор напряжения: L4940V5
- Батарея: 9 В

Глава 10

Создаем мобильное приложение

Мобильное приложение (*app*, сокращение от *application*[53]) представляет собой компьютерную программу для мобильных устройств, таких как Android-планшет или мобильный телефон. Приложения доступны для мобильных игр, предоставления информации, определения местоположения и маршрута, воспроизведения музыки и управления внешними устройствами. *MIT App Inventor*[54] предоставляет возможность создавать собственное приложение, а не использовать приложение, загруженное из *Google Play Store*. После завершения этапа разработки приложение сразу же доступно для загрузки на планшет или мобильный телефон Android.

Веб-сайт разработки приложений *MIT App Inventor* – ai2.appinventor.mit.edu, доступ к которому осуществляется нажатием кнопки *Create Apps* на appinventor.mit.edu. При создании приложения возможность одновременного отображения разрабатываемого приложения на планшете или мобильном телефоне Android предоставляется приложением *MIT App Inventor Companion* (*MIT AI2 Companion*), которое загружается из *Google Play Store*. Например, эффект изменения положения кнопки или изображения на экране в *MIT App Inventor* мгновенно реализуется в приложении *Companion*. Оба компьютера, на которых приложение разрабатывается, и планшет или мобильный телефон Android, на котором размещено приложение *MIT App Inventor Companion*, должны быть подключены к той же локальной сети Wi-Fi. Более подробная информация доступна по адресу appinventor.mit.edu/explore/ai2/setup-device-wifi.html.

---

[53] Уточним, что в английском языке сокращение *app*, несомненно, хорошо знакомое читателю хотя бы по названию Whatsapp, означает прикладную программу (приложение) именно для мобильных устройств, в отличие от полного термина *application*, который означает прикладные программы вообще, для любых операционных систем. Разница между *app* и *application* не только в назначении, но и в общем построении, структуре и методах создания и исполнения программного кода. – *Прим. перев.*

[54] MIT App Inventor – среда визуальной разработки Android-приложений. Первоначально разработана в Google Labs в 2011 году, после закрытия этого подразделения была передана Массачусетскому технологическому институту (MIT). – *Прим. перев.*

После входа на веб-сайт *MIT App Inventor* и выбора пункта *Start new project* отображается окно конструктора приложений (см. рис. 10-1). В левой части окна конструктора из палитры *User Interface* можно добавить кнопки, изображения, метки и ползунки на экран приложения с помощью перетаскивания. В правой части окна конструктора в разделе *Properties* (Свойства) изменяются свойства выбранных элементов, такие как название приложения, цвет фона экрана, ориентация и заголовок. Экран приложения разделяется на отдельные области с помощью вставки горизонтальных областей *HorizontalArrangements* из палитры *Layout* в левой части окна конструктора. Для программирования приложений используется визуальный язык программирования на основе блоков, напоминающий *Scratch*, о котором рассказывается на scratch.mit.edu/about. Для каждого элемента рекомендуется выбирать читаемое имя. Например, кнопка для управления светодиодом называется *LEDbutton*, а размер шрифта, высота и ширина кнопки и текст определяются в разделе *Properties* в правой части окна конструктора. Изображения загружаются в палитру *Media* в правой нижней части окна конструктора. Учебные пособия MIT App Inventor доступны по адресу appinventor.mit.edu/explore/ai2/tutorials.html.

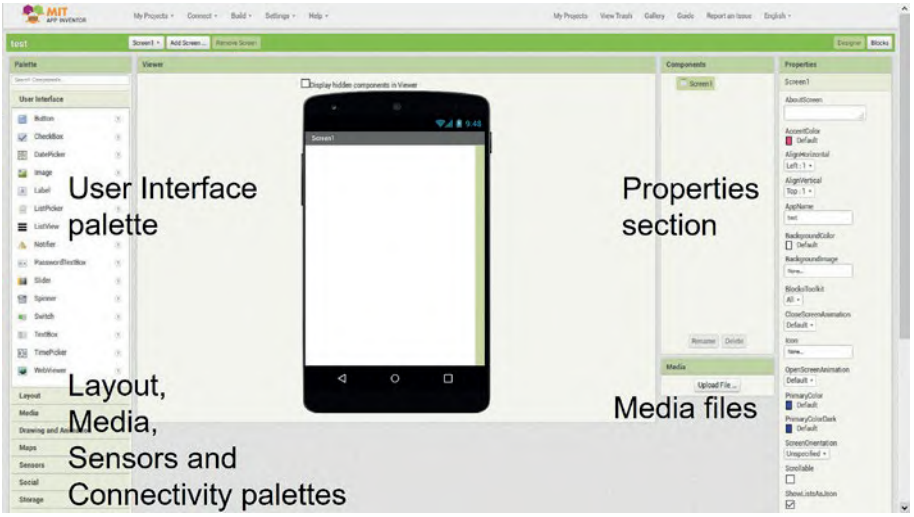


Рисунок 10-1. Стартовое окно MIT App Inventor

Приложение для управления с обратной связью

В примере приложение управляет яркостью светодиода, подключенного к микроконтроллеру ESP8266 или ESP32, с обратной связью, обеспечиваемой фоторезистором (LDR) (см. рис. 10-2). Приложение включает в себя кнопку для включения или выключения светодиода и показывает изображение, соответствующее состоянию светодиода. Когда ползунок в приложении перемещается при включенном светодиоде, приложение передает сигнал на микроконтроллер для изменения яркости светодиода. Каждые две секунды микроконтроллер передает показания фотодиода в приложение. Чем ярче

горит светодиод, тем выше значение показаний фотодиода. Управление яркостью светодиода – это только пример приложения. Можно, например, контролировать состояние и скорость удаленно расположенного электродвигателя через приложение, которое будет указывать, работает ли двигатель и с какой скоростью, без того, чтобы пользователь был вынужден наблюдать непосредственно за двигателем.

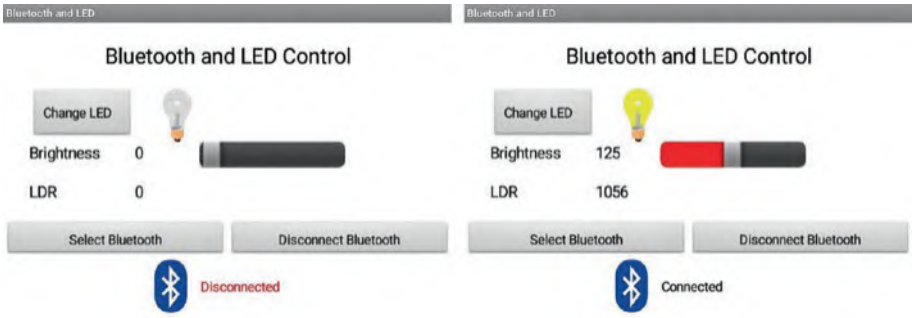


Рисунок 10-2. Приложение для управления светодиодом через Bluetooth

Приложение использует Bluetooth для связи с микроконтроллером ESP8266 или ESP32. Платы ESP32 имеют функцию Bluetooth, но платы ESP8266 должны быть подключены к отдельному модулю Bluetooth HC-05. Модуль Bluetooth HC-05 питается от 3,6 В до 6 В и подключается к выводу 5V платы ESP8266. Модуль Bluetooth HC-05 передает (TXD) и принимает (RXD) последовательные данные при напряжении 3,3 В, что является рабочим напряжением микроконтроллера ESP8266, и напрямую подключен к выводам RX и TX платы ESP8266. Микроконтроллер ESP8266 использует последовательный порт также для загрузки скомпилированного скетча, поэтому во время загрузки RX-вывод платы ESP8266 должен быть отключен, или загрузка завершится ошибкой. Соединения для обеих плат ESP8266 и ESP32 показаны на рис. 10-3 и приведены в табл. 10-1.

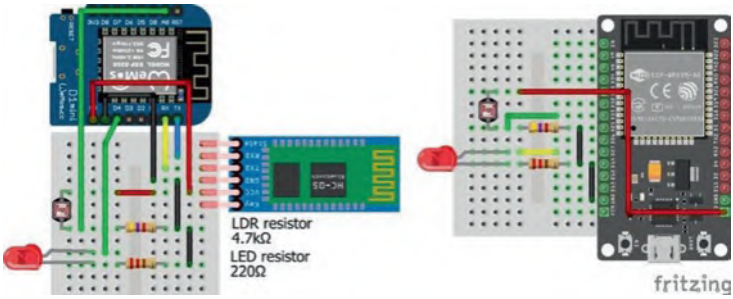


Рисунок 10-3. Bluetooth-управление светодиодом с платами ESP8266 и ESP32

Таблица 10-1. Bluetooth-управление светодиодом с платами ESP8266 и ESP32

Компонент	Подключено к	ESP8266	ESP32
Bluetooth HC-05 VCC		5V	
Bluetooth HC-05 GND		GND	
Bluetooth HC-05 TXD		RX	
Bluetooth HC-05 RXD		TX	
LED длинный вывод		D4	GPIO 27
LED короткий вывод	Резистор 220 Ом	GND	GND
LDR нижний вывод		A0	GPIO 33
LDR нижний вывод	Резистор 4.7 кОм	GND	GND
LDR верхний вывод		5V	3V3

Скетч в левой части табл. 10-2 предназначен для микроконтроллера ESP8266. Когда микроконтроллер ESP8266 получает из приложения сообщение, состоящее из строки "*C*", микроконтроллер ESP8266 изменяет состояние светодиода и передает в приложение строку "*H*" для высокого уровня (`HIGH`) или "*L*" для низкого уровня (`LOW`), чтобы указать новое состояние светодиода. Если полученное сообщение, считанное микроконтроллером, не является "*C*", то яркость светодиода устанавливается на значение, содержащееся в сообщении, с помощью парсинга полученной строки, до получения целого числа, получаемого через инструкцию `toInt(string)`, но только в том случае, если светодиод уже включен. Через каждые две секунды микроконтроллер передает в приложение показания фоторезистора (LDR). Задержка в 20 мс после того, как микроконтроллер передаст сигнал состояния светодиода, блокирует передачу одновременного сигнала со считыванием фоторезистора.

Приложение передает на микроконтроллер либо строку из одной буквы "*C*" при нажатии кнопки *Change LED*, либо значение ползунка яркости при его перемещении. Когда приложение получает строку из одной буквы "*H*" или "*L*", приложение отображает изображение `bulbon` или `bulboff`, а ползунок яркости перемещается в положение 255 или 0 соответственно. Приложение постоянно отображает обновленные значения данных фоторезистора.

Соответствующий скетч для микроконтроллера ESP32 в правой части табл. 10-2 отличается от скетча для микроконтроллера ESP8266 включением библиотеки *BluetoothSerial*, управлением широтно-импульсной модуляцией (ШИМ) и способом создания строки сообщения для микроконтроллера ESP32. Различия между скетчами, кроме используемых выводов, выделены жирным шрифтом. Микроконтроллер ESP32 собирает символы из буфера последовательного порта в строку, в то время как команда ESP8266 `Serial.ReadString()` считывает весь буфер в виде строки. Если ползунок приложения перемещается быстро, то инструкция `Serial.ReadString()` объединяет несколько значений ползунка в строку. Высокая частота процессора микроконтроллера ESP32

позволяет различать значения при быстром перемещении ползунка. ШИМ-инструкция ESP32 `ledcWrite(channel, value)` – это ESP8266-эквивалент инструкций `digitalWrite` и `analogWrite`. Инструкция ESP32 `ledcSetup(channel, 1000, 8)` устанавливает частоту ШИМ для управления светодиодом на 1000 Гц с 8-битным разрешением, что приводит к максимальному значению 255, соответствующему скетчу для ESP8266. В главе 21 («Микроконтроллеры») можно посмотреть подробную информацию о ШИМ у микроконтроллера ESP32.

Таблица 10-2. Скетчи Bluetooth-управления светодиодом для микроконтроллеров ESP8266 и ESP32

Микроконтроллер ESP8266	Микроконтроллер ESP32
	`#include <BluetoothSerial.h>`
	`BluetoothSerial SerialBT;`
`int LEDpin = D4;`	`int LEDpin = 27;`
`int LDRpin = A0;`	`int LDRpin = 33;`
`int bright, LDR;`	`int bright, LDR;`
`int LEDstate = 0;`	`int LEDstate = 0;`
	`int channel = 0;`
	`char c;`
`String str;`	`String str;`
`unsigned int lastTime = 0;`	`unsigned int lastTime = 0;`
`void setup()`	`void setup()`
`{`	`{`
`  Serial.begin(9600);`	`  SerialBT.begin("ESP32 Bluetooth");`
`  pinMode(LEDpin, OUTPUT);`	`  pinMode(LEDpin, OUTPUT);`
	`  ledcAttachPin(LEDpin, channel);`
	`  ledcSetup(channel, 1000, 8);`
	`  ledcWrite(channel, LEDstate);`
`  digitalWrite(LEDpin, LEDstate);`	
`}`	`}`
`void loop()`	`void loop()`
`{`	`{`
`if(Serial.available()>0)`	`if(SerialBT.available()>0)`
`{`	`{`
	`str = «»;`
	`while(SerialBT.available()>0)`
	`{`
`str = Serial.readString();`	`c = SerialBT.read();`
	`str = str + String(c);`
	`}`
`if(str == «C»)`	`if(str == «C» )`
`{`	`{`
`LEDstate = 1- LEDstate;`	`LEDstate = 1 - LEDstate;`
`if(LEDstate == 1)`	`if(LEDstate == 1)`
`Serial.print(«H»);`	`SerialBT.print(«H»);`
`else Serial.print(«L»);`	`else SerialBT.print(«L»);`
`digitalWrite(LEDpin,`	`ledcWrite(channel,`
`LEDstate);`	`LEDstate*255);`
`delay(20);`	`delay(20);`
`}`	`}`
`else if(LEDstate == 1)`	`else if(LEDstate == 1)`

Окончание табл. 10-2

Микроконтроллер ESP8266	Микроконтроллер ESP32
{	{
bright = str.toInt();	bright = str.toInt();
analogWrite(LEDpin,bright);	**ledcWrite(channel, bright);**
}	}
}	}
if(millis()-lastTime > 2000)	if(millis()-lastTime > 2000)
{	{
lastTime = millis();	lastTime = millis();
LDR = analogRead(LDRpin);	LDR = analogRead(LDRpin);
Serial.println(LDR);	**SerialBT.println(LDR);**
}	}
}	}

Макет приложения состоит из нескольких горизонтальных областей (*HorizontalArrangements*), содержащих заголовок приложения, кнопку *ChangeLED* и изображение *LEDimage*, текстовые метки (*labels*) *BrightLabel* и *BrightValue* рядом с ползунком *BrightSlider*, надписи *LDRlabel* и *LDRvalue*, кнопку выбора Bluetooth *SelectBluetooth ListPicker* и кнопку отключения *DisconnectBluetooth* с изображением *Bluetoothimage* и текстовой меткой *StatusLabel* (см. рис. 10-4). Все элементы перечислены в палитре пользовательского интерфейса в левой части окна конструктора. Для программирования приложений рекомендуется выбирать читаемое название для каждого элемента. Изображение *Bluetoothimage* хранится в файле *bluetooth.png*, который загружается в палитру *Media* в правом нижнем углу окна конструктора, если щелкнуть *Bluetoothimage* в разделе *Components* и выбрать имя файла изображения в поле *Picture* раздела *Properties*. Высота и ширина изображения устанавливаются равными 40 и 50 пикселям. При нажатии кнопки *ListPicker* отображается список, представляющий собой перечень доступных подключений Bluetooth. Компоненты *Clock* и *BluetoothClient*, расположенные в палитрах *Sensors* (Датчики) и *Connectivity* (Подключения) в левой части окна конструктора, отображаются под макетом приложения. Текст для кнопок добавляется в разделе *Properties* каждой кнопки. *VerticalArrangements* используются для вставки промежутков между объектами в *HorizontalArrangement*.

Рисунок 10-4. Макет приложения для Bluetooth-управления светодиодом

Как только макет будет завершен, начнется программирование приложения. Для этого щелкните *Blocks* в правом верхнем углу окна конструктора. Программа приложения определяет переменные и процедуры, устанавливает связь по Bluetooth, управляет светодиодом и обрабатывает принятый сигнал от микроконтроллера (см. рис. 10-5, 10-6, 10-7 и 10-8). Блоки имеют цветовую маркировку, и важность читаемых названий для каждого элемента на экране приложения очевидна.

Переменные и процедуры определяются в начале блочной программы, точно так же, как при создании скетча в Arduino IDE. В левой части окна блоков при щелчке по *Variables* (Переменные) отображается ряд опций, включая *initialize global name to*, которая перетаскивается в окно блоков, а *name* изменяется на *Receive*, соответственно сигналу, принятому приложением. Из *Text* в левой части окна блоков выберите и перетащите пустой блок “ ” в окно блоков. Опять же в левой части окна блоков после щелчка по *Procedures* выберите *to Procedure do*, которая перетаскивается в окно блоков, и имя процедуры изменяется на *LED*, соответственно управлению светодиодом. Процедура аналогична функции в среде IDE Arduino, и два входных параметра для процедуры определяются щелчком по сине-белому символу шестеренки и перетаскиванием блоков *input x* в блок *inputs*. В блоке процедуры *LED* измените *x* и *x2* на *picture* и *slider* соответственно (см. рис. 10-5).

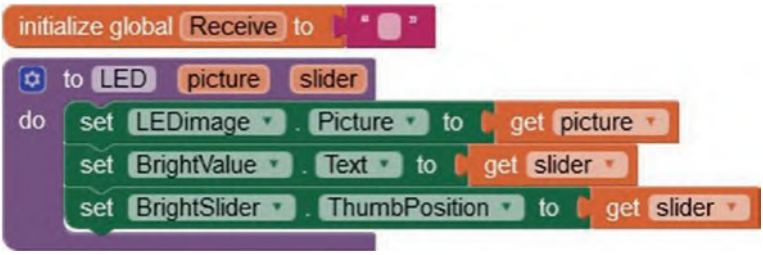


Рисунок 10-5. Определение переменных и процедур

Процедура *LED* устанавливает для *LEDimage* соответствующее изображение при смене состояния светодиода и устанавливает переменные *BrightValue* и *BrightSlider* в положение ползунка при его перемещении. Эти три действия процедуры *LED* определяются так:

1. Щелкните *LEDimage* в левой части окна блоков, выберите опцию *set LEDimage.Picture to* и перетащите ее в окно блоков.
2. Щелкните *BrightValue* в левой части окна блоков, выберите опцию *set BrightValue.Text to* и перетащите ее в окно блоков.
3. Нажмите кнопку *BrightSlider*, выберите опцию *set BrightSlider. ThumbPosition to* и перетащите ее в окно блоков.

Из *Variables* в левой части окна блоков три раза выберите опцию *get*. Девять блоков соединены между собой, как показано на рис. 10-5. В первом блоке *get* выберите *picture*, а в двух других блоках *get* выберите *slider*.

Для связи по Bluetooth существуют три функции: отображение доступных подключений по Bluetooth и подключение или отключение Bluetooth. Доступные подключения Bluetooth отображаются в виде списка с помощью кнопки *SelectBluetooth ListPicker*. В левой части окна блоков при нажатии этой кнопки отображаются опции, в том числе *when SelectBluetooth.BeforePicking* и *set SelectBluetooth.Elements to*, которые нужно перетащить в окно блоков. При нажатии на *BluetoothClient1* отображаются опции, включая *BluetoothClient1. AddressesAndNames*, который также перетаскивается в окно блоков. Три блока соединяются друг с другом, как показано на рис. 10-6.

При нажатии кнопки *SelectBluetooth ListPicker* производится выбранное подключение по Bluetooth, название подключеного *Connected* отображается черным текстом, и вызывается процедура *LED*. Блоки добавляются нажатием кнопки *SelectBluetooth ListPicker* и текстовых меток *BluetoothClient1* и *status-Label*. Блок *if then* выбирается после нажатия *Control* и пустого блока “ ”, если щелкнуть *Text*, а затем изменить пустой блок “ ” на *Connected* или на *bulbon.gif* и нажать кнопку *Procedures*, потом выбрать процедуру *LED*, созданную ранее (см. рис. 10-5). Блок для нулевого значения ползунка (*slider*) выбирается после нажатия кнопки *Math*.

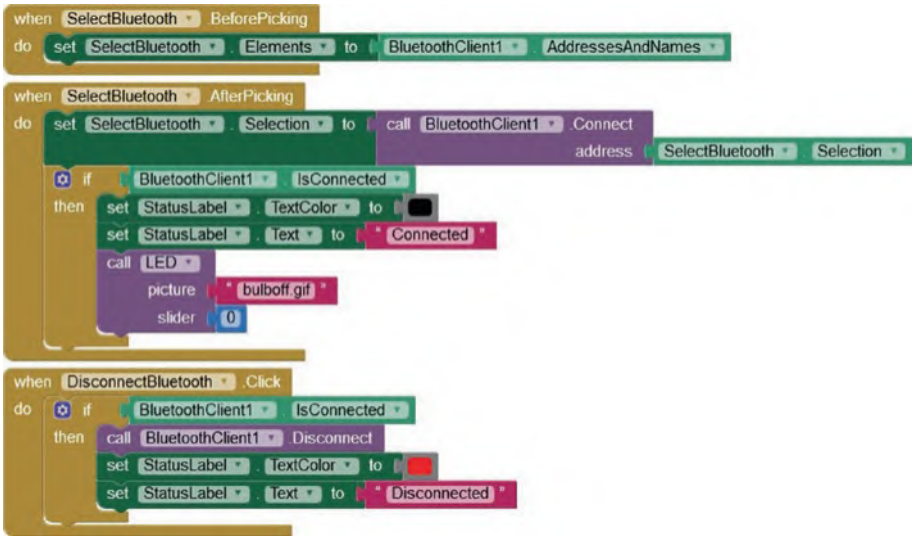


Рисунок 10-6. Bluetooth-подключение

Блоки для отключения соединения Bluetooth добавляются нажатием кнопки *BluetoothDisconnect* и включают опцию *when DisconnectBluetooth.Click*. Цвет текста метки *statusLabel* устанавливается выбором из списка цветов в разделе *Colors*. Видимость текстовой метки определяется в разделе *Properties* в правой части окна конструктора или в окне блоков, если щелкнуть по ней и выбрать для опции *set Label.visible* значение *true* или *false*.

Щелчок по блоку в разделе блоков, а затем щелчок правой кнопкой мыши на *Duplicate* – альтернатива выбору блоков щелчком в левой части окна блоков.

Нажатие кнопки *ChangeLED* в приложении отправляет сообщение со строкой "*C*", и блок строится так, как показано на рис. 10-7. Когда ползунок перемещается, приложение отправляет сообщение со значением ползунка, которое представляет собой действительное число. Блок *round*, расположенный на палитре *Math*, округляет действительное число в большую или меньшую сторону до целого числа. В левой части окна блоков при щелчке на *BrightSlider* появляется опция *when BrightSlider.PositionChanged*. В блок *call Bluetooth1.SendText text* добавьте блок *get ThumbPosition*, который получается при наведении курсора мыши на *ThumbPosition* в блоке *when BrightSlider.PositionChanged*, и перетащите блок *get ThumbPosition* в нужное положение (см. рис. 10-7).

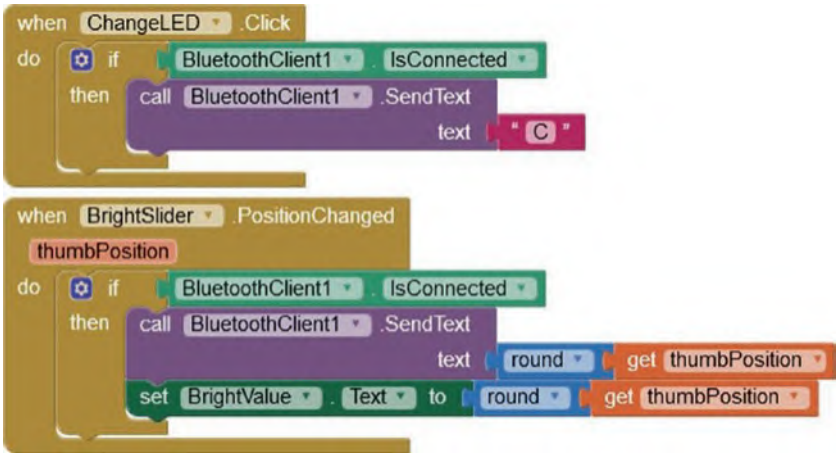


Рисунок 10-7. Управление светодиодом

Когда приложение получает сообщение, содержащее строку "*H*" или "*L*", процедура *LED* демонстрирует изображение *bulbon.gif* или *bulboff.gif* и перемещает ползунок в положение 225 или 0 соответственно. В противном случае обновляется текстовая метка *LDRvalue*.

В левой части окна блоков при нажатии кнопки *Clock1* необходимо выбрать опцию *when Clock1.Timer do*, при щелчке по *Text* – опцию *contains text piece*, а при щелчке по *Variables* – опции *get* и *global Receive*. Блок *length* получается щелчком мыши по *Text*. Блок условия *greater than* (больше чем) получается нажатием кнопки *Math* и выбором блока *equals* (равно), который и нужно изменить на *greater than*. Блоки *if then*, *else if then*, *else* создаются из блока *if then*, если щелкнуть по символу сине-белой шестеренки и перетащить блоки *else if* и *else* в блок *if*. Блоки компонентов добавляются, как показано на рис. 10-8.

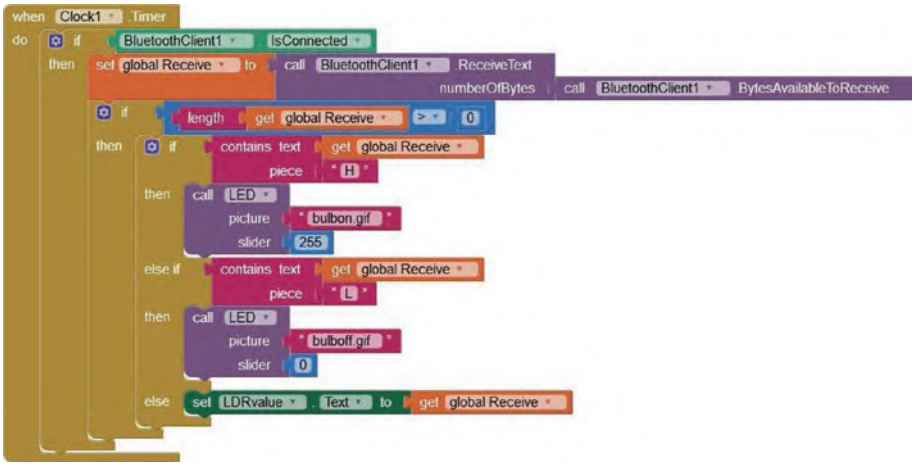


Рисунок 10-8. Обработка полученного сигнала Bluetooth

Блок *is number?*, полученный при нажатии кнопки *Math*, применяется к текстовой переменной, чтобы обеспечить действие условия над полученным текстом. Например, блок *else* перед блоком *set LDRvalue.Text* внизу рис. 10-8 заменяется блоком *else if then* и блоком *is number?*, как показано на рис. 10-9.

Рисунок 10-9. Условие для числового значения

Установка приложения

После разработки макета приложения и программирования приложения с помощью визуального языка на основе блоков приложение создается путем выбора *App (save .apk to my computer)* в меню *Build* в верхней части окна конструктора. Затем файл *.apk*[55] загружается на планшет или мобильный телефон Android с установкой приложения с помощью установщика. В качестве альтернативы можно создать QR-код для быстрой загрузки с помощью выбора App (provide QR code for .apk). Далее QR-код сканируется, чтобы начать загрузку файла .apk, и приложение устанавливается с помощью установщика.

Чтобы установить приложение, необходимо на планшете или мобильном телефоне Android в настройке *Security* (Безопасность) отметить пункт *Unknown sources* (Разрешить установку приложений из неизвестных источников)[56]. После установки приложения сбросьте настройки безопасности в исходное состояние.

Значок приложения по умолчанию *MIT App Inventor* (см. рис. 10-10) может быть заменен пользовательским изображением. В окне конструктора приложения в разделе *Properties* для *Screen1* свойство *Icon* установлено в *None* по умолчанию. Значок приложения заменяется загрузкой требуемого изображения в формате JPG или PNG, размер которого не должен превышать 50×50 пикселей.

Рисунок 10-10. Логотип MIT App Inventor

[55] .apk (Android Package Kit) – формат архивированных файлов приложений для ОС Android. Apk-файл не является исполняемым файлом Android, подобно exe-файлам в Windows, это аналог архива (*cab*, *zip* или, скорее, *msi* в Windows), который для работы приложения еще требуется распаковать и установить. При загрузке с официальных сайтов (например, *Google Play*) установка произойдет автоматически с помощью встроенного установщика. При загрузке из сторонних источников требуется выполнить ряд дополнительных процедур, для простых случаев описанных автором далее. – *Прим. перев.*

[56] Для разных версий Android пути и названия пунктов меню могут отличаться. – *Прим. перев.*

Приложение для управления сервороботом

В главе 9 («WebSocket») сервопривод, подключенный к микроконтроллеру, управлялся перемещением ползунка на веб-странице клиента, при этом положение ползунка передавалось на сервер, установленный на микроконтроллере, для перемещения сервопривода на требуемый угол. В этой главе два сервопривода перемещаются в требуемые положения с помощью приложения, управляющего сервоприводами, подключенными к плате ESP8266 или ESP32. Положения сервопривода сохраняются, а затем повторяются для имитации автоматической последовательности движений робота. Приложение управляет одним сервоприводом для перемещения влево-вправо (LR) и вторым сервоприводом для перемещения вперед-назад (FB) поворотно-наклонного кронштейна (см. рис. 10-11). Нажатие соответствующей кнопки приложения приводит в движение сервопривод непрерывно до тех пор, пока не будет нажата кнопка *Stop*. Положения сервоприводов сохраняются нажатием кнопки *Save position*, при этом приложение отображает количество сохраненных позиций. Сохраненная последовательность движений воспроизводится нажатием кнопки *Playback*. Кнопка *Reset* отменяет сохраненные положения сервоприводов.

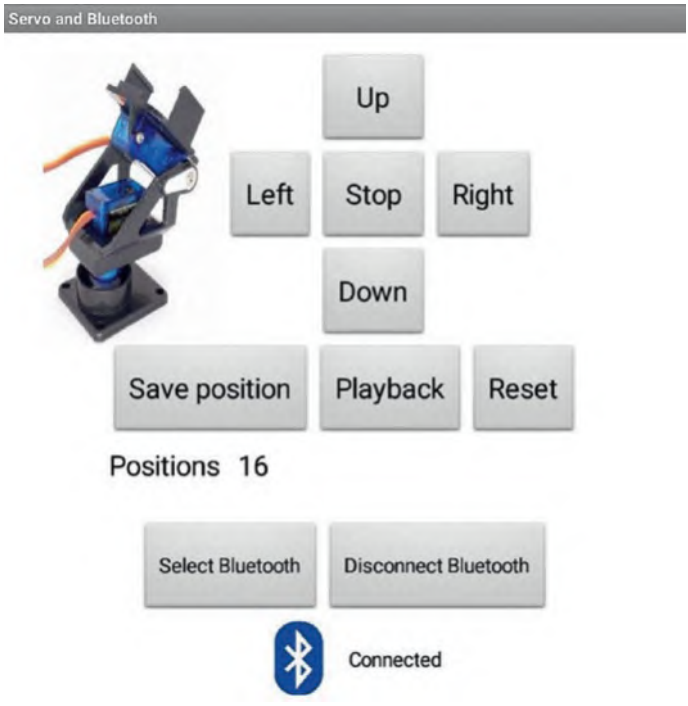


Рисунок 10-11. Управление сервоприводами

Приложение на планшете или мобильном телефоне Android взаимодействует по Bluetooth с микроконтроллером ESP8266 или ESP32, подключенным к двум сервоприводам (см. рис. 10-12 и подключения в табл. 10-3). Нажатие

кнопки в приложении передает командное сообщение, которое перемещает соответствующий сервопривод в нужном направлении. Сервоприводы питаются от внешнего источника питания напряжением 5 В, так как их двигатели могут использовать сотни миллиампер в течение нескольких миллисекунд[57], пока вращается ротор, что больше, чем могут выдать платы ESP8266 или ESP32 на выводах 5V или VIN.

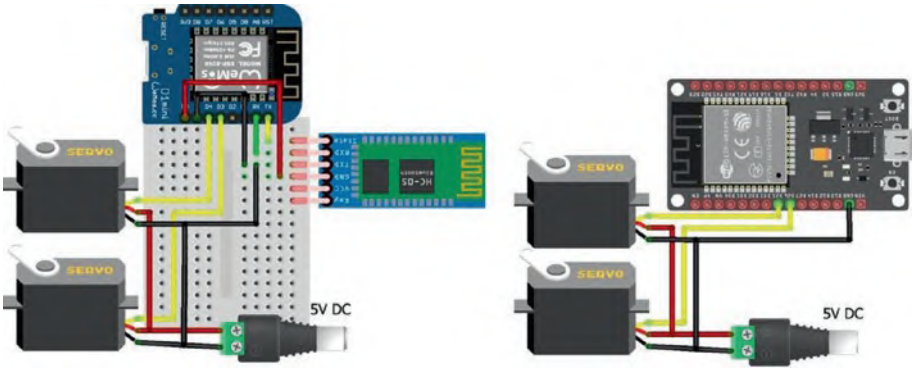


Рисунок 10-12. Сервоприводы и Bluetooth с платами ESP8266 и ESP32

Для микроконтроллера ESP32 требуется специальная библиотека сервоприводов для ESP32, а не встроенная библиотека *Servo* Arduino IDE. Рекомендуется библиотека *ESP32Servo* Кевина Харрингтона (Kevin Harrington) и Джона К. Беннетта (John K. Bennett), также доступная в Arduino IDE. Инструкции встроенной библиотеки *Servo* для микроконтроллера ESP8266

```
#include <Servo.h>     // include Servo library
servoFB.attach(FBpin) // initialise servo motor to FBpin
```

заменяются следующими инструкциями библиотеки ESP32Servo для микроконтроллера ESP32:

```
#include <ESP32Servo.h>
servoFB.setPeriodHertz(F)            // define servo frequency (F)
servoFB.attach(FBpin, min, max)      // initialise servo motor to FBpin
```

Частота прямоугольных колебаний *F* для управления сервоприводом содержится в команде `servofb.setPeriodHertz(F)` и обычно составляет 50 Гц. В инструкции `servoFB.attach(FBpin, min, max)` параметры `min` и `max` относятся к длительности импульса в микросекундах для перемещения сервопривода на 0° и 180° соответственно. Значения по умолчанию для параметров `min` и `max` составляют 1000 мкс и 2000 мкс, а для сервопривода Tower Pro SG90 – 500 мкс и 2500 мкс.

В следующих инструкциях изменений нет:

```
Servo servoFB                // associate servoFB with Servo lib
servoFB.writeMicroseconds(T) // move to position mapped to Tμs
servoFb.write(N)             // move to angle N°
```

[57] См. сноску 47 на стр. 157. – *Прим. перев.*

Микроконтроллер ESP8266 использует последовательный порт для загрузки скомпилированного скетча. Во время загрузки необходимо отсоединить RX-вывод микроконтроллера ESP8266, иначе загрузка завершится ошибкой.

Таблица 10-3. Серводвигатели и связь по Bluetooth с платой ESP8266 и ESP32

Компонент	Подключено к	ESP8266	ESP32
Сервопривод VCC (красный)	Внешний источник 5 V		
Сервопривод GND (коричневый или черный)	Внешний источник GND	GND	GND
Сигнальный вывод сервопривода (оранжевый или белый[58])		D3, D4	GPIO 25, GPIO 26
Bluetooth HC-05 VCC		5V	
Bluetooth HC-05 GND		GND	
Bluetooth HC-05 TXD		RX	
Bluetooth HC-05 RXD		TX	

Скетч в листинге 10-1 предназначен для платы ESP8266. Инструкции для платы ESP32 включают библиотеку *BluetoothSerial* с заменой инструкции `Serial` на `SerialBT` (см. табл. 10-2), поскольку передаваемые сообщения содержат только один символ. Полученный микроконтроллером ESP8266 или ESP32 командный символ либо изменяет положение сервопривода, либо вызывает функцию для сохранения, воспроизведения или сброса положения сервопривода. При перемещении сервоприводов скорость перемещения определяется задержкой между перемещениями, а задержка в 100 мс позволяет точно перемещать сервоприводы в требуемые положения. После сохранения состояния сервоприводов микроконтроллер передает номер состояния для отображения в приложении. При воспроизведении последовательности положений сервоприводов задержка в 15 мс является компромиссом между быстрым перемещением сервоприводов и обеспечением достаточного времени для их перемещения в сохраненные положения. Сервоприводы перемещаются в начальное или исходное положение перед воспроизведением сохраненной последовательности положений серводвигателя, но не после воспроизведения, так как конечное положение не обязательно должно совпадать с начальным положением.

Последовательность положений сервоприводов сохраняется в двух массивах, по одному для каждого привода, но информация не сохраняется в памяти при отключении питания платы ESP8266 или ESP32. Сохранение состояний сервоприводов в энергонезависимой памяти EEPROM сохранит информацию, когда плата ESP8266 или ESP32 отключается, как описано в главе 20 («OTA и сохранение данных в EEPROM, SPIFFS и Microsoft Excel»).

Функция `playServo` управляет воспроизведением сохраненной последовательности положений сервоприводов. Два сохраненных положения могут отличаться только для одного сервопривода, если изменение положения было

[58] См. сноску 48 на стр. 158. – *Прим. перев.*

только поворотом (влево-вправо) или наклоном (вперед-назад). Если новое положение отличается от предыдущего положения только для одного сервопривода, определяется количество и направление перемещений с шагом по 5° в новое положение без изменения положения другого сервопривода.

Листинг 10-1. Серводвигатели и Bluetooth

```
#include <Servo.h>                    // include Servo library
Servo servoFB;                        // associate servos with library
Servo servoLR;
int FBpin = D3;                       // servo motor pins
int LRpin = D4;
int FBpos[20];                        // arrays for saved servo positions
int LRpos[20];
int Nservo = 0;                       // number of saved positions
int FB, LR, steps, stepsize;
char c;
void setup()
{
  Serial.begin(9600);                 // Bluetooth module baud rate
  servoFB.attach(FBpin);              // servo motor pin to Servo lib
  servoLR.attach(LRpin);
  startPosition();                    // set initial servo positions
}
void loop()
{                                     // read character in Serial buffer
  if(Serial.available()>0) c = Se rial.read();
  if(c == 'U') FB = FB-5;             // move servo forward (up)
  else if(c == 'D') FB = FB+5;        // move servo backward
  else if(c == 'L') LR = LR+5;        // move servo left
  else if(c == 'R') LR = LR-5;        // move servo right
  else if(c == 'Z') delay(100);       // stop moving both servos
  else if(c == 'S') saveServo();      // save both servo positions
  else if(c == 'P') playServo(15);    // playback servo positions
  else if(c == 'E') resetServo();     // reset saved positions
  if(c != 'Z' && c !=' ') moveServo(FB, LR, 100);
}                                     // move both servos
void startPosition()                  // function to set initial servo positions
{
  FB = 50;                            // arbitrary home position
  LR = 70;
  moveServo(FB, LR, 100);             // move servos to initial position
}
void moveServo(int vFB, int vLR, i nt lag)
{                                     // function to move servos
  vFB = constrain(vFB, 5, 100);       // constrain servo positions
  vLR = constrain(vLR, 5, 175);
  servoFB.write(vFB);                 // move forward-backward servo
  delay(lag);                         // time between servo movements
  servoLR.write(vLR);                 // move left-right servo
  delay(lag);
}
void saveServo()                      // function to save servo positions
{
  Nservo++;                           // increment number of positions
  Serial.println(Nservo);             // transmit position number to app
```

```
    FBpos[Nservo] = FB;                // save forward-backward position
    LRpos[Nservo] = LR;                // save left-right position
    c = ' ';                           // reset command value
}
void playServo(int lag)                // function to play back servo positions
{
    startPosition();                   // move servos to initial position
    FBpos[0] = FB;
    LRpos[0] = LR;
    for (int i=1; i<Nservo+1; i++)     // cycle through saved positions
    {
      if(FBpos[i] != FBpos[i-1])       // forward-back position change
      {
        steps = abs((FBpos[i] - FBpos [i-1])/5); // number of steps
        stepsize = 5;                  // magnitude of step size
        if(FBpos[i] < FBpos[i-1]) steps ize = -5;
                                       // change in FB from FBpos[i-1],LRpos[i-1]
        for (int j = 0; j<steps; j++)
            moveServo(FBpos[i-1]+j*stepsize, LRpos[i-1], lag);
      }
      if(LRpos[i] != LRpos[i-1])       // left-right position change
      {
        steps = abs((LRpos[i] - LRpos[i-1])/5);
        stepsize = 5;
        if(LRpos[i] < LRpos[i-1]) stepsize = -5;
                              // now at FBpos[i], so change in LR to LRpos[i]
        for (int j = 0; j<steps; j++)
              moveServo(FBpos[i], LRpos[i-1]+j*stepsize, lag);
      }
    }
    c = ' ';                           // reset command value
}
void resetServo()                      // function to reset saved positions
{
    Nservo = 0;                        // reset position number to zero
    Serial.println(Nservo);            // transmit position number to app
    c = ' ';                           // reset command value
}
```

Макет приложения состоит из пяти кнопок для управления движениями сервоприводов, причем кнопки расположены в таблице *TableArrangement*, находящейся в палитре *Layout* в левой части окна конструктора, состоящей из трех строк и столбцов (см. рис. 10-13). Вторая область *HorizontalArrangement* содержит три кнопки для сохранения, воспроизведения и сброса положения сервопривода. Построение приложения для подключения по Bluetooth такое же, как описано для приложения согласно рис. 10-4. В палитре *User Interface* поле для изображения логотипа *Image* расположено в первой области *HorizontalArrangement*. Изображение поворотно-наклонного кронштейна с сервоприводами загружено в палитру *Media* в правом нижнем углу окна конструктора. В разделе *Properties* в правой части окна конструктора изображение выбирается в пункте *Picture*, при этом высота и ширина изображения устанавливаются в состояние *Fill parent* и 90 пикселей соответственно.

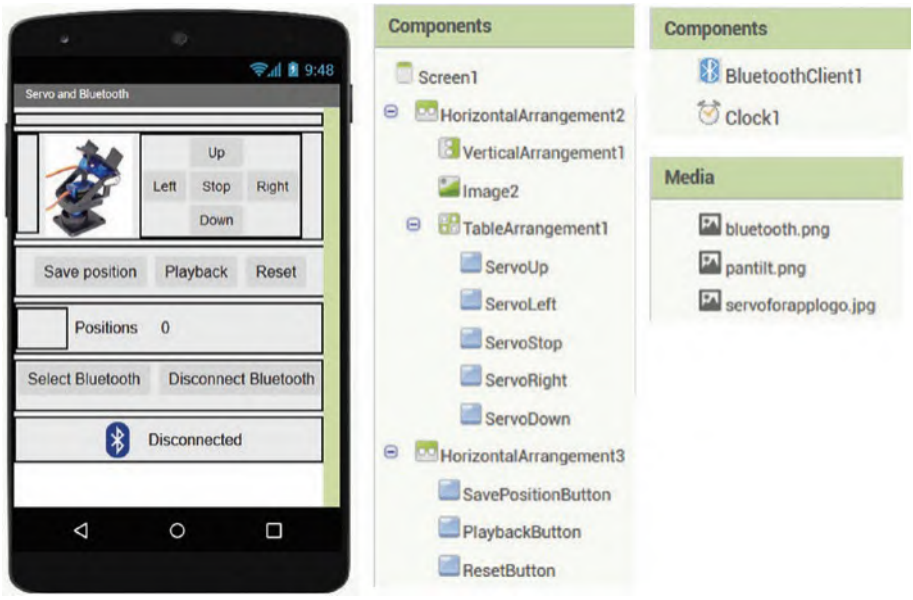


Рисунок 10-13. Макет приложения для управления сервоприводом

Все блоки приложений основаны на нажатии кнопки, которая передает соответствующий командный символ на микроконтроллер ESP8266 или ESP32, подключенный к двум сервоприводам (см. рис. 10-14). Символы команд – это первая буква названия каждой команды, например "*L*" для движения влево (*left*), за исключением "*Z*" и "*E*" для остановки (*stop*) и сброса (*reset*).

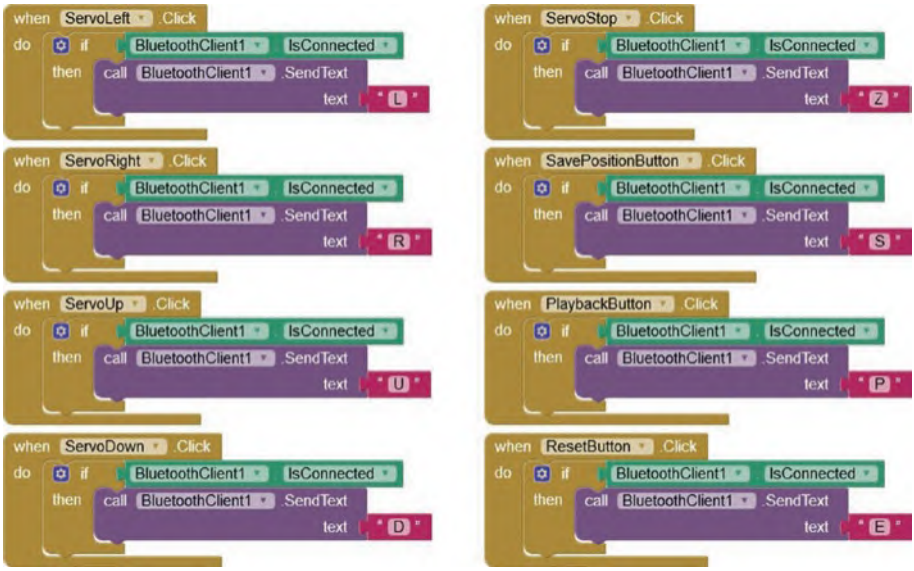


Рисунок 10-14. Команды сервоприводов

Блок, позволяющий приложению получать переданное количество сохраненных положений серводвигателя, задает текстовая метка *PositionsLabel* для передаваемого сообщения (см. рис. 10-15).

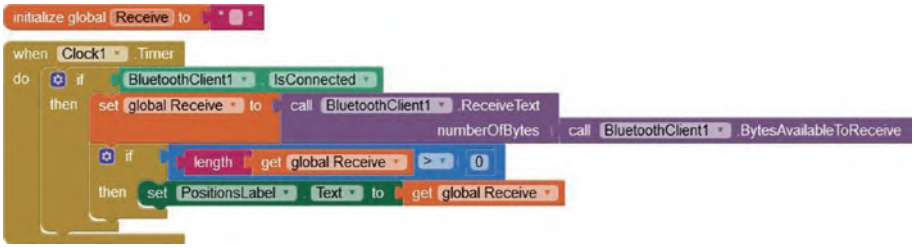


Рисунок 10-15. Количество сохраненных положений сервоприводов

Блоки *SelectBluetooth.BeforePicking*, *SelectBluetooth.AfterPicking* и *DisconnectBluetooth* идентичны изображенным на рис. 10-6.

Приложение для распознавания речи

Распознавание речи используется для управления приложением вместо нажатия экранных кнопок. Приложение запрашивает речь пользователя и преобразует произносимый звук в текст, используя функцию распознавания речи планшета или мобильного телефона Android, для чего может потребоваться доступ в интернет. Макет приложения состоит из областей *HorizontalArrangements*, содержащих заголовок приложения, кнопку *Speech* и текстовое поле *Spokentext*. Изображение говорящего человека *speech.png* загружается в палитру *Media* и сопоставляется с кнопкой *Speech* в разделе *Properties*. Текстовая метка *SpokenText* содержит текст, сгенерированный функцией распознавания речи (см. рис. 10-16).

Рисунок 10-16. Макет приложения для распознавания речи

Окно блоков состоит всего из четырех блоков (см. рис. 10-17). Когда пользователь нажимает кнопку *Speech*, функция распознавания речи *SpeechRecognizer*, расположенная в палитре *Media* в правой части окна конструктора, очищает содержимое метки *Spokentext*, просит пользователя говорить и преобразует речь в текст, который отображается в этой метке.

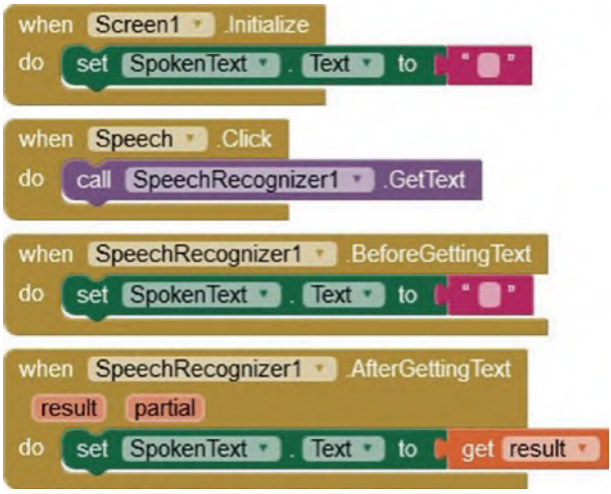


Рисунок 10-17. Распознавание речи

Устройства управляются на основе содержимого текста, полученного с помощью функции распознавания речи *SpeechRecognizer*. Например, если речь и соответствующий текст в *SpokenText* содержат фразу «*LED on*» или «*LED off*», то в приложении отображается соответствующее изображение, указывающее состояние светодиода (см. рис. 10-18). Изображения *bulbon.png* и *bulboff.png*[59] загружаются в палитру *Media* в окне конструктора, причем изображение *bulboff.png* сопоставляется с *LEDimage* в разделе *Properties*.

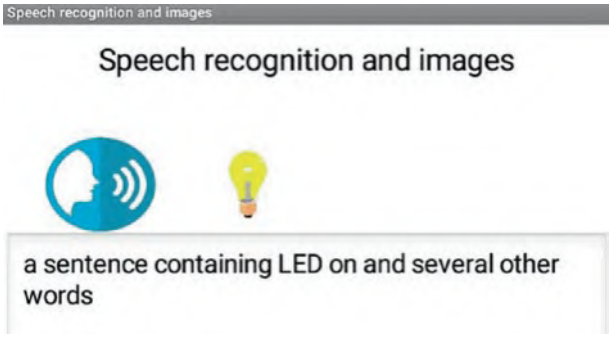


Рисунок 10-18. Распознавание речи с управлением изображением

[59] Следует обратить внимание на то, что ранее упоминались рисунки в формате GIF (*bulbon.gif* и *bulboff.gif*), которые рекомендовалось получать с сайта www.w3schools.com (см. стр. 162 и текст скетча на стр. 164). Разумеется, для данной цели подойдут изображения в любом из этих форматов. – *Прим. перев.*

Функция `SpeechRecognizer.AfterGetting` на рис. 10-17 теперь вызывает процедуру *LED* с изображением, зависящим от содержимого метки *SpokenText* – фразы «*LED on*» или «*LED off*» (см. рис. 10-19). Процедура *LED* на рис. 10-5, которая отображает изображение, указывающее состояние светодиода, комбинируется с процедурой *Change LED* на рис. 10-7, отправляющей сообщение, состоящее из символа "*C*", на микроконтроллер ESP8266 или ESP32. Скетчи для микроконтроллера ESP8266 или ESP32 в табл. 10-2 для включения или выключения светодиода по сообщению "*C*" не изменяются, так как не меняются реакции микроконтроллера на сообщения, отправляемые приложением.

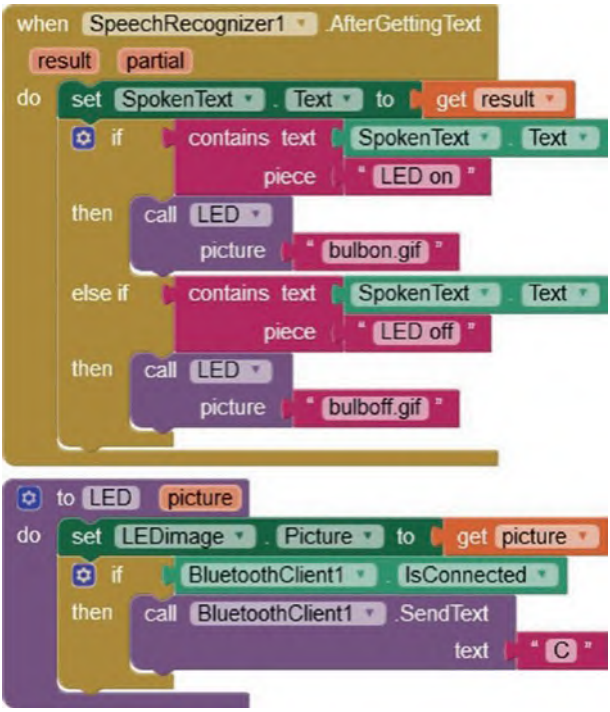


Рисунок 10-19. Распознавание речи с управлением изображением и Bluetooth

Обратите внимание, что кнопка *SelectBluetooth ListPicker*, кнопка *DisconnectBluetooth* и изображение Bluetooth с меткой *StatusLabel* включены в макет приложения (см. рис. 10-4) вместе с соответствующими блоками для подключения по Bluetooth (см. рис. 10-16 и 10-20).

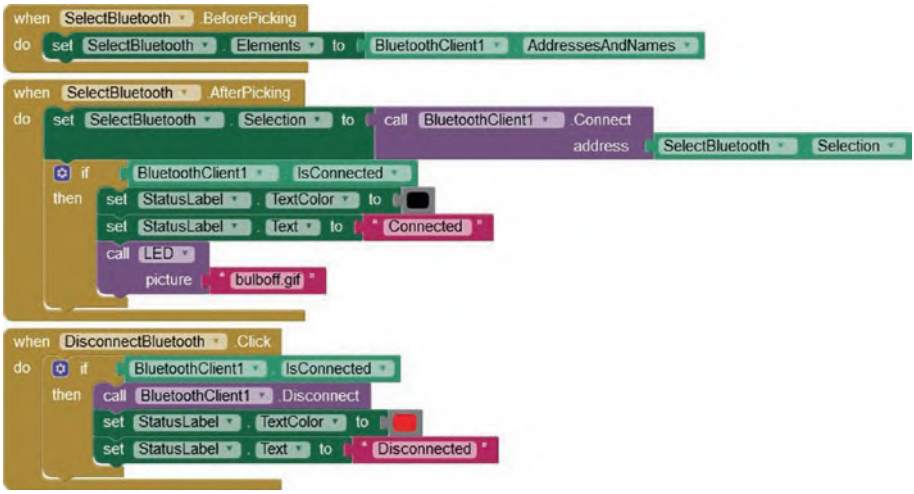


Рисунок 10-20. Подключение по Bluetooth

Вместо того чтобы приложение распознавания речи включало или выключало светодиод непосредственно, управление осуществляется с помощью реле, подключенного к плате ESP8266 или ESP32. Скетчи в табл. 10-2 не изменяются, но текст в метке *SpokenText* (см. рис. 10-19) изменен с «*LED on*» и «*LED off*» на «*Relay on*» и «*Relay off*». Глава 15 («Радиочастотная связь») включает описание управления устройством с помощью реле. Например, релейный модуль MOSFET IRF520 может переключать постоянный ток до 100 В (DC) при 10 А, чтобы обеспечить питание двигателя, источника света или других устройств (см. рис. 10-21).

Если нагрузкой является двигатель постоянного тока, то индуктивность его обмоток будет генерировать скачок напряжения для поддержания тока через обмотки при отключении питания. Установка диода навстречу питания двигателя предотвращает скачок напряжения и рассеивает энергию через двигатель при выключении питания. Рекомендуется использовать диод Шоттки, который представляет собой быстродействующий диод с низким прямым падением напряжения.

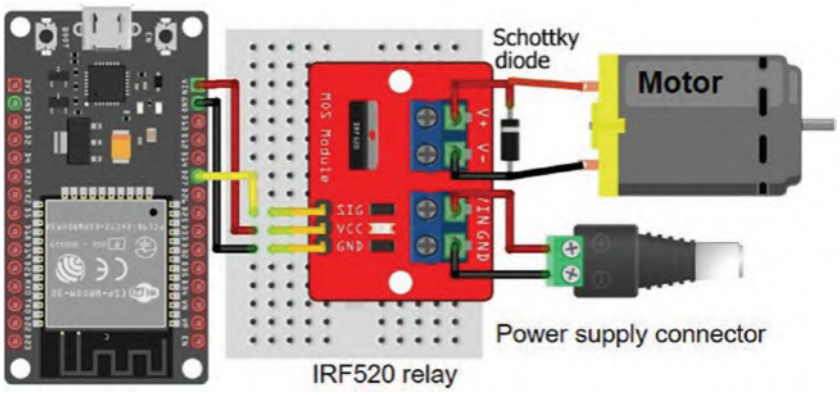


Рисунок 10-21. Релейный модуль MOSFET IRF520 и ESP32

Итоги

Было разработано приложение для планшета или мобильного телефона Android для управления яркостью светодиода, подключенного к плате ESP8266 или ESP32, с обратной связью от фоторезистора. Приложение запрограммировано с помощью *MIT App Inventor* с визуальным языком программирования на основе блоков. Плата ESP8266 или ESP32 и планшет или мобильный телефон Android взаимодействуют с помощью Bluetooth. Плата ESP32 имеет встроенную функциональность Bluetooth, а к плате ESP8266 подключен модуль Bluetooth HC-05. Два сервопривода управлялись с помощью приложения с сохранением положений сервоприводов для повторения, чтобы имитировать автоматическую последовательность движений робота. Приложение, основанное на распознавании речи, управляло состоянием светодиода, подключенного к плате ESP8266 или ESP32, при этом изображения демонстрировали состояние светодиода в приложении на планшете или мобильном телефоне Android.

Перечень компонентов

- Микроконтроллер ESP8266: LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: DEVKIT DOIT или NodeMCU
- Светодиод
- Фоторезистор
- Резисторы: 220 Ом, 4.7 кОм
- Модуль Bluetooth: HC-05
- Реле: модуль IRF520

Глава 11

Приложение базы данных и Google Maps

MIT App Inventor содержит функцию базы данных для хранения в приложении данных *Google Maps* (карты Google), доступных с помощью *MIT App Inventor*, при условии что планшет или мобильный телефон Android, на котором размещено приложение, имеют доступ в интернет. В этой главе описывается метод хранения данных о маршруте в базе данных приложения, при этом приложение отображает информацию о маршруте с помощью *Google Maps*. Создание базы данных приложения и интеграция с *Google Maps* являются основой для главы 12 («Приложение для GPS-трекинга в Google Maps»), где разрабатывается приложение для GPS-трекинга с помощью модуля спутниковой навигации, подключенного к плате ESP8266 или ESP32.

База данных MIT App Inventor

Функция базы данных *MIT App Inventor* сохраняет данные, когда приложение выключается. Базы хранят данные в структурированном формате, с данными, привязанными к индексу, называемому *MIT App Inventor tag*. База данных позволяет добавлять, изменять и удалять данные с поиском по определенному тегу базы данных с помощью *MIT App Inventor ListPicker* и отображением содержимого с помощью *MIT App Inventor ListView*. Элемент базы данных состоит из двух компонентов: тега базы данных и значения, связанного с этим тегом. Например, элемент базы данных может состоять из "*Longest_day, June, 21*", где "*Longest_day*" («самый длинный день») является тегом базы данных, а "*June, 21*" («21 июня») – значением, связанным с тегом "*Longest_day*". Значение, связанное с тегом, представляет собой строку, включающую несколько компонентов, каждый из которых разделен запятой “,”, обратной косой чертой “\”, хешем (“#”) или амперсандом (“&”). Тег базы данных представляет собой буквенно-цифровой текст, поэтому формат для элемента базы данных адреса и дня рождения – "*Александр, Скотланд-стрит, 44, 1 января*", где "*Александр*" – это тег базы данных, который ссылается на значение, состоящее из двух компонентов: адрес

"*Скотланд-стрит, 44*" и день рождения "*1 января*". Доступ к информации базы данных осуществляется путем поиска по тегу базы данных, а не по соответствующему значению.

В примере базы данных, который демонстрирует добавление, удаление, изменение и отображение содержимого базы данных, приложение состоит из двух экранов (см. рис. 11-1 и 11-2). Элемент базы данных включает в себя тег базы данных *Text1*, который чувствителен к регистру, и значение базы данных, состоящее из двух текстовых компонентов *Text2* и *Text3*, которые являются буквенно-цифровыми. На экране *Screen1* элемент базы данных вводится в соответствующие три текстовых поля и сохраняется в базе данных нажатием кнопки *SavetoDbase* (см. рис. 11-1). Элемент базы данных ищется с помощью ввода текста в текстовое поле *FindDbaseText1* и нажатия кнопки *FindTextButton* для запуска поиска в базе данных тега базы данных, соответствующего введенному тексту. После этого элемент базы данных отображается в трех текстовых полях: *Text1*, *Text2* и *Text3*. Отображаемый элемент базы данных удаляется из базы данных нажатием кнопки *DeleteDbaseItem*. Экран *Screen1* очищается нажатием кнопки *ClearScreen*.

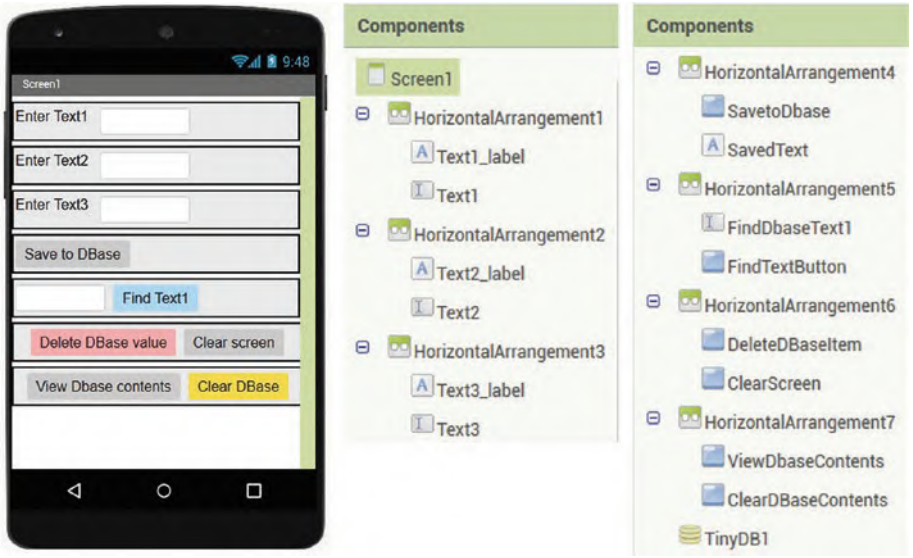


Рисунок 11-1. Макет экрана Screen1 приложения базы данных

Содержимое базы данных отображается нажатием кнопки *ViewDbaseContents*, которая переходит на второй экран *Screen2*, где элементы базы данных отображаются в списке *ListView* как "*Text1 и Text2&Text3*" (см. рис. 11-2). Нажатие кнопки *ClearDbaseDisplay* очищает отображение элементов базы данных на *Screen2*. Нажатие кнопки *SelectandDisplay ListPicker* выводит список тегов базы данных в *Текст1*. Когда выбран тег базы данных, элемент базы данных отображается как "*Text1 Text2&Text3*" в текстовые поля *DbaseTag* и *DbaseValue* соответственно. Нажатие кнопки *ClearDisplay* очищает элемент базы данных на экране *Screen2*. Действие кнопки *GotoScreen1* говорит само за себя.

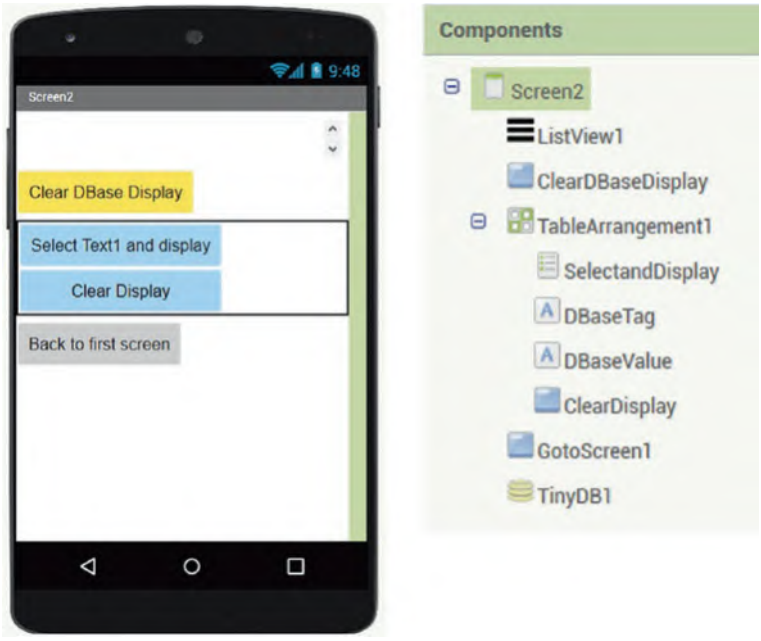


Рисунок 11-2. Макет экрана Screen2 приложения базы данных

Доступ к базе данных *TinyDB1 MIT App Inventor* осуществляется из палитры *Storage* в левой части окна конструктора, а компонент *TinyDB* перетаскивается на *Screen1*. *ListView* и *ListPicker* расположены в палитре *User Interface*.

Элемент базы данных сохраняется с помощью TinyDB-функции *storeValue*, при условии что и *Text1*, и *Text3* не являются пустыми, при этом компоненты значения базы данных, *Text2* и *Text3*, соединяются символом амперсанда "&" (см. рис. 11-3).

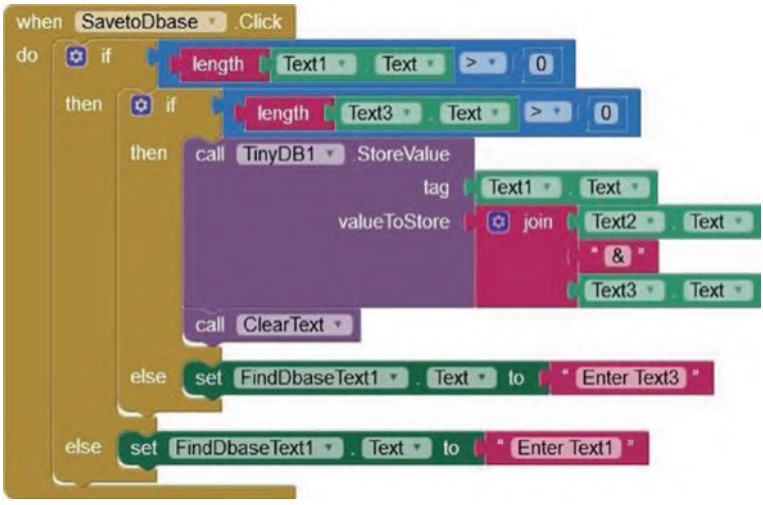


Рисунок 11-3. Сохранение элемента базы данных

Содержимое *Screen1* очищается с помощью процедуры *ClearText*, которая заменяет текстовые поля пустым текстом (см. рис. 11-4). Элемент базы данных удаляется из базы данных с помощью функции TinyDB *ClearTag*, которая удаляет *Text1* из базы данных, поскольку *Text2* и *Text3* впоследствии перезаписываются. Все содержимое базы данных удаляется с помощью TinyDB-функции *ClearAll*.

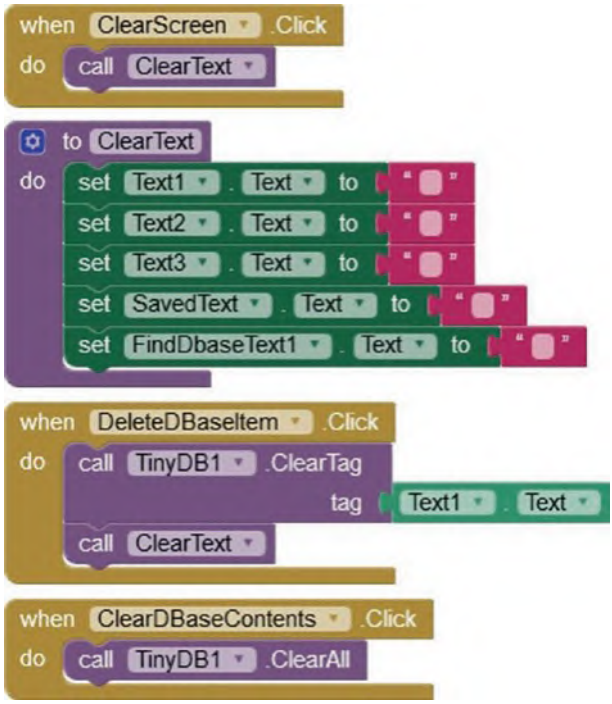


Рисунок 11-4. Удаление элемента базы данных или очистка всех элементов базы данных

Доступ к элементу базы данных осуществляется с помощью тега базы данных *Text1*, при этом компоненты значений базы данных индексируются как элементы списка 1, 2 и т. д. Если тег базы данных найден, значение базы данных разбивается на компоненты с использованием разделяющего символа амперсанда “&”; в противном случае отображается сообщение об ошибке (см. рис. 11-3 и 11-5).

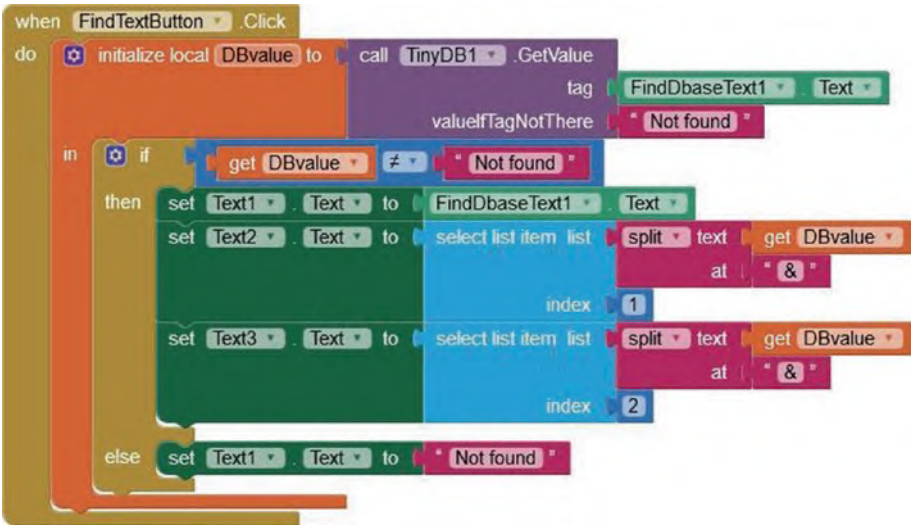


Рисунок 11-5. Обнаружение элемента базы данных

Содержимое базы данных отображается путем выделения тегов базы данных в список *DBList* с помощью TinyDB-функции *GetTags*. Соответствующие значения базы данных добавляются в список с помощью TinyDB-функции *GetValue*, а содержимое списка отображается с помощью функции *ListView* (см. рис. 11-6).

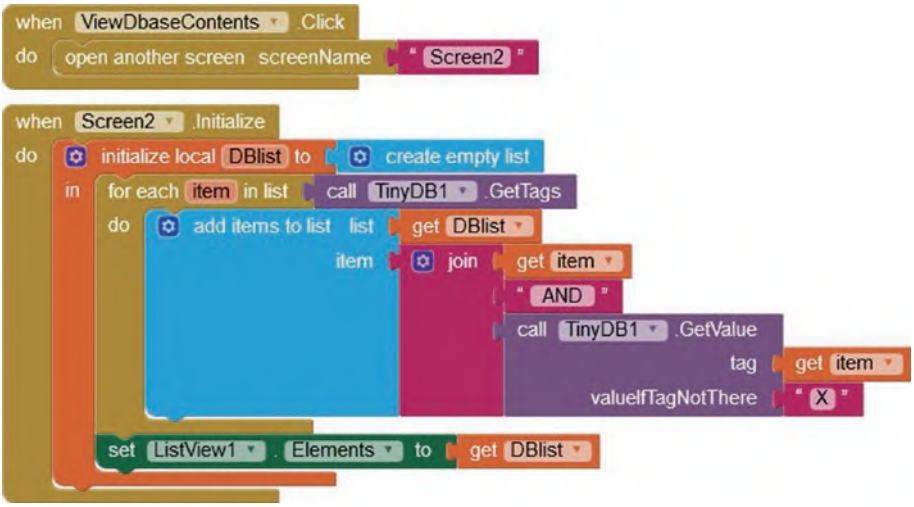


Рисунок 11-6. Отображение содержимого базы данных

Элемент базы данных выбирается из списка с помощью TinyDB-функций *GetTags* и *GetValue*. Список удаляется с помощью TinyDB-функции *create empty list*. Приложение перемещается на другой экран с помощью опции управления *open another screen* (см. рис. 11-7).

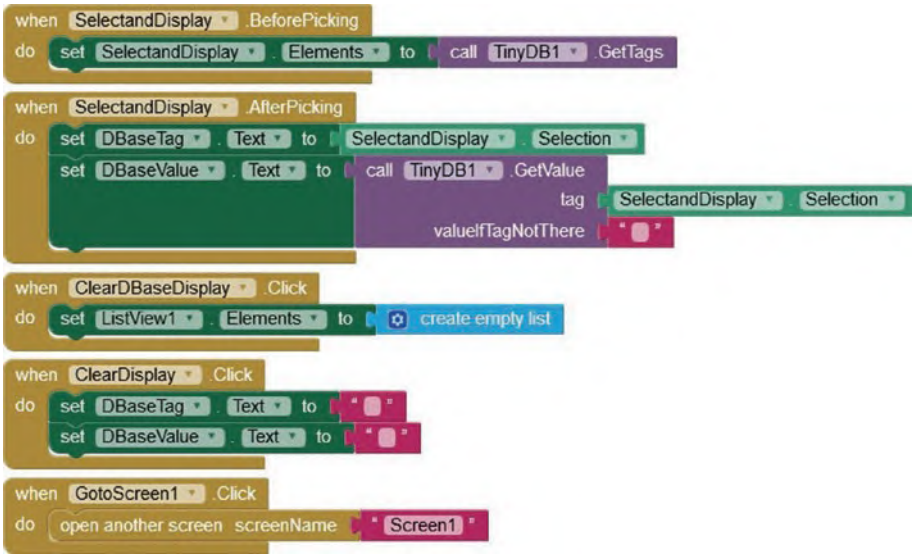


Рисунок 11-7. Отобразить или очистить отображение выбранного элемента базы данных

MIT App Inventor и Google Maps

Карты Google доступны с помощью *MIT App Inventor*, при условии что планшет или мобильный телефон Android, на котором размещено приложение, имеет доступ в интернет. Пример приложения демонстрирует доступ к *Google Maps*, сохранение широты местоположения (положение север–юг) и информацию о долготе (положение восток–запад) в базе данных приложения, размещение маркеров на карте, увеличение и уменьшение масштаба карты и отображение дорожных или аэрофотоснимков (см. рис. 11-8). Карта перемещается перетаскиванием, двойное касание увеличивает масштаб карты, и при касании отображаются широта и долгота указываемой позиции. Местоположение сохраняется в базе данных приложения после ввода имени местоположения и нажатия кнопки *AddLocation*. Кнопка выбора местоположения *ChooseLocation* отображает список сохраненных местоположений в базе данных приложения с картой, центрированной на выбранном местоположении. Кнопка удаления местоположения *DeleteLocation* удаляет сохраненное местоположение из базы данных приложения.

Рисунок 11-8. Приложение базы данных местоположений

Первая область *HorizontalArrangement* макета базы данных местоположений включает карту с маркером, доступ к которой осуществляется из палитры *Maps* (см. рис. 11-9). Карта центрирована, а маркер расположен на широте–долготе, определенной в свойстве *CenterFromString*. Свойства карты включают опцию *EnableZoom* с высотой и шириной 200 пикселей и родительской заливкой соответственно, а также опцию *ShowScale* (Показывать масштаб). Метки широты и долготы для отображения значений и кнопки для увеличения и уменьшения масштаба карты, а также для удаления маркеров с карты включены во вторую и третью области *HorizontalArrangements*. В *MIT App Inventor* метки (*labels*) предназначены только для отображения буквенно-цифрового текста, в то время как текстовые поля (*textboxes*) позволяют пользователю вводить текст. Свойства текстового поля подсказки *Hint* позволяют размещать в нем текст для предоставления информации пользователю. Подсказка для текстового поля *Location* – "*enter name then click*" («введите имя, затем щелкните»). Кнопки *AddLocation*, *ChooseLocation* и *DeleteLocation* нажимаются для обновления информации о местоположении в базе данных приложения. Наконец, кнопки *RoadView* и *AerialView* изменяют масштаб просмотра карты, хотя высокое разрешение аэрофотосъемки доступно только для США[60].

[60] Аэрофотосъемка в настоящее время применяется только для уточнения карт и панорам при необходимости особо высокого разрешения, поэтому автор, очевидно,

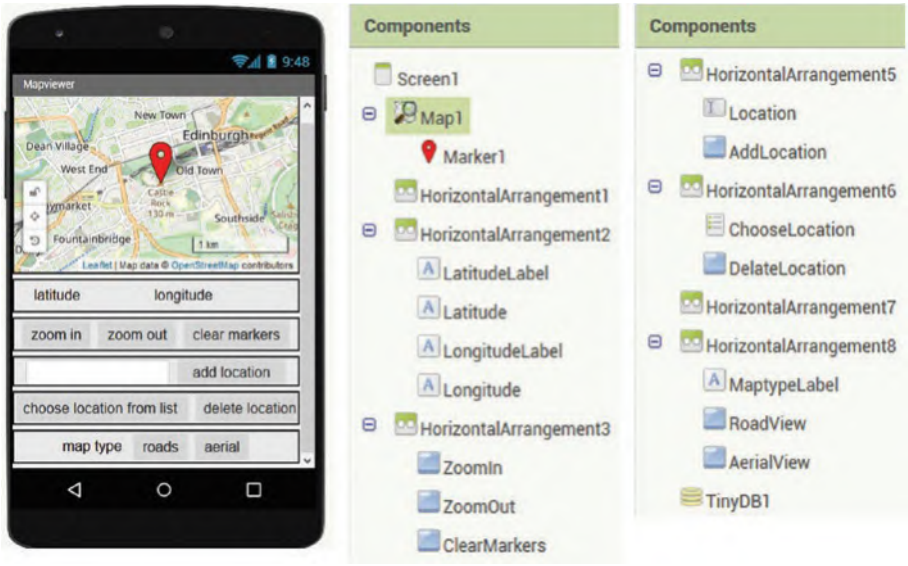


Рисунок 11-9. Макет приложения базы данных местоположений

Блоки для отображения широты (*latitude*) и долготы (*longitude*) местоположения при нажатии или увеличения масштаба при двойном нажатии на карту показаны на рис. 11-10. Процедура *setText* обновляет текстовые поля широты и долготы для позиции карты, к которой прикоснулся пользователь. Глобальная переменная *zoom* означает уровень масштабирования карты. Процедура *drawMap* перерисовывает карту с обновленным уровнем масштабирования и центрирует карту в выбранном местоположении. Блоки для изменения уровня масштабирования карты показаны на рис. 11-11, с процедурой масштабирования *zoom*, обновляющей уровень масштабирования.

подразумевает спутниковые снимки. Представление о том, что доступное разрешение карт и спутниковых снимков разнится в зависимости от региона пребывания, устарело и характерно для начала 2000-х, когда коммерческая спутниковая съемка делала первые шаги. В Google Maps разрешение карт и спутниковых снимков одинаково по всему миру и составляет 50 см и менее (в 2014 году Министерство торговли США разрешило торговлю снимками с разрешением до 25 см), причем в высоком разрешении представлены лишь самые интересные области поверхности земного шара (например, большие города). Что же касается спутниковых снимков военного назначения со сверхвысоким разрешением (по слухам, с орбиты можно даже распознавать автомобильные номера), то они засекречены всеми сторонами независимо от региона и их характеристики не уточняются. – *Прим. перев.*

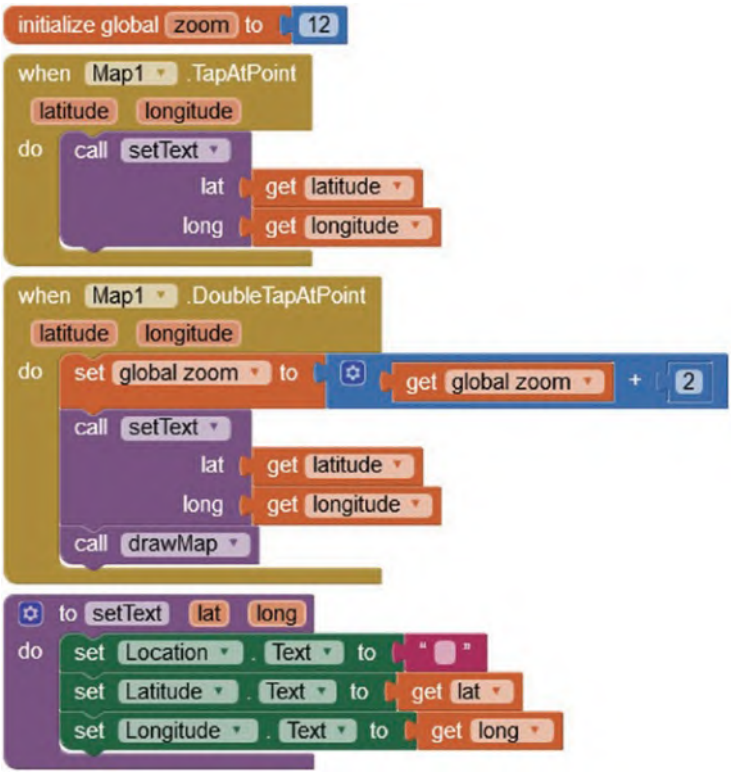


Рисунок 11-10. Отображение широты и долготы местоположения при нажатии на карте

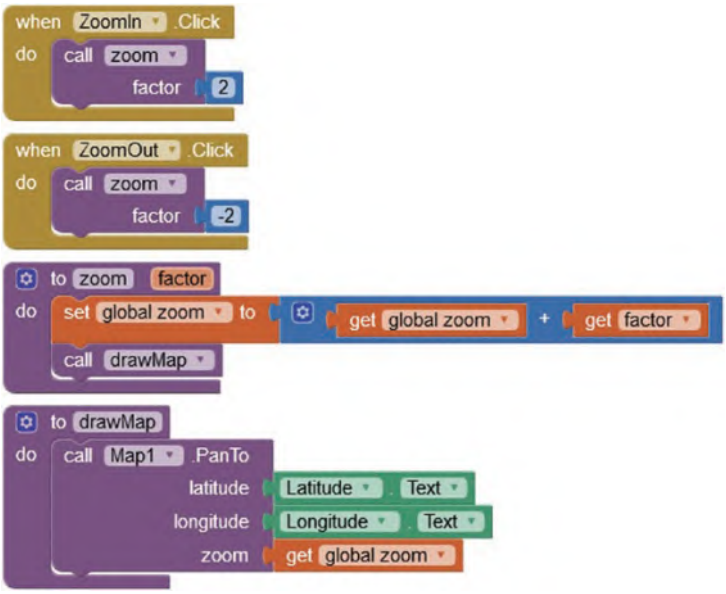


Рисунок 11-11. Изменение масштаба карты

Данные о местоположении сохраняются в базе данных с помощью TinyDB-функции *StoreValue* с тегом, представляющим собой наименование местоположения, введенное в текстовое поле *Location*. Значением базы данных являются широта (*latitude*) и долгота (*longitude*) местоположения, которые соединяются символом амперсанда (см. рис. 11-12). Местоположение удаляется из базы данных приложения с помощью TinyDB-функции *ClearTag*.

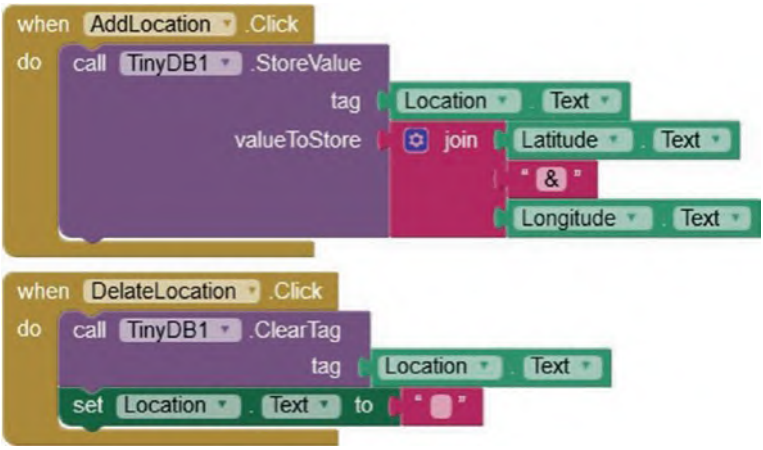


Рисунок 11-12. Сохранение данных о местоположении в базе данных приложения

Когда местоположение выбирается из списка, сохраненного в базе данных приложения, тегу базы данных присваивается выбранное местоположение. Соответствующее тегу значение, полученное с помощью TinyDB-функции *GetValue*, состоящее из широты и долготы местоположения, соединенных символом амперсанда, разбивается на два индексированных компонента (см. рис. 11-13). Тип карты устанавливается равным единице, что соответствует представлению *road view*[61], устанавливается уровень масштаба, и карта перерисовывается с центром на широте и долготе выбранного местоположения, с маркером карты, расположенным в выбранном местоположении. Кнопка *ClearMarkers* очищает список *Map Features*, удаляя все маркеры при следующем обновлении карты.

[61] Соответствует, очевидно, представлению карты на сайте Google «По умолчанию» («Схема» в Яндекс.Картах). *Aerial view* (см. далее) соответствует представлению «Спутник». – *Прим. перев.*

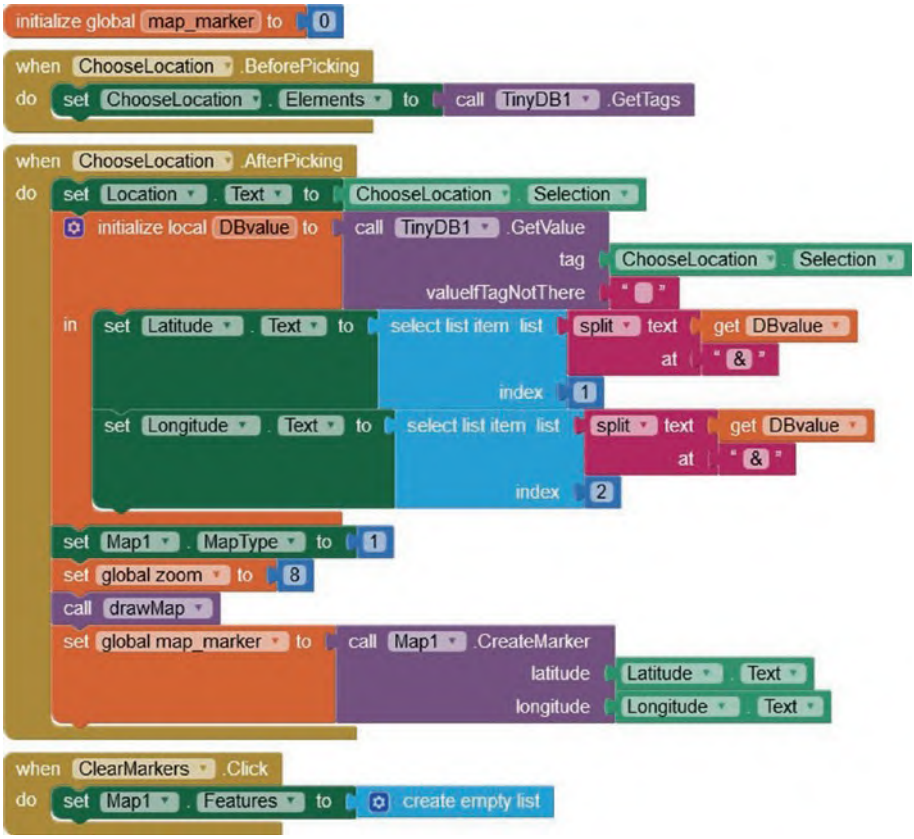


Рисунок 11-13. Выбор местоположения из базы данных приложения

Наконец, для выбора вида карты *road* или *aerial view* тип карты устанавливается равным одному или двум соответственно, и *road view* имеет более высокое разрешение, чем вид *aerial view* (см. рис. 11-14). Разрешение *aerial view* может быть недостаточно высокое для произвольного региона, поэтому перед перерисовкой карты уровень масштаба устанавливается на восемь.

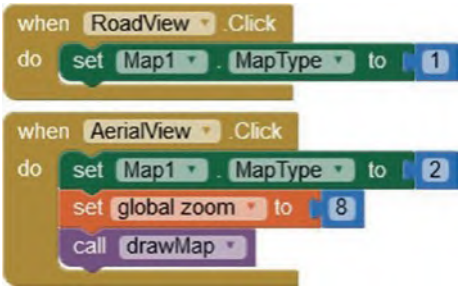


Рисунок 11-14. Установка типа карты

Итоги

Были разработаны демонстрационные приложения для иллюстрации функциональности базы данных *MIT App Inventor* и использования *Google Maps* для определения местоположения. В базе данных *MIT App Inventor* хранятся элементы, состоящие из тега и соответствующего значения, включающие несколько переменных, разделенных символом амперсанда. Элементы базы данных добавляются, изменяются или удаляются из базы данных с возможностью отображения всего содержимого или только значений, соответствующих выбранному тегу. Приложение базы данных, созданное с помощью *MIT App Inventor*, хранит данные о широте и долготе выбранных местоположений. На карте в сохраненных местоположениях отображаются маркеры. В приложении имеется функция масштабирования карты в видах *road* и *aerial views*, предоставляемых Google Maps.

Перечень компонентов

- Планшет или мобильный телефон на базе Android
- Доступ в интернет

Глава 12

Приложение для GPS-трекинга с использованием Google Maps

В главе 11 («Приложение базы данных и Google Maps») разработано приложение для отображения местоположений на карте с помощью *Google Maps* на планшете или мобильном телефоне Android с доступом в интернет. В этой главе приложение определения местоположения отображает местоположение удаленного GPS-модуля или маршрут, пройденный этим GPS-модулем с помощью *Google Maps*. Модуль nRF24L01, подключенный к плате ESP8266 или ESP32, снабженной GPS-модулем, передает местоположение на принимающий модуль nRF24L01, подключенный к плате ESP32. Информация о местоположении передается в приложение на планшете или мобильном телефоне Android микроконтроллером ESP32 с использованием Bluetooth-связи. Местоположение GPS отображается в приложении с помощью *Google Maps*, при этом мобильное устройство Android получает доступ к данным *Google Maps* из Всемирной паутины через интернет-провайдера (ISP) (см. рис. 12-1). Дальность передачи модуля приемопередатчика nRF24L01 не менее 1 км[62] в сочетании с дистанцией работы Bluetooth, равной 10 м, обеспечивает определенную гибкость в отношении расстояния между пользователем приложения, принимающим модулем nRF24L01 и удаленно расположенным GPS-модулем.

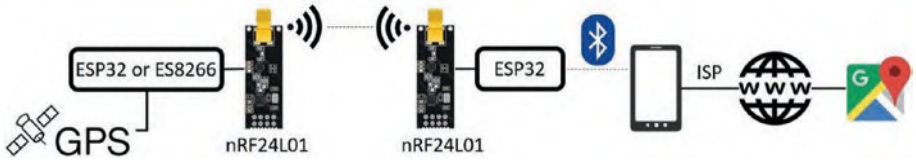


Рисунок 12-1. Приложение для GPS-трекинга с модулями приемопередатчиков nRF24L01

[62] Модуль nRF24L01 в своей приемо-передающей части аналогичен устройствам Wi-Fi (в частности, работает в том же ISM-диапазоне 2,4 ГГц), и дальность его работы в идеальных условиях прямой видимости не превышает 100 м. Дальность до 1 км может быть теоретически обеспечена при подключении к nRF24L01 дополнительного малошумящего усилителя и внешней антенны. – *Прим. перев.*

Принимающий модуль nRF24L01 может быть подключен к плате ESP8266, но для связи по Bluetooth с планшетом или мобильным телефоном Android к плате ESP8266 должен быть подключен модуль Bluetooth HC-05.

В качестве альтернативы можно заменить радиочастотную (RF) связь между передающей платой ESP8266 или ESP32, подключенной к модулю GPS, и принимающим микроконтроллером ESP32 с помощью модулей nRF24L01, на связь LoRa (long range). Связь LoRa описана в главе 14 («Обмен данными через ESP-NOW и LoRa»).

Приложение отображает текущее местоположение с функцией масштабирования карты и возможностью отображения завершенного маршрута с подключением маркеров отслеживания местоположения. Полученные с помощью GPS значения широты и долготы фильтруются, чтобы избежать нанесения на карту ошибочных положений маркеров, вызванных помехами при передаче. В приложении вводится начальная базовая позиция, с которой сравниваются полученные GPS-координаты (см. рис. 12-2).

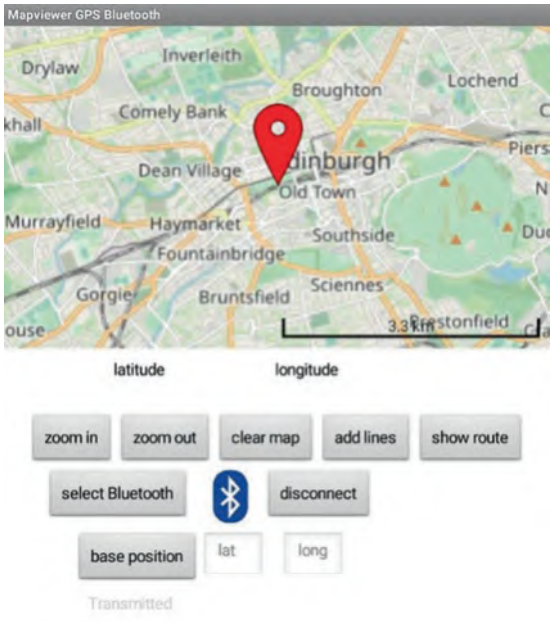


Рисунок 12-2. Приложение для GPS-трекинга

После установления Bluetooth-соединения между приложением и принимающим микроконтроллером ESP32 местоположение определяется на карте, при этом карта автоматически центрируется по показаниям GPS (см. рис. 12-3 (a)). Каждые две секунды местоположение передается на принимающий модуль nRF24L01, и карта обновляется с учетом показаний GPS (см. рис. 12-3 (b)). Нажатие кнопки *add lines* в приложении соединяет маркеры местоположения соединительной линией (см. рис. 12-3 (c)). Нажатие кнопки очистки *clear map* приложения очищает карту от GPS-маркеров и соединительной линии. При нажатии на кнопку *show route* (показать маршрут) в приложении отображается маршрут в виде линии соединения позиций GPS (см. рис. 12-3 (d)).

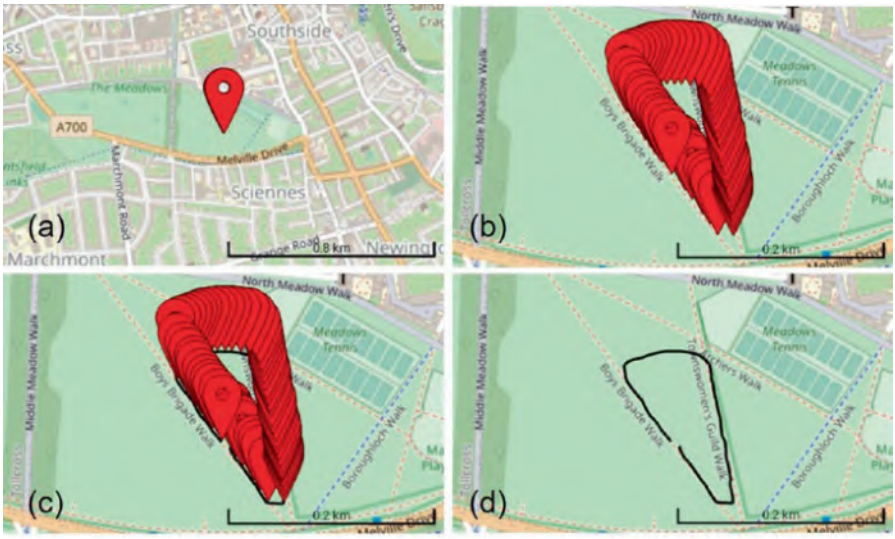


Рисунок 12-3. Приложение для отслеживания местоположения GPS-маркеров и пройденного маршрута

Макет приложения на рис. 12-4 состоит из карты с маркером местоположения и функции *LineString*, доступной в палитре *Maps* в левой части окна конструктора *MIT App Inventor*. Широта и долгота по GPS отображаются в соответствующих текстовых метках *Latitude* и *Longitude*. Вторая область *HorizontalArrangement* содержит кнопки *ZoomIn*, *ZoomOut*, *ClearMarkerLines*, *AddLines* и *ShowRoute*, которые управляют функцией масштабирования карты, удаляют все маркеры с карты, соединяют маркеры линией и отображают завершенный маршрут соединительной линией без маркеров.

Рисунок 12-4. Макет приложения для GPS-трекинга

Когда приложение загружено, при нажатии кнопки *SelectBluetooth ListPicker* отображается список доступных Bluetooth-подключений, выполняется выбранное подключение, а текст «*connected*» отображается серым цветом. Кнопка *DisconnectBluetooth* отключает соединение Bluetooth, и текст «*disconnected*» отображается красным цветом.

Ввод базовой широты и долготы в соответствующие текстовые поля *BaseLatitude* и *BaseLongitude*, а затем нажатие кнопки *BasePosition* показывает введенные значения, и потом приложение отображает полученные значения широты и долготы рядом с маркером на карте Google Maps. Последняя область *HorizontalArrangement* отображает сигнал *TransmittedLocation*, принятый модулем nRF24L01. Компоненты *Clock*, *TinyDB* и *BluetoothClient* расположены в палитрах *Sensors*, *Storage* и *Connectivity* окна конструктора.

Блоки *MIT App Inventor* для создания приложения показаны на рис. 12-5–12-10. Блоки для подключения и отключения к Bluetooth устройства, которым является микроконтроллер ESP32, показаны на рис. 12-5. Метка *StatusLabel* отображает состояние подключения Bluetooth. Построение блоков подключения Bluetooth было описано в главе 10 («Создаем мобильное приложение»).

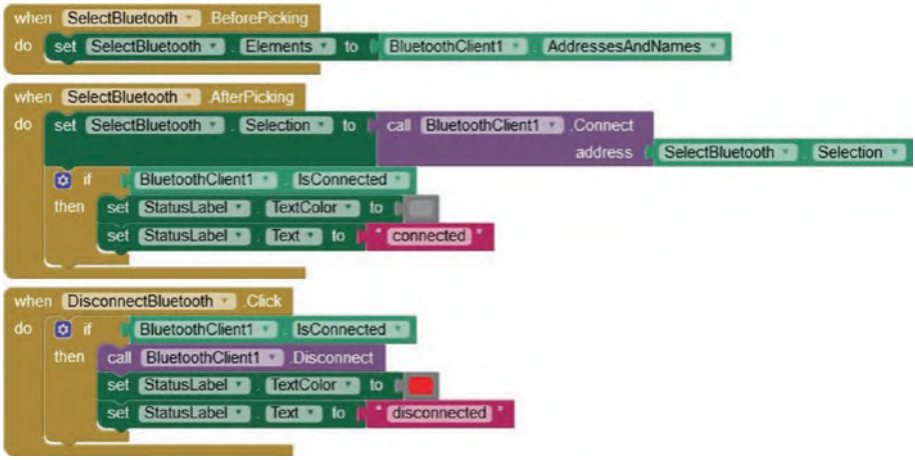


Рисунок 12-5. Подключение или отключение устройства к Bluetooth

Таймер *Clock1* с интервалом 1000 мс отслеживает устройство Bluetooth на предмет сигнала, аналогичного инструкции последовательного порта `if(Serial.available() > 0)` из Arduino IDE. Принятый сигнал отображается в нижней части экрана приложения в метке *TransmittedLocation*. Принятый сигнал должен содержать три компонента, разделенных запятыми: счетчик, GPS-широту и GPS-долготу. Если принятый сигнал содержит запятую, то анализируется текст сигнала, при этом каждый компонент сигнала индексируется 1, 2 или 3 и распределяется по компонентам списка *datalist*. Если *datalist* содержит три компонента, затем вызывается процедура проверки информации о сигнале (см. рис. 12-6). Если принятый сигнал не содержит запятой или не содержит трех компонентов, то сигнал игнорируется.

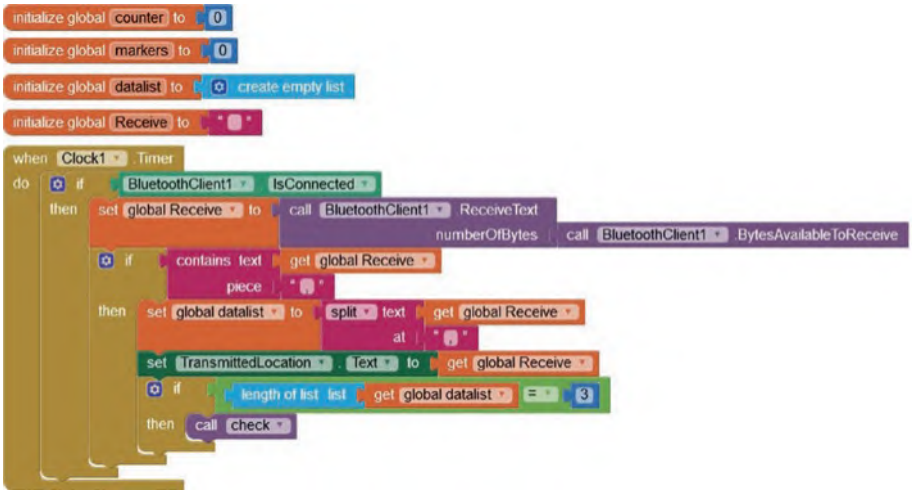


Рисунок 12-6. Прием и обработка Bluetooth-сигнала

Процедура проверки проверяет информацию о сигнале, устанавливает, что широта и долгота являются числами и что абсолютное отклонение от базового положения, которое было введено пользователем в приложении, составляет менее двух градусов (см. рис. 12-7). Один градус широты составляет 111 км, в то время как градус долготы составляет 78 км при 45° северной или южной широты, учитывая сферический радиус Земли (6371 км). Сравнение полученной широты и долготы с базовым положением гарантирует, что ложные положения из-за помех при передаче не будут нанесены на карту. Местоположение обновляется на карте с созданием маркера по полученной широте и долготе, при этом карта центрируется по новому местоположению маркера. Местоположение сохраняется в базе данных TinyDB с тегом, равным счетчику маркеров, который увеличивается только для проверенных данных о местоположении, а не для всех полученных данных.

Блоки для создания линий соединения маркеров положения показаны на рис. 12-8. Текст *linetext* по сути представляет собой последовательность координат (X, Y). При нажатии кнопки *AddLines* текст *linetext* инициализируется как пустой символ и заполняется с каждой GPS-позицией, сохраненной в базе данных TinyDB для формирования последовательности координат (X, Y). Когда все данные о местоположении включены в текст *linetext*, точки на *Map LineString* генерируются из текста *linetext*, и *Map LineString* становится видимой на *Google Maps*. *Map LineString* также делается видимой, но без маркеров карты, когда нажата кнопка *ShowRoute*, после нажатия обеих кнопок – *AddLines*, а затем *ClearMarkerLines*.

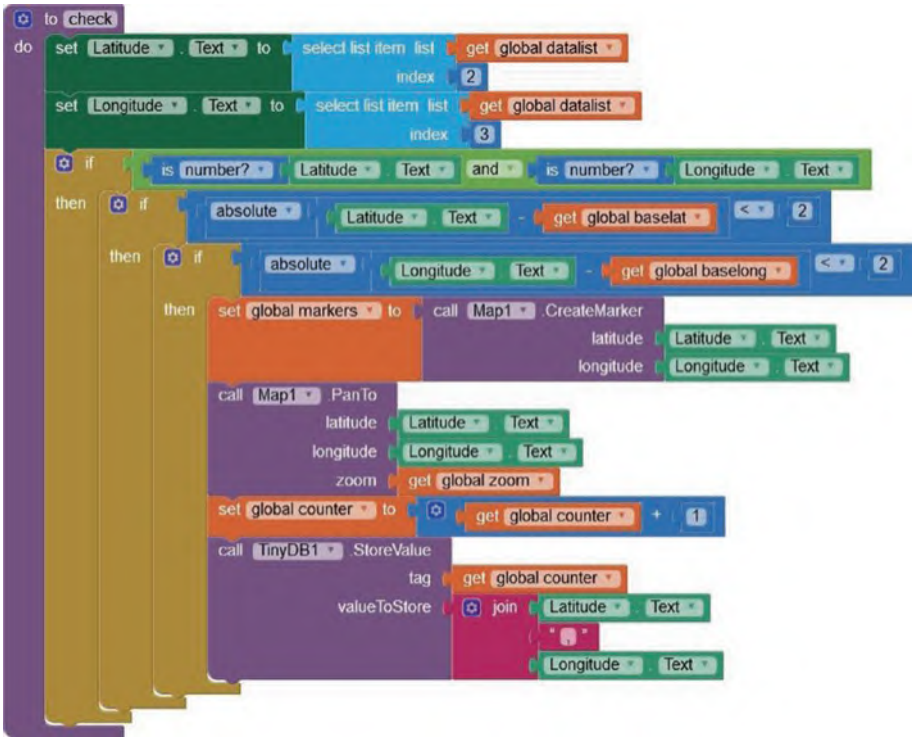


Рисунок 12-7. Проверка данных о местоположении, нанесение местоположения на карту и сохранение данных

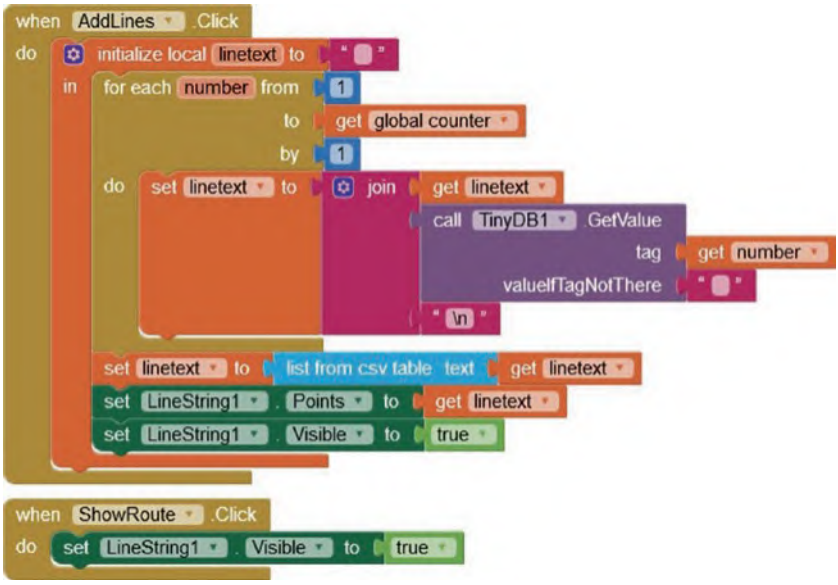


Рисунок 12-8. Создание маркерных соединительных линий

Блоки для очистки маркеров карты и/или линии соединения маркеров карты показаны на рис. 12-9. При нажатии кнопки *ClearMarkerLines* строка линий карты больше не отображается, отображаемые широта и долгота очищаются, а все объекты карты, такие как маркеры карты, сбрасываются, за исключением первого элемента *Map Features*.

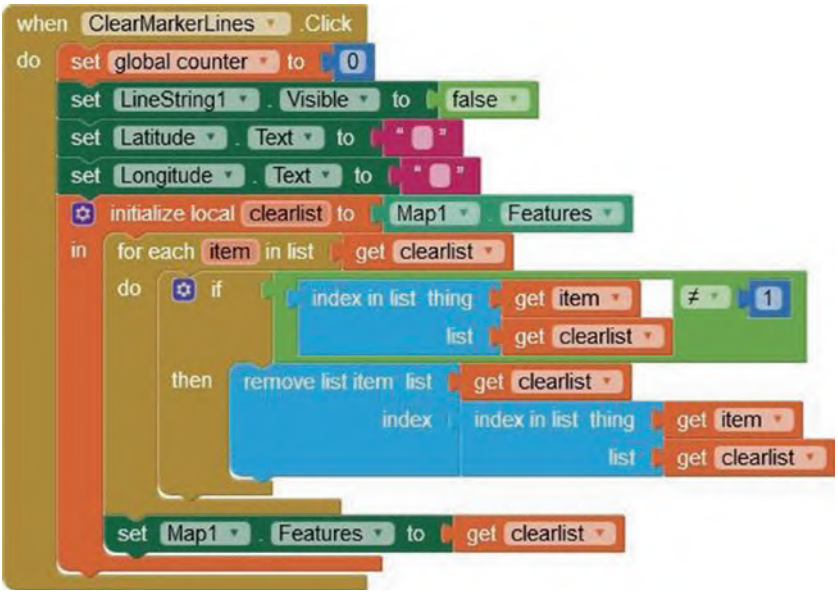


Рисунок 12-9. Очистка маркеров карты и соединительных линий

Блоки для функций масштабирования карты и включения информации о базовом местоположении показаны на рис. 12-10. При нажатии кнопки увеличения или уменьшения масштаба коэффициент масштабирования карты увеличивается или уменьшается, карта перерисовывается с обновленным масштабом и центрируется в последнем GPS-местоположении. При нажатии кнопки *BasePosition* значения, введенные пользователем в текстовых полях *BaseLatitude* и *BaseLongitude*, сохраняются в глобальных переменных *baselat* и *baselong* для проверки полученной информации о местоположении и помещения в *BaseLabel* обновленных значений переменных *baselat* и *baselong*.

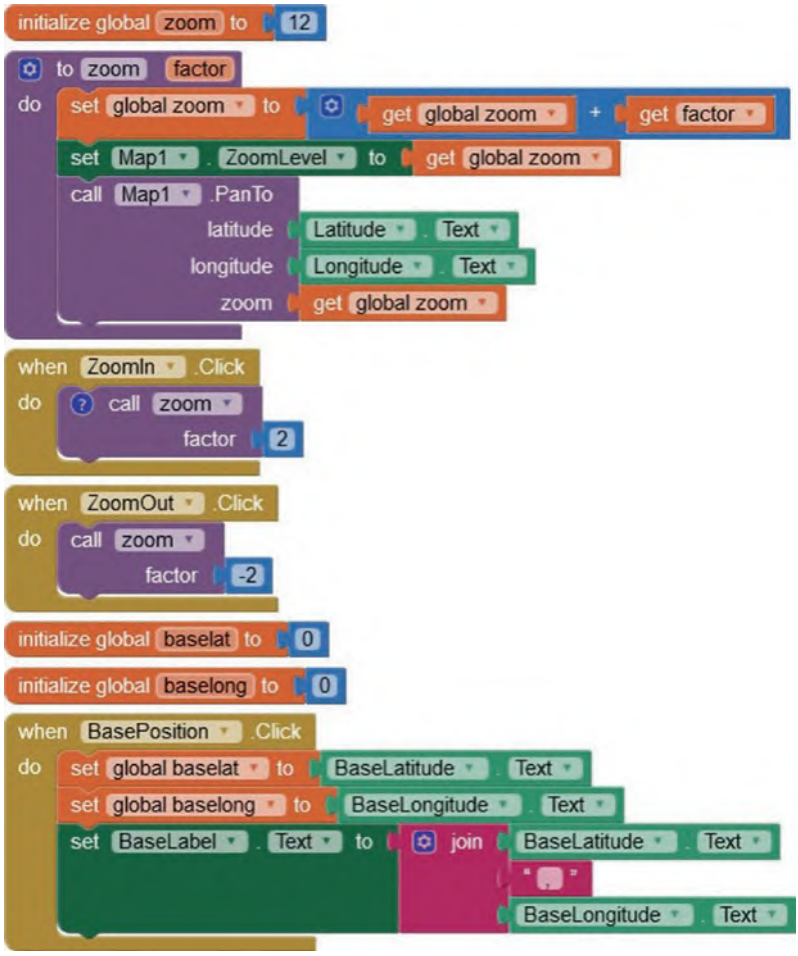


Рисунок 12-10. Функции масштабирования карты и обновления значений базового положения

Передача GPS-данных о местоположении

Информация о местоположении передается с помощью модуля приемопередатчика nRF24L01, подключенного к плате ESP8266 или ESP32. Контакты модуля nRF24L01 показаны на рис. 12-11, а вывод GND обведен квадратиком. Модуль nRF24L01 взаимодействует через интерфейс SPI, с выводами MOSI, MISO и SCK, подключенными к SPI-контактам микроконтроллера. Контакты CE (передача/прием) и CSN (режим ожидания / активный режим) не предусматривают определенного вывода микроконтроллера. Модуль nRF24L01 работает при напряжении 3,3 В, но логические выводы допускают напряжение 5 В.

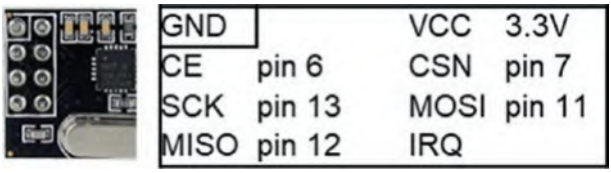


Рисунок 12-11. Разводка выводов nRF24L01

Данные о местоположении с GPS-модуля u-blox NEO-7M передаются микроконтроллером ESP8266 или ESP32, питаемым от источника питания 5 В.

Соединения передающего модуля nRF24L01 с GPS-модулем NEO-7M и платами ESP8266 или ESP32, а также принимающего модуля nRF24L01 с платой ESP32 показаны на рис. 12-12 и в табл. 12-1. Конденсатор емкостью 10 мкФ уменьшает шум в сигнале от источника питания. К микроконтроллеру подключен только вывод передачи GPS-модуля NEO-7M, так как вывод приема модуля не используется.

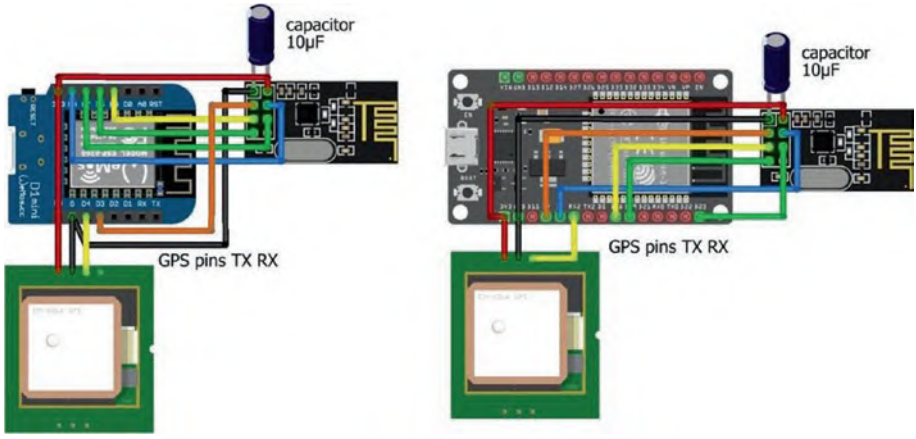


Рисунок 12-12. Передающий модуль nRF24L01, GPS-модуль u-blox NEO-7M и модуль приемопередатчика nRF24L01 с платами LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT

Таблица 12-1. Соединения для GPS-модулей nRF24L01 и u-blox NEO-7M с платами ESP8266 и ESP32

Компонент	Плата ESP8266	Плата ESP32
nRF24L01 VCC	ESP8266 3V3	ESP32 3V3
nRF24L01 CSN	ESP8266 D8	ESP32 GPIO 4
nRF24L01 MOSI	ESP8266 D7	ESP32 GPIO 23
nRF24L01 IRQ	Не подключен	Не подключен
nRF24L01 GND	ESP8266 GND	ESP32 GND
nRF24L01 CE	ESP8266 D3	ESP32 GPIO 2
nRF24L01 SCK	ESP8266 D5	ESP32 GPIO 18

Окончание табл. 12-1

Компонент	Плата ESP8266	Плата ESP32
nRF24L01 MISO	ESP8266 D6	ESP32 GPIO 19
u-blox NEO-7M VCC	ESP8266 3V3	ESP32 3V3
u-blox NEO-7M GND	ESP8266 GND	ESP32 GND
u-blox NEO-7M TXD	ESP8266 D4	ESP32 RX2 (GPIO 16)
Конденсатор 10 мкФ, положительный вывод	nRF24L01 VCC	nRF24L01 VCC
Конденсатор 10 мкФ, отрицательный вывод	nRF24L01 GND	nRF24L01 GND

Инструкции для передающего модуля nRF24L01, подключенного к GPS-модулю u-blox NEO-7M и плате ESP8266, приведены в листинге 12-1. NMEA-сообщения[63] из GPS-модуля u-blox NEO-7M извлекаются с помощью библиотеки NeoGPS, которая использует библиотеку *AltSoftSerial* для последовательного порта. Библиотека *AltSoftSerial* несовместима с микроконтроллером ESP8266, поэтому в этом случае используется библиотека *SoftwareSerial*. Библиотека *NeoGPS* Слэша Девина (Slash Devin) и библиотека *RF24* Джеймса Колиза (James Coliz) доступны в среде Arduino IDE наряду со встроенной библиотекой *SoftwareSerial*.

30-выводная плата ESP32 DEVKIT DOIT и 36-выводная плата ESP32 NodeMCU имеют второй последовательный порт на GPIO 16 (RX2) и GPIO 17 (TX2).

Листинг 12-1 предназначен для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 следующие инструкции для микроконтроллера ESP8266:

```
#include <SoftwareSerial.h>          // include SoftwareSerial library
SoftwareSerial SoftSer(D4, D0);
                                     // associate SoftSer with SoftwareSerial
RF24 radio(D3, D8);                  // associate radio with RF24 library
SoftSer.begin(9600);                 // SoftwareSerial baud rate
while(nmea.available(SoftSer)>0)     // GPS data available
```

заменяются инструкциями для микроконтроллера ESP32:

```
RF24 radio(2, 4);                          // associate radio with RF24 library
Serial2.begin(9600, SERIAL_8N1, 1 6, 17);  // ESP32 RX2 on GPIO 16
while(nmea.available(Serial2)>0)           // GPS data available
```

GPS-модуль u-blox NEO-7M ежесекундно передает сообщения NMEA. Сообщение RMC (см. сноску 63) содержит широту и долготу. В библиотеке NeoGPS откройте файл *NMEAGPS_cfg.h* в папке *src* и убедитесь, что инструкция `#define`

[63] NMEA (*National Marine Electronics Association*) – стандарт, определяющий текстовый протокол связи для различного оборудования (как правило, навигационного), находящегося на подвижных объектах (судах, поездах, самолетах). Приобрел особую популярность в связи с распространением использующих этот стандарт GPS-приемников. Содержит ряд предопределенных сообщений-строк, начинающихся с идентификатора сообщения, маркируемого символом «$» и содержащего в том числе характеристику формата строки. Например, упоминаемый далее формат RMC (Recommended Minimum Navigation Information, минимальный рекомендованный набор данных) содержит важнейшие навигационные параметры, включая точное время, координаты, скорость движения и т. д. – *Прим. перев.*

NMEAGPS_PARSE_RMC в строке 38 не закомментирована. Если требуется только сообщение RMC, то инструкция в строке 48 должна быть равна #define LAST_SENTENCE_IN_INTERVAL NMEAGPS::NMEA_RMC. Информация о структуре NMEA-сообщений доступна на github.com/SlashDevin/NeoGPS.

Значения широты и долготы GPS преобразуются в строки, которые конвертируются в массивы символов для включения в структуру данных, передаваемую модулем nRF24L01. Счетчик GPSsend увеличивается при получении действительных данных о местоположении с условием для значения счетчика, зависящего от интервала между передачами модуля nRF24L01. Подробная информация об инструкциях библиотеки *RF24* представлена в листинге 12-2.

Листинг 12-1. nRF24L01 передает данные о местоположении с помощью платы ESP8266

```
#include <SoftwareSerial.h>          // include SoftwareSerial library
SoftwareSerial SoftSer(D4, D 0);
                                     // associate SoftSer with SoftwareSerial
#include <NMEAGPS.h>                 // include NeoGPS library
NMEAGPS nmea;                        // associate nmea and gps
gps_fix gps;                         // with NMEAGPS library
float GPSlat, GPSlong;               // real numbers for GPS location
int GPSsend = 0;                     // GPS send counter
#include <SPI.h>                     // include SPI library
#include <RF24.h>                    // include RF24 library
RF24 radio(D3, D8);                  // associate radio with RF24 library
byte addresses[ ][6] = {"12" };      // data pipe address
typedef struct                       // define data structure to include
{
  char GPSlat[10];                   // character arrays for
  char GPSlong[10];                  // GPS latitude and longitude
} dataStruct;
dataStruct data;                     // name the data structure as data
int interval = 2;                    // interval (s) between GPS transmissions
void setup()
{                                    // Serial connection to GPS module
  SoftSer.begin(9600);                      // SoftwareSerial baud rate
  delay(500);
  radio.begin();                     // start radio
  radio.setChannel(50);              // set channel number,
  radio.setDataRate(RF24_2MBPS);     // baud rate
  radio.setPALevel(RF24_PA_HIGH);     // and power amplifier
  radio.setAutoAck(true);            // set auto-acknowledge
  (default)
  radio.openWritingPipe(address es[0]);
                                     // initiate data transmit pipe
  radio.stopListening();             // nRF24L01 as transmitter
}
void loop()
{
  while(nmea.available(SoftSer)>0)   // GPS data available
  {
    gps = nmea.read();               // latest satellite message
    if(gps.valid.location)           // validated GPS location
    {
      GPSlat = gps.latitude();
      GPSlong = gps.longitude();
```

```
        GPSsend++;                    // increment GPS send counter
      }
      if(GPSsend > interval)          // transmit every (interval+1)s
      {               // convert number to string and then to character array
        String(GPSlat,6).toCharArray(data.GPSlat,10);
        String(GPSlong,6).toCharArray(data.GPSlong,10);
        radio.write(&data, siz eof(data));
                                      // transmit signal as data structure
        GPSsend = 0;                  // reset GPS send counter
      }
    }
  }
```

Получение GPS-данных о местоположении

Инструкции для приемного модуля nRF24L01, подключенного к плате ESP32, приведены в листинге 12-2. Библиотека *RF24* Джеймса Колиза (James Coliz) доступна в среде Arduino IDE. Связь между модулями приемопередатчика nRF24L01 осуществляется по каналам передачи данных, для которых требуется строковый пятибайтовый адрес, такой как "*node1*" для каждого канала передачи данных, номер канала передачи, скорость передачи данных и уровень усилителя мощности. Модуль nRF24L01 работает на частотах 2,4 ГГц со 126 каналами с полосой пропускания менее 1 МГц[64], что обеспечивает диапазон частот от 2,4 ГГц (2400 МГц) до 2525 МГц, соответствующих номерам каналов 0–125. Номер канала *N* задается с помощью команды `setChannel(N)`. Скорости передачи данных 250 Кбит/с, 1 Мбит/с и 2 Мбит/с, измеряемые в битах в секунду, доступны в библиотеке RF24 при использовании команды `setDataRate()` со значениями `RF24_250KBPS`, `RF24_1MBPS` и `RF24_2MBPS`. Уровни усилителя мощности `RF24_PA_MIN`, `LOW`, `HIGH` и `MAX` равны –18, –12, –6 и 0 дБ, что соответствует выходной мощности приблизительно 1/64, 1/16, 1/4 и 1 МВт соответственно, поскольку мощность = $10^{(дБ/10)}$ МВт. Уровни усилителя мощности устанавливаются с помощью команды `setPALevel()`.

Структура данных объединяет несколько типов данных, но имеет ограничение в 32 байта с целым числом, действительным числом или символом, требующим 2, 4 или 1 байта соответственно. Принятый сигнал nRF24L01, содержащий данные о местоположении, объединяется со значением счетчика и передается в виде текстовой строки в приложение определения местоположения.

Листинг 12-2. Трансляция через nRF24L01 сигнала, передаваемого по Bluetooth

```
#include <BluetoothSerial.h>          // include Bluetooth library
BluetoothSerial SerialBT;             // associate SerialBT with library
#include <SPI.h>                      // include SPI library
#include <RF24.h>                     // include RF24 library
RF24 radio(2, 4);                     // associate radio with RF24 lib
byte addresses[ ][6] = {"12"};        // data pipe address
```

[64] В модуле nRF24L01 каждый канал занимает полосу частот менее 1 МГц при скорости передачи 250 Кбит/с и 1 МГц на скорости передачи 1 Мбит/с. Для исключения взаимовлияния каналов рекомендованное расстояние между центральными частотами должно составлять не менее 2 МГц. Некоторые подробности о nRF24L01 можно посмотреть на русском языке по адресу https://micro-pi.ru/nrf24l01-spi-модуль-беспроводной-связи/. – *Прим. перев.*

```
typedef struct                          // define data structure to include
{
   char GPSlat[10];                     // character arrays for
   char GPSlong[10];                    // GPS latitude and longitude
} dataStruct;
dataStruct data;                        // name the data structure as data
int count = 0;                          // received message counter
int textLen;
String text;
char c;
void setup()
{
   radio.begin();                       // start radio
   radio.setChannel(50);                // set channel number
   radio.setDataRate(RF24_2MBPS);       // baud rate
   radio.setPALevel(RF24_PA_HIGH);      // and power amplifier
   radio.setAutoAck(true);              // set auto-acknowledge
   (default)
   radio.openReadingPipe(0, address es[0]);
   // initiate data receive pipe
   radio.startListening();              // nRF24L01 module as receiver
   SerialBT.begin("ESP32 Bluetooth" ); // identify Bluetooth
}
void loop()
{
   if(radio.available())                // if signal received
   {                                    // received signal to data structure
      radio.read(&data, sizeof(data) );
      count++;                          // increment counter
      text = String(count) + "," + String(data.GPSlat) + "," +
      String(data.GPSlong) + ",";       // build string of position data
      textLen = text.length();
      for (int i=0; i<textLen; i++)
      {
         c = text[i];                   // for each message character
         SerialBT.write(c);             // transmit to Bluetooth device
      }
   }
}
```

Мониторинг принимающего модуля nRF24L01 осуществляется путем отображения полученных данных на мониторе последовательного порта с помощью инструкций

```
Serial.begin(115200);                  // Serial Monitor baud rate
Serial.println(text);                  // display GPS position data
```

Проверка передачи GPS-данных о местоположении

Функциональность Wi-Fi микроконтроллеров ESP8266 и ESP32 позволяет отображать данные на веб-странице в дополнение к монитору последовательного порта. При проверке разработки приложения для трекинга полученные данные о GPS-местоположении отображаются на веб-странице с помощью планшета Android или мобильного телефона, подключенного к беспроводной локальной сети (WLAN). Передача GPS-данных о местоположении проверяется изменени-

ем местоположения в радиусе подключения к беспроводной сети в несколько метров.

Создание и обновление веб-страницы описано в главе 8 («Обновление веб-страницы»). Инструкции по обновлению веб-страницы с данными о местоположении объединены с листингом 12-2, как показано в листинге 12-3. В скетче представлены различные временные интервалы для обновления, отображения и передачи данных о местоположении. Данные о местоположении с модуля Neo-7M автоматически обновляются каждую секунду, что отображается на мониторе последовательного порта. Веб-страница обновляет счетчик каждую секунду, но веб-страница с данными местоположения обновляется каждые пять секунд. Приемопередатчик nRF24L01 передает данные о местоположении GPS каждые три секунды. В листинге 12-2 широта и долгота GPS были преобразованы в строки, затем в массивы символов инструкцией `String(GPSlat,6).toCharArray(data.GPSlat,10)` и отправляются в виде текстовой строки в приложение. В листинге 12-3 широта и долгота объявляются в структуре данных `data` и передаются непосредственно с помощью команды `radio.write(&data,sizeof(data))`. Функция `flashLED` обеспечивает двойное мигание светодиода при передаче данных о местоположении в качестве индикатора того, что комбинация микроконтроллера ESP8266 или ESP32 и GPS-модуля NEO-7M работает.

Листинг 12-3 предназначен для микроконтроллера ESP8266. Если передающий модуль nRF24L01 подключен к плате ESP32, то библиотеки *WiFi*, *WebServer* и *SoftwareSerial*, контакты модуля nRF24L01 *CE* и *CSN*, а также инструкции задания вывода светодиода для микроконтроллера ESP8266

```
#include <ESP8266WiFi.h>              // include ESP8266 Wi-Fi
#include <ESP8266WebServer.h>         // web server libraries
ESP8266WebServer server;              // associate server with library
#include <SoftwareSerial.h>           // include SoftwareSerial library
SoftwareSerial SoftSer(D4, D0) ;
                                      // associate SoftSer with SoftwareSerial
SoftSer.begin(9600);                  // serial connection to GPS module
while(nmea.available(SoftSer)>0)      // GPS data available
RF24 radio(D3, D8);                   // associate radio with RF24 library
int LEDpin = D1;                      // define LED
```

заменяются следующими инструкциями для микроконтроллера ESP32:

```
#include <WiFi.h>     // include ESP8266 Wi-Fi
and
#include <WebServer.h>                // web server libraries
WebServer server (80);                // requires a port number
Serial2.begin(9600, SERIAL_8N1, 16, 17);
                                      // ESP32 RX2, TX2 on GPIO 16 and 17
while(nmea.available(Serial2)>0)      // GPS data available
RF24 radio(2, 4);                     // associate radio with RF24 library
int LEDpin = D1;                      // define LED pin
```

Листинг 12-3. Отображение на веб-странице данных о местоположении, передаваемых nRF24L01

```
#include <ESP8266WiFi.h>              // include ESP8266 Wi-Fi and
#include <ESP8266WebServer.h>         // web server libraries
ESP8266WebServer server;              // associate server with library
char ssid[] = "xxxx";                 // change xxxx to your Wi-Fi SSID
```

```
char password[] = "xxxx";              // change xxxx to your Wi-Fi password
#include "buildpage.h"                 // webpage HTML code
#include <SoftwareSerial.h>            // include SoftwareSerial library
SoftwareSerial SoftSer(D4, D0);
                                       // associate SoftSer with SoftwareSerial
#include <NMEAGPS.h>                   // include NeoGPS library
NMEAGPS nmea;                          // associate nmea with NMEAGPS lib
gps_fix gps;                           // associate gps with NMEAGPS library
float GPSlat, GPSlong, GPSalt, GPSs pd;
                                       // real numbers for GPS location
int GPSsend = 0;                       // GPS send counter
String json;
int count = 0;
String counter;                        // counter increment every second
#include <SPI.h>                       // include SPI library
#include <RF24.h>                      // include RF24 library
RF24 radio(D3, D8);                    // associate radio with RF24 library
byte addresses[ ][6] = {"12"};         // data pipe address
typedef struct                         // define data structure to include
{
  float GPSlat;                        // GPS latitude
  float GPSlong;                       // GPS longitude
  float GPSalt;                        // GPS altitude (m)
  float GPSspd;                        // GPS ground speed (kmph)
  int sigCount;                        // signal counter
} dataStruct;
dataStruct data;                       // name the data structure as data
int LEDpin = D1;                       // define LED pin
int LED = 0;                           // LED turned off
int interval = 1;                      // (interval+1)s between transmissions
void setup()
{
  Serial.begin(115200);                // define Serial output baud rate
  SoftSer.begin(9600);                 // serial connection to GPS module
  WiFi.begin(ssid, password);          // initialise and connect Wi-Fi
  while (WiFi.status() != WL_C ONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP( ));     // display web server IP address
  server.begin();                      // initialise server
  server.on("/", base);                // call base function as webpage loaded
  server.on("/GPSurl", GPSfunct);
                                       // call GPSfunct with GPSurl loaded
  server.on("/countUrl", countFunct);
  delay(500);
  radio.begin();                       // start radio
  radio.setChannel(50);                // set channel number,
  radio.setDataRate(RF24_2MBPS );      // baud rate and
  radio.setPALevel(RF24_PA_HIG H);     // power amplifier level
  radio.setAutoAck(true);              // set auto-acknowledge (default is true)
  radio.openWritingPipe(addresses[0]);
                                       // initiate data transmit pipe
  radio.stopListening();               // set nRF24L01 module as transmitter
  pinMode(LEDpin, OUTPUT);             // define LEDpin as OUTPUT
}
void GPSfunct()                        // function to transmit GPS position data
{
```

```
  while(nmea.available(SoftSer )>0)  // GPS data available
  {
    gps = nmea.read();               // latest satellite message
    if(gps.valid.location)           // validated GPS location
    {
      GPSlat = gps.latitude();
      GPSlong = gps.longitude();
      GPSsend++;                     // increment GPS send counter
    }
    if(gps.valid.altitude) GPSalt = gps.altitude( ); // altitude
    if(gps.valid.speed) GPSspd = gps.speed_kph();
                                     // ground speed
    JsonConvert(GPSlat, GPSlon g, GPSalt, GPS spd);
                                     // convert to JSON text
    server.send(200, "text/json", json);
                                     // send JSON text to client
    Serial.println(json);
    if(GPSsend > interval)           // transmit every (interval+1)s
    {
      data.GPSlat = GPSlat;
                                     // convert GPS readings to data structure
      data.GPSlong = GPSlong;
      data.GPSalt = GPSalt;
      data.GPSspd = GPSspd;
      data.sigCount++;               // increment signal counter
      radio.write(&data, sizeof(da ta));
                                     // transmit signal as data structure
      GPSsend = 0;                   // reset GPS send counter
      flashLED();
    }
  }
}
                                     // function to convert data to JSON text
String JsonConvert(float val1, float val2, float val3, float val4)
{                                    // start with open bracket
  json = "{\"var1\": \"" + String(val1,4) + "\",";
  json += " \"var2\": \"" + String(val2, 4) + "\",";
                                     // end with comma
  json += " \"var3\": \"" + String(val3) + "\",";
  json += " \"var4\": \"" + String(val4) + "\"}";
                                     // end with close bracket
  return json;
}
void countFunct()                    // function to increment counter
{                                    // and send value to client
  count++;
  counter = String(count);
  server.send (200, "text/plain", counter);
}
void flashLED()                      // function to flash LED
{
  for (int i=0; i<4; i++)
  {
    LED = 1 - LED;                   // alternate LED state four times
    digitalWrite(LEDpin, LED);       // ON – OFF – ON - OFF
    delay(50);
  }
}
```

```
void base()                    // function to return HTML code
{
  server.send (200, "text/html", page);
}
void loop()
{
  GPSfunct();                  // function to transmit GPS location data
  server.handleClient();       // manage HTTP requests
}
```

Листинг 12-4 содержит код AJAX для веб-страницы и HTTP-запросов, определяемый как символьная строка `page`. Первая часть раздела `<body>` представляет собой HTML-код веб-страницы с переменными, на которые ссылаются по их имени в HTTP-запросе. Инструкция JavaScript `setInterval()` управляет временным интервалом между HTTP-запросами, составляющим пять секунд для функции `reload` при получении данных о местоположении. Скрипты JavaScript, обозначенные тегами `<script>...</script>`, располагаются перед закрывающим HTML-тегом `</body>` для повышения скорости отображения веб-страницы.

Листинг 12-4. AJAX-запрос данных о местоположении

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head><title>ESP8266</title>
<meta charset="UTF-8">
</head>
<body>
<h2>GPS</h2>
<p>Latitude: <span id = 'latId'>0</span></p>
<p>Longitude: <span id = 'longId'>0</span></p>
<p>Altitude: <span id = 'altId'>0</span> m</p>
<p>Speed: <span id = 'speedId'>0</span> kph<p>
<p>Counter: <span id = 'countId'>0</span></p>
<script>
setInterval(reload, 5000);    // reload function called every 5s
function reload()
{
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function()
  {
  if(this.readyState == 4 && this.status == 200)
  {
    var obj = JSON.parse(this.responseText);
    document.getElementById('latId').innerHTML = obj.var1;
    document.getElementById('longId').innerHTML = obj.var2;
    document.getElementById('altId').innerHTML = obj.var3;
    document.getElementById('speedId').innerHTML = obj.var4;
    console.log(obj.var3);
  }
};
xhr.open('GET', '/GPSurl', true);
xhr.send();
}
setInterval(reload2, 1000);
function reload2()            // reload2 function called every 1s
```

```
{                              // to obtain countId from /countUrl
  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function()
  {
    if(this.readyState == 4 && this.status == 200)
    {document.getElementById('countId').innerHTML = this.
    responseText;
    console.log(this.responseText);}
  };
  xhr.open('GET', '/countUrl', true);
  xhr.send();
}
</script>
</body></html>
)";
```

В дополнение к тестированию обновления передаваемых данных о местоположении при изменении положения модуля GPS для тестирования обновления полученных данных при изменении положения принимающего модуля nRF24L01 требуется мобильный дисплей, такой как OLED-экран. Скетч в листинге 12-5 отображает данные о местоположении OLED-экрана с разрешением 128×64 пиксела (см. рис. 12-13). Соединения для плат ESP8266 с передающими и принимающими модулями nRF24L01 приведены в табл. 12-2. Библиотека *Adafruit_SSD1306* для OLED-экрана доступна в среде Arduino IDE.

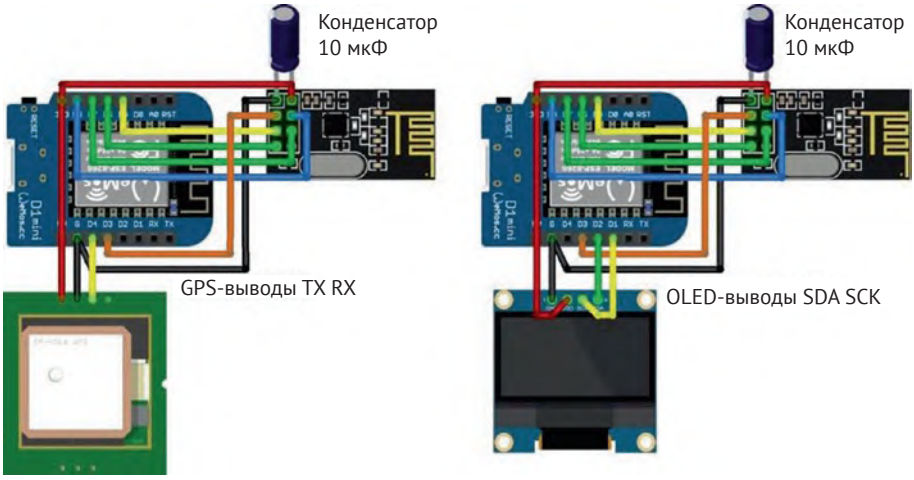


Рисунок 12-13. Передача и прием сигналов GPS с помощью модулей nRF24L01

Таблица 12-2. Подключения платы ESP8266 к передающим и принимающим модулям nRF24L01, а также к GPS-модулю NEO-7M и OLED-экрану 128×64 пиксела

Компонент	ESP8266	Подключено к
nRF24L01 VCC	3V3	Конденсатор 10 мкФ, положительный вывод
nRF24L01 CSN	D8	
nRF24L01 MOSI	D7	

Окончание табл. 12-1

Компонент	ESP8266	Подключено к
nRF24L01 IRQ	Not connected	
nRF24L01 GND	GND	Конденсатор 10 мкФ, отрицательный вывод
nRF24L01 CE	D3	

Таблица 12-2 (продолжение)

Компонент	ESP8266	Подключено к
nRF24L01 SCK	D5	
nRF24L01 MISO	D6	
u-blox NEO-7M VCC	3V3	
u-blox NEO-7M GND	GND	
u-blox NEO-7M TXD	D4	
128×64 OLED GND	GND	
128×64 OLED VCC	3V3	
128×64 OLED SCK	D1	
128×64 OLED SDA	D2	

В листинге 12-5 переданные данные о местоположении из листинга 12-3, отображаемые на веб-странице, также отображаются на OLED-экране для обеспечения оперативной проверки передачи и приема сигнала.

Листинг 12-5. Отображение на OLED-экране данных о местоположении, принятых nRF24L01

```
#include <SPI.h>                    // include SPI library
#include <RF24.h>                   // include RF24 library
RF24 radio(D3, D8);                 // associate radio with RF24 lib
byte addresses[ ][6] = {"12"};      // data pipe address
typedef struct                      // define data structure to include
{
  float GPSlat;                     // character arrays for
  float GPSlong;                    // GPS latitude and longitude
  float GPSalt;                     // GPS altitude (m)
  float GPSspd;                     // GPS ground speed (kmph)
  int sigCount;                     // signal counter
} dataStruct;
dataStruct data;                    // name the data structure as data
float lagTime = 0;
#include <Adafruit_SSD1306.h>       // library 128×64 OLED screen
int width = 128;                    // OLED screen dimensions
int height = 64;
Adafruit_SSD1306 oled(width, height, &Wire, -1);
unsigned long lastTime, nowTime = 0;
void setup()
{
  radio.begin();                    // start radio
```

```
   radio.setChannel(50);                        // set channel number
   radio.setDataRate(RF24_2MBPS);               // baud rate
   radio.setPALevel(RF24_PA_HIGH);              // and power amplifier
   radio.setAutoAck(true);                      // set auto-acknowledge
   (default)
   radio.openReadingPipe(0, address es[0]) ;
                                                // initiate data receive pipe
   radio.startListening();                      // nRF24L01 as receiver
   oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
   oled.clearDisplay();                         // initialise OLED screen
   oled.setTextColor(WHITE);
   oled.setTextSize(1);                         // text size of 6×8 pixels
   oled.display();
}
void loop()
{
   if(radio.available())                        // if signal received
   {
      radio.read(&data, sizeof(data) );         // set signal to data structure
      nowTime = millis();
      lagTime = (nowTime-lastTime)/1 00 0.0;
                                                // time since last signal received
      lastTime = nowTime;
      screen();                                 // call OLED screen function
   }
}
void screen()
{
   oled.clearDisplay();                         // clear display
   oled.setCursor(0,0);                         // position cursor
   oled.print(data.GPSlat,4);                   // display GPS latitude
   oled.setCursor(65,0);                        // and GPS longitude
   oled.print(data.GPSlong,4);
   oled.setCursor(0,10);
   oled.print("alt ");oled.print(da ta.GPSalt,1);
                                                // display GPS altitude
   oled.setCursor(65,10);                       // and GPS speed
   oled.print("spd ");oled.print(data.GPSspd);
   oled.setCursor(0, 20);
   oled.print("lag ");oled.print(la gTime,2);
                                                // time since last signal
   oled.setCursor(65, 20);
   oled.print("chk ");oled.print(da ta.sigCount); // signals sent
   oled.display();
}
```

Улучшение GPS-сигнала

Передача и прием сигнала с помощью приемопередатчиков nRF24L01 могут быть улучшены за счет выбора канала передачи с низкой загруженностью, оптимизации скорости передачи данных и уровня усилителя мощности. В листинге 12-6 микроконтроллер ESP8266 использует библиотеку *RF24* для отображения на мониторе последовательного порта загруженности несущей на каждом из 126 каналов. Если для сканирования каналов с модулем приемопе-

редатчика nRF24L01 используется плата ESP32, то инструкция `RF24 radio(D3, D8)` заменяется на `RF24 radio(2, 4)` для соответствия предыдущим листингам в этой главе.

Листинг 12-6. Сканирование каналов

```
#include <SPI.h>              // include SPI library
#include <RF24.h>             // include RF24 library
RF24 radio(D3, D8);           // associate radio with library
const int nChan = 126;        // 126 channels available
int chan[nChan];              // store counts per channel
int nScan = 100;              // number of scans per channel
int scan;
void setup()
{
  Serial.begin(115200);       // define Serial output baud rate
  radio.begin();              // start radio
}
void loop()
{
  for (int i=0;i<nChan;i++)  // for each channel
  {
    chan[i] = 0;              // reset counter
    for (scan=0; scan<nScan; scan++ )       // repeat scanning
    {
      radio.setChannel(i);                  // define channel
      radio.startListening();
      delayMicroseconds(128);               // listen for 128µs
      radio.stopListening();
      if(radio.testCarrier()>0) cha n[i]=chan[i]+1;
                                            // a carrier on the channel
    }
    delay(1);                               // avoid watchdog reset
  }
  for (int i=0; i<nChan; i++)               // for each channel
  {
    if(i%10 == 0) Serial.print("|");
    Serial.print(chan[i], HEX);             // display carrier activity
  }                                         // format in HEX for values <16
  Serial.println();                         // new line
}
```

Передача и прием сигнала двумя модулями приемопередатчика nRF24L01 оцениваются путем определения количества сигналов, успешно принятых за секунду, при условии неоднократного повторения передачи сигнала. В скетче в листинге 12-7 сообщение, содержащее время (минуты и секунды) и значение счетчика, передается повторно, при этом счетчик равен количеству передач за предыдущую секунду. При установке по умолчанию для автоматического подтверждения сигнала значения *true* количество принятых сигналов будет равно количеству переданных, так как сигнал многократно передается до тех пор, пока не будет подтвержден. Поэтому в скетче для автоматического подтверждения установлено значение *false*.

Листинг 12-7. Передача сигнала для мониторинга модулей приемопередатчика nRF24L01

```
#include <SPI.h>                        // include SPI library
#include <RF24.h>                       // include RF24 library
RF24 radio(D3, D8);                     // associate radio with library
byte addresses[ ][6] = {"12"};          // data pipe address
typedef struct                          // define data structure to include
{
   unsigned long counted;               // counter
   unsigned long mins;                  // time (minute and second)
   unsigned long secs;
} dataStruct;
dataStruct data;
unsigned long lastTime, nowTime = 0;
int count = 0;
int mins = 0, secs = 0;
void setup()
{
   radio.begin();                       // start radio
   radio.setChannel(50);                // set channel number,
   radio.setDataRate(RF24_2MBPS);       // data rate and
   radio.setPALevel(RF24_PA_HIGH);      // power amplifier
   radio.setAutoAck(false);             // set auto-acknowledge
   radio.openWritingPipe(addresses[0]);
                                        // initiate data transmit pipe
   radio.stopListening();               // set nRF24L01 as transmitter
   mins = 0;
   secs = 0;
}
void loop()
{
   nowTime = millis();
   if(nowTime - lastTime > 1000)        // determine minutes
   {                                    // and seconds
      secs++;
      if(secs > 59)                     // after 60 seconds
      {
      secs = 0;                         // reset second variable
      mins++;                           // increment minute variable
      }
      data.counted = count;             // convert values to data structure
      data.mins = mins;
      data.secs = secs;
      count = 0;                        // reset counter
      lastTime = nowTime;               // update time of “second”
   }
   radio.write(&data, sizeof(data));    // transmit signal
   count++;                             // increment signal counter
}
```

Дополнительный скетч модуля приемопередатчика nRF24L01 для приема сигнала приведен в листинге 12-8. Информация о сигнале для принимающего модуля nRF24L01 отображается на OLED-экране 128×64 пикселя для определения влияния различных положений принимающего модуля nRF24L01 относительно положения передающего модуля nRF24L01 (см. рис. 12-14). Библиотека *Adafruit SSD1306* ссылается на библиотеки *Adafruit GFX* и *Wire*, поэтому ин-

струкции `#include <Adafruit_GFX.h>` и `#include <Wire.h>` не требуются. Количество сигналов, передаваемых и принимаемых ежесекундно, сохраняется в массиве, действующем как циклический буфер, для вычисления скользящего среднего числа переданных и принятых сигналов.

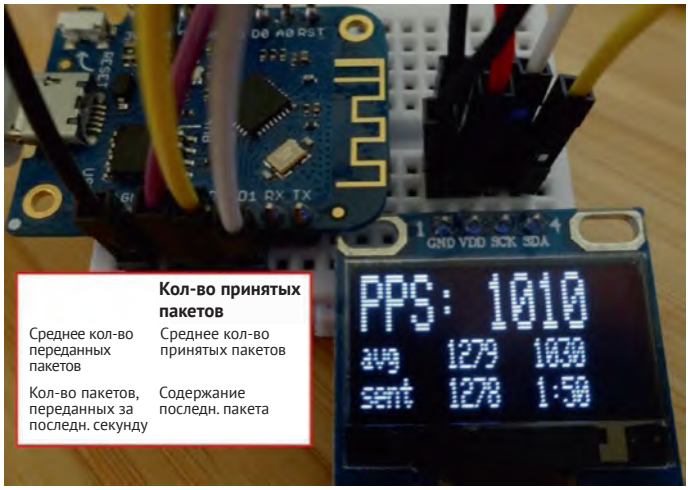


Рисунок 12-14. Пакеты nRF24L01 (принятые/переданные) в секунду

Листинг 12-8. Прием сигнала для мониторинга модулей приемопередатчика nRF24L01

```
#include <SPI.h>                        // include SPI library
#include <RF24.h>                       // include RF24 library
RF24 radio(D3, D8);                     // associate radio with library
byte addresses[ ][6] = {"12"};          // data pipe address
typedef struct                          // define data structure to include
{
  unsigned long sent;                   // sent signals
  unsigned long mins;                   // time (minute and second)
  unsigned long secs;
} dataStruct;
dataStruct data;
#include <Adafruit_SSD1306.h>           // library 128×64 OLED screen
int width = 128;                        // OLED screen dimensions
int height = 64;
Adafruit_SSD1306 oled(width, height, &Wire, -1);
const int Nval = 20;                    // size of circular buffer
int pkts[Nval], sents[Nval];            // arrays for circular buffer
int N = 0, pkt = 0;
unsigned long sumPkt = 0, sumSent = 0;
float avgPkt, avgSent;
unsigned long lastTime, nowTime = 0; // variables to store time values
void setup()
{
  radio.begin();                        // start radio
  radio.setChannel(50);                 // set channel number
  radio.setDataRate(RF24_2MBPS);        // data rate
  radio.setPALevel(RF24_PA_HIGH);       // and power amplifier
  radio.setAutoAck(false);              // set auto-acknowledge
```

```
   radio.openReadingPipe(0, addresses[0]) ;
                                    // initiate data receive pipe
   radio.startListening();          // set nRF24L01 as transmitter
   oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
   oled.clearDisplay();             // initialise OLED screen
   oled.setTextColor(WHITE);
   oled.display();
   data.sent = 0;
   for (int i=0; i<10; i++)         // set circular buffer arrays to zero
   {
      pkts[i] = 0;
      sents[i] = 0;
   }
}
void loop()
{
   if(radio.available())            // signal available
   {
      radio.read(&data, sizeof(data) );       // read signal and
      pkt++;                                  // increment signal counter
   }
   nowTime = millis();
   if(nowTime - lastTime > 1000)               // update values every second
   {
      sumPkt = sumPkt - pkts[N];               // subtract oldest value from sum
      sumPkt = sumPkt + pkt;                   // add current value to sum
      pkts[N] = pkt;                           // update circular buffer
      sumSent = sumSent - sents[N];
      sumSent = sumSent + data.sent;
      sents[N] = data.sent;
      N++;
      if(N > Nval-1) N = 0;           // back to “start” of circular buffer
      avgPkt = 1.0*sumPkt / Nval;     // calculate moving averages
      avgSent = 1.0*sumSent / Nval;
      screen();                       // call OLED screen function
      pkt = 0;
      data.sent = 0;
      lastTime = nowTime;             // update time of last “second”
   }
}
void screen()
{
   oled.clearDisplay();               // clear display
   oled.setCursor(0,0);               // position cursor
   oled.setTextSize(2);               // text size of 12×16 pixels
   oled.print("PPS: ");oled.print(p kt);      // signal (packets) per second
   oled.setCursor(0,16);
   oled.setTextSize(1);                       // text size of 6×8 pixels
   oled.setCursor(0,16);
   oled.print("avg");
   oled.setCursor(40,16);
   oled.print(avgSent,0);                     // average of transmitted signals
   oled.setCursor(80,16);
   oled.print(avgPkt,0);                      // average of received signals
   oled.setCursor(0,25);
   oled.print("sent");
```

```
    oled.setCursor(40,25);
    oled.print(data.sent);                      // last number of sent signals
    oled.setCursor(80,25);
    oled.print(data.mins);oled.print (":");     // signal content
    oled.print(data.secs);
    oled.display();
}
```

Влияние различных каналов передачи, скоростей передачи данных и уровней усилителя мощности оценивается количественно с использованием скетчей в листингах 12-7 и 12-8 для определения оптимальных настроек в конкретной среде. Обратите внимание, что скорости передачи данных для двух модулей приемопередатчика nRF24L01 должны быть одинаковыми. Для пары приемопередатчиков NRF24L01 и микроконтроллеров ESP8266 скорость передачи 2 Мбит/с и уровень усилителя мощности `RF24_PA_HIGH` привели к наибольшему количеству принятых пакетов, хотя количество пакетов, отправленных для скорости 2 Мбит/с, относительно значения скорости передачи было ниже, чем для скоростей 250 Кбит/с и 1 Мбит/с. При скорости передачи 1 Мбит/с количество принимаемых пакетов уменьшалось при более высоких уровнях усиления (см. табл. 12-3). Скорость приема пакетов будет зависеть от приемопередатчиков nRF24L01, расстояния передачи и окружающего электрического шума.

Таблица 12-3. Пакеты nRF24L01, принимаемые/передаваемые в секунду

Уровень усиления	Скорость передачи (бит/с)		
	250 к	1 М	2 М
MIN	150/250	420/1040	880/1280
LOW	10/250	300/1040	900/1280
HIGH	–	100/1040	1030/1280
MAX	–	–	400/1280

Итоги

Приложение GPS-трекинга, созданное с помощью *MIT App Inventor*, отображает в *Google Maps* местоположение удаленно расположенного GPS-модуля u-blox NEO-7M или маршрут, пройденный GPS-модулем. Модуль nRF24L01, подключенный к плате ESP8266 или ESP32, соединенной с модулем GPS, передавал местоположение на принимающий модуль nRF24L01, подключенный к плате ESP32. Информация о местоположении была передана в приложение микроконтроллером ESP32 с использованием связи по Bluetooth. Приложение отображало информацию о местоположении на планшете или мобильном телефоне Android, получив доступ к Google Maps из Всемирной паутины через интернет-провайдера. Приложение для трекинга включало функцию масштабирования карты и возможность подключения маркеров отслеживания местоположения и отображения пройденного маршрута. Для тестирования передачи сигнала с GPS-данными была создана веб-страница для отображения передаваемых и принимаемых данных. Влияние различных каналов передачи,

скоростей передачи данных и уровней усилителя мощности оценивалось по количеству сигналов, успешно принятых за секунду, при условии повторной передачи сигнала.

Перечень компонентов

- Микроконтроллер ESP8266: LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: DEVKIT DOIT или NodeMCU
- GPS-модуль: u-blox NEO-7M
- Модуль приемопередатчика: nRF24L01 2 шт.
- OLED-дисплей 128×64 пиксела

Глава 13

Связь через USB OTG[65]

В главе 12 («Приложение для GPS-трекинга с использованием Google Maps») данные передаются в приложение на планшете или мобильном телефоне Android с помощью микроконтроллера ESP32 через Bluetooth. В главе 10 («Создаем мобильное приложение») модуль Bluetooth HC-05 подключается к плате ESP8266 для связи с приложением. Если устройство для передачи данных в приложение не имеет функции Bluetooth, то соединить устройство с приложением можно через интерфейс USB OTG. В этой главе устройство без функции Bluetooth представляет собой Arduino Uno. Модуль Bluetooth HC-05 может быть подключен к Arduino Uno для связи по Bluetooth с планшетом или мобильным телефоном Android, но в этой главе основное внимание уделяется связи по USB OTG.

Связь USB OTG позволяет планшету или мобильному телефону Android обмениваться сигналами с периферийным устройством USB. Планшет или мобильный телефон Android – это устройство OTG-A, действующее в качестве хоста для этого приложения и обеспечивающее питание периферийного устройства USB или устройства OTG-B, которым является Arduino Uno. USB OTG-кабель соединяет хост с периферийным устройством USB. Приложение *Easy OTG Checker* от Кярвела (Kjarvel) в магазине *Google Play* определяет, поддерживает ли планшет или мобиль-

[65] USB OTG (*On-the-Go*, «*на ходу*») – приложение к стандартам USB 2.0 и 3.0, позволяющее переключать функции хоста и ведомого (периферийного) устройства. Например, оснащенный OTG-функциональностью смартфон при подключении к ноутбуку (хост, устройство OTG-A) выступает в качестве накопителя (периферийное устройство, OTG-B), а при подключении к USB-флешке или принтеру – в качестве хоста. Таким образом информация может передаваться напрямую, без посредника в виде стационарного компьютера. По умолчанию в качестве хоста, управляющего соединением, выступает устройство, из которого подается питание, но в процессе обмена они могут меняться ролями. Для OTG-устройств характерен разъем mini/micro USB (или современный USB-C), в котором задействованы не 4, а 5 контактов, и для перехода в режим хоста контакты 4 и 5 должны быть оба соединены с GND. Обычно это обеспечивается специальным переходником, на другом конце которого находится обычное прямоугольное гнездо USB type A (см. иллюстрацию в тексте). Есть и более сложные схемы подключения – с отдельным усиленным питанием, с возможностью подзарядки в OTG-режиме. Отметим также, что для подключения к Arduino Uno в качестве устройства OTG-B, как делается в данной главе, с помощью такого переходника дополнительно потребуется традиционный USB A-B-кабель (у автора этот факт не отражен). – *Прим. перев.*

ный телефон Android OTG при подключении периферийного устройства с помощью кабеля USB OTG (см. рис. 13-1).

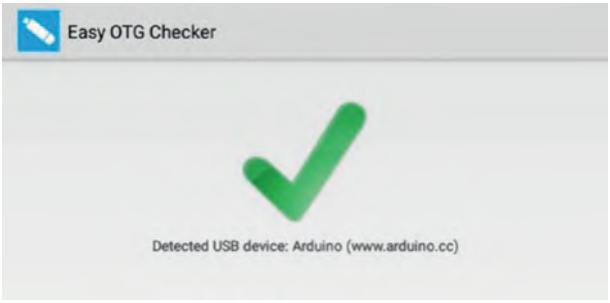


Рисунок 13-1. Простое приложение для проверки OTG, подключенное к Arduino Uno

MIT App Inventor предоставляет компонент *Serial* для связи USB OTG, расположенный в палитре *Connectivity* в левой части окна конструктора. Компонент *Serial* взаимодействует с USB-Serial преобразователем Arduino Uno ATmega16U2, но не может работать с чипом CH340 на платах Arduino Nano[66] или ESP8266. Если компонент *Serial* MIT App Inventor будет расширен до поддержки CH340, то платформа для создания коммуникационных приложений USB OTG, описанная в этой главе, будет применима к микроконтроллеру ESP8266.

Приложение для приема данных

Для демонстрации планшета или мобильного телефона Android как устройства OTG-A, получающего данные от Arduino Uno (устройства OTG-B), Arduino Uno передает пару чисел. Приложение, размещенное на устройстве OTG-A, разбивает пару чисел на компоненты и отображает как саму пару чисел, так и ее компоненты (см. рис. 13-2). Числовой парой могут быть широта и долгота от модуля GPS, температура и влажность от датчика BMP280 или напряжение и ток от датчика INA219, подключенных к Arduino Uno. Приложение предоставляет основу для более сложных проектов с передачей и приемом сигналов.

Рисунок 13-2. Приложение для получения сообщений с устройства OTG-B

[66] На платах Arduino Nano оригинального дизайна (с надписью «made in Italy» или «USA») размещен преобразователь USB-Serial FTDI FT232. Наличие преобразователя CH340 характерно для более дешевых клонов Arduino Nano китайского производства. – *Прим. перев.*

Кнопки *accessUno*, *OnButton* и *OffButton* позволяют отключать и подключать устройство OTG-B к устройству OTG-A, при этом статус подключения отображается на текстовой метке *StatusLabel*. При нажатии кнопки *accessUno*, после подключения Arduino Uno к планшету или мобильному телефону Android, выполняется запрос на доступ к USB-устройству (см. рис. 13-3). Выбирается запрос доступа *OK*, и нажимается кнопка *OnButton* приложения, чтобы подключить устройство OTG-B к устройству OTG-A.

Рисунок 13-3. Запрос доступа к USB-устройству

Скетч в листинге 13-1 выводит на монитор последовательного порта пары чисел с интервалом в одну секунду. Если Arduino Uno подключен к планшету Android или мобильному телефону с помощью кабеля USB OTG, Arduino Uno и Android-планшет или мобильный телефон начинают обмениваться данными с помощью USB OTG, как только приложение открывает последовательный порт.

Листинг 13-1. Передача пары чисел

```
String text;
int count = 0;
unsigned long timer;
void setup()
{
   Serial.begin(9600);                // define serial baud rate
}
void loop()
{
   if(millis() - timer > 1000)        // after 1sec has elapsed
   {
      count++;                        // increment count
      if(count > 254) count = 0;      // arbitrary limit on counter
      text = String(count)+ "," +Str ing(count+1);
                                      // combine numbers into text
      Serial.println(text);           // transmit text
      timer = millis();               // reset timer
   }
}
```

Макет приложения состоит из заголовка приложения и двух областей *HorizontalArrangements*, содержащих текстовые метки *sentText*, *Part1Text*, и *Part2Text* для отображения текста и кнопок (см. рис. 13-4). Кнопки *accessUno*, *OnButton* и *OffButton* для отключения-подключения устройства OTG-B к устройству OTG-A, а также

метка *StatusLabel* содержатся во второй области *HorizontalArrangement*. Компоненты *Serial* и *Clock* отображаются под макетом приложения.

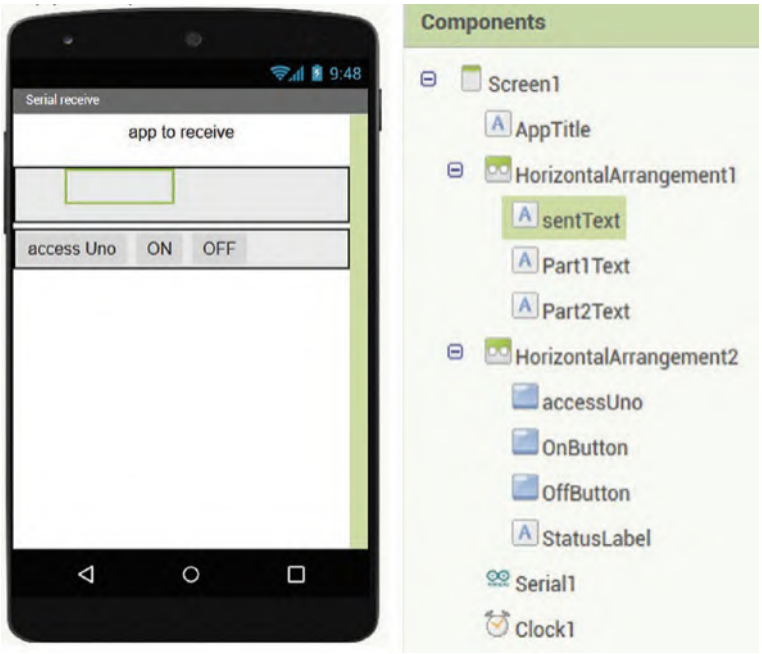


Рисунок 13-4. Макет приложения для обмена OTG-B с OTG-A

Блоки для управления последовательным подключением устройства OTG-B к устройству OTG-A, считывания сообщения, передаваемого устройством OTG-B, и отображения сообщения на экране устройства OTG-A показаны на рис. 13-5, 13-6 и 13-7. Устройство OTG-B или *Serial1* инициализируется, и определяется скорость передачи данных в бодах, точно так же, как в случае с инструкцией `Serial.begin(9600)` при настройке скорости передачи данных последовательного монитора в Arduino IDE. Метка *StatusLabel* отображает состояние соединения устройства OTG-B с устройством OTG-A (см. рис. 13-5).

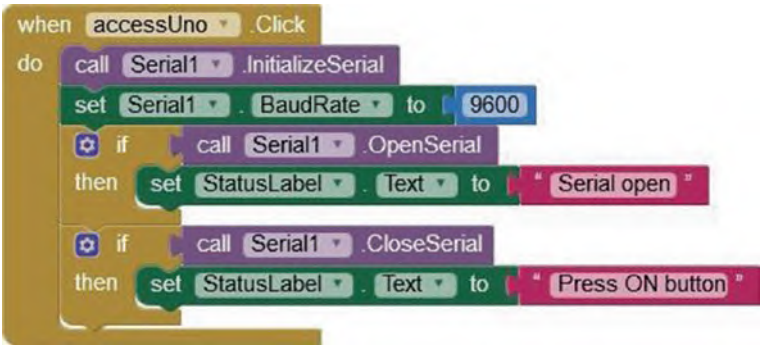


Рисунок 13-5. Подключение устройства OTG-B

Приложение включает в себя возможность подключения и отключения устройства OTG-B (см. рис. 13-6). Нажатие кнопок *OnButton* или *OffButton* подключает или отключает устройство OTG-B с помощью процедур *connect* и *close*. Статус подключения отображается меткой *StatusLabel*, поскольку процедура *connect* при подключении последовательного порта возвращает значение, равное единице. То же самое относится к процедуре *close*.

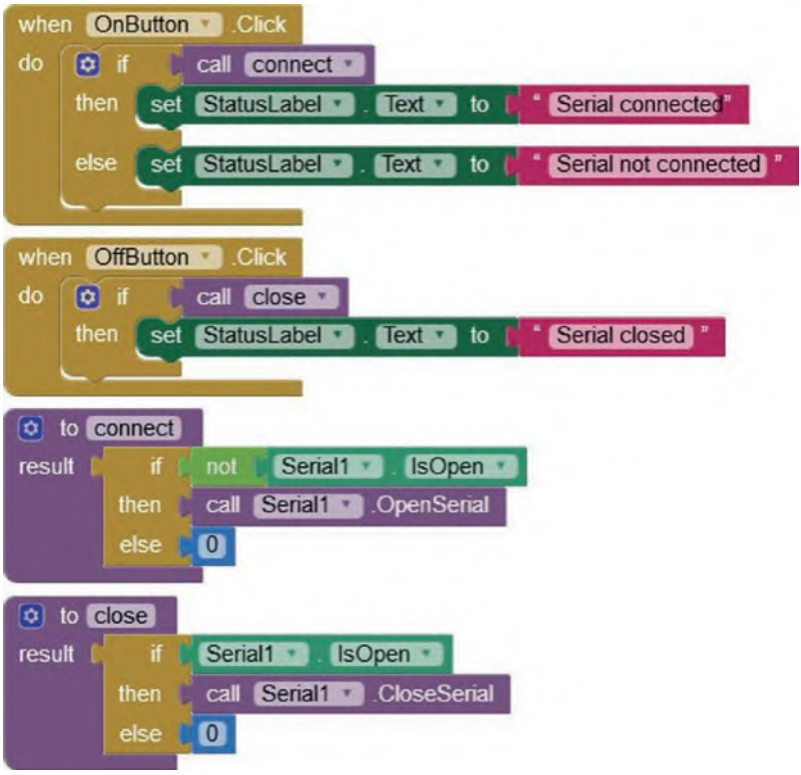


Рисунок 13-6. Подключение и отключение устройства OTG-B

Раздел получения сообщений приложения запускает таймер *Clock1* с интервалом 1000 мс для отслеживания сообщения на устройстве OTG-B. Интервал таймера *Clock1* устанавливается в разделе *Properties* в правой части окна конструктора. Если подключено устройство OTG-B и полученное сообщение содержит запятую, то сообщение разбивается на компоненты списка с индексами 1 и 2, которые отображаются на экране устройства OTG-A в виде текстовых переменных *Part1text* и *Part2text* (см. рис. 13-2 и 13-7).

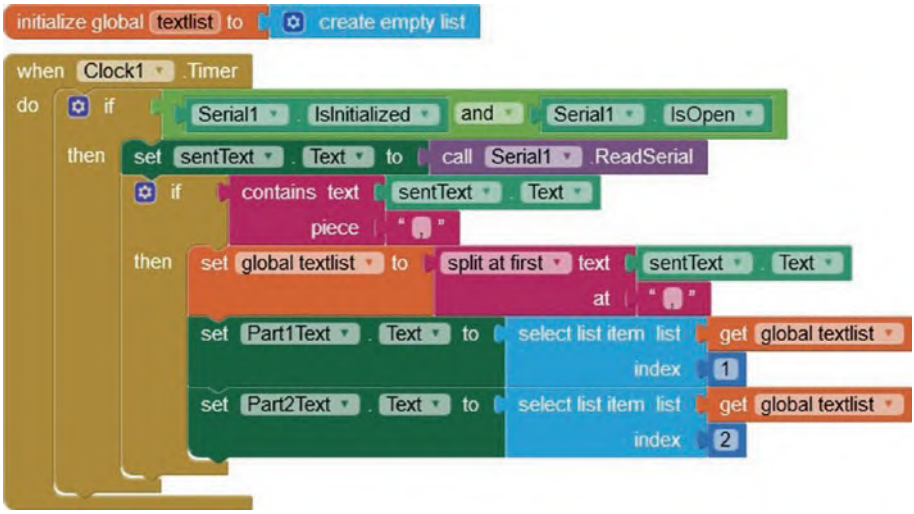


Рисунок 13-7. Получение и отображение сообщения с устройства OTG-B

Приложение для передачи данных

Противоположностью устройства OTG-B, которым является Arduino Uno, передающего на хост-устройство OTG-A, которым является планшет или мобильный телефон Android, является устройство OTG-A, передающее на периферийное устройство USB или устройство OTG-B. При нажатии кнопок *RedButton*, *GreenButton* или *BlueButton* устройство OTG-A передает на устройство OTG-B число 1, 2 или 3, соответствующее красному, зеленому или синему светодиоду и соответствующему значению ползунка, при этом цвет выбранного светодиода указывается на устройстве OTG-A (см. рис. 13-8 после нажатия *RedButton*).

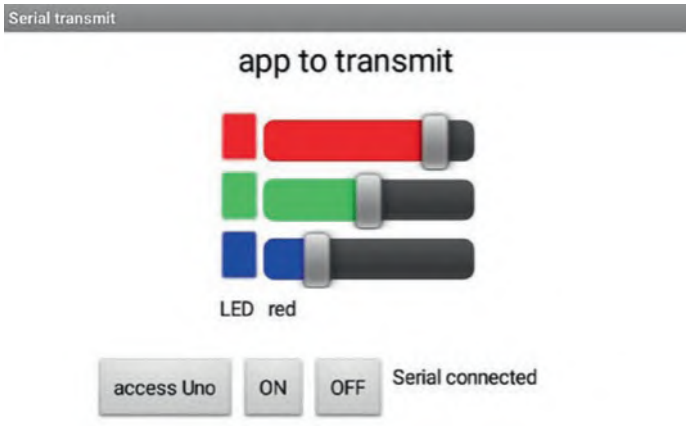


Рисунок 13-8. Приложение для управления светодиодами, подключенными к устройству OTG-B

Подключения к Arduino Uno показаны на рис. 13-9 и приведены в табл. 13-1. Приложение обеспечивает основу для более сложных проектов передачи и приема сигналов.

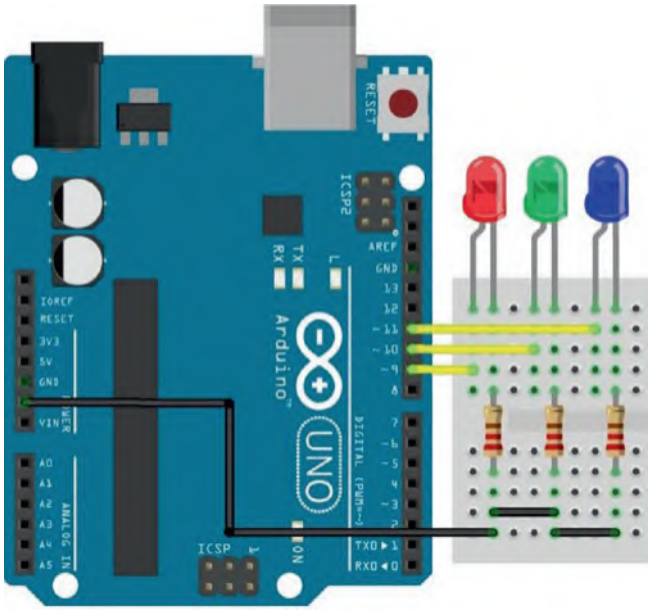


Рисунок 13-9. Arduino Uno как OTG-B-устройство

Таблица 13-1. Arduino Uno как OTG-B-устройство

Компонент	Подключено к	Также к
LED длинные выводы	Arduino Uno выводы PWM 9, 10, 11	
LED короткие выводы	Резистор 220 Ом	Arduino Uno GND

Скетч в листинге 13-2 анализирует последовательный буфер на номер обновляемого светодиода и значение широтно-импульсной модуляции PWM, задающее яркость. Затем яркость светодиода обновляется. Если номер светодиода равен нулю, то все светодиоды выключаются. Обратите внимание, что светодиоды подключены к PWM-выводам Arduino Uno.

Листинг 13-2. Arduino Uno как OTG-B-устройство

```
int redLED = 9;
int greenLED = 10;                    // define LED PWM pins
int blueLED = 11;
int LED;
int bright[] = {0,0,0,0};             // initial PWM values
void setup()
{
  Serial.begin(9600);                 // set Serial baud rate
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);          // set LED pins as output
```

```
    pinMode(blueLED, OUTPUT);
}
void loop()
{
    if(Serial.available() > 0)                  // wait for app message
    {
      LED = Serial.parseInt();                  // parse message to LED
      bright[LED] = Serial.parseInt();          // and PWM value
      if(LED < 1) for (int i=1; i<4; i++) b right[i] = 0;
                                                // all LEDs off
      analogWrite(redLED, bright[1]);
      analogWrite(greenLED, bright[2]);         // update LED brightness
      analogWrite(blueLED, bright[3]);
    }
}
```

Макет приложения содержит заголовок приложения и две области *HorizontalArrangements*. Компонент *TableArrangement*, расположенный в палитре *Layout* в левой части окна конструктора, состоит из двух столбцов по четыре строки, содержащих кнопки *RedButton*, *GreenButton* и *BlueButton* с соответствующими ползунками для управления значениями PWM, с метками *LEDlabel* и *LEDcolor*, указывающими, к какому светодиоду обращались последним (см. рис. 13-10). Вторая область *HorizontalArrangements* включает кнопки *accessUno*, *OnButton* и *OffButton* для соединения устройства OTG-B и OTG-A и метку *StatusLabel*. Компонент последовательного порта *Serial* отображается под макетом приложения. Минимальное и максимальное значения ползунка, 0 и 255 соответственно, задаются в разделе *Properties*, с *ColorLeft* и *ColorRight* – левым и правым цветами для ползунка и начальной позицией ползунка 127.

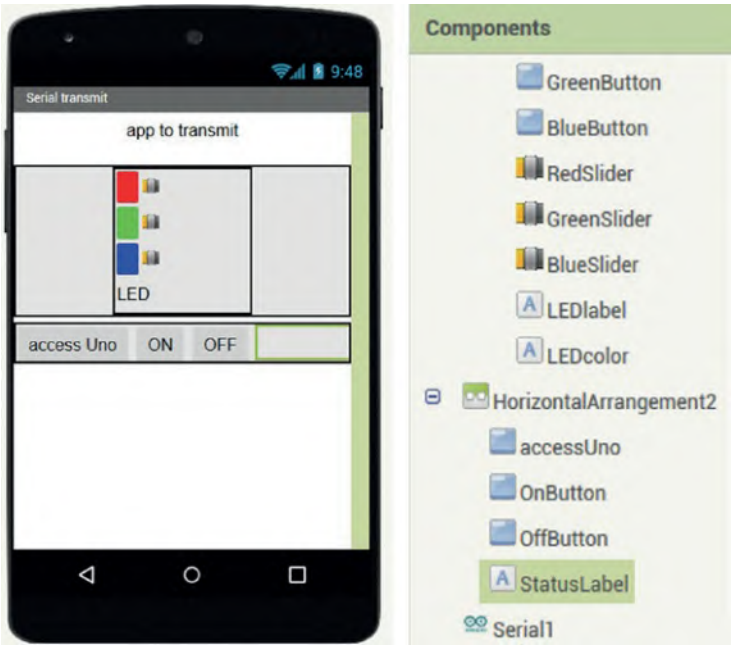


Рисунок 13-10. Макет приложения OTG-A для управления устройством OTG-B

Управление подключением устройства OTG-A к устройству OTG-B такое же, как в примере OTG-B к OTG-A (см. рис. 13-5 и 13-6). С тем отличием, что когда устройство OTG-A отключает устройство OTG-B, процедура *LED* передает нулевое значение светодиода на устройство OTG-B, которое затем выключает все светодиоды (см. рис. 13-11).

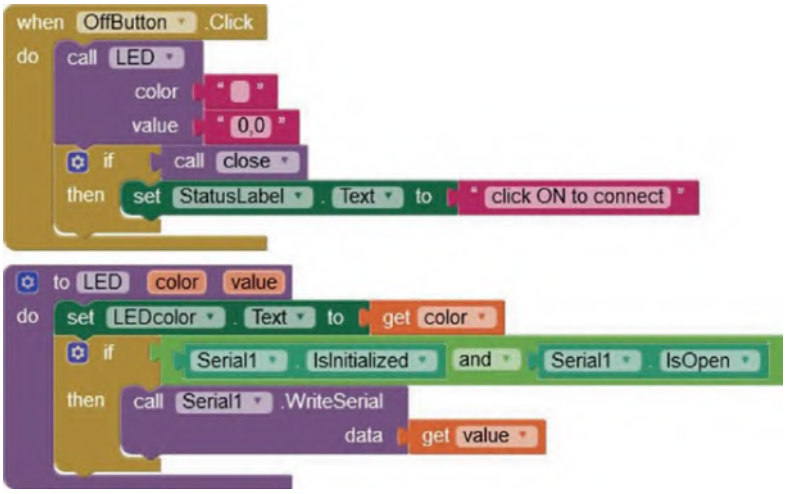


Рисунок 13-11. Отключение соединения OTG-A и OTG-B

Нажатие кнопки *RedButton* на устройстве OTG-A вызывает процедуру *LED* (см. рис. 13-12) для передачи на устройство OTG-B значения “1” и положения ползунка красного светодиода, разделенных запятыми. Блок округления округляет вверх или вниз положение ползунка, которое является действительным числом, до целого числа. Глобальная переменная *RedValue* изначально устанавливается в средней позиции ползунка 127, и в ней сохраняется текущее положение ползунка. Аналогичные блоки используются для зеленого и синего светодиодов, но со значениями номеров светодиодов “2” и “3” соответственно.

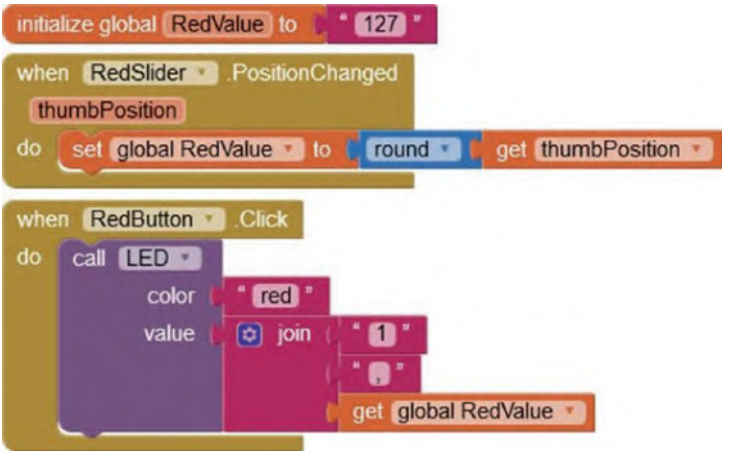


Рисунок 13-12. Управление светодиодами

Приложение для приема и передачи данных

Приложение для приема и передачи данных с устройства OTG-B и на него объединяет предыдущие примеры приложений приема и передачи. Рисунок 13-13 – это скриншот приложения приема и передачи, после того как устройство OTG-B передало сообщение, содержащее пару чисел 9, 10, которое было обработано устройством OTG-A. Скриншот сделан после нажатия кнопки *RedButton* на устройстве OTG-A для передачи сообщения на устройство OTG-B для обновления яркости красного светодиода. Приложение приема и передачи объединяет скриншоты на рис. 13-2 и 13-8.

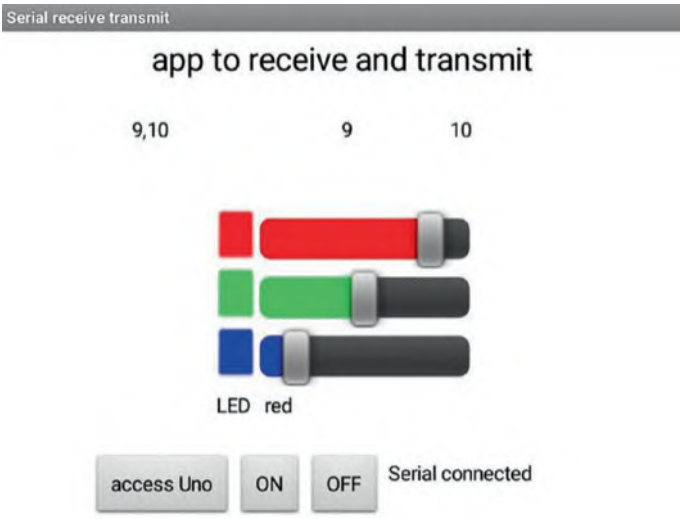


Рисунок 13-13. Приложение для приема и передачи данных

Макет дизайна приложения приема и передачи на рис. 13-14 представляет собой прямую комбинацию макетов приложений – приложения приема (см. рис. 13-4) и приложения передачи (см. рис. 13-10).

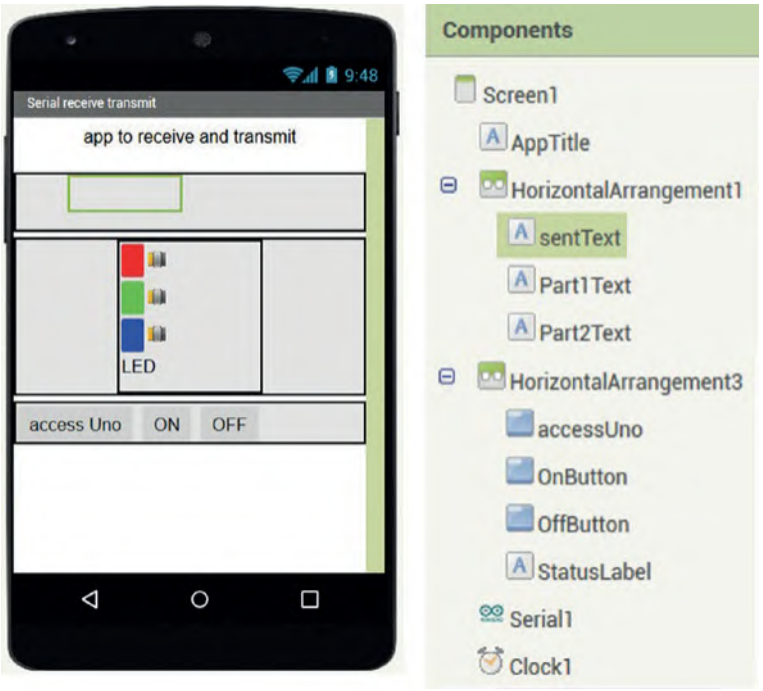


Рисунок 13-14. Макет приложения для приема и передачи данных

Скетч приложения для приема и передачи данных состоит из скетча приложения передачи OTG-A в OTG-B (листинге 13-2) и содержимого функции `loop` из скетча приложения приема OTG-B в OTG-A (листинг 13-1), а также инструкций по определению переменных

```
String text
int count = 0
unsigned long timer
```

При нажатии кнопки в приложении приема и передачи устройство OTG-A передает сообщение на устройство OTG-B для управления светодиодами во время обработки полученных сообщений с устройства OTG-B. Две условные инструкции `if(millis() - timer > 1000)` и `if(Serial.available() > 0)`, используемые в листингах 13-1 и 13-2 соответственно, позволяют устройству OTG-B (Arduino Uno) «одновременный» прием и передачу сигнала на хост-устройство OTG-A, планшет Android или мобильный телефон.

Итоги

Связь USB OTG соединила устройство OTG-B (Arduino Uno) с приложением, размещенным на устройстве OTG-A, планшете или мобильном телефоне Android, и два устройства были соединены кабелем USB OTG. Приложение приема обрабатывает данные, передаваемые устройством OTG-B на OTG-A хост-устройство. Аналогично приложение для передачи отправляло данные

на устройство OTG-B для управления яркостью нескольких светодиодов с помощью ШИМ. Два приложения были объединены в приложение для приема и передачи, при этом устройство OTG-A одновременно принимало сигнал и передавало сигнал на устройство OTG-B. Компонент *MIT App Inventor* для последовательного порта *Serial* взаимодействует с преобразователем USB-Serial на основе ATmega16U2 в Arduino Uno, но не с чипом CH340 на платах Arduino Nano или ESP8266.

Перечень компонентов

- Arduino Uno
- USB OTG-кабель: гнездо USB-A и micro-USB тип B[67]
- RGB-светодиоды или обычные разных цветов: 3 шт.
- Резистор 220 Ом: 3 шт.

[67] Напомним (см. сноску 65 на стр. 234), что для подключения смартфона (устройство OTG-A) к Arduino Uno (устройство OTG-B) дополнительно потребуется традиционный USB A-B-кабель. – *Прим. перев.*

Глава 14

Обмен данными через ESP-NOW и LoRa

Связь с помощью приемопередатчиков nRF24L01 и Bluetooth для передачи данных о GPS-местоположении описана в главе 12 («Приложение для GPS-трекинга с использованием Google Maps»), кабельная OTG-связь – в главе 13 («Связь через USB OTG»), и связь Wi-Fi для обновления информации о веб-странице – в главе 9 («WebSocket») или для интернет-радио в главе 1. Два других протокола связи, ESP-NOW и LoRa, описаны в этой главе.

ESP-NOW

Протокол ESP-NOW, разработанный фирмой Espressif Systems[68], позволяет микроконтроллерам ESP8266 и ESP32 связываться без подключения к Wi-Fi. ESP-NOW работает на частоте 2,4 ГГц, той же частоте, что Wi-Fi и Bluetooth, с помощью микроконтроллеров, сопряженных между собой перед связью. Микроконтроллер обменивается сообщениями с несколькими другими микроконтроллерами, причем сеть может включать до 20 микроконтроллеров без шифрования сообщений, и до 10 микроконтроллеров с шифрованием. Если питание микроконтроллера пропадает, а затем восстанавливается, микроконтроллер заново автоматически подключается к сопряженному. На практике между двумя микроконтроллерами ESP32 была достигнута дальность передачи 250 м на открытой местности.

Библиотека *ESP-NOW* автоматически появляется в Arduino IDE, когда установлен менеджер плат ESP8266 или ESP32. В скетче библиотека ESP-NOW должна быть включена инструкциями `<espnow.h>` или `<esp_now.h>` для микроконтроллеров ESP8266 или ESP32 соответственно.

[68] Espressif Systems (Шанхай, Китай) – ведущая в мире компания в области разработки и изготовления высокопроизводительных маломощных Wi-Fi и Wi-Fi/Bluetooth-решений для «интернета вещей» (IoT). Самые знаменитые их разработки – микроконтроллеры ESP8266 (2014) и ESP32 (2016) с интегрированными беспроводными модулями – были признаны поворотным моментом для мирового рынка IoT. – *Прим. перев.*

Для сопряжения микроконтроллеров требуется MAC-адрес[69] каждого микроконтроллера, являющийся адресом для связи внутри сети. В листинге 14-1 содержится MAC-адрес микроконтроллера ESP8266. Для микроконтроллера ESP32 инструкция `#include <ESP8266WiFi.h>` должна быть заменена на `#include <WiFi.h>`. MAC-адрес содержит шесть чисел в шестнадцатеричном формате, например 3C:71:BF:F1:CC:9C. MAC-адрес также отображается Arduino IDE при компиляции и загрузке скетча (см. рис. 14-1).

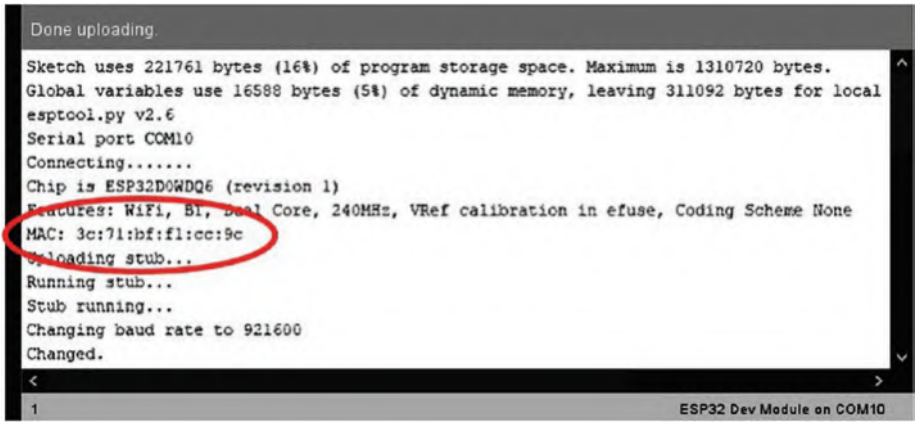


Рисунок 14-1. MAC-адрес в Arduino IDE

Листинг 14-1. MAC-адрес

```
#include <ESP8266WiFi.h>              // include Wi-Fi library
void setup()
{
   Serial.begin(115200);              // Serial Monitor baud rate
   Serial.println(WiFi.macAddress()); // get MAC address
}
void loop()                           // nothing in loop function
{}
```

Информация, передаваемая с помощью ESP-NOW, может представлять собой целое число, действительное число, текст или структуру данных, содержащую комбинацию остальных трех типов данных. Чтобы обеспечить некоторую универсальность, скетчи в этой главе включают передачу структуры данных. Для ESP-NOW максимальный размер структуры данных составляет 250 байт, при этом для целого числа, действительного числа или символа требуется 2, 4 или 1 байт соответственно. В листинге 14-2 приведен пример структуры данных, состоящей из двух целых чисел, действительного числа и массива символов. Обратите внимание, что длина массива символов равна максимальному количеству символов плюс один, чтобы добавить символ конца строки /n.

[69] MAC-адрес (media access control address) – уникальный идентификатор сетевого адаптера в сетях стандартов IEEE 802, в основном Ethernet, Wi-Fi и Bluetooth. – *Прим. перев.*

Листинг 14-2. Пример структуры данных

```
typedef struct                // define structure to include
{
   int count = 5;             // two integers,
   int total;
   float value = 3.14;        // a real number
   char text[12] = "text";    // and a character array
} dataStruct;
dataStruct payload;           // name the structure as payload
```

Структура передается или принимается с помощью имени структуры *payload*, при этом каждый компонент структуры доступен индивидуально, например `payload.value`.

Следующий раздел содержит инструкции для микроконтроллера ESP8266. Различия в командах между микроконтроллерами ESP8266 и ESP32 перечислены в табл. 14-1. Микроконтроллеру ESP8266 назначается роль передатчика (`CONTROLLER`), приемника (ведомого устройства `SLAVE`) или передатчика и приемника одновременно (`COMBO`) с помощью инструкции `esp_now_set_self_role(role)`. Терминология определена действующим соглашением для ESP-NOW с микроконтроллером ESP8266. Принимающий микроконтроллер ESP8266 идентифицируется передающим микроконтроллером ESP8266 по MAC-адресу, параметру `role` и каналу связи с помощью команды `esp_now_add_peer(receiveMAC, ESP_NOW_ROLE_SLAVE, channel, NULL, 0)`, где `receiveMAC` – это MAC-адрес принимающего микроконтроллера ESP8266. Структура данных `payload` передается на принимающий микроконтроллер ESP8266 с помощью команды `esp_now_send(receiveMAC, (uint8_t *) & payload, sizeof(payload))`. Если MAC-адрес получателя заменен на `{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}`, то передача будет осуществляться на все принимающие микроконтроллеры. Символы амперсанда & и звездочки * относятся к адресу памяти структуры `payload`. Доступ к адресу переменной в памяти осуществляется указанием перед переменной амперсанда (`&payload`), а указатель сохраняет адрес другой переменной в памяти при размещении перед указателем звездочки (`*pointer`). Указатель определяется с тем же типом данных, что и переменная, например `float *pointer`. Доступ к значению переменной осуществляется с помощью указателя, как показано в листинге 14-3.

Листинг 14-3. Переменная, указатель и адрес памяти

```
int sum = 25;         // allocate value to variable
int * pointer;        // define pointer
int number;
void setup()
{
   Serial.begin(115200);
   pointer = &sum;            // set pointer to address of sum
   number = *pointer;         // set number to pointer content
   Serial.print("\nnumber ");Serial.println(number);
}
void loop()                   // nothing in loop function
{}
```

Передающий микроконтроллер ESP8266 с помощью команды `esp_now_register_send_cb(sendData)` ожидает подтверждения того, что переданные данные

были получены принимающим микроконтроллером ESP8266. Функция `sendData` возвращает MAC-адрес принимающего микроконтроллера ESP8266 и значение подтверждения. В листинге 14-4 также отображается MAC-адрес принимающего микроконтроллера ESP8266 с использованием функции `printf` с параметром `%02x` , что означает преобразование содержимого массива `mac` в шестнадцатеричный формат с дополнением символом 0 до двух цифр. Шестнадцатеричный формат в верхнем или нижнем регистре печатается с указанием параметра %X или %x.

Листинг 14-4. Подтверждение для передатчика

```
void sendData(uint8_t * mac, uint8_t chk)
{
   for (int i=0; i<6; i++)                    // receiver MAC address
   {
      Serial.printf("%02x", mac[i]);          // convert to HEX format
      if(i < 5) Serial.print(":");            // include colons
   }
   Serial.print("\tcallback ");
   if(chk == 0) Serial.println("OK ");        // transmission received
   else Serial.println("fail");               // or not
}
```

Принимающий микроконтроллер ESP8266 формирует подтверждение с помощью команды `esp_now_register_recv_cb(receiveData)`. Функция `receiveData` возвращает MAC-адрес передающего микроконтроллера ESP8266 `mac`; копирует полученные данные `data` в адрес памяти структуры `&payload`, отображает количество байтов полученных данных `len` и массив символов `text` на мониторе последовательного порта.

Листинг 14-5. Подтверждение приемника

```
void receiveData(uint8_t * mac, uint8_t * data, uint8_t len)
{ // copy received data to payload
   memcpy(&payload, data, sizeof(pay load));
   Serial.print("bytes ");Serial.print(len);Serial.print("\t");
   Serial.print("text ");Serial.println(payload.text);
}
```

Полный текст скетча передающего микроконтроллера ESP8266 для отправки целого числа, действительного числа и текста с помощью ESP-NOW приведен в листинге 14-6. Обратите внимание, что MAC-адрес в листинге 14-6 должен быть заменен MAC-адресом вашего принимающего микроконтроллера ESP8266, или следует использовать общее значение `{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}`.

Листинг 14-6. ESP-NOW для передающего микроконтроллера ESP8266

```
#include <espnow.h>                    // include ESP-NOW library
uint8_t receiveMAC[] = {0x84,0xF3,0xEB,0x0D,0xB5,0xB3};
typedef struct                         // receiver MAC address
{
   int count = 0;                      // data structure with
   float value = 3.14;                 // integer, real number
   char text[10] = "abcdef";           // and character array
} dataStruct;
```

```
dataStruct payload;
int channel = 1;                              // set transmission channel
int chk;
void setup()
{
  Serial.begin(115200);                       // Serial Monitor baud rate
  if(esp_now_init() != 0)                     // initialise ESP-NOW
  {
    Serial.println("error initialising ESP-NOW");
    return;
  }                                           // transmitter device
  esp_now_set_self_role(ESP_NOW_ROLE_CO NTROLLER);
  chk = esp_now_add_peer(receiveMAC, ESP_NOW_ROLE_SLAVE,
  channel, NULL, 0);                          // add receiver device
  if(chk == 0) Serial.println("receiver added");
  else
  {
    Serial.println("error adding receiver");
    return;
  }
  esp_now_register_send_cb(sendData);         // link to sendData function
}
void loop()
{
  payload.count++;                            // increment counter
  payload.value = payload.value + 1.0;        // and real number
  if(strcmp(payload.text,"abcdef") == 0 )     // alternate text
  strncpy(payload.text, "xyz", sizeof(payload.text));
  else strcpy(payload.text, "abcdef");
  Serial.print(payload.count);
  Serial.print(payload.value);                // display transmitted data
  Serial.print(payload.text);
  chk = esp_now_send(receiveMAC, (uint8_t *) & payload,
  sizeof(payload));
  Serial.print("\tsent ");
  if(chk == 0) Serial.print("OK ");          // transmission sent or not
  else Serial.println("fail");
  delay(2000);
}
void sendData(uint8_t * mac, uint8_ t chk)    // callback function
{
  for (int i = 0; i < 6; i++)                 // display receiving MAC address
  {
    Serial.printf("%02x", mac[i]);
    if (i < 5)Serial.print(":");
  }
  Serial.print("\tcallback ");                // transmission received or not
  if(chk == 0) Serial.println("OK ");
  else Serial.println("fail");
}
```

Дополнительный скетч для принимающего микроконтроллера ESP8266 приведен в листинге 14-7, при этом полученное сообщение отображается на мониторе последовательного порта.

Листинг 14-7. ESP-NOW для принимающего микроконтроллера ESP8266

```
#include <espnow.h>          // include ESP-NOW library
typedef struct
{
   int count;                          // data structure with
   float value;                        // integer, real number
   char text[10];                      // and character array
} dataStruct;
dataStruct payload;
int rcv = 0;                           // counter of received signals
void setup()
{
   Serial.begin(115200);
   if(esp_now_init() != 0)    // initialise ESP-NOW
   {
      Serial.println("error initialising ESP-NOW");
      return;
   }
   esp_now_set_self_role(ESP_NOW_ROLE_SLAVE); // receiver device
   esp_now_register_recv_cb(receiveD ata);
}                                      // link to receiveData function
void receiveData(uint8_t * mac, uint8_t * data, uint8_t len)
{
   rcv++;                              // increment signal counter
   memcpy(&payload, data, sizeof(pay load));
                                  // copy received data to payload
   for (int i = 0; i < 6; i++)
   {                                   // display transmitting MAC address
      Serial.printf("%02x", mac[i]);
      if (i < 5)Serial.print(":");
   }                                   // display contents of payload
   Serial.print("\t");
   Serial.print("received ");Serial.print(rcv);Serial.print("\t");
   Serial.print("bytes ");Serial.print(len);Serial.print("\t");
   Serial.print("count ");Serial.print(payload.count);
   Serial.print("\t");
   Serial.print("value ");Serial.print(payload.value);
   Serial.print("\t");
   Serial.print("text ");Serial.println(payload.text);
}
void loop()                            // nothing in loop function
{}
```

Протокол WebSocket позволяет серверу передавать информацию клиенту, при этом информация отображается на веб-странице, как описано в главе 9 («WebSocket»). В листинге 14-8 автоматически обновляется веб-страница с сообщением и временем получения (см. рис. 14-2). Приложение – это отображение информации с датчиков.

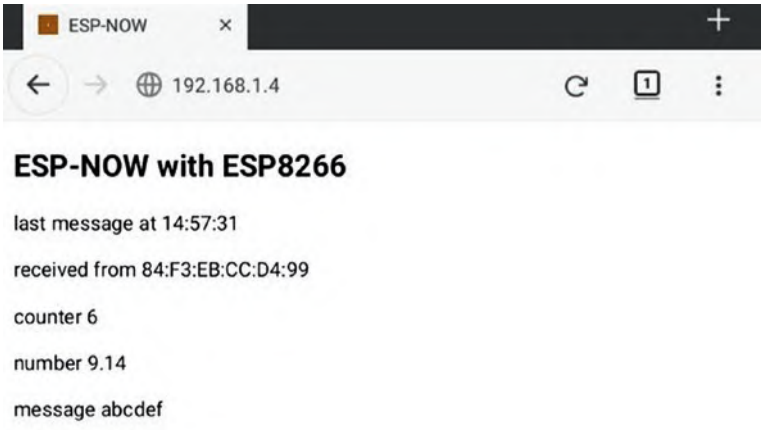


Рисунок 14-2. Прием на микроконтроллер ESP8266 с помощью ESP-NOW и WebSocket

Листинг 14-8. Прием на микроконтроллер ESP8266 с помощью ESP-NOW и WebSocket

```
#include <ESP8266WebServer.h>         // include web server library
ESP8266WebServer server;
#include <WebSocketsServer.h>         // include WebSocket library
WebSocketsServer websocket = WebSoc ketsServer(81);
                                      // set WebSocket port 81
#include "buildpage.h"                // webpage AJAX code
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to Wi-Fi password
#include <espnow.h>                   // include ESP-NOW library
typedef struct
{
  int count;                          // data structure with
  float value;                        // integer, real number
  char text[10];                      // and character array
} dataStruct;
dataStruct payload;
String strMAC, message, json;
void setup()
{
  Serial.begin(115200);               // Serial.Monitor baud rate
  WiFi.begin(ssid, password);         // initialise Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
                                      // display web server IP address
  Serial.print("MAC address: ");      // and MAC address
  Serial.println(WiFi.macAddress());
  server.begin();
  server.on("/", base);               // default webpage
  websocket.begin();
  if(esp_now_init() != 0)             // initialise ESP-NOW
  {
    Serial.println("error initialising ESP-NOW");
    return;
  }                                   // receiver device
```

```
    esp_now_set_self_role(ESP_NOW_ROLE_ SLAVE);
    esp_now_register_recv_cb(receiveDat a);
  }                                       // link to receiveData function
  void receiveData(uint8_t * mac, uint8_t * data, uint8_t len)
  {                                       // copy received data to payload
    memcpy(&payload, data, sizeof(pa yload));
    strMAC = "";
    for (int i = 0; i < 6; i++)           // transmitting MAC address
    {                                     // convert to HEX format
      strMAC = strMAC + String(mac[i ], HEX);
      if (i < 5)strMAC = strMAC + ":";
    }
    strMAC.toUpperCase();                 // convert to upper case
    JsonConvert(strMAC, payload.count, payload.value,
    payload.text);
    websocket.broadcastTXT(json.c_str(), json.length());
  }
  String JsonConvert(String val1, int val2, float val3, String val4)
  {                                       // start with open bracket
    json = "{\"var1\": \"" + val1 + "\",";
                                          // partition with comma
    json += " \"var2\": \"" + String(val2) + "\",";
    json += " \"var3\": \"" + String(val3) + "\",";
    json += " \"var4\": \"" + String( val4) + "\"}";
    // end with close bracket
    return json;
  }
  void base()                             // function to return HTML code
  {
    server.send (200, "text/html", page);
  }
  void loop()
  {
    server.hAndleClient();                // handle server requests
    websocket.loop();                     // handle WebSocket data
  }
```

Реализация протокола WebSocket содержится в разделе JavaScript листинга 14-9, демонстрирующего размещенный на вкладке *buildpage.h* AJAX-код веб-страницы в виде символьной строки `page[ ]`.

Листинг 14-9. AJAX-код веб-страницы WebSocket

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head>
<meta name='viewport' content='width=device-width,
initial-scale=1.0'>
<meta charset='UTF-8'>
<title>ESP-NOW</title>
<style>
body {font-family:Arial}
</style></head>
<body id='initialise'>
<h2>ESP-NOW with ESP8266</h2>
<p>last message at <span id='timeNow'</span></p>
```

```
<p>received from <span id='MAC'> </span></p>
<p>counter <span id='count'>0</span></p>
<p>number <span id='value'>0</span></p>
<p>message <span id='text'> </span></p>
<script>
var wskt;
document.getElementById('initialise').onload = function() {init()};
function init()
{
 wskt = new WebSocket('ws://'+ window.location.hostname + ':81/');
   wskt.onmessage = function(rx)
   {
      var obj = JSON.parse(rx.data);
      document.getElementById('MAC').innerHTML = obj.var1;
      document.getElementById('count').innerHTML = obj.var2;
      document.getElementById('value').innerHTML = obj.var3;
      document.getElementById('text').innerHTML = obj.var4;
      var dt = new Date();
      var tm = dt.toLocaleTimeString ('en-GB');
      document.getElementById('timeNow').innerHTML = tm;
   };
}
</script>
</body></html>
)";
```

Данные от нескольких микроконтроллеров принимаются одним микроконтроллером путем присвоения определенного номера канала каждому передающему микроконтроллеру инструкцией

```
chk = esp_now_add_peer(receiveMAC, ESP_NOW_ROLE_SLAVE, channel, NULL, 0)
```

В скетче MAC-адрес передающего микроконтроллера или значение индекса, включенное в сообщение, используется для назначения принятого сообщения определенному передающему микроконтроллеру. Например, два микроконтроллера ESP8266, выполняющие скетч из листинга 14-6, передавали сообщения каждые две и девять секунд, которые были получены микроконтроллером ESP8266, выполняющим скетч в листинге 14-7. Включение команды `if((mac[5]-0x9B)==0) Serial.print("\t\t\t\t\t\t")` задает смещение, отображающее на мониторе последовательного порта сообщение от микроконтроллера с MAC-адресом, заканчивающимся на 0x9B (см. рис. 14-3).

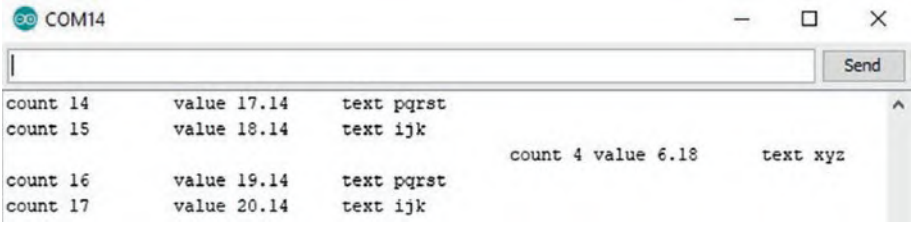


Рисунок 14-3. Сообщения от двух передающих микроконтроллеров ESP8266

Листинги с 14-4 по 14-8 предназначены для микроконтроллера ESP8266. Существует несколько различий в инструкциях ESP-NOW между библиотеками

для микроконтроллеров ESP8266 и ESP32, которые перечислены в табл. 14-1 с подробностями, выделенными жирным шрифтом. Передающий и принимающий микроконтроллеры ESP32 требуют библиотеки *Wi-Fi* и определяются как режим станции[70] Wi-Fi, (`WIFI_STA`) сразу после инструкции `Serial.begin(115200)`. Функция `esp_now_set_self_role` не требуется, и функция `esp_now_peer_info_t` для определения принимающего микроконтроллера ESP32 заменяет функцию `esp_now_add_peer`. В функции `sendData` специально определена переменная состояния отправки; а в функции `receiveData` типы параметров определяются как `const uint8_t` или `int` вместо `uint8_t`.

Таблица 14-1. Инструкции ESP-NOW для микроконтроллеров ESP8266 и ESP32

ESP8266	ESP32
	`#include <WiFi.h>`
`#include <espnow.h>`	`#include <esp_now.h>`
	`WiFi.mode(WIFI_STA )`
`esp_now_set_self_role(ESP_NOW_ROLE_XX)`	`Not required`
`where XX is CONTROLER or SLAVE`	
`chk = esp_now_add_`	`esp_now_peer_info_t receiver;`
`peer(receiveMAC,`	`memcpy(receiver.peer_addr,`
`ESP_NOW_ROLE_SLAVE, channel, NULL, 0)`	`receiveMAC, 6);`
	`receiver.channel = channel;`
	`receiver.encrypt = false;`
	`chk = esp_now_add_peer(&receiver);`
`void sendData(uint8_t * mac,`	`void sendData(const uint8_t * mac, esp_`
`uint8_t chk)`	`now_send_status_t chk)`
`void receiveData(uint8_t *mac,`	`void receiveData(const uint8_t * mac,`
`uint8_t *data, uint8_t len)`	`const uint8_t * data, int len)`

Если полученное сообщение отображается на веб-странице, то требуется канал ESP-NOW между передающим и принимающим микроконтроллерами ESP32 и канал Wi-Fi между принимающим микроконтроллером ESP32, а также беспроводная сеть WLAN.

Передающий микроконтроллер ESP32 определяется как программная точка доступа SoftAP, по умолчанию на первом канале Wi-Fi с настройкой IP-адреса по умолчанию 192.168.4.1, который отображается с помощью инструкции `Serial. println(WiFi.softAPIP())`. Пароль SoftAP должен содержать не менее восьми буквенно-цифровых символов. Инструкции по заданию SoftAP с SSID, паролем и номером канала для передающего микроконтроллера ESP32 приведены в табл. 14-2.

[70] Микроконтроллеры ESP8266 и ESP32 имеют несколько режимов работы в сети Wi-Fi: *режим станции* (Station mode, STA) – контроллер не создает собственную сеть, а подключается к существующей сети Wi-Fi; режим (программной) точки доступа (SoftAP) и совмещенный режим STA + SoftAP (см. также начало главы 7 «Беспроводная локальная сеть» и далее в этой главе). – *Прим. перев.*

Таблица 14-2. Инструкции для передачи и получения на веб-странице для микроконтроллера ESP32

Передатчик ESP32	Приемник ESP32
	`#include <WebServer.h>`
	`WebServer server(80)`
`char ssidAP[ ] = "abcdefg"`	`char ssid[ ] = "xxxx"`
`char passwordAP[ ] = "12345678"`	`char password[ ] = "xxxx"`
`int channelAP = 3`	
`WiFi.mode(WIFI_AP)`	`WiFi.mode(WIFI_AP_STA )`
`WiFi.softAP(ssidAP, passwordAP, channelAP)`	
`Serial.print("Soft-AP IP address ")`	
`Serial.println(WiFi.softAPIP())`	
`WiFi.begin(ssidAP, passwordAP)`	`WiFi.begin(ssid, password);`
`receiver.channel = channelAP`	

Принимающий микроконтроллер ESP32 имеет режим совместной точки доступа и станции, устанавливаемый инструкцией `WiFi.mode(WIFI_AP_STA)` для подключения к SoftAP и к сети Wi-Fi (см. табл. 14-2). SSID сети Wi-Fi и пароль, обозначенные как `xxxx` в табл. 14-2, заменяются SSID и паролем вашего маршрутизатора Wi-Fi. Обратите внимание, что инструкция `WiFi.begin()` для передающего микроконтроллера ESP32 ссылается на сгенерированный SSID и пароль SoftAP, в то время как принимающий микроконтроллер ESP32 ссылается на SSID и пароль маршрутизатора Wi-Fi.

Дополнительная информация об ESP-NOW доступна по адресу www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf, в разделе 3.8 для микроконтроллера ESP8266, а подробная информация о микроконтроллере ESP32 – на docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html.

LoRa

Имеется несколько систем связи, каждая из которых работает на разных частотах, на разных расстояниях и с разной скоростью передачи данных RFID[71], Bluetooth и Bluetooth Low Energy (BLE) имеют более низкие скорости передачи данных, чем связь Wi-Fi, которая, в свою очередь, имеет меньшую дальность действия, чем стандарты мобильных технологий 2G–5G (см. рис. 14-4). LoRa (от *long range*, *большая дальность*) – технология маломощной глобальной сети (LPWAN), разработанная компанией Semtech. Связь LoRa работает с использованием частотной модуляции (FM), а не амплитудной модуляции (AM), использует более низкие

[71] RFID (Radio Frequency IDentification, радиочастотная идентификация) – всем знакомая система обмена данными между различными картами (RFID-метками, транспондерами) и считывателями (ридерами). Отличается тем, что мобильная RFID-метка может работать без источника питания, получая энергию за счет индуцированного в антенне электромагнитного сигнала от считывателя. Не путать с радиочастотными (RF) коммуникациями, описанными в главе 15 «Радиочастотная связь». – *Прим. перев.*

частоты, чем 2,4 ГГц, применяемые для связи Wi-Fi и Bluetooth, и имеет низкое энергопотребление. LoRaWAN – это протокол для создания сетей на основе LoRa. В этой главе описывается связь «точка–точка» на основе LoRa.

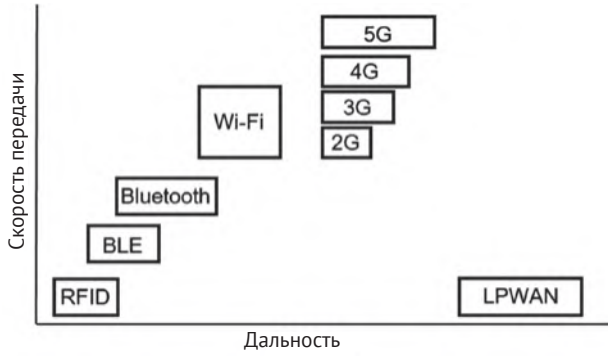


Рисунок 14-4. Коммуникационные технологии

Модули Semtech SX1276 и SX1278 LoRa питаются от 1,8–3,6 В, но отличаются диапазоном частот 137–1020 МГц и 137–525 МГц соответственно, при этом модуль SX1278 применим в Европе (433 МГц), в то время как модуль SX1276 подходит для Европы (868 МГц), Австралии, Северной Америки (915 МГц) и Азии (923 МГц). Оба модуля, SX1276 и SX1278, имеют LoRa SF (spreading factor[72]) 6–12 и полосу пропускания 8–500 кГц, что обеспечивает эффективную скорость передачи данных до 37,5 Кбит/с.

В этой главе используются модуль SX1278 LoRa с частотой 433 МГц и библиотека LoRa от Сандипа Мистры (Sandeep Mistry), доступная в среде Arduino IDE (подробную информацию см. по адресу github.com/sandeepmistry/arduino-LoRa/blob/master/API.md). Модуль LoRa взаимодействует с микроконтроллером ESP8266 или ESP32 с помощью SPI, соединения между модулем LoRa и платой ESP8266 или ESP32 показаны на рис. 14-5 и перечислены в табл. 14-3. Длина антенны модуля LoRa (вывод ANT) равна λ/4, где λ – длина волны, равная *c*/*f*, где *c* есть скорость света (299 792 458 м/с, или приблизительно 300 Мм/с), а *f* есть частота передачи LoRa. Частота передачи LoRa может составлять 433, 868 или 915 МГц, соответственно, требуется антенна длиной 173, 86 или 82 мм[73]. Контакты цифрового ввода-вывода модуля LoRa имеют маркировку DIO.

[72] LoRa spreading factor (SF) – число каналов LoRa с частотным разделением, доступных в определенной полосе спектра, см. далее в данной главе. Подробнее об этом на русском языке см. в статье «*Связь в интернете вещей: LoRa против UNB. Часть 3: технические тонкости*» (https://habr.com/ru/company/unwds/blog/372645/). – *Прим. перев.*

[73] Или специальная направленная антенна для выбранной частоты, например 433 МГц, что позволяет увеличить дальность передачи до 7–10 км на открытой местности. Выпускаются LoRa-модули с уже установленным разъемом для такой антенны. – *Прим. перев.*

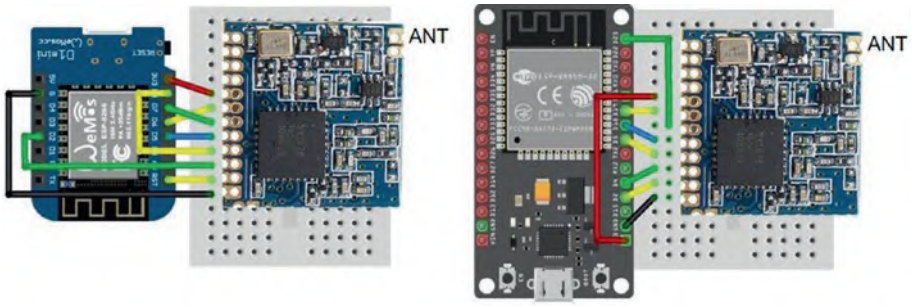


Рисунок 14-5. Модуль SX1278 LoRa с платами LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT

Изображение модуля SX1278 LoRa на рис. 14-5 было увеличено в размерах, чтобы соответствовать отверстиям мини-макета и проиллюстрировать соединения. В действительности 12 соединительных контактов модуля SX1278 LoRa занимают 14 мм, а не 30 мм на макетной плате. Одно из решений состоит в том, чтобы согнуть восемь контактов разъема с длинными контактами, дабы выровнять их с соединениями модуля SX1278 LoRa (см. рис. 14-6)[74].

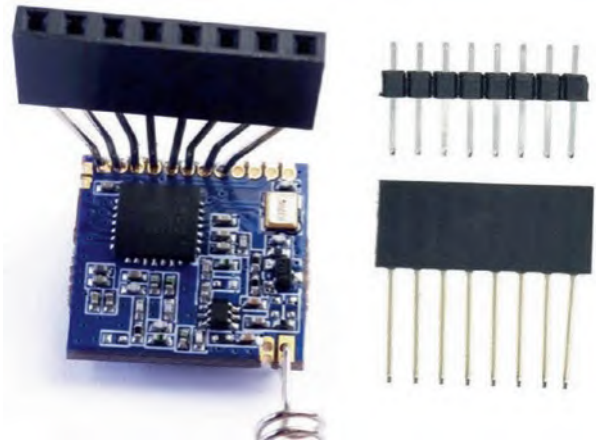


Рисунок 14-6. Разъемы с длинными и обычными выводами

Таблица 14-3. Соединения модуля SX1278 LoRa с микроконтроллерами ESP8266 и ESP32

Компонент	ESP8266	ESP32
SX1278 GND (pin 1)		
SX1278 DIO1		
SX1278 DIO2		
SX1278 DIO3		

[74] Или приобрести специально для макетирования модуль SX1278 LoRa с контактами с обычным шагом 2,54 мм, а не 1,27 мм, как у автора. Модули выпускаются во множестве модификаций, см. интернет-поиск по запросу «SX1278 LoRa». – *Прим. перев.*

Окончание табл. 14-3

Компонент	ESP8266	ESP32
SX1278 VCC	3V3	3V3
SX1278 MISO	D6	GPIO 19
SX1278 MOSI	D7	GPIO 23
SX1278 SLCK	D5	GPIO 18
SX1278 NSS (CSS)	D8	GPIO 5
SX1278 DIO0	D2	GPIO 4
SX1278 REST	RST	GPIO 2
SX1278 GND (pin 12)	GND	GND
OLED GND	GND	GND
OLED VCC	3V3	3V3
OLED SCK	D1	GPIO 22
OLED SDA	D2	GPIO 21

Скетч связи LoRa с микроконтроллером ESP8266 приведен в листинге 14-10. Частота передачи LoRa, равная 433, 868 или 915 МГц, определяется с помощью команды `LoRa.begin(NE6)`, где *N* равно 433, 868 или 915. Контакты подключения модуля LoRa для выбора микросхемы (CSS), сброса (RST) и вывода прерывания (DIO0) определены в скетче, поскольку значения библиотеки LoRa по умолчанию иные – GPIO 10, GPIO 9 и GPIO 2.

Когда вывод сброса модуля LoRa подключен к выводу сброса микроконтроллера ESP8266 или ESP32, вывод сброса модуля LoRa определяется как –1. Контакты SPI определяются по умолчанию. LoRa-фактор SF (SF6–SF12) распределяет передачу по доступной полосе пропускания, при этом более высокий SF увеличивает дальность, но снижает скорость передачи. Библиотека *LoRa* может устанавливать значения полосы пропускания от 7,8 кГц до 250 кГц, при этом меньшая полоса пропускания соответствует большей дальности передачи. Полоса пропускания сигнала по умолчанию 125 кГц может изменяться с помощью инструкции `LoRa.setSignalBandwidth(N)` для полосы пропускания *N* с возможными значениями, перечисленными на веб-сайте библиотеки *LoRa*. Например, для полосы пропускания 31,25 кГц *N* устанавливается равным 31.25E3. Максимальный размер одной передачи LoRa с библиотекой *LoRa* составляет 255 байт.

В скетче с интервалом в пять секунд передается сообщение. Интервал отображается на OLED-экране вместе с полученным сообщением от принимающего микроконтроллера (см. рис. 14-7). Значение в полученном сообщении равно переданному пакету, при условии что сообщения не были потеряны. Возвращаемое сообщение подтверждает получение переданного сообщения принимающим микроконтроллером. Принимающий микроконтроллер отображает количество секунд между полученными сообщениями, сигнал RSSI (индикатор уровня принятого сигнала) и SNR (отношение сигнал/шум), а затем полученное сообщение.

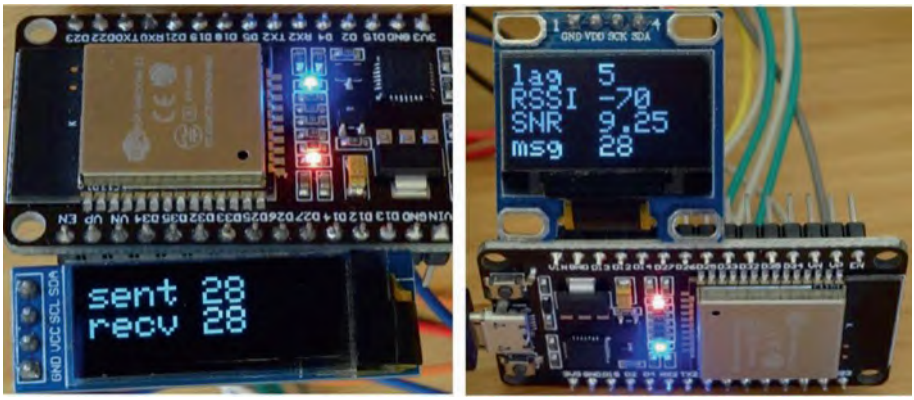


Рисунок 14-7. Типовой пример приема сообщений LoRa с обратной связью

Листинг 14-10. Передающий модуль LoRa с обратной связью

```
#include <SPI.h>                                  // include SPI and
#include <LoRa.h>                                 // LoRa libraries
int CSS = D8;                                     // define SX1278 pins
int RST = -1;                                     // RESET pin
int DIO0 = D2;                                    // interrupt pin
#include <Wire.h>                                 // include libraries for OLED
#include <Adafruit_SSD1306.h>
int width = 128;                                  // OLED dimensions
int height = 32;                                  // associate oled with library
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int counter = 0;
unsigned long lastTime;
String packet, recv;
int packetSize;                                   // size of received message
void setup()
{
  digitalPinToInterrupt(DIO0);                    // set pin as interrupt
  LoRa.setPins(CSS, RST, DIO0);                   // define LoRa module pins
  LoRa.setSpreadingFactor(9);                     // define spreading factor
  LoRa.setSignalBandwidth(62.5E3);                // set bandwidth to 62.5kHz
  while (!LoRa.begin(433E6)) delay(500);          // 433MHz transmission
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3 C);
                                                  // OLED display I2C address
  oled.setTextColor(WHITE);                       // set font color
  oled.setTextSize(2);                            // text size 12×16 pixels
  oled.display();
}
void loop()
{
  if(millis() - lastTime > 5000)                  // 5s transmission interval
  {
    screen();                                     // OLED display function
    packet = String(counter);                     // create packet
    LoRa.beginPacket();                           // start LoRa transmission
    LoRa.print(packet);                           // send packet
    LoRa.endPacket();                             // close LoRa transmission
    counter++;                                    // increment counter
```

```
      lastTime = millis();                    // update transmission time
    }
    packetSize = LoRa.parsePacket();          // detect received packet
    if (packetSize > 0)
    {
      recv = "";                              // read packet
      while(LoRa.available()) recv = recv + ((char)(LoRa.read()));
      screen();                               // OLED display function
    }
}
void screen()                                 // function for OLED display
{
  oled.clearDisplay();
  oled.setCursor(0,0);
  oled.print("sent ");oled.println(pac ket); // transmitted value
  oled.print(recv);                           // received message
  oled.display();
}
```

Во время приема пакетов модуль LoRa измеряет в децибелах (дБ) показатель уровня принимаемого сигнала (RSSI) и отношение сигнал/шум (SNR). Мощность принимаемого сигнала составляет $10^{(дБ/10)}$ МВт. RSSI колеблется от 0 дБм[75] до –120 дБм, при этом значение, превышающее –50 дБм, указывает на сильный сигнал, в то время как слабый сигнал имеет RSSI менее –90 дБм. SNR – это разница между значениями RSSI сигнала и фонового шума, при этом положительный SNR указывает на то, что принятый сигнал превышает базовый уровень шума. LoRa может работать ниже базового уровня шума, представляющего собой нормальный предел чувствительности сигнала.

Типовой скетч для приема сообщений LoRa приведен в листинге 14-11, в котором на OLED-дисплее отображается интервал между приемом сообщения, RSSI, SNR и само сообщение. Полученное сообщение считывается как байт, который преобразуется в символ с помощью инструкции `(char)(LoRa.read())`. При получении сообщения скетч также возвращает полученное сообщение передающему микроконтроллеру в качестве обратной связи. Библиотека *Adafruit SSD1306* ссылается на библиотеку *Adafruit GFX*, поэтому инструкция `#include <Adafruit_GFX.h>` не требуется.

Листинг 14-11. Типовой пример приема сообщений LoRa

```
#include <SPI.h>                      // include SPI and
#include <LoRa.h>                     // LoRa libraries
int CSS = D8;                         // define SX1278 pins
int RST = -1;                         // RESET pin
int DIO0 = D2;                        // interrupt pin
int width = 128;                      // OLED dimensions
int height = 64;
#include <Wire.h>                     // include libraries for OLED
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 oled(width, height, & Wire, -1);
                                     // Reset pin not required
String packet;
```

[75] дБм – единица измерения мощности сигнала относительно опорной мощности 1 мВт. – *Прим. перев.*

```
int RSSI, packetSize, interval;
float SNR;
unsigned long lastTime = 0;
void setup()
{
  digitalPinToInterrupt(DIO0);              // set pin as interrupt
  LoRa.setPins(CSS, RST, DIO0);             // define LoRa module pins
  while (!LoRa.begin(433E6)) delay(500 );   // 433MHz transmission
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3 C);
                                            // OLED display I2C address
  oled.setTextColor(WHITE);                 // set font color
  oled.setTextSize(2);                      // text size 12×16 pixels
  oled.display();
}
void loop()
{
  packetSize = LoRa.parsePacket();          // detect received packet
  if (packetSize > 0)
  {
    interval = round((millis() - lastT ime)/1000);  // interval (s)
    lastTime = millis();                    // update message time
    packet = "";                            // read packet
    while(LoRa.available()) packet = packet + ((char)(LoRa.read()));
    RSSI = LoRa.packetRssi();
    SNR = LoRa.packetSnr();                 // signal : noise
    screen();                               // OLED display function
    LoRa.beginPacket();                     // start LoRa transmission
    LoRa.print("recv " + packet);           // send packet
    LoRa.endPacket();                       // close LoRa transmission
  }
}
void screen()                               // function for OLED display
{
  oled.clearDisplay();
  oled.setCursor(0,0);
  oled.print("lag ");oled.pr intln(interval);
                                            // display time since last message
  oled.print("RSSI ");oled.pr intln(RSSI);
                                            // display interval, RSSI and SNR
  oled.print("SNR ");oled.println(SNR);
  oled.print("msg ");oled.print (packet);
                                            // display received message
  oled.display();
}
```

Протокол WebSocket позволяет серверу передавать информацию клиенту, при этом информация отображается на веб-странице. Листинг 14-12 автоматически обновляет веб-страницу с сообщением и временем получения (см. рис. 14-8). Приложение отображает информацию с датчиков.

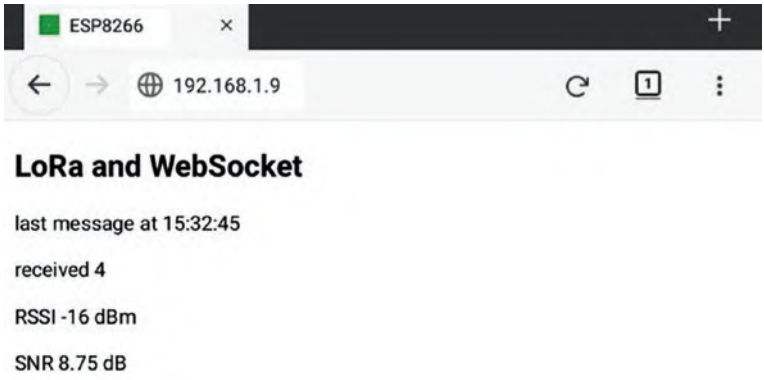


Рисунок 14-8. Прием LoRa с помощью WebSocket

Листинг 14-12 предназначен для платы разработки ESP8266 в качестве сервера. Инструкции библиотеки веб-сервера для микроконтроллера ESP8266

```
#include <ESP8266WebServer.h>
ESP8266WebServer server
```

при использовании микроконтроллера ESP32 заменяются на

```
#include <WebServer.h>
WebServer server(80); // requires a port number
```

Библиотека веб-сервера ссылается на библиотеку Wi-Fi, поэтому инструкции `#include <ESP8266WiFi.h>` или `#include <WiFi.h>` не требуются. Номера выводов *CSS*, *RST* и *DIO0* для микроконтроллера ESP32 изменены на 5, 2 и 4.

Листинг 14-12. Приемный модуль LoRa и WebSocket

```
#include <ESP8266WebServer.h>         // include Webserver library
ESP8266WebServer server;              // associate server with library
#include <WebSocketsServer.h>         // include WebSocket library
WebSocketsServer websocket = WebSoc ketsServer(81); // set WebSocket port 81
#include "buildpage.h"                // webpage AJAX code
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi SSID
char password[] = "xxxx";             // change xxxx to Wi-Fi password
String message, json;
int RSSI;
float SNR;
#include <SPI.h>                      // include SPI library
#include <LoRa.h>                     // and LoRa library
int CSS = D8;                         // define SX1278 pins
int RST = -1;                         // RESET pin
int DIO0 = D2;                        // interrupt pin
int packetSize;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  WiFi.begin(ssid, password);         // initialise and connect Wi-Fi
  while (WiFi.status() != WL_CONNECTED) delay(500);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());     // display web server IP address
```

```
    server.begin();
    server.on("/", base);              // load default webpage
    websocket.begin();                 // initialise WebSocket
    digitalPinToInterrupt(DIO0);       // set pin as interrupt
    LoRa.setPins(CSS, RST, DIO0);      // define LoRa module pins
    while (!LoRa.begin(433E6)) delay(500) ; // 433MHz transmission
    Serial.println("LoRa connected");
}
void loop()
{
    server.handleClient();             // handle HTTP requests
    websocket.loop();                  // handle WebSocket data
    packetSize = LoRa.parsePacket();   // detect received packet
    if (packetSize > 0)
    {
      message = "";                    // read packet
      while(LoRa.available()) message = message + ((char)(LoRa.read()));
      RSSI = LoRa.packetRssi();
      SNR = LoRa.packetSnr(); // signal : noise
      JsonConvert(message, RSSI, SNR); // convert to JSON format
      websocket.broadcastTXT(json.c_str(), json.length());
    }
}
String JsonConvert(String val1, int val2, float val3)
{                                      // start with open bracket
    json = "{\"var1\": \"" + String(val1 ) + "\",";
                                       // partition with comma
    json += " \"var2\": \"" + String(val2) + "\",";
    json += " \"var3\": \"" + String(val3 ) + "\"}";   // end with close bracket
    return json;
}
    void base()                        // function to return HTML code
{
    server.send (200, "text/html", page);
}
```

Соответствующий код AJAX для веб-страницы приведен в листинге 14-13. Когда веб-страница загружена, функция инициализации открывает подключение к WebSocket по адресу `ws://web server IP address:81/`. Когда модуль SX1278 LoRa получает сообщение, протокол WebSocket пересылает сообщение клиенту в виде переменной `rx.data`, которая разбирается на переменные `rxText`, `RSSI` и SNR для отображения на веб-странице. Время получения сообщения определяется из JavaScript-ссылки на дату, как описано в главе 9 («WebSocket»), подробная информация доступна по адресу www.w3schools.com/Jsref/jsref_ obj_date.asp.

Листинг 14-13. WebSocket и AJAX-код для веб-страницы

```
char page[] PROGMEM = R"(
<!DOCTYPE html><html>
<head>
<meta name='viewport' content='width=device-width,
initial-scale=1.0'>
<meta charset='UTF-8'>
<title>ESP8266</title>
<style>
body {font-family:Arial}
```

```
</style></head>
<body id='initialise'>
<h2>LoRa and WebSocket</h2>
<p>last message at <span id='timeNow'</span></p>
<p>received <span id='rxText'> </span></p>
<p>RSSI <span id='RSSI'>0</span> dBm</p>
<p>SNR <span id='SNR'>0</span> dB</p>
<script>
var wskt;
document.getElementById('initialise').onload = function()
{init()};
function init()                                  // open WebSocket
{
   wskt = new WebSocket('ws://' + window.location.hostname + ':81/');
   wskt.onmessage = function(rx)
   {
      var obj = JSON.parse(rx.data);
      document.getElementById('rxText').innerHTML = obj.var1;
      document.getElementById('RSSI').innerHTML = obj.var2;
      document.getElementById('SNR').innerHTML = obj.var3;
      var dt = new Date();
      var tm = dt.toLocaleTimeString ('en-GB');
      document.getElementById('timeNow').innerHTML = tm;
   };
}
</script>
</body></html>
)";
```

Итоги

Технологии ESP-NOW и LoRa были описаны для связи между микроконтроллерами ESP8266 и ESP32. ESP-NOW не требует дополнительных компонентов, кроме платы ESP8266 или ESP32, в то время как LoRa требует модуля SX1278. ESP-NOW работает на частоте 2,4 ГГц, что выше, чем рабочие частоты LoRa 137–525 МГц, при этом протоколы имеют ограничения в 250 и 255 байт на сообщение соответственно. LoRa имеет большую дальность передачи, чем ESP-NOW. С помощью ESP-NOW передающий микроконтроллер получает ответ о том, что сообщение было получено, с информацией, отображаемой на последовательном мониторе или на веб-странице, обновленной с помощью процедуры WebSocket. Полученное с помощью LoRa сообщение отображалось либо на OLED-экране, подключенном к плате ESP8266 или ESP32, либо на веб-странице, с использованием процедуры WebSocket.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN D1 mini (WeMos) или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU 2 шт.
- Приемопередатчик LoRa: модуль SX1278 2 шт.
- OLED-дисплей: 128×32 и 128×64 пиксела

Глава 15

Радиочастотная связь

В телекоммуникациях и обработке сигналов информация передается на несущей радиоволне с помощью модуляции ее амплитуды или частоты. Радиоволны имеют частоты между 20 кГц и 300 ГГц, соответствующие длинам волн от 15 км до 1 мм, и перемещаются со скоростью света. Радиоволны с частотами выше 300 МГц называются микроволнами[76]. Для радиочастотной (RF) связи с амплитудной модуляцией (AM) амплитуда передачи пропорциональна амплитуде сигнала, а частота передачи постоянна (см. рис. 15-1). И наоборот, для частотной модуляции (FM) амплитуда передаваемого сигнала постоянна, а частота передачи пропорциональна амплитуде сигнала. AM используется для связи между двусторонними рациями, радиостанциями в гражданских диапазонах частот и авиационными УКВ-радиостанциями. FM используется в телеметрии и радиовещании, так как FM имеет более высокое отношение сигнал/шум, чем AM при той же мощности передачи.

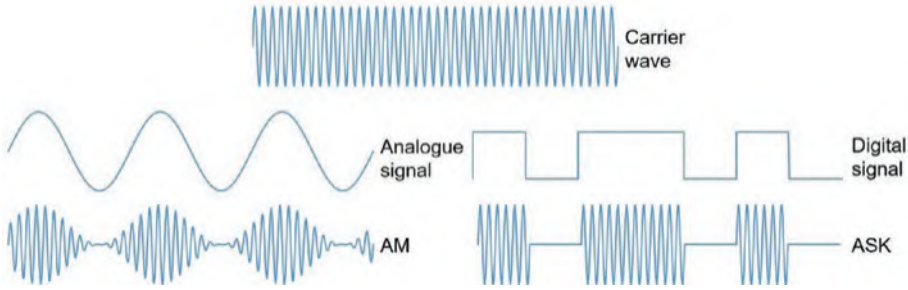


Рисунок 15-1. Амплитудная модуляция (AM) и амплитудная манипуляция (ASK)

Амплитудная манипуляция (*amplitude shift keying*, ASK) – это разновидность амплитудной модуляции для представления цифрового сигнала, такого как *1234*, для которого потребуется четыре различные амплитуды сигнала. Простейшая форма ASK типа «включение–выключение», называемая OOK (On-Off Keying), состоит из двоичного сигнала, представленного либо передачей сиг-

[76] Согласно отечественным и западным стандартам, диапазон от 300 МГц до 3 ГГц носит название ультравысоких частот (УВЧ, англ. *ultra high frequency*, UHF). Название «микроволны» (microwaves) как термин для обозначения какого-либо диапазона, в силу его неопределенности, употреблять не принято. – *Прим. перев.*

нала несущей, либо отсутствием сигнала. На рис. 15-1 сигнал 1011010 представлен в цифровой форме и как сигнал ASK.

Пары передатчика и приемника 433 МГц (см. рис. 15-2) представляют собой простые устройства, исключающие передачу голоса, малой дальности и предназначенные для передачи сигнала при низком уровне мощности, обычно в режиме, не требующем лицензирования. Передатчик – меньший из пары передатчика и приемника на рис. 15-2. Стандартная пара передатчика прямой модуляции и приемника прямого усиления часто встречается под названиями MX-FS-03V и MX-05V или FS1000A и MX-RM-5V соответственно. Супергетеродинная пара передатчик и приемник (рис. 15-2, справа) часто называется WL102-341 и WL101-341. Супергетеродинный передатчик использует смешивание частот для преобразования передаваемого сигнала в сигнал с промежуточной частотой, который обрабатывается более эффективно, чем сигнал на исходной несущей частоте. Супергетеродин – это соединение слов *супер* (например, supersonic – частота выше человеческого слуха), *гетеро* (разных) и *дин* (единица силы)[77].

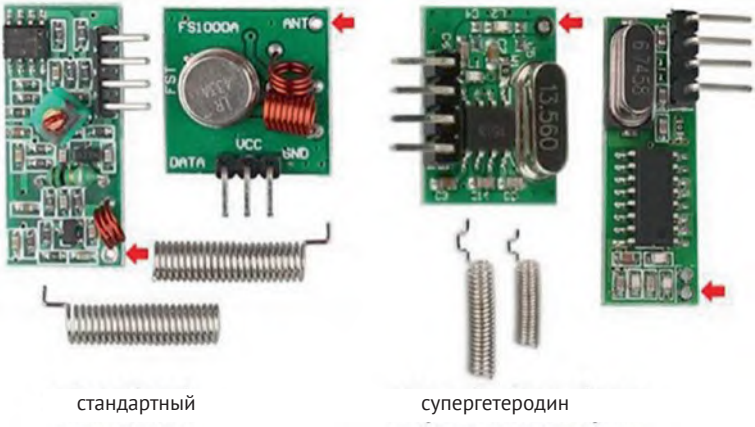


Рисунок 15-2. Пары передатчик/приемник стандарта 433 МГц (обычного прямого усиления и супергетеродинные)

Пары передатчика и приемника на рис. 15-2 показаны со спиральными антеннами, но передача и прием улучшаются с помощью прямых антенн длиной $\lambda/2$ или $\lambda/4$, где λ – длина волны, равная c/f, где c – скорость света (299 792 458 м/с или приблизительно 300 Мм/с) и f – частота несущей волны. Для частоты несущей волны 433,29 МГц антенна длиной в четверть длины волны имеет длину 173 мм. На рис. 15-2 точки подключения антенны обозначены красными стрелками.

Соединения для передатчика и приемника с платами ESP8266 и ESP32 показаны на рис. 15-3 и 15-4 соответственно, одинаково как для супергетеродин-

[77] Термин *гетеродин* (генератор промежуточной частоты) происходит от греч. ἕτερος (гетеро) – иной и δύναμις (динамис) – сила. Приставка «супер» несет в термине «супергетеродин» смысл «поверх гетеродина», а не означает какое-либо преувеличение. – *Прим. перев.*

ных, так и для стандартных устройств. У платы супергетеродинного передатчика нужно соединить между собой контакты + и EN, у супергетеродинного приемника контакты вывода данных DO или два немаркированных контакта на плате обычного приемника[78]. Понадобится библиотека *RH_ASK* от Майка Макколи (Mike McCauley), которая загружается с www.airspayce.com/mikem/arduino/RadioHead, и потом библиотека *rc-switch* от Суата Озгура (Suat Özgür), доступная в Arduino IDE.

Для обеих библиотек контакты передачи и приема данных RF подключены к контактам D1 и D2 платы ESP8266 или контактам 26 и 27 платы ESP32 (см. табл. 15-1), при этом вывод приема данных определен как вывод прерывания для библиотеки *rc-switch*. В листингах 15-1 и 15-2 инструкция для определения выводов платы ESP8266, `RH_ASK rf (2000, D2, D1, 0)`, заменяется инструкцией `RH_ASK rf (2000, 27, 26, 0)` для микроконтроллера ESP32. Выбор контактов передачи и приема является произвольным.

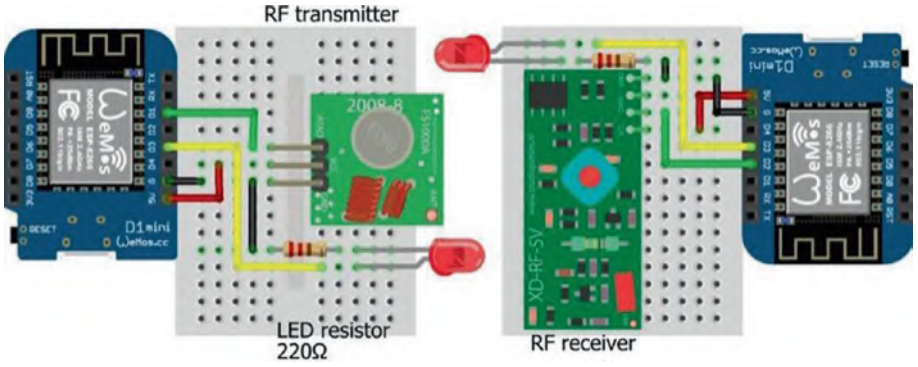


Рисунок 15-3. Радиочастотный передатчик и приемник 433 МГц с LOLIN (WeMos) D1 mini

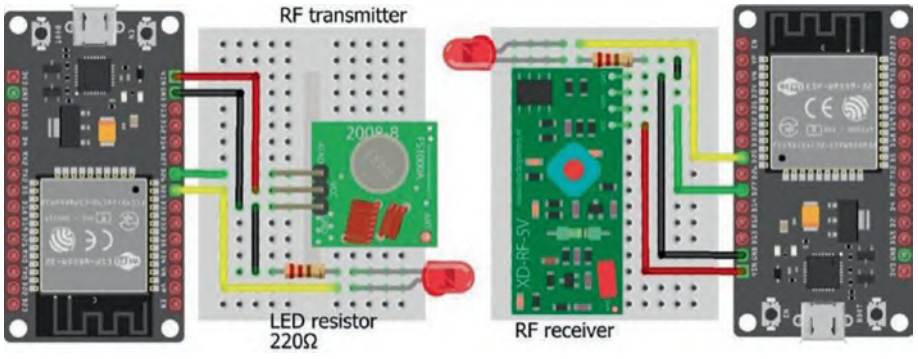


Рисунок 15-4. Радиочастотный передатчик и приемник 433 МГц с платой ESP32 DEVKIT DOIT

[78] Контакты, помеченные DO (выход данных) на плате супергетеродинного варианта, а также пара контактов аналогичного назначения (обычно помечены надписью *Data*, иногда немаркированные) у приемника прямого усиления соединять между собой необязательно, они запараллелены на плате, можно подключаться к любому из них. – *Прим. перев.*

Таблица 15-1. Соединения передатчика и приемника для микроконтроллеров ESP8266 и ESP32

	Передатчик			Приемник		
Супергетеродин	– (минус)	+ (плюс), EN	DAT	GND	DO	VCC
Стандартный	GND	VCC	Data	GND	Data или немаркир.	VCC
Подключить к						
ESP8266	GND	5V	D1	GND	D2	5V
ESP32	GND	VIN	GPIO 26	GND	GPIO 27	VIN

Передача и прием текста

Библиотека *RH_ASK* используется для передачи сообщений, содержащих символьные строки. Драйвер библиотеки *RH_ASK* использует прерывание, управляемое таймером, для генерации восьми прерываний на бит за период, при этом Timer1 является таймером по умолчанию. Несколько других библиотек используют Timer1, например библиотека Servo, но драйвер *RH_ASK* может использовать Timer2 вместо Timer1, если раскомментировать строку `#define RH_ASK_ARDUINO_USE_TIMER2` в строке 32 файла *RH_ASK.cpp*, который находится в папке *Arduino/libraries/RadioHead*. Инструкция `RH_ASK rf (2000, receive pin, transmit pin, 0)` определяет скорость передачи в битах в секунду (бит/с) вместе с выводами приема данных и передачи данных соответственно.

Для библиотеки *RH-ASK* сигналы форматируются с помощью 36-битной обучающей преамбулы, состоящей из 18 пар битов 0 и 1, 12-битного начального символа (0xB38), 1-байтового числа для размера поля данных (payload), 4-байтового заголовка, *N* байт, содержащих *N* символов сообщения с максимумом 60 символов на сообщение и 2 байта для контрольной последовательности (Frame Check Sequence, FCS[79]). Размер поля данных – количество символов в сообщении плюс семь (1 для длины поля данных; 4 для заголовка и 2 для FCS). Кроме преамбулы и начального символа, каждый байт разбивается на старшие и младшие 4-разрядные последовательности, которые перекодируются в 6-разрядные последовательности и передаются младшим битом (LSB) вперед. Таблица 15-2 обеспечивает отображение 4-разрядных последовательностей (от 0x0 до 0xF) в 6-разрядные последовательности (от 0xD до 0x34), представленные в шестнадцатеричной форме. Длина всего сигнала составляет $36 + 12 + (N + 7) \times 12$ бит для сообщения из *N* символов. Передача сообщения из 60 символов занимает 426 мс при скорости передачи 2 Кбит/с.

Таблица 15-2. Отображение 4-разрядных последовательностей в 6-разрядные в библиотеке RH_ASK

4-bit	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
6-bit	0xD	0xE	0x13	0x15	0x16	0x19	0x1A	0x1C
4-bit	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
6-bit	0x23	0x25	0x26	0x29	0x2A	0x2C	0x32	0x34

[79] FCS (Frame Check Sequences) и payload – термины из стандартов передачи в сети Ethernet. – *Прим. перев.*

Например, символ A имеет код ASCII 65 и шестнадцатеричный код *0x41* со старшим и младшим шестнадцатеричными разрядами (тетрадами, 4-разрядными последовательностями) *0x4* (0100) и *0x1* (0001). Они перекодируются в 6-разрядные последовательности *0x16* (010110) и *0xE* (001110) соответственно. 6-разрядные последовательности сначала переворачиваются из последовательности старшим битом (MSB) вперед в последовательность младшим битом (LSB) вперед. Перекодированная старшая последовательность, а затем младшая последовательность тогда выглядит как (011010)(011100), что эквивалентно (0110)(1001)(1100), выражению в виде трех тетрад, соответствующих шестнадцатеричным кодам *0x6, 0x9* и *0xC*. Таким образом символ сообщения «A» преобразуется в сигнал, соответствующий *69c*, как показано на рис. 15-5.

Цель перекодирования состоит в том, чтобы гарантировать, что 12-битная последовательность содержит 6 бит со значением один и 6 бит со значением ноль, она называется сбалансированной по постоянному току (DC-balanced)[80]. В примере передачи символа «A» две 4-разрядные последовательности, соответствующие шестнадцатеричному коду 0x41, являются (0100)(0001), которые не сбалансированы по постоянному току, но после перекодирования три 4-разрядные последовательности (0110)(1001)(1100) будут отвечать требованию DC-balanced.

Анализ полного сигнала для сообщения «*ABC*», захваченного логическим анализатором с помощью программы *sigrok PulseView*, с использованием OOK-декодера типа *NRZ* и длины фильтра 4 с отображением OOK-визуализации *Nibble-Hex* и смещением синхронизации на –1 показан на рис. 15-5. Программу и руководство *sigrok PulseView* можно скачать по адресу sigrok.org. После 36-битной обучающей преамбулы и начального символа *0xB38* длины поля данных = 10 (длина сообщения из трех символов плюс семь), которая отображается как *b19*. Эта величина выводится точно так же, как для предыдущего примера с символом «A». Число 10 имеет шестнадцатеричный код 0x0A со старшими и младшими шестнадцатеричными кодами (*0x0* и *0xA*), соответствующими *0xD* и *0x26*, которые имеют транспонированное двоичное представление (1011)(0001)(1001), соответствующее шестнадцатеричному коду *b19*. Далее следует последовательность заголовка сигнала, подобно тому, как шестнадцатеричная последовательность *0x6*, *0x9, 0xC* для символа сообщения A отображается как *69c*, аналогично символы сообщения B и C с шестнадцатеричными последовательностями *0x6, 0xB, 0x2* и *0x6, 0xA, 0xA* отображаются как *6b2* и *6aa* соответственно.

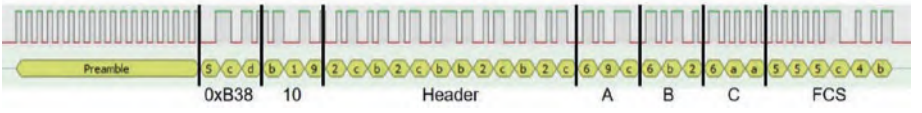


Рисунок 15-5. Результат логического анализа сигналов в формате RH_ASK

[80] Термин DC-balanced означает, что последовательность битов при передаче в виде «есть импульс – нет импульса» будет в среднем сбалансирована по постоянной составляющей, не имеющей выбросов. Для наглядности можно пропустить такую последовательность через фильтр высокой частоты, тогда среднее за период значение радиосигнала ASK будет равно нулю. – *Прим. перев.*

Листинги 15-1 и 15-2 иллюстрируют использование библиотеки *RH_ASK* для передачи и приема сообщения переменной длины, состоящего из текста, действительного числа и целого числа, с максимальной длиной сообщения 60 символов. В листинге 15-1 генерируется пример сообщения с номером передачи `val`, определяемым на основе времени выполнения скетча, деленного на время интервала между передачами, *timelag*. Отдельная текстовая строка для каждого сообщения генерируется инструкцией `text[val%3]`, которая вычисляет остаток от деления переменной `val` на три, учитывая три строки в текстовом массиве. Сообщение содержит текст, действительное число `1.2*val` и целое число `val`, разделенные запятыми. Например, пятое переданное сообщение будет `ijkl,6.0,5`[81]. Интервал между передачами устанавливается равным 2 с, и светодиод включается или выключается после каждого переданного или принятого сообщения. Возможно, потребуется повторить инструкцию `rf.send((uint8_t *) msg, strlen(msg))` для длинного сообщения[82].

Инструкция `msg = message.c_str()` использует команду C++ для создания массива `msg`, равного строке сообщения с завершающим нулевым символом \0, и возвращает указатель `msg` на массив. Если строка определена с помощью инструкции `String text = "abc"`, то нулевой символ \0 автоматически включается в конец строки. Когда строка массива определяется как `String text[4] = {'a', 'b', 'c', '\0'}`, нулевой символ включается в качестве последнего элемента массива, при этом элементы массива определяются как символы, на что указывает использование одинарного апострофа, а не кавычки, как в случае строки.

Листинг 15-1. Передача сообщения с помощью библиотеки RH_ASK

```
#include <RH_ASK.h>                    // include the RH_ASK library
#include <SPI.h>                       // SPI library required to compile
RH_ASK rf (2000, D2, D1, 0);           // associate rf with RH_ASK lib
int LEDpin = D3;                       // define LED pin
String text[] = {"abcdef", "ijkl", "rst"};
                                       // strings of different lengths
const char * msg;                      // pointer to array with message
String message;
int timelag = 2000;                    // interval between transmissions
int LED = 0;                           // initial LED state
int val, len, spd;
void setup()
{
   Serial.begin(115200);               // Serial Monitor baud rate
   rf.init();                          // initialise radio transmission
   pinMode(LEDpin, OUTPUT);            // define LED pin as output
```

[81] Внесем некоторую ясность в это запутанное объяснение, ориентируясь на текст скетча 15-1: номер передачи *val* представляет собой частное от деления числа прошедших миллисекунд *millis()* по системному счетчику на величину *timelag* (=2000). Несложно убедиться, что остаток от деления на три полученной таким образом величины *val* может принимать только три значения 0, 1 и 2, т. е. не выходит за пределы индекса строкового массива *text[]* (см. 5-ю строку скетча). Иными словами, текст в начале сообщения будет принимать одно из трех предопределенных значений "abcdef", "ijkl" или "rst", выбранных автором произвольно. – *Прим. перев.*

[82] Очевидно, повторять необходимо в случае, если длина сообщения превышает оговоренные выше 60 символов и его приходится делить на части. – *Прим. перев.*

```
  len = rf.maxMessageLength();       // get maximum message length
  spd = rf.speed();          // get transmission speed
  Serial.print("max message length: "); // display message length
  Serial.println(len);
  Serial.print("transmission speed: "); // and transmission speed
  Serial.println(spd);
}
void loop()
{
  val = millis()/timelag;    // transmission number
  message = text[val%3] + "," + String(1.2*val) + "," + String(val) + ",";
  Serial.println(message);   // display transmitted string
  msg = message.c_str();             // convert string
  rf.send((uint8_t *)msg, strlen(msg)); // transmit signal
  rf.waitPacketSent();               // wait for transmission to finish
  LED = 1 - LED;
  digitalWrite(LEDpin, LED);         // turn on or off LED
  delay(timelag);
}
```

В листинге 15-2 количество разделенных запятыми элементов в сообщении определяется в скетче. Инструкция `const int nItem = 3` позволяет неявно определять размер двух массивов, `text[nItem]` и `comma[nItem+1]`. При получении нового сообщения пустая строка увеличивается с каждым символом сообщения, а позиции запятой определяются с помощью инструкции `message.indexOf("x", y)`, которая определяет положение подстроки *x* в строке `message`, начиная с позиции *y*. Подстроки сообщения генерируются с помощью инструкции `message.substring(x, y)`, которая является подстрокой строки `message` между позициями *x* (включительно) и *y* (исключительно). Инструкции `string.toFloat()` и `string.toInt()` преобразуют строку в действительное число и целое число соответственно. Более подробная информация о разборе текста на подстроки приведена в главе 8 («Обновление веб-страницы»). Переменные `valFlt` и `valInt` в листинге 15-2 созданы для демонстрации использования преобразованного текста в вычислениях.

Листинг 15-2. Получение сообщений с помощью библиотеки RH_ASK

```
#include <RH_ASK.h>                  // include the RH_ASK library
#include <SPI.h>                     // SPI library required to compile
RH_ASK rf (2000, D2, D1, 0);         // associate rf with RH_ASK lib
int LEDpin = D3;                     // define LED pin
uint8_t msg[RH_ASK_MAX_MESSAGE_LEN] ; // maximum message length
const int nItem = 3;                 // number of items in message
String text[nItem];                  // define text array
int comma[nItem+1];                  // comma positions in message
int LED = 0;                         // initial LED state
String message;
float valFlt;
int valInt;
void setup()
{
  Serial.begin(115200);              // Serial Monitor baud rate
  rf.init();                         // initialise radio transmission
  pinMode(LEDpin, OUTPUT);           // define LED pin as output
}
```

```
void loop()
{                                       // message length based on new message
   uint8_t msglen = sizeof(ms g);
   if (rf.recv(msg, &msglen) )
                                        // message of correct length available
   {
   message = "";                        // increment message with each character
      for (int i=0; i<msglen; i++) message = message +
      char(msg[i]);
      comma[0] = -1;
      for (int i=0; i<nItem; i++)       // comma positions in message
      {                                 // get substrings between commas
         comma[i+1] = message.indexOf(",", comma[i]+1);
         text[i] = message.substring(comma[i]+1, comma[i+1]);
         Serial.print(text[i]);         // print message substring
         Serial.print(" ");
      }
      valFlt = text[1].toFloat();       // second substring to float
      valInt = text[2].toInt();         // third substring to integer
      Serial.print(text[0]);Serial.print("\t");
      Serial.print(valFlt + 0.05);Serial.print("\t");
      Serial.println(valInt * 2);
      LED = 1 - LED;
      digitalWrite(LEDpin, LED);        // turn on or off LED
   }
}
```

Декодирование сигналов дистанционного управления

Библиотека *rc-switch* используется для декодирования сигналов от беспроводных цифровых радиопультов дистанционного управления[83]. Библиотека *rc-switch* от Суата Озгура (Suat Özgür) доступна в среде Arduino IDE. Передаваемый сигнал имеет специальный трехуровневый (tri-state) формат кода в виде пар битов (00), (11) и (01), представляющих значения 0, 1 и F. Например, десятичное число 19 имеет двоичное представление 010011; если считать старший бит (MSB) первым, то пары битов равны (01), (00) и (11), что приводит к трехуровневому формату *F01*. Коды для беспроводных цифровых кнопок дистанционного управления получены с помощью скетча в листинге 15-3.

Например, радиочастотные коды для кнопки питания и кнопки «25 %» беспроводного радиочастотного пульта дистанционного управления (показанного на рисунке) равны *3163905* и *3163913* соответственно в десятичном формате. Двоичное представление кода *3163905* из (00)(11)(00) (00)(01)(00)(01)(11)

[83] Подчеркнем, что речь здесь идет именно о радиопультах, а не о привычных для поклонников Arduino инфракрасных пультах ДУ. Следует также учесть, что с помощью информации из этого раздела у вас получится работать далеко не со всеми радиопультами – радиопульты для сигнализации (например, автомобильные брелки) используют особые алгоритмы шифрования, делающие невозможным их перехват и расшифровку. – *Прим. перев.*

(00)(00)(00)(01) имеет трехуровневый формат 0100F0F1000F, аналогично код *3163913* имеет формат *0100F0F100UF*, где обозначение *U* относится к значению 10 битовой пары. Анализ сигналов для *3163905* и *3163913*, захваченных логическим анализатором с помощью программы *sigrok PulseView* и настройки декодера RC encode, показан на рис. 15-6.

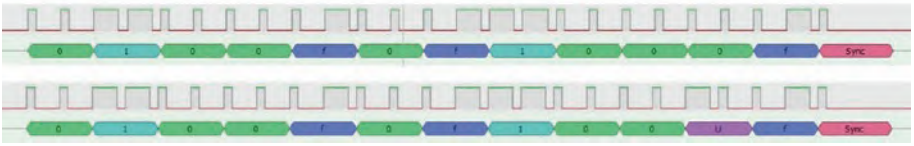


Рисунок 15-6. Анализ сигналов логического анализатора с трехстатусным кодированием

Листинги 15-3 и 15-4 иллюстрируют декодирование и передачу радиочастотного сигнала с помощью библиотеки *rc-switch*. В листинге 15-3 после приема радиочастотного сигнала монитор последовательного порта отображает сигнал в десятичном формате с длиной сигнала в битах и номером протокола. Также включается и выключается светодиод, указывая на то, что сигнал был получен. Переменная `value` для хранения полученного кода кнопки определяется как `unsigned long` с верхним пределом $2^{32} - 1$ для размещения кодов беспроводного дистанционного управления RF. Переменная `unsigned int` имеет верхний предел $2^{16} - 1$, или 65 535. Для библиотеки *rc-switch* для вывода приема данных подключаются прерывания с помощью инструкций `digitalPinToInterrupt()` и `rc.enableReceive()` с номером вывода, равным D2 или 27 для микроконтроллера ESP8266 или ESP32 соответственно[84].

Листинг 15-3. Прием и декодирование RF-кода с помощью библиотеки rc-switch

```
#include <RCSwitch.h>                  // include the rc-switch library
RCSwitch rc = RCSwitch();              // associate rc with rc-switch lib
int LEDpin = D3;                       // define LED pin and state
int LED = 0;
unsigned long value;
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  digitalPinToInterrupt(D2);           // set pin as interrupt
  rc.enableReceive(D2);                // receive data on interrupt pin
  pinMode(LEDpin, OUTPUT);             // define LED pin as output
}
void loop()
{
  if (rc.available())                  // if a signal is received
  {
    value = rc.getReceivedValue();     // signal in decimal format
    if (value != 0)                    // non-zero signal value
    {
      Serial.print("Decimal ");Serial.print(value);
      Serial.print(" (");
```

[84] Автор не указывает приемники сигналов, однако, очевидно, речь идет о тех же RF-приемниках (на частоту 315/433 МГц), показанных на рис. 15-3 и 15-4.

```
        Serial.print(rc.getReceivedBi tlength());     // signal bit length
        Serial.print("bit)\tProtocol ");      // print a tab between text
        Serial.println(rc.getReceived Protocol());
                                            // signal protocol class
        LED = 1 - LED;
        digitalWrite(LEDpin, LED);          // turn on or off the LED
      }
      else Serial.println("Unknown encoding");
      rc.resetAvailable();                  // ready to receive signal
    }
  }
```

С помощью библиотеки *rc-switch* радиочастотные коды в десятичном, двоичном или трехуровневом формате передаются с помощью команд `send(number, bitlength)`, `send(pointer)` или `sendTriState(pointer)` соответственно, где `pointer` – указатель на массив, содержащий строку, включающую число в двоичной или трехуровневой форме. Преобразованный радиочастотный код форматируется как строка с завершающим нулевым символом \0 с помощью инструкции `pointer = code.c_str()`. Независимо от формата радиочастотного кода, библиотека *rc-switch* преобразует радиочастотный код в двоичный формат для передачи. Сигнал передается многократно, со значением по умолчанию десять повторов, соответствующих трем принятым сигналам. Минимальное количество повторных передач для приема сигнала равно четырем с использованием команды `setRepeatTransmit(4)`. Передачи, повторяемые от четырех до десяти раз, приводят к приему сигнала 1, 1, 2, 2, 3, 3 и 3 раза.

В листинге 15-4 передаются коды RC *3163905-3163913*, соответствующие нескольким кнопкам беспроводного радиочастотного пульта дистанционного управления, при этом коды передаются в десятичном, двоичном и трехуровневом форматах. Индикатор включается во время передачи сигнала.

В листинге 15-4 функции `binary` и `tristate` преобразуют десятичное число в двоичный формат и двоичное число в трехуровневый формат соответственно. Десятичное число преобразуется в двоичный формат путем многократного деления целой части числа на два и сохранения остатков, причем первый остаток является младшим битом (LSB) двоичного представления. Например, для преобразования десятичного числа 19 в двоичный формат повторяющееся деление на два приводит к остаткам 1, 1, 0, 0, 1 и 0, что эквивалентно (19 – 2 × 9, 9 – 2 × 4, 4 – 2 × 2, 2 – 2 × 1, 1 – 2 × 0). Первый остаток – это LSB, поэтому двоичное представление 19 равно 010011, причем LSB записывается последним. Библиотека *rc-switch* использует 24-разрядный формат для представления числа в двоичном формате, поэтому для преобразования десятичного числа в двоичный формат требуется 24 деления на два.

Преобразование числа в двоичном формате в трехуровневый формат начинается с пары MSB и заканчивается парой LSB. Для десятичного числа 19 пары битов равны (01), (00) и (11), которые соответствуют трехуровневой записи F01. В листинге 15-4 символ в позиции *x* строки `bin` идентифицируется инструкцией `bin.charAt(x)`. В листинге 15-4 трехуровневый формат корректен для чисел 3163905, 3163907, 3163908, 3163909 и 3163911, но переданные значения некорректны для чисел 3163906, 3163910, 316312 и 3163913, поэтому трехуровневый формат дополнен обозначением *U* для значения 10 битовой пары.

Листинг 15-4. Передача радиочастотного кода с помощью библиотеки rc-switch

```
#include <RCSwitch.h>                  // include the rc-switch library
RCSwitch rc = RCSwitch();              // associate rc with rc-switch lib
unsigned long value = 3163905;         // code to be transmitted
const char * biCode;                   // pointers to arrays with number
const char * triCode;                  // in binary or Tri-state format
int LEDpin = D3;                       // define LED pin
int LED = 0;                           // initial LED state
int delTime = 1000;                    // delay between transmissions
String bin, tri;
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  pinMode(LEDpin, OUTPUT);             // define LED pin as output
  rc.enableTransmit(D1);               // transmit data pin
  rc.setPulseLength(351);              // optional with time in ms
  rc.setProtocol(1);                   // default is 1
  rc.setRepeatTransmit(4);             // define number of transmissions
}
void loop()
{
  rc.send(value, 24);                  // send number in decimal format
  Serial.print(value);Serial.prin t("\t");   // display value
  digitalWrite(LEDpin, LED);           // turn on or off LED
  LED = 1 - LED;
  delay(delTime);                      // interval between transmissions
  binary(value);                       // convert to binary format
  rc.send(biCode);                     // send array in binary format
  Serial.print(biCode);Serial.pri nt("\t");
                                       // display value in binary format
  delay(delTime);
  tristate(bin);                       // convert to Tri-state format
  rc.sendTriState(triCode);            // send array in Tri-state format
  Serial.println(triCode);             // display in Tri-state format
  delay(delTime);
  value++;
  if(value > 3163913) for (;;) de lay(1000);
}                                      // stop after sending all codes
void binary(long number)               // function to convert to binary format
{
  bin = "";
  for (int i=0; i<24; i++)             // 24 bits starting with LSB
  {                                    // next bit precedes lower significant bits
    if(number%2 == 1) bin = "1" + bin;
                                       // number is an unsigned long integer
    else bin = "0" + bin;
    number = number/2;
  }
  biCode = bin.c_str();                // create pointer to array
}
void tristate(String val)              // function to convert to Tri-state format
{
  tri = "";
  for (int i=0; i<12; i++)             // start with MSB which is charAt(0)
  {                                    // next bit follows higher significant bits
    if(val.charAt(2*i)=='0' && val.charAt(2*i+1)=='0')
    tri = tri + "0";
```

```
    else if(val.charAt(2*i)=='1' && val.charAt(2*i+1)=='1')
    tri = tri + "1";
    else if(val.charAt(2*i)=='0' && val.charAt(2*i+1)=='1')
    tri = tri + "F";
    else if(val.charAt(2*i)=='1' && val.charAt(2*i+1)=='0')
    tri = tri + "U";
    }
    triCode = tri.c_str();             // create pointer to array
}
```

Управление сервоприводами поворота и наклона с помощью RF-связи

Лазер, установленный на поворотно-наклонном кронштейне, вращается двумя серводвигателями и управляется путем передачи сообщения, содержащего положение джойстика и состояние лазера. Джойстик, такой как модуль KY-023, состоит из двух потенциометров для управления направлением влево-вправо (ось X) и вперед-назад (ось Y). Значения джойстика меняются от 0 до 1023, причем 0 соответствует значениям вправо и вперед, в то время как 1023 соответствует значениям влево и назад. Нажатие на джойстик активирует переключатель включения или выключения лазера. RF-передатчик и джойстик подключены к Arduino Nano или к плате ESP32 (см. рис. 15-7 и табл. 15-3); а RF-приемник, два серводвигателя с внешним питанием и лазерный модуль подключены к плате ESP8266 (см. рис. 15-8 и табл. 15-4). Лазер KY-008 работает на длине волны 650 нм в диапазоне длин волн красного света 635–700 нм.

Для считывания значений джойстика требуются два аналоговых вывода. Микроконтроллер ESP8266 имеет только один аналоговый вывод, поэтому либо к плате ESP8266 подключается мультиплексор 74HC4051, чтобы микроконтроллер ESP8266 мог получать доступ к обоим устройствам аналогового ввода (см. главу 4 «Интернет-часы»), либо плата ESP8266 заменяется Arduino Nano или платой ESP32.

Выводами передачи и приема по умолчанию для Arduino Nano с библиотекой *RH_ASK* являются выводы 12 и 11 соответственно. Вывод GPIO 27 микроконтроллера ESP32 устанавливается как вывод передачи инструкцией `RH_ASK rf (2000, 26, 27, 0)`. Назначение GPIO 26 для вывода приема является произвольным, так как в скетче микроконтроллер ESP32 не принимает сообщения. АЦП микроконтроллера ESP32 имеет 12-битное разрешение со значениями от 0 до 4095, в то время как АЦП Arduino Nano имеет 10-битное разрешение. Поэтому либо значения АЦП микроконтроллера ESP32 нужно делить на четыре перед передачей в листинге 15-5, либо для преобразования принятого значения джойстика в угол сервопривода в листинге 15-6 изменить в обоих случаях (для FB и LR) инструкции на `map(text[0].toInt(),0,4095,5,100)`.

Рисунок 15-7. Передача сигналов джойстика

Таблица 15-3. Джойстик, RF-передатчик и Arduino Nano или ESP32

Компонент	Arduino Nano	ESP32
Джойстик VCC (+)	5V	VIN
Джойстик SEL (B)	A0	GPIO 25
Джойстик HOR (X)	A1	GPIO 32
Джойстик VER (Y)	A2	GPIO 33
Джойстик GND (–)	GND	GND
RF-передатчик VCC (+)	5V	VIN
RF-передатчик GND (–)	GND	GND
RF-передатчик DAT	D12	GPIO 27

Скетч в листинге 15-5 считывает положение джойстика и состояние контактов кнопки, а затем объединяет информацию в строку для передачи. При нажатии кнопка джойстика выдает высокий уровень (HIGH). Библиотека *RH_ASK* используется для передачи и приема радиосигналов. Установки выводов джойстика и RF-передатчика в листинге 15-5 предназначены для Arduino Nano, для микроконтроллера ESP32 определения выводов заменяются на следующие:

```
RH_ASK rf (2000, 26, 27, 0)   // associate rf with RH_ASK lib
int switchPin = 25            // joystick switch pin
int LRpin = 32                // left-right joystick pin
int FBpin = 33                // forward-backward
joystick pin
pinMode(switchPin, INPUT)     // define switchPin as input
```

Листинг 15-5. Передача сигнала джойстика для управления серводвигателями и лазером

```
#include <RH_ASK.h>           // include the RH_ASK library
#include <SPI.h>              // SPI library required to compile
RH_ASK rf;                    // associate rf with RH_ASK lib
int timelag = 50;             // interval between transmissions
int switchPin = A0;           // joystick switch pin
int LRpin = A1;               // left-right joystick pin
```

```
int FBpin = A2;                                  // forward-backward
joystick pin
int FB, LR, SW;
const char * msg;
String message;
void setup()
{
   rf.init();                                    // initialise radio transmission
}
void loop()
{
   // string for joystick forward-backward, left-right and switch state
   FB = analogRead(FBpin);
   LR = analogRead(LRpin);                       // get joystick position
   SW = digitalRead(switchPin);                  // and switch state
   message = String(FB) +","+ String(LR) + "," + String(SW) + ",";
   msg = message.c_str();                        // convert string
   rf.send((uint8_t *)msg, strlen(msg));         // transmit message
   rf.waitPacketSent();                          // wait for transmission to finish
   delay(timelag);                               // delay between transmissions
}
```

Лазер и два сервопривода для поворотно-наклонного кронштейна подключены к плате ESP8266 с RF-приемником (см. рис. 15-8 и табл. 15-4). Сервоприводы питаются от внешнего источника 5 В, так как двигатель может потреблять сотни миллиампер в течение долей секунды, пока вращается ротор, что превышает выходную мощность 5V-вывода платы ESP8266.

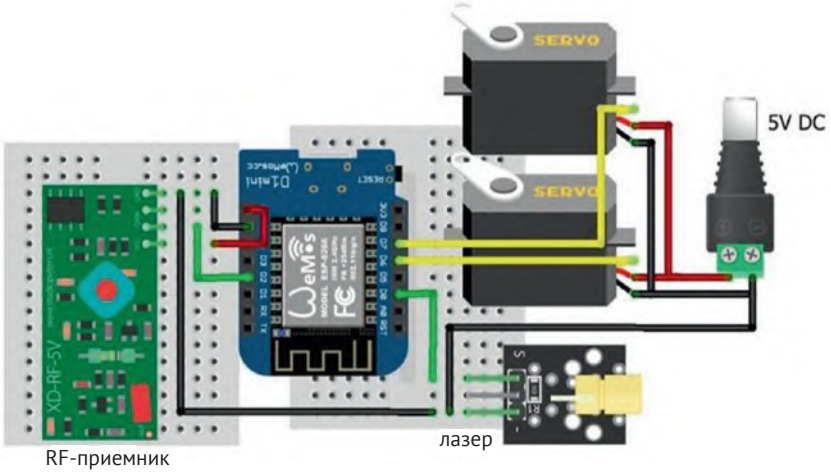


Рисунок 15-8. Управление сервоприводами и лазером с помощью полученного сигнала

Таблица 15-4. Сервоприводы, лазер, RF-приемник и плата ESP8266

Компонент	Подключено к	Также к
Сервоприводы VCC (красный)	Внешний разъем +5 В	
Сервоприводы GND (коричневый или черный)	ESP8266 GND	Внешний разъем GND
Сигнальные выводы сервоприводов (оранжевый или белый[85])	ESP8266 D6 (FB вперед-назад) и D7(LF влево-вправо)	
Лазер сигнал (S)	ESP8266 D0	
Лазер GND (–)	ESP8266 GND	
RF-приемник VCC (+)		
RF-приемник GND (–)		
RF-приемник DAT	ESP8266 D2	

Листинг 15-6 содержит скетч RF-приемника, сервоприводов и лазерного модуля. Как отмечалось в начале этой главы, драйвер *RH_ASK* использует по умолчанию таймер Timer1, который также применяется библиотекой *Servo*. Драйвер *RH_ASK* будет использовать Timer2, если раскомментировать строку `#define RH_ASK_ARDUINO_USE_TIMER2` в строке 32 файла *RH_ASK.cpp*. Скетч связывает объекты `servoFB` и `servoLR` с библиотекой *Servo*, поскольку в ней есть поддержка двух серводвигателей. Инструкция `const int nItem = 3` позволяет неявно определить размер двух массивов, `text[nItem]` и `comma[nItem+1]`. Когда RF-приемник принимает сообщение, подстроки сообщения преобразуются в целые числа, которые преобразуются в углы сервоприводов вперед-назад и влево-вправо между 5° и 100° и между 5° и 175° соответственно. Высокий (`HIGH`) уровень на лазерном выводе указывает на то, что была нажата кнопка джойстика.

Сервоприводы могут «дрожать», когда угол наклона не меняется. Чтобы остановить колебания, сервопривод можно отключить, а затем снова подключить перед переходом на новый угол. Например, команда перемещения на угол `LR` для левого-правого сервопривода `servoLR.write(LR)` в этом случае заменяется инструкциями

```
servoLR.attach(LRpin)
servoLR.write(LR)
servoLR.detach()
```

Листинг 15-6. Прием сигнала положений джойстика для управления сервоприводами и лазером

```
#include <RH_ASK.h>           // include the RH_ASK library
#include <SPI.h>              // SPI library required to compile
#include <Servo.h>            // include Servo library
RH_ASK rf (2000, D2, D1, 0); // associate rf with RH_ASK lib
Servo servoFB;                // associate servoFB and servoLR
Servo servoLR;                // with Servo library
int FBpin = D6;               // forward-backward
servo pin
```

[85] См. сноску 48 на стр. 158. – *Прим. перев.*

```
int LRpin = D7;               // left-right servo pin
int laserPin = D0;            // define laser pin
uint8_t msg[RH_ASK_MAX_MESSAGE_LEN];
const int nItem = 3;          // number of items in message
String text[nItem];           // array of strings
int comma[nItem+1];           // array of comma positions
String string;
int laser, FB, LR;
void setup()
{
  rf.init();                  // initialise radio transmission
  servoFB.attach(FBpin);      // initialise servo motors
  servoLR.attach(LRpin);
  pinMode(laserPin, OUTPUT); // define laser pin as output
}
void loop()
{                             // define message length based on new message
  uint8_t msglen = siz eof(msg);
  if (rf.recv(msg, &msglen)) // message of correct length available
  {
    string = "";              // increment string by each message character
    for (int i=0; i<msglen; i++) string = string + char(msg[i]);
    comma[0] = -1;
    for (int i=0; i<nItem; i++)      // get message comma positions
    {                                // get substrings between commas
      comma[i+1] = string.indexOf(",", comma[i]+1);
      text[i] = string.substring(comma[i]+1, comma[i+1]);
    }                                // map joystick signals to angles
    FB = map(text[0].toInt(),0,10 23,5,100);
    LR = map(text[1].toInt(),0,1023,5,175);
    laser = text[2].toInt();         // update laser status
    if(laser == HIGH) digitalWrite(laserPin,
    !digitalRead(laserPin));
    servoFB.write(FB);               // move servos to angles
    servoLR.write(LR);
  }
}
```

Если сервоприводы подключены к плате ESP32, то требуется специальная библиотека для ESP32 взамен встроенной библиотеки *Servo* Arduino IDE. Рекомендуется библиотека *ESP32Servo* Кевина Харрингтона (Kevin Harrington) и Джона К. Беннетта (John K. Bennett), также доступная в Arduino IDE. Инструкции встроенной библиотеки *Servo* для сервопривода вперед-назад (FB) с микроконтроллером ESP8266

```
#include <Servo.h>            // include Servo library
servoFB.attach(FBpin)         // initialise servo motor to FBpin
```

заменяются инструкциями библиотеки ESP32Servo для микроконтроллера ESP32

```
#include <ESP32Servo.h>
servoFB.setPeriodHertz(F)     // define servo frequency (F)
servoFB.attach(FBpin, minPW, maxPW)
                              // initialise servo motor to FBpin
```

В следующих инструкциях изменений нет:

```
Servo servoFB    // associate servoFB with Servo lib
servoFB.writeMicroseconds(T) // move to position mapped to Tμs
servoFB.write(N)    // move to angle N°
```

Аналогичные изменения следует внести для сервопривода влево-вправо (LR).

Частота прямоугольного колебания *F* задана в команде `servoFB.setPeriodHertz(F)`, которая обычно составляет 50 Гц. В инструкции `servoFB.attach(FBpin, minPW, maxPW)` параметры `minPW` и `maxPW` относятся к длительности импульса прямоугольного колебания в микросекундах для перемещения сервопривода на 0° и 180° соответственно. Значения по умолчанию для этих параметров составляют 1000 мкс и 2000 мкс, а для сервопривода Tower Pro SG90 – 500 мкс и 2500 мкс. Для сервопривода, используемого в этой главе, требовалось 250 мкс и 2050 мкс для перемещения на 0° и 180° соответственно. Скетч для калибровки сервопривода приведен в главе 9 («WebSocket»). Предыдущие инструкции для сервопривода вперед-назад `servoFB` повторяются для сервопривода влево-вправо `servoLR`.

Управление реле по RF-связи

Нажатие определенной кнопки на беспроводном RF-пульте дистанционного управления используется для включения или выключения MOSFET-транзистора *IRF520* или релейного модуля *KY-019*, который коммутирует источник питания на большую нагрузку, такую как двигатель. МОП-транзистор IRF520 коммутирует до 100 В постоянного тока (DC) при 10 А, в то время как релейный модуль KY-019 допускает либо 240 В переменного тока (AC), либо 30 В DC при 10 А. Библиотека *rc-switch* используется для передачи и приема кодов кнопок дистанционного управления. Соединения для RF-приемника, модуля MOSFET IRF520 или релейного модуля KY-019 и нагрузки, такой как двигатель, показаны на рис. 15-9 и перечислены в табл. 15-5 и 15-6.

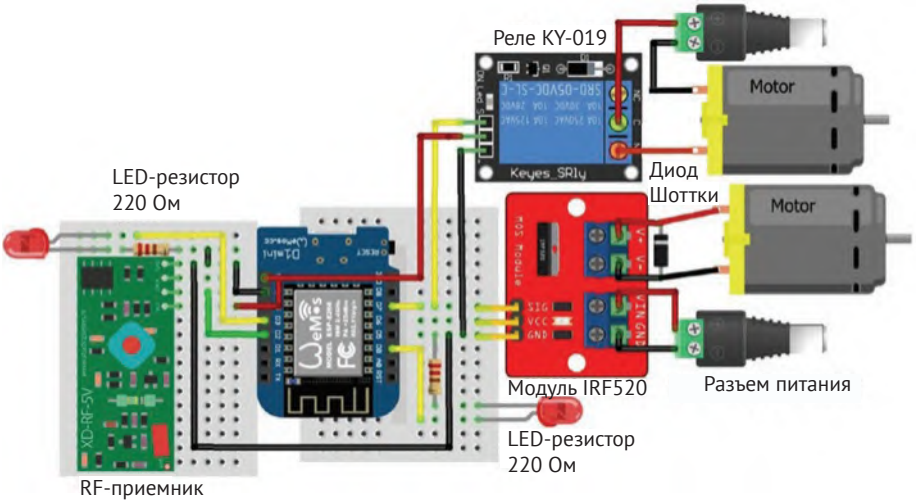


Рисунок 15-9. Принятый сигнал управляет модулем IRF520 MOSFET или релейным модулем KY-019

Таблица 15-5. Модули MOSFET IRF520 и KY-019, RF-приемник и плата ESP8266

Компонент	Подключено к	Также к
IRF520 или KY-019 SIG	ESP8266 D7	
IRF520 VCC	Not connected	
KY-019 VCC	ESP8266 5V	
IRF520 или KY-019 GND	ESP8266 GND	
LED длинный вывод	ESP8266 D3 and D0	
LED короткий вывод	резистор 220 Ом	ESP8266 GND
RF-приемник VCC (+)	ESP8266 5V	
RF-приемник GND (–)	ESP8266 GND	
RF-приемник DAT	ESP8266 D2	

Модули реле IRF520 MOSFET и KY-019 представляют собой два варианта управления внешним источником питания для нагрузки. Модуль MOSFET IRF520 имеет две пары соединений: одну пару для нагрузки и одну пару для внешнего источника питания (круглый разъем). Для релейного модуля KY-019 положительный вывод внешнего источника питания подключен к выводу реле модуля KY-019 «общий» (*common*, C), отрицательный вывод внешнего источника – к отрицательному выводу нагрузки. Нормально разомкнутый (*normally open*, NO) вывод реле модуля KY-019 подключен к положительному выводу нагрузки. Рекомендуется использовать нормально разомкнутый KY-019 (NO), а не нормально замкнутый (*normally closed*, NC) контакт.

Таблица 15-6. Подключение модулей IRF520 MOSFET и релейного модуля KY-019 к нагрузке и питанию

Компонент	Подключен к	Также к
IRF520 V+	Нагрузка (двигатель) плюс	Диод Шоттки катод (полоса)
IRF520 V–	Нагрузка (двигатель) минус	Диод Шоттки анод
IRF520 VIN	Внешнее питание +	
IRF520 GND	Внешнее питание GND	
KY-019 нормально разомкнутый (NO)		Нагрузка (двигатель) плюс
KY-019 общий (C)	Внешнее питание +	
	Внешнее питание GND	Нагрузка (двигатель) минус

В листинге 15-7 нажатие кнопки *LIGHT* на беспроводном RF-пульте дистанционного управления включает или выключает светодиод, подключенный к контакту D3 платы ESP8266. RF-коды кнопок в листинге 15-7 относятся только к беспроводному RF-пульту дистанционного управления, используемому в этой главе. При нажатии кнопок *BRIGHT*+ или *BRIGHT*– на беспроводном RF-пульте модуль MOSFET IRF520 или KY-019, подающие питание на нагрузку, включается или выключается, как и светодиод, подключенный к контакту D0

платы ESP8266. Релейный модуль KY-019 включается, когда вывод `relayPin` D7 переключается в высокий уровень (*HIGH*).

Листинг 15-7. Прием сигнала для управления нагрузкой

```
#include <RCSwitch.h>                 // include the rc-switch library
RCSwitch rc = RCSwitch();             // associate rc with rc-switch lib
int LEDpin = D3;                      // LED to change state
int LEDrelayPin = D0;                 // LED associated with relay
int relayPin = D7;                    // define MOSFET/relay pin
unsigned long value;
void setup()
{
  Serial.begin(9600);                 // Serial output baud rate
  digitalPinToInterrupt(D2);          // set pin as interrupt
  rc.enableReceive(D2);               // receive data on interrupt pin
  // digitalWrite(relayPin, HIGH);    // set relayPin HIGH before
  pinMode(relayPin, OUTPUT);          // defining relayPin
  pinMode(LEDpin, OUTPUT);
  pinMode(LEDrelayPin, OUTPUT);
}
void loop()
{
  if (rc.available())                 // if a signal is received
  {
       value = rc.getReceivedValue();
       if (value != 0)                // signal value not equal to zero
       {                              // display signal value
       Serial.print("code ");Serial.print(value);
       if(value == 3163908)           // Light button pressed
       {
          Serial.print("\tchange LED");       // display action
          digitalWrite(LEDpin, !digita lRead(LEDpin));
       }                                      // turn on or off LED
       else if(value == 3163909)              // Bright+ button pressed
       {
          Serial.print("\trelay on");
          digitalWrite(LEDrelayPin, HI GH);   // turn on relay LED
          digitalWrite(relayPin, HIGH);       // turn on relay
       }
       else if (value == 3163910)             // Bright- button pressed
       {
          Serial.print("\trelay off");
          digitalWrite(LEDrelayPin, LOW);     // turn off relay LED
          digitalWrite(relayPin, LOW);        // turn off relay
       }
       else Serial.print("\tno action");
       Serial.print("\tLED ");
       Serial.print(digitalRead(LEDpin ));
                                              // display LED and relay states
       Serial.print("\trelay ");
       Serial.println(digitalRead(relayPin));
       }
       else Serial.println("Unknown encoding");
       rc.resetAvailable();                   // ready to receive new signal
  }
}
```

Реле

Реле – это электромагнитный переключатель, управляемый небольшим током, который используется для включения или выключения большого тока. Когда управляющий вывод микроконтроллера ESP8266 или ESP32 имеет высокий уровень (*HIGH*), небольшой ток через базу транзистора позволяет большему току от вывода 5V платы ESP8266 или ESP32 проходить через транзистор, активируя магнитное поле в обмотке реле, которое притягивает и замыкает металлические контакты (см. рис. 15-10). Когда вывод микроконтроллера ESP8266 или ESP32 имеет низкий уровень (*LOW*), ток через транзистор не протекает и пружина возвращает контакты реле в нормально разомкнутое положение. В релейном модуле имеется резистор для ограничения тока через базу ключевого транзистора и диод для короткого замыкания катушки реле при отключении тока. Энергия в катушке поглощается через внутреннее сопротивление катушки реле и падение напряжения на диоде. Без диода большой скачок напряжения может разрушить транзистор. Рисунки 15-10–15-16 иллюстрируют микроконтроллер ESP8266, управляющий реле, но обсуждение также относится к микроконтроллеру ESP32.

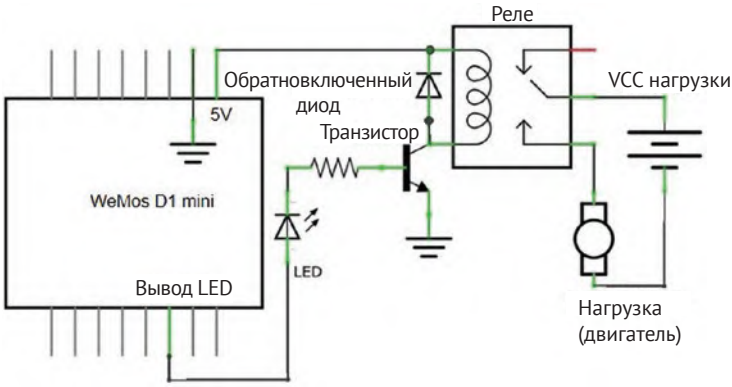


Рисунок 15-10. Схема релейного модуля

Для управления реле, таким как модуль KY-019 на рис. 15-11, высокий уровень сигнала на выводе D4 платы ESP8266 включает ключевой транзистор, активируя релейный переключатель для подачи внешнего питания на нагрузку, такую как двигатель. Контакт платы ESP8266 GND подключен к модулю KY-019, поскольку светодиодный индикатор модуля, ключевой транзистор и обмотка электромагнитного реле питаются от платы ESP8266.

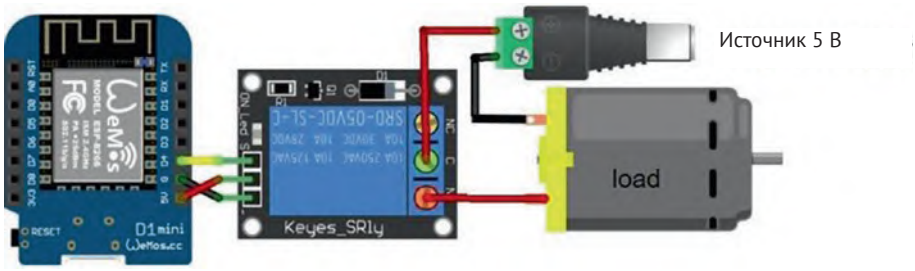


Рисунок 15-11. Релейный модуль и плата LOLIN (WeMos) D1 mini

Релейный модуль с оптроном расширяет возможности изоляции микроконтроллера ESP8266 от нагрузки с помощью внешнего источника питания (см. рис. 15-12 и 15-13). Оптрон FOD817C состоит из инфракрасного светодиода и фототранзистора, который включается при освещении светодиода. Как оптрон, так и ключевой транзистор питаются от платы ESP8266. Перемычка между контактами модуля реле с маркировкой RY-VCC или JD-VCC и VCC подключают ключевой транзистор к выводу 5V платы ESP8266. Вывод микроконтроллера ESP8266 принимает ток от модуля реле с оптроном, в отличие от реле KY-019, на которое ток подается, и реле с оптроном включается, когда вывод микроконтроллера ESP8266 находится на низком уровне[86]. Пунктирная соединительная линия между VCC оптрона и реле VCC на рис. 15-12 представляет перемычку.

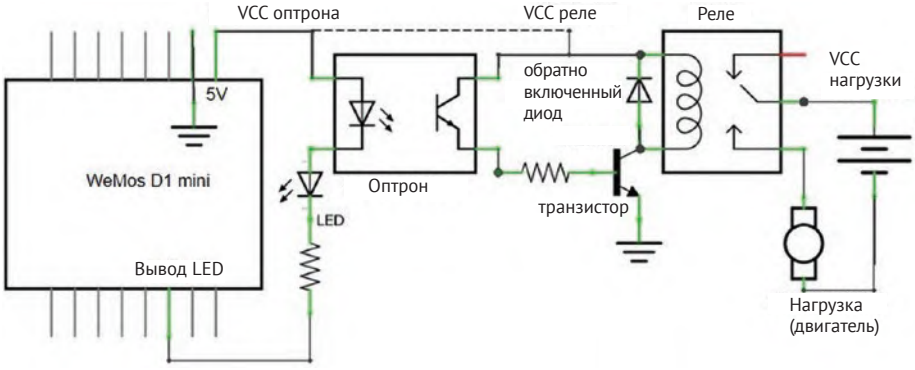


Рисунок 15-12

[86] Логика, когда срабатывание элемента происходит при подаче низкого уровня (иначе говоря, когда низкий уровень является активным), называется отрицательной логикой. Отметим, что условие срабатывания реле в принципе несложно перевести в привычную положительную логику (когда реле будет срабатывать, как обычно, при высоком уровне на выводе), если немного модифицировать схемы, подавая управляющий сигнал на плюс светодиода оптрона, а не на цепь, подключенную к его отрицательному выводу (вывод IN2 модуля в данном случае, согласно рис. 15-13 и 15-14), которую тогда необходимо будет подключить к GND контроллера. На практике с данным модулем такое не получится – за счет прямого падения напряжения на двух последовательно включенных светодиодах на маломощном управляющем выводе контроллера просто не хватит напряжения для их открытия, даже если применить 5-вольтовый Arduino, а не ESP8266 с 3,3-вольтовым выходом. – *Прим. перев.*

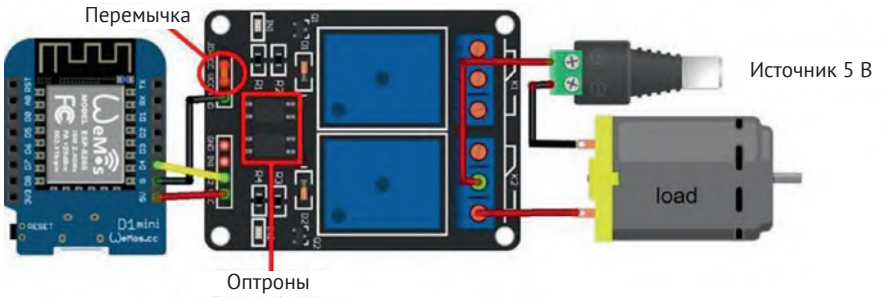


Рисунок 15-13. Релейный модуль с оптроном (1)

Микроконтроллер ESP8266 будет полностью изолирован от нагрузки и внешнего источника питания, если обеспечить реле отдельным источником питания (см. рис. 15-14 с подключениями, приведенными в табл. 15-7). Перемычка между контактами модуля реле с маркировкой RY-VCC или JD-VCC и VCC снимается, и внешнее питание подается отдельно на контакты модуля реле RY-VCC или JD-VCC и GND. Вывод GND микроконтроллера ESP8266 не связан с GND релейного модуля, так как оптрон и светодиодный индикатор пропускают ток через вывод микроконтроллера ESP8266, установленный в низкий уровень (LOW).

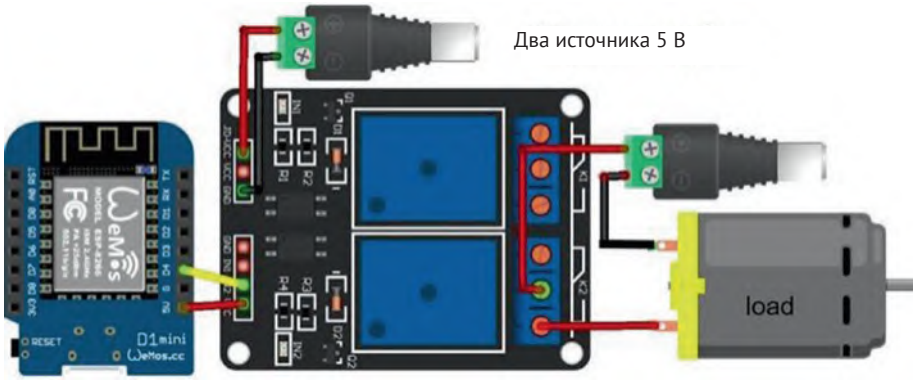


Рисунок 15-14. Релейный модуль с оптроном (2)

Таблица 15-7. Подключения оптронного релейного модуля без внешнего источника питания и с внешним источником питания для реле

Релейный модуль	Компонент	Подключено к
Без внешнего питания	Релейный модуль JD-VCC и VCC	Перемычка
	Релейный модуль GND	ESP8266 GND
	Релейный модуль SIG или IN2 [87]	ESP8266 D7
	Релейный модуль VCC	ESP8266 5V

[87] Автором используется двухканальный оптронный релейный модуль (фирмы SONGLE) с двумя управляющими входами IN1 и IN2 для каждого из каналов. В схемах рис. 15-13 и 15-14 задействован второй канал. – *Прим. перев.*

Окончание табл. 15-7

Релейный модуль	Компонент	Подключено к
С внешним питанием	Релейный модуль JD-VCC	Внешнее питание плюс
	Релейный модуль GND	Внешнее питание минус
	Релейный модуль SIG или IN2	ESP8266 D7
	Релейный модуль VCC	ESP8266 5V

В листинге 15-7 реле KY-019 активируется переводом вывода микроконтроллера ESP8266 в высокий уровень с помощью инструкций:

```
digitalWrite(relayPin, HIGH);        // turn on relay
digitalWrite(relayPin, LOW);         // turn off relay
Serial.print("\trelay ");
Serial.println(digitalRead(relayPin));
```

Для реле с оптроном состояние выводов микроконтроллера ESP8266 инвертировано, а изменения выделены жирным шрифтом:

```
digitalWrite(relayPin, LOW);         // turn on relay
digitalWrite(relayPin, HIGH);        // turn off relay
Serial.print("\trelay ");
Serial.println(!digitalRead(relayPin));
```

Команда `pinMode(relayPin, OUTPUT)` автоматически устанавливает вывод управления реле на низкий уровень, что вызовет срабатывание реле оптронного модуля. Чтобы предотвратить случайное включение реле с оптроном, команда `digitalWrite(relayPin,HIGH)` предшествует команде `pinMode(relaypin, OUTPUT)`.

Твердотельное реле

Твердотельное реле включает полупроводниковые компоненты без механических движущихся частей для высокоскоростной коммутации переменного тока (AC) до 240 В при 2 А[88]. Основой полупроводниковых реле служит фотосимистор, который состоит из светодиода инфракрасного диапазона и фоточувствительного симистора (триака), позволяющего току поступать в нагрузку (см. рис. 15-15). Твердотельное реле включает в себя дополнительные схемы как на входной стороне инфракрасного светодиода, так и на выходной стороне фото-

[88] Справедливо для модуля с реле G3MB-202P фирмы OMRON, применяемого автором. Существуют твердотельные реле, способные пропускать переменный ток до 600 В при сотнях ампер нагрузки. Следует учесть, что твердотельные реле греются при прохождении тока (падение напряжения в открытом состоянии до 1,5–2,5 В), потому реле для больших токов требуют установки на радиатор с принудительным охлаждением. Для данного типа реле выделение тепла не превышает 3–4 Вт при максимальной средней мощности, что безопасно для корпуса таких размеров, однако при эксплуатации в предельных режимах (0,5 кВт в нагрузке) не следует устанавливать модуль в непроветриваемый тесный корпус. – *Прим. перев.*

симистора. Более подробная информация доступна по адресу www.ia.omron.com/support/guide/18/introduction.html.

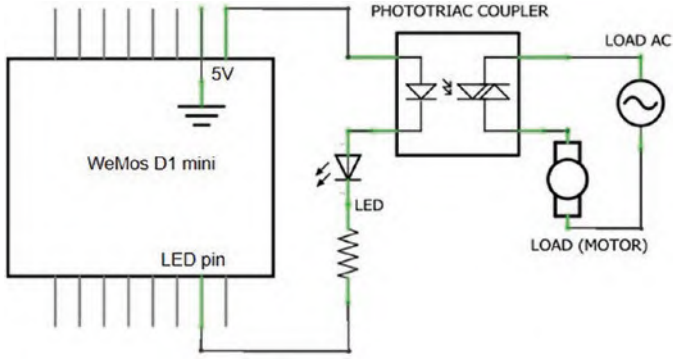


Рисунок 15-15. Упрощенная схема твердотельного реле

Твердотельное реле включено, когда вывод микроконтроллера ESP8266, подключенный к фотодиодному входу реле, имеет низкий уровень, аналогично механическому реле с оптроном. Соединения платы ESP8266 с твердотельным релейным модулем показаны на рис. 15-16 и приведены в табл. 15-8.

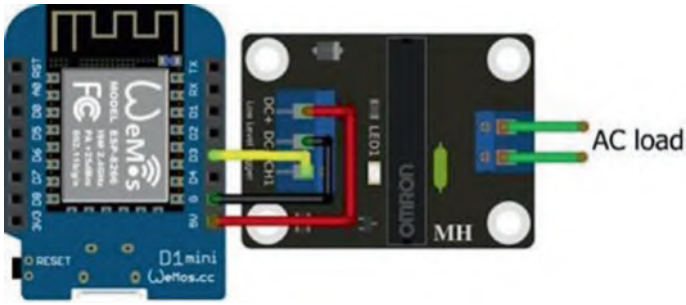


Рисунок 15-16. Твердотельное реле и плата LOLIN (WeMos) D1 mini

Таблица 15-8. Твердотельное реле и плата ESP8266

Компонент	Подключение к ESP8266
Твердотельное реле DC+	5V
Твердотельное реле DC–	GND
Твердотельное реле CH1	D3

Итоги

Описана радиочастотная связь с амплитудной манипуляцией (ASK) в варианте OOK («включение-выключение», On-Off Keying). Супергетеродинная пара передатчика и приемника с частотой 433 МГц передавала и принимала символьно-цифровые данные по радиосвязи ASK на расстоянии 150 м. Сигналы

на частоте 433 МГц с беспроводного цифрового пульта дистанционного управления декодировались с помощью супергетеродинного приемника. Обсуждалось форматирование данных в двоичном и троичном состояниях. Состояние и положение лазера, установленного на поворотно-наклонный кронштейн с сервоприводами, подключенными к микроконтроллеру ESP8266 или ESP32, дистанционно управлялись джойстиком. При этом информация о положении джойстика и состоянии лазера передавалась по ASK-радиосвязи супергетеродинным передатчиком, подключенным к микроконтроллеру ESP32. Дистанционное включение через электромеханическое реле нагрузки, такой как двигатели, управлялось передачей управляющих кодов с помощью беспроводного RF-пульта дистанционного управления. Были описаны электромеханические реле с оптроном и без него для подачи постоянного тока на нагрузку, а также твердотельное реле.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN D1 mini (WeMos) или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU
- Мультиплексор 74HC4051 или Arduino Nano
- 433 MHz передатчик: стандартный MX-FS-03V; супергетеродинный WL102-341
- 433 MHz приемник: стандартный MX-05V; супергетеродинный WL101-341
- Лазерный модуль KY-008
- Светодиод 2 шт.
- Резисторы 220 Ом 2 шт.
- Джойстик KY-023
- Сервопривод SG90 2 шт.
- Поворотно-наклонный сервокронштейн
- Беспроводной RF-пульт
- Модуль электромагнитного реле: KY-019
- Релейный модуль с оптронной парой
- Модуль MOSFET IRF520
- Твердотельное реле

Глава 16

Генерация сигналов

Телекоммуникации используют цифровые сигналы для передачи информации. Например, команда `Serial.print("12AB")` передает ASCII-код для каждого символа (см. рис. 16-1) в двоичном формате с помощью микроконтроллерного последовательного порта UART (Universal Asynchronous Receiver-Transmitter, *универсальный асинхронный приемник-передатчик*). ASCII (American Standard Code for Information Interchange, *американский стандартный код для обмена информацией*) – это 8-битный стандарт кодирования символов для электронного обмена информацией. При такой коммуникации первым передается младший значащий бит (LSB). Например, буква A имеет ASCII-код 65, который равен B01000001, или ($1 \times 2^6 + 1 \times 2^0$). Переданные биты <S> и <T> на рисунке – это стартовый и стоповый биты (рис. 16-1).

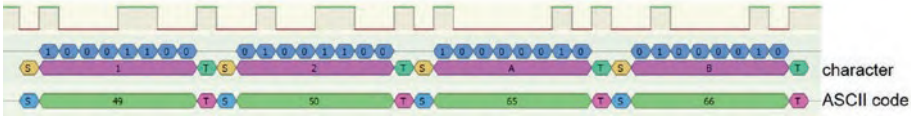


Рисунок 16-1. Передача данных через UART

Обратите внимание, что порядок передачи важен. Например, если код ASCII для символа «2», равный числу 50 (0x32), или в двоичном виде B00110010 ($1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1$), передается старшим битом (MSB) вперед, сигнал может быть интерпретирован как ($1 \times 2^6 + 1 \times 2^3 + 1 \times 2^2$) = 76, что является ASCII-кодом для буквы «L».

Сигналы, состоящие из прямоугольных импульсов с заданной частотой, также используются для управления скоростью сервоприводов и других двигателей и яркостью светодиодов будильника для усиления звука. Широтно-импульсная модуляция (ШИМ, PWM) изменяет ширину импульса (т. е. время, в течение которого прямоугольное колебание имеет высокий уровень), и большая длительность импульса соответствует более высокой частоте вращения двигателя, даже несмотря на многократное включение и выключение питания двигателя при подаче колебания. Несколько выводов микроконтроллеров ESP8266 и ESP32 поддерживают сигналы ШИМ, а рис. 16-2 иллюстрирует два прямоугольных колебания, различающихся по частоте и величине коэффициента заполнения (duty cycle). Для контекста: человеческое ухо может слышать звуки с частотой от 20 Гц до 20 кГц, FM-радиостанции вещают на частоте 100 МГц, а беспроводные сети работают на частоте 2,4 ГГц.

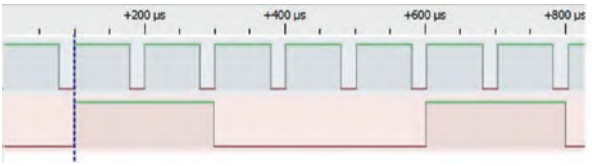


Рисунок 16-2. ШИМ-сигналы с различными частотами и коэффициентами заполнения

Прямоугольные колебания, показанные на рис. 16-2, одновременно генерируются с помощью микроконтроллера ESP8266 или ESP32 с использованием скетчей из табл. 16-1. Частота ШИМ микроконтроллера ESP8266 по умолчанию составляет 1 кГц с разрешением 10 бит. Частота *N* Гц устанавливается с помощью команды `analogWriteFreq(N)`, а коэффициент заполнения (*duty*) определяется как доля от 1023 (или $2^{10} - 1$) инструкцией `analogWrite(wavePin, duty)`.

Микроконтроллер ESP32 PWM имеет разрешение (*resolution*) 8, 10, 12 или 15 бит с максимальной частотой 80 МГц/$2^{resolution}$. Для каждого ШИМ-сигнала определен отдельный канал с инструкциями ШИМ для микроконтроллера ESP32

```
ledcAttachPin(wavePin, channel)          // attach channel to pin
ledcSetup(channel, freq, resolution)     // define frequency
ledcWrite(channel, duty)                 // generate square wave
```

с параметрами выходной канал PWM (*channel*), вывод прямоугольного колебания GPIO (*wavePin*), частота колебания (*freq*), коэффициент заполнения (*duty*) и разрешение PWM (*resolution*). Разрешение АЦП микроконтроллера ESP32 в табл. 16-1 было установлено равным 10, что соответствует разрешению АЦП ESP8266.

Если светодиоды с резисторами 220 Ом присоединены к какому-то из контактов *wavePin*, то прямоугольное колебание с бóльшим коэффициентом заполнения приведет к большей яркости светодиода.

Таблица 16-1. ШИМ у микроконтроллеров ESP8266 и ESP32

Микроконтроллер ESP8266	Микроконтроллер ESP32
`int wave1Pin = D1, wave2Pin = D2;`	`int wave1Pin = 25, wave2Pin = 26;`
`int freq1 = 10000, freq2 = 2000;`	`int freq1 = 10000, freq2 = 2000;`
`float duty1, duty2;`	`float duty1, duty2;`
	`int channel1 = 1, channel2 = 2;`
	`int resolution = 10;`
`void setup()`	`void setup()`
`{`	`{`
`pinMode(wave1Pin, OUTPUT);`	`pinMode(wave1Pin, OUTPUT);`
`pinMode(wave2Pin, OUTPUT);`	`pinMode(wave2Pin, OUTPUT);`
	`ledcAttachPin(wave1Pin, channel1);`
	`ledcAttachPin(wave2Pin, channel2);`
	`ledcSetup(channel1, freq1, resolution);`
	`ledcSetup(channel2, freq2, resolution);`
`duty1 = 0.8*1023;`	`duty1 = 0.8*1023;`
`duty2 = 0.4*1023;`	`duty2 = 0.4*1023;`
`}`	`}`
`void loop()`	`void loop()`
`{`	`{`
`analogWriteFreq(freq1);`	

Окончание табл. 16-1

Микроконтроллер ESP8266	Микроконтроллер ESP32

`analogWrite(wave1Pin, duty1);`
`analogWriteFreq(freq2);`
`analogWrite(wave2Pin, duty2);`
`}`
	`ledcWrite(channel1, duty1);`
	
	`ledcWrite(channel2, duty2);`
	`}`

Выходное напряжение ШИМ, колеблющееся между высоким и низким уровнями, имеет среднее напряжение VIN × коэффициент заполнения, где VIN – входное напряжение, а коэффициент заполнения – процент времени, в течение которого прямоугольное колебание остается на высоком уровне. Например, в ШИМ 5 В с коэффициентом заполнения 20 % и 70 % имеет среднее напряжение 1 В и 3,5 В соответственно, что приводит к более высокой скорости двигателя, когда двигатель питается от прямоугольных колебаний с коэффициентом заполнения 70 %.

Непрерывная синусоидальная волна заданной частоты генерирует звук, который аппроксимируется прямоугольным колебанием (см. рис. 16-3). Подключим пьезопреобразователь к контакту микроконтроллера ESP8266 или ESP32, генерирующему прямоугольную волну с частотой 440 Гц, что соответствует музыкальной ноте «ля» (A) в середине диапазона октав[89]. Инструкции для микроконтроллера ESP8266 следующие:

```
analogWriteFreq(440);       // define square wave frequency
analogWrite(wavePin, 512);  // square wave, 50% duty cycle
```

Соответствующие инструкции с 10-битным разрешением для микроконтроллера ESP32 будут:

```
ledcSetup(channel, 440, 10)
ledcWrite(channel, 512)
```

Прямоугольное колебание с частотой 440 Гц и рабочим циклом 50 % переключается из высокого на низкий уровень каждые 1136 мкс, что равно $(2 \times \text{частота})^{-1} = 1136 \times 10^{-6}$ с. Альтернативным способом звук может генерироваться с помощью инструкций микроконтроллера ESP8266 и ESP32:

```
digitalWrite(wavePin, !digitalRead(wavePin));
delayMicroseconds(1136);
```

Обратите внимание, что восклицательный знак – это логическое "НЕТ", что переключает вывод к значению `false` (LOW), если до этого значение digitalRead(wavePin) было `true` (HIGH), и наоборот.

[89] 440 Гц (точное значение) соответствует ноте «ля» первой октавы, т. н. «основному тону», относительно которого производится настройка всех музыкальных инструментов. – *Прим. перев.*

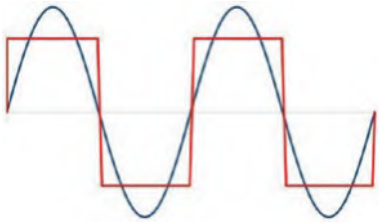


Рисунок 16-3. Прямоугольное колебание, аппроксимирующее синусоидальную волну

Другой способ состоит в том, что цифровая информация о звуке используется для генерации аналогового сигнала и непосредственного воспроизведения звука, вместо того чтобы аппроксимировать звук синусоидальной формы прямоугольным колебанием той же частоты.

Генерация колебаний

Один из способов получения синусоидальных или треугольных колебаний с частотами от 1 Гц до многих МГц – применение модуля генератора сигналов AD9833 (см. рис. 16-4 и табл. 16-2). Библиотека *MD_AD9833* Марко Колли (Marco Colli) доступна в Arduino IDE. Модуль AD9833 может переключаться между двумя формами сигналов, но через один выходной канал. Модуль AD9833 управляется с помощью выводов MOSI (SDATA) и синхронизации CLK (SCLK) аппаратного SPI с помощью инструкции `MD_AD9833 AD(FSYNC)`, где `FSYNC` – это сигнал синхронизации. Выводы модуля AD9833 цифровой земли (DGND) и аналоговой земли (AGND) должны быть предварительно подключены. Для синусоидальной и треугольной форм колебания на выходе вывод VCC модуля AD9833 подключается к напряжению 3,3 В (как показано на рис. 16-4).

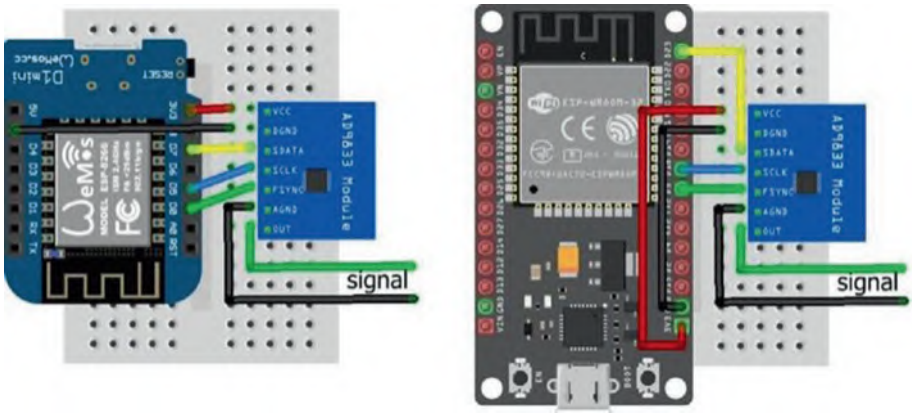


Рисунок 16-4. Генератор сигналов AD9833 синусоидальной или треугольной формы

Таблица 16-2. Модуль генератора сигналов AD9833

Компонент	Выводы ESP8266	Выводы ESP32
AD9833 VCC	3.3V	3.3V
AD9833 DGND	GND	GND
AD9833 SDATA (SPI MOSI)	D7	GPIO 23
AD9833 SCLK (SPI CLK)	D5	GPIO 18
AD9833 FSYNC	D0	GPIO 5
AD9833 AGND	Земля сигнала	Земля сигнала
AD9833 OUT	Выход сигнала	Выход сигнала

Скетч в листинге 16-1 попеременно отображает в течение пяти секунд синусоидальную волну и треугольную волну с частотами 30 кГц и 20 кГц соответственно (см. рис. 16-5). Команда `AD.setActiveFrequency(chan)` задает канал сигнала. Форма колебания задается как `MODE_SINE` или `MODE_TRIANGLE` и устанавливается с помощью команды `AD.setMode(mode)`. Генерация сигнала начинается с инструкции установки частоты `AD.setFrequency(chan, freq)`. Вывод для синхронизации в листинге 16-1 определен для микроконтроллера ESP8266 и должен быть изменен на GPIO 5 для микроконтроллера ESP32 (см. табл. 16-2).

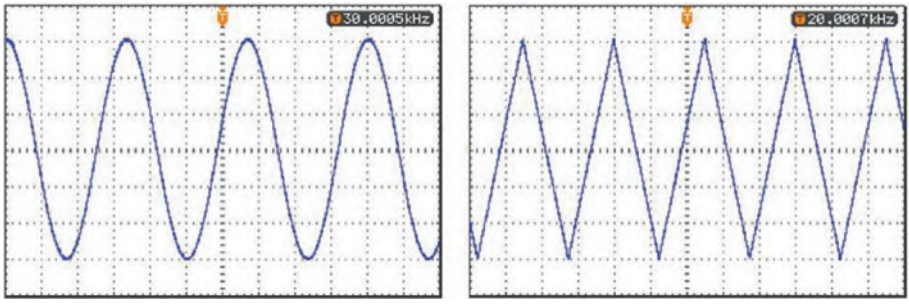


Рисунок 16-5. Генерируемые AD9833 синусоидальные и треугольные колебания

Листинг 16-1. Генератор синусоидальных и треугольных колебаний AD9833

```
#include <SPI.h>                        // include SPI library
#include <MD_AD9833.h>                  // include MD-AD9833 library
int FSYNC = D0;                         // define data synchronisation pin
MD_AD9833 AD(FSYNC);
MD_AD9833::channel_t chan;              // library channel variable
MD_AD9833::mode_t mode;                 // library signal mode variable
unsigned long freq;
void setup()
{
  AD.begin();                           // initialise library
  chan = MD_AD9833::CHAN_0;             // set channel as 0 or 1
  AD.setActiveFrequency(chan);          // activate signal generator
}
void loop()
```

```
{
  wave('s', 30000, 5000);            // call sine and triangle
  wave('t', 20000, 5000);            // wave display function
}
void wave(char shape, unsigned long freq, int timeint)
{                                    // sine wave
  if(shape == 's') mode = MD_AD983 3::MODE_SINE;
  else if(shape == 't') mode = MD_ AD9833::MODE_TRIANGLE;
                                     // triangle wave
  AD.setMode(mode);                  // set the wave form
  AD.setFrequency(chan, freq);       // set signal frequency for channel
  delay(timeint);                    // time to generate signal
  clear();                           // call clear function
}
void clear()                         // function to clear signal
{
  mode = MD_AD9833::MODE_OFF;        // set wave mode to off
  AD.setMode(mode);                  // turn off wave generation
  delay(500);                        // time with no signal
}
```

Цифроаналоговый преобразователь

Цифроаналоговый преобразователь (ЦАП) преобразует цифровое значение в аналоговый сигнал. ЦАП преобразует число из десятичного формата в двоичный формат, формирующий выходное напряжение ЦАП, равное сумме напряжений для каждого бита. При этом напряжения уменьшаются вдвое каждый раз при переходе от старшего бита (MSB) до младшего бита (LSB). Например, число 50 в 8-битном двоичном коде выглядит как B00110010, тогда напряжение ЦАП будет равно $(1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1) \times VCC/2^8 = 50 \times VCC/2^8$, или 0,195 × VCC. 8-разрядный ЦАП с напряжением на опорном выводе, равным VCC, преобразует цифровое значение N в аналоговое напряжение $N \times VCC/2^8$. Микроконтроллер ESP32 включает в себя два ЦАП, каждый с 8-битным разрешением, которые описаны далее в этой главе. Микроконтроллер ESP8266 не включает ЦАП, и в этом разделе ЦАП с микроконтроллером ESP8266 построен отдельно.

Основой уменьшения напряжения вдвое для каждого бита ЦАП служит делитель напряжения. Делитель напряжения с входным напряжением V_{IN} и двумя резисторами R1 и R2 имеет выходное напряжение V_{OUT} в точке соединения этих двух резисторов: $V_{IN} \times \left(\frac{R2}{R1+R2} \right)$ (см. рис. 16-6). Если два резистора имеют равные значения, то выходное напряжение составляет половину входного напряжения. В иллюстративных целях ЦАП может быть представлен в виде цепочки делителей напряжения, выдающей величины напряжений, соответствующие значениям каждого бита между MSB и LSB в двоичном формате[90].

[90] То есть цепочки, содержащей резисторы с номиналами, например, 1 кОм, 2 кОм, 4 кОм... 64 кОм, 128 кОм. Причем автор не акцентирует внимание на том, что сопротивления резисторов старших 2–3 разрядов в цепочке должны быть согласованы с погрешностью (для 8 разрядов) не более 0,2 %, что, конечно, в микроэлектронном исполнении недостижимый идеал. На практике подобные делители строятся в виде набора звеньев, состоящих всего из двух номиналов R и 2R, для которых достичь согласования с нужной погрешностью намного проще (см. далее). – *Прим. перев.*

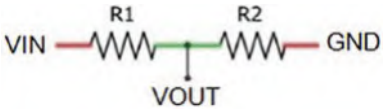


Рисунок 16-6. Делитель напряжения

На практике делитель напряжения строится в виде звеньев R-2R. 8-разрядный ЦАП, показанный на рис. 16-7, содержит восемь пар резисторов R и 2R, где номинал R равен 1 кОм (см. соединения в табл. 16-3). Самый правый резистор 2 кОм представляет старший разряд (MSB) R-2R-цепочки 8-разрядного ЦАП. Делитель напряжения, образованный парой резисторов 2 кОм с левого края, представляющий младший разряд (LSB), формирует в средней точке напряжение 0,5 от входного. В этой левой паре резисторы 2 кОм подключаются параллельно, когда на обоих резисторах напряжение равно нулю. Общее суммарное сопротивление двух параллельных резисторов R1 и R2 равно $\frac{1}{R_{>1I}} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{R_1 + R_2}{R_1 \times R_2}$. Следовательно, при параллельном соединении левая пара резисторов 2 кОм имеет суммарное сопротивление 1 кОм и соединена последовательно с резистором A со значением также 1 ком, так что пара 2 кОм и 1 кОм имеет суммарное сопротивление опять 2 кОм. Комбинация левой пары резисторов 2 кОм и резистора A с резистором B образует второй делитель напряжения, который в своей средней точке снова делит входное напряжение пополам. Третий делитель напряжения формируется комбинацией левой пары резисторов 2 кОм и резисторов A, B и C с резистором D. На каждом этапе напряжение на средней точке делителя напряжения уменьшается вдвое, поэтому единичное (HIGH) значение бита для младшего разряда (LSB) соответствует напряжению $V_{IN}/2^8$, так как вся цепочка состоит из восьми делителей напряжения.

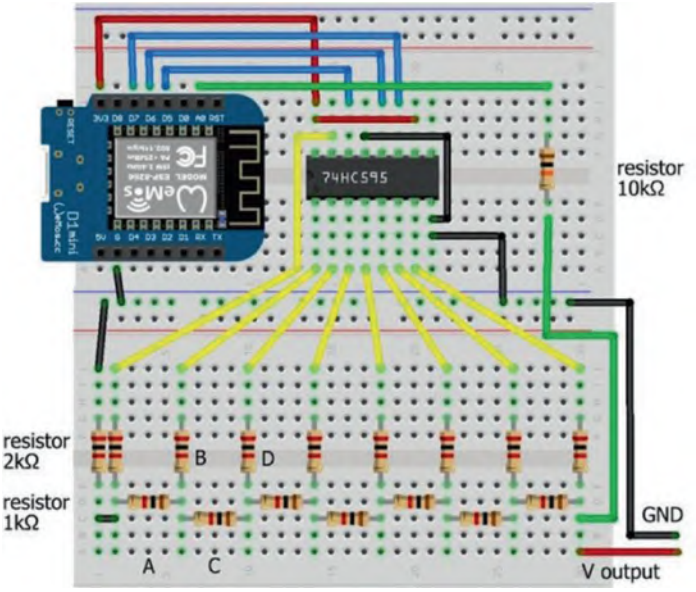


Рисунок 16-7. Цифроаналоговый преобразователь R-2R с платой LOLIN (WeMos) D1 мини

Выходное напряжение от 8-разрядного ЦАП R-2R, соответствующее цифровому значению, формируется с помощью использования двоичного представления для установки каждого из восьми выводов делителей напряжения в высокий (HIGH) или низкий (LOW) уровень. Например, 8-разрядное двоичное представление числа 156 равно B10011100; тогда выводы делителей напряжения 2, 3, 4 и 7 (пронумерованные справа налево, начиная с нуля) установлены в высокий уровень, формируя напряжение на выходе ЦАП, равное $(1 \times 2^7 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2) \times VCC/2^8 = 156 \times VCC/2^8 = 0{,}6094 \times VCC$.

На плате ESP8266 имеется восемь цифровых выводов, но выводы D3, D4 и D8 имеют встроенные подтягивающие или заземляющие резисторы для поддержания состояния выводов во время процесса загрузки микроконтроллера ESP8266. Вместо установки состояний выводов 8-разрядного ЦАП R-2R с помощью микроконтроллера ESP8266 напрямую состояния выводов устанавливаются с помощью сдвигового регистра с последовательным входом и параллельным выходом. Регистр сдвига 74HC595 загружает байт данных, состоящий из 8 бит данных, по одному биту за такт. При установленном в низкий уровень выводе сдвига SRCLK биты данных загружаются в регистр через вывод последовательного ввода данных (SER), управляемый тактовыми импульсами на выводе RCLK. После загрузки всех 8 бит на вывод сдвига подается высокий уровень. Инструкции для передачи данных через регистр сдвига следующие:

```
digitalWrite(latchPin, LOW);          // set the latch to LOW
shiftOut(dataPin, clockPin, MSBFIRST, number)
                                      // load number as a byte
digitalWrite(latchPin, HIGH)          // set latch to HIGH
```

MSBFIRST указывает, что сначала загружается старший бит (MSB), соответствующий делителю напряжения в правой части рис. 16-7.

Выводы микросхемы регистра сдвига 74HC595 пронумерованы от 1 до 16, а вырез или точка на краю корпуса указывает на сторону с выводами 1 и 16. Подключения к регистру сдвига 74HC595 показаны в табл. 16-3, при этом для выводов $\overline{\text{SRCLR}}$ (очистить регистр) и $\overline{\text{OE}}$ (разрешение выхода) надстрочная черта указывает, что вывод активен при низком уровне. Вывод 3,3 В платы ESP8266 питает регистр сдвига, напряжение питания которого составляет 2–6 В.

Таблица 16-3. Цифроаналоговый преобразователь R-2R с платой ESP8266

Компонент	Подключено к	Также к
Резистор 2 кОм с левого края (LSB)	Второй резистор 2 кОм	ESP8266 GND
7 резисторов 2 кОм	74HC595 выв. 15, 1–6 QA-QG	Резисторы 1 кОм
Правый резистор 2 кОм (MSB)	74HC595 выв. 7 QH	
Правый резистор 2 кОм (MSB)	ESP8266 A0	Выход напряжения ЦАП
74HC595 выв. 8 GND	ESP8266 GND	
74HC595 выв. 9 QH'	Не подключен	
74HC595 выв. 10 $\overline{\text{SRCLR}}$	ESP8266 3V3	
74HC595 выв. 11 SRCLK	ESP8266 D7	

Окончание табл. 16-3

Компонент	Подключено к	Также к
74HC595 выв. 12 RCLK	ESP8266 D6	
74HC595 выв. 13 $\overline{OE}$	ESP8266 GND	
74HC595 выв. 14 SER	ESP8266 D5	
74HC595 выв. 16 VCC	ESP8266 3V3	

10-разрядный аналого-цифровой преобразователь (АЦП) микроконтроллера ESP8266 преобразует напряжение от 0 до 3,2 В на выводе аналогового входа A0 для численных значений от 0 до 1023. Команда `analogRead(A0)` считывает напряжение на выводе аналогового входа. Опорное напряжение АЦП микроконтроллера ESP8266 составляет 1 В; внутренний делитель напряжения, состоящий из резисторов 220 кОм и 100 кОм (см. рис. 16-8), увеличивает максимальное напряжение на выводе аналогового входа до 3,3 В. При подаче напряжения V_{IN} на вывод аналогового входа платы ESP8266 соответствующее значение АЦП будет равно $V_{IN} \times \frac{100\ кОм}{(220+100)\ кОм} \times 1024$.

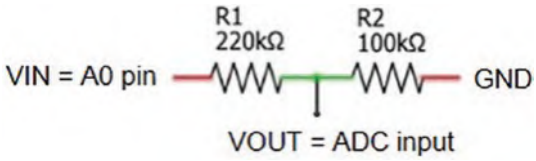


Рисунок 16-8. Аналого-цифровой преобразователь

Значения АЦП для входных напряжений от 3,2 В до 3,3 В ограничены значением 1023. Резистор 10 кОм, подключенный между входным напряжением и выводом аналогового входа платы ESP8266, увеличивает предел 3,2 В на выводе аналогового входа до 3,3 В. Входному напряжению V_{IN} соответствуют показания АЦП, равные $V_{IN} \times \frac{100\ кОм}{(10+220+100)\ кОм} \times 1024$. Учтите, что выходное напряжение с 8-разрядного ЦАП типа R-2R, соответствующее цифровому значению N, равно $N \times 3{,}3\ В/2^8$, так как микроконтроллер ESP8266 имеет напряжение питания 3,3 В.

В листинге 16-2 показано использование 8-разрядного ЦАП с резистором R-2R для преобразования цифровых значений от 0 до 255 в напряжения. Регистр сдвига 74HC595 устанавливает восемь состояний делителя напряжения на основе двоичного формата исходного числа, а выходное напряжение 8-разрядного ЦАП R-2R считывается аналоговым выводом A0 платы ESP8266.

Встроенный делитель напряжения микроконтроллера ESP8266 и последовательный резистор 10 кОм уменьшают входное напряжение на выводе аналогового входа платы ESP8266, который имеет максимальное входное напряжение 1 В. Выходное напряжение 8-разрядного ЦАП R-2R – это показания

АЦП микроконтроллера ESP8266, масштабируемые в соответствии с формулой $\frac{(10+220+100)\text{ кОм}}{100\text{ кОм}} \times \frac{1}{1024}$.

Листинг 16-2. Цифроаналоговый преобразователь R-2R

```
int dataPin = D5;                              // shift register data
int latchPin = D6;                             // latch and clock pins
int clockPin = D7;
int Vin;                                       // voltage on analog pin
float voltage, predict;                        // voltage divider effect and
float voltDivid = 1000.0*(10+220+100)/100;
                                               // adjustment to mV
void setup()
{
  Serial.begin(115200);                        // Serial Monitor baud rate
  Serial.println();
  pinMode(dataPin, OUTPUT);                    // shift register pins as output
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}
void loop()
{
  for (int i=0; i<256; i=i+50)                 // incremental increases
  {
    digitalWrite(latchPin, LOW);               // start loading shift register
    shiftOut(dataPin, clockPin, MSBFIR ST, i);
                                               // load 8-bit number
    digitalWrite(latchPin, HIGH);              // end loading shift register
    Vin = analogRead(A0);                      // read R-2R ladder voltage
    voltage = Vin * voltDivid / 1024.0 ;
                                               // scaled R-2R ladder voltage
    predict = i * 3300.0 / 256;                // predicted output voltage
    Serial.print(i);Serial.print("\t");
    Serial.print(Vin);Serial.print("\t ");     // display results
    Serial.print(voltage,0);Serial.print("\t");
    Serial.println(predict,0);
    delay(200);
  }
}
```

Максимальное напряжение 8-разрядного ЦАП с делителем R-2R равно $\text{VCC} \times \sum_{i=1}^{8} 1/2^i = 0,996 \times \text{VCC} = 3,287$ В, что соответствует цифровому значению 255 и аналоговому показанию 1020. Для выходного напряжения 3,2999 В требуется 16-разрядный ЦАП R-2R, тогда указанное значение будет соответствовать аналоговому показанию 1023.

Генерация колебаний

Синусоидальные, квадратные, треугольные и пилообразные колебания также генерируются с помощью 8-разрядного ЦАП R-2R. Значения синусоиды не всегда положительны, так как при сдвиге фазы 90° и 270° они равны 1 и –1 соответственно. Если к значениям синусоидальной кривой, находящейся

в пределах 8-битного диапазона ЦАП (0, 255), добавить константу 120, диапазон сдвинется от (–120, 120) до (8, 248). Константу 128 использовать нельзя, так как для значения 256 требуется 9-разрядный ЦАП. Масштабированное значение преобразуется в двоичный формат, при этом состояния делителя напряжения устанавливаются на соответствующий бит числа в двоичном формате. 8-разрядный ЦАП R-2R, состоящий из восьми делителей напряжения, генерирует напряжения на каждом выводе делителя. Сумма напряжений с выводов 8-разрядного ЦАП R-2R – это требуемое аналоговое напряжение. Например, масштабированная синусоидальная волна при значении фазы 45° имеет значение 212 = 128 + 120 × √0,5, которое имеет двоичный формат B11010100, и напряжение ЦАП будет равно $(1 \times 2^7 + 1 \times 2^6 + 1 \times 2^4 + 1 \times 2^2) \times VCC/2^8 = 0{,}828 \times VCC$.

Когда 8-разрядный выход ЦАП R-2R подключен к мини-динамику, раздается звук, соответствующий масштабированной синусоидальной волне. Частота звука зависит от времени, затрачиваемого микроконтроллером ESP8266 на генерацию масштабированных значений синусоиды и преобразование их в двоичный формат для 8-разрядного ЦАП R-2R. Микроконтроллеру ESP8266 требуется 15,5 мс для генерации и загрузки в регистр сдвига масштабированных значений синусоиды за один период 360° с интервалами в один градус, что соответствует частоте 64 Гц.

Различные частоты генерируются путем умножения угла синусоиды на постоянную, что изменяет количество периодов за единицу времени, причем с увеличением значения множителя разрешение[91] синусоидальной кривой уменьшается. Значения с вывода потенциометра устанавливают множитель (см. рис. 16-9), а команда `mult = 20.0*analogRead(potPin)/1023.0` масштабирует показания потенциометра до произвольного максимума, равного 20, с результирующей максимальной частотой синусоиды 1,28 кГц. Подключение потенциометра приведено в табл. 16-4.

[91] То есть количество сгенерированных значений колебания в секунду (в оцифровке звука это называется частота дискретизации). – *Прим. перев.*

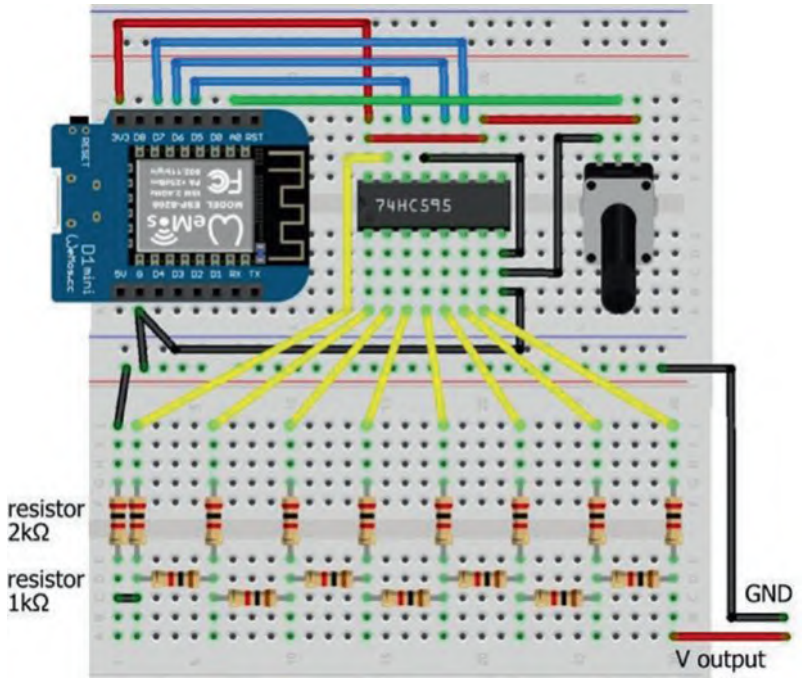


Рисунок 16-9. 8-разрядный ЦАП с цепочкой R-2R, регистр сдвига и плата LOLIN (WeMos) D1 mini

Таблица 16-4. Подключение потенциометра для генерации колебаний

Компонент	Подключено к
Потенциометр VCC	ESP8266 VCC
Движок потенциометра (средний)	ESP8266 A0
Потенциометр GND	ESP8266 GND

Скетч в листинге 16-3 изменяет множитель угла синусоиды для изменения частоты звука, генерируемого выходом 8-разрядного ЦАП R-2R. В функции `loop` множителем для угла синусоиды является масштабированное значение сигнала потенциометра. Длительность периода и частота синусоиды отображаются каждые 1000 периодов. Для вычисления значений синусоидальной кривой требуется, чтобы угол был определен в радианах, так что угол в градусах преобразуется в радианы по формуле *радиан = угол* × *π*/180. В листинге 16-3 используется константа *PI*, определенная в Arduino IDE как 3,14159.

Листинг 16-3. Генерация синусоиды с помощью 8-разрядного ЦАП R-2R

```
int dataPin = D5;                    // shift register data
int latchPin = D6;                   // latch and clock pins
int clockPin = D7;
int count = 0;
float sum, angle, Hz;
```

```
unsigned long lastTime;
int val, sign, mult, cycleTime;
void setup()
{
   Serial.begin(115200);                    // Serial Monitor baud rate
   pinMode(dataPin, OUTPUT);                // shift register pins as output
   pinMode(latchPin, OUTPUT);
   pinMode(clockPin, OUTPUT);
}
void loop()
{
   mult = 20.0*analogRead(A0)/1023.0;       // scale potentiometer value
   for (int deg=0; deg<360; deg=deg+mul t)  // cycle through 360°
   {
   angle = deg*PI/180.0;                    // convert degrees to radians
      sum = sin(angle);
      val = round(128+120.0*sum);           // scaled sine wave value
      digitalWrite(latchPin, LOW);          // start loading shift register
      shiftOut(dataPin, clockPin, MSBFIR ST, val );
                                            // load 8-bit number
      digitalWrite(latchPin, HIGH);         // end loading shift register
   }
   count ++;
   if(count > 999)                          // display every 1000 cycles
   {
      cycleTime = millis() - lastTime;      // time (ms) for cycle
      Hz = 1000.0 * count / cycleTime;      // frequency (Hz)
      Serial.print(mult);Serial.print("\t");
      Serial.print(Hz);Serial.print("Hz\t");
      Serial.print(1.0*cycleTime/count);Serial.println("ms");
      count = 0;                            // reset counter
      lastTime = millis();                  // reset timer
   }
}
```

В листинге 16-3 значения синусоидальной кривой для угла (*angle*) рассчитываются с помощью команды `sum = sin (angle)`.

Прямоугольное колебание строится из шести нечетных гармоник (синусоидальных составляющих) в виде ряда Фурье с помощью инструкций:

```
sum = 0;
for (int i=1; i<12; i=i+2) sum = sum + sin(i*angle)/i;
```

Колебание треугольной формы строится с теми же нечетными гармониками, что и прямоугольное, но с чередующимися знаками и делителем числа гармоник в квадрате. Инструкции:

```
sum = 0;
sign = -1;
for (int i=1; i<12; i=i+2)
{
   sign = -sign;
   sum = sum + sign*sin(i*angle)/(i*i);
}
sum = sum/2;
```

Колебание пилообразной формы строится со всеми гармониками (четными и нечетными), но с разными знаками. Инструкции, использующие первые девять гармоник, следующие:

```
sum = 0;
sign = -1;
for (int i=1; i<10; i++)
{
   sign = -sign;
   sum = sum + sign*sin(i*angle)/i;
}
sum = sum/2;
```

Треугольное колебание с прямым углом просто генерируется путем замены инструкций `for (int deg=0; deg<360; deg=deg+mult) {...}` на инструкции:

```
for (int i=0; i<256; i++)
{
   digitalWrite(latchPin, LOW);
   shiftOut(dataPin, clockPin, MSBFIRST, i);
   digitalWrite(latchPin, HIGH);
}
```

а треугольная форма создается с помощью добавления инструкций:

```
for (int i=1; i<255; i++)
{
   digitalWrite(latchPin, LOW);
   shiftOut(dataPin, clockPin, MSBFIRST, 255-i);
   digitalWrite(latchPin, HIGH);
}
```

Синусоидальная кривая на рис. 16-10 иллюстрирует разрешение 8-разрядного ЦАП с R-2R-делителем. Когда число в двоичном формате представлено 8 битами, получается, по существу, непрерывная синусоида с шагом, равным единице. По мере уменьшения количества битов числа размер шага увеличивается, а разрешение уменьшается. Аппроксимации синусоидальной кривой, создаваемые ЦАП R-2R со всеми 8 битами и с тремя и двумя старшими битами, демонстрируют быстрое улучшение разрешения с увеличением числа битов.

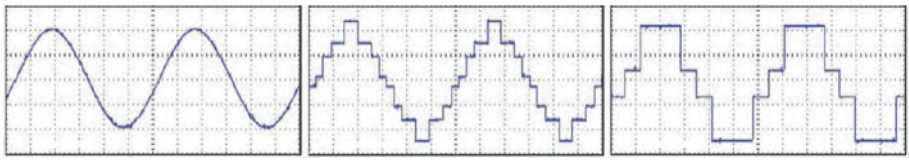


Рисунок 16-10. Приближения синусоиды с разрешением 8 бит, три и два старших бита

8-разрядный ЦАП ESP32

Микроконтроллер ESP32 включает в себя два 8-разрядных ЦАП, называемых DAC1 и DAC2, на выводах GPIO 25 и GPIO 26. Команда `dacWrite(DACpin, value)` генерирует напряжение величиной *value* × VCC/2^8, где VCC – напряжение питания микроконтроллера ESP32. Скетч в листинге 16-4 генерирует синусоиду из листинга 16-3 с меньшим количеством инструкций и с более высокой частотой, поскольку не требуется внешний ЦАП с R-2R-цепочкой и регистром сдвига. Частота меняется за счет множителя, как и в листинге 16-3.

Листинг 16-4. Генерация синусоидальной волны с помощью ЦАП ESP32

```
int DACpin = DAC1;                          // DAC pin on GPIO 25
float angle;
int val;
void setup()                                // nothing in setup function
{}
void loop()
{
   for (int deg=0; deg<360; deg++)          // cycle through 360°
   {
      angle = deg*PI/180.0;                 // convert degrees to radians
      val = round(128+120.0*sin(angle));    // scaled sine wave value
      dacWrite(DACpin, val);                // output voltage
   }
}
```

12-разрядный ЦАП

12-разрядный модуль ЦАП MCP4725 (см. рис. 16-11 и подключения в табл. 16-5) является альтернативой самодельному 8-разрядному ЦАП R-2R. Модуль MCP4725 управляется через интерфейс I2C с адресом I2C от 0x60 до 0x67, в зависимости от версии модуля.

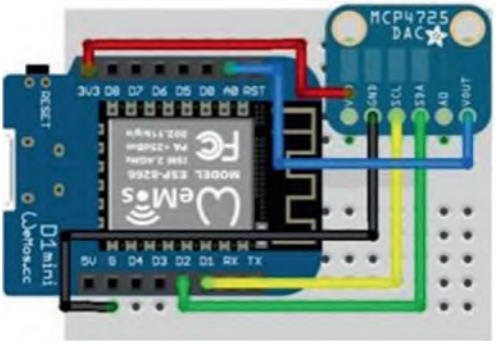


Рисунок 16-11. 12-разрядный модуль ЦАП MCP4725 и плата LOLIN (WeMos) D1 mini

Таблица 16-5. 12-разрядный модуль ЦАП MCP4725 и плата ESP8266

Компонент	Подключается к
MCP4725 VOUT	ESP8266 A0
MCP4725 GND	ESP8266 GND
MCP4725 SCL	ESP8266 D1
MCP4725 SDA	ESP8266 D2
MCP4725 VCC	ESP8266 3.3V

Адрес I2C получен с помощью скетча сканирования I2C, приведенного в листинге 16-5. При передаче на устройство I2C устройство возвращает значение 0, указывающее на успешную передачу. Адреса I2C от 0x00 до 0x07 и от 0x78 до 0x7F зарезервированы и не сканируются. Адреса датчиков и модулей I2C доступны по адресу learn.adafruit.com/i2c-addresses/the-list.

Листинг 16-5. Сканер I2C

```
#include <Wire.h>                    // include Wire library
int device = 0;                      // set device counter
void setup()
{
   Serial.begin (115200);            // Serial Monitor baud rate
   Serial.println();
   Wire.begin();                     // start I2C bus
   for (int i=8; i<127; i++)         // scan through channels 8 to 126
   {
      Wire.beginTransmission (i);    // transmit to device at address i
      if (Wire.endTransmission () == 0)
      {                              // device response to transmission
         Serial.print("Address 0x");
         Serial.println(i, HEX);     // display I2C address in HEX
         device++;                   // increment device count
         delay(10);
      }
   }
   Serial.print(device);             // display device count
   Serial.println(" device found");
}
void loop()                          // nothing in loop function
{}
```

Команда `setVoltage(value, false)` библиотеки *Adafruit MCP4725* устанавливает выходное напряжение VCC×*value*/2^{12}, где VCC – напряжение питания модуля MCP4725 3,3–5 В, которое также является опорным выходным напряжением. Выходное напряжение 12-разрядного модуля ЦАП MCP4725 можно проконтролировать подключением вывода модуля VOUT к аналоговому входу микроконтроллера ESP8266 A0, получив таким образом значение, равное `VCC*analogRead(A0)/1024`, где VCC – напряжение питания микроконтроллера.

Скетч для генерации различных напряжений и синусоиды с помощью 12-разрядного модуля ЦАП MCP4725 приведен в листинге 16-6. При генерации синусоидальной кривой угол определяется в радианах, поэтому угол,

измеренный в градусах, преобразуется в радианы по формуле *радиан = угол* × *π*/180. Единица добавляется к синусоидальному значению, чтобы получить положительное число для генерации выходного напряжения. Используется множитель 2047 для значения синуса +1, а не 2048, поскольку команда `dac.setVoltage(4096, false)` приводит к выходному напряжению 0 В. Различные частоты генерируются умножением угла синусоидальной волны на постоянную, что увеличивает количество периодов в заданный промежуток времени, причем при более высоких значениях множителя разрешение синусоидальной кривой уменьшается.

Листинг 16-6. Синусоида при помощи 12-разрядного модуля ЦАП MCP4725

```
#include <Adafruit_MCP4725.h>          // include Adafruit MCP4725 lib
Adafruit_MCP4725 dac;                  // associate dac and MCP4725 lib
float voltDivid = 3200;                // voltage divider adjustment
float VCC = 3300;                      // operating voltage
int flag = 0;                          // flag to switch to sine wave
float predict, voltage, value;
int Vin;
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  dac.begin(0x60);                     // I2C address of MCP4725
}
void loop()
{
  if(flag < 1)
  {
    for(int i=0; i<4096; i=i+50)
    {
      dac.setVoltage(i, false);              // set output voltage
      Vin = analogRead(A0);                  // read DAC voltage
      voltage = Vin * voltDivid /1024.0;     // scaled output voltage
      predict = i * VCC / 4096.0;            // predicted output voltage
      Serial.print(voltage);Serial.print("\t");
      Serial.println(predict);               // display voltages
    }
    delay(2000);
    flag = 1;                                // switch to sine wave only
  }
  for (int deg=0; deg<360; deg++)            // generate sine wave
  {
    value = 2047.0*(sin(deg*PI/180.0)+1);    // scaled sine wave value
    dac.setVoltage(value, false);            // set output voltage
  }
}
```

Считывание значений синусоиды из справочной таблицы сокращает время обработки по сравнению с вычислением, что увеличивает количество периодов в единицу времени и, следовательно, частоту. Микроконтроллер ESP8266 имеет 4 МБ флеш-памяти и всего 50 кБ SRAM, поэтому таблица значений хранится во флеш-памяти программ PROGMEM вместе со скетчем, а не в SRAM, где переменные создаются и обрабатываются в скетче. Инструкция для хранения массива во флеш-памяти представляет собой константу `arrayname[] PROGMEM = {array`

values}. Доступ к *i*-му значению массива осуществляется с помощью команды pgm_read_type(arrayname + i). Символьные или целочисленные данные хранятся во флеш-памяти с типом переменной, определенным как unsigned char или uint16_t, и доступны с помощью pgm_read_byte или pgm_ read_word соответственно.

В листинге 16-7 значения синусоиды с интервалом в 10° сохраняются во флеш-памяти. Для примера, время генерации 500 периодов измеряется для определения частоты генерируемых колебаний. Синусоида также генерируется путем непосредственного вычисления значений для сравнения с методом хранения значений во флеш-памяти.

Листинг 16-7. Хранение значений синусоиды во флеш-памяти

```
#include <Adafruit_MCP4725.h>                // include Adafruit MCP4725 lib
Adafruit_MCP4725 dac;                        // associate dac and MCP4725 lib
const uint16_t lookup[] PROGMEM = {          // sine wave in flash memory
2047,2402,2747,3071,3363,3615,3820,3971,4063,4094,
4063,3971,3820,3615,3363,3071,2747,2402,2047,1692,
1347,1024, 731, 479, 274, 123, 31, 0, 31, 123,
274, 479, 731,1024,1347,1692                 // 2047×(sin(x°×π/180) + 1)
};
int value, cycle;
unsigned long lastTime = 0;
float freq;
void setup()
{
  Serial.begin(115200);                      // Serial Monitor baud rate
  Serial.println();
  dac.begin(0x60);                           // I2C address of MCP4725
}
void loop()
{
  lastTime = millis();                       // start timer
  while (cycle < 500)                        // sine wave for 500 cycles
  {
    for (int i=0; i<36; i++)
    {
      value = pgm_read_word(lookup+i);       // values from flash memory
      dac.setVoltage(value, false);          // set output voltage
    }
    cycle++;
    yield();                                 // required to prevent timeout
  }
  freq = 1000 * 500.0/(millis() - last Time); // wave frequency
  Serial.print("lookup ");Serial.print(freq);
  cycle = 0;
  lastTime = millis();
  while (cycle < 500)
  {
  for (int deg=0; deg<360; deg=deg+10)       // generate sine wave
  {                                          // scaled sine wave value
    value = 2047.0*(sin(deg*PI/180.0)+1) ;
    dac.setVoltage(value, false);            // set output voltage
  }
  cycle++;
  yield();
```

```
    }
    freq = 1000 * 500.0/(millis()-lastTime);
    Serial.print("\tcalc ");Serial.println(freq);
    cycle = 0;
  }
```

Частоты синусоиды при различных методах генерации значений синуса с интервалом 10° суммированы в табл. 16-6. Синусоидальные колебания генерировались 8-разрядным ЦАП микроконтроллера ESP32 и микроконтроллером ESP8266 с 8-разрядным ЦАП R-2R, а также 12-разрядным ЦАП MCP4725. Частота синусоидальных колебаний, генерируемых микроконтроллером ESP32, была вдвое выше, чем у микроконтроллера ESP8266 с 8-разрядным ЦАП R-2R, и в 20 раз выше, чем у 12-разрядного модуля ЦАП MCP4725. Более высокое разрешение 12-разрядного ЦАП MCP4725 по сравнению с 8-разрядным ЦАП R-2R компенсируется более медленной генерацией колебаний. Для 12-разрядного модуля MCP4725 хранение значений синусоиды во флеш-памяти приводило к незначительно более высокой частоте колебаний, чем при вычислении значений синусоидальной волны. Модуль AD9833 – самый мощный метод генерации синусоидальной волны[92].

Таблица 16-6. Способы и частоты генерации синусоиды

Метод генерации значений с интервалом 10°	Частота (Гц)
Модуль AD9833	До МГц
ESP32 8-bit DAC	1467
ESP8266 8-разрядный ЦАП с цепочкой R-2R	647
ESP8266 + 12-битовый ЦАП MCP4725	66
С таблицей поиска в памяти	71

Итоги

Прямоугольные колебания с переменным коэффициентом заполнения периода создавались микроконтроллерами ESP8266 и ESP32 для управления внешними устройствами. Цифроаналоговый преобразователь (ЦАП) с резисторной цепочкой R-2R описан для микроконтроллера ESP8266. Синусоидальные, прямоугольные, треугольные и пилообразные колебания генерировались способом суммирования гармоник в виде ряда Фурье. Различные звуки производились от выходного напряжения 8-разрядного R-2R ЦАП при изменении

[92] Описанный выше (см. раздел «Генерация колебаний» в этой главе) модуль AD9833 отличается тем, что генерация сигналов в нем происходит аппаратно, команды контроллера только запускают или останавливают этот процесс. Тогда как любые ЦАП (не важно, встроенные или внешние) требуют со стороны контроллера задания каждой точки генерируемой кривой. Аппаратные средства всегда превышают программные по скорости выполнения. Не забывайте также о необходимости точной (не хуже 0,2 %) подгонки сопротивлений резисторов в самодельном ЦАП с цепочкой R-2R, особенно в старших 2–3 разрядах. Автор на этом не заостряет внимания, однако отсутствие такой подгонки равносильно снижению количества разрядов, т. е. искажению формы синусоиды. – *Прим. перев.*

частоты синусоиды. Выходные напряжения также генерировались с помощью 12-разрядного модуля ЦАП MCP4725. Аппаратный AD9833 модуль генерировал синусоидальные и треугольные волны с частотами в несколько мегагерц. 8-разрядный ЦАП микроконтроллера ESP32 создает синусоидальные колебания с удвоенной частотой.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN D1 mini (WeMos) или NodeMCU
- Микроконтроллер ESP32: плата ESP32 DEVKIT DOIT или NodeMCU
- Модуль MCP4725 12-бит ЦАП
- Модуль генератора колебаний: AD9833
- Сдвиговый регистр: 74HC595
- Потенциометр: 10 кОм
- Резисторы: 1 кОм 7 шт., 2 кОм 9 шт.

Глава 17

Генерация сигнала с помощью микросхемы таймера 555

В главе 16 («Генерация сигналов») микроконтроллер ESP8266 или ESP32 генерировал прямоугольные импульсы с переменным коэффициентом заполнения или синусоидальные и треугольные колебания в сочетании с модулем генератора сигналов AD9833. Синусоидальные колебания также генерировались с добавлением внешнего модуля ЦАП MCP4725 к микроконтроллеру ESP8266 или встроенного ЦАП микроконтроллера ESP32. В главе 5 («MP3-плеер») микроконтроллер сигнализировал модулю MP3-плеера о воспроизведении трека, когда перемещение приводило к срабатыванию пироэлектрического датчика (PIR). Даже несмотря на то, что скетчи сигнализации часто содержали менее 20 строк кода, для генерации сигнала всегда требовался микроконтроллер. Генерация сигнала с помощью интегральной схемы (IC) таймера 555 является альтернативой использованию микроконтроллера и дает представление о том, как комбинировать электронные компоненты для конкретного приложения. Микросхема таймера 555 предназначена для приложений, требующих создания таймеров и генераторов, начиная от генерации импульсов и звука, часов, таймеров и охранной сигнализации до управления мощностью с помощью ШИМ или любого другого приложения, требующего контроля времени. В этой главе описывается генерация сигналов с помощью микросхемы таймера 555 в дополнение к главе 16 («Генерация сигналов»). В качестве иллюстративного приложения пример MP3-плеера и датчика PIR из главы 5 («MP3-плеер») переработан с использованием микросхемы таймера 555.

Микросхема таймера 555

Микросхема таймера 555 имеет восемь выводов, обозначения выводов приведены на рис. 17-1 слева. Однако представление микросхемы таймера 555, как показано на рис. 17-2, с выводами *Threshold, Trigger* и *Control*, а также выводами *Output* и *Discharge*, сгруппированными по отдельности, облегчает интерпретацию схем с этой микросхемой. Для наглядных монтажных схем микросхема таймера 555 представлена как на рис. 17-1 (справа), чтобы свести к минимуму перекрытие соединений.

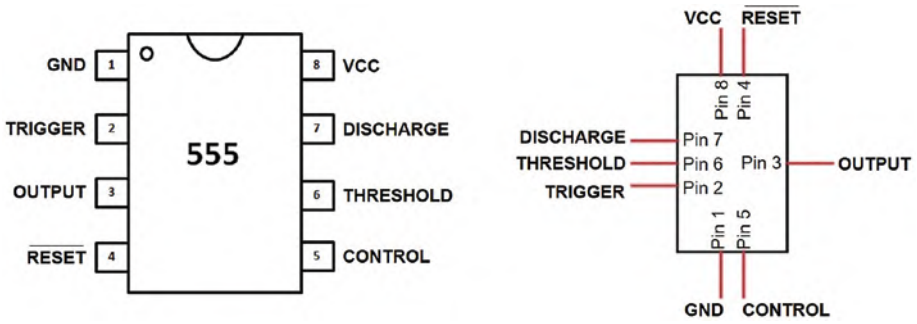


Рисунок 17-1. Разводка выводов микросхемы таймера 555

Микросхема таймера 555 включает в себя два компаратора напряжений, представленных треугольниками на рис. 17-2, с опорными напряжениями 2/3 VCC и 1/3 VCC, которые генерируются двумя делителями напряжения, образованными тремя резисторами 5 кОм. Микросхема таймера 555 в основном управляется напряжениями на выводах *Threshold* и *Trigger*. Когда напряжение на пороговом выводе *Threshold* превышает 2/3 VCC, триггер установки/сброса, представленный квадратиком *Reset/Set* на рис. 17-2, сбрасывается, состояние выхода *Output* изменяется с высокого (*HIGH*) на низкий (*LOW*) уровень, а NPN-транзистор включается для замыкания вывода *Discharge* на GND. И наоборот, когда напряжение на выводе *Trigger* становится меньше 1/3 VCC, триггер *Reset/Set* устанавливается, состояние вывода *Output* изменяется с *LOW* на *HIGH*, и разрядный NPN-транзистор выключается, отключая вывод *Discharge* от GND.

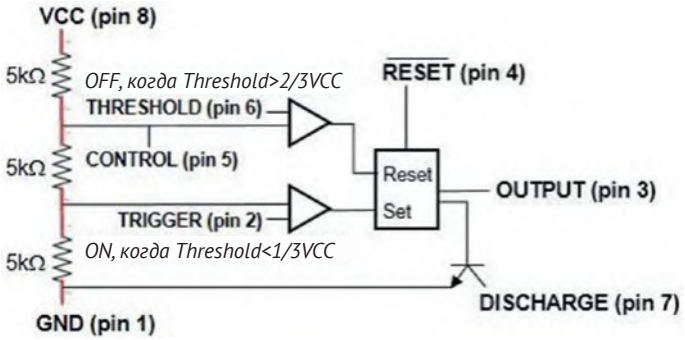


Рисунок 17-2. Схема таймера 555

Для контроля времени, в течение которого напряжения на выводах триггера и порога находятся в пределах 1/3 VCC и 2/3 VCC, к микросхеме таймера 555 подключается комбинация из резистора и конденсатора (*RC*-цепочка). Напряжение на конденсаторе в момент *t* с зарядки или разрядки равно $V(1 - e^{t/RC})$, или $Ve^{t/RC}$, где *V* – напряжение питания. Чем выше значение произведения *RC*, тем больше интервал времени между изменениями состояния триггера и напряжения на выводе *Output*. Подробное описание приведено в следующем разделе «Режим генерации». Вкратце: пока конденсатор заряжается, вывод *Output* находится на высоком уровне, и увеличивающееся напряжение конденсатора

действует на вывод *Threshold*. Когда напряжение достигает 2/3 VCC, состояние вывода *Output* изменяется с *HIGH* на *LOW*, и конденсатор разряжается через NPN-транзистор (разрядный вывод *Discharge*) на землю GND. Когда конденсатор разряжается, уменьшающееся напряжение на конденсаторе действует на вывод *Trigger*; когда напряжение достигает 1/3 VCC, состояние вывода изменяется с *LOW* на *HIGH*, и конденсатор снова начинает заряжаться.

Микросхема таймера 555 имеет три режима работы: моностабильный, бистабильный и нестабильный (режим генератора). В моностабильном режиме запуск микросхемы таймера 555 приводит к положительному импульсу в течение фиксированного времени на выходе, который затем возвращается в нулевое состояние до повторного включения. Приложения моностабильного режима включают временные задержки. В бистабильном режиме высокое или низкое состояние выхода устанавливается внешним переключателем и сохраняется при его отключении[93]. В нестабильном (генераторном) режиме микросхема таймера 555 генерирует прямоугольные колебания с фиксированной частотой и коэффициентом заполнения, определяемыми величинами резистора и конденсатора. Применения режима генерации включают управление двигателями или лампами с широтно-импульсной модуляцией, генерацию тона и звука, синхронизацию тактовых импульсов, а также прямоугольные, треугольные и синусоидальные колебания, когда к микросхеме таймера 555 подключена резисторно-конденсаторная (*RC*) или индуктивно-конденсаторная (*LC*) цепочка.

Вывод сброса *Reset* активен при подаче низкого уровня, как показывает надстрочная черта на рис. 17-2, и отключается при подключении вывода сброса к VCC. Напряжение на управляющем выводе *Control* позволяет изменять опорные напряжения компаратора напряжения, равные 2/3 VCC и 1/3 VCC, на напряжение вывода *Control* и половину напряжения вывода *Control* соответственно. Вывод *Control* отключается при подключении его к GND через конденсатор 10 нФ (0,01 мкФ) для устранения шума.

CMOS-версия таймера 555, такая как TLC555, включает в себя МОП-транзисторы вместо биполярных, входящих в стандартный таймер 555, например NE555. Микросхема CMOS-таймера 555 требует меньшей мощности и не имеет выбросов напряжения при изменении состояния выходных выводов, но допускает меньший выходной ток, чем стандартная микросхема таймера 555. Вывод VCC микросхемы таймера CMOS 555 поддерживает напряжение питания 2–15 В, а выходной вывод принимает втекающий ток до 100 мА, но выдать может только 10 мА, в то время как стандартный таймер 555 поддерживает напряжение питания 5–15 В[94], а выходной вывод при-

[93] Как следует из раздела «Бистабильный режим» далее, автор под ним имеет в виду использование таймера 555 в качестве RS-триггера – простейшей запоминающей ячейки. В литературе вы встретите упоминание о двух основных режимах таймера 555: в качестве собственно таймера (то, что автор называет моностабильным режимом; обычно называется моностабильный мультивибратор, или, проще, *одновибратор*) и в качестве генератора импульсов (то, что автор называет нестабильным режимом, часто именуется просто режим *мультивибратора*). В дальнейшем *нестабильный режим* (*astable mode*) мы переводим как *режим генерации* во избежание разночтений в терминологии. – *Прим. перев.*

[94] Уточним, что упомянутая автором модель NE555 в классическом варианте поддерживает питание от 4,5 до 18 В. – *Прим. перев.*

нимает или выдает ток до 200 мА. Выбросы напряжения, возникающие при изменении состояний выходных выводов стандартной микросхемы таймера 555, устраняются установкой конденсатора емкостью 0,1 мкФ на выводах VCC и GND микросхемы.

В этой главе подразумевается CMOS-вариант таймера 555, за исключением раздела «Преобразование прямоугольного колебания в синусоидальное», в котором используется стандартная микросхема таймера 555.

Моностабильный режим

В моностабильном режиме для микросхемы таймера 555 (см. рис. 17-3 и подключения в табл. 17-1) замыкание кнопки приводит к включению светодиода в течение длительности импульса. Когда кнопка разомкнута, вывод *Trigger* соединен с VCC через подтягивающий резистор R_2, а вывод *Threshold* соединен с GND через вывод *Discharge*, так что вывод *Output* имеет низкий уровень, и светодиод не горит. Когда кнопка на мгновение замыкается, вывод *Trigger* подключается к GND, и напряжение на нем составляет 0 В, что меньше, чем 1/3*VCC*. Так что триггер устанавливается, выходной вывод переключается с низкого на высокий уровень, светодиод включается, а разрядный NPN-транзистор выключается, чтобы отсоединить вывод *Discharge* от GND.

Конденсатор C_1 начинает заряжаться через резистор R_1, так как конденсатор больше не подключен к GND, и напряжение на выводе *Threshold* увеличивается. Когда напряжение на пороговом выводе превысит 2/3 VCC, триггер сбрасывается, вывод *Output* переключается с высокого на низкий уровень, светодиод выключается, а разрядный транзистор NPN включается, замыкая вывод *Discharge* и GND. Конденсатор быстро разряжается через вывод *Discharge*, вывод *Threshold* снова подключается к GND, а вывод *Trigger* – к VCC. Цикл повторяется только при следующем замыкании кнопки.

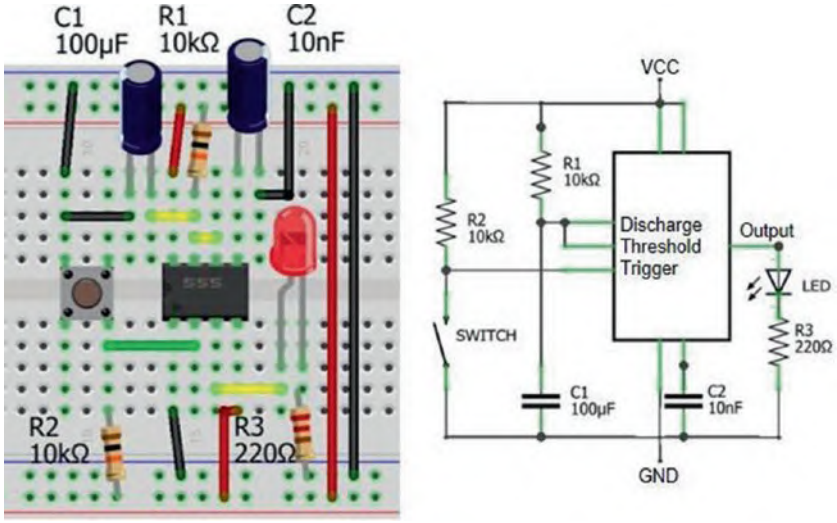


Рисунок 17-3. Моностабильный режим таймера 555

Таблица 17-1 . Моностабильный режим таймера 555

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	Кнопка (правый вывод)	
555 Output (вывод 3)	LED длинный вывод	
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	10 нФ конденсатор[95]	GND (второй вывод конденсатора 10 нФ)
555 Threshold (вывод 6)	555 Discharge (вывод 7)	
555 Discharge (вывод 7)	Резистор 10 кОм (R_1)	VCC (второй вывод резистора R_1)
555 Discharge (вывод 7)	Конденсатор 100 мкФ, положительный вывод	GND (отрицательный вывод конденсатора 100 мкФ)
555 VCC (вывод 8)	VCC	
LED короткий вывод	Резистор 220 Ом (R_3)	GND (второй вывод резистора R_3)
Кнопка (правый вывод)	Резистор 10 кОм (R_2)	VCC (второй вывод резистора R_2)
Кнопка (левый вывод)	GND	

Время, в течение которого конденсатор заряжается от 0 В до 2/3 VCC, зависит от значений в паре резистор–конденсатор R_1 и C_1, рис. 17-3. Решая уравнение $⅔VCC = VCC\ (1 - e^{t/RC})$ относительно времени *t*, получаем $t = \ln(3) \times RC$, или приблизительно $1{,}1 \times RC$. Например, фактическое время 1246 мс, в течение которого вывод *Output* остается в высоком уровне после замыкания кнопки, при сопротивлении резистора R_1 9,97 кОм и емкости конденсатора 110 мкФ, было сопоставимо с расчетным временем 1205 мс (см. рис. 17-4).

Рисунок 17-4. Моностабильный сигнал таймера 555

Сопротивление фоторезистора (LDR) уменьшается с увеличением падающего света. Можно сконструировать ночник, если заменить кнопку и подтягивающий резистор R_2 на рис. 17-3 на заземляющий резистор и LDR соответственно. Сопротивление LDR составляет от 3 кОм до 5 кОм при среднем уровне дневного света, поэтому резистор 4,7 кОм обеспечивает сбалансированное сопротивление для делителя напряжения с LDR. По мере уменьшения освещенности сопротивление LDR увеличивается, что уменьшает выходное напряжение делителя напряжения. Вывод триггера подключен к выходу этого делителя напряжения, т. е. напряжение на выводе *Trigger* уменьшается; и когда

[95] У автора на рис. 17-3 ошибочно показан конденсатор 10 нФ как электролитический полярный. Электролитические конденсаторы такой малой емкости (0,01 мкФ) не производятся, обычно употребляется неполярный керамический тип, поэтому безразлично, какой из выводов куда подключать. – *Прим. перев.*

оно становится ниже 1/3 VCC, триггер срабатывает, выходной вывод переключается от низкого к высокому уровню, и зажигается светодиод ночника[96].

Бистабильный режим

В бистабильном режиме для микросхемы таймера 555 (см. рис. 17-5 и подключения в табл. 17-2) нажатие кнопок *ON* или *OFF* приводит к включению или выключению светодиода. Когда контакты кнопок *ON* или *OFF* разомкнуты, выводы *Trigger* и *Reset* подключены к VCC через подтягивающие резисторы R_1 и R_2, поэтому вывод *Output* находится в низком уровне, а светодиод выключен. Обратите внимание, что вывод *Reset* имеет активный низкий уровень. Когда кнопка *ON* на мгновение замыкается, вывод *Trigger* подключается к GND, а напряжение на нем составляет 0 В, что меньше 1/3VCC, так что триггер срабатывает, вывод *Output* переключается с низкого на высокий уровень, и индикатор включается. Когда нажимается кнопка *OFF*, таймер 555 сбрасывается, вывод *Output* сбрасывается, и светодиод выключается. Микросхема таймера 555 в бистабильном режиме предохраняет от дребезга кнопок *ON* и *OFF*.

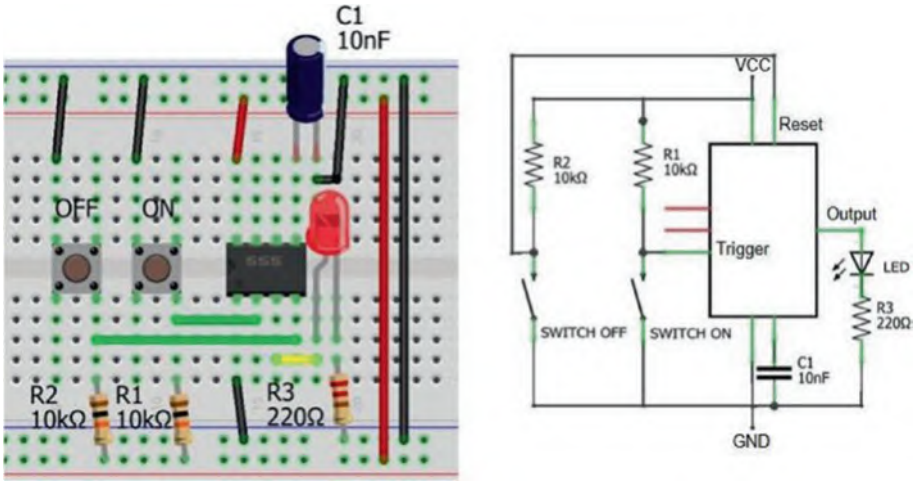


Рисунок 17-5. Бистабильный режим таймера 555

[96] Отметим, что одновибратор на основе таймера 555 обладает той особенностью, что наличие низкого уровня на входе запуска сверх заданной длительности импульса фиксирует состояние выхода в высоком уровне до тех пор, пока не сменится уровень на входе. Эта особенность отличает его от других конструкций одновибраторов, реагирующих обычно не на уровень, а на перепад на входе. Фактически автор описывает режим обычного компаратора с гистерезисом, с единственной положительной особенностью в сравнении с более простыми схемами – никакой дребезг на входе в момент переключения не отразится на состоянии выхода, будучи изолирован двойным барьером: гистерезисом за счет большой разности порогов переключения (1/3 от VCC) и временем до истечения заданного интервала таймера, в течение которого таймер не реагирует на входные импульсы. А это как раз важно для датчиков освещенности, которая меняется крайне медленно в сравнении с быстродействием электронных схем, и поэтому вблизи порога переключения неизбежен длительный дребезг выходного сигнала датчика. – *Прим. перев.*

Таблица 17-2. Бистабильный режим таймера 555

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	Кнопка ON (правый вывод)	
555 Output (вывод 3)	LED длинный вывод	
555 Reset (вывод 4)	Кнопка OFF (правый вывод)	
555 Control (вывод 5)	Конденсатор 10 нФ[97]	GND (второй вывод 10 нФ)
555 VCC (вывод 8)	VCC	
LED короткий вывод	Резистор 220 Ом	GND (второй вывод резистора)
Кнопки ON и OFF (правые выводы)	Резисторы 10 кОм	VCC (вторые выводы резисторов)
Кнопки ON и OFF (левые выводы)	GND	

Режим генерации

В режиме генерации микросхема таймера 555 (см. рис. 17-6 и подключения в табл. 17-3) генерирует прямоугольное колебание, что иллюстрируется двумя светодиодами, включающимися и выключающимися попеременно. Коэффициент заполнения периода и частота, которая является обратной величиной по отношению к длительности периода колебаний, определяются значениями конденсатора C_1 и двух резисторов R_1 и R_2. Светодиод, подключенный между выводом *Output* и GND, потребляет вытекающий ток с выходного вывода, в то время как светодиод, подключенный между VCC и выводом *Output*, подает втекающий ток в выходной вывод. Когда выходной вывод *Output* находится на высоком уровне, потребляющий светодиод включен, а подающий выключен. И наоборот, когда выходной вывод на низком уровне, потребляющий светодиод отключается, а подающий включается.

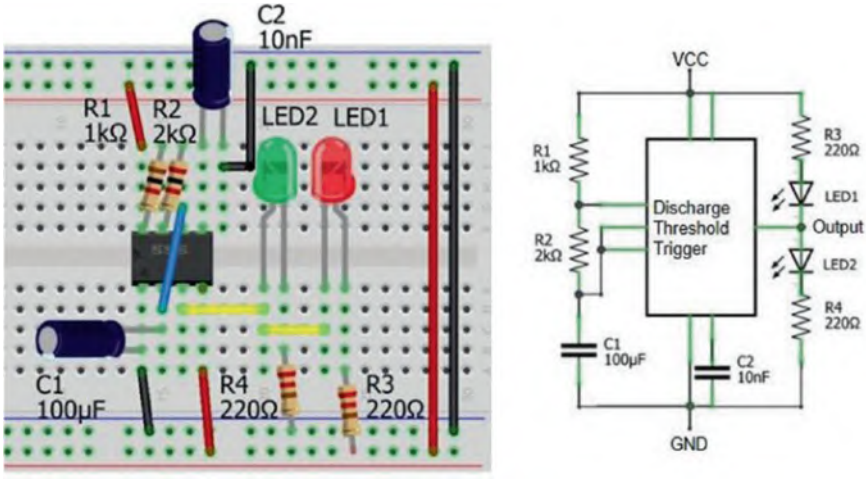


Рисунок 17-6. Режим генерации таймера 555

[97] См. сноску _ на стр. 472. – *Прим. перев.*

Таблица 17-3. Режим генерации таймера 555

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	LED2 длинный вывод	LED1 короткий вывод
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[98]	GND (второй вывод 10 нФ)
555 Threshold (вывод 6)	Резистор R_2 2 кОм	555 Discharge (вывод 7) (второй вывод R_2)
555 Discharge (вывод 7)	Резистор R_1 1 кОм	VCC (второй вывод R_1)
555 VCC (вывод 8)	VCC	
LED1 длинный вывод	Резистор R_3 220 Ом	VCC (второй вывод R_3)
LED2 короткий вывод	Резистор R_4 220 Ом	GND (второй вывод R_4)
Конденсатор 100 мкФ, положительный вывод	555 Trigger (вывод 2)	
Конденсатор 100 мкФ, отрицательный вывод	GND (555 выв. 1)	

Конденсатор C_1 заряжается через два резистора R_1 и R_2, и напряжение на пороговом выводе *Threshold* увеличивается. Когда напряжение на конденсаторе и пороговом выводе больше, чем 2/3 VCC, триггер сбрасывается, выходной вывод *Output* переключается с высокого на низкий уровень, ток через светодиод LED2 выключается, ток через светодиод LED1 включается, а NPN-транзистор включается для замыкания вывода *Discharge* и GND. Теперь конденсатор разряжается через резистор R_2 и вывод *Discharge*.

Когда напряжение на разрядном конденсаторе и выводе триггера меньше, чем 1/3 VCC, триггер устанавливается, выходной вывод переключается с низкого на высокий уровень, включается ток через светодиод LED_2, ток через светодиод LED1 выключается, и выключается NPN-транзистор, чтобы отключить вывод *Discharge* от GND. Теперь конденсатор опять заряжается через два резистора R_1 и R_2, и цикл повторяется.

Значения сопротивлений резисторов R_1 и R_2 и емкости конденсатора C_1 определяют время, в течение которого конденсатор заряжается с 1/3*VCC* до 2/3*VCC*, но время разряда определяется только резистором R_1 и конденсатором. Время, необходимое для увеличения напряжения конденсатора с 1/3*VCC* до 2/3*VCC*, – это разница во времени между зарядкой от 0 В до 2/3*VCC* и от 0 В до 1/3*VCC*. Напряжение на конденсаторе при *t* секунд зарядки равно $VCC(1 - e^{t/RC})$. Два уравнения $2/3VCC = VCC(1 - e^{t2/RC})$ и $1/3VCC = VCC(1 - e^{t1/RC})$ решаются для t_1 и t_2, двух составляющих времени заряда. Разница между временами заряда, $t_2 - t_1$, составляет $\ln(2) \times (R_1 + R_2) \times C_1$, или $0{,}693 \times (R_1 + R_2) \times C_1$ с секунд. Уравнение для напряжения на конденсаторе при *t* секунд разряда равно $VCC \cdot e^{t/RC}$. Время,

[98] См. сноску 95 на стр. 315. – *Прим. перев.*

в течение которого конденсатор разряжается с 2/3*VCC* до 1/3*VCC*, аналогично вычисляется как ln(2) × R_2C_1 секунд.

Итого продолжительность периода составляет ln(2) × (R_1 + 2R_2) × C_1 секунд, а коэффициент заполнения равен (R_1 + R_2)/(R_1 + 2R_2). Частота колебаний обратно пропорциональна продолжительности периода и обычно записывается как 1,44/[(R_1 + 2R_2) × C_1], так как ln(2) – 1 = 1,44. Например, при значениях сопротивления 986 Ом и 1981 Ом и конденсаторе 110 мкФ фактическое время зарядки и разрядки конденсатора 220 мс и 152 мс соответственно с коэффициентом заполнения 59 %, периодом 372 мс и частотой 2,69 Гц соответствовали расчетным значениям времени зарядки и разрядки 226 мс и 151 мс, коэффициента заполнения 60 %, периода 377 мс и частоты 2,65 Гц (см. рис. 17-7).

Рисунок 17-7. Генерация прямоугольных колебаний таймером 555

При фиксированных значениях для резистора R_1 и конденсатора C значение резистора R_2 выбирается для генерации прямоугольной волны с частотой [ln(2) × (R_1 + 2R_2) × C]$^{-1}$ Гц. Электронное пианино создается путем подключения к микросхеме таймера 555 нескольких пар кнопок и резисторов R_2 при фиксированных значениях для резистора R_1 (1 кОм) и конденсатора (1 мкФ) (см. рис. 17-8 и подключения в табл. 17-5). Нажатие кнопки соединяет ряд резисторов R_2 с выводами *Trigger* и *Discharge*, и конденсатор начинает заряжаться-разряжаться, что создает прямоугольное колебание до тех пор, пока нажата кнопка. Например, для ноты G4 с частотой 392 Гц (см. табл. 17-4 далее) требуется общее сопротивление 1340 Ом. Резисторы R_2 расположены последовательно друг с другом, с общим сопротивлением, равным сумме резисторов. Резисторы R_2 для нот C5, B4 и A4, которые предшествуют резистору для ноты G4, составляют в сумме 1130 Ом, поэтому требуемый резистор R_2 для ноты G4 составляет 210 Ом, а поставлен резистор 200 Ом, так как это легкодоступная величина сопротивления резистора. На практике частоты генерируемых звуков находились в пределах 7 Гц от истинных частот (см. табл. 17-4).

Таблица 17-4. Комбинации резисторов для электронного пианино[99]

Нота	C4	D4	E4	F4	G4	A4	B4	C5
Действительная частота (Гц)	262	294	330	349	392	440	494	523
Резистор (Ом)	330	220	150	220	200	150	100	880
Сумма резисторов (Ом)	2250	1920	1700	1550	1330	1130	980	880
Расчетная частота (Гц)	262	298	328	352	394	443	487	523

[99] Частоты в таблице соответствуют округленным до целых чисел частотам чистых нот от «до» до «си» первой октавы плюс «до» второй октавы. Отметим, что это чисто иллюстративный пример – для более-менее полноценного воспроизведения реальных мелодий необходимо воспроизвести не 7 нот, а все 12, входящих в октаву, и с большей точностью, чем получается в этой схеме. – *Прим. перев.*

Динамик подключается к выходному контакту через конденсатор емкостью 10 мкФ.

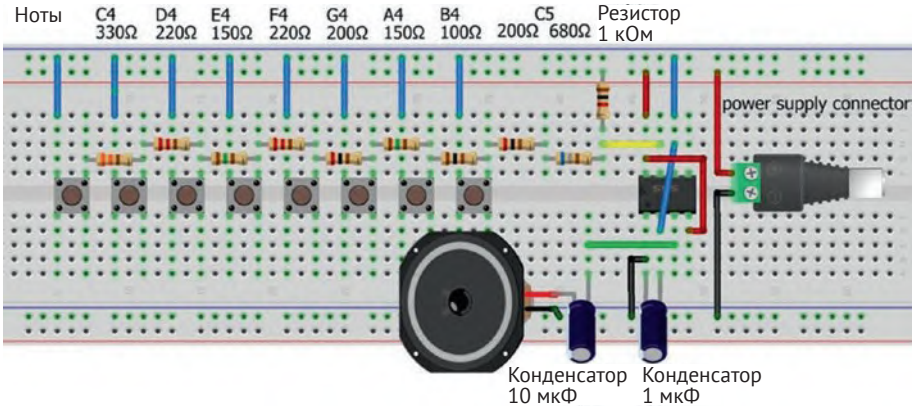


Рисунок 17-8. Таймер 555 и электронное пианино

Таблица 17-5. Таймер 555 и электронное пианино

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	Конденсатор 10 мкФ, положительный вывод	Динамик положительный вывод (отрицательный вывод 10мкФ)
555 Reset (вывод 4)	VCC	
555 Threshold (вывод 6)	Кнопка (левый вывод)	
555 Discharge (вывод 7)	Резисторы	
555 VCC (вывод 8)	VCC	
Конденсатор 1 мкФ[100]	555 Trigger (вывод 2)	
Конденсатор 1 мкФ	GND (555 выв. 1)	
Динамик отрицательный вывод	GND	

Переменный коэффициент заполнения

Коэффициент заполнения периода прямоугольного колебания в режиме генерации всегда больше, чем 50 %, поскольку конденсатор заряжается через резисторы R_1 и R_2, но разряжается только через резистор R_2 (см. рис. 17-6). Если подключить два диода, например IN4001 (см. рис. 17-9 и подключения в табл. 17-6), то конденсатор заряжается только через резистор R_1 и разряжается только через резистор R_2.

[100] На схеме времязадающий конденсатор 1 мкФ обозначен как электролитический полярный. Здесь, конечно, следует установить неполярный керамический (К10-17), который и меньше, и дешевле и лучше работает во времязадающих схемах. – *Прим. перев.*

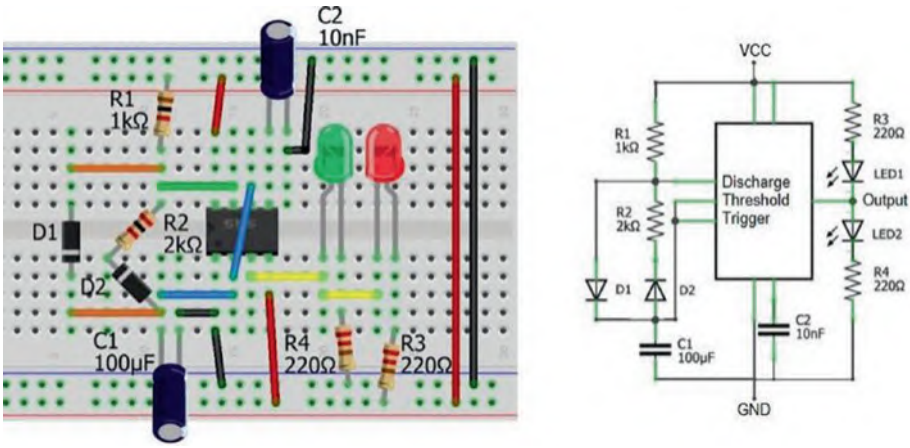


Рисунок 17-9. Режим генерации таймера 555 с полным диапазоном коэффициента заполнения

Таблица 17-6. Режим генерации таймера 555 с полным диапазоном коэффициента заполнения

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Trigger (вывод 2)	D1 катод (отрицательный вывод)	
555 Trigger (вывод 2)	D2 анод (положительный вывод)	Резистор 2 кОм
555 Output (вывод 3)	LED2 длинный вывод	LED1 короткий вывод
555 *R*eset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[101]	GND (второй вывод 10 нФ)
555 Discharge (вывод 7)	Резистор R_1 1 кОм	VCC (второй вывод R_1)
555 VCC (вывод 8)	VCC	
LED1 длинный вывод	Резистор R_3 220 Ом	VCC (второй вывод R_3)
LED2 короткий вывод	Резистор R_4 220 Ом	GND (второй вывод R_4)
Конденсатор 100 мкФ, положительный вывод	555 Trigger (вывод 2)	
Конденсатор 100 мкФ, отрицательный вывод	GND (555 выв. 1)	

Прямое падение напряжения на диодах приводит к увеличению времени заряда и разряда, которое увеличивается с $\ln(2) \times R_1C_1$ и $\ln(2) \times R_2C_1$ с до $\alpha \times R_1C_1$ и $\alpha \times R_2C_1$, при этом константа α равна $\ln\left[1+\frac{VCC}{VCC-3V_d}\right]$, где

[101] См. сноску 95 на стр. 315. – *Прим. перев.*

V_d – прямое падение напряжения на диоде. Например, если *VCC* составляет 5 В, а прямое падение напряжения на диоде равно 0,6 В, то время разряда составляет $\ln(2{,}56) \times R_2C_1$, а не $\ln(2) \times R_2C_1$, как в случае, когда в цепь не включен диод. Коэффициент заполнения прямоугольного колебания не зависит от включения двух диодов и составляет менее или более 50 %, когда резистор R_1 меньше или больше резистора R_2. Например, при значениях сопротивлений 986 Ом и 1981 Ом и конденсаторе 110 мкФ (расчетное время зарядки и разрядки конденсатора 102 мс и 205 мс) фактическое время зарядки и разрядки конденсатора составило 99 мс и 203 мс, что соответствует коэффициенту заполнения 33 % (см. рис. 17-10).

Рисунок 17-10. Переменный рабочий цикл в режиме генерации

Напряжение на конденсаторе, заряжающемся или разряжающемся через резистор R, равно $Vt = V_S + (V_0 - V_S)e^{-t/RC}$, где V_0 – начальное напряжение на конденсаторе с емкостью *C*, а V_S – напряжение питания (конечное напряжение заряда). Если конденсатор полностью разряжен при $V_0 = 0$, а конечное напряжение обозначить через *V*, то уравнение сводится к обычной формуле для заряда конденсатора *V*. Аналогично для разряда конденсатора при $V_0 = V$ и конечном напряжении $V_s = 0$ разряд сводится к обычной формуле $V(e^{t/RC})$.

Когда диод с прямым напряжением V_d включен последовательно с зарядным резистором R_1, а конденсатор заряжается от 1/3VCC до 2/3VCC, время зарядки составляет $R_1C_1 \times \ln\left[1 + \frac{VCC}{VCC - 3V_d}\right]$ для конечного напряжения $V_S = VCC - V_d$, $V_0 = 1/3VCC$ и $V_t = 2/3VCC$. Аналогично, когда диод включен последовательно с разрядным резистором R_2, а конденсатор разряжается с $2/3VCC$ до $1/3VCC$, время разрядки равно $R_2C_1 \times \ln\left[1 + \frac{VCC}{VCC - 3V_d}\right]$ для конечного напряжения $V_S = V_d$, $V_0 = 2/3VCC$, а $V_t = 1/3VCC$. Заметьте, что когда в формулах прямое напряжение диода V_d устанавливается равным нулю, время зарядки и разрядки равно $\ln(2) \times R_1C_1$ и $\ln(2) \times R_2C_1$, как и раньше.

50%-ный коэффициент заполнения

Если конденсатор заряжается и разряжается через один и тот же резистор, то рабочий цикл близок к 50 %, независимо от значения резистора или конденсатора. Резистор подключен между выходным выводом *Output* и выводами *Trigger* и *Threshold* (см. рис. 17-11 и подключения в табл. 17-7).

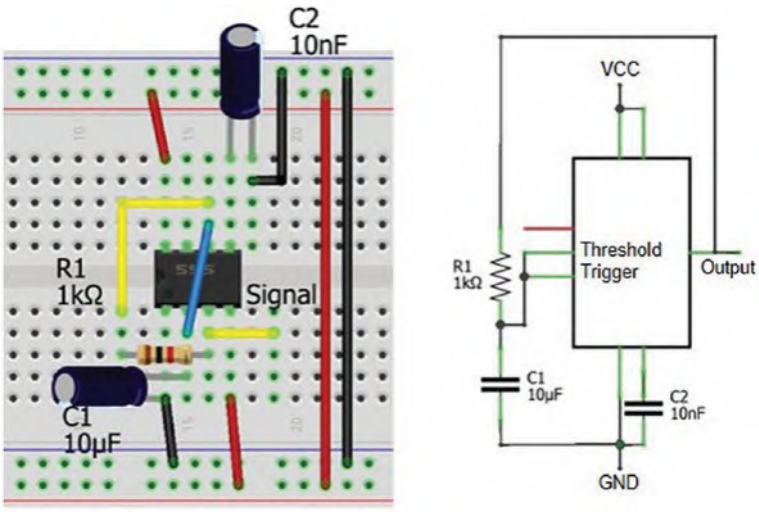


Рисунок 17-11. Режим генерации таймера 555 с коэффициентом заполнения 50 %

Таблица 17-7. Режим генерации таймера 555 с коэффициентом заполнения 50 % (см. также рис. 17-12 далее)

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	Выход сигнала	
555 Output (вывод 3)	Резистор R_1 1 кОм	555 Threshold (вывод 6) (второй вывод R_1)
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[102]	GND (второй вывод 10 нФ)
555 VCC (вывод 8)	VCC	
Конденсатор 10 мкФ, положительный вывод	555 Trigger (вывод 2)	
Конденсатор 10 мкФ, отрицательный вывод	GND (555 выв. 1)	
Коллектор NPN-транзистора	Резистор R_3 220 Ом	LED короткий вывод (второй вывод R_3)
База NPN-транзистора	Резистор R_2 10 кОм	555 Output (вывод 3) (второй вывод R_2)
Эмиттер NPN-транзистора	GND	
LED длинный вывод	VCC	

[102] См. сноску 95 на стр. 315. – *Прим. перев.*

Конденсатор заряжается через резистор R_1, и напряжение на выводе *Threshold* увеличивается. Когда напряжение на конденсаторе и выводе *Threshold* становится больше 2/3 VCC, триггер сбрасывается, выходной вывод переключается с высокого на низкий уровень. Теперь конденсатор разряжается через резистор R_1 и выходной вывод *Output*. Когда напряжение на разряжающемся конденсаторе и выводе Trigger становится меньше, чем 1/3 VCC, триггер устанавливается, выходной вывод переключается с низкого на высокий уровень. Теперь конденсатор заряжается через резистор R_1, и цикл повторяется.

Длительность периода колебания составляет $\ln(2) \times 2R_1C_1$ секунд. Например, при значениях резистора и конденсатора 986 Ом и 110 мкФ частота прямоугольной волны 6,55 Гц была близка к расчетной частоте 6,65 Гц (см. рис. 17-12).

Рисунок 17-12. Прямоугольное колебание с коэффициентом заполнения 50 %

Когда к выходному выводу подключается нагрузка, даже просто светодиод, то период импульса прямоугольной формы увеличивается, коэффициент заполнения увеличивается, а частота уменьшается. Если требуется управлять нагрузкой, то к выходному выводу следует подключить базу транзистора, при этом нагрузка будет питаться через транзистор (см. рис. 17-13 и подключения в табл. 17-7). Подойдут транзисторы BC548 или 2N2222, с резистором 10 кОм между выводом *Output* и базой транзистора, служащим для ограничения тока через базу.

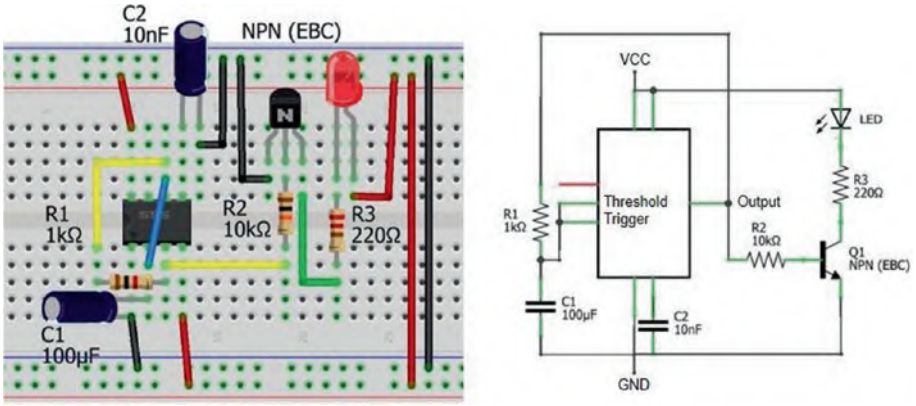


Рисунок 17-13. Режим генератора таймера 555 с 50%-ным коэффициентом заполнения и нагрузкой

Частоту колебаний можно регулировать при подключении переменного резистора последовательно с резистором R_1. Наличие постоянного резистора 1 кОм поддерживает временную задержку для разрядного конденсатора, когда переменный резистор выкручен до конца на нулевое сопротивление[103].

[103] Номинал в 1 кОм – минимальное рекомендуемое значение зарядных/разрядных сопротивлений для таймера 555. Если необходимо увеличить частоту, то это мож-

Режим ШИМ

Широтно-импульсная модуляция (ШИМ) прямоугольного колебания управляется заменой резистора R_1 на рис. 17-13 потенциометром сопротивлением P и двумя диодами, такими как IN4001 (см. рис. 17-14 и подключения в табл. 17-8). Конденсатор заряжается и разряжается через сопротивления каждой из сторон потенциометра, $\beta \times P$ и $(1 - \beta) \times P$, так что сумма времени зарядки и разрядки постоянна. Включение диодов увеличивает продолжительность периода с $\ln(2) \times 2R_1C_1$ с до $\ln\left[1+\frac{VCC}{VCC-3V_d}\right] \times PC_1$ секунд, как описано в разделе «Переменный коэффициент заполнения» этой главы. Например, потенциометр 10 кОм и конденсатор 100 мкФ обеспечивают частоту прямоугольного колебания 1,06 Гц с коэффициентом заполнения от 10 % до 90 %.

Для управления нагрузкой с помощью ШИМ выходной вывод подключается к базе транзистора, например 2N2222 (см. рис. 17-14), через который питается нагрузка. Скажем, сервопривод может управляться с помощью микросхемы таймера 555 при подключении вывода *Output* непосредственно к сигнальному выводу серводвигателя. Сервопривод вращается на углы 0° и 180° при подаче прямоугольного колебания частотой 50 Гц с длительностью импульсов 0,5 мс и 2,5 мс. Прямоугольное колебание частотой 50,6 Гц генерируется микросхемой таймера 555 с потенциометром 10 кОм и конденсатором 2,1 мкФ[104] (C_1).

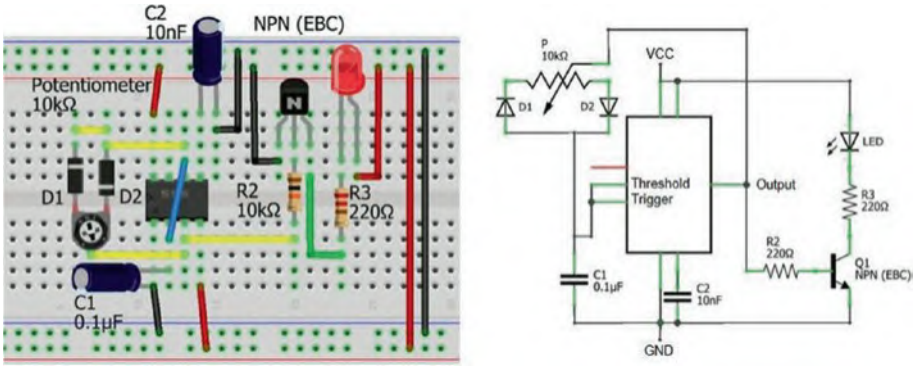


Рисунок 17-14. Таймер 555 в режиме генерации с ШИМ

но сделать снижением емкости конденсатора C_1 (минимально допустимое значение 1 нФ, что вместе с сопротивлением 1 кОм соответствует частоте около 500–700 кГц в зависимости от схемы включения). – *Прим. перев.*

[104] Конденсатор емкостью 2,1 мкФ (соответствующий ряду номиналов E96 с разбросом 1 %) теоретически должен существовать, но в продаже вы его не встретите. Поэтому получить такую емкость можно только отбором из конденсаторов емкостью 2,0 или 2,2 мкФ. – *Прим. перев.*

Таблица 17-8. Таймер 555 в режиме генерации с ШИМ

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	Потенциометр центральный вывод	
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[105]	GND (второй вывод 10 нФ)
555 Threshold (вывод 6)	D1 катод (отрицательный вывод)	Потенциометр (D1 анод)
555 Threshold (вывод 6)	D2 анод (положительный вывод)	Потенциометр (D2 катод)
555 VCC (вывод 8)	VCC	
Коллектор транзистора	Резистор R_3 220 Ом	LED короткий вывод
База транзистора	Резистор R_2 10 кОм	555 Output (вывод 3) (второй вывод R_2)
Эмиттер транзистора	GND	
LED длинный вывод	VCC	
Конденсатор 0,1 мкФ[106]	555 Trigger (вывод 2)	
Конденсатор 0,1 мкФ	GND (555 выв. 1)	

Функциональный генератор

Прямоугольное колебание представляет собой несинусоидальное периодическое колебание с мгновенными изменениями от минимального до максимального значения, которое находит применение в коммутации и обработке сигналов. Напротив, синусоидальное колебание описывается как плавное периодическое колебание, которое находит применение в механических, электрических и звуковых приложениях. Например, звук может быть представлен как комбинация синусоидальных волн с различными частотами и амплитудами. Микросхема таймера 555 генерирует колебания прямоугольной формы, однако существует несколько приложений, требующих генерации колебаний синусоидальной формы.

Прямоугольное колебание с частотой *f* может быть представлено в виде ряда Фурье – суммы синусоид за время *t*:

$$x(t) = \frac{4}{\pi}\left\{ sin(2\pi ft) + sin(3\times 2\pi ft) + \frac{1}{5} sin(5\times 2\pi ft) + ... \right\},$$

представляющих собой ряд Фурье, состоящий из нечетных целых гармоник. Рисунок 17-15 иллюстрирует синусоидальную волну с частотой 2 Гц и аппрок-

[105] См. сноску 95 на стр. 315. – *Прим. перев.*

[106] На схеме конденсатор 0,1 мкФ обозначен как электролитический полярный. Электролитические конденсаторы такой малой емкости не применяются, здесь следует установить неполярный керамический (К10-17). – *Прим. перев.*

симацию прямоугольного колебания в виде ряда Фурье путем суммирования первых десяти нечетных гармоник.

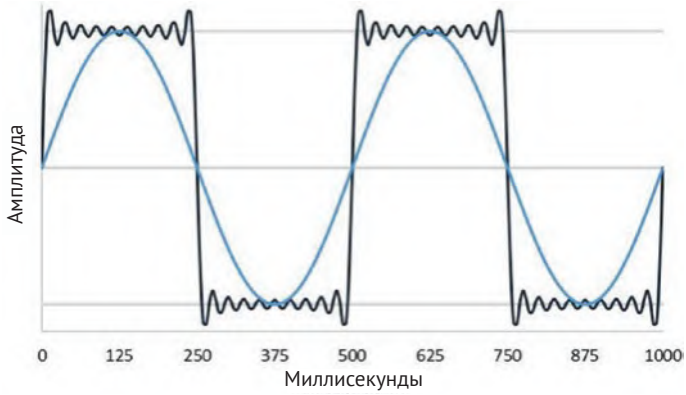


Рисунок 17-15. Синусоида и сумма нечетных гармоник

Базовая синусоида прямоугольного колебания получается путем фильтрации высших гармоник с помощью фильтра нижних частот. Фильтр нижних частот позволяет низкочастотным сигналам проходить через фильтр, в то время как фильтр высоких частот ослабляет низкочастотные сигналы. Резисторно-конденсаторная схема делителя напряжения образует RC-фильтр нижних частот, при этом конденсатор закорачивает высокие частоты на землю, а более низкие частоты доступны на выходе *VOUT* (см. рис. 17-16). Частота среза фильтра нижних частот составляет $(2\pi RC)^{-1}$ Гц.

Рисунок 17-16. Делитель напряжения и резисторно-конденсаторный фильтр нижних частот

Значение резистора *RC*-фильтра нижних частот выбирается на основе нагрузки фильтра, а затем подсчитывается значение конденсатора. Например, прямоугольный сигнал с частотой 153 Гц и коэффициентом заполнения 50 % генерируется микросхемой таймера 555 с резистором 4,7 кОм и конденсатором емкостью 1 мкФ. Для получения базовой синусоиды с *RC*-фильтром нижних частот значение сопротивления для *RC*-фильтра выбирается равным 1 кОм; а для получения частоты среза *fcut*, равной 153 Гц, требуемое значение конденсатора составит 1,04 мкФ, что следует из формулы $(2\pi Rfcut)^{-1}$ Ф. С резистором 1 кОм и конденсатором 1 мкФ фактическая частота среза *RC*-фильтра нижних частот составит 159 Гц. *RC*-фильтр нижних частот показан на рис. 17-17, подключения – в табл. 17-9.

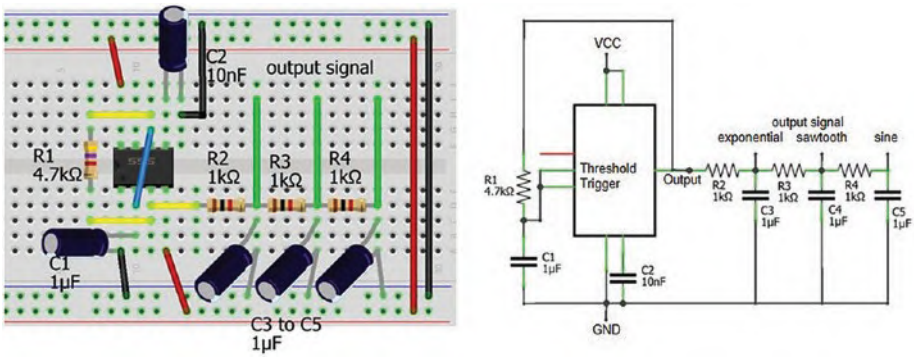


Рисунок 17-17. Микросхема таймера 555 с RC-фильтрами нижних частот

Таблица 17-9. Микросхема таймера 555 с RC-фильтрами нижних частот

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	Резистор 4,7 кОм R_1	555 Threshold (вывод 6) (второй вывод резистора R_1)
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[107]	GND (второй вывод 10 нФ)
555 VCC (вывод 8)	VCC	
Резистор 1 кОм R_2	555 Output (вывод 3)	Экспоненциальный сигнал
Резистор 1 кОм R_3 и R_4	Резистор 1 кОм R_2 и R_3	Пилообразный и синусоидальный сигнал
3 конденсатора 1 мкФ[108] верхние выводы		
3 конденсатора 1 мкФ нижние выводы		
Конденсатор 1 мкФ C_1	555 Trigger (вывод 2)	
Конденсатор 1 мкФ C_1	GND (555 выв. 1)	

Выходной сигнал от *RC*-фильтра нижних частот имеет ту же форму, что и для зарядного и разрядного конденсатора (см. рис. 17-18, верхний график). Если выходной сигнал *RC*-фильтра нижних частот является входом для второго такого же фильтра с теми же значениями сопротивления и конденсатора 1 кОм и 1 мкФ, то выходной сигнал второго фильтра представляет собой колебания треугольной формы (см. рис. 17-18, средний график). Добавление третьего фильтра нижних частот преобразует треугольное колебание в синусоиду с той же частотой, что и исходное прямоугольное колебание (см. рис. 17-18, нижний

[107] См. сноску 95 на стр. 315. – *Прим. перев.*

[108] См. сноску 100 на стр. 320. – *Прим. перев.*

график). Микросхема таймера 555 и три последовательных *RC*-фильтра нижних частот образуют функциональный генератор для получения колебаний прямоугольной, треугольной и синусоидальной форм.

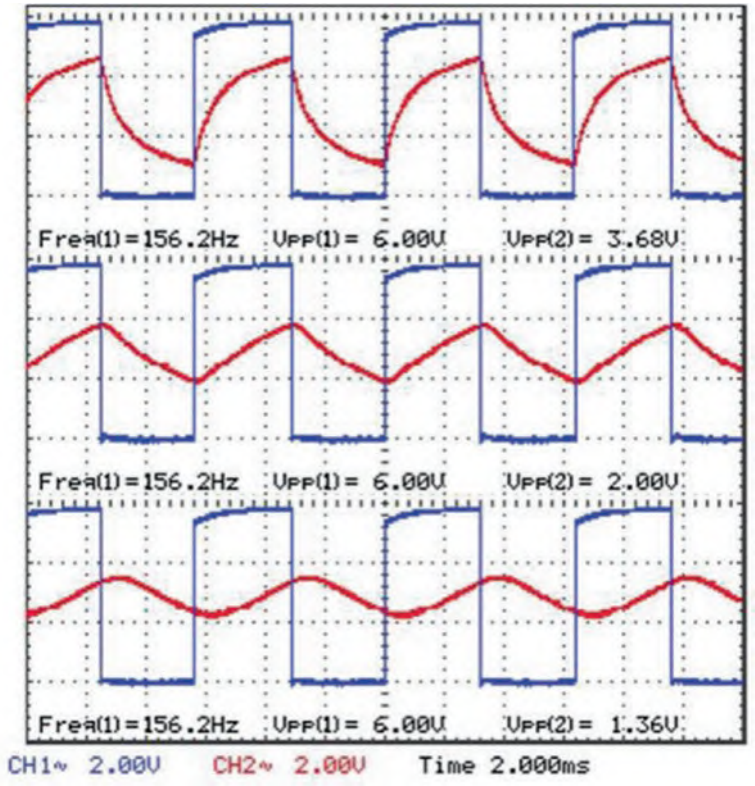


Рисунок 17-18. Выходные сигналы от RC-фильтров нижних частот

На рис. 17-18 показаны выходные сигналы от трех *RC*-фильтров нижних частот относительно прямоугольного сигнала, генерируемого той же микросхемой таймера 555. Три колебания имеют требуемую форму сигнала, но амплитуды сигнала уменьшаются с каждым *RC*-фильтром, что в некоторой степени ожидаемо. Один *RC*-фильтр нижних частот имеет частоту среза $(2\pi RC)^{-1}$ Гц, а конденсатор имеет реактивное сопротивление X_c, равное $(2\pi Cf)^{-1}$ Ом, что является сопротивлением конденсатора для переменного напряжения с частотой *f*. Если синусоида с амплитудой *V* В подается на *RC*-фильтр нижних частот, то выходная волна амплитуды уменьшается до $V \times X_C / \sqrt{R^2 + X_C^2}$ В. Когда значение конденсатора равно $\sqrt{2}Rfcut)^{-1}$ Ф, то реактивное сопротивление конденсатора равно сопротивлению резистора *R*, и уравнение для амплитуды выходной кривой упрощается до $V/\sqrt{2}$. Амплитуда синусоидальных сигналов на выходе первого, второго и третьего *RC*-фильтров нижних частот составляет $V/\sqrt{2}$, $V/2$ и $V/2\sqrt{2}$ В соответственно. Амплитуды сигналов от трех *RC*-фильтров нижних частот, учитывая начальный прямоугольный сигнал, также систематически снижаются.

Преобразование прямоугольного колебания в синусоидальное

Индуктивно-конденсаторный фильтр (*LC*) может служить заменой каскада *RC*-фильтров нижних частот для преобразования прямоугольного колебания в синусоиду. *LC*-фильтр и его схема описаны также в главе 18 («Электрические измерения»). Используя тот же принцип делителя напряжения, *LC*-цепочка также является фильтром нижних частот, поскольку катушка индуктивности блокирует высокие частоты, а конденсатор – низкие, что при соответствующем включении приводит к пропусканию низких частот на выход (см. рис. 17-19).

Рисунок 17-19. Делитель напряжения и индуктивно-конденсаторный фильтр нижних частот

LC-контур имеет частоту резонанса ($2\pi\sqrt{LC}$) Гц. Если микросхема таймера 555 генерирует прямоугольное колебание с коэффициентом заполнения 50 % на резонансной частоте *LC*-контура, то его выход представляет собой синусоидальную кривую (см. рис. 17-21). Например, *LC*-цепочка с индуктивностью 470 мкГн и конденсатором 1 мкФ имеет резонансную частоту 7,3 кГц, а микросхема таймера 555 с резистором 1 кОм и конденсатором емкостью 0,1 мкФ генерирует прямоугольное колебание с коэффициентом заполнения 50 % и частотой 7,2 кГц. Для генерации прямоугольного колебания здесь используется стандартная микросхема таймера 555, такая как NE555. Выбросы напряжения, возникающие у стандартной микросхемы таймера 555 при изменении состояний выходных выводов, устраняются путем установки конденсатора емкостью 0,1 мкФ по выводам VCC и GND микросхемы (см. рис. 17-20 и подключения в табл. 17-10).

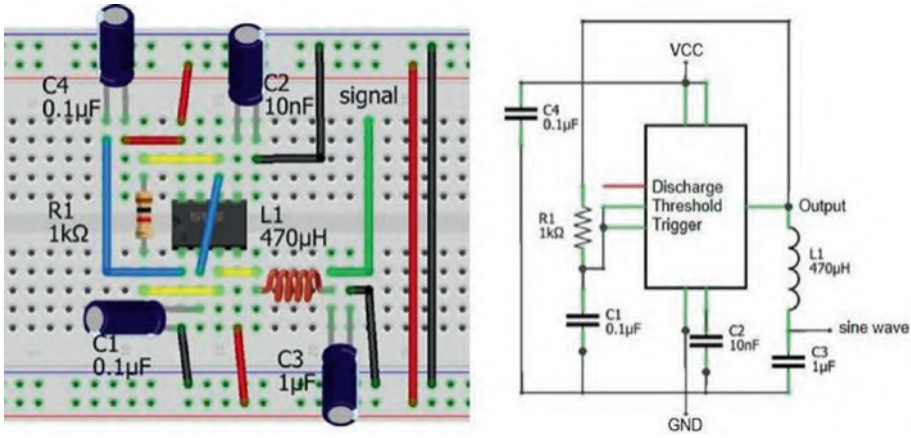


Рисунок 17-20. Таймер 555 и LC-фильтр нижних частот

Таблица 17-10. Таймер 555 и LC-фильтр нижних частот

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	555 Threshold (вывод 6)	
555 Output (вывод 3)	Резистор 1 кОм R_1	555 Threshold (вывод 6) (второй вывод резистора R_1)
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[109]	GND (второй вывод 10 нФ)
555 VCC (вывод 8)	VCC	
Резистор 1 кОм R_2	555 Output (вывод 3)	Экспоненциальный сигнал
Резистор 1 кОм R_3 и R_4	Резистор 1 кОм R_2 и R_3	Пилообразный и синусоидальный сигнал
Индуктивность 470 мкГн левый вывод	555 Output (вывод 3)	
Индуктивность 470 мкГн правый вывод	Выход синуса	Конденсатор 1 мкФ (C_3) левый вывод
Конденсатор 1 мкФ (C_3)[110] правый вывод	GND	
Конденсатор 0,1 мкФ (C_4) правый вывод	VCC	
Конденсатор 0,1 мкФ (C_1) верхний вывод	555 Trigger (вывод 2)	
Конденсатор 0,1 мкФ (C_1) нижний вывод	GND	

На практике коэффициент заполнения и частота прямоугольного колебания, генерируемого стандартной микросхемой таймера 555, работающей при питании 6 В с резистором 1 кОм и конденсатором 0,1 мкФ, составляли 66,7 % и 5,6 кГц. Результирующая синусоида, создаваемая *LC*-цепочкой, имела частоту 5,0 кГц (см. рис. 17-21).

[109] См. сноску 95 на стр. 315. – *Прим. перев.*

[110] Относительно типов конденсаторов C1, C3 и C4 см. сноску 100 на стр. 320. – *Прим. перев.*

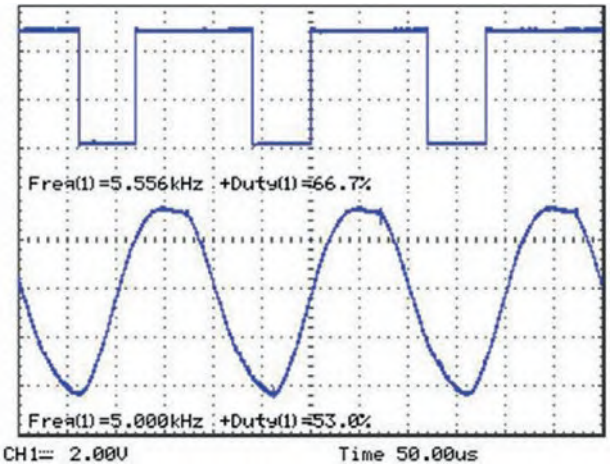
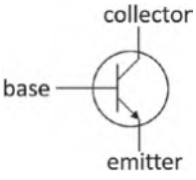


Рисунок 17-21. Прямоугольное колебание от микросхемы таймера 555 и синусоида после LC-фильтра

Биполярный транзистор в качестве ключа



В разделах главы «50%-ный коэффициент заполнения» и «Режим ШИМ» биполярный NPN-транзистор (BJT) функционировал как ключ для включения или выключения питания нагрузки, представлявшей собой светодиод. Когда состояние выходного вывода микросхемы таймера 555 имело высокий уровень, транзистор подключал нагрузку к питанию; и наоборот, питание нагрузки отключалось, когда состояние выходного вывода имело низкий уровень. Подробное описание биполярного транзистора выходит за рамки этой главы, но далее описаны некоторые практические аспекты использования его в качестве ключа-коммутатора.

Биполярный транзистор имеет три вывода: эмиттер (*emitter* E), база (*base* B) и коллектор (*collector* C). Расположение выводов зависит от конкретной модели, иногда с различиями в расположении даже между производителями одной и той же модели. Разводка выводов доступна в спецификации производителя или может определяться с помощью мультиметра, при подключении положительного провода мультиметра к одному контакту и измерении напряжения на других выводах транзистора. Если на обоих других выводах обнаруживается напряжение относительно вывода, подключенного к положительному выводу мультиметра, значит, этот вывод мультиметра подключен к базовому выводу NPN-транзистора[111]. Отличие выводов эмиттера и коллектора заключается в том, что вывод эмиттера имеет несколько более высокое падение напряжения, чем вывод коллектора.

[111] В реальности это делается «прозвонкой» выводов транзистора: база должна «звониться» с обоими другими выводами (в режиме диода); если при этом к ней подключен вывод «плюс» мультиметра, то, значит, перед нами транзистор *NPN*-типа. – *Прим. перев.*

Когда NPN-транзистор функционирует как усилитель и на базовый вывод подается небольшой ток, то между выводами коллектора и эмиттера протекает ток большей величины. Отношение тока между выводами коллектора и эмиттера к величине тока через базу называется коэффициентом усиления по постоянному току и обозначается β или *hFE*[112] в справочных данных транзистора. Изменение базового тока на величину Δ отражается изменением тока коллектор–эмиттер на величину β × Δ. Коэффициент усиления транзистора определяется из спецификации производителя или путем измерения тока на выводах базы и коллектора. Например, для транзистора 2N2222 на рис. 17-22 (слева) токи на выводах базы и коллектора составляли 255 мкА и 25,4 мА соответственно, что дает коэффициент усиления, равный 100.

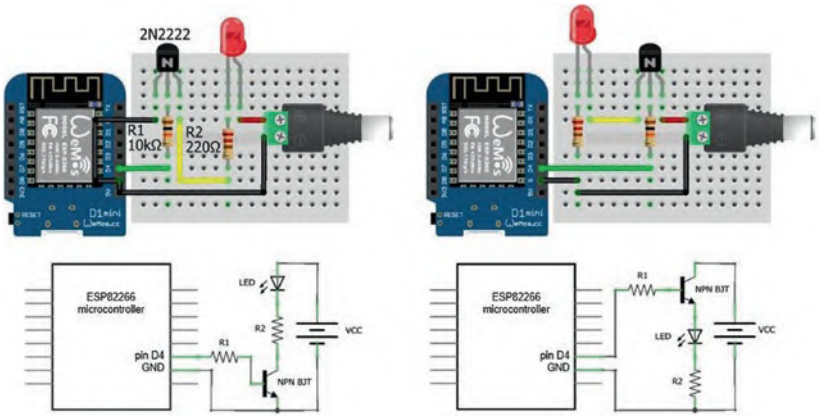


Рисунок 17-22. Биполярный транзистор в качестве ключа со стороны GND (слева) или напряжения питания (справа)

В отличие от усилительного режима, в ключевом режиме транзистора базовый ток устанавливается достаточно высоким, чтобы его увеличение больше не приводило к увеличению тока от коллектора к эмиттеру, поскольку транзистор оказывается в состоянии насыщения. В режиме насыщения ток от коллектора к эмиттеру может либо включаться, либо выключаться.

NPN-транзистор в ключевом режиме, эмиттер которого подключается к GND, назовем ключом нижнего уровня (см. рис. 17-22, слева). Чтобы проиллюстрировать работу транзистора в качестве ключа для включения или выключения питания нагрузки, базовый вывод транзистора 2N2222 в корпусе TO-92 подключен к выводу D4 платы ESP8266, вывод коллектора подключен к резистору светодиода, а вывод эмиттера к GND. Когда вывод ESP8266 D4 имеет высокий уровень, ток на базе транзистора позволяет току протекать между коллектором и эмиттером. Ток теперь течет от внешнего источника питания 5 В через светодиод и резистор к транзистору и далее к GND; что зажигает светодиод.

Для примера на рис. 17-22 (слева), когда вывод D4 ESP8266 находится в состоянии высокого уровня, базовый ток транзистора составляет 260 мкА =

[112] Между коэффициентом усиления большого сигнала β и коэффициентом усиления малого сигнала *hFE* ($h_{б9}$ в отечественных обозначениях) имеется разница, но в рассматриваемом случае ключевого режима работы она несущественна. – *Прим. перев.*

(3,3 – 0,7) В / 10 кОм, поскольку транзистор между базой и эмиттером имеет падение напряжения, равное 0,7 В. Учитывая коэффициент усиления, равный 100, ток между коллектором и эмиттером составляет 26 мА = 100 × 260 мкА, что представляет собой ток, проходящий через светодиод. Красный светодиод, используемый в этой главе, имеет прямое падение напряжения 2,12 В при 26 мА, а напряжение на светодиодном резисторе 2,6 В = 26 мА × 100 Ом приводит к напряжению 280 мВ на выводах коллектора и эмиттера.

На схеме рис. 17-22 (справа) NPN-транзистор подключен к VCC, назовем его ключом верхнего уровня. Ставить ключ на стороне питания не рекомендуется, так как ток через светодиод ограничен, даже несмотря на то, что компоненты идентичны в двух схемах на рис. 17-22. На практике базовый ток транзистора 34 мкА приводит к напряжению на базовом резисторе 338 мВ = 34 мкА × 10 кОм. Падения напряжения светодиода и базы–эмиттера на 1,92 В и 0,7 В соответственно оставляют напряжение на светодиодном резисторе 342 мВ, что соответствует току 3,4 мА – одна восьмая от тока через светодиод с помощью транзистора в качестве нижнего ключа. Обратите внимание на более низкое прямое напряжение светодиода при меньшем токе. Существенная разница между NPN-транзистором, функционирующим как ключ со стороны GND или питания, заключается в более низком напряжении (280 мВ относительно 2,74 В) между коллектором и эмиттером при подключении со стороны GND.

Когда транзистор функционирует как усилитель, значение базового резистора определяется из коэффициента усиления β и требуемого тока через нагрузку I_L. Базовый ток равен I_L/β, и после вычитания падения напряжения база–эмиттер, равного 0,7 В, из базового напряжения V_B значение базового резистора составит $(V_B - 0{,}7) \times \beta/I_L$. Например, базовый резистор R_{BASE} у ключа нижнего уровня для питания светодиода 20 мА от источника питания 5 В на рис. 17-22 (слева) равен (3,3 – 0,7) В × 100/0,02 А = 13 кОм. Однако когда транзистор функционирует как ключ, базовый резистор с более низким значением, например вдвое меньшим расчетного, т. е. равный 6,8 кОм, гарантирует, что транзистор находится в режиме насыщения.

Приложение с MP3-плеером и PIR-датчиком

В этой главе продемонстрирована генерация сигнала микросхемой таймера 555 с приложениями для управления переключателями, синхронизацией, генерацией звука, управления нагрузкой с помощью ШИМ, а также использования таймера в качестве функционального генератора прямоугольных, треугольных и синусоидальных колебаний. Таймер 555 также применяется для преобразования сигнала с одного устройства в требуемый формат для другого устройства. В главе 5 («MP3-плеер»), когда пироэлектрический датчик (PIR) обнаруживает движение, он посылает сигнал высокого уровня в течение 5 с, который считывается микроконтроллером, передающим затем последовательность команд на MP3-плеер для воспроизведения звукового файла. Если короткий сигнал низкого уровня (*LOW*) поступает на вывод IO2 MP3-плеера, то воспроизводится следующая дорожка. Однако если на вывод MP3-плеера поступает длинный сигнал низкого уровня, вместо переключения дорожек увеличивается громкость. Чтобы заменить микроконтроллер в функции переклю-

чения дорожек, сигнал высокого уровня PIR-датчика длительностью 5 с должен быть преобразован в сигнал низкого уровня длительностью всего 0,5 с.

Таймер 555 в моностабильном режиме с заданным соотношением резистора и конденсатора генерирует сигнал высокого уровня с длительностью 0,5 с, когда вывод *Trigger* подключен к GND. Следовательно, для начала требуется инвертирование длинного сигнала PIR-датчика высокого уровня. Затем можно запускать микросхему таймера 555 для генерации короткого сигнала высокого уровня, который также инвертируется и отправляется на вывод IO2 MP3-плеера.

Сигнал можно инвертировать с помощью биполярного транзистора, такого как транзистор BC548 или 2N2222. Чтобы проиллюстрировать инверсию сигнала, в схеме далее попеременно включаются и выключаются два светодиода. В примере Arduino IDE *Blink sketch* красный светодиод повторяет сигнал микроконтроллера ESP8266, а синий светодиод воспроизводит инвертированный сигнал (см. рис. 17-23). Красный светодиод и базовый вывод транзистора подключены к сигнальному выводу платы ESP8266. Когда состояние сигнального вывода имеет низкий уровень, в базе транзистора ток отсутствует, как и в цепи коллектор–эмиттер, поэтому ток от источника питания протекает через синий светодиод. И наоборот, когда состояние сигнального вывода имеет высокий уровень, базовый ток транзистора позволяет протекать току между коллектором и эмиттером. Падение напряжения на синем светодиоде больше, чем на промежутке коллектор–эмиттер открытого транзистора, поэтому транзистор закорачивает светодиод, и он выключается.

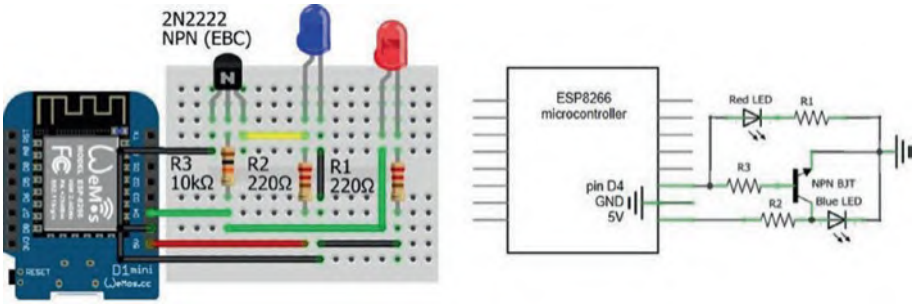


Рисунок 17-23. Инверсия сигнала с помощью биполярного транзистора

Схема PIR-датчика для запуска MP3-плеера на воспроизведение следующего звукового сопровождения показана на рис. 17-24, а подключения приведены в табл. 17-11. Когда PIR-датчик обнаруживает движение, он выдает сигнал высокого уровня в течение пяти секунд. NPN-транзистор может инвертировать длинный высокий сигнал PIR, превратив его в сигнал низкого уровня, запускающий микросхему таймера 555 для генерации короткого сигнала высокого уровня, который должен быть инвертирован с помощью второго NPN-транзистора, для получения короткого сигнала низкого уровня в соответствии с требованиями MP3-плеера для воспроизведения следующего саундтрека[113].

[113] Следует отметить, что указанный алгоритм может быть реализован с помощью куда менее громоздких схем, например на основе логических КМОП-инверторов. – *Прим. перев.*

Инвертированный длинный сигнал PIR-датчика на выводе *Trigger* микросхемы таймера 555 в натуральном виде будет перезапускать микросхему таймера 555 в течение пятисекундной длительности низкого уровня сигнала. Комбинация резистора (R_2) и конденсатора (C_3) используется для получения короткого импульса низкого уровня, посылаемого на вывод *Trigger* таймера 555. Резистор R_4 ограничивает ток от выхода сигнала PIR-датчика до необходимого уровня тока для базы NPN-транзистора, позволяющего транзистору открыть путь току через коллекторно-эмиттерную цепь. Точка соединения конденсатора C_3 и вывода *Trigger* таймера 555 оказывается на время подключена к GND через транзистор, так как конденсатор разряжен; затем конденсатор C_3 перезаряжается через резистор R_2, и вывод *Trigger* таймера 555 снова подключается к VCC через резистор R_2.

Вывод *Trigger* микросхемы таймера 555 подключается к низкому уровню на 41 мс, чего достаточно для запуска микросхемы таймера 555 в моностабильном режиме. На выходе *Output* устанавливается высокий уровень в соответствии с расчетной формулой $\ln(3) \times R_1C_1$, что дает 0,55 с при сопротивлении резистора 5,1 кОм и емкости конденсатора 10 мкФ. Второй NPN-транзистор инвертирует сигнал с выхода таймера 555, который затем отправляется на вывод IO2 MP3-плеера для воспроизведения следующего звукового сопровождения.

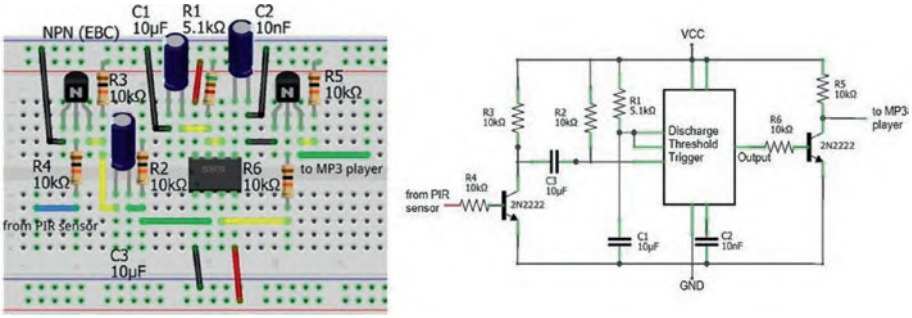


Рисунок 17-24. PIR-датчик и MP3-плеер

Таблица 17-11. PIR-датчик и MP3-плеер

Компонент	Подключено к	Также к
555 GND (вывод 1)	GND	
555 Trigger (вывод 2)	Резистор 1 кОм (R_2)	Конденсатор 10 мкФ (C_3) положительный вывод
555 Reset (вывод 4)	VCC	
555 Control (вывод 5)	Конденсатор 10 нФ[114]	GND (второй вывод 10 нФ)
555 Threshold (вывод 6)	555 Discharge (вывод 7)	
555 Discharge (вывод 7)	Резистор 5,1 кОм (R_1)	VCC (второй вывод R_1)
555 Discharge (вывод 7)	Конденсатор 10 мкФ (C_1) положительный вывод	GND (отрицательный вывод C_1)

[114] См. сноску 95 на стр. 315. – *Прим. перев.*

Окончание табл. 17-11

Компонент	Подключено к	Также к
555 VCC (вывод 8)	VCC	
Транзистор (от PIR) коллектор	Резистор 10 кОм (R_3)	Конденсатор 10 мкФ (C_3) отрицательный вывод
Транзистор (от PIR) база	Резистор 10 кОм (R_4)	От PIR-датчика
Транзистор (к MP3) коллектор	Резистор 10 кОм (R_5)	К MP3-плейеру
Транзистор (к MP3) база	Резистор 10 кОм (R_6)	555 Output (вывод 3)
Транзисторы эмиттер	GND	
Конденсаторы C_1 и C_2, отрицательный вывод	GND	

Последовательность сигналов показана на рис. 17-25, начиная с длинного сигнала высокого уровня с выхода PIR-датчика, который затем инвертируется NPN-транзистором, вызывая разрядку конденсатора C_3, а потом его перезарядку, что запускает микросхему таймера 555 для генерации короткого сигнала высокого уровня, инвертируемого вторым NPN-транзистором в короткий сигнал низкого уровня для подачи на MP3-плеер.

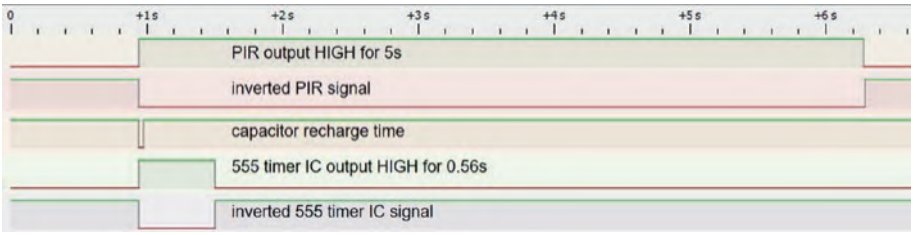


Рисунок 17-25. Последовательность сигналов PIR-датчика и MP3-плеера

Итоги

Функциональность микросхемы таймера 555 описана на примерах в моностабильном, бистабильном и генераторном режимах. Примеры включали моностабильный режим с фоторезистором, используемым для запуска временной задержки, режим генерации для создания электронного пианино, генерацию прямоугольного колебания с фиксированной частотой и переменным коэффициентом заполнения для управления сервоприводом и прямоугольного колебания с фиксированным коэффициентом заполнения и переменной частотой. Таймер 555 с *RC*-фильтрами нижних частот составили функциональный генератор для получения прямоугольных, треугольных и синусоидальных колебаний. Индуктивно-конденсаторный (*LC*) фильтр преобразовывал прямоугольное колебание в синусоиду. Преобразование сигнала с помощью микросхемы таймера 555 и транзисторов позволило с помощью обнаруживающего движение PIR-датчика запускать MP3-плеер для воспроизведения звуковой дорожки без участия микроконтроллера.

Перечень компонентов

- Микросхема таймера 555: стандартная NE555 и CMOS TLC555
- Резисторы 220 Ом (2 шт.), 1 кОм (3 шт.), 2 кОм, 4,7 кОм, 10 кОм (2 шт.), 20 кОм
- Конденсаторы: 10 нФ, 0,1 мкФ, 1 мкФ (4 шт.), 47 мкФ, 100 мкФ
- Индуктивность: 470 мкГн
- Тактовая кнопка: 2 шт.
- Светодиод: 2 шт.
- Диод IN4001 2 шт.
- Потенциометр 10 кОм
- NPN-транзистор: BC548 или 2N2222

Глава 18

Электрические измерения

Напряжение, ток, сопротивление и емкость измеряются с помощью мультиметра. Понимание требуемой схемы и программирование микроконтроллера ESP8266 или ESP32 для измерения напряжения, тока, сопротивления, емкости и индуктивности являются ценным упражнением при изучении электроники. Для измерения этих параметров необходимы делитель напряжения и аналого-цифровой преобразователь (АЦП), которые будут описаны перед проектами для различных измерений. Для каждого проекта подключение всех компонентов к общему проводу определяет точку отсчета напряжения. Если проект дает необычные результаты, то убедитесь, что подключение к общему проводу не нарушено.

Делитель напряжения

Делитель напряжения уменьшает входное напряжение до более низкого выходного напряжения. Делитель напряжения имеет входное напряжение V_{IN} и включает два резистора R_1 и R_2, при этом выходное напряжение измеряется на стыке этих двух резисторов (см. рис. 18-1). Выходное напряжение делителя V_{OUT} равно $V_{IN} \times \left(\frac{R_2}{R_1 + R_2} \right)$. Например, если два резистора имеют одинаковое значение, то выходное напряжение составляет половину входного напряжения. И наоборот, при известном входном напряжении, требуемом выходном и заданном значении резистора R_1 значение резистора R_2 можно подсчитать из формулы $R_1 \times \left(\frac{V_{OUT}}{V_{IN} - V_{OUT}} \right)$. Например, максимально возможное входное напряжение 12 В слишком велико для считывания на аналоговом выводе микроконтроллера ESP8266, и требуется делитель напряжения для снижения максимального напряжения до 3,2 В. Принимая значение 100 кОм для резистора R_1, значение резистора R_2 будет равно 36,4 кОм.

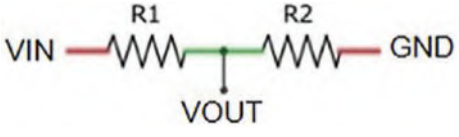


Рисунок 18-1. Делитель напряжения

Делитель напряжения, состоящий из пары резисторов 1 кОм и 2 кОм, уменьшает входное напряжение на треть, как и пара резисторов 10 кОм и 20 кОм. Разница между двумя такими парами резисторов заключается в том, что парой 10 кОм и 20 кОм потребляется меньшая мощность и выделяется меньше тепла. Мощность – это произведение напряжения и тока, равное в данном случае $\frac{V_{IN}^2}{R_1 + R_2}$. При входном напряжении 5 В мощность, потребляемая делителями напряжения с парой резисторов 1 кОм и 2 кОм и с парой резисторов 10 кОм и 20 кОм, составит 8,33 мВт и 0,83 мВт соответственно. Поэтому резисторы с высоким сопротивлением используются в делителях напряжения для снижения энергопотребления и выделения тепла[115].

Плата ESP8266 содержит внутренний делитель напряжения, состоящий из резисторов 100 кОм и 220 кОм (см. рис. 18-2), который увеличивает максимальное допустимое напряжение на выводе аналогового входа с опорного напряжения 1 В до 3,2 В. 10-разрядный аналого-цифровой преобразователь (АЦП) микроконтроллера ESP8266 преобразует напряжение в диапазоне от 0 до 3,2 В на выводе аналогового входа A0 в цифровое значение от 0 до 1023 (= $2^{10} - 1$). При заданном напряжении V_{IN} на выводе аналогового входа платы ESP8266 соответствующее значение АЦП равно $V_{IN} \times \frac{100\ \text{кОм}}{(220+100)\ \text{кОм}} \times 1024$.

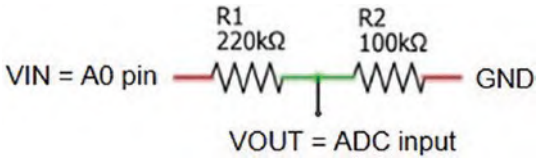


Рисунок 18-2. Делитель для входа аналого-цифрового преобразователя

Значения АЦП для входных напряжений от 3,2 В до 3,3 В ограничены значением 1023. Дополнительный резистор 10 кОм, подключенный между входным напряжением и выводом аналогового входа платы ESP8266, увеличивает предел 3,2 В на выводе аналогового входа до 3,3 В.

Функциональность 12-разрядного аналого-цифрового преобразователя (АЦП) микроконтроллера ESP32 преобразует напряжение от 0 до 3,3 В на выводе аналогового входа в цифровое значение от 0 до 4095 (= $2^{12} - 1$).

[115] Зато чем меньше сопротивления делителя, тем меньше «просадка» напряжения при подключении нагрузки и тем меньше уровень шума от электромагнитных наводок. Поэтому каждый практический случай требует отдельного рассмотрения; в общем случае можно рекомендовать выбирать сопротивления делителя настолько малыми, насколько это возможно без существенной перегрузки источника входного напряжения. – *Прим. перев.*

Аналого-цифровой преобразователь

В нескольких проектах в этой главе используется аналого-цифровой преобразователь последовательного приближения (SAR ADC), встроенный в микроконтроллер ESP8266 или ESP32. 10-разрядный (ESP8266) или 12-разрядный (ESP32) АЦП преобразует напряжение V_{IN} на выводе аналогового входа в цифровое значение между 0 и 1023 = 2^{10} – 1 или 4095 = 2^{12} – 1 относительно опорного напряжения V_{REF}, равного 1 В. Изменение разрешения АЦП микроконтроллера ESP32 описано в главе 21 («Микроконтроллеры»). Для каждого из 10 (ESP8266) или 12 (ESP32) шагов последовательного приближения напряжение на входе V_{IN} сравнивается с заданным напряжением $V_{SET(N)}$, сгенерированным цифроаналоговым преобразователем (ЦАП). Сравнение V_{IN} с первым заданным напряжением, равным $V_{SET(1)} = V_{REF}/2$, определяет старший бит (MSB) выходного цифрового значения АЦП. Если при последовательных сравнениях с заданными напряжениями $V_{SET(N)}$, равными $V_{SET(T-1)} \pm V_{REF}/2^N$ (знак ± зависит от предыдущего результата), V_{IN} оказывается больше, чем $V_{SET(N-1)}$, устанавливается последующий бит выходного цифрового значения АЦП.

Процесс преобразования АЦП можно проиллюстрировать на примере микроконтроллера ESP8266. Для входного напряжения 1,28 В, которое уменьшается до 0,40 В с помощью внутреннего делителя напряжения микроконтроллера, первое установленное напряжение $V_{SET(1)}$ равно $V_{REF}/2$ = 0,5 В. Поскольку V_{IN} на 0,1 В ниже, чем $V_{SET(1)}$, то значение старшего разряда (MSB) устанавливается равным нулю. Во втором сравнении $V_{SET(2)} = V_{SET(1)} - 1\ В/2^2$ = 0,25 В, и поскольку V_{IN} больше, чем $V_{SET(2)}$, то второй бит цифрового значения АЦП устанавливается равным единице, а третье заданное напряжение равно $V_{SET(3)} = V_{SET(2)} + 1\ В/2^3$ = 0,375 В. Десять последовательных сравнений приводят к 10-разрядному двоичному значению B0110011001, соответствующему цифровому выходу АЦП с десятичным значением 409. Иными словами, таким образом входное напряжение 1,28 В на выводе A0 платы ESP8266, уменьшенное до 0,4 В, масштабируется в соответствии с величиной 1024, равной 1 В, что приводит к округленному до целого результату (1024 × 0,4 В/1 В) = 409.

Преобразование аналогового напряжения в цифровое значение занимает у микроконтроллера ESP8266 135 мкс, за это время V_{IN} может изменяться. В АЦП встроен модуль выборки-хранения, включающий в себя MOSFET-ключ, который размыкается после запуска процесса АЦП, и конденсатор, на котором сохраняется постоянное значение V_{IN} (см. рис. 18-3). Для 10-разрядного АЦП существует 1024 уровня напряжения 0–1023; отсюда с учетом делителя напряжения и опорного напряжения 1 В дискретность напряжения АЦП составляет 3,125 мВ. Значения АЦП 0, 1 и 1023 соответствуют диапазонам значений V_{IN} от 0 В до менее чем 3,125 мВ, от 3,125 мВ до менее чем 6,25 мВ и от 3,197 В до менее чем 3,2 В соответственно. В предыдущем примере входное напряжение составляет 1,28 В и соответствует 410-му уровню напряжения с диапазоном от 1,278 В до менее 1,281 В.

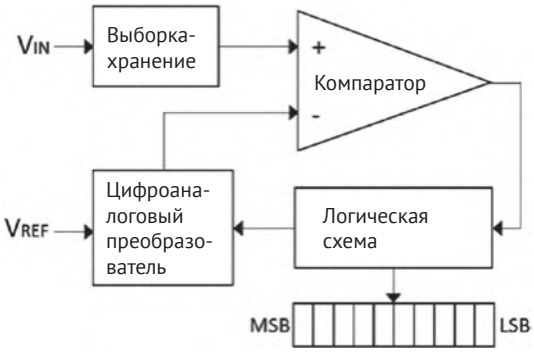


Рисунок 18-3. Аналого-цифровой преобразователь

Измеритель напряжения

Напряжение батареи измеряется с помощью АЦП микроконтроллера ESP8266 или ESP32. Хотя внутренний делитель напряжения микроконтроллера ESP8266 с резисторами 100 кОм и 220 кОм увеличивает максимальное входное напряжение до 3,2 В, внешний делитель напряжения дополнительно снижает напряжение батареи на выводе аналогового входа. Резисторы делителя напряжения 22 кОм и 10 кОм позволяют измерять максимальное напряжение батареи 10,24 В, равное $3{,}2\ B \times \left(\frac{10\ \text{кОм}}{(22+10)\ \text{кОм}} \right)$. Измеренное напряжение батареи представляет собой 10-разрядное выходное значение АЦП, умноженное на $10{,}24\ \text{В}/2^{10}$, которое отображается на OLED-экране с разрешением 128×32 пикселя (см. рис. 18-4 и соединения в табл. 18-1). Вместо внешнего делителя напряжения резисторы, соединенные последовательно с внутренним делителем напряжения и дополняющие его до 702 кОм, увеличивают максимальное напряжение на выводе аналогового входа микроконтроллера ESP8266 до 10,24 В.

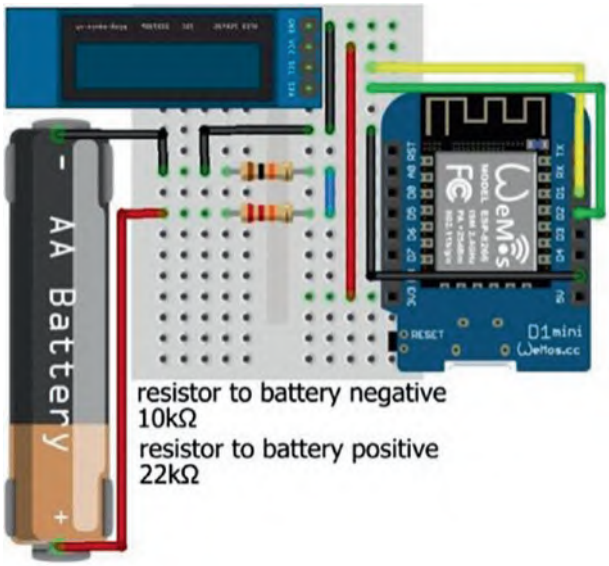


Рисунок 18-4. Измерение напряжения батареи с помощью LOLIN (WeMos) D1 mini

Таблица 18-1. Измерение напряжения батареи с помощью платы ESP8266

Компонент	Подключение к ESP8266	Также к
Батарея плюс	Резистор 22 кОм	A0 (второй вывод 22 кОм)
Батарея минус	Резистор 10 кОм	A0 (второй вывод 10 кОм)
Батарея минус	GND	OLED GND
OLED VCC	3V3	
OLED SDA	D2	
OLED SCL	D1	

В листинге 18-1 напряжение батареи отображается на OLED-дисплее в виде графического изображения батареи, иллюстрирующего уровень напряжения. Рамка батареи рисуется в виде сплошного прямоугольника размером 45×12 пикселей, начиная с позиции (0, 20) на OLED-экране, с помощью команды `oled.fillRect(0, 20, 45, 12, WHITE)`. Разряженная часть батареи представляет собой наложенный черный прямоугольник. Длина разряженной части представляет собой дополнение измеренного напряжения батареи, деленное на максимальное напряжение батареи и умноженное на 41, что равно длине рамки батареи, равной 45 пикселам, минус удвоенная ширина рамки, равная 2 пикселам. Крышка батарейного отсека добавляется в виде еще одного заполненного прямоугольника. Библиотека *Adafruit SSD1306* ссылается на библиотеки *Adafruit GFX* и *Wire*, поэтому инструкции `#include <Adafruit_GFX.h>` и `#include <Wire.h>` не требуются.

Листинг 18-1 предназначен для микроконтроллера ESP8266. Единственными изменениями в скетче при использовании микроконтроллера ESP32 являются установка вывода аналогового входа, поскольку плата разработки ESP32

имеет шесть таких выводов (см. главу 21 «Микроконтроллеры»), добавление инструкции `pinMode(ADCpin, INPUT)` и замена делителя 1024 на 4096 в инструкции

```
voltage = analogRead(ADCpin)*maxVolt/ 1024.0;
                                    // calculate battery voltage
```

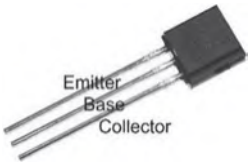
Листинг 18-1. Измерение напряжения батареи

```
#include <Adafruit_SSD1306.h>         // Adafruit SSD1306 library
int width = 128;                      // OLED screen size
int height = 32;                      // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int ADCpin = A0;                      // define analog input pin
float maxVolt = 10.24;                // maximum battery voltage
float voltage;
int battery;
void setup()
{                                     // OLED display and I2C address
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  oled.clearDisplay();                // clear OLED display
  oled.setTextColor(WHITE);           // set font color
  oled.setTextSize(2);                // set font size (1, 2, 3 or 4)
  oled.display();                     // start display instructions
}
void loop()
{                                     // calculate battery voltage
  voltage = analogRead(ADCpin)*max Volt/1024.0;
  oled.clearDisplay();
  oled.setCursor(0,0);                // position cursor at (0, 0)
  oled.print(voltage);
  oled.print("V");                    // display battery voltage
  oled.fillRect(0, 20, 45, 12, WHI TE);
                                      // battery frame 2 pixels width
  battery = 41*voltage/maxVolt;       // full battery section
  battery = constrain(battery, 0, 41);
  oled.fillRect(2+battery, 22, 41-bat tery, 8, BLACK);
                                      // empty battery section
  oled.fillRect(45, 23, 3, 6, WHITE); // battery top
  oled.display();
  delay(2000);                        // delay 2s between readings
}
```

На практике напряжения, измеренные микроконтроллерами ESP8266 и ESP32, использованными в этой главе, завышали и занижали подаваемые напряжения в диапазоне 1–3 В на 3 % и –16 % соответственно. Для сравнения, расчетное напряжение не было смещено при измерении с помощью микроконтроллера ATmega328P Arduino Uno. Смещенная оценка в *N* % корректируется введением поправки в измеренное напряжение с помощью инструкции `voltage = voltage/(1+N/100)`, где *N* имеет знак плюс, когда измеренное напряжение является заниженным, и наоборот[116].

[116] Главная причина возникновения ошибки в АЦП микроконтроллеров ESP8266 и ESP32 – нестабильность внутреннего опорного напряжения, которое зависит от экземпляра микросхемы. Для точного измерения необходимо обязательно провести индивидуальную калибровку, сравнив показания в какой-то точке (лучше бли-

Измеритель напряжения с нагрузкой



Батарея измеряется с нагрузкой и без нее с использованием биполярного транзистора (BJT) 2N2222 для включения или выключения тока батареи в нагрузке, состоящей из низкоомного резистора 10 Ом (см. рис. 18-5). Метод заимствован из проекта тестера батарей Андреаса Шписса (Andreas Spiess). При выключенном транзисторе напряжение батареи измеряется на выводе коллектора `collPin`. Базовый вывод транзистора `basePin` устанавливается в высокий уровень для включения транзистора, при этом батарея начинает обеспечивать питание нагрузки. Напряжение на нагрузке снова измеряется на выводе коллектора транзистора. Схемы для плат ESP8266 и ESP32 показаны на рис. 18-6 и 18-7, а соединения приведены в табл. 18-2.

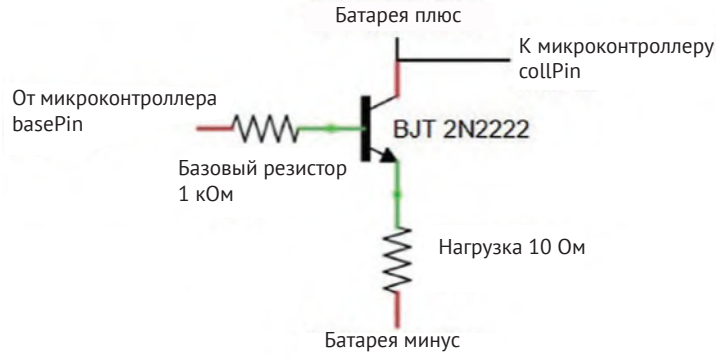


Рисунок 18-5. Измерение напряжения батареи с нагрузкой и без нагрузки

же к концу диапазона) с показаниями проверенного мультиметра. Из полученной разницы вычисляется ошибка в процентах, которая затем учитывается по формуле, приведенной автором. В микроконтроллерах Arduino ATmega328P совершенно такая же история, поскольку тип источников опорного напряжения используется тот же самый (т. н. *bandgap reference*), и автору, видимо, случайно повезло наткнуться на стабильный экземпляр. – *Прим. перев.*

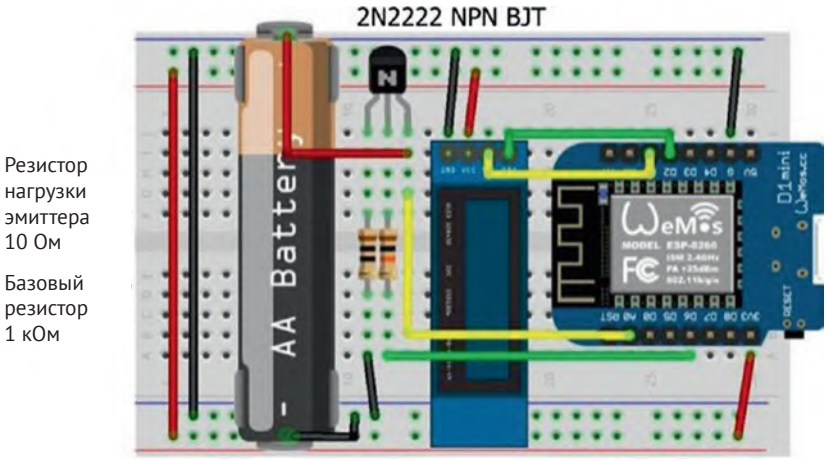


Рисунок 18-6. Тестер напряжения батареи с платой LOLIN (WeMos) D1 mini

Таблица 18-2. Тестер напряжения батареи с платами ESP8266 и ESP32

Компонент	ESP8266	Также к	ESP32	Также к
Транзистор эмиттер	Резистор 10 Ом	GND	Резистор 10 Ом	GND
Транзистор база	Резистор 1 кОм	D6	Резистор 1 кОм	GPIO 25
Транзистор коллектор	A0	Батарея плюс	GPIO 35	Батарея плюс
OLED VCC	3V3		3V3	
OLED GND	GND		GND	
OLED SDA	D2		GPIO 21	
OLED SCL	D1		GPIO 22	
Батарея минус	GND		GND	

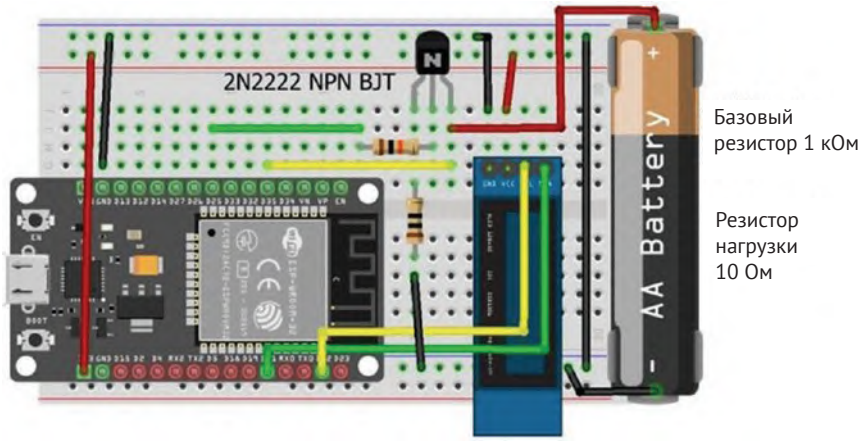


Рисунок 18-7. Тестер напряжения батареи с платой ESP32 DEVKIT DOIT

Схема в листинге 18-2 измеряет напряжение батареи с интервалом в две секунды и отображает напряжение батареи с нагрузкой и без нагрузки на OLED-экране. Повторные измерения напряжения батареи выполняются в функции `readings`, при этом среднее значение АЦП масштабируется на опорное напряжение.

Листинг 18-2 предназначен для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 номера выводов GPIO для базового и коллекторного выводов BJT изменяются (см. табл. 18-2), значение `maxVolt` увеличивается до 3,3 В, а команда `pinMode(collPin, INPUT)` включается в функцию `setup`. Делители на 1024 в функции `loop` изменены на 4096, поскольку микроконтроллер ESP32 имеет 12-разрядный АЦП по сравнению с 10-разрядным АЦП микроконтроллера ESP8266. Последующие схемы и скетчи ориентируются на плату LOLIN (WeMos) D1 mini и микроконтроллер ESP8266. Подключения к плате ESP32 и необходимые изменения в скетче для микроконтроллера ESP32 аналогичны только что описанным.

Листинг 18-2. Напряжение батареи с нагрузкой и без нее

```
#include <Adafruit_SSD1306.h>                // SSD1306 library for OLED
int width = 128;                             // OLED screen size
int height = 32;                             // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int basePin = D6;                            // BJT base pin
int collPin = A0;                            // BJT collector pin
unsigned long sum;
int reps = 10;                               // number of repeat measurements
float maxVolt = 3.2;                         // 3.2V maximum voltage
float Vbatt, Vload;
void setup()
{
  pinMode(basePin, OUTPUT);                  // define basePin as OUTPUT
  oled.begin(SSD1306_SWITCHCAPVCC , 0x3C);   // OLED I2C address
  oled.clearDisplay();                       // clear OLED display
  oled.setTextColor(WHITE);                  // OLED font color
  oled.setTextSize(1);                       // font size 1 character 6×8 pixels
  oled.display();                            // start display instructions
}
void loop()
{
  readings(LOW);                             // read BJT collector pin, BJT off
  Vbatt = maxVolt*sum/(reps*1024.0);         // battery voltage
  if(Vbatt > 0.1)
  {
    readings(HIGH);                          // read BJT collector pin, BJT on
    Vload = maxVolt*sum/(reps*102 4.0);      // battery voltage with load
    screen();                                // call screen function
  }
  else
  {
    oled.clearDisplay();                     // clear OLED when
    oled.display();                          // no battery voltage
  }
  delay(2000);                               // delay between measurements
}
```

```
void readings(int pinState)          // function to measure BJT pin
{
   digitalWrite(basePin, pinState);  // BJT base pin turned on or off
   sum = 0;
   for (int i=0; i<reps; i++)        // repeat voltage measurements
   {                                 // sum of voltage measurements
      sum = sum + analogRead(collPi n);
      delay(5);
   }
   digitalWrite(basePin, LOW);       // turn off BJT base pin
}
void screen()                        // function for OLED display
{
   oled.clearDisplay();              // clear OLED display
   oled.setCursor(0,0);              // move cursor to position (0,0)
   oled.print("battery ");oled.print(Vbatt,3);oled.println("V");
   oled.setCursor(0, 12);
   oled.print("+load ");oled.print(Vload,3);oled.print("V");
   oled.setCursor(0, 24);
   oled.print("perform ");oled.print(100.0*Vload/Vbatt,0);
   oled.print("%");
   oled.display();
}
```

Измеритель сопротивления (омметр)

Значение измеряемого резистора определяется с помощью делителя напряжения. Если входное напряжение равно V_{IN}, то соответствующее выходное значение АЦП с делителем напряжения равно $\frac{V_{IN}}{V_{REF}} \times \frac{R2}{R1+R2} \times 1024$, где V_{REF} – опорное напряжение АЦП для микроконтроллера с 10-разрядным АЦП, что показывает множитель $1024 = 2^{10}$. Если входное и опорное напряжения одинаковы, то преобразование формулы дает $R2 = \frac{R1 \times ADC}{1024 - ADC}$ Ом.

Вместо того чтобы использовать в делителе напряжения один резистор с известным сопротивлением, несколько резисторов с разными значениями обеспечивают диапазон известных сопротивлений для более точного измерения измеряемого сопротивления. Конкретные значения известных резисторов являются произвольными, но они могут охватывать некий диапазон, такой как 1 кОм, 4,7 кОм и 10 кОм, что позволяет надежно измерять значения сопротивления от 1 кОм до 10 кОм. Схема показана на рис. 18-8, подключения – в табл. 18-3.

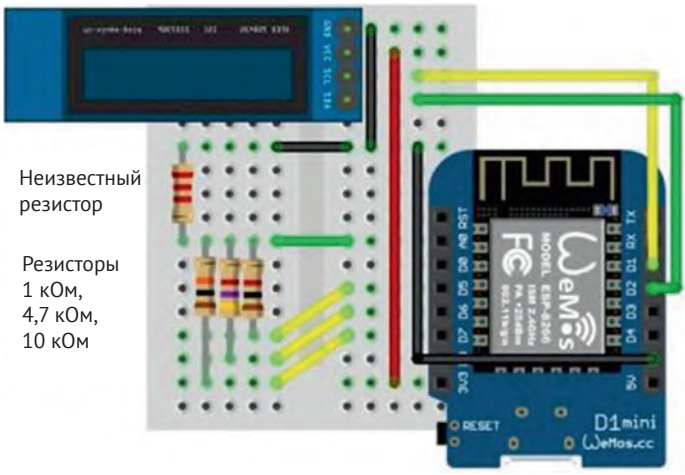


Рисунок 18-8. Измеритель сопротивления (омметр) с платой LOLIN (WeMos) D1 mini

Таблица 18-3. Измеритель сопротивления (омметр) с платой ESP8266

Компонент	Подключение к ESP8266	Также к
Известные резисторы	D5, D6, D7	A0
Измеряемый резистор	A0	GND
OLED VCC	3V3	
OLED GND	GND	
OLED SDA	D2	
OLED SCL	D1	

В листинге 18-3 перед получением показаний АЦП выводы, подключенные к известным резисторам, устанавливаются на вход (`INPUT`), что эквивалентно подключению резистора 100 МОм к каждому выводу и исключает протекание тока через эти резисторы. Затем каждый вывод, подключенный к известному резистору, последовательно устанавливается на выход (`OUTPUT`) с сигналом высокого (`HIGH`) уровня, который устанавливает входное напряжение делителя напряжения[117]. Определяется измеряемое значение сопротивления с учетом ограничений на показания АЦП, чтобы избежать экстремальных значений, и вывод известного резистора снова устанавливается на `INPUT`. На рис. 18-8 известный резистор (D5, D6 или D7) с меньшим значением подключен к меньшему номеру вывода платы ESP8266.

[117] Отметим, что такой способ установления входного напряжения делителя заключает в себе ошибку, связанную со значительным сопротивлением выхода микроконтроллера. – *Прим. перев.*

Листинг 18-3. Измеритель сопротивления (омметр)

```
#include <Adafruit_SSD1306.h>         // Adafruit SSD1306 library
int width = 128;                      // OLED screen size
int height = 32;                      // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int resistPin = A0;                   // analog input pin
int pin[] = {D5, D6, D7};             // pins for known resistor
float known[] = {1000.0, 4700.0, 10000.0}; // known resistor values
float resist, reading;
void setup()
{                                     // OLED display and I2C address
  oled.begin(SSD1306_SWITCHCA PVCC, 0 x3C);
  oled.clearDisplay();                // clear OLED display
  oled.setTextColor(WHITE);           // set font color
  oled.setTextSize(1);                // set font size (1, 2, 3 or 4)
  oled.display();                     // update display instructions
  for (int i=0; i<3; i++) pinMode(pin[i], INPUT);
}                                     // set known resistor pins to INPUT
void loop()
{
  oled.clearDisplay();                // clear OLED display
  oled.setCursor(0,0);
  oled.print("known ADC predict");    // header on OLED screen
  for (int i=0; i<3; i++)             // for each known resistor
  {
    pinMode(pin[i], OUTPUT);          // set known resistor pin
    digitalWrite(pin[i], HIGH);       // to OUTPUT and to HIGH
    reading = analogRead(resistPin); // voltage divider reading
    if(reading < 10 || reading > 1013) reading = 0;
                                      // constrain ADC reading
    resist = known[i]*reading/(1024.0-r eading);
                                      // calculate resistance
    pinMode(pin[i], INPUT);           // reset known resistor pin to INPUT
    oled.setCursor(1, (i+1)*8);       // OLED column 1, row 8, 16, 24
    oled.print(known[i],0);           // display known resistor
    oled.setCursor(50, (i+1)*8);
    oled.print(reading,0);            // display ADC reading
    oled.setCursor(90, (i+1)*8);
    oled.print(resist,0);             // display calculated resistance
    oled.display();
  }
  delay(5000);
}
```

Измеритель емкости

Емкость конденсатора измеряется по времени зарядки конденсатора. Когда резистор *R* соединен последовательно с конденсатором *C*, конденсатор будет заряжаться через резистор до тех пор, пока напряжение на конденсаторе не станет равным опорному напряжению V_{REF} (см. рис. 18-9). Напряжение на конденсаторе после *t* с зарядки равно $V_{REF}(1 - e^{-t/RC})$. После *RC* секунд зарядки напряжение на конденсаторе составляет 0,632 V_{REF}, так как $(1 - e^{-1}) = 0{,}632$.

Величина *RC* определяется в момент, когда показания аналого-цифрового преобразователя (АЦП) для напряжения на конденсаторе начинают превышать величину $647{,}3^{118} = 0{,}632 \times 1024$, поскольку ноль на входе означает ноль показаний АЦП, а V_{REF} – число 1023. Параметр *RC* (постоянная времени) также обозначается как τ, греческая буква tau.

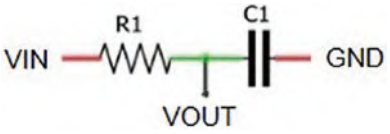


Рисунок 18-9. Последовательная RC-цепочка

Электролитические конденсаторы поляризованы, и на аноде должно быть более высокое напряжение, чем на катоде (отрицательном выводе). Катод имеет маркировку со знаком минус поверх цветной полосы на боковой стороне конденсатора. Длинный вывод электролитического конденсатора является анодом, или положительным выводом (см. рис. 18-10 и табл. 18-4). Если время зарядки конденсатора слишком мало, то резистор 10 кОм заменяется на резистор с более высоким сопротивлением; и наоборот, резистор 10 кОм заменяется на резистор с более низким сопротивлением, если время зарядки конденсатора слишком велико. Более высокие значения сопротивления резистора увеличивают время зарядки конденсатора для более точных оценок емкости. Напротив, низкое значение сопротивления резистора приводит к быстрому разряду конденсатора.

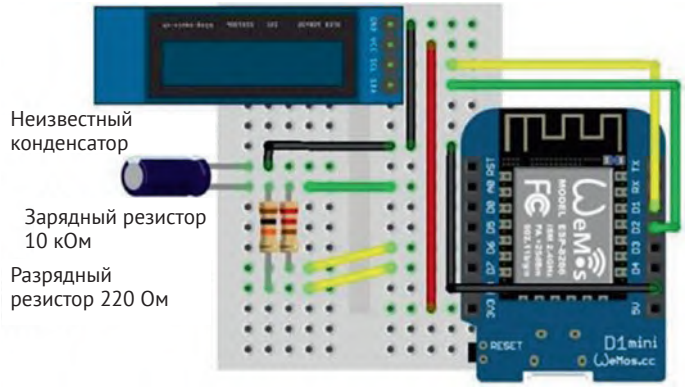


Рисунок 18-10. Измеритель емкости с платой LOLIN (WeMos) D1 mini

[118] Поскольку показания АЦП представляют собой целое число, то, очевидно, речь может идти о значении 647 ровно (см. далее описание скетча из листинга 18-4). – *Прим. перев.*

Таблица 18-4. Измеритель емкости с платой ESP8266

Компонент	Подключение к ESP8266	Также к
Зарядный резистор	D7	A0
Разрядный резистор	D6	A0
Измеряемый конденсатор, положительный вывод	A0	
Измеряемый конденсатор, отрицательный вывод	GND	OLED GND
OLED VCC	3V3	
OLED SDA	D2	
OLED SCL	D1	

В листинге 18-4 вывод заряда конденсатора устанавливается на высокий уровень (HIGH), и начинается заряд конденсатора. Когда значение АЦП для напряжения на конденсаторе достигает 648, отсчет прекращается, и емкость рассчитывается исходя из времени зарядки и значения резистора. Затем оба вывода для заряда и разряда конденсатора устанавливаются в низкий уровень, чтобы конденсатор мог разрядиться, после чего вывод разряда конденсатора устанавливается на вход (INPUT), чтобы ток не утекал через разрядный вывод, и цикл повторяется. Установка вывода на вход эквивалентна подключению резистора 100 МОм. Нижний предел расчетного значения емкости примерно равен 10 нФ. Скетч в листинге 18-4 был разработан на основе скетча в www.arduino.cc/en/Tutorial/CapacitanceMeter.

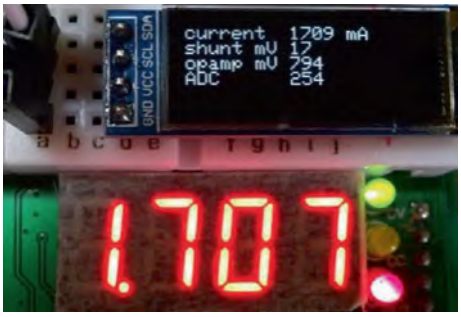
Листинг 18-4. Измеритель емкости

```
#include <Adafruit_SSD1306.h>          // Adafruit SSD1306 library
int width = 128;                       // OLED screen size
int height = 32;                       // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int capPin = A0;                       // capacitor positive pin
int chargePin = D7;                    // pin with 10kΩ charge resistor
int dischargePin = D6;                 // 220Ω discharge resistor pin
float resistor = 10000.0;              // 10kΩ charge resistor
unsigned long startTime;
float mF, uF, nF;                      // uF for microF (Greek letter μ)
void setup()
{                                      // OLED display and I2C address
  oled.begin(SSD1306_SWITCH CAPVCC, 0x3C);
  oled.clearDisplay();                 // clear OLED display
  oled.setTextColor(WHITE);            // set font color
  oled.setTextSize(2);                 // set font size (1, 2, 3 or 4)
  oled.display();                      // update display instructions
  pinMode(chargePin, OUTPUT);          // set charge pin
  digitalWrite(chargePin, LOW);        // as OUTPUT and to 0V
}
void loop()
{
  oled.clearDisplay();
  oled.setCursor(0,0);
```

```
    digitalWrite(chargePin, HIGH);      // charge pin to reference voltage
    startTime = millis();               // start timing charging capacitor
    while(analogRead(capPin) < 648) {} // do nothing while ADC < 648
    mF = (millis() - startTime) / resistor;
                                        // calculate capacitance = time/R
    uF = 1000.0 * mF;                   // change millifarad to microfarad
    if (uF > 1)
    {
      if (uF < 10) oled.print(uF, 1); // display capacitance with 1DP
      else oled.print(uF, 0);           // or 0DP depending on value
      oled.print(" uF");
    }
    else
    {
      nF = 1000.0 * uF;                 // convert to nanofarad
      if (nF > 10)                      // only display if value > 10nF
      {
      oled.print(nF, 0);                // display capacitance
      oled.print(" nF");
      }
    }
    digitalWrite(chargePin, LOW);       // set charge pin to 0V
    pinMode(dischargePin, OUTPUT);      // set discharge pin
    digitalWrite(dischargePin, LOW);    // to OUTPUT and 0V
    while(analogRead(capPin) > 0) {}    // do nothing while capacitor
                                        // discharges
    pinMode(dischargePin, INPUT);       // set discharge pin to INPUT
    oled.display();
    delay(2000);                        // to ensure no current flows
}
```

Более точные цифры для конденсаторов с низкими значениями получаются при замене функции `millis` функцией `micros` для измерения временного интервала в микросекундах, а не в миллисекундах, и деления `(micros()-startTime)` на 1000. 32-разрядный таймер отсчитывает до $2^{32} - 1$, а затем сбрасывается, поэтому функции `millis` и `micros` сбрасываются через 49,7 дня и 71,58 минуты соответственно.

Измеритель тока (амперметр)



Ток (I) в цепи измеряется по напряжению (V) на резисторе с известным сопротивлением (R), специально включенным в цепь, по формуле $I = V/R$. Резистор должен иметь низкое сопротивление, чтобы не допустить значительного падения напряжения. Токовый шунт 0,01 Ом представляет собой резистор, подбираемый по максимально допустимым току и падению напряжения при этом токе, например 100 мВ при 10 А. При маломощном резисторе выделяющаяся мощность V^2/R получается слишком большой, но токовый шунт специально предназначен для отвода выделяемого тепла.

Ожидаемое напряжение на шунтирующем резисторе 0,01 Ом составляет 10 мВ на ампер тока, поэтому измерение тока только с помощью шунтирующего резистора имеет низкое разрешение (подключение см. на рис. 18-11). С 10-разрядным аналого-цифровым преобразователем (АЦП) микроконтроллера ESP8266 увеличение тока на 1 А увеличивает значение АЦП на 3,2 = 10 мВ × $2^{10}/V_{REF}$, учитывая опорное напряжение V_{REF} микроконтроллера ESP8266, равное 3,2 В[119]. И наоборот, увеличение значения АЦП на единицу соответствует увеличению напряжения на 3,125 мВ или 3,2 В/2^{10} на шунтирующем резисторе 0,01 Ом, т. е. увеличению тока на 312 мА.

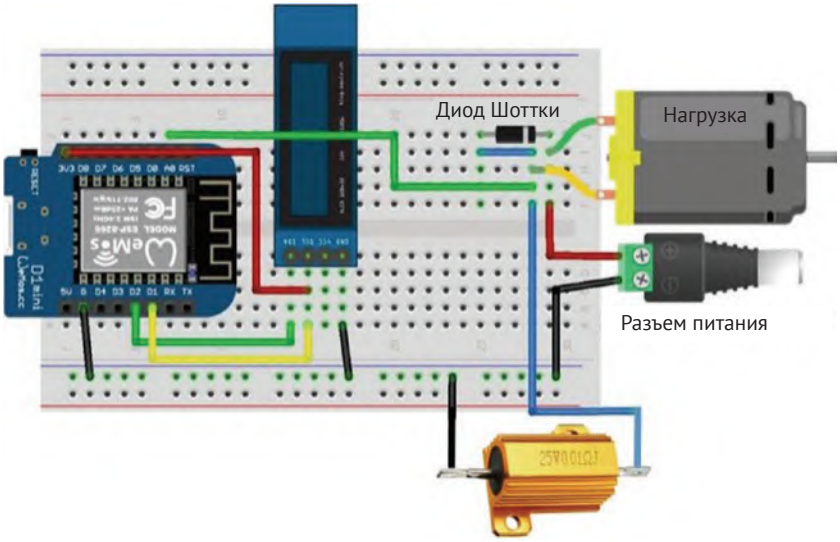


Рисунок 18-11. Шунтирующий резистор и плата LOLIN (WeMos) D1 mini

С помощью операционного усилителя, такого как LM358, или инструментального усилителя, такого как AD623, можно усилить напряжение на шунтирующем резисторе для увеличения разрешения измерения тока. Коэффициент усиления неинвертирующего усилителя на основе LM358 определяется соотношением резисторов обратной связи R_{fb} и заземления R_{GND} согласно выражению $1 + R_{fb}/R_{GND}$ (см. рис. 18-12). В этой схеме два резистора, R_{GND} и R_{fb}, образуют делитель напряжения. Напряжения на выходе операционного усилителя и выводе инвертирующего входа эквивалентны входному и выходному напряжениям обычного делителя. Выходное напряжение V_{OUT} делителя напряжения равно $V_{IN} \times \left(\frac{R2}{R1+R2} \right)$. Подстановка обозначений резисторов в эту формулу дает $V_{opIN} = V_{poOUT} \times \left(\frac{R_{GND}}{R_{fb}+R_{GND}} \right)$ и выражение для коэффициента усиления $V_{opOUT}/$

[119] В данном случае под опорным напряжением (V_{REF}) автор подразумевает верхний предел измерения АЦП, определяемый установленным на входе делителем (см. раздел «Измеритель напряжения» в этой главе). Внутреннее опорное напряжение АЦП не изменилось и принимается равным примерно 1 В, как и ранее. – *Прим. перев.*

$V_{opIN} = 1+R_{fb}/R_{GND}$. Например, коэффициент усиления, равный 46, получается с помощью пары резисторов 100 кОм и 2,2 кОм. Для уменьшения шума к выходу операционного усилителя подключен конденсатор емкостью 0,01 мкФ.

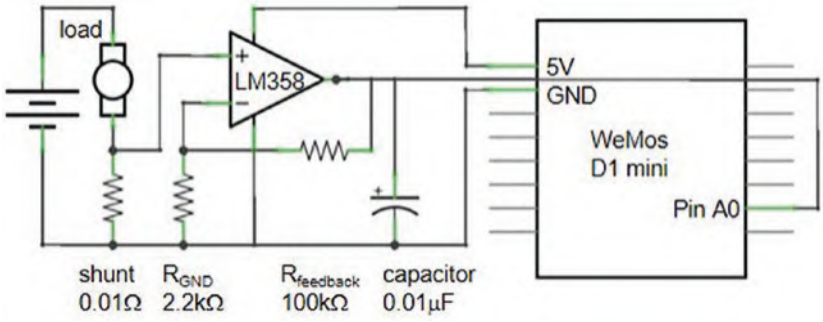


Рисунок 18-12. Неинвертирующий усилитель

Коэффициент усиления (*gain*) операционного усилителя по напряжению увеличивает разрешающую способность измерителя тока. Например, при токе 1 А пара резисторов 100 кОм и 2,2 кОм увеличивает выходное напряжение операционного усилителя LM358 с 10 мВ до 465 мВ, создавая значение АЦП, равное 149 = *коэффициент усиления* × 10 мВ × 1024/V_{REF}, что существенно выше, чем значение АЦП, равное 3 без усиления. Аналогично единичное изменение значения АЦП соответствует изменению напряжения на резисторе шунтирования на 0,067 мВ = V_{REF}/(1024 × *коэффициент усиления*), что соответствует разности токов всего 6,7 мА.

На рис. 18-13 к схеме на рис. 18-11 добавлен операционный усилитель LM358 для усиления напряжения от шунтирующего резистора, для увеличения значения АЦП и, соответственно, увеличения разрешения измерителя тока. Соединения приведены в табл. 18-5.

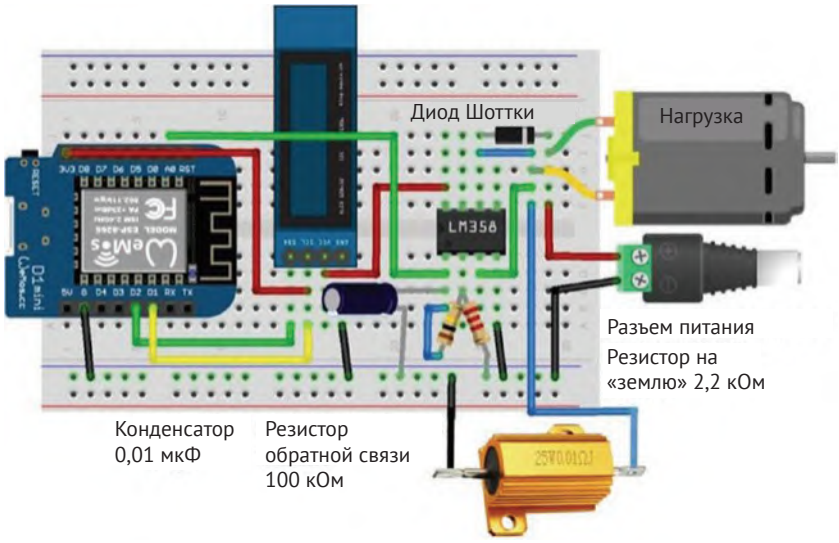


Рисунок 18-13. Операционный усилитель, шунт и плата LOLIN (WeMos) D1 mini

Если нагрузкой является двигатель постоянного тока, который частично является индуктивностью, то он будет генерировать напряжение для поддержания тока при отключении питания. Установка диода параллельно обмотке двигателя предотвращает скачок напряжения и рассеивает энергию через двигатель при отключении питания. Рекомендуется использовать диод Шоттки, который представляет собой быстро переключающийся диод с низким прямым падением напряжения.

Таблица 18-5. Операционный усилитель, шунтирующий резистор и плата ESP8266

Компонент	Подключено к	Также к
LM358 выход (вывод 1)	ESP8266 A0	
LM358 выход (вывод 1)	Конденсатор 0,01 мкФ[120]	ESP8266 GND (второй вывод конденсатора)
LM358 инвертирующий вход (вывод 2)	Резистор обратной связи R_{fb}	LM358 выход (вывод 1) (второй вывод резистора)
LM358 инвертирующий вход (вывод 2)	Заземляющий резистор R_{GND}	ESP8266 GND (второй вывод резистора)
LM358 неинвертирующий вход (вывод 3)	Нагрузка отрицательный вывод	
LM358 GND (вывод 4)	ESP8266 GND	
LM358 VCC (вывод 8)	ESP8266 3V3	
Нагрузка положительный вывод	Катод диода Шоттки (полоса)	Внешний источник, положительный вывод
Нагрузка отрицательный вывод	Анод диода Шоттки	Шунт 0,01 Ом
ESP8266 GND	Шунт 0,01 Ом	Внешний источник, отрицательный вывод
OLED VCC	ESP8266 3V3	
OLED GND	ESP8266 GND	
OLED SDA	ESP8266 D2	
OLED SCL	ESP8266 D1	

Скетч (см. листинг 18-5) преобразует показания АЦП, соответствующие напряжению на шунте, усиленного в соответствии с коэффициентом усиления операционного усилителя, в ток через нагрузку. Повторяющиеся показания АЦП усредняются для снижения влияния шума.

[120] Конденсатор 0,01 мкф (10 нФ) ошибочно показан на схемах 18-12 и 18-13 как полярный электролитический. Конденсатор такой емкости встречается только среди неполярных типов, самый распространенный – керамический К10-17. – *Прим. перев.*

Листинг 18-5. Амперметр с шунтирующим резистором

```
#include <Adafruit_SSD1306.h>          // Adafruit SSD1306 library
int width = 128;                       // OLED screen size
int height = 32;                       // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int ADCpin = A0;
unsigned long ADC;
int Rfeedback = 100000;                // feedback resistor value
int RGND = 2200;                       // resistor GND value
float current, opAmpmV, shuntmV, gain;
void setup()
{
   Serial.begin(115200);
   oled.begin(SSD1306_SWITCH CAPVCC, 0x3C);
                                       // OLED display and I2C address
   oled.clearDisplay();                // clear OLED display
   oled.setTextColor(WHITE);           // set font color
   oled.setTextSize(1);                // set font size (1, 2, 3 or 4)
   oled.display();                     // update display instructions
   gain = 1.0 + Rfeedback/RGND;        // calculate op amp gain
}
void loop()
{
   ADC = 0;
   for (int i=0; i<100; i++)           // repeated analog readings
   {
      ADC = ADC + analogRead(ADCpin);
      delay(10);
   }
   ADC = ADC/100;                      // average of analog readings
   opAmpmV = ADC*3200.0/1024;          // op amp output voltage
   shuntmV = opAmpmV / gain;           // voltage on shunt
   current = 100.0 * shuntmV;          // shunt mV to current in mA
   oled.clearDisplay();
   oled.setCursor(0,0);                // display results on OLED
   oled.print("current");oled.print(current,0);
   oled.println(" mA");
   oled.print("shunt mV ");oled.println(shuntmV,0);
   oled.print("opamp mV ");oled.println(opAmpmV,0);
   oled.print("ADC ");oled.println(ADC);
   oled.display();
   delay(1000);
}
```

На рис. 18-14 показано измерение тока с помощью мультиметра и амперметра со светодиодным дисплеем. В обоих случаях измеритель устанавливается последовательно с нагрузкой, хотя из компоновки схемы с шунтирующим резистором и операционным усилителем на рис. 18-13 это не так очевидно. Амперметр со светодиодным дисплеем имеет шунтирующий резистор 0,05 Ом (R050) и питается от внешнего источника.

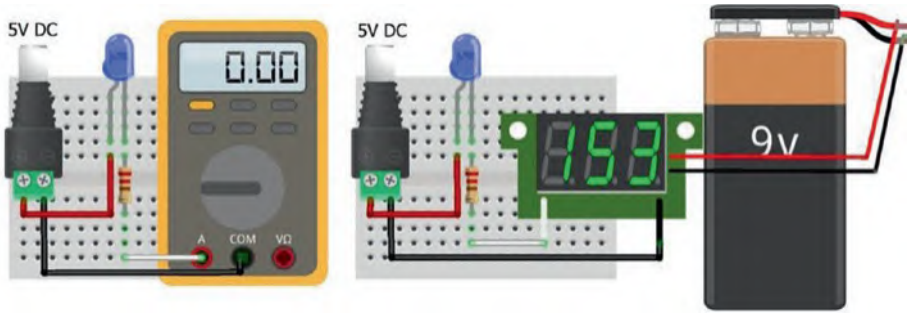
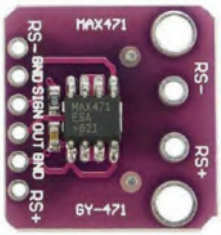


Рисунок 18-14. Измеритель тока (амперметр)

Датчик тока



Ток измеряется модулем MAX471 с верхним пределом 3 А. Положительный вывод нагрузки подключен к контакту *RS*– MAX471, а отрицательный – к GND, при этом положительный вывод источника питания нагрузки подключен к контакту *RS*+ MAX471 (см. рис. 18-15 и подключения в табл. 18-6). Разрешение по умолчанию для модуля MAX471 составляет 3,125 мА, что равно 3,2 А/2^{10}, учитывая 10-разрядный АЦП микроконтроллера ESP8266. Напряжение на выводе *OUT* MAX471 в идеале эквивалентно 1 В на ампер тока. Вывод знака *SIGN* MAX471 указывает направление тока, например разряд или заряд внешнего аккумулятора.

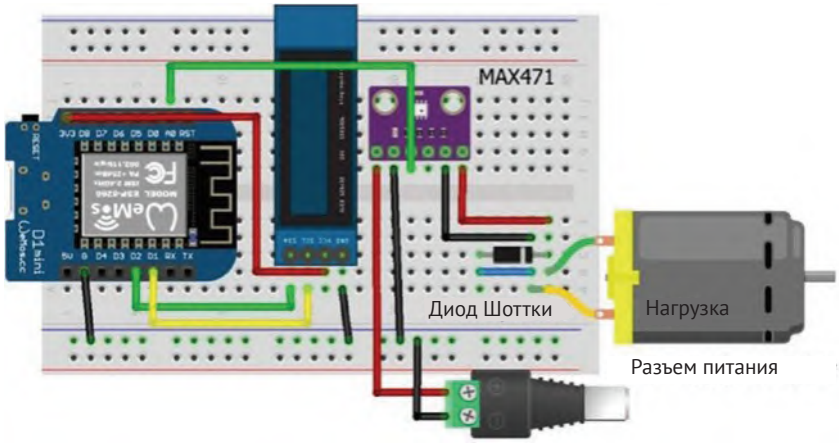


Рисунок 18-15. Датчик тока MAX471 и плата LOLIN (WeMos) D1 mini

Таблица 18-6. Датчик тока MAX471 и плата ESP8266

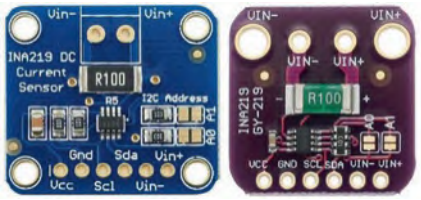
Компонент	Подключено к	Также к
MAX471 RS+	Внешний источник, положительный вывод	
MAX471 GND	Внешний источник, отрицательный вывод	ESP8266 GND
MAX471 OUT	ESP8266 A0	
MAX471 SIGN		
MAX471 GND	Нагрузка отрицательный вывод	
MAX471 RS-	Нагрузка положительный вывод	
OLED VCC	ESP8266 3V3	
OLED GND	ESP8266 GND	
OLED SDA	ESP8266 D2	
OLED SCL	ESP8266 D1	

В скетче (см. листинг 18-6) ток нагрузки измеряется с помощью модуля MAX471, причем измерения отображаются каждые пять секунд.

Листинг 18-6. Датчик тока MAX471 и плата ESP8266

```
#include <Adafruit_SSD1306.h>                // Adafruit SSD1306 library
int width = 128;                             // OLED screen size
int height = 32;                             // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int currentPin = A0;                         // analog input pin
float current;
void setup()
{
  oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);    // OLED I2C address
  oled.clearDisplay();                       // clear OLED display
  oled.setTextColor(WHITE);                  // set font color
  oled.setTextSize(3);                       // set font size (1, 2, 3 or 4)
  oled.display();                            // update display instructions
}
void loop()
{
  current = analogRead(currentPin);          // analog MAX471 reading
  current = 1000*current*3.2/1024;           // convert to current
  oled.clearDisplay();
  oled.setCursor(0,0);                       // position cursor
  oled.print(current,0); oled.print(" mA");  // display current
  oled.display();
  delay(5000);
}
```

Датчик тока и напряжения



Ток и напряжение совместно измеряются с помощью модуля INA219, который имеет как минимум два режима, из которых можно вывести соответствующие мощность и количество потребленной энергии с течением времени. Инструкции библиотеки *Adafruit INA219* `getShuntVoltage_mV()` и `getBusVoltage_V()` считывают напряжение на шунтирующем резисторе модуля INA219 0,1 Ом (R100) и на выводах нагрузки, равное напряжению питания минус напряжение шунта. Измерение мощности производится с помощью команды `getPower_mW()` или рассчитывается как произведение напряжения и тока (см. рис. 18-16 и подключения в табл. 18-7).

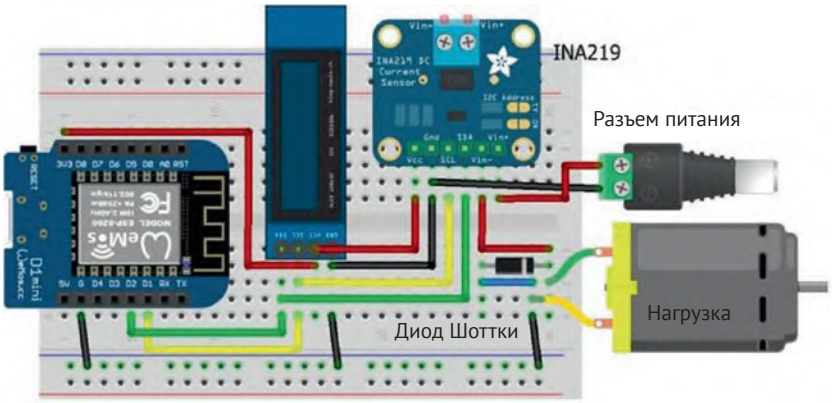


Рисунок 18-16. Датчик тока INA219 и плата LOLIN (WeMos) D1 mini

Модуль INA219 измеряет напряжение на шунтирующем резисторе 0,1 Ом с максимальным входным напряжением 320 мВ. Максимальный ток, который измеряет модуль INA219, равен 3,2 А, что соответствует 320 мВ / 0,1 Ом. Разрешение модуля INA219 по умолчанию составляет 0,8 мА, что равно 3,2 А/2^{12} = 3,2 А/4096, с учетом 12-разрядного встроенного аналого-цифрового преобразователя INA219. Разрешение может увеличиваться до 0,1 мА, равного 400 мА/2^{12}, за счет уменьшения коэффициента усиления усилителя INA219 в восемь раз, что приводит к максимальному входному напряжению 40 мВ и максимальному току 400 мА. Команды для настройки стандартного и высокого разрешения – `setCalibration_32V_2A()` и `setCalibration_16V_400mA()` соответственно. Опция промежуточного разрешения доступна с помощью команды `setCalibration_32V_1A()`.

Модуль INA219 использует интерфейс I2C для связи с микроконтроллером ESP8266 или ESP32, а библиотека *Adafruit INA219* ссылается на библиотеку *Wire*, поэтому инструкция `#include <Wire.h>` не требуется. Адрес I2C по умолчанию для модуля INA219 имеет значение 0x40, которое изменяется на 0x41, 0x44 или 0x45 с помощью установки на модуле перемычки A0, перемычки A1 или обеих перемычек A0 и A1 соответственно. Например, если адрес I2C установлен как 0x41, инициализация выполняется инструкцией `Adafruit_INA219 ina219(0x41)`.

Обратите внимание, что если используется адрес I2C по умолчанию, то достаточно инструкции `Adafruit_INA219 ina219`.

Если нагрузкой является двигатель постоянного тока, который частично является индуктивностью, то он будет генерировать напряжение для поддержания тока при отключении питания. Установка диода параллельно обмотке двигателя предотвращает скачок напряжения и рассеивает энергию через двигатель при отключении питания. Рекомендуется использовать диод Шоттки, который представляет собой быстро переключающийся диод с низким прямым падением напряжения.

Таблица 18-7. Датчик тока INA219 с платой ESP8266

Компонент	Подключено к ESP8266	Также к
INA219 VCC	3V3	
INA219 GND	GND	Внешний источник отрицательный вывод
INA219 SCL	D1	
INA219 SDA	D2	
INA219 VIN-	Нагрузка положительный вывод	
INA219 VIN+	Внешний источник положительный вывод	
Катод диода Шоттки (полоса)	Нагрузка положительный вывод	
Анод диода Шоттки	Нагрузка отрицательный вывод	INA219 GND
OLED VCC	3V3	
OLED GND	GND	
OLED SDA	D2	
OLED SCL	D1	

В листинге 18-7 с помощью модуля INA219 измеряются ток, мощность и потребленная нагрузкой энергия, причем измерения отображаются каждые пять секунд. На втором OLED-экране отображаются напряжения питания и нагрузки. Выбрана опция высокой точности с максимальным током 400 мА.

Листинг 18-7. Датчик тока INA219

```
#include <Adafruit_INA219.h>          // define Adafruit_INA219 lib
Adafruit_INA219 ina219;               // default I2C, associate ina219 with lib
#include <Adafruit_SSD1306.h>         // Adafruit SSD1306 library
int width = 128;                      // OLED screen size
int height = 32;                      // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
float shunt, load, supply, current, power;
float energy = 0;
void setup()
{
   ina219.begin();
// ina219.setCalibration_32V_2A();    // default precision option
```

```
// ina219.setCalibration_32V_1A();           // intermediate option
   ina219.setCalibration_16V_400mA();        // high precision option
   oled.begin(SSD1306_SWITCHCAPVCC, 0x3 C);  // OLED I2C address
   oled.clearDisplay();                      // clear OLED display
   oled.setTextColor(WHITE);                 // set font color
   oled.setTextSize(1);                      // set font size (1, 2, 3 or 4)
oled.display();                              // update display instructions
}
void loop()
{
   shunt = ina219.getShuntVoltage_mV();      // shunt voltage in mV
   load = ina219.getBusVoltage_V();          // load voltage in V
   supply = load + shunt / 1000.0;           // supply voltage in V
   current = ina219.getCurrent_mA();         // current in mA
   power = ina219.getPower_mW();             // power in mW
   energy = energy + power / 3600.0;         // energy in mAh
   oled.clearDisplay();
   oled.setCursor(0,0);                      // display results
   oled.print("current ");oled.print(current,0);
   oled.println(" mA");
   oled.print("shunt ");oled.print(shunt,0); oled.println(" mV");
   oled.print("power ");oled.print(power,0); oled.println(" mW");
   oled.print("energy ");oled.print(energy,0); oled.print(" mAh");
   oled.display();
   delay(5000);
   oled.clearDisplay();
   oled.setCursor(0,0);                      // display supply and load V
   oled.print("supply ");oled.print(supply); oled.println(" V");
   oled.print("load ");oled.print(load); oled.println(" V");
   oled.display();
   delay(5000);
}
```

Измеритель для солнечной панели с аккумулятором

Приложение для измерения тока и напряжения отслеживает чистый ток заряда или разряда аккумуляторной батареи, обеспечивающей питание нагрузки и заряжающейся через солнечную панель. Выходные токи солнечной панели и батареи, а также напряжение батареи выводятся графически на TFT ЖК-экран ST7735 с указанием минимального, текущего и максимального выходных токов батареи, текущего напряжения батареи и суммарной потребленной энергии (см. рис. 18-17). На графике отображается выходной ток батареи с положительными значениями при разряде батареи и отрицательными значениями при заряде.

Время находится на оси X, выходная мощность солнечной панели и аккумулятора – на левой оси Y, а напряжение аккумулятора – на правой оси Y. Рисунок 18-17 (левый график) демонстрирует комбинацию солнечной панели и аккумулятора, обеспечивающих питание светодиода. Ток солнечной панели (белая линия на левой оси) увеличился с начальных 5 мА с увеличением солнечного света, выходной ток аккумулятора (зеленая линия на левой оси) уменьшился, а напряжение аккумулятора увеличилось до 4,08 В (значение для желтой линии на правой оси). Текущий ток солнечной панели на выходе равен

23 мА (надпись «sol 23»). Минимальные, текущие и максимальные выходные значения тока аккумулятора отображаются под графиком. В течение этого периода времени потребление энергии от аккумулятора составило 7 мАч (описание элементов экрана см. на рис. 18-18).

Рисунок 18-17. Работа солнечной панели с отображением тока и напряжения аккумуляторной батареи

Когда светодиод был удален (рис. 18-17, правый график), выходной ток солнечной панели (белая линия на левой оси) составлял 20 мА, выходной ток батареи –5 мА указывал на зарядку батареи (красная линия на левой оси), а общая энергия, подаваемая на батарею, составила 3 мАч. Напряжение батареи поддерживалось на уровне 4,4 В (желтая линия на правой оси).

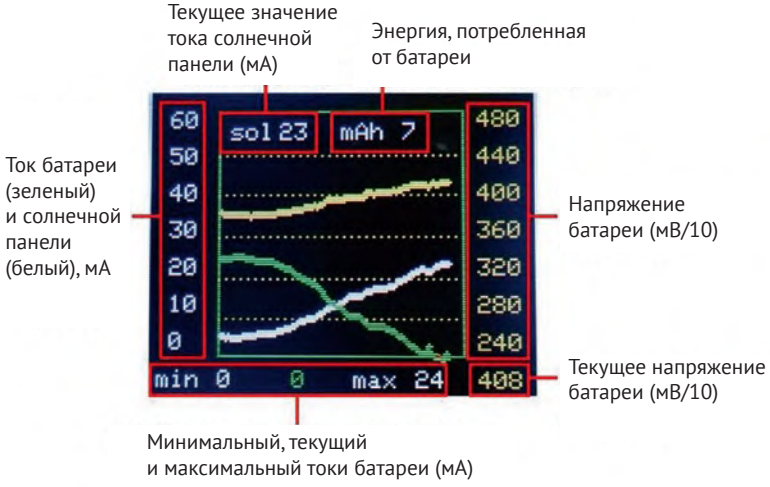


Рисунок 18-18. Расположение элементов отображения на экране

Солнечная панель заряжает литиевый аккумулятор напряжением 3,7 В через модуль заряда TP4056 (см. рис. 18-19). Пара выводов модуля заряда TP4056 IN- и OUT- имеет внутреннее соединение, как и пара OUT+ и B+. Напротив, пара выводов B- и OUT- соединяется только при наличии напряжения на клем-

ме B+. Более подробная информация о модуле зарядки аккумулятора TP4056 приведена в главе 6 («Bluetooth-динамик»). Модуль питания с повышающим преобразователем постоянного тока MT3608 увеличивает выходное напряжение литиевого элемента от 3,7 В до 5 В в соответствии с требованиями нагрузки. Выводы повышающего преобразователя MT3608 VIN- и VOUT- соединены внутри платы.

Выходные токи солнечной панели и аккумулятора измеряются с помощью модулей INA219. Изменение шкалы графика для выходного тока с 0–60 мА на 0–300 мА выполняется с помощью кнопки, запускающей процедуру прерывания (ISR), в которой задается максимальное значение графика. Максимальное напряжение на выводе аналогового входа микроконтроллера ESP8266 составляет 3,2 В; потому делитель напряжения, состоящий из двух резисторов 10 кОм, вдвое снижает напряжение аккумулятора на аналоговом входе. Соединения компонентов приведены в табл. 18-8.

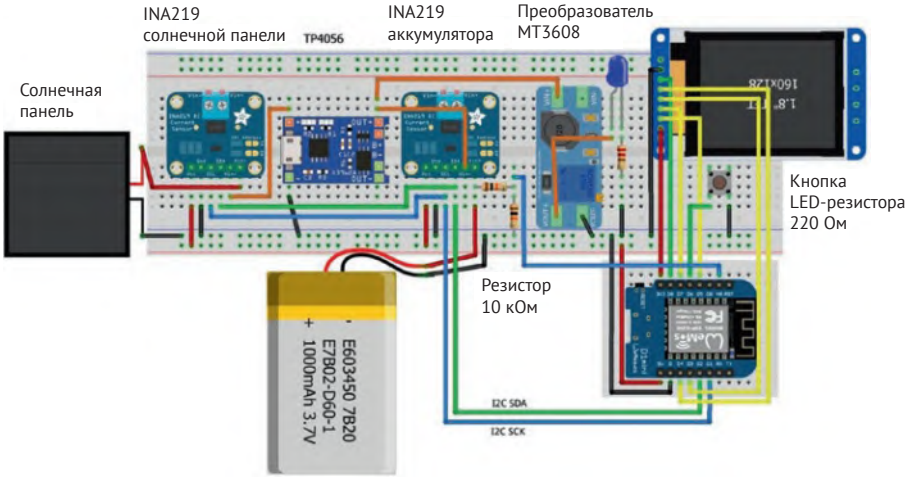


Рисунок 18-19. Измеритель тока солнечной панели и аккумуляторной батареи

Таблица 18-8. Измеритель тока солнечной панели и аккумуляторной батареи

Компонент	Подключено к	Также к
INA219 солнечной панели VIN+	Солнечная панель плюс	
INA219 солнечной панели VIN-	TP4056 (IN) +	
INA219 аккумулятора VIN+	Аккумулятор плюс	
INA219 аккумулятора VIN+	TP4056 OUT+/B+	
Преобразователь MT3608 VIN+	TP4056 OUT+/B+	
Преобразователь MT3608 VOUT+	Нагрузка положительный вывод	
ESP8266 D2 (SDA)	INA219 солнечной панели SDA	INA219 аккумулятора SDA
ESP8266 D1 (SCL)	INA219 солнечной панели SCL	INA219 аккумулятора SCL

Окончание табл. 18-8

Компонент	Подключено к	Также к
ST7735 TFT LCD GND	ESP8266 GND	
ST7735 TFT LCD CS	ESP8266 D8 (SPI CS)	
ST7735 TFT LCD RESET	ESP8266 D4	
ST7735 TFT LCD A0 или DC	ESP8266 D3	
ST7735 TFT LCD SDA	ESP8266 D7 (SPI MOSI)	
ST7735 TFT LCD SCK	ESP8266 D5 (SPI SCL)	
ST7735 TFT LCD LED	ESP8266 3V3	
Кнопка	ESP8266 D6	ESP8266 GND
ESP8266 5V	INA219 солнечной панели VCC	INA219 аккумулятора VCC
ESP8266 GND	INA219 солнечной панели GND	INA219 аккумулятора GND
ESP8266 GND	Солнечная панель минус	TP4056 (IN) –
ESP8266 GND	Преобразователь MT3608 VOUT–	Нагрузка отрицательный вывод
ESP8266 GND	Аккумулятор минус	

В первом разделе скетча в листинге 18-8 определяются библиотеки и переменные, а функция `setup` инициализирует модули INA219, TFT-ЖК-экран ST7735 и прерывание. Функция `graph` создает отображение графика с нанесенными точками данных и обновлениями графика, выполняемыми с помощью функций `screenVal` и `detail` соответственно. Функция `screenVal` отображает минимальные, текущие и максимальные выходные значения тока аккумулятора под графиком. Функция `detail` вычисляет совокупную выработку энергии батареей, которая отображается вместе с выходным током солнечной панели. Как в функции `screenVal`, так и в функции `detail` черные прямоугольники заменяют предыдущий текст и значения, поскольку для нового отображаемого текста и значений может потребоваться меньше символов. Команда `setTextColor(color, BLACK)` относится только к существующему тексту. Шестнадцатеричные коды для цветов определены в файле *Adafruit_ST77xx.h* библиотеки *Adafruit_ST7735_and_ST7789*.

В функции `loop` измеряются и выводятся на график выходные токи аккумулятора и солнечной панели, а также напряжение аккумулятора, при этом для отображения значений вызывается функция `screenVal`. Чтобы различать разряжающуюся и заряжающуюся батарею, цвет отображаемого значения и точек данных графика меняется с зеленого на красный, при этом отрицательная (зарядная) кривая графика отображается с изменением знака тока аккумулятора. Базовая линия графика находится в строке 110 ЖК-экрана ST7735, а при высоте графика 100 в строке 110 отображается точка, соответствующая *току аккумулятора* – 100 × (*текущее/максимальное значение*). Например, если максимальное значение оси тока равно 60 мА, то выходной ток аккумулятора, равный 18 мА, отображается в строке 80 = 110 – 100 × 18/60. Функция `graph` отображает границу графика с помощью библиотечных функций `fillRect` и `drawRect`,

вычисляет и отображает метки по оси Y для тока и напряжения аккумулятора, а также выводит пять пунктирных линий поперек графика, чтобы облегчить его интерпретацию. Процедура обработки прерывания `scale` изменяет максимальное значение оси тока аккумулятора на графике.

С двумя модулями INA219 для различения модулей требуются два адреса I2C. Адрес I2C по умолчанию 0x40 используется модулем солнечной панели INA219, а модуль аккумулятора INA219 имеет адрес I2C 0x41, заданный инструкцией `Adafruit_INA219 inaBatt(0x41)` при подключенной перемычке модуля A0. ЖК-экран ST7735 взаимодействует с контроллером через последовательный интерфейс SPI, хотя вывод MOSI ЖК-экрана ST7735 обозначен как SDA. Библиотека *Adafruit ST7735* ссылается на библиотеку *Adafruit_GFX*, поэтому инструкция `#include <Adafruit_GFX. h>` не требуется. Ориентация экрана TFT LCD ST7735 устанавливается на книжную или альбомную ориентацию с помощью команды `setRotation(N)` со значением 0 или 1 соответственно или со значениями 2 или 3 для переворота изображения на 180° в портретной или альбомной ориентации соответственно.

Листинг 18-8. Измеритель для солнечной панели и аккумулятора

```
#include <Adafruit_INA219.h>          // define Adafruit INA219 library
Adafruit_INA219 inaSolar;             // associate inaSolar and inaBatt with lib
Adafruit_INA219 inaBatt(0x41);        // I2C address 0x41, 0x40 is default
#include <Adafruit_ST7735.h>          // include the ST7735 library
int TFT_CS = D8;                      // ST7735 screen chip select pin
int DCpin = D3;                       // ST7725 screen DC pin
int RSTpin = D4;                      // ST7735 screen reset pin
                                      // associate tft with Adafruit_ST7735 lib
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, DCpin, RSTpin);
int batVPin = A0;                     // battery voltage reading pin
int scalePin = D6;                    // scale pin for interrupt
unsigned int BLACK = 0x0000;
unsigned int YELLOW = 0xFFE0;
unsigned int GREEN = 0x07E0;          // HEX codes for colors
unsigned int RED = 0xF800;
unsigned int WHITE = 0xFFFF;
int maxRead = 99;                     // maximum readings per screen
int n = 0;                            // reading number
int maxVal = 0;                       // initial maximum and minimum
int minVal = 1000;                    // for battery output
int initialY = 60;                    // initial maximum of Y axis
int alterY = 300;                     // alternative maximum of Y axis
volatile int maxY;                    // actual maximum Y axis
float maxV = 4.80;                    // maximum and minimum value
float minV = 2.40;                    // for battery voltage
float labelV = (maxV-minV)/6.0;       // 6 labels on battery voltage axis
int delayTime = 5;                    // delay (s) between readings
float energy = 0;                     // cumulative energy output
float batt, solar;                    // battery and solar current
float battV;                          // battery voltage
float battS, solarS, battVS;          // scaled values for graph
int labelY, newmaxY, batDirect;
void setup()
{
```

```
    inaSolar.begin();                          // initialise INA219 modules
    inaBatt.begin();
    inaSolar.setCalibration_16V_400 mA();      // 16V, 400mA range
    inaBatt.setCalibration_16V_400mA();
    tft.initR(INITR_BLACKTAB);                 // initialise ST7735 screen
    tft.fillScreen(BLACK);                     // clear screen
    tft.setRotation(3);                        // orientate ST7735 screen
    tft.drawRect(0,0,160,128,WHITE);           // draw white frame line
    tft.drawRect(1,1,158,126,WHITE);           // and second frame line
    maxY = initialY;                           // set graph axes
    newmaxY = initialY;
    graph();                                   // call graph function
    pinMode(scalePin, INPUT_PULLUP);           // set pin state HIGH
    attachInterrupt(digitalPinToInterrupt(scalePin), scale,
    FALLING);
}
void loop()
{
    if(newmaxY != maxY)                        // change graph Y axis
    {
      newmaxY = maxY;
      graph();                                 // call graph function
      n = 0;                                   // set number of readings to zero
    }
    n++;                                       // increment reading number
    if(n > maxRead)
    {                                          // new screen if n > maximum
      graph();
      n = 0;
    }
    batt = inaBatt.getCurrent_mA();            // battery current output
    if(batt < minVal) minVal = batt;           // update minimum and
    if(batt > maxVal) maxVal = batt;           // maximum battery current
    screenVal(WHITE, minVal, 5, "min");        // display minimum and
    screenVal(WHITE, maxVal, 85, "max");       // maximum battery current
    solar = inaSolar.getCurrent_mA();          // solar panel current reading
    solarS = 110-100.0*solar/maxY;             // scale solar panel reading
    tft.fillCircle(25+n, solarS, 1, W HITE);
                                               // plot scaled solar panel reading
    if(batt < 0)
    {                                          // when battery charging
      screenVal(RED, batt, 35, "");            // change line color to RED
      battS = 110+100.0*batt/maxY;             // scale battery reading
      tft.fillCircle(25+n, battS, 1, RED );
    }                                          // plot scaled battery reading
    else
    {                                          // when battery discharging
      screenVal(GREEN, batt, 35, "");          // change line color to GREEN
      battS = 110-100.0*batt/maxY;
      tft.fillCircle(25+n, battS, 1, GREEN);
    }                                          // double battery reading
    battV = 2.0*analogRead(batVPin);           // due to voltage divider
    battV = 3.2 * battV /1024.0;               // battery voltage
    screenVal(YELLOW, battV*100, 105, "");
                                               // change color to YELLOW
    battVS = 110-100.0*(battV-minV)/ (m axV-minV);
                                               // scale battery voltage
```

```
    tft.fillCircle(25+n, battVS, 1, YE LLOW);
                                        // plot scaled battery voltage
    detail();                           // function for solar and energy
    delay(delayTime * 1000);            // delay (s) between readings
  }
  void graph()                          // function to draw graph
  {
    labelY = maxY/6;                    // 7 labels on each Y axis
    tft.fillRect(2,2,156,124,BLACK);    // fill screen in BLACK
    tft.drawRect(25,10,100,100,GREEN); // graph rectangle in GREEN
    for (int i=0; i<7; i++)             // label y axis
    {
      tft.setCursor(5,10+i*(100/6-1)); // position labels
      tft.setTextColor(WHITE, BLACK);
      tft.print(maxY-i*labelY);         // left side Y axis value
      tft.setCursor(130,10+i*(100/6-1));
      tft.setTextColor(YELLOW, BLACK);
      tft.print(maxV-i*labelV,2);       // right side Y axis value
    }
    for (int j=0; j<5; j++)             // draw 5 dashed lines on graph
    for (int i=0; i<33; i++) tft.drawPixel(25+3*i, 28+j*100/6,
    YELLOW);
    tft.setTextColor(WHITE, BLACK);
    tft.setCursor(30, 15);              // headings for solar and energy
    tft.print("sol");
    tft.setCursor(75, 15);
    tft.print("mAh");
  }
  void screenVal(unsigned int color, int val, int x, String text)
  {                            // function to display text and value below graph
    tft.setTextSize(1);
    tft.setTextColor(color, BLACK);
    tft.setCursor(x, 115);              // row number 115
    tft.print(text);
    tft.setCursor(x + 25, 115);         // position in row
    if(x == 5) tft.fillRect(30,115,20,8, BLACK);
                                        // over-write previous value
    else if(x == 35) tft.fillRect(60,115,20,8,BLACK);
    else if(x == 85) tft.fillRect(110,115,20,8,BLACK);
    tft.setTextColor(color, BLACK);
    if(x != 105) tft.print(val);        // print new value
    else tft.print(val/100.0,2);        // 2DP for battery voltage
    }
    IRAM_ATTR void scale()              // ISR to change Y axis scale
    {
    if(newmaxY == alterY) maxY = initialY;
    else maxY = alterY;
  }
  void detail()
  {                                     // battery energy output
    energy = energy + battV * delayTime * batt /3600.0;
                                        // energy (mAh)
    tft.setTextColor(WHITE, BLACK);
    tft.fillRect(50,15,15,8,BLACK);     // overlay if fewer digits
    tft.setCursor(50,15);
    tft.print(solar,0);                 // solar panel current output
```

```
    tft.fillRect(100,15,20,8,BLACK);
    tft.setCursor(100,15);
    tft.print(energy,0);                        // battery current output
}
```

Измеритель индуктивности

Резисторно-конденсаторные (*RC*) фильтры блокируют определенные частоты сигнала, при этом фильтры нижних и верхних частот блокируют высокие и низкие частоты соответственно. Для фильтра нижних частот конденсатор отводит высокие частоты на землю (GND), а низкие частоты оказываются доступными на выходе VOUT (см. рис. 18-20). В фильтре высоких частот низкие частоты блокируются конденсатором, а более высокие частоты оказываются доступны на VOUT. Индуктивно-конденсаторный (*LC*) фильтр также является фильтром нижних частот, поскольку катушка индуктивности блокирует высокие частоты, а конденсатор передает высокие частоты на GND, что приводит к низким частотам на VOUT.

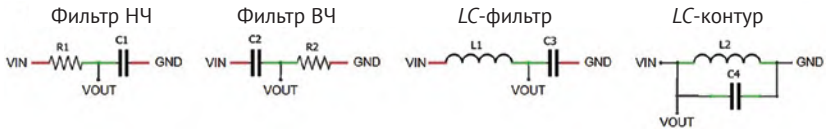


Рисунок 18-20. Схемы фильтров и LC-контур

Катушки индуктивности, обозначенные L в честь Эмилия Ленца (Emil Lenz), используются в индуктивно-конденсаторной схеме *LC*-контура (см. рис. 18-20). Она служит либо для генерации сигналов с определенной частотой, либо для фильтрации сигнала для выделения определенной частоты. Конденсатор накапливает энергию электрического поля на своих пластинах, в то время как катушка индуктивности накапливает энергию в виде магнитного поля. Когда катушка индуктивности подключена к заряженному конденсатору, напряжение на конденсаторе генерирует ток через катушку индуктивности и увеличивает ее магнитное поле. Когда напряжение на конденсаторе падает до нуля, энергия, запасенная в катушке индуктивности, индуцирует напряжение на катушке индуктивности и ток через контур, который снова заряжает конденсатор. Цикл разряда конденсатора – накопления энергии катушки индуктивности и снижения энергии катушки индуктивности – заряда конденсатора повторяется с частотой $(2\pi\sqrt{LC})^{-1}$ Гц.

Когда на *LC*-контур подается импульс напряжения, цепь начинает резонировать с определенной частотой, но из-за внутреннего сопротивления компонентов колебания со временем затухают (см. рис. 18-21). При подключении к выходу *LC*-контура компаратора напряжения схема выдает серию прямоугольных импульсов с частотой, заданной параметрами *LC*. Таким образом можно определить индуктивность неизвестной катушки по частоте импульсов и известному значению конденсатора. На практике измеряется ширина прямоугольного импульса; так как частота равна $(2 \times \textit{ширина импульса})^{-1}$ Гц, то индуктивность выводится непосредственно из ширины импульса, равной

$\pi\sqrt{LC}$. Единица измерения индуктивности – генри (Гн), в честь Джозефа Генри (Joseph Henry), пионера исследований явления индуктивности.

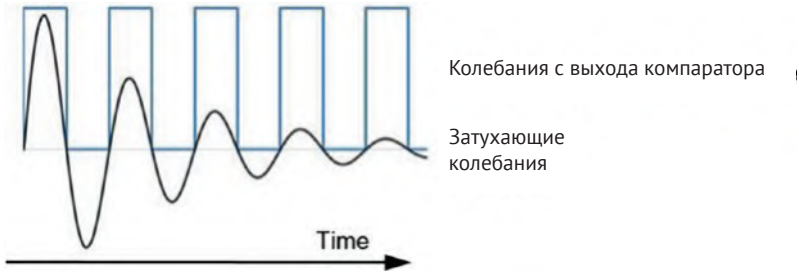


Рисунок 18-21. Колебания LC-контура

Выходное напряжение компаратора напряжения равно напряжению питания, когда напряжение на положительном (+) входном выводе больше, чем напряжение на отрицательном (–). Компаратор напряжения LM393 превращает затухающие колебания LC-контура в прямоугольные импульсы с той же частотой. Выходной вывод компаратора LM393 подтягивается к высокому уровню с помощью резистора 1 кОм между выходом LM393 и выводом VCC. Схема и соединения показаны на рис. 18-22 и в табл. 18-9 соответственно.

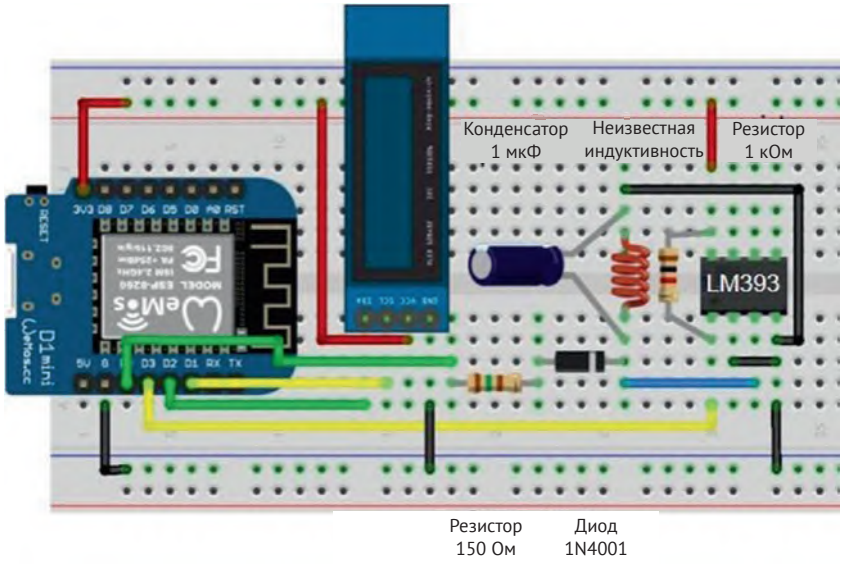


Рисунок 18-22. Измеритель индуктивности с компаратором напряжения LM393

Таблица 18-9. Измеритель индуктивности с компаратором напряжения LM393

Компонент	Подключено к	Также к
LM393 выход (выв. 1)	ESP8266 D3	Резистор 1 кОм
LM393 вход – (выв. 2)	ESP8266 GND	
LM393 вход + (выв. 3)	Индуктивность	
LM393 вход + (выв. 3)	Катод диода (с полосой)	Конденсатор (+)
LM393 GND (выв. 4)	ESP8266 GND	
LM393 VCC (выв. 8)	ESP8266 3V3	Резистор 1 кОм
ESP8266 D0	Резистор 150 Ом	
Анод диода	Резистор 150 Ом	
ESP8266 GND	Индуктивность	Конденсатор (–)
OLED VCC	3V3	
OLED GND	GND	
OLED SDA	D2	
OLED SCL	D1	

В скетче в листинге 18-9 на *LC*-контур подается импульс напряжения, и продолжительность высокого уровня, полученного с помощью компаратора LM393, измеряется с помощью команды `pulse = pulseIn(LM393pin, HIGH, timeout)`. Если сигнал имеет высокий уровень, то функция `pulseIn` ожидает сброса в низкий уровень, затем начинает отсчет времени, когда сигнал опять установится в высокий уровень, и прекращает отсчет, когда сигнал снова сбросится до низкого уровня. Если перепад из низкого в высокий не обнаруживается до истечения `timeout` микросекунд, то возвращается нулевое значение. Частота измеренного прямоугольного колебания составляет $10^6 \times (2 \times \textit{ширина импульса})^{-1}$ Гц, так как ширина импульса измеряется в микросекундах, а значение индуктивности представляет собой $\frac{\textit{ширина импульса}^2}{\pi^2 \times \textit{емкость}}$. Измеритель индуктивности достаточно точен для индуктивностей 50 мкГн и выше. Например, результат измерения для индуктивности 470 мкГн с электролитическим или керамическим конденсатором емкостью 1 мкФ составлял 462 мкГн и 415 мкГн соответственно. Длительность импульса с керамическим конденсатором обычно на 2–3 мкс меньше, чем с электролитическим конденсатором (см. рис. 18-23).

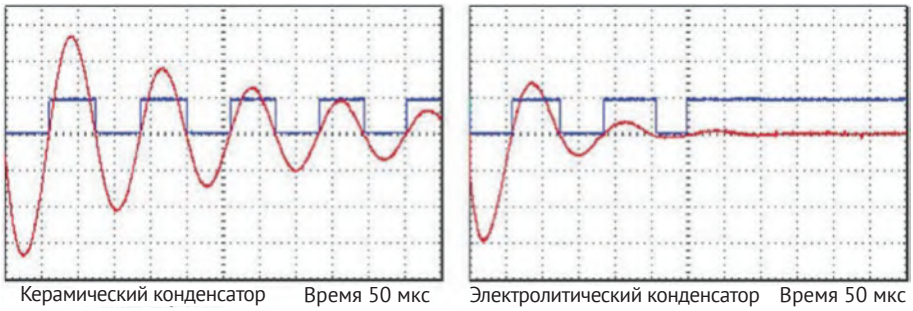


Рисунок 18-23. Колебания в LC-контуре с керамическим и электролитическим конденсатором

Контур с керамическим конденсатором, установленным параллельно катушке индуктивности, дольше генерирует колебания, чем с электролитическим (см. рис. 18-23). Электролитический конденсатор менее эффективен на высоких частотах, чем керамический конденсатор, из-за более высокого эквивалентного последовательного сопротивления (ESR) и паразитной индуктивности (ESL), что приводит к более длительному времени его зарядки и разрядки. Например, длительность импульса конденсатора емкостью 1 мкФ в паре с индуктивностью 470 мкГн составляет в течение всего времени колебаний 64 мкс, если конденсатор керамический, но с электролитическим конденсатором фиксируются только первые два импульса длительностью 65 мкс и 70 мкс (см. рис. 18-23). При индуктивности менее 50 мкГн при использовании электролитического конденсатора импульсы вообще не обнаруживаются, учитывая к тому же высокую частоту колебаний 25 кГц.

Листинг 18-9. Измеритель индуктивности

```
#include <Adafruit_SSD1306.h>          // Adafruit SSD1306 library
int width = 128;                       // OLED screen size
int height = 32;                       // associate oled with SSD1306
Adafruit_SSD1306 oled(width, height, &Wire, -1);
int voltPin = D0;                      // voltage burst pin
int LM393pin = D3;                     // LM393 output pin
float capacitor = 1.0;                 // measured in µF
int timeout = 1000;                    // timeout limit in µs
float pulse, pulse1, pulse2, frequency, inductance;
void setup()
   {
   pinMode(voltPin, OUTPUT);           // define voltPin as output
   oled.begin(SSD1306_SWITCHCAPVCC, 0x 3C); // OLED I2C address
   oled.clearDisplay();                // clear OLED display
   oled.setTextColor(WHITE);           // set font color
   oled.setTextSize(1);                // set font size (1, 2, 3 or 4)
   oled.display();                     // update display instructions
}
void loop()
{
   digitalWrite(voltPin, HIGH);        // apply 3.2V on voltPin
   delay(5);                           // time to charge the inductor
   digitalWrite(voltPin,LOW);          // end of voltage burst
```

```
    pulse1 = pulseIn(LM393pin, HIGH, timeout);
                                        // measure HIGH pulse duration
    pulse2 = pulseIn(LM393pin, HIGH, timeout);
    if(pulse2 > 0) pulse = (pulse1+p ulse2)/2.0;
                                        // average pulse length
    else pulse = pulse1;
    if(pulse > 0)
    {
    frequency = 1E6/(2.0*pulse);        // shorthand for 10 to the power 6
    inductance = pulse*pulse/(PI*P I*capacitor);
                                        // calculate inductance
    oled.clearDisplay();
    oled.setCursor(0,0);                // display results
    oled.print("inductance uH ");
    oled.println(inductance,0);
    oled.print("frequency Hz ");oled.println(frequency,0);
    oled.print("high time us ");oled.print(pulse,0);
    oled.display();
    }
    delay(1000);
}
```

Итоги

Напряжение на нагрузке, сопротивление нагрузки и ток через нагрузку измерялись с помощью аналого-цифрового преобразователя (АЦП) микроконтроллеров ESP8366 или ESP32. Напряжение, сопротивление и ток определялись по напряжению на нагрузке, а также по напряжению на шунтирующем резисторе для получения тока через нагрузку. Емкость конденсатора определялась по его времени зарядки. Модуль измерения тока и напряжения INA219 количественно определяет мощность и количество энергии, использованной нагрузкой. Чистый ток заряда или разряда аккумулятора, обеспечивающий питание нагрузки при зарядке от солнечной панели, был измерен с помощью двух INA219 модулей с отображением изменения информации во времени на ЖК-дисплее ST7735. Индуктивность измерялась с помощью подачи импульса напряжения на *LC*-контур с измерением частоты резонанса. Затухающая синусоида с *LC*-контура преобразовывалась в прямоугольное колебание с помощью компаратора напряжения, а индуктивность была получена из измерения частоты прямоугольных импульсов.

Перечень компонентов

- ESP8266 микроконтроллер: плата LOLIN (WeMos) D1 mini или NodeMCU
- ESP32 микроконтроллер: плата DEVKIT DOIT или NodeMCU
- OLED-дисплей: 128 × 32 пиксела
- Внешний источник: 5 V или 9 V
- Резисторы: 150 Ом, 220 Ом, 1 кОм (2 шт.), 2,2 кОм, 10 кОм, 22 кОм и 47 кОм
- Светодиод
- Шунт 0,01 Ом
- Операционный усилитель LM358

- Датчик тока: INA219 2 шт.
- Солнечная панель с выходом 5 В
- Литиевый аккумулятор: 3,7 V
- Зарядный модуль для литиевых аккумуляторов: TP4056
- DC-DC повышающий преобразователь MT3608
- TFT LCD-дисплей ST7735, 1,8 дюйма, 128 × 160 пикселов
- Тактовая кнопка
- Компаратор LM393
- Конденсаторы 0,1 мкФ, 1 мкФ (керамический или электролитический)
- Диод IN4001

Глава 19

Поворотный энкодер

Поворотные энкодеры обнаруживают вращение центрального вала и используются для контроля положения в механических конструкциях или частоты вращения двигателя, управления громкостью звука, положения курсора на ЖК-дисплее или просто яркостью светодиода[121]. Например, в главе 1 («Интернет-радио») радиостанция и громкость выбирались поворотом ротора энкодера. Инкрементный поворотный энкодер имеет 20 положений, и ротор должен вращаться по часовой стрелке или против часовой стрелки для увеличения или уменьшения управляющей переменной. Энкодер имеет два внешних контакта, называемых A или CLK (тактовые импульсы) и B или DT (данные), а также общий контакт внутри, подключенный к высокому уровню. Поворот на одно положение вызывает переключение уровней на выходах A и B. Когда вращается ротор поворотного энкодера, контакты A и B каждый входят в контакт с общим контактом, но поскольку контакты смещены, импульсы не совпадают по фазе на 90° (см. рис. 19-1). Количество прямоугольных импульсов при повороте указывает величину поворота, которая измеряется либо на выводе A, либо на выводе B.

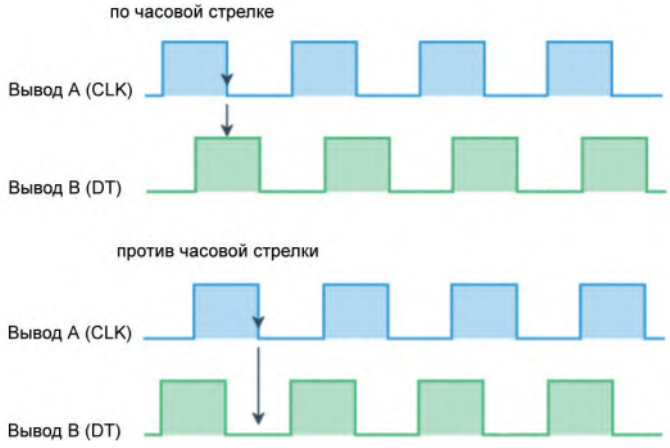


Рисунок 19-1 . Прямоугольные импульсы с поворотного энкодера

[121] Заметим, что для решения многих из перечисленных задач (например, измерения частоты вращения двигателя) простейший энкодер с механическими контактами, описанный в этой главе, не подойдет, для них выпускаются более надежные (и дорогие) конструкции. – *Прим. перев.*

Когда вал поворачивается по часовой стрелке, сначала подключается контакт A, а затем контакт B. Когда ротор поворачивается против часовой стрелки, сначала подключается контакт B, а потом контакт A. Отсюда можно установить направление вращения: при вращении по часовой стрелке падающий фронт импульса на выводе A совпадает с высоким уровнем на выводе B, и наоборот – при вращении против часовой стрелки падающий фронт импульса на выводе A совпадает с низким уровнем на выводе B.

Скетч в листинге 19-1 представляет поворотный энкодер, генерирующий прямоугольные импульсы для управления двумя светодиодами (см. рис. 19-2). Контролируется состояние вывода A, и при обнаружении падающего фронта считывается состояние вывода B. Функция `LED` включает красный или зеленый светодиод, отображает направление вращения и подсчитывает количество приращений положений ротора энкодера. Обратите внимание, что для определения направления вращения вал должен быть повернут не менее, чем на два положения. Для листинга 19-1 пары резисторов и конденсаторов на рис. 19-2 не подключаются.

Чтобы проиллюстрировать, что решение для одного микроконтроллера не обязательно подходит для микроконтроллера с более высокой частотой процессора, скетч в листинге 19-1 выполняется как на Arduino Nano, так и на микроконтроллере ESP8266. С микроконтроллером Arduino Nano ATmega328P количество и направление поворотов поворотного энкодера определяются достаточно правильно, при условии что поворотный энкодер не поворачивается слишком быстро. Напротив, появляются существенные ошибки при обнаружении поворота поворотного энкодера микроконтроллером ESP8266, который имеет более высокую частоту процессора, 160 МГц по сравнению с 16 МГц микроконтроллера ATmega328P. Обнаружение поворота поворотного энкодера хуже с микроконтроллером ESP32, который имеет частоту процессора 240 МГц, так как обнаруживается каждое нарушение контакта при дребезге во время замыкания и размыкания.

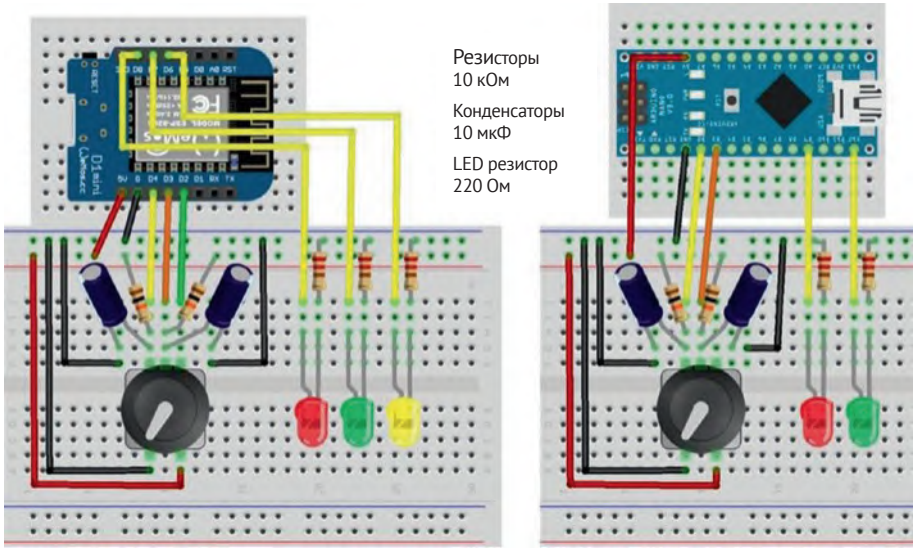


Рисунок 19-2. Поворотный энкодер и светодиоды с платой LOLIN (WeMos) D1 mini и Arduino Nano

Соединения поворотного энкодера и светодиода приведены в табл. 19-1 и показаны на рис. 19-2. Для скетчей 19-1, 19-2 и 19-3 требуются два светодиода, в то время как для скетча 19-5 потребуются три светодиода, поэтому все три светодиода включены в рис. 19-2, чтобы его не повторять.

Выводы GPIO D3 и D4 нельзя устанавливать на низкий уровень, поэтому при загрузке скетча в микроконтроллер ESP8266 они отсоединяются от контактов поворотного энкодера DT и CLK.

Таблица 19-1. Поворотный энкодер и светодиоды

Компонент	Подключено к	Также к	Также к
Энкодер CLK	ESP8266 D4 или Nano выв. 2	Конденсатор плюс	
Энкодер CLK		Резистор 10 кОм	ESP8266 или Nano 5V
Энкодер DT	ESP8266 D3 или Nano выв. 3	Конденсатор плюс	
Энкодер DT		Резистор 10 кОм	ESP8266 или Nano 5V
Энкодер SW	ESP8266 D2		
Энкодер VCC	ESP8266 или Nano 5V		
Энкодер GND	ESP8266 или Nano GND		
Конденсаторы минус	ESP8266 или Nano GND		
Светодиоды длинные выводы	ESP8266 D8, D7, D6, или Nano выв. 9 и 12		
Светодиоды короткие выводы	Резисторы 220 Ом	ESP8266 или Nano GND	

Листинг 19-1. Поворотный энкодер со светодиодами

```
int CLKpin = D4;                    // rotary encoder pins
int DTpin = D3;                     // CLK = pin A and DT = pin B
int redLED = D8;
int greenLED = D7;                  // define LED pins
int count = 0;
int oldCLK = LOW;
int newCLK;
void setup()
{
   Serial.begin(115200);            // Serial Monitor baud rate
   pinMode(redLED, OUTPUT);         // set LED pins as output
   pinMode(greenLED, OUTPUT);
}
void loop()
{
   newCLK = digitalRead(CLKpin);    // state of pin A (CLK)
   if(newCLK == LOW && oldCLK == HIGH) // falling edge of pin A
   {                                // pin B HIGH
```

```
      if(digitalRead(DTpin) == HIGH) LED (HIGH, 1, "up");
      else LED(LOW, -1, "down ");      // pin B (DT) LOW
    }
    oldCLK = newCLK;                   // reset pin A (CLK) state
// delay(100);                         // to confirm microcontroller
}                                      // can miss rotary encoder turns
void LED(int state, int increment, String text)
{
    digitalWrite(redLED, 1-state);     // turn on or off the LEDs
    digitalWrite(greenLED, state);
    Serial.print(text);
    count = count + increment;         // update and display count
    Serial.println(count);
}
```

Устранение дребезга контактов

Со скетчем в листинге 19-1 связаны две проблемы. Во-первых, дребезг при контакте выводов A и B поворотного энкодера с общим выводом приводит к ложным изменениям обнаруженного направления и количества приращений энкодера. Аппаратным решением для устранения этого дребезга является включение подтягивающих резисторов 10 кОм (R) и конденсаторов 10 мкФ (C) между выводами A, B и выводом GND (см. рис. 19-2 и табл. 19-1). Такая *RC*-цепь должна создавать задержку отключения 69 мс, равную $RC \times \ln(2)$ секунд. Дополнительная информация о типе конденсатора и расчете времени разрядки приведена в главе 17 («Генерация сигнала с помощью микросхемы таймера 555»). Некоторые модули поворотного энкодера, такие как модуль KY-040, уже включают подтягивающие резисторы 10 кОм на контактах CLK и DT, поэтому для устранения дребезга требуются только конденсаторы емкостью 10 мкФ.

Постоянная времени *RC*-цепи для устранения дребезга контактов выводов A и B поворотного энкодера должна гарантировать, что прошло достаточно времени для прекращения дребезга контактов, но не слишком много, чтобы не пропустить поворот энкодера. Задержка в 69 мс, достигнутая с помощью резистора 10 кОм и конденсатора 10 мкФ, является компромиссом.

Прерывания

Вторая проблема со скетчем в листинге 19-1 заключается в том, что микроконтроллер непрерывно считывает состояние вывода A для обнаружения падающего фронта, а затем считывает состояние вывода B для определения направления вращения. Если скетч включает другие действия, то микроконтроллер пропустит обнаружение падающих фронтов на выводе. Включение короткой задержки в 100 мс в функцию `loop` из листинга 19-1 иллюстрирует этот момент. Проблема пропуска перепада уровней решается путем включения прерывания (см. листинг 19-2). Прерывание настроено на падающий фронт на выводе A (CLK), и процедура обработки прерывания (ISR) считывает состояние вывода B (DT) для определения направления вращения. Изменение переменной происходит как в функции `loop`, так и в ISR, поэтому она определяется как `volatile`. В листинге 19-2 детализированы только инструкции, относящиеся к прерыванию, чтобы подчеркнуть различия между листингами 19-1 и 19-2.

Листинг 19-2 предназначен для микроконтроллера ESP8266. При использовании микроконтроллера ESP32 контакты поворотного энкодера определяются инструкциями

```
pinMode(CLKpin, INPUT);
pinMode(DTpin, INPUT);
```

В листинге 19-2 обработчик прерывания определяется с помощью команды `IRAM_ATTR void isr()` для микроконтроллера ESP8266 или ESP32 для сохранения прерывания во внутренней оперативной памяти (IRAM) вместо более медленной флеш-памяти. Для Arduino Nano обработчик прерывания определяется как `void isr()`.

Листинг 19-2. Поворотный энкодер и прерывание

```
int CLKpin = D4;
int DTpin = D3;
int redLED = D8;
int greenLED = D7;
int count = 0;
volatile int change;                          // variable used in ISR
void setup()
{
   Serial.begin(115200);
   pinMode(redLED, OUTPUT);
   pinMode(greenLED, OUTPUT);                 // attached interrupt
   attachInterrupt(digitalPinToInterrupt(CLKpin), isr, FALLING);
}
void loop()
{
   if(change != 0)                            // rotary encoder direction
   {
      if(change > 0) LED(HIGH, "up ");        // clockwise
      else if(change < 0) LED(LOW, "down ");  // anti-clockwise
      change = 0;
   }
}
void LED(int state, String text)              // no change to LED function
{
   digitalWrite(redLED, 1-state);
   digitalWrite(greenLED, state);
   Serial.print(text);
   count = count + change;
   Serial.println(count);
}
IRAM_ATTR void isr()                          // interrupt service routine
{
   change = 2*digitalRead(DTpin) - 1 ;
}                                            // determine direction of rotation
```

Включение прерывания для обнаружения изменений в состоянии вывода A и подключение *RC*-цепей для устранения дребезга контактов выводов энкодера улучшают обнаружение поворотов энкодера для микроконтроллера ATmega328P, но не существенны для микроконтроллеров ESP8266 и ESP32. В следующем разделе описывается более эффективное решение для устранения дребезга контактов энкодера.

Подсчет состояний[122]

Дребезг может быть полностью устранен путем подсчета изменений в состояниях вывода, согласованных с вращением по часовой стрелке или против часовой стрелки. Состояния выводов A и B легко контролировать визуально, если подключить по светодиоду к этим выводам (см. рис. 19-3 и соединения в табл. 19-2).

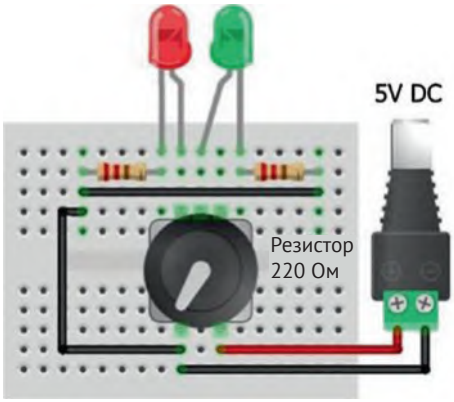


Рисунок 19-3. Поворотный энкодер и тестовые светодиоды

Таблица 19-2. Поворотный энкодер и тестовые светодиоды

Компонент	Подключено к	Также к
Энкодер A (CLK)	Красный светодиод длинный вывод	
Энкодер B (DT)	Зеленый светодиод длинный вывод	
Энкодер VCC	Внешний источник 5 В	
Энкодер GND	Внешний источник GND	
Светодиоды короткие выводы	Резисторы 220 Ом	Внешний источник GND

Когда инкрементный поворотный энкодер медленно вращается по часовой стрелке, сначала загорается светодиод, подключенный к выводу A; перед завершением прохождения очередного шага загорается светодиод, подключенный к выводу B (см. рис. 19-1). Если энкодер снова медленно поворачивается на один шаг, два светодиода гаснут, причем светодиод, подключенный к вы-

[122] В этом разделе речь пойдет об устранении дребезга контактов достаточно сложным путем. Читатель должен иметь в виду, что способы устранения дребезга, описанные далее, не являются единственно возможными. Примером альтернативного способа может служить Arduino-библиотека *Bounce*. Большинство таких способов аналогичны по принципу работы способу задержки с *RC*-цепочкой, описанному автором ранее, только реализуются программно и потому гораздо более надежны, не требуя дополнительных компонентов, и поддаются настройке в достаточно широких пределах. – *Прим. перев.*

воду A, выключается первым. Последовательность состояний светодиодов, записанная в виде двух бит LEDA и LEDB, будет 00, 10, 11, 01 и обратно к 00 (см. рис. 19-1). Последовательность, состоящая из 2 бит с 1 битом для каждого вывода, известная как код Грея, в честь Фрэнка Грея (Frank Gray), представляет последовательные значения, отличающиеся друг от друга всегда ровно на 1 бит. Аналогичная 2-разрядная последовательность, возникающая при медленном вращении энкодера против часовой стрелки, выглядит как 00, 01, 11, 10 и обратно до 00.

Изменения состояний выводов A и B записываются в виде 4-разрядных чисел, описывающих эти изменения по мере вращения. Например, если вращение происходит по часовой стрелке, состояния выводов A и B последовательно принимают значения `LOW, LOW`, затем `HIGH, LOW`, потом `HIGH, HIGH`, затем `LOW, HIGH` и наконец `LOW, LOW`. Записанные в виде 4-разрядных чисел, последовательности выглядят как (00)(10), (10)(11), (11)(01), и (01)(00) (см. табл. 19-3).

Таблица 19-3. Последовательность битов состояний поворотного энкодера

Начальное состояние	Следующее состояние	4-битное число
LOW, LOW	HIGH, LOW	0010
HIGH, LOW	HIGH, HIGH	1011
HIGH, HIGH	LOW, HIGH	1101
LOW, HIGH	LOW, LOW	0100

4-разрядное число имеет 16 возможных значений, и каждое значение сопоставляется с приращением направления поворотного энкодера (см. табл. 19-4). 4-разрядные числа с нулевой оценкой (*score*) представляют собой либо отсутствие изменений состояния, либо изменения состояния двух бит одновременно, как, например, 0101 или 0011.

Таблица 19-4. Состояния поворотного энкодера в виде 4-разрядных чисел

4-бит. число	Десятичное	Выв. A (CLK) и выв. B (DT)		Изменения	Направление	Оценка
		предыдущее	текущее			
0000	0	LOW, LOW	LOW, LOW	нет		0
0001	1	LOW, LOW	LOW, HIGH		Против час. стрелки	–1
0010	2	LOW, LOW	HIGH, LOW		По часовой стрелке	1
0011	3	LOW, LOW	HIGH, HIGH	2 вывода		0
0100	4	LOW, HIGH	LOW, LOW		По час. стрелке	1
0101	5	LOW, HIGH	LOW, HIGH	Нет		0
0110	6	LOW, HIGH	HIGH, LOW	2 вывода		0
0111	7	LOW, HIGH	HIGH, HIGH		Против час. стрелки	–1
1000	8	HIGH, LOW	LOW, LOW		Против час. стрелки	–1

Окончание табл. 19-4

4-бит. число	Десятичное	Выв. A (CLK) и выв. B (DT)		Изменения	Направление	Оценка
		предыдущее	текущее			
1001	9	HIGH, LOW	LOW, HIGH	2 вывода		0
1010	10	HIGH, LOW	HIGH, LOW	Нет		0
1011	11	HIGH, LOW	HIGH, HIGH		По час. стрелке	1
1100	12	HIGH, HIGH	LOW, LOW	2 вывода		0
1101	13	HIGH, HIGH	LOW, HIGH		По час. стрелке	1
1110	14	HIGH, HIGH	HIGH, LOW		Против час. стрелки	–1
1111	15	HIGH, HIGH	HIGH, HIGH	Нет		0

Оба состояния должны определяться всякий раз, когда изменяется одно из состояний на выводах, поэтому требуется два прерывания, по одному для каждого из выводов энкодера A и B. Два состояния преобразуются в 4-разрядное число, состоящее из предыдущего и текущего состояний, для получения соответствующей оценки. Когда суммарная оценка равна +2 или –2, то поворотный энкодер повернулся по часовой стрелке или против часовой стрелки на один шаг, так как вращение энкодера на один шаг приводит к двум изменениям состояния. Если требуется более точное определение вращения поворотного энкодера, то требуется вращение на два шага с получением оценки +4 или –4.

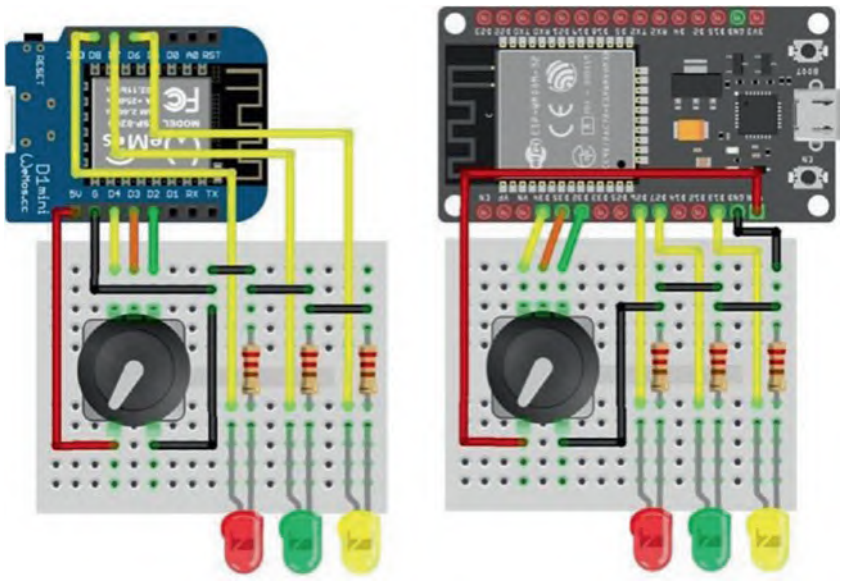


Рисунок 19-4. Поворотный энкодер и светодиоды с платами LOLIN (WeMos) D1 mini и ESP32 DEVKIT DOIT

Таблица 19-5. Поворотный энкодер и светодиоды с микроконтроллерами ESP8266 и ESP32

Компонент	Подключено к ESP8266	Подключено к ESP32
Энкодер CLK	D4	GPIO 34
Энкодер DT	D3	GPIO 35
Энкодер SW	D2	GPIO 32
Энкодер VCC	5V	VIN
Энкодер GND	GND	GND
Светодиоды длинные выводы	D8, D7, D6	GPIO 26, GPIO 27, GPIO 13
Светодиоды короткие выводы	Резисторы 220 Ом на GND	Резисторы 220 Ом на GND

Обработчик прерывания ISR строит 4-разрядное число из сохраненных предыдущих состояний и текущих состояний выводов A и B, а затем увеличивает счет. Оператор битового сдвига << перемещает бит со значением *N* в положение *r* с помощью команды `(N<<r)`. Для представления текущих состояний 4-разрядное число уменьшается до 2-разрядного числа с помощью сдвига бита на две позиции, что эквивалентно нахождению остатка от деления 4-разрядного числа на четыре при выполнении команды `oldState = newState % 4`. Например, если энкодер перемещается по часовой стрелке из начального состояния `HIGH, HIGH` в состояние `LOW, LOW`, 2-разрядная числовая последовательность равна 11, 01 и 00, что соответствует двум 4-разрядным числам 1101 и 0100 (см. табл. 19-4). 4-разрядное число для второго этапа вращения 0100 имеет 2-разрядное значение 00, когда бит сдвинут на две позиции, что эквивалентно остатку от 0, когда десятичное значение 8 (4-разрядное значение 0100) делится на 4. В качестве альтернативы команды `oldState = (digitalRead(CLKpin)<<1)+digitalRead (DTpin)` было бы достаточно, но состояния, возможно, изменились с момента активации прерывания.

Новые состояния представляют собой комбинацию предыдущих состояний и текущих состояний вывода A (CLK) и вывода B (DT). Новое состояние состоит из предыдущих состояний для выводов A и B, но сдвинутых на две позиции (<<2), текущего состояния вывода A, сдвинутого на одну позицию (<<1), и текущего состояния вывода B.

Инструкция `switch..case` является альтернативой размещению значений оценок в массиве. Например, инструкция `score = score + vals[newState]` в листинге 19-3 заменяется на

```
switch (newState)
{
  case 0: case 3: case 5: case 6: case 9: case 10: case 12: case 15:
  break;                      // без изменений
  case 1: case 7: case 8: case 14:
  score = score - 1; break;  // reduce score by one
  case 2: case 4: case 11: case 13:
  score++; break;             // increase score by one
  default: break;
}
```

Листинг 19-3. Состояния выводов поворотного энкодера

```
int CLKpin = D4;              // define rotary encoder pins
int DTpin = D3;
int redLED = D8;              // define LED pins
int greenLED = D7;
int vals[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0 };
                              // array of scores
int count = 0;
volatile int score = 0;
volatile int change = 0;      // variables used in isr and loop functions
volatile int oldState = 0;
void setup()
{
  Serial.begin(115200);       // Serial Monitor baud rate
  pinMode(redLED, OUTPUT);    // set LED pins as output
  pinMode(greenLED, OUTPUT); // attach interrupt
  pinMode(CLKpin, INPUT);     // required by ESP32
  pinMode(DTpin, INPUT);      // required by ESP32
  attachInterrupt(digitalPinToInterrupt(CLKpin), isr, CHANGE);
  attachInterrupt(digitalPinToInterrupt(DTpin), isr, CHANGE);
}
void loop()
{
  if(change != 0)                          // rotary encoder rotated
  {
    if(change > 0) LED(HIGH, "up ");       // clockwise
    else if(change < 0) LED(LOW, "down "); // anti-clockwise
    change = 0;
  }
}
void LED(int state, String text)
{
  digitalWrite(redLED, 1-state);     // turn LEDs on or off
  digitalWrite(greenLED, state);
  Serial.print(text);
  count = count + change;            // update count
  Serial.println(count);
}
IRAM_ATTR void isr()                 // interrupt service routine
{                                    // construct 4-bit number
  int newState = (oldState<<2)+(digitalRead(CLKpin)<<1)+
  digitalRead(DTpin);
  score = score + vals[newState];    // allocate score from array
  oldState = newState % 4;
                                     // remainder to leave new CLK and DT state
  if(score == 2 || score == -2)      // 2 steps for complete rotation
  {
    change = score/2;                // unit change per two steps
    score = 0;                       // reset score
  }
}
```

Переключение состояний

Кнопочный переключатель энкодера, срабатывающий при нажатии на вал, используется для изменения состояния двоичной переменной, которую можно применить для включения или выключения светодиода. Контроль времени нажатия дает возможность различать длительное и короткое нажатия, позволяя управлять переменной с тремя значениями, а не только двоичной. В листинге 19-4 вывод микроконтроллера ESP8266 или ESP32, подключенный к контакту энкодера SW, использует внутренний подтягивающий резистор вместо включения отдельного резистора в схему. Внутренний подтягивающий резистор активируется с помощью инструкции `pinMode(pin, INPUT_PULLUP)`, задающей низкий активный уровень вывода. Когда переключатель энкодера нажимается, а затем отпускается, состояние вывода изменяется с высокого уровня на низкий, а затем обратно на высокий.

В листинге 19-4 вывод D2 платы ESP8266 определяется как вывод прерывания, и в зависимости от продолжительности нажатия переменной `changeSW` присваивается символьное значение "*L*", "*S*" или "*B*" для обозначения длительного или короткого нажатия или случайного скачка из-за дребезга контактов. Переменной `changeSW` не присваивается строковое значение "*Long*", "*Short*" или "*Bounce*", поскольку строки не передаются в прерывание. Обработчик прерывания ISR сравнивает промежуток времени от нажатия до отпускания с предопределенными значениями длительного нажатия и короткого нажатия, чтобы определить длительность нажатия. Когда переключатель отпущен, время между нажатием и отпусканием контактов определяет принадлежность события к категории «длительное нажатие» ("*long press*"), «короткое нажатие» ("*short press*") или «дребезг контактов» ("*switch bounce*"), при этом состояние светодиода меняется только после длительного или короткого нажатия.

Листинг 19-4. Переключатель поворотного энкодера

```
int SWpin = D2;              // rotary encoder switch pin
int SWled = D6;              // LED pin
int longPress = 1000;        // time (ms) for long or short press
int shortPress = 500;
volatile unsigned long newTime, oldTime;
volatile char changeSW;      // variables in loop and isr functions
volatile unsigned int lagTime;
void setup()
{
   Serial.begin(115200);             // Serial Monitor baud rate
   pinMode(SWled, OUTPUT);           // set LED pin as output
   pinMode(SWpin, INPUT_PULLUP);     // activate pull-up resistor
   attachInterrupt(digitalPinToInterrupt(SWpin), isr, CHANGE);
}                    // SWpin as interrupt pin and attache ISR
void loop()
{
   if(changeSW != ' ')               // switch released, turn on or off
   {                                 // LED if long or short press
      Serial.print(lagTime);Serial.p rint("\t");
                                     // display switch press time lag
      if(changeSW != 'B') digitalWrite(SWled,
```

```
      !digitalRead(SWled));
      if(changeSW == 'L') Serial.println("long press");
      else if(changeSW == 'S') Serial.println("short press");
      else Serial.println("switch bo unce");
                                    // display only, no effect on LED
      changeSW = ' ';               // reset change variable
    }
  }
  IRAM_ATTR void isr()
  {
    newTime = millis();             // get time ISR triggered
    if(digitalRead(SWpin) == HIGH)  // switch now released
    {
      lagTime = newTime - oldTime;  // time between switch presses
      if(lagTime > longPress) change SW = 'L'; // L for long press
      else if(lagTime > shortPress) changeSW = 'S ';
                                    // S for short press
      else changeSW = 'B';          // B for bounce
    }
    oldTime = newTime;              // reset switch press/release time
  }
```

Увеличение значения

При вычислении количества прошедших прямоугольных импульсов, которое указывает степень поворота энкодера, переключатель можно использовать для управления размером инкрементного счетчика. Например, нажатие переключателя в течение длительного или короткого времени увеличивает изменение счетчика с 1 до 10. Список 19-3 легко расширить, включив инструкции для переключателя энкодера:

```
int SWpin = D2;                     // rotary encoder switchpin
int SWled = D6;                     // LED pin
int longPress = 1000;               // time (ms) for long press
volatile unsigned long newTime, oldTime, lagTime;
volatile int increment = 1;
```

Добавлены инструкции для светодиода, подтягивающего резистора и подключения прерывания с помощью команды `ISR isrSwitch` в функции `setup`:

```
pinMode(SWled, OUTPUT);             // set LED pin as output
pinMode(SWpin, INPUT_PULLUP);       // activate pull-up resistor
attachInterrupt(digitalPinToInterrupt(SWpin), isrSwitch, CHANGE);
```

В функции `LED` изменяется инструкция счета:

```
count = count + change*increment;    // update count
```

и добавляется инструкция в функцию `LED`, указывающая значение приращения:

```
digitalWrite(SWled, increment>1);    // turn on LED if increment >1
```

Инструкции для обработчика прерывания по изменению размера приращения следующие:

```
IRAM_ATTR void isrSwitch()
{
   newTime = millis();                          // get time ISR triggered
   if(digitalRead(SWpin) == HIGH)               // switch released
   {
      lagTime = newTime - oldTime;
      if(lagTime > longPress) increment = 10; // update increment
      else if(lagTime > 100) increment = 1;    // debounce switch
   }
   oldTime = newTime;                           // reset switch press/release time
}
```

Величина инкрементного счета регулируется также скоростью вращения поворотного энкодера, при этом высокая скорость приводит к большему приращению, чем низкая скорость. Обработчик прерывания отображает изменения в состояниях выводов A и B для оценки вращения и вычисляет время вращения из интервала между возрастающим и падающим фронтами импульса. Инструкций в обработчике прерывания должно быть как можно меньше, чтобы свести к минимуму время обработки, и обработчик может быть разделен на две части. Они не выполняются параллельно, одно прерывание начинается после завершения предыдущего прерывания. Разделение обработчика на два отдельных не приводит к большей эффективности, но позволяет одновременно выполнять другие инструкции без риска потери фронтов.

Для управления инкрементным отсчетом с помощью скорости вращения энкодера листинг 19-3 также легко адаптируется. Определения переменных и вывод светодиода указаны в начале скетча:

```
int incLED = D6;      // LED pin indicates when increment > 1
int rotation = 500;   // threshold rotation time (ms)
int increment = 1;
volatile unsigned long newTime, oldTime, lagTime;
volatile int newCLK;  // variables used in isr() and loop()
volatile int oldCLK = 0;
```

Индикатор, указывающий, когда активируется большее приращение, включается в функцию `setup`:

```
pinMode(incLED, OUTPUT);
```

В функции *LED* изменяется инструкция счета:

```
count = count + change*increment;  // update count
```

и добавляется инструкция для светодиода, указывающего величину приращения:

```
digitalWrite(incLED, increment>1); // turn LED on if increment > 1
```

В существующем обработчике, чтобы избежать повторного чтения состояния вывода A (CLK), инструкция

```
int newState = (oldState<<2)+(digitalRead(CLKpin)<<1)+ digitalRead(DTpin);
```

заменяется на инструкции

```
newCLK = digitalRead(CLKpin);
int newState = (oldState<<2)+(newCLK<<1)+digitalRead(DTpin);
```

Инструкции по вычислению скорости вращения и соответствующего приращения добавляются в существующий обработчик прерывания:

```
newTime = millis();                          // get time ISR triggered
if(newCLK == HIGH && oldCLK == LOW)          // interval between falling
{                                            // and rising edge on pin A (CLK)
  lagTime = newTime - oldTime;
  if(lagTime < rotation && lagTime > 10 0) increment = 10;
                                             // fast rotation
  else if(lagTime > rotation) increment = 1; // slow rotation
  oldTime = newTime;                         // reset rising/falling edge time
}
oldCLK = newCLK;                             // reset state of pin A (CLK)
```

В листинг 19-5 включены эти дополнения к листингу 19-3 для определения величины инкремента, определяемого вращением энкодера, с комментариями к дополнительным инструкциям.

Листинг 19-5. Управление инкрементным счетом с помощью скорости вращения поворотного энкодера

```
int CLKpin = D4;
int DTpin = D3;
int redLED = D8;
int greenLED = D7;
int vals[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0};
int count = 0;
volatile int score = 0; volatile int change = 0;
volatile int oldState = 0;
int incLED = D6;              // increment indicator LED pin
int rotation = 500;           // threshold rotation time (ms)
int increment = 1;            // default increment
volatile unsigned long newTime, oldTime, lagTime;
volatile int newCLK;          // variables used in ISR
volatile int oldCLK = 0;
void setup()
{
  Serial.begin(115200);
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  pinMode(CLKpin, INPUT);
  pinMode(DTpin, INPUT);
  attachInterrupt(digitalPinToInterrupt(CLKpin), isr, CHANGE);
  attachInterrupt(digitalPinToInterrupt(DTpin), isr, CHANGE);
  pinMode(incLED, OUTPUT);   // increment indicator LED
}
void loop()
{
  if(change != 0)
  {
    if(change > 0) LED(HIGH, "up ");
    else if(change < 0) LED(LOW, "down ");
    change = 0;
  }
}
void LED(int state, String text)
```

```
{
   digitalWrite(redLED, 1-state);
   digitalWrite(greenLED, state);
   Serial.print(text);
   count = count + change*increment;  // incremented count
   digitalWrite(incLED, increment>1);
                                      // turn LED on, increment = 10
   Serial.print(increment);Serial.pri nt("\t");
                                      // display updated increment
   Serial.println(count);
}
IRAM_ATTR void isr()
   {                                  // avoid re-reading the square
   newCLK = digitalRead(CLKpin);      // wave state of pin A (CLK)
   int newState = (oldState<<2)+(newCLK<<1)+digitalRead(DTpin);
   score = score + vals[newState];
   oldState = newState % 4;
   if(score == 2 || score == -2)
   {
      change = score/2;
      score = 0;
   }
   newTime = millis();                // get time ISR triggered
   if(newCLK == HIGH && oldCLK == LOW) // interval between falling
   {                                  // and rising edge on pin A (CLK)
      lagTime = newTime - oldTime;
      if(lagTime < rotation && lagTim e > 100) increme nt = 10;
                                      // fast rotation
      else if(lagTime > rotation) inc rement = 1; // slow rotation
      oldTime = newTime;              // reset rising/falling edge time
   }
   oldCLK = newCLK;                   // reset state of pin A (CLK)
}
```

Преимущество использования прерываний для обнаружения восходящих и нисходящих фронтов, для определения изменений в состояниях энкодера, заключается в том, что микроконтроллер ESP8266 или ESP32 может обрабатывать другие команды одновременно, не пропуская изменения состояния контактов энкодера. Отображение изменений в состояниях, чтобы затем определить как направление, так и степень поворота энкодера, также решает проблему дребезга контактов энкодера. В этом случае уже не требуются резисторы и конденсаторы для создания задержки.

Итоги

Проиллюстрировано управление устройствами с помощью поворотного энкодера как по направлению, так и по величине поворота. Описаны способы улучшения измерения направления и поворота энкодера. Прерывания сами по себе не позволяли обнаруживать все импульсы при вращении. Изменения в состояниях выводов внутренних контактов энкодера сопоставлены с 4-разрядными числами. Реализация только изменений состояния с соответствующими 4-разрядными числами эффективно устраняла дребезг внутренних контактов энкодера. Постепенное изменение приращения счета при вращении энкодера управлялось кнопочным переключателем энкодера или скоростью вращения.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата DEVKIT DOIT или NodeMCU
- Arduino Nano
- Поворотный энкодер: KY-040
- Светодиоды: 3 шт. разных цветов
- Резисторы: 220 Ом 3 шт., 10 кОм 2 шт.
- Конденсатор: 10 мкФ 2 шт.

Глава 20

OTA и сохранение данных в EEPROM, SPIFFS и Microsoft Excel

Запоминающее устройство для компьютеров и микроконтроллеров называется флеш-памятью, которая сохраняет данные при выключении питания. Напротив, данные, хранящиеся в оперативной памяти (RAM), теряются при выключении питания. Флеш-память ESP8266 и ESP32 делится на несколько разделов, хранящих загрузчик, приложение, обновления OTA[123], файловую систему SPIFFS[124], энергонезависимую память EEPROM, настройки Wi-Fi и информацию о конфигурации. Скетч сохраняется в памяти приложения, а переменные, создаваемые и изменяемые в программе, хранятся в оперативной памяти. В этой главе описаны возможность удаленной загрузки скетча с помощью OTA и применение опций сохранения данных в разделах флеш-памяти SPIFFS и EEPROM. Также описано сохранение данных непосредственно в файл Microsoft Excel вместо использования SD-карты для хранения log-файлов.

OTA-обновление

При обновлении методом OTA скетч удаленно загружается через Wi-Fi-соединение с микроконтроллером ESP8266 или ESP32. Скетч первоначально загружается с помощью последовательного порта, но в дальнейшем используется метод OTA-обновления через подключение Wi-Fi. Ноутбук или компьютер для передачи и микроконтроллер ESP8266 или ESP32 для приема

[123] OTA (*over-the-air*, букв. «по воздуху») – метод установки и обновления программ и данных, отличающийся тем, что вместо обычного «ручного» скачивания файла и последующего запуска специальной программы-установщика закачка, установка или обновление происходит полностью автоматически, во многих случаях незаметно для пользователя, без каких-то действий с его стороны. Термин OTA обычно применяется для мобильных устройств, но, например, автоматическое обновление операционной системы, браузеров или антивирусов в настольных компьютерах также относится к категории OTA. – *Прим. перев.*

[124] SPIFFS (Serial Peripheral Interface Flash File System) – специальная файловая система для флеш-памяти, подключаемой по SPI-интерфейсу, см. далее. – *Прим. перев.*

обновленного скетча должны быть подключены к одной и той же сети Wi-Fi. Библиотека *ArduinoOTA*, которая предустановлена в Arduino IDE, требует установки Python 3.x на ноутбуке или компьютере. Python 3.x загружается с www.python.org/downloads, должна быть выбрана опция *Add Python 3.x to PATH* (см. рис. 20-1).

Рисунок 20-1. Установка Python 3.x

При первой загрузке скетча через последовательный порт имя и IP-адрес сетевого порта микроконтроллера по умолчанию – *esp8266-[Chip identity] at IP address* или *esp32-[MAC address] at IP address*. Присвоение имени сетевому порту вместо использования IP-адреса позволяет пользователю идентифицировать конкретный микроконтроллер по определяемому пользователем имени с помощью команды `ArduinoOTA.setHostname("name")`. Аналогичным образом пользователь устанавливает пароль для разрешения OTA-обновления с помощью инструкции `ArduinoOTA.setPassword("password")`, так как по умолчанию пароль отсутствует. Если пароль был задан, то он запрашивается при перезапуске Arduino IDE или при запросе на его изменение. Список доступных сетевых портов отображается при выборе меню *Tools* в Arduino IDE и опции *Port* (см. рис. 20-2). OTA-обновление включает механизм *mDNS*[125] для сопоставления имени сетевого порта с IP-адресом в небольших сетях, использующий протокол UDP для отправления и получения UDP-сообщения. Более подробная информация доступна на arduino-esp8266.readthedocs.io/en/latest/ota_updates/readme.html. Для микроконтроллеров ESP8266 и ESP32 библиотека *ArduinoOTA* ссылается на библиотеки ESP8266WiFi и WiFi соответственно, поэтому инструкции `#include <ESP8266WiFi.h>` и `#include <WiFi.h>` не требуются.

[125] mDNS (multicast DNS) – специальная система доменных имен (DNS) для локальной сети. Слово *multicast* («многоадресность») означает, что запрос рассылается всем устройствам в данном домене, за который принимается локальный участок сети с подключенными устройствами. Система mDNS первоначально была разработана Apple с целью дать возможность устройствам работать в сети без ручной настройки. – *Прим. перев.*

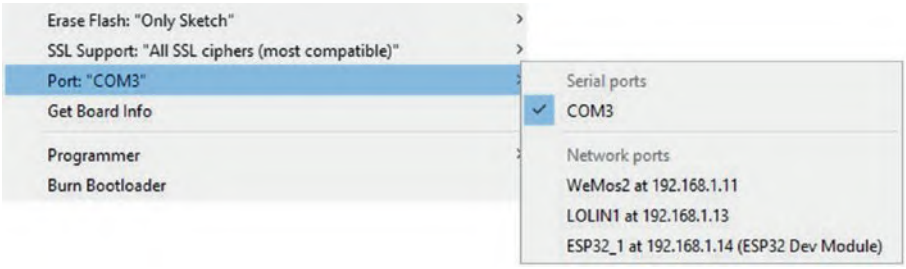


Рисунок 20-2. Доступные сетевые порты для OTA-обновления

Пример OTA-обновления демонстрируется в скетче в листинге 20-1, включающем или выключающем светодиод на определенное время. Листинг 20-1 предназначен для микроконтроллера ESP8266; для микроконтроллера ESP32 библиотека *mDNS* задается с помощью инструкции `#include <ESPmDNS.h>`. Библиотеки *mDNS* и *WiFiUdp* для микроконтроллеров ESP8266 и ESP32 оказываются доступны сразу после установки менеджеров плат ESP8266 и ESP32 в Arduino IDE. При первой загрузке скетча через последовательный порт монитор последовательного порта доступен для печати сообщений, но он оказывается недоступен при OTA-обновлении. В функции `setup` выполняется подключение по Wi-Fi и указывается имя сетевого порта микроконтроллера. Скетч включает в себя функцию `flash` для мигания встроенного светодиода каждые 500 мс, пока микроконтроллер ESP8266 или ESP32 подключается к сети Wi-Fi. В функции `loop` OTA-обновления отслеживаются с помощью команды `ArduinoOTA.handle()`, при этом команда повторяется после длительной задержки в скетче. Никаких других изменений в инструкциях в функции `loop` скетча нет по сравнению со скетчем для загрузки с помощью USB-соединения.

Удаленное OTA-обновление исправленного скетча использует именованный сетевой порт микроконтроллера ESP8266 или ESP32 для подключения к Wi-Fi, а не к последовательному порту. Теперь, когда плата ESP8266 или ESP32 работает удаленно, в меню *Tools* Arduino IDE выберите *Port* из списка доступных сетевых портов, как показано на рис. 20-2. Выберите подходящий порт; внесите необходимые изменения в текст программы, например отрегулируйте время задержки светодиода, и загрузите обновленный скетч с помощью OTA.

Листинг 20-1. OTA-обновление

```
#include <ArduinoOTA.h>       // include OTA library
#include <ESP8266mDNS.h>      // and mDNS libraries
#include <WiFiUdp.h>          // include Wi-FI UDP library
char ssid[] = "xxxx";         // change xxxx to Wi-Fi ssid
char password[] = "xxxx";     // change xxxx to Wi-Fi password
int LEDpin = 2;               // built-in LED
void setup()
{
  Serial.begin(115200);       // Serial Monitor baud rate
  pinMode(LEDpin, OUTPUT);
  WiFi.mode(WIFI_STA);        // initialise Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
```

```
  {
    delay(500);                    // flash LED while
    flash();                       // connecting to Wi-Fi
  }
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());  // display network port address
  ArduinoOTA.setHostname("WeMos2"); // name network port
  ArduinoOTA.setPassword("admin1"); // set password
  ArduinoOTA.begin();              // initialise ArduinoOTA
}
void loop()
{
  ArduinoOTA.handle();             // check for OTA updates
  digitalWrite(LEDpin, !digitalRead(LEDpin)); // turn on or off LED
  delay(1000);
}
void flash()                       // function to flash LED
{
  digitalWrite(LEDpin, HIGH);
  delay(100);
  digitalWrite(LEDpin, LOW);
}
```

Сохранение данных

Log-файлы с данными произведенных действий хранятся обычно на внешней SD-карте, доступной через SPI-интерфейс. Встроенная библиотека *SD* Arduino IDE предоставляет необходимые инструкции для создания, открытия и закрытия файлов, выполнения записи и чтения из файлов на SD-карте. Хранение данных на SD-карте для приложений с невысокими требованиями к хранению в течение небольших промежутков времени неэффективно – проще хранить данные в памяти самого микроконтроллера ESP8266 или ESP32. Например, сохранение настроек устройства, таких как состояние реле или яркость светодиода, во флеш-памяти гарантирует восстановление состояния устройства при сбросе микроконтроллера либо после выключения или потери питания. Другие примеры включают хранение файлов изображений с камеры ESP32 (см. главу 2 «Сетевая фотокамера») или положения сервопривода из приложения в главе 10 («Создаем мобильное приложение»).

Микроконтроллеры ESP8266 и ESP32 имеют до 50 КБ ОЗУ и 520 КБ SRAM, в которых создаются и обрабатываются переменные. Оперативная память является энергозависимой памятью, и ее содержимое недоступно после выключения питания микроконтроллера.

Напротив, микроконтроллеры ESP8266 и ESP32 имеют 4 МБ флеш-памяти, которая является энергонезависимой и, наоборот, сохраняется при выключении питания микроконтроллера. Скетч хранится в области флеш-памяти, отведенной для приложений, как и большие объемы данных для таблиц поиска (см., например, главу 16 «Генерация сигналов»). Массив сохраняется в области приложений флеш-памяти с помощью команды `const datatype arrayname[] PROGMEM = {значения массива}`, а доступ к *i*-му значению массива осуществляется с помощью команды `pgm_read_datatype(arrayname + i)`. Символьные или целочисленные данные хранятся с параметром `datatype`, определенным как `unsigned`

char или uint16_t, и доступны с помощью pgm_read_byte или pgm_read_word соответственно. Символьная строка, содержащая, например, AJAX-код веб-страницы, также хранится в разделе приложения флеш-памяти, как в главах 7 («Беспроводная локальная сеть»), 8 («Обновление веб-страницы»), 9 («WebSocket») и 12 («Приложение для GPS-трекинга с использованием Google Maps»).

Сохранение в EEPROM

EEPROM (Electrically Erasable Programmable Read-Only Memory, электрически стираемая программируемая память только для чтения) – это энергонезависимая память, которая сохраняет информацию при выключении микроконтроллера ESP8266 или ESP32. Емкость EEPROM равна 4096 байт, что составляет один сектор флеш-памяти, с доступом на чтение и запись для каждого байта в отдельности. EEPROM имеет ограничение в 100 тыс. циклов записи в каждой ячейке памяти. В главе 2 («Сетевая фотокамера») количество изображений, сохраненных на SD-карте, записывалось в EEPROM.

Для доступа к данным, хранящимся в EEPROM, используются инструкции встроенной библиотеки *EEPROM* Arduino IDE. Например, *i*-й байт в EEPROM записывается или считывается с помощью команды EEPROM.write(i, val) или EEPROM.read(i) соответственно, где val имеет целое значение между 0 и 255 (2^8 – 1). Нумерация байтов в EEPROM начинается с нуля. Целые числа, действительные числа, строки и структуры, требующие более 1 байта памяти, записываются в EEPROM или считываются из нее с помощью команды EEPROM.put(EEaddress, val) или EEPROM.get(EEaddress,val) соответственно, где EEaddress – номер байта в EEPROM для начала записи или чтения, а val – значение целого числа, действительного числа, строки или структуры. Как для целого, так и для действительного числа требуется 4 байта памяти EEPROM, независимо от величины числа. При сохранении данных с фиксированной структурой количество байтов записи является постоянным, а количество записей сохраняется в нулевом байте EEPROM, при этом адрес новой записи кратен количеству записей плюс четыре. Например, если в EEPROM уже хранится восемь записей данных, при этом каждая запись состоит из времени (двух целых чисел), целого числа и действительного числа, для чего требуется 16 байт памяти, тогда адрес EEPROM для девятой записи равен 132, что равно 8 (записей) × 16 (байт) + 4 (объем счетчика по адресу 0).

Инструкциям записи и чтения EEPROM должна предшествовать команда EEPROM.begin(N), где N – количество байтов EEPROM, к которым необходимо получить доступ. Например, для доступа до 1000 байт EEPROM требуется инструкция EEPROM.begin(1000). Запись в EEPROM должна сопровождаться инструкцией EEPROM.commit()[126]. Количество байтов, которое необходимо выделить для EEPROM, определяется с помощью команды EEPROM.length().

[126] Выполнение команды EEPROM.commit() гарантирует, что данные действительно запишутся в EEPROM, а не застрянут в буфере записи. Заметьте, что в Arduino-библиотеке EEPROM для контроллеров ATmega328 команды begin() и commit() отсутствуют, как и многие другие, приведенные здесь, так как в контроллерах ATmega более простой механизм доступа к EEPROM. – *Прим. перев.*

Скетч в листинге 20-2 сохраняет показания ультрафиолетового датчика в EEPROM каждые пять секунд и отображает сохраненные данные на последовательном мониторе. Количество записей получается с помощью команды `EEPROM.get(0, records)`, а не `EEPROM.read(0)`, которая является значением одного байта по нулевому адресу, так как количество записей может быть более 255. В скетче через монитор последовательного порта вводятся команды для записи, чтения и отображения, а также сброса данных, хранящихся в EEPROM. Когда количество байтов, требуемых для записей, приближается к заданной емкости EEPROM, количество записей сбрасывается на ноль. Количество необходимых байтов для записи определяется с помощью команды `sizeof()`, которая может ссылаться либо на тип переменной, либо на саму переменную, как это демонстрирует инструкция `Nbytes = sizeof(float) + sizeof(data.minuteTime) + sizeof(int)`.

Листинг 20-2. Сохранение данных в EEPROM

```
#include <EEPROM.h>                    // include EEPROM library
int EEaddress, records;
unsigned long seconds, nowTime, lastTime = 0;
char cmd = ' ';                        // command character
typedef struct                         // structure to hold data record
{
  float UV;                            // a real number (4 bytes) and an
  int minuteTime;                      // integer (4 bytes in EEPROM)
  int secondTime;
} dataStruct;
dataStruct data;                       // number of bytes to store data
int Nbytes = sizeof(float) + sizeof(data.minuteTime) +
sizeof(int);
void setup()
{
  Serial.begin(115200);                // Serial Monitor baud rate
  EEPROM.begin(1000);                  // set EEPROM capacity
  EEPROM.put(0, 0);                    // set record number to zero
  EEPROM.commit();                     // write to EEPROM
  Serial.print("Enter R: record, D: display");
  Serial.println(" or Z: zero UV record");
}
void loop()
{                                      // command from Serial buffer
  if(Serial.available() > 0) cmd = Serial.read();
  nowTime = millis();                  // start of time interval
  if((nowTime - lastTime > 5000) && (cmd == 'R'))
  {                                    // collect data every 5s
    data.UV = analogRead(A0)*3200.0/ 1024; // convert reading to mV
    seconds = (nowTime/1000);
    data.minuteTime = seconds / 60;  // calculate elapsed minutes
    data.secondTime = seconds % 60;  // and seconds
    EEPROM.get(0, records);          // number of EEPROM records
    EEaddress = records * Nbytes + 4; // address of new record
    if((EEaddress + Nbytes) > EEPRO M .length())
```

```
      {                                // check if exceeding the set EEPROM capacity
        records = 0;                   // reset EEPROM record number
        EEaddress = 4;                 // avoid over-flowing EEPROM
      }
      records++;                       // increment number of records
      EEPROM.put(0, records);          // update number of records
      EEPROM.put(EEaddress, data);     // write data to EEPROM
      EEPROM.commit();
      Serial.print("UV index ");Seria l.println(data.UV);
                                       // display data
      lastTime = nowTime;              // update time interval
    }
    if(cmd == 'D')                     // display data held on EEPROM
    {                                  // number of EEPROM records
      records = EEPROM.get(0, records );
      for (int i=0; i<records; i++)
      {
        EEaddress = i * Nbytes + 4;    // EEPROM address of ith record
        EEPROM.get(EEaddress, data);   // read and display EEPROM
        Serial.print(data.minuteTime);Serial.print(":");
        Serial.print(data.secondTime);Serial.print("\t");
        Serial.println(data.UV);
      }
      cmd = ' ';                       // reset command
      Serial.print("Enter R: record, D: display");
      Serial.println(" or Z: zero UV record");
    }
    if(cmd == 'Z')                     // command to reset records
    {
       EEPROM.put(0, 0);               // set record number to zero
       EEPROM.commit();
       cmd = ' ';
       Serial.print("Enter R: record, D: display");
       Serial.println(" or Z: zero UV record");
    }
}
```

На рис. 20-3 показано использование микроконтроллера ESP8266 или ESP32 EEPROM для хранения измерений от датчика ультрафиолета с подключениями, приведенными в табл. 20-1. Для платы ESP32 инструкция `analogRead` в листинге 20-2 заменена на `data.UV = analogRead(N)*3300.0/4096`, с номером вывода аналогового входа *N*.

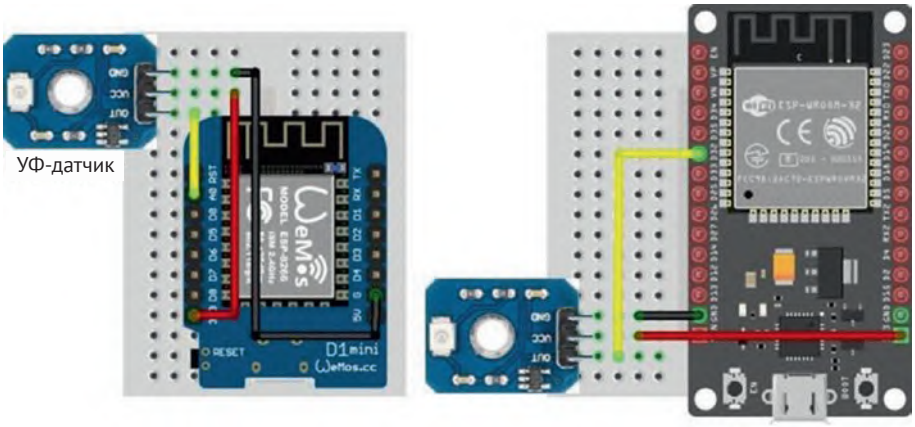


Рисунок 20-3. EEPROM и датчик ультрафиолета

Таблица 20-1. EEPROM и датчик ультрафиолета с микроконтроллерами ESP8266 и ESP32

Компонент	ESP8266	ESP32
УФ-датчик OUT	A0	GPIO 32
УФ-датчик VCC	3V3	3V3
УФ-датчик GND	GND	GND

Количество байтов, необходимое для хранения записи данных в EEPROM для символа, целого числа или действительного числа, равно 1, 4 или 4 соответственно. При хранении строки количество байтов зависит от типа строковых данных. Строка, определенная как `String s = "ABCDEFGH"`, требует 12 байт EEPROM, но строка, определенная как `char * s = "ABCDEFGH"`, требует всего 4 байт, так как `s` определяется как число-указатель на `char`, который уже указывает на объект типа `array of char` длиной 8 байт. Длина строки `s` получается с помощью команды `s.length()` или `strlen(s)`, когда строковый тип данных определен как `String` или `char s[]`.

Сохранение в SPIFFS

SPIFFS – это файловая система для записи и чтения файлов, хранящихся во флеш-памяти микроконтроллеров. Раздел флеш-памяти для SPIFFS настраивается в среде Arduino IDE в зависимости от требований скетча. Например, раздел SPIFFS по умолчанию для микроконтроллера ESP32 составляет 1472 кБ. SPIFFS используется для хранения файлов данных, файлов, содержащих HTML и AJAX-код для веб-страницы, а также файлов изображений.

Микроконтроллеры ES8266 и ESP32 используют библиотеки *LittleFS* и *SPIFFS* соответственно. Библиотека *LittleFS_esp32* доступна в Arduino IDE, но в настоящее время компания Espressif, разработчик ESP8266 и ESP32, поддерживает для микроконтроллера ESP32 встроенную библиотеку *SPIFFS*. Параметры инструкций для доступа к файлу с помощью библиотек *LittleFS* и *SPIFFS* перечислены в табл. 20-2. Обратите внимание в этой таблице, что заключение параметров команд в двойные кавычки требуется для микроконтроллера ESP8266,

но не для микроконтроллера ESP32. SPIFFS имеет одноуровневую структуру, структура в виде каталогов не поддерживается. Файл с указанием пути *temp/filename.txt* создает файл с именем *temp/filename.txt*, а не файл с именем *filename.txt* в каталоге *temp*. Библиотека *FS* для ESP8266 SPIFFS в настоящее время устарела; рекомендуется использовать библиотеку *LittleFS*, которая работает быстрее. Единственным изменением в инструкциях является замена инструкций `#include<FS.h>` и `SPIFFS.function()` на инструкции `#include<LittleFS.h>` и `LittleFS.function()`. Команда `SPIFFS.rename()` включает как компоненты `dir`, так и `filename` из `/dir/filename.txt`, подлежащего изменению, в то время как *LittleFS.rename()* изменяет только компонент `filename`. Дополнительная информация доступна на сайте arduino-esp8266.readthedocs.io/en/2.7.4_a/filesystem.html.

Таблица 20-2. Параметры SPIFFS-команд для микроконтроллеров ESP8266 и ESP32

Доступ к файлу	Библиотека ESP8266 LittleFS	Библиотека ESP32 SPIFFS
Чтение файла с начала	“r”	FILE_READ
Создать файл для записи с начала	“w”	FILE_WRITE
Добавить с конца	“a”	FILE_APPEND

Скетч в листинге 20-3 демонстрирует открытие, запись, чтение, переименование и удаление файла, хранящегося в SPIFFS. Хотя SPIFFS имеет одноуровневую структуру, файлы распределяются по каталогам с помощью префикса имени файла с именем каталога, например */temp/testfile.txt*. Учитывая ограничение в 31 символ для каталога и имени файла, рекомендуется использовать соглашение об именовании файлов 8-3, что означает восемь символов имени файла и три символа расширения. Параметры доступа к файлам библиотеки SPIFFS, “r”, “w” и “a” из табл. 20-2, достаточны для приложений, хотя также доступны расширенные параметры “r+”, “w+” и “a+”.

Листинг 20-3. Запись, чтение и добавление файла в SPIFFS для микроконтроллера ESP8266

```
#include <LittleFS.h>                 // include LittleFS library
String filename = "/temp/testfile.txt"; // structure /dir/file
String newname = "/temp/newfile.txt";
void setup()
{
   Serial.begin(115200);              // Serial Monitor baud rate
   if(LittleFS.begin()) Serial.println("initialised OK");
   dirContent("/");                   // contents of main directory
   dirContent("/temp");               // contents of sub directory
   File file = LittleFS.open(filename, " w"); // open file to write
   file.println("ABC");
   file.println("123");               // instead of print("xxx/n")
   file.close();
   fileContent(filename);             // function display file content
   dirContent("/temp");
   file = LittleFS.open(filename, "a"); // append to file
   file.println("XYZ");
   file.close();
```

```
   LittleFS.rename(filename, newnam e);
                                        // change filename not directory
   fileContent(newname);
   dirContent("/temp");
   if(LittleFS.exists(filename)) LittleF S.remove(filename);
}                                       // delete file
void dirContent(String dname)           // function to display directory content
{
   Serial.print(dname);
   S erial.println(" content");
   Dir dir = LittleFS.openDir(dname);
   while(dir.next())
   {
      File file = dir.openFile("r");    // read file
      Serial.print("file ");
      Serial.print(file.name());Serial.
      print("\t");
      Serial.print("size ");
      Serial.println (file.size());     // filesize
   }
}
void fileContent(String fname)          // function to display file content
{
   File file = LittleFS.open(fname, "r");
   while(file.available()) Serial.write(file.read());
   file.close();
}
void loop()                             // nothing in loop function
{}
```

Для микроконтроллера ESP32, кроме изменения библиотеки *LittleFS* на *SPIFFS* и параметров доступа к файлам, соответствующий скетч отличается от листинга 20-2 только функцией для отображения файлов в каталоге (см. листинг 20-4). Команда `SPIFFS.rename()` действует на оба подлежащих изменению компонента *dir* и *filename* в */dir/filename.txt*, аналогично библиотеке *FS* для микроконтроллера ESP8266.

Листинг 20-4. Составление списка файлов в каталоге с помощью SPIFFS для микроконтроллера ESP32

```
void dirContent(String dname)
{
   Serial.print(dname);Serial.println(" content");
   File dir = SPIFFS.open(dname);       // SPIFFS library
   File file = dir.openNextFile();      // openNextFile
   while(file)
   {
      Serial.print("file ");Serial.print(file.name());
      Serial.print("\t");
      Serial.print("size ");Serial.println(file.size());
      file = dir.openNextFile();        // openNextFile
   }
}
```

В листинге 20-3 показано использование SPIFFS для приложений ведения log-файлов, например, с помощью инструкций типа `file.println("123")` и `file.println("XYZ")`. SPIFFS также доступен для загрузки файлов, содержащих HTML и AJAX-код веб-страницы или файлов изображений. В Arduino IDE при программировании загрузки файлов в SPIFFS требуются различные плагины для микроконтроллеров ESP8266 и ESP32. Инструкции по установке и запуску плагинов доступны на сайте arduino-esp8266.readthedocs.io/en/latest/filesystem.html и github.com/me-no-dev/arduino-esp32fs-plugin соответственно. Разархивированный файл *esp8266littlefs.jar* или *esp32fs.jar*, содержащий плагин, должен находиться в папке *<папка со скетчами>* ➤ *tools* ➤ *ESP8266LittleFS* ➤ *tool* или *<папка со скетчами>* ➤ *tools* ➤ *ESP32FS* ➤ *tool*. Местоположение папки со скетчами в Arduino IDE можно определить через меню *File* ➤ *Preferences* (*Файл* ➤ *Настройки*). Если Arduino IDE была открыта, то закройте и перезапустите ее, тогда в меню *Tools* (*Инструменты*) появятся опции *ESP8266 LittleFS Data Upload* или *ESP32 Sketch DataUpload*.

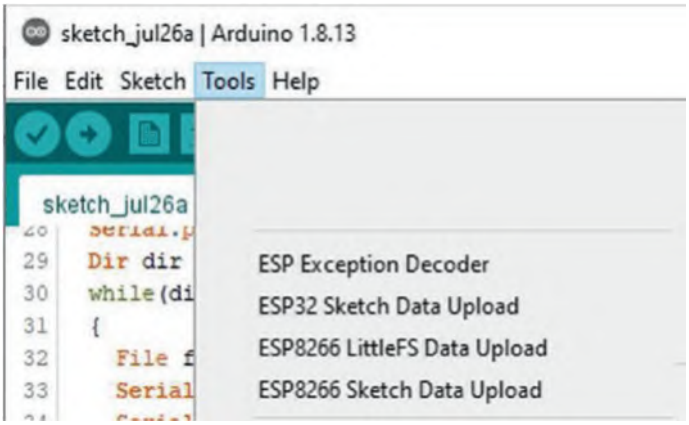


Рисунок 20-4. Плагины загрузки файлов в SPIFFS в Arduino IDE

Текстовый файл, содержащий HTML или AJAX-код веб-страницы, должен храниться в папке *data* внутри папки со скетчами. В среде IDE Arduino выберите меню *Sketch* (*Скетч*), щелкните по пункту *Show Sketch Folder* (*Показать папку скетча*) и создайте папку с именем *data*. Файл *filename.txt* упоминается как */filename.txt* в скетче. Перед загрузкой текстового файла в Arduino IDE убедитесь, что выбраны как плата, так и порт, а монитор последовательного порта закрыт. В среде IDE Arduino выберите меню *Tools* (*Инструменты*) и нажмите *ESP8266 LittleFS Data Upload* или *ESP32 Sketch Data Upload*. Как только появится сообщение *LittleFS Image Uploaded* или *SPIFFS Image Uploaded*, скомпилируйте и загрузите скетч.

В главе 8 («Обновление веб-страницы») код HTML и AJAX для веб-страницы был включен в отдельную вкладку *buildpage.h* с инструкцией для основного скетча *#include "buildpage.h"*. Веб-страница по умолчанию загружалась инструкцией `server.on("/", base)`, ссылающейся на функцию `base` для отправки клиенту строки `page`, содержащей HTML или AJAX-код для веб-страницы, с помощью команды `server.send (200, "text/html", page)`. Когда код HTML или

AJAX включен в отдельный файл, например *webcode.txt*, который загружается в SPIFFS, тогда отдельная вкладка *buildpage.h* не требуется, и функция `base` теперь загружает файл в SPIFFS, который затем отправляется клиенту. Функция `base` изменяется на

```
void base()
{
  File file = SPIFFS.open("/webcode.txt","r");        // file for webpage
  server.streamFile(file, "text/html");               // send file to server
  file.close();
}
```

Более того, необходимо удалить первую и последнюю строки текста `page`, содержащего код HTML или AJAX: `char page[] PROGMEM = R"(` и `)"`. Первая и последняя строки файла *webpage.txt* теперь должны быть `<!DOCTYPE html><html>` и `</body> </html>` соответственно, как в примере из листинга 8-4.

В главе 2 («Сетевая фотокамера») изображения сохраняются на карте micro-SD модуля ESP32-CAM в соответствии с инструкциями, приведенными в листинге 2-1:

```
fs::FS & fs = SD_MMC;     // with SD_MMC library,
file file = fs.open(filename.c_str(), FILE_WRITE);
                                      // access SD card
file.write(frame->buf, frame->len);  // save file to SD card
```

Модуль ESP32-CAM сохраняет изображение в формате JPEG в SPIFFS с помощью инструкций:

```
file file = SPIFFS.open("/photo.jpg", FILE_WRITE);
                                      // access SPIFFS
file.write(frame->buf, frame->len);  // write file to SPIFFS
```

а размер изображения отображается инструкциями:

```
Serial.print("Image size: ");
Serial.println(String(frame->len));
```

В главе 2 («Сетевая фотокамера») изображения ESP32-CAM загружаются на веб-страницу из PROGMEM, поскольку флеш-память (память программ) имеет больший объем, чем оперативная, а загрузка из PROGMEM происходит значительно быстрее, чем загрузка из SPIFFS, занимающая несколько секунд.

Загрузка файлов из SPIFFS

В то время как файлы данных, содержащие HTML и AJAX-код для веб-страницы, могут оставаться сохраненными в SPIFFS, файлы данных, хранящиеся в SPIFFS, например при ведении log-файлов, могут потребовать загрузки на компьютер или ноутбук для последующего анализа с помощью специального программного обеспечения. Еще одним вариантом, требующим загрузки, является скетч для анализа файлов данных, хранящихся в SPIFFS.

Файлы, хранящиеся в SPIFFS, загружаются на компьютер или ноутбук с помощью скетча, приведенного в листинге 20-5. Требуются библиотеки *ESPAsyncWebServer* и *AsyncTCP* за авторством Христо Гочкова (*Hristo Gochkov*).

Zip-файлы, содержащие библиотеки, загружаются с github.com/me-no-dev/ESPAsyncWebServer и github.com/me-no-dev/AsyncTCP соответственно. Библиотека *ESPAsyncWebServer* ссылается на библиотеки *AsyncTCP* и *Wi-Fi*, поэтому инструкции `#include <AsyncTCP.h>` и `#include <WiFi.h>` или `#include <ESP8266WiFi.h>` не требуются. Список каталогов, хранящихся в SPIFFS, отображается через монитор последовательного порта, а затем выбранный файл загружается на компьютер или ноутбук. Скетч включает в себя HTTP GET-запрос на загрузку файла, хранящегося в SPIFFS. Значение по умолчанию последнего параметра инструкции `request->send` равно `false`, чтобы указать формат файла как содержащего HTML-код, а не загрузочного. Пользователь определяет, где файл будет сохранен на компьютере или ноутбуке.

Листинг 20-5. Загрузка файла данных из SPIFFS

```
#include <LittleFS.h>                 // include LittleFS and
#include <ESPAsyncWebServer.h>        // ESPAsyncWebServer libraries
AsyncWebServer server(80);
char ssid[] = "xxxx";                 // change xxxx to Wi-Fi ssid
char password[] = "xxxx";             // change xxxx to Wi-Fi password
String filename;                      // file to be downloaded
void setup()
{
   Serial.begin(115200);              // define Serial Monitor baud rate
   WiFi.begin(ssid, password);        // initialise Wi-Fi
   while (WiFi.status() != WL_CONNECTED) delay(500);
   Serial.print("IP Address: ");
   Serial.println(WiFi.localIP());    // display WLAN IP address
   server.begin();                    // initialise server
   server.on("/download", HTTP_GET, [](AsyncWebServerRequest * request)
   { request->send(LittleFS, filename, "text/plain", true); });
   LittleFS.begin();                  // initialise SPIFFS
   dirContent("");                    // contents of main directory
   dirContent("temp");                // content of "temp" sub-directory
   Serial.println("\nEnter directory/filename to download");
}
void loop()
{
   if(Serial.available() > 0)         // filename entered on Serial Monitor
   {
      filename = Serial.readString();  // read Serial buffer
      Serial.print("In the browser, enter ");
      Serial.print(WiFi.localIP());
      Serial.print("/download to download file: ");
      Serial.println(filename);
      Serial.println("\n\nEnter directory/filename to download");
   }
}
void dirContent(String dname)
                        // function to display directory content
{
   Serial.print("\nContent of directory: ");
   Serial.println(dname);
   Dir dir = LittleFS.openDir("/"+dname);
   while(dir.next())
```

```
    {
      File file = dir.openFile("r");    // read file
      Serial.print(dname);Serial.print("/");
      Serial.print(file.name());Serial.print("\t");
      Serial.print("size ");Serial.println(file.size());
    }                                   // filesize
}
```

Листинг 20-5 предназначен для микроконтроллера ESP8266. Для микроконтроллера ESP32 библиотека *LittleFS* заменена библиотекой *SPIFFS*; кроме того, как и в листинге 20-3, функция `dirContent` заменена на функцию из листинга 20-4. В функции `dirContent` инструкция `File dir = SPIFFS.open(dname)` также изменяется на `File dir = SPIFFS.open("/"+dname)`. Содержимое главного каталога и директории *temp* отображается отдельно для микроконтроллера ESP8266, но при использовании микроконтроллера ESP32 содержимое любых каталогов отображается с помощью команды `dirContent("")`.

Сохранение данных в Excel

Микроконтроллеры ESP8266 и ESP32 не могут эмулировать USB-устройство, такое как клавиатура, для отправки символа или строки на подключенный ноутбук или компьютер. Однако это может делать Arduino Pro Micro. Покажем на примере, как данные, собранные датчиком, подключенным к Pro Micro, отправляются на подключенный ноутбук или компьютер и записываются непосредственно в файл Microsoft Excel. Данные датчика выводятся на график немедленно, в отличие от обычной практики сохранения данных на SD-карте с последующим импортом в файл Microsoft Excel.

Эмуляция клавиатуры в Pro Micro запускается с помощью команды `Keyboard.begin()`; при этом важно иметь возможность управлять этим процессом для завершения эмуляции – в противном случае не будет функционировать компьютерная клавиатура. Например, изменение состояния вывода GPIO нажатием кнопки, подключенной к Pro Micro, может вызвать команду `Keyboard.end()` и завершить эмуляцию со стороны Pro Micro. Символ или строка `str` отправляется на подключенный компьютер с помощью инструкции `Keyboard.print(str)` или `Keyboard.println(str)`, причем последняя включает в себя символы ASCII для возврата каретки и перевода строки. Перед загрузкой скетча с инструкциями по эмуляции клавиатуры рекомендуется протестировать скетч с помощью команд `Serial.print()` или `Serial.println()` и убедиться, что система остановки программы работает правильно. Эмуляция клавиатуры Pro Micro приведет к выводу текста в окно, которое в данный момент открыто на подключенном компьютере. Во время компиляции в среде Arduino IDE скетча, содержащего инструкцию `Keyboard.print()`, в нашем примере курсор компьютера должен быть установлен на открытом листе файла Microsoft Excel.

Скетч в листинге 20-6 считывает температуру с датчика BMP280, измеряет интенсивность света с помощью фоторезистора (LDR) и записывает данные не-

посредственно в открытый рабочий лист Microsoft Excel. Эмуляция клавиатуры с помощью Pro Micro прекращается нажатием кнопки с подключенным заземляющим резистором, что изменяет состояние вывода GPIO, соединенного с кнопкой, и запускает команду `Keyboard.end()`. В скетче данные записываются в лист Microsoft Excel каждые пять секунд. Управляющие символы ASCII 9, 10, 11 и 13 для горизонтальной табуляции, перевода строки, вертикальной табуляции и возврата каретки используются для форматирования данных на листе Microsoft Excel. Например, символ табуляции отправляется на подключенный ноутбук или компьютер с помощью `Keyboard.print(char(9))`.

Листинг 20-6. Сохранение данных непосредственно в Excel-файл

```
#include <Keyboard.h>                 // include Keyboard library
#include <Adafruit_Sensor.h>          // include Unified Sensor library
#include <Adafruit_BMP280.h>          // include BMP280 library
Adafruit_BMP280 bmp;                  // associate bmp with BMP280
int BMPaddress = 0x76;                // I2C address of BMP280
int switchPin = A3;                   // define switch and LDR pins
int LDRpin = 9;
unsigned long nowTime, lastTime = 0;
float temp, bright;
int counter = 0;
void setup()
{
  Keyboard.begin();                   // initialise Keyboard
  bmp.begin(BMPaddress);              // initialise BMP280
  header();                           // call header function
}
void loop()
{                                     // MUST BE ABLE TO STOP KEYBOARD
  if(digitalRead(switchPin) == HIGH) // switch to stop Keyboard
  {
    Keyboard.end();                   // stop Keyboard
    while(1);                         // and do nothing else
  }
  nowTime = millis();                 // set start of time interval
  if(nowTime - lastTime > 5000)       // collect data every 5s
  {
    counter++;                        // increment counter
    temp = bmp.readTemperature();     // get BMP280 reading
    bright = analogRead(LDRpin);      // and brightness reading
    Keyboard.print(counter);          // print counter to Excel
    Keyboard.print(char(9));          // print tab character
    Keyboard.print(temp);             // print temp to Excel
    Keyboard.print(char(9));          // print bright to Excel, with
    Keyboard.println(bright);         // carriage return and new line
    lastTime = nowTime;               // update start of time interval
  }
}
void header()                 // function to print columns headers to Excel
{
  Keyboard.print("counter");          // print "counter" to Excel
  Keyboard.print(char(9));            // print tab character
  Keyboard.print("temp");
  Keyboard.print(char(9));
```

```
    Keyboard.print("bright");
    Keyboard.print(char(13));  // print carriage return character
    Keyboard.print(char(10));  // print new line character
}
```

Соединения Pro Micro, BMP280 и фоторезистора (LDR) показаны на рис. 20-5 и перечислены в табл. 20-3. Обратите внимание на важность обеспечения наличия надежной системы остановки функции эмуляции клавиатуры во время выполнения скетча, поскольку в противном случае управление подключенным компьютером с его клавиатуры будет невозможно.

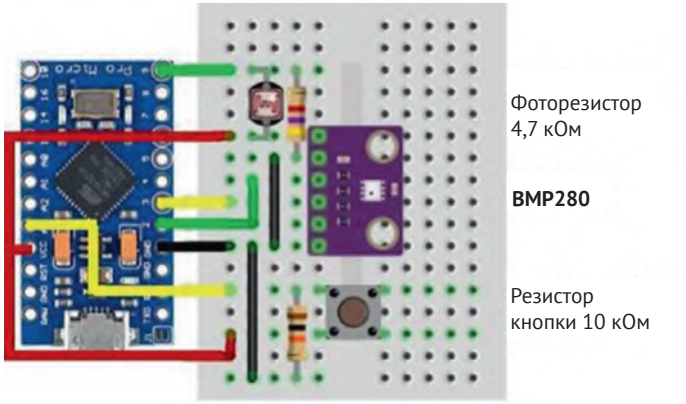


Рисунок 20-5. Pro Micro с датчиком BMP280 и фоторезистором

Таблица 20-3. Сохранение данных непосредственно в Excel-файл

Компонент	Подключено к	Также к
BMP280 VCC	Pro Micro 3.3V	
BMP280 GND	Pro Micro GND	
BMP280 SDI (SDA)	Pro Micro выв. 2	
BMP280 SCK (SCL)	Pro Micro выв. 3	
BMP280 CSB	Не подключен	
BMP280 SDO	Не подключен	
LDR верхний вывод	Pro Micro выв. 9	
LDR верхний вывод	Резистор 10 кОм	Pro Micro GND
LDR нижний вывод	Pro Micro 3.3V	
Кнопка верхний вывод	Pro Micro выв. A3	
Кнопка верхний вывод	Резистор 10 кОм	Pro Micro GND
Кнопка нижний вывод	Pro Micro 3.3V	

Итоги

Обновление методом OTA описано для удаленной загрузки скетча на микроконтроллер ESP8266 или ESP32. Разделы EEPROM и SPIFFS флеш-памяти микроконтроллеров ESP8266 и ESP32 используются для сохранения данных, файлов или изображений. Сохранение данных во внутренней EEPROM микроконтроллера описано с примером сохранения и извлечения измерений датчика. Хранение файлов на SPIFFS описано с примером создания, записи и чтения, переименования и удаления файла. Файл, содержащий HTML или AJAX-код для веб-страницы, загружается в SPIFFS с помощью плагина *Sketch Data Upload* для среды разработки Arduino IDE. Данные также сохраняются непосредственно на листе Microsoft Excel с помощью Arduino Pro Micro, эмулирующего клавиатуру, с приведенным примером измерений температуры и интенсивности света.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата DEVKIT DOIT или NodeMCUboard
- Arduino Pro Micro
- УФ-датчик
- Датчик температуры: BMP280
- Фоторезистор
- Тактовая кнопка
- Резисторы: 4,7 ком, 10 кОм

Глава 21

Микроконтроллеры

В книге используется несколько микроконтроллеров, в зависимости от требуемой функциональности. Плата LOLIN (WeMos) D1 mini на основе микроконтроллера ESP8266 используется в приложениях, требующих подключения к Wi-Fi. Плата ESP32 DEVKIT DOIT, основанная на микроконтроллере ESP32, идеально подходит для приложений, требующих интенсивной обработки данных с подключением по Wi-Fi или Bluetooth. Плата ESP32-CAM, основанная на микроконтроллере ESP32-S, является очевидным выбором для приложений, требующих наличия камеры в совокупности с Wi-Fi. Arduino Pro Micro эмулирует USB-устройство, например клавиатуру, в приложениях, сохраняющих данные непосредственно в формате электронной таблицы Microsoft Excel. Arduino Uno или компактный Arduino Nano, оба основанные на микроконтроллере ATmega328P, подходят для приложений, где не требуется подключение к Wi-Fi или высокая вычислительная мощность. В табл. 21-1 приведены некоторые свойства микроконтроллеров, а дополнительная информация доступна из www.arduino.cc/en/Main/Products и на www.espressif.com/ru/products/devkits. Микроконтроллеры ESP8266 и ESP32 имеют значительно более высокие тактовые частоты процессора, большее количество флеш-памяти и оперативной памяти (RAM), чем микроконтроллер ATmega328P. Программа хранится во флеш-памяти, в то время как переменные хранятся в оперативной памяти.

Таблица 21-1. Свойства микроконтроллеров

Параметр	ESP32	ESP8266	ATmega328P	ATmega32U4
Название платы модуля	DEVKIT DOIT	LOLIN (WeMos) D1 mini	Arduino Uno/Nano	Arduino Pro Micro
Архитектура	32 бит	32 бит	8 бит	8 бит
Частота CPU	240 МГц	160 МГц	16 МГц	16 МГц при 5 В
Флеш-память	4 МБ	4 МБ	32 кБ	32 кБ
(S)RAM	520 кБ	<50 кБ	2 кБ	2,5 кБ
Время расчета первых 10 тыс. простых чисел	385 мс	1786 мс	56 732 мс	57 027 мс

Все показанные платы-модули имеют выводы ШИМ и АЦП. Микроконтроллеры ESP8266 и ESP32 имеют подключение по Wi-Fi, при этом микроконтрол-

лер ESP32 также имеет обычный Bluetooth и вариант с низким энергопотреблением Bluetooth Low Energy, модуль цифроаналогового преобразователя ЦАП, сенсорные контакты и датчик Холла (см. главу 22 «Особенности микроконтроллера ESP32»).

Результатом более высокой частоты процессора, скорости работы флеш-памяти и оперативной памяти является меньшее время обработки скетча. Например, скетч в листинге 21-1 измеряет время, необходимое для определения первых 10 тысяч простых чисел. Скетч может быть неоптимальным с точки зрения минимизации времени обработки, но его достаточно для сравнения. Микроконтроллеру ATmega328P для выполнения этой задачи потребовалось 57 с, в то время как микроконтроллерам ESP8266 и ESP32 потребовалось менее двух секунд и менее половины секунды соответственно. Можно изменить частоту процессора микроконтроллера ESP32 в Arduino IDE до 10, 20, 40, 80 или 160 МГц, выбрав *Tools* (*Инструменты*) ➤ *CPU Frequency* (*Частота процессора*). Поскольку частота процессора удваивается на каждом шаге, время для определения первых 10K простых чисел на каждом шаге сокращается примерно вдвое: с 11 607 мс при 10 МГц до 5060 мс при 20 МГц, до 2403 мс при 40 МГц, до 1172 мс при 80 МГц и до 579 мс при 160 МГц.

Листинг 21-1. Определение первых 10 тысяч простых чисел

```
int Nprimes = 9999;           // required number of primes - 1
unsigned long number = 2;     // start from number 2
int count = 1;                // prime number counter
unsigned int start = 0;       // store processing time
unsigned long ms;
int chk, limit, mod, divid;
void setup()
{
   Serial.begin(115200);      // Serial Monitor baud rate
   while(!Serial);            // wait for Pro Micro to connect Serial
   Serial.print("\nCPU "); Serial.println(F_CPU/1000000);
   start = millis();          // start of processing time
}
void loop()
{
   number++;                  // increment number to check
   chk = is_prime(number);
   if (chk > 0) count++;      // increment counter when prime
   if (count > Nprimes)
   {
      ms = millis() - start;  // display results
      Serial.print("Found ");
      Serial.print(count);
      Serial.print(" primes in ");
      Serial.print(ms);
      Serial.println(" ms");
      Serial.print("Highest prime is ");
      Serial.println(number);
      delay(50000);           // long delay when finished
   }
}
```

```
int is_prime(unsigned long num)       // function to check if prime number
{
   mod = num % 2;                     // exclude even numbers
   if (mod == 0) return 0;
   limit = sqrt(num);                 // check divisors less than square root
   for (int divid = 3; divid <= limit; divid = divid + 2)
   {
      mod = num % divid;              // remainder after dividing
      if (mod == 0) return 0;         // not prime if zero remainder
   }
   return 1;                          // no divisor with zero remainder
}
```

Платы имеют различное расположение выводов, а также требования к установке и загрузке скетчей, поэтому эти особенности для каждой платы описаны отдельно. Информация о расположении контактов плат, доступная с помощью Arduino IDE, указана в файле *pins_arduino*, который находится по адресу *<пользователь>* ➤ *AppData* ➤ *Local* ➤ *Arduino15* ➤ *packages* ➤ *<microcontroller>* ➤ *hardware* ➤ *<category>* ➤ *<version>* ➤ *variants* ➤ *<board>*, где *microcontroller*, *category*, *version* и *board* соответствуют конкретной плате. Например, информация о расположении контактов LOLIN (WeMos) D1 mini находится в файле по адресу *<пользователь>* ➤ *AppData* ➤ *Local* ➤ *Arduino15* ➤ *packages* ➤ *esp8266* ➤ *hardware* ➤ *esp8266* ➤ *version* ➤ *variants* ➤ *d1_mini*.

Информация о частоте процессора микроконтроллера и контактах, выделенных для интерфейсов SPI и I2C, доступна в рамках Arduino IDE. Скетч в листинге 21-2 также содержит информацию о самом скетче. Скетч запускается на любом микроконтроллере, доступном с помощью Arduino IDE.

Листинг 21-2. Информация о микроконтроллере

```
String str, adjStr;
void setup()
{
   Serial.begin(115200);              // Serial Monitor baud rate
   Serial.println();
   while(!Serial);                    // Pro Micro wait for serial port
   Serial.print("Arduino IDE version ");
   str = String(ARDUINO);             // Arduino IDE version
   adjStr = str.substring(1,str.length()-5)+".";
   adjStr = adjStr + str.substring(str.length()-4,
   str.length()-2)+".";
   adjStr = adjStr + str.substring(str.length()-2);
   Serial.println(adjStr);            // date and time sketch compiled
   Serial.print("Compiler version");
   Serial.println(__VERSION__);
   Serial.print("Compiled date"); Serial.println(__DATE__);
   Serial.print("Compiled time"); Serial.println(__TIME__);
   Serial.print("Sketch location"); Serial.
   println(__FILE__);
   Serial.print("CPU frequency(MHz)"); // CPU frequency
   Serial.println(F_CPU/1000000);
   Serial.print("Development board");
   Serial.println(ARDUINO_BOARD);
   #ifdef __AVR__                     // development board
```

```
    Serial.print("Microcontroller");
    Serial.println(ARDUINO_MCU);
  #endif                                          // microcontroller
  Serial.print("SPI MOSI");Ser ial.println(MOSI);  // pin layout SPI
  Serial.print("SPI MISO");Serial.println(MISO);
  Serial.print("SPI SCK");Serial.println(SCK);
  Serial.print("SPI SS");Serial.println(SS);
  Serial.print("I2C SDA");Ser ial.println(SDA);    // pin layout I2C
  Serial.print("I2C SCL");Serial.println(SCL);
  #ifndef ESP32
  Serial.print("LED");Serial.println(LED_BUILTIN);  // built-in LED
  #endif
}
void loop()                                   // nothing in loop function
{}
```

Параметры ARDUINO_BOARD и ARDUINO_MCU генерируются инструкциями, основанными на строке 58 файла *platform.txt*, который ссылается на компиляцию файлов C++. Строка начинается с `recipe.cpp.o.pattern`. Файл *platform.txt* находится в папке <*пользователь*> ➤ *AppData* ➤ *Local* ➤ *Arduino15* ➤ *packages* ➤ *arduino* ➤ *hardware* ➤ *avr* ➤ *version*. Строка из файла *platform.txt* вставляется в новый файл, *platform.local.txt*, с добавлением `-DARDUINO_BOARD="{build.board}"` и `-DARDUINO_ MCU="{build.mcu}"`. Файл *platform.local.txt* хранится в той же папке, что и *platform.txt*. Когда скетч компилируется, обе переменные оказываются доступными. Параметр `ARDUINO_BOARD` уже существует для микроконтроллеров ESP8266 и ESP32, а параметр `ARDUINO_MCU` для микроконтроллеров ESP8266 и ESP32 отображать не требуется.

Листинг 21-2 включает директивы компилятора `#ifdef` и `#endif`, позволяющие компилятору установить, определен ли микроконтроллер как основанный на AVR (__AVR__), а затем включить соответствующие инструкции в процесс компиляции. Директивы компилятора `#if`, `#elif` и `#else` соответствуют операторам `if` с условием `else if` и `else` в языке C. Директива `#ifndef` означает *if not defined as* («*если не определено как*...»). За директивой компилятора `#ifdef` должна следовать директива `#endif`.

Микроконтроллеры в целом сгруппированы как основанные на AVR, на ESP8266 или ESP32. AVR – вероятно, аббревиатура от имени изобретателей архитектуры: Альф-Эгил Боген (Alf-Egil Bogen) и Вегард Воллен (Vegard Wollan), RISC – процессор. Подробная информация о конкретных разновидностях микроконтроллеров приведена в Arduino, в разделах ESP8266 или ESP32, файле *boards.txt*, который находится в том же каталоге, что и *platform.txt*, в группе *board.build*. Примеры разновидностей микроконтроллеров ESP8266 или ESP32 – это `ESP8266_WEMOS_ D1MINI` и `ESP8266_NODEMCU` или `ESP32_DEV` и `FEATHER_ESP32`. Если определенная разновидность микроконтроллеров используется в директиве компилятора, то названию предшествует `ARDUINO_`, как, например, в `#ifdef ARDUINO_ESP32_DEV`.

Arduino IDE имеет предопределенные значения для констант π, e, $\pi/180$ и $180/\pi$, определяемых как `PI`, `EULER`, `DEG_TO_RAD` и `RAD_TO_DEG` соответственно. Значения находятся в файле *Arduino* ➤ *hardware* ➤ *arduino* ➤ *avr* ➤ *cores* ➤ *arduino* ➤ *Arduino*.

Arduino Uno

Arduino Uno основан на микроконтроллере ATmega328P[127]. Модуль работает при напряжении 5 В, питание осуществляется через USB-соединение с напряжением 5 В или через входной разъем-гнездо от источника постоянного тока с напряжением 7–12 В. Максимальный ток, подаваемый с вывода GPIO[128], составляет 40 мА, при этом максимальный суммарный ток со всех выходных выводов составляет 200 мА. В скетче контакты GPIO обозначаются нумерацией, приведенной на плате Uno, а не номерами выводов микроконтроллера ATmega328P. Исключением являются контакты АЦП A0–A5, которые также могут быть обозначены как номера контактов 14–19. Коммуникационные контакты для интерфейса I2C – A4 (SDA) и A5 (SCL), для SPI – 11 (MOSI), 12 (MISO), 13 (SCK) и 10 (SS). Контакты ШИМ – это 3, 5, 6, 9, 10 и 11. Контакты внешних прерываний INT0 и INT1 – 2 и 3. Встроенный светодиод подключен к выводу 13.

Arduino Nano

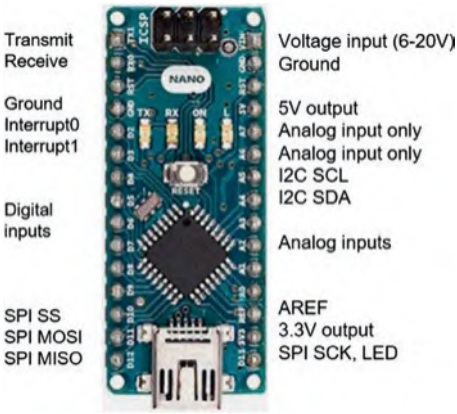
Arduino Nano имеет тот же самый микроконтроллер ATmega328P, как и Arduino Uno, и имеет ту же функциональность контактов, за исключением того, что контакты A6 и A7 являются только аналоговыми входами[129]. Цифровые контакты имеют префикс D на плате Nano[130]. Плата работает при напряжении 5 В и питается через разъем мини-USB при напряжении 5 В. В январе 2018 года

[127] Буква P после цифр в наименовании типа AVR-микроконтроллеров означает малопотребляющие версии (*pico-power*, до 100 нА в режиме Power-down). Не путать с той же буквой после дефиса в конце наименования (ATmega328P-PU), которая означает тип корпуса DIP. Подробнее см. Маркировка микроконтроллеров AVR на *microkontroller.ru*. – *Прим. перев.*

[128] Для выводов плат Arduino на основе 8-разрядных ATmega328P не принято употреблять название GPIO (General Purpose Input Output, ввод/вывод общего назначения), это сокращение стало модным с появлением 32-разрядных платформ (таких как Rasberry PI, например). Хотя по смыслу оно совершенно точно подходит и к 8-разрядным AVR, для последних принято обозначать выводы просто цифрами, иногда с добавлением «D» (от *digital*, если вывод употребляется как цифровой) или «A» (если существенны его аналоговые функции, см. далее). – *Прим. перев.*

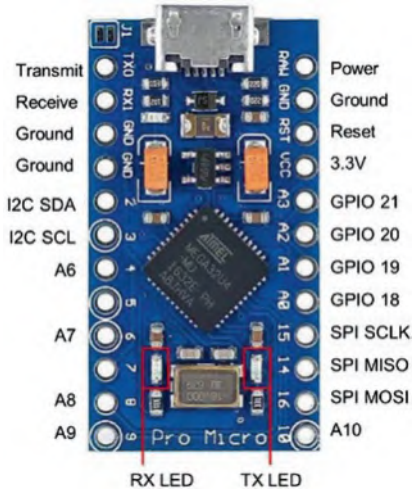
[129] Это связано с разными корпусами микроконтроллера ATmega328P – у плоского миниатюрного корпуса TQFP-32 выводов больше, и к ним подключены 6-й и 7-й выводы АЦП, не задействованные в 28-выводном DIP-корпусе, установленном в Arduino Uno.

[130] Уточним, что в скетчах для всех 8-разрядных Arduino в обозначениях выводов общего назначения употребляются голые номера, без добавления «D». Для аналоговых входов, наоборот, добавление «A» обязательно, но можно и их тоже обозначать числами в диапазоне 14–19 (см. выше). – *Прим. перев.*



Arduino выпустила новый загрузчик для Arduino Nano, поэтому в Arduino IDE необходимо выбрать соответствующую опцию. В меню *Tools (Инструменты)* ➤*Processor* (*Процессор*) выберите либо опцию *ATmega328P*, либо опцию *ATmega328P (Old Bootloader)* (*Старый загрузчик*).

Arduino Pro Micro



В среде Arduino IDE Arduino Pro Micro можно найти под названием Arduino Leonardo в меню *Tools* (*Инструменты*) ➤ *Board* (*Плата*). Модуль работает от 5 В и питается через разъем мини-USB при напряжении 5 В[131]. Как и для других Arduino, в скетче выводы обозначаются номерами, помещенными на плате, а не номерами выводов GPIO самого микроконтроллера ATmega32U4. Контакты АЦП A0–A3 также могут обозначаться как номера контактов 18–21. Выводы 4, 6, 8, 9 и 10 соответствуют выводам АЦП A6–A10. Контакты интерфейса I2C – номер 2 (SDA) и 3 (SCL), SPI – 16 (MOSI), 14 (MISO) и 15 (SCK). ШИМ-контакты – 3, 5, 6, 9 и 10. Контакты внешних прерываний INT0, INT1, INT2, INT3 и INT6 – выводы 3, 2, 0 (также RX), 1 (также TX) и 7. Встроенные светодиоды RX и TX соединены с внутренними выводами 17 и 30, с активным низким уровнем и автоматически определяются только как выходные (*OUTPUT*) контакты. Индикатор TX выключается или включается с помощью встроенного макроса TXLED0 или TXLED1 соответственно (см. листинг 21-3). Если макросы TXLED0 и TXLED1 не используются, необходимо определить вывод TXLED, как показано в скетче ниже.

[131] Arduino Pro Micro – переработанный сторонними фирмами вариант оригинальной платы Arduino Micro, которая, в свою очередь, является уменьшенной версией Arduino Leonardo (подобно тому, как Arduino Nano – уменьшенный вариант Arduino Uno). Как и другие Arduino, Arduino Pro Micro имеет встроенный стабилизатор напряжения, только его вход носит название не VIN, как обычно, а RAW (крайний верхний справа). На RAW можно подавать напряжение от 6 до 12 В, впрочем, как и другие Arduino, модуль прекрасно работает при подаче на VIN/RAW напряжения 5 В. – *Прим. перев.*

Листинг 21-3. Управление встроенными светодиодами Pro Micro

```
int RXLED = 17;       // define RXLED pin
// int TXLED = 30;    // required if not using macros
void setup()
{}                    // nothing in setup function
void loop()
{
  digitalWrite(RXLED, HIGH);                // turn off RXLED
  // digitalWrite(TXLED, HIGH);             // turn off TXLED
  TXLED0;                                   // macro to turn off TXLED
  delay(1000);
  digitalWrite(RXLED, LOW);                 // turn on RXLED
  // digitalWrite(TXLED, LOW);              // turn on TXLED
  TXLED1;                                   // macro to turn on TXLED
  delay(1000);
}
```

При загрузке скетча открывается последовательный порт загрузчика ATmega32U4, по окончании загрузки он закрывается. Затем открывается последовательный порт ATmega32U4, поэтому соответствующий COM-порт может измениться после загрузки скетча. Инструкция `while(!Serial)` ожидает, пока не будет установлено последовательное соединение. Микроконтроллер ATmega32U4 не сбрасывается при открытии COM-порта, в отличие от ATmega328P в Arduino Uno.

Микроконтроллер Arduino Pro Micro сбрасывается двукратным подключением вывода сброса (RST) к GND, для того чтобы перевести Pro Micro в режим загрузчика (*bootloader*) на восьмисекундный период перед запуском скетча. Микроконтроллер ATmega32U4 может заблокироваться, если возникает проблема при загрузке скетча, в котором используется библиотека *Keyboard*, или с неправильно определенным микроконтроллером, таким как микроконтроллер 16 МГц/5 В, определенный как 8 МГц / 3,3 В. Может возникнуть необходимость в переустановке загрузчика. Рекомендуется использовать *Atmega_Board_Programmer* от Ника Гэммона (Nick Gammon), который загружается с githib.com/nickgammon/arduino_sketches.

Модули ESP8266

Плата LOLIN (WeMos) D1 mini (см. рис. 21-1) основана на микроконтроллере ESP-8266EX и имеет функцию Wi-Fi. Плата работает при напряжении 3,3 В и питается через соединение micro-USB при напряжении 5 В через регулятор напряжения 3,3 В или непосредственно от вывода 3,3 V. Выводы не выдерживают внешнего напряжения 5 В, а максимальный ток, подаваемый с одного контакта, составляет 12 мА.

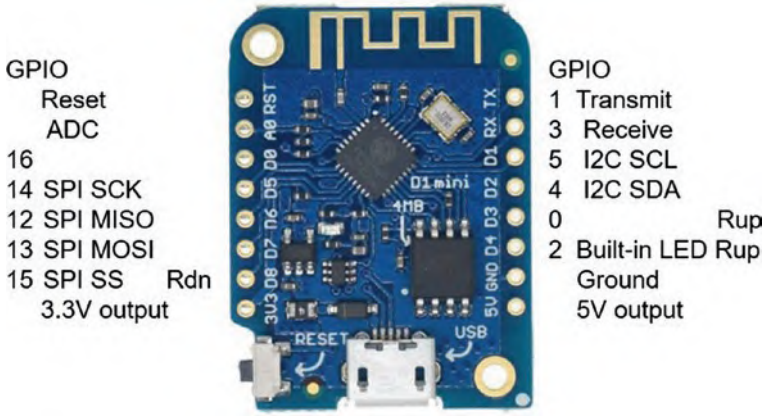


Рисунок 21-1. Плата LOLIN (WeMos) D1 mini

В скетче контакты обозначаются либо нумерацией на плате с добавлением буквы D для цифровых контактов, либо номерами GPIO микроконтроллера ESP-8266EX. Вывод АЦП обозначается A0 и имеет 10-битное разрешение. Выводами для интерфейса I2C являются D2/GPIO 4 (SDA) и D1/GPIO 5 (SCL) для SPI – D7/GPIO 13 (MOSI), D6/GPIO 12 (MISO), D5/GPIO 14 (SCK) и D8/GPIO 15 (SS). Четыре цифровых вывода, D2, D6, D5 и D8 (GPIO 4, GPIO 12, GPIO 14 и GPIO 15), имеют 10-битное разрешение ШИМ и функциональность прерывания. Встроенный светодиод на выводе D4 (GPIO 2) активен при подаче низкого уровня (LOW). Обратите внимание, что контакты D3/GPIO 0 и D4/GPIO 2 имеют встроенные подтягивающие резисторы, в то время как вывод D8/GPIO 15 имеет встроенный заземляющий резистор. При загрузке скетча контакты D3 и D4 не должны быть подключены к низкому уровню, и аналогично вывод D8 не должен быть подключен к высокому. На рис. 21-1 и 21-2 встроенный подтягивающий или заземляющий резистор обозначен *Rup* или *Rdn*.

Драйвер USB-UART-преобразователя CH340G для LOLIN (WeMos) D1 mini загружается с docs.wemos.cc/en/latest/ch340_driver.html. Сохраните файл *CH341SER_WIN_3.5 zip* на рабочем столе, выберите правой кнопкой мыши пункт *Извлечь все* (*Extract All*), в извлеченной папке *CH341SER_WIN_3.5* щелкните правой кнопкой мыши приложение *Setup*, выберите *Запуск от имени администратора* (*Run as administrator*) и установите драйвер *CH341S64.SYS*. Драйверы находятся в папке *Windows* ➤ *System32* ➤ *drivers*.

Плата NodeMCU ESP8266 (см. рис. 21-2) обладает той же эффективной функциональностью, что и плата LOLIN (WeMos) D1 mini. Плата NodeMCU ESP8266 имеет три контакта 3,3 В и четыре контакта GND для подключения к другим устройствам, а также два встроенных светодиода на контактах D4/GPIO 2 и D0/GPIO 16, которые включаются при подаче низкого уровня. Микроконтроллер ESP8266 сохраняет скетч во внешней микросхеме флеш-памяти и взаимодействует с этой микросхемой через SDIO-интерфейс. Контакты GPIO 6–11 платы NodeMCU ESP8266 соответствуют контактам интерфейса SDIO с маркировкой CLK, SD0, SD1, SD2, SD3 и CMD.

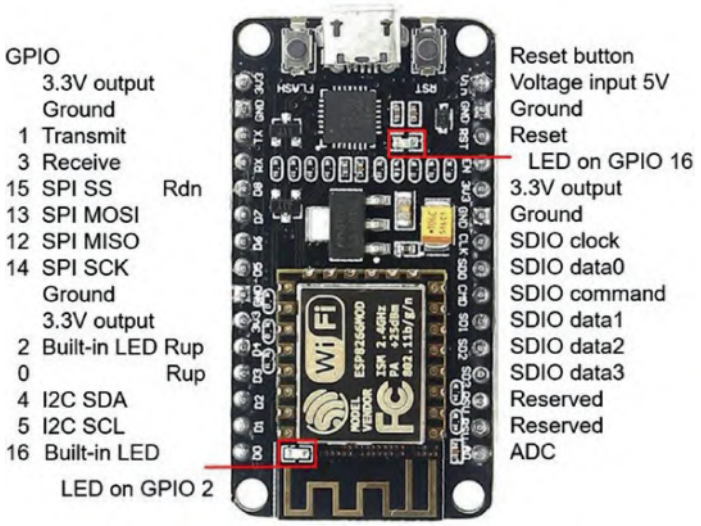


Рисунок 21-2. Плата NodeMCU ESP8266

Драйвер моста CP210x USB-to-UART для платы NodeMCU ESP8266 загружается с www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers. Сохраните zip-файл *CP210x_Universal_Windows_Driver* на рабочем столе, выберите правой кнопкой мыши пункт *Извлечь все* (*Extract All*). В извлеченной папке *CHP210x_Universal_Windows_Driver* щелкните правой кнопкой мыши либо 64-разрядную, либо x86-версию приложения *CP210xVCPInstaller* для 64-разрядной или 32-разрядной операционной системы соответственно. Чтобы определить, установлена ли на компьютере 64-разрядная или 32-разрядная операционная система, обратитесь по адресу *Панель управления* ➤ *Система и безопасность* ➤ *Система*. Выберите *Запуск от имени администратора* (*Run as administrator*) и установите драйвер *silabser.sys*. Драйверы находятся в папке *Windows* ➤ *System32* ➤ *Drivers*. Возможно, потребуется перезагрузить компьютер для завершения процедуры установки.

Для обеих плат разработки LOLIN (WeMos) D1 mini и NodeMCU ESP8266 выберите *Файл* ➤ *Настройки* (*File*➤ *Preferences*) в меню среды IDE Arduino и введите URL-адрес *http://arduino.esp8266.com/stable/package_ esp8266com_index.json* в поле URL-адреса менеджера дополнительных плат. Если в поле уже есть URL-адрес, разделите URL-адреса запятой.

Библиотеки ESP8266 устанавливаются в Arduino IDE, если выбрать *Tools* ➤ *Board* ➤ *Boards Manager* (*Инструменты* ➤ *Плата* ➤ *Менеджер плат*), ввести «8266» в опцию *Filter* (*Фильтр*), чтобы отобразить *esp8266* от *ESP8266 Community*, и нажать *Install* (*Установить*). В раскрывающемся списке *Tools* ➤ *Board* (*Инструменты* ➤ *Плата*) выберите LOLIN(WEMOS) D1 R2 & mini, находящуюся в разделе *ESP8266 Boards*. В меню *Tools* ➤ *CPU Frequency* (*Инструменты* ➤ *Частота процессора*) выберите 160 МГц; а в меню *Tools* ➤ *Port* (*Инструменты* ➤ *Порт*) выберите соответствующий COM-порт.

Справочная документация для микроконтроллера ESP8266 доступна по адресу arduino-esp8266.readthedocs.io/en/latest/index.html.

Аналоговый вход ESP8266

10-разрядный аналого-цифровой преобразователь (АЦП) микроконтроллера ESP8266 преобразует напряжение от 0 до 3,2 В на выводе аналогового входа A0 в цифровое значение от 0 до 1023. Команда `analogRead(A0)` считывает напряжение на выводе аналогового входа A0. Опорное напряжение АЦП микроконтроллера ESP8266 составляет 1 В; внутренний делитель напряжения, состоящий из резисторов 100 кОм и 220 кОм (см. рис. 21-3), увеличивает максимальное напряжение на выводе аналогового входа до 3,2 В. При заданном напряжении VIN на выводе аналогового входа платы ESP8266 соответствующее значение АЦП равно $V_{IN} = \frac{100\ \text{кОм}}{(220 + 100)\ \text{кОм}} \times 1024$.

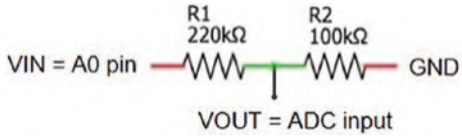


Рисунок 21-3. Аналого-цифровой преобразователь

Значения на выходе АЦП для входных напряжений от 3,2 В до 3,3 В ограничены значением 1023. Резистор 10 кОм, подключенный между входным напряжением и выводом аналогового входа платы ESP8266, увеличивает предел 3,2 В на выводе аналогового входа до 3,3 В.

Прерывания ESP8266

Внешнее прерывание подключается к контакту на плате ESP8266, такому как контакты D1–D7 LOLIN (WeMos) D1 mini, инструкцией `attachInterrupt(digitalPinToInterrupt(switchPin), change, FALLING)` (см. рис. 21-4, слева). Кнопка подключена к GND и контакту на плате ESP8266. Инструкция `pinMode(switchPin,INPUT_PULLUP)` активирует внутренний подтягивающий резистор на выводе, к которому подключена кнопка, поэтому нормальное состояние этого вывода – высокий уровень (`HIGH`). Когда кнопка нажата, вывод подключается к GND, изменяя состояние на низкий уровень (`LOW`), при этом падающий фронт (`FALLING`) активирует прерывание. Микроконтроллер ESP8266 хранит скомпилированный код во внутренней оперативной памяти (IRAM), а не в более медленной флеш-памяти, если добавить к инструкциям в скетче атрибут `IRAM_ATTR`. Обработчик прерывания (ISR) определяется как `IRAM_ATTR void ISR()`, а не как просто `void ISR()`. Если обработчик прерывания определен перед функцией `setup` скетча, то команда определения ISR изменяется на `void IRAM_ATTR ISR()`.

Напротив, если вывод переключателя равен D8, то кнопка подключена к 3,3 В (см. рис. 21-4, справа). Вывод D8 имеет встроенный заземляющий резистор, поэтому в нормальном состоянии вывод находится на низком уровне (`LOW`). Прерывание активируется на выводе D8 с помощью команды `attachInterrupt (digitalPinToInterrupt(switchPin), change, RISING)`. Когда нажата кнопка, состояние вывода изменяется на высокий уровень, при этом возрастающий фронт (*RISING*) активирует прерывание. Прерывание по возрастающему

фронту с выводом D8, подключенному к кнопке, менее подвержено дребезгу контактов, чем прерывание по падающему фронту на выводах D1–D7. Обратите внимание, что разница между двумя схемами на рис. 21-4 заключается только в выводе кнопки, подключенной к контактам D7 и GND или к контактам D8 и 3,3 В.

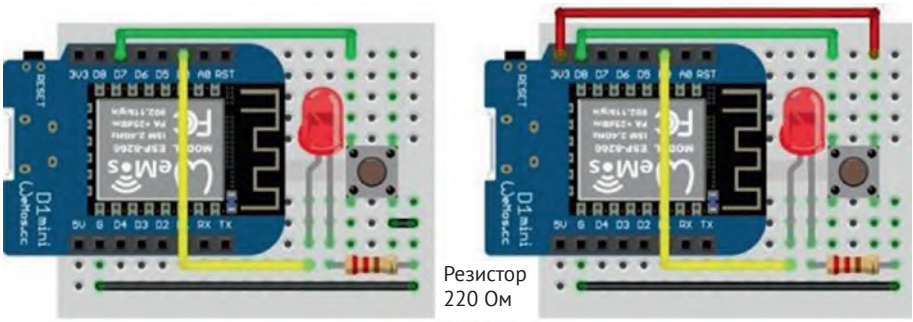


Рисунок 21-4. Прерывания в LOLIN (WeMos) D1 mini

Скетч в листинге 21-4 иллюстрирует разницу между прерываниями, подключенными к выводам D7 и D8 платы разработки ESP8266. Когда прерывание подключено к выводу D7, вывод определяется как INPUT_PULLUP для активации встроенного подтягивающего резистора, и прерывание активируется при падающем уровне. Если прерывание подключено к выводу D8, то прерывание активирует возрастающий уровень.

Листинг 21-4. Прерывания с платой ESP8266

```
int LEDpin = D0;                        // define LED pin
int switchPin = D7;                     // define switch pin
volatile int LEDstate = LOW;            // initial LED state
                                        // volatile as LEDstate in ISR
void setup()
{
   Serial.begin(115200);                // Serial Monitor baud rate
   pinMode(LEDpin, OUTPUT);             // define LEDpin as OUTPUT
   pinMode(switchPin, INPUT_PULLUP);    // when switch pin is D1 to D7
   attachInterrupt(digitalPinToInterrupt(switchPin), change,
   FALLING);                            // when switch pin is D8
   // attachInterrupt(digitalPinToInterrupt(switchPin), change,  // RISING);
}
void loop()
{
   Serial.println(digitalRead(switchPin));
                                        // display switch pin state
   delay(1000);
}
IRAM_ATTR void change()                 // interrupt service routine (ISR)
{
   LEDstate = 1-LEDstate;               // change LED state
   digitalWrite(LEDpin, LEDstate);      // turn LED on or off
}
```

Сторожевой таймер ESP8266

Если микроконтроллер ESP8266 не может выполнять фоновые задачи из-за длительного периода бездействия или зависания во время выполнения программы, то сторожевой таймер (Watchdog) может инициировать сброс программного или аппаратного обеспечения. Фоновые задачи включают в себя поддержание подключения к Wi-Fi или управление TCP/IP-подключением к интернету. На программный сброс указывают сообщения `Soft WDT reset` и `rst cause:2, boot mode:(3,6)`. Включение команды задержки `delay(1)` и `yield()` в скетч в момент бездействия может разрешить программный сброс. Инструкция `yield()` позволяет выполнение фоновых задач.

Момент программного сброса скетча сторожевым таймером определяется с помощью декодера исключений *ESP Exception Decoder*. Инструкции по установке и запуску декодера исключений изложены в arduino-esp8266.readthedocs.io/en/latest/faq/a02-my-esp-crashes.html. Декодер исключений ESP загружается с github.com/me-no-dev/EspExceptionDecoder. Разархивированный файл *EspExceptionDecoder.jar*, содержащий декодер исключений ESP, должен находиться в папке *Sketchbook* ➤ *tools* ➤ *EspExceptionDecoder* ➤ *tool*, где папка расположения скетчей *Sketchbook* определяется в среде IDE Arduino, через меню *File* ➤ *Preferences* (*Файл* ➤ *Настройки*).

Аппаратный сброс с помощью сторожевого таймера сопровождается сообщениями `wdt reset` и `rst cause:4, boot mode:(3,6)`. Момент аппаратного сброса в скетче не определяется с помощью *ESP Exception Decoder*, поэтому для определения последней успешной команды перед аппаратным сбросом можно использовать серию инструкций `Serial.println()`.

Модули ESP32

Плата ESP32 DEVKIT DOIT основана на микроконтроллере ESP32 и имеет функции Wi-Fi и Bluetooth. Плата работает при напряжении 3,3 В и питается через разъем micro-USB при напряжении 5 В или непосредственно от VIN-контакта 3,3 В, но рекомендуется использовать первое подключение. Контакты GPIO не выдерживают подачи напряжения 5 В, а максимальный ток, подаваемый с контакта, составляет 12 мА.

В тексте скетча контакты обозначаются номерами GPIO микроконтроллера ESP32. Имеется шесть выводов АЦП (GPIO 32, GPIO 33, GPIO 34, GPIO 35, GPIO 36 и GPIO 39) с 12-битным разрешением и двумя ЦАП на контактах GPIO 25 и GPIO 26 с 8-битным разрешением. Коммуникационные контакты для I2C – это GPIO 21 (SDA) и GPIO 22 (SCL), а для SPI – GPIO 23 (MOSI), GPIO 19 (MISO), GPIO 18 (CLK) и GPIO 5 (CS). Все выводы GPIO, за исключением выводов только для ввода (GPIO 34, GPIO 35, GPIO 36 и GPIO 39), имеют ШИМ-функцию; и все контакты GPIO имеют функцию прерывания. Есть девять емкостных сенсорных

выводов[132] (GPIO 2, GPIO 4, GPIO 12, GPIO 13, GPIO 14, GPIO 15, GPIO 27, GPIO 32 и GPIO 33). Встроенный светодиод подключен к GPIO 2 и зажигается подачей высокого уровня (HIGH). На нескольких контактах доступны функции подсистемы RTC[133] для вывода микроконтроллера ESP32 из спящего режима. Внутренние подтягивающие резисторы подключены к выводам GPIO 0, 5, 14 и 15, заземляющие резисторы – к выводам GPIO 2, 4 и 12. Микроконтроллер ESP32 содержит датчик Холла. Разводка контактов плат ESP32 DEVKIT DOIT и NodeMCU с 30 и 38 контактами показана на рис. 21-5 и 21-6 соответственно, для иллюстрации различия в платах. Функции контактов обозначены как: *A* – аналоговый вход; *T* – емкостный сенсорный; *input* – только вход; *RTC* – вывод подсистемы RTC; *Rup* – встроенный подтягивающий резистор и *Rdn* – встроенный подтягивающий резистор. Контакты GPIO 6–11 платы разработки NodeMCU подключены к встроенной флеш-памяти, и использование этих контактов не рекомендуется.

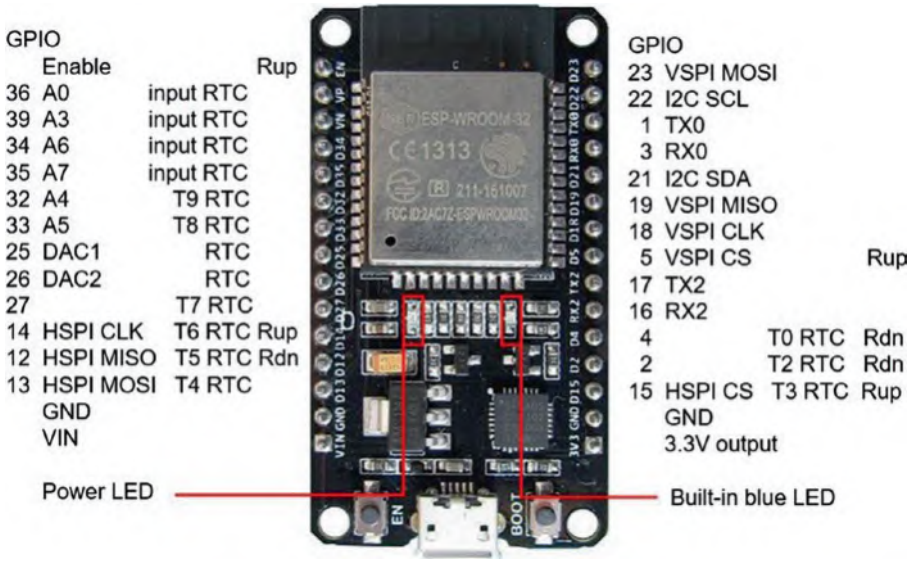


Рисунок 21-5. 30-контактная плата ESP32 DEVKIT DOIT

[132] Емкостный сенсорный вывод – вывод со встроенным механизмом определения изменения электрической емкости между металлической площадкой, подключенной к этому выводу, и общим проводом. Сенсор настраивается на изменение емкости, происходящее в результате поднесения пальца к такой площадке, поверх которой размещается слой диэлектрика. Подробнее см. раздел «Емкостный сенсорный датчик» в главе 22. – *Прим. перев.*

[133] В описаниях ESP32 часто встречается аббревиатура RTC, которая в большинстве случаев не имеет отношения к привычной расшифровке Real Time Clock (часы реального времени). Здесь термином RTC обозначается отдельная подсистема (ядро) низкого потребления, имеющая свои источники тактирования, память, выводы и т. д. Часы реального времени также имеются в составе этой подсистемы, видимо, поэтому название локального устройства – часов – распространилось на всю подсистему. – *Прим. перев.*

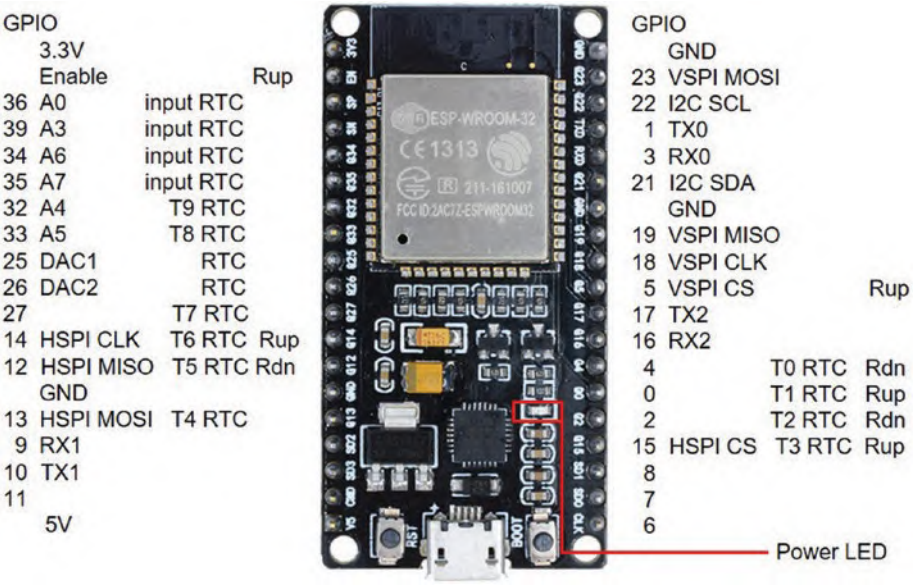


Рисунок 21-6. 36-контактная плата ESP32 NodeMCU

Установка драйвера моста USB-to-UART CP210x для платы ESP32 NodeMCU проводится точно так же, как для платы ESP8266 NodeMCU (описание процедуры см. на стр. 416).

В среде IDE Arduino выберите *Файл* ➤ *Настройки* (*File* ➤ *Preferences*) и введите URL-адрес https://dl.espressif.com/dl/package_esp32_index.json в поле *Additional Boards Manager URLs*. Если в поле уже есть URL-адрес, разделите URL-адреса запятой.

Библиотеки ESP32 устанавливаются в Arduino IDE, если выбрать *Tools* ➤ *Board* ➤ *Boards Manager* (*Инструменты* ➤ *Плата* ➤ *Менеджер плат*), ввести «ESP32» в опцию *Filter* (*Фильтр*), чтобы отобразить *esp32* от Espressif Systems, и нажать кнопку *Install* (*Установить*). В раскрывающемся списке *Tools* ➤ *Board* (*Инструменты* ➤ *Плата*) из раздела *ESP32 Boards* (*Платы ESP32*) выберите плату ESP32 NodeMCU или вашу конкретную плату ESP32. В разделах меню *Tools* (*Инструменты*) установите *Upload Speed* (*Скорость загрузки*) = 921 600; *CPU Frequency* (*Частота процессора*) = 240 МГц (WiFi/BT), *Flash Frequency* (*Частота флеш-памяти*) = 80 МГц, *Flash Mode* (*Режим флеш-памяти*) = DIO, *Flash Size* (*Размер флеш-памяти*) = 4 МБ (32 МБ) и *Partition Scheme* (*Схема разделов*) = '*Default 4MB with spiffs*'. Настройка SPI Flash Mode, равная QIO (quad IO – четырехкратный ввод/вывод), делает загрузку быстрее, чем настройка DIO (двойной ввод/вывод), но не все настройки доступны для каждого устройства. Если настройка QIO не поддерживается, после загрузки скетча отображается сообщение `rst:0x3 (SW_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)`. Более подробная информация доступна на github.com/espressif/esptool/wiki/SPI-Flash-Modes. Наконец, в меню *Tools* ➤ *Port* (*Инструменты* ➤ *Порт*) выберите соответствующий COM-порт.

Справочная документация для микроконтроллера ESP32 доступна по адресу docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html, где вы найдете, в частности, справочный раздел программного интерфейса *API Reference*.

Микроконтроллер ESP32 включает в себя несколько функций, специфичных для микроконтроллера ESP32, которые описаны в главе 22 («Функциональность микроконтроллера ESP32»). Инструкции для платы ESP32 в нескольких моментах отличаются от инструкций для плат, содержащих микроконтроллер ATmega328P или ESP8266. Следующие примеры иллюстрируют программирование платы ESP32 при использовании функций, характерных для микроконтроллера ATmega328P или ESP8266.

Цифровой вход ESP32

Чтобы считывать состояние вывода GPIO, он должен быть задан с помощью команды `pinMode(pin, INPUT)`. Если вывод GPIO должен удерживаться на низком уровне, то внутренний заземляющий резистор активируется с помощью команды `pinMode(pin, INPUT_PULLDOWN)`. Аналогично команда `pinMode(pin, INPUT_PULLUP)` активирует внутренний подтягивающий резистор на выводе GPIO, точно так же, как на микроконтроллере ATmega328P или ESP8266. Сценарии подключения подтягивающего или заземляющего резистора на выводе GPIO предназначены, например, для поддержания состояния кнопки по умолчанию на высоком или низком уровне соответственно или для запуска прерывания по падающему или возрастающему фронту.

Аналоговый вход ESP32

Функция 12-разрядного аналого-цифрового преобразователя (АЦП) преобразует напряжение от 0 до 3,3 В на выводе аналогового входа в цифровое значение от 0 до 4095. Команда `analogRead(ADCpin)` считывает напряжение на выводе ADC, при этом `ADCpin` определяется либо как номер GPIO 32, 33, 34, 35, 36 (VP) или 39 (VN), либо как A4, A5, A6, A7, A0 и A3 соответственно. Преобразование напряжения в цифровое значение является линейным для входных напряжений от 0,5 В до 2,5 В с шагом 0,8 мВ. Микроконтроллер ESP32 включает в себя больше контактов с функциональностью АЦП, но эти контакты недоступны на платах ESP32 DEVKIT DOIT или NodeMCU.

Разрешение АЦП на выводе АЦП увеличивается с помощью команды `analogSetPinAttenuation(ADCpin, attenuation)` со значениями ослабления *ADC_11dB* (по умолчанию), *ADC_6dB*, *ADC_2_5dB* или *ADC_0dB*. Входное напряжение V_{IN} на выводе АЦП уменьшается до $V_{IN} / \sqrt{10^{(db/10)}}$ для заданного значения децибел. Например, ослабление на 2,5 дБ снижает входное напряжение с 1 В до 0,75 В, что приводит к значению АЦП 3072 = 4096*0,75, в то время как значение по умолчанию 11 дБ приводит к значению АЦП 1154. Скетч в листинге 21-5 иллюстрирует изменение настройки ослабления вывода АЦП.

Листинг 21-5. Аналого-цифровое преобразование

```
int ADCpin = 36;              // define ADC pin
void setup()
{
   Serial.begin(115200);      // Serial Monitor baud rate
   Serial.println();
```

```
    analogSetPinAttenuation(ADCpin, ADC_11d b);
                                // default setting of 11dB
    Serial.println(analogRead(ADCpin)); // read ADC pin
    analogSetPinAttenuation(ADCpin, ADC_6db ) ;
                                // change setting to 6dB
    Serial.println(analogRead(ADCpin));
    analogSetPinAttenuation(ADCpin, ADC_2_5db);
    Serial.println(analogRead(ADCpin));
    analogSetPinAttenuation(ADCpin, ADC_0db);
    Serial.println(analogRead(ADCpin));
}
void loop()
{}                              // nothing in loop function
```

Широтно-импульсная модуляция в ESP32

Все контакты GPIO, за исключением контактов, работающих только на вход (GPIO 34, GPIO 35, GPIO 36 и GPIO 39), являются ШИМ-контактами и могут генерировать прямоугольное колебание с переменным коэффициентом заполнения. Для активации ШИМ требуются три инструкции

```
ledcAttachPin(wavePin, channel)
ledcSetup(channel, freq, resolution)
ledcWrite(channel, duty)
```

с параметрами: выходной канал ШИМ (`channel`), вывод GPIO для вывода ШИМ (`wavePin`), частота колебаний (`freq`), разрешение ШИМ (*resolution*) и коэффициент заполнения периода (`duty`). Микроконтроллер ESP32 использует 8-, 10-, 12- или 15-битное разрешение для ШИМ, обеспечивая диапазоны от 0 до 255, 1023, 4095 или 32767 соответственно. Для сравнения, микроконтроллеры Arduino Uno ATmega328P и ESP8266 используют 8-битное и 10-битное разрешение соответственно. Максимальная частота ШИМ-колебания равна 80 МГц/$2^{resolution}$. Например, при 8-битном и 15-битном разрешениях максимальная частота составляет 312,5 кГц и 2,44 кГц, что обусловлено компромиссом между разрешением ШИМ и максимальной частотой колебаний. Скетч в листинге 21-6 увеличивает, а затем уменьшает яркость светодиода, изменяя коэффициент заполнения периода ШИМ с частотой 5 кГц.

Листинг 21-6. ШИМ-сигнал

```
int channel = 0;                    // define PWM output channel
int wavePin = 25;                   // square wave output pin
int freq = 5000;                    // square wave frequency
int resolution = 8;                 // PWM resolution
int bright = 0;
int increm = 5;                     // increment in duty cycle
int lag = 25;                       // time between PWM changes
void setup()
{
  pinMode(wavePin, OUTPUT);         // square wave pin as output
  ledcAttachPin(wavePin, channel);  // attached channel to pin
  ledcSetup(channel, freq, resolution);
}
```

```
void loop()
{
  ledcWrite(channel, bright);        // set channel duty cycle
  delay(lag);
  bright = bright + increm;  // increment duty cycle
  if(bright <= 0 || bright >= 255) increm = - increm;
}                                    // reverse duty cycle increment
```

Вход последовательного порта ESP32

30-контактная плата ESP32 DEVKIT DOIT имеет два последовательных порта с выводами последовательной передачи и приема для порта *Serial1* GPIO 1 (TX0) и GPIO 3 (RX0) и для порта *Serial2* GPIO 17 (TX2) и GPIO 16 (RX2). В главе 12 («Приложение для GPS-трекинга с использованием Google Maps») микроконтроллер ESP8266 используется для обновления информации о местоположении GPS-модуля u-blox NEO-7M, для чего требовалась библиотека программного последовательного порта *SoftwareSerial*. Аналогично в главе 5 («MP3-плеер») библиотека *SoftwareSerial* требовалась для связи с MP3-плеером. Последовательные порты платы ESP32 обеспечивают связь с несколькими устройствами без необходимости использования специальных библиотек. Инструкция `Serial2.begin(baud, SERIAL_8N1, RXD2, TXD2)` устанавливает связь на втором последовательном порту на выводах `RXD2` и `TXD2` со скоростью передачи в бодах, определяемой параметром `baud`.

36-контактная плата ESP32 NodeMCU имеет три последовательных порта с выводами передачи и приема для порта *Serial1* GPIO 10 (TX1) и GPIO 9 (RX1). Команда `Serial1.begin(baud, SERIAL_8N1, RXD1, TXD1)` устанавливает последовательную связь через последовательный порт на выводах RXD1 и TXD1, со скоростью передачи в бодах, определяемой параметром `baud`.

Связь по Wi-Fi и веб-сервер

Для нескольких проектов в книге требуется связь по Wi-Fi с функциональностью веб-сервера. Для микроконтроллера ESP8266 необходимыми инструкциями являются

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
ESP8266WebServer server
```

В соответствующих инструкциях для микроконтроллера ESP32:

```
#include <WiFi.h>
#include <WebServer.h>
WebServer server(80)
```

обратите внимание, что номер порта должен быть специально определен для микроконтроллера ESP32. Библиотеки *ESP8266WebServer* и *WebServer* ссылаются на соответствующую библиотеку Wi-Fi, поэтому инструкции `#include <ESP8266WiFi.h>` или `#include <WiFi.h>` не требуются.

Прерывания ESP8266 и ESP32

Прерывания подключаются с помощью команды `attachInterrupt(digitalPinToInterrupt(switchPin, ISR, <state change>)` с параметрами, определяющими `switchPin`, подключенный к прерыванию, процедуру обработки прерывания (`ISR`) и изменение состояния вывода `state change`, которое может быть равно любому изменению `CHANGE`, падающему фронту `FALLING`, возрастающему фронту `RISING`, высокому уровню `HIGH` или низкому уровню `LOW`. Микроконтроллеры ESP8266 и ESP32 хранят скомпилированный код во внутренней оперативной памяти (IRAM), а не в более медленной флеш-памяти, если добавить к коду атрибут IRAM_ATTR. Обработчик прерывания определяется как `IRAM_ATTR void ISR()`, а не `void ISR()`. Если обработчик прерывания определен перед функцией `setup`, то команда изменяется на `void IRAM_ATTR ISR()`.

ESP8266, ESP32 и OLED-экран

Библиотека *Wire* требуется при использовании библиотеки *Adafruit SSD1306* для отображения на OLED-экране с интерфейсом I2C, подключенным к плате ESP8266 или ESP32. Библиотека *Adafruit SSD1306* ссылается на библиотеки *Adafruit GFX* и *Wire*, поэтому инструкции `#include <Adafruit_GFX.h>` и `#include <Wire.h>` не требуются. Размер OLED-экрана, ширина (`width`) и высота (`height`) в пикселях определяются в инструкции `Adafruit_SSD1306 oled(width, height, &Wire, -1)`, где значение –1 указывает на то, что вывод сброса не определен для OLED-экрана. Ширина, высота и параметр `&Wire` не нуждаются в явном определении при использовании OLED-экрана с микроконтроллером ATmega328P.

ESP32 и сервопривод

Для платы ESP32 требуется специальная библиотека сервоприводов ESP32, а не встроенная библиотека *Servo* Arduino IDE. Рекомендуется библиотека *ESP32Servo* Кевина Харрингтона (Kevin Harrington) и Джона К. Беннетта (John K. Bennett), также доступная в Arduino IDE. Инструкции встроенной библиотеки *Servo*

```
#include <Servo.h>                  // include Servo library
servoFB.attach(FBpin)               // initialise servo motor to FBpin
```

заменяются инструкциями библиотеки ESP32Servo:

```
#include <ESP32Servo.h>
servoFB.setPeriodHertz(F)           // define servo frequency (F)
servoFB.attach(FBpin, minPW, maxPW)
                                    // initialise servo motor to FBpin
```

В следующих инструкциях изменения не требуются:

```
Servo servoFB                       // associate servoFB with servo lib
servoFB.writeMicroseconds(T)        // move to position mapped to Tμs
servoFb.write(N)                    // move to angle N°
```

В инструкции `servoFB.attach(FBpin, minPW, maxPW)` параметры `minPW` и `maxPW` относятся к длительности импульса в микросекундах для перемещения серво-

двигателя на 0° и 180° соответственно. Значения по умолчанию для параметров `minPW` и `maxPW` составляют 1000 мкс и 2000 мкс, а для сервопривода Tower Pro SG90 – 500 мкс и 2500 мкс. Частота управляющего колебания *F* задается инструкцией `servoFB.setPeriodHertz(F)` и обычно составляет 50 Гц. Скетч для калибровки серводвигателя приведен в главе 9 («WebSocket»).

Итоги

Описаны свойства микроконтроллеров ESP8266 и ESP32 с примерами, иллюстрирующими некоторые функции. Выделены различия в инструкциях микроконтроллера ESP32 и микроконтроллера ESP8266, требующиеся в некоторых случаях. Разобрано подключение прерываний к плате ESP8266 и устранение проблем с зависанием с помощью сторожевого таймера Arduino Uno, Nano и Pro Micro. В нескольких скетчах приведены примеры некоторых функций микроконтроллеров.

Перечень компонентов

- Микроконтроллер ESP8266: плата LOLIN (WeMos) D1 mini или NodeMCU
- Микроконтроллер ESP32: плата DEVKIT DOIT или NodeMCUboard
- Светодиод
- Резистор: 220 Ом
- Тактовая кнопка

Глава 22

Особенности микроконтроллера ESP32

Особенности, характерные для микроконтроллера ESP32, описаны в этой главе. В главе 21 («Микроконтроллеры») были описаны различия в инструкциях для микроконтроллеров ESP8266 и ESP32 в отношении функций, доступных для обоих микроконтроллеров. Микроконтроллер ESP32 имеет два ядра, которые управляются независимо, интерфейс связи Bluetooth и Bluetooth Low Energy (BLE), четыре независимых таймера, цифроаналоговый преобразователь (ЦАП), емкостные сенсорные датчики и датчик эффекта Холла. Плата ESP32 DEVKIT DOIT показана на рис. 22-1.

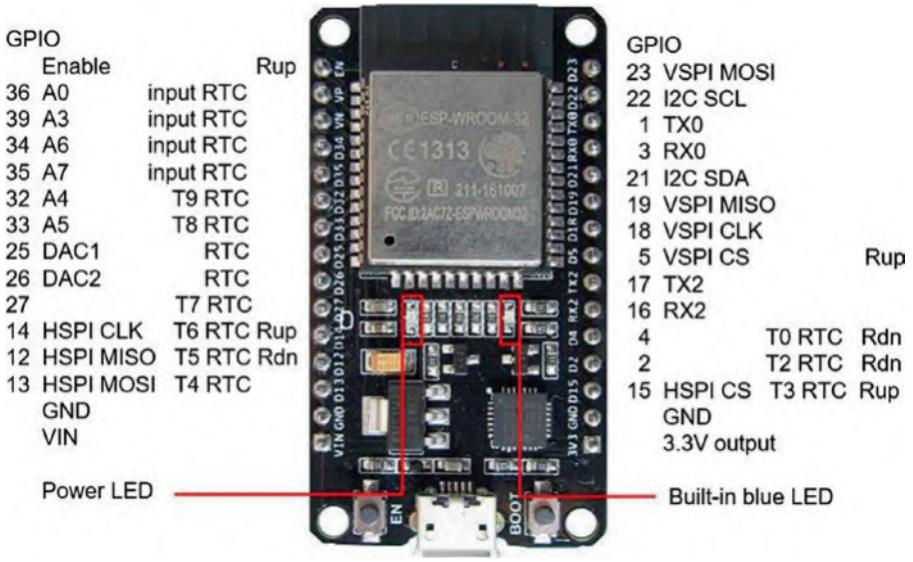


Рисунок 22-1. Модуль ESP32 DEVKIT DOIT

Процессор и память

Частота процессора микроконтроллера ESP32 может устанавливаться в Arduino IDE на 10, 20, 40, 80, 160 или 240 МГц, если выбрать *Tools* ➤ *CPU Frequency* (*Ин-*

струменты ➤ *Частота процессора*). Влияние различных частот процессора определяется путем измерения времени, затраченного на вычисление первых 10 тысяч простых чисел (см. главу 21 «Микроконтроллеры»). По мере того как частота процессора удваивается, время определения первых 10 тыс. простых чисел существенно уменьшается: с 11 607 мс при 10 МГц до 5060 мс при 20 МГц, до 2403 мс при 40 МГц, до 1172 мс при 80 МГц и 579 мс при 160 МГц. Время, затраченное при частоте 240 МГц, составило всего 385 мс.

Флеш-память ESP8266 и ESP32 делится на несколько разделов: энергонезависимое хранилище (NVS), секцию обновлений (OTA), секцию приложения, SPIFFS и EEPROM и т.д. (см. главу 20 «OTA и сохранение данных в EEPROM, SPIFFS и Microsoft Excel»), – которые настраиваются в среде IDE Arduino в зависимости от требований скетча к памяти. Выберите меню *Tools* (*Инструменты*), пункт *Partition scheme* (*Схема разделов*) для настройки распределения памяти при различных комбинациях приложений, SPIFFS и OTA. Например, распределение по умолчанию: основная память 36 кБ; NVS 20 кБ; OTA 8 кБ; ядра протоколов и приложений 2 × 1280 кБ и SPIFFS 1472 кБ.

Ядра ESP32

Микроконтроллер ESP32 включает в себя два ядра, каждое из которых представляет собой 32-разрядный микропроцессор Tensilica Xtensa LX6, в отличие от микроконтроллера ESP8266, содержащего один 32-разрядный процессор Tensilica L106. Одно из ядер, называемое ядром протокола PRO_CPU, управляет связью Wi-Fi, Bluetooth, SPI и I2C, в то время как ядро приложения APP_CPU предназначено для выполнения приложений[134]. Два ядра управляются ESP-IDF FreeRTOS[135], которая является модификацией FreeRTOS, подробная информация доступна по адресу www.freertos.org/a00106.html. Реализация Arduino IDE для ESP32 включает в себя ESP-IDF FreeRTOS. Библиотека *FreeRTOS* Ричарда Барри (Richard Barry), доступная в Arduino IDE, совместима с микроконтроллерами ATmega328P и ATmega32U4 Arduino Uno, Nano и Pro Micro, однако в настоящее время нет библиотеки FreeRTOS для микроконтроллера ESP8266.

Задачи в рамках *FreeRTOS* назначаются определенному ядру инструкцией

```
xTaskCreatePinnedToCore(code, "detail", 1000, NULL, pr, &TaskName, core);
```

где `code` – функция, содержащая инструкции по выполнению задачи, `"detail"` – строка, описывающая задачу, `pr` – приоритет задачи от 0 (самый низкий приоритет) до 24, `&TaskName` – указатель на дескриптор задачи, а `core` – номер ядра, 0 для ядра протокола или 1 для ядра приложения. Значение 1000 – это распределение памяти по умолчанию в байтах, а значение `NULL` указывает, что пара-

[134] Ядро протокола (CPU0) в ESP32 используется для запуска различных беспроводных протоколов, таких как Wi-Fi, Bluetooth и BLE, а ядро приложения (CPU1) используется для запуска прикладных программ отдельно от сетевого уровня. Два ядра ESP32 совместно применяют шину для доступа к памяти и другим периферийным устройствам. Такое разделение совместно с RTOS позволяет создавать высоконадежные решения без опасности полного отказа при неисправности какого-то из узлов. – *Прим. перев.*

[135] ESP-ID – ESP Espressif IoT Development Framework, среда для разработки IoT-приложений для ESP; RTOS – операционная система реального времени для микроконтроллеров. – *Прим. перев.*

метры не передаются. Подтверждение того, что задача назначена ядру, можно получить с помощью параметра `xPortGetCoreID()`. Дескриптор задачи определяется с помощью инструкции `TaskHandle_t TaskName`, но требуется только для удаления задачи с помощью инструкции `vTaskDelete(TaskName)`. Термин `&TaskName` можно заменить на NULL в инструкции `xTaskCreatePinnedToCore()`.

Инструкции по выполнению задач выделяются следующим образом:

```
void code(void * parameter)   // code function
{
  for(;;)                     // equivalent to a “for” instruction
  {
    Task instructions         // task instructions
  }
}
```

Выражение `(;;)` – то же самое, что `(int i=0; i<max; i++)`, но без параметров и более эффективно выполняет инструкции задачи в бесконечном цикле.

Время выполнения задач в ESP-IDF FreeRTOS основано на количестве тактов по 1 мс каждый, как определено параметром `portTICK_PERIOD_MS`. Команда `vTaskDelay(xOneSec)` представляет собой задержку в одну секунду с переменной `xOneSec` типа `TickType_t`, определяемой как xOneSec = `1000/portTICK_PERIOD_MS`. Временной интервал измеряется как количество прошедших тактов инструкциями

```
int tickTime = xTaskGetTickCount();          // tick count at start
vTaskDelay(xOneSec);                         // time interval
int tick = xTaskGetTickCount() - tickTime;   // change in tick count
```

Скетч в листинге 22-1 одновременно включает или выключает два светодиода с разными временными интервалами, при этом задачи, управляющие светодиодами, распределены на разные ядра ESP32. Задача назначается ядру ESP32 с помощью инструкции xTaskCreatePinnedToCore. Для сравнения, функция `codeRed` использует временны́е инструкции ESP-IDF FreeRTOS, в то время как функция `codeBlue` использует стандартные инструкции Arduino IDE. В пустой функции `loop` команда `vTaskDelay(NULL)` предотвращает выделение процессорного времени для нее.

Листинг 22-1. Задача для каждого ядра

```
int redLED = 26;                              // define LED pins
int blueLED = 27;
TickType_t xOneSec;                           // create time delay variable
void setup()
{
  Serial.begin(115200);                       // Serial Monitor baud rate
  pinMode(redLED, OUTPUT);                    // set LED pins as output
  pinMode(blueLED, OUTPUT);
  xOneSec = 1000 / portTICK_PERIOD_MS;        // define number of ticks
  xTaskCreatePinnedToCore(codeRed, "red LED one sec",
  1000, NULL, 2, NULL, 0);                    // allocate tasks to cores
  xTaskCreatePinnedToCore(codeBlue, "blue LED quarter sec",
  1000, NULL, 1, NULL, 1);
}
void codeRed(void * parameter)                // function for red LED
```

```
{
  for (;;)
  {
    int tickTime = xTaskGetTickCount();      // tick count at start
    digitalWrite(redLED, HIGH);              // turn on or off LED
    vTaskDelay(xOneSec);                     // task delay for one second
    digitalWrite(redLED, LOW);
    vTaskDelay(xOneSec);
    int tick = xTaskGetTickCount() - tickTime; // change in tick
                                             // count
    Serial.print("Core ");Serial.print(xPortGetCoreID());
    Serial.print(" red ");Serial.println(tick);
  }
}
void codeBlue(void * parameter)              // similar task with 250ms delay
{
  for (;;)
  {
    unsigned long start = millis();          // time at start
    digitalWrite(blueLED, HIGH);
    delay(250);                              // task delay of 250ms
    digitalWrite(blueLED, LOW);
    delay(250);
    start = millis() - start;
    Serial.print("Core ");Serial.print(xPortGetCoreID());
    Serial.print(" blue ");Serial.println(start);
  }
}
void loop()
{                            // no instructions in loop function
  vTaskDelay(NULL);                          // other than zero delay
}
```

Включение или выключение двух светодиодов с помощью 32-разрядного микроконтроллера, работающего на 240 МГц, – это чересчур просто, но скетч демонстрирует одновременное использование обоих ядер ESP32.

Скетч в листинге 22-2 определяет первые 5 тыс. и 10 тыс. простых чисел одновременно, при этом каждая задача назначается другому ESP32-ядру.

Распределение задач на разные ядра эффективно удваивает производительность задачи по сравнению с выполнением задачи на одном ядре ESP32. Последняя команда `vTaskDelay(1)` в каждой задаче предотвращает сброс сторожевого таймера микроконтроллера ESP32. Выполнение задачи по определению 10 тыс. простых чисел занимает 334 мс, но при удалении команды `vTaskDelay(NULL)` из пустой функции `loop` выполнение задачи занимает 662 мс. Функция `loop` выделяется ядру приложения, а команда `vTaskDelay(NULL)` предотвращает ненужное использование процессорного времени.

Листинг 22-2. Одновременное определение первых 5 тыс. и 10 тыс. простых чисел

```
unsigned long num5k = 2, num10k = 2;         // start from number 2
int count5k = 1, count10k = 1;               // prime number counters
unsigned int start5k = 0, start10k = 0;      // processing times
void setup()
{
```

```
   Serial.begin(115200);                    // Serial Monitor baud rate
   xTaskCreatePinnedToCore(code5k, "5k", 1000, NULL, 1, NULL, 0);
   xTaskCreatePinnedToCore(code10k, "10k", 1000, NULL, 1, NULL, 1);
}
void code5k(void * parameter)               // function for 5k primes
{
   for (;;)
   {
      num5k++;                              // increment number to check
      int chk = is_prime(num5k);      // call function to test for prime
      if (chk > 0) count5k++;         // increment counter when prime
      if (count5k > 4999)             // count up to 5k numbers
      {
         printLine(start5k, count5k, n um5k);
                                      // function to display results
         num5k = 2;
         count5k = 1;                 // reset parameters
         start5k = millis();
         vTaskDelay(1);                       // delay for watchdog timer
      }
   }
}
void code10k(void * parameter)       // function for 10k primes
{
   for (;;)
   {
      num10k++;
      int chk = is_prime(num10k);
      if (chk > 0) count10k++;
      if (count10k > 9999)
      {
         printLine(start10k, count10k, num10k);
         num10k = 2;
         count10k = 1;
         start10k = millis();
         vTaskDelay(1);
      }
   }
}
void printLine(unsigned long start, int count, unsigned long number)
{
   int ms = millis() - start;
   Serial.print("Core ");Serial.print(xPortGetCoreID());
   Serial.print(" Found ");Serial.print(count);
   Serial.print(" primes in "); Serial.print(ms);
   Serial.print(" ms");
   Serial.print(" highest prime is ");Serial.println(number);
}
int is_prime(unsigned long num)      // function to check if prime number
{
   int mod = num % 2;                // exclude even numbers
   if (mod == 0) return 0;
   int limit = sqrt(num);            // check divisors less than square root
   for (int divid = 3; divid <= limit; divid = divid + 2)
   {
      mod = num % divid;             // remainder after dividing
```

```
      if (mod == 0) return 0;  // not prime if zero remainder
    }
    return 1;                          // no divisor with zero remainder
}
void loop()
{
  vTaskDelay(NULL);
}
```

Информация передается между задачами, работающими на одном и том же ядре ESP32 или на различных ядрах ESP32, с использованием либо метода семафора, либо очереди для управления передачей информации. Метод семафора в некоторой степени аналогичен эстафете, в которой один бегун передает эстафетную палочку второму бегуну, чтобы позволить ему начать бег. Однако в методе семафора при двух ядрах первый бегун продолжает бежать! Инструкции `SemaphoreHandle_t baton` и `baton = xSemaphoreCreateMutex()` создают переменную `baton` для семафора, заданного одной задачей и принятого другой задачей с помощью инструкций `xSemaphoreGive(baton)` и `xSemaphoreTake(baton, portMAX_DELAY)` соответственно. Позиция инструкции `xSemaphoreGive(baton)` в инструкциях первой задачи определяет, когда инициируется вторая задача. В методе семафора любая информация, необходимая для выполнения второй задачи, содержится в глобальной переменной, которая определяется с помощью задания типа переменной `volatile`.

Метод очереди аналогичен менеджеру, распределяющему задания работнику, при этом работник выполняет задания в порядке очереди. Инструкции `QueueHandle_t queue` и `queue = xQueueCreate(N, sizeof(int))` создают переменную очереди `queue`, содержащую до *N* заданий. Задание добавляется в очередь или удаляется из очереди с помощью инструкций `xQueueSend(queue, &work, portMAX_DELAY)` и `xQueueReceive(queue, &work, portMAX_DELAY)` соответственно, где `&work` – указатель на задание.

Разница между методами семафора и очереди для передачи информации между задачами заключается в использовании глобальных переменных методом семафора. Инструкции в табл. 22-1 иллюстрируют сходство и различие между методами семафора и очереди. Различия между методами выделены жирным шрифтом. В скетче задача, выделенная ядру протокола ESP32, включает красный светодиод на одну секунду и выключается на случайный период времени. Задача, выделенная ядру приложений ESP32, включает синий светодиод для периода времени, в течение которого красный светодиод не горит. Команда `delay(1)`, следующая за командой для присвоения значения переменной `redOff`, учитывает время обработки.

При использовании метода семафора переменная `redOff` является глобальной переменной, доступной для задачи, управляющей синим светодиодом. В методе очереди переменная `redOff` добавляется в очередь задачей, управляющей красным светодиодом, и считывается из очереди, как переменная `BlueOn`, задачей, управляющей синим светодиодом. Для обоих методов, когда инструкция `xSemaphoreGive()` или `xQueueSend()` предшествует инструкции по включению красного светодиода, два светодиода включаются одновременно. В отличие от этого, два светодиода включаются попеременно, когда инструкция `xSemaphoreGive()` или `xQueueSend()` предшествует инструкции по выключению красного светодиода. Скетч иллюстрирует выбор времени для одновременного или попеременного выполнения двух задач.

Таблица 22-1. Методы семафора и очереди

Семафор	Очередь
int redLED = 26;	int redLED = 26;
int blueLED= 27;	int blueLED = 27;
SemaphoreHandle_t baton;	**QueueHandle_t queue;**
volatile int redOff;	
void setup() {	void setup() {
pinMode(redLED, OUTPUT);	pinMode(redLED, OUTPUT);
pinMode(blueLED, OUTPUT);	pinMode(blueLED, OUTPUT);
xTaskCreatePinnedToCore(codeRed,	xTaskCreatePinnedToCore(codeRed,
"red LED ", 1000, NULL, 1, NULL, 0);	"red LED ", 1000, NULL, 1, NULL, 0);
xTaskCreatePinnedToCore(codeBlu,	xTaskCreatePinnedToCore(codeBlue,
"blue LED", 1000, NULL, 1, NULL, 1);	"blue LED", 1000, NULL, 1, NULL, 1);
baton = xSemaphoreCreateMutex();	**queue = xQueueCreate(1,**
	sizeof(int));
}	}
void codeRed(void * parameter) {	void codeRed(void * parameter) {
for (;;) {	for (;;) {
redOff = random(500, 2000);	int redOff = random(500, 2000);
delay(1);	delay(1);
xSemaphoreGive(baton);	**xQueueSend(queue, &redOff,**
	portMAX_DELAY);
digitalWrite(redLED, HIGH);	digitalWrite(redLED, HIGH);
delay(1000);	delay(1000);
digitalWrite(redLED, LOW);	digitalWrite(redLED, LOW);
delay(redOff);	delay(redOff);
} }	} }
void codeBlu(void * parameter) {	void codeBlue(void * parameter) {
for (;;)	for (;;)
{	{
	int blueOn;
xSemaphoreTake(baton,	**xQueueReceive(queue, &blueOn,**
portMAX_DELAY);	**portMAX_DELAY);**
digitalWrite(blueLED, HIGH);	digitalWrite(blueLED, HIGH);
delay(redOff);	**delay(blueOn);**
digitalWrite(blueLED, LOW);	digitalWrite(blueLED, LOW);
} }	} }
void loop() { vTaskDelay(NULL); }	void loop() { vTaskDelay(NULL); }

Связь по Bluetooth

Микроконтроллер ESP32 работает как по протоколу Bluetooth, так и по протоколу Bluetooth Low Energy[136]. Для связи по Bluetooth требуется установить последовательное соединение Bluetooth, что выполняется с помощью инструкций

```
#include <BluetoothSerial.h>          // include Bluetooth library
BluetoothSerial SerialBT;             // associate SerialBT with library
SerialBT.begin("ESP32 Bluetooth");    // identify Bluetooth
```

Устройства Bluetooth обмениваются данными, посылая по одному символу за раз. Текст, введенный на мониторе последовательного порта, отправляется с помощью одной из команд `SerialBT.write(Serial.read())` или `SerialBT.write(c)`, где `c` – посылаемый символ. Скетч в листинге 22-3 устанавливает последовательное соединение Bluetooth и отображает полученное сообщение от устройства Bluetooth на мониторе последовательного порта. Команда `Serial.write()` отсылает код ASCII для буквенно-цифровых символов, в то время как `Serial.print()` отсылает ASCII-код для каждого символа в сообщении[137]. Чтобы оба передавали сообщение на устройство Bluetooth и переданное сообщение отображалось на мониторе последовательного порта, каждый символ сообщения передается индивидуально, поскольку последовательный буфер не считывается дважды.

Листинг 22-3. Связь по Bluetooth

```
#include <BluetoothSerial.h>          // include Bluetooth library
BluetoothSerial SerialBT;             // associate SerialBT with library
String str;
int strLen;
char c;
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  SerialBT.begin("ESP32 Bluetooth"); // identify Bluetooth device
}
void loop()
{                                     // received message from Bluetooth device
  if(SerialBT.available()) Serial.write(SerialBT.read());
  if(Serial.available())              // message to transmit
  {
    str = Serial.readString();        // read and display
    Serial.print("\t\t\t\t");
    Serial.println(str);              // Serial buffer
    strLen = str.length();
    for (int i=0; i<strLen; i++)
```

[136] Bluetooth Low Energy (BLE) – экономичный вариант беспроводного коммуникационного протокола Bluetooth. Отличается низким и сверхнизким потреблением энергии (в разы и десятки раз меньше, чем стандартный Bluetooth). Экономия в том числе достигается за счет пониженной скорости обмена, отчего на основе BLE отлично получаются автономные датчики для широкого диапазона применений, но невозможно организовать передачу аудиопотока для таких применений, как беспроводные наушники. Подробнее о BLE см. далее в этой главе. – *Прим. перев.*

[137] Для иллюстрации: в результате команды `Serial.write(88)` на мониторе порта отобразится буква *X* (ASCII-код равен 88 = 0x58), тогда как в результате команды `Serial.print(88)` отобразится строка «88». – *Прим. перев.*

```
      {
        c = str[i];                    // for each message character
        SerialBT.write(c);             // transmit to Bluetooth device
      }
      SerialBT.write('\n');            // add new line character
    }
    delay(50);
}
```

Существует несколько приложений для связи по Bluetooth, которые можно загрузить из Google Play Store для связи Android-устройств с ESP32 Bluetooth. Рекомендуется использовать приложение *Bluetooth Terminal HC-05* от mightyIT. После запуска приложения *Bluetooth Terminal HC-05* Android-клиент сканирует устройства в зоне видимости для поиска необходимого (см. рис. 22-2) и затем устанавливает соединение с Bluetooth-сервером.

Рисунок 22-2. Клиентское сканирование для сервера Bluetooth

Как только установлено соединение клиент–сервер, связь по Bluetooth между двумя устройствами позволяет передавать буквенно-цифровой текст, как показано на рис. 22-3.

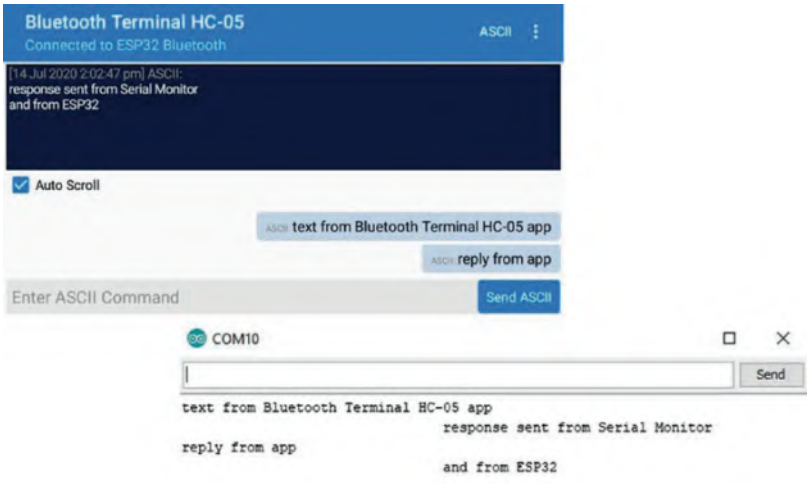


Рисунок 22-3. Передача текстовых сообщений по Bluetooth

Связь Bluetooth Low Energy

Bluetooth Low Energy (BLE) поддерживает связь на той же частоте 2,4 ГГц, что и Bluetooth, с аналогичным диапазоном передачи, но с уменьшенным энергопотреблением. С помощью BLE данные передаются только тогда, когда установлено соединение с клиентом и клиент отправил уведомление об обновлении данных. Сервер объявляет о своем существовании, клиент сканирует устройства, и когда клиент обнаруживает требуемое устройство, то устанавливает соединение с сервером. Например, если микроконтроллер ESP32 является сервером, а планшет или мобильный телефон Android – клиентом, то в данном случае реализуется связь «точка–точка». Топологии широковещательной и ячеистой сетей предназначены для связи между устройствами «один ко многим» и «многие ко многим», а в этом разделе основное внимание уделяется связи «точка–точка».

BLE используется для периодической передачи небольших объемов данных, например в датчиках мониторинга окружающей среды, устройствах домашней автоматизации, а также в медицинском и спортивном оборудовании, допустим в мониторах сердечного ритма и артериального давления на смарт-часах. Данные, передаваемые с помощью BLE, имеют определенный формат в зависимости от категории передаваемых данных, в том числе формат, называемый *Generic ATTribute (GATT) profile*, который используется устройствами, взаимодействующими с BLE. Категории данных состоят из BLE-сервисов, таких, например, как служба экологического зондирования *Environmental Sensing*. Она включает характеристики, такие как температура или влажность, причем как сервис, так и каждая характеристика имеют универсальный уникальный идентификатор (UUID). Подробная информация об услугах и характеристиках GATT доступна по адресу www.bluetooth.com/specifications/gatt/services. Например, идентификаторы UUID для датчиков окружающей среды и температуры равны 0x181A и 0x2A63 соответственно, где температура с двумя знаками после запятой сдвигается на два десятичных разряда, чтобы получить целый формат `uint16_t`.

Скетч в листинге 22-10 иллюстрирует микроконтроллер ESP32, передающий три характеристики измерения окружающей среды с помощью BLE-связи. Три характеристики – это *температура*, *тепловой индекс* (heat index) и *UV (ультрафиолетовый) индекс* (UV index)[138], они форматируются для BLE как uint16_t, uint8_t и uint8_t соответственно. Для иллюстрации форматирования данных датчиков для BLE характеристика теплового индекса выводится из действительного числа `temp`, а УФ-индекс выводится из целого числа `UV`. Обратите внимание, что существуют ограничения на возможные значения характеристик, такие как характеристика уровня заряда батареи, 0x2A19, которая представляет собой процент от 0 до 100.

[138] *Тепловой индекс* – то же самое, что эффективная температура, учитывает совместное действие температуры воздуха и относительной влажности на ощущение температуры человеком. Рассчитывается по специальной таблице или формуле; в примере далее это просто преобразованное значение температуры. *Ультрафиолетовый индекс* – число, рассчитываемое по уровню УФ-излучения от солнца, достигающего поверхности, показывает степень опасности УФ-излучения для незащищенного человека. – *Прим. перев.*

Библиотека *ESP32 BLE Arduino* от Нейла Колбана (Neil Kolban) автоматически включается в Arduino IDE при установке ESP32 с помощью менеджера плат. Листинги 22-10 и 22-11 основаны на примерах скетчей в *ESP32 BLE Arduino* и на примерах Андреаса Шписса (Andreas Spiess) по адресу github.com/SensorsIot/Bluetooth-BLE-on-Arduino-IDE.

В первом разделе скетча в листинге 22-4 загружаются библиотеки BLE и определяются идентификаторы UUID для службы и трех характеристик. UUID характеристик и требуемые форматы доступны по адресу www.bluetooth.com/specifications/gatt/characteristics. Каждая характеристика имеет до четырех свойств: PROPERTY_READ, PROPERTY_WRITE, PROPERTY_NOTIFY и PROPERTY_INDICATE, причем свойство PROPERTY_NOTIFY присваивается характеристикам, передаваемым сервером клиенту. Класс `ServerConnect` определяет, установил ли клиент соединение с сервером. В функции `setup` определяются сервер и служба BLE, при этом каждая характеристика определяется и добавляется в службу BLE.

В функции `loop`, если клиент установил соединение с сервером, он получает уведомление об обновленных значениях характеристик. В примере значения `temp` и `UV` преобразуются в требуемые форматы BLE для характеристик температуры, теплового индекса и УФ-индекса. Температурная характеристика преобразуется в формат `uint16-t` путем умножения действительного числа *temp* на 100, а затем выделения старшего и младшего байтов в массив `TempData` из двух элементов. При этом старший байт получается с помощью сдвига числа `tempValue` в формате `uint16-t` на восемь бит вправо. Характеристика теплового индекса имеет формат `uint8_t`, получаемый непосредственно преобразованием формата действительного числа. Характеристика UV-индекса получается с помощью команды `(uint8_t*)&UV`, которая указывает (*) на адрес (&) целого числа `UV`. В главе 14 («Обмен данными через ESP-NOW и LoRa») описываются указатели и адреса переменных.

В результате форматирования характеристики для BLE ее значение обновляется, и клиент уведомляется об этом с помощью инструкций `setValue()` и `notify()`. На библиотеки *BLEUtils* и *BLEServer* ссылается библиотека *BLEDevice*, поэтому инструкции `#include <BLEUtils.h>` и `#include <BLEServer.h>` в явном виде не требуются.

Листинг 22-4. BLE-устройство в качестве сервера

```
#include <BLEDevice.h>        // include BLE libraries
#include <BLE2902.h>
BLEServer * pServer;          // define BLE server,
BLEService * pService;        // BLE service and
BLECharacteristic * pChar;    // BLE Characteristic
int devConnect = 0;
#define SERVICE_UUID BLEUUID((uint16 t)0x181A)
                              // environmental service
BLECharacteristic tempChar(BLEUUID((uint16_t)0x2A6E),
BLECharacteristic::PROPERTY_NOTIFY);
BLECharacteristic UVChar (BLEUUID((uint16_t)0x2A76),
BLECharacteristic::PROPERTY_NOTIFY);
BLECharacteristic heatChar(BLEUUID((uint16_t)0x2A7A),
BLECharacteristic::PROPERTY_NOTIFY);
class ServerConnect: public BLEServerCallba cks
```

```
{    // to check if connected
   void onConnect(BLEServer * pServer) {devConnect = 1;}
   void onDisconnect(BLEServer * pServer) {devConnect = 0;}
};
float temp = 0;
uint8_t tempData[2];
uint16_t tempValue;
int UV = 0, heat;
void setup()
{
   Serial.begin(115200);                      // Serial monitor baud rate
   BLEDevice::init("ESP32");                  // define BLE device
   pServer = BLEDevice::createServer();       // define BLE server
   pServer->setCallbacks(new ServerConnect()) ;
                                              // check if connected
   pService = pServer->createService(SERVICE_UU ID);
                                              //define BLE service
   pService->addCharacteristic(&tempChar);    // define temperature
   tempChar.addDescriptor(new BLE2902());     // characteristic
   pService->addCharacteristic(&UVChar);      // define UV index
   UVChar.addDescriptor(new BLE2902());       // characteristic
   pService->addCharacteristic(&heatChar);    // define heat index
   heatChar.addDescriptor(new BLE2902());     // characteristic
   pServer->getAdvertising()->addServiceUUID(SERVICE_UUID);
   pService->start();                         // start service
   pServer->getAdvertising()->start();        // advertise service
   Serial.println("Waiting for client to connect");
}
void loop()
{
   if(devConnect == 1)                        // if the client is connected
   {
      temp = temp + 1.11;
      tempValue = (uint16_t)(temp*100);       // convert to uint16_t with 2DP
      tempData[0]= tempValue;                 // LSB (least significant byte)
      tempData[1]= tempValue >> 8;            // MSB (most significant byte)
      tempChar.setValue(tempData, 2);         // update characteristic
      tempChar.notify();                      // notify client
      heat = (uint8_t)temp;                   // convert to uint8_t with 0DP
      heatChar.setValue(heat);
      heatChar.notify();
      UV = UV + 1;
      UVChar.setValue((uint8_t*)&UV, 4);      // point to address of UV
      UVChar.notify();
      Serial.print(temp);Serial.print("\t ");
                                              // display values on Monitor
      Serial.print(heat);Serial.print("\t");
      Serial.println(UV);
      delay(1000);                            // delay between sensor readings
   }
}
```

Приложения *nRF Connect* и *nRF Toolbox* от Nordic Semiconductor рекомендуются для BLE-связи и доступны для загрузки из магазина Google Play для планшета или мобильного телефона Android. После запуска приложения *nRF Connect* клиент находит необходимое устройство (см. рис. 22-4) и устанавливает соединение с сервером.

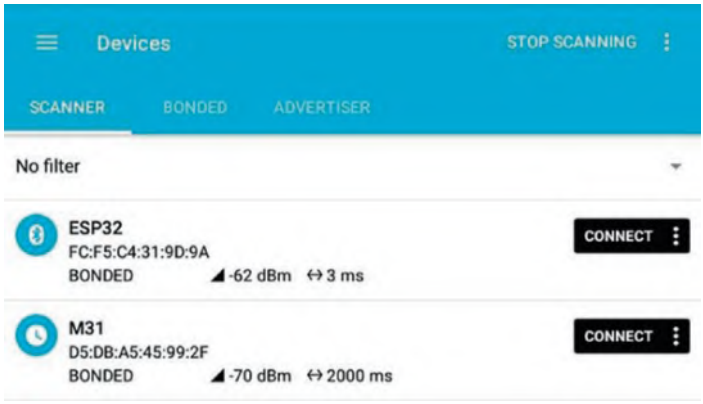


Рисунок 22-4. Сканирование на клиенте для поиска BLE-сервера

Отображение службы экологического зондирования Environmental Sensing с обновленными значениями характеристик температуры, теплового индекса и УФ-индекса показано на рис. 22-5. Щелчок по стрелкам в характеристиках BLE (обведенным кружком на рис. 22-5) разрешает отображение обновленных значений характеристик приложением, но только в контексте скетча из листинга 22-4. Обратите внимание, что на рис. 22-5 обновление характеристики теплового индекса в настоящее время отключено.

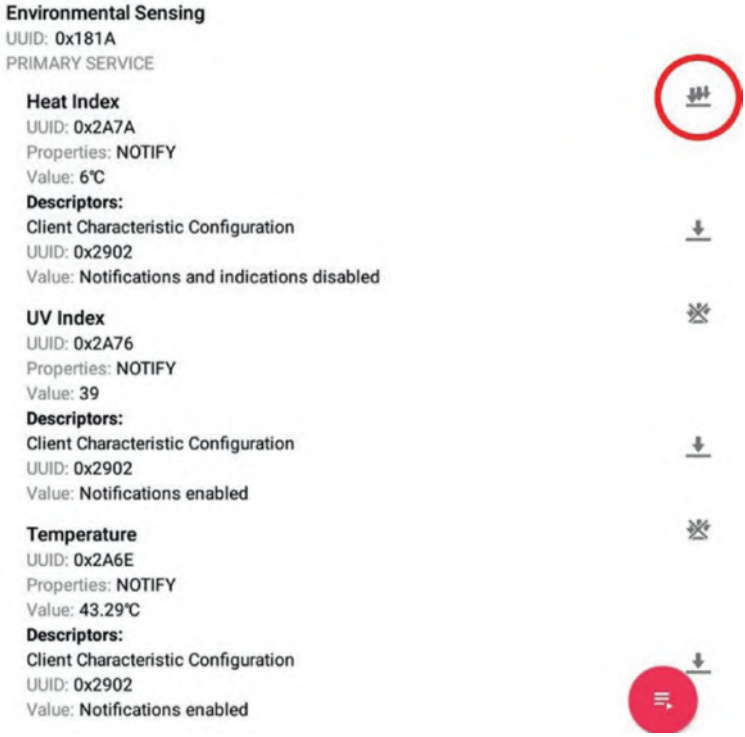


Рисунок 22-5. Служба BLE Environmental Sensing

Как передача, так и прием данных сервером и клиентом демонстрируются с помощью скетча в листинге 22-5. Сервер передает обновленные значения характеристик клиенту, как в листинге 22-4; но в листинге 22-5 клиент передает текстовые сообщения на сервер, а также уведомляет сервер, если клиенту требуются обновленные значения характеристик. Чтобы проиллюстрировать реакцию сервера на текст, полученный от клиента, индикатор, подключенный к плате ESP32, включается или выключается, когда клиент передает текст «LED».

В первом разделе листинга 22-5 устанавливаются библиотеки BLE, как в листинге 22-4, но служба UART не определена как служба GATT BLE, поэтому UUID для службы UART и характеристики передачи и приема определяются явно, а не неявно, как в листинге 22-4. Три UUID – это UUID по умолчанию, используемые Nordic Semiconductor для передачи UART. Класс `RXCallback` управляет приемом данных сервером с помощью строки `RXstr`, содержащей полученный буквенно-цифровой текст. Класс `notifyCallback` задает значение переменной `devNotify`, которая определяется независимо от того, передает ли сервер обновленные значения характеристик клиенту. В функции `setup` характеристики передачи (TX) и приема (RX) определяются с помощью `PROPERTY_NOTIFY` и `PROPERTY_WRITE` соответственно, с помощью классов `notifyCallback` и `RXCallback`, подключенных к TX и RX, соответственно (см. рис. 22-6 далее).

В функции `loop` сервер передает обновленные значения характеристик, но только в том случае, если клиент установил соединение и запросил обновленные значения. Функция языка C `dtostrf` – это метод форматирования действительного числа в виде массива символов с требуемым BLE-форматом. Команда `dtostrf(value, 8, 1, valueStr)` преобразует значение действительного числа в массив символов длиной восемь байт, представляющий значение с одним знаком после запятой.

Листинг 22-5. Передача и прием с помощью BLE

```
#include <BLEDevice.h>                // include BLE libraries
#include <BLE2902.h>
BLEServer * pServer;                  // define BLE server,
BLEService * pService;                // BLE service,
BLECharacteristic * pTXChar;          // BLE transmit and
BLECharacteristic * pRXChar;          // receive characteristics,
BLEDescriptor * pTXDesc;              // transmit descriptor
int devConnect = 0;
int devNotify = 0;
int LEDpin = 25;
int LEDstate = 0;
float value;
char valueStr[8];
char SERVICE_UUID[] = "6E400001-B5A3-F393-E0A9-E50E24DCCA9E";
                                      // UUIDs
char RXChar_UUID[] = "6E400002-B5A3-F393-E0A9-E50E24DCCA9E";
char TXChar_UUID[] = "6E400003-B5A3-F393-E0A9-E50E24DCCA9E";
void changeLED()                      // function to change LED state
{
  LEDstate = 1 - LEDstate;            // turn on or off LED
  digitalWrite(LEDpin, LEDstate);
}
```

```
class ServerConnect: public BLEServe rCallbacks
{                                                  // to check if connected
   void onConnect(BLEServer * pServer) {devConnect = 1;}
   void onDisconnect(BLEServer * pServer) {devConnect = 0;}
};
class RXCallback: public BLECharacteristicC allbacks
{                                                  // receive client data
   void onWrite(BLECharacteristic * pCharacteristic)
   {
      std::string RXstr = pCharacteristic->getValue();
      if (RXstr.length() > 0)                      // client data available
      {
         Serial.print("Received: ");               // read client data
         for (int i=0; i<RXstr.length(); i++) Serial.
         print(RXstr[i]);
         Serial.println();
         if(RXstr == "LED") changeLED();           // call changeLED function
      }
   }
};
class NotifyCallback: public BLEDescript orCallbacks
{                                                  // client data notification
   void onWrite(BLEDescriptor * pDescriptor)
   {                                               // obtain TX descriptor
      uint8_t * TXvalue = pDescriptor->getV al ue();
      devNotify = 0;
      if (pDescriptor->getLength() > 0)            // client data available
      {
         if(TXvalue[0] == 1) devNotify = 1;        // update data notification
         Serial.print("Notify: ");Serial.println(devNotify);
      }
   }
};
void setup()
{
   Serial.begin(115200);                           // Serial monitor baud rate
   pinMode(LEDpin, OUTPUT);
   BLEDevice::init("ESP32");                       // define BLE device
   pServer = BLEDevice::createServer();            // define BLE server
   pServer->setCallbacks(new ServerConnect ());
                                                   // check if connected
   pService = pServer->createService(SER VICE_U UID);
                                                   // define BLE service
                                                   // define TX characteristic
   pTXChar = pService->createCharacteristic(
   TXChar_UUID, BLECharacteristic::PROPERTY_NOTIFY);
   pTXDesc = new BLE2902();                        // define TX descriptor
   pTXDesc->setCallbacks(new NotifyCallb ack ());
                                                   // attach notify callback
   pTXChar->addDescriptor(pTXDesc);
                                                   // define RX characteristic
   pRXChar = pService->createCharacteristic(
   RXChar_UUID, BLECharacteristic::PROPERTY_WRITE);
   pRXChar->setCallbacks(new RXCallback());        // attach RX callback
   pService->start();                              // start service
   pServer->getAdvertising()->start();             // advertise service
   Serial.println("Waiting for client to connect");
}
```

```
void loop()
{                                          // if the client is connected and
   if(devConnect == 1 && devNotify == 1)
                                           // requests data notification
   {
      value = random(10, 200)*1.5;         // generate real number
      dtostrf(value, 8, 1, valueStr);      // convert 8-char string with 1DP
      pTXChar->setValue(valueStr);         // update characteristic
      pTXChar->notify();                   // notify client
      Serial.print(value);
      Serial.print("\t");Serial.
      println(valueStr);
      delay(3000);                         // delay between updates
   }
}
```

После открытия приложения *nRF Connect*, сканирования в поисках необходимого устройства и установления соединения с сервером отображается служба *Nordic UART*. Обновление характеристики TX (см. рис. 22-6) сервером включается или отключается нажатием стрелок BLE (см. рис. 22-5). При нажатии на одиночную стрелку BLE напротив *RX characteristic* появляется всплывающее текстовое окно, и введенный текст отображается как значение характеристики RX (см. рис. 22-6). Ввод текста «LED» приводит к включению или выключению светодиода, подключенного к плате ESP32.

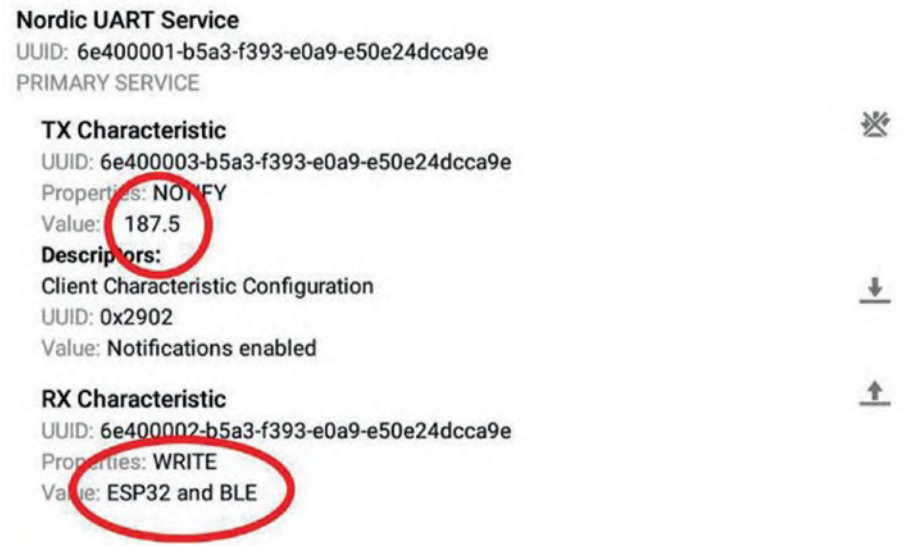


Рисунок 22-6. Передача и прием с помощью BLE

Приложение *nRF Toolbox* имеет виртуальную клавишную панель для передачи буквенно-цифрового текста с помощью BLE-связи на сервер. При открытии приложения *nRF Toolbox* выберите кнопку *UART* и нажмите кнопку *CONNECT*, чтобы отобразить доступные устройства (см. рис. 22-7). Выберите нужное устройство или выберите *SCAN*, чтобы вывести список устройств, если их много.

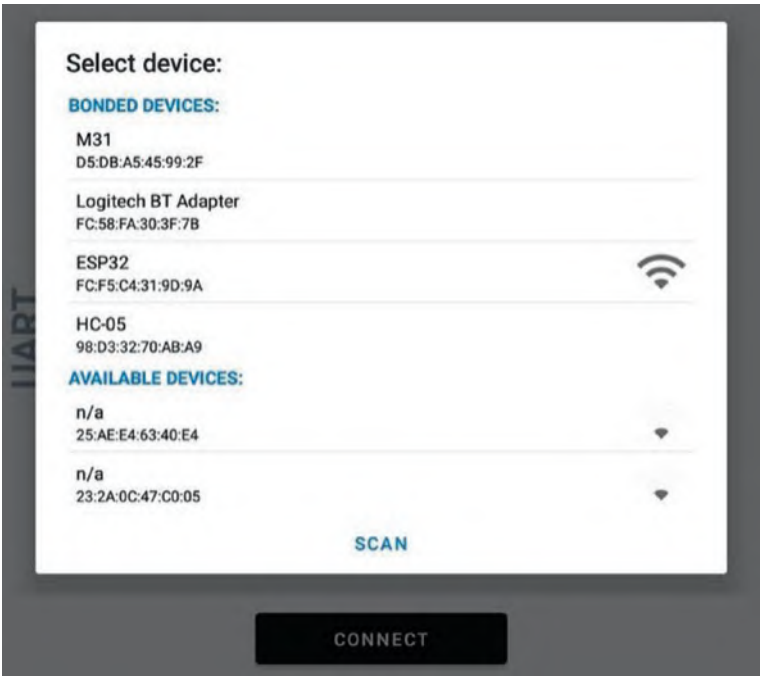


Рисунок 22-7. Приложение nRF Toolbox

Отображаемая клавишная панель (см. рис. 22-8) настраивается при выборе пункта *EDIT*. Необходимо нажать соответствующую кнопку панели, ввести буквенно-цифровой текст, который будет передаваться при нажатии кнопки, и выбрать символ, который будет отображаться на кнопке. В контексте листинга 22-5 текст «LED» был выделен центральной кнопке панели.

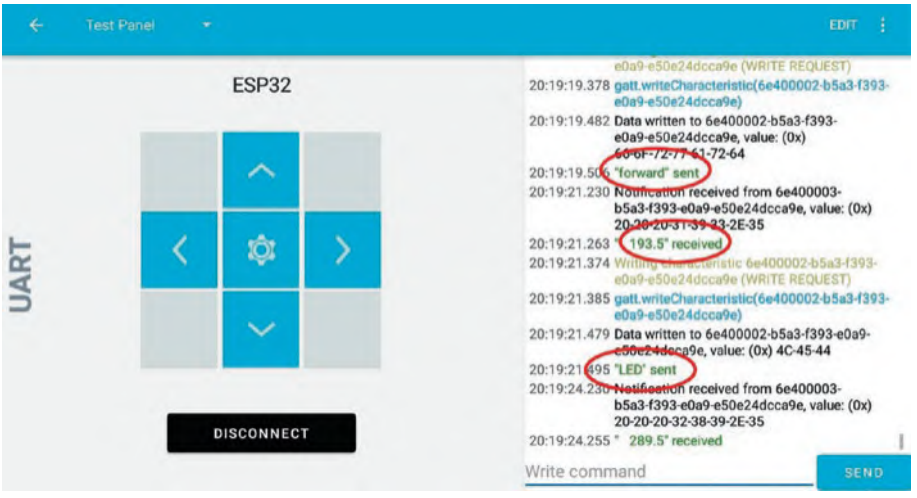


Рисунок 22-8. Клавишная панель в приложении nRF Toolbox

Для скетча, использующего связь BLE для управления двигателями, подключенными к удаленной плате разработки ESP32, кнопкам клавиатуры будет назначен текст «*вперед*» (*forward*), «*назад*» (*backward*), «*влево*» (*left*) и «*вправо*» (*right*) или «*быстро*» (*fast*) и «*медленно*» (*slow*) (см. рис. 22-8). Текст, полученный микроконтроллером ESP32, интерпретируется аналогично тексту «LED» в классе *RXCallback* в листинге 22-5 с соответствующими функциями, вызываемыми для управления двигателями или другими устройствами.

В другом примере BLE микроконтроллер ESP32, действующий в качестве клиента, сканирует устройства BLE и включает светодиод при обнаружении конкретного устройства (см. рис. 22-9). Микроконтроллер ESP32 может включать реле, а не просто светодиод, для активации устройства при обнаружении устройства BLE, такого как ваши смарт-часы. Адрес BLE ваших смарт-часов доступен на часах или определяется приложением для сканирования BLE, таким как приложения *nRF Connect* и *nRF Toolbox* от Nordic Semiconductor. Для скетча BLE-адрес часов должен быть изменен на нижний регистр, например *D5:DB:A5:45:99:2F* на *d5:db:a5:45:99:2f*.

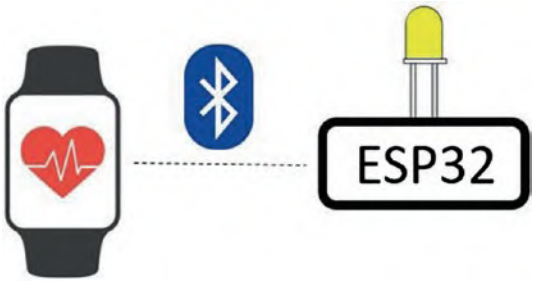


Рисунок 22-9. Сканирование для конкретного устройства BLE

Листинг 22-6. Управление часами BLE

```
#include <BLEDevice.h>                          // include BLE library
BLEScan * pBLEScan;                             // pointer to BLE scanner
BLEAddress * pAddress;                          // and to BLE address
BLEScanResults devices;
String watch = "d5:db:a5:45:99:2f";             // change upper to lower case
String scan;
int scanTime = 3;                               // scan devices for 3s
int paired = 0, lastPair = 0;
int LEDpin = 25;                                // define LED pin
class watchCallback: public BLEAdvertisedDeviceCallbacks
{                                               // BLE advertising devices
  void onResult(BLEAdvertisedDevice a dvertised)
  {                                             // option to display device data
// Serial.printf("found device: %s \n",
// advertised.toString().c_str());
    pAddress = new BLEAddress(advertised.getAd dress());
                                                // BLE address
    scan = pAddress->toString().c_str();        // convert to string
    if(scan == watch) paired = 1;               // device matches watch address
  }
};
```

```
void setup()
{
   Serial.begin(115200);                    // Serial Monitor baud rate
   pinMode(LEDpin, OUTPUT);                 // LED pin as output
   BLEDevice::init("");                     // initialise BLE client
   pBLEScan = BLEDevice::getScan();         //create new scan
   pBLEScan->setAdvertisedDeviceCallbacks(new watchCallback());
   pBLEScan->setActiveScan(true);
}
void loop()
{
   devices = pBLEScan->start(scanTime, false);       // start scanning
// Serial.print("scanned devices ");
// Serial.print(devices.getCount());
   digitalWrite(LEDpin, paired);                     // turn on or off LED
   if(paired == 0 && lastPair == 1) digitalWrite(LEDpin, 1);
   lastPair = paired;                 // 2 consecutive non-pairings to turn off
   paired = 0;                                       // reset paired variable
   pBLEScan->clearResults();                         // delete scan results
}
```

Таймеры

Микроконтроллер ESP32 имеет четыре независимых таймера, *timer0–timer3*, со входной частотой 80 МГц каждый, так что один тик таймера длится 0,0125 мкс. 16-разрядный предварительный делитель определяет количество тиков, составляющих один отсчет регистра таймера. Например, установка предварительного делителя, равная 80, означает, что регистр таймера увеличивается каждую микросекунду (1 мкс = 80 × 0,0125), и каждую секунду таймер досчитывает до миллиона. Прерывание подключается к таймеру для запуска события. Для подключения таймера требуются четыре инструкции, определяющие его свойства:

```
timer = timerBegin(0, 80, true);             // timer0, pre-scalar of 80
timerAttachInterrupt(timer, &timerISR, tr ue );
                                             // attach interrupt ISR
timerAlarmWrite(timer, 1000000, true);       // alarm count = 10^6
timerAlarmEnable(timer);                     // enable the timer alarm
```

где переменная `timer` задается инструкцией `hw_timer_t * timer = NULL`, а `timerISR` – подпрограмма обслуживания прерываний (ISR). Если `timer0` должен вести обратный отсчет, то первой инструкцией будет `timerBegin(0, 80, false)`. Обработчик прерывания срабатывает, когда значение в регистре таймера равно заданному числу *alarm count*, которое в примере равно 10^6. Первая и последняя инструкции обработчика прерывания следующие:

```
portENTER_CRITICAL_ISR(&timerMux)
portEXIT_CRITICAL_ISR(&timerMux)
```

где переменная `timerMux` задается инструкцией

```
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED
```

Если переменная `value`, имеющая тип *float*, доступна в прерывании и изменяется в функции *loop*, то она определяется с помощью команды `volatile float value`, а в функции `loop` выделяется инструкциями:

```
portENTER_CRITICAL(&timerMux)
value = value + 1
portEXIT_CRITICAL(&timerMux)
```

Связь по Wi-Fi и Bluetooth может повлиять на моменты возникновения прерываний. Функции связи могут быть приостановлены с помощью инструкций `WiFi.mode(WIFI_OFF)` и `btStop()` соответственно.

Чтобы проиллюстрировать использование таймеров, два светодиода включаются и выключаются с разными интервалами, причем интервалы регулируются двумя таймерами (см. рис. 22-10 и листинг 22-7).

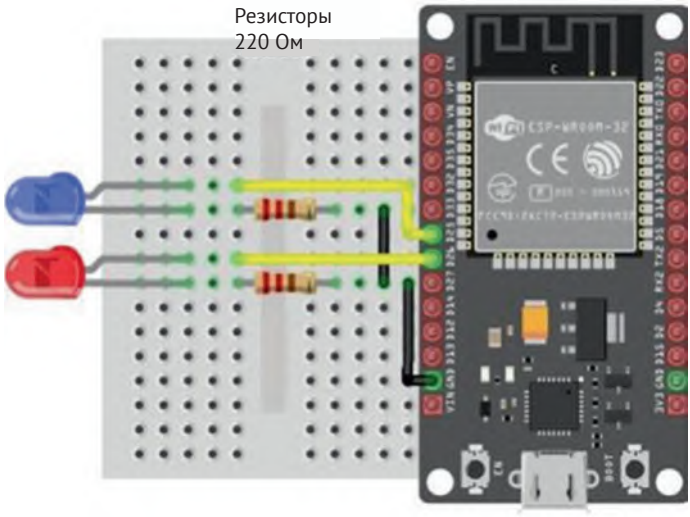


Рисунок 22-10. Таймеры ESP32

Листинг 22-7. Управление по таймеру двумя независимыми событиями

```
hw_timer_t * timer1 = NULL;                    // define timer1
portMUX_TYPE timer1Mux = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t * timer2 = NULL;                    // define timer2
portMUX_TYPE timer2Mux = portMUX_INITIALIZER_UNLOCKED;
int LED1pin = 25;
int LED2pin = 26;                              // define LED pins
void setup()
{
  Serial.begin(115200);
  pinMode(LED1pin, OUTPUT);
  pinMode(LED2pin, OUTPUT);
  timer1 = timerBegin(1, 80, true);  // set timer1 properties
  timerAttachInterrupt(timer1, &timer1ISR, true);
  timerAlarmWrite(timer1, 250000, true);     // interval of 0.25s
  timerAlarmEnable(timer1);
  timer2 = timerBegin(2, 80, true);          // set timer2 properties
  timerAttachInterrupt(timer2, &timer2ISR, true);
  timerAlarmWrite(timer2, 1000000, true);    // interval of 1s
  timerAlarmEnable(timer2);
}
```

```
void loop()
{
   vTaskDelay(NULL);
}
IRAM_ATTR void timer1ISR()                    // ISR for timer1
{
   portENTER_CRITICAL_ISR(&timer1Mux);
   digitalWrite(LED1pin, !digitalRead(LED1pin));
   portEXIT_CRITICAL_ISR(&timer1Mux);
}
IRAM_ATTR void timer2ISR()                    // ISR for timer 2
{
   portENTER_CRITICAL_ISR(&timer2Mux);
   digitalWrite(LED2pin, !digitalRead(LED2pin));
   portEXIT_CRITICAL_ISR(&timer2Mux);
}
```

RTC[139] и спящий режим

Несколько контактов GPIO доступны с помощью библиотеки доступа к выводам RTC-подсистемы *rtc_io*, в том числе для вывода микроконтроллера ESP32 из спящего режима. Библиотека *rtc-io* включена в библиотеки ESP32 в Arduino IDE. Команда `esp_sleep_enable_ext0_wakeup(pin, state)` выводит микроконтроллер ESP32 из спящего режима, когда состояние `pin` равно `state`. Например, если нажатие кнопки, подключенной к GPIO 32, должно приводить к выходу микроконтроллера ESP32 из спящего режима, параметр `pin` определяется как `GPIO_NUM_32` или как `(gpio_num_t) switchPin`, учитывая задание вывода инструкцией `int switchPin = 32` (см. рис. 22-11). Значение `state` равно нулю или единице, если переключатель имеет подтягивающий или заземляющий резистор соответственно. Подтягивающие и заземляющие резисторы микроконтроллера ESP32 отключены во время сна, поэтому команда `rtc_gpio_pullup_en(pin)` или `rtc_gpio_ pulldown_en(pin)` подключает соответствующий резистор на выводе GPIO, подключенном к кнопке. Подробная информация о режимах сна и вариантах пробуждения доступна по адресу docs.espressif.com/projects/espidf/en/latest/esp32/api-reference/system/sleep_modes.html.

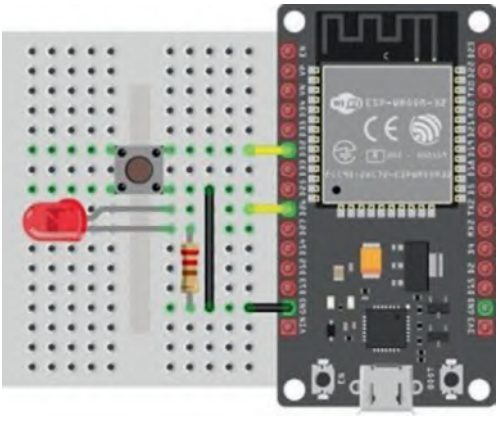


Рисунок 22-11. Выводы RTC и спящий режим

[139] См. сноску 133 на стр. 420. – *Прим перев.*

Обратите внимание, что подтягивающий или заземляющий резистор для использования с кнопкой подключается к выводу с помощью команды `pinMode(pin, INPUT_PULLUP)` или `pinMode(pin, INPUT_ PULLDOWN)` соответственно. Скетч в листинге 22-8 позволяет кнопке на выводе 32 вывести ESP32 из спящего режима, помигать внешним и встроенным светодиодом, а затем возвращает ESP32 в спящий режим.

Листинг 22-8. Выводы RTC и спящий режим

```
#include <driver/rtc_io.h>    // include rtc input-output library
int switchPin = 32;           // define switch pin
int LEDpin = 26;              // and LED pin
int builtinLED = 2;
void setup()
{
  Serial.begin(115200);       // Serial Monitor baud rate
  pinMode(LEDpin, OUTPUT);    // LED pins as output
  pinMode(builtinLED, OUTPUT);
  flash();                    // call flash function
  rtc_gpio_pullup_en((gpio_num_t)s witchPin); // pull-up switch pin
  esp_sleep_enable_ext0_wakeup((gpio_num_t)switchPin, 0);
}                             // wakeup on switch pin with state 0
void loop()
{
  Serial.print("sleep mode on pin ");Serial.println(switchPin);
  esp_deep_sleep_start();     // ESP32 in sleep mode
}
void flash()
{
  for (int i=0; i<3; i++)     // flash LEDs three times
  {
    digitalWrite(LEDpin, HIGH);      // LED on pin
    digitalWrite(builtinLED, HIGH);  // and built-in LED
    delay(200);
    digitalWrite(LEDpin, LOW);
    digitalWrite(builtinLED, LOW);
    delay(100);
  }
}
```

Таймер подсистемы RTC ESP32 выводит микроконтроллер ESP32 из спящего режима по истечении определенного периода времени с помощью команды `esp_sleep_enable_timer_wakeup(N)`, где *N* – требуемое количество микросекунд.

В спящем режиме память подсистемы RTC ESP32 функционирует, в то время как основной процессор и память микроконтроллера ESP32 отключены. Если информация должна сохраняться в спящем режиме, то данные сохраняются в памяти RTC при указании RTC_DATA_ATTR в инструкциях по определению данных. Например, если целочисленная переменная `var` должна сохраняться во время спящего режима, переменная определяется как `RTC_DATA_ATTR int var`.

Скетч в листинге 22-9 определяет счетчик, который будет сохранен в памяти RTC, печатает его измененное значение и переводит микроконтроллер в спящий режим, который будет разбужен с помощью таймера RTC через 5 с.

Листинг 22-9. Таймер RTC и спящий режим

```
RTC_DATA_ATTR int count = 0;          // store count in RTC memory
unsigned long micro = 5000000;        // time interval in μs
void setup()
{
  Serial.begin(115200);               // Serial Monitor baud rate
  esp_sleep_enable_timer_wakeup(micro) ; // RTC timer interval in μs
}
void loop()
{
  count++;                            // increment and print count
  Serial.print("count ");Serial.println(count);
  esp_deep_sleep_start();
}                                     // microcontroller in sleep mode
```

Цифроаналоговый преобразователь

8-разрядный цифроаналоговый преобразователь (ЦАП) преобразует цифровое значение в диапазоне от 0 до 255 в напряжение от 0 до 3,3 В на выводе ЦАП. У 8-разрядного ЦАП имеется 256 ступенек напряжения, от 0 до 255; для опорного напряжения 3,3 В разница между соседними ступеньками напряжения ЦАП (цена разряда) составляет 12,9 мВ. Команда `dacWrite(DACpin, N)` выводит напряжение *N*-й ступеньки на `DACpin`, при этом `DACpin` определяется либо как GPIO 25 или 26, либо термином DAC1 или DAC2 соответственно. Скетч в листинге 22-10 генерирует диапазон напряжений от 0,5 В до 3 В.

Листинг 22-10. Цифроаналоговый преобразователь

```
int DACpin = DAC1;            // define DAC pin
void setup()
{} // nothing in setup function
void loop()
{
  for (int i=0; i<255; i=i+39)
  {
    dacWrite(DACpin, i);      // output voltage 0.5V, 1V...
    delay(2000);
  }
}
```

Емкостный сенсорный датчик

Емкостные сенсорные датчики обнаруживают изменения емкости на сенсорном выводе для использования такого вывода вместо механического переключателя. При поднесении пальца к проводнику, подключенному к сенсорному контакту, значение напряжения сенсорного контакта снижается. Команда `touchRead (touchPin)` считывает значение `touchPin`, при этом `touchPin` определяется либо как номер GPIO 2, 4, 12, 13, 14, 15, 27, 32 или 33, либо как T2, T0, T5, T4, T6, T3, T7, T9 или T8 соответственно. Прерывание, подключенное к сенсорному контакту, срабатывает, когда уровень напряжения сенсорного контакта падает ниже порогового значения. Команда `touchAttachInterrupt(touchPin, ISR,`

threshold) определяет сенсорный контакт, процедуру обработки прерываний (ISR) и пороговое значение (threshold), ниже которого запускается прерывание. Чтобы предотвратить повторное срабатывание прерывания, при нажатии сенсорного вывода должен пройти интервал времени с момента касания до срабатывания прерывания. Скетч в листинге 22-11 включает или выключает светодиод при касании провода, подключенного к сенсорному контакту. Обратите внимание, что обработчик сенсорного прерывания (change) не обязательно должен быть определен как IRAM_ATTR.

Листинг 22-11. Емкостный сенсорный датчик

```
int touchPin = T7;                    // define touch pin
int LEDpin = 32;                      // and LED pin
int threshold = 50;                   // limit for touch pin
volatile unsigned long lastTouch = 0; // time touch pin pressed
void setup()
{
  pinMode(LEDpin, OUTPUT);            // LED pin as output
  touchAttachInterrupt(touchPin, change, threshold);
}                                     // define interrupt
void change()                         // ISR
{                                     // touch pin recently pressed
  if (millis() - lastTouch < 1000) return;
  lastTouch = millis();               // update touch time
  digitalWrite(LEDpin, 1 - digitalRead(LEDpi n));
}                                     // change LED state
void loop()                           // nothing in loop function
{}
```

Датчик Холла

Микроконтроллер ESP32 содержит датчик на основе эффекта Холла, который активируется магнитным полем. Команда hallRead() возвращает значение датчика Холла, при этом абсолютная величина значения указывает на напряженность магнитного поля, а положительный или отрицательный знак указывает направление магнитного поля. Датчик Холла используется, например, для измерения скорости вращения колеса или вала с помощью прикрепленного к ним магнита. Скетч в листинге 22-12 включает или выключает светодиод при обнаружении магнитного поля датчиком Холла ESP32.

Листинг 22-12. Датчик Холла

```
int LEDpin = 32;                      // define LED pin
unsigned long lastHall = 0;           // time Hall value changed
void setup()
{
  pinMode(LEDpin, OUTPUT);            // LED pin as output
}
void loop()
{
  if(abs(hallRead()) > 30) change(); // call change function
}                                     // when magnetic field detected
```

```
void change()
{
   if(millis() - lastHall < 1000) return;     // check time last change
   lastHall = millis();                       // update change time
   digitalWrite(LEDpin, 1 - digitalRead(LE Dpin));
}                                             // change LED state
```

Итоги

Микроконтроллер ESP32 имеет два ядра, каждое из которых содержит 32-разрядный микропроцессор Tensilica Xtensa LX6; распределение задач по разным ядрам эффективно удваивает производительность контроллера по сравнению с выполнением задачи на одном ядре ESP32. Микроконтроллер ESP32 поддерживает связь по Wi-Fi, SPI и I2C, аналого-цифровое преобразование, широтно-импульсную модуляцию и функции прерывания. Микроконтроллер ESP32 имеет ряд дополнительных функций, в том числе связь по Bluetooth и Bluetooth Low Energy, четыре независимых таймера для управления событиями, подсистему RTC для управления спящим режимом микроконтроллера, цифроаналоговый преобразователь, емкостные сенсорные датчики для использования сенсорного контакта вместо механического переключателя и датчик Холла, который активируется магнитным полем. Примеры иллюстрируют использование функций ESP32 в комплексе.

Перечень компонентов

- Микроконтроллер ESP32: плата DEVKIT DOIT или NodeMCU
- Светодиод 2 шт.
- Резистор 220 Ом 2 шт.
- Тактовая кнопка

Приложение

Программные библиотеки

Большинство необходимых библиотек загружаются в Arduino IDE вместе с программным обеспечением плат, другие библиотеки доступны через GitHub (www.github.com) или через конкретные веб-сайты, перечисленные в табл. А-1. Доступ к примерам скетчей в каждой библиотеке осуществляется в среде IDE Arduino выбором *Файл* ➤ *Пример* ➤ <*имя библиотеки*> (*File* ➤ *Example* ➤ <*library name*>). Библиотека включается в скетч инструкцией `#include <libraryname.h>`, которая ссылается на библиотеку, расположенную в папке библиотек Arduino IDE. Чтобы определить местоположение папки Arduino IDE *libraries*, выберите *Файл* ➤ *Настройки* (*File* ➤ *Preferences*) в Arduino IDE, и папка *libraries* будет находиться внутри указанной там папки для размещения скетчей (например, *C:\Users\<имя пользователя>\Documents\Arduino*)[140].

Когда библиотека включена в скетч, переменная обычно ассоциируется с библиотекой. Это действие называется созданием экземпляра класса, где класс задается библиотекой. Переменная обладает свойствами, заданными в библиотеке, аналогично тому, как переменная, определенная как целое число, обладает свойствами целого числа, принятыми для конкретной архитектуры. Инструкции, специфичные для библиотеки, имеют определенный префикс имени переменной. Например, библиотека *ESP8266WebServer* устанавливается инструкцией `#include <ESP8266WebServer.h>` и инструкция `ESP8266WebServer server` связывает переменную `server` с библиотекой. Тогда команда, специфичная для библиотеки *ESP8266WebServer*, должна иметь префикс `server`, как, например, в `server.handleClient()`.

Существует три способа установки библиотеки:

1. Воспользоваться менеджером библиотек *Library Manager*.
   Откройте Arduino IDE, выберите меню *Скетч* (*Sketch*) и выберите *Подключить библиотеку* ➤ *Управление библиотеками* (*Include Library* ➤ *Manage Libraries*). В окне Менеджера библиотек используйте опцию фильтрации по-

[140] Отметим, что кроме пользовательской папки *libraries*, находящейся внутри папки со скетчами, есть основная папка по адресу *Arduino/libraries*, в которой хранятся библиотеки, входящие в Arduino IDE по умолчанию, т. е. поступающие в комплекте первоначальной установки. Для среды Arduino IDE совершенно безразлично, в каком из этих двух мест хранится библиотека, на которую вы ссылаетесь; при размещении одной и той же библиотеки в двух местах преимущество получит пользовательская, то есть та, что хранится в папке скетчей. Поэтому если вы хотите внести правку в какую-либо оригинальную библиотеку, не меняя имен заголовочных файлов, целесообразно ее перед этим скопировать в пользовательскую папку: у вас окажется под рукой оба варианта, причем рабочим будет исправленный вами вариант, а для возвращения к старому достаточно просто удалить библиотеку из пользовательской папки. – *Прим. перев.*

иска, чтобы найти нужную библиотеку. Выберите номер версии библиотеки и нажмите кнопку *Установить* (*Install*). Опция *More info* предоставляет доступ к GitHub для получения документации и обновлений библиотеки.

2. Скачать zip-файл.
   Загрузите библиотеку в виде архивного zip-файла и сохраните его на своем компьютере или ноутбуке. В среде IDE Arduino выберите меню *Скетч* (*Sketch*) и выберите *Подключить библиотеку* ➤ *Добавить .ZIP-библиотеку* (*Include Library* ➤ *Add .ZIP library*). Выберите папку, в которой был сохранен zip-файл, укажите zip-файл и нажмите кнопку *Открыть* (*Open*).
3. Ручная установка.
   Загрузите библиотеку в zip-файле и извлеките содержимое zip-файла в папку библиотеки по умолчанию, например *C:\Users\<имя пользователя>\Documents\Arduino\libraries*. Arduino IDE необходимо перезапустить, после чего следует добавить библиотеку через *Скетч* ➤ *Подключить библиотеку* (*Sketch* ➤ *Include Library*), тогда библиотека будет доступна в Arduino IDE для использования в скетчах.

Таблица А-1. Библиотеки с информацией об источнике и авторе

Библиотека	Автор и источник библиотеки, если она недоступна в Arduino IDE
Adafruit BMP280	Adafruit
Adafruit GFX	Adafruit
Adafruit ILI9341	Adafruit
Adafruit ILI9341 esp	NailBuster Software nailbuster.com/?page_id=341
Adafruit INA219	Adafruit
Adafruit MCP4725	Adafruit
Adafruit NeoPixel	Adafruit
Adafruit SSD1306	Adafruit
Adafruit S7735 and ST7789	Adafruit
Adafruit Unified Sensor	Adafruit
ArduinoJson	Benoit Blanchon
ArduinoOTA	Доступна
AsyncTCP	Hristo Gochkov github.com/me-no-dev/AsyncTCP
BLEDevice	Доступна с ESP32
BLEScan	Доступна с ESP32
BLEServer	Доступна с ESP32
BLEUtils	Доступна с ESP32
BLE2902	Доступна с ESP32
BluetoothSerial	Доступна с ESP32 Espressif Systems
CayenneMQTT	Cayenne
EEPROM	David Mellis
ESP IDF FreeRTOS	Доступна с ESP32 Espressif Systems

Продолжение табл. А-1

Библиотека	Автор и источник библиотеки, если она недоступна в Arduino IDE
esp_camera	Доступна с ESP32 Espressif Systems
esp_http_server	Доступна с ESP32 Espressif Systems
ESP32 BLE Arduino	Neil Kolban, Доступна с ESP32
ESP32 vs1053 ext	Wolle github.com/schreibfaul1/ESP32-vs1053_ext
ESP32Servo	Kevin Harrington and John K. Bennett
ESPAsyncWebServer	Hristo Gochkov github.com/me-no-dev/ESPAsyncWebServer
ESPmDNS	Доступна с ESP32 Espressif Systems
ESP-NOW	Доступна с ESP8266 и ESP32 Espressif Systems
ESP8266mDNS	Доступна с ESP8266
ESP8266WebServer	Доступна с ESP8266
ESP8266WiFi	Доступна с ESP8266
FS	Доступна с ESP8266
IRremote	Ken Shirriff github.com/z3t0/Arduino-IRremote
IRremoteESP8266	David Conran, Sebastien Warin, Mark Szabo, and Ken Shirriff
Keyboard	Доступна
LittleFS	Доступна с ESP8266
LoRa	Sandeep Mistry
MD_AD9833	Marco Colli, MajicDesigns
MD_DS3231	Marco Colli, MajicDesigns
NeoGPS	Slash Devin
NewPing	Tim Eckel
NewPingESP8266	Tim Eckel and Jordan Shaw github.com/jshaw/NewPingESP8266
NTPtimeESP	Andreas Spiess github.com/SensorsIot/NTPtimeESP
printf	Included in RF24
rc-switch	Saut Özgür
RF24	James Coliz
RH_ASK	Mike McCauley www.airspayce.com/mikem/arduino/RadioHead
rtc-io	Доступна с ESP32
RunningMedian	Rob Tillaart
SD	Доступна
SD_MMC	Доступна с ESP32 Espressif Systems
Servo	Доступна, Michael Margolis

Окончание табл. А-1

Библиотека	Автор и источник библиотеки, если она недоступна в Arduino IDE
SoftwareSerial	Доступна
SPI	Доступна
SPIFFS	Доступна с ESP32 Espressif Systems
TFT_eSPI	Bodmer
Ticker	Доступна с ESP8266
Time	Michael Margolis
VS1053	Ed Smallenburg and James Coliz github.com/baldram/ESP_VS1053_Library
WebServer	Доступна с ESP32 Espressif Systems
WebSocketsServer	Markus Sattler
WiFi	Доступна с ESP32 Espressif Systems
WiFiUdp	Доступна с ESP8266 и ESP32 Espressif Systems
Wire	Доступна, Nicholas Zambetti
XPT2046	Spiros Papadimitriou github.com/spapadim/XPT2046

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «КТК Галактика» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, пр. Андропова д. 38 оф. 10.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.galaktika-dmk.com**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Нил Кэмерон

Электронные проекты на основе ESP8266 и ESP32

Создание приложений и устройств с поддержкой Wi-Fi

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Ревич Ю. В.*
Корректор *Синяева Г. И.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 36,89. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com

Откройте для себя
мощные микроконтроллеры ESP8266 и ESP32 с Wi-Fi-связью!

Микроконтроллеры ESP8266 и ESP32 необычайно популярны во всем мире как основа для построения интернета вещей и систем умного дома. Они сочетают простоту применения и дешевизну с достаточно высокими возможностями, характерными для 32-разрядных платформ. Популярность их в значительной мере обусловлена наличием легкодоступного и бесплатного ПО, совместимого с уже ставшей стандартом в любительских кругах средой разработки Arduino IDE.

В книге делается акцент на практических проектах - начиная от создания мобильных приложений для удаленного управления устройствами с распознаванием речи до GPS-трекинга с использованием Google Maps.

Вы сможете:

- создавать электронные проекты с возможностью связи по Wi-Fi;
- использовать микроконтроллеры для обновления веб-страниц;
- общаться с мобильным телефоном или смарт-часами по интерфейсу с низким энергопотреблением Bluetooth Low Energy;
- передавать и получать информацию для управления удаленными устройствами через интернет;
- освоить дизайн и сборку мобильных приложений;
- приобрести навыки компьютерного программирования на C++, JavaScript, AJAX и JSON;
- научиться использовать WebSocket, MQTT-брокеры и IFTTT для быстрой двусторонней связи с веб-страницами.

Книга адресована всем любителям DIY, умеющим работать с Arduino и заинтересованным в создании настоящих IoT-устройств и интеграции их в систему умного дома.

Предполагается, что читатель имеет некоторый опыт программирования на C++ в Arduino IDE.

ISBN 978-5-93700-141-2

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК "Галактика"
books@alians-kniga.ru

ДМК
издательство
www.дмк.рф