

ESP32-C3

Беспроводное приключение

Полное руководство по IoT

RISC-V

Wi-Fi

Bluetooth

ESP-IDF

ESP RainMaker



ESP32-C3: Беспроводное подключение

Полное руководство по IoT

ESP32-C3 Wireless Adventure

A Comprehensive Guide to IoT



ESP32-C3: Беспроводное подключение

Полное руководство по IoT



Москва, 2024

УДК 004.75

ББК 32.81

Э41

Э41 ESP32-C3: Беспроводное приключение: Полное руководство по IoT / пер. с англ. Ю. В. Ревича. – М.: ДМК Пресс, 2023. – 442 с.: ил.

ISBN 978-5-93700-248-8

Эта книга демонстрирует различные приложения семейства ESP32-C3 – одноядерного микроконтроллера, представляющего собой «систему на кристалле» (SoC) с интегрированными Wi-Fi и Bluetooth 5 (LE). Он обеспечивает необходимый баланс мощности, возможностей ввода-вывода и безопасности, предлагая таким образом оптимальное экономичное решение для подключаемых устройств. Вы освоите основы разработки проектов интернета вещей (IoT) и настройки среды и выполните практические примеры вплоть до запуска изделия в серийное производство.

Издание будет полезно инженерам, разработчикам программного обеспечения, преподавателям, студентам и просто любителям оборудования для IoT.

УДК 004.75

ББК 32.81

Copyright title of the English-language edition: 'ESP32-C3 Wireless Adventure: A Comprehensive Guide to IoT' published by Expressif Systems.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Оглавление

https://t.me/it_books/2

Предисловие от издательства	13
Введение	14
Предисловие.....	15
ЧАСТЬ I. ПОДГОТОВКА	21
Глава 1. Введение в IoT.....	23
1.1. Архитектура интернета вещей	23
Уровень восприятия и управления	23
Сетевой уровень	24
Уровень платформы	25
Прикладной уровень	26
1.2. Применение IoT в проекте «Умного дома»	26
Глава 2. Введение в практику IoT-проектов	29
2.1. Введение в типовые проекты IoT	29
2.1.1. Базовые модули для обычных устройств IoT	29
2.1.2. Базовые модули клиентских приложений	30
2.1.3. Введение в общие облачные платформы IoT	31
2.2. Практика: проект Smart Light.....	32
2.2.1. Структура проекта	33
2.2.2. Функции проекта	33
2.2.3. Подготовка оборудования	34
2.2.4. Процесс разработки	36
2.3. Резюме	37
Глава 3. Введение в ESP RainMaker.....	38
3.1. Что такое ESP RainMaker?	39
3.2. Реализация ESP RainMaker	40
3.2.1. Служба обработки заявок	41
3.2.2. RainMaker Agent	42
3.2.3. Облачный сервер	43
3.2.4. Клиент RainMaker.....	44
3.3. Практика: ключевые моменты разработки с ESP RainMaker	44
3.4. Особенности ESP RainMaker	46

3.4.1. Управление пользователями.....	46
3.4.2. Функции конечного пользователя.....	47
3.4.3. Функции администратора.....	48
3.5. Резюме	48

Глава 4. Настройка среды разработки..... 50

4.1. Обзор ESP-IDF	50
4.1.1. Версии ESP-IDF	51
4.1.2. Рабочий процесс ESP-DIFF Git	52
4.1.3. Выбор подходящей версии.....	53
4.1.4. Обзор каталога ESP-IDF SDK	53
4.2. Настройка среды разработки ESP-IDF	58
4.2.1. Настройка среды разработки ESP-IDF в Linux	58
4.2.2. Настройка среды разработки ESP-IDF в Windows.....	60
4.2.3. Настройка среды разработки ESP-IDF на Mac	66
4.2.4. Установка VS Code	67
4.2.5. Знакомство со сторонними средами разработки	67
4.3. Система компиляции ESP-IDF	68
4.3.1. Основные концепции системы компиляции.....	68
4.3.2. Структура файла проекта	69
4.3.3. Правила построения системы компиляции по умолчанию	71
4.3.4. Введение в сценарий компиляции	72
4.3.5. Введение в общие команды	73
4.4. Практика: компиляция примера программы Blink.....	74
4.4.1. Анализ примера	74
4.4.2. Компиляция программы Blink.....	77
4.4.3. Прошивка программы Blink.....	81
4.4.4. Анализ логов последовательного порта программы Blink	82
4.5. Резюме	85

ЧАСТЬ II. РАЗРАБОТКА ОБОРУДОВАНИЯ И ДРАЙВЕРОВ 87

Глава 5. Аппаратный дизайн продуктов Smart Light на базе ESP32-C3..... 89

5.1. Характеристики и состав продуктов Smart Light.....	89
5.2. Аппаратный дизайн базовой системы ESP32-C3	93
5.2.1. Источник питания	97
5.2.2. Порядок включения питания и сброс системы	97
5.2.3. SPI флеш-память.....	98
5.2.4. Источник тактовых импульсов	98
5.2.5. Радиочастотный сигнал (RF) и антенна	99
5.2.6. Выводы управления загрузкой ПО (Strapping Pins).....	102
5.2.7. GPIO и ШИМ-контроллер.....	102

5.3. Практика: создание системы умного освещения с помощью ESP32-C3.....	103
5.3.1. Выбор модулей.....	103
5.3.2. Настройка ШИМ-сигналов на выводах GPIO.....	105
5.3.3. Загрузка встроенного ПО и интерфейс отладки.....	105
5.3.4. Рекомендации по проектированию радиочастотой части	108
5.3.5. Рекомендации по проектированию источника питания.....	109
5.4. Резюме	109

Глава 6. Разработка драйверов 111

6.1. Процесс разработки драйверов	111
6.2. Периферийные приложения ESP32-C3	112
6.3. Основы построения драйверов светодиодов.....	113
6.3.1. Цветовые пространства.....	114
6.3.2. Светодиодный драйвер	118
6.3.3. Диммирование светодиодов.....	119
6.3.4. Введение в ШИМ	120
6.4. Разработка драйвера для регулирования светодиодов.....	122
6.4.1. Энергонезависимая память (NVS).....	122
6.4.2. Светодиодный ШИМ-контроллер (LEDC).....	123
6.4.3. Программирование ШИМ для светодиодов.....	125
6.5. Практика: добавление драйверов в проект Smart Light.....	128
6.5.1 Драйвер кнопки	128
6.5.2. Драйвер регулировки яркости светодиода	130
6.6. Резюме	134

ЧАСТЬ III. БЕСПРОВОДНАЯ СВЯЗЬ И УПРАВЛЕНИЕ 135

Глава 7. Настройка Wi-Fi-соединения 137

7.1. Основы Wi-Fi	137
7.1.1. Введение в Wi-Fi.....	137
7.1.2. Эволюция IEEE 802.11	138
7.1.3. Концепции Wi-Fi	139
7.1.4. Wi-Fi-соединение	141
7.2. Основы Bluetooth	149
7.2.1. Введение в Bluetooth.....	149
7.2.2. Концепции Bluetooth	150
7.2.3. Bluetooth-соединение	154
7.3. Конфигурация сети Wi-Fi	158
7.3.1. Руководство по настройке сети Wi-Fi.....	158
7.3.2. Программная точка доступа (Soft access point, SoftAP)	158
7.3.3. SmartConfig	159
7.3.4. Bluetooth	162
7.3.5. Другие методы.....	164

7.4. Программирование Wi-Fi	166
7.4.1. Компоненты Wi-Fi в ESP-IDF	166
7.4.2. Упражнение: соединение Wi-Fi	168
7.4.3. Упражнение: интеллектуальное подключение к Wi-Fi	172
7.5. Практика: конфигурация Wi-Fi в проекте Smart Light	184
7.5.1 Соединение Wi-Fi в проекте Smart Light	184
7.5.2. Умная настройка Wi-Fi	185
7.6. Резюме	186

Глава 8. Локальное управление..... 187

8.1. Введение в локальное управление	187
8.1.1. Применение локального управления	189
8.1.2. Преимущества локального управления	189
8.1.3. Обнаружение управляемых устройств с помощью смартфонов	189
8.1.4. Передача данных между смартфонами и устройствами	190
8.2. Общие методы локального обнаружения	190
8.2.1. Широковещательная передача	191
8.2.2. Групповая передача (Multicast)	197
8.2.3. Сравнение широковещательной и групповой рассылки	202
8.2.4. Протокол групповых приложений mDNS для локального обнаружения	203
8.3. Общие протоколы связи для локальных данных	206
8.3.1. Протокол управления передачей (TCP)	206
8.3.2. Протокол передачи гипертекста (HyperText Transfer Protocol, HTTP)	211
8.3.3. Протокол пользовательских датаграмм (User Datagram Protocol, UDP)	214
8.3.4. Протокол ограниченных приложений (Constrained Application Protocol, CoAP)	218
8.3.5. Протокол Bluetooth	222
8.3.6. Обзор протоколов передачи данных	228
8.4. Гарантии безопасности данных	230
8.4.1. Введение в безопасность транспортного уровня (TLS)	232
8.4.2. Введение в датаграмм-протокол безопасности транспортного уровня (DTLS)	238
8.5. Практика: локальное управление в проекте Smart Light	241
8.5.1. Создание локального управляющего сервера на базе Wi-Fi	241
8.5.2. Проверка функциональности локального управления с помощью скриптов	245
8.5.3. Создание локального сервера управления на базе Bluetooth	246
8.6. Резюме	248

Глава 9. Управление через облако 250

9.1. Введение в удаленное управление	250
9.2. Облачные протоколы передачи данных	251
9.2.1. Введение в MQTT	251

9.2.2. Принципы MQTT.....	252
9.2.3. Формат сообщения MQTT	254
9.2.4. Сравнение протоколов	258
9.2.5. Настройка MQTT Broker в Linux и Windows	259
9.2.6. Настройка клиента MQTT на основе ESP-IDF	260
9.3. Обеспечение безопасности данных MQTT.....	262
9.3.1. Значение и функция сертификатов.....	262
9.3.2. Локальная генерация сертификатов	263
9.3.3. Настройка MQTT Broker.....	266
9.3.4. Настройка клиента MQTT.....	266
9.4. Практика: дистанционное управление через ESP RainMaker	268
9.4.1. Основы ESP RainMaker.....	268
9.4.2. Протокол связи между узлом и серверной частью облака.....	269
9.4.3. Взаимодействие между клиентом и облачным бэкендом	274
9.4.4. Типы пользователей	277
9.4.5. Основные сервисы	278
9.4.6. Пример Smart Light	280
9.4.7. Приложение RainMaker и интеграция сторонних платформ	286
9.5. Резюме	293

Глава 10. Разработка приложений для смартфонов..... 295

10.1. Введение в разработку приложений для смартфонов	295
10.1.1. Обзор разработки приложений для смартфонов	296
10.1.2. Структура проекта Android.....	296
10.1.3. Структура проекта iOS.....	297
10.1.4. Жизненный цикл Android Activity	298
10.1.5. Жизненный цикл iOS ViewController.....	299
10.2. Создание нового проекта приложения для смартфона	301
10.2.1. Подготовка к разработке под Android	301
10.2.2. Создание нового проекта Android	301
10.2.3. Добавление зависимостей для MyRainmaker	302
10.2.4. Запрос разрешений в Android	303
10.2.5. Подготовка к разработке iOS.....	304
10.2.6. Создание нового проекта iOS.....	304
10.2.7. Добавление зависимостей для MyRainmaker	305
10.2.8. Запрос разрешений в iOS	307
10.3. Анализ функциональных требований приложения	307
10.3.1. Анализ функциональных требований проекта.....	307
10.3.2. Анализ требований к управлению пользователями	308
10.3.3. Анализ требований к подготовке и привязке устройства....	309
10.3.4. Анализ требований к удаленному управлению.....	310
10.3.5. Анализ требований к планированию	311
10.3.6. Анализ требований к пользовательскому центру	312
10.4. Разработка системы управления пользователями.....	312
10.4.1. Введение в API RainMaker.....	312
10.4.2. Инициализация связи через смартфон.....	313

10.4.3. Регистрация учетной записи.....	313
10.4.4. Вход в учетную запись.....	316
10.5. Разработка системы подготовки устройств	319
10.5.1. Сканирование устройств	320
10.5.2. Подключение устройств	321
10.5.3. Генерация секретных ключей	324
10.5.4. Получение идентификатора (ИД) узла	324
10.5.5. Подготовка устройств	326
10.6. Разработка управления устройствами	328
10.6.1. Привязка устройств к облачным учетным записям.....	328
10.6.2. Получение списка устройств	330
10.6.3. Получение статуса устройства	333
10.6.4. Изменение статуса устройства.....	336
10.7. Разработка расписания и пользовательского центра.....	338
10.7.1. Реализация функции планирования.....	338
10.7.2. Реализация центра пользователей	340
10.7.3. Дополнительные облачные API.....	343
10.8. Резюме	344

Глава 11. Обновление встроенного ПО и управление версиями..... 345

11.1. Обновление прошивки	345
11.1.1. Обзор таблицы разделов	346
11.1.2. Процесс загрузки прошивки	348
11.1.3. Обзор механизма OTA	350
11.2. Управление версиями прошивки.....	353
11.2.1. Маркировка прошивки	353
11.2.2. Откат и защита от отката	354
11.3. Практика: пример OTA-обновления.....	355
11.3.1. Обновление прошивки через локальный хост.....	355
11.3.2. Обновление прошивки через ESP RainMaker.....	358
11.4. Резюме	364

ЧАСТЬ IV. ОПТИМИЗАЦИЯ И СЕРИЙНОЕ ПРОИЗВОДСТВО365

Глава 12. Управление питанием и оптимизация энергопотребления..... 367

12.1. Управление питанием ESP32-C3.....	367
12.1.1. Динамическое масштабирование частоты	368
12.1.2. Настройка управления питанием.....	370
12.2. Режимы пониженного энергопотребления ESP32-C3	370
12.2.1. Режим Modem-sleep	371
12.2.2. Режим Light-sleep	373
12.2.3. Режим глубокого сна Deep-sleep	378
12.2.4. Потребление тока в различных режимах питания.....	380

12.3. Управление питанием и отладка режима низкого энергопотребления	381
12.3.1. Отладка через логи	381
12.3.2. Отладка по состояниям GPIO	383
12.4. Практика: управление питанием в проекте Smart Light.....	385
12.4.1. Настройка функции управления питанием.....	386
12.4.2. Использование блокировки управления питанием	387
12.4.3. Проверка энергопотребления	388
12.5. Резюме	388

Глава 13. Расширенные функции безопасности устройства.... 389

13.1. Обзор безопасности данных IoT-устройств	389
13.1.1. Зачем защищать данные устройств интернета вещей?	390
13.1.2. Основные требования к безопасности данных IoT-устройств	391
13.2. Защита целостности данных.....	392
13.2.1. Основы метода проверки целостности	392
13.2.2. Проверка целостности данных прошивки	393
13.2.3. Пример.....	394
13.3. Защита конфиденциальности данных	394
13.3.1. Введение в шифрование данных	394
13.3.2. Введение в систему флеш-шифрования.....	396
13.3.3. Хранение ключей флеш-шифрования.....	399
13.3.4. Рабочие режимы флеш-шифрования	400
13.3.5. Процесс флеш-шифрования.....	402
13.3.6. Введение в шифрование NVS.....	403
13.3.7. Примеры флеш-шифрования и шифрования NVS	404
13.4. Защита легитимности данных	406
13.4.1. Введение в цифровую подпись	406
13.4.2. Обзор системы безопасной загрузки	408
13.4.3. Введение в программную безопасную загрузку	408
13.4.4. Введение в аппаратную безопасную загрузку	410
13.4.5. Примеры	415
13.5. Практика: функции безопасности в серийном производстве	416
13.5.1. Флеш-шифрование и безопасная загрузка	416
13.5.2. Включение флеш-шифрования и безопасной загрузки с помощью инструментов пакетной прошивки.....	417
13.5.3. Включение флеш-шифрования и безопасной загрузки в проекте Smart Light	418
13.6. Резюме	418

Глава 14. Запись и тестирование прошивки для серийного производства 420

14.1. Загрузка прошивки при серийном производстве	420
14.1.1. Определение разделов данных.....	421
14.1.2. Запись прошивки	423

14.2. Тестирование серийной продукции	424
14.3. Практика: производственные данные в проекте Smart Light.....	426
14.4. Резюме	426

Глава 15. ESP Insights: платформа удаленного мониторинга427

15.1. Введение в ESP Insights	427
15.2. Начало работы с ESP Insights	431
15.2.1. Начало работы с ESP Insights в проекте esp-insights.....	431
15.2.2. Пример выполнения в проекте esp-insights	433
15.2.3. Отчетность об информации дампа памяти	433
15.2.4. Настройка интересующих логов	434
15.2.5. Сообщение о причине перезагрузки	435
15.2.6. Отчетность по заданным показателям	435
15.3. Практика: использование ESP Insights в проекте Smart Light	438
15.4. Резюме	440

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.

Введение

ESP32-C3 – это одноядерный микроконтроллер, представляющий собой «систему на кристалле» (SoC) с интегрированными Wi-Fi и Bluetooth 5 (LE), основанный на архитектуре RISC-V с открытым исходным кодом. Он обеспечивает необходимый баланс мощности, возможностей ввода-вывода и безопасности, предлагая таким образом оптимальное экономичное решение для подключаемых устройств. Эта книга от компании Espressif продемонстрирует различные приложения семейства ESP32-C3, проведя вас по увлекательному пути, начинающемуся с основ разработки проектов интернета вещей (IoT) и настройки среды и заканчивающемуся практическими примерами. Первые четыре главы рассказывают об IoT, ESP RainMaker и ESP-IDF. В главах 5 и 6 кратко говорится о проектировании оборудования и разработке драйверов. По мере продвижения вы узнаете, как настроить свой проект через сети Wi-Fi и мобильные приложения. Наконец, вы научитесь оптимизировать свой проект и запускать его в серийное производство.

Если вы инженер в смежных областях, разработчик программного обеспечения, преподаватель, студент или просто любитель, интересующийся IoT, – эта книга для вас.

Примеры кода, использованные в данной книге, вы можете скачать с сайта Espressif на GitHub. Для получения последней информации о разработке интернета вещей, пожалуйста, следите за нашим официальным аккаунтом.

Предисловие

Информатизирующийся мир

Со времени появления Всемирной сети интернет вещей (IoT) совершил грандиозный скачок, став новым типом инфраструктуры в цифровой экономике. Чтобы приблизить технологию к широкой публике, компания Espressif Systems работает над тем, чтобы разработчики из всех слоев общества могли использовать IoT для решения некоторых насущных проблем современности. Мир «интеллектуальной сети вещей» – это то, чего мы ожидаем от будущего.

Разработка все новых чипов является важнейшим компонентом этого видения. Это марафон, требующий постоянных прорывов через технологические границы. От «изменившего правила игры» ESP8266 до серии ESP32, интегрирующей Wi-Fi и Bluetooth (LE), а затем ESP32-S3, оснащенной ускорителем искусственного интеллекта, компания Espressif никогда не прекращает исследования и разработку продуктов для решений IoT. С помощью нашего программного обеспечения с открытым исходным кодом, такого как фреймворк ESP-IDF для разработки интернета вещей, фреймворк разработки распределенных сетей ESP-MDF и платформа ESP RainMaker для подключения устройств, мы создали независимую платформу для создания приложений AIoT¹.

По состоянию на июль 2022 года совокупные поставки IoT-чипсетов Espressif превысили 800 млн, что обеспечивает огромное количество подключенных устройств во всем мире и вывело компанию в лидеры рынка Wi-Fi-микроконтроллеров. Стремление к совершенству делает каждый продукт Espressif популярным благодаря высокому уровню интеграции и экономической эффективности. Выпуск ESP32-C3 знаменует собой важную веху в развитии технологий собственных разработок Espressif. Это одноядерный 32-рядный микроконтроллер на базе RISC-V с 400 Кбайт SRAM, который может работать на частоте 160 МГц. В ESP32-C3 встроены Wi-Fi с частотой 2,4 ГГц и Bluetooth 5 (LE) с поддержкой большой дальности связи. Он обеспечивает прекрасный баланс мощности, возможностей ввода-вывода и безопасности, предлагая таким образом оптимальное экономичное решение для подключенных устройств. На примере столь мощного ESP32-C3 эта книга с подробными иллюстрациями и практическими примерами призвана помочь читателям разобраться в знаниях, связанных с IoT.

¹ AIoT – в отличие от всем известного сокращения IoT, означающего «интернет вещей» (Internet of Things), сокращение AIoT означает «искусственный интеллект вещей» (Artificial intelligence of things) – появившееся в последние годы направление, сочетающее технологии интернета вещей с технологиями искусственного интеллекта (ИИ). – *Здесь и далее прим. перев.*

Почему мы написали эту книгу?

Espressif Systems – это нечто большее, чем полупроводниковая компания. Это также компания, занимающаяся платформой интернета вещей, которая всегда стремится к прорывам и инновациям в области технологий. В то же время Espressif открыла исходный код и поделилась с сообществом своей самостоятельно разработанной операционной системой и программным обеспечением, сформировав уникальную экосистему. Инженеры, производители и энтузиасты технологий активно разрабатывают новые программные приложения на основе продуктов Espressif, свободно общаются и делятся своим опытом. Вы можете увидеть увлекательные идеи разработчиков, постоянно появляющиеся на различных платформах, таких как YouTube и GitHub. Популярность продуктов Espressif стимулировала рост числа авторов, которые выпустили более 100 книг на основе наборов микросхем Espressif, более чем на десяти языках, включая английский, китайский, немецкий, французский и японский.

Именно поддержка и доверие партнеров сообщества поощряют непрерывные инновации Espressif. «Мы стремимся сделать наши чипы, операционные системы, фреймворки, решения, облако, бизнес-практики, инструменты, документацию, тексты, идеи и т. д. еще более актуальными для ответов, необходимых людям для решения самых насущных проблем современной жизни. Это высочайшие амбиции и моральный ориентир Espressif», – сказал г-н Тео Суи Энн, основатель и генеральный директор компании Espressif.

Espressif ценит изложение идей. Поскольку непрерывное совершенствование технологий интернета вещей предъявляет все более высокие требования к инженерам, как мы можем помочь большему числу людей быстро освоить IoT-чипы, операционные системы, программные платформы, типовые схемы приложений и продукты облачных сервисов? Как говорится, лучше научить человека ловить рыбу, чем давать ему эту самую рыбу. Во время мозгового штурма нам пришло в голову, что мы могли бы написать книгу, в которой систематизировались бы ключевые знания об IoT-разработке. Мы договорились, быстро собрали группу старших инженеров, обобщивших опыт технической команды в области встроенного программирования, разработки аппаратного и программного обеспечения интернета вещей, что внесло свой вклад в публикацию этой книги. В процессе написания мы изо всех сил старались быть объективными и справедливыми, избавляться от шор и использовать максимально краткие выражения, чтобы рассказать о сложности и очаровании интернета вещей. Мы тщательно обобщили общие проблемы, отзывы и предложения сообщества, чтобы четко ответить на вопросы, возникающие в процессе разработки, и предоставить практические рекомендации по разработке интернета вещей для ответствующих технических специалистов и лиц, принимающих решения.

Структура книги

Эта книга ориентирована на инженеров и излагает необходимые знания для разработки проекта IoT шаг за шагом. Он состоит из четырех частей, а именно:

- **«Подготовка»** (главы 1–4): эта часть знакомит с архитектурой IoT, типичными структурами проектов, облачной платформой ESP RainMaker

и средой разработки ESP-IDF, закладывая прочную основу для разработки проекта IoT;

- **«Разработка оборудования и драйверов»** (главы 5–6): на основе чипсета ESP32-C3 в этой части подробно рассматриваются минимальная аппаратная система и разработка драйверов, а также реализуются диммирование, цветокоррекция и управление беспроводной связью;
- **«Беспроводная связь и управление»** (главы 7–11): в этой части объясняется интеллектуальная схема конфигурации Wi-Fi, основанная на чипе ESP32-C3, протоколах локального и облачного управления, а также локальном и удаленном управлении устройствами. Здесь также предоставляются способы разработки приложений для смартфонов, обновления встроенного ПО и управления версиями;
- **«Оптимизация и серийное производство»** (главы 12–15): эта часть предназначена для продвинутых приложений IoT, ориентированных на оптимизацию продуктов в области управления питанием, низкого энергопотребления и повышенной безопасности. Представлены примеры записи прошивки и тестирования в серийном производстве, а также диагностики рабочего состояния и ведения логов устройств через платформу удаленного мониторинга ESP Insights.

Об исходном коде

Читатели могут запускать примеры программ из этой книги либо путем ввода кода вручную, либо с использованием исходного кода, прилагаемого к книге. Мы подчеркиваем сочетание теории и практики и почти в каждую главу включаем раздел «Практика» на основе проекта Smart Light («Умного освещения»). Все коды открыты. Читатели могут скачать исходный код и обсудить его в разделах, связанных с этой книгой, на GitHub и нашем официальном форуме *esp32.com*. Открытый исходный код этой книги подчиняется условиям Apache License 2.0.

Авторское примечание

Эта книга официально выпущена компанией Espressif Systems и написана старшими инженерами компании. Она подходит для менеджеров и специалистов по исследованиям и разработкам в отраслях, связанных с IoT, преподавателей и студентов смежных специальностей, а также энтузиастам в области интернета вещей. Мы надеемся, что эта книга может служить и рабочим пособием, и справочником, и прикроватной книжкой, как хороший наставник и друг.

При составлении книги мы ссылались на некоторые актуальные результаты исследований экспертов, ученых и технических специалистов в стране² и за рубежом и сделали все возможное, чтобы привести их в соответствие с академическими нормами. Однако неизбежны некоторые упущения, поэтому здесь мы хотели бы выразить наше глубокое уважение и благодарность всем соответствующим авторам. Кроме того, мы цитируем информацию из интернета, поэтому хотели бы поблагодарить авторов и издателей оригиналов и прино-

² Так как Espressif Systems является китайской компанией со штаб-квартирой в Шанхае, то, очевидно, речь идет о Китае.

сим свои извинения за то, что не можем указать источник каждого фрагмента информации.

Для выпуска качественной книги мы организовали регулярные внутренние обсуждения, изучили предложения и отзывы читателей предварительных версий и редакторов издательства. Здесь мы хотели бы еще раз поблагодарить вас за вашу помощь, которая способствовала этой успешной работе.

Последнее, но самое главное: спасибо всем в Espressif, кто усердно работал для создания и популяризации нашей продукции.

Разработка IoT-проектов требует широкого круга знаний. В связи с ограниченным объемом книги, а также уровнем знаний и опыта авторов упущения неизбежны. Поэтому просим экспертов и читателей критиковать и исправлять наши ошибки. Если у вас есть какие-либо предложения по этой книге, пожалуйста, свяжитесь с нами по адресу book@espressif.com. Мы с нетерпением ждем ваших отзывов.

Как пользоваться этой книгой

Код изложенных в этой книге проектов находится в открытом доступе. Вы можете скачать его с нашего репозитория GitHub, поделиться своими мыслями и задать вопросы на нашем официальном форуме.

GitHub: <https://github.com/espressif/book-esp32c3-iot-projects>

Форум: <https://www.esp32.com/bookc3>

В книге выделяются следующие части текста:

Исходный код

В этой книге мы подчеркиваем сочетание теории и практики, и, таким образом, раздел о проекте Smart Light найдется почти в каждой главе. Соответствующие шаги и ссылка на исходный код будут размещаться между двумя линиями и помечаться заголовком **Исходный код**.

Примечание/Совет

Здесь вы можете найти важную информацию и напоминания об успешной отладке вашей программы. Она будет размещена между двумя толстыми линиями и помечена заголовком **Примечание** или **Совет**.

Большинство команд в этой книге выполняются под Linux и помечаются символом системного приглашения «\$». Если для выполнения команды требуются привилегии суперпользователя, приглашение будет заменено на "#". Командная строка в системах Mac помечается «%», как указано в разделе 4.2.3 «Установка ESP-IDF на Mac».

Основной текст этой книги будет напечатан обычным (пропорциональным) шрифтом, а примеры кода, компоненты, функции, переменные будут обозначены моноширинным шрифтом. Имена файлов, каталоги, пути и гиперссылки обозначаются курсивом.

Команды или тексты, которые должны быть введены пользователем, и команды, которые могут быть введены нажатием клавиши **Enter**, будут напечатаны **жирным моноширинным шрифтом**. Логи и блоки кода будут помечены голубым фоном. Гиперссылки обозначены синим курсивом: *[гиперссылка](#)*.

Пример:

Во-вторых, используйте `esp-idf/components/nvs_flash/nvs_partition_generator/nvs_partition_gen.py` для создания раздела бинарных файлов NVS на хосте разработки с помощью следующей команды:

```
$ python $IDF_PATH/components/nvs_flash/nvs_partition_generator/ nvs_partition_gen.py  
--input mass prod.csv --output mass prod.bin --size NVS_PARTITION_SIZE.
```


Часть I

Подготовка

https://t.me/it_boooks/2

Здесь начинается наше путешествие к полноценному проекту интернета вещей. В этой части мы сначала познакомимся с архитектурой и базовыми знаниями IoT, затем с сервисами ESP RainMaker на основе проекта Smart Light. Наконец, установим платформу разработки интернета вещей Espressif (фреймворк ESP-IDF) для дальнейшего изучения. К концу данной части вы будете знакомы с концепциями, описанными в этой книге далее, и сможете разрабатывать и компилировать примеры проектов.

ГЛАВА 1

Введение в IoT

ГЛАВА 2

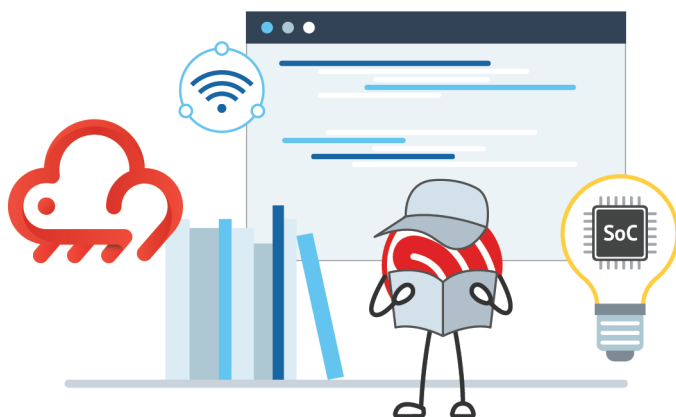
**Введение в практику
IoT-проектов**

ГЛАВА 3

Введение в ESP RainMaker

ГЛАВА 4

Настройка среды разработки



В конце XX века, с появлением компьютерных сетей и коммуникационных технологий, интернет быстро интегрировался в жизнь людей. В процессе развития интернет-технологий родилась идея интернета вещей (IoT). Буквально IoT означает связь вещей между собой с помощью интернета. В то время как обычный интернет ломает границы пространства и времени и сужает дистанцию между «человеком и человеком», IoT делает «вещи» важным участником, сближает «людей» и «вещи». В обозримом будущем интернет вещей станет движущей силой информационной индустрии.

Итак, что такое интернет вещей?

Точное определение интернету вещей дать трудно, поскольку его значение и масштабы постоянно развиваются. В 1995 году Билл Гейтс впервые поднял идею IoT в своей книге «*The Road Ahead*» («Дорога в будущее»). В его изложении IoT позволяет объектам обмениваться информацией друг с другом через интернет. Конечной целью является создание «интернета всего». Это ранняя интерпретация IoT, а также фантазия о технологиях будущего. Тридцать лет спустя, с бурным развитием экономики и технологии, фантазия становится реальностью. От «умных устройств», «умных домов», «умных городов», интернета транспортных средств и носимых устройств – к виртуальной «метавселенной», поддерживаемой технологиями IoT. Постоянно появляются новые концепции. В этой главе мы начнем с объяснения архитектуры интернета вещей, а затем познакомимся с одним из наиболее общих приложений IoT – «умным домом», чтобы помочь вам получить четкое представление.

1.1. Архитектура интернета вещей

Интернет вещей включает в себя несколько технологий, которые в различных отраслях имеют разные прикладные свойства и формы. Для того чтобы разобраться в структуре, ключевых технологиях и прикладных характеристиках IoT, необходимо установить единую архитектуру и стандартную техническую систему. В этой книге архитектура IoT просто разделена на четыре уровня: уровень восприятия и управления, сетевой уровень, уровень платформы и уровень приложений.

Уровень восприятия и управления

Как самый основной элемент архитектуры IoT, уровень восприятия и управления является ядром, реализующим всестороннее представление IoT. Его основная функция состоит в том, чтобы собирать, идентифицировать и управлять информацией. Он состоит из множества устройств, обладающих способностью

сбора информации, идентификации, управления и исполнения, а также отвечает за получение и анализ данных, таких как свойства материалов, поведенческие тенденции и состояние устройства. Таким образом, IoT получает способность познавать реальный физический мир. Кроме того, этот слой также может контролировать статус устройства.

Наиболее распространенными устройствами этого слоя являются различные датчики, играющие важную роль в сборе и идентификации информации. Датчики похожи на органы чувств человека: например, фоточувствительные датчики эквивалентны зрению, акустические датчики – слуху, газовые датчики – обонянию, а датчики, чувствительные к давлению и температуре, – осязанию. Со всеми этими «органами чувств» предметы становятся «живыми» и способными к осмысленному восприятию, узнаванию и манипулированию физическим миром.

Сетевой уровень

Основная функция сетевого уровня заключается в передаче информации, включая данные, полученные с уровня восприятия и управления, заданному целевому объекту, а также команд, выданных с прикладного уровня, обратно на уровень восприятия и управления. Сетевой уровень служит основным коммуникационным мостом, соединяющим различные уровни системы интернета вещей. Для создания базовой модели интернета вещей необходимо выполнить две операции по интеграции объектов в сеть: доступ к интернету и передачу данных через интернет.

Доступ к интернету

Интернет обеспечивает взаимосвязь между людьми, но пасует при включении вещей в сообщество. До появления интернета вещей большинство вещей не были «подключены к сети». Только непрерывное развитие технологий позволило подключать объекты к интернету, тем самым реализуя взаимосвязь между «людьми и вещами» и «вещами и вещами». Существует два распространенных способа подключения к интернету: доступ к проводной сети и доступ к беспроводной сети.

К проводным методам доступа относятся Ethernet, последовательная связь (например, RS-232, RS-485) и USB. Доступ через беспроводную сеть зависит от беспроводной связи, которую можно разделить на беспроводную связь ближнего действия и дальнюю беспроводную связь.

Беспроводная связь ближнего действия включает ZigBee, Bluetooth®, Wi-Fi, NFC³ и радиочастотную идентификацию (RFID). Беспроводная связь большого радиуса действия включает стандарты мобильной связи (2G, 3G, 4G, 5G), а так-

³ NFC (Near field communication, «связь в близком поле») – стандарт связи устройств на расстояниях 10 см и менее, представляющий собой расширение стандарта бесконтактных карт. В отличие от Bluetooth, может осуществлять связь только «точка–точка»; в отличие от радиочастотной идентификации (RFID), связь полноценно двухсторонняя. Всем известный пример применения NFC в быту – эмуляция бесконтактных банковских карт с помощью мобильного устройства при оплате покупок или проезда в общественном транспорте.

же такие технологии, как LoRa, eMTC, узкополосный интернет вещей (NB-IoT)⁴ и т. д.

Передача данных через интернет

Различные методы доступа в интернет определяют соответствующий физический канал передачи данных. Следующее, что нужно сделать, – это решить, какой протокол связи использовать для передачи. По сравнению с интернет-терминалами, большинство IoT-терминалов в настоящее время имеют меньше доступных ресурсов, таких как производительность обработки, емкость памяти, скорость передачи и т. д., поэтому в IoT-приложениях необходимо выбрать протокол связи, который занимает меньше ресурсов. В настоящее время широко используются два протокола связи: MQTT (Message Queuing Telemetry Transport, передача телеметрии с очередью сообщений) и CoAP (Constrained Application Protocol, протокол ограниченного приложения).

Уровень платформы

Уровень платформы в основном относится к облачным платформам IoT. Когда все терминалы IoT объединены в сеть, их данные должны быть агрегированы на облачной платформе IoT для расчетов и хранения. Уровень платформы в основном поддерживает приложения IoT для облегчения доступа и управления крупными устройствами. На уровне платформы IoT-терминалы подключаются к облачной платформе, собираются данные терминалов и выдаются команды дистанционного управления. Как промежуточный сервис для назначения оборудования отраслевым приложениям, уровень платформы играет связующую роль во всей архитектуре IoT, неся абстрактную бизнес-логику и стандартизированную базовую модель данных, которая может не только реализовать быстрый доступ к устройствам, но также обеспечивает мощные модульные возможности для удовлетворения различных потребностей в промышленных прикладных сценариях. Уровень платформы в основном включает в себя функциональные модули, такие как доступ к устройствам, управление устройствами, управление безопасностью, обмен сообщениями, мониторинг операций и эксплуатации, техническое обслуживание, а также приложения для работы с данными:

- доступ к устройствам, реализующий соединение и связь между терминалами и облачными платформами IoT;
- управление устройствами, включая такие функции, как подключение и обслуживание устройств, преобразование и синхронизация данных и распределение устройств;
- управление безопасностью, обеспечение передачи данных IoT с точки зрения безопасной аутентификации и связи;
- обмен сообщениями, включая три направления передачи, т. е. терминал отправляет данные на облачную платформу IoT, облачная платформа IoT

⁴ LoRa (Long Range) – технология маломощной сети передачи данных со скоростью 0,3–50 кбит/с и дальностью от 1 до 15 км в нелицензируемых диапазонах частот (ISM). eMTC (Enhanced Machine Type Communication) и NB-IoT (Narrow Band Internet of Things) – специально разработанные для IoT разновидности сотовой связи, поверх LTE (eMTC) или LTE/GSM (NB-IoT).

отправляет данные на сторонний сервер или другие облачные платформы IoT, а серверная сторона удаленно управляет устройствами IoT;

- мониторинг O&M⁵, включая мониторинг и диагностику, обновление прошивки, онлайн-отладку, сервисы ведения лог-файлов и т. д.;
- приложения данных, включая хранение, анализ и применение данных.

Прикладной уровень

Прикладной уровень использует данные уровня платформы для управления приложением, производит их фильтрацию и обработку с помощью таких инструментов, как базы данных и аналитическое программное обеспечение. Полученные данные можно использовать для реальных приложений IoT, таких как интеллектуальное здравоохранение, интеллектуальное сельское хозяйство, умные дома и умные города.

Конечно, архитектуру IoT можно разделить на большее количество уровней, но сколько бы ни было слоев, из которых она состоит, основной принцип остается по существу одним и тем же. Изучение архитектуры IoT помогает углубить наше понимание технологий и построить полнофункциональные IoT-проекты.

1.2. Применение IoT в проекте «Умного дома»

IoT проник во все сферы жизни. Наиболее близкое к нам приложение IoT – «Умный дом». Многие традиционные бытовые приборы теперь оснащены одним или несколькими IoT-устройствами, и многие недавно построенные дома с самого начала проектируются с использованием технологий IoT.

На рис. 1.1 показаны некоторые распространенные устройства умного дома.

Развитие умного дома можно просто разделить на стадию подбора «умного» продукта, этап взаимодействия сценариев и интеллектуальный этап, как показано на рис. 1.2.

Первый этап касается **умных продуктов**. В отличие от традиционных домов, в умных домах устройства IoT получают сигналы с датчиков и объединяются в сеть посредством беспроводной связи с помощью таких технологий, как Wi-Fi, Bluetooth LE или ZigBee. Пользователи могут управлять интеллектуальными продуктами различными способами – через приложения на смартфонах, с помощью голосовых помощников, интеллектуального управления колонками и т. д.

Второй этап фокусируется на **взаимодействии сценариев**. На этом этапе разработчики больше не рассматривают возможность управления одним умным продуктом, а соединяют два или более таких продукта, автоматизируя взаимодействие до определенной степени и, наконец, формируя общий пользовательский режим всего окружения. Например, когда пользователь нажимает любую кнопку установки режима, свет, шторы и кондиционеры выключаются, автоматически возвращаясь к предустановкам. Конечно, есть условие, что ло-

⁵ О функции Operation and Maintenance (O&M) см. стр. 32. Не путать с международным стандартом O&M (Observations and Measurements) на форматы наблюдений и измерений, относящимся к географическим информационным системам.

гика взаимодействия должна легко настраиваться, включая условия срабатывания и действия для выполнения команды. Представьте, что режим обогрева кондиционера срабатывает при снижении температуры в помещении ниже 10 °С; что в 7 часов утра играет музыка, чтобы разбудить пользователя, умные шторы открываются и рисоварка или тостер запускается через смарт-розетку. Пользователь встает и заканчивает умываться, а завтрак уже подан, так что можно не опаздывать на работу. Как удобна стала наша жизнь!

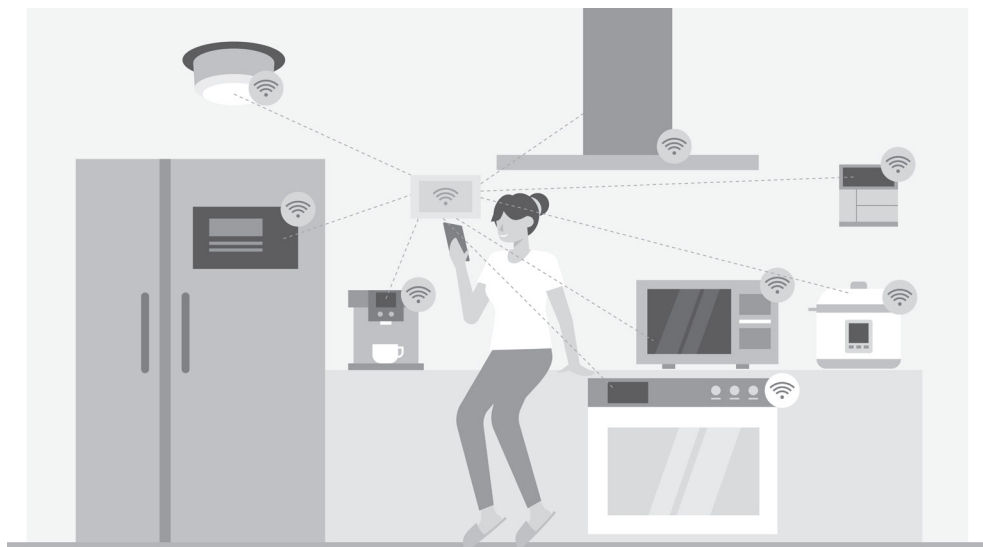


Рисунок 1.1. Обычные устройства умного дома

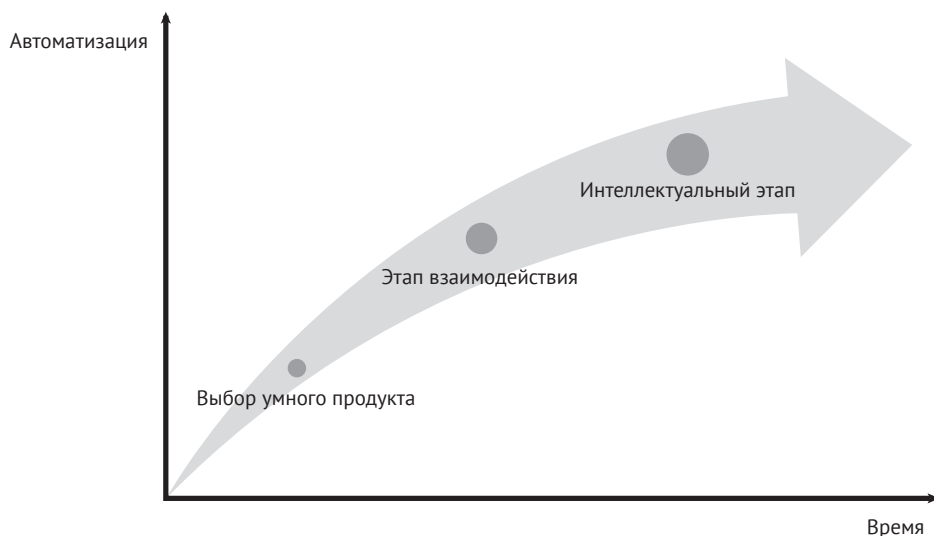


Рисунок 1.2. Этапы разработки умного дома

Третий этап – **интеллектуальный**. По мере доступа к большему количеству умных домашних устройств растет количество типов генерируемых данных. При участии облачных вычислений, технологий больших данных и искусственного интеллекта это похоже на то, как в умные дома подсадили «умный мозг», который уже не требует частых команд от пользователя. Он собирает данные о предыдущих действиях, изучает модели поведения и предпочтения пользователя, чтобы автоматизировать действия, в том числе предоставляя рекомендации для принятия решения.

В настоящее время большинство умных домов находятся на стадии межсетевого взаимодействия. По мере развития увеличиваются скорость и интеллект умных продуктов, устраняются барьеры между коммуникационными протоколами. В будущем умные дома обязательно станут действительно «умными», как ИИ Джарвис в «Железном человеке»⁶, который может не только помочь пользователю управлять различными устройствами и справляться с повседневными делами, но также обладает супервычислительной мощностью и выдающимися мыслительными способностями. На интеллектуальной стадии люди будут получать более качественные услуги как по количеству, так и по качеству.

⁶ Джарвис (Jarvis) – верный дворецкий на основе искусственного интеллекта, вымышленный персонаж американских комиксов, издаваемых Marvel Comics. Джарвис чаще всего изображается в виде супергероя, как, например, в фильме «Железный человек» (2008).

Глава 2

Введение в практику IoT-проектов

https://t.me/it_books/2

В главе 1 мы представили архитектуру интернета вещей, роли и взаимосвязи уровней восприятия и управления, сетевого уровня, уровня платформы и приложений, обозначили развитие проекта умного дома. Однако так же, как при обучении рисованию, теоретических знаний далеко недостаточно, чтобы по-настоящему овладеть технологией, мы должны «запачкать руки», чтобы внедрить IoT-проекты на практике. Кроме того, когда проект переходит на стадию серийного производства, необходимо учитывать большое количество факторов, таких как подключение сети, ее настройка, взаимодействие с облачной платформой IoT, управление прошивкой и обновлениями, управление массовым производством и настройки безопасности.

Итак, на что нам нужно обратить внимание при разработке полного проекта IoT?

В главе 1 мы упомянули, что умный дом является одним из наиболее распространенных приложений интернета вещей, а умное освещение – одно из самых простых и практичных устройств, которое можно использовать в домах, отелях, спортзалах, больницах и т. д. Поэтому в этой книге мы возьмем создание проекта умного освещения в качестве отправной точки, объясним его компоненты и функции и дадим рекомендации по развитию проекта. Мы надеемся, что из этого случая вы сможете сделать выводы для создания большего количества IoT-приложений.

2.1. Введение в типовые проекты IoT

С точки зрения развития основные функциональные модули IoT-проектов можно разделить на разработку программного и аппаратного обеспечения IoT-устройств, разработку клиентских приложений и разработку облачной платформы IoT. Важно уточнить содержание этих основных модулей проекта, которые будут дополнительно описаны в данном разделе.

2.1.1. Базовые модули для обычных устройств IoT

Программно-аппаратная разработка IoT-устройств включает в себя следующие основные модули:

Сбор данных

В качестве нижнего уровня архитектуры IoT устройства для наблюдения и управления соединяют датчики и устройства через их чипы и периферийные устройства для сбора данных и контроля работы.

Привязка аккаунта и первоначальная настройка

Для большинства IoT-устройств привязка учетной записи и первоначальная настройка объединяются в один рабочий процесс, например подключение устройств и пользователей путем настройки сети Wi-Fi.

Управление устройством

При подключении к облачным платформам IoT устройства могут взаимодействовать с облаком – регистрироваться, связываться или управляться. Пользователи могут запрашивать статус продукта и выполнять другие операции в приложении для смартфона через облачные платформы IoT или протоколы локальной связи.

Обновление прошивки

Устройства IoT также могут обновлять прошивку в зависимости от потребностей производителей. Обновление прошивки и управление версиями осуществляются через команды, отправленные из облака. С помощью этой функции обновления прошивки вы можете постоянно совершенствовать функции устройств IoT, исправлять дефекты и улучшать взаимодействие с пользователем.

2.1.2. Базовые модули клиентских приложений

Клиентские приложения (например, приложения для смартфонов) в основном включают следующие базовые модули:

Система аккаунта и авторизации

Эта система поддерживает авторизацию учетной записи и регистрацию устройства.

Управление устройством

Приложения для смартфонов обычно снабжены управляющими функциями. Пользователи могут легко подключаться к IoT-устройствам и управлять ими в любое время и в любом месте с помощью приложений для смартфонов. В реальном умном доме устройства в основном управляются через приложения для смартфонов, которые не только обеспечивают интеллектуальное управление устройствами, но и экономят затраты на рабочую силу. Таким образом, управление устройством является обязательным для клиентских приложений, таких как управление функциями устройства, управление сценариями, планирование, дистанционное управление, привязка устройств и т. д. Пользователи умного дома также могут настраивать сценарии в соответствии с личными потребностями, контролируя освещение, бытовую технику, входную дверь и т. д., чтобы сделать домашнюю жизнь более комфортной и удобной. Они могут задать время кондиционирования, выключить кондиционер удаленно, включить свет в коридоре автоматически, как только дверь разблокирована, или переключиться в режим «театр» одной-единственной кнопкой.

Уведомления

Клиентские приложения контролируют состояние устройств IoT в режиме реального времени и отправляют оповещения, когда устройства ведут себя ненормально.

Послепродажное обслуживание клиентов

Приложения для смартфонов могут предоставлять послепродажное обслуживание продуктов для своевременного решения проблем, связанных с отказами IoT-устройств и техническими операциями.

Рекомендуемые функции

Для удовлетворения потребностей разных пользователей могут быть добавлены другие функции, такие как управление жестами, NFC, GPS и т. д. GPS может помочь установить точность операций в зависимости от местоположения и расстояния, а функция управления жестами позволяет пользователям устанавливать команды, которые будут выполняться для определенного устройства или сценария с помощью перемещения или встряхивания смартфона.

2.1.3. Введение в общие облачные платформы IoT

Облачная платформа IoT – это универсальная платформа, объединяющая такие функции, как управление устройствами, связь для обеспечения безопасности данных и управление уведомлениями. Согласно их целевому назначению и доступности, облачные платформы IoT можно разделить на общедоступные (далее именуемые «**публичное облако**») и частные (далее называются «**частным облаком**»).

Публичное облако обычно означает общие облачные платформы IoT для предприятий или частных лиц, которые эксплуатируются и обслуживаются поставщиками платформ и управляются через интернет. Могут быть бесплатными или по низкой цене, а также предоставлять услуги в открытой общедоступной сети, такой как Alibaba Cloud, Tencent Cloud, Baidu Cloud, AWS IoT, Google IoT и т. д. В качестве вспомогательной платформы общедоступное облако может интегрировать вышестоящих поставщиков услуг и конечных пользователей для создания новой экосистемы.

Частное облако создано только для корпоративного использования, что гарантирует лучший контроль над данными, лучшую безопасность и качество обслуживания. Его услуги и инфраструктура обслуживаются отдельными предприятиями, а вспомогательное аппаратное и программное обеспечение также предназначено для конкретных пользователей. Предприятия могут настраивать облачные сервисы в соответствии с потребностями своего бизнеса. В настоящий момент некоторые производители умных домов уже получили частные облачные платформы IoT и разработали приложения для умного дома на их основе.

Публичное облако и частное облако имеют свои преимущества, которые будут объяснены позже. Для достижения коммуникационной связности необходимо завершить как минимум встроенную разработку на стороне устройства наряду с бизнес-серверами, облачными платформами IoT и приложениями для смартфонов. Столкнувшись с таким огромным проектом, общедоступное облако обычно предоставляет комплекты для разработки ПО приложений на стороне устройства и смартфона, чтобы ускорить процесс. Как общедоступное, так и частное облако предоставляет услуги, включая доступ к устройству, управление устройством, теневые копии устройства, а также эксплуатацию и техническое обслуживание.

Доступ к устройству

Облачные платформы IoT должны предоставлять не только интерфейсы для доступа к устройствам с использованием таких протоколов, как MQTT, CoAP, HTTPS и WebSocket, но также и функции безопасности – аутентификацию для блокировки поддельных и незаконных устройств, эффективно снижающую риск быть скомпрометированным. Такая аутентификация обычно поддерживает различные механизмы, поэтому, когда устройства выпускаются серийно, необходимо предварительное присвоение сертификата устройства согласно выбранному механизму аутентификации и запись его на устройство.

Управление устройствами

Функция управления устройствами, предоставляемая облачными платформами IoT, может не только помочь производителям следить за статусом активации и онлайн-статусом своих устройств в режиме реального времени, но также позволяет такие действия, как добавление/удаление устройств, извлечение, добавление/удаление группы, обновление прошивки и управление версиями.

Теневые копии устройства

Облачные платформы IoT могут создавать постоянную виртуальную версию (теневую копию) для каждого устройства, и статус теневой копии может быть синхронизирован и получен приложением или другими устройствами через интернет-протоколы. Теневая копия устройства хранит последний отчетный статус и ожидаемый статус каждого устройства, и даже если устройство находится в автономном режиме, приложение по-прежнему может получать его статус, вызвав API. Теневая копия обеспечивает постоянно активные API, что упрощает создание приложений для смартфонов, взаимодействующих с устройствами.

Эксплуатация и обслуживание (O&M)

Функция O&M включает в себя три аспекта:

- демонстрация статистической информации об устройствах и уведомлениях интернета вещей;
- управление лог-файлами позволяет извлекать информацию о поведении устройства, потоке сообщений о включении/отключении и их содержании;
- отладка устройства поддерживает доставку команд, обновление конфигурации и проверку взаимодействия облачных платформ IoT с сообщениями от устройств.

2.2. Практика: проект Smart Light

После теоретического введения в каждой главе вы найдете практический раздел, связанный с проектом Smart Light, который поможет вам получить практический опыт. Проект основан на чипе Espressif ESP32-C3 и облачной платформе ESP RainMaker IoT Cloud и включает аппаратный беспроводной модуль в продуктах для умного освещения, встроенное программное обеспечение для интеллектуальных устройств на основе ESP32-C3, приложение для смартфонов и взаимодействие с ESP RainMaker.

Исходный код

Для лучшего обучения и развития опыта проект, описанный в этой книге, находится в открытом доступе. Вы можете загрузить исходный код из нашего репозитория GitHub по адресу <https://github.com/espressif/book-esp32c3-iot-projects>.

2.2.1. Структура проекта

Проект Smart Light состоит из трех частей:

- I. **Устройства умного света на базе ESP32-C3**, отвечающего за взаимодействие с облаком IoT-платформы, а также осуществляющего управление выключением, яркостью и цветовой температурой светодиодов ламповой гирлянды.
- II. **Приложения для смартфонов** (включая приложения для планшетов, работающие на Android и iOS), ответственные за сетевую конфигурацию продуктов Smart Light, а также служащие для запроса и управления их статусом.
- III. **Облачной платформы IoT на основе ESP RainMaker**. Для упрощения в этой книге считаем облачную платформу IoT и бизнес-сервер единым целым. Подробности об ESP RainMaker будут представлены в главе 3.

Соответствие структуры проекта Smart Light общей архитектуре IoT показано на рис. 2.1.



Рисунок 2.1. Структура проекта Smart Light

2.2.2. Функции проекта

В соответствии со структурой функции каждого уровня следующие.

Умные LED-светильники:

- настройка сети и подключение;
- ШИМ-управление светодиодами, например выключение, изменение яркости, цветовой температуры и т. д.;
- автоматизация или управление сценариями, например таймер;
- шифрование и безопасная загрузка с флеш-памяти;
- обновление прошивки и управление версиями.

Мобильные приложения:

- конфигурация сети и привязка устройств;
- интеллектуальное управление освещением, например выключение, изменение яркости, цветовой температуры и т. д.;
- настройки автоматизации или сценария, например таймер;
- местное/дистанционное управление;
- регистрация пользователя, вход в систему и т. д.

Облачная платформа интернета вещей ESP RainMaker IoT cloud:

- включение доступа к IoT-устройствам;
- предоставление API-интерфейсов управления устройством, доступных для мобильных приложений;
- обновление прошивки и управление версиями.

2.2.3. Подготовка оборудования

При реализации проекта вам потребуется следующее оборудование: умные светильники, смартфоны, Wi-Fi-роутеры и компьютер, соответствующий требованиям установки среды разработки.

Умные светильники

Умные светильники – это новый тип ламп, форма которых такая же, как у обычных ламп накаливания. Умный светильник состоит из понижающего регулируемого источника питания, беспроводного модуля (со встроенным ESP32-C3), светодиодного контроллера и светодиодной RGB-матрицы. При подключении к питанию выходное напряжение 15 В постоянного тока после понижения, диодного выпрямления и регулирования обеспечивает питанием светодиодный контроллер и светодиодную матрицу. Светодиодный контроллер может автоматически отправлять высокие и низкие уровни через определенные промежутки времени, переключая светодиодную RGB-матрицу между включенным (горит свет) и выключенным (свет выключен) состояниями, чтобы она могла излучать голубой, желтый, зеленый, фиолетовый, синий, красный и белый свет. Беспроводной модуль отвечает за подключение к Wi-Fi-роутеру, получающему и сообщаящему сведения о состоянии умных светильников и отправляющему команды для управления светодиодом.

На ранней стадии разработки вы можете смоделировать умный свет, используя плату ESP32-C3 DevKitM-1, подключенную к светодиодной RGB-лампе (см. рис. 2.2). Но вы должны учитывать, что это не единственный способ собрать интеллектуальное осветительное устройство. Аппаратная часть проекта в этой книге содержит только беспроводной модуль (со встроенным ESP32-C3), но не полный аппаратный набор для реального проекта.

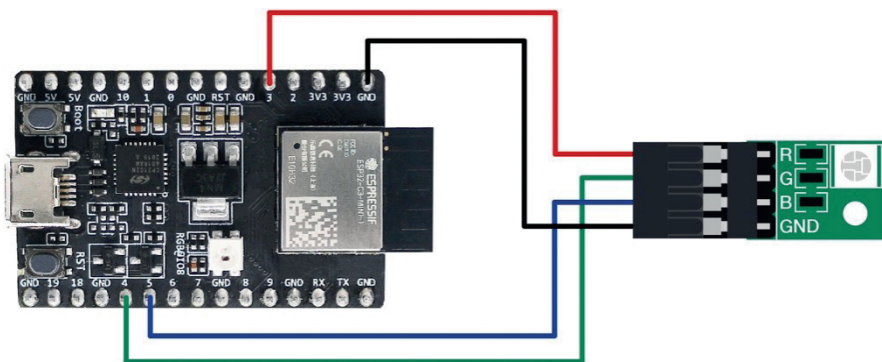


Рисунок 2.2. Имитатор умного света

Кроме того, Espressif также производит плату для разработки аудио на базе ESP32-C3 (ESP32-C3-Lyra) для управления светом со звуковым сопровождением. Плата имеет интерфейсы для микрофонов и динамиков и может управлять светодиодными лентами. Ее можно использовать для разработки сверхдешевых высокопроизводительных звуковых вещателей с ритмическим световым сопровождением. На рис. 2.3 показана плата ESP32-C3-Lyra, соединенная со световой полосой из 40 светодиодов.

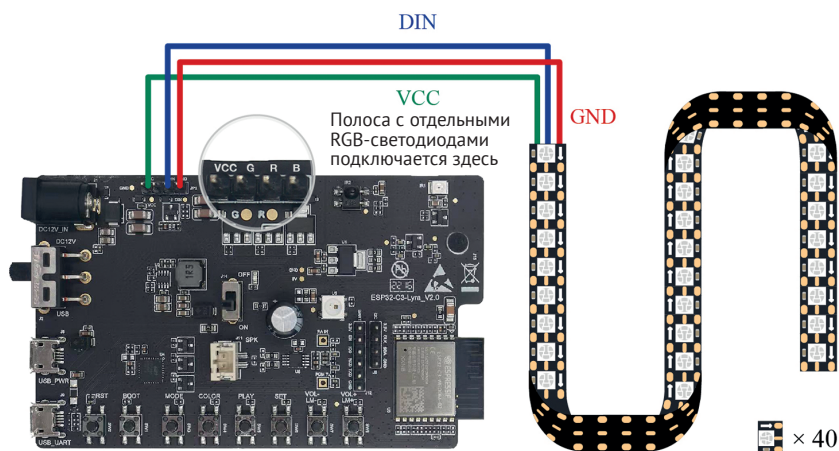


Рисунок 2.3. ESP32-C3-Lyra с подключенной полосой из 40 светодиодов

Смартфоны (Android/iOS)

Проект Smart Light предполагает разработку мобильного приложения для настройки и управления интеллектуальным осветительным устройством.

Wi-Fi-роутеры

Wi-Fi-роутеры преобразуют сигналы проводной сети и сигналы мобильной сети в сигналы беспроводной сети для компьютеров, смартфонов, планшетов и других беспроводных устройств, объединенных в сеть. Например, для широкополосного доступа в доме требуется только подключение к Wi-

Fi-роутеру для обеспечения беспроводной сети устройств Wi-Fi. Основной стандартный протокол, поддерживаемый маршрутизаторами Wi-Fi, – IEEE 802.11n со средней скоростью передачи 300 Мбит/с или максимум 600 Мбит/с. Он обратно совместим с IEEE 802.11b и IEEE 802.11g. Чип ESP32-C3 от Espressif поддерживает IEEE 802.11b/g/n, поэтому вы можете выбирать однодиапазонный (2,4 ГГц) или двухдиапазонный (2,4 ГГц и 5 ГГц) Wi-Fi-роутер.

Компьютер (Linux/macOS/Windows)

Среда разработки будет представлена в главе 4.

2.2.4. Процесс разработки

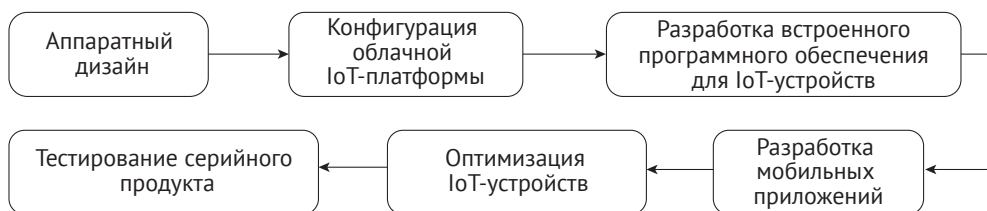


Рисунок 2.4. Этапы разработки проекта Smart Light

Аппаратный дизайн

Аппаратный дизайн устройств IoT имеет важное значение для проекта IoT. Полный проект Smart Light предназначен для изготовления лампы, управляемой по сети. Разные производители производят лампы разных стилей и типов драйверов, но их беспроводные модули обычно несут одни и те же функции. Для упрощения процесса разработки проекта Smart Light эта книга охватывает только проектирование оборудования и разработку программного обеспечения для беспроводных модулей.

Конфигурация облачной IoT-платформы

Чтобы использовать облачные платформы IoT, вам необходимо настроить проекты на серверной части, например задать продукты, создание устройств, настройку свойств устройств и т. д.

Разработка встроенного программного обеспечения для устройств IoT

Реализуйте необходимые функции с помощью ESP-IDF, Espressif SDK на стороне устройства, включая подключение к облачным платформам IoT, разработку драйверов светодиодов и обновление прошивки.

Разработка мобильного приложения

Разрабатывайте мобильные приложения для систем Android и iOS, чтобы реализовать регистрацию пользователей, вход в систему, управление устройством и другие функции.

Оптимизация IoT-устройств

После того как базовая разработка функций устройства IoT завершена, вы можете перейти к оптимизации задач, таких как оптимизация энергопотребления.

Тестирование серийного продукта

Проведение испытаний серийного продукта в соответствии с требованиями на функционирование оборудования, испытание старения, RF-тест⁷ и т. д. Несмотря на этапы, перечисленные выше, проект Smart Light не обязательно подлежит такой процедуре полностью, поскольку различные задачи могут выполняться одновременно. Например, встроенное программное обеспечение и мобильные приложения для смартфонов могут разрабатываться параллельно. Некоторые шаги также может потребоваться повторить, например оптимизацию устройств IoT и тестирование серийного производства.

2.3. Резюме

В этой главе мы изложили основные компоненты и функциональные модули проекта IoT, затем представили кейс практического проекта Smart Light, ориентируясь на его структуру, функции, подготовку оборудования и процесс разработки. Читатели могут сделать выводы для практики и уверенно выполнять проекты IoT с минимальными ошибками в будущем.

⁷ RF-тест – измерение излучаемых сигналов радиочастотного спектра и их мощности на соответствие стандартам, гарантирующим отсутствие помех иным радиоустройствам, а также для исключения вредоносного воздействия на окружающую среду (см. главу 14).

Глава 3

Введение в ESP RainMaker

Интернет вещей (IoT) предлагает бесконечные возможности изменить образ жизни людей, но инженерная разработка IoT полна проблем. При наличии общедоступных облачных хранилищ производители оборудования могут реализовать функциональность продукта с помощью следующих решений.

На основе облачных платформ поставщиков решений

В этом случае производителям оборудования нужно только разработать аппаратное обеспечение продукта, а затем вы его подключаете к облаку, используя предоставленный коммуникационный модуль, и после настройки продукт работает в соответствии с рекомендациями. Это эффективный подход, поскольку он устраняет необходимость в разработке на стороне сервера и на стороне приложения и исключает этап эксплуатации и обслуживания (O&M). Это также позволяет производителям оборудования сосредоточиться на разработке аппаратного обеспечения без необходимости рассмотрения облачной реализации. Однако готовые решения (например, прошивка устройства и мобильное приложение), как правило, не имеют открытого исходного кода, поэтому функции продукта будут ограничены облачной платформой провайдера, которую нельзя настроить. При этом и пользователь, и функциональность устройства также оказываются завязаны на данную облачную платформу.

На базе облачных продуктов

В этом случае после завершения проектирования оборудования производителям не только необходимо реализовать облачные функции с использованием одного или нескольких продуктов, предоставляемых общедоступным облаком, но также нужно связать оборудование с этим облаком. Например, для подключения к Amazon Web Services (AWS) производители терминалов должны использовать такие продукты AWS, как Amazon API Gateway, AWS IoT Core и AWS Lambda, чтобы обеспечить доступ к устройствам, удаленное управление ими, хранение данных, управление пользователями и другие основные функции. Подобное решение не только требует от производителей уметь гибко использовать и настраивать облачные продукты с глубоким пониманием и наличием опыта, но и требует от них затрат на техническое обслуживание на начальном и последующих этапах. Это создает большие проблемы для компании с точки зрения затраченной энергии и ресурсов.

По сравнению с общедоступными облаками частные облака обычно создаются для конкретных проектов и продуктов. Разработчикам частного облака предоставляется максимальный уровень свободы в разработке протоколов и ло-

гичная реализация в бизнесе. Производители оборудования могут изготавливать продукцию и проектные схемы по желанию, а также легко интегрировать пользовательские данные и расширять их возможности. Объединив высокий уровень безопасности, масштабируемости и надежности общедоступного облака с преимуществами частного облака, компания Espressif выпустила ESP RainMaker – глубоко интегрированное частное облачное решение на базе облака Amazon. Пользователи могут развернуть ESP RainMaker и создать свое частное облако, просто используя учетную запись AWS.

3.1. Что такое ESP RainMaker?

ESP RainMaker – это полноценная платформа AIoT, созданная с использованием нескольких продуктов AWS. Платформа предоставляет различные услуги, необходимые для серийного производства, такие как доступ к облаку устройств, обновление, управление серверной частью, сторонний вход в систему, голосовая интеграция и управление пользователями. Используя репозиторий бессерверных приложений (SAR), предоставляемый AWS, производители терминалов могут быстро развернуть ESP RainMaker в своих AWS-аккаунтах, что экономит время и упрощает эксплуатацию. Управляемый и поддерживаемый компанией Espressif, SAR в ESP RainMaker помогает разработчикам сократить расходы на обслуживание облака и ускорить разработку AIoT-продуктов, таким образом способствуя созданию безопасных, стабильных и настраиваемых решений AIoT. На рис. 3.1 показана архитектура ESP RainMaker.

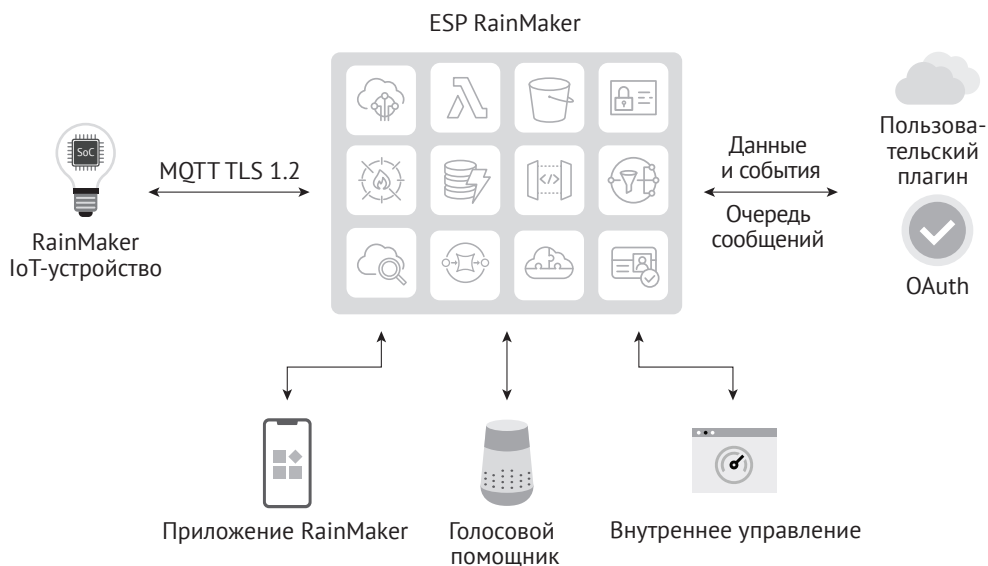


Рисунок 3.1. Архитектура ESP RainMaker

Общедоступный сервер ESP RainMaker от Espressif для оценки решений доступен бесплатно для всех энтузиастов ESP, разработчиков и преподавателей. Разработчики могут войти в систему с помощью учетных записей Apple, Google

или GitHub и быстро создавать свои собственные прототипы IoT-приложений. Общедоступный сервер интегрирует сервисы Alexa и Google Home, а также предоставляет услуги голосового управления, которые поддерживаются функциями Alexa Skill и Google Actions. Функции семантического распознавания ESP RainMaker также поддерживаются третьими лицами. Устройства интернета вещей RainMaker реагируют только на определенные действия. Для получения исчерпывающего перечня поддерживаемых голосовых команд, пожалуйста, обратитесь к перечисленным сторонним платформам. Кроме того, Espressif предлагает общедоступное приложение RainMaker, позволяющее пользователям управлять продуктами с помощью смартфонов.

3.2. Реализация ESP RainMaker

Как показано на рис. 3.2, ESP RainMaker состоит из четырех частей:

- **служба обработки заявок**, позволяющая устройствам RainMaker динамически получать сертификаты;
- **RainMaker Cloud** (также известный как облачный сервер), предоставляющий такие услуги, как фильтрация сообщений, управление пользователями, хранение данных и интеграция со сторонними серверами;
- **RainMaker Agent**, позволяющий устройствам RainMaker подключаться к RainMaker Cloud;
- **клиент RainMaker** (приложение RainMaker или CLI⁸-скрипты) для подготовки, создания пользователей, объединения устройств и управления ими и т. д.

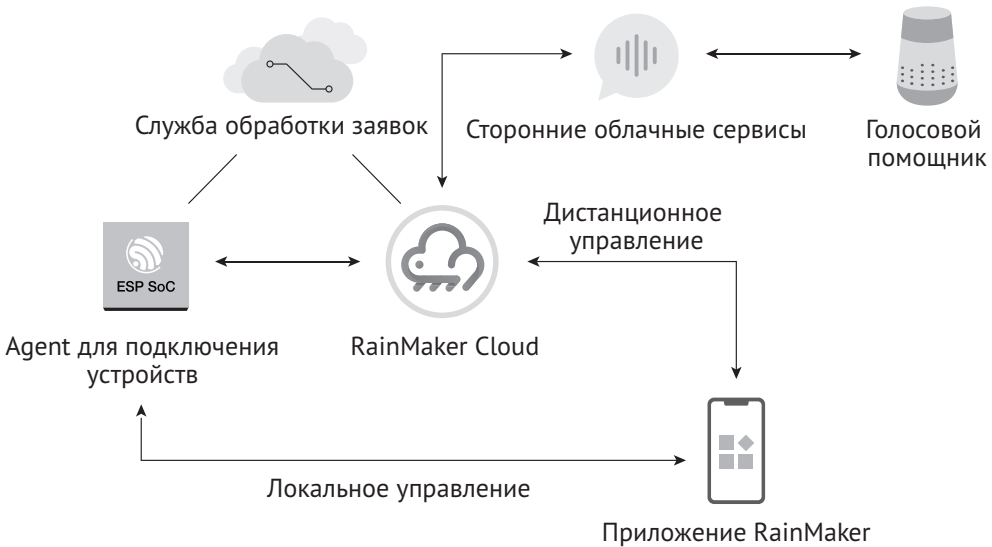


Рисунок 3.2. Структура ESP RainMaker

ESP RainMaker предоставляет полный набор инструментов для разработки продукта и серийного производства, включая:

⁸ Command-line interface, интерфейс командной строки.

Пакет SDK для RainMaker

RainMaker SDK основан на ESP-IDF и предоставляет исходный код агента на стороне устройства и связанные с ним C API для разработки встроенного ПО. Разработчикам нужно только написать логику приложения, а остальное оставить фреймворку RainMaker. Для получения дополнительной информации о C API, пожалуйста, посетите <https://book3.espressif.com/rm/c-api-reference>.

Приложение RainMaker

Общедоступная версия приложения RainMaker позволяет разработчикам выполнять подготовку устройств, а также контролировать и запрашивать статус устройств (например, умных осветительных приборов). Он доступен как в магазинах приложений для iOS, так и для Android. Для получения более подробной информации, пожалуйста, обратитесь к главе 10.

REST API

API-интерфейсы REST помогают пользователям создавать собственные приложения, аналогичные приложению RainMaker. Для получения подробной информации, пожалуйста, посетите <https://swaggerapis.rainmaker.espressif.com/>.

API-интерфейсы Python

CLI на основе Python, который поставляется вместе с RainMaker SDK, предназначены для реализации всех функций, аналогичных функциям смартфона. Для получения дополнительной информации об API-интерфейсах Python, пожалуйста, посетите <https://book3.espressif.com/rm/python-api-reference>.

Интерфейс администратора (Admin CLI)

Для частного развертывания ESP RainMaker предусмотрен интерфейс администратора с более высоким уровнем доступа, позволяющий серийно генерировать сертификаты устройств.

3.2.1. Служба обработки заявок

Вся связь между устройствами RainMaker и облачным бэкендом осуществляется через MQTT+TLS. В контексте ESP RainMaker «заявка» – это процесс, в ходе которого устройства получают сертификаты от заявляющей службы для подключения к серверной части облака. Обратите внимание, что служба обработки заявок применима только к общедоступной службе RainMaker, в то время как для частного развертывания сертификаты устройств необходимо генерировать через интерфейс администратора.

RainMaker поддерживает три типа сервиса обработки заявок:

Автозаявка (Self Claiming)

Само устройство извлекает сертификаты с помощью секретного ключа, предварительно записанного в блоке eFuse, после подключения к интернету.

Заявка, управляемая хостом (Host Driven Claiming)

Сертификаты получены от хостинга разработки с учетной записью RainMaker.

Ассистируемые заявки (Assisted Claiming)

Сертификаты получаются с помощью приложений для смартфонов во время подготовки.

3.2.2. RainMaker Agent

Основная функция RainMaker Agent заключается в обеспечении подключения и оказании помощи прикладному уровню в обработке облачных данных по восходящей/нисходящей линии связи. Agent создан с помощью RainMaker SDK и разработан на основе проверенной платформы ESP-IDF с использованием ее компонентов, таких как RTOS, NVS и MQTT. На рис. 3.3 показана структура RainMaker SDK.

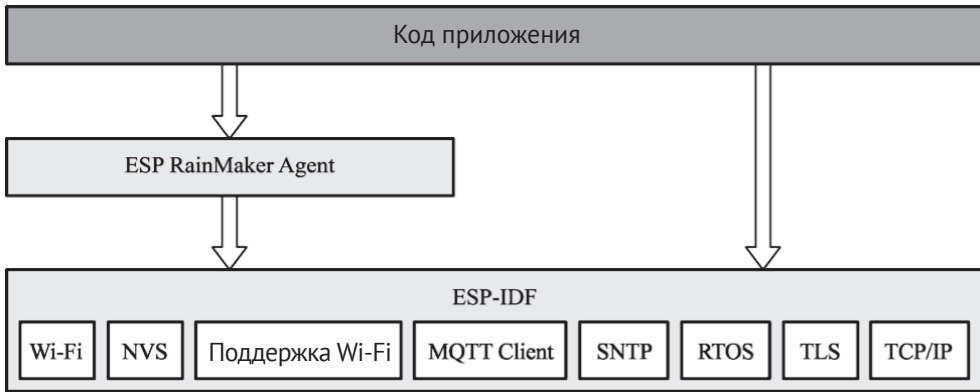


Рисунок 3.3. Структура SDK RainMaker

Пакет RainMaker SDK включает в себя две основные функции:

Подключение

- I. Сотрудничество со службой обработки заявок для получения сертификатов устройств.
- II. Подключение к серверной части облака с использованием защищенного протокола MQTT для обеспечения удаленного подключения и реализации удаленного контроля, создания отчетов о сообщениях, управления пользователями, устройствами и т. д. Оно использует компонент MQTT в ESP-IDF по умолчанию и предоставляет уровень абстракции для взаимодействия с другими стеками протоколов.
- III. Предоставление компонента поддержки Wi-Fi `wifi_provisioning` для подключения и подготовки к работе по Wi-Fi, компонента `esp_https_ota` для OTA-обновлений⁹ и компонента `esp_local_ctrl` для обнаружения и подключения локального устройства. Все эти цели могут быть достигнуты с помощью простой настройки.

⁹ OTA-обновления («Over The Air», букв. «по воздуху») – способ обновлений, особенно характерный для мобильных устройств (смартфонов), когда вместо «ручного» скачивания файла прошивки и его запуска устройство самостоятельно находит необходимые файлы на серверах производителя и производит их установку в автоматическом режиме.

Обработка данных

- I. Хранение сертификатов устройств, выданных службой обработки заявок, и данных, необходимых при запуске RainMaker, по умолчанию с использованием интерфейса, предоставляемого компонентом `nvs_flash`, и предоставление разработчикам API для непосредственного использования.
- II. Использование механизма обратного вызова для обработки облачных данных восходящей/нисходящей линии связи и автоматического разблокирования данных на прикладном уровне для удобства обработки разработчиками. Например, RainMaker SDK предоставляет расширенные интерфейсы для установки данных TSL (Thing Specification Language, «язык спецификации вещей»), необходимых для определения моделей TSL, для описания устройств и реализации таких функций, как хронометраж, обратный отсчет и голосовое управление. Для базовых интерактивных функций, таких как синхронизация, RainMaker SDK предоставляет решение, не требующее разработки, которое можно просто включить при необходимости. Затем агент RainMaker напрямую обрабатывает данные, отправит их в облако через соответствующую тему MQTT и вернет измененные данные в облачный сервер через механизм обратного вызова.

3.2.3. Облачный сервер

Облачный сервер построен на базе AWS Serverless Computing и реализуется с помощью системы управления идентификацией AWS Cognito, шлюза Amazon API Gateway, службы бессерверных вычислений AWS Lambda, базы данных Amazon DynamoDB, AWS IoT Core (ядро доступа IoT, обеспечивающее доступ MQTT и фильтрацию правил), почтового сервера Amazon Simple Email Service, сети быстрой доставки сообщений Amazon CloudFront, обработчика очереди сообщений Amazon Simple Queue Service и службы пакетного хранения Amazon S3. Сервер направлен на оптимизацию масштабируемости и безопасности. С помощью ESP RainMaker разработчики могут управлять устройствами без необходимости писать код в облаке. Сообщения, передаваемые устройствами, прозрачно передаются на приложения клиентов или другие сторонние сервисы.

В табл. 3.1 показаны облачные продукты AWS и функции, используемые в облачном сервере, а также другие продукты и функции, находящиеся в стадии разработки.

Таблица 3.1. Облачные продукты и функции AWS, используемые серверной частью облака

Облачный продукт AWS, используемый RainMaker	Функция
AWS Cognito	Управление учетными данными пользователей и поддержка сторонних входов в систему
AWS Lambda	Реализация основной бизнес-логики облачной серверной части
Amazon Timestream	Хранение данных во времени
Amazon DynamoDB	Хранение личной информации клиентов

Облачный продукт AWS, используемый RainMaker	Функция
AWS IoT Core	Поддержка связи по MQTT
Amazon SES	Предоставление услуг по отправке электронной почты
Amazon CloudFront	Ускорение управления доступом к серверному веб-сайту
Amazon SQS	Пересылка сообщений из AWS IoT Core

3.2.4. Клиент RainMaker

Клиенты RainMaker, такие как App и CLI, взаимодействуют с облачным сервером через REST API-интерфейсы. Подробную информацию и инструкции о REST API можно найти в документации Swagger, предоставленной Espressif. Клиент мобильного приложения RainMaker доступен как для систем iOS, так и для Android. Это позволяет подготавливать устройства, управлять ими и совместно использовать их, а также создавать и включать задачи обратного отсчета и подключаться к сторонним платформам. Клиент может автоматически загружать пользовательский интерфейс и значки в соответствии с конфигурацией, сообщаемой устройствами, и полностью отображать TSL-устройства.

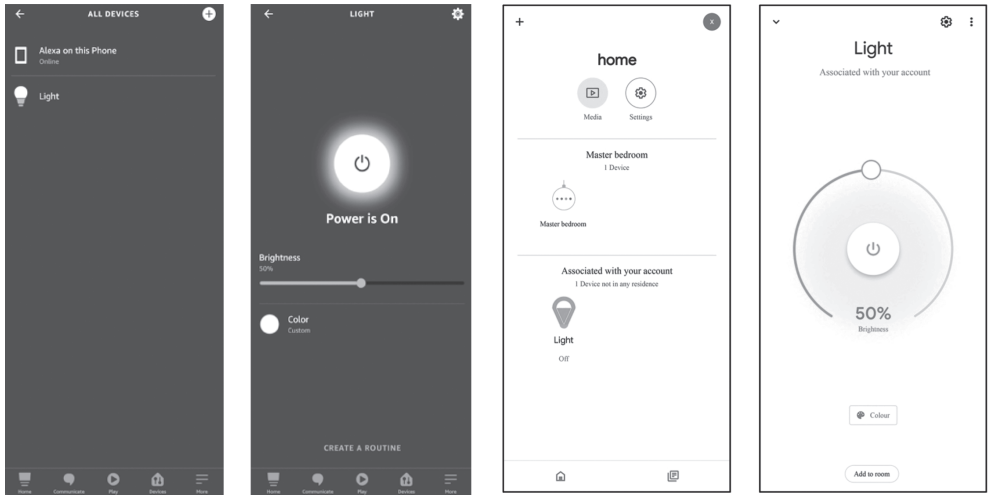
Например, если интеллектуальный источник света создан на основе примеров, предоставленных RainMaker SDK, значок и пользовательский интерфейс лампочки будут загружены автоматически после завершения подготовки. Пользователи могут изменять цвет и яркость подсветки через этот интерфейс и осуществлять стороннее управление, связав Alexa Smart Home Skill или Google Smart Home Actions со своими учетными записями ESP RainMaker. На рис. 3.4 показаны примеры значка и пользовательского интерфейса лампочки соответственно в приложениях Alexa, Google Home и ESP RainMaker.

3.3. Практика: ключевые моменты разработки с ESP RainMaker

Как только уровень драйвера устройства будет завершен, разработчики могут приступить к созданию моделей TSL и обработке данных нисходящей линии связи с использованием API, предоставляемых RainMaker SDK, и подключить базовые услуги ESP RainMaker на основе свойств и требований к продукту.

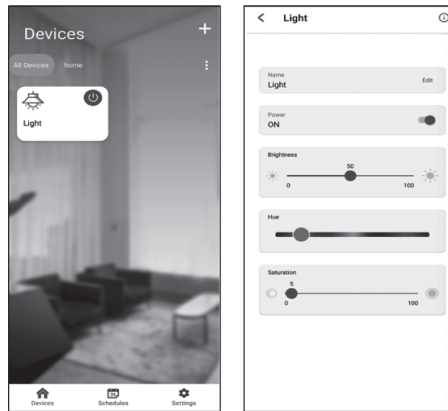
В разделе 9.4 этой книги будет рассказано о внедрении светодиодного умного освещения в RainMaker. Во время отладки разработчики могут использовать инструменты CLI в RainMaker SDK для взаимодействия с умным светильником (или вызывать REST API из Swagger).

В главе 10 будет подробно рассмотрено использование REST API при разработке приложений для смартфонов. О модернизации светодиодных умных светильников через OTA-обновления будет рассказано в главе 11. Если разработчики включили удаленный мониторинг ESP Insights, серверная часть ESP RainMaker отобразит данные ESP Insights. Подробности будут представлены в главе 15.



(a) Пример Alexa

(b) Пример Google Home



(c) Пример ESP RainMaker

Рисунок 3.4. Примеры значка и пользовательского интерфейса лампочки в приложениях Alexa, Google Home и ESP RainMaker

ESP RainMaker поддерживает частное развертывание, которое отличается от публичного развертывания на сервере RainMaker следующими особенностями:

Служба обработки заявок

Для создания сертификатов в частных развертываниях необходимо использовать интерфейс администратора RainMaker Admin CLI вместо подачи заявок. При применении общедоступного сервера разработчикам должны быть предоставлены права администратора для реализации обновления прошивки, но это нежелательно в коммерческих развертываниях. Поэтому ни отдельная служба аутентификации не может быть предоставлена для саморегистрации, ни права администратора для управления хостом или поддержки заявок.

Приложения для телефона

В частных развертываниях приложения необходимо настраивать и компилировать отдельно, для того чтобы убедиться, что системы учетных записей совместимы.

Сторонние логины и голосовая интеграция

Разработчики должны настраивать отдельно учетные записи Google и Apple Developer, чтобы включить сторонние входы в систему, точно так же осуществляется интеграция Alexa Skill и Google Voice Assistant.

Совет

Подробную информацию о развертывании в облаке см. на сайте <https://customer.rainmaker.espressif.com>. С точки зрения прошивки, миграция с общедоступного сервера на частный сервер требует только замены сертификатов устройств, что значительно повышает эффективность миграции и снижает стоимость миграции и вторичной отладки.

3.4. Особенности ESP RainMaker

Функции ESP RainMaker в основном ориентированы на три аспекта: управление пользователями, поддержку конечных пользователей и администраторов. Все функции поддерживаются как на общедоступных, так и на частных серверах, если не указано иное.

3.4.1. Управление пользователями

Функции управления пользователями позволяют конечным пользователям регистрироваться, входить в систему, менять пароли, извлекать пароли и т. д.

Регистрация и вход в систему

Методы регистрации и входа в систему, поддерживаемые RainMaker, включают:

- идентификатор электронной почты + пароль;
 - номер телефона + пароль;
 - учетную запись Google;
 - учетную запись Apple;
 - учетную запись GitHub (только для общедоступного сервера);
 - учетную запись Amazon (только для частного сервера).
-

Примечание

Зарегистрируйтесь с помощью Google/Amazon, предоставив RainMaker общий адрес электронной почты пользователя. Зарегистрируйтесь с помощью Apple, используя фиктивный адрес, который Apple назначает пользователю специально для сервиса RainMaker. Учетная запись RainMaker будет автоматически создана для пользователей, впервые вошедших в систему с учетной записью Google, Apple или Amazon.

Изменение пароля

Действительно только для входа на основе идентификатора электронной почты / номера телефона. Все другие активные сеансы после смены пароля будут закрыты. Согласно правилам AWS Cognito, завершённые сеансы могут оставаться активными до 1 часа.

Получение пароля

Действительно только для входа на основе идентификатора электронной почты / номера телефона.

3.4.2. Функции конечного пользователя

Функции, доступные конечным пользователям, включают локальное и удаленное управление, мониторинг, планирование, группировку устройств, общий доступ к устройствам, push-уведомления и стороннюю интеграцию.

Дистанционное управление и мониторинг

- Запрос конфигурации, значений параметров и состояние подключения для одного или всех устройств.
- Установка параметров для одного или нескольких устройств.

Локальный контроль и мониторинг

Мобильный телефон и устройство должны быть подключены к одной сети для локального управления.

Планирование

- Пользователи предварительно устанавливают определенные действия в определенное время.
- Не требуется подключение к интернету для устройства при выполнении расписания.
- Однократное действие или повтор (с указанием дат) для одного или нескольких устройств.

Группировка устройств

Поддерживает многоуровневую абстрактную группировку. Метаданные группы можно использовать для создания структуры «дом-комнаты».

Совместное использование устройств

Одно или несколько устройств могут использоваться совместно с одним или несколькими пользователями.

Всплывающее (push) уведомление

Конечные пользователи будут получать push-уведомления о таких событиях, как:

- добавлено или удалено новое устройство;
- устройство подключено к облаку;
- устройство отключено от облака;
- запросы на совместное использование устройств созданы/приняты/отклонены;
- предупреждающие сообщения, выдаваемые устройствами.

Интеграция со стороны

Alexa и Google Voice Assistant поддерживаются для управления устройствами RainMaker, включая светильники, выключатели, розетки, вентиляторы и датчики температуры.

3.4.3. Функции администратора

Функции администратора позволяют администраторам осуществлять регистрацию устройств, группировку устройств и OTA-обновления, а также для просмотра статистики и данных ESP Insights.

Регистрация устройства

Создание сертификатов устройств и регистрация в Admin CLI (только для частного сервера).

Группировка устройств

Создание абстрактных или структурированных групп на основе информации об устройстве (только для частного сервера).

OTA-обновления

Загрузка прошивки в зависимости от версии и модели на одно или несколько устройств или группу. Можно мониторить, отменить или заархивировать задания OTA.

Просмотр статистики

Доступная для просмотра статистика включает:

- регистрацию устройств (сертификаты, зарегистрированные администратором);
- активацию устройства (устройство подключено впервые);
- учетные записи пользователей;
- связь пользователя с устройством.

Просмотр данных ESP Insights

Доступные для просмотра данные ESP Insights включают:

- ошибки, предупреждения и ведение пользовательских логов;
- отчеты о сбоях и анализ;
- причины перезагрузки;
- такие показатели, как использование памяти, уровень мощности сигнала RSSI и т. д.;
- пользовательские показатели и переменные.

3.5. Резюме

В этой главе мы представили некоторые ключевые различия между общедоступным и частным развертываниями RainMaker. Частное решение ESP RainMaker, представленное компанией Espressif, отличается высокой надежностью и возможностью расширения. Все чипы серии ESP32 могут быть подключены и адаптированы к AWS, что значительно снижает стоимость. Разра-

ботчики могут сосредоточиться на проверке прототипов, не изучая облачные продукты AWS. Мы изложили реализацию и особенности ESP RainMaker, а также некоторые ключевые моменты разработки с использованием платформы.



Отсканируйте для загрузки ESP RainMaker для Android



Отсканируйте для загрузки ESP RainMaker для iOS

Глава 4

Настройка среды разработки

В этой главе основное внимание уделяется ESP-IDF, официальной среде разработки программного обеспечения для ESP32-C3. Мы расскажем, как настроить среду в различных операционных системах, представим структуру проекта и систему сборки ESP-IDF, а также использование соответствующих инструментов. Затем мы представим процесс компиляции и запуска примера проекта, в то же время предлагая подробное объяснение выходных лог-записей на каждом этапе.

4.1. Обзор ESP-IDF

ESP-IDF (Espressif IoT Development Framework) – это универсальная платформа для разработки IoT, предоставленная компанией Espressif Technology. В качестве основного языка разработки используется C/C++, поддерживает кросс-компиляцию под основными операционными системами, такими как Linux, Mac и Windows. Примеры программ, включенные в эту книгу, разработаны с использованием ESP-IDF, предлагающего следующие функции:

- **драйверы системного уровня SoC.** ESP-IDF включает в себя драйверы для ESP 32, ESP32-S2, ESP32-C3 и других микросхем. Эти драйверы включают периферийную библиотеку низкого уровня (LL), библиотеку аппаратного уровня абстракции (HAL), поддержку RTOS¹⁰ и программное обеспечение драйверов верхнего уровня и т. д.;
- **основные компоненты.** ESP-IDF включает в себя фундаментальные компоненты, необходимые для разработки интернета вещей. Они включают несколько стеков сетевых протоколов, таких как HTTP и MQTT, платформу управления питанием с динамической частотной модуляцией и такие функции, как flash-шифрование, безопасная загрузка и т. д.;
- **инструменты разработки и производства.** ESP-IDF предоставляет широко используемые инструменты для сборки, прошивки и отладки во время разработки и серийного производства (см. рис. 4.1), такие как система сборки на основе CMake, тулчейн¹¹ с кросс-компиляцией на основе GCC, инструмент отладки JTAG на основе OpenOCD и т. д.

Стоит отметить, что код ESP-IDF в первую очередь придерживается общедоступного кода по лицензии Apache 2.0. Пользователи могут разрабатывать лич-

¹⁰ RTOS – операционная система реального времени. О FreeRTOS для ESP32 см. далее в этой главе.

¹¹ Тулчейн (toolchain) – цепочка взаимозависимых программных инструментов для получения исполняемого кода из исходных текстов. Простой пример тулчейна: компилятор + компоновщик в классических языках программирования.

ное или коммерческое программное обеспечение без ограничений, соблюдая условия лицензии с открытым исходным кодом. Кроме того, пользователям предоставляется бессрочная патентная лицензия бесплатно, без обязательства открывать исходный код выполненных модификаций.

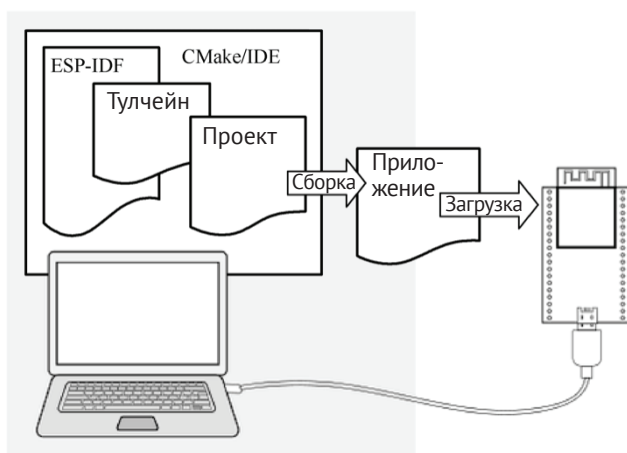


Рисунок 4.1. Инструменты сборки, прошивки и отладки для разработки и серийного производства

4.1.1. Версии ESP-IDF

Код ESP-IDF размещен на GitHub как проект с открытым исходным кодом. В настоящее время доступны три основные версии: v3, v4 и v5. Каждая основная версия обычно имеет различные подверсии, такие как v4.2, v4.3 и т. д. Espressif Systems обеспечивает 30-месячную поддержку исправлений ошибок и патчей безопасности для каждой выпущенной подверсии. Таким образом, регулярно выпускаются варианты версий, такие как v4.3.1, v4.2.2 и т. д. В табл. 4.1 показан статус поддержки различных версий ESP-IDF для чипов Espressif с указанием того, находятся ли они в стадии предварительного просмотра (т. е. поддержки предварительных версий, в которых могут отсутствовать определенные функции или полная документация) или уже поддерживаются официально.

Таблица 4.1. Статус поддержки различных версий ESP-IDF для чипов Espressif¹²

Series	v4.1	v4.2	v4.3	v4.4	v5.0
ESP32	supported	supported	supported	supported	supported
ESP32-S2		supported	supported	supported	supported
ESP32-C3			supported	supported	supported
ESP32-S3				supported	supported

¹² На июнь 2023 года.

Series	v4.1	v4.2	v4.3	v4.4	v5.0
ESP32-C2					supported
ESP32-H2				preview	preview

Изменения основных версий часто включают в себя корректировку структуры фреймворка и обновления системы компиляции. Например, основным изменением с версии 3.* на версию 4.* был постепенный перенос системы сборки с Make на CMake. С другой стороны, обновления минорных версий обычно влекут за собой добавление новых функций или поддержку новых чипов.

Важно различать и понимать взаимосвязь между стабильными версиями и ветками GitHub. Версии, помеченные как v*.* или v*.*.*, представляют собой стабильные версии, которые прошли полное внутреннее тестирование Espressif. После исправления кода, тулчейна и релиза документы для той же версии остаются без изменений. Однако ветки GitHub (например, ветвь *release/v4.3*) подвергаются частым коммитам¹³ кода, нередко ежедневным. Поэтому два фрагмента кода в одной и той же ветке могут отличаться, что требует от разработчиков немедленного обновления своего кода соответствующим образом.

4.1.2. Рабочий процесс ESP-DIFF Git

Espressif следует определенному рабочему процессу Git для ESP-IDF:

- новые изменения вносятся в ветку *master*, которая служит основной ветвью разработки. Версия ESP-IDF в основной ветке всегда содержит тег *-dev*, указывающий, что она в настоящее время находится в разработке, например *v4.3-dev*. Изменения на мастер-ветке будут сначала проверены и протестированы во внутренней репозитории Espressif, а затем отправлены на GitHub после завершения автоматического тестирования;
- как только для новой версии завершается разработка функций в основной ветке и она удовлетворяет критериям для участия в бета-тестировании, версия переходит в новую ветку, такую как *release/v4.3*. Кроме того, эта новая ветка может быть помечена как предварительная версия, например *v4.3-beta1*. Разработчики могут обратиться к платформе GitHub, чтобы получить доступ к полному списку ветвей и тегов для ESP-IDF. Важно отметить, что бета-версия (предварительная версия) все еще может содержать значительное количество известных проблем. Поскольку бета-версия постоянно тестируется, исправления ошибок добавляются как в эту версию, так и в основную ветку *master* одновременно. Тем временем на главной ветке *master*, возможно, уже приступили к разработке новых функций для следующей версии. Когда тестирование почти завершено, в ветку добавляется метка *release candidate (rc)*, указывающая на то, что она является потенциальным кандидатом для официального выпуска, например *v4.3-rc1*. На данном этапе ветка остается предварительной версией;

¹³ Коммит кода (code commits, фиксация кода) – в системах контроля версий коммитом кода называется операция, отправляющая последние изменения исходного кода в репозиторий, делая эти изменения частью главной версии.

- если не обнаружено никаких серьезных ошибок или не сообщается о них, предварительная версия в конечном итоге получает метку основной версии (например, v5.0) или метку младшей версии (например, v4.3) и становится официальной версией выпуска, которая задокументирована на странице примечаний к выпуску. Впоследствии все ошибки, выявленные в этой версии, исправляются в ветке выпуска. После завершения ручного тестирования ветке присваивается метка версии с исправлением ошибок (например, v4.3.2), которая также отражается на странице примечаний к выпуску.

4.1.3. Выбор подходящей версии

Поскольку ESP-IDF официально начал поддерживать ESP32-C3 с версии v4.3, а v4.4 еще не выпущена официально на момент написания этой книги, в данной книге используется v4.3.2, которая является исправленной версией v4.3. Однако важно отметить, что когда вы читаете эту книгу, уже может быть доступна версия 4.4 или более новые. При выборе версии мы советуем следующее:

- для **разработчиков начального уровня** рекомендуется выбирать стабильную версию v4.3 или ее исправленную версию, которая совпадает с версией примера, приведенного в этой книге;
- в целях **серийного производства** рекомендуется использовать последнюю стабильную версию, чтобы воспользоваться самой современной технической поддержкой;
- если вы собираетесь экспериментировать с **новыми чипами** или исследовать новые функции продукта, пожалуйста, используйте основную ветку *master*. Последняя версия содержит все последние функции, но имейте в виду, что могут присутствовать известные или неизвестные ошибки;
- если используемая стабильная версия не содержит желаемых новых функций и вы хотите **минимизировать риски**, связанные с основной ветвью, рассмотрите возможность использования соответствующей ветви выпуска, такой как ветвь *release/v4.4*. Репозиторий Espressif на GitHub сначала создаст ветку *release/v4.4*, а затем выпустит стабильную версию v4.4 на основе конкретного исторического снимка этой ветки, после завершения разработки и тестирования всех функций.

4.1.4. Обзор каталога ESP-IDF SDK

ESP-IDF SDK состоит из двух основных каталогов: *esp-idf* и *.espressif*. Первый содержит файлы исходного кода репозитория ESP-IDF и скрипты компиляции, в то время как второй в основном хранит тулчейны для компиляции и другое программное обеспечение. Знакомство с этими двумя каталогами поможет разработчикам лучше использовать доступные ресурсы и ускорить процесс разработки. Структура каталогов ESP-IDF следующая:

(1) **Каталог кода репозитория ESP-IDF (~/*esp/esp-idf*)**, показан на рис. 4.2.

a. Компоненты каталога *components*

Этот основной каталог объединяет многочисленные основные программные компоненты ESP-IDF. Ни один код проекта не может быть скомпилирован без использования компонентов из этого каталога. Он включает в себя поддержку драйверов для различных чипов Espressif.

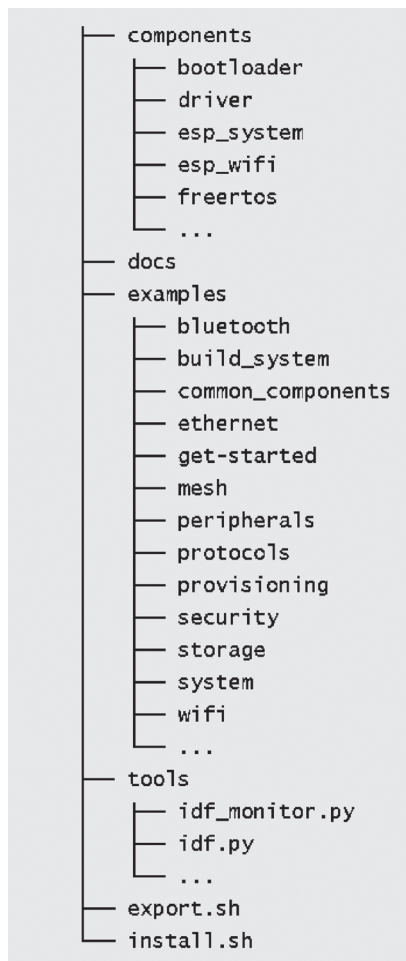


Рисунок 4.2. Каталог кода репозитория ESP-IDF

Благодаря поддержке уровней от интерфейсов библиотеки LL и библиотеки HAL для периферийных устройств до драйвера верхнего уровня и виртуальной файловой системы (VFS) разработчики могут выбирать подходящие компоненты на разных уровнях в соответствии со своими потребностями. ESP-IDF также поддерживает несколько стандартных стеков сетевых протоколов, таких как TCP/IP, HTTP, MQTT, WebSocket и т. д. Разработчики могут использовать знакомые интерфейсы, такие как Socket, для создания сетевых приложений. Компоненты обеспечивают комплексную функциональность, которая может быть легко интегрирована в приложения, что позволяет разработчикам сосредоточиться исключительно на бизнес-логике. Некоторые общие компоненты включают в себя:

- *driver*. Этот компонент содержит для различных серий микросхем Espressif программы-драйверы периферийных устройств, таких как

GPIO, I2C, SPI, UART, светодиоды (PWM) и т. д. Программы-драйверы периферийных устройств в этом компоненте предлагают абстрактные интерфейсы, не зависящие от микросхемы. Каждое периферийное устройство имеет общий заголовочный файл (например, *gpio.h*), что избавляет от необходимости решать вопросы поддержки, связанные с различными чипами;

- *esp_wifi*. Wi-Fi как специальное периферийное устройство рассматривается как отдельный компонент. Он включает в себя множество API-интерфейсов, таких как инициализация различных режимов драйвера Wi-Fi, настройка параметров и обработка событий. Некоторые функции этого компонента предоставляются в виде библиотек статических ссылок. Для удобства использования ESP-IDF также предоставляет исчерпывающую документацию по драйверам;
- *freertos*. Этот компонент содержит полный код FreeRTOS. Помимо обеспечения всесторонней поддержки этой операционной системы, Espressif также расширила поддержку для двухъядерных чипов. Для двухъядерных чипов, таких как ESP 32 и ESP32-S3, пользователи могут создавать задачи на выбранных ядрах.

b. Документы каталога docs

Этот каталог содержит документы по разработке, связанные с ESP-IDF, включая начальное руководство (Get Started Guide), справочное руководство по API (API Reference Manual), руководство по разработке (Development Guide) и т. д.

Примечание

После компиляции автоматизированными средствами содержимое этого каталога развертывается по адресу <https://docs.espressif.com/projects/esp-idf>. Пожалуйста, убедитесь, что вы переключили целевой документ на ESP32-C3 и выбрали заданную версию ESP-IDF.

c. Скриптовые инструменты tools

Этот каталог содержит часто используемые интерфейсные инструменты компиляции, такие как *idf.py*, инструмент мониторинга терминала *idf_monitor.py* и т. д. Подкаталог *cmake* также содержит файлы основных сценариев системы компиляции, служащие основой для реализации правил компиляции ESP-IDF. При добавлении переменных окружения содержимое каталога *tools* также добавляется в системные переменные окружения, что позволяет *idf.py* выполняться непосредственно из папки проекта.

d. Каталог программных примеров examples

Этот каталог содержит обширную коллекцию примеров программ ESP-IDF, демонстрирующих использование API компонентов. Примеры организованы в различные подкаталоги в зависимости от их категорий:

- *get-started*. Этот подкаталог содержит примеры начального уровня, такие как «hello world» и «blink», чтобы помочь пользователям понять основы;
- *bluetooth*. Здесь вы можете найти примеры, связанные с Bluetooth, включая Bluetooth LE Mesh, Bluetooth LE HID, BluFi и многое другое;
- *wifi*. Этот подкаталог посвящен примерам Wi-Fi, включая базовые программы, такие как WiFi SoftAP, Wi-Fi Station, espnow, а также примеры фирменных протоколов связи от Espressif. Он включает в себя еще несколько примеров прикладного уровня, основанных на Wi-Fi, таких как Iperf, Sniffer и Smart Config;
- *peripherals*. Этот обширный подкаталог далее разделен на множество подпапок в зависимости от названий периферийных устройств. В основном он содержит примеры периферийных драйверов для чипов Espressif, причем каждый пример содержит несколько под-примеров. Например, подкаталог *gpio* содержит два примера: GPIO и матричную клавиатуру GPIO. Важно отметить, что не все примеры в этом каталоге применимы к ESP32-C3. Например, примеры в разделе *usb/host* применимы только к периферийным устройствам со встроенным хостом USB (например, ESP32-S3), а ESP32-C3 этого периферийного устройства не имеет. Система компиляции обычно выдает подсказки при установке целевого значения. В файле *README* каждого примера перечислены поддерживаемые чипы;
- *protocols*: этот подкаталог содержит примеры различных протоколов связи, включая MQTT, HTTP, HTTP-сервер, PPPoS, Modbus, mDNS, SNTP, охватывающие широкий спектр примеров протоколов связи, необходимых для разработки IoT;
- *provisioning*: здесь вы найдете примеры подготовки для различных средств, такие как подготовка Wi-Fi или Bluetooth LE;
- *system*: этот подкаталог содержит примеры отладки системы (например, трассировка стека, трассировка времени выполнения, мониторинг задач), примеры управления питанием (например, различные спящие режимы, сопроцессоры) и примеры, относящиеся к общим системным компонентам, таким как консольный терминал, цикл обработки событий и системный таймер;
- *storage*: в этом подкаталоге вы найдете примеры всех файловых систем и механизмов хранения, поддерживаемых ESP-IDF (таких как чтение и запись флеш-памяти, SD-карт и других носителей информации), а также примеры использования энергонезависимого хранилища (non-volatile storage, NVS), FatFS, SPIFFS и других операций с файловой системой;
- *security*: этот подкаталог содержит примеры, связанные с аппаратным шифрованием устройств хранения.

(2) Каталог цепочки инструментов компиляции ESP-IDF (~/.espressif), показан на рис. 4.3.

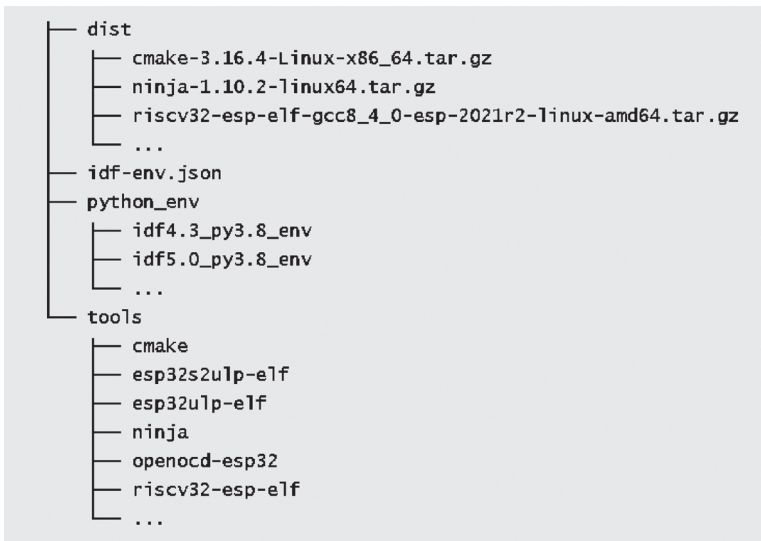


Рисунок 4.3. Каталог тулчейна компиляции ESP-IDF

a. Каталог дистрибуции программного обеспечения `dist`

Набор инструментов ESP-IDF и другое программное обеспечение распространяются в виде сжатых пакетов. В процессе установки сначала загружается сжатый пакет в каталог `dist`, затем установщик извлекает его в нужный каталог. Как только установка будет завершена, содержимое этого каталога может быть безопасно удалено.

b. Каталог виртуальной среды Python `python_env`

Различные версии ESP-IDF зависят от определенных версий пакетов Python. Установка этих пакетов непосредственно на одном и том же хосте может привести к конфликтам между версиями. Чтобы решить эту проблему, ESP-IDF использует виртуальные среды Python для изоляции различных версий. С помощью этого механизма разработчики могут устанавливать несколько версий ESP-IDF на одном хосте и легко переключаться между ними, импортируя различные переменные среды.

c. Каталог тулчейна компиляции ESP-IDF `tools`

Этот каталог в основном содержит инструменты кросс-компиляции, необходимые для компиляции проектов ESP-IDF, такие как CMake tools, Ninja build tools и тулчейн gcc, которые генерируют конечную исполняемую программу. Кроме того, в этом каталоге находится стандартная библиотека языка C/C++ вместе с соответствующими заголовочными файлами. Если программа ссылается на системный заголовочный файл, например `#include <stdio.h>`, инструменты компиляции найдут файл `stdio.h` в этом каталоге.

4.2. Настройка среды разработки ESP-IDF

Среда разработки ESP-IDF поддерживает основные операционные системы, такие как Windows, Linux и macOS. В этом разделе будет рассказано о том, как настроить среду разработки в каждой системе. Рекомендуется работать с ESP32-C3 в системе Linux, которая будет подробно представлена здесь. Многие инструкции применимы на разных платформах из-за сходства инструментов разработки. Поэтому рекомендуется внимательно ознакомиться с содержанием данного раздела.

Примечание

Вы можете обратиться к онлайн-документам, доступным по адресу <https://book3.espressif.com/esp32c3>, где представлено описание команд, упомянутых в этом разделе.

4.2.1. Настройка среды разработки ESP-IDF в Linux

Инструменты разработки и отладки GNU, необходимые для среды разработки ESP-IDF, являются родными для системы Linux. Кроме того, терминал командной строки в Linux является мощным и удобным в использовании, что делает его идеальным выбором для разработки на ESP32-C3. Вы можете выбрать предпочитаемый вами дистрибутив Linux, но мы рекомендуем использовать Ubuntu или другие системы на базе Debian. В этом разделе приведены рекомендации по настройке среды разработки ESP-IDF в Ubuntu 20.04.

1. Установка необходимых пакетов

Откройте новый терминал и выполните следующую команду, чтобы установить все необходимые пакеты. Команда автоматически пропустит пакеты, которые уже установлены.

```
$ sudo apt-get install git wget flex bison gperf python3 python3-pip python3-setuptools  
cmake ninja-build ccache libffi-dev libssl-dev dfu-util libusb-1.0-0
```

Совет

Вам нужно использовать учетную запись администратора и пароль для ввода команды выше. По умолчанию при вводе пароля информация отображаться не будет. Просто нажмите клавишу **Enter** для продолжения процедуры.

Git – это ключевой инструмент управления кодом в ESP-IDF. После успешной настройки среды разработки вы можете использовать команду `git log` для просмотра всех изменений кода, внесенных с момента создания ESP-IDF. Кроме того, он также используется в ESP-IDF для подтверждения информации о версии, которая необходима для установки правильного тулчейна, соответствующего конкретным версиям. Наряду с Git другие важные системные инструменты включают Python. ESP-IDF включает в себя множество сценариев автоматизации, написанных на Python. Такие инструменты, как CMake, Ninja-

build и Sсache, широко используются в проектах на C/C++ и по умолчанию служат инструментами компиляции и построения кода в ESP-IDF. libusb-1.0-0 и dfu-util являются основными драйверами, используемыми для последовательной связи по USB и прошивки встроеного ПО.

Как только пакеты программного обеспечения будут установлены, вы можете использовать команду `apt show <имя_пакета>` для получения подробных описаний каждого пакета. Например, используйте `apt show git` для печати информации из описания инструмента Git.

Вопрос: что делать, если версия Python не поддерживается?

Ответ: для ESP-IDF версии 4.3 требуется версия Python не ниже версии 3.6. Для более старых версий Ubuntu, пожалуйста, вручную загрузите и установите более новую версию Python и установите Python3 в качестве среды Python по умолчанию. Вы можете найти подробные инструкции, выполнив поиск по ключевому слову `update-alternatives python`.

2. Загрузка кода репозитория ESP-IDF

Откройте терминал и создайте папку с именем `esp` в вашем домашнем каталоге, используя команду `mkdir`. Вы можете выбрать другое имя для папки, если хотите. Используйте команду `cd`, чтобы войти в папку.

```
$ mkdir -p ~/esp
$ cd ~/esp
```

Используйте команду `git clone` для загрузки кода репозитория ESP-IDF, как показано ниже:

```
$ git clone -b v4.3.2 --recursive https://github.com/espressif/esp-idf.git
```

В приведенной выше команде параметр `-b v4.3.2` указывает версию для загрузки (в данном случае версию 4.3.2). Параметр `--recursive` гарантирует, что все вложенные репозитории ESP-IDF загружаются рекурсивно. Информацию о вложенных репозиториях можно найти в файле `.gitmodules`.

3. Установка тулчейна разработки ESP-IDF

Espressif предоставляет автоматизированный скрипт `install.sh`, чтобы загрузить и установить набор инструментов. Этот скрипт проверяет текущую версию ESP-IDF и среду операционной системы, а затем загружает и устанавливает соответствующую версию пакетов инструментов Python и тулчейн компиляции. Путь установки цепочки инструментов по умолчанию `~/espressif`. Все, что вам нужно сделать, – это перейти в каталог `esp-idf` и запустить `install.sh`.

```
$ cd ~/esp/esp-idf
$ ./install.sh
```

Если установка прошла успешно, терминал выведет сообщение:

```
All done!
```

Это означает, что вы успешно настроили среду разработки ESP-IDF.

4.2.2. Настройка среды разработки ESP-IDF в Windows

1. Загрузите установщик ESP-IDF tools

Совет

Рекомендуется настраивать среду разработки ESP-IDF в Windows 10 или выше. Вы можете скачать программу установки с сайта <https://dl.espressif.com/dl/esp-idf/>. Установщик также является программным обеспечением с открытым исходным кодом, и его исходный код можно просмотреть по адресу <https://github.com/espressif/idf-installer>.

- **Онлайн-установщик ESP-IDF tools**

Этот установщик относительно невелик, размером около 4 МБ, другие пакеты и код будут загружены в процессе установки. Преимущество онлайн-установщика заключается в том, что в процессе установки можно не только загружать программные пакеты и код по запросу, но и устанавливать все доступные версии ESP-IDF и последнюю ветвь кода GitHub (например, главную ветвь). Недостатком является то, что в процессе установки требуется подключение к Сети, что может привести к сбою установки из-за проблем с подключением.

- **Автономный установщик ESP-IDF tools**

Этот установщик больше, размером около 1 ГБ, и содержит все пакеты программного обеспечения и код, необходимые для настройки среды. Основным преимуществом автономной программы установки является то, что ее можно использовать на компьютерах без доступа в интернет и, как правило, она имеет более высокий процент успешной установки. Следует отметить, что автономный установщик может устанавливать только стабильные версии ESP-IDF, обозначенные v*.* или v*.*.*.

2. Запустите программу установки ESP-IDF tools

После загрузки подходящей версии установщика (возьмем, например, ESP-IDF Tools Offline 4.3.2) дважды щелкните исполняемый файл, чтобы запустить интерфейс установки ESP-IDF. Ниже показано, как установить стабильную версию ESP-IDF v4.3.2 с помощью автономного установщика.

- (1) В окне **Select installation language** (Выберите язык установки), показанном на рис. 4.4, выберите язык, который будет использоваться, из выпадающего списка:
- (2) После выбора языка нажмите **ОК**, чтобы открыть интерфейс **License agreement** (Лицензионное соглашение) (рис. 4.5). Внимательно прочитав лицензионное соглашение об установке, выберите **I accept the agreement** (Я принимаю соглашение) и нажмите **Next** (Далее).

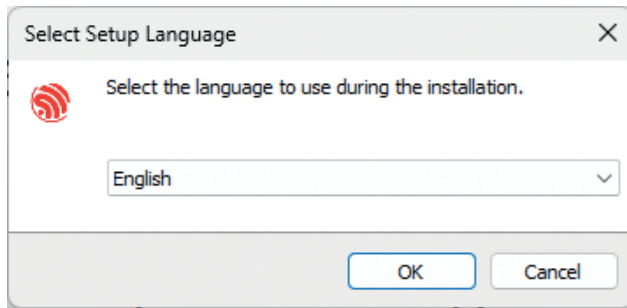


Рисунок 4.4. Интерфейс выбора языка установки

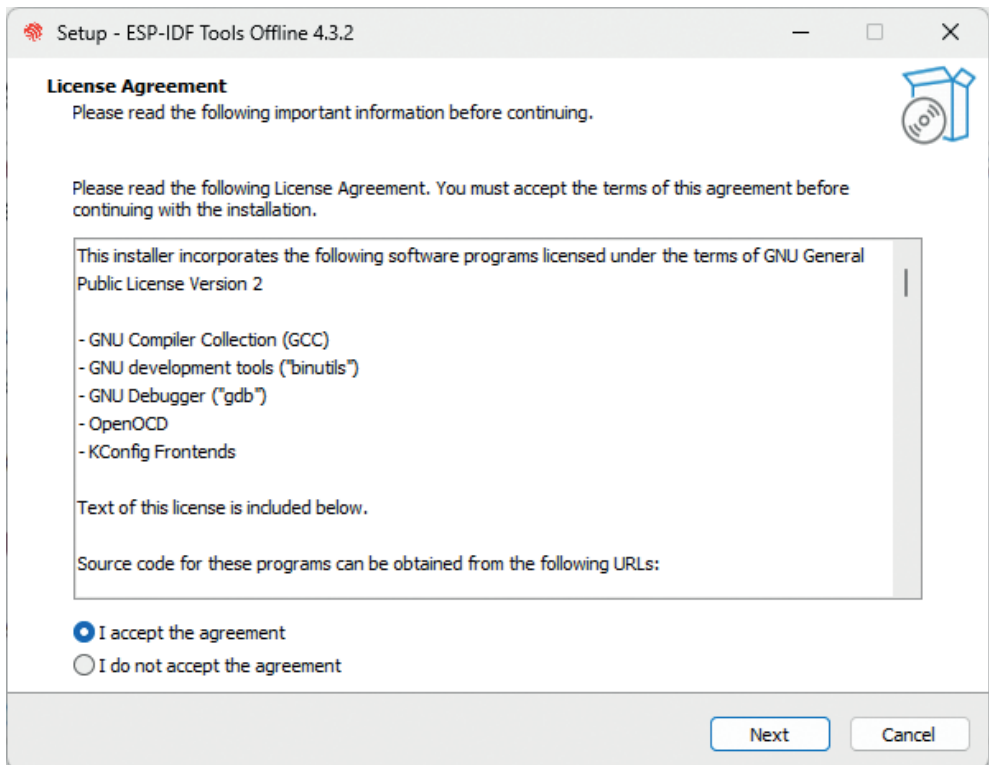


Рисунок 4.5 Интерфейс License agreement

- (3) Просмотрите конфигурацию системы в окне **Pre-installation system check** (Проверка системы перед установкой) (рис. 4.6). Проверьте версию Windows и информацию об установленном антивирусном программном обеспечении. Нажмите **Next**, если все элементы конфигурации в норме. В противном случае вы можете нажать **Full log** (Полный лог) для получения решений, основанных на ключевых элементах.

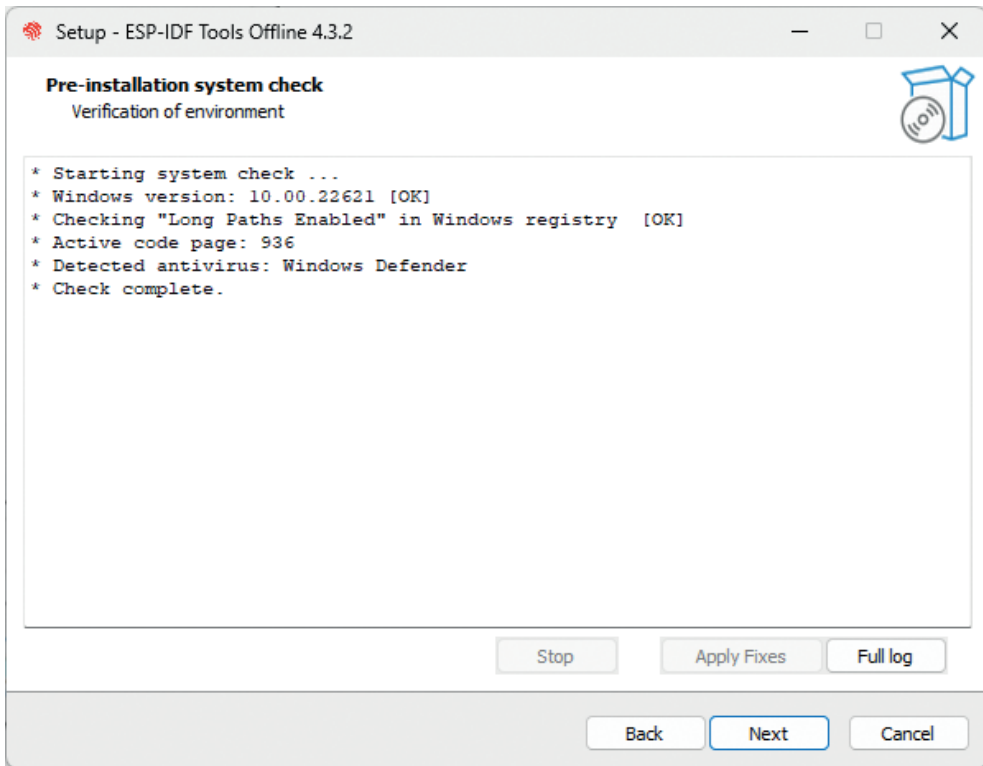


Рисунок 4.6. Интерфейс проверки системы перед установкой

Совет

Вы можете отправить логи на <https://github.com/espressif/idf-installer/issues> для получения помощи.

- (4) Выберите каталог установки ESP-IDF (на рис. 4.7 для примера выбран `D:/.espressif`) и нажмите кнопку **Next** (Далее). Пожалуйста, обратите внимание, что `.espressif` здесь является скрытым каталогом. После завершения установки вы можете просмотреть конкретное содержимое этого каталога, открыв файловый менеджер с отображением скрытых элементов.
- (5) Проверьте компоненты, которые необходимо установить, как показано на рис. 4.8. Рекомендуется использовать опцию по умолчанию, то есть завершить установку, а затем нажать **Next**.
- (6) Подтвердите установку компонентов и нажмите **Install** (Установить), чтобы запустить процесс автоматической установки, как показано на рис. 4.9. Процесс установки может длиться десятки минут, индикатор выполнения процесса установки показан на рис. 4.10. Пожалуйста, подождите немного.

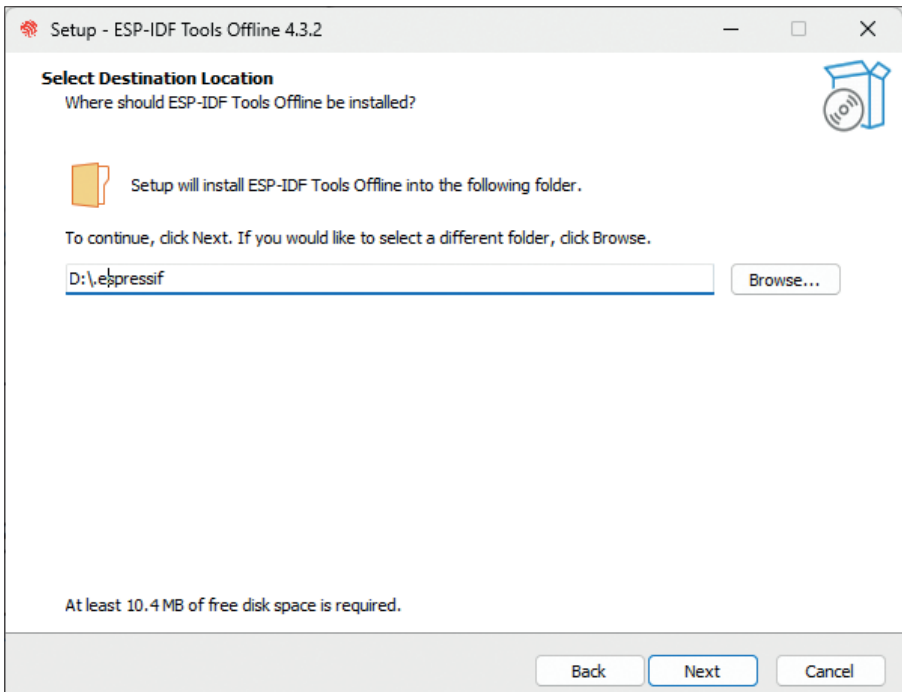


Рисунок 4.7. Выберите каталог установки ESP-IDF

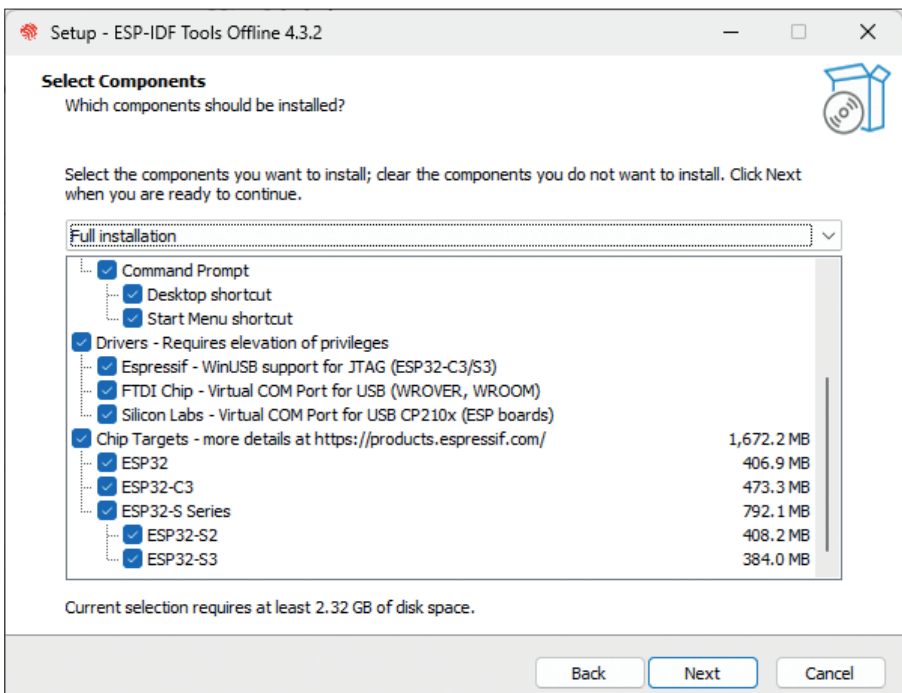


Рисунок 4.8. Выбор компонентов для установки

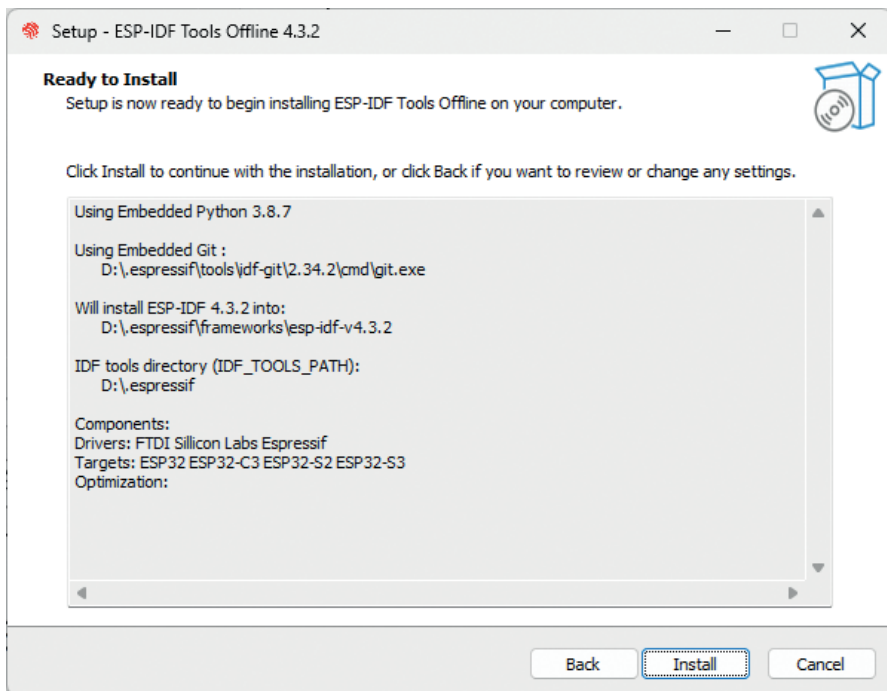


Рисунок 4.9. Подготовка к установке

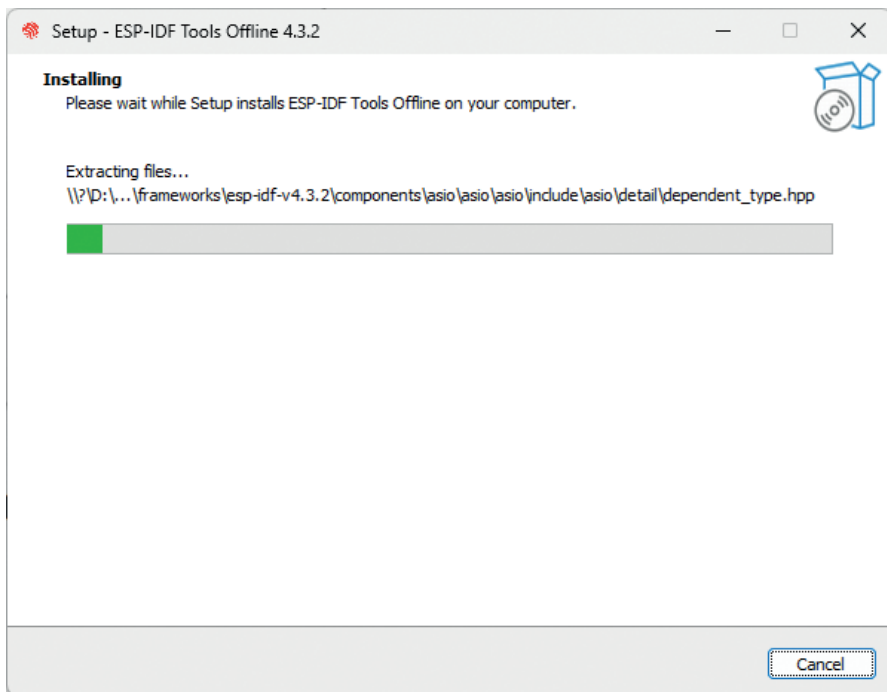


Рисунок 4.10. Индикатор выполнения установки

- (7) После завершения установки рекомендуется установить флажок **Register the ESP-IDF Tools executables as Windows Defender exclusions...** (Зарегистрировать исполняемые файлы инструментов ESP-IDF как исключения защитника Windows...), чтобы предотвратить удаление файлов антивирусным программным обеспечением. Добавление элементов в исключения также позволяет пропускать частые проверки антивирусным программным обеспечением, что значительно повышает эффективность компиляции кода в системе Windows. Нажмите **Finish** (Готово), чтобы завершить установку среды разработки (рис. 4.11). Вы можете установить флажок **Run ESP-IDF PowerShell environment** (Запустить среду ESP-IDF PowerShell) или **Run ESP-IDF command prompt** (Запустить командную строку ESP-IDF). Запустите окно компиляции сразу после установки для обеспечения нормального функционирования среды разработки.

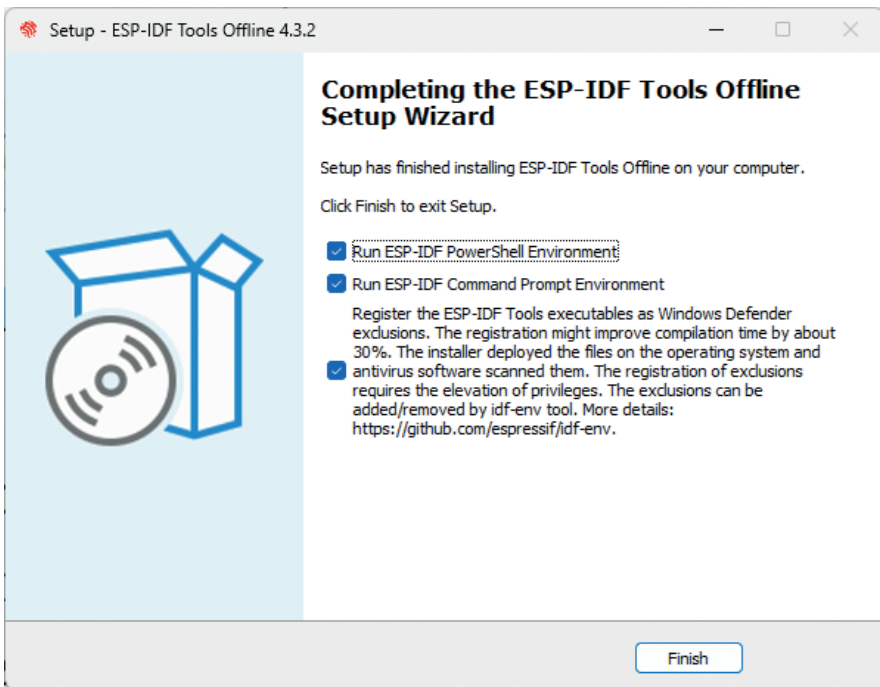


Рисунок 4.11. Установка завершена

- (8) Откройте установленную среду разработки в списке программ (либо ESP-IDF 4.3 CMD или ESP-IDF 4.3 PowerShell terminal, как показано на рис. 4.12), и переменная окружения ESP-IDF будет автоматически добавлена при запуске в терминале. После этого вы можете использовать команду *idf.py* для проведения операций. Открытый командный файл ESP-IDF 4.3 показан на рис. 4.13.

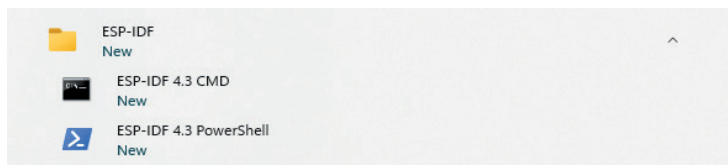


Рисунок 4.12. Установленная среда разработки

```
Administrator: ESP-IDF 4.3 CA X
Python 3.8.7
Using Git in D:\.espressif\tools\idf-git\2.34.2\cmd\
git version 2.34.1.windows.1
Setting IDF_PATH: D:\.espressif\frameworks\esp-idf-v4.3.2

Adding ESP-IDF tools to PATH...
Not using an unsupported version of tool cmake found in PATH: 3.23.2.
D:\.espressif\tools\xtensa-esp32-elf\esp-2021r2-8.4.0\xtensa-esp32-elf\bin
D:\.espressif\tools\xtensa-esp32s2-elf\esp-2021r2-8.4.0\xtensa-esp32s2-elf\bin
D:\.espressif\tools\xtensa-esp32s3-elf\esp-2021r2-8.4.0\xtensa-esp32s3-elf\bin
D:\.espressif\tools\riscv32-esp-elf\esp-2021r2-8.4.0\riscv32-esp-elf\bin
D:\.espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf-binutils\bin
D:\.espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf-binutils\bin
D:\.espressif\tools\cmake\3.16.4\bin
D:\.espressif\tools\openocd-esp32\v0.10.0-esp32-20211111\openocd-esp32\bin
D:\.espressif\tools\idf-exe\1.0.1\
D:\.espressif\tools\ccache\3.7\
D:\.espressif\tools\dfu-util\0.9\dfu-util-0.9-win64
D:\.espressif\frameworks\esp-idf-v4.3.2\tools

Checking if Python packages are up to date...
Python requirements from D:\.espressif\frameworks\esp-idf-v4.3.2\requirements.txt are satisfied.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

idf.py build

D:\.espressif\frameworks\esp-idf-v4.3.2>
```

Рисунок 4.13. ESP-IDF 4.3 CMD

4.2.3. Настройка среды разработки ESP-IDF на Mac

Процесс установки среды разработки ESP-IDF в системе Mac такой же, как и в системе Linux. Команды для загрузки кода репозитория и установки набора инструментов точно такие же. Немного отличаются только команды для установки зависимых пакетов.

1. Установка зависимых пакетов

Откройте терминал и установите *pip*, инструмент управления пакетами Python, выполнив следующую команду:

```
% sudo easy install pip
```

Установите инструмент управления пакетами для macOS Homebrew, выполнив следующую команду:

```
% /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Установите необходимые пакеты, выполнив следующую команду:

```
% brew python3 install cmake ninja ccache dfu-util
```

2. Загрузка кода репозитория ESP-IDF

Следуйте инструкциям, приведенным в разделе 4.2.1, чтобы загрузить код репозитория ESP-IDF. Действия такие же, как и для загрузки в системе Linux.

3. Установите тулчейн разработки ESP-IDF

Следуйте инструкциям, приведенным в разделе 4.2.1, чтобы установить тулчейн разработки ESP-IDF. Действия такие же, как и для установки в системе Linux.

4.2.4. Установка VS Code

По умолчанию ESP-IDF SDK не включает инструмент редактирования кода (хотя последняя версия ESP-IDF installer для Windows предлагает возможность установки ESP-IDF Eclipse). Вы можете использовать любой инструмент редактирования текста по вашему выбору, чтобы отредактировать код, а затем скомпилировать его с помощью команд терминала.

Одним из популярных инструментов редактирования кода является VS Code (Visual Studio Code), который представляет собой бесплатный и многофункциональный редактор кода с удобным интерфейсом. Он предлагает различные плагины, которые предоставляют такие функции, как навигация по коду, подсветка синтаксиса, управление версиями Git и интеграция терминала. Кроме того, Espressif разработала специальный плагин под названием Espressif IDF для VS Code, который упрощает настройку и отладку проекта.

Вы можете использовать команду `code` в терминале, чтобы быстро открыть текущую папку в VS Code. В качестве альтернативы можете использовать сочетание клавиш **Ctrl+~**, чтобы открыть системную консоль терминала по умолчанию в VS Code.

Совет

Рекомендуется применять VS Code для разработки кода ESP32-C3. Скачайте и установите последнюю версию VS Code по адресу <https://code.visualstudio.com/>.

4.2.5. Знакомство со сторонними средами разработки

В дополнение к официальной среде разработки ESP-IDF, которая в основном использует язык C, ESP32-C3 также поддерживает другие основные языки программирования и широкий спектр сторонних сред разработки. Некоторые популярные варианты приведены далее.

Arduino:

платформа с открытым исходным кодом как для аппаратного, так и для программного обеспечения, поддерживающая различные микроконтроллеры, включая ESP32-C3.

Платформа использует язык C++ и предлагает упрощенный и стандартизированный API, обычно называемый языком Arduino. Arduino широко используется при разработке прототипов и в образовательных контекстах. Она предоставляет расширяемый пакет программного обеспечения и среду IDE, которая позволяет легко выполнять компиляцию и перепрошивку.

MicroPython:

интерпретатор языка Python 3, предназначенный для работы на платформах встраиваемых микроконтроллеров.

С помощью простого скриптового языка (языка сценариев) он может напрямую обращаться к периферийным ресурсам ESP32-C3 (таким как UART, SPI и I2C) и коммуникационным функциям (таким как Wi-Fi и Bluetooth LE).

Это упрощает взаимодействие с оборудованием. MicroPython в сочетании с широкими возможностями библиотеки математических операций Python позволяет реализовать сложные алгоритмы на ESP32-C3, облегчающие разработку приложений, связанных с искусственным интеллектом. В языке сценариев нет необходимости в повторной компиляции; изменения в сценариях могут выполняться напрямую.

NodeMCU:

интерпретатор языка LUA, разработанный для чипов серии ESP.

Он поддерживает практически все периферийные функции чипов ESP и легче, чем MicroPython. Подобно MicroPython, NodeMCU использует скриптовый язык, что устраняет необходимость в повторной компиляции.

Кроме того, ESP32-C3 также поддерживает операционные системы NuttX и Zephyr. NuttX – это операционная система реального времени, предоставляющая POSIX-совместимый интерфейс, повышающий переносимость приложений. Zephyr – это небольшая операционная система реального времени, специально разработанная для приложений интернета вещей. Она включает в себя многочисленные программные библиотеки, необходимые для разработки интернета вещей, постепенно превращаясь во всеобъемлющую программную экосистему.

В этой книге не приведены подробные инструкции по установке для вышеупомянутых сред разработки. Вы можете установить среду разработки в соответствии с вашими требованиями, следуя соответствующей документации и инструкциям.

4.3. Система компиляции ESP-IDF

4.3.1. Основные концепции системы компиляции

Проект ESP-IDF представляет собой набор из основной программы с функцией ввода и множеством независимых функциональных компонентов. Например,

проект, который управляет светодиодными переключателями, в основном состоит из программы входа `main` и компонента `driver`, который управляется через GPIO. Если вы хотите реализовать дистанционное управление светодиодами, вам также необходимо добавить Wi-Fi, стек протоколов TCP/IP и т. д.

Система компиляции может компилировать, связывать и генерировать исполняемые файлы (`.bin`) для кода с помощью набора правил построения. Система компиляции ESP-IDF версии 4.0 и выше по умолчанию основана на CMake, а сценарий компиляции `CMakeLists.txt` может использоваться для управления поведением кода при компиляции. В дополнение к поддержке базового синтаксиса CMake система компиляции ESP-IDF также определяет набор правил компиляции по умолчанию и функции CMake, и вы можете написать сценарий компиляции с помощью простых инструкций.

4.3.2. Структура файла проекта

Проект – это папка, содержащая основную программу входа, пользовательские компоненты и файлы, необходимые для сборки исполняемых приложений, такие как сценарии компиляции, конфигурационные файлы, таблицы разделов и т. д. Проекты можно копировать и передавать дальше, а такой же исполняемый файл может быть скомпилирован и сгенерирован на машинах с той же версией среды разработки ESP-IDF. Типичная файловая структура проекта ESP-IDF показана на рис. 4.14.

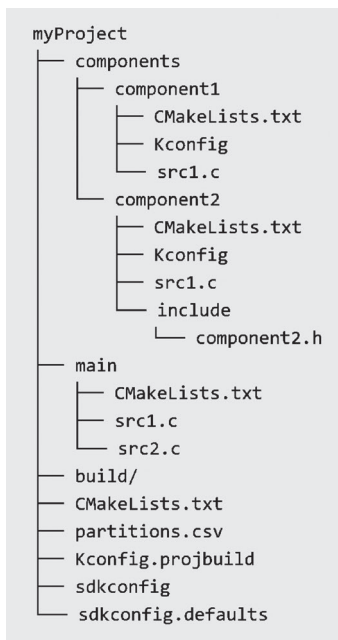


Рисунок 4.14. Типичная файловая структура проекта ESP-IDF

Поскольку ESP-IDF поддерживает несколько чипов IoT от Espressif, включая серии ESP32, ESP32-S, серию ESP32-C, ESP32-H и т. д., перед компиляцией не-

обходимо определить целевой объект. Целевым объектом может являться как аппаратное устройство, на котором выполняется прикладная программа, так и целевая сборка системы компиляции.

В зависимости от ваших потребностей вы можете указать одну или несколько целей для вашего проекта. Например, с помощью команды `idf.py set-target esp32c3` вы можете установить цель компиляции на ESP32-C3, для которой будут загружены параметры по умолчанию и путь к тулчейну компиляции для ESP32-C3. После компиляции можно сгенерировать исполняемую программу для ESP32-C3. Вы также можете запустить команду `set-target` еще раз, чтобы установить другую цель, и система компиляции автоматически очистит и перенастроит параметры.

Компоненты

Компоненты в ESP-IDF представляют собой модульные и независимые блоки кода, управляемые в рамках системы компиляции. Они организованы в виде папок, причем имя папки по умолчанию соответствует имени компонента. Каждый компонент имеет свой собственный сценарий компиляции, который определяет его параметры компиляции и зависимости. В процессе компиляции компоненты компилируются в отдельные статические библиотеки (файлы `.a`) и в итоге объединяются с другими компонентами для формирования прикладной программы.

ESP-IDF предоставляет основные функции, такие как операционная система, драйверы периферийных устройств и стек сетевых протоколов, в виде компонентов. Эти компоненты хранятся в каталоге `components`, расположенном в корневом каталоге ESP-IDF. Разработчикам не нужно копировать эти компоненты в каталог компонентов `MyProject`. Вместо этого им нужно только указать отношения зависимостей этих компонентов в файле проекта `CMakeLists.txt` с использованием директив `REQUIRES` или `PRIV_REQUIRES`. Система компиляции автоматически найдет и скомпилирует необходимые компоненты.

Следовательно, каталог `components` в каталоге проекта `myProject` не нужен. Он используется только для включения некоторых пользовательских компонентов проекта, которые могут быть сторонними библиотеками или пользовательским кодом. Кроме того, компоненты могут быть получены из любого каталога, отличного от ESP-IDF или текущего проекта, например из проекта с открытым исходным кодом, сохраненного в другом каталоге. В этом случае вам нужно только добавить путь к компоненту, установив переменную `EXTRA_COMPONENT_DIRS` в `CMakeLists.txt` в корневом каталоге. Этот каталог переопределяет любой компонент ESP-IDF с таким же именем, гарантируя использование правильного компонента.

Вход в программу `main`

Каталог `main` внутри проекта имеет ту же файловую структуру, что и другие компоненты (например, `component1`). Однако он имеет особое значение, поскольку является обязательным компонентом, который должен присутствовать в каждом проекте. Основной каталог содержит исходный код проекта и точку входа пользовательской программы, обычно называемую `app_main`. По умолчанию выполнение пользовательской программы начинается с этой точки входа. Компонент `main` также отличается тем, что он автоматически зависит от всех компонентов в пределах пути поиска. Следовательно, нет

необходимости явно указывать зависимости с помощью директив `REQUIRES` или `PRIV_REQUIRES` в файле `CMakeLists.txt`.

Конфигурационный файл

Корневой каталог проекта содержит конфигурационный файл под названием `sdkconfig`, который содержит параметры конфигурации для всех компонентов в рамках проекта. Файл `sdkconfig` автоматически генерируется системой компиляции и может быть изменен и восстановлен повторно с помощью команды `idf.py menuconfig`. Параметры `menuconfig` в основном исходят из файла проекта `Kconfig.projbuild` и `Kconfig` компонентов. Разработчики компонентов обычно добавляют элементы конфигурации в `Kconfig`, чтобы сделать компонент гибким и настраиваемым.

Каталог сборки build

По умолчанию в каталоге сборки проекта хранятся промежуточные файлы и результирующие исполняемые программы, сгенерированные командой `idf.py build`. В общем случае нет необходимости напрямую обращаться к содержимому каталога сборки. ESP-IDF предоставляет predefined команды для взаимодействия с каталогом, такие как команда `idf.py flash` для автоматического определения местоположения скомпилированного двоичного файла и его перепрошивки по указанному адресу, или с помощью команды `idf.py fullclean` для очистки всего каталога `build`.

Таблица разделов (partitions.csv)

Каждому проекту требуется таблица разделов для распределения флеш-памяти устройства, указания размера и начального адреса исполняемой программы и пространства пользовательских данных. Команда `idf.py flash` или OTA-обновление прошьет встроенное ПО по соответствующему адресу в соответствии с этой таблицей. ESP-IDF предоставляет несколько таблиц разделов по умолчанию в `components/partition_table`, таких как `partitions_single_app.csv` и `partitions_two_ota.csv`, которые можно выбрать в `menuconfig`.

Если таблица разделов системы по умолчанию не соответствует требованиям проекта, пользовательский файл `partitions.csv` может быть добавлен в каталог проекта и выбран в `menuconfig`.

4.3.3. Правила построения системы компиляции по умолчанию

Правила переопределения компонентов с тем же именем

В процессе поиска компонентов система компиляции выполняется в определенном порядке. Сначала она выполняет поиск внутренних компонентов ESP-IDF, затем ищет компоненты пользовательского проекта и, наконец, выполняет поиск компонентов в `EXTRA_COMPONENT_DIRS`. В случаях, когда несколько каталогов содержат компоненты с одинаковым именем, компонент, найденный в последнем каталоге, переопределит все предыдущие компоненты с таким же именем. Это правило позволяет настраивать компоненты ESP-IDF в рамках пользовательского проекта, сохраняя при этом исходный код ESP-IDF нетронутым.

Правила добавления общих компонентов по умолчанию

Как упоминалось в разделе 4.3.2, компоненты должны явно указывать свои зависимости от других компонентов в *CMakeLists.txt*. Однако распространенные компоненты, такие как `freertos`, автоматически включаются в систему сборки по умолчанию, даже если их отношения зависимостей явно не определены в сценарии компиляции. Общие компоненты ESP-IDF включают `freertos`, `Newlib`, `heap`, `log`, `soc`, `esp_gom`, `esp_common`, `xtensa/risc` и `sxx`. Использование этих общих компонентов позволяет избежать повторяющейся работы при написании *CMakeLists.txt* и делает его более лаконичным.

Правила для переопределения элементов конфигурации

Разработчики могут дополнять параметры конфигурации по умолчанию, добавляя в проект файл конфигурации с именем *sdk config.defaults*. Например, добавлением `CONFIG_LOG_DEFAULT_LEVEL_NONE = y` можно настроить интерфейс UART так, чтобы он по умолчанию не выводил данные логов. Кроме того, если необходимо задать конкретные параметры для конкретной цели, используется конфигурационный файл с именем *sdkconfig.defaults.TARGET_NAME*, где *TARGET_NAME* – имя целевого объекта, например `esp32s2`, `esp32c3` и т. д. Эти конфигурационные файлы импортируются в *sdkconfig* во время компиляции, при этом сначала импортируется общий файл конфигурации по умолчанию *sdkconfig.defaults*, за которым следует файл конфигурации для конкретной цели, такой как *sdkconfig.defaults.esp32c3*. В случаях когда имеются элементы конфигурации с одинаковыми именами, последний файл конфигурации будет переопределять первый.

4.3.4. Введение в сценарий компиляции

При разработке проекта с использованием ESP-IDF разработчикам необходимо не только написать исходный код, но и написать *CMakeLists.txt* для проекта и компонентов. *CMakeLists.txt* – это текстовый файл, также известный как сценарий компиляции, который определяет ряд объектов компиляции, элементов конфигурации компиляции и команд для управления процессом компиляции исходного кода. Система компиляции ESP-IDF версии 4.3.2 основана на CMake. В дополнение к поддержке собственных функций и команд CMake он также определяет ряд пользовательских функций, значительно упрощающих написание скриптов компиляции.

Сценарии компиляции в ESP-IDF в основном состоят из сценария компиляции проекта и сценариев компиляции компонентов. *CMakeLists.txt* в корневом каталоге проекта содержит скрипт, управляющий процессом компиляции всего проекта. Базовый сценарий компиляции проекта обычно включает в себя следующие три строки:

```
1. cmake_minimum_required(VERSION 3.5)
2. include($ENV{IDF_PATH}/tools/cmake/project.cmake)
3. project(myProject)
```

Из них строка `cmake_minimum_required (VERSION 3.5)` должна быть помещена в первую строку, т. к. используется для указания минимального номера версии CMake, требуемого проектом. Новые версии CMake, как правило, обратно

совместимы со старыми версиями, поэтому для обеспечения совместимости при использовании новых команд CMake следует соответствующим образом изменить номер версии.

Строка `include(${ENV{IDF_PATH}}/tools/cmake/project.cmake)` импортирует определенные элементы конфигурации и команды системы компиляции ESP-IDF, включая правила сборки системы компиляции по умолчанию, описанные в разделе 4.3.3. Строка `project(MyProject)` создает сам проект и указывает его название. Указанное имя будет использоваться в качестве окончательного имени выходного двоичного файла, например *MyProject.elf* и *MyProject.bin*.

Проект может иметь несколько компонентов, включая главный компонент. Каталог верхнего уровня каждого компонента содержит файл *CMakeLists.txt*, который называется скриптом компиляции компонента. Скрипты компиляции компонентов в основном используются для указания зависимостей, параметров конфигурации, файлов исходного кода и включают файлы заголовков для компиляции. С применением пользовательской функции ESP-IDF `idf_component_register` минимальный необходимый код для скрипта компиляции компонента выглядит следующим образом:

```
1. idf_component_register(SRCS "src1.c"
2.                       INCLUDE_DIRS "include"
3.                       REQUIRES component1)
```

Параметр `SRCS` предоставляет список исходных файлов в компоненте, разделенных пробелами, если файлов несколько. Параметр `INCLUDE_DIRS` предоставляет список общедоступных каталогов заголовочных файлов для компонента, которые будут добавлены в путь поиска `include` для других компонентов, зависящих от текущего компонента. Параметр `REQUIRED` определяет зависимости общедоступного компонента для текущего компонента. Компонентам необходимо явно указать, от каких компонентов они зависят, например `component2` зависит от `component1`. Однако для основного компонента, который по умолчанию зависит от всех компонентов, параметр `REQUIRED` может быть опущен.

Кроме того, в сценарии компиляции также могут быть использованы собственные команды CMake. Например, используйте команду `set` для установки переменных, таких как `set(VARIABLE "VALUE")`.

4.3.5. Введение в общие команды

ESP-IDF в процессе компиляции кода использует CMake (инструмент настройки проекта), Ninja (инструмент построения проекта) и `esptool` (инструмент flash-загрузки). Каждый инструмент играет различную роль в процессе компиляции, сборки и загрузки флеш-памяти, а также поддерживает различные операционные команды. Чтобы облегчить работу пользователя, ESP-IDF добавляет унифицированный внешний интерфейс *idf.py*, что позволяет быстро вызывать указанные команды.

Перед использованием *idf.py* убедитесь, что:

- переменная среды `IDF_PATH` для ESP-IDF добавлена к текущему терминалу;
- каталог выполнения команд – это корневой каталог проекта, который включает в себя скрипт компиляции проекта *CMakeLists.txt*.

Общие команды *idf.py* заключаются в следующем:

- *idf.py --help*: отображение списка команд и инструкций по их использованию;
- *idf.py set-target <target>*: настройка компиляции *taidf.py fullcleanrget*, например при замене *<target>* на *esp32c3*;
- *idf.py menuconfig*: запуск *menuconfig*, графического инструмента настройки терминала, который может выбирать или изменять параметры конфигурации, а результаты настройки сохранять в файле конфигурации *sdkconfig*;
- *idf.py build*: инициирование компиляции кода. Промежуточные файлы и конечный исполняемый файл программы, сгенерированные в результате компиляции, по умолчанию будут сохранены в каталоге *build* проекта. Процесс компиляции является инкрементным, т. е. при изменении только одного исходного файла в следующий раз будет скомпилирован только измененный файл;
- *idf.py clean*: очистка промежуточных файлов, сгенерированных при компиляции проекта. Весь проект будет принудительно скомпилирован при следующей компиляции. Обратите внимание, что конфигурация CMake и изменения конфигурации, внесенные в *menuconfig*, не будут удалены во время очистки;
- *idf.py fullclean*: удаление всего каталога сборки *build*, включая все выходные файлы конфигурации CMake. При повторном создании проекта CMake настроит проект с нуля. Пожалуйста, обратите внимание, что эта команда рекурсивно удалит все файлы в каталоге *build*, поэтому используйте ее с осторожностью, чтобы не удалить файл конфигурации проекта;
- *idf.py flash*: перепрошивка двоичного файла исполняемой программы, сгенерированного при сборке, на целевой ESP32-C3. Опциональные параметры *-p <port_name>* и *-b <baud_gate>* используются для установки имени устройства последовательного порта и скорости передачи данных в бодах для перепрошивки. Если эти два параметра не указаны, последовательный порт будет обнаружен автоматически и будет использоваться скорость передачи данных в бодах по умолчанию;
- *idf.py monitor*: отображение выходных данных последовательного порта целевого ESP32-C3. Опцию *-p* можно использовать для указания имени устройства последовательного порта на стороне хоста. Во время вывода через последовательный порт можно нажать комбинацию клавиш **Ctrl+J**, чтобы выйти из режима монитора.

Приведенные выше команды также могут быть объединены по мере необходимости. Например, команда *idf.py build flash monitor* выполнит компиляцию кода, прошивку и затем откроет монитор последовательного порта.

Вы можете посетить <https://book3.espressif.com/build-system>, чтобы узнать больше о системе компиляции ESP-IDF.

4.4. Практика: компиляция примера программы Blink

4.4.1. Анализ примера

В этом разделе мы рассмотрим программу Blink в качестве примера для детального анализа файловой структуры и правил кодирования реального проекта.

Программа Blink реализует эффект мигания светодиода, а проект находится в каталоге *examples/get-started/blink*, который содержит исходный файл, файлы конфигурации и несколько скриптов компиляции.

Проект Smart light, представленный в этой книге, основан на данном примере. В последующих главах будут постепенно добавляться функции для его окончательного завершения.

Исходный код

Чтобы продемонстрировать весь процесс разработки, программа Blink была скопирована в *esp32c3-iot-projects/device firmware/1_blink*.

Структура каталогов файлов проекта Blink показана на рис. 4.15.

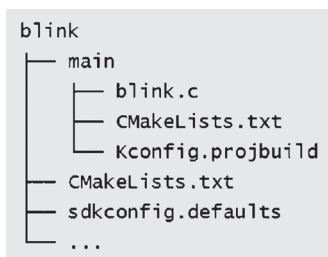


Рисунок 4.15. Структура каталогов файлов проекта Blink

Проект Blink содержит только один основной каталог, являющийся специальным компонентом, который был добавлен, как описано в разделе 4.3.2. Основной каталог в целом используется для хранения реализации функции `app_main()`, являющейся точкой входа в пользовательскую программу. Проект Blink не включает каталог *components*, поскольку в этом примере будут использоваться только компоненты, поставляемые с ESP-IDF, и дополнительные компоненты не потребуются. Включенный в проект файл *CMakeLists.txt* применяется для управления процессом компиляции, файл *Kconfig.projbuild* используется для добавления элементов конфигурации для данного примера программы в *menuconfig*. Другие ненужные файлы не повлияют на компиляцию кода, поэтому они здесь обсуждаться не будут. Ниже приводится подробный перечень добавлений в файлы проекта Blink.

```
1. /*blink.c includes the following header files*/
2. #include <stdio.h> //Standard C library header file
3. #include "freertos/freertos.h" //FreeRTOS main header file
4. #include "freertos/task.h" //FreeRTOS Task header file
5. #include "sdkconfig.h" //Configuration header file generated by kconfig
6. #include "driver/gpio.h" //GPIO driver header file
```

Исходный файл *blink.c* содержит ряд заголовочных файлов, соответствующих объявлениям функций. ESP-IDF обычно следует порядку включения заголовочных файлов стандартной библиотеки: файлы заголовков FreeRTOS, файлы заголовков драйверов, файлы заголовков других компонентов и файлы заголовков проекта. Порядок, в котором включаются заголовочные файлы, может повлиять

на конечный результат компиляции, поэтому старайтесь следовать правилам по умолчанию. Нужно отметить, что *sdkconfig.h* автоматически генерируется *kconfig* и может быть изменен только с помощью команды `idf.py menuconfig`. Прямая модификация этого заголовочного файла будет перезаписана.

```
1. /*Вы можете выбрать GPIO, соответствующий светодиоду, в idf.py menuconfig,
   и результатом модификации menuconfig является то, что значение CONFIG_BLINK_GPIO будет
   изменено. Вы также можете здесь напрямую изменить определение макроса и изменить CONFIG_
   BLINK_GPIO на фиксированное значение.*/
2. #define BLINK_GPIO CONFIG_BLINK_GPIO
3. void app_main(void)
4. {
5.     /*Configure IO as the GPIO default function, enable pull-up mode, and
6.     disable input and output modes*/
7.     gpio_reset_pin(BLINK_GPIO);
8.     /*Set GPIO to output mode*/
9.     gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
10.    while(1) {
11.        /*Print log*/
12.        printf("Turning off the LED\n");
13.        /*Turn off the LED (output low level)*/
14.        gpio_set_level(BLINK_GPIO, 0);
15.        /*Delay (1000 ms)*/
16.        vTaskDelay(1000 / portTICK_PERIOD_MS);
17.        printf("Turning on the LED\n");
18.        /*Turn on the LED (output high level)*/
19.        gpio_set_level(BLINK_GPIO, 1);
20.        vTaskDelay(1000 / portTICK_PERIOD_MS);
21.    }
22. }
```

Функция `app_main()` в примере программы `Blink` служит точкой входа для пользовательских программ. Это простая функция без параметров и возвращаемого значения. Она вызывается после завершения инициализации системы, которая включает в себя такие задачи, как инициализация последовательного порта для вывода логов, настройка одноядерного/двухъядерного процессора и настройка сторожевого таймера.

Функция `app_main()` выполняется в контексте задачи с именем `main`. Размер стека и приоритет этой задачи можно настроить в меню конфигурации `menuconfig` → `Component config` (Конфигурация компонента) → `Common ESP-related` (Общее относительно ESP).

Для простых задач, таких как мигание светодиода, весь необходимый код может быть реализован непосредственно в функции `app_main()`. Обычно это включает в себя инициализацию GPIO, соответствующего светодиоду, и использование цикла `while(1)` для включения и выключения светодиода. В качестве альтернативы вы можете использовать FreeRTOS API для создания новой задачи, которая обрабатывает мигание светодиода. Как только новая задача будет успешно создана, вы можете выйти из функции `app_main()`.

Содержание файла `main/CMakeLists.txt`, управляющего процессом компиляции основного компонента, выглядит следующим образом:

```
1. idf_component_register(SRCS "blink.c" INCLUDE_DIRS ".")
```

В соответствии с этой строкой *main/CMakeLists.txt* вызывает только одну системную функцию компиляции `idf_component_register`. Подобно *CMakeLists.txt* для большинства других компонентов, *blink.c* добавляется в список `SRCS`, и все исходные файлы, добавленные в `SRCS`, будут скомпилированы.

В то же время ".", обозначающий путь, по которому находится файл *CMakeLists.txt*, следует добавить в `INCLUDE_DIRS` в качестве каталогов поиска файлов заголовков. Содержимое *CMakeLists.txt* тогда выглядит следующим образом:

```
1. # Указываем v3.5 как самую старую версию CMake, поддерживаемую текущим проектом
2. # Версии ниже v3.5 необходимо обновить перед продолжением компиляции
3. cmake_minimum_required(VERSION 3.5)
4. # Включаем конфигурацию системы компиляции ESP-IDF CMake по умолчанию
5. include($ENV{IDF_PATH}/tools/cmake/project.cmake)
6. # Создание проекта под названием «blink»
7. project(blink )
```

CMakeLists.txt содержит главным образом ссылку на файл `$ENV{IDF_PATH}/tools/cmake/project.cmake`, который является основным файлом конфигурации CMake, предоставляемым ESP-IDF. Он используется для настройки правил по умолчанию системы компиляции ESP-IDF и определения общих функций, таких как `idf_component_register`; `project(blink)`, а также создает проект под названием `blink`, и окончательная прошивка будет называться *blink.bin*.

4.4.2. Компиляция программы Blink

В этом разделе в качестве примера рассматривается программа Blink, демонстрирующая процесс компиляции простой программы ESP-IDF. Важно отметить, что в этом разделе для управления светодиодом используется просто высокий/низкий уровень на выходе GPIO. Однако для работы, например, адресного светильника WS2812 потребуется специальный протокол связи. Вы можете ознакомиться с примером программы в *esp-idf/examples/peripherals/rmt/led_strip* для получения дополнительной информации.

1. Откройте новый терминал и импортируйте переменные окружения ESP-IDF

В системах Linux и Mac используйте команду `cd ~/esp/esp-idf` для перехода к папке ESP-IDF. Затем импортируйте переменные окружения ESP-IDF с помощью команды `./export.sh`. Этот процесс также выполняет полную проверку целостности среды разработки.

Совет

Пожалуйста, обратите внимание, что в команде `./export.sh` точка перед пробелом не должна быть опущена. Точка эквивалентна директиве `source`, которая относится к выполнению скрипта и изменению переменных окружения в текущей оболочке.

```
Administrator: ESP-IDF 4.3 CMD
Python 3.8.7
Using Git in D:\.espressif\tools\idf-git\2.34.2\cmd\
git version 2.34.1.windows.1
Setting IDF_PATH: D:\.espressif\frameworks\esp-idf-v4.3.2

Adding ESP-IDF tools to PATH...
Not using an unsupported version of tool cmake found in PATH: 3.23.2.
D:\.espressif\tools\xtensa-esp32-elf\esp-2021r2-8.4.0\xtensa-esp32-elf\bin
D:\.espressif\tools\xtensa-esp32s2-elf\esp-2021r2-8.4.0\xtensa-esp32s2-elf\bin
D:\.espressif\tools\xtensa-esp32s3-elf\esp-2021r2-8.4.0\xtensa-esp32s3-elf\bin
D:\.espressif\tools\riscv32-esp-elf\esp-2021r2-8.4.0\riscv32-esp-elf\bin
D:\.espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf\binutils\bin
D:\.espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf\binutils\bin
D:\.espressif\tools\cmake\3.16.4\bin
D:\.espressif\tools\openocd-esp32\v0.10.0-esp32-20211111\openocd-esp32\bin
D:\.espressif\tools\idf-exe\1.0.1\
D:\.espressif\tools\ccache\3.7\
D:\.espressif\tools\dfu-util\0.9\dfu-util-0.9-win64
D:\.espressif\frameworks\esp-idf-v4.3.2\tools

Checking if Python packages are up to date...
Python requirements from D:\.espressif\frameworks\esp-idf-v4.3.2\requirements.txt are satisfied.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

idf.py build

D:\.espressif\frameworks\esp-idf-v4.3.2>
```

Рисунок 4.16. Автоматическое добавление переменных среды в системе Windows

Для систем Windows вы можете напрямую найти и открыть ESP-IDF 4.3 CMD или ESP-IDF 4.3. PowerShell в списке программ. После открытия терминала переменные окружения будут добавляться автоматически, как показано на рис. 4.16.

2. Перейдите в корневой каталог проекта blink

Перед компиляцией проекта перейдите в корневой каталог проекта. Для этого воспользуйтесь командой `cd examples/get-started/blink`.

3. Установите целевое устройство компиляции на ESP32-C3

Используйте команду `idf.py set-target esp32c3` для установки целевого устройства на ESP32-C3, как показано на рис. 4.17. Если этот шаг пропущен, целью компиляции по умолчанию будет ESP32.

4. Настройте GPIO

Используйте команду `idf.py menuconfig` для входа в интерфейс конфигурации. Перемещаясь с помощью клавиш вверх/вниз, нажмите клавишу **Enter**, чтобы ввести пример конфигурации. Введите номер, чтобы изменить GPIO на указанный вывод, как показано на рис. 4.18. Сохраните конфигурацию, следуя инструкциям.

5. Создайте код

Используйте команду `idf.py build` для сборки кода. Процесс построения кода показан на рис. 4.19. Соответствующие подсказки и команды прошивки flash будут выведены после завершения сборки, как показано на рис. 4.20.

```

Administrator: ESP-IDF 4.3 CN
components/esp_wifi D:/.../esp-idf-v4.3.2/components/espcoredump D:/.../esp-idf-v4.3.2/components/esptool_py D:/.../esp-idf-v4.3.2/components/expand D:/.../esp-idf-v4.3.2/components/fatfs D:/.../esp-idf-v4.3.2/components/freemodbus D:/.../esp-idf-v4.3.2/components/freertos D:/.../esp-idf-v4.3.2/components/hal D:/.../esp-idf-v4.3.2/components/heap D:/.../esp-idf-v4.3.2/components/idf_test D:/.../esp-idf-v4.3.2/components/jsmn D:/.../esp-idf-v4.3.2/components/json D:/.../esp-idf-v4.3.2/components/libsodium D:/.../esp-idf-v4.3.2/components/Log D:/.../esp-idf-v4.3.2/components/lwip D:/.../esp-idf-v4.3.2/examples/get-started/blink/main D:/.../esp-idf-v4.3.2/components/mbdttl D:/.../esp-idf-v4.3.2/components/mdns D:/.../esp-idf-v4.3.2/components/mqtt D:/.../esp-idf-v4.3.2/components/newlib D:/.../esp-idf-v4.3.2/components/nghttp D:/.../esp-idf-v4.3.2/components/nvs_flash D:/.../esp-idf-v4.3.2/components/openssl D:/.../esp-idf-v4.3.2/components/partition_table D:/.../esp-idf-v4.3.2/components/protobuf-c D:/.../esp-idf-v4.3.2/components/protocomm D:/.../esp-idf-v4.3.2/components/pthread D:/.../esp-idf-v4.3.2/components/riscv D:/.../esp-idf-v4.3.2/components/sdmmc D:/.../esp-idf-v4.3.2/components/soc D:/.../esp-idf-v4.3.2/components/spi_flash D:/.../esp-idf-v4.3.2/components/spiffs D:/.../esp-idf-v4.3.2/components/tcp_transport D:/.../esp-idf-v4.3.2/components/tcpip_adapter D:/.../esp-idf-v4.3.2/components/tinyusb D:/.../esp-idf-v4.3.2/components/unity D:/.../esp-idf-v4.3.2/components/vfs D:/.../esp-idf-v4.3.2/components/wear_levelling D:/.../esp-idf-v4.3.2/components/wifi_provisioning D:/.../esp-idf-v4.3.2/components/wpa_supplicant
-- Configuring done
-- Generating done
-- Build files have been written to: D:/.../esp-idf-v4.3.2/examples/get-started/blink/build
D:\.../esp-idf-v4.3.2/examples/get-started/blink>

```

Рисунок 4.17. Установите целевое устройство компиляции на ESP32-C3

```

(Top) → Example Configuration
      Espressif IoT Development Framework Configuration
(5) Blink GPIO number

```

Blink GPIO number (int)

Range: 0-34

```

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                  [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

```

Рисунок 4.18. Настройте GPIO с помощью *menuconfig*


```

Administrator: ESP-IDF 4.3 C...
v4.3.2/components/openssl D:/.../frameworks/esp-idf-v4.3.2/components/partition_table D:/.../frameworks/esp-idf-v4.3.2/components/protobuf-c D:/.../frameworks/esp-idf-v4.3.2/components/protocomm D:/.../frameworks/esp-idf-v4.3.2/components/ptthread D:/.../frameworks/esp-idf-v4.3.2/components/riscv D:/.../frameworks/esp-idf-v4.3.2/components/sdmmc D:/.../frameworks/esp-idf-v4.3.2/components/soc D:/.../frameworks/esp-idf-v4.3.2/components/spi_flash D:/.../frameworks/esp-idf-v4.3.2/components/spiffs D:/.../frameworks/esp-idf-v4.3.2/components/tcp_transport D:/.../frameworks/esp-idf-v4.3.2/components/tcpip_adapter D:/.../frameworks/esp-idf-v4.3.2/components/tinyusb D:/.../frameworks/esp-idf-v4.3.2/components/unity D:/.../frameworks/esp-idf-v4.3.2/components/vfs D:/.../frameworks/esp-idf-v4.3.2/components/wear_levelling D:/.../frameworks/esp-idf-v4.3.2/components/wifi_provisioning D:/.../frameworks/esp-idf-v4.3.2/components/wpa_supplicant
-- Configuring done
-- Generating done
-- Build files have been written to: D:/.../frameworks/esp-idf-v4.3.2/examples/get-started/blink/build

D:\.../frameworks/esp-idf-v4.3.2/examples/get-started/blink>idf.py build
Executing action: all (aliases: build)
Running ninja in directory d:\.../frameworks/esp-idf-v4.3.2/examples/get-started/blink/build
Executing "ninja all"...
[7/940] Generating ../../partition_table/partition-table.bin
Partition table binary generated. Contents:
*****
# ESP-IDF Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,data,nvs,0x9000,24K,
phy_init,data,phy,0xf000,4K,
factory,app,factory,0x10000,1M,
*****
[C291/940] Building C object esp-idf/spi_flash/CM..._idf_spi_flash.dir/spi_flash_chip_winbond.c.obj

```

Рисунок 4.19. Процесс компиляции кода

```

Administrator: ESP-IDF 4.3 C...
[70/82] Building C object esp-idf/log/CMakeFiles/_idf_log.dir/log.c.obj
[71/82] Linking C static library esp-idf/log/liblog.a
[72/82] Linking C static library esp-idf/esp_rom/libesp_rom.a
[73/82] Linking C static library esp-idf/esp_hw_support/libesp_hw_support.a
[74/82] Linking C static library esp-idf/efuse/libefuse.a
[75/82] Linking C static library esp-idf/bootloader_support/libbootloader_support.a
[76/82] Linking C static library esp-idf/spi_flash/libspi_flash.a
[77/82] Linking C static library esp-idf/micro-ecc/libmicro-ecc.a
[78/82] Linking C static library esp-idf/soc/libsoc.a
[79/82] Linking C static library esp-idf/hal/libhal.a
[80/82] Linking C static library esp-idf/main/libmain.a
[81/82] Linking C executable bootloader.elf
[82/82] Generating binary image from built executable
esptool.py v3.2-dev
Merged 1 ELF section
Generated D:/.../frameworks/esp-idf-v4.3.2/examples/get-started/blink/build/bootloader/bootloader.bin
[940/940] Generating binary image from built executable
esptool.py v3.2-dev
Merged 1 ELF section
Generated D:/.../frameworks/esp-idf-v4.3.2/examples/get-started/blink/build/blink.bin

Project build complete. To flash, run this command:
D:\.../python_env/idf4.3_py3.8_env/Scripts/python.exe ../../../../components/esptool_py/esptool/esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip esp32c3 write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x0 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin 0x10000 build/blink.bin
or run 'idf.py -p (PORT) flash'

D:\.../frameworks/esp-idf-v4.3.2/examples/get-started/blink>

```

Рисунок 4.20. Подсказка после завершения компиляции кода

4.4.3. Прошивка программы Blink

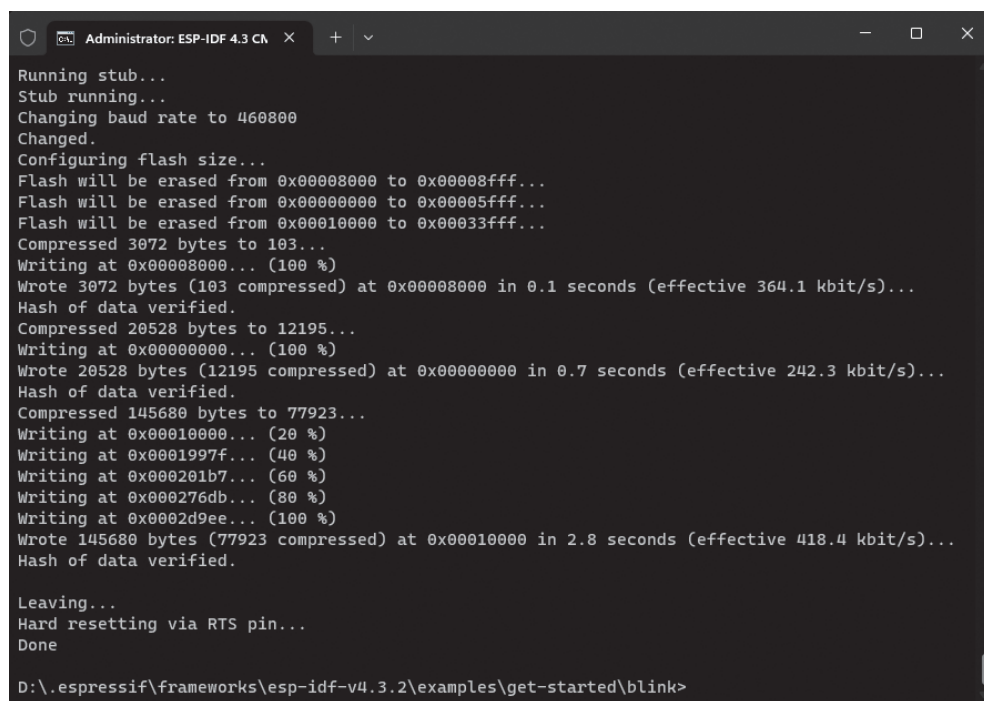
Для системы Linux подключите ESP32-C3 к компьютеру через микросхему USB-UART (например, CP2102) и используйте команду `ls /dev/ttyUSB*` для просмотра номера последовательного порта. Если текущий отображаемый номер последовательного порта, например, равен `/dev/ttyUSB0`, используйте команду `idf.py -p /dev/ttyUSB0 flash` для прошивки программы на ESP32-C3.

Для систем Mac подключите ESP32-C3 к компьютеру через микросхему USB-UART (например, CP2102) и используйте команду `ls /dev/cu.*` для просмотра номера последовательного порта. Если отобразится текущий номер последовательного порта `/dev/cu.SLAB_USBtoUART`, используйте команду `idf.py -p /dev/cu.SLAB_USBtoUART flash` для прошивки программы на ESP32-C3.

Для систем Windows подключите ESP32-C3 к компьютеру через USB-UART-чип (например, CP2102) и просмотрите номер последовательного порта через Диспетчер устройств. Если текущий номер последовательного порта равен COM5, используйте команду `idf.py -p COM 5 flash` для прошивки программы на ESP32-C3.

После завершения процесса прошивки вы увидите в консоли приглашение, как показано на рис. 4.21. Когда появится приведенная ниже запись, код начнет выполняться, и светодиод на отладочной плате начнет мигать.

```
Hard resetting via RTS pin...
Done
```



```
Administrator: ESP-IDF 4.3 CN
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00008000 to 0x00003fff...
Flash will be erased from 0x00000000 to 0x00005fff...
Flash will be erased from 0x00010000 to 0x00033fff...
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 364.1 kbit/s)...
Hash of data verified.
Compressed 20528 bytes to 12195...
Writing at 0x00000000... (100 %)
Wrote 20528 bytes (12195 compressed) at 0x00000000 in 0.7 seconds (effective 242.3 kbit/s)...
Hash of data verified.
Compressed 145680 bytes to 77923...
Writing at 0x00010000... (20 %)
Writing at 0x0001997f... (40 %)
Writing at 0x000201b7... (60 %)
Writing at 0x000276db... (80 %)
Writing at 0x0002d9ee... (100 %)
Wrote 145680 bytes (77923 compressed) at 0x00010000 in 2.8 seconds (effective 418.4 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
Done
D:\.espressif\frameworks\esp-idf-v4.3.2\examples\get-started\blink>
```

Рисунок 4.21. Вывод в консоли после завершения прошивки

4.4.4. Анализ логов последовательного порта программы Blink

Как только компиляция и загрузка встроенного ПО будут завершены, перейдите в папку проекта и запустите команду `idf.py monitor`. Это откроет монитор с цветным шрифтом. Монитор выведет лог последовательного порта целевого ESP32-C3. По умолчанию содержимое разделено на три части: **информация о загрузчике первого уровня, информация о загрузчике второго уровня и выходные данные пользовательской программы**. Во время вывода журнала вы можете нажать комбинацию клавиш **Ctrl+J** для выхода из режима вывода логов.

```
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fcd6100,len:0x1798
load:0x403ce000,len:0x8dc
load:0x403d0000,len:0x2984
entry 0x403ce000
```

Информация о загрузчике первого уровня

По умолчанию информация о загрузчике первого уровня выводится из UART и не может быть отключена посредством настройки в ESP-IDF версии 4.3.2. Эта информация включает информацию о версии кода ROM, зафиксированную внутри чипа. Разные микросхемы одной и той же серии могут иметь разные версии кода ПЗУ из-за апгрейдинга ПЗУ и расширения функциональных возможностей. В информации также указывается причина перезапуска микросхемы: например, `rst:0x1` указывает на перезапуск микросхемы при включении питания, `rst:0x3` указывает на перезапуск, вызванный программным обеспечением, `rst:0x4` указывает на перезапуск из-за исключительной ситуации программного обеспечения и т. д. Вы можете использовать эту информацию для оценки состояния чипа. Кроме того, предоставляется подробная информация о режиме загрузки чипа, например `boot:0xc` указывает режим загрузки SPI Flash (обычный режим работы, в котором загружается и выполняется код во flash), а `boot:0x4` указывает режим загрузки Flash, в котором ее содержимое может быть стерто и запрограммировано.

Информация о загрузчике второго уровня

Вывод информации о загрузчике второго уровня можно отключить, установив `menuconfig(Top) → Bootloader config (Конфигурация загрузчика) → Bootloader log verbosity (Подробность лога загрузчика) в No output (Не выводить)`. Эта информация в основном включает версию ESP-IDF, режим работы и скорость флеш-памяти, системный раздел и распределение стека, а также название и версию приложения.

```

I (30) boot: ESP-IDF v4.3.2-1-g887e7c0c73-dirty 2nd stage bootloader
I (30) boot: compile time 18:27:35
I (30) boot: chip revision: 3
I (34) boot.esp32c3: SPI Speed : 80MHz
I (38) boot.esp32c3: SPI Mode : DIO
I (43) boot.esp32c3: SPI Flash Size : 2MB
I (48) boot: Enabling RNG early entropy source...
I (53) boot: Partition Table:
I (57) boot: ## Label Usage Type ST Offset Length
I (64) boot: 0 nvs WiFi data 01 02 00009000 00006000
I (72) boot: 1 phy_init RF data 01 01 0000f000 00001000
I (79) boot: 2 factory factory app 00 00 00010000 00100000
I (86) boot: End of partition table
I (91) esp_image: segment 0: paddr=00010020 vaddr=3c020020 size =06058h (24664) map
I (103) esp_image: segment 1: paddr=00016080 vaddr=3fc89c00 size=01a88h (6792) load
I (109) esp_image: segment 2: paddr=00017b10 vaddr=40380000 size=08508h (34056) load
I (122) esp_image: segment 3: paddr=00020020 vaddr=42000020 size=15c54h (89172) map
I (138) esp_image: segment 4: paddr=00035c7c vaddr=40388508 size=0157ch (5500) load
I (139) esp_image: segment 5: paddr=00037200 vaddr=50000000 size=00010h (16) load
I (147) boot: Loaded app from partition at offset 0x10000
I (150) boot: Disabling RNG early entropy source...
I (166) cpu_start: Pro cpu up.
I (179) cpu_start: Pro cpu start user code
I (179) cpu_start: cpu freq: 160000000
I (179) cpu_start: Application information:
I (182) cpu_start: Project name: blink
I (186) cpu_start: App version: v4.3.2-1-g887e7c0c73-dirty
I (193) cpu_start: Compile time: Jan 26 2022 18:27:31
I (199) cpu_start: ELF file SHA256: dadcae8e7bb964ab...
I (205) cpu_start: ESP-IDF: v4.3.2-1-g887e7c0c73-dirty
I (212) heap_init: Initializing. RAM available for dynamic allocation:
I (219) heap_init: At 3FC8C4D0 len 00033B30 (206 KiB): DRAM
I (225) heap_init: At 3FCC0000 len 0001F060 (124 KiB): STACK/DRAM
I (232) heap_init: At 50000010 len 00001FF0 (7 KiB): RTCRAM
I (238) spi_flash: detected chip: generic
I (243) spi_flash: flash io: dio
W (247) spi_flash: Detected size(4096k) larger than the size in the binary image
header(2048k). Using the size in the binary image header.
I (260) sleep: Configure to isolate all GPIO pins in sleep state
I (267) sleep: Enable automatic switching of GPIO sleep configuration
I (274) cpu_start: Starting scheduler.

```

Вывод из пользовательской программы

Вывод из программы пользователя включает всю информацию, которая выводится с помощью стандартной функции вывода на языке C `printf()`, или функции `ESP_LOG()` – пользовательской функции вывода, предоставляемой ESP-IDF. Рекомендуется применять `ESP_LOG()`, потому что она позволяет указать уровень лога для лучшей организации и фильтрации логов.

Вы можете настроить, какие логи выше определенного уровня выводятся через `menuconfig(Top) → Component config (Конфигурация загрузчика) → Log output`

(Вывод лога). Это позволяет вам контролировать детализацию логов и настраивать уровень детализации, отображаемый во время выполнения.

```
I (278) gpio: GPIO[5]| InputEn: 0| OutfgputEn: 0| OpenDrain: 0| Pullup: 1|
Pulldown: 0| Intr:0
Turning off the LED
Turning on the LED
Turning off the LED
Turning on the LED
```

Помимо вывода логов, `idf.py monitor` также может анализировать системные исключения и отслеживать программные ошибки. Например, при сбое приложения будут сгенерированы следующий дамп регистра и информация о трассировке:

```
Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was
unhandled.
Register dump:
PC : 0x400f360d PS : 0x00060330 A0 : 0x800dbf56 A1 : 0x3ffb7e00
A2 : 0x3ffb136c A3 : 0x00000005 A4 : 0x00000000 A5 : 0x00000000
A6 : 0x00000000 A7 : 0x00000080 A8 : 0x00000000 A9 : 0x3ffb7dd0
A10 : 0x00000003 A11 : 0x00060f23 A12 : 0x00060f20 A13 : 0x3ffba6d0
A14 : 0x00000047 A15 : 0x0000000f SAR : 0x00000019 EXCCAUSE : 0x0000001d
EXCVADDR: 0x00000000 LBEG : 0x4000c46c LEND : 0x4000c477 LCOUNT : 0x00000000
Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40
0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90
```

Основываясь на адресе регистра, IDF монитор будет запрашивать скомпилированный файл ELF и отслеживать процесс вызова кода при сбое приложения, выведя информацию о вызове функции:

```
Guru Meditation Error of type StoreProhibited occurred on core 0. Exception was
unhandled.
Register dump:
PC : 0x400f360d PS : 0x00060330 A0 : 0x800dbf56 A1 : 0x3ffb7e00
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/
hello_world/main/./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
main/./hello_world_main.c:52
A2 : 0x3ffb136c A3 : 0x00000005 A4 : 0x00000000 A5 : 0x00000000
A6 : 0x00000000 A7 : 0x00000080 A8 : 0x00000000 A9 : 0x3ffb7dd0
A10 : 0x00000003 A11 : 0x00060f23 A12 : 0x00060f20 A13 : 0x3ffba6d0
A14 : 0x00000047 A15 : 0x0000000f SAR : 0x00000019 EXCCAUSE : 0x0000001d
EXCVADDR: 0x00000000 LBEG : 0x4000c46c LEND : 0x4000c477 LCOUNT : 0x00000000
Backtrace: 0x400f360d:0x3ffb7e00 0x400dbf56:0x3ffb7e20 0x400dbf5e:0x3ffb7e40
0x400dbf82:0x3ffb7e60 0x400d071d:0x3ffb7e90
0x400f360d: do_something_to_crash at /home/gus/esp/32/idf/examples/get-started/
hello_world/main/./hello_world_main.c:57
(inlined by) inner_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
main/./hello_world_main.c:52
0x400dbf56: still_dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/
main/./hello_world_main.c:47
```

```
0x400dbf5e: dont_crash at /home/gus/esp/32/idf/examples/get-started/hello_world/main./hello_world_main.c:42
0x400dbf82: app_main at /home/gus/esp/32/idf/examples/get-started/hello_world/main./hello_world_main.c:33
0x400d071d: main_task at /home/gus/esp/32/idf/components/esp32./cpu_start.c:254
```

Информация трассировки показывает, что приложение завершает работу с ошибкой в функции `do_something_to_crash()`, которая вызывается функцией `app_main()` → `dont_crash()` → `still_dont_crash()` → `inner_dont_crash()` → `do_something_to_crash()`.

Исходя из этого, входные/выходные параметры каждой функции могут быть проверены для установки причины аварии.

4.5. Резюме

В этой главе мы рассмотрели настройку официальной среды разработки программного обеспечения ESP-IDF для ESP32-C3. Мы представили ресурсы кода и файловую структуру ESP-IDF, продемонстрировали структуру проекта ESP-IDF, систему компиляции и связанные с ней инструменты разработки на простом примере.

Следуя инструкциям в данной главе, вы можете начать разработку с помощью ESP-IDF для простых проектов. Для получения более конкретных или расширенных требований к компиляции рекомендуется обратиться как к официальной документации ESP-IDF, так и к официальной документации CMake.

Часть II

Разработка оборудования и драйверов

Из чего состоят продукты IoT? Что они могут сделать?

Как спроектировать минимальную аппаратную систему для приложений ESP32-C3? Каковы сценарии применения периферийных интерфейсов ESP32-C3?

Что нужно для разработки драйверов?

Ответы мы узнаем в этой части вместе с нашим старым другом – проектом Smart Light.

ГЛАВА 5

Аппаратный дизайн продуктов Smart Light на базе ESP32-C3

ГЛАВА 6

Разработка драйверов



Аппаратный дизайн продуктов Smart Light на базе ESP32-C3

В этой главе мы сначала познакомим вас с основными компонентами продуктов Smart Light и сценариями их применения, а также возьмем светодиодные умные светильники в качестве примера, чтобы продемонстрировать их основные аппаратные блоки. Затем будем использовать чипы и модули ESP32-C3 для разработки умного светильника, способного регулировать яркость, изменять цвет и управляться по беспроводной связи. Дизайн, представленный здесь также может быть расширен и применен к различным светодиодным изделиям, таким как световые ленты, потолочные светильники, точечные светильники и т. д.

5.1. Характеристики и состав продуктов Smart Light

В устройствах Smart Light в качестве источников света обычно используются светодиоды. Светодиоды – это твердотельные источники света и полупроводниковые световые приборы, характеризующиеся низким энергопотреблением и длительным сроком службы, они просты в управлении и не загрязняют окружающую среду. По сравнению с традиционными осветительными приборами они обладают более высокой эффективностью преобразования световой энергии. Одновременно продукты Smart Light обладают функциями беспроводной связи, поддерживая подключение к беспроводным маршрутизаторам или интеллектуальным шлюзам через Wi-Fi, Bluetooth LE или ZigBee, а следовательно, могут подключаться к интернету или облачным серверам. Вы можете использовать смартфоны, планшеты, умные колонки, поддерживающие голосовое управление, и интеллектуальные панели управления для регулировки яркости и цвета, а также установки таймеров включения/выключения освещения. Вы также можете сгруппировать несколько источников света вместе и управлять их яркостью и цветом в пакетном режиме. Вы можете предварительно настроить освещение помещений для различных случаев: например, «театральный режим» для приглушения окружающего освещения, «режим чтения» для мягкой и приятной для глаз яркости, «музыкальный режим» для изменения цвета и мигания в такт ритмам музыки или «спящий режим» для выключения всех источников света, кроме ночника. Структура системы умного освещения показана на рис. 5.1.

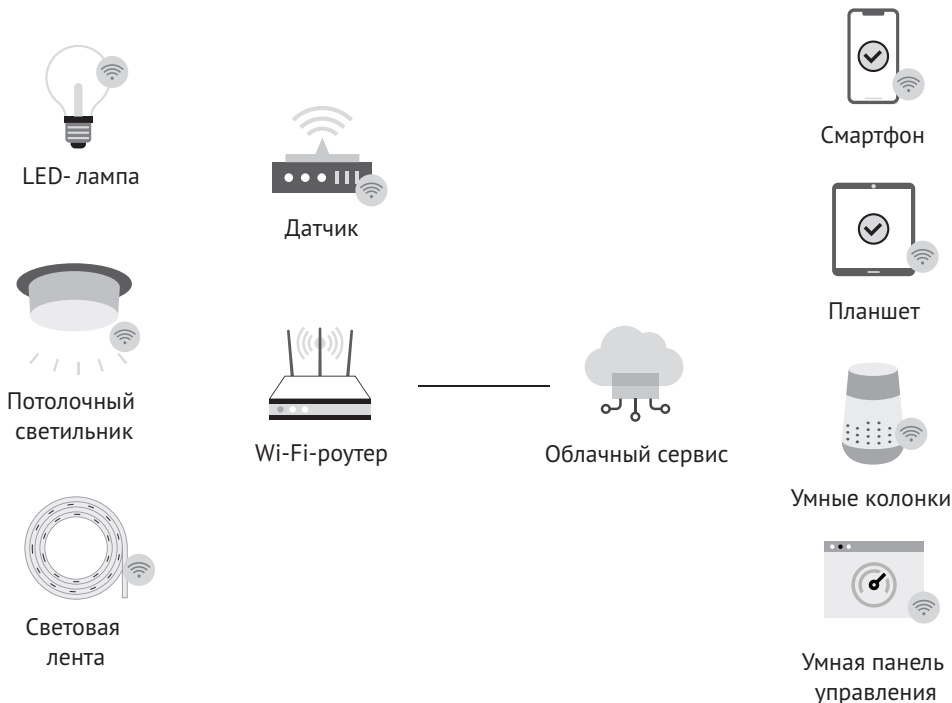


Рисунок 5.1. Структура системы Smart Light

Из приведенного описания мы можем видеть, что основными функциями продуктов Smart Light является управление с помощью беспроводного соединения. Мы возьмем смарт-светодиод с изменением цвета в качестве примера, объясняющего устройство основных компонентов продуктов Smart Light и способы управления ими.

На рис. 5.2 показана конструкция интеллектуальной светодиодной лампы со стандартным патроном E27, алюминиевым корпусом лампы в пластиковой оболочке, блоком питания и платой светодиодного драйвера, модулем Wi-Fi, индивидуальными светодиодами на алюминиевой подложке, а также высокопрозрачным рассеивателем. По сравнению с традиционными светодиодными лампочками, умная светодиодная лампа дополнительно оснащена модулем Wi-Fi. Итак, как этот модуль Wi-Fi помогает управлять освещением по беспроводной сети? В следующих разделах будет более подробно рассказано о реализации данной функции.

На рис. 5.3 показана функциональная блок-схема умной светодиодной лампы, которая в основном включает в себя модуль питания переменного/постоянного тока напряжением 220 В, светодиодный драйвер постоянного тока, дополнительный источник питания с выходным напряжением 3,3 В, модули ШИМ-управления и беспроводной связи, а также отдельные светодиоды различных цветов.

Прежде чем разбирать подробно каждую функцию, давайте сначала взглянем на то, как удастся изменять яркость и цвет освещения в целом. Ключ кроется в отдельных светодиодах лампы.

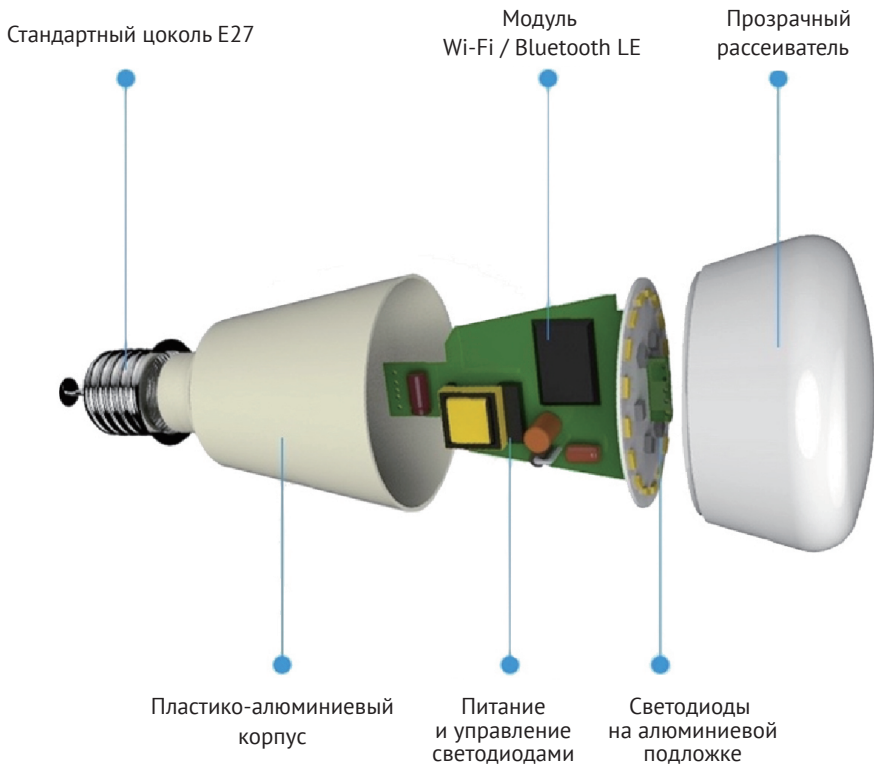


Рисунок 5.2. Устройство умной светодиодной лампы

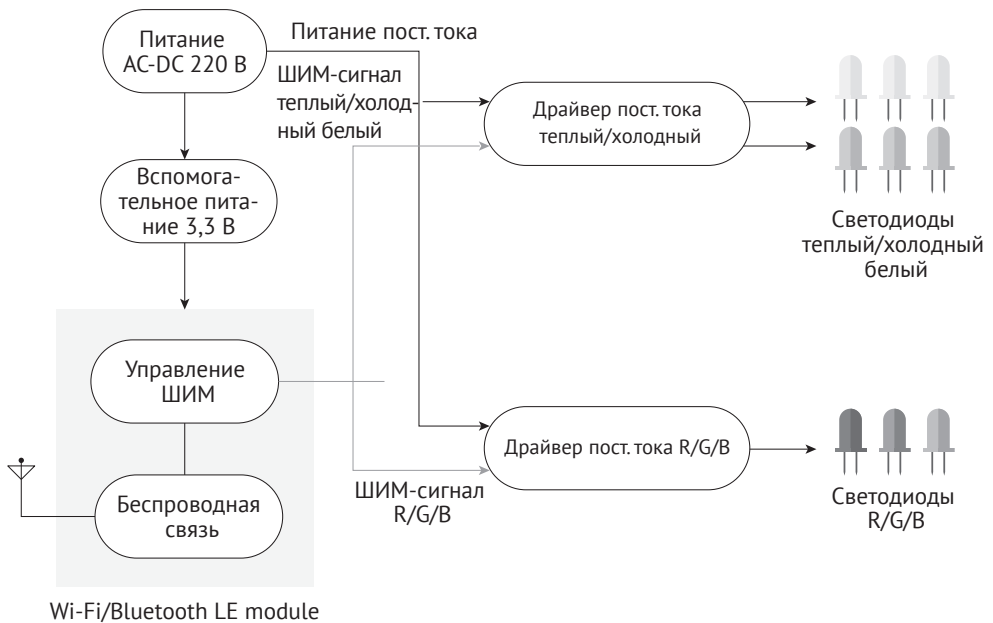


Рисунок 5.3. Блок-схема функциональных блоков умной светодиодной лампы

Регулировать яркость светодиодов в лампе можно двумя способами: аналоговым и цифровым. Аналоговая регулировка изменяет светоотдачу светодиода простым изменением постоянного тока в цепи; тогда как цифровая регулировка, также известная как широтно-импульсная модуляция (ШИМ), достигается путем изменения относительного времени прохождения прямого тока путем включения/выключения светодиодов с помощью ШИМ-сигналов с различной длительностью импульсов. ШИМ-регулировка будет подробно описана в разделе 6.3.3. Здесь мы только кратко представим ШИМ-регулировку с использованием ШИМ-сигналов.

При применении управляемого источника постоянного тока для управления светодиодами с целью регулировки цветовой температуры вы можете изменять рабочие циклы ШИМ-сигналов по двум каналам, чтобы регулировать ток светодиодов теплого белого (WW) и холодного белого (CW) цветов. Для регулировки цвета свечения вы можете изменять рабочие циклы ШИМ-сигналов по трем каналам R/G/B, что позволяет регулировать яркость соответствующих цветов таким образом, чтобы умный светильник излучал смешанный цвет отдельных светодиодов лампы.

Зная основы управления яркостью света и изменения цвета, теперь давайте рассмотрим функциональные блоки один за другим.

Источник питания переменного/постоянного тока (AC/DC) напряжением 220 В

Входное питание умных светодиодных ламп обычно составляет переменный ток высокого напряжения: стандартное напряжение бытовой сети переменного тока в Китае составляет 220 В. Модуль питания 220 В переменного/постоянного тока сначала преобразует переменный ток в постоянный через выпрямительный мост, а затем снижает его до 18–40 В для питания светодиодных драйверов постоянного тока.

Поскольку рабочее напряжение модуля ШИМ-управления и беспроводной связи равно 3,3 В, есть еще один вспомогательный источник питания для снижения напряжения постоянного тока до 3,3 В.

Светодиодный драйвер постоянного тока

Чтобы обеспечить согласованность излучения нескольких светодиодов, вы можете подключить их последовательно и питать светодиоды от одного источника постоянного тока. Яркость светодиодов можно регулировать, управляя этим источником с помощью ШИМ-сигналов. Первый светодиодный драйвер используется для управления светодиодами холодного белого (CW) и теплого белого (WW) цветов, его выходная мощность относительно более высокая. Второй светодиодный драйвер постоянного тока используется для управления красными (R), зелеными (G) и синими (B) светодиодами, в основном для изменения цвета, и его выходная мощность ниже.

Отдельные светодиоды

В интеллектуальных светодиодных светильниках обычно используются светодиоды теплого белого, холодного белого, красного, зеленого и синего цветов, среди которых большее количество теплых белых и холодных белых бусин используется для освещения и меньшее количество красных, зеленых или синих бусин – для регулировки цвета.

ШИМ-управление и беспроводная связь

В продуктах Smart Light для реализации функций ШИМ-управления и беспроводной связи обычно используется высокоинтегрированная система на

кристалле (SoC). SoC поддерживает несколько выходов ШИМ-сигнала, а также один или несколько основных протоколов беспроводной связи, таких как Wi-Fi, Bluetooth LE или ZigBee. Она может запускать встроенную операционную систему реального времени RTOS и поддерживает разработку программных приложений. Модули подключения Wi-Fi позволяют подключить продукт непосредственно к интернету и облачным серверам через Wi-Fi-маршрутизатор. При наличии модулей Bluetooth LE или ZigBee вам дополнительно необходимо настроить шлюзовое устройство для подключения к сети Ethernet или сначала установить Wi-Fi-маршрутизатор, а затем подключить его к интернету и облачным серверам.

Приведенный обзор описывает основные компоненты умных светодиодных светильников, а также реализацию функций регулировки яркости и изменения цвета. Можно сделать вывод, что самое большое различие между продуктами Smart Light и обычными световыми приборами заключается в использовании ШИМ-управления и беспроводной связи. Следующие разделы этой главы будут посвящены тому, как спроектировать минимальную аппаратную систему на базе чипа ESP32-C3 для реализации ШИМ-диммирования, изменения цвета и беспроводного управления. Этот дизайн также применим к другим типам интеллектуальных осветительных устройств, таких как точечные светильники, потолочные светильники, лампы, световые полосы и т. д.

5.2. Аппаратный дизайн базовой системы ESP32-C3

Из раздела 5.1 следует, что модуль ШИМ-управления и беспроводной связи является основным компонентом продуктов Smart Light, отличающим их от традиционных устройств освещения. Как тогда мы должны спроектировать эту базовую систему для реализации функций продуктов Smart Light? Чтобы продемонстрировать соответствующий аппаратный дизайн, в этом разделе мы будем использовать чип ESP32-C3.

ESP32-C3 – это высокоинтегрированная SoC, оснащенная 32-разрядным процессором RISC-V, поддерживающим подключение по Wi-Fi и Bluetooth LE с частотой 2,4 ГГц. Функциональная блок-схема ESP32-C3 показана на рис. 5.4.

ESP32-C3 обладает следующими функциями:

- 32-разрядный одноядерный процессор RISC-V с четырехступенчатым конвейером, работающий на частоте до 160 МГц;
- полноценная **подсистема Wi-Fi**, соответствующая протоколу IEEE 802.11b/g/n и поддерживающая режим станции (Station mode), режим SoftAP, режим SoftAP + Station mode и смешанный режим;
- подсистема **Bluetooth LE** с поддержкой Bluetooth 5 и Bluetooth mesh;
- память емкостью 400 Кбайт SRAM и 384 Кбайт ROM на чипе, а также SPI, интерфейсы Dual SPI, Quad SPI и QPI¹⁴, позволяющие подключаться к внешней флеш-памяти;

¹⁴ QPI (Quick Path Interconnect) – высокопроизводительная шина «точка–точка», разработанная фирмой Intel для соединения процессоров/памяти в многопроцессорных системах. Сочетает высокую пропускную способность (до 15 Гбит/с) и низкое энергопотребление (не более 5,0 мВт на каждый Гбит/с).

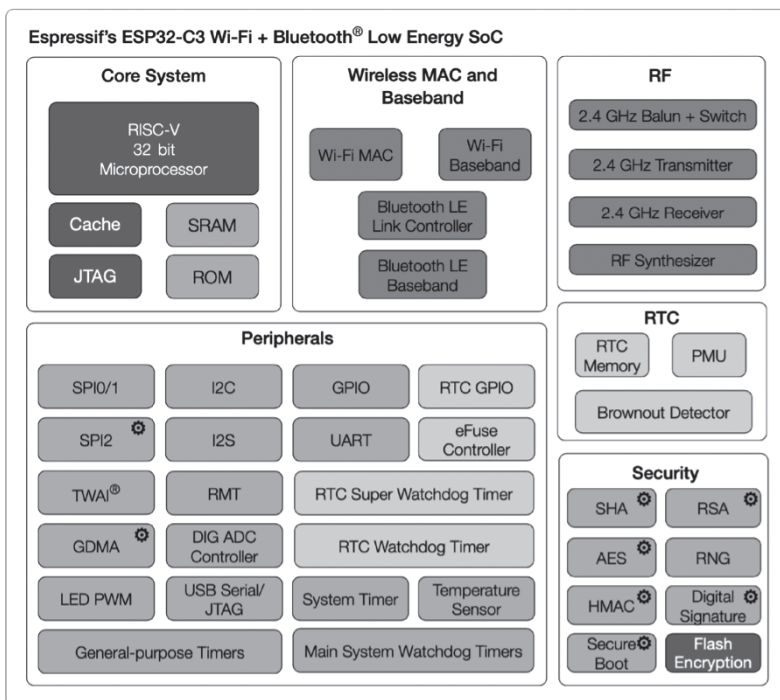


Рисунок 5.4. Блок-схема функций ESP32-C3

- **надежные механизмы безопасности**, обеспечиваемые аппаратными криптографическими ускорителями, поддерживающими стандарты AES-128/256, хеширование, RSA, HMAC, цифровую подпись и безопасную загрузку, шифрование и дешифрование внешней памяти, генератор случайных чисел и контроль разрешений на доступ к внутренней памяти, внешней памяти и периферийным устройствам;
- **богатый набор периферийных интерфейсов**, которые идеально подходят для различных сценариев использования и сложных приложений; 22 программируемых GPIO, которые могут быть гибко сконфигурированы для поддержки приложений PWM, UART, I2C, SPI, I2S¹⁵, АЦП (ADC), TWAI¹⁶, RMT¹⁷ и USB Serial / JTAG.

Серия чипов ESP32-C3 имеет несколько вариантов, включая версии со встроенной флеш-памятью с интерфейсом SPI. ESP8685 – это уменьшенная версия ESP32-C3 (см. табл. 5.1).

¹⁵ I2S (Integrated Inter-chip Sound) – интерфейс внутрочиповой последовательной шины, использующийся для соединения цифровых аудиоустройств.

¹⁶ Two-Wire Automotive Interface (TWAI) – вариант известного протокола CAN, введенный Espressif для своих чипов начиная с ESP32.

¹⁷ Модуль RMT (Remote Control) предназначен для воспроизведения интерфейса инфракрасных пультов дистанционного управления.

Таблица 5.1. Серия ESP32-C3

Версия	Flash (Мбайт)	Раб. температура (°C)	Корпус (мм)
ESP32-C3	–	-40÷105	QFN32 (5×5)
ESP32-C3-FN4	4	-40÷85	QFN32 (5×5)
ESP32-C3-FH4	4	-40÷105	QFN32 (5×5)
ESP32-C3-FH4AZ	4	-40÷105	QFN32 (5×5)
ESP8685H2	2	-40÷105	QFN32 (4×4)
ESP8685H4	4	-40÷105	QFN32 (4×4)

Примечание

- В ESP32-C3FH4AZ, ESP8685H2 и ESP8685H4 контакты для подключения внешней флеш-памяти не подключены.
- Номенклатура серий ESP32-C3: F означает интегрированную флеш-память, H/N указывает рабочую температуру, а AZ – другой идентификационный код.

Для основной схемы подключения ESP32-C3 требуется в общей сложности около 20 резисторов, конденсаторов и индуктивностей, а также один резонатор и одна микросхема флеш-памяти с интерфейсом SPI. Высокая степень интеграции ESP32-C3 делает его подходящим для малогабаритных приложений, таких как умные осветительные приборы. На рис. 5.5 и 5.6 показаны структурная и принципиальная схемы подключения ESP32-C3.

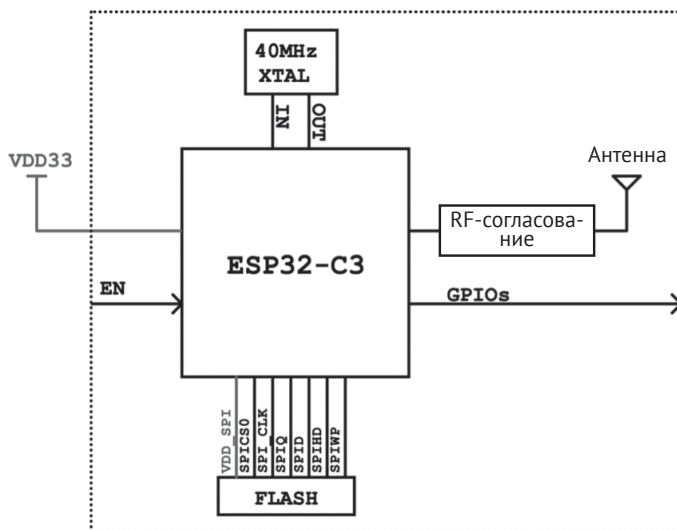


Рисунок 5.5. Базовая блок-схема подключения ESP32-C3

Ниже подробно описаны схемы и разводка печатной платы ESP32-C3.

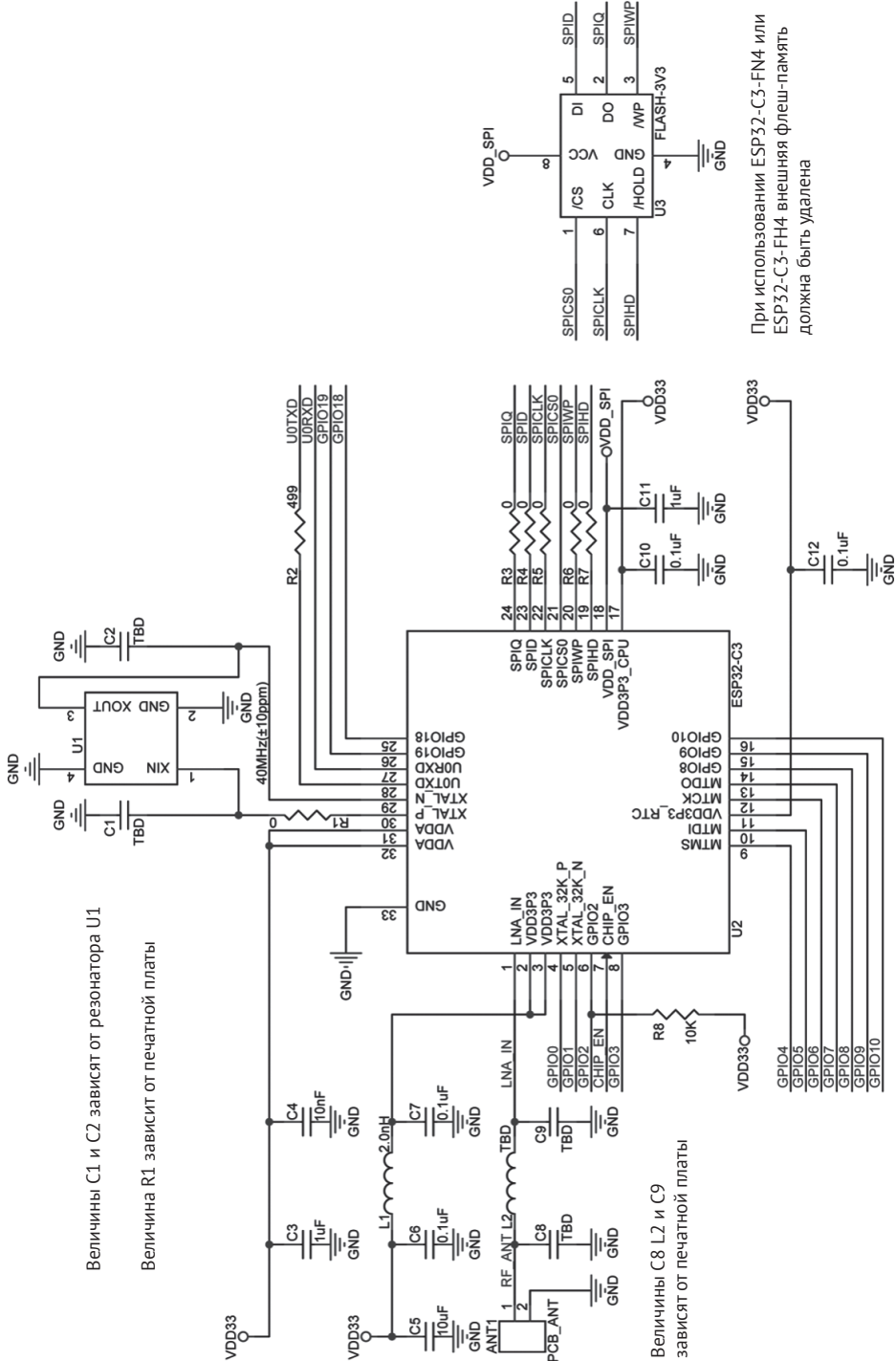


Рисунок 5.6. Базовая схема подключения ESP32-C3

5.2.1. Источник питания

Выходы 11 и 17 являются выводами источника питания для RTC¹⁸ IO и CPU IO соответственно в диапазоне напряжений 3,0–3,6 В. Мы рекомендуем добавить конденсатор емкостью 0,1 мкФ рядом с каждым выводом источника питания. При работе в качестве выходного вывода источника питания VDD SPI (вывод 18) в основном питает внешнюю SPI-память. Мы рекомендуем добавить фильтрующий конденсатор емкостью 1 мкФ между VDD SPI и землей. Когда VDD SPI работает в качестве вывода источника питания 3,3 В для встроенной или внешней флеш-памяти, напряжение питания процессора VDD3P3_CPU должно поддерживаться на уровне 3,0 В или выше, чтобы обеспечить работу флеш-памяти.

Контакты 2, 3, 31 и 32 являются выводами аналогового источника питания, работающими при напряжении 3,0–3,6 В. Пожалуйста, обратите внимание, что когда ESP32-C3 работает в режиме передачи (TX), мгновенный ток будет выше обычного и может привести к нарушениям на шине питания. Поэтому настоятельно рекомендуется добавить к схеме питания конденсатор емкостью 10 мкФ, который может работать в сочетании с конденсатором емкостью 0,1 мкФ¹⁹. Кроме того, необходимо добавить схему LC-фильтра рядом с выводом 2 и выводом 3 для подавления высокочастотных гармоник. Номинальный ток катушки индуктивности предпочтительно составляет 500 мА или выше. Обратитесь к принципиальной схеме и установите соответствующий развязывающий конденсатор рядом с каждым аналоговым выводом питания.

Для одного источника питания рекомендуемое напряжение составляет 3,3 В, а рекомендуемый выходной ток – 500 мА или выше. Мы также предлагаем добавить защитный диод от электростатического разряда на входе питания.

5.2.2. Порядок включения питания и сброс системы

ESP32-C3 использует системный источник питания 3,3 В. Чип должен быть активирован после стабилизации питания шины. Это достигается задержкой активации контакта 7 CHIP_EN после установления питания шины 3,3 В. На рис. 5.7 показано время включения и сброса ESP32-C3. Подробная информация о параметрах приведена в табл. 5.2.

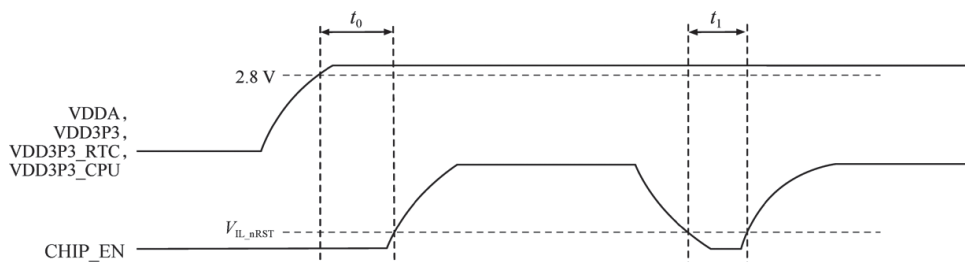


Рисунок 5.7. Время включения и сброса ESP32-C3

¹⁸ Напомним, что в архитектуре ESP32 термином RTC обозначаются не просто часы реального времени (Real Time Clock), а целая отдельная подсистема (модуль RTC, см. рис. 5.4) с низким потреблением, имеющая свои источники тактирования, таймеры, быструю память, отдельные выводы (IO), возможность питания от автономного источника и т. д.

¹⁹ Имеется в виду, что конденсатор 10 мкФ обычный (алюминиевый) электролитического типа, конденсатор 0,1 мкФ – с керамическим диэлектриком.

Таблица 5.2. Параметры задержек включения питания и сброса ESP32-C3

Параметр	Описание	Мин. значение
t_0	Время между установлением на выводах VDDA, VDD3P3, VDD3P3_RTC, VDD3P3_CPU и активацией CHIP_EN	50 μ s
t_1	Длительность уровня сигнала CHIP_EN < V_{IL_nRST} для сброса микросхемы	50 μ s

Чтобы обеспечить стабильное питание ESP32-C3 во время включения, рекомендуется добавить схему RC-задержки на выводе EN микросхемы. Рекомендуемая настройка для RC-задержки обычно $R = 10$ кОм и $C = 1$ мкФ, а конкретные параметры должны быть скорректированы на основе времени включения источника питания и времени включения и сброса чипа.

CHIP_EN также можно использовать в качестве вывода сброса ESP32-C3. Когда CHIP_EN находится на низком уровне, напряжение сброса (V_{IL_nRST}) должно быть не более $(-0,3-0,25) \times V_{DD}$ (где V_{DD} – напряжение ввода/вывода для конкретного набора выводов). Во избежание перезагрузок, вызванных внешними помехами, выполните дорожку CHIP_EN как можно короче и добавьте к выводу подтягивающий резистор, а также конденсатор к общему проводу. Обратите внимание, что контакт CHIP_EN не должен оставаться «висящим в воздухе».

5.2.3. SPI флеш-память

ESP32-C3 поддерживает внешнюю флеш-память объемом до 16 МБ, которая в основном используется для хранения встроенного программного обеспечения, системных параметров, пользовательских параметров, пользовательских данных и т. д. SPI-память питается от вывода VDD_SPI. Мы рекомендуем резервировать последовательный резистор (изначально 0 Ом) на линии SPI, чтобы снизить ток возбуждения, отрегулировать синхронизацию, уменьшить перекрестные и внешние помехи и т. д.

ESP32-C3FN4/FN4 имеет встроенную SPI флеш-память объемом 4 МБ.

5.2.4. Источник тактовых импульсов

В настоящее время прошивка ESP32-C3 поддерживает кварцевый резонатор с частотой 40 МГц. Величины емкостей C1 и C2 зависят от дальнейшего тестирования и настройки общей производительности всей схемы. Пожалуйста, последовательно добавьте резистор или индуктивность (например, R1 на рис. 5.6) в тактовую схему XTAL, чтобы свести к минимуму влияние гармоник резонатора на радиочастотные характеристики. Величина этого компонента (первоначально равная 24 нГн) зависит от дальнейшего радиочастотного тестирования. Обратите внимание, что точность выбранного резонатора должна составлять ± 10 ppm. При фактическом использовании, по мере того как температура умного осветителя растет, отклонение частоты резонатора также увеличится. Поэтому, пожалуйста, убедитесь, что отклонение частоты кварцевого резонатора не превышает 25 ppm, чтобы не повлиять на связь по Wi-Fi.

Хотя в ESP32-C3 встроен RC-генератор в качестве источника тактовой частоты RTC, он также поддерживает внешний резонатор частотой 32,768 кГц,

который выступает в качестве источника тактовой частоты RTC. На рис. 5.8 показана схема подключения внешнего резонатора с частотой 32,768 кГц.

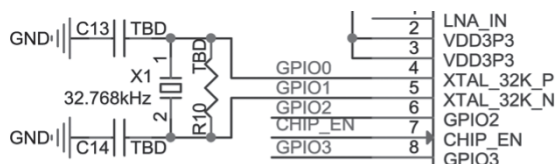


Рисунок 5.8. Схема подключения внешнего резонатора ESP32-C3 (RTC)

Примечание

- Требования к кварцевому резонатору 32,768 кГц:
 - эквивалентное последовательное сопротивление (ESR) ≤ 70 кОм;
 - емкость нагрузки на обоих концах (C13, C14) должна быть настроена в соответствии со спецификацией резонатора.
- Параллельный резистор R10 используется для смещения цепи кварца ($5 \text{ МОм} < R10 \leq 10 \text{ МОм}$). В общем случае вам не нужно подбирать R10.
- Если источник RTC не требуется, можно использовать контакты 4 (XTAL_32K_P) и 5 (XTAL_32K_N) как обычные GPIO.

5.2.5. Радиочастотный сигнал (RF) и антенна

В вашем схемотехническом проекте, пожалуйста, добавьте согласующую цепь между радиочастотным портом (LNA_IN) и антенной для целей согласования. Предпочтительна сеть конфигурации CLC, показанная на рис. 5.9. Параметры C8, L2 и C9 в согласующей сети зависят от фактической схемы антенны и печатной платы.

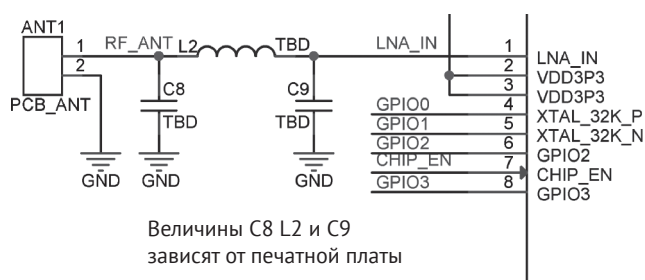


Рисунок 5.9. Цепь CLC для радиочастотного согласования ESP32-C3

Антенну можно выбрать в зависимости от дизайна изделия и общей стоимости. Вы можете выбрать встроенную антенну на печатной плате или внешнюю антенну, такую как стержневая, на гибкой печатной плате, керамическую, 3D-металлическую и т. д. Обычно используемые типы антенн показаны на рис. 5.10. Способы их установки и характеристики приведены в табл. 5.3.



На печатной плате



Стержневая



На гибкой
печатной плате



Керамическая



3D-металлическая

Рисунок 5.10. Часто используемые типы антенн

Таблица 5.3. Способы установки и характеристики широко используемых типов антенн

Тип антенны	Способы установки	Характеристики
Антенна на печатной плате	На плате	Низкая стоимость, средний коэффициент усиления, обычно интегрирована в модули
Стержневая антенна	Внешнее подключение через разъем IPEX	Высокая стоимость, высокий коэффициент усиления, меньшая чувствительность к помехам, хорошая всенаправленная эффективность
Антенна на гибкой печатной плате	Клеевой монтаж	Средняя стоимость, среднее усиление, можно подстроить к упаковке, подходит для продуктов с ограничениями по структуре
Керамическая антенна	Монтаж на печатную плату	Средняя стоимость, низкое усиление, небольшой размер, подходит для малогабаритных модулей
3D-металлическая антенна	Монтаж на печатную плату	Высокая стоимость, высокий коэффициент усиления, меньшая чувствительность к помехам, хорошая всенаправленная эффективность

Радиочастотные характеристики могут быть оптимизированы за счет согласования антенн. После согласования вы можете использовать CMW500, WT-200, IQ View, IQ Xel или другие радиочастотные тестеры для проверки радиочастотных характеристик платы ESP32-C3. Радиочастотный тест включает в себя тест соединений и тест на излучение.

Тест соединений

В ходе тестов соединений используйте радиочастотный кабель 50 Ом для подключения радиочастотного выходного порта платы ESP32-C3 к радиочастотному порту тестера и запустите программное обеспечение для радиочастотного тестирования на ПК. С помощью программного обеспечения вы можете взаимодействовать с основной платой ESP32-C3 и тестером, таким образом управляя тестом. Тестовая установка показана на рис. 5.11.

Тест на излучение

При выполнении теста на излучение поместите антенну тестера и антенну платы ESP32-C3 близко друг к другу в экранированной коробке. Рекомендуется, чтобы расстояние между антеннами составляло около 10 см. Управляйте тестом с помощью программного обеспечения для ПК. Установка для теста на излучение показана на рис. 5.12.

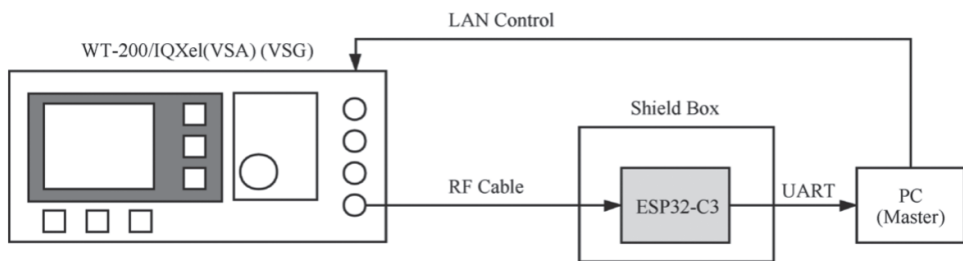


Рисунок 5.11. Установка тестирования RF-соединений основной платы ESP32-C3

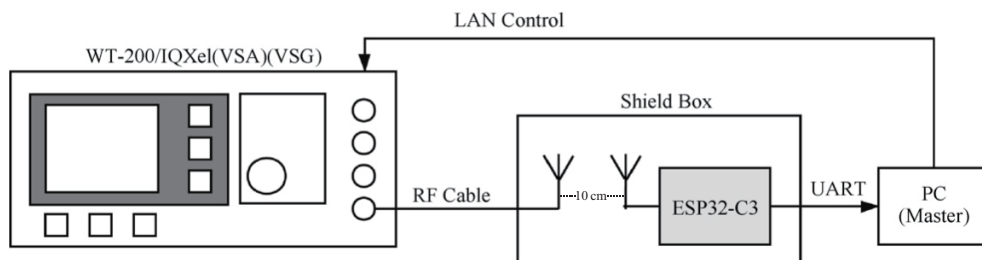


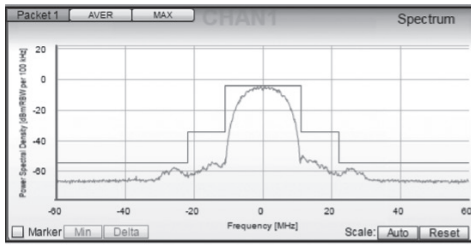
Рисунок 5.12. Установка тестирования излучений основной платы ESP32-C3

Для проверки Wi-Fi RF-эффективности основными параметрами теста являются целевая мощность передачи, EVM, чувствительность приемника и погрешность частоты, как показано в табл. 5.4.

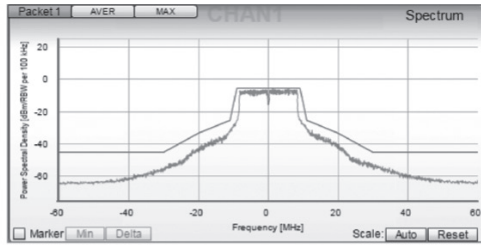
Таблица 5.4. Ключевые параметры для RF-теста Wi-Fi

Рабочий режим и скорость	Целевая мощность TX (дБм)	EVM (dB)	Чувствительность приемника (дБм)	Ошибка частоты (ppm)
IEEE 802.11b, 1 Mbit/s	21.0±2.0	< -24.5	< -98	±25
IEEE 802.11g, 54 Mbit/s	19.0±2.0	< -27.5	< -76.2	±20
IEEE 802.11n, MCS7 HT20	18.5±2.0	< -29	< -74.4	±20
IEEE 802.11n, MCS7 HT40	18.5±2.0	< -28	< -71.2	±20

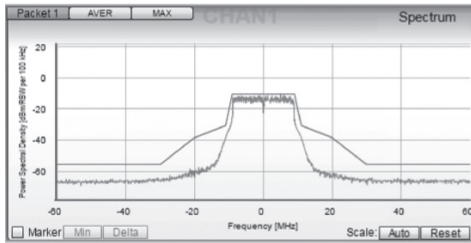
На рис. 5.13 показаны требования к спектральной маске в различных режимах работы.



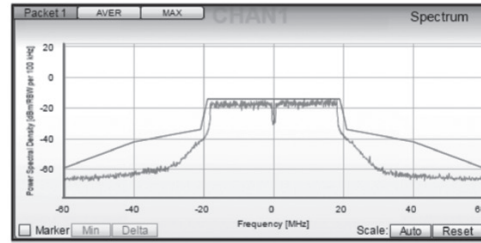
(a) IEEE 802.11 b



(b) IEEE 802.11 g



(c) IEEE 802.11n, 20 MHz



(d) IEEE 802.11n, 40 MHz

Рисунок 5.13. Требования к спектральной маске в различных режимах работы

5.2.6. Выводы управления загрузкой ПО (Strapping Pins)

ESP32-C3 имеет три вывода управления загрузкой ПО: GPIO2, GPIO8 и GPIO9. Во время системного сброса микросхемы установленный на этих контактах уровень фиксируется и сохраняется в защелке до тех пор, пока микросхема не будет выключена. В зависимости от сохраненных уровней микросхема перейдет в различные режимы загрузки ПО после сброса системы. Соответствие между уровнями и режимами загрузки показано в табл. 5.5. После сброса указанные контакты функционируют как обычные GPIO.

Таблица 5.5. Уровень напряжения на выводах управления загрузкой и соответствующий режим загрузки ПО

Выводы	По умолчанию	Загрузка с SPI	Загрузка прошивки
GPIO2	Не определено	1	1
GPIO8	Не определено	Произвольный	1
GPIO9	Слабое внутреннее подтягивание	1	0

5.2.7. GPIO и ШИМ-контроллер

ESP32-C3 имеет 22 вывода GPIO, которым можно назначить различные функции, настроив соответствующие регистры. Все GPIO могут быть сконфигурированы с внутренним подтягиванием к питанию (pull-up), заземлением (pull-down) или настроены на высокий импеданс. Мультиплексор GPIO MUX и матрица GPIO Matrix используются для коллективного управления сигналами выводов GPIO микросхемы. С использованием GPIO MUX и GPIO Matrix (как

показано на рис. 5.14) можно сконфигурировать периферийные входные сигналы с любого вывода GPIO, а периферийные выходные сигналы также могут быть подключены к любому выводу GPIO.

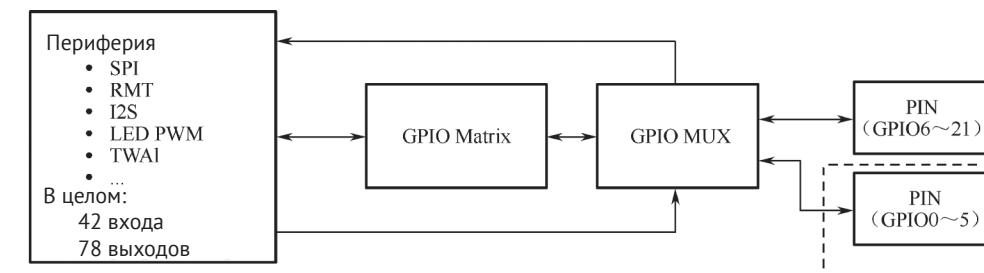


Рисунок 5.14. Мультиплексор IO MUX и матрица GPIO Matrix

ШИМ-контроллер может генерировать независимые ШИМ-сигналы по шести каналам, которые могут быть настроены на любые контакты GPIO через матрицу GPIO Matrix.

5.3. Практика: создание системы умного освещения с помощью ESP32-C3

В разделе 5.2 представлено, как спроектировать минимальную аппаратную систему (основную схему) и систему коммуникаций для умных осветителей на базе ESP32-C3. Система минимального оборудования включает в себя основные периферийные компоненты и антенную часть, которую необходимо с помощью анализатора цепей и радиочастотного тестера согласовать в соответствии с выбранным типом антенны и конструкцией радиочастотной цепи. Согласование антенны может быть затруднено для пользователей-новичков в радиочастотной теме. Итак, есть ли готовая минимальная аппаратная система, настроенная в радиочастотной области, чтобы пользователи могли быстро приступить к разработке умного светильника?

Да, есть готовые к работе аппаратные модули на базе чипа ESP32-C3. Кроме чипа контроллера, эти модули также объединяют кварцевый генератор, флеш-память, антенну, радиочастотную схему и основные периферийные компоненты. Кроме того, модули прошли сертификацию SRRC, CE, FCC и KCC и могут быть непосредственно применены к продуктам умного освещения. В следующих разделах мы выберем один из модулей ESP32-C3 для разработки интеллектуальных осветительных приборов.

5.3.1. Выбор модулей

Как показано в табл. 5.6, с точки зрения типа антенны модули ESP32-C3 можно разделить на антенные модули на печатной плате и внешние антенные модули IPEX; с точки зрения размера и выводов их можно разделить на серии ROOM и серии MINI. Каждый модуль имеет два варианта исполнения в температур-

ном диапазоне: версия от -40 до 85 °C и версия от -40 до 105 °C, – подходящие для умных светильников с различными температурными требованиями. Для осветительных приборов, таких как светодиодные лампы, которые характеризуются высокой внутренней температурой, рекомендуется использовать модуль от -40 до 105 °C. Для других осветительных приборов, которые не имеют высокой внутренней температуры, подходит модуль от -40 до 85 °C.

Таблица 5.6. ESP32-C3 модули

Модуль	Антенна	Температурный диапазон (°C)	Габариты (мм)
ESP32-C3-WROOM-02 	Антенна на печатной плате	$-40 \div 105$ $-40 \div 85$	$18 \times 20 \times 3.2$
ESP32-C3-WROOM-02U 	Внешняя антенна IPEX	$-40 \div 105$ $-40 \div 85$	$18 \times 14.3 \times 3.2$
ESP32-C3-MINI-1 	Антенна на печатной плате	$-40 \div 105$ $-40 \div 85$	$13.2 \times 16.6 \times 2.4$
ESP32-C3-MINI-1U 	Внешняя антенна IPEX	$-40 \div 105$ $-40 \div 85$	$13.2 \times 12.5 \times 2.4$

Примечание

Вы также можете выбрать один из модулей от ESP8685-WROOM-01 до ESP8685-WROOM-07 для уменьшения размеров. Для получения дополнительной информации посетите сайт products.espressif.com.

5.3.2. Настройка ШИМ-сигналов на выводах GPIO

ШИМ-контроллер ESP32-C3 может генерировать независимые ШИМ-сигналы по шести каналам, которые могут быть назначены любому GPIO через матрицу GPIO Matrix. В нашей конструкции пять каналов ШИМ-сигналов используются для управления сигналами R (красный), G (зеленый), B (синий), CW (холодный белый) и WW (теплый белый). В реальном применении мы можем использовать один канал для управления рабочими циклами светодиодов WW и CW для регулировки цветовой температуры, а другой канал для управления общим током для регулировки яркости светодиодов WW и CW. Конфигурация GPIO каждого ШИМ-сигнала показана в табл. 5.7.

Таблица 5.7. Конфигурация GPIO для сигналов PWM

Функция	Номер GPIO
R (Red)	GPIO3
G (Green)	GPIO4
B (Blue)	GPIO5
CW (холодный белый)	GPIO7
WW (теплый белый)	GPIO10

При выборе GPIO убедитесь, что они не находятся на высоком уровне после запуска чипа, в противном случае светодиодная лампа может мигать при включении. Если подходящего GPIO нет, добавьте заземляющий резистор 10 кОм к GPIO для предотвращения мерцания. Можно использовать любые GPIO на ESP32-C3 для функции ШИМ, если они настроены во время инициализации после включения микросхемы.

На рис. 5.15 показана минимальная система управления на базе модуля ESP32-C3-WROOM-02, который подключен к пяти светодиодам красного, зеленого, синего, холодного белого и теплого белого цветов.

5.3.3. Загрузка встроенного ПО и интерфейс отладки

1. Подключите ESP32-C3 к ПК

Микросхема ESP32-C3 интегрирует контроллер USB Serial/JTAG, для которого подключение внешнего моста USB-to-UART или адаптера JTAG не требуется. Интерфейс USB на ESP32-C3 использует GPIO19 в качестве D+ и GPIO18 как D- и может быть напрямую подключен к USB-интерфейсу ПК, чтобы осуществлять

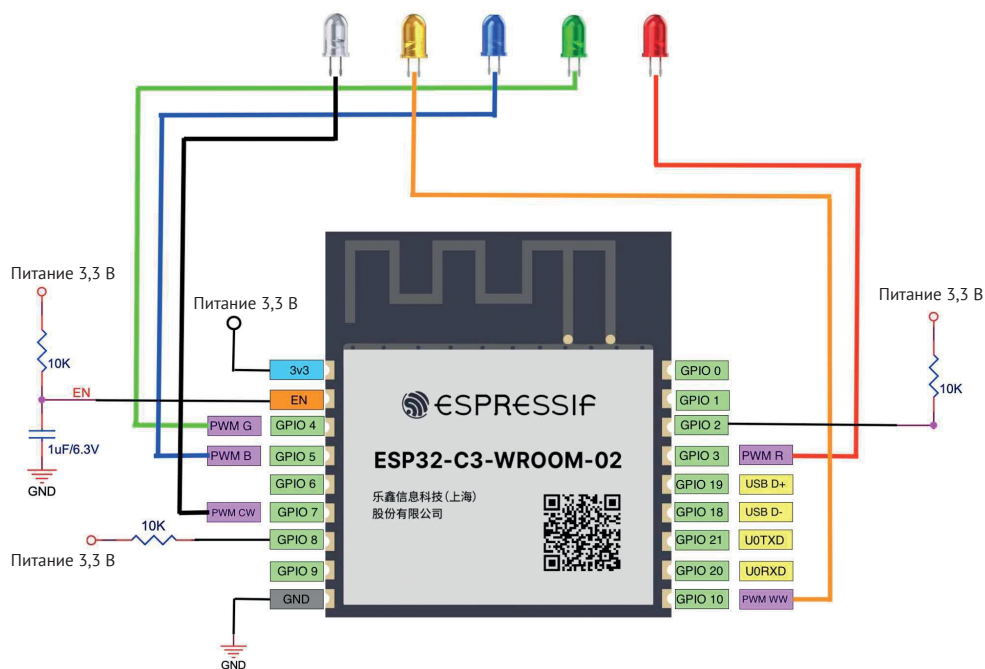


Рисунок 5.15. Минимальная система управления на базе ESP32-C3-WROOM-02

загрузку встроенного ПО, вывод логов и отладку через JTAG. На рис. 5.16 показано, как плата ESP32-C3 подключается к ПК через встроенный контроллер USB Serial/JTAG. Вы можете посетить <https://bookc3.espressif.com/usb> для получения дополнительной информации о применении контроллера USB Serial/JTAG.

Для некоторых отладочных плат ESP32-C3 мост USB-to-UART подключен к интерфейсу UART0 чипа. Разработчикам нужно только подключить USB-интерфейс ПК к отладочной плате через мост, чтобы реализовать загрузку прошивки и вывод логов, как показано на рис. 5.17.

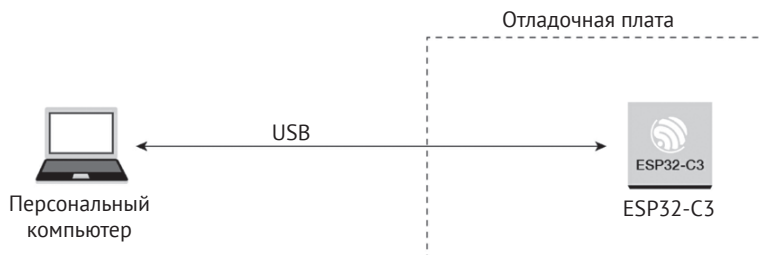


Рисунок 5.16. ESP32-C3 и ПК подключены через контроллер USB Serial/JTAG

Что касается готовой платы, то для экономии места и стоимости мы часто используем программатор с мостом USB-to-UART для подключения к интер-

фейсу UART0 на чипе ESP32-C3, для загрузки встроенного ПО и вывода логов. На рис. 5.18 показано, как программатор с мостом USB-to-UART используется для подключения платы разработки к ПК.

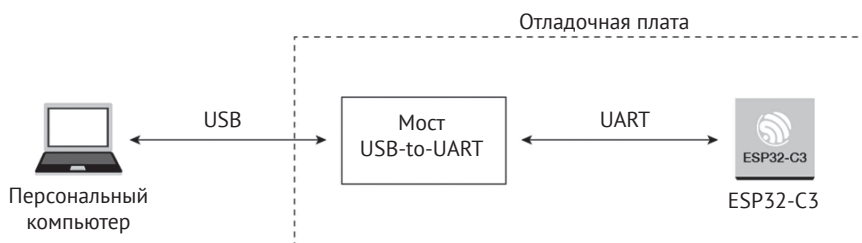


Рисунок 5.17. Мост USB-UART, соединяющий отладочную плату ESP32-C3 и ПК

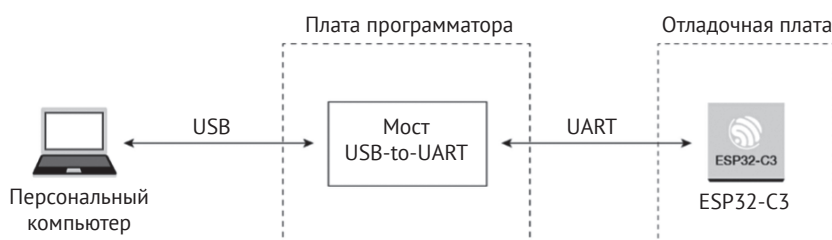


Рисунок 5.18. Программатор с мостом USB-UART, соединяющим ESP32-C3 и ПК

2. Загрузите прошивку

Прошивка и системные параметры ESP32-C3 хранятся во флеш-памяти SPI. Чтобы загрузить прошивку в чип, сначала переведите чип в режим загрузки. Согласно табл. 5.5, GPIO2 и GPIO8 должны быть на высоком уровне, а GPIO9 – на низком. Сбросьте чип, чтобы перейти в режим загрузки. Подключите ESP32-C3 к ПК любым из трех описанных выше способов, чтобы начать загрузку встроенного ПО.

3. Интерфейс отладки

Существует два способа отладки: вывод логов через последовательный порт и JTAG-отладка.

Вывод лога через последовательный порт

Код ESP32-C3 ROM и IDF SDK выводит отладочные сообщения через UART0 по умолчанию. Соедините ESP32-C3 и ПК любым из трех вышеперечисленных способов, чтобы сделать доступным вывод в терминал ПК.

JTAG-отладка

Вы можете напрямую использовать контроллер USB JTAG, встроенный в ESP32-C3, для отладки. Для этого вам нужно соединить выводы JTAG (MTMS/GPIO4, MTDI/GPIO5, MТСК/GPIO6 и MTDO/GPIO7) с внешним JTAG-адаптером для реализации отладки.

5.3.4. Рекомендации по проектированию радиочастотой части

При проектировании умного светового изделия с использованием модуля с антенной на печатной плате обратите внимание на его размещение на базовой плате, чтобы свести к минимуму влияние платы на эффективность ее антенны. Модуль должен располагаться как можно ближе к краю базовой платы. Лучше всего размещать антенну на печатной плате за пределами базовой платы и сохранять точку ее ввода поблизости от платы.

Точка ввода антенны ESP32-C3-WROOM-02 находится справа, а у ESP32-C3-MINI-1 слева. Размещение этих двух модулей показано на рис. 5.19 и 5.20.

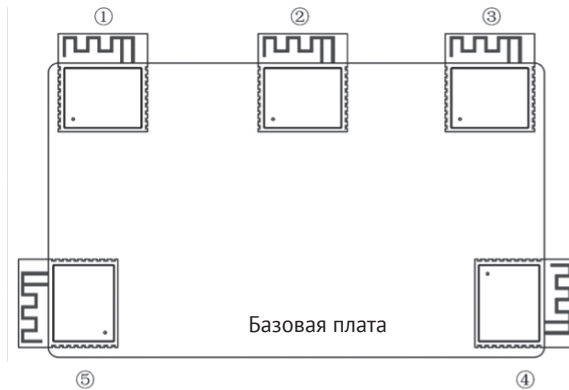


Рисунок 5.19. Модуль ESP32-C3 на базовой плате – точка ввода антенны справа

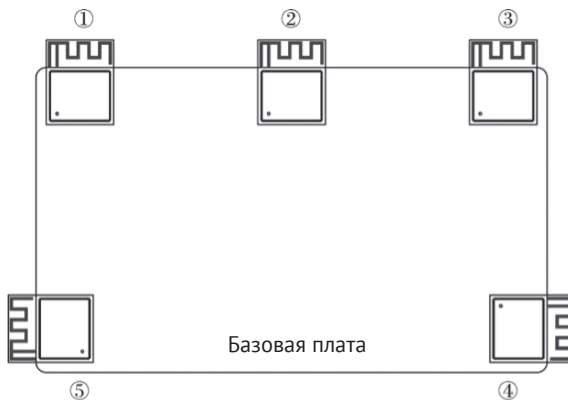


Рисунок 5.20. Модуль ESP32-C3 на базовой плате – точка ввода антенны слева

Примечание

Для точек ввода справа (рис. 5.19) предпочтительнее позиции 3 и 4. Для точки ввода слева (рис. 5.20) позиции 1 и 5 предпочтительнее.

Если рекомендуемые позиции недоступны, убедитесь, что модуль не заслоняется какой-либо металлической оболочкой. Вокруг площади, занятой антенной на печатной плате, в пределах 15 мм не должно быть никаких медных дорожек, проводников или компонентов. Оформление площади должно быть как можно больше, как показано на рис. 5.21. Кроме того, если есть базовая плата, под антенной ее рекомендуется обрезать, чтобы свести к минимуму ее воздействие. При проектировании конечного продукта также обратите внимание на воздействие корпуса на антенну.

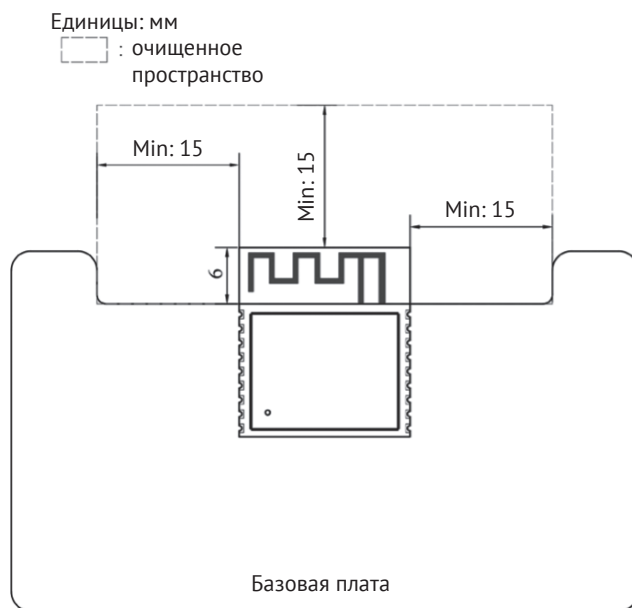


Рисунок 5.21. Зазор на базовой плате

5.3.5. Рекомендации по проектированию источника питания

При включении модуля ESP32-C3 через один контакт источник питания должен быть 3,3 В с выходным током 500 мА или более. Пульсации мощности могут существенно повлиять на радиочастотную эффективность передачи TX. Как правило, пиковое значение пульсаций должно быть менее 80 мВ при передаче пакетов IEEE 802.11n MCS7 и менее 120 мВ при передаче на скорости 11 Мбит/с.

5.4. Резюме

После прочтения этой главы вы должны получить знания по следующим пунктам и быть в состоянии создать свою собственную аппаратную систему для продукта Smart Light:

- компоненты системы, реализация функций умного освещения и функциональные модули интеллектуальных светодиодных светильников;

- принципы и методы регулирования светодиодов и изменения их цвета;
- внедрение ШИМ-управления и беспроводной связи на базе ESP32-C3;
- выбор антенны для беспроводной связи, а также основные параметры и методы тестирования радиочастотных характеристик Wi-Fi;
- особенности ESP32-C3 и его базовой схемотехники;
- выбор модуля ESP32-C3 для упрощения разработки приложений для продуктов Smart Light;
- рекомендации по проектированию продуктов Smart Light на основе ESP32-C3.

В предыдущей главе мы представили функции и аппаратные компоненты продукта IoT (умный свет). В этой главе мы перейдем к разработке его драйвера. Среди четырех уровней архитектуры IoT уровень восприятия и управления предназначен для управления объектами, например чтобы включить/выключить свет или открыть/закрыть занавеску. Для управления различными объектами требуются соответствующие аппаратные драйверы, такие как драйверы светодиодов или двигателей. Уровень восприятия и управления можно комбинировать с облачными вычислениями, интеллектуальным анализом данных, нечетким распознаванием и другими технологиями искусственного интеллекта верхнего уровня для анализа и обработки массовых данных и информации, интеллектуального управления объектами и осуществления контроля в реальном времени, точного управления и принятия научно обоснованных решений.

6.1. Процесс разработки драйверов

Чтобы разработать драйвер датчика, обычно требуется выполнить три шага: изучить датчик, разработать драйвер датчика и протестировать его.

1. Изучение датчика

Из технического описания датчика или какими-то другими средствами узнайте характеристики датчика: его тип, интерфейс связи (например, I2C, SPI), цикл измерения, режим работы, режим питания и т. д.

2. Разработка драйвера датчика

Основной целью разработки драйвера датчика является управление поведением датчика через периферийные интерфейсы контроллера.

3. Тестирование драйвера

После завершения разработки составьте тестовые примеры для проверки, может ли драйвер читать данные и успешно управляться периферийным интерфейсом.

Для разработки драйвера контроллера требуются аналогичные шаги: изучить контроллер, разработать драйвер и протестировать драйвер.

1. Изучение контроллера

Ознакомившись с техническим описанием контроллера, узнайте о принципах работы контроллера, чтобы выбрать подходящий периферийный интерфейс.

2. Разработка драйвера

На основе периферийного интерфейса, выбранного ранее, разработайте соответствующие API-интерфейсы драйверов для других встроенных программных модулей.

3. Тестирование драйвера

Составьте тестовые примеры, чтобы проверить, может ли каждый API драйвера вызываться и работать должным образом.

6.2. Периферийные приложения ESP32-C3

Чип ESP32-C3 имеет богатый набор периферийных интерфейсов, показанных на рис. 6.1. В этом разделе мы представим сценарии применения периферийных интерфейсов ESP32-C3 с точки зрения уровня восприятия и управления.

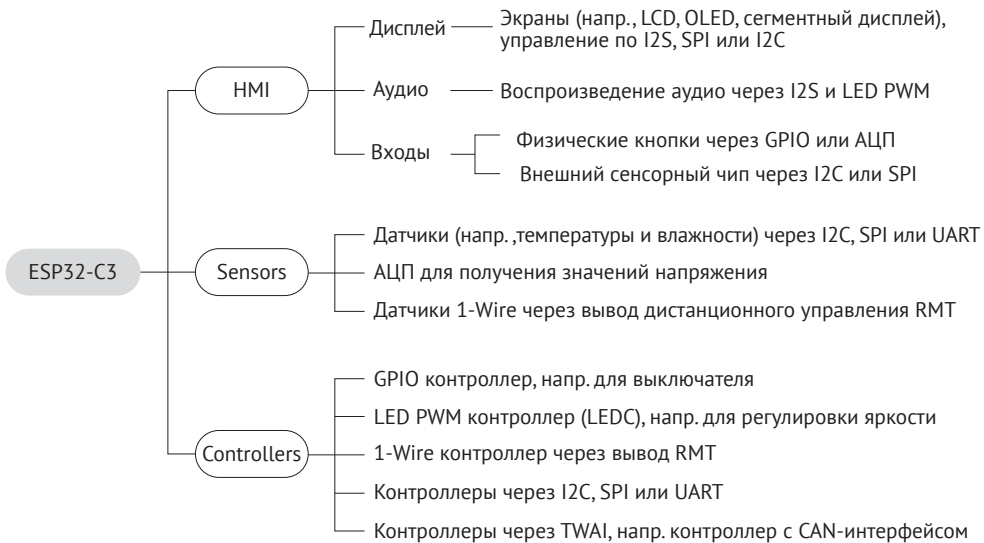


Рисунок 6.1. Периферийные приложения ESP32-C3

Человеко-машинный интерфейс (HMI)

Интерфейсы HMI представляют собой цифровые устройства, состоящие из блока ввода (например, сенсорного экрана и кнопок) для приема команд и дисплея для отображения информации, таким образом реализуя взаимодействие человека и компьютера. В соответствии со сценариями применения к ESP32-C3 могут быть подключены ЖК-дисплеи, монохромные дисплеи или OLED-дисплеи через интерфейсы SPI либо I2C. GPIO и АЦП используются для считывания данных, вводимых физическими кнопками от пользователей. Кроме того, емкостные сенсорные контакты ESP32, ESP32-S2 и ESP32-S3 могут использоваться для подключения сенсорных кнопок, матричных кнопок, линейных ползунков, 2D-сенсорных панелей и датчиков приближения. Эти функции, связанные с кнопками и дисплеем, применимы

к интеллектуальным дверным замкам и другим устройствам с экранами. Интерфейс I2S можно использовать для подключения внешних аудиокодеков для устройств с функциями голосового управления. Интерфейс I2C может использоваться для управления цифровыми шкалами или светодиодными матричными дисплеями, которые обычно используются для встраиваемых приложений. По сравнению с обычными ЖК-дисплеями, эти дисплеи занимают меньше контактов и внутренней памяти, и их реализация проще. Они больше подходят для сценариев с простыми требованиями, такими как время, подсчет и отображение состояний.

Датчики

Датчики относятся к устройствам и компонентам, которые могут преобразовывать различные физические, химические и биологические величины в природе в измеримые электрические сигналы. В разных случаях необходимы различные типы датчиков. Датчики – это нервные окончания интернета вещей и основные компоненты, позволяющие людям в полной мере воспринимать природу. Для разработки интернета вещей необходимо большое количество различных датчиков. Мы можем использовать датчики температуры и влажности, инерционные датчики, датчики освещенности, датчики давления воздуха, датчики жестов и т. д., в зависимости от сценариев применения. Для функционирования и сбора данных они должны быть подключены через различные периферийные интерфейсы. Что касается ESP32-C3, то I2C, SPI и АЦП являются общими периферийными интерфейсами для управления датчиками.

Для справки: драйверы, совместимые с различными типами датчиков, представлены в репозитории *espessif/esp-iot-solution* на нашем GitHub.

Контроллеры

Управление объектами – важная функция уровня восприятия и контроля. Системы управления можно разделить на две категории: системы с разомкнутым контуром и системы с замкнутым контуром. Система управления с разомкнутым контуром, не имеющая механизма обратной связи, использует приводы для непосредственного управления объектами. Их выходные сигналы не оказывают никакого влияния на другие управляющие воздействия внутри системы. В системах управления с замкнутым контуром выходной сигнал обычно измеряется датчиками и подается обратно для сравнения с заданным значением. Отклонение между фактическим результатом и ожидаемой точкой затем используется для автоматической генерации следующей команды. В приложениях для умного дома обычными управляемыми объектами являются освещение, двигатели и выключатели, которые в основном управляются цифровыми и аналоговыми сигналами с контроллера. Периферийные интерфейсы LED PWM, GPIO и АЦП ESP32-C3 могут использоваться для передачи вышеуказанных сигналов.

6.3. Основы построения драйверов светодиодов

В этом разделе представлены основные сведения о драйверах светодиодов, в том числе о цветовых пространствах в освещении, типах светодиодных драйверов, методах диммирования светодиодов и о ШИМ.

6.3.1. Цветовые пространства

Голубой, пурпурный и желтый (*cyan*, *magenta* и *yellow*, CMY) – три основных цвета для рисования. При смешивании друг с другом они создают набор цветов, составляющих цветовое пространство CMY. Мы определяем количество пурпурного по оси *x*, желтого по оси *y* и голубого по оси *z*, таким образом создавая трехмерное пространство, в котором каждый цвет имеет уникальную позицию. CMY – не единственное цветовое пространство. Компьютерные мониторы обычно используют цветовое пространство RGB (красный, зеленый, синий), в котором количество красного, зеленого и синего присваивается осям *x*, *y* и *z*. Другим цветовым пространством является HSV, которое описывает цвета с точки зрения оттенка (*hue*, ось *x*), насыщенности (*saturation*, ось *y*) и интенсивности (*value*, ось *z*). В светотехнической промышленности обычно используется цветовое пространство HSL, которое генерирует цвета, изменяя оттенок (*hue*), насыщенность (*saturation*) и светлоту (*lightness*).

1. Цветовое пространство RGB

Цветовое пространство RGB – это то, с чем мы больше всего знакомы. Как показано на рис. 6.2, это цветовое пространство представлено путем смешивания трех основных цветов – красного, зеленого и синего (*red*, *green*, *blue*) для воспроизведения практически любого цвета²⁰. Это базовое, аппаратно ориентированное цветовое пространство, обычно используемое при обработке изображений, относительно простое для понимания. Оно использует линейную комбинацию из трех основных цветов для представления производного цвета. Три основных компонента сильно коррелированы, поэтому непрерывное переключение цветов не является интуитивно понятным. Чтобы настроить цвет светодиода, вам нужно изменить интенсивность всех трех основных цветов.

На изображения, полученные в естественных условиях, легко влияют естественный свет, преграды и тени. То есть они чувствительны к яркости. Количество трех основных цветов в цветовом пространстве RGB тесно связано с яркостью. Пока яркость меняется, количество всех трех цветов будет меняться соответствующим образом. Однако не существует интуитивного способа отразить это изменение – человеческий глаз не одинаково чувствителен к трем основным цветам. При монохромном зрении человеческий глаз наименее чувствителен

²⁰ Следует отметить, что RGB относится к т. н. *аддитивным* цветовым пространствам, получившим свое название от того, что цвета в них складываются друг с другом (отсутствие цвета – чистый белый). Для того чтобы получить аддитивные цвета, необходимо иметь самосветящиеся источники каждого из основных цветов (например, светодиоды или подсвеченные точки на цветной ЖК-матрице). В отличие от аддитивных, в *субтрактивных* цветовых пространствах цвета вычитаются друг из друга (отсутствие цвета – чистый черный). Этот случай характерен для отражающих поверхностей (например, бумаги). К субтрактивным цветовым пространствам относится упомянутое выше пространство CMY. По рис. 6.7 далее вы можете установить, что RGB-цвета являются дополнительными по отношению к CMY (пары *magenta*–*green*, *cyan*–*red*, *yellow*–*blue*), поэтому теоретически оба пространства должны однозначно переводиться друг в друга. Однако на практике в CMY-палитре трудно получить полное отсутствие цвета (что означало бы 100%-ное отсутствие отраженных лучей), потому что к трем субтрактивным цветам обычно добавляют черный *black*, и полностью пространство называется CMYK.

к красному и наиболее чувствителен к зеленому. Из-за этого считается, что цветовое пространство RGB обладает плохой однородностью к изменению чувствительности. То, как человеческий глаз воспринимает разницу цветов, сильно отличается от евклидова расстояния в цветовом пространстве RGB. Поэтому людям трудно точно представить цвет по количеству трех основных цветов.

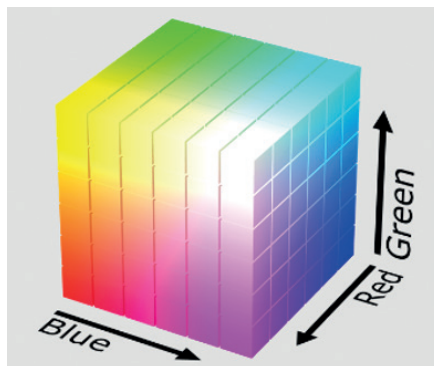


Рисунок 6.2. Цветовое пространство RGB

2. Цветовое пространство HSV

Цветовое пространство HSV, показанное на рис. 6.3, широко используется в компьютерах. По сравнению с цветовым пространством RGB, HSV ближе к человеческому восприятию цветов. Здесь можно интуитивно представлять значения оттенка, насыщенности и яркости цветов для сравнения.

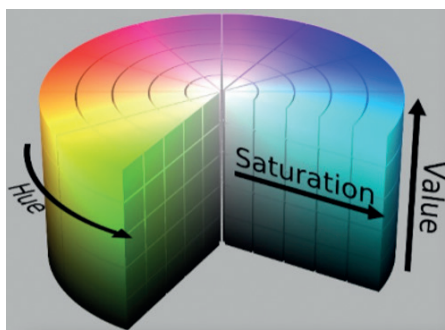


Рисунок 6.3. Цветовое пространство HSV

Объект определенного цвета легче отслеживать в пространстве HSV, чем в пространстве RGB, поэтому цветовое пространство HSV часто используется для классификации объектов определенного цвета. HSV-пространство определяет цвета с точки зрения оттенка (*hue*), насыщенности (*saturation*) и интенсивности (*value*).

Обычно цветовое пространство HSV сопоставляется с цилиндром. Поперечное сечение цилиндра можно рассматривать как полярную систему координат, в которой полярный угол интерпретируется как оттенок, длина полярной оси интерпретируется как насыщенность, а высота оси цилиндра – как интенсив-

ность. Оттенок измеряется в углах и колеблется от 0 до 360°, указывая спектральное положение цвета. На рис. 6.4 показан оттенок в цветовом пространстве HSV.

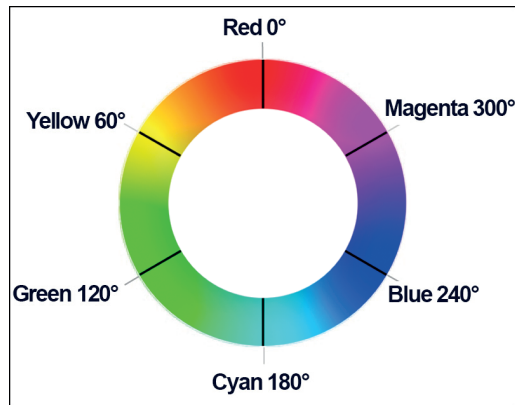


Рисунок 6.4. Цветовое пространство HSV – оттенок (*hue*)

На рис. 6.4 цвета на окружности являются цветами спектра. Угол отсчитывается против часовой стрелки от красного, т. е. 0° соответствует красному, 120° – зеленому, а 240° – синему.

В цветовом пространстве RGB один цвет определяется тремя значениями. Например, желтый цвет представлен как (255,255,0). В цветовом пространстве HSV желтый цвет представлен только одним значением, например *Hue* = 60 (градусов).

Рисунок 6.5 представляет собой половину сечения цилиндра по радиусу при *Hue* = 60° и иллюстрирует насыщенность и интенсивность в цветовом пространстве HSV.

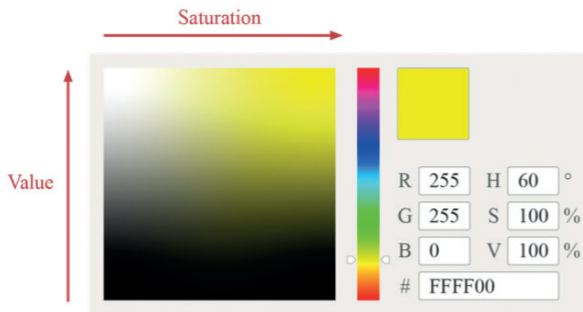


Рисунок 6.5. Цветовое пространство HSV – насыщенность (*saturation*) и интенсивность (*value*)

На рис. 6.5 горизонтальная ось представляет насыщенность, которая указывает на отклонение от чистых цветов спектра. Она колеблется от 0 % до 100 %, где 0 означает чистый белый цвет. Чем выше насыщенность, тем он ближе к цвету спектра, и наоборот.

Вертикальная ось представляет значение, которое указывает на интенсивность (яркость) цвета в цветовом пространстве HSV. Интенсивность колеблется от 0 % до 100 %, где 0 означает обычный черный цвет. Чем выше значение, тем ярче цвет.

3. Цветовое пространство HSL

Цветовое пространство HSL аналогично цветовому пространству HSV. Оно также состоит из трех компонентов: оттенка (*hue*), насыщенности (*saturation*) и светлоты (*lightness*). Разница заключается в последнем компоненте: светлота в HSL означает интенсивность цвета (тогда как яркость в HSV представляет яркость свечения). Светлота в HSL, равная 100, означает белый цвет, в то время как светлота, равная 0, означает черный. Для HSV значение яркости = 100 означает спектральный цвет, тогда как значение яркости = 0 также означает черный. На рис. 6.6 показано цветовое пространство HSL.

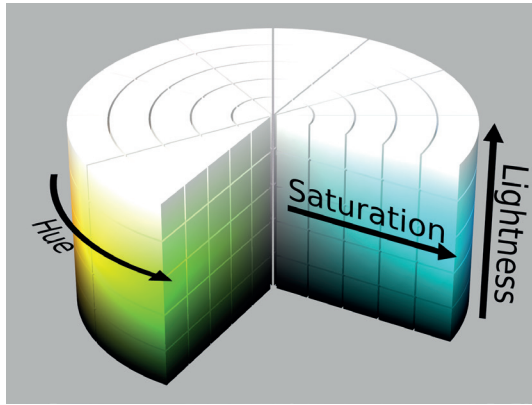


Рисунок 6.6 . Цветовое пространство HSL

На рис. 6.7 показаны оттенки в цветовом пространстве HSL, представляющие диапазон цветов, воспринимаемых человеческим глазом. Они распределены по плоскому цветовому кругу; каждый представлен оттенком от 0 до 360°. Таким образом мы можем изменить цвет, вращая цветовой круг без изменения насыщенности или светлоты.

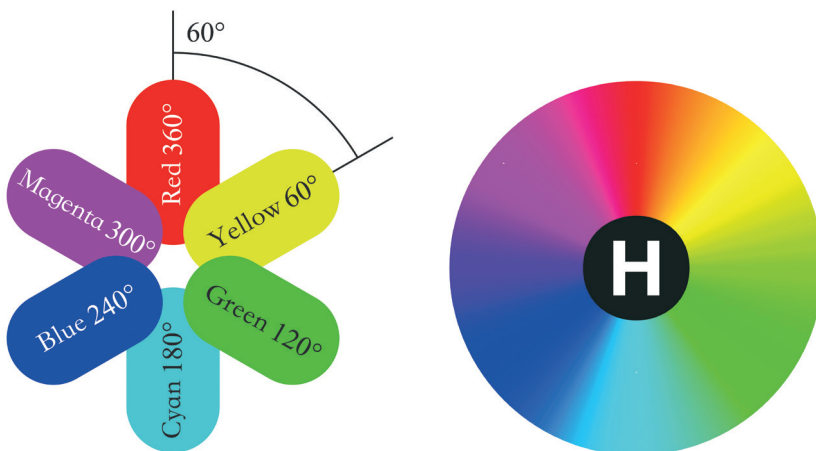


Рисунок 6.7. Цветовое пространство HSL – оттенок (*hue*)

На рис. 6.8 показана насыщенность в цветовом пространстве HSL в диапазоне от 0 % до 100 %. Он описывает изменение чистоты цвета при том же оттенке и светлоте. Чем больше насыщенность, тем более яркий и менее серый цвет.



Рисунок 6.8. Цветовое пространство HSL – насыщенность (*saturation*)

На рис. 6.9 показана светлота в цветовом пространстве HSL, которая представляет собой интенсивность цвета. Она колеблется от 0 % до 100 %. Чем меньше значение, тем темнее цвет и тем ближе к черному, и наоборот.



Рисунок 6.9. Цветовое пространство HSL – светлота (*lightness*)

Представленные три цветовых пространства просто описывают цвета разными способами и, таким образом, могут быть взаимно преобразованы. На практике светодиодные источники используют цветовое пространство RGB, а яркость красных, зеленых и синих светодиодов регулируется для получения различных цветов. Однако пользовательский интерфейс и команды управления обычно используют цветовое пространство HSV или HSL. Поэтому драйвер светодиода должен преобразовать значения из формата HSV или HSL в RGB-формат для получения ожидаемого цвета светодиода.

6.3.2. Светодиодный драйвер²¹

По сравнению с традиционными источниками света светодиоды более энергоэффективны и экологичны, а срок их службы больше. Осветительный светодиод – низковольтный сильноточный полупроводниковый компонент. Его интенсивность свечения увеличивается с увеличением прямого тока. При выборе светодиодного драйвера необходимо учитывать условия работы. Если драйвер чувствителен к температуре окружающей среды, нам следует использовать компоненты, которые выделяют меньше тепла или рассеивают его. Светодиодный драйвер является основным компонентом интеллектуальных светильников и напрямую влияет на срок службы и удобство использования умных лампочек. В настоящее время существует в основном два типа светодиодных драйверов.

²¹ В данном разделе «драйвер» употребляется в значении аппаратного устройства, управляющего светодиодами лампочки.

Драйверы постоянного напряжения

Драйверы постоянного напряжения обеспечивают стабильное напряжение на клеммах светодиода, а ток изменяется в зависимости от нагрузки. При питании от источника постоянного напряжения каждому светодиоду требуется подходящий резистор, чтобы излучать свет одинаковой яркости.

Драйверы постоянного тока

Драйверы постоянного тока стабилизируют ток, протекающий через светодиод, а напряжение на светодиоде изменяется в зависимости от нагрузки. При питании от источника постоянного тока можно управлять яркостью светодиода (осуществлять диммирование), управляя протекающим через него током.

6.3.3. Диммирование светодиодов

Диммирование – основная функция интеллектуальных светодиодных светильников, включая изменение цвета, яркости и статуса включения/выключения. Пользователи могут настраивать светодиодные светильники с помощью приложения для смартфона, пульта дистанционного управления и т. д. Существует три метода диммирования светодиодов.

Диммирование симистором (триак-диммирование)

При использовании триак-диммирования форма сигнала входного напряжения изменяется в зависимости от угла включения симистора, тем самым изменяя действующее значение входного напряжения и в конечном итоге яркость светодиода. Триак-диммирование подходит для традиционных ламп, таких как лампы накаливания и люминесцентные лампы²².

ШИМ-диммирование

По сути, ШИМ периодически включает/выключает светодиоды и приглушает свет, изменяя сигналы ШИМ их частоты и рабочего цикла.

I2C-диммирование

Микросхема линейного контроллера светодиодов постоянного тока с интерфейсом I2C подходит для управления маломощными светодиодными источниками света. Такие микросхемы получают управляющие сигналы через входные интерфейсы I2C и могут регулировать ток нескольких независимых выходных интерфейсов.

Среди трех перечисленных методов управления светодиодами ШИМ-диммирование работает лучше всего. Оно гарантирует отсутствие цветовых сдвигов и стабильность при низкой яркости и поэтому используется очень широко.

На рис. 6.10 показана блок-схема ШИМ-управления яркостью, которая в основном содержит тактовую схему переключения, основную схему управления и ШИМ-контроллер. Схема переключения начинает генерировать тактовый сигнал после прихода сигнала включения. Основная схема управления принимает тактовый

²² Для люминесцентных ламп регулировка яркости с помощью действующего значения напряжения работает только для определенных конструкций и, как правило, в небольшом диапазоне.

сигнал и генерирует три импульсных сигнала, которые соответственно выводятся на три ШИМ-контроллера. ШИМ-контроллеры выводят импульсы тока различной продолжительности для регулировки яркости соответствующего светодиода.

Основная схема управления обычно включает в себя блок микроконтроллера, вход которого подключен к схеме переключающего тактового сигнала и три выхода соответственно подключены к трем ШИМ-контроллерам. Выходы ШИМ-контроллеров подключены к красному, зеленому и синему светодиодам соответственно, тем самым контролируя их яркость, чтобы получить ожидаемый цвет. Все светодиоды упакованы в один корпус-рассеиватель.

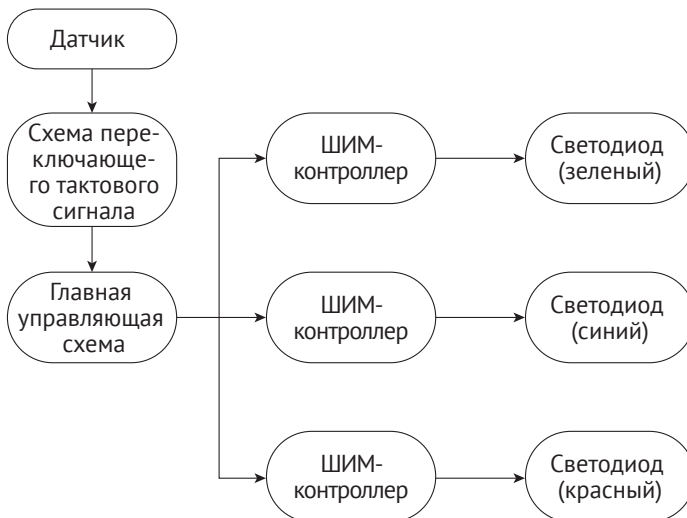


Рисунок 6.10. Блок-схема ШИМ-диммирования

6.3.4. Введение в ШИМ

Широтно-импульсная модуляция (ШИМ) – это метод, который преобразует аналоговые сигналы в импульсные сигналы различной длительности (средство формирования аналогового выхода с помощью цифровых сигналов). Его можно использовать для управления яркостью светодиодов, частотой вращения двигателей постоянного тока и т. д.

Он имеет три основных параметра: частоту, период и коэффициент заполнения. *Частота ШИМ* – это количество раз, когда ШИМ-сигнал переходит с высокого уровня на низкий и обратно в течение одной секунды. Частота измеряется в герцах. *Период ШИМ* является величиной, обратной частоте ШИМ.

Коэффициент заполнения ШИМ равен отношению времени высокого уровня к длительности периода ШИМ и варьируется от 0 % до 100 %²³. На рис. 6.11 по-

²³ В электронике для прямоугольных импульсов с переменной шириной чаще употребляется величина, обратная коэффициенту заполнения, называемая *скважностью*. Соответственно, коэффициент заполнения 50 % эквивалентен скважности, равной 2, 100 % – скважности, равной единице, а 0 % равносителен скважности, равной бесконечности (что имеет вполне логичную физическую интерпретацию, как полное отсутствие импульсов).

казан коэффициент заполнения ШИМ. Например, если период ШИМ составляет 10 мс, а длительность импульса – 8 мс, то коэффициент заполнения равен $8/10 = 80\%$.

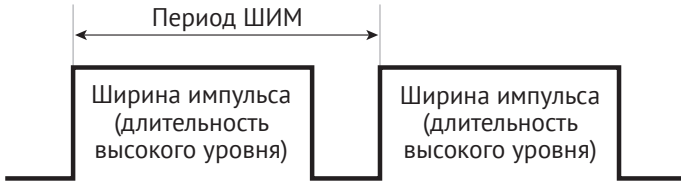
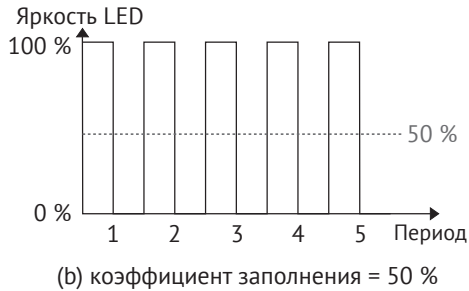
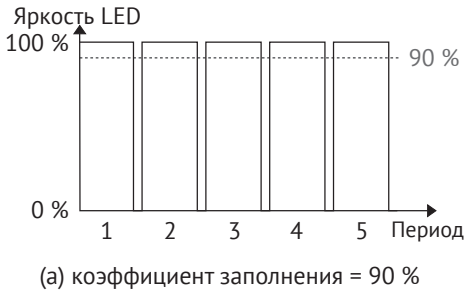


Рисунок 6.11. Коэффициент заполнения ШИМ

При использовании ШИМ для управления светодиодом, если индикатор включается на 1 секунду, а затем выключается на 1 секунду (т. е. период = 2 с, рабочий цикл = 50%), светодиод будет мигать. Если период сократить до 200 мс, то есть светодиод горит в течение 100 мс, а затем выключается на 100 мс, то будет казаться, что светодиод мигает с более высокой частотой. Из-за инерционности зрения, по мере того как цикл продолжает уменьшаться, будет достигнут критический порог, при котором человеческий глаз не сможет воспринимать мигание светодиода. В этот момент видимая яркость сочетает в себе включенное и выключенное состояния, что приводит к стабильной средней яркости. Эта средняя яркость напрямую связана с коэффициентом заполнения ШИМ, как показано на рис. 6.12. Таким образом, мы можем приглушить яркость светодиодов, отрегулировав рабочий цикл ШИМ.



..... Средняя яркость

Рисунок 6.12. Связь между коэффициентом заполнения ШИМ и средней яркостью

6.4. Разработка драйвера для регулирования светодиодов

Разобравшись с основами светодиодных драйверов, мы можем приступить к разработке драйвера регулировки яркости на основе чипа ESP32-C3. В основном это включает разработку функциональных API-интерфейсов для управления переключателем, яркостью, цветом и цветовой температурой. В повседневной жизни обычно ожидается, что при включении цвет, яркость и цветовая температура источника света будут соответствовать его предыдущему состоянию. Это требует сохранения состояния индикатора, когда он выключен. Чтобы достичь этого, мы можем использовать функцию энергонезависимого хранилища (NVS), предоставляемую ESP-IDF.

Поэтому, прежде чем писать код драйвера, также необходимо ознакомиться со светодиодным ШИМ-контроллером, входящим в состав ESP32-C3, процедурами его программирования и энергонезависимой памятью.

6.4.1. Энергонезависимая память (NVS)

Энергонезависимая память в ESP-IDF использует часть основной флеш-памяти через API-интерфейсы для хранения пар ключ–значение `esp_partition.h`. Поскольку NVS сохраняет состояние, даже если устройство перезапущено или выключено, данные не будут потеряны. NVS была специально разработана для предотвращения повреждения данных, вызванного сбоем питания, и для распределения записанных данных по NVS на случай ее износа. Выделенный раздел во флеш-памяти, используемый NVS, хранит данные различных типов, такие как целые числа, строки, заканчивающиеся нулем, и двоичные данные.

NVS подходит для хранения данных небольшого размера, а не таких объемных, как строки или большие двоичные объекты (BLOB-объекты), которые должны обрабатываться файловой системой FAT для выравнивания износа памяти. В проектах интернета вещей NVS может хранить не только уникальные данные при серийном производстве продуктов, но и пользовательские данные, относящиеся к конкретному приложению.

Ниже приведено несколько ключевых концепций NVS: пары ключ–значение, пространства имен, безопасность, устойчивость к несанкционированному доступу и надежность.

Пары ключ–значение

NVS работает с парами ключ–значение вида «key:value». Ключи представляют собой строки ASCII длиной до 15 символов, а значения могут быть любого из следующих типов:

- целые числа: `uint8_t`, `int8_t`, `uint16_t`, `int16_t`, `uint32_t`, `int32_t`, `uint64_t` и `int64_t`;
- нуль-терминированные строки;
- двоичные данные переменной длины.

Пространства имен

Чтобы смягчить потенциальные конфликты в именах ключей между различными компонентами, NVS присваивает пространство имен для каж-

дой пары ключ–значение, которое следует тому же правилу именования, что и ключи, т. е. максимальная длина составляет 15 символов. Эти имена указываются при вызове `nvs_open()` или `nvs_open_from_part()`. Такой вызов возвращает непрозрачный дескриптор, который используется в последующих вызовах функций `nvs_get_*()`, `nvs_set_*()` и `nvs_commit()`. Таким образом, с каждым пространством имен связан дескриптор, и имена ключей не будут конфликтовать с теми же именами в других пространствах имен. Обратите внимание, что пространства имен с одинаковыми именами в разных разделах NVS считаются отдельными пространствами имен.

Безопасность, устойчивость к несанкционированному доступу и надежность

После шифрования NVS-данные будут храниться в зашифрованном виде. Если шифрование NVS не включено, любой пользователь, имеющий физический доступ к флеш-памяти, может изменять, стирать или добавлять пары ключ–значение. Если включено шифрование NVS, пары ключ–значение не могут быть изменены или добавлены без знания соответствующего ключа шифрования. Однако операция стирания не защищена от несанкционированного доступа.

Когда флеш-память оказывается нарушенной, NVS попытается выполнить восстановление. Отключение питания устройства в произвольный момент, а затем повторное включение не приведет к потере данных. Однако если устройство выключено во время записи новой пары ключ–значение, эта конкретная пара может быть потеряна.

6.4.2. Светодиодный ШИМ-контроллер (LEDC)

Светодиодный ШИМ-контроллер ESP32-C3 может генерировать шесть независимых цифровых сигналов со следующими функциями:

- шесть независимых ШИМ-генераторов (т. е. шесть каналов);
- четыре независимых таймера, поддерживающих деление на части;
- автоматическое плавное изменение коэффициента заполнения (т. е. постепенное увеличение/уменьшение коэффициента заполнения ШИМ без вмешательства ESP32-C3) с генерацией прерывания по завершении процесса;
- регулируемая фаза выходного ШИМ-сигнала;
- ШИМ-сигнал в режиме Light-sleep (подробнее о режимах с низким энергопотреблением см. в главе 12);
- максимальное разрешение ШИМ 14 бит.

Четыре таймера идентичны по своим характеристикам и принципу работы. В следующих разделах таймеры в совокупности называются `Timerx` (где x находится в диапазоне от 0 до 3). Аналогичным образом шесть ШИМ-генераторов также идентичны по характеристикам и работе и, соответственно, называются `PWMn` (где n меняется от 0 до 5). На рис. 6.13 показана структура ШИМ-таймера с управлением светодиодами.

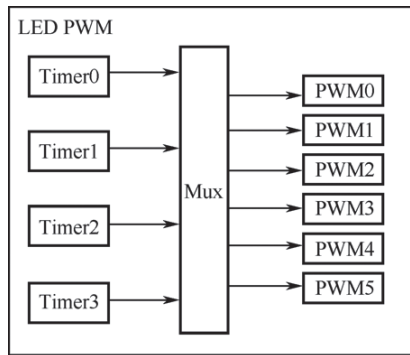


Рисунок 6.13. Светодиодный ШИМ-таймер

Четыре таймера могут быть настроены независимо (например, настраиваемый делитель тактов и значение переполнения счетчика), и каждый поддерживает счетчик тактов (т. е. счетчик, который считает такты эталонного тактового сигнала). Каждый генератор ШИМ выбирает один из четырех таймеров, используя значение счетчика таймера в качестве эталона для генерации ШИМ-сигналов, и выводит сигналы на таймер.

На рис. 6.14 показаны основные функциональные блоки таймера и ШИМ-генератора.

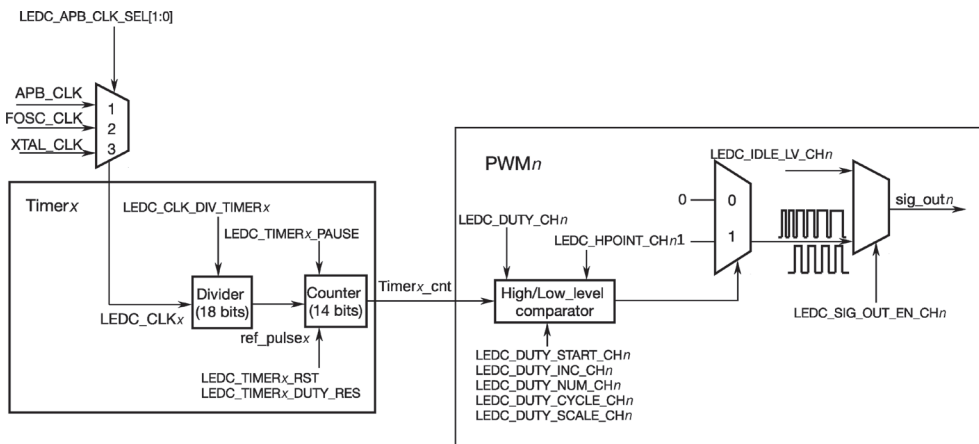


Рисунок 6.14 . Функциональные блоки таймера и ШИМ-генератора

Для генерации ШИМ-сигналов ШИМ-генератору (PWM_n) необходимо выбрать один из четырех таймеров ($Timer_x$) и использовать значение его счетчика в качестве эталона для генерации сигналов. Каждый ШИМ-генератор имеет компаратор и два мультиплексора. Компаратор сравнивает значение 14-разрядного счетчика таймера ($timer_x_cnt$) с двумя триггерными значениями компаратора $hpoint_n$ и $lpoint_n$. Когда $timer_x_cnt$ равен $hpoint_n$ или $lpoint_n$, будет сгенерирован ШИМ-сигнал высокого или низкого уровня соответственно.

На рис. 6.15 показано, как $hpoint_n$ или $lpoint_n$ используется для генерации ШИМ-сигналов с заданным коэффициентом заполнения.

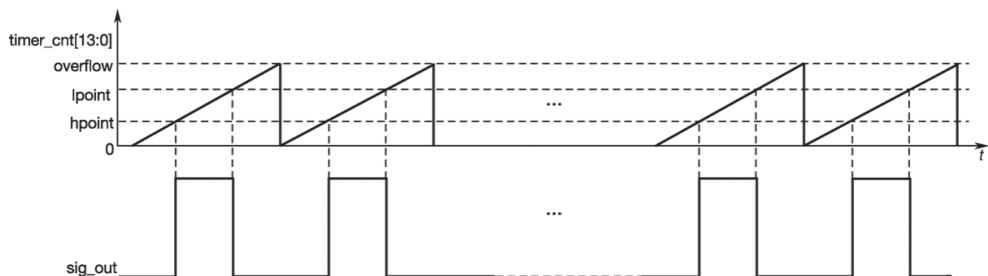


Рисунок 6.15. Генерация ШИМ-сигналов с заданным коэффициентом заполнения с использованием $hpoint_n$ или $lpoint_n$

Генераторы ШИМ могут плавно изменять коэффициент заполнения выходного сигнала ШИМ. Когда включено изменение коэффициента заполнения, значение $lpoint_n$ будет увеличиваться/уменьшаться каждый раз, когда счетчик переполняется определенное количество раз. На рис. 6.16 показан процесс плавного изменения коэффициента заполнения.

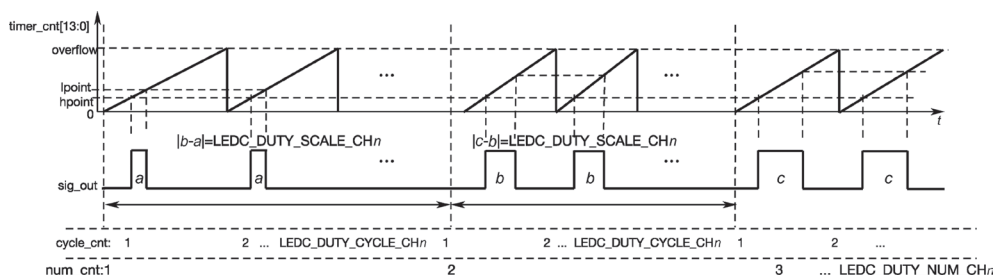


Рисунок 6.16. Плавное изменение коэффициента заполнения

6.4.3. Программирование ШИМ для светодиодов

Ознакомившись с устройством LEDC в составе ESP32-C3, теперь можно настроить контроллер с использованием API-интерфейсов LED PWM, предоставляемых ESP-IDF. Конфигурация включает три шага, как показано на рис. 6.17:

1. **Настройте таймер**, указав частоту и разрешение ШИМ-сигналов.
2. **Настройте канал**, сопоставив таймер с GPIO, на которые выводятся сигналы ШИМ.
3. **Настройте выходные сигналы ШИМ** для управления светодиодом. Яркость светодиода можно изменить с помощью программного управления или аппаратной функции плавного изменения коэффициента заполнения.

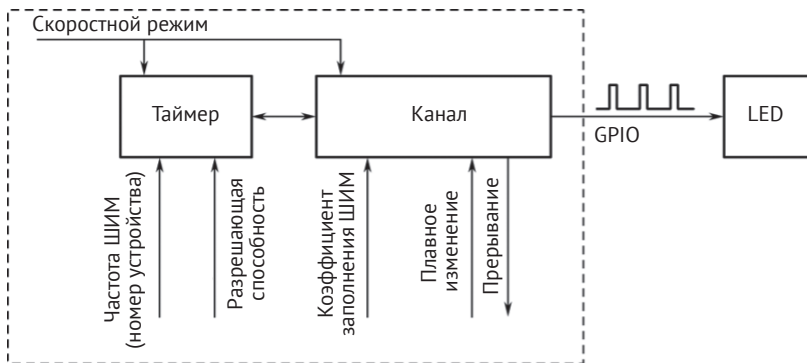


Рисунок 6.17. Этапы настройки ШИМ-контроллера

1. Настройка таймера

Таймеры можно настроить, вызвав `ledc_timer_config()`; в эту функцию необходимо передать структуру `ledc_timer_config_t` со следующими параметрами:

- скоростной режим (значение этого параметра должно быть `LEDC_LOW_SPEED_MODE`);
- номер таймера `timer_num`;
- частота ШИМ;
- разрешение коэффициента заполнения ШИМ.

Частота ШИМ обратно пропорциональна разрешению коэффициента заполнения, так как более высокая частота приводит к меньшему доступному числу градаций длительностей для данного периода и наоборот. Эта взаимосвязь может быть более важной, если контроллер LEDC используется для целей, отличных от изменения яркости светодиодов.

2. Настройка канала

После настройки таймера вам также необходимо настроить требуемый канал (один из `ledc_channel_t`), вызвав `ledc_channel_config()`. В функцию необходимо передать структуру `ledc_channel_config_t` с параметрами конфигурации канала.

Затем канал начнет работать в соответствии с параметрами этой структуры и генерировать ШИМ-сигналы на выбранных GPIO с частотой, указанной на шаге 1, и коэффициентом заполнения, указанным на шаге 2. Этот процесс можно приостановить в любое время, вызвав функцию `ledc_stop()`.

3. Настройка ШИМ-сигналов

Есть несколько способов изменить ШИМ-сигнал, работающий с заданной скважностью и частотой. Для диммирования светодиодов (изменения цвета и яркости) мы в первую очередь должны регулировать коэффициент заполнения.

Программное изменение коэффициента заполнения ШИМ

Чтобы задать коэффициент заполнения, используйте специальную функцию `ledc_set_duty()`. После этого вызовите `ledc_update_duty()`, дабы активировать изменения. Чтобы проверить текущее установленное значение, используйте функцию `ledc_get_duty()`.

Другим способом задать коэффициент заполнения, а также некоторые иные параметры канала, является вызов функции `ledc_channel_config()`.

Коэффициент заполнения, передаваемый этим функциям, зависит от заданной разрешающей способности `duty_resolution`, и его значение находится в диапазоне от 0 до $2^{\text{duty_resolution}} - 1$. Например, если `duty_resolution` равно 10, то значения рабочего цикла могут варьироваться от 0 до 1023.

Изменение коэффициента заполнения ШИМ аппаратным способом

LEDC обеспечивают средства для постепенного изменения (затухания) коэффициента заполнения. Чтобы использовать эту функцию, включите ее с помощью `ledc_fade_func_install()`, а затем настройте, вызвав одну из следующих функций:

```
1. esp_err_t ledc_set_fade_with_time(ledc_mode_t speed_mode,
2.                                 ledc_channel_t channel,
3.                                 uint32_t target_duty,
4.                                 int max_fade_time_ms);
5.
6. esp_err_t ledc_set_fade_with_step(ledc_mode_t speed_mode,
7.                                  ledc_channel_t channel,
8.                                  uint32_t target_duty,
9.                                  uint32_t scale,
10.                                 uint32_t cycle_num);
11.
12. esp_err_t ledc_set_fade(ledc_mode_t speed_mode,
13.                         ledc_channel_t channel,
14.                         uint32_t duty,
15.                         ledc_duty_direction_t fade_direction,
16.                         uint32_t step_num,
17.                         uint32_t duty_cycle_num,
18.                         uint32_t duty_scale);
```

Наконец, вызовите `ledc_fade_start()`, чтобы инициировать плавное изменение. Если оно больше не требуется, плавное изменение можно отключить с помощью `ledc_fade_func_uninstall()`.

4. Диапазон частоты ШИМ и коэффициента заполнения

Светодиодный ШИМ-контроллер LEDC в основном используется для управления яркостью светодиодов. Это обеспечивается большой гибкостью настроек рабочего цикла ШИМ. Например, частота ШИМ 5 кГц может иметь максимальное рабочее разрешение 13 бит. Это означает, что режим работы может быть установлен в диапазоне от 0 % до 100 % с разрешением $\approx 0,012\%$ ($2^{13} = 8192$ дискретных уровня яркости светодиода). Пожалуйста, обратите внимание, что эти параметры зависят от тактового сигнала, синхронизирующего таймер светодиодного контроллера, который, в свою очередь, синхронизирует канал.

LEDC может использоваться для генерации сигналов с более высокими частотами, достаточными для синхронизации других устройств, таких как модули цифровых камер. В этом случае максимальная частота может составлять 40 МГц с рабочим разрешением 1 бит. Это означает, что рабочий цикл зафиксирован на уровне 50 % и не может быть скорректирован.

API LEDC сообщит об ошибке, когда сконфигурированная частота и рабочее разрешение превысят допустимые значения для оборудования LEDC. Например, попытка установить частоту 20 МГц и разрешение нагрузки до 3 бит приведет к следующему сообщению об ошибке на мониторе последовательного порта:

```
[E (196) ledc: requested frequency and duty resolution cannot be achieved, try reducing freq_hz or duty_resolution. div_param=128]
```

В сообщении говорится, что необходимо уменьшить либо разрешение `duty_resolution`, либо частоту `freq_hz`. Например, установка рабочего разрешения в 2 бита может решить эту проблему и позволит установить коэффициент заполнения с шагом 25 %, т. е. на значения 25 %, 50 % или 75 %.

Драйвер LEDC также будет фиксировать и сообщать о попытках настройки комбинаций частоты / рабочего разрешения, которые ниже поддерживаемого минимума, например:

```
[[E (196) ledc: requested frequency and duty resolution cannot be achieved, try increasing freq_hz or duty_resolution. div_param=128000000]]
```

Разрешение коэффициента заполнения `duty_resolution` обычно устанавливается параметром `ledc_timer_bit_t` в диапазоне от 10 до 15 бит. Для меньшего разрешения (от 10 до 1) просто введите эквивалентное числовое значение напрямую.

6.5. Практика: добавление драйверов в проект Smart Light

В рамках проекта Smart Light необходимо разработать два драйвера – драйвер для кнопок и драйвер для регулировки яркости светодиодов. С помощью этих двух драйверов мы можем использовать кнопку для управления светодиодами.

6.5.1 Драйвер кнопки

При использовании ESP32-C3-DevKitM-1 для имитации интеллектуального освещения при разработке мы можем найти две кнопки на плате, а именно кнопку Boot и кнопку RST. Кнопка RST используется для сброса и перезапуска ESP32-C3, а кнопка Boot, после того как прошивка загрузится, работает как обычная кнопка. Другими словами, кнопка загрузки Boot может использоваться для имитации выключателя света.

Для этой цели мы вводим компонент кнопки в виде драйвера кнопки. Вы можете прочитать исходный код, чтобы узнать о его разработке, так как мы не будем подробно описывать его в этой книге. Чтобы добавить драйвер кнопки в проект умного освещения, выполните следующие действия:

1. Добавление исходных файлов драйверов

Создайте папку с именем *components* в каталоге проекта Smart Light и поместите в нее компоненты, используемые проектом.

Создайте подпапку с именем *button* в разделе *components*.

Затем создайте исходные файлы и файлы заголовков для драйвера кнопки в *button* и соответствующим образом отредактируйте код.

Конкретный код можно найти по адресу https://github.com/espressif/book-es-p32c3-iot-projects/tree/main/device_firmware/components/button.

В главной папке проекта *main* создайте исходный файл с именем *app_driver.c* для обработки всех драйверов. Также создайте заголовочные файлы в папке *include* в разделе *main*. Добавьте в соответствующие файлы операции драйвера и объявления функций, такие как инициализация драйвера и обработка событий кнопки. Код кнопки выглядит следующим образом:

```
1. //Callback function for pressing the button
2. static void push_btn_cb(void *arg)
3. {
4.     //Code Omitted
5. }
6. void app_driver_init()
7. {
8.     //Initializing button driver
9.     button_config_t btn_cfg = {
10.         .type = BUTTON_TYPE_GPIO,
11.         .gpio_button_config = {
12.             .gpio_num = LIGHT_BUTTON_GPIO,
13.             .active_level = LIGHT_BUTTON_ACTIVE_LEVEL,
14.         },
15.     };
16.     button_handle_t btn_handle = iot_button_create(&btn_cfg);
17.     if (btn_handle) {
18.         iot_button_register_cb(btn_handle, BUTTON_PRESS_UP, push_btn_cb);
19.     }
20.     //Code Omitted
21. }
```

2. Добавление исходных файлов в систему компиляции

Сначала отредактируйте файл *CMakeLists.txt* в каталоге проекта. Добавьте путь к компонентам в разделе *components* в соответствии со следующим кодом:

```
1. //Code Omitted
2. set(EXTRA_COMPONENT_DIRS ${CMAKE_CURRENT_LIST_DIR}/../components)
3. //Code Omitted
```

Затем отредактируйте файл *CMakeLists.txt* в папке *main*. Добавьте исходный файл *app_driver.c* в систему компиляции с помощью следующего кода:

```
1. set(srcs "app_main.c"
2.         "app_driver.c")
3. set(include_dirs "include")
4. idf_component_register(SRCS "${srcs}"
5.                       INCLUDE_DIRS "${include_dirs}")
```

Кроме того, компонент кнопки также имеет файл *CMakeLists.txt*, который используется для добавления исходного кода драйвера кнопки в систему компиляции. Вы можете обратиться к https://github.com/esp8266/arduino-esp8266-core/blob/main/device_firmware/components/button/CMakeLists.txt для получения подробной информации.

Другие компоненты, которые будут добавляться в дальнейшем, имеют аналогичную структуру и не будут повторно объясняться в последующих главах.

6.5.2. Драйвер регулировки яркости светодиода

Светодиодные лампы, используемые в нашем проекте, имеют пять цветовых вариантов: красный, зеленый, синий, теплый белый (WW) и холодный белый (CW), поэтому для управления ими нужно пять ШИМ-каналов. Целевые функции включают включение/выключение светодиодных ламп и управление их цветом, цветовой температурой, яркостью, мерцанием и плавным изменением свечения. Однако поскольку на практике для симуляции мы используем ESP32-C3-DevKitM-1, который имеет только каналы R, G и B, мы можем программно менять лишь цвета светодиодов, но не их цветовую температуру.

В соответствии с требованиями проекта интеллектуального освещения драйвер диммирования светодиодов внедрен в модуль и предоставляется в виде компонента `light_driver`, который можно найти по адресу https://github.com/esp8266/arduino-esp8266-core/tree/main/device_firmware/components/light_driver.

Кроме того, для сохранения состояния светодиодных светильников в проект также включается компонент `app_storage`.

На внутреннем уровне он использует NVS-память, хранящую пары ключ-значение в основном разделе флеш-памяти путем вызова API в `esp_partition.h`. Компонент можно найти по адресу https://github.com/esp8266/arduino-esp8266-core/tree/main/device_firmware/components/app_storage.

1. Компонент драйвера источника света

В соответствии с требованиями этого проекта компонент `light_driver` реализует такие функции, как инициализация/деинициализация драйвера затемнения светодиодов, включение/выключение источников света, управление их цветом, яркостью, цветовой температурой и т. д. В табл. 6.1 перечислены API-интерфейсы, предоставляемые для основного приложения драйвером регулировки яркости светодиодов в компоненте `light_driver`.

Таблица 6.1. API-интерфейсы, предоставляемые драйвером диммирования светодиодов в компоненте `light_driver`

API	Функция
<code>light_driver_init()</code>	Инициализация <code>light_driver</code>
<code>light_driver_deinit()</code>	Деинициализация <code>light_driver</code>
<code>light_driver_config()</code>	Настройка плавного изменения и цикла мигания <code>light_driver</code>
<code>light_driver_set_switch()</code>	Включение/выключение светодиода
<code>light_driver_get_switch()</code>	Получить состояние вкл/выкл светодиода

API	Функция
<code>light_driver_set_hue()</code>	Установить цвет
<code>light_driver_get_hue()</code>	Получить установленный цвет
<code>light_driver_set_saturation()</code>	Установить насыщенность
<code>light_driver_get_saturation()</code>	Получить установленное значение насыщенности
<code>light_driver_set_value()</code>	Установить интенсивность (в соответствии с моделью HSV)
<code>light_driver_get_value()</code>	Получить установленное значение интенсивности
<code>light_driver_set_hsv()</code>	Установить три компоненты HSV в один прием
<code>light_driver_get_hsv()</code>	Получить значения трех компонент HSV в один прием
<code>light_driver_set_lightness()</code>	Установить светлоту (в соответствии с моделью HSL)
<code>light_driver_get_lightness()</code>	Получить установленное значение светлоты
<code>light_driver_set_hsl()</code>	Установить три компоненты HSL в один прием
<code>light_driver_get_hsl()</code>	Получить значения трех компонент HSL в один прием
<code>light_driver_set_color_temperature()</code>	Установить цветовую температуру
<code>light_driver_get_color_temperature()</code>	Получить установленное значение цветовой температуры
<code>light_driver_set_brightness()</code>	Установить яркость
<code>light_driver_get_brightness()</code>	Получить установленное значение яркости
<code>light_driver_set_ctb()</code>	Установить цветовую температуру и яркость в один прием
<code>light_driver_get_ctb()</code>	Получить установленные значения цветовой температуры и яркости
<code>light_driver_set_rgb()</code>	Установить компоненты RGB в один прием
<code>light_driver_breath_start()</code>	Установить цвет и запустить мерцание
<code>light_driver_breath_stop()</code>	Остановить мерцание
<code>light_driver_blink_start()</code>	Установить цвет и запустить мигание
<code>light_driver_blink_stop()</code>	Остановить мигание

2. Компонент `app_storage`

Компонент `app_storage` использует энергонезависимое хранилище (NVS) на своем базовом уровне. Таблица 6.2 показывает API, предоставляемые основному приложению компонентом `app_storage`.

Таблица 6.2. API, предоставляемые компонентом `app_storage`

API	Функция
<code>app_storage_init()</code>	Инициализация <code>app_storage</code>
<code>app_storage_set()</code>	Сохранить данные в формате ключ-значение
<code>app_storage_get()</code>	Получить данные
<code>app_storage_erase()</code>	Удалить заданную пару ключ-значение

3. Сохранение состояния светильника

Когда светодиоды включаются, мы можем ожидать, что они будут настроены на цвет и яркость последнего использования. Чтобы реализовать эту функцию, нам нужно сохранять состояние светильника после каждого изменения и загружать последний статус при инициализации драйвера в момент включения. Эта функция сохранения состояния реализована в компоненте `light_driver`. Каждый раз, когда API компонента `light_driver` вызывается для изменения состояния светодиодов, новое состояние будет сохраняться и впоследствии будет загружено при вызове API инициализации.

```
1. //Save LED status
2. if (app_storage_get(LIGHT_STATUS_STORE_KEY, &g_light_status,
3.     sizeof(light_status_t)) != ESP_OK) {
4.     //Code Omitted
5. }
6. //Load LED status
7. if (app_storage_set(LIGHT_STATUS_STORE_KEY, &g_light_status,
8.     sizeof(light_status_t)) != ESP_OK) {
9.     //Code Omitted
10. }
```

4. Инициализация драйвера

При добавлении драйвера регулировки светодиодов в проект умного освещения необходимо написать код для инициализации драйвера в функции `app_main()`. Чтобы использовать такой код, должен быть предоставлен параметр `light_driver_config_t`, который указывает GPIO, используемые пятью каналами ШИМ, время плавного изменения, циклы мерцания, частоту ШИМ, источник синхронизации модуля LEDC, разрешение режима ШИМ и т. д. Код инициализации драйвера в `app_main()` определяется как функция `app_driver_init()` и вызывается следующим образом:

```
1. void app_driver_init()
2. {
3.     //Code Omitted
4.     //Initialize LED dimming driver
5.     light_driver_config_t driver_config = {
6.         .gpio_red= LIGHT_GPIO_RED,
7.         .gpio_green = LIGHT_GPIO_GREEN,
8.         .gpio_blue = LIGHT_GPIO_BLUE,
```

```

9.     .gpio_cold = LIGHT_GPIO_COLD,
10.    .gpio_warm = LIGHT_GPIO_WARM,
11.    .fade_period_ms = LIGHT_FADE_PERIOD_MS,
12.    .blink_period_ms = LIGHT_BLINK_PERIOD_MS,
13.    .freq_hz = LIGHT_FREQ_HZ,
14.    .clk_cfg = LEDC_USE_APB_CLK,
15.    .duty_resolution = LEDC_TIMER_11_BIT,
16.    };
17.    ESP_ERROR_CHECK(light_driver_init(&driver_config));
18.    //Code Omitted
19. }

```

5. Контроль состояния светодиода

После инициализации драйвера регулировки светодиодов мы можем использовать API, предоставляемые компонентом `light_driver`, для управления светодиодными лампами. Наличие драйвера кнопки позволяет включать/выключать светодиодные светильники с ее помощью. Здесь мы сосредоточимся на управлении состоянием включения/выключения, цветом и цветовой температурой.

(1) Состояние включения/выключения

Следующий API можно использовать для управления включением/выключением светодиодных светильников:

```

1. //Turn on the light
2. light_driver_set_switch(true);
3. //Turn off the light
4. light_driver_set_switch(false);

```

(2) Цвет

После инициализации драйвера затемнения светодиодов и включения света цветом светодиодов можно управлять на основе цветовых пространств RGB, HSL или HSV. При использовании API диммирования светодиодов обратите внимание на диапазон значений параметров этих API:

- HSV: оттенок ≤ 360 , насыщенность ≤ 100 , яркость ≤ 100 ;
- HSL: оттенок ≤ 360 , насыщенность ≤ 100 , светлота ≤ 100 ;
- RGB: красный ≤ 255 , зеленый ≤ 255 , синий ≤ 255 .

Следующие API используются для настройки всех трех компонентов цвета RGB, HSL или HSV-моделей одним вызовом функции. Вы также можете настроить только один из компонентов, используя API, перечисленные в табл. 6.1.

```

1. //RGB color control
2. light_driver_set_rgb(uint8_t red, uint8_t green, uint8_t blue);
3. //HSL color control
4. light_driver_set_hsl(uint16_t hue, uint8_t saturation, uint8_t lightness);
5. //HSV color control
6. light_driver_set_hsv(uint16_t hue, uint8_t saturation, uint8_t value);

```

(3) Цветовая температура

В дополнение к управлению статусом включения/выключения и цветом также можно использовать API-интерфейсы `light_driver` для управления цвето-

вой температурой (но это недоступно на симуляторе освещения ESP32-C3-DevKitM-1). Следующий API можно использовать для изменения цветовой температуры и яркости:

```
1. light_driver_set_ctb(uint8_t color_temperature, uint8_t brightness);
```

Исходный код

Пожалуйста, обратитесь к полному коду по адресу https://github.com/espressif/book-esp32c3-iot-projects/tree/main/device_firmware/2_light_drivers для добавления драйвера затемнения светодиодов и драйвера кнопок в проект умного освещения. Вы также можете проверить результаты работы после компиляции кода и прошивки его на макетной плате.

6.6. Резюме

ESP32-C3 обладает богатым набором периферийных интерфейсов, которые могут использоваться в различных сценариях применения, таких как отображение на экране, сбор данных с датчиков, воспроизведение аудио и т. д. Для приложений интеллектуального освещения обычно используются светодиодные ШИМ-контроллеры для реализации драйверов регулировки светодиодов.

Изучив этапы разработки драйвера в этой главе, вы сможете самостоятельно запрограммировать управление светодиодными лампами. В последующих главах мы познакомим вас с широко используемыми технологиями и протоколами беспроводной связи в IoT и с тем, как их применять в проекте Smart Light.

Часть III

Беспроводная связь и управление

В приложениях интернета вещей объекты подключаются через сеть. Пользователи могут получать доступ к IoT-устройствам и управлять ими удаленно через интернет или локально с помощью локальной сети. Это помогает пользователям контролировать устройства и обновлять встроенное ПО. В настоящее время Wi-Fi и Bluetooth являются двумя основными подходами к сетевому подключению, в то время как новые облачные платформы предоставляют общие функции, необходимые для IoT-приложений. В этой части мы покажем вам, как добавить сетевое подключение и облачное управление в ваш IoT-проект.

ГЛАВА 7

Настройка связи Wi-Fi

ГЛАВА 8

Локальное управление

ГЛАВА 9

Управление через облако

ГЛАВА 10

Разработка приложений для смартфонов

ГЛАВА 11

Обновление встроенного ПО и управление версиями



Глава 7

Настройка Wi-Fi-соединения

В этой главе мы сосредоточимся на технических характеристиках конфигурации Wi-Fi-сети и Wi-Fi-подключения, начиная с основ Wi-Fi и Bluetooth и заканчивая распространенными методами настройки Wi-Fi. Затем будут приведены примеры, которые помогут вам лучше понять механизм работы сети и способы интеллектуальной настройки Wi-Fi. Наконец, мы познакомим вас с практикой настройки Wi-Fi с помощью проекта Smart Light на основе ESP 32-C3.

Поскольку технология беспроводной связи имеет долгую историю, множество документации позволяет глубоко вникнуть в эту тему. Поэтому в этой главе вы найдете лишь краткое введение. Для получения более подробной информации, пожалуйста, ознакомьтесь со списком литературы, приведенным в конце данной книги.

7.1. Основы Wi-Fi

В этом разделе вы познакомитесь с со следующими аспектами технологии Wi-Fi:

- что такое Wi-Fi;
- как развивается Wi-Fi;
- что означают концепции Wi-Fi;
- как работает соединение Wi-Fi.

7.1.1. Введение в Wi-Fi

Wi-Fi – это торговая марка технологии беспроводной связи, принадлежащая Wi-Fi Alliance (WECA²⁴), контролирующему сертификацию Wi-Fi. Wi-Fi – это семейство протоколов беспроводной сети, основанных на стандарте IEEE 802.11. Продуктам, прошедшим тщательные испытания, будет присвоен сертификат Wi-Fi CERTIFIED™, подтверждающий, что они соответствуют согласованным в отрасли стандартам функциональной совместимости, безопасности и целому ряду протоколов, специфичных для конкретного применения.

²⁴ Wireless Ethernet Compatibility Alliance – альянс совместимости с беспроводным Ethernet.

По сравнению с другими технологиями беспроводной связи Wi-Fi может похвастаться более широким охватом, лучшей способностью проникать сквозь стены²⁵ и более высокой пропускной способностью.

7.1.2. Эволюция IEEE 802.11

Как пользователи Wi-Fi вы наверняка должны были слышать о стандарте IEEE 802.11. Но что это именно значит?

IEEE – это аббревиатура от названия Institute of Electrical and Electronics Engineers (*Институт инженеров электротехники и электроники*). 802 – это комитет института сетевых стандартов, еще известный как LMSC – Комитет по стандартам LAN/MAN. Комитет охватывает такое большое семейство стандартов, что его необходимо разделить на группы, посвященные конкретным областям. Каждая группа имеет свой собственный номер (следующий за «802», отделенный точкой), поэтому 802.11 относится к 11-й группе комитета 802, который разрабатывает технические характеристики протоколов управления доступом к среде (MAC) и физический уровень (PHY) беспроводных локальных вычислительных сетей (WLAN). В стандарт IEEE 802.11 несколько раз вносились изменения, как показано в табл. 7.1.

Таблица 7.1. Поколения IEEE 802.11²⁶

Версия	Частота (ГГц)	Ширина канала (МГц)	Скорость (Мбит/с)	Метод модуляции	Псевдоним
IEEE 802.11a	5	20	54	OFDM	–
IEEE 802.11b	2,4	22	11	CCK/DSSS	–
IEEE 802.11g	2,4	20	54	OFDM	–
IEEE 802.11n	2,4, 5	20, 40	72–600 MIMO 4×4	OFDM	Wi-Fi 4
IEEE 802.11ac	5	20, 40, 80, 80+80, 160	433-1733 MIMO 4×4	OFDM	Wi-Fi 5
IEEE 802.11ax	2,4, 5	20, 40, 80, 80+80, 160	600-2401 (MIMO:4×4)	OFDMA	Wi-Fi 6

²⁵ Проникающая способность через преграды в первую очередь зависит от длины волны и материала препятствия, а также, естественно, мощности передатчика и чувствительности приемника. Сигнал Wi-Fi (стандарты 802.11b/g/n, частота 2,4 ГГц) практически полностью потеряется уже на 10 см железобетонной перегородки; для 802.11n на частоте 5 ГГц непреодолимым препятствием будет даже кирпичная/гипсолитовая перегородка или тройной стеклопакет. Wi-Fi выигрывает в сравнении с Bluetooth, работающим на той же частоте, за счет более мощных передатчиков (ведь Bluetooth и рассчитан на расстояние в единицы метров, см. далее в этой главе). Однако в сравнении, например, с сотовой связью GSM (частоты 0,9–2 ГГц) Wi-Fi решительно проигрывает из-за меньших разрешенных мощностей передатчиков. Поэтому надо иметь в виду, что при наличии толстых стен и на существенных расстояниях (более десятка-другого метров) предпочтительные беспроводные технологии связи для IoT-компонентов – те, что используют более низкие частоты (433/868 МГц), например LoRa или обычные RF-приемопередатчики; в этой книге они не рассматриваются.

²⁶ Следует отметить, что в табл. 7.1 приведены не все существующие разновидности IEEE 802.11.

Пояснения к таб. 7.1:

- MIMO: Multiple Input Multiple Output («много входов – много выходов»);
- OFDM: Orthogonal Frequency Division Multiplexing («мультиплексирование с ортогональным частотным разделением»);
- CCK: Complementary Code Keying («кодирование комплементарным кодом»);
- DSSS: Direct Sequence Spread Spectrum («расширение спектра методом прямой последовательности»);
- OFDMA: Orthogonal Frequency Division Multiple Access («множественный доступ с ортогональным частотным разделением каналов»).

7.1.3. Концепции Wi-Fi

В этом разделе мы представим сетевые технологии, связанные с IEEE 802.11, включая эталонную модель взаимодействия открытых систем (OSI/RM) и физические компоненты IEEE 802.11.

Эталонная модель взаимодействия открытых систем (Open System Interconnection Reference Model, OSI/RM)

В модели OSI/RM компьютерная сетевая система задумана как семиуровневая структура. Их названия и взаимосвязи показаны на рис. 7.1.

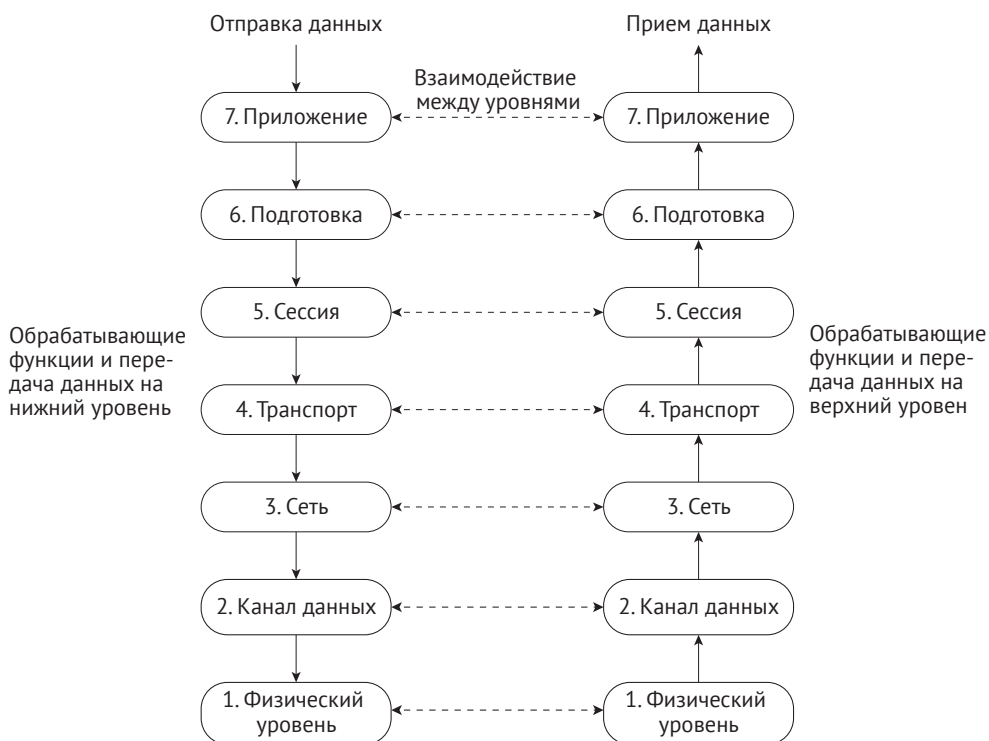


Рисунок 7.1. Архитектура OSI/RM

Физические компоненты IEEE 802.11

Архитектура IEEE 802.11 состоит из четырех основных физических компонентов, показанных на рис. 7.2.

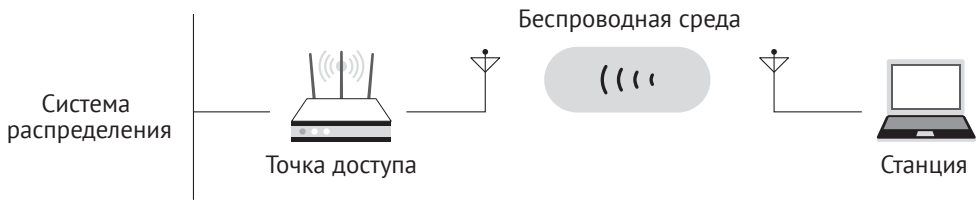


Рисунок 7.2. Физические компоненты IEEE 802.11

Беспроводная среда (Wireless Medium, WM)

WM относится к физическому уровню, на котором передаются MAC-кадры данных. Изначально были введены два физических уровня радиочастот (RF) и один физический уровень инфракрасного излучения (IR), но уровни RF оказались более популярными.

Станции (STA)

Станции включают в себя все устройства и оборудование, подключенные к беспроводной локальной сети. Ноутбуки и карманные компьютеры с батарейным питанием являются типичными STA, а «портативные» – необязательно. В некоторых случаях настольные компьютеры также подключаются к беспроводным локальным сетям, чтобы избежать прокладки кабелей.

Точки доступа (AP)

Как ветвь STA, AP предоставляет доступ к службам распространения для STA, связанных между собой.

Система распределения (DS)

Когда несколько точек доступа подключены для покрытия большей территории, они должны взаимодействовать друг с другом для отслеживания перемещений мобильных станций. Это происходит через систему распределения, т. е. внешнюю сеть передачи данных. Она отвечает за передачу кадров данных в места назначения.

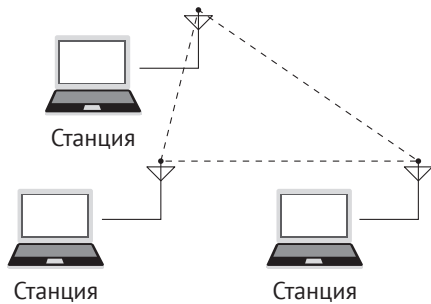
Построение беспроводных сетей

Вышеуказанные физические компоненты составляют беспроводную сеть. Основной строительный блок сети IEEE 802.11 – это базовый набор услуг (Basic Service Set, BSS). Он бывает двух категорий: независимый BSS и инфраструктурный BSS, как показано на рис. 7.3.

Независимый BSS

Станции в независимой системе BSS взаимодействуют друг с другом напрямую, без промежуточных точек доступа.

Независимый BSS



Инфраструктурный BSS

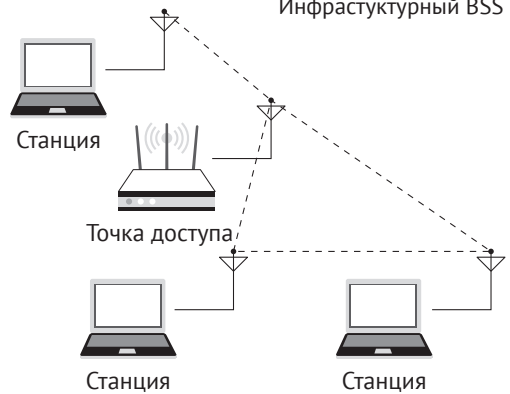


Рисунок 7.3. Независимые BSS и инфраструктурные BSS

Инфраструктурный BSS

В инфраструктурном BSS станции должны подключаться к точке доступа для получения сетевых услуг. Точки доступа функционируют как центр управления комплексом. Каждая станция должна пройти авторизацию и подключение, прежде чем присоединиться к определенному BSS. Инфраструктурный BSS – это наиболее распространенная сетевая архитектура.

Сетевые архитектуры сопровождаются идентификационными данными:

Идентификация BSS (BSSID)

Каждый BSS имеет физический адрес для идентификации, называемый BSSID. Для инфраструктурного BSS BSSID – это MAC-адрес точки доступа. Он поставляется с заводским значением по умолчанию и может быть изменен в соответствии с фиксированным форматом именования.

Идентификатор набора услуг (SSID)

Каждая точка доступа имеет идентификатор для идентификации пользователя. В большинстве случаев один BSSID связан с одним SSID. Обычно это читаемая строка, которую мы называем «имя Wi-Fi-сети».

7.1.4. Wi-Fi-соединение

STA сначала ищет близлежащие беспроводные сети с помощью активного или пассивного сканирования, а затем устанавливает соединение с AP после аутентификации и подключения, наконец, получает доступ к беспроводной локальной сети. На рис. 7.4 показан процесс подключения к сети Wi-Fi.

1. Сканирование

STA может активно или пассивно сканировать беспроводные сети.

Пассивное сканирование

Пассивное сканирование относится к обнаружению близлежащих беспроводных сетей посредством мониторинга сигнальных сообщений, периодически отправляемых точкой доступа. Это рекомендуется, когда пользователям необходимо экономить электроэнергию.

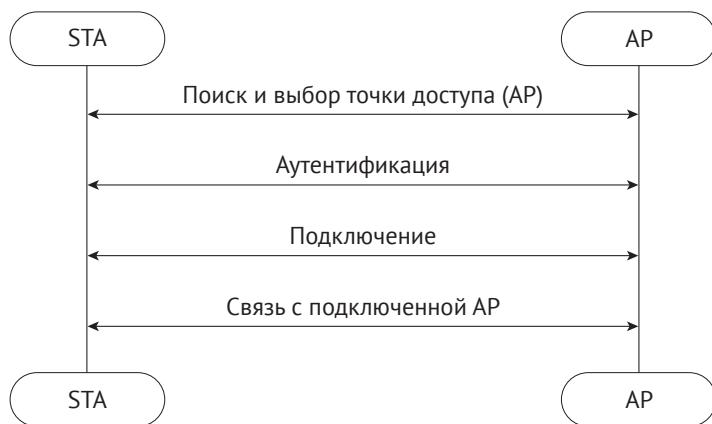


Рисунок 7.4. Процесс подключения к Wi-Fi

Активное сканирование

Во время активного сканирования STA активно отправляет запросы на проверку и получает ответы от точки доступа. Далее этот способ делится на два режима, основанных на использовании SSID.

Активное сканирование без SSID

STA периодически отправляет пробные запросы по поддерживаемым каналам для поиска беспроводных сетей. Приложения, получившие запрос на проверку, будут возвращать ответы, содержащие информацию о доступных беспроводных сетях. Это позволяет STA получать все доступные услуги беспроводной связи поблизости.

Активное сканирование с определенными SSID

Если STA необходимо настроить беспроводную сеть для подключения или станция ранее обращалась к беспроводной сети, она будет периодически отправлять пробные запросы с информацией о конфигурации или SSID беспроводной сети, к которой осуществляется доступ. Когда точка доступа с определенным SSID получит запрос, он вернет пробный ответ. Таким образом, STA может получить доступ к указанной беспроводной сети.

Для скрытых точек доступа рекомендуется активное сканирование с использованием определенного SSID.

2. Аутентификация

Когда STA найдет доступную беспроводную сеть, она выберет одну из точек доступа с соответствующим SSID в соответствии с определенной стратегией подключения, такой как выбор устройства с самым сильным сигналом или с соответствующим MAC-адресом. Следующий шаг – аутентификация. Существует открытая аутентификация и неоткрытая аутентификация.

Открытая аутентификация

По сути, открытая аутентификация не требует аутентификации или шифрования. Любая станция может получить доступ к сети. AP не проверяет идентификатор станции в этом процессе, как показано на рис. 7.5.

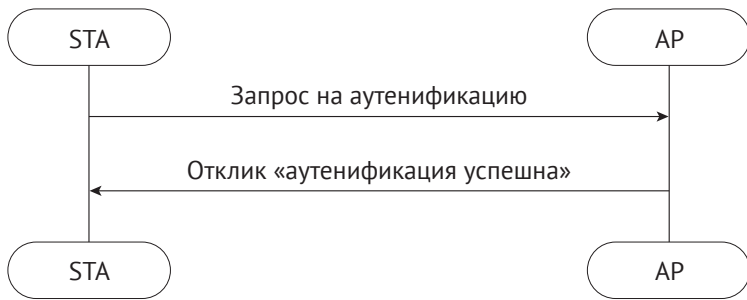


Рисунок 7.5. Процесс открытой аутентификации

STA отправляет запрос на аутентификацию, а AP возвращает результат. Если результат успешен, аутентификация завершена.

Неоткрытая аутентификация

Неоткрытая аутентификация включает в себя аутентификацию с использованием открытого ключа защищенного доступа Wi-Fi (WPA) и надежной сетевой безопасности (RSN).

Открытый ключ

Аутентификация с использованием открытого ключа основана на методе Wired Equivalent Privacy (WEP). Это базовая технология шифрования с ограниченной безопасностью.

STA и AP могут интерпретировать данные, передаваемые между собой, только тогда, когда они настроены на один и тот же ключ. Есть 64-битные ключи и 128-битные ключи. Пользователи могут настроить до четырех групп различных клавиш. На рис. 7.6 показан процесс использования открытого ключа для аутентификации.

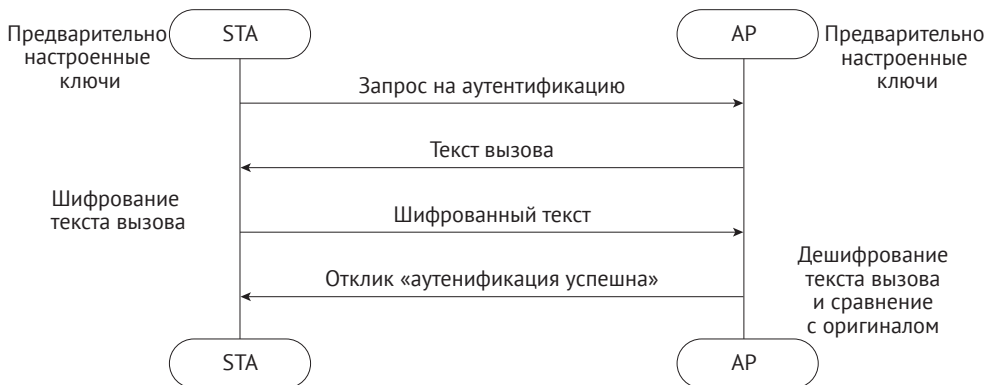


Рисунок 7.6. Процесс аутентификации с открытым ключом

STA отправляет запрос аутентификации на AP. Затем AP генерирует текст вызова и отправляет его на STA. STA использует предварительно настроенные ключи для шифрования текста и отправляет обратно. Точка доступа

применяет предварительно настроенные ключи для расшифровки текста и сравнения его с исходным текстом. Если два текста идентичны, то аутентификация завершена.

Защищенный доступ Wi-Fi (WPA)

WPA – это промежуточное решение для замены WEP перед официальным выпуском IEEE.802.11i. Оно использует новый алгоритм проверки целостности сообщений (MIC) для замены CRC-алгоритма в WEP. Оно также использует протокол целостности временного ключа (TKIP) для создания разных ключей для разных кадров MAC. TKIP является переходным протоколом шифрования, и его защищенность оказалась невысокой.

Надежная сеть безопасности (RSN)

RSN также называется WPA2. Она принимает новый метод шифрования, режим счетчика с протоколом CBC-MAC (CCMP), блочным протоколом безопасности, основанным на расширенном стандарте шифрования (AES). Подробнее об этом – ниже в рассказе об авторизации.

Защищенный доступ Wi-Fi 3 (WPA3)

Хотя WPA2 в определенной степени делает сети Wi-Fi более защищенными, продолжают появляться новые уязвимости безопасности, такие как автономные атаки по словарю, атаки методом перебора и атаки переустановки (KRACK). По этой причине WFA в 2018 году выпустила WPA3 – новую версию протокола шифрования Wi-Fi, который снижает риски WPA2 и обеспечивает новые возможности. По сравнению с WPA2, WPA3 имеет следующие преимущества:

- использование шифрования AES является обязательным вместо TKIP;
- защита управляющих сообщений;
- более безопасный метод, одновременная аутентификация равных (Simultaneous Authentication of Equals, SAE), используется для замены аутентификации с предварительно настроенным открытым ключом (PSK) в WPA2. Во-первых, SAE отказывает службам STA, которые неоднократно пытаются подключиться к точке доступа, предотвращая атаки методом перебора или взлом пароля. Во-вторых, метод гарантирует, что ключ будет меняться часто и автоматически, так что даже в случае взлома самого последнего ключа будет раскрыт лишь минимальный объем данных. Наконец, SAE рассматривает устройства как одноранговые. Любая из сторон может инициировать соединение и отправить аутентификационную информацию независимо, отменяя процесс обмена сообщениями и тем самым не оставляя возможности для взлома;
- 192-разрядный пакет безопасности используется для усиления защиты паролем;
- алгоритм HMAC-SHA-384 используется для экспорта и подтверждения ключей на этапе четырехстороннего подтверждения;
- протокол режима счетчика Galois-Counter Mode Protocol-256 (GCMP-256) применяется для защиты беспроводного трафика, после того как станция выходит в интернет;
- код аутентификации сообщения Galois-256 (GMAC-256) GCMP используется для защиты многоадресных управляющих сообщений;

- WPA3 вводит расширенный режим открытой аутентификации Wi-Fi – Opportunistic Wireless Encryption (OWE), который позволяет подключаться без пароля, сохраняя при этом упрощенный доступ к открытым сетям. Он применяет алгоритм обмена ключами Диффи–Хеллмана для шифрования данных в сети Wi-Fi, тем самым защищая обмен данными между STA и сетью Wi-Fi.

3. Подключение

После того как AP вернет STA успешный результат аутентификации, следующим шагом будет подключение для получения полного доступа к сети (рис. 7.7).

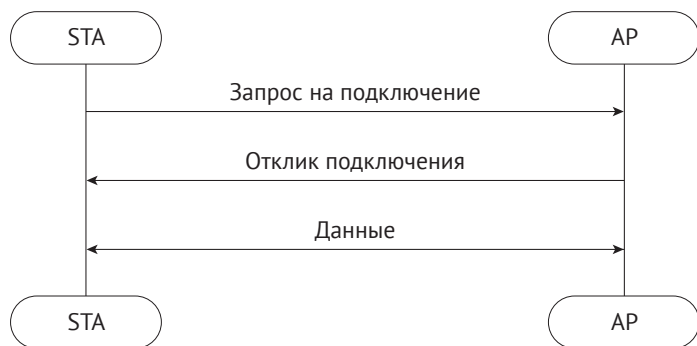


Рисунок 7.7. Процесс подключения

4. Авторизация

После сканирования, аутентификации и подключения давайте сосредоточимся на последнем шаге – авторизации. В этом разделе мы представим расширяемый протокол аутентификации (Extensible Authentication Protocol, EAP) и соглашение о ключах – протокол четырехстороннего взаимного подтверждения.

Расширяемый протокол аутентификации (EAP)

EAP – это самый простой протокол безопасности для проверки личности, который является не просто протоколом, а целой структурой. Основываясь на этой структуре протокола, хорошо поддерживаются различные методы проверки подлинности. Заявители отправляют запросы на аутентификацию через протокол EAP поверх локальной сети (EAP over LAN, EAPOL) и получают разрешение на однократное использование сети, если проверка проходит успешно. На рис. 7.8 показана архитектура EAP.

Эта книга затрагивает только основы EAP. Чтобы узнать больше, обратитесь к RFC 3748.

- **Заявитель:** объект, который инициирует запрос аутентификации. Для беспроводных сетей STA является заявителем.
- **Аутентификатор:** объект, который отвечает на запрос аутентификации. Для беспроводных сетей аутентификатором является точка доступа.
- **Внутренний сервер аутентификации (BAS).** В некоторых случаях, например в корпоративных приложениях, аутентификатор не занимается авторизацией напрямую. Вместо этого он отправляет запрос аутентификации в BAS. Таким образом EAP расширяет диапазон своего применения.

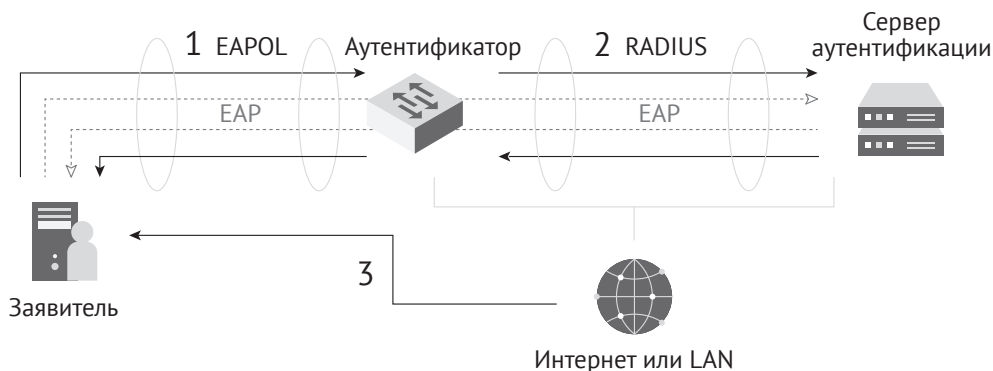


Рисунок 7.8. Архитектура EAP

- **Аутентификация, авторизация и учет (AAA):** еще один протокол на основе EAP.
- Сущностью, реализующей этот протокол, является определенный тип BAS, например сервер RADIUS.
- **Сервер EAP:** это то, что фактически обрабатывает авторизацию. Если BAS нет, роль сервера EAP играет аутентификатор, в противном случае для этой цели употребляется BAS.

Соглашение о ключах

Ассоциация надежных защищенных сетей (Robust Secure Network Association, RSNA) установила набор процедур, определенных в стандарте IEEE 802.11 для обеспечения безопасности беспроводной сети. Набор состоит из шифрования данных и проверки целостности. RSNA использует вышеупомянутые TKIP и CCMP. Временный ключ (ТК), применяемый в TKIP и CCMP, берется из функции получения ключа, определенной RSNA. Основываясь на стандарте IEEE 802.1X, RSNA также определяет четырехстороннее подтверждение при генерации ключей для одноадресного шифрования данных и групповое подтверждение ключей для многоадресного шифрования данных.

Но зачем нам нужно извлекать ключи? В режиме шифрования WEP все STA используют один и тот же ключ WEP для шифрования, что приводит к снижению уровня безопасности. В то время как RSNA требует, чтобы разные STA использовали разные ключи для шифрования после подключения к точкам доступа. Означает ли это, что точка доступа должна устанавливать разные пароли для разных STA? Очевидно, что ответ: нет, не должна. В реальной жизни мы используем один и тот же пароль, чтобы связать разные STA с одной и той же точкой доступа.

Тогда как разные STA могут использовать разные пароли? Пароль, который мы устанавливаем в STA, называется парным мастер-ключом (ПМК). Он генерируется из PSK, а именно из пароля, установленного в беспроводном маршрутизаторе дома. Пароль устанавливается без какого-либо сервера аутентификации, и соответствующей настройкой является WPA/WPA2-PSK. На рис. 7.9 показано, как ПМК генерируется из PSK.

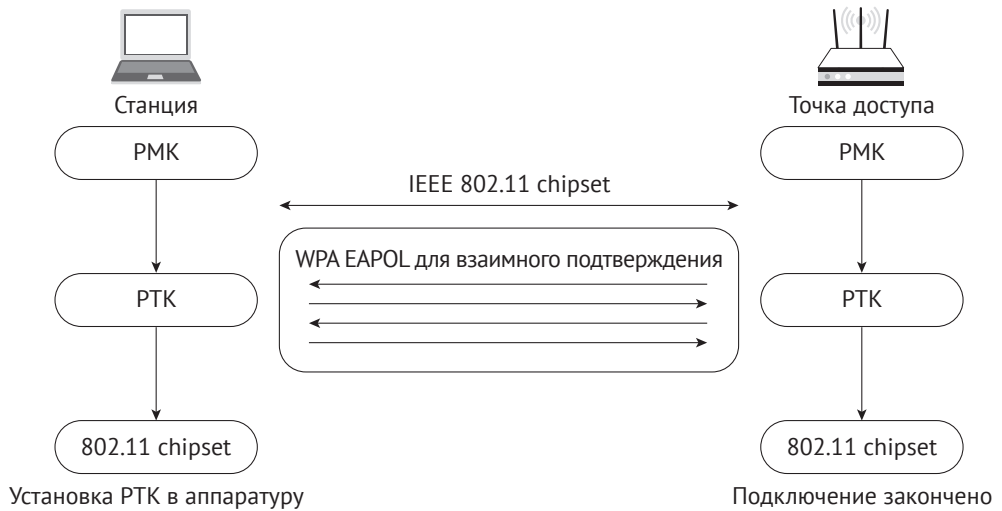


Рисунок 7.9. Генерация PMK из PSK

В WPA2-PSK PMK идентичен PSK. Но в WPA3 генерируется новый PMK с помощью метода SAE, для гарантии, что каждая STA имеет уникальный PMK на разных этапах. На рис. 7.10 показано, как генерируется PMK с помощью SAE.

SAE рассматривает заявителя и аутентификатора как одноранговые сущности. Любой из них может инициировать аутентификацию. Две стороны обмениваются данными друг с другом, чтобы доказать свои знания ключа и сгенерировать PMK. SAE включает две фазы: Commit (заверения) и Confirm (подтверждения). На этапе заверения обе стороны отправляют кадры заверения SAE для определения PSK. Затем на этапе подтверждения они отправляют кадры подтверждения SAE для проверки PSK. После успешной проверки PMK будет сгенерирован, и подключение продолжится.

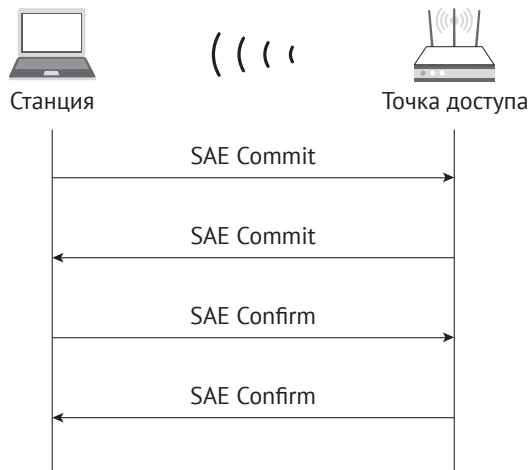


Рисунок 7.10. Генерация PMK с помощью SAE

- **Заверение (Commit)**

Отправитель использует специальный алгоритм Hunting and Pecking для генерации элемента пароля (PWE) на основе PSK и MAC-адресов отправителя и получателя. Затем скалярное целое число и координаты элемента генерируются отправителем на основе PWE и случайного значения с помощью операций на эллиптических кривых. После получения кадра заверения получатель проверяет кадр и использует как локальное известное значение, так и полученное содержимое кадра для генерации ключа подтверждения (КСК) и РМК. Сгенерированный КСК проверит содержимое сообщения на этапе подтверждения.

- **Подтверждение (Confirm)**

Обе стороны генерируют проверочный код из КСК, локальных и одноранговых скаляров, а также локальных и одноранговых элементов пароля PWE через один и тот же алгоритм хеширования. Если коды окажутся идентичными, проверка считается пройденной.

После того как STA и AP получают РМК, они приступят к генерации ключей. Во время этого процесса AP и разные STA генерируют разные ключи, которые настраиваются на аппаратное обеспечение для шифрования/дешифрования. Поскольку AP и STA должны повторно генерировать эти ключи каждый раз, когда они связываются, они называются парными временными ключами (РТК). При проведении четырехстороннего подтверждения AP и STA используют кадр EAPOL-Key для обмена одноразовыми и другими сообщениями. Этот процесс показан на рис. 7.11.

- (1) Аутентификатор генерирует одноразовый номер (ANonce) и отправляет его заявителю через EAPOL-Key (сообщение 1).
- (2) Заявитель выполняет формирование ключа на основе ANonce, своего собственного одноразового номера (SNonce) и РМК, а также MAC-адреса аутентификатора. Затем он отправляет аутентификатору другое сообщение EAPOL-Key (сообщение 2), содержащее SNonce. Сообщение 2 также содержит значение для проверки целостности MIC, зашифрованное КСК. Как только аутентификатор получает SNonce в сообщении 2, он выполняет вычисления, идентичные заявителю, чтобы проверить, является ли возвращенное сообщение корректным. Ошибка означает, что РМК заявителя неверен, подтверждение будет прекращено.
- (3) Если РМК заявителя верен, аутентификатор также выполнит вывод ключа. После этого аутентификатор отправляет третье сообщение EAPOL-Key (сообщение 3) заявителю. Это сообщение содержит групповой временный ключ (GTK, используется для обновления группового ключа, зашифрованного КЕК) и значение проверки целостности MIC (зашифрованное КСК). Когда заявитель получит сообщение 3, он с помощью вычислений проверит, является ли РМК точки доступа корректным.
- (4) Заявитель отправляет последнее сообщение EAPOL-Key (сообщение 4) аутентификатору для подтверждения. Затем обе стороны будут использовать полученный ключ для шифрования данных.

На данный момент заявитель и аутентификатор завершили вывод ключа и подключение и смогут общаться друг с другом.

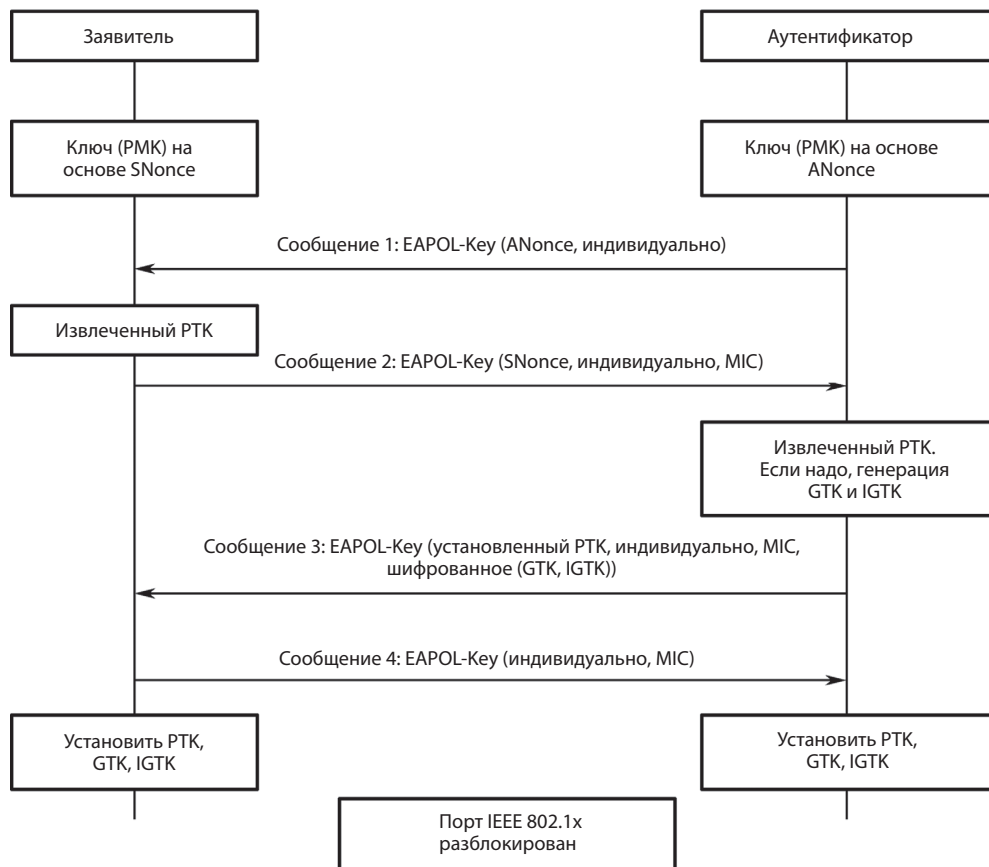


Рисунок 7.11. Процесс четырехстороннего подтверждения

7.2. Основы Bluetooth

В этом разделе будут представлены следующие аспекты Bluetooth:

- что такое Bluetooth;
- как развивается Bluetooth;
- что означают концепции Bluetooth;
- как работает соединение Bluetooth.

7.2.1. Введение в Bluetooth

Bluetooth – это технология беспроводной связи малого радиуса действия, первоначально разработанная компанией Ericsson в 1994 году. Целью Bluetooth является облегчение передачи и совместного использования информации на коротких расстояниях без кабельных соединений между мобильными устройствами, встроенными устройствами, компьютерной периферией и бытовой техникой. По сравнению с другими технологиями беспроводной связи Bluetooth может похвастаться высокой безопасностью и простотой подключения.

Примечание: почему «Bluetooth»?

Слово «Bluetooth» («синий зуб») появилось более тысячи лет назад благодаря датскому королю Харальду Синезубому. Королю Харальду приписывают первое объединение Скандинавии. Легенда гласит, что король Харальд так любил чернику, что его зубы все время были окрашены в синий цвет. Поэтому его прозвали Синезубым. В 1998 году Intel, Nokia, Ericsson и IBM организовали группу особого интереса (Special Interest Group, SIG), названную Bluetooth. Слово Bluetooth быстро завоевало популярность и стало синонимом технологии беспроводной связи малого радиуса действия.

Bluetooth использует децентрализованную сетевую структуру, быструю скачкообразную перестройку частоты и технологию коротких пакетов с поддержкой связи типа «точка–точка» и «точка – много точек». Он работает в диапазоне ISM (промышленный, научный, медицинский) 2,4 ГГц, который обычно открыт для свободного использования во всем мире. Технологию Bluetooth можно разделить на две категории: классический Bluetooth Classic и экономичный Bluetooth Low Energy (Bluetooth LE, BLE).

Bluetooth Classic

Bluetooth Classic (BR/EDR) обычно относится к модулям, поддерживающим протокол Bluetooth ниже версии 4.0 и используемым для передачи больших объемов данных, таких как голос и музыка. Протоколы Bluetooth Classic имеют разные профили персональных сетей для разных сценариев. Обычно используемые профили: Advanced Audio Distribution (A2DP) для аудиосвязи, профиль громкой связи/гарнитуры (HFP/HSP) для устройств громкой связи, профиль последовательного порта (SPP) для прозрачной передачи текста через последовательный порт и устройство интерфейса пользователя (HID) для беспроводных устройств ввода/вывода.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) – это новый тип технологии беспроводной связи со сверхнизким энергопотреблением, предназначенный для недорогих и менее сложных беспроводных телесетей и персональных компьютеров. Стоит отметить, что чипы Bluetooth LE могут питаться от элементов-«таблеток». Вместе с микросенсорами вы можете использовать эти чипы для создания встроенных или носимых датчиков и приложений сенсорных сетей.

Что касается протоколов, Bluetooth 1.1, 1.2, 2.0, 2.1 и 3.0 применимы к Bluetooth Classic, Bluetooth 4.0 поддерживает как Bluetooth Classic, так и Bluetooth LE, а все более поздние версии используют Bluetooth LE.

7.2.2. Концепции Bluetooth

В этом разделе представлены сетевые технологии, связанные с Bluetooth, включая основную архитектуру и компоненты в спецификациях Bluetooth.

Базовая архитектура

На базовом уровне Bluetooth состоит из хоста, контроллера и интерфейса хост–контроллер (Host Controller Interface, HCI). Хост используется для разработки

приложений, контроллер – для отправки и получения сообщений, управления физическим подключением и другими основными функциями, которые реализуются специализированными производителями чипов Bluetooth.

Первоначально концепция дизайна заключалась в том, чтобы запускать хост и контроллер независимо друг от друга на двух микросхемах или даже системах, и они могли бы общаться через HCI. При таком способе легче заменить и обновить любой модуль. Хотя появилось много чипов, объединяющих хост и контроллер вместе, они по-прежнему следуют этой архитектуре, за исключением того, что HCI изменен с аппаратного порта связи на программный.

Стек протоколов Bluetooth LE включает физический уровень (PHY), канальный уровень (LL), логический протокол управления каналом и адаптации (L2CAP), протокол атрибутов (ATT), протокол диспетчера безопасности (SMP), общий профиль атрибутов (GATT) и общий профиль доступа (GAP), показанные на рис. 7.12.

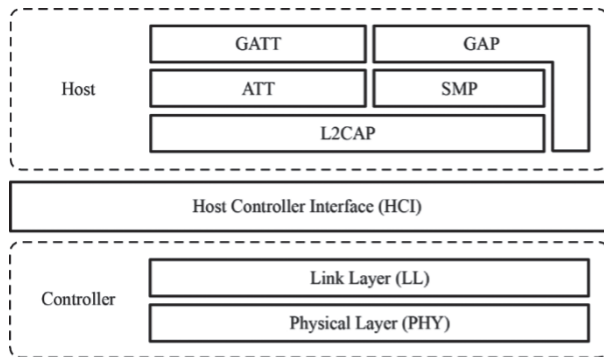


Рисунок 7.12. Уровни стека протоколов Bluetooth LE

- **Физический уровень** (Physical Layer, PHY) определяет полосу частот беспроводной сети и режим модуляции, используемые Bluetooth LE. То, как работает PHY, определяет энергопотребление, чувствительность и избирательность чипа Bluetooth LE и влияет на другие радиочастотные устройства.
- **Канальный уровень** (Link Layer, LL) только отправляет или получает данные, оставляя анализ данных протоколам GAP или ATT на вышележащих уровнях. LL лежит в основе стека протоколов Bluetooth LE, так как он решает, какой радиочастотный канал выбрать для связи, как идентифицировать пакеты данных, передающиеся по воздуху, когда отправлять пакеты данных, как обеспечить целостность данных, как управлять ссылками и контролировать их, как получать и ретранслировать подтверждения (ACK) и т. д.
- **Интерфейс хост–контроллер** (Host Controller Interface, HCI) обеспечивает средства связи между хостом и контроллером. Этот уровень может быть реализован либо с помощью аппаратного интерфейса, например UART или USB в двухчиповых архитектурах, либо через программный API в одночиповых.
- **Логический протокол управления каналом и адаптации** (Logical Link Control and Adaptation Protocol, L2CAP) обеспечивает службы данных при

установленном и неустановленном соединении для протоколов верхнего уровня с возможностью мультиплексирования протоколов, сегментации и повторной сборки, а также групповых абстракций. Он также разрешает управление потоком и повтор передачи.

- **Протокол атрибутов** (Attribute Protocol, ATT) определяет данные для пользовательских команд и командных операций, таких как чтение или запись определенных данных. Bluetooth LE вводит понятие атрибутов, используемых для детального описания данных. Кроме того, ATT также определяет ATT-команды, которые могут использоваться данными. Это слой, с которым вы будете иметь дело чаще всего.
- **Протокол диспетчера безопасности** (Security Manager Protocol, SMP) отвечает за шифрование и безопасность соединений Bluetooth LE, не влияющие на работу пользователя.
- **Общий профиль атрибутов** (Generic Attribute Profile, GATT) стандартизирует содержание данных в атрибутах и использует концепцию групп для классификации и управления атрибутами. Хотя стек протокола BLE может работать без GATT, его способность к взаимодействию будет нарушена. GATT и богатый набор профилей избавляют Bluetooth LE от проблемы совместимости, с которой сталкиваются другие беспроводные протоколы, такие как ZigBee.
- **Общий профиль доступа** (Generic Access Profile, GAP) определяет эффективные пакеты данных в LL, предлагая самый простой способ анализа информационно-формационной части LL. Он ограничен такими функциями, как вещание, сканирование и инициирование соединений.

Режимы Bluetooth

Bluetooth может быть распространителем оповещений (advertiser), сканером или инициатором. Ведущее устройство играет роль инициатора и сканера, в то время как ведомое устройство играет роль распространителя оповещений.

Связь по Bluetooth относится к связи между двумя или более устройствами. Это происходит только между ведущими и ведомыми устройствами, поскольку ведомые устройства не могут взаимодействовать напрямую.

Режим ведущего

Ведущие (или «центральные») устройства сканируют ведомые устройства и иницируют подключение. Теоретически, если технология Bluetooth LE Mesh не используется для обеспечения связи устройств «многие ко многим», устройства могут быть объединены только в пикосети (см. далее).

Устройство с технологией Bluetooth может переключаться между ведущим и ведомым режимами.

Обычно устройство работает в ведомом режиме и ожидает подключения ведущих устройств. При необходимости оно переключается в ведущий режим и вызывает другие устройства. Чтобы инициировать вызов в ведущем режиме, устройству необходимо знать Bluetooth-адрес и пароль подключения другого устройства и начинать обмен после успешного соединения.

Режим ведомого

Ведомые (или «периферийные») устройства рассылают оповещения и ждут подключения. После подключения ведомые устройства могут обмениваться данными с ведущим.

Таким образом, ведущий может искать ведомых и активно с ними соединяться, в то время как ведомый не может инициировать какое-либо соединение самостоятельно, но ожидает подключения.

Создание Bluetooth-сетей

Теперь, когда мы узнали о ведущем и ведомом устройствах, давайте посмотрим, как создать Bluetooth-сеть. По топологическим структурам сети Bluetooth можно разделить на пикосети (piconet), распределенные (scatternet) и ячеистые (mesh) сети.

Пикосеть

Каждый раз, когда формируется беспроводная связь Bluetooth, она образует пикосеть. Пикосеть состоит из двух или более устройств, занимающих один и тот же физический канал, что означает, что устройства синхронизируются в соответствии с общими часами и последовательностью скачкообразной перестройки частоты. На рис. 7.13 показана топология пикосети.

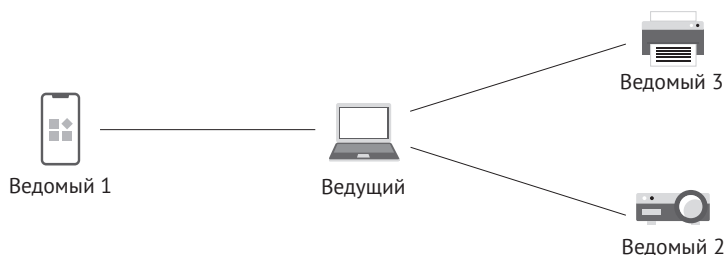


Рисунок 7.13. Топология пикосети

Распределенная сеть (Scatternet)

Распределенная сеть формируется, когда несколько пикосетей перекрываются. На рис. 7.14 показана топология распределенной сети. Каждая пикосеть, составляющая распределенную сеть, поддерживает своего собственного ведущего. Ведущий одной пикосети может одновременно действовать как ведомый другой пикосети. На рис. 7.14 мобильный телефон является ведущим левой пикосети, а также ведомым правой пикосети.

Ячеистая сеть (Mesh)

Технология Bluetooth Mesh родилась с появлением Bluetooth 4.0. Это сеть Bluetooth LE, используемая для установки связи «многие ко многим». Позволяет создавать масштабные сети, где десятки, сотни или даже тысячи устройств Bluetooth могут передавать данные друг другу. Ячеистая сеть Bluetooth не является предметом рассмотрения в этой книге, поэтому вам только нужно знать ее определение на данный момент.

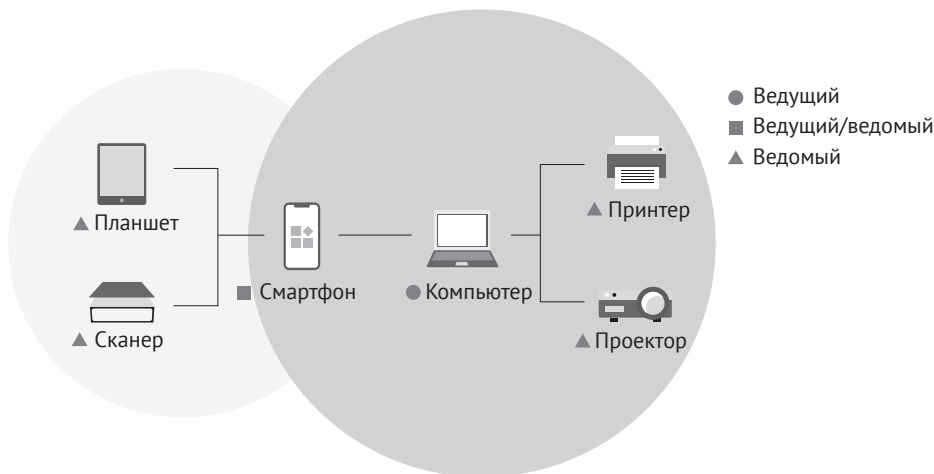


Рисунок 7.14. Топология распределенной сети

7.2.3. Bluetooth-соединение

Bluetooth сначала ищет близлежащие устройства с помощью рассылки оповещений²⁷ или сканирования, затем устанавливает соединение и, наконец, формирует сеть для передачи данных.

1. Рассылка ведомого устройства

Обычно периферийное устройство (ведомое) объявляет о себе и ждет ответа от центрального устройства (ведущего), чтобы обнаружить его и установить соединение по протоколу GATT для дальнейшего обмена. В некоторых случаях ведомое устройство только распространяет свою информацию на несколько центральных устройств, без потребности в подключении.

Чтобы быть обнаруженным ведущим, ведомое устройство будет периодически рассылать оповещения с определенным интервалом t . Мы называем каждое отправленное оповещение «событием», поэтому время t также называют интервалом между событиями оповещений (advertising event interval) (см. рис. 7.15).



Рисунок 7.15. Интервалы между оповещениями

²⁷ В англоязычной терминологии, относящейся к Bluetooth, используется термин «advertising», по аналогии с рассылкой рекламных сообщений. В переводе мы стараемся избегать маркетинговых ассоциаций, поэтому этот термин здесь везде переводится как просто «рассылка оповещений».

События оповещений происходят время от времени, и каждое событие длится определенный период. Bluetooth-чип только позволяет радиочастотному модулю отправлять пакеты в заданное время, и тогда энергопотребление возрастает. В остальное время чип простаивает, поэтому средняя мощность совсем невелика.

Каждое событие содержит три пакета для одного и того же оповещения, которые будут распространяться на каналах 37, 38 и 39 одновременно. Процесс рассылки оповещений ведомого показан на рис. 7.16.

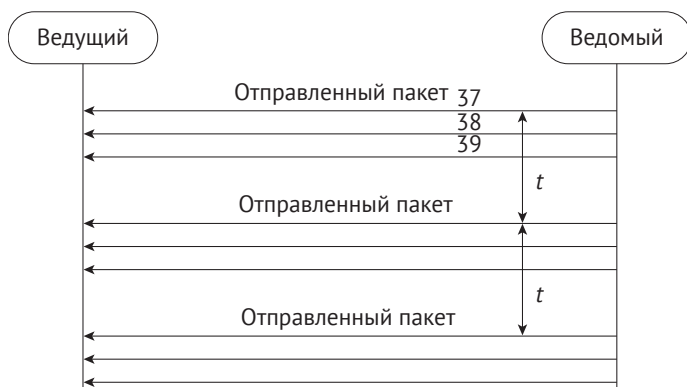


Рисунок 7.16. Рассылка оповещений ведомого

2. Сканирование ведущим

Сканирование относится к процессу, когда ведущий пытается найти другие устройства Bluetooth LE в определенном диапазоне с помощью обнаружения оповещений. В отличие от рассылки оповещений, ведущий настроен на сканирование без настройки интервалов или каналов. Ведущий может настраивать свои собственные параметры.

Пассивное сканирование

При пассивном сканировании сканер только прослушивает оповещения, не отправляя никаких данных отправителю, как показано на рис. 7.17.

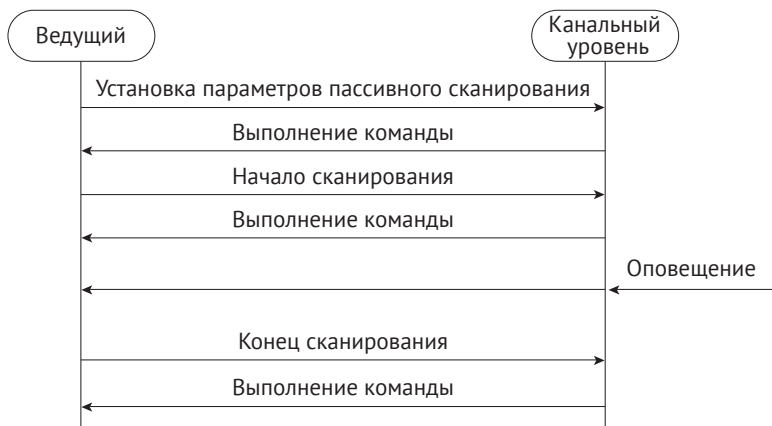


Рисунок 7.17. Пассивное сканирование

После того как параметры сканирования установлены, ведущий может отправлять команды в стеке протоколов, чтобы начать сканирование. В процессе сканирования, если контроллер получает оповещение, соответствующее политике фильтрации или другим требованиям, он сообщит о событии главному устройству. Помимо адреса отправителя, событие также включает в себя данные и измеренные значения уровня сигнала (RSSI) полученного пакета. Разработчики могут оценить потери на пути прохождения сигнала на основе RSSI и мощности передатчика оповещений. Эту функцию можно использовать для разработки трекеров с исправлением ошибок и решений для позиционирования.

Активное сканирование

При активном сканировании ведущий может перехватывать не только пакеты оповещений, отправляемые подчиненными устройствами, но и пакеты ответов на сканирование и различать эти две разновидности. Процесс активного сканирования показан на рис. 7.18.

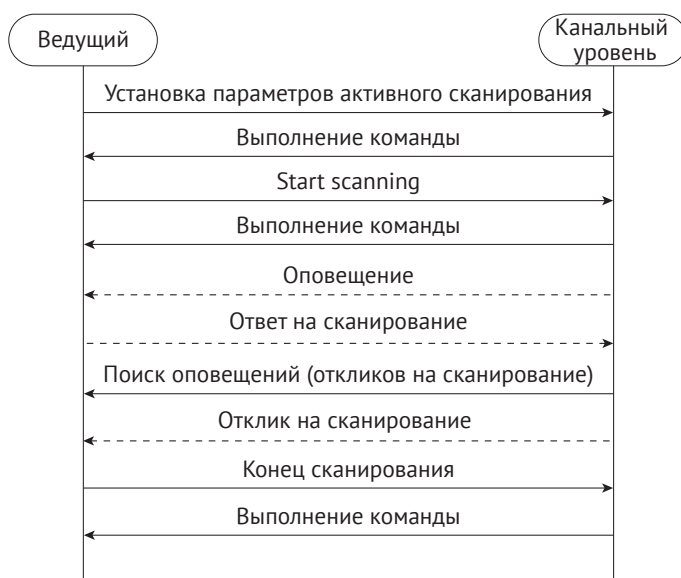


Рисунок 7.18. Активное сканирование

После того как контроллер получит какие-либо данные, он сообщит ведущему о событии, содержащем тип пакета оповещения LL. Ведущий может, таким образом, решить, следует ли подключиться или сканировать ведомое устройство и отличать пакеты оповещений от пакетов ответа на сканирование.

3. Основное соединение

- (1) Периферийное устройство запускает оповещение. Через время t_{IFS} после отправки пакета оно может получать пакеты от центрального устройства. (t_{IFS} : Inter Frame Space, интервал времени между передачей двух пакетов на одном и том же канале.)

- (2) Ведущее устройство сканирует наличие оповещений. В рамках интервала T_{IFS} после получения пакета оповещения, если оно активирует ответ на сканирование, на периферийное устройство отправляется ответ.
- (3) Как только периферийное устройство получит ответ на сканирование, оно вернет пакет подтверждения и подготовится к приему данных.
- (4) Если центральное устройство не получит пакет подтверждения, оно продолжит отправлять ответы на сканирование до истечения времени ожидания. В течение этого периода для установления соединения достаточно получить только один подтверждающий пакет.
- (5) Теперь, когда два устройства подключены, они начинают обмениваться данными. Центральное устройство будет отправлять пакеты данных на периферийное устройство с интервалами подключения, начиная с момента получения оповещения. Пакеты данных используются для синхронизации тактовых импульсов двух устройств и установления связи в режиме ведущий–ведомый. Процесс заключается в следующем:
 - а) каждый раз, когда периферийное устройство получает пакет от центрального устройства, оно сбрасывает начальную точку синхронизации с центральным устройством (служба синхронизируется с клиентом);
 - б) связь по протоколу Bluetooth LE установлена в режиме ведущий–ведомый. Центральное устройство становится ведущим, а периферийное устройство – ведомым. Ведомое устройство может отправлять данные обратно ведущему устройству только в течение определенного времени после того, как ведущее устройство отправит ему пакет;
 - в) соединение установлено;
 - г) периферийное устройство автоматически останавливает отправку оповещений, и другие устройства больше не могут его найти;
 - д) в течение интервала между передачами пакетов центральным устройством периферийное устройство может отправлять несколько оповещений.

Последовательность подключения показана на рис. 7.19.

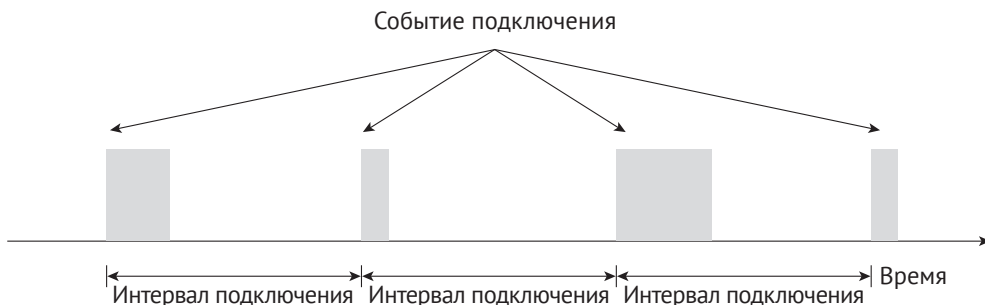


Рисунок 7.19. Последовательность подключений

Чтобы отключиться, центральному устройству нужно только прекратить отправку пакетов. Оно может записать MAC-адрес периферийного устройства во флеш-память, статическое ОЗУ или другие устройства хранения, продолжая следить за адресом и восстанавливать связь, когда получает пакет оповещения с периферии снова. В целях экономии энергии ведомый не будет рассылать оповещения, если нет данных для передачи, и две стороны будут отключены

из-за истечения времени соединения. В это время центральное устройство должно начать мониторинг, чтобы они могли снова соединиться при необходимости ведомого отправить данные.

7.3. Конфигурация сети Wi-Fi

С развитием интернета вещей все больше и больше устройств подключаются через интернет. Однако такие устройства не имеют развитого интерфейса хост-контроллер (HCI), как смартфоны и планшеты. Чтобы подключить их к интернету, пользователи не могут напрямую вводить SSID и пароль маршрутизатора. Как дать возможность таким устройствам подключаться к интернету или локальным сетям с помощью маршрутизатора? Это одна из важных задач устройств Wi-Fi. В данном разделе будут рассмотрены некоторые общие методы настройки сети.

7.3.1. Руководство по настройке сети Wi-Fi

Конфигурация сети заключается в предоставлении SSID и пароля устройствам Wi-Fi, чтобы они могли подключаться к указанной точке доступа и присоединяться к ее сети.

Конечная цель здесь состоит в том, чтобы отправить SSID и пароль точки доступа к устройству Wi-Fi различными способами и подключить устройство к ней, дабы присоединиться к локальной сети или интернету. На рис. 7.20 показан процесс настройки сети Wi-Fi.

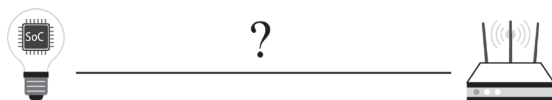


Рисунок 7.20. Процесс настройки Wi-Fi

Устройства интернета вещей, ожидающие подключения, также должны быть связаны с учетной записью, поэтому здесь появляются некоторые новые концепции:

- **конфигурация сети в узком смысле:** устройство Wi-Fi получает информацию о точке доступа (SSID, пароль и т. д.) и подключается к ней;
- **привязка:** связывание учетных записей пользовательских приложений с настроенным устройством;
- **конфигурация сети в широком смысле:** конфигурация сети в узком смысле + привязка.

В этом разделе основное внимание будет уделено конфигурации сети в узком смысле, опуская процесс привязки. В настоящее время наиболее популярными методами настройки сетей являются SoftAP, SmartConfig и Bluetooth.

7.3.2. Программная точка доступа (Soft access point, SoftAP)

1. Краткое описание

SoftAP – это традиционный метод. В начале настраиваемое устройство IoT устанавливает точку доступа, пользователь подключает смартфон, планшет

или другие устройства с HCI (Host Controller Interface) к этой точке доступа и отправляет информацию об устройстве, предоставляющем сеть. Затем устройство IoT ищет соответствующую сеть и соединяется с ней. На рис. 7.21 показаны этапы настройки сети SoftAP.

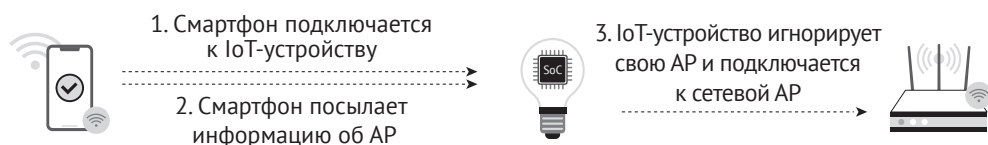


Рисунок 7.21. Этапы настройки сети SoftAP

Механизм SoftAP подключает устройства непосредственно к локальной сети без маршрутизаторов, тем самым предотвращая проблемы с их совместимостью. Это упрощает успешную настройку сети по сравнению с методом SmartConfig. Но недостатком является то, что для подключения требуется дополнительный шаг, так как нам нужно вручную переключиться на IoT SoftAP в списке Wi-Fi. Если мы хотим получить доступ к облачным сервисам, нам все равно будет нужен маршрутизатор. Некоторые смартфоны могут автоматически переключаться по списку AP, но с iOS 11.0 или предыдущими версиями нам придется выполнить дополнительные настройки вручную.

2. Конфигурация

На рис. 7.22 показано, как настраивать сети через SoftAP. Подробное введение в SoftAP будет дано позже вместе с программированием Wi-Fi.

7.3.3. SmartConfig

1. Краткое описание

SmartConfig позволяет смартфонам вводить SSID и пароль в незашифрованном заголовке MAC-пакета в соответствии с определенным форматом кодировки и отправлять их частями на IoT-устройство множество раз с помощью широко-вещательной и многоадресной рассылки. Как правило, нам нужно установить приложение на смартфон для взаимодействия между двумя сторонами. Этапы настройки сети SmartConfig показаны на рис. 7.23.

После того как приемник активирует SmartConfig, IoT-устройство начинает отслеживать данные на маршрутизаторе с канала 1. Как только обнаруживаются пакеты данных, соответствующие правилам, он прекращает переключение каналов и остается на текущем канале для получения остальных данных. В противном случае IoT-устройство автоматически переключается на последующие каналы вплоть до канала 13 и начинает все сначала с канала 1.

Формат кадра MAC-уровня в IEEE 802.11 позволяет четко идентифицировать данные канального уровня LL, которые включают заголовок и данные сетевого уровня. Это позволяет немедленно извлекать и вычислять длину полезных данных, как только получены кадры MAC.

Данные здесь обычно представляют собой пароль. На рис. 7.24 показана структура конфигурации пакета данных сети SmartConfig.

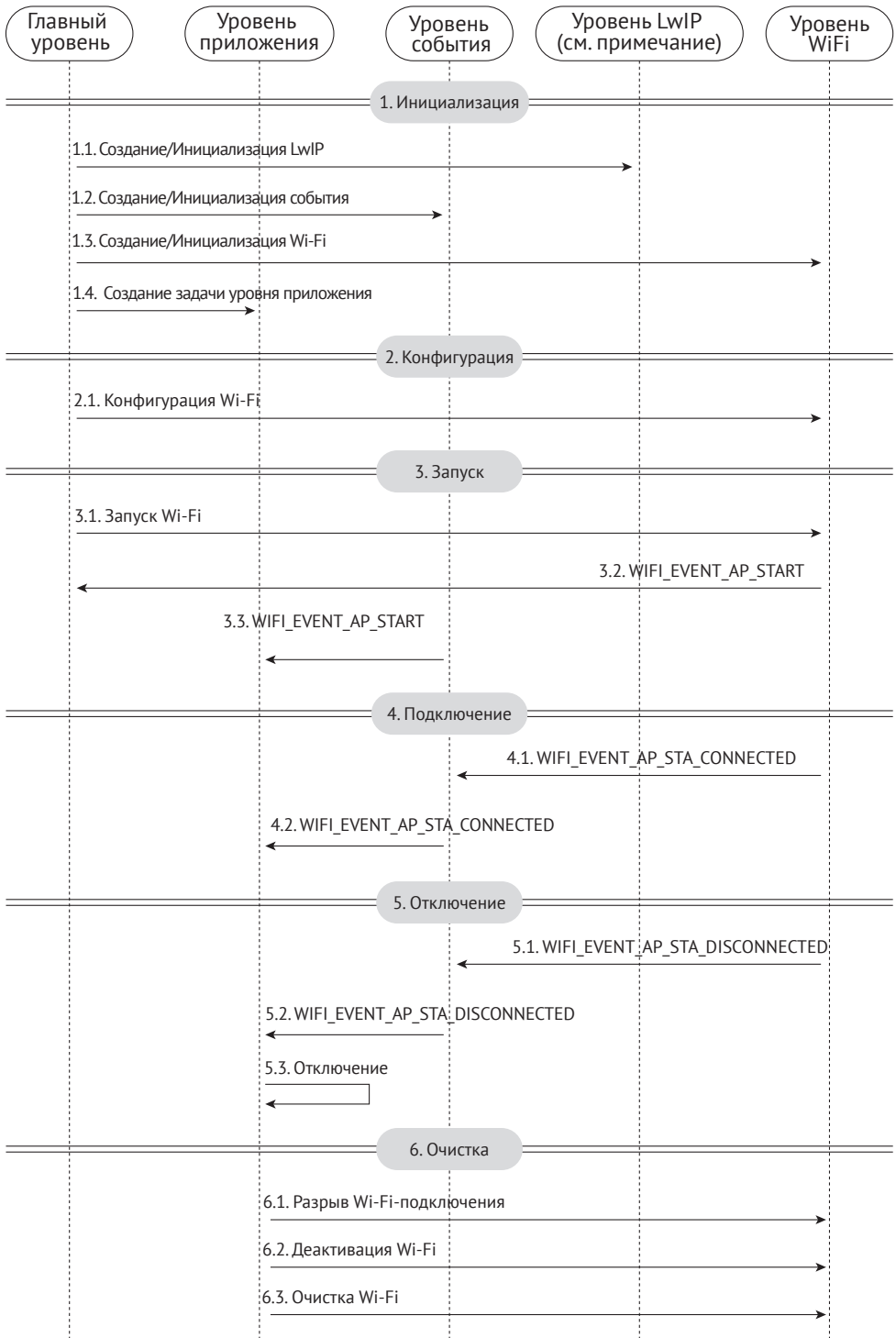


Рисунок 7.22. Настройка сети через SoftAP

Примечание: на рисунке **LwIP** (облегченный IP) – вариант стека протоколов TCP/IP, специально разработанный для встраиваемых систем.

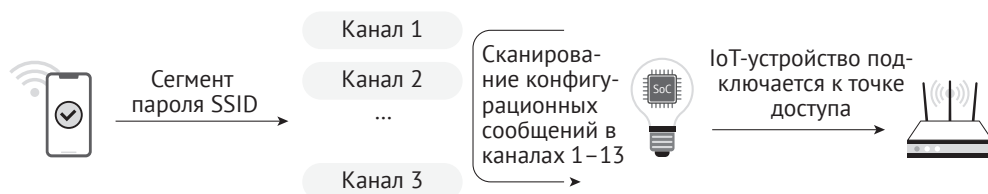


Рисунок 7.23. Этапы настройки сети Smart Config

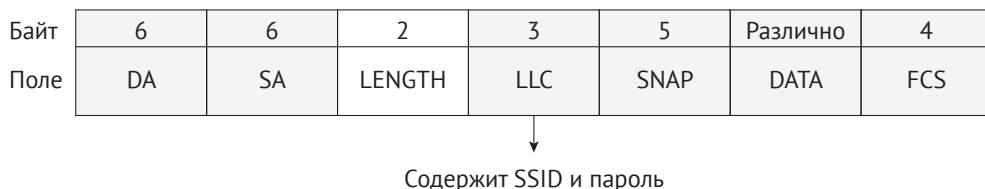


Рисунок 7.24 . Структура пакетов сетевой конфигурации SmartConfig

В табл. 7.2 приведены данные полей пакета сетевой конфигурации SmartConfig.

Таблица 7.2. Поля пакета сетевой конфигурации SmartConfig

Поле	Описание
DA	Целевой MAC-адрес
SA	MAC-адрес источника
LENGTH	Длина данных информационной части
LLC	Заголовок канального уровня
SNAP	3 байта кода производителя + 2 байта типа протокола
DATA	Информационная часть
FCS	Контрольный код

Отправитель обычно использует следующие методы для отправки данных.

Широковещательный UDP

Формат кадра MAC IEEE 802.11 гарантирует, что поля DA, SA, LENGTH, LLC, SNAP и FCS всегда видны мониторам беспроводных сигналов для получения достоверной информации, независимо от того, зашифрованы ли каналы. При трансляции отправитель ограничивает операционную систему, оставив в ее распоряжении только поле LENGTH. Однако при указании конкретного коммуникационного протокола поля LENGTH достаточно для отправления всех необходимых данных.

Многоадресный UDP

Групповой адрес представляет собой зарезервированный адрес класса D с диапазоном от 224.0.0.0 до 239.255.255.255. Сопоставление между IP-адресами и MAC-адресами выполняется путем установки первых 25 бит MAC-адреса на 01.00.5E, а последних 23 бит MAC-адреса – на соответствующие биты IP-адреса. В результате отправитель может кодировать данные в последних 23 битах многоадресного IP-адреса и передать их получателю через многоадресный пакет.

SmartConfig предлагает удобный и гибкий интерфейс, но предъявляет строгие требования к совместимости смартфонов и роутеров. Например, некоторые маршрутизаторы могут отключать переадресацию широковещательных/многоадресных пакетов по умолчанию, предотвращающую получение пакетов устройствами при пересылке через маршрутизатор. В других случаях разные полосы частот, используемые смартфонами и IoT-устройствами, также могут привести к сбою конфигурации. Если смартфон подключен к роутеру в полосе частот 5 ГГц, устройство, использующее полосу частот 2,4 ГГц, может не принимать данные. Такие неуправляемые факторы могут значительно снизить общую совместимость и затруднить настройку сети.

2. Конфигурация

Механизмы SmartConfig, разработанные Espressif:

- **ESP-TOUCH V2**: широковещательное и многоадресное кодирование UDP;
- **ESP-TOUCH**: кодирование вещания UDP;
- **AIRKISS**: мини-программа WeChat.

Исходный код

Подробное введение в SmartConfig будет дано позже вместе с программированием Wi-Fi. Посетите <https://github.com/espressif/esp-idf>; пример кода вы найдете в https://github.com/espressif/esp-idf/tree/master/examples/wifi/smart_config.

7.3.4. Bluetooth

1. Краткое описание

Если настраиваемое IoT-устройство поддерживает Bluetooth, информация о привязке к сети может быть отправлена по Bluetooth-каналу.

Принцип настройки сети с помощью Bluetooth аналогичен принципу настройки с помощью SoftAP, за исключением того, что метод связи, используемый для передачи информации Wi-Fi, изменен с Wi-Fi (режим точки доступа) на Bluetooth. Настраиваемое IoT-устройство создает профиль Bluetooth, затем пользователь подключает к нему смартфоны, планшеты или другие устройства с интерфейсом HCI через канал Bluetooth и отправляет информацию, необходимую для настройки сети. После получения информации IoT-устройство ищет соответствующую точку доступа и подключается к ней.

Этапы настройки сети через Bluetooth показаны на рис. 7.25.

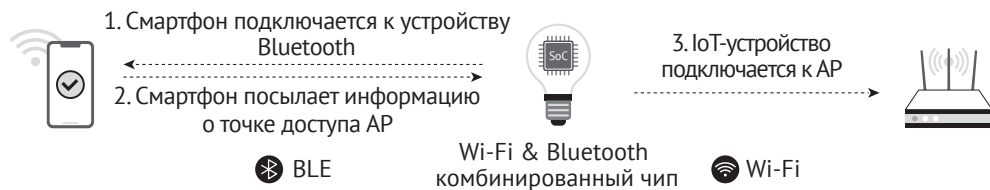


Рисунок 7.25. Этапы настройки сети через Bluetooth

Преимущество сетевой конфигурации через Bluetooth заключается в том, что устраняются проблемы совместимости, связанные с маршрутизаторами, обеспечивающими более высокую скорость соединения. Также можно обнаруживать и подключать устройства напрямую, поэтому нет необходимости подключать устройство к его собственной точке доступа. Однако совместимость между модулем Bluetooth и мобильным телефоном может повлиять на конфигурацию сети. Кроме того, использование модулей Bluetooth увеличивает стоимость устройства.

2. Конфигурация

Чип ESP32-C3 поддерживает как Wi-Fi, так и Bluetooth LE, таким образом поддерживая различные методы конфигурации сети. Когда дело доходит до конфигурации сети через Bluetooth, ESP32-C3 предлагает комплексное решение под названием BluFi. На рис. 7.26 показано, как настраивать сети через BluFi.

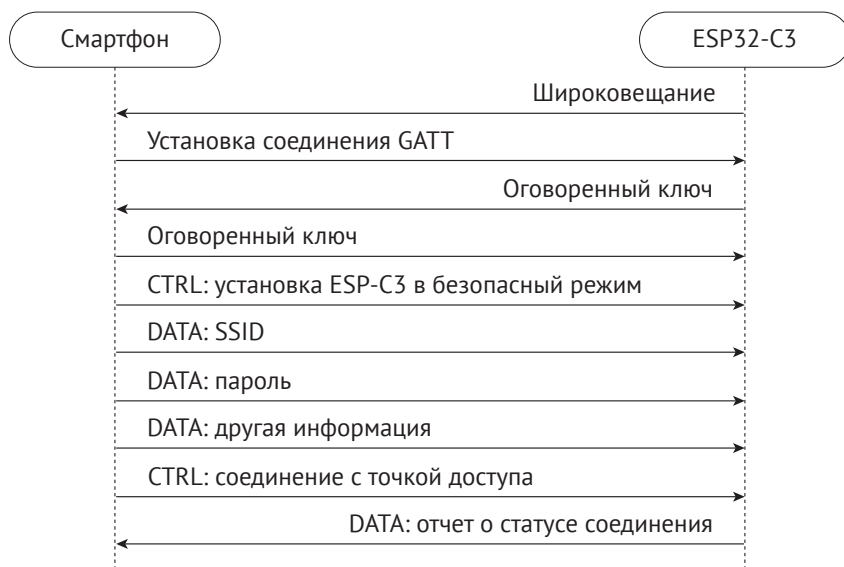


Рисунок 7.26. Настройка сети через BluFi

Исходный код

Более подробное введение в конфигурацию сети с помощью Bluetooth будет дано позже вместе с программированием Wi-Fi. Посетите <https://github.com/espressif/esp-idf>, пример кода вы найдете в <https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluifi>.

7.3.5. Другие методы

1. Прямая настройка сети

Прямая настройка сети означает отправку SSID и пароля непосредственно на устройства IoT через периферийные интерфейсы, такие как UART, SPI, SDIO или I2C, в соответствии с определенным протоколом связи. Этот метод также известен как конфигурация через проводную сеть. Как только IoT-устройство получает SSID и пароль, оно подключается к точке доступа и затем возвращает результат подключения через главный интерфейс.

Кроме того, некоторые устройства поставляются с предварительно установленной информацией о Wi-Fi, такой как SSID и пароль. Когда такие устройства запускаются в указанной среде Wi-Fi, они могут автоматически подключаться к соответствующей точке доступа. Такие устройства обычно используются в крупномасштабных сетях, при заводских испытаниях или в промышленных сценариях. Этапы прямой настройки сети показаны на рис. 7.27.



Рисунок 7.27. Этапы прямой настройки сети

Этот метод использует программное решение и прост в реализации. Он хорошо подходит для устройства с чипами Wi-Fi или проводными линиями передачи других протоколов. Однако эти линии передачи должны быть предварительно протянуты между системами.

Командная прошивка Espressif AT (ESP-AT), предоставляемая Espressif, может быть непосредственно использована в массовых IoT-приложениях. Разработчики могут легко присоединиться к беспроводным сетям, запустив команды Wi-Fi. Подробнее см. в руководстве пользователя ESP-AT по адресу <https://docs.espressif.com/projects/esp-at/en/latest/esp32/index.html>.

2. Настройка маршрутизатора RouterConfig

RouterConfig основан на стандарте Wi-Fi Protected Setup (WPS), представленном альянсом Wi-Fi для решения сложного процесса настройки шифрования и настройки аутентификации в беспроводной сети. Цель WPS – упростить безопасность Wi-Fi и управление сетью для пользователей. Стандарт предлагает два метода: метод персонального идентификационного номера (PIN) и метод настройки нажатием кнопки (PBC). На рис. 7.28 показаны этапы RouterConfig.

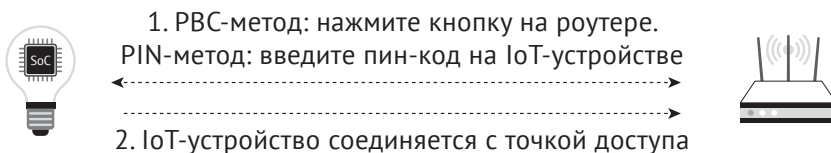


Рисунок 7.28. Этапы RouterConfig

Этот процесс сравнительно простой, но требует, чтобы и маршрутизатор, и устройство поддерживали WPS. К сожалению, многие пользователи прене-

брегают настройками шифрования из-за громоздкости действия, которые могут привести к серьезным проблемам с безопасностью. В результате увеличивается количество маршрутизаторов, отказывающихся от поддержки WPS или отключающих ее по умолчанию. Метод имеет тенденцию становиться менее популярным в последние годы.

Исходный код

ESP-IDF, официальная среда разработки IoT от Espressif, служит примером этого решения для настройки сети. Процесс там достаточно простой. Посетите <https://github.com/espressif/esp-idf>, пример кода вы найдете в <https://github.com/espressif/esp-idf/tree/master/examples/wifi/wps>.

3. ZeroConfig

ZeroConfig – это метод использования одного подключенного устройства при настройке сети для другого. Данный метод не предполагает использование смартфонов, так как вместо них могут быть использованы другие IoT-устройства, такие как смарт-колонки.

Чтобы инициировать процесс, подключаемое устройство отправляет свой MAC-адрес подключенному к сети устройству через пользовательское сообщение. Подключенное устройство отвечает, отправляя обратно сохраненный SSID и пароль маршрутизатора через другое пользовательское сообщение. После подключения устройство может выполнить дальнейшую настройку, такую как привязка к внешней сети. Этапы настройки сети методом ZeroConfig показаны на рис. 7.29.

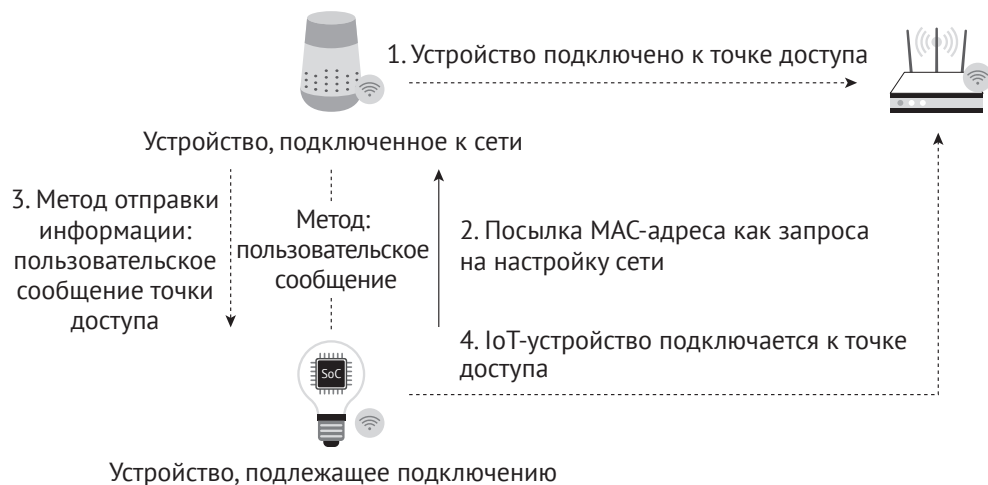


Рисунок 7.29. Этапы ZeroConfig

Поскольку сетевое устройство хранит SSID и пароль маршрутизатора, пользователям не нужно вводить их вручную. Конфигурация будет проще, что более удобно пользователям. Однако этот метод не может быть широко распространен, так как должны быть подключены сетевые устройства с роутером. В то же время, поскольку мобильные приложения имеют ограниченный доступ,

здесь невозможно собрать все данные об управлении Wi-Fi через сторонние программы. Поэтому смартфоны могут использоваться для реализации этого метода только при определенных обстоятельствах.

4. Настройка сетевой телефонной точки доступа

Конфигурация сетевой телефонной точки доступа (Phone AP) означает настройку смартфона в качестве точки доступа с уникальным именем и паролем, подключение IoT-устройства к этой точке доступа и отправку информации о сетевой привязке. На рис. 7.30 показаны этапы настройки сетевой телефонной точки доступа.



Рисунок 7.30. Этапы настройки сетевой телефонной точки доступа

Конфигурация сетевой телефонной точки доступа не требует, чтобы устройство IoT поддерживало режим точки доступа, поэтому пользователям не нужно много заниматься разработкой устройства. Его можно использовать одновременно со SmartConfig, что делает его хорошим кандидатом для резервной конфигурации сети.

Тем не менее полученный пользовательский опыт едва ли удовлетворителен. Многим пользователям сложно настроить имя точки доступа смартфона или даже включение точки доступа на нем. В частности, на устройствах iOS приложение не может автоматически создать точку доступа, поэтому пользователям приходится вручную изменять имя устройства и включать точку доступа. В результате этот метод не подходит для потребительских устройств.

В дополнение к описанным выше методам настройки Espressif также поддерживает Wi-Fi Easy Connect, известный как протокол подготовки устройств (DPP). Для получения дополнительной информации, пожалуйста, посетите <https://book3/espressif.com/esp-dpp>.

7.4. Программирование Wi-Fi

В этом разделе представлен обзор API-интерфейсов Wi-Fi, в котором рассказывается, как использовать API-интерфейсы, чтобы установить подключение устройства STA и подключиться оптимальным способом.

При разработке Wi-Fi-приложения наиболее эффективным способом является адаптация подходящего примера для ваших требований. Поэтому если вы хотите создать надежное приложение, прежде чем начать свой собственный проект, изучите этот раздел и выполните практические примеры.

7.4.1. Компоненты Wi-Fi в ESP-IDF

1. Особенности

Компоненты Wi-Fi можно использовать для настройки и мониторинга сетевого подключения Wi-Fi ESP32-C3. Поддерживаются следующие функции:

- **режим STA:** также известный как режим станции или режим клиента Wi-Fi. В этом режиме ESP32-C3 подключается к внешней точке доступа (AP);
- **режим точки доступа:** также известный как режим SoftAP или режим точки доступа. В этом режиме точка доступа подключена к ESP32-C3;
- **совмещенный режим AP-STA:** ESP32-C3 подключен к другой AP как AP;
- **стандарты безопасности** для указанных выше режимов: WPA, WPA2, WPA3, WEP и т. д.;
- **сканирование точек доступа**, включая активное и пассивное сканирование;
- **смешанный режим** для мониторинга пакетов Wi-Fi IEEE 802.11.

2. API

Файл `esp_wifi.h` для Wi-Fi-компонентов определяет API, показанные в табл. 7.3.

Таблица 7.3. API для Wi-Fi-компонентов

Название функции	Описание
<code>esp_wifi_init()</code>	Инициализировать ресурсы для драйвера Wi-Fi, такие как структуры управления и задачи Wi-Fi
<code>esp_wifi_deinit()</code>	Освободить ресурсы, выделенные <code>esp_wifi_init()</code> , и остановить задачи
<code>esp_wifi_set_mode()</code>	Установить режим работы WiFi для ESP32-C3
<code>esp_wifi_get_mode()</code>	Получить установленный режим работы Wi-Fi для ESP32-C3
<code>esp_wifi_start()</code>	Запустить Wi-Fi в соответствии с текущей конфигурацией
<code>esp_wifi_stop()</code>	Остановить Wi-Fi в соответствии с текущей конфигурацией
<code>esp_wifi_connect()</code>	Подключить ESP32-C3 к точке доступа
<code>esp_wifi_disconnect()</code>	Отключить ESP32-C3 от точки доступа
<code>esp_wifi_scan_start()</code>	Сканировать все доступные точки доступа
<code>esp_wifi_scan_stop()</code>	Остановить сканирование
<code>esp_wifi_scan_get_ap_num()</code>	Получить количество точек доступа, найденных ESP32-C3
<code>esp_wifi_scan_get_ap_records()</code>	Получить информацию о точках доступа, найденных ESP32-C3
<code>esp_wifi_set_config()</code>	Установить конфигурацию ESP32-C3 (STA или AP)
<code>esp_wifi_get_config()</code>	Получить установленную конфигурацию ESP32-C3 (STA или AP)

3. Модель программирования

Модель программирования Wi-Fi ESP32-C3 изображена на рис. 7.31.

Драйвер Wi-Fi можно рассматривать как черный ящик, который ничего не знает о коде верхнего уровня, таком как стеки TCP, задачи приложений и задачи обработки событий. Код приложения вызывает API драйвера Wi-Fi для инициализации Wi-Fi и при необходимости обрабатывает события Wi-Fi. Wi-Fi-драйвер получает вызовы API, обрабатывает их и отправляет результат в приложение.

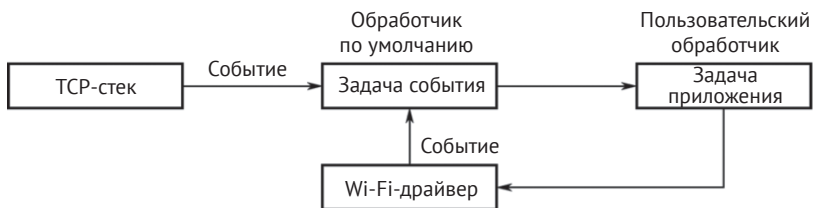


Рисунок 7.31. Модель программирования Wi-Fi ESP32-C3

Обработка событий Wi-Fi основана на библиотеке `esp_event`. События отправляются драйвером Wi-Fi в цикл событий по умолчанию. Приложения могут обрабатывать эти события в обратных вызовах, зарегистрированных с помощью `esp_event_handler_register()`. События Wi-Fi также обрабатываются компонентом `esp_netif` для обеспечения правил поведения по умолчанию. Например, когда станция Wi-Fi подключается к точке доступа, `esp_netif` автоматически запустит клиента протокола динамической конфигурации хоста (Dynamic Host Configuration Protocol, DHCP) по умолчанию.

7.4.2. Упражнение: соединение Wi-Fi

1. Переведите ESP32-C3 в режим STA и подключитесь к точке доступа.

При работе в режиме STA ESP32-C3 может подключаться к точке доступа как STA.

BSS (базовый набор услуг), основанный на центральной точке доступа, позволяет нескольким STA строить беспроводную сеть с точкой доступа, перенаправляющей все сообщения в сети. В этом режиме устройство может использоваться для прямого доступа как к внешней, так и к внутренней сети, используя IP-адрес, назначенный точкой доступа. Рисунок 7.32 поясняет режим Wi-Fi STA.

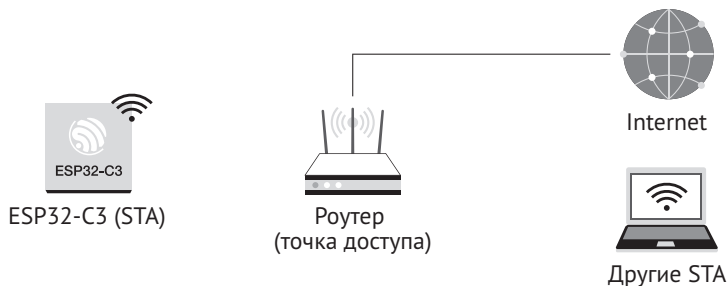


Рисунок 7.32. STA-режим сети Wi-Fi

2. Используйте компоненты ESP-IDF для подключения устройств к маршрутизаторам.

На рис. 7.33 показано, как использовать компоненты ESP-IDF для подключения устройств к маршрутизаторам.

(1) **Инициализация.** См. 1.1, 1.2 и 1.3 на рис. 7.33.

а. Инициализировать LwIP. Создайте основную задачу LwIP и инициализируйте работу, связанную с LwIP:

```
1. ESP_ERROR_CHECK(esp_netif_init());
```

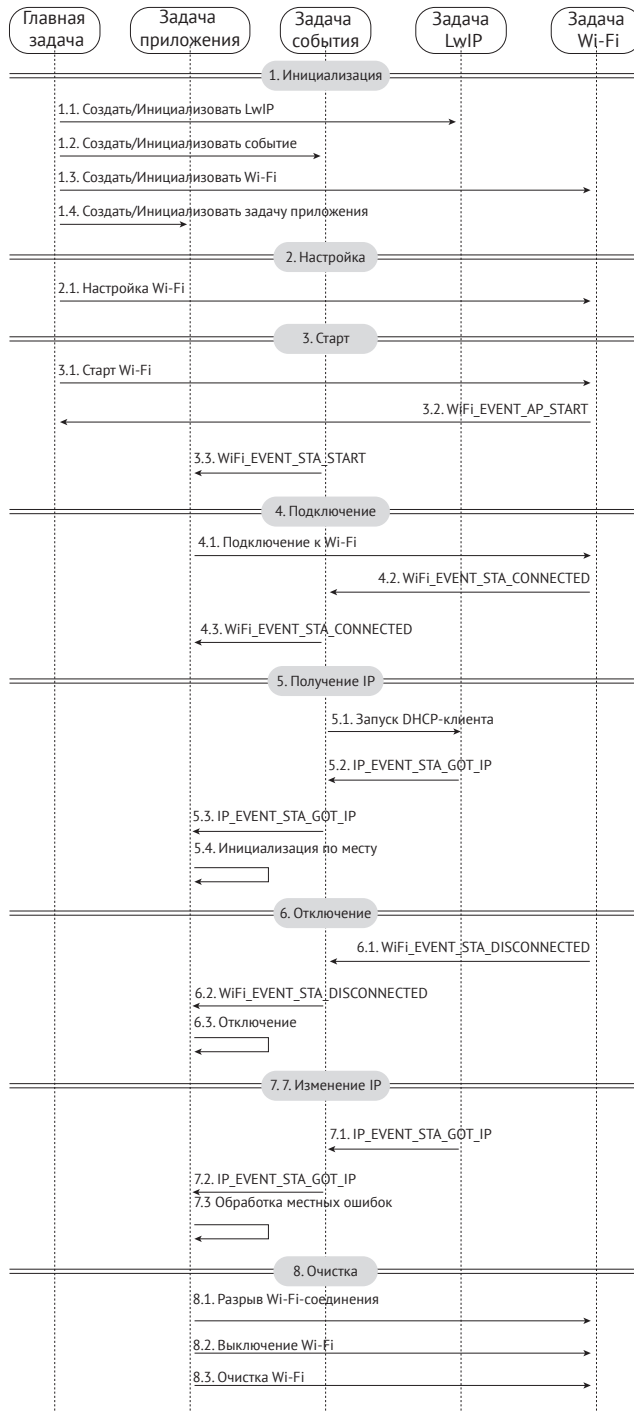


Рисунок 7.33. Использование компонентов ESP-IDF для подключения устройств к маршрутизаторам

в. Инициализация события. Как было сказано ранее, обработка событий Wi-Fi основана на библиотеке `esp_event`, поддерживаемой компонентом `esp_netif`. Код для инициализации события выглядит следующим образом:

```
1. ESP_ERROR_CHECK(esp_event_loop_create_default());
2. esp_netif_create_default_wifi_sta();
3. esp_event_handler_instance_t instance_any_id;
4. esp_event_handler_instance_t instance_got_ip;
5. ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
6.                                                     ESP_EVENT_ANY_ID,
7.                                                     &event_handler,
8.                                                     NULL,
9.                                                     &instance_any_id));
10. ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
11.                                                      IP_EVENT_STA_GOT_IP,
12.                                                      &event_handler,
13.                                                      NULL,
14.                                                      &instance_got_ip));
```

с. Инициализация Wi-Fi. Создайте задачу драйвера Wi-Fi и инициализируйте драйвер Wi-Fi. Код для инициализации Wi-Fi выглядит следующим образом:

```
1. wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
2. ESP_ERROR_CHECK(esp_wifi_init(&cfg));
```

(2) **Настройка.** После инициализации драйвера Wi-Fi можно приступить к настройке. На этом этапе драйвер находится в режиме STA, поэтому вы можете вызвать `esp_wifi_set_mode(WIFI_MODE_STA)`, переведя ESP32-C3 в режим STA. Соответствующий код выглядит следующим образом:

```
1. wifi_config_t wifi_config = {
2.     .sta = {
3.         .ssid = EXAMPLE_ESP_WIFI_SSID,
4.         .password = EXAMPLE_ESP_WIFI_PASS,
5.     },
6. };
7. ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
8. ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
```

(3) **Старт.** Вызовите `esp_wifi_start()`, чтобы запустить драйвер Wi-Fi.

```
1. ESP_ERROR_CHECK(esp_wifi_start());
```

Драйвер Wi-Fi отправляет `WIFI_EVENT_STA_START` в задачу события; затем эта задача выполнит некоторую рутинную работу и вызовет функцию обратного вызова события приложения. Функция обратного вызова события приложения передает `WIFI_EVENT_STA_START` приложению, а затем вызывает `esp_wifi_connect()`.

(4) **Соединение.** После вызова `esp_wifi_connect()` драйвер Wi-Fi запустит внутренний процесс сканирования/подключения. Если внутреннее сканирова-

ние/подключение прошло успешно, будет сгенерировано `WIFI_EVENT_STA_CONNECTED`.

В задаче события будет запущен клиент DHCP и иницируется процесс конфигурации хоста. Соответствующий код выглядит следующим образом:

```
1. static void event_handler(void* arg, esp_event_base_t event_base,
2.                          int32_t event_id, void* event_data)
3. {
4.     if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
5.         esp_wifi_connect();
6.     } else if
7.         (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
8.         if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY) {
9.             esp_wifi_connect();
10.            s_retry_num++;
11.            ESP_LOGI(TAG, "retry to connect to the AP");
12.        } else {
13.            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
14.        }
15.        ESP_LOGI(TAG, "connect to the AP fail");
16.    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
17.        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
18.        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
19.        s_retry_num = 0;
20.        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
21.    }
22. }
```

(5) **Получение IP.** После инициализации DHCP-клиента начнется фаза «получения IP». Если IP-адрес успешно получен от DHCP-сервера, сработает `IP_EVENT_STA_GOT_IP` и будет обработан в задаче события.

В обратном вызове события приложения `IP_EVENT_STA_GOT_IP` передается задаче приложения. Для приложений на базе LwIP это особое событие, означающее, что все готово для выполнения приложением последующих задач. Но помните, что не следует запускать работу, связанную с подключением, до получения IP.

(6) **Отключение.** Соединение Wi-Fi может не работать из-за отключения активности, неправильного пароля, ненайденной точки доступа и т. д. В этом случае возникнет событие `WIFI_EVENT_STA_DISCONNECTED` и укажет причину сбоя, например вызовет `esp_wifi_disconnect()` при отключении активности.

```
1. ESP_ERROR_CHECK(esp_wifi_disconnect());
```

(7) **Изменение IP.** Если IP-адрес изменен, будет активировано событие `IP_EVENT_STA_GOT_IP` с `ip_change`, установленное в `true`.

(8) **Очистка**, включая разрыв соединения Wi-Fi, остановку и выгрузку драйвера Wi-Fi и т. д. Код выглядит следующим образом:

```

1. ESP_ERROR_CHECK(esp_event_handler_instance_unregister(IP_EVENT,
2.                                     IP_EVENT_STA_GOT_IP,
3.                                     instance_got_ip));
4. ESP_ERROR_CHECK(esp_event_handler_instance_unregister(WIFI_EVENT,
5.                                     ESP_EVENT_ANY_ID,
6.                                     instance_any_id));
7. ESP_ERROR_CHECK(esp_wifi_stop());
8. ESP_ERROR_CHECK(esp_wifi_deinit());
9. ESP_ERROR_CHECK(esp_wifi_clear_default_wifi_driver_and_handlers(
10.                station_netif));
11. esp_netif_destroy(station_netif);

```

7.4.3. Упражнение: интеллектуальное подключение к Wi-Fi

1. SoftAP

Компоненты `wifi_provisioning`, предоставляемые ESP32-C3, могут передавать SSID и пароль точки доступа через SoftAP или Bluetooth LE, а затем использовать их для подключения к точке доступа.

1) API-интерфейсы

API-интерфейсы компонентов `wifi_provisioning` определены в https://github.com/espressif/esp-idf/blob/master/components/wifi_provisioning/include/wifi_provisioning_manager.h и показаны в табл. 7.4.

Таблица 7.4. API-интерфейсы компонентов `wifi_provisioning`

API	Описание
<code>wifi_prov_mgr_init()</code>	Инициализировать интерфейс диспетчера provisioning в соответствии с текущей конфигурацией
<code>wifi_prov_mgr_deinit()</code>	Освободить интерфейс менеджера provisioning
<code>wifi_prov_mgr_is_provisioned()</code>	Проверка статуса менеджера provisioning ESP32-C3
<code>wifi_prov_mgr_start_provisioning()</code>	Запустить службу provisioning
<code>wifi_prov_mgr_stop_provisioning()</code>	Остановить службу provisioning
<code>wifi_prov_mgr_wait()</code>	Ожидание, пока служба provisioning завершит работу
<code>wifi_prov_mgr_disable_auto_stop()</code>	Отключить автоматическую остановку службы provisioning после завершения
<code>wifi_prov_mgr_endpoint_create()</code>	Создать endpoint и выделить для нее внутренние ресурсы
<code>wifi_prov_mgr_endpoint_register()</code>	Зарегистрировать обработчик для созданной endpoint

API	Описание
wifi_prov_mgr_endpoint_unregister()	Отменить регистрацию обработчика для созданной endpoint
wifi_prov_mgr_get_wifi_state()	Получить состояние Wi-Fi STA во время provisioning
wifi_prov_mgr_get_wifi_disconnect_reason()	Получить код причины отключения Wi-Fi STA во время provisioning

(2) Структура программы

- Инициализация:

```
1. wifi_prov_mgr_config_t config = {
2.     .scheme = wifi_prov_scheme_softap,
3.     .scheme_event_handler = WIFI_PROV_EVENT_HANDLER_NONE
4. };
5.
6. ESP_ERR_CHECK(wifi_prov_mgr_init(config));
```

- Проверка статуса provisioning:

```
1. bool provisioned = false;
2. ESP_ERROR_CHECK(wifi_prov_mgr_is_provisioned(&provisioned));
```

- Запуск службы provisioning:

```
1. const char *service_name = "my_device";
2. const char *service_key = "password";
3.
4. wifi_prov_security_t security = WIFI_PROV_SECURITY_1;
5. const char *pop = "abcd1234";
6.
7. ESP_ERR_CHECK(wifi_prov_mgr_start_provisioning(security, pop, service_name,
8.     service_key));
```

- Высвобождение ресурсов provisioning.

Как только служба provisioning будет завершена, основное приложение освобождает ресурсы provisioning и начинает выполнение собственной логики. Есть два способа сделать это. Более простой способ – вызвать `wifi_prov_mgr_wait()`. Код следующий:

```
1. //Wait for the provisioning service to finish
2. wifi_prov_mgr_wait();
3.
4. //Release the resources for provisioning
5. wifi_prov_mgr_deinit();
```

Другой способ – использовать функцию обратного вызова события. Код следующий:

```
1. static void event_handler(void* arg, esp_event_base_t event_base,
2.                             int event_id, void* event_data)
3. {
4.     if (event_base == WIFI_PROV_EVENT && event_id == WIFI_PROV_END) {
5.         //Release the resources for provisioning upon completion
6.         wifi_prov_mgr_deinit();
7.     }
8. }
```

(3) Функциональная проверка

Для начала установите ESP SoftAP Provisioning на свой телефон. Затем включите Wi-Fi и включите устройство. Убедитесь, что в логе (см. рис. 7.34) через последовательный порт выводится информация, начинающаяся с PROV_.

Примечание

Вы можете скачать приложение по адресу:

<https://www.espressif.com/en/support/download/apps>.

```
I (904) wifi_prov_mgr: Provisioning started with service name : PROV_DAED2C
I (914) app: Provisioning started
I (914) app: Scan this QR code from the provisioning application for Provisioning.
I (924) QRCODE: Encoding below text with ECC LVL 0 & QR Code Version 10
I (934) QRCODE: {"ver":"v1","name":"PROV_DAED2C","pop":"abcd1234","transport":"softap"}
```



```
I (1154) app: If QR code is not visible, copy paste the below URL in a browser.
https://espressif.github.io/esp-jumpstart/qrcode.html?data={"ver":"v1","name":"PROV_DAED2C",
"pop":"abcd1234","transport":"softap"}
```

Рисунок 7.34. Вывод лога через последовательный порт

а. Запуск

Откройте приложение на телефоне и нажмите **Start Provisioning** (Начать подготовку). Тогда вы найдете устройство PROV DAED2CXXXXXX на экране (см. рис. 7.35).

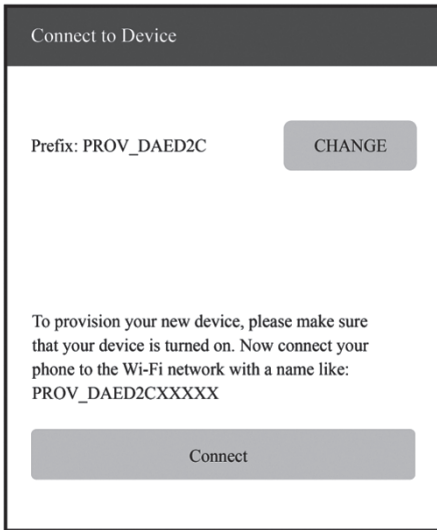


Рисунок 7.35. Запуск

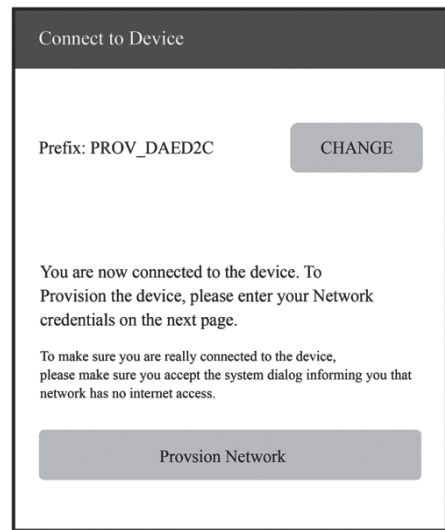


Рисунок 7.36. Соединение SoftAP

b. Соединение

Нажмите **Connect** (Подключиться), чтобы перейти к интерфейсу настройки Wi-Fi. Выберите, чтобы подключить устройство PROV_DAED2CXXXXX. При подключении вы увидите на экране то, что показано на рис. 7.36.

Выходной лог выглядит следующим образом:

```
I (102906) wifi:station: 88:40:3b:40:c1:13 join, AID=1, bgn, 40U
I (103056) esp_netif_lwip: DHCP server assigned IP to a station, IP is: 192.168.4.2
I (124286) wifi:station: 88:40:3b:40:c1:13 leave, AID = 1, bss_flags is 134259, bss:0x3fca7844
I (124286) wifi:new: <1,0>, old: <1,1>, ap: <1,1>, sta: <0,0>, prof:1
I (149036) wifi:new: <1,1>, old: <1,0>, ap: <1,1>, sta: <0,0>, prof:1
I (149036) wifi:station: 88:40:3b:40:c1:13 join, AID=1, bgn, 40U
I (149246) esp_netif_lwip: DHCP server assigned IP to a station, IP is: 192.168.4.2
```

c. Предоставление сети

Нажмите **Provision Network** (Предоставить сеть), чтобы открыть экран настройки, показанный на рис. 7.37.

d. Завершение

Нажмите **Provision**, чтобы открыть экран завершения, показанный на рис. 7.38.

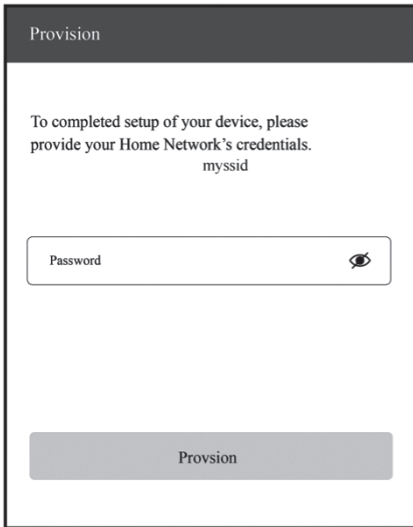


Рисунок 7.37. Предоставление сети

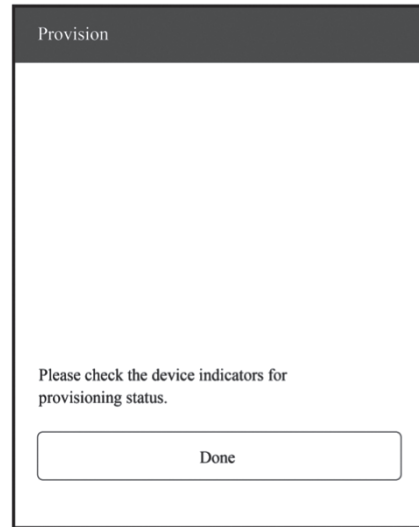


Рисунок 7.38. Завершение

В выходном логе будет записано следующее:

```
I (139471) app: Received Wi-Fi credentials
SSID : myssid
Password : mypassword
.
.
.
I (144091) app: Connected with IP Address:192.168.50.31
I (144091) esp_netif_handlers: sta ip: 192.168.50.31, mask: 255.255.255.0, gw: 192.168.50.1
I (144091) wifi_prov_mgr: STA Got IP
I (144101) app: provisioningssuccessful
I (144101) app: Hello World!
I (145101) app: Hello World!
.
.
.
I (146091) wifi_prov_mgr: Provisioning stopped
I (146101) app: Hello World!
I (147101) app: Hello World!
I (148101) app: Hello World!
```

2. SmartConfig

Компонент SmartConfig, предоставляемый ESP32-C3, может передавать SSID и пароль точки доступа AP в смешанном режиме, а затем использовать их для подключения к AP.

(1) API

API для SmartConfig определены в `esp-idf/components/esp_wifi/include/esp_smartconfig.h` и показаны в табл. 7.5.

Таблица 7.5. API для SmartConfig

API	Описание
esp_smartconfig_get_version()	Получить текущую версию SmartConfig
esp_smartconfig_start()	Запуск SmartConfig
esp_smartconfig_stop()	Остановка SmartConfig
esp_esptouch_set_timeout()	Установить тайм-аут для процесса SmartConfig
esp_smartconfig_set_type()	Задать тип протокола SmartConfig
esp_smartconfig_fast_mode()	Установить режим SmartConfig
esp_smartconfig_get_rvd_data()	Получить резервные данные ESPTouch v2

(2) Структура программы

- Обработка событий Wi-Fi

Этот модуль заботится о подключении к Wi-Fi, отключении, повторном подключении, сканировании и т. д., как подробно описано в предыдущих разделах. Кроме того, когда произойдет событие WIFI_EVENT_STA_START, также будет создана задача SmartConfig.

- Обработка событий NETIF

Этот модуль помогает получить IP-адрес. Подробная информация представлена в предыдущих разделах. Когда произойдет событие IP_EVENT_STA_GOT_IP, будет установлен флаг соединения.

- Обработка событий SmartConfig

Полученный запрос определяет способ обращения и обработки событий. События SmartConfig показаны в табл. 7.6.

Таблица 7.6. События SmartConfig

Событие	Описание
SC_EVENT_SCAN_DONE	Сканирование для получения информации о ближайших точках доступа
SC_EVENT_FOUND_CHANNEL	Получить канал заданной точки доступа
SC_EVENT_GOT_SSID_PSWD	Войти в режим STA, чтобы получить SSID и пароль заданной точки доступа
SC_EVENT_SEND_ACK_DONE	Установить флаг завершения SmartConfig

- Задачи SmartConfig

Код задач SmartConfig выглядит следующим образом:

```

1. static void smartconfig_example_task (void *param)
2. {
3.     EventBits_t uxBits;
4.     ESP_ERROR_CHECK(esp_smartconfig_set_type(SC_TYPE_ESPTOUCH));
5.     smartconfig_start_config_t cfg = SMARTCONFIG_START_CONFIG_DEFAULT();
6.     ESP_ERROR_CHECK(esp_smartconfig_start(&cfg));
7.     while (1) {
8.         uxBits = xEventGroupWaitBits (s_wifi_event_group,
9.                                     CONNECTED_BIT | ESPTOUCH_DONE_BIT,
10.                                    true,
11.                                    false,
12.                                    portMAX_DELAY);
13.         if (uxBits & CONNECTED_BIT) {
14.             ESP_LOGI (TAG, "WiFi Connected to ap");
15.         }
16.         if (uxBits & ESPTOUCH_DONE_BIT) {
17.             ESP_LOGI (TAG, "smartconfig over");
18.             esp_smartconfig_stop();
19.             vTaskDelete (NULL);
20.         }
21.     }
22. }

```

Как показано в приведенном коде, задачи SmartConfig в основном выполняют три функции. Во-первых, устанавливается тип SmartConfig, такой как ESP-TOUCH и ESP-TOUCH V2. Во-вторых, после настройки вызовом `esp_smartconfig_start()` SmartConfig включается. Наконец, проверяется группа событий в цикле. Получив событие `SC_EVENT_SEND_ACK_DONE`, задача останавливает SmartConfig, вызывая `esp_smartconfig_stop()`.

- Основная программа

Она создает группу событий для установки флага при запуске соответствующего события, а затем инициализирует Wi-Fi.

(3) Функциональная проверка

Для начала установите Espressif Esptouch на свой телефон. Затем включите Wi-Fi и питание на устройстве. В выводе лога через последовательный порт вы увидите следующее:

```

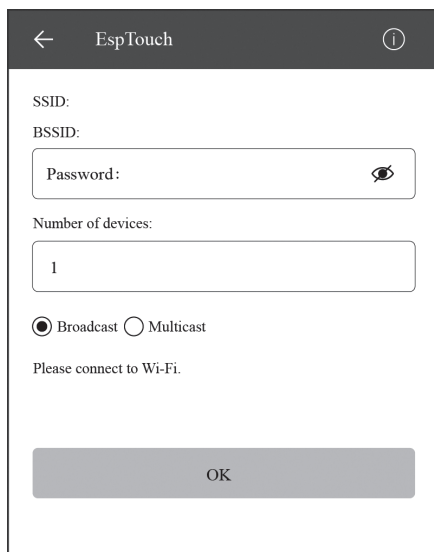
I (1084) wifi:mode : sta (30:ae:a4:80:65:7c)
I (1084) wifi:enable tsf
I (1134) smartconfig: SC version: V3.0.1
I (5234) wifi:ic_enable_sniffer
I (5234) smartconfig: Start to find channel...
I (5234) smartconfig_example: Scan done

```

Примечание

Вы можете скачать приложение по адресу: <https://www.espressif.com/en/support/download/apps>.

Подключите телефон к Wi-Fi и введите пароль, чтобы начать настройку. Интерфейс SmartConfig показан на рис. 7.39.



← EspTouch ⓘ

SSID:
BSSID:

Password:

Number of devices:

Broadcast Multicast

Please connect to Wi-Fi.

OK

Рисунок 7.39. Настройка SmartConfig

Выходной лог выглядит следующим образом:

```
I (234592) smartconfig: TYPE: ESPTOUCH
I (234592) smartconfig: T|PHONE MAC:68:3e:34:88:59:bf
I (234592) smartconfig: T|AP MAC:a4:56:02:47:30:07
I (234592) sc: SC_STATUS_GETTING_SSID_PSWD
I (239922) smartconfig: T|pswd: 123456789
I (239922) smartconfig: T|ssid: IOT_DEMO_TEST
I (239922) smartconfig: T|bssid: a4:56:02:47:30:07
I (239922) wifi: ic_disable_sniffer
I (239922) sc: SC_STATUS_LINK
I (239932) sc: SSID:IOT_DEMO_TEST
I (239932) sc: PASSWORD:123456789
I (240062) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (241042) wifi: state: init -> auth (b0)
I (241042) wifi: state: auth -> assoc (0)
I (241052) wifi: state: assoc -> run (10)
I (241102) wifi: connected with IOT_DEMO_TEST, channel 1
I (244892) event: ip: 192.168.0.152, mask: 255.255.255.0, gw: 192.168.0.1
I (244892) sc: WiFi Connected to ap
I (247952) sc: SC_STATUS_LINK_OVER
I (247952) sc: Phone ip: 192.168.0.31
I (247952) sc: smartconfig over
```

3. Bluetooth

Компоненты BluFi ESP32-C3 помогают передавать через Bluetooth LE SSID и пароль, которые можно использовать для подключения к точке доступа.

(1) API

API для компонентов BluFi определены в файле *esp_blufi_api.h* и показаны в табл. 7.7.

Таблица 7.7. API для компонентов BluFi

API	Описание
<code>esp_blufi_register_callbacks()</code>	Регистрация событий обратного вызова BluFi
<code>esp_blufi_profile_init()</code>	Инициализировать профиль BluFi
<code>esp_blufi_profile_deinit()</code>	Деинициализировать профиль BluFi
<code>esp_blufi_send_wifi_conn_report()</code>	Отправка отчетов о подключении к Wi-Fi
<code>esp_blufi_send_wifi_list()</code>	Отправить список Wi-Fi
<code>esp_blufi_get_version()</code>	Получить версию текущего профиля BluFi
<code>esp_blufi_close()</code>	Отключить устройство
<code>esp_blufi_send_error_info()</code>	Отправлять сообщения об ошибках BluFi
<code>esp_blufi_send_custom_data()</code>	Отправить пользовательские данные

(2) Структура программы

- **Обработка событий Wi-Fi:** надзор за подключением к Wi-Fi, отключением, повторным подключением, сканированием и т. д., как подробно описано в предыдущих разделах.
- **Обработка событий NETIF:** получение IP-адреса. Подробная информация представлена в предыдущих разделах.
- **Обработка событий BluFi:** определяется полученным запросом. События BluFi перечислены в табл. 7.8.

Таблица 7.8. События Blu-Fi

Событие	Описание
<code>ESP_BLUFI_EVENT_INIT_FINISH</code>	Инициализация функции BluFi, наименование устройства и отправка указанных широковещательных данных
<code>ESP_BLUFI_EVENT_DEINIT_FINISH</code>	Обработка события настройки деинициализации
<code>ESP_BLUFI_EVENT_BLE_CONNECT</code>	Подключение к Bluetooth LE и перевод устройства в безопасный режим
<code>ESP_BLUFI_EVENT_BLE_DISCONNECT</code>	Установка отключения Bluetooth LE и повторного подключения
<code>ESP_BLUFI_EVENT_SET_WIFI_OPMODE</code>	Перевод ESP32-C3 в рабочий режим
<code>ESP_BLUFI_EVENT_REQ_CONNECT_TO_AP</code>	Отключение от оригинального Wi-Fi и подключение к указанному Wi-Fi

Событие	Описание
ESP_BLUFI_EVENT_REQ_DISCONNECT_FROM_AP	Отключиться от точки доступа, которая в данный момент подключена к ESP32-C3
ESP_BLUFI_EVENT_REPORT_ERROR	Отправлять сообщения об ошибках
ESP_BLUFI_EVENT_GET_WIFI_STATUS	Получить статус Wi-Fi, включая текущий режим Wi-Fi и подключен ли он
ESP_BLUFI_EVENT_RECV_SLAVE_DISCONNECT_BLE	Уведомить BluFi о закрытии соединения GATT
ESP_BLUFI_EVENT_RECV_STA_BSSID	Войти в режим STA и получить BSSID ²⁸ целевой точки доступа
ESP_BLUFI_EVENT_RECV_STA_SSID	Войти в режим STA и получить SSID целевой точки доступа
ESP_BLUFI_EVENT_RECV_STA_PASSWD	Войти в режим STA и получить пароль целевой точки доступа
ESP_BLUFI_EVENT_RECV_SOFTAP_SSID	Войти в режим SoftAP и получить пользовательский SSID точки доступа
ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD	Войти в режим SoftAP и получить пользовательский пароль точки доступа
ESP_BLUFI_EVENT_RECV_SOFTAP_MAX_CONN_NUM	Установить максимальное количество подключенных устройств в режиме SoftAP
ESP_BLUFI_EVENT_RECV_SOFTAP_AUTH_MODE	Вход в режим аутентификации в режиме SoftAP
ESP_BLUFI_EVENT_RECV_SOFTAP_CHANNEL	Установить канал в режиме SoftAP
ESP_BLUFI_EVENT_GET_WIFI_LIST	Получить список SSID, канал и MAC-адрес STA при сканировании
ESP_BLUFI_EVENT_RECV_CUSTOM_DATA	Распечатать полученные данные и отформатировать их в соответствии с приложением

- **Основная программа:** инициализация Wi-Fi, инициализация и включение контроллера Bluetooth, инициализация и включение протокола Bluetooth, получение адреса Bluetooth и версии BluFi, обработка событий Bluetooth GAP и создание событий BluFi.

²⁸ В отличие от обычного SSID (service set identification) – текстового имени точки доступа в Wi-Fi-сеть, BSSID (basic service set identification) – базовое имя точки доступа, обычно представляющее собой MAC-адрес сетевого адаптера точки доступа (см. стр. 141). В отличие от SSID, которое можно скрыть от отображения в списке доступных сетей, BSSID легко определяется при сканировании Wi-Fi-сетей.

(3) Функциональная проверка

Для начала установите EspBlufi на свой телефон. Включите Wi-Fi и включите устройство.

Через последовательный порт вы получите выходной лог, содержащий следующие строки:

```
I (516) phy_init: phy_version 500,985899c, Apr 19 2021, 16:05:08
I (696) wifi:set rx active PTI: 0, rx ack PTI: 12, and default PTI: 1
I (908) wifi:mode : sta (30:ae:a4:80:41:55)
I (908) wifi:enable tsf
W (706) BTDM_INIT: esp_bt_controller_mem_release not implemented, return OK
I (706) BTDM_INIT: BT controller compile version [9c99115]
I (716) coexist: coexist rom version 9387209
I (726) BTDM_INIT: Bluetooth MAC: 30:ae:a4:80:41:56
I (746) BLUFI_EXAMPLE: BD ADDR: 30:ae:a4:80:41:56
I (1198) BLUFI_EXAMPLE: BLUFI VERSION 0102
I (1198) BLUFI_EXAMPLE: BLUFI init finish
```

Примечание

Вы можете скачать приложение по следующему адресу:
<https://www.espressif.com/en/support/download/apps>.

а. Запуск

Откройте приложение на телефоне и потяните вниз для обновления данных. Вы увидите информацию о ближайших устройствах Bluetooth на экране, как показано на рис. 7.40.

б. Связь

Выберите модуль ESP32-C3 BLUFI DEVICE, чтобы получить подробную информацию об устройстве. Нажмите **Connect** (Подключиться) для подключения по Bluetooth. При подключении вы увидите интерфейс, показанный на рис. 7.41.

Выходной лог выглядит следующим образом:

```
I (32736) BLUFI_EXAMPLE: BLUFI ble connect
```

с. Предоставление сети

Нажмите **Provision network** (Предоставить сеть) на рис. 7.41, чтобы войти в интерфейс инициализации, показанный на рис. 7.42.

д. Стационарное соединение

Нажмите **OK** на рис. 7.42, чтобы настроить сеть. Если конфигурация прошла успешно, вы увидите подключенный интерфейс STA, показанный на рис. 7.43. Подробности о подключении STA в Wi-Fi-режиме будут отображаться в нижней части экрана, в том числе BSSID и SSID точки доступа и состояние соединения.

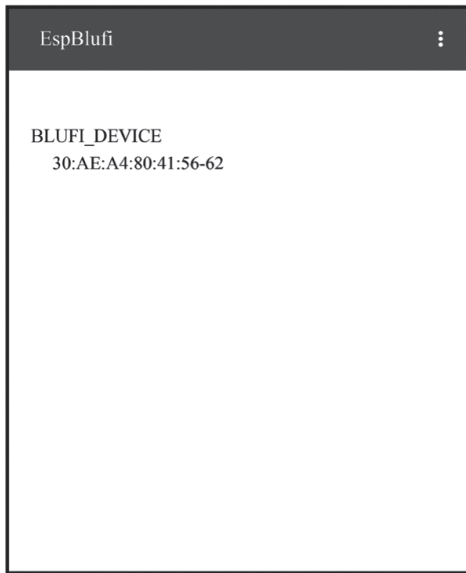


Рисунок 7.40. Запуск EspBlufi

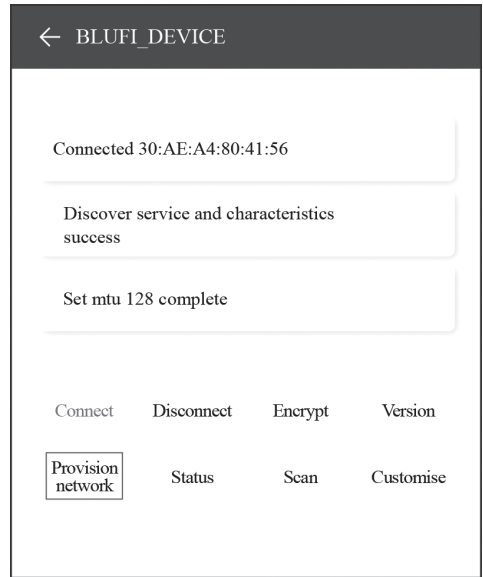


Рисунок 7.41. Подключение Bluetooth



Рисунок 7.42. Предоставление сети

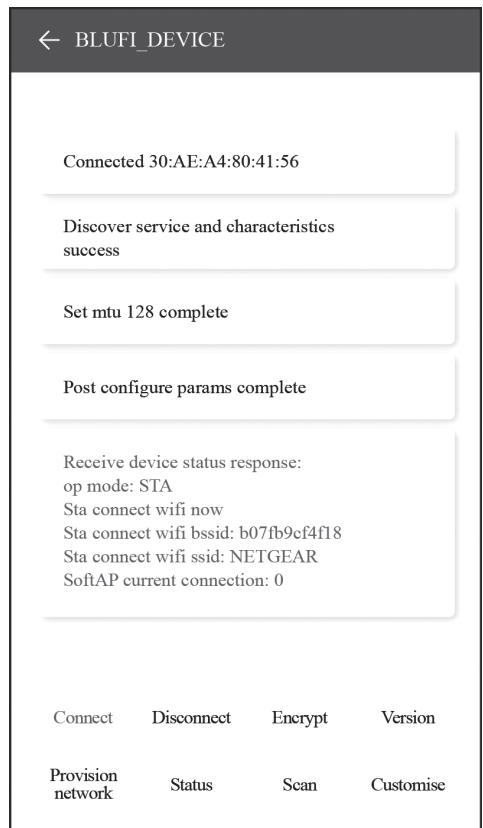


Рисунок 7.43. STA подключена

Выходной лог будет содержать следующие строки:

```
I (63756) BLUFI_EXAMPLE: BLUFI Set WIFI opmode 1
I (63826) BLUFI_EXAMPLE: Recv STA SSID NETGEAR
I (63866) BLUFI_EXAMPLE: Recv STA PASSWORD 12345678
I (63936) BLUFI_EXAMPLE: BLUFI request wifi connect to AP
I (65746) wifi:new: <8,2>, old: <1,0>, ap: <255,255>, sta: <8,2>, prof:1
I (66326) wifi:state: init -> auth (b0)
I (67326) wifi:state: auth -> init (200)
I (67326) wifi:new: <8,0>, old: <8,2>, ap: <255,255>, sta: <8,2>, prof:1
I (69516) wifi:new: <10,0>, old: <8,0>, ap: <255,255>, sta: <10,0>, prof:1
I (69516) wifi:state: init -> auth (b0)
I (69566) wifi:state: auth -> assoc (0)
I (69626) wifi:state: assoc -> run (10)
I (69816) wifi:connected with NETGEAR, aid = 1, channel 10, BW20, bssid =
5c:02:14:03:a5:7d
I (69816) wifi:security: WPA2-PSK, phy: bgn, rssi: -48
I (69826) wifi:pm start, type: 1
I (69826) wifi:set rx beacon pti, rx_bcn_pti: 14, bcn_timeout: 14, mt_pti:
25000, mt_time: 10000
I (69926) wifi:BCnInt:102400, DTIM:1 W (70566) wifi:idx:0 (ifx:0,
5c:02:14:03:a5:7d), tid:0, ssn:2, winSize:64
I (71406) esp_netif_handlers: sta ip: 192.168.31.145, mask: 255.255.255.0,
gw:192.168.31.1
```

7.5. Практика: конфигурация Wi-Fi в проекте Smart Light

В этом разделе мы начнем с программирования соединения Wi-Fi на основе драйвера регулировки светодиода, а затем приведем пример умной настройки Wi-Fi в проекте Smart Light.

7.5.1 Соединение Wi-Fi в проекте Smart Light

Изучив основы подключения к Wi-Fi, мы можем применить их на практике на основе ESP32-C3 и интегрировать функции Wi-Fi в соответствии с требованиями приложения, чтобы обеспечить API для инициализации сети и соединения с Wi-Fi.

1. Инициализация драйвера

Этот API определяет параметры для ESP32-C3, такие как контакты GPIO, время затухания, цикл мигания, частоту ШИМ, источник тактового сигнала контроллера ШИМ и разрешение рабочего цикла ШИМ. Подробности см. в главе 5.

```
1. app_driver_init();
```

2. Инициализация NVS

Перед инициализацией Wi-Fi необходимо инициализировать библиотеку NVS, так как для компонента Wi-Fi нужно получать и сохранять определенные параметры в энергонезависимой памяти. API выглядит следующим образом:

```
1. nvs_flash_init();
```

3. Инициализация Wi-Fi

Этот API обрабатывает события LwIP и Wi-Fi и инициализирует драйверы Wi-Fi.

```
1. wifi_initialize();
```

4. Инициализация соединения Wi-Fi

Этот API реализует настройку Wi-Fi, запускает драйвер Wi-Fi и ожидает завершения подключения к Wi-Fi.

```
1. wifi_station_initialize();
```

Исходный код

Чтобы получить код соединения Wi-Fi из проекта драйвера диммирования светодиодов, см. https://github.com/espressif/book-esp32c3-iot-projects/tree/main/device_firmware/3_wifi_connection.

7.5.2. Умная настройка Wi-Fi

Теперь перейдем к настройке Wi-Fi на основе ESP32-C3. Аналогично соединению Wi-Fi, мы интегрируем интеллектуальные функции настройки Wi-Fi в соответствии с требованиями приложения, чтобы предоставить API для умной настройки Wi-Fi.

После инициализации подготовки программа проверит ее статус. Если устройство было предоставлено, программа закончит соединение Wi-Fi, используя информацию о маршрутизаторе; в противном случае она выведет QR-код, чтобы вы могли начать предоставление.

```
1. wifi_prov_mgr_initialize();
```

Чтобы интегрировать в проект код для настройки сети Bluetooth из раздела 7.5.1, пожалуйста, обратитесь к https://github.com/espressif/book-esp32c3-iot-projects/tree/main/device_firmware/4_network_config.

С помощью приложения ESP BLE Provisioning App вы можете скомпилировать и запустить код на плате разработчика. Результат следующий.

Примечание

Вы можете скачать приложение по адресу:
<https://www.espressif.com/en/support/download/apps>.

Если устройство не было инициализировано, вы получите вывод в логе, показанный на рис. 7.44.

Если устройство было подготовлено, вы в логе получите следующий текст:

```
I (399) wifi station: Application driver initialization
I (399) gpio: GPIO[9]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1|
PullDown: 0| Intr:0
I (429) wifi station: NVS Flash initialization
I (429) wifi station: Wi-Fi initialization
I (549) wifi station: Wi-Fi Provisioning initialization
I (549) wifi station: Already provisioned, starting Wi-Fi STA
I (809) wifi station: wifi_station_initialize finished.
I (1939) wifi station: got ip:192.168.3.105
```

```
I (679) wifi_prov_mgr: Provisioning started with service name : PROV_082118
I (689) wifi station: Scan this QR code from the provisioning application for Provisioning.
I (699) QRCODE: Encoding below text with ECC LVL 0 & QR Code Version 10
I (709) QRCODE: {"ver":"v1","name":"PROV_082118","pop":"abcd1234","transport":"ble"}
```



I (899) wifi station: If QR code is not visible, copy paste the below URL in a browser.
[https://espressif.github.io/esp-jumpstart/qrcode.html?data={"ver":"v1","name":"PROV_082118","pop":"abcd1234","transport":"ble"}](https://espressif.github.io/esp-jumpstart/qrcode.html?data={)

Рисунок 7.44. Вывод лога, если устройство не подготовлено

7.6. Резюме

В этой главе мы впервые представили две важные технологии настройки устройств IoT в сети Wi-Fi и Bluetooth. Затем рассмотрели некоторые концепции и механизмы конфигурации сети Wi-Fi, включая SoftAP, SmartConfig, Bluetooth, прямую настройку сети, RouterConfig, ZeroConfig и конфигурацию сети для телефонной точки доступа. Мы также проанализировали код для настройки сети SoftAP, SmartConfig и Bluetooth в сочетании с программированием Wi-Fi. Наконец, опробовали умную настройку Wi-Fi для проекта Smart Light.

В главе 7 мы узнали об основах Wi-Fi и Bluetooth, а также о некоторых методах настройки сети. В главе 7 вы должны были научиться настройке устройств и подключению их к маршрутизаторам Wi-Fi. Учитывая полученные знания, в данной главе мы рассказываем, как обеспечить локальное управление устройствами на основе Wi-Fi и Bluetooth и реализовать его с помощью ESP32-C3. Приводятся пояснения определений и самого процесса локального управления, а также некоторые общие протоколы связи для этой цели, что поможет вам создать собственную инфраструктуру локального управления для умных источников света на основе ESP32-C3.

8.1. Введение в локальное управление

В этом разделе сначала разбирается, что такое локальное управление, а также каковы условия его использования, сценарии и преимущества. Затем будет рассказано о функции обнаружения устройств и протоколах передачи данных, участвующих в локальном управлении, и о том, как выбрать среду передачи данных для этой цели. Прочитав данный раздел, вы получите полное представление о локальном управлении устройств.

Как следует из названия, локальное (местное) управление относится к управлению управляемыми устройствами в пределах определенного расстояния с помощью ряда методов, таких как аппаратные выключатели, сенсорные кнопки, инфракрасное дистанционное управление, смартфоны и компьютерные сети. Он вездесущ в нашей повседневной жизни: настройка кондиционеров с помощью инфракрасных пультов дистанционного управления, управление оборудованием с помощью голосовых команд, а также включение бытового освещения с помощью выключателей или приложения для смартфона. Понятие и технология локального управления стали глубоко интегрированы во все аспекты нашей повседневной жизни.

Вы могли заметить, что некоторые из перечисленных выше примеров выполняются аппаратно, как выключатели в цепи или инфракрасное дистанционное управление, в то время как другие включают распознавание голоса и технологии передачи данных. В этой книге мы сосредоточимся на технологии передачи данных для интернета вещей и поможем вам создать собственную локальную систему управления для управления умными светильниками.

В интернете вещей каждое устройство должно передавать команды с помощью определенного способа связи. Вот некоторые распространенные способы.

- **Использование Wi-Fi или Ethernet.** Как правило, устройства на основе Wi-Fi и Ethernet изначально работают через стек протоколов TCP/IP, что

значительно снижает трудоемкость адаптации протокола при разработке. При выполнении локальной связи понадобится шлюз или Wi-Fi-роутер.

- **Использование технологий беспроводной связи малого радиуса действия**, таких как Bluetooth и Zig-Bee, подходящих для передачи данных между низкоскоростными и маломощными устройствами.

В соответствии с функциональными характеристиками ESP32-C3 мы в этой главе представим две обычно используемые технологии локального управления, а именно через Wi-Fi или Bluetooth в пределах локальной сети.

Топология сети, основанная на Wi-Fi, показана на рис. 8.1. Устройства, посылающие команды (смартфон или ПК), должны находиться в той же локальной сети, что и контролируемое устройство, и отправлять данные через Wi-Fi. Для управления через Bluetooth нет необходимости в маршрутизаторах Wi-Fi, поскольку данные через Bluetooth могут напрямую передаваться между смартфоном и управляемым устройством.

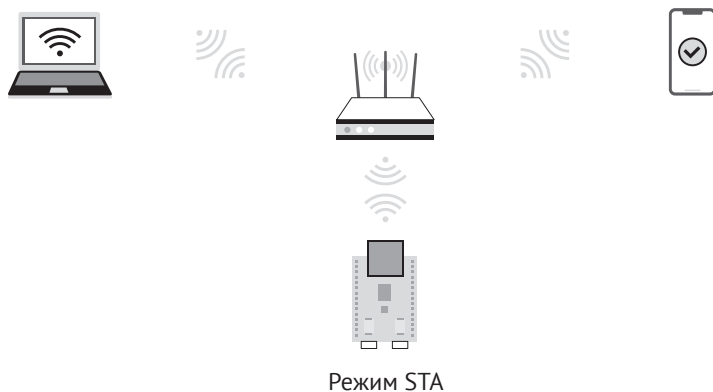


Рисунок 8.1. Топология сети, основанная на Wi-Fi внутри локальной сети

Использование Bluetooth проще, чем использование Wi-Fi, поскольку Bluetooth не требует маршрутизаторов Wi-Fi. Однако на практике, если устройство IoT хочет получить доступ к облачным платформам, ему нужны маршрутизаторы Wi-Fi для подключения к интернету, а затем к облачной платформе. Кроме того, смартфоны в основном подключены к Wi-Fi-роутерам. Поэтому локальная сеть обычно включает маршрутизаторы Wi-Fi, что делает удобным использование Wi-Fi для локального управления. Если IoT-устройству не нужен доступ к облаку, Bluetooth может быть хорошим вариантом для локального управления. Вы можете выбрать один из способов в зависимости от того, нужен ли вашему устройству доступ к облачным платформам.

- Если нужен, рекомендуется выбрать Wi-Fi, так как он поддерживает управление одним устройством несколькими смартфонами одновременно и его пропускная способность больше, чем Bluetooth. Вы можете использовать Bluetooth только для настройки сети, а затем остановить его для экономии ресурсов ESP32-C3.
- Если *не нужен*, вы можете использовать Bluetooth вместо Wi-Fi для обмена данными между смартфоном и управляемым устройством.

8.1.1. Применение локального управления

Большинство IoT-устройств подключены к облачным платформам, пересылающим со смартфонов команды дистанционного управления. Такое управление зависит от интернета, предоставляемого маршрутизаторами Wi-Fi для поддержания связи между контролируемым устройством и облачными платформами. Но как только маршрутизаторы Wi-Fi отключатся от интернета, удаленное управление будет парализовано. В этом случае локальное управление будет хорошим дополнением к отправке команд, предотвращая тем самым IoT-устройства от полной поломки из-за сетевых сбоев.

Как показано на рис. 8.1, инфраструктура локального управления на основе Wi-Fi в локальной сети состоит из Wi-Fi-маршрутизатора, управляющего устройства и управляемого (контролируемого) устройства. Управляющим устройством может быть смартфон или компьютер, поддерживающий стек протоколов TCP/IP. Он должен быть подключен к тому же маршрутизатору Wi-Fi, что и управляемое устройство, чтобы наверняка оказаться в той же локальной сети во время передачи данных.

Что касается локального управления по Bluetooth, Wi-Fi-роутеры не нужны. Смартфоны могут напрямую подключаться к контролируемому устройству через Bluetooth и реализовать передачу данных по типу «точка-точка».

8.1.2. Преимущества локального управления

Локальное управление требует только передачи данных внутри локальной сети, а не через интернет. Поэтому оно работает с меньшей задержкой и более быстрым откликом. Более того, поскольку данные взаимодействуют только внутри локальной сети, состоящей из контролируемого устройства, маршрутизатора Wi-Fi и управляющих устройств, вероятность кражи или изменения данных ниже, поэтому налицо повышение конфиденциальности и безопасности данных. Кроме того, локальное управление экономит пропускную способность интернета и не уязвимо к его отключению. Пока две стороны находятся в одной локальной сети или связаны через Bluetooth, управление может быть сохранено. Для некоторых продуктов без доступа к облачным платформам локальное управление – единственный способ, с помощью которого смартфоны могут ими управлять.

Учитывая эти преимущества, компании IoT все чаще отдают предпочтение локальному управлению. Все больше и больше SDK и продуктов поддерживают функции локального управления. Например, ESP RainMaker, полная платформа IoT, разработанная Espressif, включает приложение для смартфона для реализации локального управления.

8.1.3. Обнаружение управляемых устройств с помощью смартфонов

Для локального управления на основе беспроводной среды передачи данных Wi-Fi данные передаются с помощью стека протокола TCP/IP. Следующие две проблемы должны быть решены, так как смартфон не подключен непосредственно к управляемому устройству. Как смартфон находит устройство и как смартфон с ним общается?

Вопрос, как смартфон находит контролируемое устройство, равносильно вопросу, как смартфон узнает IP-адрес устройства. Поскольку все данные передаются на уровне IP, получение IP-адреса контролируемого устройства является обязательным условием для последующих обменов данными. Вы можете подумать: «Я могу войти в интерфейс маршрутизатора Wi-Fi и проверить IP-адрес контролируемого устройства прямо через интерфейс Wi-Fi-роутера, верно?» Да, таким образом, безусловно, можно получить IP-адрес устройства. Однако ручной запрос IP-адресов полностью противоречит первоначальному замыслу IoT-технологий. Таким образом, требуется технология для автоматического обнаружения контролируемых устройств. Эта проблема будет подробно рассмотрена в разделе 8.2.

Для локальных систем управления, основанных на управлении через Bluetooth, как вы можете узнать из главы 7, Bluetooth контролируемого устройства будет транслировать собственную информацию, и смартфону нужно только сканировать Bluetooth. Обнаружение контролируемого устройства через Bluetooth намного проще, чем через Wi-Fi. После того как смартфон подключится к Bluetooth управляемого устройства, он может на него отправлять данные. Кроме того, передача Bluetooth не зависит от стека протоколов TCP/IP, поскольку имеет собственный протокол передачи. Эти вопросы будут представлены подробно в разделе 8.3.

8.1.4. Передача данных между смартфонами и устройствами

Как смартфоны взаимодействуют с управляемыми устройствами?

При использовании беспроводной среды передачи Wi-Fi после получения IP-адреса контролируемого устройства смартфон может обмениваться данными с ним по протоколу TCP/IP или протоколу UDP. Вообще говоря, как приемник управляемое устройство получает команды управления, отправляемые смартфоном; а смартфон как отправитель отправляет команды управления на управляемое устройство. Таким образом, управляемое устройство воспроизводит роль сервера; а смартфон играет роль клиента, что позволяет нескольким клиентам отправлять управляющие команды на сервер. Эти вопросы будут подробно представлены в разделе 8.3.

8.2. Общие методы локального обнаружения

В разделе 8.1.4 мы упомянули, как обнаружить контролируемые устройства в локальной сети с помощью беспроводной среды передачи Wi-Fi. В стеке протоколов TCP/IP обнаружение контролируемого устройства означает получение его IP-адреса.

В локальной сети вопрос о том, как получить IP-адрес узла, заслуживает изучения. Общий протокол для прямого получения IP-адреса называется Reverse Address Resolution Protocol (RARP, протокол обратного разрешения адресов).

Этот протокол отправляет пакеты запросов, зная MAC-адрес партнера и входной сервер, анализирует свою собственную таблицу ARP, чтобы получить IP-адрес запрошенного устройства MAC. Если вы знакомы с локальными сетями

ми, вы можете сразу ассоциироваться с протоколом разрешения адресов (ARP), который отправляет пакеты запросов, содержащие IP-адрес. Одноуровневое устройство или шлюз отвечает MAC-адресом, соответствующим IP-адресу, после запроса собственной таблицы ARP. ARP и RARP – это пара протоколов, которые взаимно разрешают адреса сетевого уровня (IP-адреса) и адреса канального уровня (MAC-адреса). Однако этим двум протоколам необходимо изначально знать либо IP-адрес, либо MAC-адрес. Эта особенность создает трудности для приложений IoT, поскольку эти адреса трудно получить в локальной сети. В этом подразделе будет представлена технология локального обнаружения, которая действительно подходит для интернета вещей.

Локальное обнаружение заключается в обнаружении информации об узлах в локальной сети, включая адресную информацию для связи с ними, информацию службы приложений, поддерживаемую узлами, и персонализированную информацию. Например, mDNS (Multicast DNS, о котором будет рассказываться в подразделе 8.2.4) является широко используемым локальным протоколом обнаружения. Принцип локального обнаружения заключается в отправке сообщения, и одноранговый узел сообщит отправителю информацию о своем устройстве после получения сообщения. Теперь необходимо решить проблему, как партнер может получить сообщение, отправленное отправителем.

На самом деле если вы разбираетесь в классификации IP-адресов, то будете знать, что помимо более широко используемой связи «один к одному» (одноадресной рассылки) есть также групповая («один ко многим») и прямая (широковещательная) связь «один ко всем». IP-адреса можно разделить на индивидуальные адреса, групповые адреса и широковещательные адреса. Одноадресная рассылка предполагает знание IP-адреса партнера, поэтому не подходит для локального обнаружения. Групповая и широковещательная рассылки не требуют знания IP-адреса партнера. Устройства отправляют сообщения на определенные адреса и могут получать сообщения, пока отслеживается данный адрес. Таким образом, групповая и широковещательная рассылки подходят для обнаружения устройств в локальной сети, и одноранговый узел может получить отправленное сообщение с помощью этих двух технологий.

8.2.1. Широковещательная передача

Широковещательная передача относится к отправке сообщений всем возможным получателям в сети. Есть два основных приложения такого вещания:

- поиск хоста в локальной сети;
- уменьшение потока пакетов в локальной сети, чтобы одно сообщение могло уведомить все хосты.

Общие сообщения широковещательных приложений включают в себя:

Протокол разрешения адресов (ARP)

Его можно использовать для трансляции ARP-запроса в локальную сеть: «Сообщите, пожалуйста, какой MAC-адрес устройства с IP-адресом a.b.c.d». Широковещательная рассылка ARP – это широковещательная передача MAC на уровне 2 (канальный уровень), а не IP-трансляция на уровне 3 (сетевой уровень).

Протокол динамической конфигурации хоста (DHCP)

Если в локальной сети есть DHCP-сервер, DHCP-клиент отправляет DHCP-запрос на IP-адрес назначения (обычно 255.255.255.255), и DHCP-сервер в той же сети может получить запрос и ответить соответствующим IP-адресом.

Широковещательная передача в основном использует протокол UDP (подробности см. в подразделе 8.3.3) вместо TCP-протокола (подробности см. в подразделе 8.3.1), так как он больше подходит для одноадресной рассылки.

1. Широковещательные адреса

Широковещательные адреса включают широковещательные MAC-адреса на уровне 2 (канальном уровне) (FF:FF:FF:FF:FF:FF) и широковещательные IP-адреса на уровне 3 (сетевом уровне) (255.255.255.255). Далее они называются адресами уровня 2 и адресами уровня 3. Этот раздел в основном знакомит с адресами уровня 3. Как правило, когда адрес уровня 3 сообщения равен 255, адрес уровня 2 обычно содержит все FF. Это связано с тем, что сообщение с адресом уровня 3, содержащее только 255, означает, что все устройства в локальной сети получат это сообщение. Если сообщение с адресом уровня 2 содержит фрагменты, отличные от FF, сообщение будет отброшено во время обработки адреса уровня 2 приемного устройства. Для принимающего устройства, если адрес сообщения уровня 2 не является широковещательным адресом, то есть ни локальным MAC-адресом, ни групповым MAC-адресом (например, 01:00:5E:XX:XX:XX), он будет отброшен и не будет обработан. Поэтому, как правило, если адрес уровня 3 является широковещательным адресом, адрес уровня 2 такой же.

Адреса IPv4 состоят из идентификатора подсети и идентификатора узла. Например, для устройства с IP-адресом 192.168.3.4 и маской подсети 255.255.255.0 идентификаторы подсети и хоста вычисляются из IP-адреса и маски подсети. В этом примере идентификатор подсети 192.168.3.0 и идентификатор узла 4. Если идентификатор подсети и идентификатор узла равны 255, это широковещательная рассылка; это также широковещательный адрес, когда только идентификатор хоста равен всем 255. Например, если вы имеете подсеть 192.168.1/24, то 192.168.1.255 – широковещательный адрес этой подсети.

Вы можете задаться вопросом, в чем разница между широковещательным адресом с идентификатором подсети и идентификатором хоста из всех 255 и широковещательным адресом только с идентификатором хоста 255?

Диапазон широковещательной рассылки первого адреса больше, чем у конкретной подсети. Например, маршрутизатор Wi-Fi имеет две подсети: 192.168.1/24 и 192.168.2/24. Если хост с IP-адресом 192.168.1.2 в подсети 192.168.1/24 отправляет сообщение на адрес назначения 192.168.1.255, маршрутизатор Wi-Fi будет пересылать сообщение только на хост в подсети 192.168.1/24 и не будет пересылать его на хост в подсети 192.168.2/24. Если хост отправляет сообщение на адрес назначения 255.255.255.255, маршрутизатор Wi-Fi пересылает сообщение хостам в обеих подсетях. Следовательно, широковещательный адрес с идентификатором хоста 255 также называется широковещательным адресом, направленным на подсеть. С помощью широковещательной рассылки, направленной на адрес подсети, вы можете отправлять сообщения в указанную подсеть, так что эти сообщения не будут отправляться в подсети, которые не нуждаются в них в локальной сети, тем самым экономя сетевые ресурсы.

2. Реализация широковещательного отправителя с использованием сокета

Исходный код

Исходный код функции `esp_send_broadcast()` доступен по адресу https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/broadcast_discovery.

Функция `esp_send_broadcast()` отправляет широковещательные пакеты UDP с запросом «Are you Espressif IOT Smart Light» («являетесь вы IOT-устройством Espressif Smart Light») в локальную сеть, а затем ждет ответа от партнера. Эта функция использует стандартный интерфейс сокетов Berkeley, также известный как сокет BSD. Сокет Berkeley является общим сетевым интерфейсом в системах UNIX, который не только поддерживает различные сетевые типы, но и является механизмом связи между внутренними процессами. Программирование протоколов TCP/UDP, описываемое в этой книге, использует сокеты Berkeley. Если вы заинтересовались, то можете прочитать «UNIX Network Programming (Volume 1): Socket Networking API», опубликованное Posts & Telecommunications Press, чтобы узнать больше о программировании сокетов Berkeley. В этой книге мы лишь кратко познакомим вас с программированием сокетов.

В данном разделе мы расскажем, как использовать `socket(AF_INET, SOCK_DGRAM, 0)` для создания сокета UDP, а потом используем `setsockopt()`, чтобы включить поддержку сокета для вещания.

Затем установим адрес назначения для вещания на все 255 и порт на 3333 и вызовем `sendto()` для отправки сообщения. Вы можете определить, отправляются ли данные успешно в соответствии с возвращаемым значением функции `sendto()`. Код выглядит следующим образом:

```
1. esp_err_t esp_send_broadcast(void)
2. {
3.     int opt_val = 1;
4.     esp_err_t err = ESP_FAIL;
5.     struct sockaddr_in from_addr = {0};
6.     socklen_t from_addr_len = sizeof(struct sockaddr_in);
7.     char udp_recv_buf[64 + 1] = {0};
8.
9.     //Create an IPv4 UDP socket
10.    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
11.    if (sockfd == -1) {
12.        ESP_LOGE(TAG, "Create UDP socket fail");
13.        return err;
14.    }
15.
16.    //Set SO_BROADCAST socket option, and use it to send broadcast
17.    int ret = setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &opt_val,
18.                        sizeof(int));
19.    if (ret < 0) {
20.        ESP_LOGE(TAG, "Set SO_BROADCAST option fail");
21.        goto exit;
22.    }
23.
24.    //Set broadcast destination address and port
```

```

25.  struct sockaddr_in dest_addr = {
26.      .sin_family = AF_INET,
27.      .sin_port = htons(3333),
28.      .sin_addr.s_addr = htonl(INADDR_BROADCAST),
29.  };
30.
31.  char *broadcast_msg_buf = "Are you Espressif IOT Smart Light";
32.
33.  //Call sendto() to send broadcast data
34.  ret = sendto(sockfd, broadcast_msg_buf, strlen(broadcast_msg_buf), 0,
35.              (struct sockaddr *)&dest_addr,
36.              sizeof(struct sockaddr));
37.  if (ret < 0) {
38.      ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
39.  } else {
40.      ESP_LOGI(TAG, "Message sent successfully");
41.      ret = recvfrom(sockfd, udp_rcv_buf, sizeof(udp_rcv_buf) - 1, 0,
42.                   (struct sockaddr *)&from_addr,
43.                   (socklen_t *)&from_addr_len);
44.      if (ret > 0) {
45.          ESP_LOGI(TAG, "Receive udp unicast from %s:%d, data is %s",
46.                  inet_ntoa (((struct sockaddr_in *)&from_addr)->sin_addr),
47.                  ntohs(((struct sockaddr_in *)&from_addr)->sin_port),
48.                  udp_rcv_buf);
49.          err = ESP_OK;
50.      }
51.  }
52.  exit:
53.      close(sockfd);
54.      return err;
55.  }

```

3. Реализация широковещательного приемника с использованием сокета

Исходный код

Для получения исходного кода функции `esp_receive_broadcast()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/broadcast_discovery.

Функция `esp_receive_broadcast()` реализует прием широковещательных пакетов и одноадресные ответы. Логика реализации получателя такая же, как у отправителя.

Сначала создайте сокет UDP и установите исходный адрес и номер порта сообщения для прослушивания. Как правило, сокет используется в качестве сервера. Адрес источника сообщения установлен на 0.0.0.0, что означает, что исходный адрес сообщения не определен. Вызовите `bind()` для привязки сокета, а затем используйте `recvfrom()` для получения сообщения. Когда широковещательный пакет принесет данные «Are you Espressif IOT Smart Light», IP-адрес и номер порта отправителя сохранятся в `from_addr`, что будет отправлено ему в виде одноадресной передачи. Код выглядит следующим образом:

```

1. esp_err_t esp_receive_broadcast(void)
2. {
3.     esp_err_t err = ESP_FAIL;
4.     struct sockaddr_in from_addr = {0};
5.     socklen_t from_addr_len = sizeof(struct sockaddr_in);
6.     char udp_server_buf[64+1] = {0};
7.     char *udp_server_send_buf = "ESP32-C3 Smart Light https 443";
8.
9.     //Create an IPv4 UDP socket
10.    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
11.    if (sockfd == -1) {
12.        ESP_LOGE(TAG, "Create UDP socket fail");
13.        return err;
14.    }
15.
16.    //Set broadcast destination address and port
17.    struct sockaddr_in server_addr = {
18.        .sin_family = AF_INET,
19.        .sin_port = htons(3333),
20.        .sin_addr.s_addr = htonl(INADDR_ANY),
21.    };
22.
23.    int ret = bind(sockfd, (struct sockaddr *)&server_addr,
24.                  sizeof(server_addr));
25.    if (ret < 0) {
26.        ESP_LOGE(TAG, "Bind socket fail");
27.        goto exit;
28.    }
29.
30.    //Call recvfrom()to receive broadcast data
31.    while (1) {
32.        ret = recvfrom(sockfd, udp_server_buf, sizeof(udp_server_buf) - 1, 0,
33.                      (struct sockaddr *)&from_addr,
34.                      (socklen_t *)&from_addr_len);
35.        if (ret > 0) {
36.            ESP_LOGI(TAG, "Receive udp broadcast from %s:%d, data is %s",
37.                    inet_ntoa(((struct sockaddr_in *)&from_addr->sin_addr),
38.                    ntohs(((struct sockaddr_in *)&from_addr->sin_port),
39.                    udp_server_buf);
40.        //Upon reception of broadcast request, send data communication port of peer through unicast
41.        if (!strcmp(udp_server_buf, "Are you espressif IOT Smart Light")){
42.            ret = sendto(sockfd, udp_server_send_buf, strlen(udp_server_send_buf),
43.                        0, (struct sockaddr *)&from_addr, from_addr_len);
44.            if (ret < 0) {
45.                ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
46.            } else {
47.                ESP_LOGI(TAG, "Message sent successfully");
48.            }
49.        }
50.    }
51. }
52. exit:
53.     close(sockfd);
54.     return err;
55. }

```

4. Результат запуска

Добавьте код отправителя и получателя в пример станции Wi-Fi, чтобы убедиться, что они подключены к тому же Wi-Fi-роутеру. Лог отправки рассылки выглядит следующим образом:

```
I (774) wifi:mode : sta (c4:4f:33:24:65:f1)
I (774) wifi: enable tsf
I (774) wifi station: wifi_init_sta finished
I (784) wifi:new: <6,0>, old: <1,0>, ap: <255,255>, sta: <6,0>, prof:1
I (794) wifi:state: auth -> assoc (0)
I (814) wifi:state: assoc -> run (10)
I (834) wifi: connected with myssid, aid = 1, channel 6, BW20, bssid = 34:29:12:43:c5:40
I (834) wifi:security: WPA2-PSK, phy: bgn, rssi: -23
I (834) wifi: pm start, type: 1
I (884) wifi: AP's beacon interval = 102400 us, DTIM period = 1
I (1544) esp_netif handlers: sta ip: 192.168.3.5, mask: 255.255.255.0, gw: 192.168.3.1
I (1544) wifi station: got ip:192.168.3.5 I (1544) wifi station: connected to ap SSID: myssid password: 12345678
I (1554) wifi station: Message sent successfully
I (1624) wifi station: Receive udp unicast from 192.168.3.80:3333, data is ESP32-C3 Smart Light https 443
```

Лог приема рассылки выглядит так:

```
I (1450) wifi:new: <6,0>, old: <1,0>, ap: <255,255>, sta: <6,0>, prof:1
I (2200) wifi:state: init -> auth (b0)
I (2370) wifi:state: auth -> assoc (0)
I (2380) wifi:state: assoc -> run (10)
I (2440) wifi: connected with myssid, aid = 2, channel 6, BW20, bssid = 34:29:12:43:c5:40
Chapter 8. Local Control 161
12:43:c5:40
I (2450) wifi:security: WPA2-PSK, phy: bgn, rssi: -30
I (2460) wifi: pm start, type: 1
I (2530) wifi: AP's beacon interval = 102400 us, DTIM period = 1
I (3050) esp_netif_handlers: sta ip: 192.168.3.80, mask: 255.255.255.0, gw: 192.168.3.1
I (3050) wifi station: got ip:192.168.3.80
I (3050) wifi station: connected to ap SSID: myssid password: 12345678
W (17430) wifi: <ba-add>idx:0 (ifx:0, 34:29:12:43:c5:40), tid:5, ssn:0, winSize:64
I (26490) wifi station: Receive udp broadcast from 192.168.3.5:60520, data is Are you Espressif IOT Smart Light
I (26500) wifi station: Message sent successfully
I (382550) wifi station: Receive udp broadcast from 192.168.3.5:63439, data is Are you Espressif IOT Smart Light
I (382550) wifi station: Message sent successfully
```

В логe широковещательной рассылки указано, что отправитель отправил широковещательный UDP-пакет с данными «Are you Espressif IOT Smart Light». В логe приема указано, что приемник прослушивает широковещательный пакет локальной сети и при получении такого пакета отвечает одноадресным пакетом с данными «ESP32-C3 Smart Light https 443». Таким образом, локальные устройства могут быть обнаружены. После получения одноадресного ответа от

получателя отправитель может подтвердить IP-адрес и знать протокол приложения и номер порта для последующей передачи произведенных данных.

Широковещательный протокол локальной сети может выполнять функцию обнаружения устройств. Однако широковещательная рассылка запроса на обнаружение всем устройствам в локальной сети вызовет определенную нагрузку на локальную сеть и хост. Таким образом, обнаружение устройств с помощью широковещательной рассылки – не самый хороший выбор.

8.2.2. Групповая передача (Multicast)

Групповая рассылка относится к отправке сообщений заинтересованным получателям. По сравнению с двумя «крайностями» схем одноадресной и широковещательной адресации (одному или всем), групповая является компромиссным решением. Как следует из названия, групповая рассылка поддерживает концепцию группы, то есть хост может отправить сообщение на групповой адрес, и все хосты, присоединившиеся к этой группе, могут получить сообщение. Это чем-то похоже на широковещательную рассылку по подсетям, но более гибкое решение, потому что хост может присоединиться к определенной группе или выйти из нее в любое время, тем самым уменьшая нагрузку на локальную сеть и хосты.

Протокол управления группами Интернета (Internet Group Management Protocol, IGMP) – это протокол управления групповой (multicast) передачей данных в сетях, основанных на протоколе IP, непосредственно между членами группы и соседними маршрутизаторами. Для групповой рассылки маршрутизаторы Wi-Fi должны поддерживать протокол IGMP.

1. Групповая адресация

Адреса назначения групповых сообщений используют IP-адрес класса D. Первый байт начинается с двоичных чисел 1110 и находится в диапазоне от 224.0.0.0 до 239.255.255.255. Так как групповой IP-адрес идентифицирует группу хостов, он может использоваться только в качестве адреса назначения, а не адреса источника, который всегда является адресом индивидуальной рассылки.

Мультикаст-группа – это группа, идентифицируемая по определенному адресу. Когда участники внутри или за пределами группы отправляют сообщение на этот адрес, члены группы, идентифицированные по групповому адресу, могут получить сообщение. Мультикаст-группы могут быть постоянными или временными. Официально назначенные групповые адреса присваиваются постоянным мультикаст-группам, в то время как те, которые не являются ни зарезервированными, ни постоянными, присваиваются временным мультикаст-группам. Количество хостов в постоянной и временной группах является динамическим и может даже быть равным нулю.

Групповые адреса классифицируются следующим образом:

- 224.0.0.0 ~ 224.0.0.255: зарезервированные групповые адреса (постоянные мультикаст-группы). Адрес 224.0.0.0 не выделяется, остальные используют его для протоколов маршрутизации.
- 224.0.1.0 ~ 224.0.1.255: общедоступные групповые адреса, которые можно использовать в интернете.

- 224.0.2.0 ~ 238.255.255.255: адреса мультикаст-рассылки, доступные пользователям (временные группы), которые действительны во всей сети.
- 239.0.0.0 ~ 239.255.255.255: групповые адреса для локального управления, действуют только в пределах определенных локальных диапазонов.

2. Реализация группового отправителя с использованием сокета

Исходный код

Для получения исходного кода функции `esp_join_multicast_group()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/multicast_discovery.

Реализация групповой отправки более сложна, чем широковещательной. Групповая отправка требует настройки интерфейса отправки многоадресных пакетов. Если вам нужно получать пакеты от определенной мультикаст-группы, к ней нужно присоединиться. Функция `esp_join_multicast_group()` реализует настройку интерфейса групповой рассылки и функцию присоединения к мультикаст-группе. Функция `esp_send_multicast()` реализует создание, привязку, настройку адреса доставки и порта обычных сокетов UDP, а также функции отправки и получения. Кроме того, также добавлены настройки времени жизни TTL, чтобы гарантировать, что мультикаст-группа может выполняться только в локальной сети этого маршрута. Код выглядит следующим образом:

```
1. #define MULTICAST_IPV4_ADDR "232.10.11.12"
2. int esp_join_multicast_group(int sockfd)
3. {
4.     struct ip_mreq imreq = { 0 };
5.     struct in_addr iaddr = { 0 };
6.     int err = 0;
7.
8.     //Configure sending interface of multicast group
9.     esp_netif_ip_info_t ip_info = { 0 };
10.    err = esp_netif_get_ip_info(esp_netif_get_handle_from_ifkey("WIFI_STA_DEF"),
11.                               &ip_info);
12.    if (err != ESP_OK) {
13.        ESP_LOGE(TAG, "Failed to get IP address info. Error 0x%x", err);
14.        goto err;
15.    }
16.    inet_addr_from_ip4addr(&iaddr, &ip_info.ip);
17.    err = setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_IF, &iaddr, sizeof(struct in_addr));
18.    if (err < 0) {
19.        ESP_LOGE(TAG, "Failed to set IP_MULTICAST_IF. Error %d", errno);
20.        goto err;
21.    }
22.
23.    //Configure the address of monitoring multicast group
24.    inet_aton(MULTICAST_IPV4_ADDR, &imreq.imr_multiaddr.s_addr);
25.
26.    //Configure the socket to join the multicast group
27.    err = setsockopt(sockfd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
28.                   &imreq, sizeof(struct ip_mreq));
29.    if (err < 0) {
30.        ESP_LOGE(TAG, "Failed to set IP_ADD_MEMBERSHIP. Error %d", errno);
31.    }
```

```

32. err:
33.     return err;
34. }
35.
36. esp_err_t esp_send_multicast(void)
37. {
38.     esp_err_t err = ESP_FAIL;
39.     struct sockaddr_in saddr = {0};
40.     struct sockaddr_in from_addr = {0};
41.     socklen_t from_addr_len = sizeof(struct sockaddr_in);
42.     char udp_rcv_buf[64 + 1] = {0};
43.
44.     //Create an IPv4 UDP socket
45.     int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
46.     if (sockfd == -1) {
47.         ESP_LOGE(TAG, "Create UDP socket fail");
48.         return err;
49.     }
50.
51.     //Bind socket
52.     saddr.sin_family = PF_INET;
53.     saddr.sin_port = htons(3333);
54.     saddr.sin_addr.s_addr = htonl(INADDR_ANY);
55.     int ret = bind(sockfd, (struct sockaddr *)&saddr, sizeof(struct sockaddr_in));
56.     if (ret < 0) {
57.         ESP_LOGE(TAG, "Failed to bind socket. Error %d", errno);
58.         goto exit;
59.     }
60.
61.     //Set multicast TTL to 1, limiting the multicast packet to one route
62.     uint8_t ttl = 1;
63.     ret = setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(uint8_t));
64.     if (ret < 0) {
65.         ESP_LOGE(TAG, "Failed to set IP_MULTICAST_TTL. Error %d", errno);
66.         goto exit;
67.     }
68.
69.     //Join the multicast group
70.     ret = esp_join_multicast_group(sockfd);
71.     if (ret < 0) {
72.         ESP_LOGE(TAG, "Failed to join multicast group");
73.         goto exit;
74.     }
75.
76.     //Set multicast destination address and port
77.     struct sockaddr_in dest_addr = {
78.         .sin_family = AF_INET,
79.         .sin_port = htons(3333),
80.     };
81.     inet_aton(MULTICAST_IPV4_ADDR, &dest_addr.sin_addr.s_addr);
82.     char *multicast_msg_buf = "Are you espressif IOT Smart Light";
83.
84.     //Call sendto() to send multicast data
85.     ret = sendto(sockfd, multicast_msg_buf, strlen(multicast_msg_buf), 0,
86.                 (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
87.     if (ret < 0) {

```



```

88.     ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
89. } else {
90.     ESP_LOGI(TAG, "Message sent successfully");
91.     ret = recvfrom(sockfd, udp_recv_buf, sizeof(udp_recv_buf) - 1, 0,
92.                  (struct sockaddr *)&from_addr,
93.                  (socklen_t *)&from_addr_len);
94.     if (ret > 0) {
95.         ESP_LOGI(TAG, "Receive udp unicast from %s:%d, data is %s",
96.                  inet_ntoa(((struct sockaddr_in *)&from_addr)->sin_addr),
97.                  ntohs(((struct sockaddr_in *)&from_addr)->sin_port),
98.                  udp_recv_buf);
99.         err = ESP_OK;
100.    }
101. }
102.exit:
103. close(sockfd);
104. return err;
105.}

```

3. Реализация группового приемника с использованием сокета

Исходный код

Для получения исходного кода функции `esp_recv_multicast()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/multicast_discovery.

Реализация получателя групповой рассылки аналогична реализации отправителя групповой рассылки, для которой требуется указание интерфейса групповых пакетов и присоединяемой группы. Функция `esp_recv_multicast()` реализует создание, привязку, настройку адреса назначения и порта обычных сокетов UDP, а также функции отправки и получения. Кроме того, поскольку в этом примере необходимо отправить групповую рассылку, устанавливается время жизни (TTL). Код выглядит следующим образом:

```

1. esp_err_t esp_recv_multicast(void)
2. {
3.     esp_err_t err = ESP_FAIL;
4.     struct sockaddr_in saddr = {0};
5.     struct sockaddr_in from_addr = {0};
6.     socklen_t from_addr_len = sizeof(struct sockaddr_in);
7.     char udp_server_buf[64+1] = {0};
8.     char *udp_server_send_buf = "ESP32-C3 Smart Light https 443";
9.
10.    //Create an IPv4 UDP socket
11.    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
12.    if (sockfd == -1) {
13.        ESP_LOGE(TAG, "Create UDP socket fail");
14.        return err;
15.    }
16.
17.    //Bind socket
18.    saddr.sin_family = PF_INET;

```

```

19.  saddr.sin_port = htons(3333);
20.  saddr.sin_addr.s_addr = htonl(INADDR_ANY);
21.  int ret = bind(sockfd, (struct sockaddr *)&saddr, sizeof(struct sockaddr_in));
22.  if (ret < 0) {
23.      ESP_LOGE(TAG, "Failed to bind socket. Error %d", errno);
24.      goto exit;
25.  }
26.
27.  //Set multicast TTL to 1, limiting the multicast packet to one route
28.  uint8_t ttl = 1;
29.  ret = setsockopt(sockfd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(uint8_t));
30.  if (ret < 0) {
31.      ESP_LOGE(TAG, "Failed to set IP_MULTICAST_TTL. Error %d", errno);
32.      goto exit;
33.  }
34.
35.  //Join the multicast group
36.  ret = esp_join_multicast_group(sockfd);
37.  if (ret < 0) {
38.      ESP_LOGE(TAG, "Failed to join multicast group");
39.      goto exit;
40.  }
41.
42.  //Call recvfrom() to receive multicast data
43.  while (1) {
44.      ret = recvfrom(sockfd, udp_server_buf, sizeof(udp_server_buf) - 1, 0,
45.                  (struct sockaddr *)&from_addr,
46.                  (socklen_t *)&from_addr_len);
47.      if (ret > 0) {
48.          ESP_LOGI(TAG, "Receive udp multicast from %s:%d, data is %s",
49.                  inet_ntoa(((struct sockaddr_in *)&from_addr->sin_addr),
50.                  ntohs(((struct sockaddr_in *)&from_addr->sin_port),
51.                  udp_server_buf);
52.          //Upon reception of multicast request, send data communication port of peer
          through unicast
53.          if (!strcmp(udp_server_buf, "Are you Espressif IOT Smart Light")) {
54.              ret = sendto(sockfd, udp_server_send_buf, strlen(udp_server_send_buf),
55.                          0, (struct sockaddr *)&from_addr, from_addr_len);
56.              if (ret < 0) {
57.                  ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
58.              } else {
59.                  ESP_LOGI(TAG, "Message sent successfully");
60.              }
61.          }
62.      }
63.  }
64.  exit:
65.      close(sockfd);
66.      return err;
67. }

```

4. Результат запуска

Добавьте код отправителя и получателя в пример станции Wi-Fi, чтобы убедиться, что они подключены к одному и тому же Wi-Fi-роутеру. Лог групповой рассылки выглядит следующим образом:

```

I (752) wifi :mode : sta (c4:4f:33:24:65:f1)
I (752) wifi:enable tsf
I (752) wifi station: wifi init sta finished.
I (772) wifi:new: <6,0>, old: <1,0>, ap: <255,255>, sta: <6,0>, prof:1
I (772) wifi:state: init -> auth (b0)
I (792) wifi:state: auth -> assoc (0)
I (802) wifi:state: assoc -> run (10)
I (822) wifi:connected with myssid, aid = 2, channel 6, BW20, bssid = 34:29:12:43:c5:40
I (822) wifi:security: WPA2-PSK, phy: bgn, rssi: -17
I (822) wifi: pm start, type: 1
I(882) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (1542) esp_netif_handlers: sta ip: 192.168.3.5, mask: 255.255.255.0, gw: 192.168.3.1
I (1542) wifi station: got ip:192.168.3.5 I (1542) wifi station: connected to ap
SSID: myssid password: 123456 8
I (1552) wifi station: Message sent successfully
I (1632) wifi station: Receive udp unicast from 192.168.3.80:3333, data is ESP32-C3 Smart Light https 443

```

Лог приема групповой рассылки выглядит следующим образом:

```

I (806) wifi:state: init -> auth (b0)
I (816) wifi:state: auth -> assoc (0)
I (836) wifi:state: assoc -> run (10)
I (966) wifi:connected with myssid, aid = 1, channel 6, BW20, bssid = 34:29:12:43:c5:40
I (966) wifi:security: WPA2-PSKI phy: bgn, rssi: -29
I (976) wifi:pm start, type: 1
I (1066) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (2056) esp_netif_handlers: sta ip: 192.168.3.80, mask: 255.255.255.0, gw: 192.168.3.1
I (2056) wifi station: got ip:192.168.3.80
I (2056) wifi station: connected to ap SSID: myssid password: 12345678
W (18476) wifi: <ba-add>idx:0 (ifx:0, 34:29:12:43:c5:40), tid:0, ssn:4, winSize: 64
W (23706) wifi: <ba-add>idx:1 (ifx:0, 34:29:12:43:c5:40), tid:5, ssn:0, winSize: 64
I (23706) wifi station: Receive udp multicast from 192.168.3.5:3333, data is Are you Espressif IOT Smart Light

```

Как и в логах отправки, отправитель отправляет пакеты с определенными данными, а получатель сообщает отправителю протокол приложения и номер порта передачи данных.

8.2.3. Сравнение широковещательной и групповой рассылки

Сравнение широковещательной и групповой рассылки показано в табл. 8.1. Как можно видеть, групповая рассылка имеет меньшую нагрузку на полосу пропускания, и устройства в локальной сети могут подключаться и отключаться от групп, предварительно созданных для получения и отправки определенных данных, что делает ее более гибкой. Для широковещательной рассылки все устройства в локальной сети получают пакет, который увеличит нагрузку на другие устройства в локальной сети, а также нагрузку на ее пропускную способность.

Таблица 8.1. Сравнение широковещательной и групповой рассылки

Сравнение	Широковещательная (broadcast)	Групповая (multicast)
Принцип	Пакеты отправляются на все хосты, подключенные к сети	Пакеты отправляются только определенным получателям в сети
Передача	«Один ко всем»	«Один ко многим»
Управление	Нет необходимости в управлении группой	Требуется управление группой
Сеть	Может вызвать потери и задержки пропускной способности сети	Контролируемая пропускная способность сети
Скорость	Медленно	Быстро

8.2.4. Протокол групповых приложений mDNS для локального обнаружения

В компьютерных сетях протокол Multicast DNS (mDNS) преобразует имена хостов в IP-адреса в небольших сетях, которые не включают локальные DNS-серверы. Это сервер нулевой конфигурации²⁹. mDNS имеет в основе тот же интерфейс программирования, формат пакета и принцип действия, как и традиционная система доменных имен (DNS).

Протокол mDNS был впервые предложен Биллом Вудкоком и Биллом Мэннингом в IETF в 2000 году. Согласно Википедии, он был окончательно опубликован в качестве стандартного протокола в RFC 6762 Стюартом Чеширом и Марком Крохмалом в 2013 году и реализован Apple Bonjour и программными пакетами Avahi с открытым исходным кодом. Включен в большинство дистрибутивов Linux.

1. Введение в протокол mDNS

mDNS – это протокол разрешения доменных имен для локальных сетей, который использует порт 5353 и групповой адрес 224.0.0.251. Это прикладной протокол, работающий на UDP. В отличие от традиционных протоколов DNS, mDNS не требует DNS-сервера для получения доменного имени, что позволяет избежать настройки DNS-серверов в локальных сетях.

После того как хост с включенной службой mDNS присоединится к локальной сети, он сначала отправит групповое сообщение по адресу локальной сети 224.0.0.251: «Who am I? What is my IP address? What are the services and port numbers I provide?» («Кто я? Какой у меня IP-адрес? Какие я предоставляю сервисы и номера портов?»). После получения сообщения другие хосты со службой mDNS, включенные в локальной сети, ответят на эти вопросы. Если хост хочет запросить имя домена mDNS, он сначала запросит информацию в собственном кеше. Если там имя не найдено, он отправит групповой запрос в локальную сеть, чтобы запросить IP-адрес, службы и номера портов доменного имени.

²⁹ Zero Configuration – набор технологий, которые автоматически создают IP-сеть без необходимости конфигурации вручную или специальных сервисов (например, DHCP и DNS).

Тогда как хост может отличить доменное имя DNS от mDNS, когда идет запрос доменного имени? Доменные имена mDNS отличаются от доменных имен DNS суффиксом «.local».

2. Использование компонента mDNS на основе ESP-IDF

Примечание: компонент mDNS

ESP-IDF предоставляет компонент mDNS, который помогает разрабатывать приложения. Вы можете найти сведения о службе mDNS в руководстве по программированию ESP-IDF Programming Guide для соответствующих интерфейсов. Компонент mDNS можно найти на странице <https://github.com/espressif/esp-idf/tree/v4.3.2/components/mdns>. По поводу службы mDNS, пожалуйста, посетите <https://bookc3.espressif.com/mdns>.

В этом разделе в основном рассказывается, как использовать компонент mDNS для разработки устройств, которые должны быть обнаруженными.

```
1. esp_err_t esp_mdns_discovery_start(void)
2. {
3.     char *host_name = "my_smart_light";
4.     char *instance_name = "esp32c3_smart_light";
5.
6.     //Initialise the mDNS component
7.     if (mdns_init() != ESP_OK) {
8.         ESP_LOGE(TAG, "mdns_init fail");
9.         return ESP_FAIL;
10.    }
11.
12.    //Set host name (the DNS domain name tag to be queried by other hosts)
13.    if (mdns_hostname_set(host_name) != ESP_OK) {
14.        ESP_LOGE(TAG, "mdns_hostname_set fail");
15.        goto err;
16.    }
17.    ESP_LOGI(TAG, "mdns hostname set to: [%s]", host_name);
18.
19.    //Set mDNS instance name to be discovered by mDNS LAN
20.    if (mdns_instance_name_set(instance_name) != ESP_OK) {
21.        ESP_LOGE(TAG, "mdns_instance_name_set fail");
22.        goto err;
23.    }
24.
25.    //Set service TXT field data (optional)
26.    mdns_txt_item_t serviceTxtData[1] = {
27.        {"board", "esp32c3"}
28.    };
29.
30.    //Add HTTP service; port 80 corresponds to mDNS service.
    //The second parameter (application layer
31.    protocol) and the third parameter (transport layer protocol) need to
    correspond to each other.
```

```

32.   if (mdns_service_add(instance_name, "_http", "_tcp", 80,
33.                           serviceTxtData, 1) != ESP_OK) {
34.       ESP_LOGE(TAG, "mdns_instance_name_set fail");
35.       goto err;
36.   }
37.
38.   //Set service TXT field data
39.   if (mdns_service_txt_item_set("_http", "_tcp", "path", "/foobar") !=ESP_OK){
40.       ESP_LOGE(TAG, "mdns_service_txt_item_set fail");
41.       goto err;
42.   }
43.   return ESP_OK;
44. err:
45.   mdns_free();
46.   return ESP_FAIL;
47. }

```

Приведенный код реализует службу mDNS с доменным именем `my_smart_light` и сетевой узел `esp32c3_smart_light`. Другие ваши хосты могут запрашивать узел `esp32c3_smart_light` через службу mDNS. Хост умного освещения сообщит свое доменное имя (`my_smart_light`), соответствующий IP-адрес, предоставляемый сервис (HTTP), соответствующий порт сервера (80) и текстовое имя узла (`path=/foobar board=esp32c3`).

Исходный код

Полный код примера см. https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/mdns_discovery.

Вы можете использовать команду Windows `dns-sd -L esp32c3_smart_light_http` для запроса информации о хосте в локальной сети. Команда выглядит следующим образом:

```

c:\Users> dns-sd -L esp32c3 smart light http
Lookup esp32c3_smart_light._http._tcp.local
14:25:09.682 esp32c3_smart_light._http._tcp.local. can be reached at my_smart_
light.local.:80 (interface 6)
  path=/foobar board=esp32c3

```

Примечание: «Bonjour»

«Bonjour» – это программное обеспечение для настройки сети, которое поддерживает службу сети с нулевой конфигурацией и может автоматически обнаруживать компьютеры, устройства и службы в IP-сети. Его необходимо установить перед использованием команды `dns-sd`. Вы можете скачать «Bonjour» по адресу <https://bonjour.en.softonic.com/>.

Вы также можете использовать команду Linux `avahi-browse -a --resolve` для запроса службы информации обо всех хостах mDNS в локальной сети. Команда выглядит следующим образом:

```
# avahi-browse -a --resolve
= enp1s0 IPv4 esp32c3_smart_light Локальный веб-сайт
  имя хоста = [my_smart_light.local]
  адрес = [192.168.3.5]
  порт = [80]
  txt = ["board=esp32c3" "путь=/foobar"]
```

8.3. Общие протоколы связи для локальных данных

8.3.1. Протокол управления передачей (TCP)

TCP (Transmission Control Protocol) является одним из основных протоколов в семействе интернет-протоколов. В модели TCP/IP протокол TCP служит протоколом транспортного уровня, обеспечивая надежную передачу данных для протоколов уровня приложений, таких как HTTP, MQTT, FTP и т. д. Модель TCP/IP показана на рис. 8.2.

1. Введение в TCP

TCP – это ориентированный на установление соединения, надежный протокол связи транспортного уровня на основе потока байтов, определенный RFC 793 IETF.

- **Ориентированность на соединение.** Перед отправкой данных по TCP необходимо установить соединение между отправителем и получателем, что обычно называют трехсторонним подтверждением (three-way handshake).

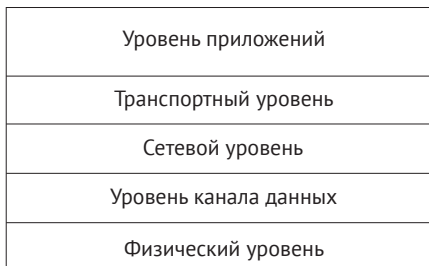


Рисунок 8.2. Модель TCP/IP

- **Надежность.** При отправке данных по протоколу TCP прием получателем может быть гарантирован. Если данные потеряны, они будут переданы повторно. TCP также может гарантировать, что получатель получит данные в порядке отправления.
- **На основе потока байтов.** При отправке данных с помощью TCP данные прикладного уровня идут первыми и записываются в буфер TCP. Затем TCP управляет передачей данных на основе байтового потока, созданного на прикладном уровне и не зависящего от длины сообщения. Это и называется протоколом на основе потока байтов.

Процесс TCP, отправляющий данные приложения верхнего уровня получателю, выглядит следующим образом:

- (1) прикладная программа верхнего уровня записывает данные приложения в буфер TCP;

- (2) буфер TCP упаковывает данные в сообщение TCP и отправляет его в сетевой слой;
- (3) получатель получает сообщение TCP и помещает его в буфер TCP;
- (4) после получения определенного количества данных данные сортируются и реорганизуются перед передачей прикладному уровню.

Процесс отправки и получения данных с помощью TCP показан на рис. 8.3.

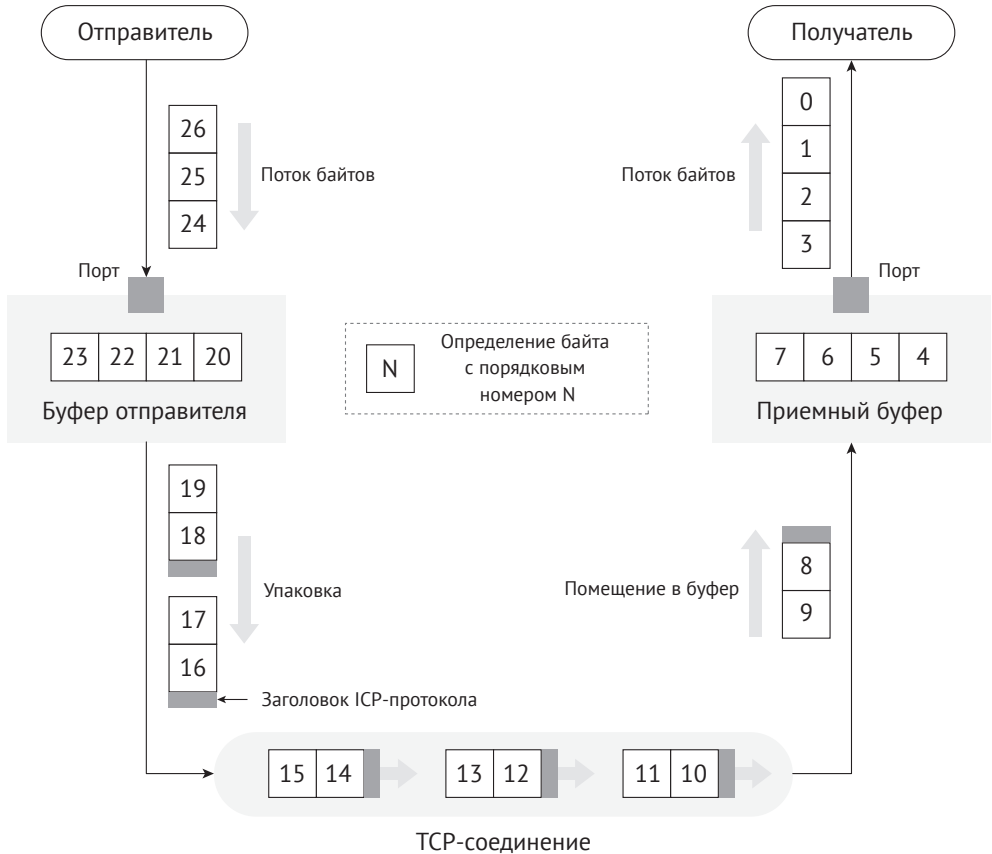


Рисунок 8.3. Процесс отправки и получения данных с использованием TCP

2. Создание TCP-сервера с использованием сокета

Исходный код

Для получения исходного кода функции `esp_create_tcp_server()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/tcp_socket.

Функция `esp_create_tcp_server()` может создать TCP-сервер, включая создание сокета TCP, настройку и привязку порта, прослушивание, получение и отправку данных. По сравнению с TCP-клиентами, UDP-серверами и клиентами поток

кода TCP-серверов более сложный и включает в себя две функции сокета, listen и accept, уникальные для TCP-серверов. Код выглядит следующим образом:

```
1. esp_err_t esp_create_tcp_server(void)
2. {
3.     int len;
4.     int keepAlive = 1;
5.     int keepIdle = 5;
6.     int keepInterval = 5;
7.     int keepCount = 3;
8.     char rx_buffer[128] = {0};
9.     char addr_str[32] = {0};
10.    esp_err_t err = ESP_FAIL;
11.    struct sockaddr_in server_addr;
12.
13.    //Create a TCP socket
14.    int listenfd = socket(AF_INET, SOCK_STREAM, 0);
15.    if (listenfd < 0) {
16.        ESP_LOGE(TAG, "create socket error");
17.        return err;
18.    }
19.    ESP_LOGI(TAG, "create socket success, listenfd : %d", listenfd);
20.
21.    //Enable SO_REUSEADDR, allowing the server to bind the connected address
22.    int opt = 1;
23.    int ret = setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &opt,
24.        sizeof(opt));
25.    if (ret < 0) {
26.        ESP_LOGE(TAG, "Failed to set SO_REUSEADDR. Error %d", errno);
27.        goto exit;
28.    }
29.
30.    //Bind the server to an interface with all-zero IP address and port number 3333
31.    server_addr.sin_family = AF_INET;
32.    server_addr.sin_addr.s_addr = INADDR_ANY;
33.    server_addr.sin_port = htons(3333);
34.    ret = bind(listenfd, (struct sockaddr *) &server_addr, sizeof(server_addr));
35.    if (ret < 0) {
36.        ESP_LOGE(TAG, "bind socket failed, socketfd: %d, errno : %d",
37.            listenfd, errno);
38.        goto exit;
39.    }
40.    ESP_LOGI(TAG, "bind socket success");
41.    ret = listen(listenfd, 1);
42.    if (ret < 0) {
43.        ESP_LOGE(TAG, "listen socket failed, socketfd : %d, errno : %d",
44.            listenfd, errno);
45.        goto exit;
46.    }
47.    ESP_LOGI(TAG, "listen socket success");
48.    while (1) {
49.        struct sockaddr_in source_addr;
50.        socklen_t addr_len = sizeof(source_addr);
51.
52.        //Wait for new TCP connection, and return the communicating socket with the peer
53.        int sock = accept(listenfd, (struct sockaddr *)&source_addr, &addr_len);
```

```

54.     if (sock < 0) {
55.         ESP_LOGE(TAG, "Unable to accept connection: errno %d", errno);
56.         break;
57.     }
58.
59.     //Enable TCP keep-alive function to prevent zombie clients
60.     setsockopt(sock, SOL_SOCKET, SO_KEEPALIVE, &keepAlive, sizeof(int));
61.     setsockopt(sock, IPPROTO_TCP, TCP_KEEPIDLE, &keepIdle, sizeof(int));
62.     setsockopt(sock, IPPROTO_TCP, TCP_KEEPINTVL, &keepInterval, sizeof(int));
63.     setsockopt(sock, IPPROTO_TCP, TCP_KEEPCNT, &keepCount, sizeof(int));
64.     if (source_addr.sin_family == PF_INET) {
65.         inet_ntoa_r(((struct sockaddr_in *)&source_addr)->sin_addr,
66.             addr_str, sizeof(addr_str) - 1);
67.     }
68.     ESP_LOGI(TAG, "Socket accepted ip address: %s", addr_str);
69.     do {
70.         len = recv(sock, rx_buffer, sizeof(rx_buffer) - 1, 0);
71.         if (len < 0) {
72.             ESP_LOGE(TAG, "Error occurred during receiving: errno %d", errno);
73.         } else if (len == 0) {
74.             ESP_LOGW(TAG, "Connection closed");
75.         } else {
76.             rx_buffer[len] = 0;
77.             ESP_LOGI(TAG, "Received %d bytes: %s", len, rx_buffer);
78.         }
79.     } while (len > 0);
80.     shutdown(sock, 0);
81.     close(sock);
82. }
83. exit:
84.     close(listenfd);
85.     return err;
86. }

```

Приведенный код создает TCP-сервер и прослушивает данные приложения на порту 3333. Опция сокета `SO_REUSEADDR` позволяет серверу привязываться к адресу уже установленного подключения, что полезно для кода на стороне сервера. Вариант сокета `SO_KEEPALIVE` включает функцию проверки активности TCP, которая может обнаруживать ненормально отключенных клиентов и предотвращать их нагрузку на серверные процессы. Варианты сокетов `TCP_KEEPIDLE`, `TCP_KEEPINTVL` и `TCP_KEEPCNT` соответствуют времени простоя со времени последних данных, отправленных узлом, интервалу времени для отправки сообщений проверки активности TCP и максимальному количеству повторных попыток отправки сообщения соответственно. Например, если TCP-клиент устанавливает для `TCP_KEEPIDLE` значение 5, это означает, что если нет обмена данными между клиентом и сервером в течение 5 секунд, клиент должен отправить сообщение проверки активности TCP на сервер; если клиент устанавливает для `TCP_KEEPINTVL` значение 5, это означает, что если клиент отправляет TCP-сообщение проверки активности на сервер и сервер не отвечает в течение 5 секунд, клиенту необходимо повторно отправить сообщение на сервер; если клиент устанавливает `TCP_KEEPCNT` в 3, это означает, что клиент может повторить отправку TCP-сообщения подтверждения активности на сервер максимум 3 раза.

3. Создание TCP-клиента с использованием сокета

Исходный код

Для получения исходного кода функции `esp_create_tcp_client()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/tcp_socket.

Функция `esp_create_tcp_client()` может создать TCP-соединение между клиентом и сервером, включая создание TCP-сокета, настройку адреса назначения и порта, подключение и отправку данных. Код выглядит следующим образом:

```
1. #define HOST_IP "192.168.3.80"
2. #define PORT 3333
3.
4. esp_err_t esp_create_tcp_client(void)
5. {
6.     esp_err_t err = ESP_FAIL;
7.     char *payload = "Open the light";
8.     struct sockaddr_in dest_addr;
9.     dest_addr.sin_addr.s_addr = inet_addr(HOST_IP);
10.    dest_addr.sin_family = AF_INET;
11.    dest_addr.sin_port = htons(PORT);
12.
13.    //Create a TCP socket
14.    int sock = socket(AF_INET, SOCK_STREAM, 0);
15.    if (sock < 0) {
16.        ESP_LOGE(TAG, "Unable to create socket: errno %d", errno);
17.        return err;
18.    }
19.    ESP_LOGI(TAG, "Socket created, connecting to %s:%d", HOST_IP, PORT);
20.
21.    //Connect to the TCP server
22.    int ret = connect(sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
23.    if (ret != 0) {
24.        ESP_LOGE(TAG, "Socket unable to connect: errno %d", errno);
25.        close(sock);
26.        return err;
27.    }
28.    ESP_LOGI(TAG, "Successfully connected");
29.
30.    //Send TCP data
31.    ret = send(sock, payload, strlen(payload), 0);
32.    if (ret < 0) {
33.        ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
34.        goto exit;
35.    }
36.    err = ESP_OK;
37. exit:
38.    shutdown(sock, 0);
39.    close(sock);
40.    return err;
41. }
```

После того как клиент устанавливает TCP-соединение с сервером, он отправляет TCP-данные **Open the light** (Включить свет) на сервер. Помимо использо-

вания сокетов TCP, клиент также может применять инструменты отладки TCP для имитации клиента TCP-соединения.

На основе приведенного выше кода клиента и сервера TCP, при реализации управления умным светом через смартфон, вы можете загрузить код TCP-сервера на умный светильник и код TCP-клиента на смартфон. После установления TCP-соединения между смартфоном и умным светильником смартфон может отправлять данные. Например, в приведенном выше коде клиент TCP отправляет данные **Open the light**; и после того, как умный светильник получит данные, он может включить свет, подтянув уровень соответствующего вывода GPIO.

8.3.2. Протокол передачи гипертекста (HyperText Transfer Protocol, HTTP)

HTTP – это прикладной протокол транспортного уровня. Передача данных с помощью HTTP – основа Всемирной паутины (WWW, или Web). Протокол определяет формат и метод передачи данных между клиентами и серверами. Клиенты могут использовать HTTP для получения (GET) состояния (включено/выключено) умного освещения или осуществлять (POST) его включение/выключение через HTTP-запросы, и каждая операция будет иметь ответ от партнера. Таким образом, HTTP является более функциональным и более удобен в приложениях, чем простой TCP.

1. Введение в HTTP

HTTP – это стандарт для запросов и ответов между клиентами (пользователями) и серверами (веб-сайтами). Клиент устанавливает TCP-соединение с сервером через веб-браузер, поисковый робот или другие инструменты, а затем отправляет запросы на чтение данных сервера, загрузку данных или форм на сервер и сначала читает статус ответа сервера, например «HTTP/1.1 200 OK», а потом возвращенный контент (например, запрошенные файлы, сообщения об ошибках или другую информацию). Ресурсы, запрашиваемые через HTTP, идентифицируются унифицированными идентификаторами ресурсов (URI).

В версиях HTTP 0.9 и 1.0 TCP-соединение закрывается после каждого запроса и ответа. В версии 1.1 HTTP был введен механизм поддержания соединения, позволяя соединению повторять несколько запросов и ответов, уменьшая время, затраченное на установление соединений TCP и сетевые накладные расходы перед каждым запросом данных.

Общие методы HTTP-запроса включают в себя:

- **GET**: запрос указанного ресурса URI;
- **POST**: отправка данных на указанный ресурс URI и запрос на их обработку сервером (например, отправка формы или загрузка файла);
- **DELETE**: запрос сервера на удаление ресурса, указанного в URI.

Для локального управления умными источниками света вы можете использовать метод GET для получения их статуса и использовать метод POST для управления ими.

2. Создание HTTP-сервера с использованием компонента ESP-IDF

Исходный код

Для получения исходного кода функции `esp_start_webserver()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/https_server.

Функция `esp_start_webserver()` может создать HTTP-сервер. Функции обратного вызова, соответствующие операциям GET и POST на стороне сервера, определяются как `esp_light_get_handler()` и `esp_light_set_handler()` соответственно и должны быть зарегистрированы через функцию `httpd_register_uri_handler()` после вызова функции `httpd_start()` на стороне сервера.

```
1. char buf[100] = "{\"status\": true}";
2. //Callback function of the HTTP GET request
3. esp_err_t esp_light_get_handler(httpd_req_t *req)
4. {
5.     //Send data in JSON containing the status of smart lights to the client
6.     httpd_resp_send(req, buf, strlen(buf));
7.     return ESP_OK;
8. }
9.
10. //Callback function of the HTTP POST request
11. esp_err_t esp_light_set_handler(httpd_req_t *req)
12. {
13.     int ret, remaining = req->content_len;
14.     memset(buf, 0, sizeof(buf));
15.     while (remaining > 0) {
16.         //Read HTTP request data
17.         if ((ret = httpd_req_recv(req, buf, remaining)) <= 0) {
18.             if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
19.                 continue;
20.             }
21.             return ESP_FAIL;
22.         }
23.         remaining -= ret;
24.     }
25.     ESP_LOGI(TAG, "%.s", req->content_len, buf);
26.
27.     //TODO: Read and parse the data; then control the smart light
28.     return ESP_OK;
29. }
30.
31. //Callback function corresponding to GET
32. static const httpd_uri_t status = {
33.     .uri = "/light",
34.     .method = HTTP_GET,
35.     .handler = esp_light_get_handler,
36. };
37.
38. //Callback function corresponding to POST
39. static const httpd_uri_t ctrl = {
40.     .uri = "/light",
```

```

41.     .method = HTTP_POST,
42.     .handler = esp_light_set_handler,
43. };
44.
45. esp_err_t esp_start_webserver()
46. {
47.     httpd_handle_t server = NULL;
48.     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
49.     config.lru_purge_enable = true;
50.
51.     //Start the HTTP server
52.     ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);
53.     if (httpd_start(&server, &config) == ESP_OK) {
54.         //Set the callback function corresponding to the HTTP URI
55.         ESP_LOGI(TAG, "Registering URI handlers");
56.         httpd_register_uri_handler(server, &status);
57.         httpd_register_uri_handler(server, &ctrl);
58.         return ESP_OK;
59.     }
60.     ESP_LOGI(TAG, "Error starting server!" );
61.     return ESP_FAIL;
62. }

```

Приведенный выше код реализует HTTP-сервер для запроса и установки состояния умного светильника. При доступе к `http://[ip]/light` через браузер он вернет состояние умного света: `{"status": true}` (как показано на рис. 8.4) или `{"status": false}`.



Рисунок 8.4. Использование HTTP для запроса состояния умного светильника

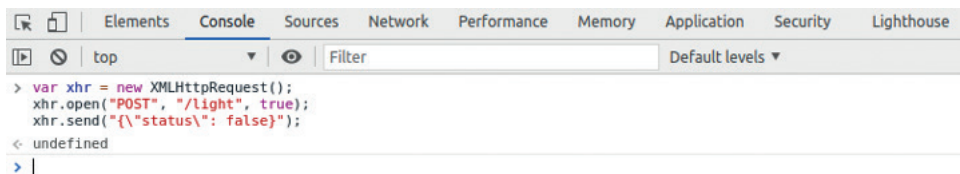
Нажмите **F12** на текущей странице, чтобы войти в консоль. Введите следующую команду и нажмите **Enter**, чтобы отправить POST-запрос:

```

$ var xhr = new XMLHttpRequest();
$ xhr.open("POST", "192.168.3.80/light", true);
$ xhr.send("{\"status\": false}");

```

На рис. 8.5 показано, как использовать HTTP для установки состояния умного светильника.



```
> var xhr = new XMLHttpRequest();
  xhr.open("POST", "/light", true);
  xhr.send({"status": false});
< undefined
> |
```

Рисунок 8.5. Использование HTTP для установки статуса умного светильника

В этот момент сервер получит запрос HTTP POST {"status": false}. Лог использования HTTP для установки состояния умного светильника выглядит следующим образом:

```
I (773) wifi:mode:sta (30:ae:a4:80:48:98)
I (773) wifi:enable tsf
I (773) wifi station: wifi init sta finished.
I (793) wifi:new: <6,0>, old: <1,0>, ap: <255,255>, sta: <6,0>, prof:1
I (793) wifi:state: init -> auth (be)
I (813) wifi:state: auth -> assoc (0)
I (823) wifi:state: assoc -> run (10)
I (873) wifi:connected with myssid, aid = 1, channel 6, Bw20, bssid =
34:29:12:43:c5:40
I (873) wifi:security: WPA2-PSK, phy: bgn, rssi: -21
I (883) wifi:pm start, type: 1
I (943) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (1543) esp netif handlers: sta ip: 192.168.3.80, mask: 255.255.255.0, gw:
192.168.3.1
I (1543) wifi station: got ip:192.168.3.80
I (1543) wifi station: connected to ap SSID: myssid password: 12345678
I (1553) wifi station: Starting server on port: '80'
I (1563) wifi station: Registering URI handlers
W (11393) wifi:<ba-add>idx:0 (ifx:0,34:29:12:43:c5:40), tid:7, ssn:4, winSize:64
I (11413) wifi station: {"status": false}
```

Обновление текущей страницы в это время может продолжать запрашивать состояние умного светильника, и тогда отобразится установленный командой статус, как показано на рис. 8.6.



```
← → ↻ 🏠 🔒 Not secure | 192.168.3.80/light
{"status": false}
```

Рисунок 8.6. Отображение измененного состояния умного светильника

8.3.3. Протокол пользовательских датаграмм (User Datagram Protocol, UDP)

Подразделы 8.3.1 и 8.3.2 вводят соответственно TCP и HTTP, оба из которых характеризуются надежной передачей. В этом подразделе будет представлен еще один протокол транспортного уровня UDP. В отличие от TCP, UDP является

ненадежным протоколом передачи. Общие прикладные протоколы, основанные на UDP, включают DNS, TFTP и SNMP.

1. Введение в UDP

UDP – это простой протокол связи, ориентированный на датаграммы³⁰, который расположен на транспортном уровне подобно TCP. Согласно Википедии, протокол UDP был разработан Дэвидом П. Ридом в 1980 году и определен в RFC 768. UDP – ненадежный протокол передачи данных. После отправки данных по протоколу UDP нижележащий уровень не сохраняет данные, чтобы предотвратить потерю во время передачи. Сам UDP не поддерживает исправление ошибок, управление очередями или контроль перегрузки каналов, но поддерживает контрольные суммы.

UDP – это протокол без установления соединения. Ему не нужно устанавливать соединение перед отправкой данных, в отличие от TCP. Данные могут быть отправлены непосредственно одноуровневому узлу без установления соединения. Поскольку во время передачи данных не требуется устанавливать соединение, нет необходимости поддерживать статус соединения, включая статус отправки и получения.

UDP отвечает только за передачу, поэтому приложениям, использующим этот протокол, необходимо самим контролировать то, как отправляются и обрабатываются данные: например, гарантировать, что одноранговые приложения получают данные правильно и по порядку.

По сравнению с TCP, UDP не может гарантировать безопасную и надежную передачу данных. Вы можете поинтересоваться, почему до сих пор используется протокол UDP. Отсутствие необходимости в установке соединения UDP приводит к меньшим сетевым и временным затратам, чем у TCP. Ненадежная передача UDP (главным образом невозможность гарантировать повторную передачу при потере пакета) больше подходит для таких приложений, как потоковое мультимедиа, многопользовательские игры в реальном времени и IP-телефония, где потеря нескольких пакетов не повлияет на работу приложения. С другой стороны, если для повторной передачи используется протокол TCP, это значительно увеличит задержку в сети.

2. Создание UDP-сервера с использованием сокета

Исходный код

Для получения исходного кода функции `esp_create_udp_server()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/udp_socket.

Создание UDP-сервера с использованием сокета аналогично созданию группового приемника групповой рассылки, как было показано в подразделе 8.2.2. Оба включают создание сокета UDP, настройку привязанного порта, прием и отправку данных. Функция `esp_create_udp_server()` устанавливает опцию `SO_REUSEADDR`, позволяющую серверу привязать адрес уже установленной связи. Код выглядит следующим образом:

³⁰ Датаграммы (Datagram) – так в терминологии UDP называют пользовательские сообщения.


```

1. esp_err_t esp_create_udp_server(void)
2. {
3.     char rx_buffer[128];
4.     char addr_str[32];
5.     esp_err_t err = ESP_FAIL;
6.     struct sockaddr_in server_addr;
7.     //Create a UDP socket
8.     int sock = socket(AF_INET, SOCK_DGRAM, 0);
9.     if (sock < 0) {
10.         ESP_LOGE(TAG, "create socket error");
11.         return err;
12.     }
13.     ESP_LOGI(TAG, "create socket success, sock : %d", sock);
14.     //Enable SO_REUSEADDR, allowing the server to bind connected address
15.     int opt = 1;
16.     int ret = setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
17.     if (ret < 0) {
18.         ESP_LOGE(TAG, "Failed to set SO_REUSEADDR. Error %d", errno);
19.         goto exit;
20.     }
21.     //Bind the server to an interface with all-zero IP address and port number 3333
22.     server_addr.sin_family = AF_INET;
23.     server_addr.sin_addr.s_addr = INADDR_ANY;
24.     server_addr.sin_port = htons(PORT);
25.     ret = bind(sock, (struct sockaddr *) &server_addr, sizeof(server_addr));
26.     if (ret < 0) {
27.         ESP_LOGE(TAG, "bind socket failed, sockfd: %d, errno : %d", sock, errno);
28.         goto exit;
29.     }
30.     ESP_LOGI(TAG, "bind socket success");
31.     while (1) {
32.         struct sockaddr_in source_addr;
33.         socklen_t addr_len = sizeof(source_addr);
34.         memset(rx_buffer, 0, sizeof(rx_buffer));
35.         int len = recvfrom(sock, rx_buffer, sizeof(rx_buffer) - 1, 0,
36.             (struct sockaddr *)&source_addr, &addr_len);
37.         // Reception error
38.         if (len < 0) {
39.             ESP_LOGE(TAG, "recvfrom failed: errno %d", errno);
40.             break;
41.         } else { //Data is received
42.             if (source_addr.sin_family == PF_INET) {
43.                 inet_ntoa_r(((struct sockaddr_in *)&source_addr)->sin_addr,
44.                     addr_str, sizeof(addr_str) - 1);
45.             }
46.             //String ends with NULL
47.             rx_buffer[len] = 0;
48.             ESP_LOGI(TAG, "Received %d bytes from %s: ", len, addr_str);
49.             ESP_LOGI(TAG, "%s", rx_buffer);
50.         }
51.     }
52. exit:
53.     close(sock);
54.     return err;
55. }

```

3. Создание UDP-клиента с использованием сокета

Исходный код

Для получения исходного кода функции `esp_create_udp_client()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/udp_socket.

С помощью функции `esp_create_udp_client()` клиент UDP может отправлять данные, включая создание сокетов UDP, настройку адресов и портов назначения, вызов интерфейса сокетов `sendto()` для отправки данных. Код выглядит следующим образом:

```
1. esp_err_t esp_create_udp_client(void)
2. {
3.     esp_err_t err = ESP_FAIL;
4.     char *payload = "Open the light";
5.     struct sockaddr_in dest_addr;
6.     dest_addr.sin_addr.s_addr = inet_addr(HOST_IP);
7.     dest_addr.sin_family = AF_INET;
8.     dest_addr.sin_port = htons(PORT);
9.
10.    //Create a UDP socket
11.    int sock = socket(AF_INET, SOCK_DGRAM, 0);
12.    if (sock < 0) {
13.        ESP_LOGE(TAG, "Unable to create socket: errno %d", errno);
14.        return err;
15.    }
16.
17.    //Send data
18.    int ret = sendto(sock, payload, strlen(payload), 0,
19.                    (struct sockaddr *)&dest_addr, sizeof(dest_addr));
20.    if (ret < 0) {
21.        ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
22.        goto exit;
23.    }
24.    ESP_LOGI(TAG, "Message send successfully");
25.    err = ESP_OK;
26. exit:
27.    close(sock);
28.    return err;
29. }
```

Клиентам UDP не нужно устанавливать соединение с сервером, они могут напрямую отправлять данные на сервер. Поскольку UDP создает ненадежные соединения, отправляемые данные, такие как «Open the light», могут быть потеряны, в результате чего одноранговый узел их не получит. Поэтому при написании кода для обмена между клиентом и сервером в код прикладного уровня следует добавить некоторую логику, чтобы гарантировать, что данные не потеряются. Например, когда клиент отправляет серверу «Open the light», сервер возвращает «Open the light OK» после его успешного получения. Если клиент получает ответ в течение 1 секунды, это означает, что данные были отправлены на сервер корректно. Если клиент не получает его в течение 1 секунды, ему нужно повторно отправить данные «Open the light».

8.3.4. Протокол ограниченных приложений (Constrained Application Protocol, CoAP)

С быстрым развитием технологий интернета вещей была создана серия протоколов для IoT-устройств. Большинство таких устройств имеют ограниченные ресурсы: ОЗУ, флеш-память, ЦП, пропускную способность сети и т. д. Если использовать протоколы TCP и HTTP для передачи данных, часто требуется увеличение памяти и повышение пропускной способности сети. Если UDP можно использовать для передачи данных в условиях дефицита ресурсов, существует ли протокол приложения, аналогичный HTTP? Да, CoAP был разработан в соответствии с архитектурой REST HTTP.

1. Введение в CoAP

CoAP – это протокол, аналогичный веб-приложениям в устройствах IoT. Он определен в RFC 7252 и может использоваться для устройств IoT с ограниченными ресурсами, что позволяет этим устройствам, называемым узлами (nodes), связываться с более широким диапазоном интернет-ресурсов с использованием аналогичных протоколов. CoAP предназначен для устройств в одной и той же ограниченной сети (например, маломощных или в сетях с потерями), для связи между устройствами и общими узлами в интернете, а также между устройствами в различных ограниченных сетях, соединенных через интернет.

CoAP основан на модели запрос–ответ, аналогичной HTTP, которая может компенсировать недостатки ненадежной передачи UDP и гарантировать, что данные не будут потеряны или перемешаны. Ресурсы сервера идентифицируются URL-адресами (например, *coap://[IP]/id/light_status* для доступа к статусу умного светильника). Клиент обращается к ресурсам сервера через URL-адрес ресурса и управляет ресурсами сервера с помощью четырех методов запроса (GET, PUT, POST и DELETE).

CoAP также имеет следующие функции:

- и клиент, и сервер могут независимо отправлять друг другу запросы;
- поддерживает надежную передачу данных;
- поддерживает групповую и широковещательную рассылку, обеспечивая передачу данных «один ко многим»;
- поддерживает связь с низким энергопотреблением и непостоянными соединениями;
- по сравнению с HTTP его заголовок меньше по объему.

2. Создание сервера CoAP с использованием компонента ESP-IDF

В следующем коде показано, как создать сервер CoAP с помощью компонента ESP-IDF, предоставляющего операции GET и PUT для извлечения и модификации ресурсов через CoAP. Операции протокола CoAP, как правило, фиксированы, и вам нужно сосредоточиться только на URI вашего собственного ресурса, путях и операциях, которые необходимо предоставить. Функция *coap_resource_init()* может использоваться для установки URI доступа к ресурсам, а функция *coap_register_handler()* может использоваться для регистрации функций обратного вызова GET и PUT, соответствующих установленному URI ресурса.

Исходный код

Для получения исходного кода функций `coap_resource_init()` и `coap_register_handler()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/coap.

```
1. static char buf[100] = "{\"status\": true}";
2.
3. //Callback function of GET method in CoAP
4. static void esp_coap_get(coap_context_t *ctx, coap_resource_t *resource,
5.     coap_session_t *session, coap_pdu_t *request,
6.     coap_binary_t *token, coap_string_t *query,
7.     coap_pdu_t *response)
8. {
9.     coap_add_data_blocked_response(resource, session, request, response,
10.        token, COAP_MEDIATYPE_TEXT_PLAIN, 0,
11.        strlen(buf), (const u_char *)buf);
12. }
13.
14. //Callback function of PUT method in CoAP
15. static void esp_coap_put(coap_context_t *ctx,
16.     coap_resource_t *resource,
17.     coap_session_t *session,
18.     coap_pdu_t *request,
19.     coap_binary_t *token,
20.     coap_string_t *query,
21.     coap_pdu_t *response)
22. {
23.     size_t size;
24.     const unsigned char *data;
25.     coap_resource_notify_observers(resource, NULL);
26.
27.     //Read the received CoAP protocol data
28.     (void)coap_get_data(request, &size, &data);
29.     if (size) {
30.         if (strcmp((char *)data, buf, size)) {
31.             memcpy(buf, data, size);
32.             buf[size] = 0;
33.             response->code = COAP_RESPONSE_CODE(204);
34.         } else {
35.             response->code = COAP_RESPONSE_CODE(500);
36.         }
37.     } else { //A size of 0 indicates a receiving error
38.         response->code = COAP_RESPONSE_CODE(500);
39.     }
40. }
41.
42. static void esp_create_coap_server(void)
43. {
44.     coap_context_t *ctx = NULL;
45.     coap_address_t serv_addr;
46.     coap_resource_t *resource = NULL;
47.     while (1) {
48.         coap_endpoint_t *ep = NULL;
49.         unsigned wait_ms;
50.
```

```

51.     //Create a CoAP server socket
52.     coap_address_init(&serv_addr);
53.     serv_addr.addr.sin6.sin6_family = AF_INET6;
54.     serv_addr.addr.sin6.sin6_port = htons(COAP_DEFAULT_PORT);
55.
56.     //Create CoAP ctx
57.     ctx = coap_new_context(NULL);
58.     if (!ctx) {
59.         ESP_LOGE(TAG, "coap_new_context() failed");
60.         continue;
61.     }
62.
63.     //Set the CoAP protocol node
64.     ep = coap_new_endpoint(ctx, &serv_addr, COAP_PROTO_UDP);
65.     if (!ep) {
66.         ESP_LOGE(TAG, "udp: coap_new_endpoint() failed");
67.         goto clean_up;
68.     }
69.
70.     //Set CoAP protocol resource URI
71.     resource = coap_resource_init(coap_make_str_const("light"), 0);
72.     if (!resource) {
73.         ESP_LOGE(TAG, "coap_resource_init() failed");
74.         goto clean_up;
75.     }
76.
77. //Register callback functions of GET and PUT methods corresponding to CoAP
resource URI
78.     coap_register_handler(resource, COAP_REQUEST_GET, esp_coap_get);
79.     coap_register_handler(resource, COAP_REQUEST_PUT, esp_coap_put);
80.
81.     //Set CoAP GET resource observable
82.     coap_resource_set_get_observable(resource, 1);
83.
84.     //Add resource to CoAP ctx
85.     coap_add_resource(ctx, resource);
86.     wait_ms = COAP_RESOURCE_CHECK_TIME * 1000;
87.     while (1) {
88.         //Wait to receive CoAP data
89.         int result = coap_run_once(ctx, wait_ms);
90.         if (result < 0) {
91.             break;
92.         } else if (result && (unsigned)result < wait_ms) {
93.             //Decrease waiting time
94.             wait_ms -= result;
95.         } else {
96.             //Reset waiting time
97.             wait_ms = COAP_RESOURCE_CHECK_TIME * 1000;
98.         }
99.     }
100. }
101.clean_up:
102.     coap_free_context(ctx);
103.     coap_cleanup();
104.}

```

Приведенный код создает сервер CoAP и предоставляет метод GET для запроса состояния умного светильника и метод PUT для его установки. Вы можете использовать браузер Chrome для установки CoAP для отладки подключаемого модуля клиента Sorreg и имитации CoAP-клиента.

Откройте плагин Chrome Sorreg, введите URL-адрес `coap://[ip]/light` и нажмите **Enter** для подключения к серверу. На рис. 8.7 показано подключение модуля CoAP.

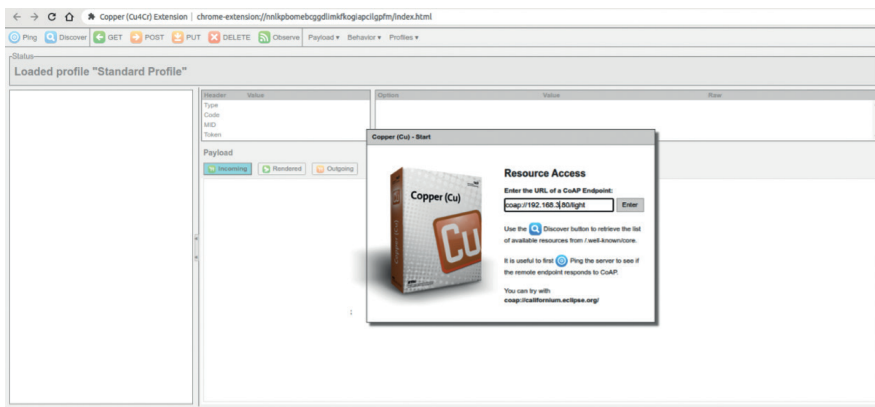


Рисунок 8.7. Подключение плагина CoAP

После успешного подключения нажмите кнопку **GET** в верхнем левом углу, чтобы получить статус и отобразить `{"status": true}`, запрос статуса подключаемого модуля CoAP показан на рис. 8.8.

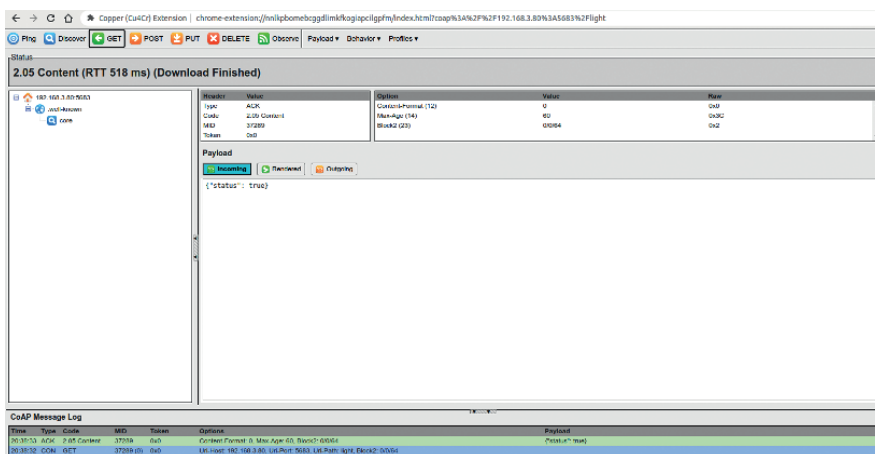


Рисунок 8.8. Запрос статуса плагина CoAP

Нажмите кнопку **PUT** в левом верхнем углу и измените данные в разделе **Payload** (Полезная нагрузка) → **Outgoing** (Исходящая), чтобы установить статус умного светильника на `false`. Рисунок 8.9 показывает состояние конфигурации подключаемого модуля CoAP.

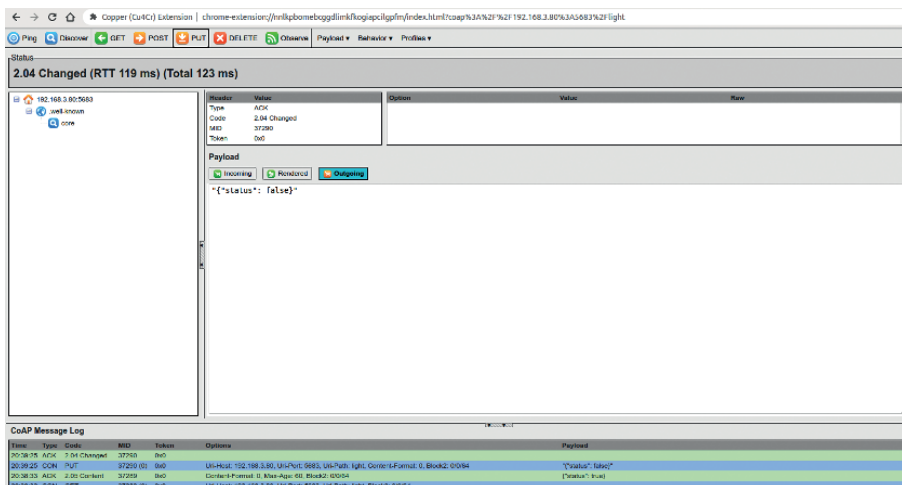


Рисунок 8.9. Настройка статуса плагина CoAP

В это время снова нажмите кнопку **GET** в левом верхнем углу, чтобы получить статус, который отобразится как `{"status": false}`. На рис. 8.10 показано состояние настройки запроса CoAP-плагина.

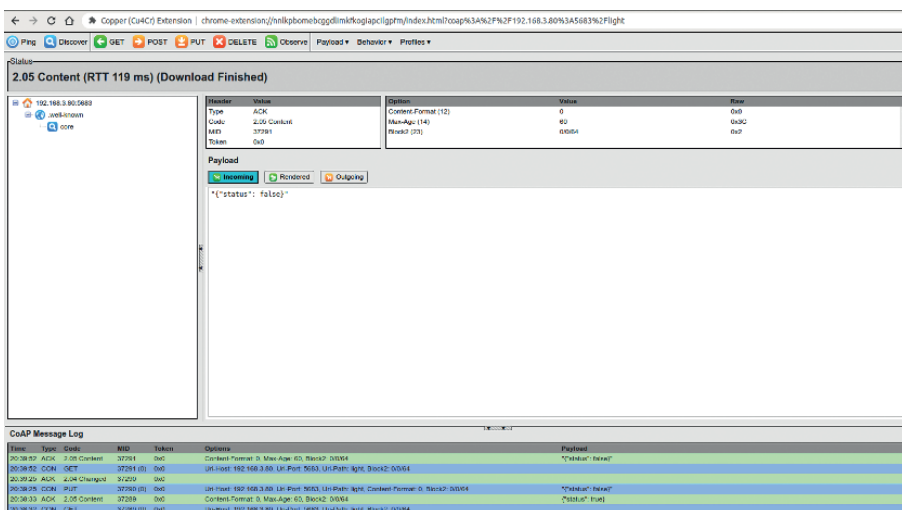


Рисунок 8.10. Статус настройки запроса плагина CoAP

8.3.5. Протокол Bluetooth

1. Введение в протокол Bluetooth

В главе 7 вы познакомились с протоколом и архитектурой Bluetooth. Протокол Bluetooth определяет форматы сообщений и процесс выполнения определенных функций, таких как управление связью, службы безопасности, службы обмена информацией и передачи данных. Этот раздел знакомит с основами

протокола атрибутов (ATT) спецификации Bluetooth. Bluetooth-данные существуют в виде атрибутов, и каждый атрибут состоит из четырех элементов.

Дескриптор атрибута

Точно так же, как адреса памяти используются для поиска содержимого в памяти, дескрипторы атрибутов могут помочь найти соответствующий атрибут. Например, первый дескриптор атрибута – 0x0001, дескриптор второго атрибута – 0x0002 и т. д., максимум до 0xFFFF.

UUID атрибута

Каждые данные имеют определенные свойства. Например, умный светильник имеет два основных атрибута: один для установки состояния включения/выключения, а другой для чтения этого состояния.

Значение атрибута

Значение атрибута – это информация, которую несет атрибут; остальные три элемента предназначены лишь для того, чтобы одноранговый узел мог получить значение атрибута как можно проще. Например, для умного светильника значение атрибута настройки состояния включения/выключения может быть установлено на «1», чтобы включить свет, или на «0», чтобы выключить свет; значение атрибута для чтения состояния включения/выключения может быть «1» для состояния «включено» или «0» для состояния «выключено».

Разрешения атрибутов

Каждый атрибут имеет соответствующие ограничения доступа для собственных значений атрибута, например некоторые атрибуты доступны для чтения, некоторые доступны для записи, а некоторые доступны для чтения и записи.

Сторона, которой принадлежат данные, может управлять разрешениями атрибутов локальных данных. Например, разрешение атрибута установки выключателя умного светильника может быть установлено как доступное для записи, но не для чтения, а разрешение атрибута статуса выключателя умного светильника может быть установлено только для чтения и недоступно для записи.

В табл. 8.2 перечислены атрибуты Bluetooth для основных функций умного светильника.

Таблица 8.2. Атрибуты Bluetooth для основных функций умного светильника

Дескриптор атрибута	UUID атрибута	Значение атрибута	Разрешения атрибута
0x0001	Установить статус вкл/выкл	1/0	Доступно для записи, но не для чтения
0x0002	Прочсть статус вкл/выкл	1/0	Доступно для чтения, но не для записи

Устройство, на котором хранятся данные (т. е. атрибуты), обычно называется **сервером**, а устройство, которое получает данные от других устройств, называется **клиентом**. В случае умного светильника и смартфона светильник –

сервер, а смартфон – клиент. Следующие операции являются общими между сервером и клиентом:

(1) Клиент отправляет данные на сервер.

Данные передаются путем записи данных на сервер. Существует два типа операций записи: **запрос на запись** и **команда записи**. Основное отличие между ними заключается в том, что первый требует ответа (**write response**) от сервера, а второй не требует. Для умного светильника команда включения/выключения света, отправленная смартфоном, является операцией записи, то есть запросом на запись, требующим ответа от умного светильника. Такой ответ не является простым ответом АСК. На смартфон нужно вернуть результат действия включения/выключения света, т. е. сообщить текущее состояние.

(2) Сервер отправляет данные клиенту.

Обновленные данные отправляются с сервера клиенту в основном в виде **указания** или **уведомления** сервера. Подобно операциям записи, основное различие между указанием и уведомлением заключается в том, что первое требует, чтобы другое устройство ответило (подтвердило) получение данных указанием. Для умного светильника, если он включен/выключен через физическую кнопку-переключатель, ее статус нужно сообщить на смартфон через указание или уведомление, и смартфон тогда будет отображать последний статус.

(3) Клиент активно считывает данные с сервера.

Как правило, клиент получает значения соответствующих атрибутов от сервера через операции чтения. В случае, упомянутом выше, когда умный светильник включается/выключается с помощью физической кнопки, кроме ожидания уведомления от сервера, смартфон также может получать статус в реальном времени через операции чтения.

2. Создание сервера Bluetooth с помощью компонента ESP-IDF

Исходный код

Исходный код Bluetooth см. в <https://github.com/espressif/esp-idf/tree/master/examples/provisioning>. Пример кода конкретной конфигурации см.: <https://github.com/espressif/esp-idf/tree/master/examples/provisioning>.

В следующем примере компонент `protocomm` используется для реализации сервера умного светильника, а настроенная конфигурация применяет протокол `custom-proto`. Как упоминалось ранее, для реализации управления включением/выключением и запроса состояния светильника необходимо указать два определенных атрибута. Код выглядит следующим образом:

```
1. static esp_err_t wifi_prov_config_set_light_handler(uint32_t session_id,
2.                                                     const uint8_t *inbuf,
3.                                                     ssize_t inlen,
4.                                                     uint8_t **outbuf,
5.                                                     ssize_t *outlen,
6.                                                     void *priv_data)
7. {
```

```

8.     CustomConfigRequest *req;
9.     CustomConfigResponse resp;
10.    req = custom_config_request_unpack(NULL, inlen, inbuf);
11.    if (!req) {
12.        ESP_LOGE(TAG, "Unable to unpack config data");
13.        return ESP_ERR_INVALID_ARG;
14.    }
15.    custom_config_response_init(&resp);
16.    resp.status = CUSTOM_CONFIG_STATUS_ConfigFail;
17.    if (req->open_light) { //Turn on the smart light
18.        //Pull up GPIO level according to the status
19.        ESP_LOGI(TAG, "Open the light");
20.    } else {
21.        //Pull down GPIO level according to the status
22.        ESP_LOGI(TAG, "Close the light");
23.    }
24.
25.    //Set response status and smart light status according to the light's execution result
26.    resp.status = CUSTOM_CONFIG_STATUS_ConfigSuccess;
27.    custom_config_request_free_unpacked(req, NULL);
28.    resp.light_status = 1; //Respond according to the light status
29.    *outlen = custom_config_response_get_packed_size(&resp);
30.    if (*outlen <= 0) {
31.        ESP_LOGE(TAG, "Invalid encoding for response");
32.        return ESP_FAIL;
33.    }
34.    *outbuf = (uint8_t *) malloc(*outlen);
35.    if (*outbuf == NULL) {
36.        ESP_LOGE(TAG, "System out of memory");
37.        return ESP_ERR_NO_MEM;
38.    }
39.
40.    custom_config_response_pack(&resp, *outbuf);
41.    return ESP_OK;
42. }
43.
44. static int wifi_prov_config_get_light_handler(uint32_t session_id,
45.                                                const uint8_t *inbuf,
46.                                                ssize_t inlen,
47.                                                uint8_t **outbuf,
48.                                                ssize_t *outlen,
49.                                                void *priv_data)
50. {
51.     CustomConfigResponse resp;
52.     custom_config_response_init(&resp);
53.     resp.status = CUSTOM_CONFIG_STATUS_ConfigSuccess;
54.     resp.light_status = 1; //Respond according to the light status
55.     *outlen = custom_config_response_get_packed_size(&resp);
56.     if (*outlen <= 0) {
57.         ESP_LOGE(TAG, "Invalid encoding for response");
58.         return ESP_FAIL;
59.     }
60.     *outbuf = (uint8_t *) malloc(*outlen);
61.     if (*outbuf == NULL) {
62.         ESP_LOGE(TAG, "System out of memory");
63.         return ESP_ERR_NO_MEM;
64.     }

```



```

122. //Set protocomm version verification endpoint for the protocol
123. protocomm_set_version(g_prov->pc, "proto-ver", "V0.1");
124. //Set protocomm security type for the endpoint
125. if (g_prov->security == 0) {
126.     protocomm_set_security(g_prov->pc, "prov-session",
127.         &protocomm_security0, NULL);
128. } else if (g_prov->security == 1) {
129.     protocomm_set_security(g_prov->pc, "prov-session",
130.         &protocomm_security1, g_prov->pop);
131. }
132. //Add an endpoint for Wi-Fi configuration
133. if(protocomm_add_endpoint(g_prov->pc, "prov-config",
134.     wifi_prov_config_data_handler,
135.     (void *) &wifi_prov_handlers) !=ESP_OK){
136.     ESP_LOGE(TAG, "Failed to set provisioning endpoint");
137.     protocomm_ble_stop(g_prov->pc);
138.     return ESP_FAIL;
139. }
140. //Add an endpoint for setting smart light status
141. if (protocomm_add_endpoint(g_prov->pc, "set-light",
142.     wifi_prov_config_set_light_handler,
143.     NULL) != ESP_OK) {
144.     ESP_LOGE(TAG, "Failed to set set-light endpoint");
145.     protocomm_ble_stop(g_prov->pc);
146.     return ESP_FAIL;
147. }
148. //Add an endpoint for getting smart light status
149. if (protocomm_add_endpoint(g_prov->pc, "get-light",
150.     wifi_prov_config_get_light_handler,
151.     NULL) != ESP_OK) {
152.     ESP_LOGE(TAG, "Failed to set get-light endpoint");
153.     protocomm_ble_stop(g_prov->pc);
154.     return ESP_FAIL;
155. }
156. ESP_LOGI(TAG, "Provisioning started with BLE devname : '%s'",
157.     config.device_name);
158. return ESP_OK;
159.}

```

В приведенном примере представлены два атрибута: set-light и get-light, дескрипторы этих атрибутов имеют значения 0x0004 и 0x0005 соответственно. Когда смартфон отправляет команду на установку света, функция обратного вызова wifi_prov_config_set_light_handler() будет выполняться для обработки действия включения/выключения и информирования смартфона о текущем состоянии умного светильника. Когда смартфон отправляет команду чтения, функция обратного вызова wifi_prov_config_get_light_handler() будет выполнена для информирования смартфона о текущем состоянии светильника. Вы можете использовать на смартфоне помощник Bluetooth по отладке для сканирования устройств, подключенных к Bluetooth, и функция каждой службы интуитивно будет более понятна с помощью услуг, предоставляемых Bluetooth-устройством.

В приведенном выше примере реализовано локальное управление через Bluetooth на основе компонента protocomm, так что структура данных отно-

сительно сложна. Если вы опытный разработчик, то можете попробовать использовать идеи приведенного примера для реализации локального управления. Кроме того, в этой книге представлен самый простой пример сервера на основе Bluetooth для начинающих. Чтобы понять процесс локального управления с помощью Bluetooth, можно обратиться к примеру кода в подразделе 8.5.3.

8.3.6. Обзор протоколов передачи данных

Протоколы UDP и TCP на транспортном уровне могут напрямую использоваться как коммуникационные протоколы для пересылки прикладных данных. В табл. 8.3 перечислены различия между UDP и TCP.

Таблица 8.3. Различия между TCP и UDP

Сравнение	TCP	UDP
Надежность	Надежная передача; поддерживает повтор пакетов, управление потоком и контроль перегрузки	Ненадежная передача; не поддерживает повтор пакетов, управление потоком или контроль перегрузки
Соединение	Ориентированный на процедуру подключения, с тремя подтверждениями для установления соединения и четырьмя для отключения; длительное соединение	Нет процедуры подключения; прямая передача данных; короткое соединение
Объект соединения	Соединение «один к одному»	Индивидуальная одноадресная передача «один к одному», прямая трансляция «один ко всем» и групповая рассылка «один ко многим»
Накладные расходы на заголовок	≥ 20 байт	8 байт
Скорость передачи	Зависит от сетевой среды; в случае потери пакета происходит повторная передача, что снижает скорость	Быстро, независимо от сетевой среды, и отвечает только за передачу данных в сеть
Типовое применение	Подходит для надежной передачи, например передачи файлов	Подходит для передачи в реальном времени, например IP-телефония, видеотелефония, потоковое мультимедиа и т. д.

Для передачи данных локального управления может быть выбран протокол TCP на транспортном уровне, поскольку он может обеспечить точность передачи данных. При использовании UDP приложение для смартфона отправит команду на включение освещения. Команда может пропасть из-за проблем с сетевым окружением, и ESP32-C3 может не получить команду. В то время как для TCP, даже если пакет данных будет испорчен, базовый уровень приложения для смартфона автоматически отправит команду повторно.

Однако недостатком отправки данных с использованием протокола чисто транспортного уровня является то, что вам необходимо разработать бизнес-логику приложений верхнего уровня. Таким образом, в этом разделе также представлены прикладные протоколы HTTP и CoAP, основанные на TCP и UDP.

И HTTP, и CoAP являются сетевыми протоколами передачи данных, основанными на модели REST, которые используются для отправки запросов и ответа на запросы. Единственное различие заключается в том, что первый основан на TCP, а второй на UDP, и каждый наследует соответствующие характеристики протокола транспортного уровня. В табл. 8.4 перечислены различия между HTTP и CoAP.

Таблица 8.4. Различия между HTTP и CoAP

Сравнение	HTTP	CoAP
Транспортный уровень	TCP	UDP
Накладные расходы на заголовки	Может содержать большой объем данных заголовка сообщения с большими накладными расходами	Заголовки пакетов сжимаются в двоичном формате для снижения накладных расходов
Потребляемая мощность	Долгая связь, высокое энергопотребление	Короткое соединение, низкое энергопотребление
Обнаружение ресурса	Не поддерживается	Поддерживается
Способ запроса	Обычно запускается клиентом; нет активного запуска со стороны сервера	Как клиент, так и сервер могут быть инициаторами запросов
Типовое применение	Подходит для устройств с хорошей производительностью и большим объемом памяти	Подходит для устройств с низкой производительностью и маленькой памятью

По сравнению с HTTP, CoAP больше подходит для устройств интернета вещей с ограниченными ресурсами. Если устройство обладает большим количеством ресурсов и лучшей производительностью, то надо учитывать, что HTTP обладает большей функциональностью, чем CoAP.

После сравнения протоколов связи в семействе протоколов TCP/IP мы сравнили эти протоколы с протоколом Bluetooth. Наиболее очевидное различие между ними заключается в том, что Bluetooth – это протокол «точка–точка», в то время как TCP/IP – это сквозной протокол, который может передаваться через маршрутизаторы. Таким образом, с точки зрения скорости отклика, хотя Bluetooth и Wi-Fi оба являются технологиями беспроводной передачи данных на канале 2,4 ГГц, Bluetooth быстрее Wi-Fi при передаче данных между смартфонами и ESP32-C3. Размер пакета Bluetooth меньше, чем у данных приложения, использующего стек протоколов TCP/IP, а энергопотребление Bluetooth, естественно, ниже, чем у Wi-Fi. Протокол Bluetooth поддерживает обнаружение ресурсов и не требует локального обнаружения, поскольку Bluetooth – это соединение «точка–точка», которое очень подходит для локального управления. Однако поскольку большинство продуктов интернета

вещей в настоящее время нуждаются в подключении к облаку, функциональность Wi-Fi имеет важное значение. Многие IoT-продукты могут использовать только Wi-Fi или только Bluetooth для настройки сети. Если IoT-продукт не нуждается в подключении к облаку, Bluetooth можно использовать лишь для локального управления. Если такому продукту необходимо будет подключиться к облаку, ему необходимо использовать Wi-Fi и для подключения к облаку, и для локального управления.

8.4. Гарантии безопасности данных

Как мы все знаем, TCP и UDP, а также прикладные протоколы HTTP и CoAP, работающие поверх них, передают данные в виде открытого текста. Это может привести к перехвату или искажению данных во время передачи. Если в данные будет включена конфиденциальная информация, такая как пароли или номера учетных записей, могут возникнуть невосполнимые потери. Поэтому необходимо шифровать данные, передаваемые в виде открытого текста. Данные, передаваемые по Bluetooth, не будут попадать в сеть, поскольку Bluetooth является протоколом «точка–точка», и вероятность того, что они будут перехвачены, очень мала. Кроме того, сам протокол Bluetooth шифрует пользовательские данные. Поэтому в этом разделе в основном обсуждается шифрование данных по протоколу TCP/IP.

Шифрование используется для обеспечения конфиденциальности и целостности передаваемых данных. Общие системы шифрования обычно кодируют данные перед передачей. Например, в предыдущих войнах телеграммы были закодированы, и у отправителя и получателя была одна и та же кодовая книга. Получатель использовал цифры или буквы в кодовой книге, чтобы заменить слова или предложения в телеграмме. Даже если содержание телеграммы было перехвачено третьим лицом, третье лицо не могло в короткие сроки расшифровать истинное содержание телеграммы. Однако этот метод имеет недостаток, заключающийся в том, что содержимое телеграммы все же может быть расшифровано, что является лишь вопросом времени. Кроме того, для предотвращения расшифровки телеграммы получателю и отправителю необходимо периодически менять кодовую книгу. Это может привести к тому, что кодовая книга утечет и содержание телеграммы будет расшифровано.

Приведенный выше пример телеграммы представляет собой распространенный алгоритм шифрования – симметричное шифрование. В алгоритме симметричного шифрования один и тот же алгоритм используется для шифрования и дешифрования, и их ключи также одинаковы. Симметричное шифрование имеет преимущества открытого алгоритма, небольшой вычислительной сложности, высокой скорости и высокой эффективности шифрования. Однако перед передачей данных отправитель и получатель должны согласовать ключ, а для того чтобы данные не были расшифрованы, обе стороны должны также периодически обновлять ключ, что усложняет управление ключами для обеих сторон.

Общие алгоритмы симметричного шифрования включают AES, DES и RC4. На рис. 8.11 показан процесс симметричного шифрования.

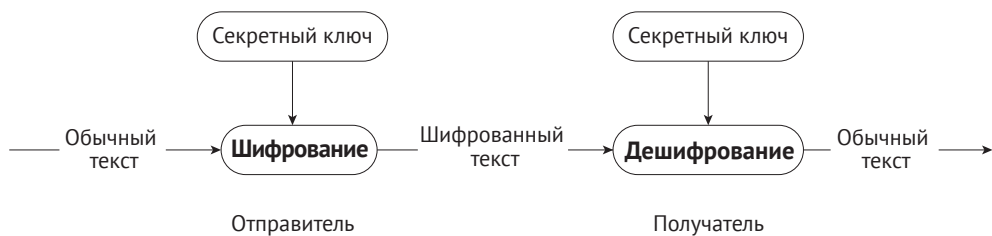


Рисунок 8.11. Процесс симметричного шифрования

В этом разделе мы представим альтернативный алгоритм – асимметричное шифрование. Обе стороны в асимметричном шифровании имеют пару ключей: открытый и закрытый. Данные шифруются с помощью открытого ключа, а расшифровываются с помощью закрытого ключа. Поскольку для шифрования и дешифрования используются разные ключи, этот алгоритм и называется асимметричным шифрованием. По сравнению с симметричным шифрованием асимметричное шифрование более безопасно. Поскольку асимметричное шифрование сложнее, чем симметричное, для расшифровки требуется больше времени, и третьим сторонам трудно расшифровать данные напрямую. Алгоритм асимметричного шифрования имеет высокую сложность и закрытый ключ, используемый для расшифровки, не передается по сети – его можно получить только у получателя, что значительно повышает безопасность данных. Обычные алгоритмы асимметричного шифрования включают RSA, Diffie-Hellman, DSA и т. д.

Преимуществом асимметричного шифрования является его безопасность. Пользователь А может сохранить закрытый ключ и передать открытый ключ пользователю В через сеть. Даже если пользователь С получит открытый ключ, он не может расшифровать данные, потому что у пользователя С нет закрытого ключа пользователя А. Таким образом, пользователь А и пользователь В могут уверенно передавать свои соответствующие открытые ключи через сеть. Помните, что открытый ключ используется для шифрования, а закрытый – для расшифровки. На рис. 8.12 показан процесс асимметричного шифрования.

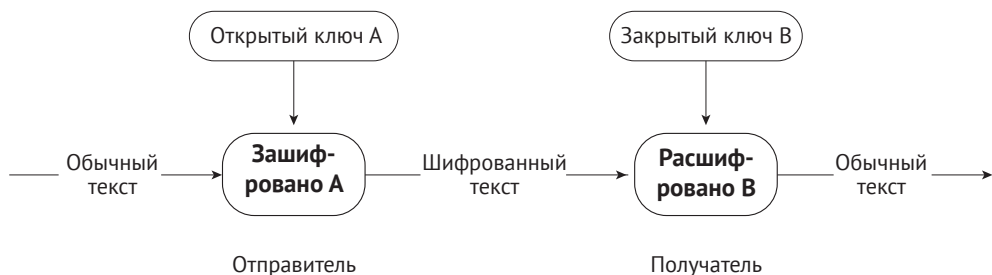


Рисунок 8.12. Процесс асимметричного шифрования

Асимметричное шифрование кажется очень безопасным, но задумывались ли вы когда-нибудь над этим вопросом: что, если пользователь С заменит все открытые ключи, отправленные пользователям А и В, своими соответствующими?

ющими ключами? Пользователь А не знает, принадлежит ли этот открытый ключ пользователю В, поэтому когда пользователь А отправляет данные, он будет использовать для шифрования открытый ключ пользователя С. В это время пользователь С может перехватить зашифрованные данные и расшифровать их с помощью своего закрытого ключа.

Поэтому крайне важно обеспечить легитимность открытого ключа. В действительности легитимность открытого ключа может быть гарантирована через центр сертификации (Certificate Authority, CA). CA также работает на основе алгоритмов асимметричного шифрования. При наличии CA пользователь В сначала предоставит свой открытый ключ и некоторую другую информацию для CA. CA шифрует эти данные, используя свой закрытый ключ; эти зашифрованные данные называются цифровым сертификатом пользователя В. Открытый ключ, переданный пользователем В пользователю А, является цифровым сертификатом, зашифрованным СА. После получения цифрового сертификата пользователь А будет применять цифровой сертификат, опубликованный СА (который содержит открытый ключ СА), для расшифровки данных цифрового сертификата пользователя В и, наконец, получит открытый ключ пользователя В.

8.4.1. Введение в безопасность транспортного уровня (TLS)

TLS – это протокол, основанный на TCP и обслуживающий прикладной уровень. Его предшественник – протокол уровня защищенных сокетов (Secure Sockets Layer, SSL). Через протокол TLS пакеты прикладного уровня могут быть зашифрованы и доставлены на уровень TCP для передачи.

1. Что делает TLS?

Протокол TLS в основном решает следующие три сетевые проблемы:

- гарантия конфиденциальности данных. Все данные передаются в зашифрованном виде для обеспечения защиты от несанкционированного доступа или кражи данных третьими лицами;
- гарантия целостности данных. Все данные защищены механизмом проверки, поэтому любое вмешательство будет немедленно обнаружено обеими сторонами, участвующими в общении;
- гарантия аутентификации и проверки личности обеих сторон, вовлеченных в данную коммуникацию. Аутентификация с помощью сертификатов может использоваться в общении обеими сторонами для обеспечения легитимности их личности.

2. Как работает TLS?

Протокол TLS можно разделить на две части. Уровень записи (**record layer**) использует ключ, согласованный клиентом и сервером для шифрования и передачи данных. Уровень подтверждения связи (**handshake layer**) выполняет согласование между клиентом и сервером набора ключевых строк для передачи зашифрованных данных. Модель протокола TLS показана на рис. 8.13, где уровень подтверждения включает четыре подпротокола: протокол подтверждения связи, протокол изменения спецификации шифра, протокол прикладных данных и протокол предупреждений.

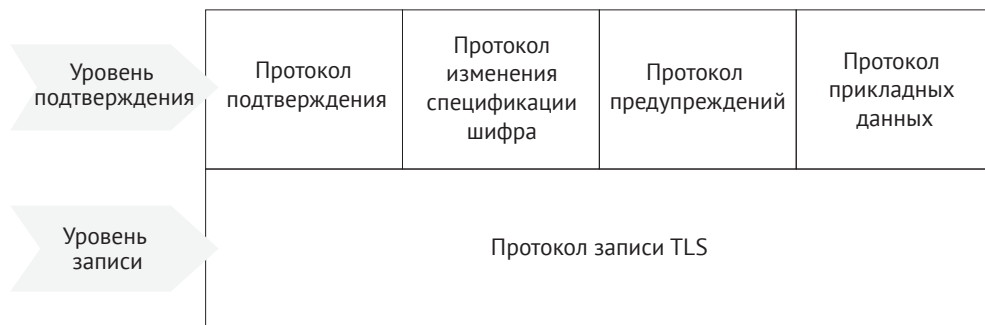


Рисунок 8.13. Модель протокола TLS

Уровень записи отвечает за все базовые данные, которыми обмениваются на транспортном уровне, и может шифровать данные. Каждая запись TLS начинается с короткого заголовка, который включает в себя поля типа содержимого (или название подпротокола), версии протокола и длину данных. Базовые данные сегментируются (или, наоборот, объединяются), сжимаются, дополняются кодом аутентификации сообщения, шифруются, а затем преобразуются в часть данных записи TLS. На рис. 8.14 показана структура пакета записей TLS.

Уровень подтверждения связи имеет четыре подпротокола, представленных ниже.

Протокол подтверждения связи

Отвечает за генерацию общего ключа, необходимого для процесса обмена данными и выполнения аутентификации личности. Обратите внимание, что протокол подтверждения не использует наборы шифров напрямую. Вместо этого он полагается на криптографию с открытым ключом или обмен ключами Диффи–Хеллмана, чтобы установить безопасную связь и предотвратить прослушивание или перехват данных.

Протокол изменения спецификации шифра

Отвечает за синхронизацию переключения паролей и используется после протокола подтверждения связи. Во время процесса подтверждения используется набор шифров «null», что означает отсутствие шифрования. После завершения подтверждения согласованный набор шифров используется для обеспечения безопасности последующей передачи данных.

Протокол передачи прикладных данных

Используется сторонами, обменивающимися данными, для передачи данных. Процесс передачи осуществляется через протокол прикладных данных и протокол записи TLS уровня подтверждения связи.

Протокол предупреждений

Используется для уведомления другой стороны при возникновении ошибки, такой как исключение во время процесса подтверждения связи, ошибка кода аутентификации сообщения или наличие данных, которые не могут быть распакованы.

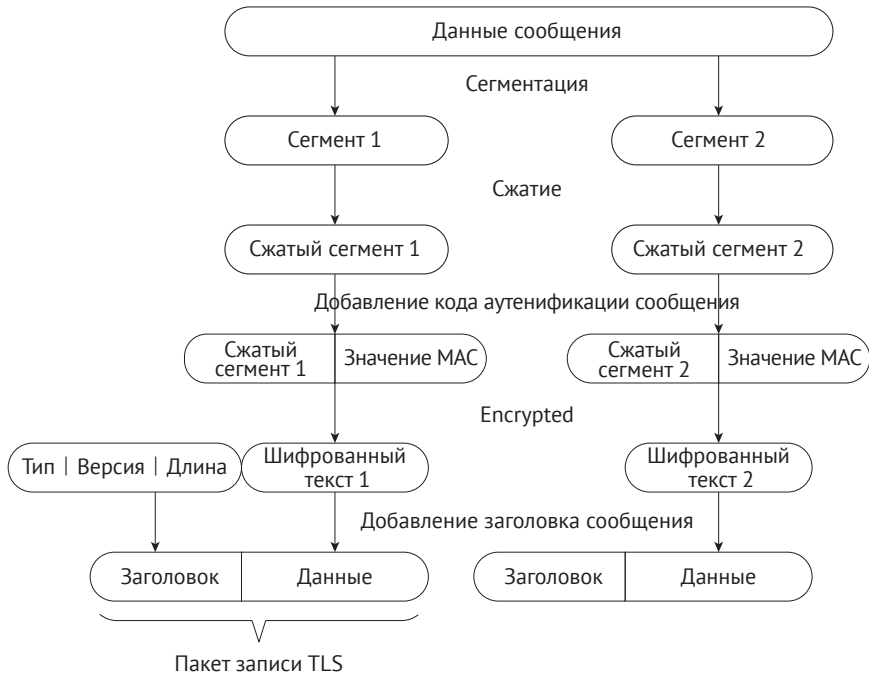


Рисунок 8.14. Пакет записи TLS

Алгоритм, используемый при шифровании TLS, следующий:

- **хеш-функция** проверяет целостность данных. Распространенные алгоритмы шифрования включают MD5, SHA и т. д.;
- **алгоритм симметричного шифрования** шифрует прикладные данные. Распространенные алгоритмы шифрования включают AES, RC4, DES и т. д.;
- **алгоритм асимметричного шифрования** для аутентификации личности и согласования ключей. Распространенные алгоритмы шифрования включают RSA, DH и т. д.

При использовании TLS клиент и сервер применяют алгоритм асимметричного шифрования для аутентификации личности и согласования ключа алгоритма симметричного шифрования, а затем симметричное шифрование и упаковку для передачи данных. На рис. 8.15 показан процесс подтверждения связи по протоколу TLS.

- (1) Приветствие клиента. Клиент отправляет на сервер самую высокую поддерживаемую версию протокола TLS и все поддерживаемые им наборы шифров, которые используются для отправки такой информации, как случайное число для генерации сеансового ключа.
- (2) Приветствие сервера. После получения приветственного сообщения клиента сервер выбирает версию протокола TLS и набор шифров в соответствии с информацией, отправленной клиентом, и возвращает их клиенту.
- (3) (Необязательно) Отправка сертификата. Сервер отправляет свой собственный сертификат на стороне сервера, используемый клиентом для проверки легитимности сервера.

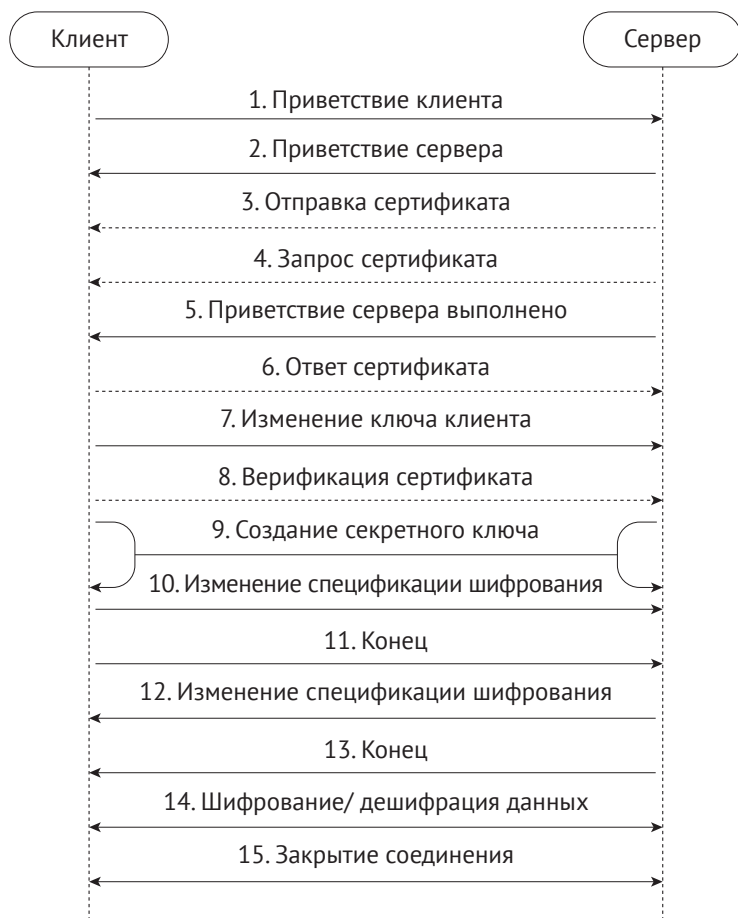


Рисунок 8.15. Процесс подтверждения связи по протоколу TLS

- (4) (Необязательно) Запрос сертификата. Когда серверу потребуется подтвердить сертификат клиента, сервер отправит клиенту сообщение с запросом сертификата, если была выбрана взаимная аутентификация.
- (5) Приветствие сервера выполнено. Сервер информирует клиента о том, что сервер отправил все сообщения о подтверждении связи, и сервер будет ждать отправки сообщений клиентом.
- (6) (Необязательно) Сертификат ответа. Если была выбрана взаимная аутентификация, клиент отправит свой сертификат на сервер. Затем сервер проверит личность клиента.
- (7) Обмен клиентскими ключами. Клиент использует открытый ключ сервера для шифрования открытого ключа клиента и начального значения генератора ключей перед отправкой их на сервер.
- (8) (Необязательно) Проверка сертификата. Если выбрана взаимная аутентификация, клиент использует локальный закрытый ключ для генерации цифровой подписи и отправляет ее на сервер для аутентификации с помощью полученного открытого ключа клиента.

- (9) Создание секретного ключа. Взаимодействующие стороны генерируют коммуникационный ключ на основе начального значения генератора ключей.
- (10) Измените спецификацию шифра. Клиент уведомляет сервер о том, что метод связи был переключен в зашифрованный режим.
- (11) Закончено. Клиент готов к зашифрованному общению.
- (12) Измените спецификацию шифра. Сервер уведомляет клиента о том, что способ связи был переключен в зашифрованный режим.
- (13) Закончено. Подготовьтесь к зашифрованному обмену данными на стороне сервера.
- (14) Зашифрованные/расшифрованные данные. Обе стороны используют клиентский ключ для шифрования/дешифрования содержимого сообщения с помощью алгоритма симметричного шифрования.
- (15) Закрытие соединения. После завершения связи любая из сторон отправляет сообщение с просьбой отключить TLS-соединение.

3. Создание сервера HTTP+TLS с помощью ESP-IDF

HTTPS (HTTP over SSL) шифрует HTTP-данные с помощью протокола SSL или TLS. По сравнению с HTTP, HTTPS может предотвратить кражу или изменение данных во время передачи, обеспечивая тем самым их целостность. В разделе 8.3.2 рассказывалось о том, как использовать ESP-IDF для создания HTTP-сервера. На самом деле создание HTTPS-сервера аналогично. Вызовите `httpd_ssl_start()`, чтобы запустить службу HTTP+TLS, и вызовите `httpd_register_uri_handler()`, чтобы зарегистрировать соответствующую функцию обратного вызова.

Исходный код

Для получения исходного кода функций `httpd_ssl_start()` и `httpd_register_uri_handler()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/https_server.

```
1. static esp_err_t root_get_handler(httpd_req_t *req)
2. {
3.     httpd_resp_set_type(req, "text/html");
4.     httpd_resp_send(req, "<h1>Hello Secure World! </h1>", HTTPD_RESP_USE_STRLEN);
5.     return ESP_OK;
6. }
7.
8. static const httpd_uri_t root = {
9.     .uri = "/",
10.    .method = HTTP_GET,
11.    .handler = root_get_handler
12. };
13.
14. esp_err_t esp_create_https_server(void)
15. {
16.     httpd_handle_t server = NULL;
17.     ESP_LOGI(TAG, "Starting server");
18.     httpd_ssl_config_t conf = HTTPD_SSL_CONFIG_DEFAULT();
19.     //Configure CA certificate and private key for the server
20.     extern const unsigned char cacert_pem_start[] asm("_binary_cacert_pem_start");
21.     extern const unsigned char cacert_pem_end[] asm("_binary_cacert_pem_end");
22.     conf.cacert_pem = cacert_pem_start;
```

```

23.     conf.cacert_len = cacert_pem_end - cacert_pem_start;
24.     extern const unsigned char prvtkey_pem_start[] asm("_binary_prvtkey_pem_start");
25.     extern const unsigned char prvtkey_pem_end[] asm("_binary_prvtkey_pem_end");
26.     conf.prvtkey_pem = prvtkey_pem_start;
27.     conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;
28.     //Start the HTTP+TLS server
29.     esp_err_t ret = httpd_ssl_start(&server, &conf);
30.     if (ESP_OK != ret) {
31.         ESP_LOGI(TAG, "Error starting server!");
32.         return ESP_FAIL;
33.     }
34.     //Set URI callback function
35.     ESP_LOGI(TAG, "Registering URI handlers");
36.     httpd_register_uri_handler(server, &root);
37.     return ESP_OK;
38. }

```

Приведенный код представляет собой пример создания HTTPS-сервера. Перед использованием этого примера, пожалуйста, вручную создайте сертификат CA и закрытый ключ в главном каталоге, используя следующую команду:

```
$ openssl req -newkey rsa:2048 -nodes -keyout prvtkey.pem -x509 -days 3650 -out cacert.pem -subj "/CN=ESP32 HTTPS server example"
```

Затем измените файл *MakeLists.txt*, чтобы скомпилировать сертификат в код:

```

1. idf_component_register(SRCS "station_example_main.c"
2.                        INCLUDE_DIRS ".")
3.                        EMBED_TXTFILES "cacert.pem"
4.                        "prvtkey.pem")

```

Кроме того, вам также необходимо зайти в меню конфигурации *idf.py* *menuconfig* → *Component config* (Конфигурация компонента) → **ESP HTTPS server** и настроить `CONFIG_ESP_HTTPS_SERVER_ENABLE`.

Введите `https://[IP-адрес вашего устройства]:443/` в браузере Chrome. Сертификат CA на стороне сервера не выдан центром сертификации, поэтому он не является доверенным. Таким образом, вы увидите экран, показанный на рис. 8.16.

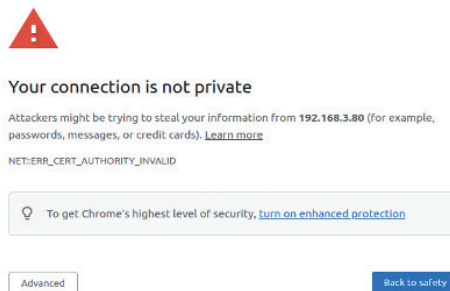


Рисунок 8.16. Сообщение ненадежного HTTPS-соединения

Пользователям необходимо нажать кнопку **Advanced** (Дополнительно), чтобы разрешить это ненадежное соединение. Рисунок 8.17 показывает интерфейс успешного соединения HTTPS.



Рисунок 8.17. Сообщение успешного соединения HTTPS

Если вы столкнетесь с сообщением **Header fields are too long for server to interpret** (Поля заголовка слишком длинные для интерпретации сервером), просто перейдите на *idf.py* `menuconfig` → `Component config` (Конфигурация компонента) → HTTP-сервер → `Max HTTP Request Header Length` (Максимальная длина заголовка HTTP-запроса) и увеличьте `HTTPD_MAX_REQ_HDR_LEN`. На рис. 8.18 показано сообщение о сбое HTTPS-соединения.

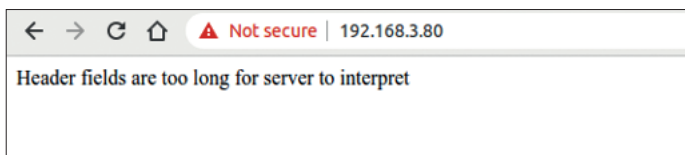


Рисунок 8.18. Сообщение о сбое соединения HTTPS

8.4.2. Введение в датаграмм-протокол безопасности транспортного уровня (DTLS)

DTLS – это протокол на основе UDP, который обслуживает прикладной уровень. Протокол TLS не может гарантировать безопасность данных, передаваемых по протоколу UDP. Таким образом, протокол DTLS был расширен на существующей архитектуре протокола TLS для поддержки UDP и стал версией протокола TLS, поддерживающей пакетную передачу данных. DTLS 1.0 основан на TLS 1.1, а DTLS 1.2 основан на TLS 1.2. Алгоритм шифрования, сертификация и процесс шифрования протокола DTLS в основном такие же, как у протокола TLS, поэтому не будут описываться в этом разделе.

1. Различия между DTLS и TLS

Принцип работы протокола DTLS в основном такой же, как у протокола TLS, за исключением следующих отличий:

- на этапе подтверждения связи протокол DTLS добавил механизм cookie. В версии 1.0 протокола DTLS добавлен механизм cookie, используемый сервером для проверки клиента и позволяющий избежать DoS-атак. Когда клиент отправляет сообщение «Client Hello» («Привет от клиента») на сервер, сервер напрямую не отвечает сообщением «Server Hello» («Привет от сервера») для выполнения процесса подтверждения. Вместо этого сервер отвечает клиенту сообщением «Hello Verify Request» («Привет, запрос на проверку»), содержащим значение файла cookie. Когда клиент получает это сообщение, он запишет значение файла cookie в приветственное сообщение клиента и повторно отправит его на сервер. После его получения

сервер проверяет список локальных файлов cookie, чтобы определить, требуется ли подтверждение подлинности;

- DTLS поддерживает механизм повторной передачи. Поскольку сам протокол UDP не поддерживает повторную передачу, как протокол TCP, протокол DTLS вводит такой механизм. Приведем пример с приведенным выше сообщением «Client Hello»: после того как клиент отправит это сообщение, клиент запустит таймер для получения «Hello Verify Request», которым сервер должен ответить; если сервер не ответит в течение определенного периода времени, клиент повторно отправит сообщение «Client Hello». Аналогично, как только сообщение отправлено, сервер активирует таймер для отслеживания таймаутов и определения необходимости повторной отправки сообщения;
- DTLS поддерживает упорядоченный прием. UDP не гарантирует порядок доставки пакетов, однако протокол DTLS добавил поле `message_seq` в сообщение о подтверждении связи. Получатель предоставит принимающий буфер для приема сообщений не по порядку (аналогично TCP) и обработает сообщения по порядку в соответствии с полем `message_seq`;
- DTLS поддерживает ограничение размера пакета. UDP – это пакетно ориентированный протокол, а TCP – потокоориентированный протокол. Протокол TCP поддерживает фрагментацию и повторную сборку пакетов. Однако когда UDP-сообщение превышает максимальную единицу передачи (MTU) канального уровня, оно может быть принудительно фрагментировано на IP-уровне. Затем получателю необходимо обработать фрагментированный пакет на основе IP-заголовка и повторно собрать исходные данные. Если один пакет будет потерян, все UDP-сообщение будет недействительным. Следовательно, в протоколе DTLS сообщения подтверждения связи сегментируются поверх UDP. Это делается путем добавления полей `fragment_offset` и `fragment_length` к сообщению о подтверждении, представляющих собой смещение этого сообщения относительно начала данных и длину этого сообщения соответственно.

2. Создание сервера CoAP+DTLS с помощью ESP-IDF

В следующем примере показано, как создать сервер CoAP+DTLS. Этот пример фактически совпадает с примером CoAP, представленным в разделе 8.3.4, за исключением двух функций, добавленных для поддержки протокола DTLS. Функция `coap_context_set_psk()` предназначена для установки шифрования PSK в протоколе DTLS, а также может использовать сертификат (PKI) для подтверждения протокола TLS. Функция `coap_new_endpoint(ctx, &serv_addr, COAP_PROTO_DTLS)` указывает, что узел поддерживает протокол DTLS.

Исходный код

Для получения исходного кода функции `coap_context_set_psk()` обратитесь по адресу: https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/coap. Инструкции по использованию PKI см.: https://github.com/espressif/esp-idf/tree/master/examples/protocols/coap_server.

```
1. static char psk_key[] = "esp32c3_key";  
2. static void esp_create_coaps_server(void)
```



```

3. {
4.     coap_context_t *ctx = NULL;
5.     coap_address_t serv_addr;
6.     coap_resource_t *resource = NULL;
7.     while (1) {
8.         coap_endpoint_t *ep = NULL;
9.         unsigned wait_ms;
10.
11.        //Create a CoAP server socket
12.        coap_address_init(&serv_addr);
13.        serv_addr.addr.sin6.sin6_family = AF_INET6;
14.        serv_addr.addr.sin6.sin6_port = htons(COAP_DEFAULT_PORT);
15.
16.        //Create CoAP ctx
17.        ctx = coap_new_context(NULL);
18.        if (!ctx) {
19.            ESP_LOGE(TAG, "coap_new_context() failed");
20.            continue;
21.        }
22.
23.        //Add PSK encryption key
24.        coap_context_set_psk(ctx, "CoAP",
25.                               (const uint8_t *)psk_key,
26.                               sizeof(psk_key) - 1);
27.
28.        //Set CoAP node
29.        ep = coap_new_endpoint(ctx, &serv_addr, COAP_PROTO_UDP);
30.        if (!ep) {
31.            ESP_LOGE(TAG, "udp: coap_new_endpoint() failed");
32.            goto clean_up;
33.        }
34.
35.        //Add DTLS node and port
36.        if (coap_dtls_is_supported()) {
37.            serv_addr.addr.sin6.sin6_port = htons(COAPS_DEFAULT_PORT);
38.            ep = coap_new_endpoint(ctx, &serv_addr, COAP_PROTO_DTLS);
39.            if (!ep) {
40.                ESP_LOGE(TAG, "dtls: coap_new_endpoint() failed");
41.                goto clean_up;
42.            } else {
43.                ESP_LOGI(TAG, "MbedTLS (D)TLS Server Mode not configured");
44.            }
45.        }
46.
47.        //Set CoAP resource URI
48.        resource = coap_resource_init(coap_make_str_const("light"), 0);
49.        if (!resource) {
50.            ESP_LOGE(TAG, "coap_resource_init() failed");
51.            goto clean_up;
52.        }
53.
54.        //Register callback functions for GET and PUT method corresponding to
CoAP resource URI
55.        coap_register_handler(resource, COAP_REQUEST_GET, esp_coap_get);
56.        coap_register_handler(resource, COAP_REQUEST_PUT, esp_coap_put);
57.

```

```

58. //Set CoAP GET resource visible
59. coap_resource_set_get_observable(resource, 1);
60.
61. //Add resource to CoAP ctx
62. coap_add_resource(ctx, resource);
63. wait_ms = COAP_RESOURCE_CHECK_TIME * 1000;
64. while (1) {
65.     //Wait to receive CoAP data
66.     int result = coap_run_once(ctx, wait_ms);
67.     if (result < 0) {
68.         break;
69.     } else if (result && (unsigned)result < wait_ms) {
70.         //Decrease waiting time
71.         wait_ms -= result;
72.     } else {
73.         //Reset waiting time
74.         wait_ms = COAP_RESOURCE_CHECK_TIME * 1000;
75.     }
76. }
77. }
78. clean_up:
79.     coap_free_context(ctx);
80.     coap_cleanup();
81. }

```

8.5 Практика: локальное управление в проекте Smart Light

Компонент локального управления (`esp_local_ctrl`) ESP-IDF позволяет легко управлять чипами Espressif через Wi-Fi + HTTPS или просто Bluetooth LE. С помощью этого компонента вы можете получить доступ к свойствам, определяемым приложением, которые можно считывать или записывать с помощью набора настраиваемых обработчиков. В этом разделе в основном представлен модуль локального управления на базе Wi-Fi. Если взять в качестве примера Smart Light, то локальный модуль управления может быть сконфигурирован следующим образом:

- настройте локальное устройство для обнаружения протокола mDNS;
- настройте локальный HTTPS-сервер и сертификат для передачи данных;
- настройте умное освещение.

В предыдущих разделах было представлено управление ESP32-C3 через Bluetooth LE. При использовании Bluetooth LE стек протоколов TCP/IP не задействован, и обнаружение локального устройства не требуется. Bluetooth имеет свою собственную службу обнаружения ресурсов.

8.5.1. Создание локального управляющего сервера на базе Wi-Fi

Следующий пример кода реализует локальный управляющий сервер на базе Wi-Fi. Локальный элемент управления основан на протоколе HTTP для передачи данных, а данные шифруются с использованием протокола TLS. Кроме того, в примере добавлен модуль mDNS для обнаружения устройств.

Исходный код

Полностью пример кода см. по адресу:

https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/local_control.

```
1. #define PROPERTY_NAME_STATUS "status"
2. static char light_status[64] = "{\"status\": true}";
3.
4. //Property type definition, used with scripts
5. enum property_types {
6.     PROP_TYPE_TIMESTAMP = 0,
7.     PROP_TYPE_INT32,
8.     PROP_TYPE_BOOLEAN,
9.     PROP_TYPE_STRING,
10. };
11.
12. //Get attribute value
13. esp_err_t get_property_values(size_t props_count,
14.                               const esp_local_ctrl_prop_t props[],
15.                               esp_local_ctrl_prop_val_t prop_values[],
16.                               void *usr_ctx)
17. {
18.     int i = 0;
19.     for (i = 0; i < props_count; i++) {
20.         ESP_LOGI(TAG, "Reading property : %s", props[i].name);
21.         if (!strncmp(PROPERTY_NAME_STATUS,
22.                     props[i].name,
23.                     strlen(props[i].name))) {
24.             prop_values[i].size = strlen(light_status);
25.             prop_values[i].data = &light_status; //prop_values[i].data is
26.                                             //just a pointer, and cannot be assigned.
27.             break;
28.         }
29.     }
30.     if (i == props_count) {
31.         ESP_LOGE(TAG, "Not found property %s", props[i].name);
32.         return ESP_FAIL;
33.     }
34.     return ESP_OK;
35. }
36.
37. //Set property value
38. esp_err_t set_property_values(size_t props_count,
39.                               const esp_local_ctrl_prop_t props[],
40.                               const esp_local_ctrl_prop_val_t prop_values[],
41.                               void *usr_ctx)
42. {
43.     int i = 0;
44.     for (i = 0; i < props_count; i++) {
45.         ESP_LOGI(TAG, "Setting property : %s", props[i].name);
46.         if (!strncmp(PROPERTY_NAME_STATUS,
47.                     props[i].name,
48.                     strlen(props[i].name))) {
49.             memset(light_status, 0, sizeof(light_status));
50.             strncpy(light_status,
```

```

51.         (const char *)prop_values[i].data,
52.         prop_values[i].size);
53.     if (strstr(light_status, "true")) {
54.         app_driver_set_state(true); //Turn on the smart light
55.     } else {
56.         app_driver_set_state(false); //Turn off the smart light
57.     }
58.     break;
59. }
60. }
61. if (i == props_count) {
62.     ESP_LOGE(TAG, "Not found property %s", props[i].name);
63.     return ESP_FAIL;
64. }
65. return ESP_OK;
66. }
67. #define SERVICE_NAME "my_esp_ctrl_device"
68. void esp_local_ctrl_service_start(void)
69. {
70.     //Initialise the HTTPS server-side configuration
71.     httpd_ssl_config_t https_conf = HTTPD_SSL_CONFIG_DEFAULT();
72.     //Load the server certificate
73.     extern const unsigned char cacert_pem_start[] asm("_binary_cacert_pem_start");
74.     extern const unsigned char cacert_pem_end[] asm("_binary_cacert_pem_end");
75.     https_conf.cacert_pem = cacert_pem_start;
76.     https_conf.cacert_len = cacert_pem_end - cacert_pem_start;
77.     //Load server-side private key
78.     extern const unsigned char prvtkey_pem_start[] asm("_binary_prvtkey_pem_start");
79.     extern const unsigned char prvtkey_pem_end[] asm("_binary_prvtkey_pem_end");
80.     https_conf.prvtkey_pem = prvtkey_pem_start;
81.     https_conf.prvtkey_len = prvtkey_pem_end - prvtkey_pem_start;
82.     esp_local_ctrl_config_t config = {
83.         .transport = ESP_LOCAL_CTRL_TRANSPORT_HTTPD,
84.         .transport_config = {
85.             .httpd = &https_conf
86.         },
87.         .proto_sec = {
88.             .version = PROTOCOLCOM_SEC0,
89.             .custom_handle = NULL,
90.             .pop = NULL,
91.         },
92.         .handlers = {
93.
94.         //User-defined processing function
95.             .get_prop_values = get_property_values,
96.             .set_prop_values = set_property_values,
97.             .usr_ctx = NULL,
98.             .usr_ctx_free_fn = NULL
99.         },
100.
101.         //Set the maximum number of attributes
102.         .max_properties = 10
103.     };
104.
105.     //Initialise local discovery
106.     mdns_init();
107.     mdns_hostname_set(SERVICE_NAME);
108.

```

```

109. //Start the local control service
110. ESP_ERROR_CHECK(esp_local_ctrl_start(&config));
111. ESP_LOGI(TAG, "esp_local_ctrl service started with name : %s", SERVICE_NAME);
112. esp_local_ctrl_prop_t status = {
113.     .name = PROPERTY_NAME_STATUS,
114.     .type = PROP_TYPE_STRING,
115.     .size = 0,
116.     .flags = 0,
117.     .ctx = NULL,
118.     .ctx_free_fn = NULL
119. };
120. //Add attribute value
121. ESP_ERROR_CHECK(esp_local_ctrl_add_property(&status));
122.}

```

Приведенный пример кода реализует обнаружение устройства (доменное имя: `my_esp_ctrl_device.local`) через протокол локального обнаружения (mDNS), устанавливает локальное управляющее соединение HTTPS и позволяет клиенту устанавливать и запрашивать значения атрибутов через зарегистрированную конечную точку обмена (endpoint).

Пользователи могут включить защиту безопасности передачи для местного управления с помощью следующих параметров:

- PROTOCOL SEC0: указывает используемый алгоритм сквозного шифрования;
- PROTOCOL SEC1: указывает, что обмен данными осуществляется в виде обычного текста;
- PROTOCOL SEC CUSTOM: настраивает требования безопасности.

Каждый атрибут должен иметь уникальное имя (строку), тип (например, `int`, `bool` или `string`), флаг (например, доступный только для чтения или доступный для чтения и записи) и размер. Если ожидается, что значение свойства будет переменной длины (например, если значение свойства представляет собой строку или поток байтов), размер следует сохранить равным 0. Для типов данных фиксированной длины, таких как `int`, `float` и т. д., установка в поле `size` правильного значения помогает функции `esp_local_ctrl` выполнять внутренние проверки параметров, полученных с помощью запросов на запись.

Вы можете обработать их, сопоставив `props[i].name` с соответствующим именем атрибута и последующей проверкой флагов и типа данных, чтобы определить, удовлетворяет ли атрибут соответствующим требованиям.

Конечные точки ввода /вывода по умолчанию приведены в табл. 8.5.

Таблица 8.5. Конечные точки по умолчанию

Имя конечной точки (сервер BLE + GATT)	URI (HTTPS-сервер + mDNS)	Описание
<code>esp_local_ctrl/version</code>	<code>https://my_esp_ctrl_device.local/esp_local_ctrl/version</code>	Для возвращения строки версии
<code>esp_local_ctrl/control</code>	<code>https://my_esp_ctrl_device.local/esp_local_ctrl/control</code>	Для отправки/получения управляющих сообщений

8.5.2. Проверка функциональности локального управления с помощью скриптов

В предыдущем разделе вы ознакомились с тем, как создать локальный модуль управления. В этом разделе будет рассказано о том, как использовать скрипты для проверки. Здесь мы используем созданный пример `esp_local_ctrl` в качестве примера для проверки.

1. Создайте сертификат для подтверждения связи TLS между клиентом и сервером.

- a. Сгенерируйте корневой центр сертификации, который будет использоваться для подписи сертификата на стороне сервера; клиент будет использовать его для проверки сертификата сервера во время подтверждения SSL. Необходимо задать кодовую фразу для шифрования сгенерированного `rootkey.pem`.

```
$ openssl req -new -x509 -subj "/CN=root" -days 3650 -sha256 -out rootCA.pem -keyout rootkey.pem
```

- b. Создайте запрос на подпись сертификата и его закрытый ключ `prvtkey.pem` для сервера.

```
$ openssl req -newkey rsa:2048 -nodes -keyout prvtkey.pem -days 3650 -out server.csr -subj "/CN=my esp ctrl device.local"
```

- c. Используйте ранее сгенерированный `rootCA` для обработки запроса на подпись сертификата на стороне сервера и сгенерируйте сертификат подписи `cacert.pem`. На этом шаге необходимо ввести парольную фразу, установленную ранее для зашифрованного `rootkey.pem`.

```
$ openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootkey.pem -CAcreateserial -out cacert.pem -days 500 -sha256
```

Среди сгенерированных сертификатов `cacert.pem` и `privkey.pem` компилируются на сервере, а `rootkey.pem` подходит для клиентских скриптов проверки на стороне сервера. Каталог сертификата можно задать в скрипте `esp_local_ctrl.py`.

```
1. def get_transport(sel_transport, service_name, check_hostname):
2. ...
3.     example_path = os.environ['IDF_PATH'] + '/examples/protocols/esp_local_ctrl'
4.                     cert_path = example_path + '/main/certs/rootCA.pem'
5. ...
```

2. Используйте следующую команду для подключения к локальному серверу управления с помощью скрипта. Если `sec_ver` равно 0, это означает, что на сервере установлен `PROTOCOLCOM_SEC0`.

```
$ python esp local ctrl.py --sec ver 0
```

Скрипт автоматически получит значение свойства, т. е.:

```
Connecting to my_esp_ctrl_device.local
```

```
==== Starting Session ====  
==== Session Established ====  
  
==== Available Properties ====  
S.N. Name Type Flags Value  
[1] status STRING {"status": true}
```

3. В соответствии с подсказкой сценария введите номер атрибута «1» и установите значение атрибута {"status": false}. Затем скрипт автоматически отправит запрос и обнаружит, что значение свойства было изменено:

```
Select properties to set (0 to re-read, 'q' to quit) : 1  
Enter value to set for property (status) : {"status": false}  
==== Available Properties ====  
S.N. Name Type Flags Value  
[1] status STRING {"status": false}  
  
Select properties to set (0 to re-read, 'q' to quit) :
```

8.5.3. Создание локального сервера управления на базе Bluetooth

Следующий пример кода создает локальный сервер управления на базе Bluetooth, который можно использовать для передачи данных (можно также обратиться к примеру `gatt_server`). Вы можете скомпилировать и прошить этот пример, затем использовать средство отладки Bluetooth на вашем смартфоне для сканирования и подключения к устройству Bluetooth с именем ESP32C3-LIGHT. После подключения вы можете получить доступ к Bluetooth-сервисам, предоставляемым устройством.

Исходный код

Полностью пример кода `gatt_server` см. по адресу:

https://github.com/espressif/book-esp32c3-iot-projects/tree/main/test_case/gatt_server.

В этом примере представлены две службы Bluetooth: одна для получения статуса устройства (UUID: FF01) и еще одна для установки статуса устройства (UUID: EE01), как показано на рис. 8.19.

Лог работы выглядит следующим образом:

```
I (387) GATTS_DEMO: NVS Flash initialization  
I (387) GATTS_DEMO: Application driver initialization  
I (397) gpio: GPIO[9]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1|  
Pulldown: 0| Intr:0  
W (437) BTDM_INIT: esp_bt_controller_mem_release not implemented, return OK  
I (437) BTDM_INIT: BT controller compile version [501d88d]  
I (437) coexist: coexist rom version 9387209  
I (437) phy_init: phy_version 500,985899c, Apr 19 2021, 16:05:08  
I (617) system_api: Base MAC address is not set  
I (617) system_api: read default base MAC address from EFUSE
```

```

I (617) BTDM_INIT: Bluetooth MAC: 68:ab:bc:a7:d8:d5
I (637) GATTS_DEMO: REGISTER_APP_EVT, status 0, app_id 0
I (647) GATTS_DEMO: CREATE_SERVICE_EVT, status 0, service_handle 40
I (647) GATTS_DEMO: SERVICE_START_EVT, status 0, service_handle 40
I (647) GATTS_DEMO: ADD_CHAR_EVT, status 0, attr_handle 42, service_handle 40
I (657) GATTS_DEMO: ADD_DESCR_EVT, status 0, attr_handle 43, service_handle 40
I (667) GATTS_DEMO: REGISTER_APP_EVT, status 0, app_id 1

```

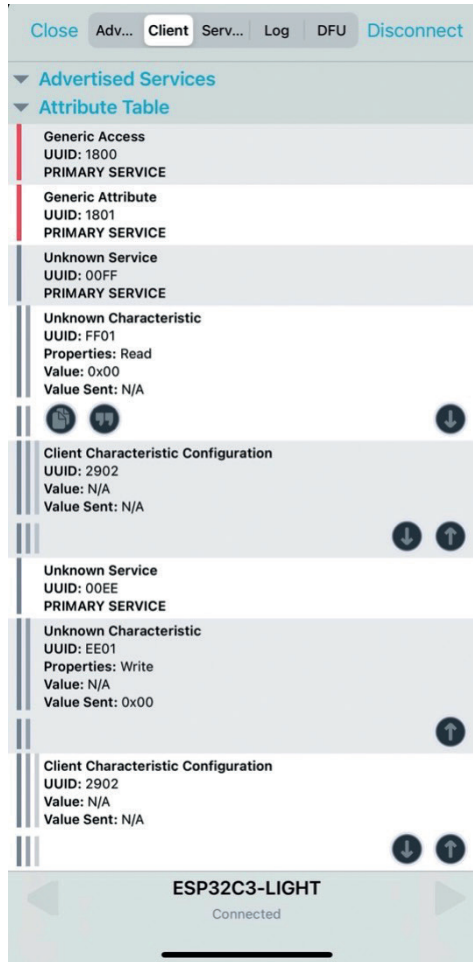


Рисунок 8.19. Сервисы Bluetooth из примера

```

I (677) GATTS_DEMO: CREATE_SERVICE_EVT, status 0, service_handle 44
I (677) GATTS_DEMO: SERVICE_START_EVT, status 0, service_handle 44
I (687) GATTS_DEMO: ADD_CHAR_EVT, status 0, attr_handle 46, service_handle 44
I (697) GATTS_DEMO: ADD_DESCR_EVT, status 0, attr_handle 47, service_handle 44
I (6687) GATTS_DEMO: ESP_GATTS_CONNECT_EVT, conn_id 0, remote 4a:13:d8:ca:b3:cf:
I (6687) GATTS_DEMO: CONNECT_EVT, conn_id 0, remote 4a:13:d8:ca:b3:cf:
I (6987) GATTS_DEMO: ESP_GATTS_MTU_EVT, MTU 500
I (6987) GATTS_DEMO: ESP_GATTS_MTU_EVT, MTU 500
I (7347) GATTS_DEMO: update connection params status = 0, min_int = 16, max_int

```



```
= 32,conn_int = 24,latency = 0, timeout = 400
I (15117) GATTS_DEMO: GATT_READ_EVT, conn_id 0, trans_id 3, handle 42
I (23037) GATTS_DEMO: GATT_WRITE_EVT, conn_id 0, trans_id 4, handle 46
I (23037) GATTS_DEMO: GATT_WRITE_EVT, value len 1, value :
I (23037) GATTS_DEMO: 00
I (23037) app_driver: Light OFF
I (30987) GATTS_DEMO: GATT_WRITE_EVT, conn_id 0, trans_id 5, handle 46
I (30987) GATTS_DEMO: GATT_WRITE_EVT, value len 1, value :
I (30987) GATTS_DEMO: 01
I (30987) app_driver: Light ON
```

8.6. Резюме

В этой главе мы впервые представили обзор рамочной модели, условий и сценариев применения локального управления. Мы также сравнили его с удаленным управлением, что позволило вам оценить пригодность функций локального управления в соответствии с вашими требованиями к проекту. Локальное обнаружение играет ключевую роль в локальном управлении, поскольку оно определяет способность смартфонов выполнять поиск устройств в локальной сети. Это позволяет смартфонам извлекать характеристики устройств и облегчает последующее управление идентифицированными устройствами. Следовательно, мы также углубились в режим работы локальных протоколов обнаружения, в частности с точки зрения основного уровня. Мы также провели сравнительный анализ характеристик двух режимов: широковещательного и многоадресного, с анализом их сходств и различий. Наиболее часто используемым протоколом локального обнаружения является mDNS, который предоставляет гибкие возможности реализации функций локального обнаружения в IoT-проектах. При применении Bluetooth для целей управления вы также можете напрямую использовать технологию обнаружения ресурсов, заложенную в этом протоколе.

Затем мы рассмотрели наиболее важные протоколы передачи данных и соответствующие алгоритмы шифрования данных в локальном управлении. Основными протоколами для передачи данных в локальном управлении являются TCP и UDP. Хотя вы можете напрямую использовать эти протоколы для локального управления, обычно это не рекомендуется из-за определенных ограничений. TCP и UDP классифицируются как протоколы транспортного уровня и по своей сути не передают данные прикладного формата, в отличие от таких протоколов, как HTTP и CoAP, которые включают прикладной уровень поверх транспортного. Кроме того, важно отметить, что протоколы транспортного уровня, такие как TCP и UDP, не поддерживают прямое шифрование данных с использованием TLS или DTLS. Следовательно, применение исключительно протокола транспортного уровня для передачи данных может не гарантировать безопасность передаваемых данных. Следовательно, рекомендуется использовать такие протоколы, как HTTP с TLS или CoAP с DTLS, для передачи данных в задачах локального управления.

Наконец, мы представили, как реализовать полную функцию локального управления на основе компонента `esp_local_ctrl` в ESP-IDF. Компонент `esp_local_ctrl` поддерживает локальное управление на основе Wi-Fi и Bluetooth с протоколами передачи данных, которые включают HTTPS для Wi-Fi и разнообразные

протоколы Bluetooth. Вы можете приступить к разработке локального элемента управления с помощью компонента `esp_local_ctrl`. Этот компонент сам по себе не поддерживает функцию обнаружения устройств локальной сети, потому что вам необходимо реализовать функциональность обнаружения устройств с помощью модуля `mDNS`. Компонент `esp_local_ctrl` широко используется в ESP-IDF, и вы можете найти функции подготовки сети и локальной связи в компоненте `wifi_provisioning`. Конечно, существуют различные протоколы и методы реализации для локального контроля. Если компонент `esp_local_ctrl` не соответствует вашим требованиям, вы можете следовать примерам кодов в разделах 8.2, 8.3 и 8.4, чтобы создать свою собственную локальную среду управления.

Глава 9

Управление через облако

Ознакомившись с локальным управлением, представленным в главе 8, вы должны теперь знать, как спроектировать функцию локального управления для проектов интернета вещей. Однако этой функции далеко недостаточно, поскольку полноценный IoT-проект нацелен на подключение всего и всюду, а локальное управление имеет географическое ограничение: смартфон должен находиться в той же локальной сети (LAN), что и управляемое устройство. Если вы хотите удаленно управлять домашними IoT-устройствами с помощью своего смартфона, вам понадобится функция удаленного управления.

В этой главе в основном рассказывается о том, как удаленно управлять устройствами на базе ESP32-C3. Цель состоит в том, чтобы помочь вам понять, что такое удаленное управление и его реализации, какой протокол используется, как создать локальный сервер MQTT для имитации облачного сервера и как создать модель продукта для удаленного управления устройством с помощью ESP RainMaker.

9.1. Введение в удаленное управление

Что такое удаленное управление? Как следует из названия, удаленное управление относится к поведению одного устройства (такого как смартфоны, компьютеры или другие сетевые устройства), управляющего другим устройством через глобальную сеть (WAN). Это не ограничено регионом. Например, вы можете управлять умным освещением своего дома с помощью смартфона на рабочем месте. Как правило, как устройство удаленного управления, так и управляемое устройство должно быть подключено к облачному серверу, и команды, отправляемые устройством управления, передаются на управляемое устройство через облачный сервер.

Подобно локальному управлению (рассмотренному в главе 8), удаленное управление также является способом передачи данных, но оно осуществляется не по локальной, а по глобальной сети. При локальном управлении сервером может быть само управляемое устройство или хост в локальной сети; управляющее устройство (например, мобильные телефоны или компьютеры) должно находиться в той же локальной сети, что и сервер, что является ограничением. При удаленном управлении сервером, как правило, является облачный сервер (несколько крупных поставщиков облачных серверов – Alibaba Cloud, Amazon Cloud, Tencent Cloud и т. д.). Управляющее и управляемое устройства нуждаются в подключении к нему, а пересылка и хранение данных также осуществляются облачным сервером.

Преимущество удаленного управления заключается в том, что оно является более гибким и может преодолевать пространственные ограничения. Однако, по сравнению с локальным управлением, для этого требуется оплата облачных сервисов и сетевого трафика, таким образом, этот метод более дорогостоящий. Кроме того, он обычно имеет более высокую задержку, что приводит к повышенному риску утечки данных.

Как следует из принципа реализации и компонентов ESP RainMaker, описанных в разделе 3.2, при удаленном управлении как управляющее устройство (смартфон), так и управляемое устройство (например, ESP32-C3) напрямую подключаются к облачному серверу, что облегчает передачу данных между устройствами. В результате важно иметь полное представление о том, как эти устройства взаимодействуют с облачным сервером.

Удаленное управление обходится дороже, чем локальное, поскольку для него требуются облачные серверы, но это компенсируется удобством удаленного просмотра текущего состояния управляемого устройства. И то, и другое имеет свои преимущества и недостатки. В настоящее время большинство устройств интернета вещей, представленных на рынке, могут быть подключены к различным облакам. Например, продукты Xiaomi, Alibaba и JD подключены к их собственной облачной платформе. Пользователю нужно только загрузить соответствующее приложение и выполнить настройку и привязку для просмотра и управления IoT-устройствами.

Если ваш смартфон и управляемое устройство находятся в одной локальной сети, лучше использовать вариант локального управления. В противном случае необходимо применять удаленное управление. Локальное управление имеет свои преимущества и сценарии использования. Преимущества обеих разновидностей должны быть полностью использованы для разработки наиболее подходящей технологии управления.

9.2. Облачные протоколы передачи данных

В разделе 9.1 описано, что такое дистанционное управление. Как показывает топологическая структура устройства дистанционного управления, смартфон и управляемое устройство напрямую не подключены. Они подключены к облачному серверу, и данные, отправляемые ими, пересылаются облаком. Тогда каков протокол для подключения устройства к облаку? Каков протокол передачи данных? Только разобравшись с этими протоколами, вы сможете получить базовое представление об удаленном управлении.

В настоящее время распространены протоколами для подключения устройств к облаку являются протоколы MQTT и HTTP. В этой главе будет рассмотрен только первый, поскольку второй был представлен в главе 8.

9.2.1. Введение в MQTT

MQTT (Message Queue Telemetry Transport) – это протокол передачи сообщений типа «издатель–подписчик» между сервером и клиентом. Протокол открытый, простой, облегченный, стандартизированный и несложный в реализации. Эти характеристики делают его стандартным протоколом передачи данных интернета вещей, который идеально подходит для устройств с ограниченными

ресурсами. Протокол был выпущен IBM в 1999 году. В настоящее время разработана версия 5.x, но ESP-IDF поддерживает версию 3.1.1. Две версии имеют существенные различия и несовместимы друг с другом. Большинство облачных платформ в настоящее время все еще полагаются на более старые версии v3.x. Поэтому в этой главе мы сосредоточимся на MQTT версии 3.x.

Протокол MQTT работает поверх протокола TCP. Он имеет следующие особенности:

- шаблон «издатель/подписчик», поддерживающий рассылку сообщений «один ко многим» и разделение приложений;
- транспорт обмена сообщениями, который не зависит от полезного содержимого;
- три качества обслуживания (Quality of service, QoS) для доставки сообщений;
- небольшие транспортные издержки; обмен в рамках протокола сведен к минимуму для уменьшения сетевого трафика;
- возможны уведомления заинтересованных сторон об аварийном отключении.

9.2.2. Принципы MQTT

Протокол MQTT основан на взаимодействии клиент–сервер. В нем определены три роли: издатель, брокер и подписчик. Серверы издателя и подписчика выступают в качестве клиентов, которые могут как публиковать сообщения, так и подписываться на них. Брокер выступает в роли сервера. На рис. 9.1 показана архитектура протокола.

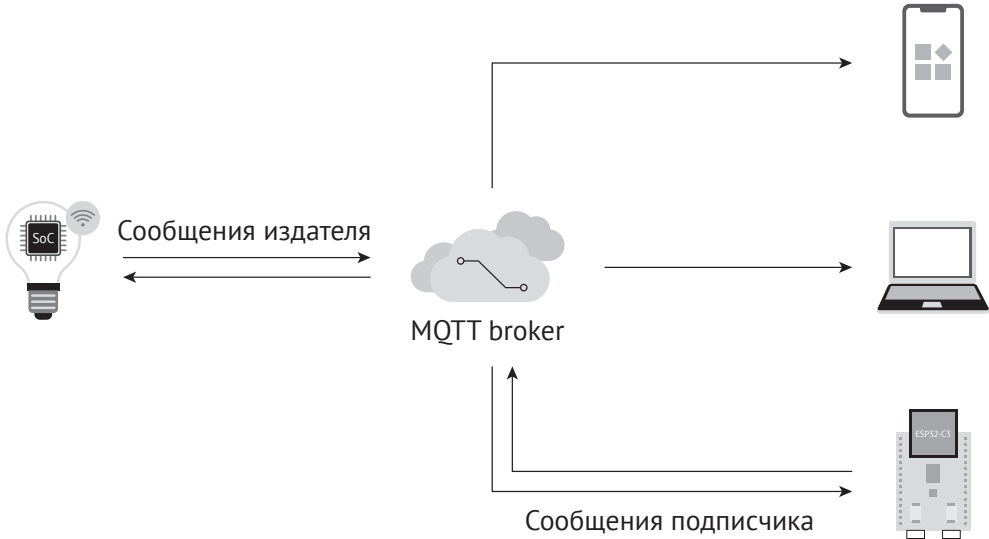


Рисунок 9.1. Архитектура протокола MQTT

(1) **Клиент:** устройства, на которых запущены приложения MQTT, такие как смартфоны и управляемые устройства. Клиент может работать как издатель или подписчик. Клиент всегда подключается к серверу по сети. Он может:

- публиковать прикладные сообщения, которые могут заинтересовать других клиентов;
 - подписываться на запрос прикладных сообщений, в получении которых он заинтересован;
 - отписываться, чтобы удалить запрос на получение прикладных сообщений;
 - отключаться от сервера.
- (2) **Сервер:** брокер, который действует как посредник между клиентами, публикующими прикладные сообщения, и клиентами, запрашивающими подписку на них. Брокером могут служить облачные платформы и облачные серверы. Он может:
- принимать сетевые подключения клиентов;
 - принимать прикладные сообщения от клиентов;
 - обрабатывать запросы на подписку и отмену подписки от клиентов;
 - пересылать прикладные сообщения, соответствующие клиентским подпискам.
- (3) **Подписка:** включает фильтр тем и максимальное качество обслуживания. Подписка ориентирована на один сеанс, в то время как сеанс может содержать несколько подписок. Каждая подписка в рамках сеанса имеет свой тематический фильтр.
- (4) **Тема:** ярлык, прикрепленный к сообщению приложения. Она сопоставляется с подписками, известными серверу. Сервер отправляет копию прикладного сообщения каждому клиенту, у которого есть соответствующая подписка.
- (5) **Фильтр по темам:** выражение, содержащееся в подписке, указывающее на интерес к одной или более тем. Фильтр темы может включать шаблоны для представления одного или нескольких символов.
- (6) **Сеанс:** взаимодействие между клиентом и сервером с отслеживанием состояния от начала до конца соединения. Некоторые сеансы длятся только до тех пор, пока существует сетевое подключение.
- (7) **Публикация/подписка:** ядро протокола MQTT. Позволяет осуществлять связь между подписчиками и издателями без знания IP-адреса или номера порта друг друга. Прямое подключение не требуется, подписчики и издатели могут работать, не зная о существовании друг друга. Обмен сообщениями между ними осуществляется брокером, который фильтрует все опубликованные сообщения и затем рассылает их соответствующим подписчикам.

Как подписчики, так и издатели связаны темой сообщения. Например, смартфон хочет проверить состояние умного светильника А. В этом случае смартфон может выступать в качестве подписчика, подписанного на сообщение с темой A/light_state от брокера. Умный светильник А может выступать в качестве издателя. Когда его состояние меняется, он публикует брокеру сообщение о состоянии с той же темой. Затем брокер фильтрует подписчиков, которые подписались на тему, и публикует для них сообщение о статусе. Таким образом смартфон может запрашивать статус светильника А.

9.2.3. Формат сообщения MQTT

Согласно определению протокола MQTT, управляющий пакет MQTT состоит из трех частей: фиксированного заголовка, переменного заголовка и данных.

(1) **Фиксированный заголовок** присутствует во всех управляющих пакетах MQTT.

Как показано на рис. 9.2, тип управляющего пакета занимает 4 бита.

Всего существует 14 типов управляющих пакетов, перечисленных в табл. 9.1.

Бит	7	6	5	4	3	2	1	0
Байт 1	Тип управляющего пакета				Специфический флаг типа управляющего пакета			
Байт 2	Оставшийся объем							

Рисунок 9.2. Фиксированный заголовок управляющих пакетов MQTT

Таблица 9.1. Типы управляющих пакетов MQTT

Название	Значение	Направление передачи	Описание
Зарезервировано	0	Запрещено	Зарезервировано
CONNECT	1	Сервер <= Клиент	Запрос клиента на подключение к серверу
CONNACK	2	Сервер => Клиент	Подтверждение соединения
PUBLISH	3	Сервер <=> Клиент	Опубликовать сообщение
PUBACK	4	Сервер <=> Клиент	Публикация сообщения QoS 1 подтверждена
PUBREC	5	Сервер <=> Клиент	Опубликовать полученное (гарантированная доставка, часть 1)
PUBREL	6	Сервер <=> Клиент	Публикация реализована (гарантированная доставка, часть 2)
PUBCOMP	7	Сервер <=> Клиент	Публикация сообщения QoS 2 завершена (гарантированная доставка, часть 3)
SUBSCRIBE	8	Сервер <= Клиент	Подписка по запросу клиента
SUBACK	9	Сервер => Клиент	Подтверждение подписки
UNSUBSCRIBE	10	Сервер <= Клиент	Запрос на отмену подписки
UNSUBACK	11	Сервер => Клиент	Подтверждение отмены подписки
PINGREQ	12	Сервер <= Клиент	PING-запрос
PINGRESP	13	Сервер => Клиент	PING-отклик
DISCONNECT	14	Сервер <= Клиент	Клиент отключен

Существует три уровня качества обслуживания MQTT: QoS 0, QoS 1 и QoS 2.

a. QoS 0: доставлено не более одного раза.

Передача сообщений полностью зависит от базовой сети TCP/IP. Поскольку протокол MQTT здесь не подразумевает ответ и повторную попытку, сообщение либо достигнет сервера только один раз, либо не достигнет вовсе. Информационный поток MQTT QoS 0 показан на рис. 9.3.

b. QoS 1: доставлено как минимум один раз.

Подтверждение получения сообщения предоставляется сообщением PUBACK. Если канал связи или передающее устройство неисправны либо сообщение не получено в течение указанного времени, отправитель повторно доставит сообщение, а в фиксированном заголовке пакета управления MQTT установится флаг дублирования (DUP). Поток информации MQTT QoS 1 показан на рис. 9.4.

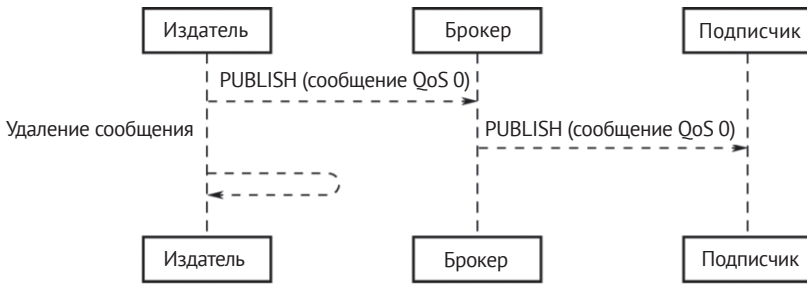


Рисунок 9.3. Информационный поток MQTT QoS 0

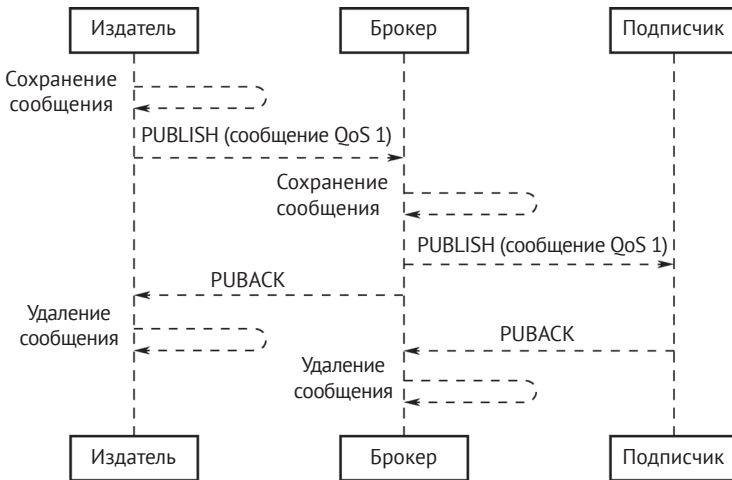


Рисунок 9.4. Информационный поток QoS MQTT 1

c. QoS 2: доставлено только один раз.

Это высочайшее качество обслуживания, при котором потеря и дублирование сообщений недопустимы и возникают повышенные накладные расходы. Информационный поток QoS 2 MQTT показан на рис. 9.5.

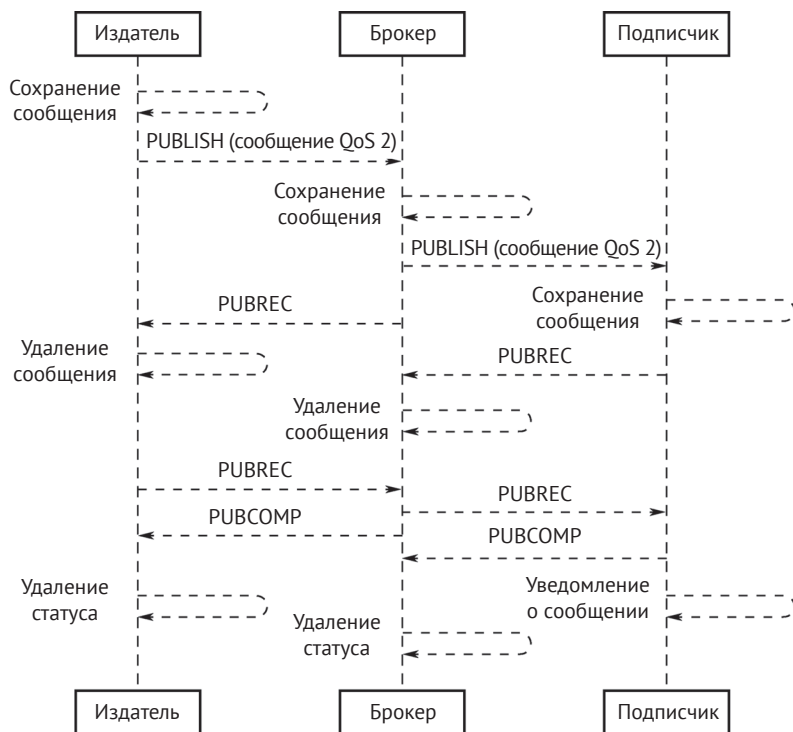


Рисунок 9.5. Информационный поток MQTT QoS 2

Биты [3–0] фиксированного заголовка содержат флаги, характерные для каждого типа управляющих пакетов. Кроме PUBLISH, биты флагов других типов берутся системой. Для типов без флагов биты зарезервированы. Если получены недопустимые флаги, получатель должен закрыть связь. Биты [3–0] в байте 1 заголовка пакета PUBLISH следующие:

- a. **DUP** (бит3): дублирование доставки.
 - «0» означает, что клиент или сервер запрашивает отправку PUBLISH первый раз.
 - «1» указывает, что это может быть повторная доставка более раннего сообщения. Флаг DUP для сообщений QoS 0 должен быть установлен на 0.
- b. **QoS** (биты [2–1]): определение количества доставляемых сообщений.
 - В табл. 9.2 показано, как представлять значения QoS в битах [2–1].

Таблица 9.2. Представление значений QoS в битах [2–1]

Значение QoS	Бит 2	Бит 1	Описание
0	0	0	Доставка не более одного раза
1	0	1	Доставка как минимум один раз
2	1	0	Доставка только один раз
-	1	1	Зарезервировано

с. **RETAIN** (bit0): определение необходимости сохранения сообщения.

Если этот флаг установлен в 1 в сообщении PUBLISH, отправленном клиентом на сервер, сервер должен сохранить это сообщение и его QoS, чтобы позже передать их подписчикам на соответствующие темы. Каждый клиент, подписывающийся на шаблон темы, который соответствует теме сохраненного сообщения, получает его сразу после подписки. Флаг RETAIN обычно используется для будущих сообщений. Например, после неожиданного отключения устройства брокер отправит сообщение о событии на смартфон, где устройство будет отображаться как отключенное.

Второй и последующие байты указывают оставшийся объем: они содержат количество оставшихся байтов в текущем пакете, включая данные в переменном заголовке и данные сообщения. Остаточный объем кодируется с использованием схемы кодирования переменной длины, которая применяет один байт для значений до 127. Для больших значений данные кодируются младшими семью битами каждого байта, а старший бит используется для указания того, есть ли в представлении следующие байты. Таким образом, каждый байт кодирует 128 значений и «бит продолжения». Максимальное количество байтов в поле оставшегося объема равно 4. Значения количества байтов оставшегося объема показаны в табл. 9.3.

Таблица 9.3. Значения количества байтов оставшегося объема

Байт	Минимальное значение	Максимальное значение
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

(2) Переменный заголовок

Некоторые типы пакетов управления MQTT содержат переменный компонент заголовка. Он находится между фиксированным заголовком и данными. Содержимое переменного заголовка зависит от типа пакета. Поле Packet Identifier в переменном заголовке представляет несколько типов пакетов, таких как PUBLISH (при QoS=0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, и UNSUBACK.

(3) **Данные:** третья часть управляющего пакета MQTT, представляющая содержание сообщения.

Данные бывают пяти типов пакетов: CONNECT, SUBSCRIBE, SUBACKUP, UNSUBSCRIBE и PUBLISH.

- a. CONNECT: идентификатор клиента, тема, сообщение, имя пользователя и пароль.
- b. SUBSCRIBE: тема для подписки и уровень QoS.
- c. SUBACKUP: подтверждение сервера, ответ на тему (на которую клиент запрашивает подписку) и QoS.

d. UNSUBSCRIBE: тема для отписки.

e. PUBLISH: прикладное сообщение, которое должно быть опубликовано, может быть нулевой длины.

9.2.4. Сравнение протоколов

В главе 8 были представлены такие протоколы, как TCP, HTTP, UDP и CoAP, которые могут служить для местного управления. Их также можно использовать для удаленного управления.

MQTT и TCP

MQTT – это прикладной протокол, основанный на протоколе TCP. И то, и другое можно использовать для удаленной передачи данных. Для программирования сокетов протокол TCP требует, чтобы пользователи разрабатывали свои собственные прикладные протоколы, которые имеют ограниченное удобство использования в существующей среде интернета вещей. С другой стороны, MQTT – это стандартизированный облегченный протокол для IoT, широко использующийся большинством облачных серверов, таких как Alibaba Cloud и Amazon Cloud, что делает его удобным для интеграции продуктов.

MQTT и HTTP

Оба являются протоколами клиент-серверных приложений, основанных на TCP. Однако, по сравнению с MQTT, HTTP имеет гораздо большие накладные расходы по размеру сообщений, а HTTP-серверу, как правило, сложно инициировать отправку данных клиентам, что может не соответствовать требованиям удаленного управления в IoT. В случаях, когда требуется только односторонняя передача данных от клиентов к серверу, можно использовать протокол HTTP.

MQTT и CoAP

Подобно HTTP, CoAP применяет модель REST, в которой сервер создает ресурсы в формате URI, а клиенты получают доступ к этим ресурсам с помощью методов GET, PUT, POST и DELETE. Кроме того, CoAP также имеет стиль протокола, аналогичный HTTP, но требует меньше ресурсов устройства и накладных расходов на коммуникацию, что делает его подходящим для интернета вещей. Однако CoAP может оказаться неподходящим выбором для дистанционного управления. Если смартфоны отправляют управляющие команды для удаленного управления, может потребоваться сложная архитектура CoAP + Web + база данных + приложение. Для протокола CoAP управляющие команды должны проходить через базу данных, прежде чем попасть на устройство, поскольку CoAP не имеет подключения. Когда смартфоны отправляют управляющие команды, сервер сначала сохраняет управляющие команды в базе данных, и устройство запрашивает сервер с помощью метода GET, чтобы проверить наличие управляющих команд, а затем принимает решение о том, следует ли действовать. В отличие от этой длинной и ресурсоемкой процедуры, MQTT ориентирован на подключение к серверу, и сервер будет прямо пересылать управляющие команды со смартфонов на все подписанные устройства, не сохраняя их. Для реализации удаленного управления необходимо только клиент MQTT + сервер MQTT + приложение, и можно утверждать, что MQTT более выгоден с точки зрения развертывания.

9.2.5. Настройка MQTT Broker в Linux и Windows

Некоторыми широко используемыми брокерами MQTT являются Mosquitto, MQTT и HiveMQ. HiveMQ не выступает открытым исходным кодом и является платным, поэтому он может не подходить для локального тестирования. EMQTT обладает мощными функциями, такими как просмотр трафика данных в веб-интерфейсе, и может использоваться на большинстве облачных серверов, причем доступны как бесплатные, так и платные пользовательские версии.

В этом разделе рассказывается о том, как использовать Mosquitto для настройки Mosquitto broker в Windows или Linux. Mosquitto – это брокер сообщений с открытым исходным кодом (лицензия EPL/EDL), который реализует протокол MQTT версий 5.0, 3.1.1 и 3.11. Считается облегченным программным обеспечением с открытым исходным кодом. Проект Mosquitto предоставляет библиотеку языка C для реализации популярных MQTT-клиентов командной строки `mosquitto_pub` и `mosquitto_sub`. Для получения дополнительной информации, пожалуйста, посетите его официальный веб-сайт.

1. Настройка Mosquitto broker в Linux

Все команды терминала в этом разделе должны выполняться от имени пользователя. Символ \$ представляет приглашение командной строки.

(1) Загрузите `mosquitto-2.0.12.tar.gz` с <https://mosquitto.org/files/source>.

(2) Извлеките Mosquitto.

```
$ tar -zxvf mosquitto-2.0.12.tar.gz
```

Проверьте успешность установки с помощью `mosquitto --help`.

```
$ cd mosquitto-2.0.12/src
$ mosquitto --help
mosquitto version 2.0.12
mosquitto is an MQTT v5.0/v3.1.1/v3.1 broker.
Usage: mosquitto [-c config_file] [-d] [-h] [-p port]
-c : specify the broker config file.
-d : put the broker into the background after starting.
-h : display this help.
-p : start the broker listening on the specified port.
    Not recommended in conjunction with the -c option.
-v : verbose mode - enable all logging types. This overrides
    any logging options given in the config file.
See https://mosquitto.org/ for more information.
```

(3) Запустите Mosquitto broker и протестируйте его в клиенте MQTT.

а. Запустите MQTT:

```
$ mosquitto
```

б. Используйте `mosquitto_sub`, чтобы подписаться на тему topic:

```
$ mosquitto sub -t 'test/topic' -v
```

в. Откройте новый терминал и используйте `mosquitto_pub` для публикации данных:

```
$ mosquitto pub -t 'test/topic' -m 'hello world'
```

d. В исходном терминале, где была подписана тема, просмотрите полученные данные:

```
$ mosquitto sub -t 'test/topic' -v  
test/topic hello world
```

2. Настройка Mosquitto broker в Windows

- (1) Загрузите 32-разрядный или 64-разрядный установочный пакет MQTT в зависимости от архитектуры вашего компьютера. Дважды щелкните для установки.
- (2) Откройте окно командной строки, перейдите в каталог, в котором установлен Mosquitto, и запустите Mosquitto broker:

```
cd C:\Program Files\mosquitto\
```

- (3) Используйте *mosquitto_sub.exe*, чтобы подписаться на тему topic:

```
C:\Program Files\mosquitto>mosquitto sub.exe -t 'test/topic' -v
```

- (4) Используйте *mosquitto_pub.exe* для публикации данных:

```
C:\Program Files\mosquitto>mosquitto pub.exe -t 'test/topic' -m 'hello world'
```

9.2.6. Настройка клиента MQTT на основе ESP-IDF

Компонент, применяемый в ESP-IDF для реализации MQTT-клиента, – это ESP-MQTT, имеющий следующие особенности:

- поддержка MQTT, MQTT через TLS, MQTT через WebSocket и MQTT через WebSocket и TLS;
- простота настройки с помощью URI;
- несколько клиентов в одном приложении;
- поддержка подписки, публикации, аутентификации, последних сообщений, проверки активности и QoS сообщения.

Следующий код основан на ESP-IDF и создает соединение с локальным Mosquitto broker. Для получения полного кода откройте проект ESP-IDF на GitHub и перейдите в каталог *esp-idf/examples/protocols/mqtt/tcp*.

```
1. //MQTT event handler function  
2. static void mqtt_event_handler(void *handler_args, esp_event_base_t base,  
3.                               int32_t event_id, void *event_data)  
4. {  
5.     esp_mqtt_event_handle_t event = event_data;  
6.     esp_mqtt_client_handle_t client = event->client;  
7.     int msg_id;  
8.     switch ((esp_mqtt_event_id_t)event_id) {  
9.         case MQTT_EVENT_CONNECTED:  
10.            ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");  
11.            //Subscribe to topic /topic/test  
12.            msg_id = esp_mqtt_client_subscribe(client, "/topic/test", 0);
```

```

13.     ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);
14.     break;
15.     case MQTT_EVENT_DISCONNECTED:
16.     ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
17.     break;
18.     case MQTT_EVENT_SUBSCRIBED:
19.     ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
20.     break;
21.     case MQTT_EVENT_UNSUBSCRIBED:
22.     ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
23.     break;
24.     case MQTT_EVENT_PUBLISHED:
25.     ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
26.     break;
27.     case MQTT_EVENT_DATA:
28.     ESP_LOGI(TAG, "MQTT_EVENT_DATA");
29.     ESP_LOGI(TAG, "TOPIC=%.*s\r\n", event->topic_len, event->topic);
30.     ESP_LOGI(TAG, "DATA=%.*s\r\n", event->data_len, event->data);
31.     break;
32.     case MQTT_EVENT_ERROR:
33.     ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
34.     break;
35.     default:
36.     ESP_LOGI(TAG, "Other event id:%d", event->event_id);
37.     break;
38.     }
39. }
40.
41. #define CONFIG_BROKER_URL "mqtt://192.168.3.4/"
42.
43. static void esp_mqtt_start(void)
44. {
45.     //Configure MQTT URI
46.     esp_mqtt_client_config_t mqtt_cfg = {
47.         .uri = CONFIG_BROKER_URL,
48.     };
49.
50.     //Initialise MQTT client
51.     esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
52.
53.     //Register event handler function
54.     esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
55.
56.     //Start MQTT client
57.     esp_mqtt_client_start(client);
58. }

```

Клиентское устройство подключается к MQTT Broker и подписывается на тему /topic/test. После того как другой клиент MQTT опубликует сообщение «hello world» в теме /topic/test, в устройстве появится следующий текст:

```

I (2598) wifi station: MQTT_EVENT_CONNECTED
I (2598) wifi station: sent subscribe successful, msg_id=25677
I (2648) wifi station: MQTT_EVENT_SUBSCRIBED, msg_id=25677
I (314258) wifi station: MQTT_EVENT_DATA
I (314258) wifi station: TOPIC=/topic/test
I (314258) wifi station: DATA=hello world

```

9.3. Обеспечение безопасности данных MQTT

Данные, передаваемые по протоколу MQTT, представляют собой обычный текст, значит, их можно перехватить, если они не зашифрованы. В разделе 8.4.1 протокол TLS представлен как средство обеспечения того, что данные могут быть расшифрованы только взаимодействующими сторонами, тем самым обеспечивая безопасность и легитимность данных.

Точно так же TLS можно использовать для шифрования в связи через облако и MQTT. Это уже было рассмотрено в разделе 8.4.1, в данном разделе рассказывается только о том, что за сертификаты в TLS-подтверждении имеются в виду и какие функции они выполняют; а также как можно генерировать сертификаты локально и как настроить среду TLS для взаимной аутентификации на основе MQTT Broker.

9.3.1. Значение и функция сертификатов

1. Введение

Сертификаты, также известные как сертификаты открытого ключа (public-key certificates, PKC), содержат личную информацию, такую как имя пользователя, название организации, адрес электронной почты, открытый ключ пользователя и цифровая подпись центра сертификации (CA). Вы можете рассматривать сертификат как личную идентификационную карту, где открытый ключ служит номером карты, идентифицирующим, какое физическое лицо она представляет. Центр сертификации CA – это как полицейский участок, который выдает ИД-карту. Это может быть международная организация, государственное учреждение, коммерческая компания или обычное физическое лицо.

2. Генерация сертификата

- (1) Пользователь А локально генерирует пару секретных ключей, используя алгоритм асимметричного шифрования.
- (2) Пользователь А отправляет локально сгенерированный файл с информацией об открытом ключе и сертификате в центр сертификации для цифровой подписи и генерации сертификата.
- (3) Центр сертификации генерирует локально пару закрытый–открытый ключ, которая является ключом `ca.key`, упомянутым в примере далее. Центр сертификации использует свой закрытый ключ для цифровой подписи открытого ключа пользователя А и выдает сертификат.
- (4) Пользователь В получает открытый ключ центра сертификации, который находится в открытом доступе, и использует его для проверки легитимности цифровой подписи в сертификате пользователя А. Успешная проверка подтверждает, что открытый ключ в сертификате пользователя А принадлежит пользователю А.
- (5) Чтобы отправить данные пользователю А, пользователю В нужно только зашифровать данные с помощью открытого ключа из сертификата пользователя А и отправить их. Затем пользователь А расшифровывает данные с помощью своего собственного закрытого ключа.

До сих пор мы рассматривали генерацию сертификата пользователя А и процесс передачи данных между пользователем А и пользователем В. Однако он включает только одностороннюю аутентификацию сертификата TLS – пользователь А подобен серверной стороне, пользователь В подобен клиентской стороне, и этот процесс можно рассматривать как проверку клиентом сертификата сервера. Аналогичный процесс происходит при проверке сервером сертификата клиента.

3. Функция сертификата

Как видно из приведенного выше процесса генерации сертификатов, сертификаты являются средством проверки легитимности одноуровневого устройства. Только когда она осуществляется, передаваемые данные могут быть гарантированы от риска утечек.

4. Стандарт сертификата

Все сертификаты соответствуют международному стандарту X.509 ITU-T³¹. Структура сертификатов X.509 описана и закодирована с использованием абстрактной синтаксической нотации один (Abstract Syntax Notation One, ASN1).

Обычно сертификат состоит из следующих полей:

- **номер версии:** номер версии спецификации. Текущая версия равна 3, что соответствует значению 0x2;
- **серийный номер:** уникальный серийный номер, который поддерживается центром сертификации и присваивается каждому сертификату для отслеживания и отзыва. Максимальный размер составляет 20 байт;
- **алгоритм подписи:** алгоритм, используемый для создания цифровых подписей;
- **срок действия:** срок действия сертификата, включая даты начала и окончания;
- **субъект:** идентификационная информация владельца сертификата, а именно личная информация, упомянутая выше;
- **информация об открытом ключе субъекта:** защищенная информация, относящаяся к открытому ключу, включая алгоритм открытого ключа и уникальный идентификатор субъекта.

5. Формат сертификата

Письмо повышенной конфиденциальности (Privacy Enhanced Mail, PEM) – распространенный формат сертификатов X.509. Файлы PEM обычно встречаются с расширениями *.crt* или *.cer* (для сертификатов), *.key* (для закрытых ключей) и *.csr* (для запроса подписи сертификата).

Файл PEM представляет собой текстовый файл, который обычно содержит заголовки, нижние колонтитулы и блоки содержимого, закодированные в Base64.

9.3.2. Локальная генерация сертификатов

OpenSSL – это криптографическая библиотека Secure Socket Layer (SSL) с открытым исходным кодом, которая обеспечивает функции для алгоритмов,

³¹ ITU-T полностью расшифровывается как International Telecommunication Union Telecommunication Standardization Sector – Сектор стандартизации телекоммуникаций Международного союза электросвязи.

управление упаковкой ключей и сертификатов и выполнение протокола SSL. Он состоит из трех частей: библиотеки протокола SSL, инструментов командной строки для приложений и библиотеки криптографических алгоритмов. Следующие примеры демонстрируют, как использовать библиотеку для создания сертификатов и ключей в Linux.

1. Генерация закрытых ключей для сертификата

Эта команда создает закрытый ключ (2048 бит) для сертификата. Открытый ключ может быть извлечен из него.

```
$ openssl genrsa -out ca.key 2048
```

```
Generating RSA private key, 2048 bit long modulus
.....
.....+++
.....+++
e is 65537 (0x10001)
```

Следующая команда создает закрытый ключ (2048 бит) для сертификата сервера:

```
$ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

Следующая команда создает закрытый ключ (2048 бит) для сертификата клиента:

```
$ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

Рекомендуемая минимальная длина ключа для алгоритма RSA составляет 2048 бит. Если длина ключа 1024 бита, `mbdctl` отклонит согласование TLS из-за низкой безопасности.

2. Запрос подписи для сертификата (CSR)

Эта команда создает запрос подписи сертификата (certificate sign request, CSR), требуемой для сертификата CA. Введите необходимую информацию в соответствии с запросом. Название организации (Organization Name) может вводиться по желанию, потому что подпись предназначена только для локального использования.

```
$ openssl req -out ca.csr -key ca.key -new
Вас попросят ввести информацию, которая будет включена в ваш запрос сертификата. То, что вы собираетесь ввести, называется отличительным (distinguished) именем, или DN. Полей довольно много, но вы можете оставить некоторые пустыми. Для некоторых полей будет значение по умолчанию, если вы введете «.», поле останется пустым.
-----
```

```
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IOT Certificate Test
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Следующая команда создаст CSR, требуемый сертификатом сервера. Обратите внимание, что поле Common Name должно быть заполнено доменным именем или IP-адресом сервера.

```
$ openssl req -out server.csr -key server.key -new
```

Вас попросят ввести информацию, которая будет включена в ваш запрос сертификата. То, что вы собираетесь ввести, называется отличительным (distinguished) именем, или DN. Полей довольно много, но вы можете оставить некоторые пустыми. Для некоторых полей будет значение по умолчанию, если вы введете «.», поле останется пустым.

```
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MQTT Server
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.3.4
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Создание сертификата CA, сертификата сервера и сертификата клиента

Следующая команда создаст сертификат CA ca.crt:

```
$ openssl x509 -req -in ca.csr -out ca.crt -sha256 -days 5000 -signkey ca.key
Signature ok
subject=/C=CN/ST=Some-State/O=IOT Certificate Test
Getting Private key
```

Следующая команда создаст сертификат сервера server.crt:

```
$ openssl x509 -req -in server.csr -out server.crt -sha256 -CAcreateserial -days 5000 -CA ca.crt -CAkey ca.key
Signature ok
subject=/C=CN/ST=Some-State/O=MQTT Server/CN=192.168.3.4
Getting CA Private Key
```

Следующая команда создаст клиентский сертификат client.crt:

```
$ openssl x509 -req -in client.csr -out client.crt -sha256 -CAcreateserial -days 5000 -CA ca.crt -CAkey ca.key
Signature ok
subject=/C=CN/ST=Some-State/O=MQTT Client/CN=192.168.3.5
Getting CA Private Key
```

Примечание

Не используйте алгоритм SHA1, так как `mbedtls` может отклонить согласование TLS из-за низкого уровня безопасности.

9.3.3. Настройка MQTT Broker

В разделе 9.2.5 описано, как настроить MQTT Broker в Windows или Linux с помощью Mosquitto, а в этом разделе будет рассказано, как это сделать через TLS.

Во-первых, в корневом каталоге `mosquitto` откройте файл конфигурации `mosquitto.conf` и добавьте абсолютный путь к файлам, созданным в разделе 9.3.2, включая сертификат CA (`ca.crt`), сертификат сервера (`server.crt`) и сертификат закрытого ключа сервера (`server.key`). Команда выглядит следующим образом:

```
$ port 8883
certfile {absolute path}/server.crt
keyfile {absolute path}/server.key
cafile {absolute path}/ca.crt
require_certificate true
use_identity_as_username true
```

Затем перезапустите Mosquitto и загрузите файл конфигурации:

```
$ mosquitto -c mosquitto.conf -v
1635927859: mosquitto version 1.6.3 starting
1635927859: Config loaded from mosquitto.conf.
1635927859: Opening ipv4 listen socket on port 8883.
1635927859: Opening ipv6 listen socket on port 8883.
```

9.3.4. Настройка клиента MQTT

В разделе 9.2.5 мы рассмотрели, как настроить клиент MQTT на основе ESP-IDF, используя MQTT через TCP, но такой подход не может гарантировать безопасность данных. В этом разделе мы представим более безопасный подход, который заключается в настройке клиента с использованием MQTT через TLS.

```
1. extern const uint8_t client_cert_pem_start[] asm("_binary_client_cert_start");
2. extern const uint8_t client_cert_pem_end[] asm("_binary_client_cert_end");
3. extern const uint8_t client_key_pem_start[] asm("_binary_client_key_start");
4. extern const uint8_t client_key_pem_end[] asm("_binary_client_key_end");
5. extern const uint8_t server_cert_pem_start[] asm("_binary_ca_cert_start");
6. extern const uint8_t server_cert_pem_end[] asm("_binary_ca_cert_end");
7.
8. #define CONFIG_BROKER_URL "mqtt://192.168.3.4/"
9.
10. //Configure MQTT URI
11. esp_mqtt_client_config_t mqtt_cfg = {
12.     .uri = CONFIG_BROKER_URL,
13.     .client_cert_pem = (const char *)client_cert_pem_start,
14.     .client_key_pem = (const char *)client_key_pem_start,
15.     .cert_pem = (const char *)server_cert_pem_start,
16. };
```

- (1) Загрузите сертификат клиента (`client.crt`), закрытый ключ клиента (`client.key`) и сертификат СА, который аутентифицирует сервер (`ca.crt`).
- (2) Измените предыдущее соединение MQTT на `mqtts`.
- (3) Используйте номер порта по умолчанию 8883.
- (4) Измените файл `CMakeLists.txt`, чтобы сертификаты загружались во встроенное ПО во время компиляции. Код выглядит следующим образом:

```
1. target_add_binary_data(${CMAKE_PROJECT_NAME}.elf "main/client.crt" TEXT)
2. target_add_binary_data(${CMAKE_PROJECT_NAME}.elf "main/client.key" TEXT)
3. target_add_binary_data(${CMAKE_PROJECT_NAME}.elf "main/ca.crt" TEXT)
```

После компиляции и прошивки подключите устройство к Wi-Fi. Затем и клиент, и сервер покажут лог успешного подключения, а клиент подпишется на тему `/topic/test`. Лог сервера следующий:

```
1635927859: mosquitto version 1.6.3 starting
1635927859: Config loaded from mosquitto.conf.
1635927859: Opening ipv4 listen socket on port 8883.
1635927859: Opening ipv6 listen socket on port 8883.
1635927867: New connection from 192.168.3.5 on port 8883.
1635927869: New client connected from 192.168.3.5 as ESP32_2465F1 (p2, c1, k120, u'192.168.3.5').
1635927869: No will message specified.
1635927869: Sending CONNACK to ESP32_2465F1 (0, 0)
1635927869: Received SUBSCRIBE from ESP32_2465F1
1635927869: /topic/test (QoS 0)
1635927869: ESP32_2465F1 0 /topic/test
1635927869: Sending SUBACK to ESP32_2465F1
```

Теперь используйте `mosquitto_pub`, чтобы отправить «hello world» в тему `/topic/test`. Проверьте, что устройство получило сообщение. Команда выглядит следующим образом:

```
$ mosquitto pub -h 192.168.3.4 -p 8883 -t /topic/test" -m 'hello world'
--cafile ca.crt --cert client.crt --key client.key
```

На стороне устройства будет выведен следующий лог:

```
I (1600) esp_netif_handlers: sta ip: 192.168.3.5, mask: 255.255.255.0, gw: 192.168.3.1
I (1600) wifi station: got ip:192.168.3.5
I (1600) wifi station: connected to ap SSID:myssid password:12345678
I (1610) wifi station: Other event id:7
W (1630) wifi:<ba-add>idx:0 (ifx:0, 34:29:12:43:c5:40), tid:0, ssn:4, winSize:64
I (4110) wifi station: MQTT_EVENT_CONNECTED
I (4120) wifi station: sent subscribe successful, msg_id=42634
I (4140) wifi station: MQTT_EVENT_SUBSCRIBED, msg_id=42634
I (10290) wifi station: MQTT_EVENT_DATA
I (10290) wifi station: TOPIC=/topic/test
I (10290) wifi station: DATA=hello world
```

9.4. Практика: дистанционное управление через ESP RainMaker

После прочтения предыдущих глав у вас должно быть общее представление о настройке Wi-Fi и протокола MQTT. В этом разделе мы углубимся в проект Smart Light, обсуждаемый в данной книге. Мы будем использовать ESP RainMaker для обеспечения умного освещения дополнительными функциями, включая дистанционное управление, локальное управление, OTA-обновление³² и планирование. Кроме того, мы разрешим сторонним приложениям Alexa и Google Home управлять умным светом с помощью Skill и дополним голосовым управлением с помощью голосовых помощников, таких как Alexa и Google.Assistant.

Стандартный голосовой помощник позволяет включать и выключать умный свет, а также настраивать его яркость. Если умный свет поддерживает регулировку цвета и цветовой температуры, вы можете отправлять заданные голосовые команды для их изменения. Кроме того, вы можете использовать для обнаружения и управления светом динамик со встроенным голосовым помощником, такой как Echo и Nest.

9.4.1. Основы ESP RainMaker

Прежде чем углубиться в функции ESP RainMaker, в этом разделе сначала объясняются некоторые фундаментальные концепции, которые будут упомянуты в описании фреймворка ESP RainMaker (бэкенд и фронтенд³³). Платформа ESP RainMaker показана на рис. 9.6.



Рисунок 9.6. Фреймворк ESP RainMaker

Узел (node)

Это наименование относится к модели, представляющей физическое устройство (например, ESP32-C3) в облаке. Каждый узел имеет уникальный идентификатор, а именно node ID. Это самый маленький операционный блок во фреймворке ESP RainMaker.

³² См. сноску на стр. 42.

³³ Backend – функциональность на стороне сервера; frontend – функциональность на стороне клиента/пользователя. В настоящее время в отечественной программистской среде эти термины часто употребляют без перевода.

Атрибут узла (node attribute)

Атрибут узла используется для лучшего описания и определения функций узлов. ESP RainMaker устанавливает для узла метаданные по умолчанию, включая `fw_version` и `model`. Имя и тип, которые задаются при создании узла, также относятся к метаданным по умолчанию. Вы можете добавить вашу собственную информацию в метаданные, чтобы лучше описать узел.

Устройство (device)

Это логический объект, которым пользователь может управлять, такой как выключатель, умное освещение, датчик температуры или вентилятор. В отличие от узла, устройство – это самый маленький блок, которым можно управлять на уровне пользователя.

Атрибут устройства (device attribute)

Аналогично атрибуту узла, атрибут устройства используется для лучшего описания и определения функций устройств.

Сервис

В фреймворке ESP RainMaker сервис – это объект, очень похожий на устройство. Основное отличие заключается в том, что сервис не требует операций от пользователя. Например, служба OTA-обновлений имеет некоторые состояния, не требующие каких-либо операций и управления со стороны пользователя.

Параметр

Параметр используется для реализации функций устройств и сервисов, таких как состояние питания, яркость и цвет умного светильника, а также обновление статуса во время OTA-обновлений.

Концепции узлов, устройств, параметров и служб в фреймворке ESP RainMaker позволяют точно описать состояние и функции продукта. Например, для создания интеллектуального источника света с регулируемым состоянием питания, яркостью, цветом и переключением по расписанию источник света представлен узлом и устройством, состояние его питания, яркость и цвет управляются параметрами, а функция планирования выполняется с помощью сервиса.

9.4.2. Протокол связи между узлом и серверной частью облака

Связь между узлом и облачным бэкендом шифруется с использованием протокола TLS поверх MQTT, и их идентификационные данные взаимно аутентифицируются с применением сертификатов X.509. Закрытый ключ, используемый для подключения к узлу, автоматически генерируется на узле.

Во время первой настройки Wi-Fi ESP32-C3 получает сертификат через службу Assisted Claiming³⁴ и сохраняет его во флеш-памяти. Процесс использования ESP32-C3 Assisted Claiming показан на рис. 9.7.

³⁴ Assisted Claiming – букв. «оказывающий помощь в подаче запросов».

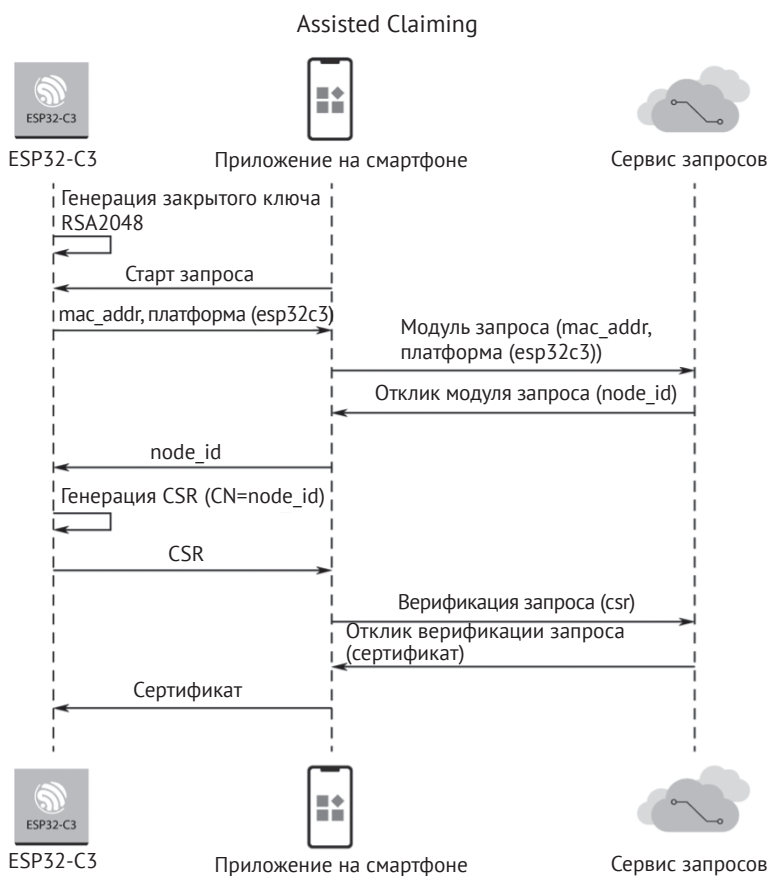


Рисунок 9.7. Служба ESP32-C3 Assisted Claiming

- (1) ESP32-C3 генерирует закрытый ключ RSA2048, используя MAC-адрес как начальный идентификатор узла (`node_id`), а затем отправляет соответствующее сообщение в приложение на смартфоне.
- (2) Во время первой подготовки приложение и служба запроса аутентифицируют личность друг друга. Как только аутентификация проходит успешно, принимающий сервер выдает идентификатор узла, который затем пересылается приложением на ESP32-C3.
- (3) ESP32-C3 генерирует CSR с полем CN, заданным в виде идентификатора узла. Затем приложение пересылает CSR в службу запросов.
- (4) Служба запросов проверяет CSR и выдает сертификат, который потом пересылается приложением на ESP32-C3.

Идентификатор узла служит не только средством идентификации узла для применения сертификата, но и способом привязки к пользователю и фильтрации сообщений MQTT. Например, узел может подписываться только на темы с определенным префиксом (`node/<node_id>/*`) и публиковать сообщения в этих темах.

ESP RainMaker определяет некоторые сообщения по умолчанию, включая сообщения конфигурации, управляющие сообщения, сообщения о состоянии, сообщения о начальном состоянии, сообщения о сопоставлении, сообщения об OTA-обновлении и предупреждающие сообщения. Эти сообщения упаковываются в JSON³⁵ и отправляются в облачный сервер через MQTT.

Сообщение о конфигурации публикуется узлами через `node/<node_id>/config`. Оно содержит информацию о самом узле, его атрибутах, устройствах, атрибутах устройств, службах и параметрах. Пример:

```
1. //Configuration message for led_light
2. {
3.     "node_id": "xxxxxxxxxx", //Node ID
4.     "config_version": "2020-03-20", //Configuration version
5.     "info": { //Node information
6.         "name": "ESP RainMaker Device",
7.         "fw_version": "1.0",
8.         "type": "Lightbulb",
9.         "model": "led_light"
10.    },
11.    "devices": [ //Devices of this node
12.        {
13.            "name": "Light",
14.            "type": "esp.device.lightbulb",
15.            "primary": "Power",
16.            "params": [ //Device parameters
17.                {
18.                    "name": "Name",
19.                    "type": "esp.param.name",
20.                    "data_type": "string",
21.                    "properties": ["read", "write"]
22.                },
23.                {
24.                    "name": "Power",
25.                    "type": "esp.param.power",
26.                    "data_type": "bool",
27.                    "properties": ["read", "write"],
28.                    "ui_type": "esp.ui.toggle"
29.                },
30.                .....
31.            ]
32.        }
33.    ],
34.    "services": [ //Node services
35.        {
36.            "name": "OTA",
37.            "type": "esp.service.ota",
38.            "params": [
39.                {
40.                    "name": "Status",
41.                    "type": "esp.param.ota_status",
42.                    "data_type": "string",
43.                    "properties": ["read"]
44.                }
45.            ]
46.        }
47.    ]
48. }
```

³⁵ JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript.


```
45.         .....
46.         ]
47.     }
48. ]
49. }
```

Приложение на смартфоне может получить уникальный идентификатор продукта, проанализировав раздел `node_id`, информацию и номер устройства, проанализировав раздел `devices`, сервисы, проанализировав раздел `services`, а также разрешения на чтение и запись приложения, проанализировав раздел `properties`. Если задан `ui_type` в секции `params` раздела `devices`, приложение отобразит соответствующий пользовательский интерфейс. Для получения дополнительной информации об использовании стандартных параметров, стандартных устройств и стандартного пользовательского интерфейса обратитесь к разделу 9.4.7.

Управляющее сообщение по нисходящей линии связи используется приложением и сторонними приложениями для управления узлами. Оно содержит параметры устройства, которые необходимо обновить. Чтобы получать такие сообщения, узлу необходимо подписаться на тему `node/<node_id>/remote`. Пример управляющих сообщений:

```
1. {
2.     "Light": {
3.         "Power": false
4.     }
5. }
```

Узел может активно сообщать о своем состоянии в теме `node/<node_id>/params/local`. Облачный сервер кеширует параметры в сообщении и отправляет их клиентам, которые включили функцию приема.

Сопоставляющее сообщение (`mapping message`) используется для сопоставления узлов и пользователей. Узел сначала должен быть сопоставлен с пользователем, чтобы гарантировать, что только этот пользователь имеет к нему доступ. Запрос на сопоставление выполняется на этапе настройки Wi-Fi. Во время настройки Wi-Fi устройство получает идентификатор (ИД) пользователя и ключ безопасности со смартфона. Как только узел подключится к облачному серверу, он объединит идентификатор пользователя и ключ безопасности со своим собственным идентификатором узла и отправит их в облачный сервер. Пример сопоставляющего сообщения:

```
1. {
2.     "node_id": "112233AABBCS",
3.     "user_id": "02e95749-8d9d-4b8e-972c-43325ad27c63",
4.     "secret_key": "9140ef1d-72be-48d5-a6a1-455a27d77dee"
5. }
```

После получения этого сообщения облачный сервер проверит, получил ли он соответствующий ключ безопасности от приложения. Если да, он сопоставит пользователя с устройством. Процесс сопоставления показан на рис. 9.8.

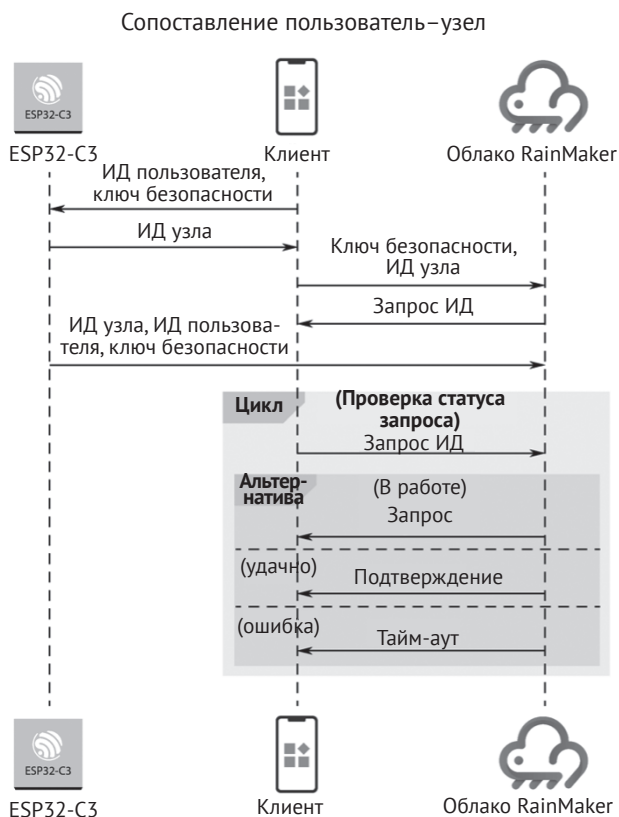


Рисунок 9.8. Процесс сопоставления

Сообщение об OTA-обновлении встроенного ПО используется для реализации обновлений узлов. Оно использует три MQTT-темы: `node/<node_id>/otafetch`, `node/<node_id>/status` и `node/<node_id>/otaurl`. В этих темах распространяется OTA-обновление, сообщается о состоянии OTA-обновления и запрашиваются задачи обновления OTA. Код выглядит следующим образом:

```

1. //Distribute OTA upgrade firmware
2. {
3.     "url": "<ota_image_url>",
4.     "ota_job_id": "<ota_job_id>",
5.     "file_size": "<num_bytes>"
6. }
7.
8. //Query OTA upgrade tasks
9. {
10.    "node_id": "<node_id>",
11.    "fw_version": "<fw_version>"
12. }
13.
14. //Report OTA upgrade status
  
```

```
15. {
16.     "ota_job_id": "<ota_job_id>",
17.     "status": "<in-progress/success/fail>",
18.     "additional_info": "<additional_info>"
19. }
```

Предупреждающее сообщение (warning message) – это тип всплывающих сообщений, используемых для уведомления и напоминания пользователям. Узел может публиковать предупреждающие сообщения через тему `node/<node_id>/alert`. После получения предупреждающего сообщения приложение отправляет его на панель уведомлений смартфона. Все сообщения с данными в облачном бэкенде имеют свойства всплывающих, и использование этого раздела явно помечает данные как уведомление, которое необходимо активно отправлять на панель уведомлений смартфона. Пример предупреждающих сообщений:

```
1.
2. "esp.alert.str": "alert"
3.
```

9.4.3. Взаимодействие между клиентом и облачным бэкендом

ESP RainMaker предлагает два клиентских инструмента: `app` и `CLI`, причем оба реализованы с использованием RESTful API. В этом разделе кратко объясняется, как применять инструмент `CLI`, входящий в комплект поставки `device SDK`, для взаимодействия с облачной серверной частью.

Инструмент `CLI` представляет собой подмодуль на основе Python-репозитория `esp-rainmaker` в каталоге `esprainmaker/cli`. Чтобы использовать его, пожалуйста, обратитесь к главе 4 о настройке среды ESP-IDF и экспорте переменных среды ESP-IDF. Вы можете проверить установку ESP-IDF и готовность среды Python, выполнив следующие команды:

```
# Print ESP-IDF version
$ idf.py --version
ESP-IDF v4.3.2
# Print Python version
$ python3 --version
Python 3.6.9
```

Ответ оболочки, аналогичный приведенному, указывает на то, что среда ESP-IDF готова. Примечание: инструмент `CLI` зависит от Python 3.x, а более старые версии требуют обновления.

После того как среда ESP-IDF будет готова, используйте команду `pip` для установки зависимостей Python и инструмента `CLI` следующим образом:

```
$ cd your RainMaker path/esp-rainmaker/cli
$ pip install -r requirements.txt
Collecting argparse
Using cached argparse-1.4.0-py2.py3-none-any.whl (23 kB)
...
...
...
```

```

Installing collected packages: cryptography, argparse
Attempting uninstall: cryptography
Found existing installation: cryptography 2.9.2
Uninstalling cryptography-2.9.2:
Successfully uninstalled cryptography-2.9.2
Successfully installed argparse-1.4.0 cryptography-2.4.2
WARNING: You are using pip version 21.1.2; however, version 21.3.1 is available.

```

Как только среда настроена, вы можете использовать инструмент CLI для взаимодействия с облачным бэкендом. Все команды, поддерживаемые CLI, перечислены в табл. 9.4, и вы можете просмотреть использование каждой команды, запустив `python3 rainmaker.py -h`. Кроме того, вы можете использовать параметр `-h` вместе с каждой командой для просмотра дополнительной справочной информации.

Таблица 9.4. Команды CLI

Команда	Описание
<code>signup</code>	Регистрация в ESP RainMaker
<code>login</code>	Вход в ESP RainMaker
<code>logout</code>	Выход текущего (подключенного) пользователя
<code>forgotpassword</code>	Сброс пароля
<code>getnodes</code>	Список всех узлов, связанных с пользователем
<code>getnodeconfig</code>	Получить конфигурацию узла
<code>getnodestatus</code>	Получить статус онлайн/офлайн узла
<code>setparams</code>	Установить параметры узла
<code>getparams</code>	Получить последние параметры узла в облаке
<code>removenode</code>	Удалить сопоставление узла с пользователем
<code>provision</code>	Предоставление узла для подключения к сети Wi-Fi
<code>getmqttthost</code>	Получить адрес хоста MQTT, к которому подключается узел
<code>claim</code>	Выполнить запрос, управляемый хостом, и получить сертификат MQTT
<code>test</code>	Проверка, был ли узел сопоставлен пользователю
<code>otaupgrade</code>	Доставка информации об OTA-обновлении
<code>getuserinfo</code>	Получить подробную информацию о подключенном пользователе
<code>sharing</code>	Поделиться узлом

Команда `claim` в инструменте CLI предназначена для управляемого хостом запроса, который больше не поддерживается в ESP32-C3. Вместо этого в ESP32-C3 поддерживается более удобное самоподтверждение.

Прежде чем использовать какую-либо другую команду, вам необходимо сначала запустить команду `signup`, чтобы зарегистрироваться в учетной записи ESP RainMaker:

```
$ cd your RainMaker path/esp-rainmaker/cli
$ cd python3 rainmaker.py signup someone@example.com
Choose a password
Password :
Confirm Password :
Enter verification code sent on your Email.
Verification Code : 973854
Signup Successful
Please login to continue with ESP Rainmaker CLI
```

Проверьте код подтверждения в присланном электронном письме, как показано на рис. 9.9.

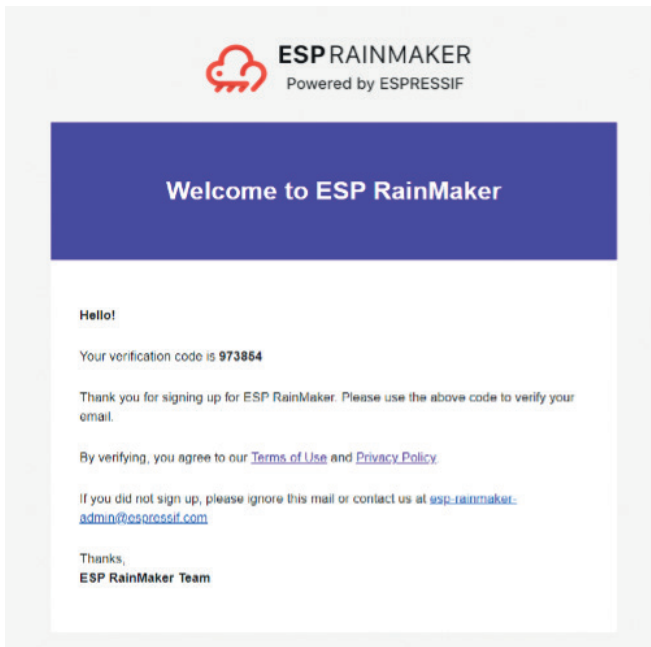


Рисунок 9.9. Код подтверждения в электронном письме

Затем войдите в систему.

Выполните команду входа в систему, и командная консоль откроет веб-страницу, показанную на рис. 9.10.

Введите свою учетную запись и пароль в соответствующие поля.

Другим способом вы можете ввести команду `login` с параметром `-email` для входа в систему напрямую с помощью CLI:

```
$ python3 rainmaker.py login --email someone@example.com
Password:
Login Successful
```

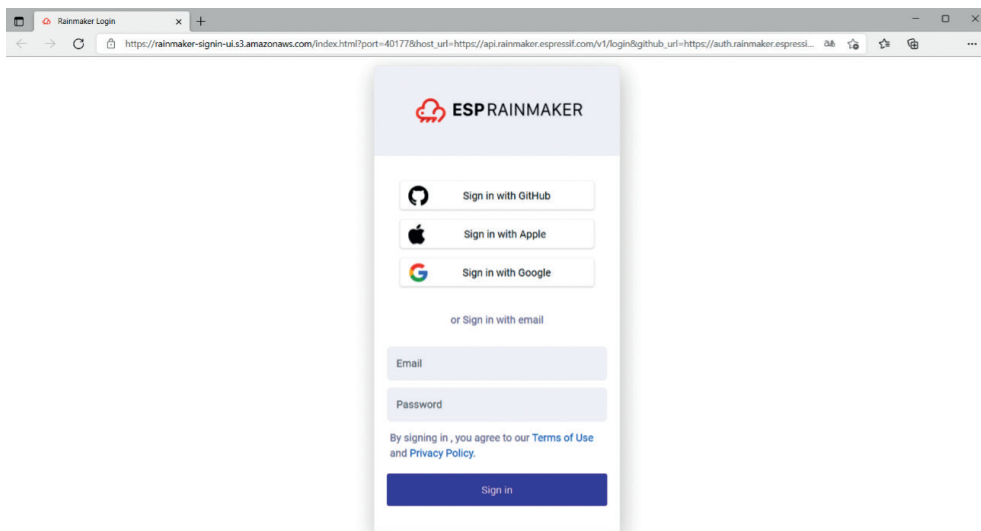


Рисунок 9.10. Вход в ESP RainMaker с помощью веб-браузера

9.4.4. Типы пользователей

Как упоминалось ранее, сопоставление пользовательских узлов направлено на то, чтобы гарантировать, что каждый узел управляется уникальным пользователем. В ESP RainMaker есть два типа пользователей: пользователи-администраторы и конечные пользователи.

Пользователь-администратор:

пользователь, который владеет сертификатом MQTT данного узла или получает сертификат через службу запроса. Пользователи с правами администратора могут получать доступ к узлам через панель управления ESP RainMaker, посылать обновления ПО через OTA-обновления и использовать ESP Insight для удаленного мониторинга, но не могут считывать или записывать параметры узла.

Конечный пользователь:

пользователь, у которого есть права управления данным узлом. Конечные пользователи могут устанавливать и получать параметры и конфигурацию узлов, но не могут просматривать узлы через панель мониторинга ESP RainMaker.

Они подразделяются на первичных пользователей и вторичных пользователей.

Первичный пользователь:

пользователь, с которым последним выполнялось сопоставление пользовательского узла. Первичные пользователи могут получать доступ к конфигурации узла, читать/записывать параметры узла и добавлять/удалять/просматривать вторичных пользователей.

Вторичный пользователь:

любой пользователь, который получает доступ к узлу в рамках совместного использования. Вторичные пользователи могут получить доступ к конфигурации узла и к чтению/записи параметров узла, но не могут добавлять/удалять/просматривать других вторичных пользователей.

9.4.5. Основные сервисы

Сервисы ESP RainMaker являются практическими примерами, которые интегрируют конкретные функции для облегчения вторичной разработки и обогащения функциональности узлов. Например, служба планирования предоставляет устройствам автономную функцию синхронизации / обратного отсчета; системная служба предлагает функции удаленной перезагрузки и сброса настроек к заводским; служба времени и часовых поясов включает функцию переключения часовых поясов; служба обновления OTA предоставляет функцию удаленного обновления; а служба локального управления обеспечивает быструю стабильную работу и защищенную связь по локальной сети. Эти службы могут быть быстро внедрены с помощью простой настройки.

1. Служба времени и часовых поясов

Время выборки является одной из наиболее важных задач для устройства интернета вещей после подключения к интернету, особенно когда включена служба планирования. В ESP RainMaker есть два важных понятия: время и часовой пояс.

Время обычно извлекается с помощью простого сетевого протокола времени (SNTP). ESP RainMaker SDK предоставляет доступ к компоненту SNTP в ESP-IDF, что упрощает синхронизацию и проверку времени. Код выглядит следующим образом:

```
1. /*Initialise time synchronisation. This will call the SNTP component
internally and
2. set the SNTP server through sntp_server_name passed by esp_rmaker_time_
config_t*/
3. esp_err_t esp_rmaker_time_sync_init(esp_rmaker_time_config_t *config);
4.
5. //Check if time has been synchronised by comparison with the standard time
1546300800
6. bool esp_rmaker_time_check(void);
7.
8. //Wait for time to be synchronised
9. esp_err_t esp_rmaker_time_wait_for_sync(uint32_t ticks_to_wait);
```

Поскольку страны и регионы, расположенные на разной долготе, имеют разное местное время и часовые пояса, переменная среды TZ и функция tz_set() предоставляются ESP-IDF для установки часового пояса. RainMaker обеспечивает несколько способов его установки. Например:

(1) Используем C API.

```
1. //Set time zone using the timezone region string
2. esp_err_t esp_rmaker_time_set_timezone(const char *tz);
3.
```

```
4. //Set time zone using the POSIX format
5. esp_err_t esp_rmaker_time_set_timezone_posix(const char *tz_posix);
```

- (2) Изменение часового пояса по умолчанию Default Timezone в меню конфигурации menuconfig. Чтобы использовать этот метод, вам необходимо иметь некоторое базовое понимание ESP-IDF. Дополнительную информацию см. в главе 4. Конфигурация для этого метода следующая:

```
(Top) → Component config → ESP RainMaker Common
Espressif IoT Development Framework Configuration
...
(Asia/Shanghai) Default Timezone
...
```

- (3) Установка часового пояса непосредственно на стороне клиента через службу часового пояса. Чтобы включить эту службу, вызовите на стороне устройства следующую функцию:

```
1. esp_err_t esp_rmaker_timezone_service_enable(void);
```

2. Служба планирования

Служба планирования выполняет периодические изменения параметров устройства. Например, если вам нужно включать свет в 19:00 и выключать его в 23:00 каждый день, эта услуга может избавить вас от ручного включения и выключения. После настройки эта служба запускается независимо на устройстве и не зависит от сети, а это означает, что устройство может выполнять запланированные действия корректно даже при отключении устройства от сети.

Чтобы включить эту услугу, вызовите на стороне устройства следующую функцию:

```
1. esp_err_t esp_rmaker_schedule_enable(void);
```

3. Услуга OTA-обновления

ESP RainMaker предоставляет услугу OTA-обновления для обновления прошивки. Вам нужно только вызвать простой API для его включения. Существует два метода выполнения OTA-обновления:

- (1) **OTA-обновление с использованием параметров.** Это самый простой способ обновления OTA-прошивки для разработчиков. Вам нужно только загрузить прошивку на любой защищенный веб-сервер и предоставить URL-адрес узла. Этот метод можно запустить из клиента CLI ESP RainMaker. Команда `otaupgrade` в инструменте CLI используется для проведения обновления, как показано в следующем коде:

```
1. esp_rmaker_ota_config_t ota_config = {
2.     .server_cert = ESP_RMAKER_OTA_DEFAULT_SERVER_CERT,
3. };
4. esp_rmaker_ota_enable(&ota_config, OTA_USING_PARAMS);
```

- (2) **OTA-обновление с использованием тем MQTT.** Это более продвинутый метод, доступный пользователю-администратору, который также может обновить прошивку. Необходимо загрузить прошивку в рабочую панель

и создать там задачу для включения OTA-обновления. Устройство получит URL для OTA-обновления и сообщит о ходе обновления через MQTT. Код выглядит следующим образом:

```
1. esp_rmaker_ota_config_t ota_config = {
2.   .server_cert = ESP_RMAKER_OTA_DEFAULT_SERVER_CERT,
3. };
4. esp_rmaker_ota_enable(&ota_config, OTA_USING_TOPICS);
```

4. Местная служба управления

Помимо удаленного управления, ESP RainMaker также позволяет клиенту локально управлять узлами, находящимися в той же сети Wi-Fi, что и клиент. Процесс управления и реакция устройств оказываются быстрее и надежнее. ESP-IDF предоставляет компонент под названием ESP Local Control, который использует обнаружение на основе mDNS и управление на основе HTTP. Теперь он также интегрирован в ESP RainMaker SDK.

Локальное управление не требует добавления службы в сообщение о конфигурации узла. Служба локального управления защищает данные с помощью алгоритмов асимметричного шифрования и передает подтверждение владения (proof of possession, PoP) в приложение. Приложение для смартфона выполняет шифрование с помощью PoP.

```
# Enable local control
CONFIG_ESP_RMAKER_LOCAL_CTRL_ENABLE=y

# Enable local control encryption
CONFIG_ESP_RMAKER_LOCAL_CTRL_SECURITY_1=y
```

5. Системный сервис

ESP RainMaker предварительно настраивает набор системных служб для сброса настроек и удаленной перезагрузки. Приложения на смартфоне могут использовать эти службы для удаления информации на устройстве о конфигурации сети и отмены сопоставления пользователей с устройствами. Чтобы включить эту службу, вызовите следующий API на стороне устройства:

```
1. esp_err_t esp_rmaker_system_service_enable(esp_rmaker_system_serv_config_t *config)
```

9.4.6. Пример Smart Light

RainMaker SDK построен на базе ESP-IDF и предоставляет простые API для создания приложений на основе спецификации ESP RainMaker. В этом разделе будет объяснен и запущен пример Smart Light. Код выглядит следующим образом:

```
1. esp_rmaker_device_t *light_device;
2. //Callback function to handle commands received from ESP RainMaker
3. static esp_err_t write_cb(const esp_rmaker_device_t *device,
4.                           constesp_rmaker_param_t *param,
5.                           const esp_rmaker_param_val_t val,
6.                           void *priv_data,
7.                           esp_rmaker_write_ctx_t *ctx)
```

```

8. {
9.     if (ctx) {
10.         ESP_LOGI(TAG,
11.             "Received write request via : %s",
12.             esp_rmaker_device_cb_src_to_str(ctx->src));
13.     }
14.     const char *device_name = esp_rmaker_device_get_name(device);
15.     const char *param_name = esp_rmaker_param_get_name(param);
16.     if (strcmp(param_name, ESP_RMAKER_DEF_POWER_NAME) == 0) {
17.         ESP_LOGI(TAG,
18.             "Received value = %s for %s - %s",
19.             val.val.b? "true" : "false",
20.             device_name,
21.             param_name);
22.         app_light_set_power(val.val.b);
23.     } else if (strcmp(param_name, ESP_RMAKER_DEF_BRIGHTNESS_NAME) == 0) {
24.         ESP_LOGI(TAG,
25.             "Received value = %d for %s - %s",
26.             val.val.i,
27.             device_name,
28.             param_name);
29.         app_light_set_brightness(val.val.i);
30.     } else if (strcmp(param_name, ESP_RMAKER_DEF_HUE_NAME) == 0) {
31.         ESP_LOGI(TAG,
32.             "Received value = %d for %s - %s",
33.             val.val.i,
34.             device_name,
35.             param_name);
36.         app_light_set_hue(val.val.i);
37.     } else if (strcmp(param_name, ESP_RMAKER_DEF_SATURATION_NAME) == 0) {
38.         ESP_LOGI(TAG,
39.             "Received value = %d for %s - %s",
40.             val.val.i,
41.             device_name,
42.             param_name);
43.         app_light_set_saturation(val.val.i);
44.     } else {
45.         //Omit parameters that do not need processing
46.         return ESP_OK;
47.     }
48.     esp_rmaker_param_update_and_report(param, val);
49.     return ESP_OK;
50. }
51.
52. void app_main()
53. {
54.     //Initialise the driver layer
55.     app_driver_init();
56.
57.     //Initialise the NVS partition
58.     esp_err_t err = nvs_flash_init();
59.     if (err == ESP_ERR_NVS_NO_FREE_PAGES || err ==
60.         ESP_ERR_NVS_NEW_VERSION_FOUND) {
61.         ESP_ERROR_CHECK(nvs_flash_erase());
62.         err = nvs_flash_init();
63.     }
64.     ESP_ERROR_CHECK( err );

```

```

65.
66. //Initialise Wi-Fi
67. app_wifi_init();
68.
69. //Initialise ESP RainMaker Agent
70. esp_rmaker_config_t rainmaker_cfg = {
71.     .enable_time_sync = false,
72. };
73. esp_rmaker_node_t *node = esp_rmaker_node_init(&rainmaker_cfg,
74.     "ESP RainMakerDevice",
75.     "Lightbulb");
76. if (!node) {
77.     ESP_LOGE(TAG, "Could not initialise node. Aborting!!!");
78.     vTaskDelay(5000/portTICK_PERIOD_MS);
79.     abort();
80. }
81.
82. //Create the device and add parameters
83. light_device = esp_rmaker_lightbulb_device_create("Light",
84.     NULL,
85.     DEFAULT_POWER);
86. esp_rmaker_device_add_cb(light_device, write_cb, NULL);
87. esp_rmaker_device_add_param(light_device,
88.     esp_rmaker_brightness_param_create(
89.     ESP_RMAKER_DEF_BRIGHTNESS_NAME,
90.     DEFAULT_BRIGHTNESS));
91. esp_rmaker_device_add_param(light_device, esp_rmaker_hue_param_creat(
92.     ESP_RMAKER_DEF_HUE_NAME,
93.     DEFAULT_HUE));
94. esp_rmaker_device_add_param(light_device,
95.     esp_rmaker_saturation_param_create(
96.     ESP_RMAKER_DEF_SATURATION_NAME,
97.     DEFAULT_SATURATION));
98. esp_rmaker_node_add_device(node, light_device);
99.
100. //Enable OTA upgrade
101. esp_rmaker_ota_config_t ota_config = {
102.     .server_cert = ota_server_cert,
103. };
104. esp_rmaker_ota_enable(&ota_config, OTA_USING_PARAMS);
105.
106. //Enable time & time zone service
107. esp_rmaker_timezone_service_enable();
108.
109. //Enable scheduling service
110. esp_rmaker_schedule_enable();
111.
112. //Enable ESP Insight
113. app_insights_enable();
114.
115. //Enable ESP RainMaker Agent
116. esp_rmaker_start();
117.
118. //Enable Wi-Fi
119. err = app_wifi_start(POP_TYPE_RANDOM);
120. if (err != ESP_OK) {

```

```

121.     ESP_LOGE(TAG, "Could not start Wifi. Aborting!!!");
122.     vTaskDelay(5000/portTICK_PERIOD_MS);
123.     abort();
124. }
125.}

```

В начале приведенный пример инициализирует аппаратный драйвер с настройкой GPIO и инициализацией периферийных устройств. Далее раздел NVS инициализируется для подготовки к чтению данных из флеш-памяти. Таблица разделов `partitions.csv` выглядит следующим образом:

1. #	Name,	Type,	SubType,	Offset,	Size,	Flags
2. #	Note: Firmware partition offset needs to be 64K aligned, initial 36K (9					
3. #	sectors) are reserved for bootloader and partition table					
4.	sec_cert,	0x3F,	,	0xd000,	0x3000,	,
5.	nvs,	data,	nvs,	0x10000,	0x6000,	
6.	otadata,	data,	ota,	,	0x2000	
7.	phy_init,	data,	phy,	,	0x1000,	
8.	ota_0,	app,	ota_0,	0x20000,	1600K,	
9.	ota_1,	app,	ota_1,	,	1600K,	
10.	fctry,	data,	nvs,	0x340000,	0x6000	

Как показано в этой таблице, в примере проекта имеется два раздела NVS: `nvs` и `fctry`. Первый используется для хранения конфигурации сети и информации о локальном планировании, а последний – для хранения информации о сертификате.

Затем инициализируется Wi-Fi. Этот шаг должен быть выполнен до вызова функции `esp_maker_node_init()`. Если раздел `fctry` не содержит сертификата, будет включена служба Assisted Claiming, поскольку при инициализированном Wi-Fi в качестве начального идентификатора узла может использоваться MAC-адрес. После этого создается модель устройства и добавляется функция обратного вызова. Все данные нисходящей облачной связи передаются через эту функцию обратного вызова, и будет запущена основная задача ESP RainMaker. Наконец, Wi-Fi включен. Если устройство не имеет подключения к Wi-Fi, приложение подготовки будет запущено автоматически. Приложение реализовано с использованием компонента `wifi_provisioning` в ESP-IDF и начинается с вызова `app_wifi_start()` в ESP RainMaker SDK.

Запустите `idf.py`, чтобы скомпилировать и прошить проект `led_light`, и `idf.py monitor`, дабы открыть монитор, после чего вы увидите следующий лог:

```

I (30) boot: ESP-IDF v4.3.2-dirty 2nd stage bootloader
...
...
...
I (488) cpu_start: Starting scheduler.
I (493) gpio: GPIO[9]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1|
Pulldown: 0| Intr:3
I (503) coexist: coexist rom version 9387209
I (503) pp: pp rom version: 9387209
I (503) net80211: net80211 rom version: 9387209
I (523) wifi:wifi driver task: 3fca4d8c, prio:23, stack:6656, core=0
I (523) system_api: Base MAC address is not set

```

```
I (523) system_api: read default base MAC address from EFUSE
...
...
...
I (623) esp_rmaker_work_queue: Work Queue created.
I (623) esp_claim: Initialising Assisted Claiming. This may take time.
W (633) esp_claim: Generating the private key. This may take time.
I (110533) esp_rmaker_node: Node ID ----- 7CDFA161BE38
I (21213) esp_rmaker_node: Node ID ----- 7CDFA1C21DA0
I (21213) esp_rmaker_ota: OTA state = 2
I (21213) esp_rmaker_ota_using_params: OTA enabled with Params
I (21223) esp_rmaker_time_service: Time service enabled
I (21223) esp_rmaker_time: Initializing SNTP. Using the SNTP server: pool.ntp.org
I (21233) app_insights: Enable CONFIG_ESP_INSIGHTS_ENABLED to get Insights.
I (21243) esp_rmaker_core: Starting RainMaker Work Queue task
I (21253) esp_rmaker_work_queue: RainMaker Work Queue task started.
I (21253) esp_claim: Waiting for assisted claim to finish.
...
...
...
```



```
I (21623) app_wifi: If QR code is not visible, copy paste the below URL in a browser.
https://rainmaker.espressif.com/qrcode.html?data={"ver":"v1","name":"PROV_8a20e0",
"pop":"827e49ae","transport":"ble"}
I (21633) app_wifi: Provisioning Started. Name : PROV_8a20e0, POP : 827e49ae
```

Используйте приложение для смартфона, чтобы отсканировать QR-код. Если в таблице нет раздела сертификата, будет включена служба Assisted Claiming, как показано на рис. 9.11:

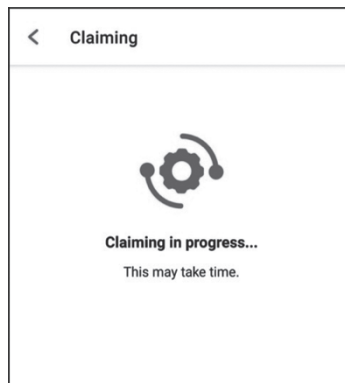


Рисунок 9.11. Интерфейс для включения Assisted Claiming

```
...
...
I (444493) esp_clain: Assisted Claiming Started.
I (447603) esp_rmaker_core: New Node ID ----- nq8xT6p53BZHтm6k8AZqN
I (472813) esp_clain: Assisted Claiming was Successful.
```

После получения сертификата устройство переходит в фазу настройки сети. Приложение для смартфона отправит выбранный SSID и пароль на устройство, которое затем попытается подключиться к роутеру и облаку, как показано на рис. 9.12.

```
...
...
I (491113) esp_rmaker_user_mapping: Received request for node details
I (491113) esp_rmaker_user_mapping: Got user_id = 764865be-e49f-49d1-afa1-696d6
a7e3233, secret_key = a3c89473-514f-4aa4-a190-a9aa38e7a9d8
I (491123) esp_rmaker_user_mapping: Sending status SUCCESS
I (491753) app_wifi: Received Wi-Fi credentials
    SSID : Xiaomi_32BD
    Password : 12345678
I (495173) wifi:new:<11,0>, old:<1,0>, ap:<255,255>, sta:<11,0>, prof:1
I (495753) wifi:state: init -> auth (b0)
I (495793) wifi:state: auth -> assoc (0)
I (495833) wifi:state: assoc -> run (10)
I (495973) wifi:connected with Xiaomi_32BD, aid = 2, channel 11, BW20, bssid =
88:c3:97:9e:32:be
I (495973) wifi:security: WPA2-PSK, phy: bgn, rssi: -25
I (495983) wifi:pm start, type: 1
I (495983) wifi:set rx beacon pti, rx_bcn_pti: 14, bcn_timeout: 14, mt_pti:
25000, mt_time: 10000
I (496043) wifi:BcnInt:102400, DTIM:1
W (496573) wifi:<ba-add>idx:0 (ifx:0, 88:c3:97:9e:32:be), tid:0, ssn:2, winSize:64
I (497503) app_wifi: Connected with IP Address:192.168.31.65
I (497503) esp_netif_handlers: sta ip: 192.168.31.65, mask: 255.255.255.0, gw:
192.168.31.1
I (497503) wifi_prov_mgr: STA Got IP
I (497503) app_wifi: Provisioning successful
I (497513) esp_mqtt_glue: Initialising MQTT
I (500973) esp_mqtt_glue: MQTT Connected
```

После завершения настройки и успешного подключения к облаку устройство отправит сообщение о сопоставлении пользователя с узлом, и приложение продолжит проверять статус сопоставления, как показано на рис. 9.13.

```
I (45959) esp_rmaker_user_mapping: User Node association message published
successfully.
```

После сопоставления вы можете проверить этот узел с помощью CLI.

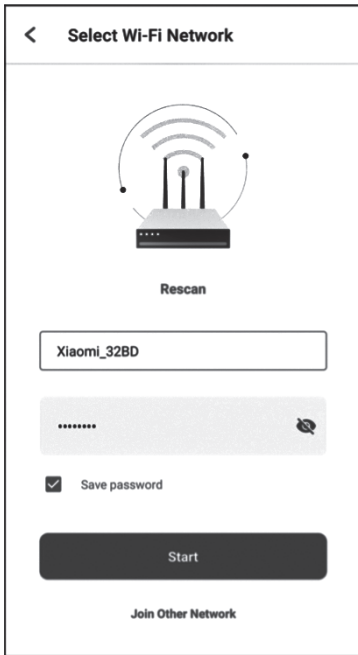


Рисунок 9.12. Подключение устройства к роутеру и облаку

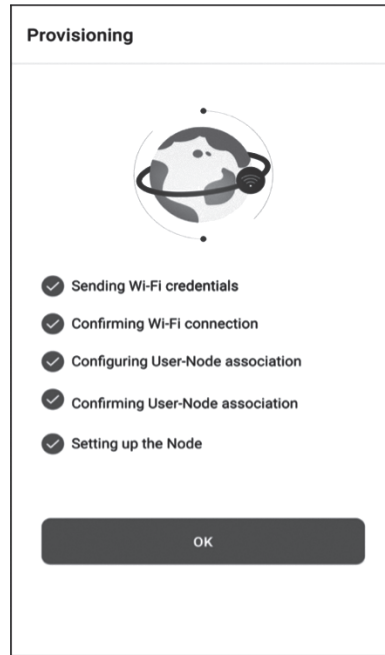



Рисунок 9.13. Проверка статуса сопоставления

9.4.7. Приложение RainMaker и интеграция сторонних платформ






В разделе 9.4.6 мы завершили подготовку устройства и сопоставление пользовательских узлов, что позволило управлять умным светом через приложение. В результате значок умного освещения и пользовательский интерфейс теперь отображаются на главной странице приложения. Стандартные параметры, устройства и упомянутые ранее пользовательские интерфейсы определяются ESP RainMaker и составляют основу его фреймворка. Используя эту структуру, приложение может точно управлять параметрами каждого устройства и поддерживаемыми услугами. Стандартные позиции, перечисленные в таблицах ниже, также применимы к сторонним платформам.


(1) Типы устройств со стандартным пользовательским интерфейсом

Таблица 9.5. Типы устройств со стандартным пользовательским интерфейсом

Название	Тип	Параметры	GVA ³⁶	Alexa	Image
Switch (выключатель)	esp.device.switch	Название (name), Питание (power)	SWITCH	SWITCH	

³⁶ GVA Lighting – канадская компания, разработчик и производитель осветительного и сопутствующего оборудования для архитектурного и коммерческого рынков.

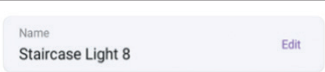




Название	Тип	Параметры	GVA ³⁶	Alexa	Image
Lightbulb (лампочка)	esp.device.lightbulb	Название (name), Питание (power) Яркость (brightness), Цветовая температура (color temperature), Насыщенность (saturation), Оттенок (hue), Интенсивность (intensity)	LIGHT	LIGHT	
Light (светильник)	esp.device.light	Название (name), Питание (power) Яркость (brightness), Цветовая температура (color temperature), Насыщенность (saturation), Оттенок (hue), Интенсивность (intensity)	LIGHT	LIGHT	–
Fan (вентилятор)	esp.device.fan	Название (name), Питание (power), Скорость (speed), Направление (direction)	FAN	FAN	
Temperature Sensor (датчик температуры)	esp.device.temperature-sensor	Название (name), Температура (temperature)	–	TEMPERATURE SENSOR	
Outlet (розетка)	esp.device.outlet	Название (name), Питание (power)	OUTLET	SMARTPLUG	
Plug (вилка)	esp.device.plug	Название (name), Питание (power)	OUTLET	SMARTPLUG	–
Socket (патрон)	esp.device.socket	Название (name), Питание (power)	OUTLET	SMARTPLUG	–
Lock (замок)	esp.device.lock	Питание (power), Состояние (lock state)	LOCK	SMARTLOCK	
Internal Blinds (внутренние жалюзи)	esp.device.blinds-internal	Название (name)	BLINDS	INTERIOR_BLIND	–
External Blinds (внешние жалюзи)	esp.device.blinds-external	Название (name)	BLINDS	EXTERIOR_BLIND	–
Garage Door (гаражная дверь)	esp.device.garage-door	Название (name)	GARAGE	GARAGE DOOR	–
Garage Lock (гаражный замок)	esp.device.garage-door-lock	Название (name)	GARAGE	SMARTLOCK	–





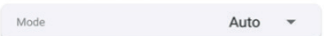
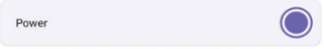
Название	Тип	Параметры	GVA ³⁶	Alexa	Image
Speaker (громкоговоритель)	esp.device.speaker	Название (name)	SPEAKER	SPEAKER	–
Air Conditioner (кондиционер)	esp.device.air-conditioner	Название (name)	AC_UNIT	AIR_CONDITIONER	–
Thermostat (термостат)	esp.device.thermostat	Название (name)	THERMOSTAT	THERMOSTAT	–
TV (телевизор)	esp.device.tv	Название (name)	TV	TV	–
Washer (стиральная машина)	esp.device.washer	Название (name)	WASHER	WASHER	–
Другое	esp.device.other	–	–	OTHER	

(2) Стандартные элементы пользовательского интерфейса

Стандартный элемент пользовательского интерфейса, добавленный к параметру, в приложении ESP RainMaker отображается как соответствующий показанному в табл. 9.6.

Таблица 9.6. Стандартные элементы пользовательского интерфейса

Название	Тип	Тип данных	Требования	Image
Text (текст, по умолчанию)	esp.ui.text	Любой	N/A	
Toggle Switch (переключатель)	esp.ui.toggle	bool		
Slider (ползунок)	esp.ui.slider	int, float	Границы (мин, макс)	
Brightness Slider (ползунок яркости)	esp.ui.slider	int	Тип параметра = esp.param.brightness	
CCT Slider (ползунок цветовой температуры)	esp.ui.slider	int	Тип параметра = esp.param.cct	

Название	Тип	Тип данных	Требования	Image
Saturation Slider (ползунок насыщенности)	esp.ui.slider	int	Тип параметра = esp.param.saturation	
Hue Slider (ползунок цветового тона)	esp.ui.hue-slider	int	Тип параметра = esp.param.hue	
Hue Circle (цветовой круг)	esp.ui.hue-slider	int	Тип параметра = esp.param.hue	
Push button (кнопка, большая)	esp.ui.push-btn-big	bool	N/A	
Dropdown (выпадающий список)	esp.ui.dropdown	int/string	Bounds (границы, min, max) для int Корректные строки для string	
Trigger (кнопка с фиксацией, только Android)	esp.ui.trigger	bool	N/A	
Скрытый (только Android)	esp.ui.hidden	bool	N/A	Параметр будет скрыт

(3) Стандартные типы параметров

Они сопоставляются с параметрами соответствующих имен и пользовательских интерфейсов в Alexa и приложениях Google Home.

Таблица 9.7. Стандартные типы параметров

Название	Тип	Тип данных	Интерфейс устройства	Свойства	min, max, шаг (step)
Power (питание)	esp.param.power	bool	esp.ui.toggle	Чтение/запись	N/A
Brightness (яркость)	esp.param.brightness	int	esp.ui.slider	Чтение/запись	0, 100, 1
ССТ (цветовая температура)	esp.param.cct	int	esp.ui.slider	Чтение/запись	2700, 6500, 100

Название	Тип	Тип данных	Интерфейс устройства	Свойства	min, max, шаг (step)
Hue (цветовой тон)	esp.param.hue	int	esp.ui.slider	Чтение/ запись	0, 360, 1
Saturation (насыщенность)	esp.param.saturation	int	esp.ui.slider	Чтение/ запись	0, 100, 1
Intensity (интенсивность)	esp.param.intensity	int	esp.ui.slider	Чтение/ запись	0, 100, 1
Speed (скорость)	esp.param.speed	int	esp.ui.slider	Чтение/ запись	0, 5, 1
Direction (направление)	esp.param.direction	int	esp.ui.dropdown	Чтение/ запись	0, 1, 1
Temperature (температура)	esp.param.temperature	float	N/A	Чтение	N/A
OTA URL (URL обновления)	esp.param.ota url	string	N/A	Запись	N/A
OTA Status (статус обновления)	esp.param.ota status	string	N/A	Чтение	N/A
OTA Info (информация обновления)	esp.param.ota info	string	N/A	Чтение	N/A
Timezone	esp.param.tz	string	N/A	Чтение/ запись	N/A
Timezone POSIX	esp.param.tz posix	string	N/A	Чтение/ запись	N/A
Schedules (расписания)	esp.param.schedules	array	N/A	Чтение, запись, сохранение	N/A
Reboot (перезагрузка)	esp.param.reboot	bool	N/A	Чтение/ запись	N/A
Factory Reset (сброс к заводским настройкам)	esp.param.factory-reset	bool	N/A	Чтение/ запись	N/A
Wi-Fi Reset	esp.param.wifi-reset	bool	N/A	Чтение/ запись	N/A
Toggle Controller (контроллер переключений)	esp.param.toggle	bool	Применим к любому типу	Чтение/ запись	N/A

Название	Тип	Тип данных	Интерфейс устройства	Свойства	min, max, шаг (step)
Range Controller (контроллер диапазона)	esp.param.range	int/float	Применим к любому типу	Чтение/запись	Зависит от приложения
Mode Controller (контроллер режима)	esp.param.mode	string	esp.ui.dropdown	Чтение/запись	N/A
Setpoint Temperature (заданная температура)	esp.param.setpoint-temperature	int/float	Применим к любому типу	Чтение/запись	N/A
Lock State (состояние запора)	esp.param.lockstate	bool	Применим к любому типу	Чтение/запись	N/A
Blinds Position (положение жалюзи)	esp.param.blinds-position	int	esp.ui.slider	Чтение/запись	0, 100, 1
Garage Position (положение дверей гаража)	esp.param.garage-position	int	esp.ui.slider	Чтение/запись	0, 100, 1
Light Mode (режим освещения)	esp.param.light-mode	int	esp.ui.dropdown/ esp.ui.hidden	Чтение/запись	0, 2, 1
					0: ошибка
					1:HSV
					2:CCT
AC Mode (вид переменного тока)	esp.param.ac-mode	string	esp.ui.dropdown	Чтение/запись	N/A

(4) Стандартные типы сервисов

Перечисленные типы используются только для быстрого создания сервисов в ESP RainMaker SDK.

Таблица 9.8. Стандартные типы сервисов

Имя	Тип	Параметры
OTA (обновление)	esp.service.ota	OTA URL, OTA Status, OTA Info
Schedule (расписание)	esp.service.schedules	Schedules
Time (время)	esp.service.time	TZ, TZ-POSIX
System (системные)	esp.service.system	Reboot, Factory-Reset, Wi-Fi-Reset

На обучающей странице Alexa (Alexa Skill) или странице совместимости служб в Google Home (Google Service Compatibility) можно выполнить синхронизацию устройств с ESP RainMaker. Привяжите свою учетную запись в ESP RainMaker. После этого вы можете использовать две разновидности приложений и голосовых команд для управления устройствами.

На рис. 9.14 показаны устройства ESP RainMaker в Alexa. Ими можно управлять с помощью голоса такими командами, как «Alexa, please turn on the light» («Алекса, пожалуйста, включи свет»).

Примечание

Узнайте больше об Alexa Skill по адресу:

<https://www.amazon.com/Espressif-Systems-ESP-RainMaker/dp/B0881W7RPV/>.

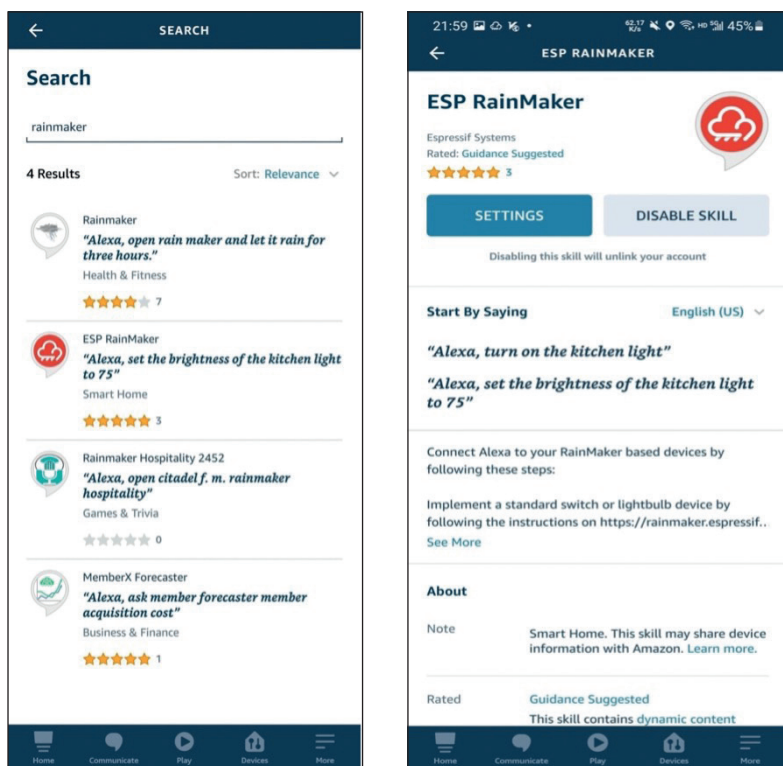


Рисунок 9.14. Устройства ESP RainMaker в Alexa

На рис. 9.15 показаны устройства ESP RainMaker в Google Home. Ими можно управлять голосовыми командами, такими как «Hey Google, please turn off the light» («Эй, Google, пожалуйста, выключи свет»).

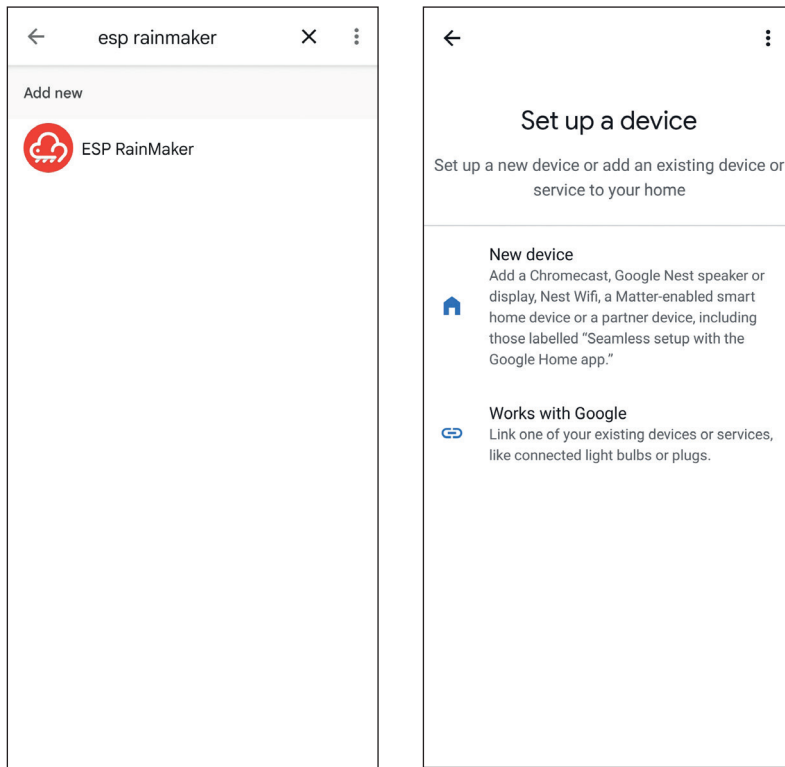


Рисунок 9.15. Устройства ESP RainMaker в Google Home

ESP RainMaker создает промежуточный уровень в облачном бэкенде. Этот уровень отображает стандартные прошитые типы параметров и типы устройств в форматах, которые Alexa Skill и Google Assistant понимают. Поэтому типы устройств в ESP RainMaker, такие как умные светильники и выключатели, сопоставляются с аналогичными типами устройств в Alexa Skill и Google Assistant и их параметры, такие как положение переключателя, яркость или цвет, сопоставляются между собой. Если вы установите только яркость, то получите умный светильник с регулируемой яркостью в Alexa и Google Home. Если вы также дополните цветом и цветовой температурой, то можете настроить и эти параметры тоже.

9.5. Резюме

В этой главе мы представили удаленное управление и протокол MQTT. Это широко используемый протокол удаленного управления и подключения IoT-устройств к облаку. Сейчас он принят всеми основными облачными платформами, такими как Amazon Cloud, Alibaba Cloud, Baidu Cloud, Tencent Cloud и ESP RainMaker, представленными в данной главе. Этот простой и легкий протокол обеспечивает надежные сетевые услуги для устройств интернета вещей в условиях низкой пропускной способности и нестабильности сетевой среды.

Кроме того, мы также рассказали, как локально создать брокер MQTT для имитации облачной платформы и как генерировать сертификаты сервера и клиента для подтверждения протокола TLS, чтобы обеспечить безопасность данных.

В практическом разделе этой главы мы рассматривали разработку умного светового продукта как пример выполнения удаленного управления устройствами с помощью платформы ESP RainMaker через MQTT. Для получения сертификатов применяется собственный запрос. Используется несколько наборов тем MQTT для управления устройством, сопоставления пользовательских узлов и состояния устройства. Встроенные базовые сервисы могут выполнить операции планирования и OTA-обновление. После завершения подключения к облаку благодаря функциональности ESP RainMaker можно быстро получить возможности голосового управления умными светильниками.

Возможности ESP RainMaker этим не ограничиваются. Сбор и анализ данных, прямая связь устройство–устройство и сторонние переключатели сценариев – это интересные функции, но они в данной главе не упомянуты. С помощью этих облачных функций мы можем примерно рассчитать энергопотребление на основе подсчета времени онлайн/офлайн и частоты включения умного освещения и выяснить, как оно работает совместно с другим оборудованием. Эти функции могут быть реализованы с помощью открытого RESTful API. В главе 10 будет представлен RESTful API и его использование в разработке приложения для смартфона.

Глава 10

Разработка приложений для смартфонов

В главе 9 мы рассказали, как управлять устройствами через облако с помощью технологий Wi-Fi, а также как общаться с облаком по протоколам MQTT и TLS с обеспечением безопасности и достоверности данных.

В этой главе мы научимся разрабатывать собственное приложение для смартфона для беспроводного управления умными светильниками. Приложение должно позволить пользователям подключать устройства умного дома к системе управления, основанной на Wi-Fi, Bluetooth и других технологиях беспроводной связи, чтобы иметь доступ, даже если устройства находятся за тысячи миль от вас. В зависимости от сети и данных вы можете получать подробную информацию об устройствах через смартфон.

В повседневной жизни мы можем просто отправлять команды на встроенный в бытовую технику модуль Wi-Fi через беспроводную сеть для осуществления соответствующего управления, например включения/выключения умного светильника и настройки его цвета или яркости.

10.1. Введение в разработку приложений для смартфонов

В разделе «Практика» главы 9 мы объяснили, как использовать клиент ESP RainMaker для беспроводного управления проектом Smart Light. В этой главе мы обратимся к приложению для смартфона и подробно представим процесс его разработки.

Проект приложения доступен как для iOS, так и для Android. Это комплексное решение, предоставляемое от Espressif для удаленного управления и мониторинга IoT-устройств на базе чипов Espressif без необходимости какой-либо конфигурации облака.

Исходный код

Исходный код приложения iOS хранится в каталоге https://github.com/espressif/book-esp32c3-iot-projects/tree/main/phone_app/app_ios. Исходный код приложения Android хранится в каталоге https://github.com/espressif/book-esp32c3-iot-projects/tree/main/phone_app/app_android.

Не волнуйтесь, если вы раньше не разрабатывали приложения для Android или iOS. В этой главе мы подробно расскажем, как разработать приложение для

смартфона, начиная с создания нового проекта приложения. После того как вы получите базовое представление о разработке приложений для Android и iOS, мы углубимся в основные особенности этого проекта с открытым исходным кодом.

10.1.1. Обзор разработки приложений для смартфонов

Приложение для смартфона для управления умным освещением включает версии для iOS и Android. Версия iOS разработана в Xcode и написана на Swift, тогда как версия Android разработана в Android Studio и запрограммирована на Java. В этой главе мы сначала составим блок-схемы прототипов в соответствии с требованиями API, затем спроектируем интерфейсы и взаимодействия и, наконец, реализуем дизайн для каждого интерфейса на основе составленных блок-схем. Мы будем использовать Git для управления кодами, получения обновленных версий, сравнения версий и фиксации изменений.

Для реализации приложения для смартфона требуется разрешение на доступ к камере смартфона, локальной сети, местоположению, Bluetooth и т. д. Реализация основана на сетевом запросе ESP Provision SDK и других сторонних фреймворках для подготовки, анализа данных, всплывающих окон и включает в себя разработку таких функций, как список устройств, планирование, центр пользователя, вход в систему и регистрацию.

В следующих разделах мы познакомим вас со структурой проекта и жизненным циклом³⁷ приложения для смартфонов на Android и iOS, чтобы вам было проще разрабатывать приложения, имея предварительное представление.

10.1.2. Структура проекта Android

В этом разделе в качестве примера представления структуры проекта для Android рассматривается приложение MyRainmaker. В корневом каталоге есть две папки под названием *app* и *Gradle Scripts*. Папка *app* содержит весь код и ресурсы для разработки приложения, а папка *Gradle Scripts* содержит скрипты, связанные с компиляцией Gradle³⁸.

Структура этого проекта для Android показана на рис. 10.1.

Папка *app*

Папка *app* содержит три подпапки: *manifests*, *java* и *res*.

- *manifests*: хранят конфигурации приложений, включая имя, версию, SDK и разрешения.
- *java*: в основном хранит исходный и тестовый коды.
- *res*: хранит все ресурсы проекта.

Папка *Gradle Scripts*

Папка *Gradle Scripts* содержит *build.gradle* (два файла с одинаковым именем), *gradle-wrapper.properties*, *proguard-rules.pro*, *gradle.properties*, *settings.gradle* и *local.properties*.

- *build.gradle*: компилирует приложение с помощью Gradle.
- *gradle-wrapper.properties*: настраивает версию Gradle.

³⁷ Жизненный цикл (lifecycle) – в терминологии Android последовательность состояний активного приложения, обусловленная действиями пользователя.

³⁸ Gradle – одна из систем сборки и компиляции проектов на языке Java.

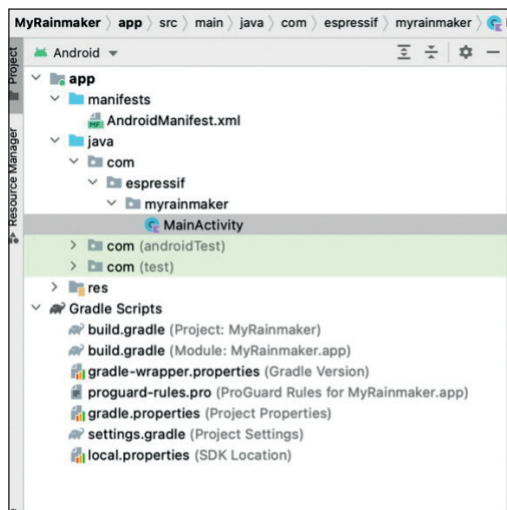


Рисунок 10.1. Структура проекта Android

- *proguard-rules.pro*: настраивает правила proguard для обфускации³⁹ кода.
- *gradle.properties*: настраивает глобальные свойства, связанные с Gradle.
- *settings.gradle*: настраивает соответствующие сценарии Gradle.
- *local.properties*: настраивает путь к SDK/NDK.

10.1.3. Структура проекта iOS

В этом разделе на примере приложения MyRainmaker представлена структура проекта приложения iOS. Как показано в представлении навигации на рис. 10.2, проект включает в себя папку исходного кода *MyRainmaker*, папку *MyRainmakerTests* для кода модульного теста и папку *MyRainmakerUITests* для тестового кода пользовательского интерфейса.

Папка *MyRainmaker*

Эта папка содержит файлы *AppDelegate*, *SceneDelegate*, *ViewController*, *Main*, *Assets*, *LaunchScreen* и *Info*.

- *AppDelegate*: входной файл всего приложения, в котором хранится класс делегата⁴⁰ приложения.
- *SceneDelegate*: в Xcode 11 добавлен новый класс, который обрабатывает ситуации, отделенные от *AppDelegate*.
- *ViewController*: класс хост-контроллера, который управляет отображением вида и обрабатывает сенсорный ввод, события и т. д.

³⁹ Обфускация (от англ. obfuscate – запутывать, сбивать с толку) в широком смысле – приведение исходного текста или исполняемого кода программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции. Обфускация также применяется для уменьшения размера программного кода.

⁴⁰ Делегат (delegate) – класс, который позволяет хранить в себе ссылку на метод с определенной сигнатурой (т. е. порядком и типами принимаемых и возвращаемого значений). Экземпляры делегатов содержат ссылки на конкретные методы конкретных классов.

- *Main*: последовательность действий основного интерфейса, содержащая обработку событий от ViewController и описывающая взаимодействие между несколькими ViewController.
- *Assets*: файл, в котором хранится большинство изображений.
- *LaunchScreen*: настраивает экран запуска приложения.
- *Info*: настройка разрешений для приложения, таких как Bluetooth, местоположение и доступ к камере.

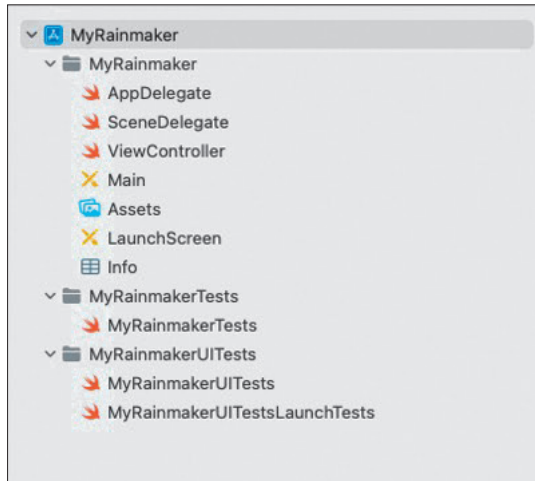


Рисунок 10.2. Структура iOS-проекта

Тестовые файлы

MyRainmakerTests, *MyRainmakerUITests* и *MyRainmakerUITestsLaunch-Tests* – это все тестовые классы. Они обычно не используются при разработке проектов и, следовательно, подробно не поясняются.

10.1.4. Жизненный цикл Android Activity

Android имеет четыре основных компонента: Activity, Service, ContentProvider и BroadcastReceiver, среди которых Activity используется очень часто и обрабатывает почти все взаимодействия с интерфейсом.

Давайте рассмотрим жизненный цикл Activity, показанный на рис. 10.3.

- `onCreate()` указывает, что действие создается. Это первый метод жизненного цикла Activity, и именно здесь вам следует выполнить инициализацию.
- `onStart()` указывает, что действие запускается и видно пользователю.
- `onRestart()` указывает, что действие перезапускается, и его следует вызывать, когда активность меняется с невидимой на видимую. Например, когда пользователи нажимают кнопку **Домой**, чтобы переключиться на рабочий стол или открыть новое действие, текущее действие будет остановлено. Когда текущая активность возвращается на передний план, будет вызван метод `onRestart()`.
- `onResume()` указывает, что действие было создано и пользователи могут взаимодействовать с интерфейсом.

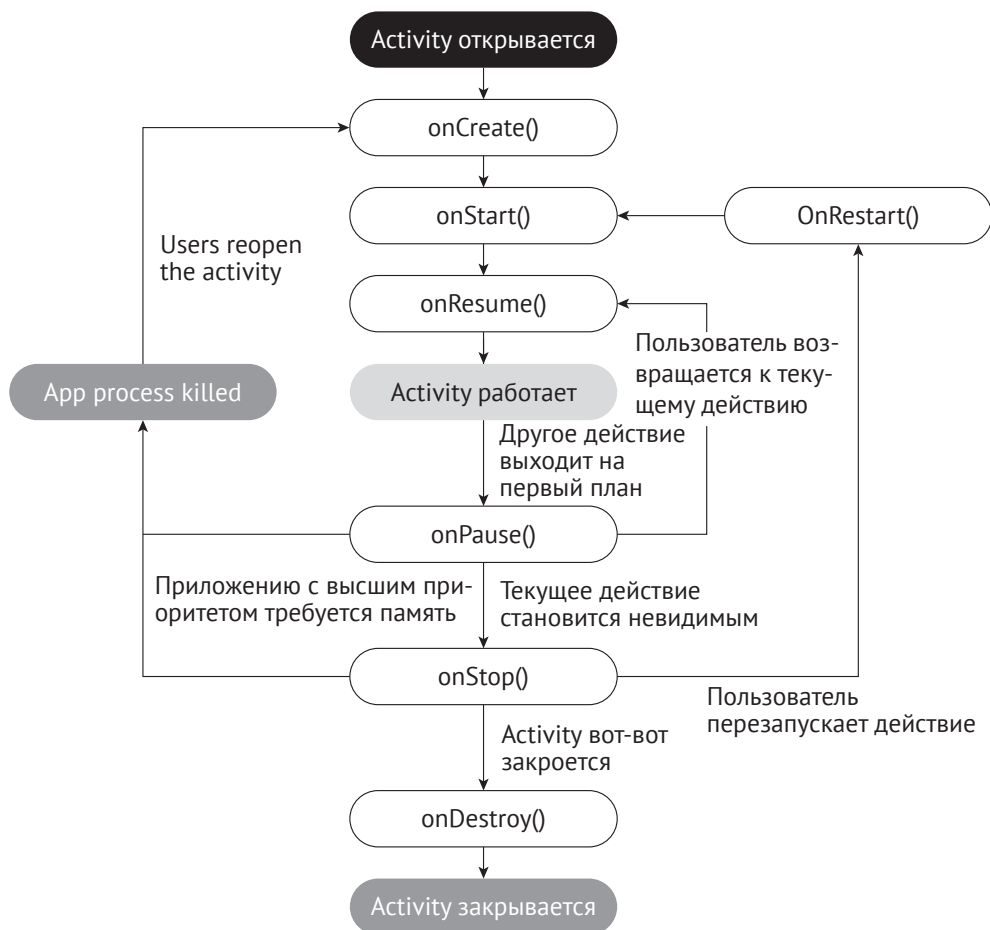


Рисунок 10.3. Жизненный цикл Activity

- `onPause()` указывает, что действие приостановлено, и обычно сразу вызывается `onStop()`. Если пользователи быстро вернутся к текущей деятельности, будет вызван `onResume()`.
- `onStop()` указывает, что действие вот-вот прекратится. Оно больше не видно пользователям и работает только в фоновом режиме.
- `onDestroy()` указывает, что действие сейчас будет закрыто. Это последний метод, выполняемый в жизненном цикле Activity, и именно здесь вы должны вернуть или освободить ресурсы.

10.1.5. Жизненный цикл iOS ViewController

Жизненный цикл ViewController относится к жизненному циклу визуализации действий, которыми он управляет. Когда состояние действия изменяется, ViewController автоматически вызывает серию методов в качестве реакции на изменения. Жизненный цикл ViewController показан на рис. 10.4.

Каждый метод используется следующим образом:

- `init()` инициализирует соответствующие критические данные;
- `loadView()` инициализирует визуализацию. Этот метод не будет вызываться напрямую, а системой автоматически;

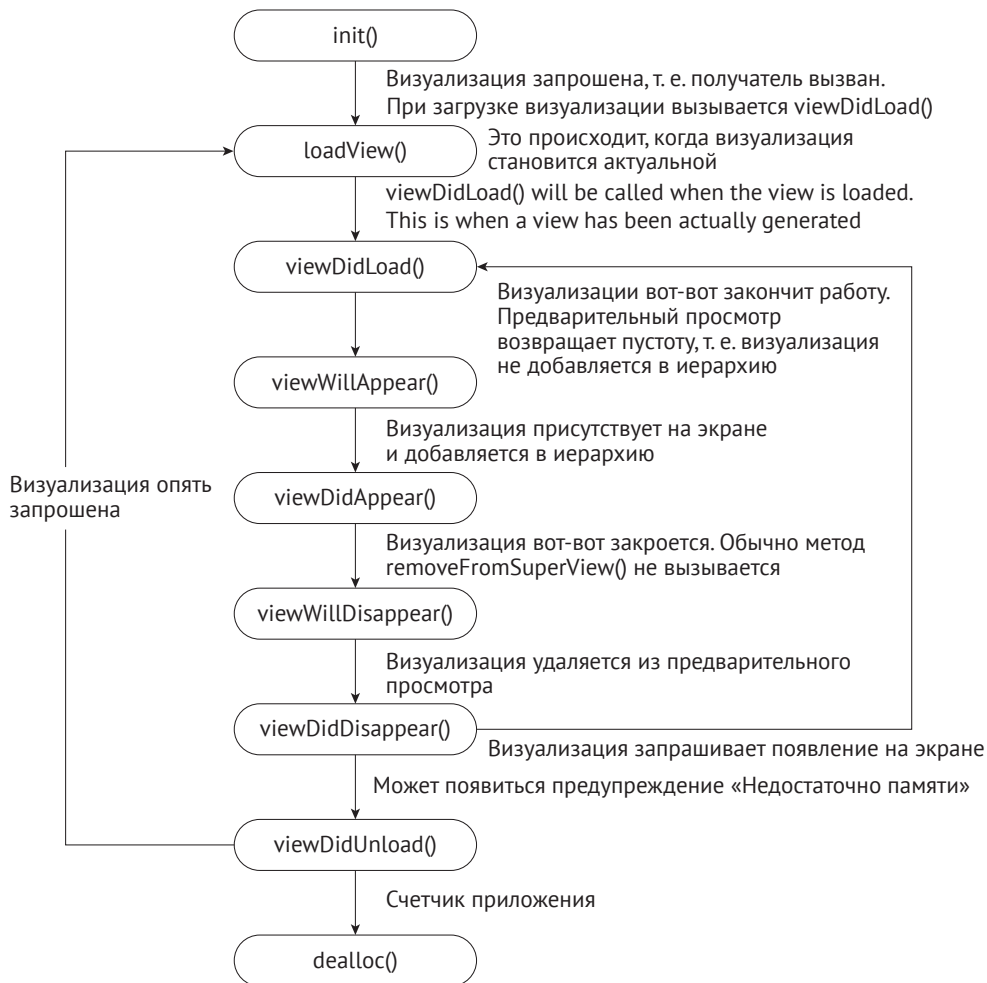


Рисунок 10.4. Жизненный цикл ViewController

- `viewDidLoad()` указывает, что визуализация загружена, но еще не отображена на экране. Переопределив этот метод, вы можете выполнить дополнительную инициализацию визуализаций, например удаление некоторых визуализаций, изменение ограничений, загрузку данных и т. д.;
- `viewWillAppear()` указывает, что визуализация скоро будет отображена на экране. Вы можете использовать этот метод, чтобы изменить ориентацию или стиль строки состояния;
- `viewDidAppear()` указывает, что визуализация была отображена на экране. Вы можете использовать этот метод, чтобы изменить способ представления на экране;

- `viewWillDisappear()` указывает, что визуализация вот-вот исчезнет, закроется или станет скрытой;
- `viewDidDisappear()` указывает, что визуализация исчезла, была закрыта или скрыта.

10.2. Создание нового проекта приложения для смартфона

Теперь, когда мы узнали о разработке проекта для Android и iOS, приступим к созданию нового проекта приложения. Поскольку для функции подготовки приложения требуется Bluetooth-модуль, симулятор не соответствует нашим требованиям. Пожалуйста, подготовьте смартфон для разработки и отладки приложения.

Прежде чем создавать новый проект, вам необходимо загрузить соответствующие инструменты разработки. Инструменты IDE интегрировали все необходимые среды и не требуют установки переменных среды и другой утомительной работы.

10.2.1. Подготовка к разработке под Android

Приложение Android можно разработать в среде Linux, Mac или Windows с использованием Android Studio. Она должна быть ориентирована на Android 6.0 и выше и может быть создана с применением Java и Kotlin.

Язык Kotlin предпочтителен для разработки приложений для Android. Будучи JVM-ориентированным языком, он полностью совместим с Java и предлагает более гибкий синтаксис и мощные функции. Еще в 2017 году Google объявляла, что будет поддерживать Kotlin на Android как первоклассный язык. Если у вас есть опыт разработки на Java, вы легко сможете начать работу с Kotlin.

10.2.2. Создание нового проекта Android

Чтобы создать новый проект Android, выполните следующие действия.

После загрузки и установки Android Studio на свой компьютер откройте ее, и вы увидите интерфейс, показанный на рис. 10.5. Затем нажмите **New Project** (Новый проект).

Выберите **Empty Activity** (Пустое действие), нажмите **Next** (Далее), и появится диалоговое окно **New Project** (Новый проект), показанное на рис. 10.6.

Укажите имя вашего проекта (например, `MyRainmaker`), имя программного пакета и установите место сохранения, язык (Kotlin) и минимальный SDK (Android 6.0 и выше). Затем нажмите **Finish** (Готово), чтобы создать новый проект.

Когда вы создаете проект впервые, автоматическая загрузка всех зависимостей может занять некоторое время, поэтому, пожалуйста, будьте терпеливы.

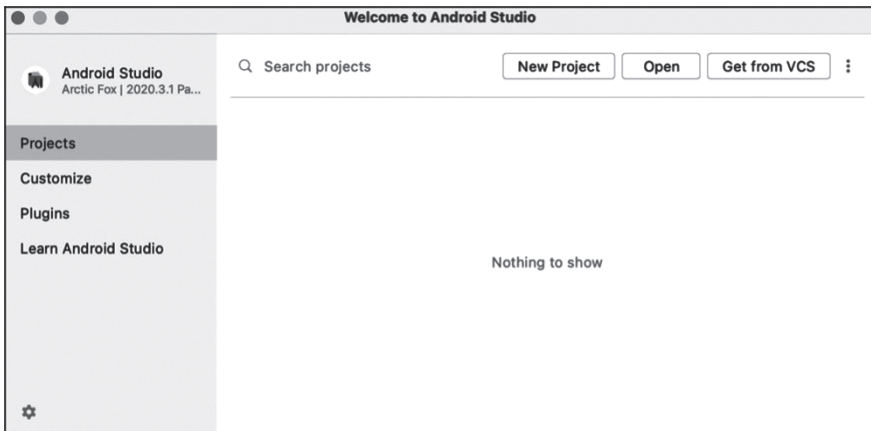


Рисунок 10.5. Интерфейс Android Studio

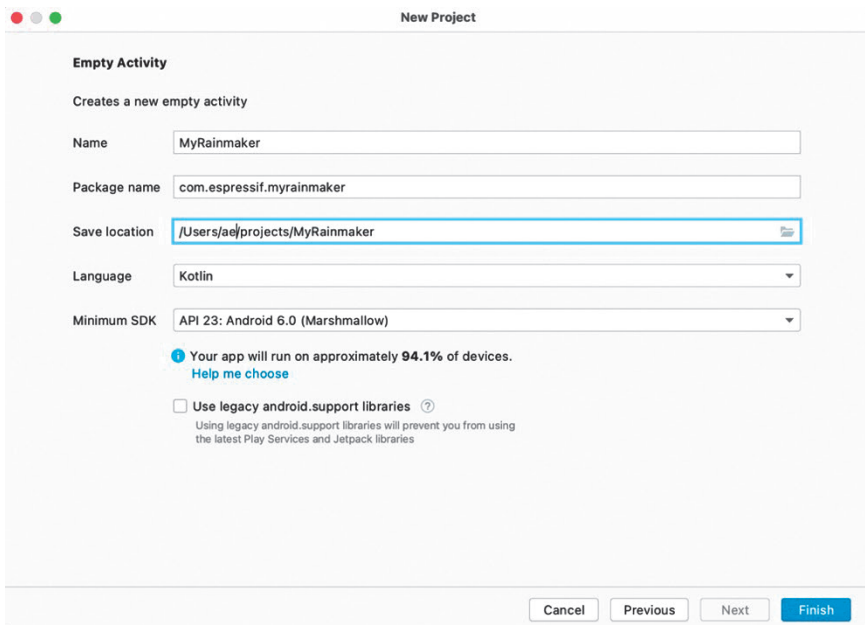


Рисунок 10.6. Диалоговое окно New Project


10.2.3. Добавление зависимостей для MyRainmaker

Добавьте репозиторий в `settings.gradle` (*Project Settings*).

```
1. repositories {  
2. ...  
3. maven { url 'https://jitpack.io' }  
4. }
```

Добавьте зависимости в build.gradle (модуль: *MyRainmaker.app*).

```
1. dependencies {
2.     implementation 'org.greenrobot:eventbus:3.2.0'
3.     implementation 'com.github.espressif:
4.         esp-idf-provisioning-android:lib-2.0.11'
5.     implementation 'com.github.espressifApp:rainmaker-proto-java:1.0.0'
6.     implementation 'com.google.protobuf:protobuf-javalite:3.14.0'
7.     implementation 'com.google.crypto.tink:tink-android:1.6.1'
8. }
```

Нажмите **Sync Now** (Синхронизировать сейчас) или кнопку  (Sync) в правом верхнем углу, чтобы загрузить зависимости, как показано на рис. 10.7.

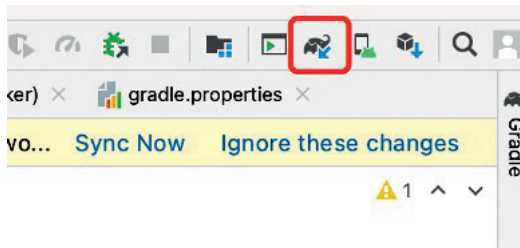


Рисунок 10.7. Загрузить зависимости

10.2.4. Запрос разрешений в Android

Добавьте необходимые разрешения в файл AndroidManifest.xml.

```
1. //Location permissions
2. <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
3. //Bluetooth permissions
4. <uses-permission android:name="android.permission.BLUETOOTH" />
5. <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
6. //Network permissions
7. <uses-permission android:name="android.permission.INTERNET" />
```

Помимо локальных разрешений, разрешения на определение местоположения также должны быть запрошены отдельно в следующем действии.

```
1. registerForActivityResult(ActivityResultContracts.RequestPermission())
2. { granted ->
3.     //Result callback
4.     if (granted) {
5.         //Permission granted
6.     } else {
7.         //Permission denied
8.     }
9. }.launch(android.Manifest.permission.ACCESS_FINE_LOCATION)
```

Добавьте приведенный код к методу `onCreate().activity`, чтобы приложение запрашивало разрешение на местоположение при каждом запуске.

10.2.5. Подготовка к разработке iOS

Приложение для iOS может быть разработано на компьютерах под управлением macOS 10.12 и выше с использованием Xcode (доступно в App Store). Он должен быть ориентирован на iOS 11.0 и выше и может быть создан с помощью Swift и Objective-C.

Предпочтение отдается Swift, поскольку он быстрее, безопаснее и более интерактивен. Он удаляет указатели и другие небезопасные моменты доступа в Objective-C и переключается с синтаксиса в стиле smalltalk, используемого Objective-C, на точечную нотацию. Все примеры кода в этой главе написаны на Swift.

10.2.6. Создание нового проекта iOS

Чтобы создать новый проект iOS, выполните следующие действия.

После загрузки и установки Xcode на свой компьютер откройте его, нажмите **Create a new Xcode project** (Создать новый проект Xcode), выберите **iOS** → **App**, как показано на рис. 10.8, и нажмите **Next** (Далее). Вы должны увидеть новое приглашение для настройки деталей вашего проекта.

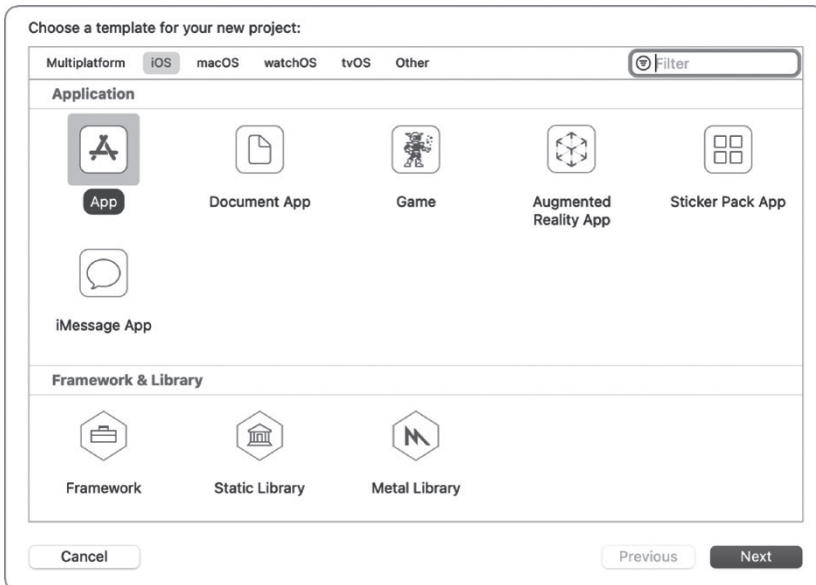


Рисунок 10.8. Выбрать iOS → App

Укажите название продукта (например, MyRainmaker), бригаду, идентификатор организации, интерфейс, срок службы и язык программирования (Swift), как показано на рис. 10.9. Нажмите **Далее**, и вы должны увидеть приглашение к указанию размещения проекта. Укажите путь к хранилищу и нажмите **Create** (Создать).

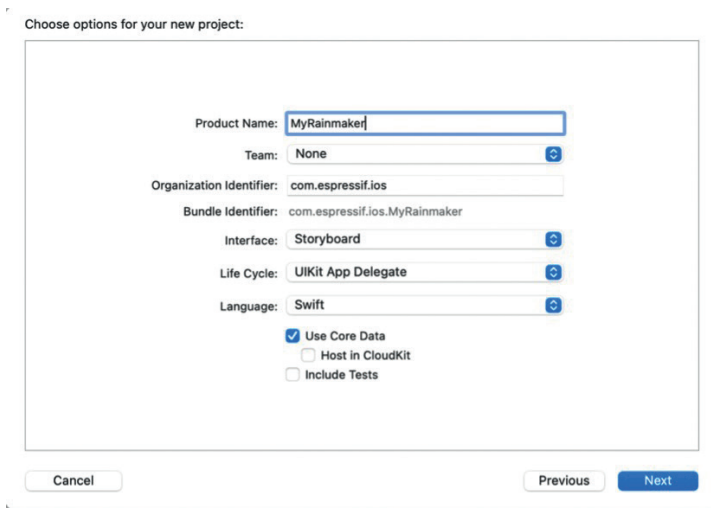


Рисунок 10.9. Настройка деталей проекта

10.2.7. Добавление зависимостей для MyRainmaker

Откройте терминал, перейдите в каталог проекта и выполните следующую команду для создания *Podfile*.

```
% touch Podfile
```

Откройте *Podfile* и добавьте зависимости:

```
1. # Uncomment the next line to define a global platform for your project
2. platform :ios, '12.0'
3.
4. target 'ESPRainMaker' do
5.     # Comment the next line if you're not using Swift and don't want to
6.     use dynamic frameworks
7.     use_frameworks!
8.
9.     # Pods for ESPRainMaker
10.
11.     pod 'MBProgressHUD', '~> 1.1.0'
12.     pod 'Alamofire', '~> 5.0.0'
13.     pod 'Toast-Swift'
14.     pod 'ReachabilitySwift'
15.     pod 'JWTDecode', '~> 2.4'
16.     pod 'M13Checkbox'
17.     pod 'ESPProvision'
18.     pod 'DropDown'
19.     pod 'FlexColorPicker'
20.
21. end
22.
```

```
23. post_install do |installer|
24.   .pods_project.targets.each do |target|
25.     target.build_configurations.each do |config|
26.       config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '12.0'
27.     end
28.   end
29. end
```

Выполните следующую команду, чтобы загрузить зависимости.

```
% pod install
```

После загрузки войдите в папку проекта и дважды щелкните *MyRainmaker.xcworkspace*, чтобы открыть проект, как показано на рис. 10.10.



Рисунок 10.10. Дважды щелкните *MyRainmaker.xcworkspace*.

Структура проекта показана на рис. 10.11.

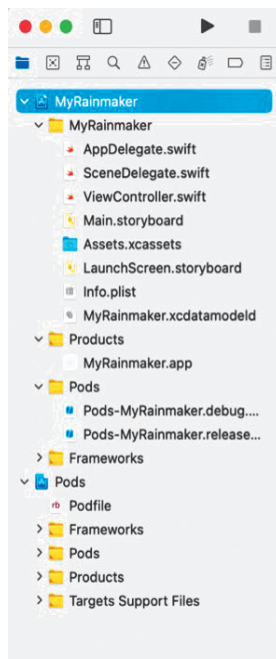


Рисунок 10.11. Структура проекта

10.2.8. Запрос разрешений в iOS

Добавьте следующие разрешения в *info.plist* в папке MyRainmaker:

- ключ `NSBluetoothAlwaysUsageDescription` и ключ `NSBluetoothPeripheralUsageDescription` для разрешения Bluetooth;
- ключ `NSCameraUsageDescription` для разрешения камеры на сканирование QR-кодов;
- ключ `NSLocationWhenInUseUsageDescription` для разрешения местоположения (это требуется устройствам под управлением iOS 13 и выше для доступа к SSID);
- ключ `NSLocalNetworkUsageDescription` для разрешения локальной сети (требуется для устройств под управлением iOS 14 и выше для связи по локальной сети).

10.3. Анализ функциональных требований приложения

В предыдущих разделах мы рассказали, как создать новый проект приложения, а также его структуру и жизненный цикл. Чтобы помочь вам понять разработку функций приложения более конкретно, мы разместили исходный код проекта приложения для смартфонов в нашем GitHub, и вы можете импортировать его в Android Studio/Xcode для использования в качестве образца.

Основная функция приложения для смартфона – настройка устройств, разработанных на базе чипов и модулей Espressif совместно с конкретным маршрутизатором и отправка команд через приложение для управления этими устройствами, например умными светильниками и датчиками. Другая функция – установка статуса устройства с помощью модуля планирования, например для включения в указанное время водяного нагревателя по дороге домой, чтобы горячая вода была доступна, как только вы приедете.

10.3.1. Анализ функциональных требований проекта

Прежде чем разрабатывать приложение, вам следует сначала определиться с функциональными модулями и конкретными функциями, которые будут реализованы в данном проекте. Проект приложения для смартфона в этой главе в основном включает в себя такие модули, как управление пользователями, подготовку и управление устройствами (подробнее функции показаны на рис. 10.12). В следующих разделах мы разберем модуль за модулем.

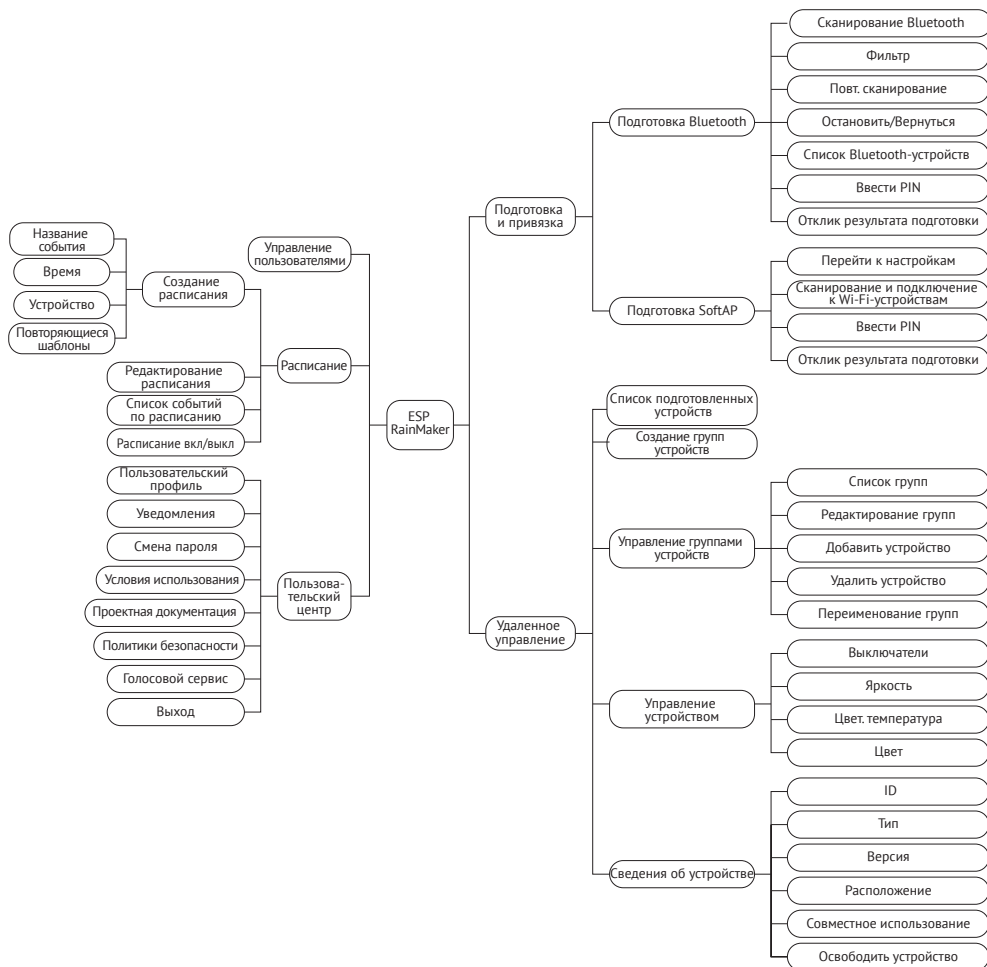


Рисунок 10.12. Функциональные требования проекта

10.3.2. Анализ требований к управлению пользователями

Регистрация и вход пользователя реализованы в одном интерфейсе и переключаются через кнопку-переключатель. Важнейшей частью этого модуля является разрешение сторонних учетных записей, сетевые запросы на регистрацию, вход в систему, получение проверочного кода и т. д., а также анализ данных.

Анализ требований к управлению пользователями показан на рис. 10.13.

- Чтобы войти в систему с помощью сторонней учетной записи, например GitHub, Apple или Google, сначала откройте веб-страницу в браузере и получите уникальный идентификатор учетной записи.
- Забытый пароль и код подтверждения можно осуществить через соответствующие облачные API.
- Введенные пароли по умолчанию отображаются в зашифрованном виде, и их можно переключить на открытый текст для подтверждения, щелкнув переключатель (значок глаза).

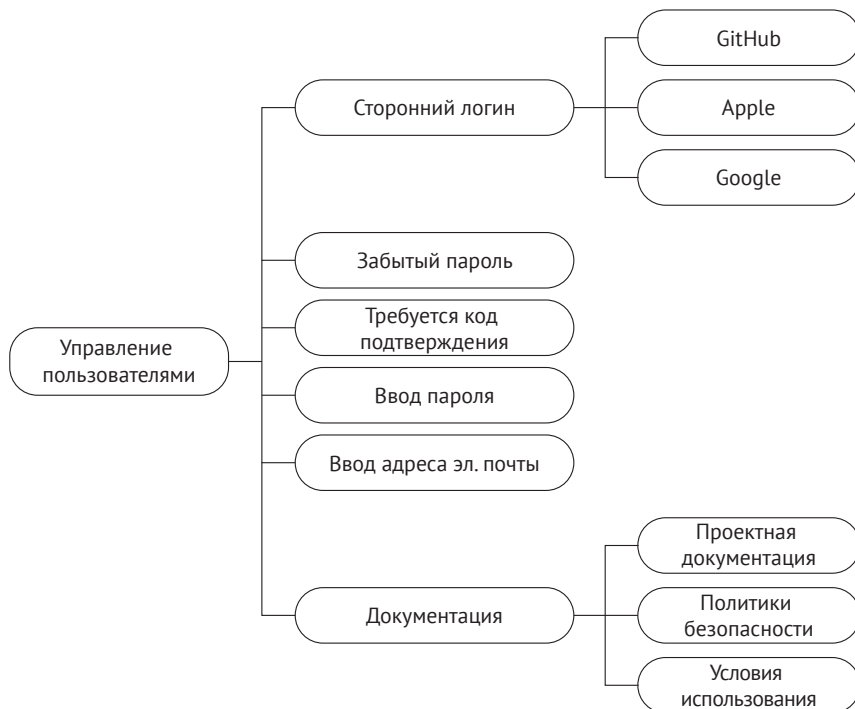


Рисунок 10.13. Анализ требований к управлению пользователями

- Введенные адреса электронной почты будут проверены с использованием регулярных выражений.
- Документация будет включать документы по всему проекту, доступ к которым можно получить из различных мест в приложении.

10.3.3. Анализ требований к подготовке и привязке устройства

Существует два способа подготовить устройство. Одним из них является подготовка по Bluetooth, когда приложение подключается и взаимодействует с устройством через Bluetooth, предоставляет данные для подготовки устройства и позволяет ему подключаться к сети. Другой способ – это инициализация SoftAP, при которой устройство запускает точку доступа Wi-Fi для подключения и обмена данными через смартфон. Как только устройство будет подготовлено, вводится PIN-код, чтобы привязать устройство к облаку. Анализ требований к подготовке и привязке устройства показан на рис. 10.14.

- Для подготовки Bluetooth приложению необходимо реализовать сканирование Bluetooth, подключение, подписку, передачу пакетов и другие функции.
- Для подготовки SoftAP приложению нужно перейти к интерфейсу настройки системы, подключиться к точке доступа Wi-Fi-устройства и отобразить информацию о подключенной точке доступа.

- Реализация привязки устройства после подготовки одинакова для обоих способов, и ее можно использовать повторно.

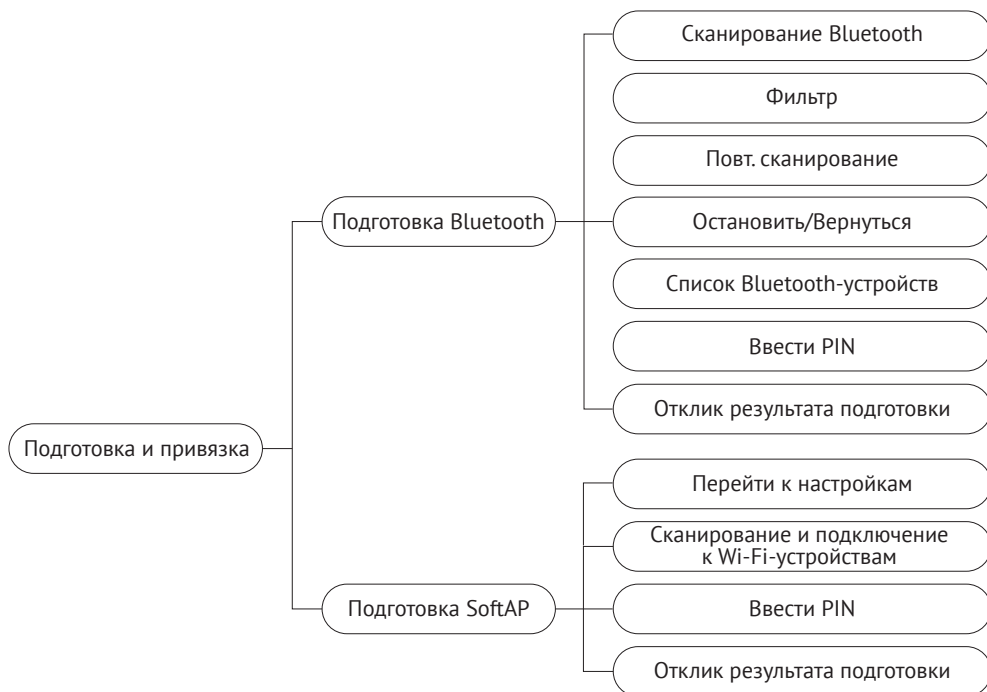


Рисунок 10.14. Анализ требований к подготовке и привязке устройства

10.3.4. Анализ требований к удаленному управлению

Как только устройство будет подготовлено и привязано, пользователи смогут использовать приложение на смартфоне для удаленного контроля и управления. Кроме того, пользователи также могут управлять несколькими устройствами одновременно и группировать их. Анализ требований к удаленному управлению показан на рис. 10.15.

- В приложении для смартфона все подготовленные устройства будут отображаться в списке и могут легко включаться и выключаться.
- Выбрав указанное устройство, пользователи войдут в его отдельный интерфейс управления, различающийся для каждого типа устройства. Например, его можно использовать для управления выключателями, яркостью или цветом.
- Интерфейс сведений об устройстве покажет идентификатор устройства, тип, версию, местоположение и т. д., а также позволит пользователям анализировать и освобождать устройства.
- Создавая группы, устройствами можно управлять совместно.



Рисунок 10.15. Анализ требований к удаленному управлению

10.3.5. Анализ требований к планированию

Функция планирования относительно проста и очень похожа на будильники, которые мы используем каждый день. В основном она включает в себя такие функции, как создание расписаний, список событий расписания, редактирование расписаний, включение/выключение расписаний и т. д. Анализ требований к расписанию показан на рис. 10.16. К деталям расписания относятся название события, дата, время, повторяющийся шаблон и т. д.



Рисунок 10.16. Анализ требований к планированию

10.3.6. Анализ требований к пользовательскому центру

Модуль пользовательского центра в основном содержит профиль пользователя, уведомление, изменение пароля, условия использования, проектные документы, политики безопасности, голосовые услуги и выход из системы. Обратите внимание, что изменения функции пароля и выхода из системы должны вызывать облачный API. Анализ требований к пользовательскому центру показан на рис. 10.17.



Рисунок 10.17. Анализ требований к пользовательскому центру

10.4. Разработка системы управления пользователями

После анализа функциональных требований проекта в разделе 10.3 у вас уже должно быть общее представление о модулях и функциях, которые необходимо разработать, а также о необходимых фреймворках и сторонних библиотеках. В этом разделе мы поместим все модули и функции в код. Основываясь на новых проектах и разрешениях, настроенных ранее, теперь вам нужно знать классы, предназначенные для каждого интерфейса, и связи между ними, чтобы добиться лучшей работы с помощью кода. Код для каждой функции будет инкапсулирован для повторного использования и размещения в модулях.

10.4.1. Введение в API RainMaker

RainMaker Cloud поддерживает два типа API: неаутентифицированный Unauthenticated и аутентифицированный (прошедший проверку подлинности) Authenticated. Неаутентифицированные API-интерфейсы не имеют никаких токенов аутентификации в заголовке HTTP и получают access_token в ответе при успешном

входе пользователей в систему. Аутентифицированные API-интерфейсы помечены в файле *Swagger* знаком «lock» спереди. Их `access_token` нуждается в прохождении авторизации, что будет передано через `Authorization` в HTTP-заголовке.

Документацию по API RainMaker можно найти по адресу <https://swaggerapis.rainmaker.espressif.com>.

Когда смартфоны взаимодействуют с облаком RainMaker, базовым протоколом является безопасный протокол передачи гипертекста (HyperText Transfer Protocol Secure, HTTPS). HTTPS может аутентифицировать сервер для защиты конфиденциальности и целостности при обмене данными. Тело HTTPS, получаемое облаком RainMaker, находится в формате JSON.

10.4.2. Инициализация связи через смартфон

Android и iOS обеспечивают хорошую встроенную поддержку HTTPS и JSON.

- Система Android использует `JSONObject` и `JSONArray` для сборки и парсинга массивов JSON, а также `URLConnection` для инициирования HTTPS-запросов.
- Система iOS использует `NSJSONSerialization` для сбора и парсинга данных JSON, а также `NSURLSession` для инициирования HTTPS-запросов.

Конечно, вы также можете использовать сторонние библиотеки HTTPS и JSON.

10.4.3. Регистрация учетной записи

Для начала нам нужно реализовать регистрацию новой учетной записи, которая будет использоваться на последующих этапах для привязки устройства и удаленного управления. В данном проекте учетная запись регистрируется по адресу электронной почты.

В интерфейсе регистрации есть кнопка переключения между **SIGN IN** (Вход) и **SIGN UP** (Регистрация). В интерфейсе **SIGN UP** есть три поля ввода: **Email** (адрес электронной почты), **Password** (пароль) и **Confirm Password** (подтверждение пароля). Содержимое полей **Password** и **Confirm Password** можно отобразить или скрыть, переключив их видимость, чтобы пользователи могли проверить, правильно ли они ввели пароль. Пароль должен содержать по крайней мере одну заглавную букву и одну цифру.

Прежде чем нажать **Register** (Регистрироваться), пользователи должны прочитать **Privacy Policy** (политику конфиденциальности) и **Terms of Use** (условия использования) и согласиться с ними. Затем программа перейдет к интерфейсу проверки, и на адрес электронной почты будет отправлен цифровой код. Пользователям необходимо ввести правильный цифровой код для завершения процедуры регистрации.

Интерфейс регистрации показан на рис. 10.18.

На рис. 10.19 представлен API для регистрации учетной записи. Для получения подробной информации, пожалуйста, обратитесь по адресу <https://swaggerapis.rainmaker.espressif.com/#/User/usercreation>.

Рисунок 10.18. Интерфейс регистрации

POST `/{version}/user` Creates the new user or confirms the user

Рисунок 10.19. API для регистрации аккаунта

Функция регистрации аккаунта реализована следующим образом.

Создать новый аккаунт. Ниже показан API регистрации учетной записи, где `user_name` относится к адресу электронной почты и `password` – к паролю, введенным при регистрации.

```

1. POST /v1/user
2. Content-Type: application/json
3.
4. {
5.     "user_name": "username@domain.com",
6.     "password": "password"
7. }
```

Чтобы создать новую учетную запись в Android, используйте:

```

1. @POST
2. Call<ResponseBody> createUser(@Url String url, @Body JsonObject body);
```

Исходный код

Исходный код создания новой учетной записи в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiInterface.java.

Чтобы создать новую учетную запись в iOS, используйте:

```
1. func createUser(name: String, password: String) {
2.     apiWorker.callAPI(endPoint: .createNewUser(url: self.url, name: name,
3.         password: password), encoding: JSONEncoding.default) { data, error in
4.         self.apiParser.parseResponse(data, withError: error) { umError in
5.             self.presenter?.verifyUser(withName: name, andPassword:
6.
7. }
```

Исходный код

Исходный код создания новой учетной записи в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_ios/ESPRainMaker/ES-PRainMaker/UserManagement/Interactors/ESPCreateUserService.swift.

Подтверждение аккаунта после получения цифрового кода. Ниже показаны подробности API, где `user_name` относится к адресу электронной почты, использованному для регистрации, а `verification_code` – к цифровому коду.

```
POST /v1/user
Content-Type: application/json

{
  "user_name": "username@domain.com",
  "verification_code": "verification_code"
}
```

Чтобы подтвердить аккаунт с помощью цифрового кода в Android, используйте:

```
1. @POST
2. Call<ResponseBody> confirmUser(@Url String url, @Body JsonObject body);
```

Исходный код

Исходный код проверки учетной записи в Android см. https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiInterface.java.

Чтобы подтвердить аккаунт с помощью цифрового кода в iOS, используйте:

```

1. func confirmUser(name: String, verificationCode: String) {
2.     apiWorker.callAPI(endPoint: .confirmUser(url: self.url, name: name,
3.         verificationCode: verificationCode),
4.         encoding: JSONEncoding.default) { data, error in
5.         self.apiParser.parseResponse(data, withError: error) { umError in
6.             self.presenter?.userVerified(withError: umError)
7.         }
8.     }
9. }

```

Исходный код

Исходный код проверки учетной записи в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_ios/ESPRainMaker/ESPRainMaker/UserManagement/Interactors/ESPCreateUserService.swift.

10.4.4. Вход в учетную запись

После регистрации учетной записи мы можем вызвать API входа в учетную запись, чтобы получить token и основной профиль для аутентификации.

Проект приложения для смартфона в этой главе поддерживает вход через сторонние учетные записи, такие как GitHub, Apple и Google. Если у пользователей есть учетные записи на этих трех платформах, они могут войти прямо в приложение без регистрации.

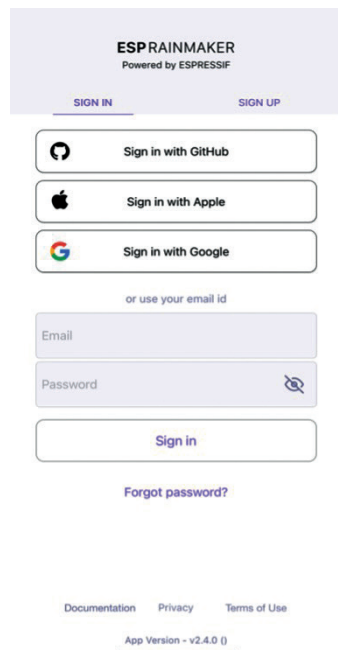


Рисунок 10.20. Интерфейс входа

Если пользователи уже зарегистрировали новые учетные записи, они также могут войти в приложение, введя свой адрес электронной почты и пароль.

Если пользователи забудут свой пароль, для сброса пароля они могут нажать **Forgot password?** (Забыли пароль?) под кнопкой **SIGN IN**.

В нижней части интерфейса входа **SIGN IN** находится документация, связанная с проектом, соглашение о конфиденциальности, условия использования и версия приложения. Интерфейс входа показан на рис. 10.20.

Функция входа в аккаунт реализована следующим образом.

Запросите токен доступа. API показан на рис. 10.21, подробную информацию можно найти по адресу <https://swaggerapis.rainmaker.espressif.com/#/User/login>.

```
POST /{version}/login Handle login or extend session request from the user
```

Рисунок 10.21. API для входа в аккаунт

```
POST /v1/login
Content-Type: application/json
{
  "user_name": "username@domain.com",
  "password": "password"
}
```

Сервер отвечает на запрос следующим образом:

```
{
  "status": "success",
  "description": "Login successful",
  "idtoken": "idtoken",
  "accesstoken": "accesstoken",
  "refresh_token": "refresh_token"
}
```

Среди этих полей `status` сообщает, успешен ли запрос; `description` обеспечивает подробности запроса; `accesstoken` – это токен, который будет добавлен в заголовок HTTP-запроса всеми API, требующими разрешения пользователя, в формате `Authorization:$accesstoken`; `idtoken` и `refresh_token` на данный момент не используются и поэтому здесь не объясняются.

Чтобы запросить токен доступа в Android, используйте:

```
1. @POST
2. Call<ResponseBody> login(@Url String url, @Body JsonObject body);
```

Исходный код

Исходный код запроса токена доступа в Android см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/expressif/cloudapi/ApiInterface.java.

Чтобы запросить токен доступа в iOS, используйте:

```
1. func loginUser(name: String, password: String) {
2.     apiWorker.callAPI(endpoint: .loginUser(url: self.url,
3.     name: name, password: password),
4.     encoding: JSONEncoding.default) { data, error in
5.     self.apiParser.parseExtendSessionResponse(data,
6.     error: error) { _, umError in
7.     self.presenter?.loginCompleted(withError: umError)
8.     }
9.     }
10. }
```

Исходный код

Исходный код запроса токена доступа в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_ios/ESPRainMaker/ESPRainMaker/UserManagement/Interactors/ESPLoginService.swift.

Получить профиль пользователя. API показан на рис. 10.22, подробную информацию можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/User/getUser>.



Рисунок 10.22. API для получения профиля пользователя

```
GET /v1/user
Authorization: $access_token
```

В ответ на запрос «получить профиль пользователя» сервер возвращает:

```
{
  "user_id": "string",
  "user_name": "string",
  "super_admin": true,
  "picture_url": "string",
  "name": "string",
  "mfa": true,
  "phone_number": "<+Mobile Number with country code>"
}
```

Среди этих полей `user_id` – это уникальный идентификатор пользователя, который будет использоваться позже при подготовке; `user_name` относится к учетной записи; `super_admin` возвращается `true` только тогда, когда пользователь является суперадминистратором; `picture_url` указывает на изображение профиля пользователя; `phone_number` – номер мобильного телефона пользователя; `name` и `mfa` не используются в этом проекте и, следовательно, здесь не объясняются.

Чтобы получить профиль пользователя в Android, используйте:

```
1. @GET
2. Call<ResponseBody> fetchUserDetails(@Url String url);
```

Исходный код

Исходный код получения профиля пользователя в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiInterface.java.

Чтобы получить профиль пользователя в iOS, используйте:

```

1. func fetchUserDetails() {
2.     sessionWorker.checkUserSession { accessToken, error in
3.         if let token = accessToken, token.count > 0 {
4.             self.apiWorker.callAPI(endPoint: .fetchUserDetails(url: self.url,
5.                 accessToken: token), encoding:
6.                 JSONEncoding.default) { data, error in
7.                 self.apiParser.parseUserDetailsResponse(data,
8.                     withError: error) { umError in
9.                     self.presenter?.userDetailsFetched(error: umError)
10.                    return
11.                }
12.            }
13.        } else {
14.            self.presenter?.userDetailsFetched(error: error)
15.        }
16.    }
17. }

```

Исходный код

Исходный код получения профиля пользователя в iOS см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/main/phone_app/app_ios/ESPRainMaker/ES-PRainMaker/UserManagement/Interactors/ESPUserService.swift.

10.5. Разработка системы подготовки устройств

Как описано в разделе 10.4, мы можем получить через API токен доступа и `user_id` учетной записи RainMaker для входа в учетную запись и получения профиля пользователя. Следующий шаг – найти устройство, подключить его к маршрутизатору и активировать в облаке. Подходящей библиотекой подготовки для приложения является `idf-provisioning`, внедренная на основе ESP-IDF provisioning.

Для получения информации о методах подготовки, пожалуйста, обратитесь к <https://bookc3.expressif.com/provisioning>.

На рис. 10.23 показан обмен данными между смартфоном и устройством во время подготовки. Этот обмен также упоминается в разделе 7.3.4 при рассказе о настройке Bluetooth.

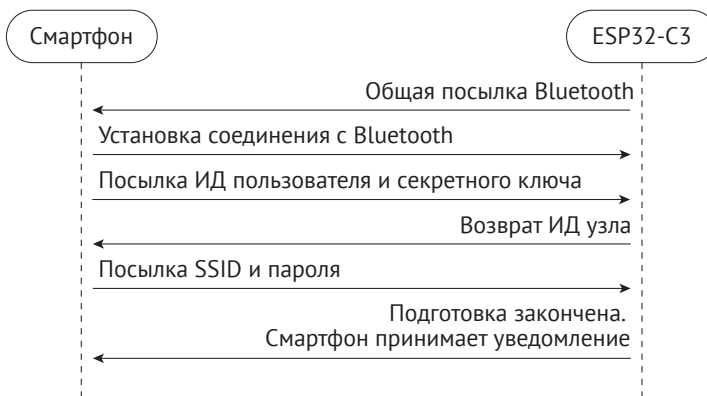


Рисунок 10.23. Обмен данными между смартфоном и устройством во время подготовки

10.5.1. Сканирование устройств

Перейдите на домашнюю страницу приложения, нажмите кнопку в правом верхнем углу, для сканирования QR-кода на устройствах будет вызвана камера. Это самый быстрый способ обнаружить умное устройство. Помимо сканирования QR-кода, пользователи также могут обнаруживать устройства с помощью Bluetooth LE или SoftAP. Интерфейс сканирования устройства показан на рис. 10.24. (Фактический интерфейс может отличаться от скриншотов в этой книге из-за обновлений приложения.)

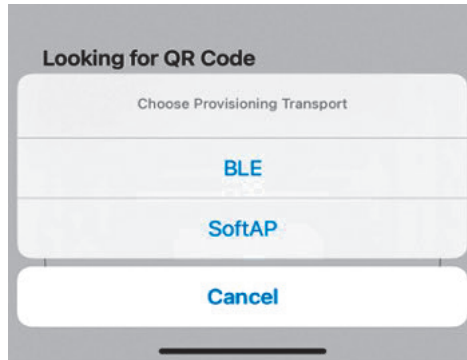


Рисунок 10.24. Интерфейс сканирования устройств

В следующих разделах мы будем использовать настройку через Bluetooth в качестве примера процесса подготовки устройства.

Сканирование устройств в Android

Пользователям следует обновить свои телефоны до Android 9.0 или выше и включить GPS для поиска сигналов Bluetooth LE. Код выглядит следующим образом:

Исходный код

Исходный код сканирования устройств в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/ui/activities/BLEProvisionLanding.java.

```
1. private void startScan() {
2.     //Code Omitted
3.     if (ActivityCompat.checkSelfPermission(this,
4.         Manifest.permission.ACCESS_FINE_LOCATION) ==
5.         PackageManager.PERMISSION_GRANTED) {
6.         provisionManager.searchBleEspDevices(deviceNamePrefix, bleScanListener);
7.         updateProgressAndScanBtn();
8.     } else {
9.         //Code Omitted
10.    }
11.}
12.
```

```

13. private BleScanListener bleScanListener = new BleScanListener() {
14.     @Override
15.     public void scanStartFailed() {
16.         Toast.makeText(BLEProvisionLanding.this,
17.             "Please turn on Bluetooth to connect BLE device",
18.             Toast.LENGTH_SHORT).show();
19.     }
20.
21.     @Override
22.     public void onPeripheralFound(BluetoothDevice device,
23.         ScanResult scanResult) {
24.         //Code Omitted
25.     }
26.
27.     @Override
28.     public void scanCompleted() {
29.         //Code Omitted
30.     }
31.
32.     @Override
33.     public void onFailure(Exception e) {
34.         //Code Omitted
35.     }
36. };

```

Сканирование устройств в iOS

В следующем коде префикс используется для фильтрации устройств по именам. Если устройство имеет свой уникальный идентификатор, его можно использовать для фильтрации. Код iOS имеет дополнительный параметр `transport` с двумя возможными значениями: `ble` и `softap`, – относящимися к двум методам подготовки.

Исходный код

Исходный код сканирования устройств в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_ios/ESPRainMaker/ESPRainMaker/Interface/Provision/BLE/BLELandingViewController.swift.

```

1. ESPProvisionManager.shared.searchESPDevices(devicePrefix: "prefix",
2. transport: .ble, security: Configuration.shared.espProvSetting.securityMode)
3. { bleDevices, _ in
4.     //Code Omitted
5. }

```

10.5.2. Подключение устройств

Подключение устройств в Android.

Исходный код

Исходный код подключения устройств в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/ui/activities/BLEProvisionLanding.java.

```

1. override fun onCreate(savedInstanceState: Bundle?) {
2.     super.onCreate(savedInstanceState)
3.     //Code Omitted
4.     EventBus.getDefault().register(this)
5. }
6.
7. @Override
8. protected void onDestroy() {
9.     EventBus.getDefault().unregister(this);
10.    super.onDestroy();
11. }
12.
13. @Subscribe(threadMode = ThreadMode.MAIN)
14. public void onEvent(DeviceConnectionEvent event) {
15.     handler.removeCallbacks(disconnectDeviceTask);
16.     switch (event.getEventType()) {
17.         case ESPConstants.EVENT_DEVICE_CONNECTED:
18.             //Code Omitted
19.             break;
20.
21.         case ESPConstants.EVENT_DEVICE_DISCONNECTED:
22.             //Code Omitted
23.             break;
24.
25.         case ESPConstants.EVENT_DEVICE_CONNECTION_FAILED:
26.             //Code Omitted
27.             break;
28.     }
29. }

```

В Android для уведомления о действиях при изменении статуса соединения Bluetooth LE используется EventBus, поэтому в activities необходимо зарегистрировать функцию обратного вызова. После обнаружения устройства мы должны сначала создать экземпляр устройства, как показано в приведенном выше коде, а затем вызвать API подключения следующим образом, чтобы приложение для смартфона могло инициировать запрос на соединение с устройством:

```

1. public void deviceClick(int deviceClickedPosition) {
2.     stopScan();
3.     isConnecting = true;
4.     isDeviceConnected = false;
5.     btnScan.setVisibility(View.GONE);
6.     rvBleDevices.setVisibility(View.GONE);
7.     progressBar.setVisibility(View.VISIBLE);
8.     this.position = deviceClickedPosition;
9.     BledDevice bleDevice = deviceList.get(deviceClickedPosition);
10.    String uuid = bluetoothDevices.get(bleDevice.getBluetoothDevice());
11.
12.    if (ActivityCompat.checkSelfPermission(BLEProvisionLanding.this,
13.        Manifest.permission.ACCESS_FINE_LOCATION) ==
14.        PackageManager.PERMISSION_GRANTED) {
15.        boolean isSec1 = true;
16.        if (AppConstants.SECURITY_0.equalsIgnoreCase(BuildConfig.SECURITY)) {
17.            isSec1 = false;
18.        }

```

```

19.     if (isSec1) {
20.         provisionManager.createESPDevice(ESPConstants.TransportType.
21.             TRANSPORT_BLE, ESPConstants.SecurityType.SECURITY_1);
22.     } else {
23.         provisionManager.createESPDevice(ESPConstants.TransportType.
24.             TRANSPORT_BLE, ESPConstants.SecurityType.SECURITY_0);
25.     }provisionManager.getEspDevice().connectBLEDevice(bleDevice.
26.         getBluetoothDevice(), uuid);
27.     handler.postDelayed(disconnectDeviceTask, DEVICE_CONNECT_TIMEOUT);
28. } else {
29.     Log.e(TAG, "Not able to connect device as Location permission is
30.         not granted." );
31. }
32. }

```

Исходный код

Исходный код инициирования соединения приложения для смартфона в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/ui/activities/BLEProvisionLanding.java.

Подключение устройств в iOS

Приложение iOS предоставляет прокси для функции обратного вызова соединения, поэтому мы можем напрямую вызывать экземпляр интерфейса подключения, возвращенный при сканировании устройства, и передать состояние прокси как параметр для `bleConnectionStatusHandler`. Код выглядит следующим образом:

Исходный код

В папке `Pods` хранятся импортированные сторонние библиотеки. Файлы в этой папке будут создаваться, только если проект скомпилирован и установлен локально, потому что онлайн-ссылка отсутствует. Для исходного кода подключения устройства в iOS см.: [book-esp32c3-iot-projects/phone_app/pods/espprovision/ESPDevice.swift](https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/pods/espprovision/ESPDevice.swift).

```

1. open func connect(delegate: ESPDeviceConnectionDelegate? = nil,
2.     completionHandler: @escaping (ESPSessionStatus) -> Void) {
3.     ESPLog.log("Connecting ESPDevice..." )
4.     self.delegate = delegate
5.     switch transport {
6.     case .ble:
7.         ESPLog.log("Start connecting ble device." )
8.         bleConnectionStatusHandler = completionHandler
9.         if espBleTransport == nil {
10.            espBleTransport = ESPBleTransport(scanTimeout: 0,
11.                deviceNamePrefix: "")
12.        }
13.        espBleTransport.connect(peripheral: peripheral,
14.            withOptions:
15.                nil,
16.            delegate: self)

```

```

17.         case .softap:
18.             ESPLog.log("Start connecting SoftAp device." )
19.             if espSoftApTransport == nil {
20.                 espSoftApTransport = ESPSoftAPTransport(baseUrl:
21.                                                             ESUtility.baseUrl)
22.             }
23.             self.connectToSoftApUsingCredentials(ssid: name,
24.                                                  completionHandler: completionHandler)
25.         }
26.     }

```

10.5.3. Генерация секретных ключей

Секретные ключи используются для аутентификации при привязке устройств пользователями и могут генерироваться случайным образом через приложение для смартфона. Чтобы сгенерировать секретный ключ в Android, добавьте:

```
1. final String secretKey = UUID.randomUUID().toString();
```

Чтобы сгенерировать секретный ключ в iOS, добавьте:

```
1. let secretKey = UUID().uuidString
```

10.5.4. Получение идентификатора (ИД) узла

Каждое устройство имеет свой уникальный ИД, называемый идентификатором узла. После подготовки устройства оно может быть привязано к облачному серверу через идентификатор его узла после отправки запроса привязки. Цель привязки заключается в обеспечении последующего дистанционного управления.

Получение ИД узла в Android

Исходный код

Исходный код получения идентификатора узла в Android см.: https://github.com/esp8266/ESP8266-Android-App/blob/main/phone_app/app/src/main/java/com/esp8266/ui/activities/ProvisionActivity.java.

Чтобы создать запрос на идентификатор узла, используйте следующий код:

```

1. EspRmakerUserMapping.CmdSetUserMapping deviceSecretRequest =
2.     EspRmakerUserMapping.CmdSetUserMapping.newBuilder()
3.         .setUserID(ApiManager.userId)
4.         .setSecretKey(secretKey)
5.         .build();
6. EspRmakerUserMapping.RMakerConfigMsgType msgType =EspRmakerUserMapping.
7.     RMakerConfigMsgType.TypeCmdSetUserMapping;
8. EspRmakerUserMapping.RMakerConfigPayload payload = EspRmakerUserMapping.
9.     RMakerConfigPayload.newBuilder()
10.        .setMsg(msgType)
11.        .setCmdSetUserMapping(deviceSecretRequest)
12.        .build();

```

Для инициализации запроса:

```
1. private void associateDevice() {
2.   provisionManager.getEspDevice().sendDataToCustomEndPoint(AppConstants.
3.   HANDLER_RM_USER_MAPPING, payload.toByteArray(), new ResponseListener() {
4.     @Override
5.     public void onSuccess(byte[] returnData) {
6.       processDetails(returnData, secretKey);
7.     }
8.     @Override
9.     public void onFailure(Exception e) {
10.      //Code Omitted
11.    }
12.  });
13. }
```

Чтобы проанализировать ответ устройства, используйте следующий код:

```
1.private void processDetails(byte[] responseData, String secretKey) {
2.
3.  try {
4.    EspRmakerUserMapping.RMakerConfigPayload payload = EspRmakerUserMapping.
5.      RMakerConfigPayload.parseFrom(responseData);
6.    EspRmakerUserMapping.RespSetUserMapping response = payload.
7.      getRespSetUserMapping();
8.
9.    if (response.getStatus() == EspRmakerUserMapping.RMakerConfigStatus.
10.      Success) {
11.      //Node ID received. Ready for device provisioning.
12.      receivedNodeId = response.getNodeId();
13.    }
14.  } catch (InvalidProtocolBufferException e) {
15.    //Code Omitted
16.  }
17.}
```

Получение ИД узла в iOS

Исходный код

Исходный код получения идентификатора узла в iOS см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/cf25c67fbc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/AWSCognito/DeviceAssociation.swift.

```
1. private func createAssociationConfigRequest() throws -> Data? {
2.   var configRequest = Rainmaker_CmdSetUserMapping()
3.   configRequest.secretKey = secretKey
4.   configRequest.userID = User.shared.userInfo.userID
5.   var payload = Rainmaker_RMakerConfigPayload()
6.   payload.msg = Rainmaker_RMakerConfigMsgType.typeCmdSetUserMapping
7.   payload.cmdSetUserMapping = configRequest
8.   return try payload.serializedData()
9. }
```

Для инициализации запроса:

```
1. func associateDeviceWithUser() {
2.     do {
3.         let payloadData = try createAssociationConfigRequest()
4.         if let data = payloadData {
5.             device.sendData(path: Constants.associationPath, data: data)
6.                 { response, error in
7.             guard error == nil, response != nil else {
8.                 self.delegate?.deviceAssociationFinishedWith(success:
9.                     false, nodeID: nil,
10.                    error: AssociationError.runtimeError
11.                    (error!.localizedDescription))
12.             return
13.         }
14.         self.processResponse(responseData: response!)
15.     }
16. } else {
17.     delegate?.deviceAssociationFinishedWith(success: false, nodeID:
18.        nil, error: AssociationError.runtimeError
19.        ("Unable to fetch request payload." ))
20. }
21. } catch {
22.     delegate?.deviceAssociationFinishedWith(success: false, nodeID: nil,
23.        error: AssociationError.runtimeError
24.        ("Unable to fetch request payload." ))
25. }
26. }
```

Чтобы проанализировать ответ устройства, используйте следующий код:

```
1. func processResponse(responseData: Data) {
2.     do {
3.         let response = try Rainmaker_RMakerConfigPayload(serializedData:
4.             responseData)
5.         if response.respSetUserMapping.status == .success {
6.             //Node ID received. Ready for device provisioning.
7.             delegate?.deviceAssociationFinishedWith(success: true, nodeID:
8.                 response.respSetUserMapping.nodeID, error: nil)
9.         } else {
10.            delegate?.deviceAssociationFinishedWith(success: false, nodeID:
11.                nil, error: AssociationError.runtimeError
12.                ("User node mapping failed." ))
13.        }
14.    } catch {
15.        delegate?.deviceAssociationFinishedWith(success: false, nodeID: nil,
16.            error: AssociationError.runtimeError
17.            (error.localizedDescription))
18.    }
19. }
```

10.5.5. Подготовка устройств

Как только приложение для смартфона установит соединение с устройством, мы сможем через Bluetooth реализовать протоколы подготовки устройства и активировать его на облаке. Весь процесс подготовки состоит из пяти шагов, как показано на рис. 10.25.

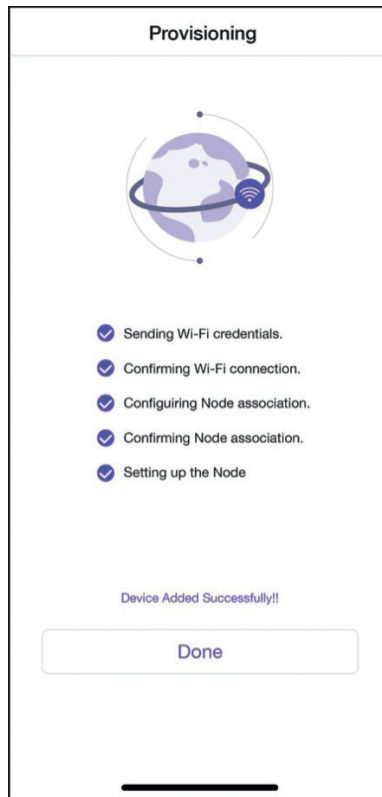


Рисунок 10.25. Интерфейс подготовки устройства

Подготовка устройств в Android

Исходный код

Исходный код подготовки в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/ui/activities/ProvisionActivity.java.

```
1. private void provision() {
2.
3.     provisionManager.getEspDevice().provision(ssidValue, passphraseValue,
4.         new ProvisionListener() {
5.             @Override
6.             public void createSessionFailed(Exception e) {}
7.             @Override
8.             public void wifiConfigSent() {}
9.             @Override
10.            public void wifiConfigFailed(Exception e) {}
11.            @Override
12.            public void wifiConfigApplied() {}
13.            @Override
14.            public void wifiConfigApplyFailed(Exception e) {}
15.            @Override
```



```

16.     public void provisioningFailedFromDevice(final ESPConstants.
17.         Provision FailureReason failureReason) {}
18.     @Override
19.     public void deviceProvisioningSuccess() {
20.         // Provisioning succeeded.
21.     }
22.     @Override
23.     public void onProvisioningFailed(Exception e) {}
24. });
25. }

```

Подготовка устройств в iOS

Исходный код

Исходный код подготовки в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ES-PRainMaker/ESPRainMaker/Interface/Provision/SuccessViewController.swift.

```

1. espDevice.provision(ssid: ssid, passphrase: passphrase) { status in
2.     switch status {
3.     case .success:
4.         //Provisioning succeeded.
5.     case let .failure(error):
6.         switch error {
7.             case .configurationError:
8.             case .sessionError:
9.             case .wifiStatusDisconnected:
10.            default:
11.            }
12.        case .configApplied:
13.        }
14.    }

```

После подготовки устройства мы готовы разработать приложение для смартфона с функцией управления устройством.

10.6. Разработка управления устройствами

В разделе 10.5 мы узнали, как подготовить и активировать устройства. В этом разделе мы приступим к привязке устройства к облачной учетной записи и сможем им управлять.

10.6.1. Привязка устройств к облачным учетным записям

API для привязки устройств к учетным записям показан на рис. 10.26 и доступен по адресу: <https://swaggerapis.rainmaker.espressif.com/#/User%20Node%20Association/addRemoveUser-NodeMapping>.

PUT `/{version}/user/nodes/mapping` Add or Remove the User Node mapping

Рисунок 10.26. API для привязки устройств

Для привязки устройства к учетной записи используйте секретный ключ, сгенерированный в разделе 10.5.3, ID устройства (`node_id`) и идентификатор операции `operation`.

```
PUT /v1/user/nodes/mapping
Content-Type: application/json
Authorization: $access_token

{
  "node_id": "$node_id",
  "secret_key": "$secretKey",
  "operation": "add"
}
```

Привязка устройств в Android

Исходный код

Исходный код привязки устройств в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiManager.java.

```
1. public void addNode(final String nodeId,
2.     String secretKey,
3.     final ApiResponseListener listener) {
4.     DeviceOperationRequest req = new DeviceOperationRequest();
5.     req.setNodeId(nodeId);
6.     req.setSecretKey(secretKey);
7.     req.setOperation(AppConstants.KEY_OPERATION_ADD);
8.
9.     apiInterface.addNode(AppConstants.URL_USER_NODE_MAPPING, accessToken,
10.        req).enqueue(new Callback<ResponseBody>() {
11.
12.         @Override
13.         public void onResponse(Call<ResponseBody> call,
14.             Response<ResponseBody> response) {
15.             //Code Omitted
16.         }
17.         @Override
18.         public void onFailure(Call<ResponseBody> call, Throwable t) {
19.         }
20.     });
21. }
```

Привязка устройств в iOS

Исходный код

Исходный код привязки устройств в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/Interface/Provision/SuccessViewController.swift.


```

1. @objc func sendRequestToAddDevice() {
2.     let parameters = ["user_id": User.shared.userInfo.userID,
3.                       "node_id":
4.                         User.shared.currentAssociationInfo!.nodeID,
5.                       "secret_key": User.shared.currentAssociationInfo!.uuid,
6.                       "operation": "add"]
7.     NetworkManager.shared.addDeviceToUser(parameter: parameters as!
8.                                           [String: String]) { requestID, error in
9.         if error != nil, self.count > 0 {
10.            self.count = self.count - 1
11.            DispatchQueue.main.asyncAfter(deadline: .now()) {
12.                self.perform(#selector(self.sendRequestToAddDevice),
13.                             with:nil,
14.                             afterDelay: 5.0)
15.            }
16.        } else {
17.            if let requestid = requestID {
18.                self.step3Indicator.stopAnimating()
19.                self.step3Image.image = UIImage(named: "checkbox_checked")
20.                self.step3Image.isHidden = false
21.                self.step4ConfirmNodeAssociation(requestID: requestid)
22.            } else {
23.                self.step3FailedWithMessage(message: error?.description ??
24.                                             "Unrecognized error. Please check your internet.")
25.            }
26.        }
27.    }
28. }

```

Как только устройство будет привязано к учетной записи, пользователи смогут инициировать запрос на удаленное общение.

10.6.2. Получение списка устройств

Когда пользователи привяжут все устройства к учетной записи, приложение для смартфона покажет их список. В верхней части интерфейса есть несколько переключателей для групп устройств. По умолчанию все устройства появятся в группе **All Devices** (Все устройства). Если пользователи хотят создать разные группы устройств, они могут щелкнуть значок  справа и увидеть варианты управления и создания групп.

Устройства, выделенные серым цветом, указывают на то, что питание устройства выключено или оно находится в автономном режиме, а выделенные устройства указывают на доступные онлайн. Карточка устройства содержит значок типа устройства, имя устройства, время автономной работы и значок тумблера включения. На рис. 10.27 показан пример списка всех устройств, привязанных к данной учетной записи.

API для получения списка всех привязанных устройств показан на рис. 10.28, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/User%20Node%20Association/getUserNodeMappingRequestStatus>.

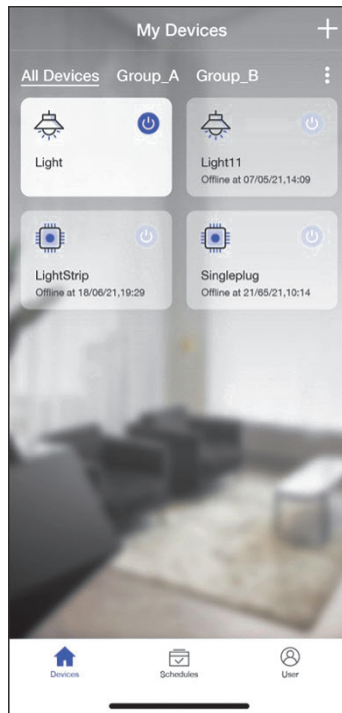


Рисунок 10.27. Интерфейс, показывающий список всех привязанных устройств

GET `/{version}/user/nodes/mapping` Get the status of User Node mapping request

Рисунок 10.28. API для получения привязанных устройств

```
GET /v1/user/nodes?node_details=true
Authorization: $access_token
```

В ответ на запрос сервер возвращает:

```
{
  "nodes": "[ nodeid1, ... ]",
  "node_details": [
    {
      "id": "nodeid1",
      "role": "primary",
      "status": {
        "connectivity": {
          "connected": true,
          "timestamp": 1584698464101
        }
      }
    },
    "config": {
      "node_id": "nodeid1",
      "config_version": "config_version",
    }
  ]
}
```

```

        "devices": [
            {}
        ],
        "info": {
            "fw_version": "fw_version",
            "name": "node_name",
            "type": "node_type"
        }
    },
    "params": {
        "Light": {
            "brightness": 0,
            "output": true
        },
        "Switch": {
            "output": true
        }
    }
}
],
"next_id": "nodeid1",
"total": 5
}

```

Среди этих возвращаемых полей узлы `nodes` представляют собой массив идентификаторов всех устройств; `node_details` – это подробная информация об устройствах, включая `id` (уникальный идентификатор), `role`, `status` (статус соединения), `config` (конфигурация), `params` (свойства устройства) и т. д.; `total` – это количество устройств; эта величина возвращается, когда информация об устройстве распространяется между страницами приложения.

Получение информации об устройстве в Android

Исходный код

Исходный код получения информации об устройстве в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiManager.java.

```

1. private void getNodesFromCloud(final String startId,
2.                               final ApiResponseListener listener) {
3.
4.     Log.d(TAG, "Get Nodes from Cloud with start id : " + startId);
5.     apiInterface.getNodes(AppConstants.URL_USER_NODES_DETAILS,
6.                           accessToken,
7.                           startId).enqueue(new Callback<ResponseBody>() {
8.         @Override
9.         public void onResponse(Call<ResponseBody> call,
10.                               Response<ResponseBody> response) {
11.             //Code Omitted
12.         }
13.         @Override

```

```

14.     public void onFailure(Call<ResponseBody> call, Throwable t) {
15.         t.printStackTrace();
16.         listener.onNetworkFailure(new Exception(t));
17.     }
18. });
19. }

```

Получение информации об устройстве в iOS

Исходный код

Исходный код получения информации об устройстве в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/AWSCognito/ESPAPIManager.swift.

```

1. func getNodes(partialList: [Node]? = nil, nextNodeID: String? = nil,
2.     completionHandler: @escaping ([Node]?, ESPNetworkError?) -> Void) {
3.     let sessionWorker = ESPEExtendUserSessionWorker()
4.     sessionWorker.checkUserSession() { accessToken, error in
5.         if let token = accessToken {
6.             let headers: HTTPHeaders = ["Content-Type": "application/json",
7.                 "Authorization": token]
8.             var url=Constants.getNodes + "?node_details=true&num_records=10"
9.             if nextNodeID != nil {
10.                 url += "&start_id=" + nextNodeID!
11.             }
12.             self.session.request(url, method: .get,
13.                 parameters: nil,
14.                 encoding: JSONEncoding.default,
15.                 headers: headers).responseJSON { response in
16.                 //Code Omitted
17.             }
18.         } else {
19.             if self.validatedRefreshToken(error: error) {
20.                 completionHandler(nil, .emptyToken)
21.             }
22.         }
23.     }
24. }

```

10.6.3. Получение статуса устройства

Нажав на определенное устройство в списке устройств, пользователи могут перейти к его интерфейсу управления, отображающему различную информацию в зависимости от типа устройства.

В этом разделе в качестве примера рассматривается интерфейс управления лампочками, содержащий название лампы, состояние включения/выключения питания, яркость, оттенок и насыщенность. Эта информация была получена ранее при получении списка привязанных устройств в разделе 10.6.2. Интерфейс управления лампочками показан на рис. 10.29.

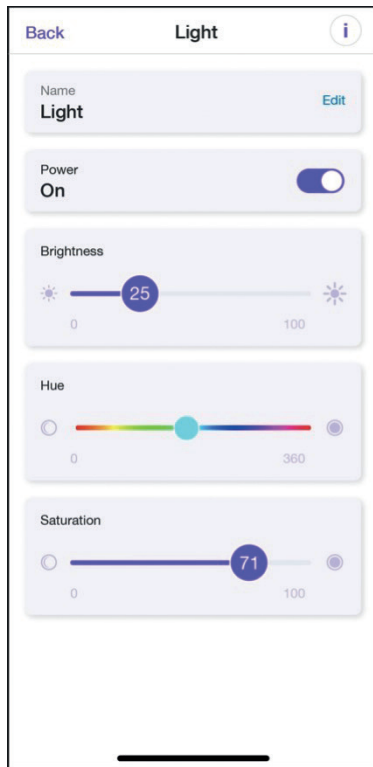


Рисунок 10.29. Интерфейс управления лампой

Учитывая, что одним устройством могут управлять разные пользователи, нам следует в приложении для смартфона регулярно обновлять информацию об устройстве.

API для получения статуса устройства показан на рис. 10.30, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/Node%20Parameter%20Operations/getnodestate>.



Рисунок 10.30. API для получения статуса устройства

```
GET /v1/user/nodes/params?node_id=string
Authorization: $access token
```

В ответ на запрос «get status» сервер возвращает:

```
{
  "Light": {
    "brightness": 0,
    "output": true
  },
}
```

```
"Switch": {
    "output": true
}
```

Среди возвращаемых полей Light представляет яркость устройства, brightness – ее конкретное значение; Switch – состояние включения/выключения устройства.

Получение статуса устройства в Android

Исходный код

Исходный код получения статуса устройства в Android см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/expressif/cloudapi/ApiManager.java.

```
1. public void getParamsValues(final String nodeId, final ApiResponseListener
2.     listener) {
3.     apiInterface.getParamValue(AppConstants.URL_USER_NODES_PARAMS,
4.         accessToken, nodeId).enqueue(new Callback<ResponseBody>() {
5.         @Override
6.         public void onResponse(Call<ResponseBody> call,
7.             Response<ResponseBody> response) {
8.             //Code Omitted
9.         }
10.        @Override
11.        public void onFailure(Call<ResponseBody> call, Throwable t) {
12.            t.printStackTrace();
13.            listener.onNetworkFailure(new Exception(t));
14.        }
15.    });
16. }
```

Получение статуса устройства в iOS

Исходный код

Исходный код получения статуса устройства в iOS см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/AWSCognito/ESPAPIManager.swift.

```
1. func getDeviceParams(device: Device, completionHandler:
2.     @escaping (ESPNetworkError?) -> Void) {
3.
4.     ESPExtendUserSessionWorker().checkUserSession(){accessToken, error in
5.     if let token = accessToken {
6.         let headers: HTTPHeaders = ["Content-Type": "application/json",
7.             "Authorization": token]
8.         let url = Constants.setParam + "?node_id=" +
9.             (device.node?.node_id ?? "")
10.        self.session.request(url, method: .get, parameters: nil,
11.            encoding: JSONEncoding.default, headers:
```



```

12.         headers).responseJSON { response in
13.             //Code Omitted
14.         }
15.     } else {
16.         if self.validatedRefreshToken(error: error) {
17.             completionHandler(.emptyToken)
18.         }
19.     }
20. }
21. }

```

10.6.4. Изменение статуса устройства

Приложение позволяет пользователям изменять имя устройства, состояние включения/выключения питания, яркость, оттенок и насыщенность. В этом разделе будет объяснено, как реализовать данную функцию на примере изменения яркости и состояния включения/выключения.

API для изменения статуса устройства показан на рис. 10.31, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/Node%20Parameter%20Operations/updatenodestate>.



Рисунок 10.31. API для изменения статуса устройства

```

1. PUT /v1/user/nodes/params
2. Authorization: $access_token
3.
4. [
5.   {
6.     "node_id": "string",
7.     "payload": {
8.       "Light": {
9.         "brightness": 100,
10.        "output": true
11.      },
12.      "Switch": {
13.        "output": true
14.      }
15.    }
16.  }
17. ]

```

Среди возвращаемых полей `node_id` представляет уникальный идентификатор устройства; `Light` представляет яркость устройства, `brightness` – ее конкретное значение; `Switch` указывает состояние включения/выключения устройства.

Изменение статуса устройства в Android

Исходный код

Исходный код изменения статуса устройства в Android см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiManager.java.

```

1. public void updateParamValue(final String nodeId,
2.                             JSONObject body,
3.                             final ApiResponseListener listener) {
4.
5.     apiInterface.updateParamValue(AppConstants.URL_USER_NODES_PARAMS,
6.                                   accessToken,
7.                                   nodeId,
8.                                   body).enqueue(new Callback<ResponseBody>(){
9.         @Override
10.        public void onResponse(Call<ResponseBody> call,
11.                               Response<ResponseBody> response) {
12. //Code Omitted
13.     }
14.
15.     @Override
16.     public void onFailure(Call<ResponseBody> call, Throwable t) {
17.         t.printStackTrace();
18.         listener.onNetworkFailure(new Exception(t));
19.     }
20. });
21. }

```

Изменение статуса устройства в iOS

Исходный код

Исходный код получения статуса устройства в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/AWSCognito/ESPAPIManager.swift.

```

1. func setDeviceParam(nodeID: String?, parameter: [String: Any],
2.                    completionHandler: ((ESPCloudResponseStatus) ->
3.                    Void)? = nil) {
4.     NotificationCenter.default.post(Notification(name: Notification.Name
5.                                                   (Constants.paramUpdateNotification)))
6.     if let nodeid = nodeID {
7.         ESPExtendUserSessionWorker().checkUserSession(){accessToken, error in
8.             if let token = accessToken {
9.                 let url = Constants.setParam + "?nodeid=" + nodeid
10.                let headers: HTTPHeaders = ["Content-Type":
11.                                             "application/json",
12.                                             "Authorization": token]
13.                self.session.request(url, method: .put,
14.                                     parameters:
15.                                     parameter,
16.                                     encoding: ESPCustomJsonEncoder.default,
17.                                     headers: headers).responseJSON {response in
18. //Code Omitted
19.             }
20.         } else {
21.             let _ = self.validatedRefreshToken(error: error)
22.         }
23.     }
24. }
25. }

```

Приведенный код позволяет пользователям изменять статус определенного устройства. Таким образом, у нас есть реализованные функции регистрации аккаунта, входа в аккаунт, сканирования устройства, подключения и подготовки устройств, привязки устройств и удаленного управления в RainMaker.

10.7. Разработка расписания и пользовательского центра

После реализации основных функциональных модулей также необходимо разработать модуль планировщика и модуль пользовательского центра в соответствии с функциональными требованиями. Обычно для полноценного приложения для смартфона необходимы оба этих модуля. В этом разделе будет описана их разработка.

10.7.1. Реализация функции планирования

Интерфейс планирования в основном используется для отображения, создания и редактирования событий расписания. Список на странице отображает такую информацию, как название, время, дата и повторяющийся шаблон запланированных событий. У каждого события есть выключатель. Интерфейс планирования показан на рис. 10.32.

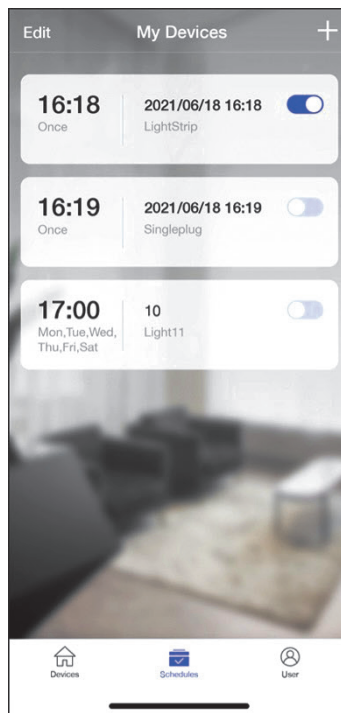


Рисунок 10.32. Интерфейс планирования

Данные, связанные с планированием, хранятся в локальной базе данных приложения для смартфона и могут дополняться, удаляться, изменяться и запрашиваться. Код см. ниже.

Реализация планирования в Android

Исходный код

Исходный код реализации планирования в Android см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/expressif/ui/activities/AddScheduleActivity.java.

В коде сохранения данных расписания, приведенном далее, KEY_OPERATION описывает событие расписания, KEY_ID – уникальный идентификатор, KEY_NAME – название расписания, KEY_DAYS относится к запланированной дате, KEY_MINUTES – к запланированному времени, KEY_TRIGGERS – к шаблону повторений (с понедельника по пятницу).

```
1. private void saveSchedule() {
2.     JSONObject scheduleJson = new JSONObject();
3.     scheduleJson.addProperty(AppConstants.KEY_OPERATION, "");
4.
5.     //Schedule JSON
6.     scheduleJson.addProperty(AppConstants.KEY_ID, "");
7.     scheduleJson.addProperty(AppConstants.KEY_NAME, "");
8.
9.     JSONObject jsonTrigger = new JSONObject();
10.    jsonTrigger.addProperty(AppConstants.KEY_DAYS, "");
11.    jsonTrigger.addProperty(AppConstants.KEY_MINUTES, "");
12.
13.    JSONArray triggerArr = new JSONArray();
14.    triggerArr.add(jsonTrigger);
15.    scheduleJson.add(AppConstants.KEY_TRIGGERS, triggerArr);
16.
17.    prepareJson();
18.    //Code Omitted
19. }
```

Код для обновления расписания выглядит следующим образом:

```
1. @SuppressWarnings("CheckResult")
2. public void updateSchedules(final HashMap<String, JSONObject> map,
3.                             final ApiResponseListener listener) {
4.    //Code Omitted
5. }
```

Исходный код

Исходный код обновления расписаний в Android см.: https://github.com/expressif/book-esp32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/expressif/cloudapi/ApiManager.java.

Реализация планирования в iOS

Исходный код

Исходный код реализации планирования в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/Storage/ESPLocalStorageSchedules.swift.

```
1. func saveSchedules(schedules: [String: ESPSchedule]) {
2.     do {
3.         let encoded = try JSONEncoder().encode(schedules)
4.         saveDataInUserDefaults(data: encoded,
5.             key: ESPLocalStorageKeys.scheduleDetails)
6.     } catch {
7.         print(error)
8.     }
9. }
```

Код для обновления расписания выглядит следующим образом:

```
1. func fetchSchedules() -> [String: ESPSchedule] {
2.     var scheduleList: [String: ESPSchedule] = [: ]
3.     do {
4.         if let scheduleData = getDataFromSharedUserDefaults(key:
5.             ESPLocalStorageKeys.scheduleDetails) {
6.             scheduleList = try JSONDecoder().decode([String:
7.                 ESPSchedule].self, from: scheduleData)
8.         }
9.         return scheduleList
10.    } catch {
11.        print(error)
12.        return scheduleList
13.    }
14. }
```

10.7.2. Реализация центра пользователей

Модуль пользовательского центра в основном включает в себя такие функции, как профиль пользователя, уведомление, изменение пароля, политики конфиденциальности, условия использования, документация, голосовые услуги и выход из системы. Центральный интерфейс центра показан на рис. 10.33.

Среди этих функций смена пароля и выход из системы должны быть реализованы путем вызова облачных API. В этом разделе мы рассмотрим смену пароля в качестве примера реализации функции пользовательского центра. Интерфейс смены пароля показан на рис. 10.34.

```
1. PUT /v1/password
2. Authorization: $access_token
3. {
4.     "password": "password",
5.     "newpassword": "newpassword"
6. }
```

В приведенном коде `password` относится к старому паролю, который требуется облаку для изменения; `newpassword` относится к новому паролю. Как только пароль будет изменен, новый пароль должен вступить в силу, а старый станет недействительным.

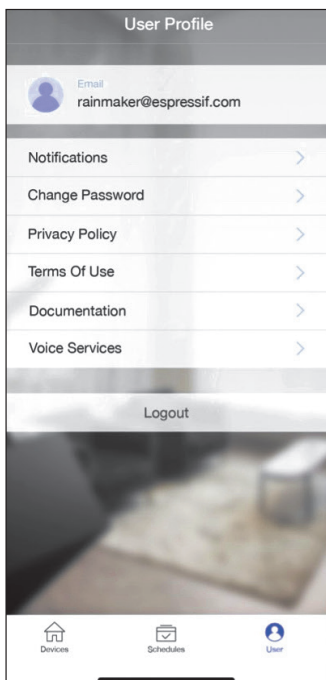


Рисунок 10.33. Интерфейс пользовательского центра

```
PUT /{version}/password Handle password change request from the user
```

Рисунок 10.34. API для смены пароля

В ответ на запрос сервер возвращает:

```
1. {
2.   "status": "success",
3.   "description": "Success description"
4. }
```

Среди возвращаемых полей `status` указывает статус смены пароля; `description` указывает описание запроса на изменение.

Смена пароля в Android

Исходный код

Исходный код смены пароля в Android см.: https://github.com/espressif/book-es-p32c3-iot-projects/blob/main/phone_app/app_android/app/src/main/java/com/espressif/cloudapi/ApiManager.java.

```

1. Public void changePassword(String oldPassword, String newPassword,
2.                             final ApiResponseListener listener) {
3.
4.     JsonObject body = new JsonObject();
5.     body.addProperty(AppConstants.KEY_PASSWORD, oldPassword);
6.     body.addProperty(AppConstants.KEY_NEW_PASSWORD, newPassword);
7.
8.     apiInterface.changePassword(AppConstants.URL_CHANGE_PASSWORD,
9.                                 accessToken,
10.                                body).enqueue(new Callback<ResponseBody>() {
11.
12.         @Override
13.         public void onResponse(Call<ResponseBody> call,
14.                                Response<ResponseBody> response) {
15. //Code Omitted
16.     }
17.
18.     @Override
19.     public void onFailure(Call<ResponseBody> call, Throwable t) {
20.         t.printStackTrace();
21.         listener.onNetworkFailure(new RuntimeException("Failed to
22.             change password"));
23.     }
24. });
25. }

```

Смена пароля в iOS

Исходный код

Исходный код смены пароля в iOS см.: https://github.com/espressif/book-esp32c3-iot-projects/blob/cf25c67fbcedc44394fd7f90637b745d659f80ff/phone_app/app_ios/ESPRainMaker/ESPRainMaker/UserManagement/Interactors/ESPChangePasswordService.swift.

```

1. func changePassword(oldPassword: String, newPassword: String) {
2.     sessionWorker.checkUserSession() { accessToken, sessionError in
3.         if let token = accessToken {
4.             self.apiWorker.callAPI(endPoint: .changePassword(url: self.url,
5.                                                         old: oldPassword, new: newPassword,
6.                                                         accessToken: token), encoding:
7.                                                         JSONEncoding.default) { data, error in
8.                 self.apiParser.parseResponse(data, withError:
9.                                                 error) { umError in
10.                    self.presenter?.passwordChanged(withError: umError)
11.                }
12.            }
13.        } else {
14.            if !self.apiParser.isRefreshTokenValid(serverError:sessionError) {
15.                if let error = sessionError {
16.                    self.noRefreshSignOutUser(error: error)
17.                }
18.            } else {
19.                self.presenter?.passwordChanged(withError: sessionError)
20.            }
21.        }
22.    }
23. }

```

10.7.3. Дополнительные облачные API

Помимо API, подробно описанных в предыдущих разделах, RainMaker также предоставляет некоторые другие API. Давайте кратко рассмотрим их.

Совместное использование устройств с другими пользователями

API для совместного использования устройств с другими пользователями показан на рис. 10.35, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/User%20Node%20Association/addUserNodeSharingRequests>.

PUT `/{{version}}/user/nodes/sharing/requests` This API creates the sharing of Nodes between users.

Рисунок 10.35. API для обмена устройствами с другими пользователями

Получение онлайн/офлайн статуса устройства

API для получения статуса онлайн/офлайн устройства показан на рис. 10.36 и может быть найден по адресу: <https://swaggerapis.rainmaker.espressif.com/#/User%20Node%20Association/getNodeStatus>.

GET `/{{version}}/user/nodes/status` Get the online or offline status for the node

Рисунок 10.36. API для получения статуса онлайн/офлайн устройства

Создание групп устройств

API для создания групп устройств показан на рис. 10.37, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/Device%20grouping/usercreatedevicegroup>.

POST `/{{version}}/user/node_group` Create the user device group

Рисунок 10.37. API для создания групп устройств

Добавление устройства в группу

API для добавления устройства в группу показан на рис. 10.38, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/Device%20grouping/userupdatedevicegroup>.

PUT `/{{version}}/user/node_group` Update user device group

Рисунок 10.38. API для добавления устройства в группу

Удаление группы устройств

API для удаления групп устройств показан на рис. 10.39, его можно найти по адресу: <https://swaggerapis.rainmaker.espressif.com/#/Device%20grouping/userdeletedevicegroup>.



Рисунок 10.39. API для удаления групп устройств

Конечно, RainMaker может гораздо больше, чем мы здесь представили. Чтобы узнать о других интересных функциях, вы можете подробно изучить документацию API.

10.8. Резюме

В этой главе в основном описано, как разработать приложение для смартфона. В начале мы представили технологии разработки приложений для смартфонов и порядок создания нового проекта приложения, чтобы дать вам общее представление о всем процессе разработки. Затем проанализировали функциональные требования к каждому модулю, а именно управление пользователями, подготовку устройств, управление устройствами, планирование и пользовательский центр. Наконец, мы подробно описали процесс разработки приложения для смартфона с исходным кодом и показали, как реализовать подготовку и управление устройствами в приложении для смартфона.

Глава 11

Обновление встроенного ПО и управление версиями

Обновление встроенного программного обеспечения устройств интернета вещей часто осуществляется согласно технологии OTA (Over-the-Air⁴¹). OTA предоставляет безопасные и надежные средства для устранения уязвимостей встроенного ПО, внедрения новых функций и оптимизации производительности продукта, тем самым повышая удобство работы конечных пользователей. В настоящее время OTA стала стандартной функцией при серийном производстве продукции.

Маркировка встроенного ПО различными версиями на основе их соответствующих функциональных возможностей является надежным и эффективным средством управления встроенным ПО. Стандартизированные методы проверки версий могут облегчить управление версиями, помочь в устранении неполадок и обеспечить эффективное отслеживание обновлений, что в конечном итоге приведет к более эффективному процессу обновлений встроенного ПО через OTA.

ESP-IDF предоставляет пример OTA и различных методов управления версиями встроенного ПО. В этой главе будут рассмотрены эти темы и продемонстрировано, как добиться OTA-обновлений для умного светильника с помощью ESP RainMaker.

11.1. Обновление прошивки

Механизм OTA позволяет устройству получать новую прошивку во время нормальной работы и установить ее в неактивный в данный момент раздел приложения. После проверки действительности прошивки устройство переключается на работу на новой прошивке. Основные этапы OTA показаны на рис. 11.1.

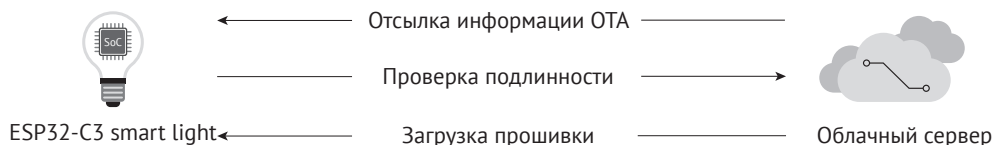


Рисунок 11.1. Основные этапы OTA

⁴¹ О технологии OTA см. сноску на стр. 42.

На рис. 11.1 показаны основные этапы OTA:

- (1) облачный сервер передает информацию OTA на устройство;
- (2) устройство проверяет подлинность облачного сервера и загружает прошивку с доверенного облачного сервера;
- (3) устройство решает, выполнять ли OTA в соответствии с информацией о версии прошивки. Если оно решает выполнить OTA, прошивка запрашивается и записывается во флеш-память. После успешного завершения проверки система переходит к запуску на новой прошивке.

Согласно основным шагам, перечисленным выше, можно сказать проще, что процесс OTA – это процесс получения, записи, проверки и переключения прошивки. Прежде чем продолжить изучение механизма OTA, мы сначала представим таблицу разделов и процесс запуска прошивки.

11.1.1. Обзор таблицы разделов

Таблица разделов в ESP-IDF относится к описательным файлам, которые на уровне пользователя делят флеш-память на определенные функциональные области. В этой книге в качестве примера используется расширенный [advanced_https_ota](#), сокращенно именуемый примером обновления OTA. В данном примере файл [partitions_two_ota.csv](#) используется компонентом `partition_table` по умолчанию. Следующий текст представляет собой обзор заготовки таблицы разделов из файла `partitions_two_ota.csv`:

```
# Name, Type, SubType, Offset, Size, Flags
# Note: if you have increased the bootloader size, make sure to update the offsets to
# avoid overlap
nvs, data, nvs, , 0x4000,
otadata, data, ota, , 0x2000,
phy_init, data, phy, , 0x1000,
factory, app, factory, , 1M,
ota_0, app, ota_0, , 1M,
ota_1, app, ota_1, , 1M,
```

Согласно приведенному обзору, каждая запись в таблице разделов состоит из имени (Name), типа (Type), подтипа (SubType), смещения (Offset), размера (Size) и флагов (Flags).

- Поле имени Name используется для идентификации имени и не должно превышать 16 байт.
- В поле типа Type можно указать либо приложение, либо данные, либо число от 0 до 254 (или соответствующее шестнадцатеричное число от 0x00 до 0xFE). В основном используется для обозначения того, что сохраненный контент – это встроенное ПО или данные приложения.
- Длина поля подтипа SubType составляет 8 бит, а конкретное содержание маркировки связано с полем типа:
 - если тип определен как приложение, подтип можно указать как `factory(0x00)`, `ota_0 (0x10)`, ..., `ota_15(0x1F)` или `test(0x20)`;
 - если тип определен как данные, подтип можно указать как `ota(0x00)`, `phy(0x01)`, `nvs(0x02)`, `nvs_keys(0x04)` или определенный подтип для других компонентов.

- Поля смещение *Offset* и размер *Size* используются для определения конкретной области.
- Поле флагов *Flags* используется для обозначения того, включено ли шифрование.

Без заполнения какого-либо значения в поле *Offset* таблица разделов остается действительной. Это связано с тем, что позиция первой записи в таблице разделов определена, поэтому адрес последующей записи может быть вычислен по полю размера (*Size*) предыдущей записи. Если адреса каждой записи в таблице разделов не являются непрерывными, необходимо использовать поле смещения *Offset* для обозначения начального адреса каждой записи. Для наглядности пример таблицы разделов в этой книге преобразован в рисунок (см. рис. 11.2).

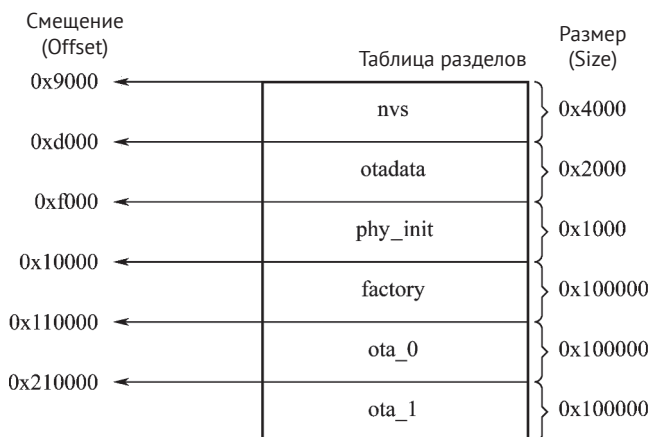


Рисунок 11.2. Схема таблицы разделов

На рис. 11.2 начальный адрес первой записи в таблице разделов равен 0x9000, то есть поле смещения записи (*Offset*), имя которой в файле *partitions_two_ota.csv* обозначено *nvs*, имеет значение 0x9000, а размер этой записи – 0x4000. По указанным выше правилам расчет смещения следующей записи равно $0x9000 + 0x4000 = 0xd000$. Последовательно рассчитанное смещение последней записи *ota_1* должно быть равно 0x210000.

Таблица разделов *parts_two_ota.csv* разделена на шесть областей: три раздела данных *nvs*, *otadata* и *phy_init* используются для хранения данных энергонезависимого хранилища NVS, данных OTA и данных инициализации РНУ⁴² соответственно, и три раздела приложений используются для хранения трех разных прошивок. Как видно из основных этапов OTA, для выполнения OTA необходимы как минимум два OTA-раздела приложений: [Type (*app*), SubType (*ota_0/ota_1*)] и один раздел данных OTA [Type (*data*), SubType (*ota*)]. Он также может включать в себя дополнительный раздел приложения, который является разделом заводской версии приложения: [Type (*app*), SubType (*factory*)].

⁴² РНУ (сокращение от *physical*) – протокол физического уровня сетевой модели, определяющий передачу пакетов через физическую среду.

- Раздел данных OTA используется для хранения информации о выбранном в данный момент разделе приложения OTA. После проведения первого OTA раздел данных OTA будет обновлен, чтобы указать, какой раздел приложения OTA загрузить следующим. Размер раздела данных OTA должен быть установлен в значение 0x2000, чтобы предотвратить проблемы, вызванные сбоем питания во время записи. Эти два сектора стираются и записываются соответствующими данными по отдельности. Если есть несоответствие, значение счетчика будет использоваться для определения сектора с последними верными данными.
- Раздел приложений используется для хранения встроенного ПО. Раздел заводских приложений – это раздел приложения по умолчанию. Если нет раздела данных OTA или раздел данных OTA испорчен, сначала будет использоваться прошивка заводского раздела приложений (если он существует), с последующей прошивкой раздела данных OTA. OTA никогда не будет обновлять содержимое раздела заводского приложения.

11.1.2. Процесс загрузки прошивки

В разделе 11.1.1 мы рассказали, что начальный адрес первой записи в разделе имеет размер 0x9000. Но почему начальный адрес не 0x0? И почему это именно 0x9000? Чтобы ответить на эти вопросы, давайте сначала посмотрим на рис. 11.3, на котором представлено конкретное содержимое, хранящееся во флеш-памяти на 4 МБ.

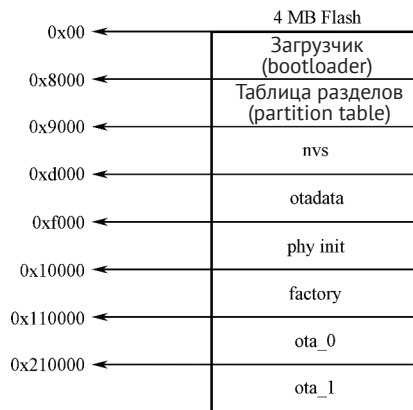


Рисунок 11.3. Конкретное содержимое флеш-памяти объемом 4 МБ

Как видно по рис. 11.3, флеш-память разделена на восемь областей: по адресу 0x00 размещен загрузчик, по адресу 0x8000 – таблица разделов, а последние шесть областей, начиная с 0x9000, – это область, занятая таблицей разделов, с которой вы, возможно, уже хорошо знакомы. Теперь мы можем ответить на первый вопрос. Причина, по которой 0x0 не является начальным адресом, заключается в том, что он хранит загрузчик (не каждый адрес 0x0 флеш-памяти любого чипа хранит загрузчик. Так, микросхемы серии ESP32 хранят загрузчик по адресу 0x1000), который используется для закачки и загрузки раздела приложения. В программном обеспечении чипов Espressif загрузчик называется

вторичным загрузчиком, который в основном повышает гибкость использования разделов флеш-памяти и облегчает реализацию функций шифрования данных, безопасной загрузки и OTA.

Вторичный загрузчик по умолчанию загружает таблицу разделов по адресу смещения флеш-памяти 0x8000. Размер таблицы разделов – 0x1000. Вторичный загрузчик извлечет данные для раздела заводского приложения и раздела данных OTA из таблицы разделов и определит, какой раздел следует загрузить, запросив раздел данных OTA.

Процесс от включения питания до запуска функции `app_main()` на ESP32-C3 можно разделить на три шага:

- (1) начальная загрузка выполняется основным загрузчиком, который хранится в ПЗУ ESP32-C3. После сброса микросхемы ЦП немедленно начинает работать, чтобы определить режим загрузки и выполнить соответствующие операции. Вторичный загрузчик в этом случае загружается в ОЗУ по адресу смещения 0x0 флеш-памяти;
- (2) начальная загрузка выполняется вторичным загрузчиком. Вторичный загрузчик сначала загрузит таблицу разделов из флеш-памяти, а затем запросит раздел данных OTA для выбора прошивки из конкретного раздела приложения для загрузки. Когда все данные обработаны, вторичный загрузчик проверит целостность прошивки и извлечет адрес входа из заголовка бинарного файла прошивки, чтобы перейти к нему для выполнения прошивки. Прошивка в разделе приложений имеет определенные статусы, влияющие на ее запуск. Эти статусы сохраняются в разделе данных OTA и определяются в ESP-IDF набором определенных переменных (`esp_ota_img_states_t`):
 - новая прошивка: определяется `ESP_OTA_IMG_NEW`, указывает, загружается ли прошивка загрузчиком в первый раз. Этот статус будет изменен на `ESP_OTA_IMG_PENDING_VERIFY` в загрузчике;
 - прошивка, ожидающая проверки: определяется `ESP_OTA_IMG_PENDING_VERIFY`, указывает, действительна ли прошивка. Если прошивка остается в том же состоянии, при второй загрузке статус изменится на `ESP_OTA_IMG_ABORTED`;
 - действительная прошивка: определяется `ESP_OTA_IMG_VALID`, указывает, что прошивка функционирует нормально. После того как отмечен этот статус, прошивку можно загружать без ограничений;
 - неверная прошивка: определяется `ESP_OTA_IMG_INVALID`, указывает, что прошивка не функционирует должным образом. После пометки этим статусом прошивка не может быть перезагружена;
 - прерванная прошивка: определяется `ESP_OTA_IMG_ABORTED`, указывает на наличие исключений в прошивке. После пометки этим статусом прошивка не может быть перезагружена;
 - неопределенная прошивка: определяется `ESP_OTA_IMG_UNDEFINED`. После пометки этим статусом прошивка может загружаться без ограничений;
- (3) фаза запуска приложения. После начальной загрузки, выполненной вторичным загрузчиком, наступает этап запуска прошивки приложения, который включает в себя все процессы от запуска приложения до создания и выполнения функции `app_main()`.

Этот этап можно разделить на три части:

- инициализация оборудования и основных портов;
- инициализация программных сервисов и системы FreeRTOS;
- запуск основной задачи и вызов функции `app_main()`.

11.1.3. Обзор механизма ОТА

Для устройств интернета вещей первым шагом ОТА является загрузка новой прошивки. Есть несколько способов сделать это, среди которых использование Wi-Fi – один из самых простых и удобных. Устройства IoT могут подключаться к маршрутизатору через Wi-Fi, а затем подключаться к ОТА-серверу для загрузки прошивки через протоколы прикладного уровня (например, HTTP, FTP). Здесь мы будем использовать расширенный протокол *advanced_https_ota* в качестве примера, чтобы представить способ загрузки прошивки через HTTPS. В примере ОТА выполняется с помощью компонента `esp_https_ota`, который использует HTTPS в качестве протокола загрузки. Следующий код взят из расширенного *advanced_https_ota_example.c*, некоторые части которого опущены для ясности:

```
1. static esp_err_t _http_client_init_cb(esp_http_client_handle_t http_client)
2. {
3.     esp_err_t err = ESP_OK;
4.     //Set HTTPS custom header
5.     //err = esp_http_client_set_header(http_client, "Custom-Header", "Value");
6.     return err;
7. }
8.
9. void advanced_ota_example_task(void *pvParameter)
10. {
11.     ESP_LOGI(TAG, "Starting Advanced OTA example");
12.
13.     //1. Configure the HTTPS connection and set the esp_https_ota parameter
14.     esp_err_t ota_finish_err = ESP_OK;
15.     esp_http_client_config_t config = {
16.         .url = CONFIG_EXAMPLE_FIRMWARE_UPGRADE_URL,
17.         .cert_pem = (char *)server_cert_pem_start,
18.         .timeout_ms = CONFIG_EXAMPLE_OTA_RECV_TIMEOUT,
19.         .keep_alive_enable = true,
20.     };
21.     .....
22.     esp_https_ota_config_t ota_config = {
23.         .http_config = &config,
24.         .http_client_init_cb = _http_client_init_cb, //Register esp_http_client
25.         // Callback function called after initialization
26.     };
27.
28.     //2. Enable OTA via esp_https_ota_begin, returning HTTP/HTTPS results
29.     esp_https_ota_handle_t https_ota_handle = NULL;
30.     esp_err_t err = esp_https_ota_begin(&ota_config, &https_ota_handle);
31.     if (err != ESP_OK) {
32.         ESP_LOGE(TAG, "ESP HTTPS OTA Begin failed");
33.         vTaskDelete(NULL);
34.     }
35.
```

```

36. //3. When connected, the information of new firmware can be obtained
37. //via esp_https_ota_get_img_desc, which can be used for verification
38. esp_app_desc_t app_desc;
39. err = esp_https_ota_get_img_desc(https_ota_handle, &app_desc);
40. if (err != ESP_OK) {
41.     ESP_LOGE(TAG, "esp_https_ota_read_img_desc failed");
42.     goto ota_end;
43. }
44. err = validate_image_header(&app_desc);
45. if (err != ESP_OK) {
46.     ESP_LOGE(TAG, "image header verification failed");
47.     goto ota_end;
48. }
49.
50. //4. Receive and write firmware by iterating esp_https_ota_perform(),
51. //and jump out of the loop when HTTPS is not in receiving mode.
52. while (1) {
53.     err = esp_https_ota_perform(https_ota_handle);
54.     if (err != ESP_ERR_HTTPS_OTA_IN_PROGRESS) {
55.         break;
56.     }
57.     /.
58. ...
59. }
60. //5. Verify the firmware integrity, and call esp_https_ota_finish or
61. //esp_https_ota_abort to release the HTTP/HTTPS connection.
62. //Esp_https_ota_finish will upgrade the OTA data partition
63. //and switch the next boot to the new firmware.
64. if (esp_https_ota_is_complete_data_received(https_ota_handle) != true) {
65.     //OTA firmware not fully received. Users can customise handling options
66.     ESP_LOGE(TAG, "Complete data was not received.");
67. } else {
68.     ota_finish_err = esp_https_ota_finish(https_ota_handle);
69.     if ((err == ESP_OK) && (ota_finish_err == ESP_OK)) {
70.         ESP_LOGI(TAG, "ESP_HTTPS_OTA upgrade successful.Rebooting ..." );
71.         vTaskDelay(1000 / portTICK_PERIOD_MS);
72.         esp_restart();
73.     } else {
74.         if (ota_finish_err == ESP_ERR_OTA_VALIDATE_FAILED) {
75.             ESP_LOGE(TAG, "Image validation failed, image is corrupted");
76.         }
77.         ESP_LOGE(TAG, "ESP_HTTPS_OTA upgrade failed 0x%x",
78.             ota_finish_err);
79.         vTaskDelete(NULL);
80.     }
81. }
82.
83. ota_end:
84.     esp_https_ota_abort(https_ota_handle);
85.     ESP_LOGE(TAG, "ESP_HTTPS_OTA upgrade failed");
86.     vTaskDelete(NULL);
87. }

```

(1) Получение прошивки

Приведенный код загружает прошивку по протоколу HTTPS, операции HTTPS внедрены в ESP-IDF. Вам нужно только настроить структуру esp_

`http_client_config_t`, куда можно передать сертификат и включить протокол TLS для защиты передаваемых данных. После настройки параметров HTTPS-соединения конфигурация должна быть передана в структуру `esp_https_ota_config_t`, предоставляющую функцию обратного вызова для облегчения настройки информации заголовка HTTPS-запроса. Информация заголовка обычно используется для объявления длины и формата данных HTTPS, посланных на сервер. После настройки для подключения вы можете вызвать `esp_https_ota_begin()`. Функция определит результат на основе возвращенного кода состояния HTTPS. После успешного подключения HTTPS последующие вызовы функции `esp_https_ota_perform()` для запроса прошивки могут быть сделаны рекурсивно.

(2) Запись прошивки

Функция `esp_https_ota_perform()` непрерывно отправляет информацию о запросе на сервер и записывает во флеш-память каждый набор данных, возвращаемых сервером. Этот процесс записи реализуется внутри функции через вызов `esp_ota_write()`.

(3) Проверка прошивки

Учитывая ограниченные ресурсы принимающего устройства и не всегда идеальную сетевую среду, чтобы полностью загрузить прошивку, часто бывает необходимо отправлять HTTPS-запросы несколько раз. ESP-IDF предоставляет функцию `esp_https_ota_is_complete_data_received()` для определения того, была ли получена прошивка полностью, через расчет размера прошивки. По окончании загрузки будет вызвана функция `esp_https_ota_finish()` для завершения HTTPS-соединения и освобождения всех занятых ресурсов. Простое сравнение размера встроенного ПО для определения того, является ли оно допустимым, не является оптимальным, нам также необходимо проверить значение SHA-256⁴⁵ прошивки, чтобы убедиться, что загруженная прошивка идентична исходной. Вызов функции `esp_https_ota_finish()` завершит проверку автоматически. Если включена безопасная загрузка, соответствующие проверки также будут выполнены на этом этапе. Для получения более подробной информации о безопасной загрузке, пожалуйста, обратитесь к разделу 13.4.2.

(4) Переключение прошивки

Функция `esp_https_ota_finish()` не только освобождает ресурсы HTTPS и проверяет прошивку, но также автоматически перезаписывает таблицу разделов OTA после успешной проверки. При этом функция обновляет статус этой загруженной прошивки до «неопределенной» (`ESP_OTA_IMG_UNDEFINED`). После завершения подготовительных работ, при перезагрузке с помощью вызова `esp_restart()`, будет загружена новая прошивка. «Неопределенность» прошивки в данном случае означает, что прошивка может быть загружена без ограничений до тех пор, пока не включен режим отката загрузчика.

⁴⁵ SHA-256 (Secure Hash Algorithm 256, безопасный алгоритм хеширования, версия 256) – криптографический алгоритм на основе однонаправленных хеш-функций, применяющийся для проверки аутентичности и целостности передаваемой информации.

11.2. Управление версиями прошивки

Маркировка прошивок с разными функциями как разных версий является важным средством облегчения обслуживания. Для информации о версии ESP-IDF предоставляет ряд маркировочных полей, которые можно использовать при вызове функции отката/антиотката для удовлетворения большей части потребностей при контроле версий.

11.2.1. Маркировка прошивки

Есть четыре редактируемых поля: `secure_version`, `project_version`, `project_name` и `App version`, а также два нередактируемых поля: `idf_ver` и `Compile time and date`.

- **secure_version**: используется для установки безопасной версии чипа. Безопасный номер версии хранится в ячейках eFuse и может отмечать до 16 версий. Способ включения следующий:

```
(Top) → Bootloader config
...
...
[*] Enable app rollback support
[*]   Enable app anti-rollback support
(0)   eFuse secure version of app (NEW)
(16)  Size of the eFuse secure version field (NEW)
...
...
```

- **project_version**: используется для установки версии проекта. Способ включения следующий:

```
(Top) → Application manager
...
...
[*] Get the project version from Kconfig
(1)   Project version
...
...
```

- **project_name**: имя проекта задается в файле `CMakeLists.txt` в директории проекта. Если в качестве примера взять проект `advanced_https_ota`, то способ включения поля будет следующим:

```
1. ...
2. ...
3. include($ENV{IDF_PATH}/tools/cmake/project.cmake)
4. project(advanced_https_ota)
5. ...
6. ...
```

`project(X)` – имя отмеченного проекта.

- **Compile time and date**, а также **idf_ver** (версия ESP-IDF) будут назначены автоматически во время компиляции со следующим выводом в логе:

```
...
...
I (304) cpu_start: Compile time: Mar 14 2022 18 : 44 : 58
I (316) cpu_start: ESP-IDF: v4.3.2
...
...
```

Как поле `secure_version`, так и `project_version` могут использоваться для обозначения информации о версии встроенного ПО, но с разными целями и реализациями. Номер `secure_version` записан в eFuse чипа. Он не может быть изменен после записи, и впоследствии на чип разрешается записывать только более высокие версии прошивки, что позволяет безопасно и эффективно управлять основными обновлениями. Поскольку основные обновления, как правило, связаны с безопасностью, такие номера версий и называются `secure_version` (защищенными версиями). Номер версии проекта `project_version` хранится во флеш-памяти вместе с прошивкой и может быть произвольно изменен во время каждой компиляции. Устройство активно не проверяет эту информацию во время обновлений, и ее использование полностью определяется разработчиком. В практических приложениях для разработки, из-за ограничений в использовании номера `secure_version`, ее обновление обычно определяется как содержащее основные функциональные обновления и исправляющее ошибки безопасности, в то время как обновление номера версии проекта `project_version` используется как функциональное обновление бизнес-уровня.

11.2.2. Откат и защита от отката

Основная цель отката приложения – обеспечить возможность возвращения ПО устройства к предыдущей версии в случае появления исключительных ситуаций после обновления, без нарушений нормальной работы устройства. При включении отката прошивка будет помечена как новая прошивка (`ESP_OTA_IMG_NEW`) после завершения проверки. После перезагрузки загрузчик выберет эту прошивку и повторно пометит ее как ожидающую проверки (`ESP_OTA_IMG_PENDING_VERIFY`). При запуске функции `app_main()` возможны два варианта:

(1) нормальное функционирование.

После тестирования и подтверждения, что все работает правильно, разработчик вызывает функцию `esp_ota_mark_app_valid_cancel_rollback()` для отметки текущего варианта как действительной прошивки (`ESP_OTA_IMG_VALID`). После этого прошивка может загружаться без каких-либо ограничений;

(2) обнаружение серьезных ошибок.

Это вызовет автоматическую вторичную перезагрузку, и загрузчик сразу отметит прошивку как прерванную (`ESP_OTA_IMG_ABORTED`) и откатится на предыдущую версию. После самотестирования разработчик должен подтвердить наличие ошибки и вызовет `esp_ota_mark_app_invalid_rollback_and_reboot()`, чтобы отметить запущенную версию как недействительную. После этого прошивка откатится до предыдущей версии автоматически и не может быть перезагружена снова.

Другая функция номера защищенной версии – предотвратить откат прошивки на более низкую безопасную версию. При выборе приложения для загрузки

загрузчик будет проверять номер защищенной версии дополнительно. Только если номер защищенной версии прошивки равен или выше значения, хранящегося в eFuse чипа, прошивка будет выбрана. Новый номер защищенной версии будет обновлен после того, как статус встроенного ПО будет помечен как действительный (ESP_OTA_IMG_VALID).

И откат, и запрет отката можно включить через меню config следующим образом:

```
(Top) → Bootloader config
...
...
[*] Enable app rollback support
[*] Enable app anti-rollback support
(0) eFuse secure version of app (NEW)
(16) Size of the eFuse secure version field (NEW)
...
...
```

11.3. Практика: пример OTA-обновления

11.3.1. Обновление прошивки через локальный хост

Для примера в ESP-IDF процедура OTA показана на рис. 11.4.

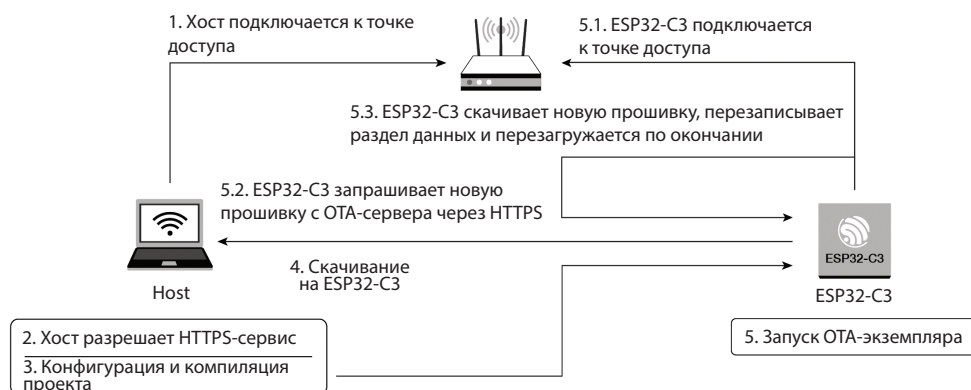


Рисунок 11.4. Процедура OTA-обновления

(1) Включение OTA-сервера

Прежде чем запускать пример OTA, вам необходимо сначала создать службу HTTPS и подать заявку на сертификат. Сертификат можно подписать самостоятельно, выполнив команду `openssl req -x509 -newkey rsa:2048 -keyout ca_key.pem -out ca_cert.pem -days 365 -nodes`. Логи этой операции следующие:

```
Generating a RSA private key
.+++++
.....+++++
writing new private key to 'ca_key.pem'
```

```
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
...
...

```

(2) Включение службы HTTPS

Команда `openssl s_server -www -key ca_key.pem -cert ca_cert.pem -port 8070` включает службу HTTPS с портом 8070 на локальном компьютере. Выходные логи следующие:

```
...
Using default temp DH parameters
ACCEPT
...
...
При успешном подключении HTTPS логи выглядят следующим образом:
139783954665920:error:14094416:SSL routines:ssl3_read_bytes:sslv3 alert
certificate unknown:../ssl/record/rec_layer_s3.c:1528:SSL alert number 46
FILE:advanced_https_ota.bin
...
...

```

(3) Использование компонента `esp_https_ota` для выполнения OTA

Перед тестом OTA давайте сначала посмотрим, как написать программу OTA. Следующий образец кода взят отсюда: https://github.com/espressif/esp-idf/blob/master/examples/system/ota/advanced_https_ota/main/advanced_https_ota_example.c.

```
1. void app_main(void)
2. {
3.     //Initialize NVS
4.     esp_err_t err = nvs_flash_init();
5.     if (err == ESP_ERR_NVS_NO_FREE_PAGES || err ==
6.         ESP_ERR_NVS_NEW_VERSION_FOUND) {
7.         ESP_ERROR_CHECK(nvs_flash_erase());
8.         err = nvs_flash_init();
9.     }
10.    ESP_ERROR_CHECK( err );
11.
12.    ESP_ERROR_CHECK(esp_netif_init());
13.    ESP_ERROR_CHECK(esp_event_loop_create_default());
14.
15.    //Initialize connection
16.    ESP_ERROR_CHECK(example_connect());
17.
18.    //firmware verification
19.    const esp_partition_t *running = esp_ota_get_running_partition();
20.    esp_ota_img_states_t ota_state;
```

```

21.     if (esp_ota_get_state_partition(running, &ota_state) == ESP_OK) {
22.         if (ota_state == ESP_OTA_IMG_PENDING_VERIFY) {
23.             if (esp_ota_mark_app_valid_cancel_rollback() == ESP_OK) {
24.                 ESP_LOGI(TAG, "App is valid, rollback cancelled successfully");
25.             } else {
26.                 ESP_LOGE(TAG, "Failed to cancel rollback");
27.             }
28.         }
29.     }
30.
31. #if CONFIG_EXAMPLE_CONNECT_WIFI
32. //Set PS mode
33.     esp_wifi_set_ps(WIFI_PS_NONE);
34. #endif
35. //Create Over-the-air (OTA) task
36.     xTaskCreate(&advanced_ota_example_task, "advanced_ota_example_task",
37.                 1024 * 8, NULL, 5, NULL);
38. }

```

Приведенный фрагмент кода демонстрирует следующие процедуры:

- a) инициализация NVS-памяти с помощью функции `nvs_flash_init()`, что обычно является первым шагом при написании приложения ESP32-C3;
- b) инициализация уровня `netif` и подключение к Wi-Fi с помощью функции `example_connect()`. Эта функция является универсальной и реализована в компоненте `protocol_examples_common`. Вы можете заменить его своей собственной функцией подключения к Wi-Fi;
- c) если откат включен, необходимо установить статус встроенного ПО;
- d) (необязательно) установка режима системы питания на значение `WIFI_PS_NONE` с помощью функции `esp_wifi_set_ps()`, чтобы отключить режим энергосбережения Wi-Fi и достичь для данных максимальной пропускной способности;
- e) создание задачи OTA и завершение процесса получения прошивки в этой задаче. Для получения информации о выполнении `advanced_ota_example_task`, пожалуйста, обратитесь к разделу 11.1.3.

Получив базовое представление на приведенном примере, теперь мы можем перейти к тестированию OTA.

Пожалуйста, последовательно выполните следующие действия.

- (1) Установите целевой чип на ESP32-C3 с помощью следующей команды:

```
$ idf.py set-target esp32c3
```

- (2) Установите информацию о Wi-Fi.

Запустите `idf.py menuconfig` и отредактируйте SSID и пароль Wi-Fi следующим образом:

```

(Top) → Example Connection Configuration
Espressif IoT Development Framework Configuration
[*] connect using WiFi interface
(Xiaomi_32BD) WiFi SSID
(12345678) WiFi Password

```

```
[ ] connect using Ethernet interface
[*] Obtain IPv6 address
Preferred IPv6 Type (Local Link Address) --->
```

(3) Определите информацию OTA.

Заполните URL-адрес обновления прошивки и пропустите проверку сертификата сервера и версии прошивки.

```
(https://192.168.31.177:8070/advanced_https_ota.bin) Firmware Upgrade URL
[*] Skip server certificate CN fieldcheck
[*] Skip firmware version check
(5000) OTA Receive Timeout
```

(4) Создайте прошивку с помощью команды `idf.py build`.

После сборки извлеките файл `Advanced_https_ota.bin` из каталога сборки в каталог HTTPS-сервера, а именно в каталог, в котором выполняется операция `openssl s_server`.

(5) Загрузите прошивку.

Запустите `idf.py flash monitor`, чтобы загрузить прошивку и открыть монитор.

По завершении всех операций ESP32-C3 включится, подключится к установленному Wi-Fi и перезагрузится после загрузки прошивки с заданного URL.

11.3.2. Обновление прошивки через ESP RainMaker

Более распространенное решение – обновить прошивку через облачную платформу. В этой секции мы покажем, как отправлять сообщения об обновлениях из облака на устройство с помощью ESP RainMaker, также с использованием компонента `esp_https_ota`. С помощью кода OTA, интегрированного в ESP RainMaker SDK, вы можете запустить процесс OTA, просто вызвав `esp_rmaker_ota_enable()`. Обратите внимание, что хотя ESP RainMaker предоставляет два способа осуществления OTA, вам необходимо выбрать получение OTA-сообщений с помощью указания темы (`topics`). Подписываясь на темы, связанные с OTA, вы можете получать сообщения MQTT, выделять URL-адрес прошивки и получать сообщения о ходе и окончательном статусе текущего обновления. Код для ESP RainMaker OTA хранится в папке https://github.com/espressif/esp-rainmaker/tree/master/components/esp_rainmaker/src/ota. Исходный код, связанный с загрузкой прошивки, хранится в файле `esp_rmaker_ota.c` в том же каталоге так же, как и следующий код:

```
1. //ESP RainMaker OTA status
2. char *esp_rmaker_ota_status_to_string(ota_status_t status)
3. {
4.     switch (status) {
5.         case OTA_STATUS_IN_PROGRESS:
6.             return "in-progress";
7.         case OTA_STATUS_SUCCESS:
8.             return "success";
9.         case OTA_STATUS_FAILED:
10.            return "failed";
11.         case OTA_STATUS_DELAYED:
12.            return "delayed";
13.         default:
```

```

14.     return "invalid";
15.     }
16.     return "invalid";
17. }
18.
19. esp_err_t esp_rmaker_ota_report_status(esp_rmaker_ota_handle_t ota_handle,
20.                                       ota_status_t status,
21.                                       char *additional_info)
22. {
23.     .....
24.     if (ota->type == OTA_USING_PARAMS) {
25.         err = esp_rmaker_ota_report_status_using_params(ota_handle, status,
26.                                                       additional_info);
27.     } else if (ota->type == OTA_USING_TOPICS) {
28.         err = esp_rmaker_ota_report_status_using_topics(ota_handle, status,
29.                                                       additional_info);
30.     }
31.     .....
32. }

```

В ESP RainMaker есть четыре статуса OTA: выполняется получение прошивки (OTA_STATUS_IN_PROGRESS), OTA выполнено удачно (OTA_STATUS_SUCCESS), OTA выполнено неудачно (OTA_STATUS_FAILED) и OTA проходит с задержкой (OTA_STATUS_DELAYED).

Получение прошивки соответствует статусу загрузки прошивки, что следует сообщать облачной платформе, когда вызывается функция `esp_https_ota_begin()`, и облачная платформа соответствующим образом обновит значок. Статус успешного и неудачного выполнения OTA указывает результаты загрузки и проверки прошивки. Статус задержки указывает, что устройство в данный момент недоступно для обработки запроса, а статус OTA может быть обновлен позже с помощью функции `esp_rmaker_ota_report_status()`.

```

1. //Firmware information verification
2. static esp_err_t validate_image_header(esp_rmaker_ota_handle_t ota_handle,
3.                                       esp_app_desc_t *new_app_info)
4. {
5.     if (new_app_info == NULL) {
6.         return ESP_ERR_INVALID_ARG;
7.     }
8.
9. //Firmware status acquisition
10.    const esp_partition_t *running = esp_ota_get_running_partition();
11.    esp_app_desc_t running_app_info;
12.    if (esp_ota_get_partition_description(running, &running_app_info) ==
13.        ESP_OK) {
14.        ESP_LOGD(TAG, "Running firmware version: %s", running_app_info.version);
15.    }
16.
17. //Verify project version number
18. #ifndef CONFIG_ESP_RMAKER_SKIP_VERSION_CHECK
19.     if (memcmp(new_app_info->version, running_app_info.version,
20.               sizeof(new_app_info->version)) == 0){
21.         ESP_LOGW(TAG, "Current running version is same as the new.
22.             We will not continue the update." );
23.         esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_FAILED,

```



```

24.         "Same version received");
25.     return ESP_FAIL;
26. }
27. #endif
28.
29. //Verify project name
30. #ifndef CONFIG_ESP_RMAKER_SKIP_PROJECT_NAME_CHECK
31.     if (memcmp(new_app_info->project_name, running_app_info.project_name,
32.             sizeof(new_app_info->project_name)) != 0 ){
33.         ESP_LOGW(TAG, "OTA Image built for Project: %s. Expected: %s",
34.                 new_app_info->project_name, running_app_info.project_name);
35.         esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_FAILED,
36.                                     "Project Name mismatch");
37.         return ESP_FAIL;
38.     }
39. #endif
40.
41.     return ESP_OK;
42. }

```

ESP RainMaker управляет прошивкой, проверяя номер версии и имя проекта. Только когда новая и старая прошивки с одинаковым названием проекта проверены на наличие разных номеров версий, будет разрешено продолжить загрузку прошивки. Вообще говоря, номер версии проекта обычно увеличивается, что более удобно для контроля версий. Сравнивая имена проектов, вы можете предотвратить любые случайные установки прошивки для других продуктов.

```

1. static esp_err_t esp_rmaker_ota_default_cb(esp_rmaker_ota_handle_t ota_handle,
2.     esp_rmaker_ota_data_t *ota_data)
3. {
4.     .....
5. //OTA http parameter configuration
6.     esp_err_t ota_finish_err = ESP_OK;
7.     esp_http_client_config_t config = {
8.         .url = ota_data->url,
9.         .cert_pem = ota_data->server_cert,
10.        .timeout_ms = 5000,
11.        .buffer_size = DEF_HTTP_RX_BUFFER_SIZE,
12.        .buffer_size_tx = buffer_size_tx
13.    };
14. #ifdef CONFIG_ESP_RMAKER_SKIP_COMMON_NAME_CHECK
15.     config.skip_cert_common_name_check = true;
16. #endif
17.     .....
18. //Report update status
19.     esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_IN_PROGRESS,
20.                                 "Starting OTA Upgrade");
21.
22.     ...
23.     ...
24. //Establish HTTPS connection and prepare to download firmware
25.     esp_err_t err = esp_https_ota_begin(&ota_config, &https_ota_handle);
26.     if (err != ESP_OK) {
27.         ESP_LOGE(TAG, "ESP HTTPS OTA Begin failed");

```

```

28.     esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_FAILED,
29.         "ESP HTTPS OTA Begin failed");
30.     return ESP_FAIL;
31. }
32. ....
33. //Acquire firmware information for verification
34.     esp_app_desc_t app_desc;
35.     err = esp_https_ota_get_img_desc(https_ota_handle, &app_desc);
36.     if (err != ESP_OK) {
37.         ESP_LOGE(TAG, "esp_https_ota_read_img_desc failed");
38.         esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_FAILED,
39.             "Failed to read image decription");
40.         goto ota_end;
41.     }
42.     err = validate_image_header(ota_handle, &app_desc);
43.     if (err != ESP_OK) {
44.         ESP_LOGE(TAG, "image header verification failed");
45.         goto ota_end;
46.     }
47.
48.     esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_IN_PROGRESS,
49.         "Downloading Firmware Image");
50.     int count = 0;
51.
52. // Download firmware cyclically
53.     while (1) {
54.         err = esp_https_ota_perform(https_ota_handle);
55.         if (err != ESP_ERR_HTTPS_OTA_IN_PROGRESS) {
56.             break;
57.         }
58.         ....
59.     }
60.
61. // Release HTTPS resource after downloading and start verification
62.     ota_finish_err = esp_https_ota_finish(https_ota_handle);
63.     if ((err == ESP_OK) && (ota_finish_err == ESP_OK)) {
64. //Report OTA succeeded status to Cloud after successful verification
65.         ESP_LOGI(TAG, "OTA upgrade successful. Rebooting in %d seconds..." ,
66.             OTA_REBOOT_TIMER_SEC);
67.         esp_rmaker_ota_report_status(ota_handle, OTA_STATUS_SUCCESS,
68.             "OTA Upgrade finished successfully");
69.         esp_rmaker_reboot(OTA_REBOOT_TIMER_SEC);
70.         return ESP_OK;
71.     }
72.     ....
73. }

```

Функция `esp_rmaker_ota_default_cb()` в ESP RainMaker является функцией обратного вызова OTA по умолчанию и будет вызываться после получения OTA-сообщения из облака. Действие этой функции аналогично действию примера функции OTA, представленной в разделе 11.1.3. Разница в том, что функция `esp_rmaker_ota_default_cb()` включает отчетность о статусе OTA, которая будет отправлять на облачную платформу текущий прогресс выполнения OTA на устройстве вовремя.

```

1. esp_err_t esp_rmaker_ota_enable(esp_rmaker_ota_config_t *ota_config,
2.                               esp_rmaker_ota_type_t type)
3. {
4. //Acquire partition information and verify firmware validity through callback
5.     const esp_partition_t *running = esp_ota_get_running_partition();
6.     esp_ota_img_states_t ota_state;
7.     if (esp_ota_get_state_partition(running, &ota_state) == ESP_OK) {
8.         if (ota_state == ESP_OTA_IMG_PENDING_VERIFY) {
9.             ESP_LOGI(TAG, "First Boot after an OTA");
10. //Run diagnostic function
11.         bool diagnostic_is_ok = true;
12.         if (ota_config->ota_diag) {
13.             diagnostic_is_ok = ota_config->ota_diag();
14.         }
15.         if (diagnostic_is_ok) {
16.             ESP_LOGI(TAG, "Diagnostics completed successfully!
17.                 Continuing execution ..." );
18.             esp_ota_mark_app_valid_cancel_rollback();
19.         } else {
20.             ESP_LOGE(TAG, "Diagnostics failed! Start rollback to the
21.                 previous version ..." );
22.             esp_ota_mark_app_invalid_rollback_and_reboot();
23.         }
24.     }
25. }
26.
27. // Over-the-air (OTA) task callback function
28.     if (ota_config->ota_cb) {
29.         ota->ota_cb = ota_config->ota_cb;
30.     } else {
31.         ota->ota_cb = esp_rmaker_ota_default_cb;
32.     }
33.     .....
34.     return err;
35. }

```

Раздел OTA в ESP RainMaker содержит сегмент самотестирования функции отката. Вы можете ввести функцию самотестирования с возвращаемым логическим значением через структуру `esp_rmaker_ota_config_t`. При вызове функции `esp_rmaker_ota_enable()` для включения OTA, как только будет установлено, что текущее состояние прошивки ожидает проверки (`ESP_OTA_IMG_PENDING_VERIFY`), функция самопроверки, введенная ранее, будет вызвана через указатель, и текущий статус прошивки будет установлен с помощью возвращаемого значения функции самопроверки. Значение `server_cert` в структуре `esp_rmaker_ota_config_t` указывает на сертификат на стороне сервера. ESP RainMaker использует службу хранения данных AWS S3, и вы можете передать сертификат напрямую с помощью макроса `ESP_RMAKER_OTA_DEFAULT_SERVER_CERT`, который используется во время OTA для выполнения проверок и предотвращения подмены DNS. Вы можете загрузить новую прошивку в серверную часть управления ESP RainMaker. Проектная версия новой прошивки должна отличаться от версии, до которой ожидается обновление, и есть два способа изменить версию в ESP RainMaker:

- (1) измените номер версии проекта, указанный в разделе 11.2.1;
- (2) измените файл *CMakeLists.txt*, добавив `set(PROJECT_VER «1.0»)`. Вы можете обратиться к примеру по этой части.

После создания новой прошивки вы можете загрузить ее, как показано на рис. 11.5.

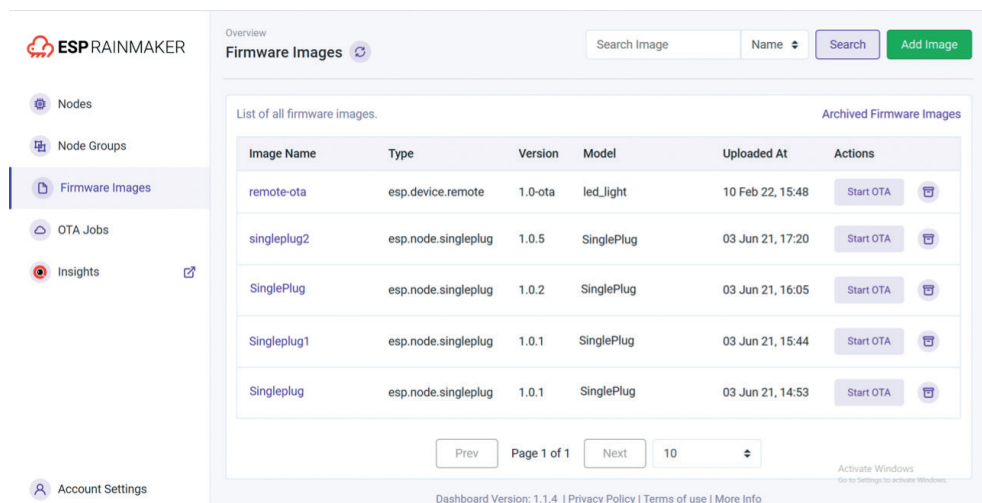


Рисунок 11.5. Загружаем новую прошивку

После загрузки прошивки вы можете включить задачу OTA, как показано на рис. 11.6.

Процедура загрузки OTA-задачи следующая:

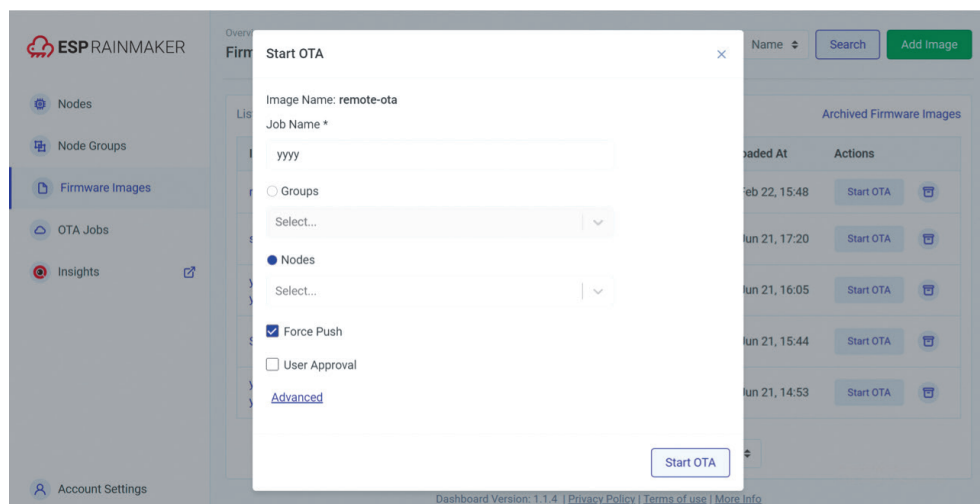


Рисунок 11.6. Загрузка задачи OTA

- (1) Выберите из списка прошивку, используемую для OTA.
- (2) Нажмите **Start OTA** (Запустить OTA) для прошивки.
- (3) Заполните информацию о задаче OTA и выберите узел для обновления. Если выбрана опция **Force Push** (Вынужденная отправка), онлайн-узлы могут немедленно получать сообщения OTA. В противном случае узлы будут получать URL-адреса для OTA на основе определенной политики OTA (проверка во время запуска, периодические проверки и т. д.), что может привести к задержкам.

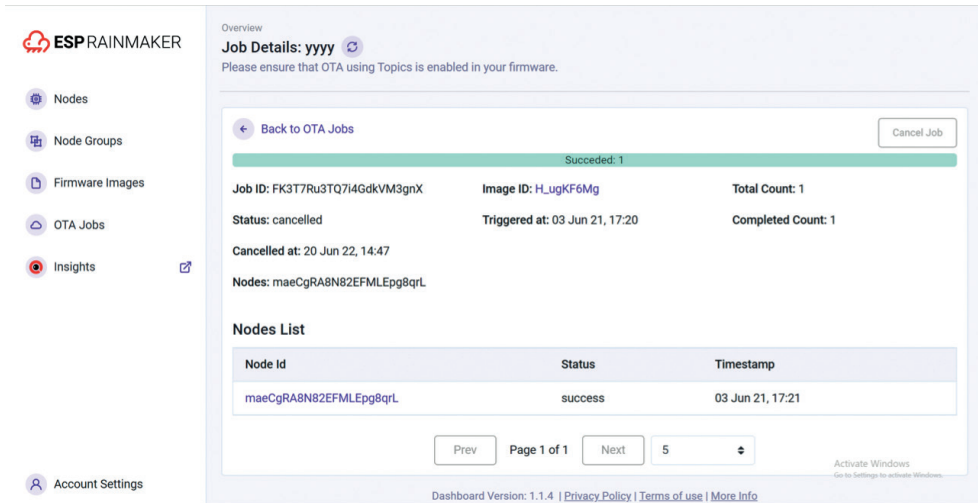


Рисунок 11.7. Интерфейс монитора заданий OTA Job Monitor

Монитор заданий OTA (см. рис. 11.7) обеспечивает обратную связь в режиме реального времени о состоянии OTA, включая информацию, доступную пользователям, и реализованные функции.

- После успешного запуска OTA статус OTA дает возможность проверить в деталях текущее задание OTA, о котором сообщает устройство.
- Монитор заданий OTA предоставит обзор задачи и проверку состояния каждого узла.
- Задача OTA может быть отменена на полпути, но узлы, уже получившие URL-адрес, будут продолжать обновляться.

11.4. Резюме

В этой главе в основном представлены механизм и процедуры OTA. Во-первых, здесь описаны основные процедуры и способы проведения OTA. Затем представлены функции таблицы разделов и поля, относящиеся к OTA. В реальном серийном производстве также необходимо включить функции отката и предотвращения отката для улучшения стабильности и надежности устройства. Наконец, согласно приведенному практическому примеру, вы можете либо завершить обновление прошивки через локальный хост, либо отправить сообщение об обновлении на устройство из облака через ESP RainMaker.

Часть IV

Оптимизация и серийное производство

Наконец-то мы выпускаем наш продукт на рынок! Но подождите – вот краткий список необходимых действий перед официальным запуском продукта.

- **Сертификация соответствия:** энергопотребление, безопасность работы и т. д.
- **Безопасность:** блокировка вредоносного вмешательства или незаконных/несанкционированных программ.
- **Серийное производство.**
- **Мониторинг и обслуживание.**

В следующих главах будут рассмотрены вопросы управления питанием, оптимизации энергопотребления, безопасности устройств, серийной прошивки и тестирования, а также представлена платформа удаленного мониторинга ESP Insights. Прочитав эту часть, вы будете готовы создать зрелый продукт интернета вещей!

ГЛАВА 12

Управление питанием и оптимизация энергопотребления

ГЛАВА 14

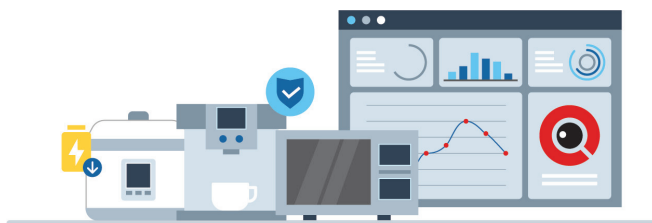
Запись и тестирование прошивки для серийного производства

ГЛАВА 13

Расширенные функции безопасности устройства

ГЛАВА 15

ESP Insights: платформа удаленного мониторинга



Управление питанием и оптимизация энергопотребления

Благодаря широкому применению продуктов интернета вещей люди могут видеть все больше и больше продуктов интернета вещей в повседневной жизни, например умные часы, умные розетки, умные лампочки, умные колонки и т. д. Многие из этих IoT-продуктов вынуждены снижать энергопотребление, поскольку они питаются от батареи или требуют сертификации энергопотребления. Например, спецификация CEC Tile 20 для получения сертификата энергопотребления в Калифорнии, США, требует, чтобы интеллектуальные лампочки в режиме ожидания не превышали потребляемую мощность 0,2 Вт. Разработчики умных часов с батарейным питанием также стремятся продлить время их работы. Разработчики таких продуктов интернета вещей должны уделять приоритетное внимание энергопотреблению как важнейшему фактору при разработке продукта. Они должны иметь полное представление по части энергосберегающих характеристик используемых чипов и владеть навыками использования их на практике в IoT-проектах. Для достижения этой цели им следует уделить приоритетное внимание использованию технологий беспроводной связи с низким энергопотреблением, таких как Bluetooth LE, и в своих реализациях применять схемы с низким энергопотреблением. Это гарантирует минимизацию энергопотребления на протяжении всего процесса разработки.

В проектах с низким энергопотреблением срок службы устройства с батарейным питанием и его энергопотребление часто определяются по его среднему току. На этот средний ток влияет несколько факторов, включая ток в различных режимах пониженной мощности, рабочий ток в активных состояниях, продолжительность активации или деактивации режима пониженного энергопотребления, а также вычислительную мощность процессора. ESP32-C3 обеспечивает поддержку на уровне чипа для сценариев с низким энергопотреблением. Он использует передовую технологию управления питанием для переключения между различными режимами питания и функции интеллектуальных периферийных устройств с низким энергопотреблением, помогающие сократить время пробуждения процессора, что приводит к дальнейшему снижению общего энергопотребления.

12.1. Управление питанием ESP32-C3

Алгоритм управления питанием, включенный в ESP-IDF, может регулировать частоту расширенной периферийной шины (advanced peripheral bus, APB), час-

тоту процессора и перевод чипа в режим Light-sleep для запуска приложения с минимально возможным энергопотреблением с учетом требований компонентов приложения. Кроме того, чип автоматически переходит в режим Light-sleep при простое, что снижает энергопотребление во время работы приложения. Различные режимы пониженного энергопотребления ESP32-C3 подробно обсуждаются в разделе 12.2. За включение функций управления питанием приходится платить увеличением задержки прерываний. Дополнительная задержка зависит от нескольких факторов, таких как частота процессора, одно-/двухъядерный режим, требуется ли переключатель частоты.

Приложения могут устанавливать/снимать блокировки для управления алгоритмом управления питанием. Если приложение ставит блокировку, работа алгоритма управления питанием ограничивается. При снятии блокировки такие ограничения снимаются. Блокировки управления питанием имеют счетчики установки/снятия. Если блокировка была установлена несколько раз подряд, ее необходимо снять столько же раз для отмены связанных с ней ограничений.

ESP32-C3 поддерживает три типа блокировок, описанных в табл. 12.1.

Таблица 12.1. Блокировки управления питанием

Блокировка управления питанием	Описание
SP_PM_CPU_FREQ_MAX	Установка максимальной частоты процессора, заданной с помощью <code>esp_pm_configure()</code> . Для ESP32-C3 это значение можно установить на 80 МГц, 160 МГц или 240 МГц
ESP_PM_APB_FREQ_MAX	Установка максимальной поддерживаемой частоты шины APB. Для ESP32-C3 это 80 МГц
ESP_PM_NO_LIGHT_SLEEP	Отключить автоматическое переключение в режим Light-sleep

Приложения могут устанавливать или снимать блокировки управления питанием таким образом, чтобы обеспечить выполнение задач, в которых управление питанием не требуется. Например, драйвер периферийного устройства, тактируемый от APB, может запросить установку частоты APB на 80 МГц на время использования периферийного устройства; RTOS может запросить для ЦП работу на самой высокой настроенной частоте, пока есть задачи, готовые к запуску; драйверу периферийного устройства могут потребоваться прерывания для включения, это означает, что ему придется запросить отключение режима Light-sleep.

Поскольку запрос более высоких частот APB и процессора или отключение режима Light-sleep приводит к увеличению потребления тока, пожалуйста, старайтесь свести к минимуму использование блокировок управления питанием.

12.1.1. Динамическое масштабирование частоты

Когда управление питанием включено, частота периферийной шины (APB) и частота ЦП могут меняться в процессе эксплуатации. Этот процесс называется динамическим масштабированием частоты (Dynamic Frequency Scaling, DFS). Если DFS включено, частоту APB можно изменить несколько раз в течение одного такта RTOS. Изменение частоты APB не влияет на работу некоторых

периферийных устройств, в то время как другие периферийные устройства могут иметь проблемы. Например, периферийные таймеры будут продолжать считать, однако скорость, с которой они считают, изменится пропорционально частоте APB. Поэтому разработчики должны понимать, какие периферийные устройства будут и какие не будут затронуты включением DFS. Благодаря постоянному совершенствованию ESP-IDF развития все больше и больше драйверов периферийных устройств не будут затронуты DFS.

Следующие периферийные устройства не подвержены воздействию DFS при синхронизации от заданного источника синхронизации:

- **UART**: если в качестве источника синхронизации используется REF_TICK, DFS не влияет на UART. В противном случае DFS будет влиять;
- **LEDC**: если в качестве источника синхронизации используется REF_TICK, DFS не влияет на LEDC. В противном случае DFS будет влиять;
- **RMT⁴⁴**: если в качестве источника синхронизации используется REF_TICK или внешний кварцевый резонатор XTAL, DFS не влияет на RMT.

В настоящее время DFS не влияет на перечисленные далее драйверы периферийных устройств. Эти драйверы включают блокировку ESP_PM_APB_FREQ_MAX на время работы и снимут блокировку после ее завершения автоматически:

- **хост SPI**;
- **I2C**;
- **I2S** (если используется тактовый сигнал APLL, I2S получает управление питанием ESP_PM_NO_LIGHT_SLEEP);
- **SPI ведомый**: между вызовами spi_slave_initialize() и spi_slave_free() ведомый SPI не зависит от DFS;
- **Wi-Fi**: между вызовами esp_wifi_start() и esp_wifi_stop() Wi-Fi не подчиняется DFS. В режиме ожидания модема, когда Wi-Fi включен, чип отключает блокировку управления питанием ESP_PM_APB_FREQ_MAX до выключения RF-модуля;
- **TWAI**: между вызовами twai_driver_install() и twai_driver_uninstall() DFS не влияет на TWAI;
- **Bluetooth**: между вызовами esp_bt_controller_enable() и esp_bt_controller_disable() DFS не влияет на Bluetooth. В режиме ожидания модема, когда Bluetooth включен, чип отключает блокировку управления питанием ESP_PM_APB_FREQ_MAX до выключения RF-модуля, но при этом удерживает блокировку ESP_PM_NO_LIGHT_SLEEP, если CONFIG_BTDM_CTRL_LOW_POWER_CLOCK не настроен на внешний кварцевый генератор 32 кГц.

DFS влияет на следующие драйверы периферийных устройств, поэтому приложениям необходимо иметь возможность отключать блокировку самостоятельно:

- **PCNT**;
- **сигма-дельта-модулятор**;
- **группа таймеров**.

⁴⁴ Remote Control Peripheral (RMT) – периферийное устройство дистанционного управления, узел ESP32-C3 с поддержкой двух каналов инфракрасного передатчика и двух каналов инфракрасного приемника. Может использоваться также для организации различных протоколов 1-Wire.

12.1.2. Настройка управления питанием

Обычно автоматический режим Light-sleep используется в сочетании с режимом Modem-sleep и функцией управления питанием. Настройка автоматического включения режима Light-sleep описана в разделе 12.2.2.

12.2. Режимы пониженного энергопотребления ESP32-C3

ESP32-C3 оснащен усовершенствованным блоком управления питанием (Power Management Unit, PMU), который позволяет гибко включать питание различных областей чипа для достижения наилучшего баланса между производительностью, энергопотреблением и задержкой выхода из режима сна. ESP32-C3 имеет четыре предустановленных режима питания, которые помогают разработчикам не только выполнять требования различных задач в приложениях интернета вещей, но и пройти строгие сертификационные испытания по энергопотреблению. Эти режимы питания успешно используются во многих проектах интернета вещей, включая умное освещение. ESP32-C3 предлагает ряд маломощных решений для этих режимов, которые могут служить эталоном, чтобы разработчики могли выбирать и настраивать их в соответствии со своими конкретными требованиями. Четыре режима питания следующие:

- **активный режим Active mode:** ЦП и RF-чип включены. Чип может принимать, передавать или ждать сигнала;
- **режим Modem-sleep mode:** ЦП работает, но тактовую частоту можно снизить. Базовый диапазон Wi-Fi, базовый диапазон Bluetooth LE и RF отключены, но соединения Wi-Fi и Bluetooth LE могут оставаться активными;
- **режим Light-sleep mode:** ЦП приостановлен. Базовый диапазон Wi-Fi, базовый диапазон Bluetooth LE и RF отключены. Любые события пробуждения (MAC, хост, таймер RTC или внешние прерывания) будут выводить чип из сна. В автоматическом режиме Light-sleep Wi-Fi или Bluetooth LE могут оставаться подключенными;
- **режим глубокого сна Deep-sleep mode:** процессор и большинство периферийных устройств отключаются. Включены только память модуля RTC и периферийные устройства RTC. Базовый диапазон Wi-Fi, базовый диапазон Bluetooth LE и RF-модуль отключены.

По умолчанию ESP32-C3 после перезагрузки перейдет в активный режим. В активном режиме все компоненты ESP32-C3 работают нормально. Когда процессору не требуется непрерывная работа, например дождавшись пробуждения внешней активности, чип может перейти в один из режимов пониженного энергопотребления. Разработчики могут выбирать различные режимы питания в зависимости от конкретного энергопотребления, задержки пробуждения и доступных требований к источнику пробуждения. За исключением активного режима, остальные три режима являются режимами с пониженным энергопотреблением. В табл. 12.2 перечислены различия между тремя режимами энергопотребления.

Таблица 12.2. Различия между тремя режимами пониженного энергопотребления

Компонент	Modem-sleep	Light-sleep		Deep-sleep
		Автоматический	Принудительный	
Wi-Fi-соединение и соединение Bluetooth LE	Удержание	Удержание	Разрыв	Разрыв
GPIO	Удержание	Удержание		Удержание
Wi-Fi	Выкл	Выкл		Выкл
Системный тактовый генератор	Вкл	Выкл		Выкл
RTC	Вкл	Вкл		Вкл
CPU	Вкл	Пауза		Выкл

12.2.1. Режим Modem-sleep

В настоящее время режим Modem-sleep на ESP32-C3 применим только при активных соединениях Wi-Fi Station и Bluetooth LE. Режим вступит в силу после того, как Wi-Fi Station подключится к роутеру или подключится Bluetooth LE, тогда чип будет периодически переключаться между активным режимом и режимом Modem-sleep. В режиме Modem-sleep основные полосы Wi-Fi и Bluetooth LE синхронизированы или выключены. Когда RF-модуль выключен, ESP32-C3 можно автоматически разбудить без каких-либо задержек (также это может сделать любой источник пробуждения без дополнительной настройки). После выхода из режима Modem-sleep RF-модуль чипа переключается в активный режим, что приводит к увеличению энергопотребления.

ESP32-C3 использует механизм сигнального сообщения трафика доставки Wi-Fi (Delivery Traffic Indication Message, DTIM) для поддержки соединения с роутером. В режиме ожидания модема ESP32-C3 отключит питание RF-модуля между двумя сигнальными сообщениями DTIM для экономии энергии и автоматического пробуждения RF-модуля непосредственно перед поступлением следующего сообщения DTIM. Продолжительность сна определяется интервалами DTIM-сигналов маршрутизатора и параметром `listen_interval` ESP32-C3. В режиме Modem-sleep ESP32-C3 остается подключенным к маршрутизатору Wi-Fi, что позволяет ему получать интерактивную информацию со смартфона или сервера.

Для DTIM можно указать частоту передачи данных при использовании маршрутизатора. Обычно интервал между сигнальными DTIM-сообщениями маршрутизатора составляет от 100 до 1000 мс.

ESP32-C3 использует событие подключения Bluetooth LE для поддержания соединения с одноранговым узлом устройства. В режиме ожидания модема ESP32-C3 отключит RF-модуль между двумя событиями подключения для экономии энергии и автоматическое пробуждение перед следующим наступлением события. Продолжительность сна определяется параметрами подключения Bluetooth LE.

Режим Modem-sleep обычно используется в приложениях с низким энергопотреблением, где процессору требуется работать постоянно и необходимо поддерживать соединение Wi-Fi или Bluetooth LE. Например, при использовании ESP32-C3 в локальных приложениях голосового пробуждения, когда ЦП постоянно собирает и обрабатывает аудиоданные.

1. Режим Wi-Fi Modem-sleep

В разработке пользователи могут использовать функцию `esp_wifi_set_ps()` для установки типа энергосбережения Wi-Fi:

- `WIFI_PS_NONE`: режим Modem-sleep не используется;
- `WIFI_PS_MIN_MODEM`: ESP32-C3 просыпается для приема каждого DTIM-сообщения маршрутизатора, т. е. через каждый сигнальный интервал маршрутизатора;
- `WIFI_PS_MAX_MODEM`: ESP32-C3 периодически просыпается для приема сигнального DTIM-сообщения. Интервал можно настроить с помощью параметра `listen_interval` в `wifi_sta_config_t` (в единицах времени DTIM-интервала маршрутизатора). Значение по умолчанию равно 3, что указывает интервал в 3 сигнальных DTIM-сообщения маршрутизатора. Код выглядит следующим образом:

```
1. typedef enum {
2.     WIFI_PS_NONE, /*< No power save*/
3.     WIFI_PS_MIN_MODEM, /*< Minimum modem power saving. In this mode,
4.         station wakes up to receive beacon every DTIM period*/
5.     WIFI_PS_MAX_MODEM, /*< Maximum modem power saving. In this mode,
6.         interval to receive beacons is determined by the
7.         listen_interval parameter in wifi_sta_config_t*/
8. } wifi_ps_type_t;
9.
10. esp_err_t esp_wifi_set_ps(wifi_ps_type_t type);
```

Если тип режима настроен как `WIFI_PS_MAX_MODEM`, настройте интервал `listen_interval` так, чтобы ESP32-C3 просыпался для приема сигнала следующим образом:

```
1. #define LISTEN_INTERVAL 3
2. wifi_config_t wifi_config = {
3.     .sta = {
4.         .ssid = "SSID",
5.         .password = "Password",
6.         .listen_interval = LISTEN_INTERVAL,
7.     },
8. };
9. ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
10. ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
11. ESP_ERROR_CHECK(esp_wifi_start());
12.
13. ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_MAX_MODEM));
```

2. Режим Bluetooth LE Modem-sleep

Чтобы включить режим Modem-sleep для Bluetooth LE, запустите команду `idf.py menuconfig` для вызова инструмента настройки Espressif IoT Development Frame-

work Configuration tool (далее именуемого menuconfig), затем перейдите в **Component config** (Конфигурация компонентов) → **Bluetooth** → **Bluetooth controller (ESP32C3 Bluetooth Low Energy)** → **MODEM SLEEP Options** (Опции режима Modem-sleep) и активируйте **Bluetooth modem sleep**; используйте конфигурацию по умолчанию **Bluetooth Modem sleep Mode 1** (Bluetooth Modem sleep режим 1) и **Bluetooth low power clock** (Экономичное тактирование Bluetooth). Режим Modem-sleep ESP32-C3 Bluetooth LE показан на рис. 12.1.

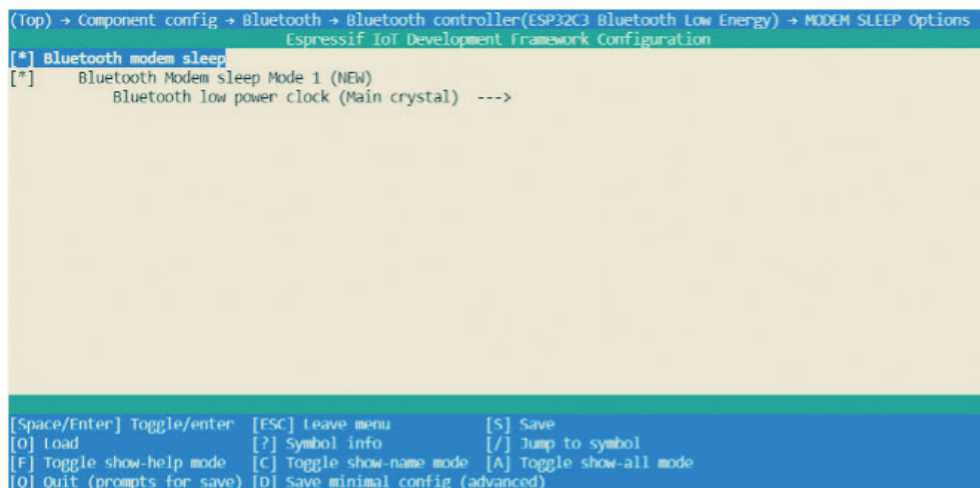


Рисунок 12.1. Режим Modem-sleep ESP32-C3 Bluetooth LE

12.2.2. Режим Light-sleep

Режим Light-sleep работает аналогично режиму Modem-sleep, за исключением того, что в режиме Light-sleep ESP32-C3 отключит RF-модуль и цифровую периферию и большая часть оперативной памяти будет работать с ограниченной тактовой частотой. Кроме того, процессор будет приостановлен, что приводит к снижению энергопотребления по сравнению с режимом Modem-sleep. После выхода из режима Light-sleep периферийные устройства и процессор ESP32-C3 возобновляют работу, и их внутреннее состояние сохраняется. Задержка пробуждения в режиме Light-sleep меньше, чем 1 мс. Есть два способа перевести ESP32-C3 в режим Light-sleep:

Вход в режим Light-sleep вручную

Это достигается путем вызова API. Чтобы войти в режим Light-sleep вручную, необходимо настроить Wi-Fi в качестве источника пробуждения, чтобы устройство могло получать интерактивную информацию со смартфона или сервера через маршрутизатор.

Автоматический вход в режим Light-sleep

После настройки на автоматический переход в режим Light-sleep ESP32-C3 автоматически войдет в него, когда процессор и RF-модуль простаивают, и может быть автоматически пробужден для получения интерактивной информации со смартфона или сервера через маршрутизатор.

1. Источники пробуждения в режиме Light-sleep

Для входа в режим Light-sleep вручную необходимо настроить источник пробуждения, которым могут служить таймеры, GPIO, UART, Wi-Fi или Bluetooth LE. ESP32-C3 поддерживает настройку одного или нескольких источников пробуждения одновременно, и в этом случае ESP32-C3 будет разбужден, когда сработает любой из них. Пользователи могут использовать функцию `esp_sleep_enable_*_wakeup()` для настройки источников пробуждения или функцию отключения источника пробуждения `esp_sleep_disable_wakeup_source()`. Пользователи могут настроить источник пробуждения в любое время, прежде чем войти в режим Light-sleep. После пробуждения пользователи могут определить, какой источник отвечал за пробуждение чипа вызовом функции `esp_sleep_get_wakeup_cause()`. Доступные источники пробуждения для Light-sleep включают в себя:

Пробуждение от GPIO

Любой GPIO-вывод можно использовать в качестве внешнего входа для вывода чипа из режима Light-sleep. Каждый контакт можно индивидуально настроить для запуска пробуждения на высоком или низком уровне с помощью функции `gpio_wakeup_enable()`. Пробуждение от GPIO можно использовать для любого типа GPIO (RTC IO или цифровой ввод-вывод). Предварительно должна быть вызвана функция `esp_sleep_enable_gpio_wakeup()`, чтобы разрешить использование данного вывода для пробуждения.

Пробуждение по таймеру

Контроллер RTC имеет встроенный таймер, который можно использовать для пробуждения чипа через заранее установленное количество времени. Время указывается с точностью до микросекунды, но фактическое разрешение зависит от источника синхронизации `SLOW_CLK`, выбранного для модуля RTC. Периферийные устройства RTC или память RTC не нужно включать во время сна в этом режиме пробуждения. Функцию `esp_sleep_enable_timer_wakeup()` можно использовать для включения вывода из сна с помощью таймера.

Пробуждение по UART

Когда ESP32-C3 получает входной сигнал UART от внешних устройств, часто необходимо вывести его из спящего режима, запустив чип при доступности входных данных. Периферийное устройство UART содержит функцию, которая позволяет вывод чипа из режима Light-sleep при достижении определенного количества нарастающих фронтов на контакте RX. Это количество нарастающих фронтов можно установить с помощью `uart_set_wakeup_threshold()`. Обратите внимание, что символ, вызывающий пробуждение (и любые символы перед ним), не будет получен UART после пробуждения. Это означает, что внешнее устройство обычно должно отправить дополнительный символ в ESP32-C3, чтобы вызвать пробуждение перед отправкой данных. Для включения UART как источника пробуждения можно использовать функцию `esp_sleep_enable_uart_wakeup()`.

Пробуждение по Wi-Fi

Если требуется поддержание соединения Wi-Fi, Wi-Fi можно настроить в режиме источника пробуждения ESP32-C3. ESP32-C3 просыпается перед при-

бытием каждого DTIM-сигнала от точки доступа и включает свой RF-модуль, поддерживая таким образом соединение Wi-Fi. Функция `esp_sleep_enable_wifi_wakeup()` может использоваться для включения этого источника пробуждения.

2. Вход в режим Light-sleep вручную

Чтобы вручную войти в режим Light-sleep, пользователи могут вызвать соответствующие API для отправки ESP32-C3 в режим Light-sleep, когда это необходимо. После входа в режим Light-sleep ESP32-C3 выключит RF-модуль и приостановит процессор. После выхода из режима Light-sleep ESP32-C3 продолжит выполнять исходную программу в том месте, где был вызов API Light-sleep. После ручного входа в режим Light-sleep ESP32-C3 может поддерживать соединение к маршрутизатору, включив Wi-Fi в качестве источника пробуждения и получая интерактивную информацию со смартфона или сервера через роутер. Однако если Wi-Fi не включен в качестве источника пробуждения, ESP32-C3 может не получать пакеты в сети или отключать Wi-Fi-связь. Ситуация с включением/отключением Bluetooth LE в качестве источника пробуждения аналогична.

Примечание

- После вызова интерфейса, который вручную отправляет чип в режим Light-sleep, ESP32-C3 не перейдет сразу в режим Light-sleep, а дождется, пока система сначала перейдет в режим ожидания.
- При включенном источнике пробуждения Wi-Fi только при входе в режим Light-sleep вручную ESP32-C3 может поддерживать соединение с маршрутизатором и получать данные, отправленные через сеть.

3. Инструкция по входу в режим Light-sleep вручную

После настройки источника пробуждения пользователи могут вручную перевести чип в режим Light-sleep вызовом функции `esp_light_sleep_start()`. Код выглядит следующим образом:

```
1. #define BUTTON_WAKEUP_LEVEL_DEFAULT 0
2. #define BUTTON_GPIO_NUM_DEFAULT    9
3.
4. /*Configure the button GPIO as input, enable wakeup*/
5. const int button_gpio_num = BUTTON_GPIO_NUM_DEFAULT;
6. const int wakeup_level = BUTTON_WAKEUP_LEVEL_DEFAULT;
7. gpio_config_t config = {
8.     .pin_bit_mask = BIT64(button_gpio_num),
9.     .mode = GPIO_MODE_INPUT
10. };
11. ESP_ERROR_CHECK(gpio_config(&config));
12. gpio_wakeup_enable(button_gpio_num, wakeup_level == 0 ?
13.     GPIO_INTR_LOW_LEVEL : GPIO_INTR_HIGH_LEVEL);
14.
15. /*Wake up in 2 seconds, or when button is pressed*/
16. esp_sleep_enable_timer_wakeup(2000000);
17. esp_sleep_enable_gpio_wakeup();
18.
```



```
19. /*Enter sleep mode*/
20. esp_light_sleep_start();
21. /*Execution continues here after wakeup*/
```

Если источник пробуждения не включен, ESP32-C3 все равно может перейти в режим Light-sleep. Однако в этом случае ESP32-C3 останется в режиме Light-sleep до тех пор, пока не будет выполнен внешний сброс чипа.

4. Автоматический переход в режим Light-sleep

ESP32-C3 можно настроить на автоматический переход в режим Light-sleep, когда он находится в режиме ожидания, не нуждается в RF-модуле все время и автоматически просыпается, когда ему нужно работать (например, поддерживать соединения Wi-Fi и Bluetooth LE или получение данных). В этом случае пользователям не нужно вручную переводить чип в режим Light-sleep или настраивать источник пробуждения отдельно. После настройки на автоматический переход в режим Light-sleep ESP32-C3 может поддерживать соединение с маршрутизатором и получать интерактивную информацию от смартфона или сервера через маршрутизатор, тем самым улучшая взаимодействие с пользователем. Подключение Bluetooth LE аналогично подключению к роутеру. Обычно автоматический режим Light-sleep используется в сочетании с режимом Modem-sleep и управлением питанием. Когда RF-модуль не требуется, ESP32-C3 сначала переходит в режим Modem-sleep. Если в это время он простаивает, ESP32-C3 перейдет в режим Light-sleep для дальнейшего снижения энергопотребления. Режим Modem-sleep ESP32-C3 показан на рис. 12.2.

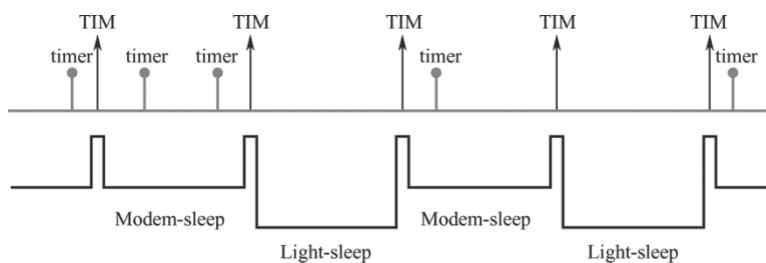


Рисунок 12.2. Режим Modem-sleep ESP32-C3

Автоматический режим Light-sleep может быть полезен в задачах, требующих от ESP32-C3 поддержки соединения с маршрутизатором и ответов на данные, отправленные маршрутизатором, в режиме реального времени, но контроллер может бездействовать, пока данные не получены. Например, в приложении умного Wi-Fi-выключателя ЦП большую часть времени простаивает, пока не получит команду управления. Только после получения этой команды ЦП активируется и управляет выключателем для включения или выключения.

5. Инструкция по автоматическому входу в режим Light-sleep

Чтобы настроить включение автоматического режима Light-sleep, пользователи могут вызвать функцию `esp_pm_configure()` и установить для параметра `light_sleep_enable` значение `true`. Когда эта функция включена, обратите внимание, что также необходимо настроить параметры `CONFIG_FREERTOS_USE_TICK-`

LESS_IDLE и CONFIG_PM_ENABLE. Для настройки параметра CONFIG_PM_ENABLE пользователи могут запустить команду `idf.py menuconfig` для старта инструмента настройки, перейти в **Component config** (Конфигурация компонента) → **Power Management** (Управление питанием), затем включить поддержку управления питанием **Support for power management**. Функция управления конфигурацией питания ESP32-C3 представлена на рис. 12.3.

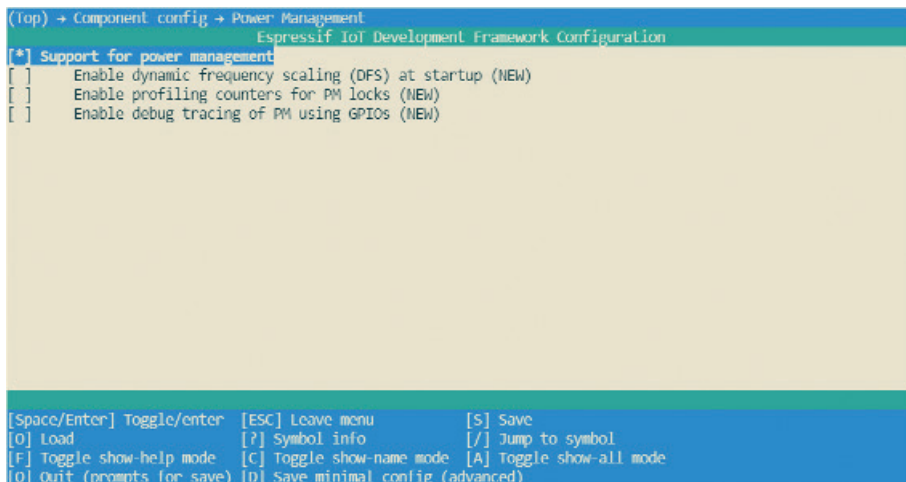


Рисунок 12.3. Конфигурация управления питанием ESP32-C3

Пользователи могут включить функцию динамического масштабирования частоты (DFS) и настроить чип для автоматического перехода в режим Light-sleep через вызов функции `esp_pm_configure()`. При использовании ESP32-C3 соответствующий параметр этой функции – `esp_config_esp32c3_t` структура, определяющая соответствующие настройки DFS и контролирующая, может ли чип автоматически входить в режим Light-sleep. В приведенной выше структуре следующие три переменные (поля) необходимо инициализировать:

- `max_freq_mhz`: максимальная частота ЦП в МГц, т. е. частота, используемая для запроса блокировки `ESP_PM_CPU_FREQ_MAX`. Обычно для этого поля установлено значение `CONFIG_ESP32C3_DEFAULT_CPU_FREQ_MHZ`;
- `min_freq_mhz`: минимальная частота ЦП в МГц, т. е. частота, используемая только при запросе блокировки `ESP_PM_APB_FREQ_MAX`. В этом поле можно установить значение частоты XTAL или частоты XTAL, разделенной на целое число. Обратите внимание, что 10 МГц – это самая низкая частота, на которой может генерироваться тактовый сигнал `REF_TICK`, по умолчанию равный 1 МГц;
- `light_sleep_enable`: должен ли ESP32-C3 автоматически переходить в режим Light-sleep, когда блокировки не установлены (тип значения `true/false`).

Автоматический режим Light-sleep основан на функциональности FreeRTOS Tickless Idle. Если режим Light-sleep запрашивается автоматически, пока опция `CONFIG_FREERTOS_USE_TICKLESS_IDLE` не включена в `menuconfig`, функция `esp_pm_configure()` вернет ошибку `ESP_ERR_NOT_SUPPORTED`. Конфигурация функции FreeRTOS Tickless Idle ESP32-C3 показана на рис. 12.4.

```

(Top) → Component config → FreeRTOS
+++++ Espressif IoT Development Framework Configuration
FreeRTOS assertions (abort() on failed assertions) --->
(2304) Idle Task stack size
(1536) ISR stack size
[ ] Use FreeRTOS legacy hooks
(16) Maximum task name length
[ ] Enable static task clean up hook
(1) FreeRTOS timer task priority
(2048) FreeRTOS timer task stack size
(10) FreeRTOS timer queue length
(8) FreeRTOS queue registry size
[ ] Enable FreeRTOS trace facility
[ ] Enable FreeRTOS to collect run time stats
* Tickless idle support
(3) Minimum number of ticks to enter sleep mode for (NEW)
[*] Enclose all task functions in a wrapper function
[*] Check that mutex semaphore is given by owner task
[ ] Tests compliance with Vanilla FreeRTOS port*_CRITICAL calls
Place FreeRTOS functions into Flash

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                    [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)

```

Рисунок 12.4. Конфигурация свойств FreeRTOS Tickless Idle ESP32-C3

```

1. #if CONFIG_PM_ENABLE
2.     /* Configure dynamic frequency scaling:
3.     automatic light sleep is enabled if tickless idle support is enabled.*/
4.     esp_pm_config_ESP32-C3_t pm_config = {
5.         .max_freq_mhz = 160, //Maximum CPU frequency
6.         .min_freq_mhz = 10, //Minimum CPU frequency
7.     #if CONFIG_FREERTOS_USE_TICKLESS_IDLE
8.         .light_sleep_enable = true
9.     #endif
10.    };
11.    ESP_ERROR_CHECK( esp_pm_configure(&pm_config) );
12. #endif //CONFIG_PM_ENABLE

```

12.2.3. Режим глубокого сна Deep-sleep

По сравнению с режимом Light-sleep, ESP32-C3 не может автоматически переходить в режим глубокого сна Deep-sleep. Вместо этого пользователи должны вызвать функцию `esp_deep_sleep_start()`, чтобы отправить чип в режим Deep-sleep. В режиме Deep-sleep ESP32-C3 не поддерживает соединения Wi-Fi и Bluetooth LE и отключает ЦП, большую часть оперативной памяти и все цифровые периферийные устройства, работающие на тактовой частоте шины APB CLK. Однако контроллеры часов RTC, периферийные устройства RTC и быстрая память RTC могут функционировать. После выхода из режима Deep-sleep процессор ESP32-C3 перезагрузится и перезапустится с нуля.

1. Источники пробуждения в режиме глубокого сна

В режиме Deep-sleep ESP32-C3 может использовать GPIO или таймер в качестве источников пробуждения и поддерживает до двух источников пробуждения одновременно. В этом случае ESP32-C3 будет разбужен, когда срабатывает любой из источников пробуждения. Прежде чем войти в режим Deep-sleep, пользователи могут в любое время либо настроить источник пробуждения, используя соответствующий API, либо отключить источник пробуждения с помощью

функции `esp_sleep_disable_wakeup_source()`. После пробуждения пользователи могут определить, какой источник пробуждения был ответственен за пробуждение чипа, вызвав функцию `esp_sleep_get_wakeup_cause()`.

Пробуждение GPIO

Любой вывод GPIO можно использовать в качестве внешнего входа для вывода чипа из режима Deep-sleep. Каждый вывод можно индивидуально настроить для запуска пробуждения на высоком или низком уровне с помощью функции `esp_deep_sleep_enable_gpio_wakeup()`. Важно отметить, что пробуждение от GPIO доступно только для RTC IO.

Пробуждение по таймеру

Контроллер RTC имеет встроенный таймер, который можно использовать для пробуждения чипа через заранее определенный промежуток времени. Время указывается с точностью до микросекунды, но фактическое разрешение зависит от источника синхронизации, выбранного для `RTC_SLOW_CLK`. Когда пробуждение по таймеру включено, периферийные устройства RTC или память RTC не нужно включать во время пребывания ESP32-C3 в режиме сна, а пробуждение по таймеру можно включить, вызвав `esp_sleep_enable_timer_wakeup()`.

2. Инструкция по входу в режим глубокого сна

После настройки источника пробуждения пользователи могут вызвать `esp_deep_sleep_start()`, чтобы войти в режим Deep-sleep. Если источник пробуждения не включен, ESP32-C3 все равно может перейти в режим глубокого сна. Однако в этом случае ESP32-C3 останется в режиме глубокого сна до внешнего сброса чипа.

Следующий код показывает, как настроить режим Deep-sleep в ESP32-C3.

- Источник пробуждения: GPIO и таймер;
- вывод GPIO4 настроен на пробуждение при высоком уровне на нем;
- предопределенное время для пробуждения чипа с помощью таймера составляет 20 с.

Учитывая, что вывод GPIO4 пробуждает ESP32-C3 на высоком уровне, необходимо добавить заземляющий резистор в аппаратной схеме или настроить программное обеспечение так, чтобы избежать ложного пробуждения.

```
1. #define DEFAULT_WAKEUP_PIN 4
2. #define DEFAULT_WAKEUP_LEVEL ESP_GPIO_WAKEUP_GPIO_HIGH
3.
4. const gpio_config_t config = {
5.     .pin_bit_mask = BIT(DEFAULT_WAKEUP_PIN),
6.     .mode = GPIO_MODE_INPUT,
7. };
8. ESP_ERROR_CHECK(gpio_config(&config));
9. ESP_ERROR_CHECK(esp_deep_sleep_enable_gpio_wakeup(BIT(DEFAULT_WAKEUP_PIN),
10.     DEFAULT_WAKEUP_LEVEL));
11. ESP_LOGI("TAG", "Enabling GPIO wakeup on pins GPIO%d\n",
12.     DEFAULT_WAKEUP_PIN);
13.
14. const int wakeup_time_sec = 20;
15. ESP_LOGI("TAG", "Enabling timer wakeup, %ds\n", wakeup_time_sec);
```

```

16. esp_sleep_enable_timer_wakeup(wakeup_time_sec * 1000000);
17.
18. /*Enter deep sleep*/
19. esp_deep_sleep_start();

```

12.2.4. Потребление тока в различных режимах питания

Измерения потребления тока проводятся для питания 3,3 В при температуре окружающей среды 25 °С в RF-порту. Все измерения передатчиков основаны на 100%-ном коэффициенте заполнения.

Потребление тока в зависимости от режимов RF- модуля показано в табл. 12.3.

Таблица 12.3. Потребление тока в зависимости от режимов RF- модуля

Режим работы	Описание		Пиковое потребление (мА)
Активный (RF работает)	TX	IEEE 802.11b, 1 Mbit/s, @21 dBm	335
		IEEE 802.11g, 54 Mbit/s, @19 dBm	285
		IEEE 802.11n, HT20, MCS7, @18.5 dBm	276
		IEEE 802.11n, HT40, MCS7, @18.5 dBm	278
	RX	IEEE 802.11b/g/n, HT20	84
		IEEE 802.11n, HT20	87

Потребление тока в других режимах представлено в табл. 12.4.

Таблица 12.4. Потребление тока в других режимах

Режим работы	Описание		Типичное значение	Единицы
Modem-sleep ^{1,2}	CPU работает ³	160 MHz	20	mA
		80 MHz	15	mA
Light-sleep	–		130	µA
Deep-sleep	Таймер RTC + память RTC		5	µA
Питание откл.	Вывод CHIP_EN установлен на низкий уровень, т. е. чип выключен		1	µA

Примечания к таблице:

¹ При измерении энергопотребления в режиме Modem-sleep ЦП работает и кеш-память простаивает.

² В сценарии при включенном Wi-Fi чип переключается между активным режимом и режимом Modem-sleep, потребление тока также будет переключаться между двумя режимами.

³ В режиме Modem-sleep частота процессора изменяется автоматически, она зависит от загрузки процессора и используемой периферии.

12.3. Управление питанием и отладка режима низкого энергопотребления

Обратите внимание, что фактическое энергопотребление может превышать теоретические значения из-за различных факторов, в том числе:

- Wi-Fi или Bluetooth LE принимает данные;
- приложение устанавливает блокировку управления питанием на длительное время и не снимает ее;
- в приложении имеется блокировка процессов (не связанная с вызовом API операционной системы);
- заданный интервал для пробуждения чипа через таймер слишком мал или прерывания происходят слишком часто.

Рекомендуется определить основную причину и поискать оптимизацию, если энергопотребление превышает теоретические значения в течение значительного периода времени. Пользователи могут определить основную причину через отладочные логи и анализ состояний GPIO или могут дополнительно использовать некоторые анализаторы сетевых протоколов, если высокое энергопотребление относится к Wi-Fi и Bluetooth LE. Обратите внимание, что этот процесс может повторяться столько раз, сколько необходимо, пока не будут выполнены фактические требования к продукту.

В следующих разделах описаны наиболее часто используемые методы оптимизации энергопотребления: отладка через анализ логов и состояний GPIO – и продемонстрировано, как реализовать эти методы на практике с получением данных об энергопотреблении в реальном времени.

12.3.1. Отладка через логи

При использовании метода отладки через логи пользователям необходимо в меню конфигурации `menuconfig` настроить параметр `CONFIG_PM_PROFILING`, чтобы отслеживать время действия каждой блокировки управления питанием, затем вызвать функцию `esp_pm_dump_locks (FILE* stream)` для вывода такого лога. Отладка через лог позволяет пользователям проанализировать, какие блокировки управления питанием препятствуют входу чипа в режим пониженного энергопотребления, и проверить, сколько времени чип проводит в каждом режиме энергопотребления. После отладки пользователи должны отключить `CONFIG_PM_PROFILING` в меню `config`.

Чтобы настроить `CONFIG_PM_PROFILING`, пользователям необходимо запустить команду `idf.py menuconfig`. Чтобы запустить инструмент настройки, перейдите в **Component config** (Конфигурация компонентов) → **Power Management** (Управление питанием) и включите пункт `Enable profiling counters for PM locks` (Включить счетчики профилирования для блокировок PM). Скриншот включения отладочного лога для ESP32-C3 показан на рис. 12.5.

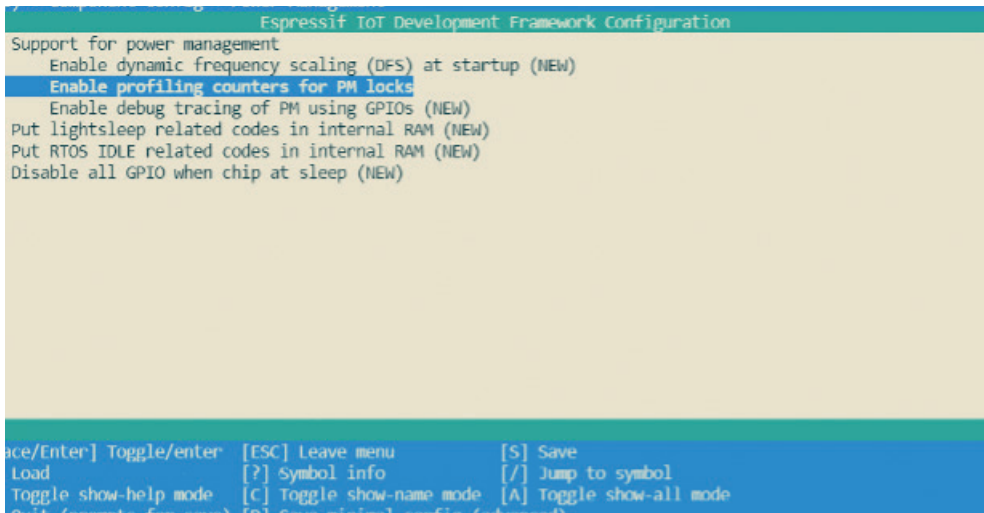


Рисунок 12.5. Конфигурация отладочного лога для низкого энергопотребления ESP32-C3

Если ESP32-C3 настроен на автоматический переход в режим Light-sleep, пользователи должны вызвать функцию `esp_pm_dump_locks (FILE* stream)`, периодически выводить лог отладки и проанализировать первопричину повышенного энергопотребления. Некоторая отладочная информация из лога представлена ниже:

```
Time: 11879660
Lock stats:
name          type          arg   cnt   times      time      percentage
wifi          APB_FREQ_MAX  0     0    107      1826662   16%
bt            APB_FREQ_MAX  0     1    126      5367607   46%
rtos0         CPU_FREQ_MAX  0     1   8185      809685    7%
Mode stats:
name          HZ      time      percentage
SLEEP         40M     4252037   35%
APB_MIN        40M      0         0%
APB_MAX        80M     6303881  53%
CPU_MAX        160M    823595    6%
```

Функция `esp_pm_dump_locks (FILE* stream)` выводит два типа отладочной информации: статистику блокировки `Lock stats` и статистику режима `Mode stats`.

В разделе `Lock stats` перечислены статусы в реальном времени всех блокировок управления питанием, используемых приложением, со следующей информацией: имя (`name`), тип блокировки управления питанием (`type`), параметр (`arg`), количество запросов блокировки управления питанием на данный момент (`cnt`), общее количество запросов блокировки (`times`), общее количество времени, в течение которого блокировка была запрошена (`time`), а также относительная доля времени запрошенного состояния (`percentage`). В разделе `Mode stats` отображается в реальном времени статус различных режимов приложения со следующей информацией: название режима (`name`), системная тактовая частота (Hz), общее количество времени нахождения в режиме (`time`) и процент времени нахождения в режиме (`percentage`).

Рассмотрев приведенный выше пример лога, пользователи могут легко узнать, что для питания Wi-Fi блокировка управления APB_FREQ_MAX:

- в настоящее время не запрошена;
- общее время, в течение которого была запрошена эта блокировка, составляет 1 826 662 мкс;
- общее количество запросов этой блокировки составляет 107 раз;
- доля времени, когда эта блокировка была запрошена, составляет 16 %.

Аналогичным образом пользователи также могут узнать про блокировку управления питанием CPU_FREQ_MAX:

- в настоящее время запрошена;
- общее время, в течение которого была запрошена эта блокировка, составляет 809 685 мкс;
- общее количество запросов блокировки составляет 8185 раз;
- доля времени, когда эта блокировка была запрошена, составляет 7 %.

Кроме того, пользователи могут аналогичным образом прочитать информацию лога о статистике режима. Например, в спящем режиме тактовая частота равна 40 МГц, общее время пребывания в этом режиме 4 252 037 мкс, что составляет 35 % всего времени. Аналогичным образом пользователи могут узнать из лога о других блокировках и их состояниях.

12.3.2. Отладка по состояниям GPIO

При использовании отладки по состояниям выводов GPIO пользователям необходимо перейти в меню конфигурации `menuconfig` и включить `CONFIG_PM_TRACE`. Если этот параметр включен, некоторые GPIO будут использоваться для сигнализации о таких событиях, как такты RTOS, частота переключения, вход/выход из состояния ожидания. Список GPIO см. в файле `pm_trace.c`. Эта особенность предназначена для использования при анализе и отладке поведения управления питанием и должна быть отключена после отладки.

Соответствующие GPIO показаны ниже, по два GPIO для каждого события, соответствующего CPU0 и CPU1. Поскольку ESP32-C3 – одноядерный чип, при отладке в данном случае эффективен только первый столбец GPIO. Во время разработки пользователи также могут изменять используемый GPIO, редактируя исходный код. Перед отладкой подключите выбранный GPIO к такому инструменту, как логический анализатор или осциллограф.

```
1. /*GPIOs to use for tracing of esp_pm events.
2. * Two entries in the array for each type, one for each CPU.
3. * Feel free to change when debugging.
4. */
5. static const int DRAM_ATTR s_trace_io[] = {
6. #ifndef CONFIG_IDF_TARGET_ESP32C3
7.     BIT(4), BIT(5), //ESP_PM_TRACE_IDLE
8.     BIT(16), BIT(17), //ESP_PM_TRACE_TICK
9.     BIT(18), BIT(18), //ESP_PM_TRACE_FREQ_SWITCH
10.    BIT(19), BIT(19), //ESP_PM_TRACE_CCOMPARE_UPDATE
11.    BIT(25), BIT(26), //ESP_PM_TRACE_ISR_HOOK
12.    BIT(27), BIT(27), //ESP_PM_TRACE_SLEEP
```



```

13. #else
14.     BIT(2), BIT(3), //ESP_PM_TRACE_IDLE
15.     BIT(4), BIT(5), //ESP_PM_TRACE_TICK
16.     BIT(6), BIT(6), //ESP_PM_TRACE_FREQ_SWITCH
17.     BIT(7), BIT(7), //ESP_PM_TRACE_CCOMPARE_UPDATE
18.     BIT(8), BIT(9), //ESP_PM_TRACE_ISR_HOOK
19.     BIT(18), BIT(18), //ESP_PM_TRACE_SLEEP
20. #endif
21. };

```

Чтобы включить CONFIG_PM_PROFILING, пользователям необходимо запустить команду `idf.py menuconfig` для запуска инструмента настройки, затем перейти в **Component config** (Конфигурация компонента) → **Power Management** (Управление питанием) и включить пункт отслеживания отладки с помощью GPIO «Enable debug tracing of PM using GPIOs». Скриншот, как это делается для ESP32-C3, показан на рис. 12.6.

Отладку можно начать после завершения вышеупомянутой настройки. Наблюдая различные состояния GPIO, пользователи увидят текущее состояние ЦП и соответствующий режим питания, а также могут понять, какие режимы питания потребляют больше мощности и могут быть оптимизированы. На рис. 12.7 представлена форма отладочного сигнала GPIO ESP32-C3. Верхняя часть показывает в реальном времени потребление ESP32-C3, а в нижней части представлена форма сигнала GPIO, соответствующая событию ESP_PM_TRACE_SLEEP.

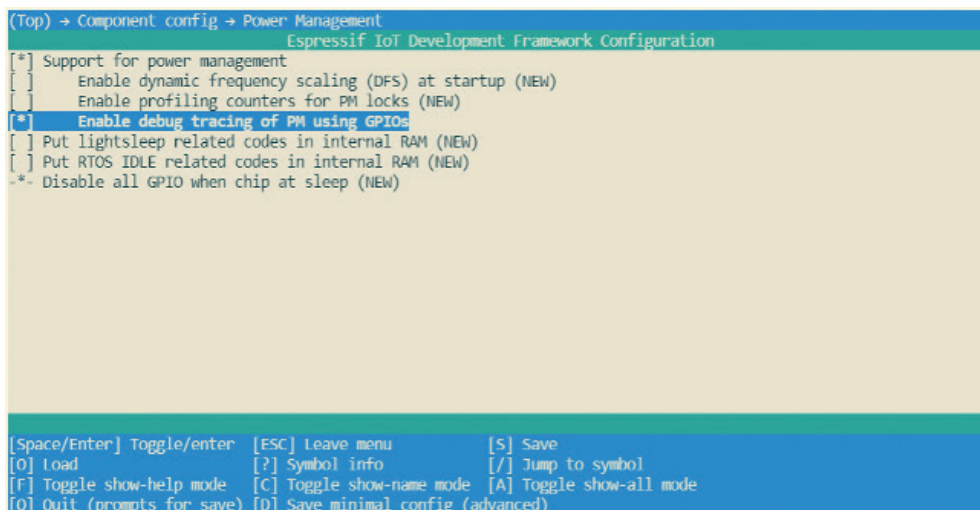


Рисунок 12.6. Включение отладки пониженного энергопотребления ESP32-C3 с помощью GPIO

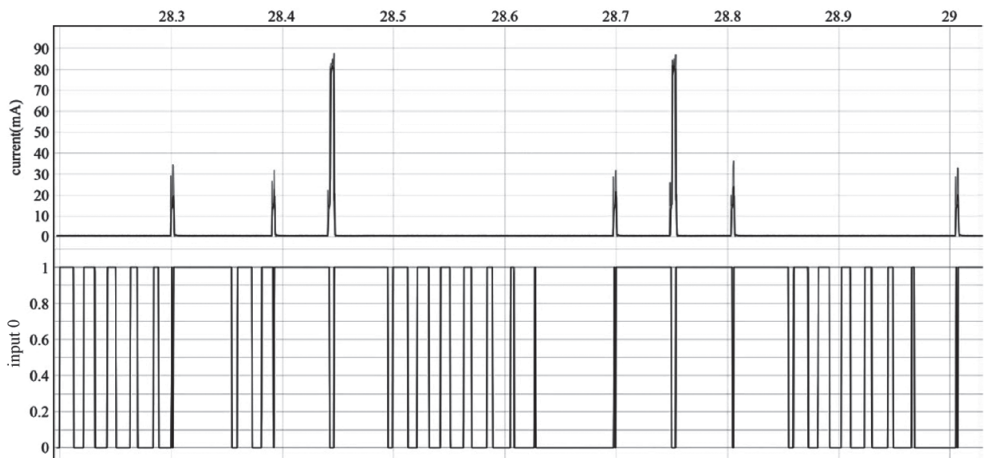


Рисунок 12.7. Форма сигнала отладки с помощью GPIO

12.4. Практика: управление питанием в проекте Smart Light

После изучения функции управления питанием ESP32-C3 и различных режимов пониженного энергопотребления пользователи могут реализовать различные системы управления питанием для снижения энергопотребления во время разработки своих реальных проектов интернета вещей. В этом разделе рассказывается, как оптимизировать энергопотребление проекта Smart Light, который также используется в качестве примера в других разделах данной книги, реализовав систему управления питанием ESP32-C3 и используя различные режимы пониженного энергопотребления.

Чтобы сэкономить энергию в целом или получить сертификацию энергопотребления, при работе над проектом Smart Light необходимо максимально снизить энергопотребление ESP32-C3. Как объяснено в разделах 12.1 и 12.2, когда ESP32-C3 работает в режиме глубокого сна, функция LEDC не работает должным образом, и не могут сохраняться соединения через Wi-Fi и BLE. В результате ESP32-C3 не может получать команды управления от пользователя. Таким образом, для минимизации энергопотребления в проектах интеллектуального освещения обычно используется система управления питанием, применяющая автоматический режим Light-sleep для Wi-Fi и Bluetooth Low Energy. После реализации этой системы:

- когда свет горит, срабатывает блокировка управления питанием, чтобы гарантировать, что светодиод LEDC работает нормально, а Wi-Fi и Bluetooth LE остаются подключенными для приема пользовательских команд управления. При использовании режима Modem-sleep время работы RF-модуля Wi-Fi и Bluetooth LE может быть уменьшено, чтобы уменьшить энергопотребление;
- когда свет выключен, блокировка управления питанием снимается, и процессор может перейти в режим Light-sleep, в котором он находится в режиме ожидания, чтобы еще больше снизить энергопотребление.

Реализация этой системы в проекте Smart Light включает в себя два этапа:

- (1) настройте функцию управления питанием ESP32-C3, включите автоматический режим Light-sleep, включите спящий режим Modem-sleep для Wi-Fi и Bluetooth LE;
- (2) завершите операцию блокировки управления питанием в приложении, чтобы драйвер регулировки LEDC работал правильно.

Чтобы узнать, как оптимизировать энергопотребление для существующих проектов, пожалуйста, посетите book-esp32c3-iot-projects/device_firmware/6_project_optimize.

12.4.1. Настройка функции управления питанием

- (1) Включение функции управления питанием и автоматического режима Light-sleep.

При включении функции управления питанием пользователям сначала необходимо включить соответствующие параметры в меню config, затем вызвать функцию `esp_pm_configure()` (или `esp_pm_config_esp32c3_t` при разработке с использованием ESP32-C3). Инструкции о том, как включить автоматический режим Light-sleep, вы найдете в разделе 12.2.2.

- (2) Настройка режима Wi-Fi Modem-sleep и Bluetooth Modem-sleep.
 - Чтобы включить Bluetooth Modem-sleep, просто включите эту опцию в меню конфигурации, как показано на скриншоте ниже.
 - Чтобы включить Wi-Fi Modem-sleep, пользователям необходимо сначала инициализировать Wi-Fi, а затем вызвать `esp_wifi_set_ps` (тип `wifi_ps_type_t`). Код для включения режима Wi-Fi Modem-sleep для проекта Smart Light выглядит следующим образом:

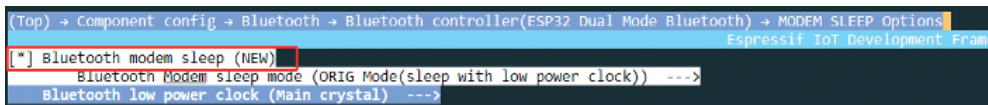


Рисунок 12.8. Включение Modem-sleep для Bluetooth LE ESP32-C3

```
1. #define LISTEN_INTERVAL 3
2. wifi_config_t wifi_config = {
3.     .sta = {
4.         .ssid = "SSID",
5.         .password = "Password",
6.         .listen_interval = LISTEN_INTERVAL,
7.     },
8. };
9. ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
10. ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
11. ESP_ERROR_CHECK(esp_wifi_start());
12.
13. ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_MAX_MODEM));
```

12.4.2. Использование блокировки управления питанием

Из раздела 12.1.1 мы знаем, что при синхронизации от источников тактирования, кроме REF_TICK, на LEDC в ESP32-C3 влияет динамическое масштабирование частоты и в код приложения для включения/выключения питания необходимо добавить блокировки управления, чтобы LEDC мог работать правильно.

Поэтому в приложении требуется блокировка управления питанием, дабы гарантировать, что тактовая частота APB не меняется во время работы LEDC. Конкретно:

- во время инициализации драйвера LEDC инициализируйте блокировку управления питанием ESP_PM_APB_FREQ_MAX;
- когда LEDC начнет работать (свет загорится), включите блокировку управления питанием;
- а когда светодиод LEDC перестанет работать (свет погаснет), выключите блокировку управления питанием.

Код для включения Wi-Fi Modem-sleep в проект Smart Light выглядит следующим образом:

```
1. static esp_pm_lock_handle_t s_pm_apb_lock = NULL;
2.
3. if (s_pm_apb_lock == NULL) {
4.     if (esp_pm_lock_create(ESP_PM_APB_FREQ_MAX, 0, "l_apb",
5.         &s_pm_apb_lock) != ESP_OK) {
6.         ESP_LOGE(TAG, "esp pm lock create failed");
7.     }
8. }
9.
10. while (1) {
11.     ESP_ERROR_CHECK(esp_pm_lock_acquire(s_pm_apb_lock));
12.     ESP_LOGI(TAG, "light turn on");
13.     for (ch = 0; ch < LEDC_TEST_CH_NUM; ch++) {
14.         ledc_set_duty(ledc_channel[ch].speed_mode,
15.             ledc_channel[ch].channel, LEDC_TEST_DUTY);
16.         ledc_update_duty(ledc_channel[ch].speed_mode,
17.             ledc_channel[ch].channel);
18.     }
19.     vTaskDelay(pdMS_TO_TICKS(5 * 1000));
20.
21.     ESP_LOGI(TAG, "light turn off");
22.     for (ch = 0; ch < LEDC_TEST_CH_NUM; ch++) {
23.         ledc_set_duty(ledc_channel[ch].speed_mode,
24.             ledc_channel[ch].channel, 0);
25.         ledc_update_duty(ledc_channel[ch].speed_mode,
26.             ledc_channel[ch].channel);
27.     }
28.     ESP_ERROR_CHECK(esp_pm_lock_release(s_pm_apb_lock));
29.     vTaskDelay(pdMS_TO_TICKS(5 * 1000));
30. }
```

12.4.3. Проверка энергопотребления

После завершения описанной настройки для снижения энергопотребления следует проверить фактическое энергопотребление и его соответствие требованиям. Согласно требованиям сертификации, фактическое тестируемое устройство (Device Under Test, DUT) может представлять собой светильник целиком или только модуль ESP32-C3. Когда модуль ESP32-C3 выбран в качестве DUT, между источником питания и микросхемой можно добавить анализатор мощности для измерения данных об энергопотреблении. В этой книге используется анализатор Joulescope⁴⁵: высокоточный анализатор мощности постоянного тока. Среди сертификационных требований, касающихся энергопотребления, для умных осветительных устройств часто необходимо измерять средний ток, когда светильник выключен, а Wi-Fi подключен.

После реализации представленной выше системы управления питанием средний ток модуля ESP32-C3 составляет 2,24 мА (см. рис. 12.9). Обратите внимание, что фактический результат теста может отличаться, поскольку на рис. 12.9 показано энергопотребление модуля ESP32-C3 только за короткий период времени.

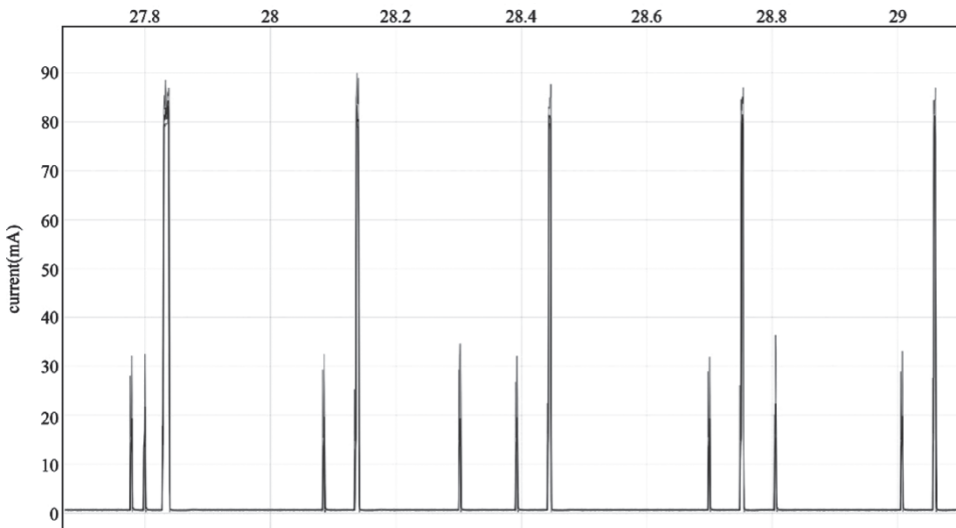


Рисунок 12.9. Потребляемый ток модуля ESP32-C3

12.5. Резюме

В этой главе мы представили функцию управления питанием ESP32-C3 и поддержку режимами пониженного энергопотребления (Modem-sleep, Light-sleep, Deep-sleep), а затем описали, как выполнить отладку пониженного энергопотребления, и, наконец, узнали, как использовать функции управления питанием и режимы низкого энергопотребления на практике, привели способы измерения фактического энергопотребления на примере проекта Smart Light.

⁴⁵ <https://www.joulescope.com/products/js220-joulescope-precision-energy-analyzer>.

Глава 13

Расширенные функции безопасности устройства

Безопасность данных – сложная и постоянно развивающаяся тема исследований. Целью этой главы является ознакомление читателя с базовыми представлениями о безопасности данных. В разделе 13.1 мы обсудим угрозы безопасности данных для устройств интернета вещей и представим базовые основы защиты данных. В разделе 13.2 представлена система проверки целостности прошивки устройств IoT. В разделе 13.3 представлены две системы шифрования – Flash Encryption и NVS Encryption, обеспечивающие конфиденциальность данных. В разделе 13.4 описывается система безопасной загрузки (Secure Boot), обеспечивающая легитимность данных прошивки IoT-устройства. Наконец, в разделе 13.5 рассматривается эффективность сочетания шифрования Flash Encryption с системой безопасной загрузки Secure Boot и даются рекомендации по использованию этих методов в серийном производстве устройств.

13.1. Обзор безопасности данных IoT-устройств

В предыдущих главах было описано, как обеспечить безопасность данных во время передачи с использованием протокола HTTPS. Хотя данные защищены во время передачи, они могут по-прежнему сталкиваться с различными угрозами на стороне устройства.

Нам нужна безопасность данных на наших устройствах, но что именно означает «безопасность»? Как мы узнаем в этой главе, безопасность – это многогранная концепция, охватывающая множество разных аспектов. Как минимум безопасность данных должна включать следующие ключевые компоненты:

Конфиденциальность

Только авторизованные разработчики могут получить доступ к реальному содержанию данных. В системе умного света такие данные, как пароль Wi-Fi, данные для входа в систему пользователя и идентификатор устройства, необходимо защищать от несанкционированного доступа или раскрытия.

Целостность

Данные могут быть злонамеренно подделаны на этапах передачи и хранения, а ошибки в коде могут возникнуть случайно. Следовательно, устройство должно иметь возможность проверять полноту данных для обеспечения их целостности. Для умного светильника такие данные, как новая прошивка и сохраненные сетевые сертификаты, полученные во время обновлений по беспроводной

сети (ОТА), должны быть проверены на целостность перед загрузкой и использованием. Эта проверка необходима для предотвращения загрузки и использования данных, которые были подделаны или содержат ошибки.

Легитимность

Важно, чтобы устройства, получающие данные, имели возможность аутентифицировать отправителя и принимать данные только из законных источников. Что касается умного освещения, команды управления и обновления прошивки должны исходить только от авторизованных устройств. Представьте себе, если кто-нибудь сможет управлять светом в вашей спальне с помощью своего смартфона – вы будете спать хорошо?

Итак, какие типы данных необходимо защищать на устройстве?

Данные прошивки

Прошивка представляет собой исполняемый бинарный файл, работающий на устройстве, отвечающий за координацию ресурсов системы и обеспечение обмена данными между устройством и внешней системой. Безопасность встроенного ПО так же важна, как и безопасность операционной системы на ПК. Если безопасность прошивки нарушена, это может серьезно повлиять на нормальные функции устройства. Поэтому крайне важно обеспечить безопасность данных прошивки. В случае умных светильников на базе ESP32-C3 прошивка обычно включает загрузчик и хранится во флеш-памяти.

Ключевые данные, используемые устройством

Например, ключ для подключения к облаку, ключ для входа в устройство и т. д.

13.1.1. Зачем защищать данные устройств интернета вещей?

Данные IoT-устройств могут подвергаться различным угрозам безопасности во время передачи и хранения. На рис. 13.1 показан обмен данными между устройством и облаком, где облако отправляет данные на устройство, а устройство получает и сохраняет данные в своей флеш-памяти. Для обеспечения безопасности данных и соблюдения конфиденциальности, целостности и легитимности для передачи данных обычно используется шифрованный протокол HTTPS. Однако существует еще вероятность того, что злоумышленники потенциально могут поставить под угрозу безопасность данных, участвуя в следующих действиях:

- нарушение **конфиденциальности данных**, например использование esptool.py для чтения данных в прошивке, воровства идентификатора устройства и пароля Wi-Fi;
- нарушение **целостности данных**, например удаление данных входа пользователя из флеш-памяти устройства, подделка сетевых сертификатов или внедрение программ, собирающих информацию о пользователях. Этот вид атаки, при которой вредоносный код внедряется в исходный код, называется атакой с внедрением кода;
- нарушение **легитимности данных**, например подделка облачного сервера и незаконная отправка данных на устройство или подслушивание

безопасной сетевой связи, а затем повторная отправка данных для входа на устройство и управления им. Атака с нарушением легитимности данных путем «воспроизведения» украденных данных называется атакой воспроизведения.

На рис. 13.1 показаны риски безопасности, с которыми может столкнуться устройство при обмене данными с облаком.

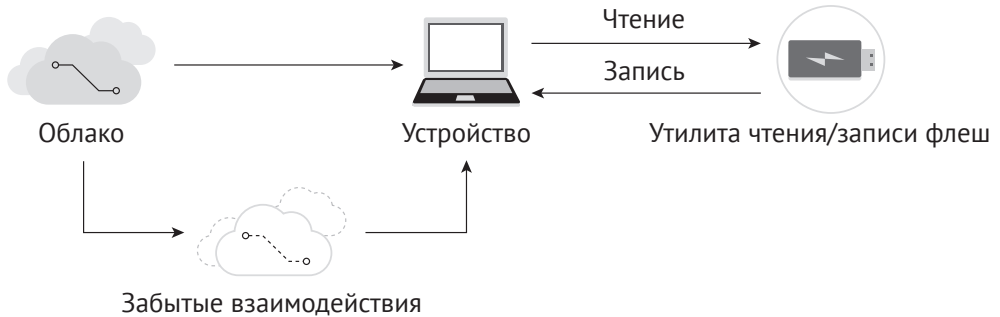


Рисунок 13.1. Риски безопасности при обмене данными с облаком

Как мы видим, безопасность данных устройства не может быть гарантирована, если не будут приняты надлежащие меры. В реальных приложениях угрозы для IoT-устройств гораздо сложнее, чем обсуждавшиеся выше. Поскольку IoT-устройства часто взаимодействуют с другими устройствами, а некоторые из них могут работать в автоматических средах, злоумышленникам становится легче получать, анализировать или подделывать данные устройства. Поэтому обеспечение безопасности данных стало еще более актуальной задачей.

13.1.2. Основные требования к безопасности данных IoT-устройств

Защита безопасности данных устройств интернета вещей должна обсуждаться с двух точек зрения: хранение данных и передача данных. Они соответственно предлагают требования по защите целостности, конфиденциальности и легитимности данных. Показаны основные компоненты безопасности данных на рис. 13.2.

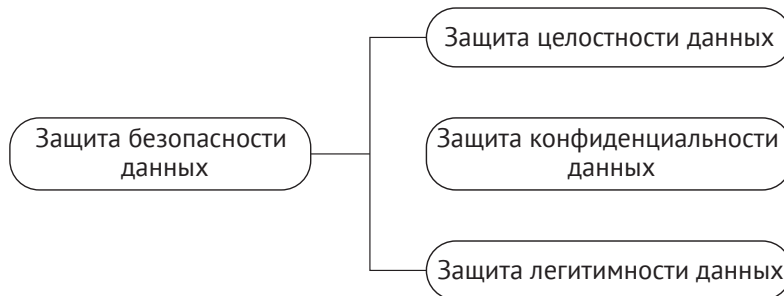


Рисунок 13.2. Основные компоненты безопасности данных

Требования к безопасности передачи данных в основном включают в себя следующие три аспекта:

- целостность: данные не должны быть подделаны или повреждены во время передачи;
- конфиденциальность: передаваемые данные зашифрованы, и злоумышленники не могут получить доступ к реальному содержанию данных;
- легитимность: взаимодействующее устройство является доверенным целевым устройством.

Требования к безопасности хранения данных включают в себя те же три аспекта:

- целостность: данные не должны быть подделаны или повреждены во время хранения;
- конфиденциальность: после прочтения сохраненных данных злоумышленники не смогут расшифровать реальное содержание данных;
- легитимность: используемые данные аутентифицированы.

Конечно, безопасность при хранении и безопасность при передаче данных не являются полностью независимыми. Они дополняют друг друга и вместе составляют единую основу безопасности данных IoT-устройств. После установления вышеуказанной структуры, разъяснения некоторых концепций и требований к безопасности данных устройств эта глава далее шаг за шагом объясняет, как защитить данные IoT-устройств.

13.2. Защита целостности данных

13.2.1. Основы метода проверки целостности

Для проверки целостности данных используется блок данных, называемый контрольной суммой (также известный как дайджест, отпечаток пальца, хеш-значение или хеш-код). Контрольная сумма представляет собой генерируемые соответствующим алгоритмом проверки целостности контрольные данные фиксированной длины. Контрольная сумма по существу представляет уникальность блока данных, так же, как отпечаток пальца или идентификационный номер может однозначно представлять человека. Алгоритм проверки целостности показан на рис. 13.3.



Текст сегмента данных Алгоритм проверки целостности Контрольная сумма

Рисунок 13.3. Алгоритм проверки целостности

Алгоритм проверки целостности обладает следующими свойствами:

Устойчивость к коллизиям

Устойчивость к коллизиям означает, что в пределах длины данных, заданной алгоритмом, невозможно (или очень сложно) найти два разных сег-

мента данных x и y , которые приводят к одной и той же контрольной сумме. Коллизия возникает, когда размер данных увеличивается, некоторые данные теряются или повреждены, но все равно получается правильная контрольная сумма.

Не могут быть извлечены необработанные данные

В случае когда контрольная сумма известна, по ней невозможно определить, что именно представляют собой исходные данные.

Коллизия возникает, когда в алгоритм вводятся разные фрагменты данных, что приводит к той же контрольной сумме. Общие алгоритмы проверки целостности включают CRC, MD5, SHA1 и SHA256. Эти алгоритмы генерируют контрольные суммы различной длины, что влияет на вероятность коллизии. Например, контрольная сумма CRC32 имеет длину 32 байта и теоретически может гарантировать, что данные в пределах 512 МБ не будут конфликтовать. Однако вероятность коллизий увеличивается за пределами этого диапазона.

Распространенный способ проверки целостности данных – добавление к данным контрольной суммы, которая может быть проверена. Рисунок 13.4 иллюстрирует основной принцип проверки целостности, когда контрольная сумма добавляется в конец блока данных. После получения блока данных или перед их использованием получатель пересчитывает контрольную сумму. Если вычисленная контрольная сумма соответствует присланной, данные считаются целыми, в противном случае считается, что данные были подделаны или содержат ошибки.

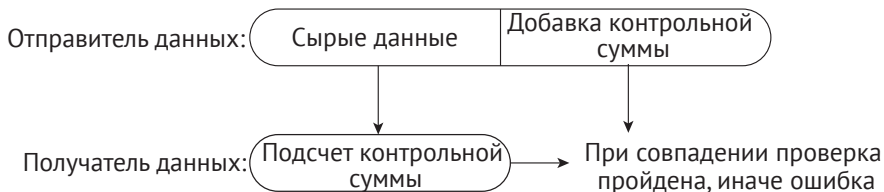


Рисунок 13.4. Основной принцип проверки целостности данных

13.2.2. Проверка целостности данных прошивки

В этом разделе проверка целостности данных прошивки во время OTA-обновлений рассматривается как пример, показывающий, как вообще устроена проверка целостности. Рисунок 13.5 показывает, что при обновлении прошивки проверка целостности выполняется перед передачей данных и загрузкой обновленной прошивки.

В процессе OTA-обновлений, если для передачи данных используется протокол HTTPS, отправитель перед передачей генерирует контрольную сумму CRC для данных, а получатель пересчитывает полученную CRC с последующей проверкой, аналогично процессу, показанному на рис. 13.4. Стоит отметить, что при использовании протокола HTTPS для передачи данных нет необходимости специально беспокоиться о проверке CRC, поскольку протокол HTTPS выполняет эту проверку автоматически.

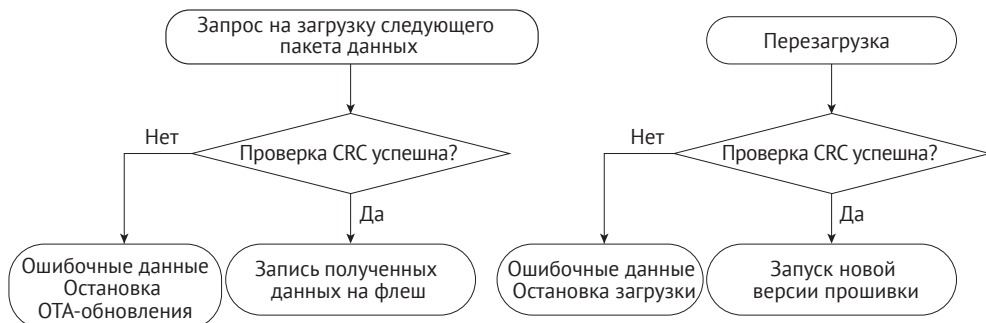


Рисунок 13.5. Проверка целостности перед передачей данных и загрузкой прошивки

Кроме того, когда устройство использует прошивку, хранящуюся во флеш-памяти, оно также проверяет целостность прошивки. Каждый раз, когда устройство перезагружается для загрузки прошивки приложения, оно будет выполнять проверку целостности, чтобы убедиться, что прошивка приложения для загрузки не повреждена. Этот процесс происходит автоматически и не требует ручного вмешательства.

Однако полагаться только на проверку целостности для обеспечения безопасности данных недостаточно. Поскольку механизмы и реализации алгоритмов проверки целостности обычно с открытым исходным кодом, злоумышленники могут использовать тот же алгоритм проверки CRC, чтобы добавить контрольную сумму CRC в заказную прошивку и прошить ее во флеш-память устройства, тем самым передав и проверочное значение CRC. Чтобы предотвратить подобные атаки, необходимо задать источник данных, включающий систему защиты легитимности – Secure Boot, о чем будет рассказано подробно в разделе 13.4.2.

13.2.3. Пример

В систему Linux интегрированы различные инструменты для расчета контрольных сумм, например `sha256sum` и `md5sum`. Мы можем использовать эти инструменты для вычисления контрольной суммы указанных файлов и сравнения контрольных сумм до и после изменения файла. Приведенные команды используют `md5sum` для расчета контрольных сумм файла `hello.c` до и после модификации:

```

$ md5sum hello.c
87cb921a75d4211a57ba747275e8bbe6 //Original MD5 checksum of hello.c
$ md5sum hello.c
79c3416910f9ea0d65a72cb720368416 //New MD5 checksum after adding one line
  
```

Видно, что изменение всего лишь одной строки кода в исходном файле приведет к значительной разнице в контрольных суммах MD5.

13.3. Защита конфиденциальности данных

13.3.1. Введение в шифрование данных

Целью шифрования данных является предотвращение того, чтобы посторонние лица узнали истинное содержание данных, в то же время позволяя авторизован-

ным пользователям правильно интерпретировать данные. Предположим, вы хотите зашифровать данные, хранящиеся во флеш-памяти, чтобы предотвратить несанкционированный доступ. Для начала вам необходимо ознакомиться с ключевыми понятиями, показанными на рис. 13.6. Оригинал данных, хранящихся во флеш-памяти, называется открытым текстом, а зашифрованные данные, генерируемые алгоритмом шифрования, называются шифрованным текстом. Этот шифрованный текст непонятен несанкционированному субъекту. Алгоритм шифрования использует ключ, который представляет собой строку чисел или букв. В примере, представленном на рис. 13.6, алгоритм шифрования добавляет 1 к коду ASCII каждого символа исходной строки и таким образом заменяет все символы.

Ключ, используемый алгоритмом шифрования, в данном случае равен целому числу 1. Процесс расшифровки представляет собой процесс, обратный процессу шифрования, при котором каждый символ изменяется с помощью вычитания 1, тем самым восстанавливая открытый текст.

Все алгоритмы шифрования данных основаны на принципе замены одного набора данных другим набором. На рис. 13.6 используется простейший алгоритм шифрования с заменой одного бита. В реальных приложениях алгоритмы шифрования намного сложнее, но принцип тот же.

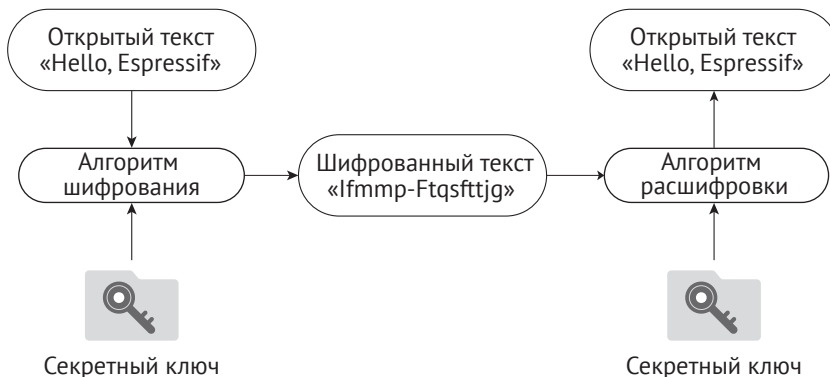


Рисунок 13.6. Основной принцип шифрования данных

Алгоритмы шифрования данных обычно можно разделить на две категории: алгоритмы симметричного шифрования и алгоритмы асимметричного шифрования.

Алгоритмы симметричного шифрования

Как следует из названия, алгоритмы симметричного шифрования используют один и тот же ключ для шифрования и расшифровки. Обычно используемые алгоритмы симметричного шифрования – это DES, 3DES и AES. Процесс шифрования, аналогичный показанному на рис. 13.6, является основной разновидностью симметричного шифрования. Ключ, используемый для шифрования и дешифрования, один и тот же, в данном случае целое число 1.

Алгоритмы асимметричного шифрования

Алгоритмы асимметричного шифрования используют два разных ключа: открытый ключ и закрытый ключ, которые представляют собой пару строк

с определенной зависимостью. Контент, зашифрованный открытым ключом, может быть расшифрован только с помощью парного закрытого ключа. Аналогично контент, зашифрованный закрытым ключом, может быть расшифрован только с помощью парного открытого ключа.

Предпосылкой для симметричного шифрования является то, что шифратор и дешифратор должны договориться об общем ключе, то есть они должны заранее знать содержимое ключа. Однако в некоторых случаях шифратор и дешифратор никогда не встречались и не обменивались данными каким-то способом, кроме как через сеть. В таких случаях без заранее согласованных ключей как шифровальщик и дешифровальщик смогут выполнить шифрование или дешифрование? Ответ – в асимметричном шифровании.

На рис. 13.7 показан основной процесс использования асимметричного и симметричного шифрований вместе для передачи зашифрованных данных, где асимметричное шифрование используется для обмена симметричным ключом. После его получения клиент и сервер используют для защиты конфиденциальности передаваемых данных менее ресурсоемкий алгоритм симметричного шифрования.



Рисунок 13.7. Сочетание асимметричного и симметричного шифрований для передачи данных

Обычно используемый алгоритм асимметричного шифрования – алгоритм RSA.

В этой книге не будут представлены технические подробности об алгоритмах шифрования. После получения базового представления о шифровании данных мы можем отправиться дальше.

13.3.2. Введение в систему флеш-шифрования

Система шифрования флеш-памяти (для краткости далее именуемая просто флеш-шифрованием) используется для усиления защиты конфиденциальности данных, чтобы гарантировать их безопасность. После включения этой функции для восстановления содержимого флеш-памяти простого чтения бу-

дет недостаточно. Как пояснялось ранее, конфиденциальность данных должна быть обеспечена как на этапе передачи, так и на этапе хранения. Флеш-шифрование может использоваться для шифрования данных, хранящихся во флеш-памяти, а для передачи данных необходима другая система шифрования, например протокол передачи HTTPS.

1. Соответствующие области хранения

И eFuse, и flash являются носителями данных, к которым применяется флеш-шифрование, но имеют различные свойства и способы использования, показанные в табл. 13.1.

Типы данных, хранящихся во флеш-памяти и зашифрованных с помощью флеш-шифрования, включают загрузчик прошивки, прошивку приложения, таблицу разделов и любой раздел, помеченный в таблице разделов флагом encrypted.

Как видно из следующей таблицы разделов, приведенной в качестве иллюстрации, включение флеш-шифрования приведет к шифрованию некоторых конкретных разделов, а именно: раздела загрузчика (bootloader partition), раздела заводской прошивки (factory partition), раздела хранения (storage partition) и раздела ключа nvs_key. Примечательно, что разделы, используемые для хранения прошивки, такие как раздел загрузчика и раздел заводской прошивки, зашифрованы по умолчанию, поэтому нет необходимости добавлять к ним флаг encrypted.

Таблица 13.1. Содержимое и свойства eFuse и флеш-памяти

Хранилище	Содержимое	Свойства
Flash	<i>Bootloader.bin</i> , <i>app.bin</i> , данные <i>nvs</i> и таблицы разделов	Флеш-память можно стирать и перепрограммировать неоднократно
eFuse	Системные параметры, такие как версия чипа и MAC-адрес, ключи и биты управления, относящиеся к системным функциям	Если бит eFuse запрограммирован на 1, его уже нельзя вернуть в 0. В частности, для некоторых блоков eFuse, если для них установлена защита от чтения, данные в этих блоках могут быть прочитаны только аппаратными криптографическими модулями

1. # Name, Type, SubType, Offset, Size, Flags
2. nvs, data, nvs, , 0x6000,
3. # Extra partition to demonstrate reading/writing of encrypted flash
4. storage, data, 0xff, , 0x1000, encrypted
5. factory, app, factory, , 1M,
6. # nvs_key partition contains the key that encrypts the NVS partition named nvs.
7. The nvs_key partition needs to be encrypted.
8. nvs_key, data, nvs_keys, , 0x1000, encrypted

Флеш-шифрование используется для шифрования данных, хранящихся во флеш-памяти. Некоторые eFuses применяются во время шифрования прошивки. Перечень используемых eFuses и их описание приведены в табл. 13.2.

Таблица 13.2. eFuses, используемые при флеш-шифровании

eFuses	Описание	Длина (бит)
BLOCK_KEYN	Используется для хранения флеш-ключа шифрования/дешифрования. N находится в диапазоне от 0 до 4	256
DIS_DOWNLOAD_MANUAL_ENCRYPT	Если установлен, отключает функцию загрузки с флеш-шифрованием в режиме скачивания загрузки	1
SPI_BOOT_CRYPT_CNT	Включает шифрование и дешифрование. Функция активна, если в поле установлен один или три бита eFuse, в противном случае отключена	3

Инструмент *espefuse.py* можно использовать на ESP32-C3 для проверки текущего статуса eFuse. Для примера выполните следующую команду, проверяющую текущее значение eFuse:

```
$ espefuse.py --port PORT summary //replace "PORT" with your port name
```

Если FLASH_CRYPT_CNT равен 0, как показано в выводе лога ниже, это означает, что флеш-шифрование выключено:

```
espefuse.py v2.6-beta1
Connecting....._____.
EFUSE_NAME Description = [Meaningful Value] [Readable/Writeable] (Hex Value)
-----
Security fuses:
FLASH_CRYPT_CNT      Flash encryption mode counter           = 0 R/W (0x0)
FLASH_CRYPT_CONFIG   Flash encryption config (key tweak bits) = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE Disable ROM BASIC interpreter fallback    = 1 R/W (0x1)
Identity fuses:
MAC                  MAC Address
= 30:ae:a4:c3:86:94 (CRC 99 OK) R/W
...

```

2. Алгоритм флеш-шифрования

Алгоритм симметричного шифрования, используемый при флеш-шифровании, – AES-XTS, который является настраиваемым блочным шифром. Во время шифрования алгоритм шифрует данные открытого текста блоками и динамически настраивает ключ в соответствии с адресом смещения данных открытого текста. Базовый принцип блочного шифрования AES-XTS-128 показан на рис. 13.8, где 64 байта данных открытого текста делятся на четыре блока и из базового ключа получают ключи шифрования (ключ1 ÷ ключ4). Объединение четырех зашифрованных блоков даст зашифрованные данные размером 64 байта.

Преимущества AES-XTS, который сначала динамически настраивает ключ шифрования, а затем шифрует данные:

- шифрование одного и того же блока данных приводит к получению отличающегося зашифрованного текста, что делает зашифрованные данные более сложными для анализа и взлома, повышая их защищенность;

- различные блоки данных могут быть зашифрованы и расшифрованы независимо. Даже если один блок данных поврежден, это не повлияет на расшифровку других блоков.

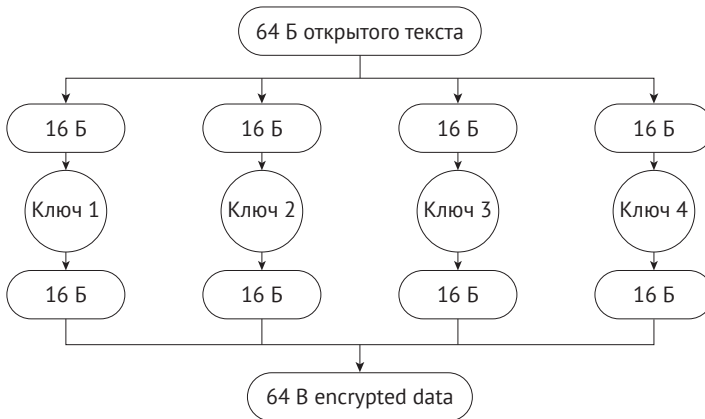


Рисунок 13.8. Основной принцип блочного шифрования AES-XTS-128

13.3.3. Хранение ключей флеш-шифрования

Ключ флеш-шифрования хранится в секторе BLOCK_KEY. Есть два способа записать ключ в eFuse:

Ручной метод

Использует *espsecure.py*, чтобы вручную сгенерировать ключ и записать его в eFuse. Этот метод можно использовать только перед первым включением флеш-шифрования.

Автоматический метод

После включения флеш-шифрования в меню конфигурации *menuconfig* устройство автоматически сгенерирует ключ в загрузчике при первом запуске и автоматически сохранит его в eFuse.

Чтобы вручную записать ключ флеш-шифрования в eFuse, сначала выполните следующую команду для генерации ключа:

```
$ espsecure.py generate flash_encryption_key my_flash_encryption_key.bin
```

Затем выполните следующую команду, чтобы записать ключ в eFuse:

```
$ espfuse.py --port PORT burn key BLOCK my_flash_encryption_key.bin XTS AES 128 KEY
```

Примечание

Поскольку запись eFuse необратима, запись ключа в eFuse вручную можно провести только один раз.

Флеш-шифрование можно включить через меню конфигурации `menuconfig` в пункте **Security features** (Функции безопасности) → **Enable flash encryption on boot** (Включить флеш-шифрование при загрузке). Если флеш-шифрование включено на этапе сборки и ключ заранее не был прописан вручную в eFuse, то после окончания прошивки устройство включит флеш-шифрование, автоматически сгенерирует ключ и запишет его в eFuse.

Основное отличие ручного метода от автоматического состоит в том, что ручным методом вы можете узнать содержимое ключа и использовать скрипт для шифрования данных перед их прошивкой в устройство. При автоматическом методе, если в eFuse включена защита от чтения сектора `BLOCK_KEY` (включенная по умолчанию), ключ генерируется внутри устройства и хранится непосредственно в защищенном от чтения eFuse, что делает невозможным внешним разработчикам получить ключ или вручную зашифровать/расшифровать данные.

В ручном режиме используйте следующую команду, чтобы зашифровать прошивку приложения и прошить ее на устройство:

```
$ espsecure.py encrypt flash data --aes xts --keyfile /path/to/key.bin --address 0x10000 --output my-app-ciphertext.bin build/my-app
```

Важно отметить, что при использовании этой команды для шифрования данных необходимо указать адрес хранения данных в таблице разделов. В приведенном примере шифруются данные `my-app` с адресом `0x10000`. Как подчеркивалось в разделе 13.3.2, флеш-шифрование основано на настраиваемом блочном шифре AES-XTS и, следовательно, должен быть указан точный адрес данных. Отсутствие или неправильное указание адреса приведет к сбою устройства после загрузки зашифрованной прошивки.

Кроме того, стоит отметить, что если ключ шифрования известен, скрипт `espsecure.py` также можно использовать для расшифровки данных. Запуск команды `espsecure.py -h` предоставит полезную информацию об использовании скрипта.

При серийном производстве устройств настоятельно рекомендуется использовать автоматический метод написания ключей. Это гарантирует, что каждому устройству будет присвоен уникальный ключ, который останется недоступен из внешних источников, что максимально повышает общую безопасность устройства.

13.3.4. Рабочие режимы флеш-шифрования

Флеш-шифрование имеет два режима работы:

Режим разработки (Development mode)

Как следует из названия, режим разработки используется на этапе разработки, когда часто возникает необходимость запрограммировать различные образы флеш-памяти в виде открытого текста и протестировать процесс шифрования флеш-памяти. Для этого необходимо, чтобы новые образы с открытым текстом могли загружаться столько раз, сколько необходимо. В режиме разработки можно отключить флеш-шифрование и установить новую прошивку с помощью команд, которые будут показаны в разделе 13.3.7.

Режим реализации (Release mode)

Режим реализации рекомендуется для серийного производства. В этом режиме последовательный порт не может выполнять операции флеш-

шифрования, и их можно обеспечить, включив свойство флеш-шифрования. В этом режиме однажды включенное флеш-шифрование нельзя отключить и новую прошивку нельзя загрузить через последовательный порт, ее можно загрузить только с помощью OTA-способа.

Рабочий режим флеш-шифрования можно выбрать через меню конфигурации `menuconfig` в пункте **Security features** (Функции безопасности) → **Enable flash encryption on boot** (Включить флеш-шифрование при загрузке) → **Enable usage mode** (Включить режим использования). Рисунок 13.9 демонстрирует, что для флеш-шифрования включен режим разработки.

```
[ ] Require signed app images
[ ] Enable hardware Secure Boot in bootloader (READ DOCS FIRST)
[*] Enable flash encryption on boot (READ DOCS FIRST)
    Enable usage mode (Development (NOT SECURE)) -->
    Potentially insecure options --->
[*] Check Flash Encryption enabled on app startup (NEW)
    UART ROM download mode (UART ROM download mode (Enabled (not recommended))) --->
```

Рисунок 13.9. Режим разработки включен для флеш-шифрования

Обратите внимание, что в режиме разработки флеш-шифрование можно отключить с помощью команды `espefuse.py --port PORT burn_efuse SPI_BOOT_CRYPT_CNT`. После отключения системы шифрования отмените выбор этой опции в меню конфигурации, затем запустите `idf.py flash`, чтобы загрузить новую прошивку. Количество раз отключения флеш-шифрования ограничено длиной флага `SPI_BOOT_CRYPT_CNT` в eFuse. Если флаг содержит нечетное число «1», это означает, что флеш-шифрование включено; если он содержит четное количество «1», это означает, что флеш-шифрование отключено. Если длина бита флага составляет 3 бита, это означает, что флеш-шифрование можно отключить только один раз.

Примечание

Посетите <https://docs.espressif.com/projects/esptool/en/latest/esp32/espsecure/index.html> для получения дополнительной информации об `espefuse.py`.

Включение флеш-шифрования может увеличить размер загрузчика. Вариантами решения этой проблемы являются следующие.

- (1) Установите смещение таблицы разделов через меню конфигурации `menuconfig` → **Partition Table** (Таблица разделов) → **Offset of partition table** (Смещение таблицы разделов). Например, изменение смещения с `0x8000` на `0xa000` увеличит пространство на 8 КБ.

Обратите внимание, что после изменения смещения раздела загрузчика необходимо проверить, следует ли обновить распределение областей в таблице разделов.

- (2) Понижьте уровень лога загрузчика с помощью меню конфигурации `menuconfig` → **Bootloader log verbosity** (Подробности лога загрузчика).

Изменение уровня лога с **Info** (Информирование) на **Warning** (Предупреждения) может уменьшить размер журнала, тем самым уменьшая размер загрузчика.

13.3.5. Процесс флеш-шифрования

После первой загрузки прошивки в открытом виде на устройство с включенным флеш-шифрованием и последующего запуска устройства будет автоматически активирована функция флеш-шифрования. Ниже описан основной рабочий процесс первоначального автоматического включения флеш-шифрования:

- (1) загрузчик прошивки считывает из eFuse значение SPI_BOOT_CRYPT_CNT. Если флеш-шифрование не включено, загрузчик включит шифрование флеш-памяти автоматически. По умолчанию значение равно 0, что означает, что флеш-шифрование еще не включено;
- (2) загрузчик проверяет, хранит ли BLOCK_KEY ключ шифрования флеш-памяти. Если ключ предварительно не прошит (см. раздел 13.3.3), он будет сгенерирован автоматически и записан в BLOCK_KEY. Биты защиты от записи и чтения для BLOCK_KEY будут установлены, так что программное обеспечение не может получить доступ к ключу;
- (3) блок шифрования флеш-памяти шифрует содержимое флеш-памяти – загрузчик прошивки, приложения и разделы, помеченные как encrypted;
- (4) загрузчик прошивки устанавливает первый доступный бит в SPI_BOOT_CRYPT_CNT в 1, чтобы пометить содержимое флеш-памяти как зашифрованное;
- (5) в режиме разработки SPI_BOOT_CRYPT_CNT и DIS_DOWNLOAD_MANUAL_ENCRYPT не защищены от записи. Загрузчик прошивки позволяет отключить шифрование флеш-памяти и перепрошить зашифрованные двоичные файлы;
- (6) в режиме реализации SPI_BOOT_CRYPT_CNT и DIS_DOWNLOAD_MANUAL_ENCRYPT защищены от записи. Флеш-шифрование включено навсегда, и перепрошивка запрещена;
- (7) устройство перезагружается, чтобы начать выполнение зашифрованного загрузчика и прошивки приложения.

Примечание

По умолчанию, когда флеш-шифрование включено, будут установлены некоторые биты флагов eFuse, таким образом отключены некоторые системные функции, например JTAG. Сохранение этих системных функций может привести к рискам безопасности. Если на этапе тестирования вам необходимо сохранить эти флаги, в руководстве по программированию ESP-IDF обратитесь к инструкциям, относящимся к флеш-шифрованию.

При включенном флеш-шифровании, когда устройство загружает и запускает зашифрованный загрузчик и прошивку приложения, оно сначала автоматически расшифровывает данные через аппаратный модуль и затем загружает расшифрованные данные в свой iRAM и кеш. Кроме того, некоторые API разработаны для беспрепятственного шифрования и дешифрования данных при выполнении чтения и операции записи в зашифрованных разделах флеш-памяти. API, отвечающие за автоматическую расшифровку данных, включают esp_partition_read(), esp_flash_read_encrypted() и bootloader_flash_read(); API, отвечающие за автоматическое шифрование данных, включают esp_partition_write(), esp_flash_write_encrypted() и bootloader_flash_write().

В частности, при включенном флеш-шифровании во время OTA-обновлений устройство получает данные в виде открытого текста, а затем вызывает `esp_partition_write()` для автоматического шифрования данных перед записью во флеш-память.

Примечание

Для устройств серийного производства можно использовать функцию обновления OTA для удаленного обновления прошивки приложения, но не загрузчика. Поэтому крайне важно тщательно настроить загрузчик, включая такие параметры, как уровень логов, перед включением флеш-шифрования.

13.3.6. Введение в шифрование NVS

Система флеш-шифрования не защищает напрямую данные, хранящиеся в разделе NVS. Для защиты конфиденциальности данных, хранящихся в разделе NVS, должна быть использована система шифрования NVS. Шифрование NVS можно включить через меню конфигурации `menuconfig` → **Component config** (Конфигурация компонента) → **NVS** → **Enable NVS encryption** (Включить шифрование NVS) или вызвав функцию `nvs_flash_secure_init()`.

Основной принцип системы шифрования NVS заключается в формировании раздела в таблице разделов размером не менее 4 КБ и с подтипом `nvs_key`. После включения системы шифрования NVS раздел NVS будет зашифрован с использованием ключа из раздела `nvs_key`. Типичная таблица разделов, поддерживающая шифрование NVS, показана ниже:

1. #	Name,	Type,	SubType,	Offset,	Size,	Flags
2.	<code>nvs</code> ,	<code>data</code> ,	<code>nvs</code> ,	,	<code>0x6000</code> ,	
3.	<code>phy_init</code> ,	<code>data</code> ,	<code>phy</code> ,	,	<code>0x1000</code> ,	
4.	<code>factory</code> ,	<code>app</code> ,	<code>factory</code> ,	,	<code>1M</code> ,	
5.	<code>nvs_key</code> ,	<code>data</code> ,	<code>nvs_keys</code> ,	,	<code>0x1000</code> ,	<code>encrypted</code> ,

Система шифрования NVS во многом похожа на систему флеш-шифрования, например:

- алгоритм шифрования, используемый системой шифрования NVS, представляет собой симметричное шифрование по алгоритму AES-XTS. Как упоминалось ранее, алгоритм симметричного шифрования требует наличия ключа, хранящегося в секрете, чтобы гарантировать невозможность анализа и взлома зашифрованных данных. Ключ системы шифрования NVS не может быть открытым текстом, поэтому эта система часто использует вместе с системой флеш-шифрования, где флеш-шифрование отвечает за защиту конфиденциальности ключа `nvs_key`, в то время как шифрование NVS использует `nvs_key` для защиты конфиденциальности раздела NVS;
- существует два метода хранения ключа шифрования NVS: один – ручной, предлагающий вручную сгенерировать ключ и записать его в назначенный раздел; другой – автоматический метод, то есть при первом включении системы шифрования NVS раздел `nvs_key` пуст, устройство автоматически вызывает функцию `nvs_flash_generate_keys()` для генерации ключа, затем записывает его в раздел `nvs_key` и впоследствии использует ключ для выполнения шифрования/дешифрования NVS.

Шаги по сохранению ключа шифрования NVS вручную следующие.
Сначала создайте файл, содержащий ключ, выполнив следующую команду:

```
$ espsecure.py generate flash encryption key my nvs encryption key.bin
```

Затем скомпилируйте и запишите таблицу разделов с помощью следующей команды:

```
$ idf.py -p (PORT) partition table-flash
```

Наконец, запишите ключ указанного раздела с помощью следующей команды:

```
$ parttool.py -p (PORT) --partition-table-offset "nvs key partition offset"
write partition --partition-name="name of nvs key partition" --input "nvs key
partition"
```

- После включения шифрования NVS API, начинающиеся с `nvs_get` или `nvs_set`, автоматически выполняют шифрование/дешифрование данных при чтении/записи данных в разделе NVS.

Обратите внимание: если включено флеш-шифрование, настоятельно рекомендуется также включить шифрование NVS (включено по умолчанию). Это важно, поскольку драйвер Wi-Fi сохраняет конфиденциальные данные, такие как SSID и пароль, по умолчанию в разделе NVS. Система шифрования позволяет использовать разные ключи (`nvs_key`) в разных разделах NVS. При инициализации конкретного раздела NVS вам нужно лишь указать соответствующий `nvs_key`.

Примечание

Посетите <https://book3.espressif.com/nvs> для получения дополнительной информации о шифровании NVS.

13.3.7. Примеры флеш-шифрования и шифрования NVS

В каталог [esp-idf/examples/security/flash_encryption](#) мы загрузили пример флеш-шифрования и NVS-шифрования. Запустив этот пример, вы можете просмотреть логи, демонстрирующие результаты флеш-шифрования и NVS-шифрования.

Как упоминалось ранее, когда в режиме разработки включено флеш-шифрование, прошивку можно загружать неоднократно. Для загрузки прошивки в устройство мы использовали следующие три команды.

Команда 1:

```
$ idf.py -p PORT flash monitor
```

Команда 2:

```
$ idf.py -p PORT encrypted-flash monitor
```

Команда 3:

```
$ idf.py -p PORT encrypted-app-flash monitor
```

Результаты выполнения трех вышеуказанных команд:

- команда 1: данные, хранящиеся во флеш-памяти, остаются в виде открытого текста, что приводит к ошибке загрузки;
- команда 2: прошиваются только зашифрованный загрузчик, прошивка приложения и таблица разделов, и прошивку можно без проблем загрузить и запустить на устройстве;
- команда 3: прошивается только зашифрованная прошивка приложения; если загрузчик также зашифрован, прошивку можно загрузить и запустить на устройстве.

Вышеупомянутые три команды фактически делают внутренний вызов `esptool.py`. Соответствующие настройки `esptool.py`:

```
$ esptool.py --chip esp32c3 -p /dev/ttyUSB0 -b 460800 --before=default reset
--after= no reset write flash --flash mode dio --flash freq 40m --flash size
2MB 0x1000 bootloader/bootloader 0x20000 flash encryption.bin 0xa000 partition
table/partition-table.bin
```

```
$ esptool.py --chip esp32c3 -p /dev/ttyUSB0 -b 460800 --before=default reset
--after= no reset write flash --flash mode dio --flash freq 40m --flash size
2MB --encrypt 0x1000 bootloader/bootloader 0x20000 flash encryption.bin 0xa000
partition table/partition-table.bin
```

```
$ esptool.py --chip esp32c3 -p /dev/ttyUSB0 -b 460800 --before=default reset
--after= no reset write flash --flash mode dio --flash freq 40m --flash size 2MB
--encrypt 0x20000 flash encryption.bin
```

Изучив настройки команд, можно сделать вывод, что при использовании `esptool.py` для прошивки добавление опции `--encrypt` включит автоматическое шифрование прошивки и запись зашифрованных данных во флеш-память.

Ниже приведено несколько типичных случаев сбоя при включении флеш-шифрования.

- Если **загрузчик** является открытым текстом, при запуске устройства может возникнуть следующая ошибка:

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
invalid header: 0xb414f76b
```

- Если **раздел** является открытым текстом, при запуске устройства может возникнуть следующая ошибка:

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsp: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:10464
```

```
ho 0 tail 12 room 4
load:0x40078000,len:19168
load:0x40080400,len:6664
entry 0x40080764
I (60) boot: ESP-IDF v4.0-dev-763-g2c55fae6c-dirty 2nd stage bootloader
```

- Если **прошивка** представляет собой открытый текст, при запуске устройства может возникнуть следующая ошибка:

```
rst:0x3 (SW_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8452
load:0x40078000,len:13616
load:0x40080400,len:6664
entry 0x40080764
I (56) boot: ESP-IDF v4.0-dev-850-gc4447462d-dirty 2nd stage bootloader
I (56) boot: compile time 15:37:14
I (58) boot: Enabling RNG early entropy source...
I (64) boot: SPI Speed : 40MHz
I (68) boot: SPI Mode : DIO
I (72) boot: SPI Flash Size : 4MB
I (76) boot: Partition Table:
I (79) boot: ## Label Usage Type ST Offset Length
I (87) boot: 0 nvs Wi-Fi data 01 02 0000a000 00006000
I (94) boot: 1 phy_init RF data 01 01 00010000 00001000
I (102) boot: 2 factory factory app 00 00 00020000 00100000
I (109) boot: End of partition table
E (113) esp_image: image at 0x20000 has invalid magic byte
W (120) esp_image: image at 0x20000 has invalid SPI mode 108
W (126) esp_image: image at 0x20000 has invalid SPI size 11
E (132) boot: Factory app partition is not bootable
E (138) boot: No bootable app partitions in the partition table
```

13.4. Защита легитимности данных

13.4.1. Введение в цифровую подпись

Возможно, вы подписали свое имя в заявлении о приеме, юридических документах или на квитанции по кредитной карте, свидетельствуя о том, что вы согласны с содержанием данных документов. В области безопасности данных устройству необходимо идентифицировать отправителя или производителя данных для уверенности, что данные не подделаны, были авторизованы, легитимны и могут безопасно использоваться. Цифровая подпись – техническое решение для проверки легитимности данных.

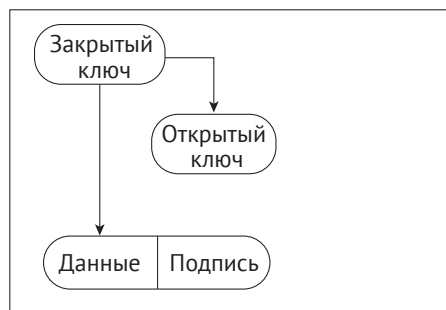
Цифровая подпись имеет два свойства: **невозможность подделки**, то есть только законные отправители данных могут подписывать данные, а другие подписи недействительны, и **проверяемость**, то есть пользователи данных должны иметь возможность для проверки действительности подписи.

Общие алгоритмы цифровой подписи включают **RSA** и **DSA**. Основной процесс проверки цифровой подписи содержит:

- (1) отправитель данных генерирует закрытый ключ, который используется для генерации открытого ключа, таким образом получается пара закрытый–открытый ключ. Обратите внимание, что только закрытый ключ может сгенерировать соответствующий открытый ключ;
- (2) отправитель данных сохраняет открытый ключ в системе хранения данных пользователя;
- (3) отправитель данных подписывает данные закрытым ключом и отправляет подписанные данные и подпись пользователю данных;
- (4) после получения данных пользователь данных использует открытый ключ, сохраненный на этапе (2), для проверки подписи, отправленной на этапе (3). Если подпись верна, данные считаются верными, полученными от законного отправителя данных; в противном случае данные считаются неавторизованными и не будут использоваться.

Основной принцип использования цифровой подписи для проверки легитимности данных показан на рис. 13.10.

Отправитель данных



Получатель данных

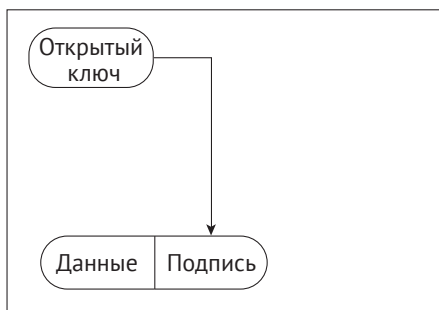


Рисунок 13.10. Основной принцип использования цифровых подписей для проверки легитимности данных

Благодаря этому механизму «выдачи секретного ключа, открытого ключа, подписания закрытым ключом и проверки подписи открытым ключом» легитимность источника данных может быть подтверждена. Однако, возможно, вы заметили, что есть необходимые условия, для того чтобы эта схема была эффективной:

- закрытый ключ отправителя данных не должен быть разглашен. Как только закрытый ключ оказывается доступным, злоумышленник может использовать опубликованный закрытый ключ для подписи незаконных данных и отправки их пользователю данных, и этот механизм не сработает;
- открытый ключ пользователя данных не может быть удален по желанию. Если злоумышленник создает пару закрытый–открытый ключ в его собственной системе и заменяет открытый ключ пользователя данных своим открытым ключом, злоумышленник может подписать незаконные данные своим собственным закрытым ключом и затем отправить их пользователю данных. Пользователь данных может использовать замененный общедоступный ключ для проверки данных, отправленных злоумышленником, и, таким образом, посчитать данные легитимными.

В следующих разделах мы увидим, как система безопасной загрузки решает эти проблемы. Продолжим наше путешествие к следующему разделу!

13.4.2. Обзор системы безопасной загрузки

Система безопасной загрузки используется для защиты легитимности данных встроенного ПО (включая загрузчик и прошивку приложения). Она использует алгоритм цифровой подписи RSA для проверки подписи, прикрепленной к прошивке, перед загрузкой и запуском новых прошивок, тем самым осуществляя проверку легитимности данных прошивки. Если включена система безопасной загрузки, устройство загружает и запускает только прошивку, авторизованную указанным закрытым ключом.

Прежде чем углубляться в принципы реализации безопасной загрузки, давайте кратко рассмотрим процесс загрузки ESP32-C3, показанный на рис. 13.11.

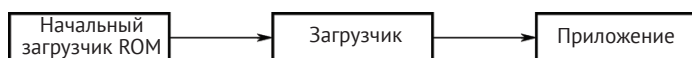


Рисунок 13.11. Процесс загрузки ESP32-C3

Когда устройство включено, процесс загрузки начинается с начального загрузчика (ROM Boot), а затем передается основному загрузчику (Bootloader) и в конечном итоге доходит до прошивки приложения. Начальный загрузчик ROM – фиксированная исполняемая программа во встроенной постоянной памяти (ROM), которая остается неизменной. Следовательно, основной загрузчик и прошивка приложения являются ключевыми компонентами, требующими защиты. Модификацию прошивки можно выполнить двумя способами: **физической перепрошивкой**, при которой новый загрузчик и прошивка с кодом приложения записываются во флеш-память устройства с помощью программы для перепрошивки; или с помощью **ОТА-обновлений**, обновляющих только прошивку приложения, исключая загрузчик.

Итак, возникает вопрос: как мы можем обеспечить целостность и легитимность данных прошивки независимо от способа их передачи на устройство? Чтобы решить эту проблему, мы рассмотрим два режима безопасной загрузки в разделах 13.4.3 и 13.4.4: **безопасная программная** и **безопасная аппаратная** загрузка.

Примечание

Существует две версии системы безопасной загрузки: v1 и v2. Поскольку ESP32-C3 поддерживает только безопасную загрузку v2, содержимое этого раздела применимо к варианту v2.

13.4.3. Введение в программную безопасную загрузку

Программная безопасная загрузка для верификации не требует аппаратной поддержки (в основном eFuse).

Прежде чем включить программную безопасную загрузку, необходимо сгенерировать закрытый ключ подписи RSA, используя следующую команду:

```
$ espsecure.py generate signing key --version 2 secure_boot_signing_key.pem
```

Сгенерированный закрытый ключ хранится в файле *secure_boot_signing_key.pem*.

Включить безопасную программную загрузку так же просто: следует выбрать **Require signed app images** (Требовать подписанные образы приложений) в *menuconfig* (как показано на рис. 13.12), после чего идут сборка и загрузка прошивки.

```
[*] Require signed app images
    App Signing Scheme (RSA) --->
[*] Verify app signature on update
[ ] Enable hardware Secure Boot in bootloader (READ DOCS FIRST)
[*] Sign binaries during build
    (secure_boot_signing_key.pem) Secure boot private signing key
[ ] Enable flash encryption on boot (READ DOCS FIRST)
```

Рисунок 13.12. Включение программной безопасной загрузки для ESP32-C3

Если включена программная безопасная загрузка, во время сборки прошивки созданная прошивка приложения (далее называемая исходным приложением) содержит открытый ключ, который будет использоваться для проверки легитимности новой прошивки *new_app*, отправленной через OTA-обновление. Как показано на рис. 13.13, во время OTA-обновлений, после получения прошивки и вызова *esp_ota_end()* или *esp_ota_set_boot_partition()*, программная загрузка автоматически будет использовать открытый ключ в *origin_app* для проверки цифровой подписи, прикрепленной к *new_app*.

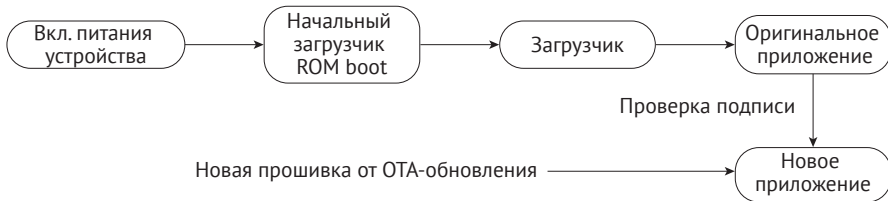


Рисунок 13.13. Программная безопасная загрузка проверяет новую прошивку приложения, отправленную через OTA-обновления

Когда включена безопасная программная загрузка, прошивка приложения, отправляемая на устройство через OTA-обновление, должна быть подписана закрытым ключом. Есть два способа добиться этого:

- (1) как показано на рис. 13.12, настройте параметр **Sign binaries during build** (Подписывать двоичные файлы во время сборки) и укажите каталог файла закрытого ключа, после чего прошивка приложения при компиляции будет подписана автоматически;
- (2) выполните следующую команду, чтобы подписать прошивку приложения:

```
$ espsecure.py sign data --version 2 --keyfile PRIVATE_SIGNING_KEY BINARY_FILE
```

Приведенная команда напрямую изменяет текущий файл и добавляет проверочную информацию непосредственно в него. Используйте опцию *--output*, чтобы поименовать файл после добавления подписи. Применение команды

подписи прошивки позволяет хранить подписанный закрытый ключ на удаленном сервере, а не на прошиваемом устройстве, поэтому удобнее для пакетной подписи при серийном выпуске устройств.

Включение безопасной программной загрузки предполагает добавление блока подписи в прошивку приложения. Этот блок подписи содержит необходимые данные для проверки подписи. В случае ESP32-C3 при использовании программной безопасной загрузки сохраняется только первичный блок подписи с определенным периодом действия. И наоборот, при выборе аппаратной безопасной загрузки разрешено до трех блоков подписи, каждый из которых может содержать отдельный закрытый ключ. Проверка считается успешной, если хотя бы одна из подписей действительна. Формат данных подписанной прошивки приложения ESP32-C3 показан на рис. 13.14.

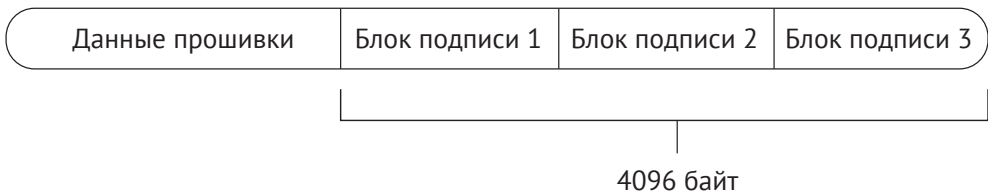


Рисунок 13.14. Формат данных подписанной прошивки приложения ESP32-C3

В системе программной безопасной загрузки компилируется открытый ключ, используемый для проверки подписи в текущей прошивке приложения, и автоматически управляется устройством. Пользователям не требуется управлять им вручную. Чтобы получить содержимое открытого ключа, используйте следующую команду, вручную экспортирующую открытый ключ, полученный из закрытого ключа:

```
$ espsecure.py extract_public_key --version 2 --keyfile secure_boot_signing_key.pem pub_key.pem
```

В этой команде закрытый ключ содержится в `secure_boot_signing_key.pem`, а `pub_key.pem` – открытый ключ, полученный из этого закрытого ключа.

Из принципов реализации программной безопасной загрузки можно сделать вывод, что система проверяет `new_app`, отправленное через OTA-обновление, с использованием файла `origin_app`. Однако злоумышленники могут прошить неавторизованный загрузчик и `origin_app` на устройстве посредством физической перепрошивки, которой нельзя управлять с помощью программной безопасной загрузки. В результате программная безопасная загрузка больше подходит для применений, в которых устройство неуязвимо к физическим атакам. В последующих разделах мы углубимся в то, как аппаратная система безопасной загрузки защищает от физических атак.

13.4.4. Введение в аппаратную безопасную загрузку

Аппаратная безопасная загрузка включает проверку, выполняемую с помощью аппаратного обеспечения. Она использует данные, хранящиеся в eFuse, для проверки легитимности данных прошивки. Соответствующие биты eFuse показаны в табл. 13.3.

Таблица 13.3. Биты eFuses для проверки легитимности данных прошивки

eFuses	Описание	Длина (бит)
SECURE_BOOT_EN	Если установлен, аппаратная безопасная загрузка включена постоянно	1
KEY_PURPOSE_X	X – натуральное число. Например, KEY_PURPOSE_1 используется для установки назначения BLOCK_KEY1	4
BLOCK_KEYX	Если соответствующий BLOCK_KEYX установлен на SECURE_BOOT_DIGEST1, BLOCK_KEYX будет содержать SHA256-дайджест ⁴⁶ открытого ключа	256

Аппаратная безопасная загрузка поддерживает не только все описанные в разделе 13.4.3 функции программной безопасной загрузки, но также дополнительные проверки загрузчика и `origin_app` прошивки. Схема аппаратной безопасной загрузки использует тот же метод создания закрытых-открытых пар ключей и тот же метод подписи прошивки приложения, представленный в разделе 13.4.3.

Если включена аппаратная безопасная загрузка, помимо прошивки приложения, загрузчик также требует подписи, используя тот же метод и формат, что и прошивка. В случае если загрузчик нужно пересобрать и подписать заново, необходимо выполнить команду `idf.py bootloader` отдельно. Кроме того, для прошивки подписанного загрузчика необходима команда `idf.py -p PORT bootloader-flash`. Команда `idf.py flash` запустит только подписанную прошивку приложения и таблицу разделов, исключая загрузчик.

Аппаратную безопасную загрузку можно включить следующим образом:

- (1) откройте меню конфигурации проекта, перейдите в меню конфигурации `menuconfig` в раздел **Security features** (Функции безопасности) и выберите параметр **Enable hardware Secure Boot** (Включить аппаратную безопасную загрузку);
- (2) если при компиляции необходимо подписать прошивку, укажите закрытый ключ подписи. Как показано на рис. 13.15, укажите файл закрытого ключа через меню конфигурации `menuconfig` → **Security features** (Функции безопасности) → **Secure Boot private key** (Закрытый ключ безопасной загрузки). Если закрытый ключ не был сгенерирован, обратитесь к разделу 13.4.3, чтобы экспортировать закрытый ключ. Кроме того, обратитесь к разделу 13.4.3, чтобы подписать прошивку с помощью `espsecure.py`;
- (3) запустите команду `idf.py bootloader`, чтобы создать загрузчик, а затем `idf.py -p PORT bootloader-flash` для прошивки загрузчика;
- (4) запустите `idf.py flash monitor`, чтобы прошить прошивку приложения и таблицу разделов;

⁴⁶ Дайджест (digest) – в терминах криптографии сокращенный «отпечаток» какой-либо последовательности данных, аналог понятия контрольной суммы, позволяющий проверить аутентичность данных.

- (5) после включения устройства оно запустит только что созданный загрузчик, который автоматически устанавливает флаг `SECURE_BOOT_EN` в eFuse, разрешая постоянное использование аппаратной безопасной загрузки. Кроме того, дайджест открытого ключа, прикрепленный к блоку подписи загрузчика, будет записан в `BLOCK_KEY`. На рис. 13.15 показано, как включить аппаратную безопасную загрузку на этапе компиляции.

```
App Signing Scheme (RSA) --->
[*] Enable hardware Secure Boot in bootloader (READ DOCS FIRST)
    Select secure boot version (Enable Secure Boot version 2) --->
[*] Sign binaries during build
    (secure_boot_signing_key.pem) Secure boot private signing key
[*] Allow potentially insecure options
[ ] Enable flash encryption on boot (READ DOCS FIRST)
    Potentially insecure options --->
```

Рисунок 13.15. Включение аппаратной безопасной загрузки во время компиляции

Примечание

1. Если включена аппаратная безопасная загрузка, обязательно сохраните подписанный файл закрытого ключа, в противном случае обновленный загрузчик и прошивка приложения могут быть не отправлены на устройство.
2. Включение безопасной загрузки приведет к увеличению размера загрузчика, что может потребовать обновления смещений таблицы разделов или уменьшения размера загрузчика. Подробную информацию см. в разделе 13.3.4.
3. Если в прошивку загрузчика было что-то добавлено, убедитесь, что размер загрузчика не превышает `0x10000`.
4. Аппаратная безопасная загрузка сохраняет в eFuse дайджест открытого ключа SHA256, а не сам открытый ключ. Это связано с тем, что сам открытый ключ содержит много данных, но пространство eFuse ограничено.

Посетите <https://book3.espressif.com/bootloader> для получения дополнительной информации о загрузчике.

Если включена аппаратная безопасная загрузка, устройство при обновлении загрузчика и прошивке приложения выполнит следующие проверки.

- (1) **Проверка открытого ключа.** При запуске устройства ROM Boot проверит eFuse. Если аппаратная безопасная загрузка включена, ROM проверяет дайджест открытого ключа в загрузчике и проверяет, соответствует ли он дайджесту открытого ключа в eFuse. Если они не совпадают, это означает, что открытый ключ был подделан или поврежден, и загрузка прерывается; в противном случае открытый ключ в загрузчике считается правильным, и процесс загрузки продолжится.
- (2) **Проверка подписи загрузчика.** ROM Boot использует открытый ключ для проверки подписи загрузчика. Если проверка не пройдена, загрузка будет прекращена; в противном случае процесс продолжится.
- (3) **Проверка подписи исходного приложения `origin_app`.** Загрузчик использует открытый ключ для проверки подписи `origin_app`. Если проверка не удалась, процесс загрузки прекращается.

(4) Проверка подписи нового приложения `new_app` во время OTA-обновлений. Это делается через `origin_app`, аналогично программной безопасной загрузке. На рис. 13.16 показан базовый процесс проверки подписи, выполняемый аппаратной безопасной загрузкой.

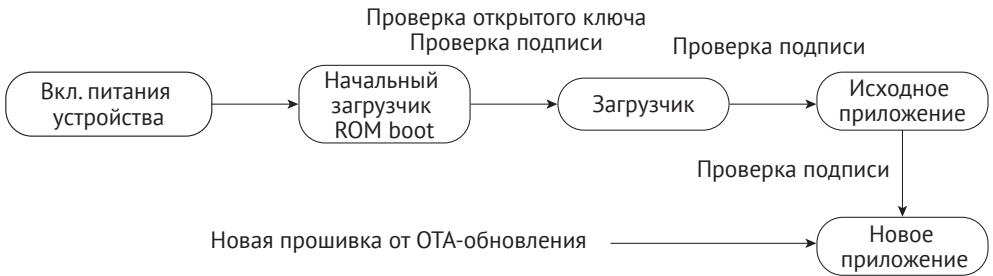


Рисунок 13.16. Базовый процесс проверки подписи с помощью аппаратной безопасной загрузки

Примечание

Полный процесс проверки подписи проверяет не только подписи, но и другие данные, например дайджест прошивки.

Аппаратная безопасная загрузка запускает проверку подписи из ROM Boot, а затем переходит к загрузчику, исходной прошивке `origin_app` и, наконец, новой прошивке `new_app`, шаг за шагом создавая полную цепочку доверия: ROM boot → bootloader → `origin_app` → `new_app`. Из вышеописанного процесса не трудно определить различия между программной и аппаратной безопасными загрузками, см. табл. 13.4.

Таблица 13.4. Различия между программной и аппаратной безопасными загрузками

Пункт	Программная безопасная загрузка	Аппаратная безопасная загрузка
Используется eFuse?	Нет	Да
Объем установленных цепочек доверия	<code>origin_app</code> → <code>new_app</code>	Полная цепочка доверия ROM boot → bootloader → <code>origin_app</code> → <code>new_app</code>
Может ли пара закрытый ключ – открытый ключ быть подменена?	Да. Перепрошивка приложения включит новый пару закрытый ключ – открытый ключ	Нет. Дайджест открытого ключа фиксирован в eFuse
Можно отключить?	Да. Можно отключить пере-прошивкой <code>app.bin</code> с отключенной программной безопасной загрузкой	Нет. Раз аппаратная безопасная загрузка включена, <code>SECURE_BOOT_EN</code> прошит в eFuse, а это означает невозможность отключения

Система аппаратной безопасной загрузки выполняет дополнительную проверку во время всего процесса от ROM Boot до `origin_app`, тем самым увеличивая время запуска устройства и размер загрузчика. В приложениях, в которых устройствам необходимо быстро запускаться или загрузчик должен иметь не большой размер, более подходящей является программная безопасная загрузка.

Если включена аппаратная безопасная загрузка, к устройству будут применены некоторые ограничения:

- на устройстве можно использовать только подписанный загрузчик и прошивку приложения. В итоге перед перепрошивкой необходимо обновить загрузчик и прошивку приложения или обновить прошивку приложения посредством OTA-обновлений, подписанных соответствующими секретными ключами;
- чтобы повысить безопасность системы при включенной аппаратной безопасной загрузке, по умолчанию отладка JTAG отключена. Кроме того, защита чтения eFuse отключена, и неиспользованный слот для подписи в eFuse игнорируется. На этапе разработки эти функции можно оставить через меню конфигурации `menuconfig` → **Security features** (Функции безопасности) → **Potentially insecure options** (Потенциально небезопасные варианты). На этапе серийного производства эти функции следует отключить по умолчанию для повышения общей безопасности устройства;
- когда включена аппаратная безопасная загрузка, функция загрузки устройства через UART изменяется в зависимости от выбранного варианта в меню конфигурации `menuconfig` → **Security features** (Функции безопасности) → **UART ROM download mode** (Режим UART-загрузки ROM). Существует три варианта режима UART-загрузки ROM, показанных в табл. 13.5.

Таблица 13.5. Варианты режима UART-загрузки ROM

Опция	Описание
Enabled	Разрешает чтение/запись флеш-памяти через последовательный порт
Switch to Secure mode	Сохраняет только базовые функции чтения/записи флеш-памяти через последовательный порт. Расширенные функции (например, загрузка зашифрованных прошивок) запрещены
Permanently disabled	Отключает чтение/запись флеш-памяти через последовательный порт

На данный момент мы изучили основные принципы и общее использование аппаратной безопасной загрузки. Существуют также расширенные варианты применения этой схемы, такие как использование нескольких подписей или отмена недействительных открытых ключей.

Примечание

Посетите <https://book3.espressif.com/secure-boot-v2> для ознакомления с руководством пользователя по Secure Boot v2.

13.4.5. Примеры

Функции безопасной загрузки органично интегрированы в ESP-IDF. Самостоятельно ознакомившись с принципами реализации и настройкой соответствующих опций в меню конфигурации, вы можете легко включить эти функции в соответствии с вашими требованиями. По сравнению с программной безопасной загрузкой, аппаратная безопасная загрузка обеспечивает более полную проверку работоспособности прошивки. Таким образом, рекомендуется использовать аппаратное решение безопасной загрузки для повышения безопасности устройства на этапе серийного производства. В этом разделе будет представлено несколько примеров включения аппаратной безопасной загрузки, которые можно использовать в целях тестирования. Кроме того, если вы столкнетесь с какими-либо ошибками при отправке новых прошивок на устройство с использованием аппаратной безопасной загрузки, приведенные далее сообщения в логах могут служить справочной информацией для устранения неполадок.

Когда аппаратная безопасная загрузка включена в соответствии с описанием в разделе 13.4.4, при включении устройства первый раз появится следующее сообщение:

```
I (10251) secure_boot_v2: Secure boot V2 is not enabled yet and eFuses digest keys are not set
I (10256) secure_boot_v2: Verifying with RSA-PSS...
I (10254) secure_boot_v2: Signature verified successfully!
I (10272) boot: boot: Loaded app from partition at offset 0X120000
I (10274) secure_boot_v2: Enabling secure boot V2...
```

При повторном включении устройства появится следующее сообщение:

```
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0xC(SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
Valid Secure Boot key blocks: 0
Secure Boot verification succeeded
load:0x3fcd6268,len:0x2ebc
load:0x403ce000,len:0x928
load:0x403d0000,len:0x4ce4
entry 0x403ce000
I (71) boot: ESP-IDF v4.3.2-2741-g7c0fa3fc70 2nd stage bootloader
```

Перепрошивка неподписанного загрузчика на устройство приведет к появлению следующего сообщения об ошибке, и процесс загрузки будет прерван:

```
ESP-ROM:esp32c3-api1-20210207
Build:Feb 7 2021
rst:0x1 (POWERON),boot:0xC(SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
Valid secure boot key blocks: 0
No signature block magic byte found at signature sector (found 0xcd not 0xe7).
Image not V2 signed?
secure boot verification failed
ets_main.c 333
```


При попытке загрузки на устройство неподписанного приложения появится следующее сообщение об ошибке, и процесс загрузки будет прерван:

```
I (310) esp_image: Verifying image signature...
I (312) secure_boot_v2: Verifying with RSA-PSS...
No signature block magic byte found at signature sector (found 0x41 not 0xe7).
Image not V2 signed?
E (326) secure_boot_v2: Secure Boot V2 verification failed.
E (332) esp_image: Secure boot signature verification failed
I (339) esp_image: Calculating simple hash to check for corruption...
W (418) esp_image: image valid, signature bad
```

Отправка неподписанной прошивки приложения на устройство посредством OTA-обновления приведет к появлению ошибки проверки подписи, прерыванию передачи данных и предотвращению загрузки прошивки:

```
I (4487) simple_ota_example: Starting OTA example
I (5657) esp_https_ota: Starting OTA...
I (5657) esp_https_ota: Writing to partition subtype 16 at offset 0x120000
I (26557) esp_image: segment 0: paddr=00120020 vaddr=3c0a0020 size=1b488h (111752) map
I (26567) esp_image: segment 1: paddr=0013b4b0 vaddr=3fc8d800 size=02b10h ( 11024)
I (26567) esp_image: segment 2: paddr=0013dfc8 vaddr=40380000 size=02050h ( 8272)
I (26577) esp_image: segment 3: paddr=00140020 vaddr=42000020 size=9d9ech (645612) map
I (26667) esp_image: segment 4: paddr=001dda14 vaddr=40382050 size=0b60ch ( 46604)
I (26667) esp_image: segment 5: paddr=001e9028 vaddr=50000000 size=00010h ( 16)
I (26667) esp_image: Verifying image signature...
I (26677) secure_boot_v2: Take trusted digest key(s) from eFuse block(s)
E (26687) esp_image: Secure boot signature verification failed
I (26687) esp_image: Calculating simple hash to check for corruption...
W (26757) esp_image: image valid, signature bad
E (26767) simple_ota_example: Firmware upgrade failed
```

13.5. Практика: функции безопасности в серийном производстве

13.5.1. Флеш-шифрование и безопасная загрузка

Схема флеш-шифрования в первую очередь используется для защиты конфиденциальности данных, сохраненных на флеш-памяти устройства, а схема безопасной загрузки ориентирована на обеспечение легитимности прошивки. Для оптимальной безопасности устройства рекомендуется использовать обе схемы совместно. Как показано на рис. 13.17, флеш-шифрование и безопасная загрузка могут выполняться одновременно в процессе сборки прошивки.

```
App Signing Scheme (RSA) --->
[*] Enable hardware Secure Boot in bootloader (READ DOCS FIRST)
    Select secure boot version (Enable Secure Boot version 2) --->
[*] Sign binaries during build
(security_signing_key.pem) Secure boot private signing key
[ ] Allow potentially insecure options (NEW)
[*] Enable flash encryption on boot (READ DOCS FIRST)
    Enable usage mode (Release) --->
[*] Check Flash Encryption enabled on app startup (NEW)
    UART ROM download mode (UART ROM download mode (Permanently disabled
```

Рисунок 13.17. Включение флеш-шифрования и безопасной загрузки через меню конфигурации

Кроме того, при использовании флеш-шифрования и безопасной загрузки для повышения безопасности устройства обратите внимание на следующее:

- генерируйте разные ключи флеш-шифрования для разных устройств;
- переключите режим загрузки ROM через UART (**UART ROM download mode**) в безопасный режим `Secure mode` или режим отключения `disabled mode` с помощью меню конфигурации `menuconfig` → **Security features** (Функции безопасности);
- сохраните закрытый ключ для подписи в защищенном месте, чтобы он не был утерян или раскрыт. Входите только на безопасном устройстве. Если экспортирован ключ для флеш-шифрования, также сохраните его в защищенном месте.

13.5.2. Включение флеш-шифрования и безопасной загрузки с помощью инструментов пакетной прошивки

В системах Linux для настройки можно использовать такие инструменты, как `esptool.py` и `espssecure.py`, которые могут быть использованы для настройки функций безопасности или защиты данных прошивки. Эти инструменты помогают применять функции безопасности с большей гибкостью.

Для систем Windows используйте инструменты загрузки флеш-памяти (см. <https://www.espressif.com/zh-hans/support/download/other-tools>), которые могут загружать прошивку в пакетном режиме, одновременно как с безопасной загрузкой, так и с шифрованием флеш-памяти. Откройте файл `configure/esp32c3/security.conf` в папке инструментов загрузки и настройте параметры безопасной загрузки и шифрования флеш-памяти. Конфигурационный файл показан на рис. 13.18.

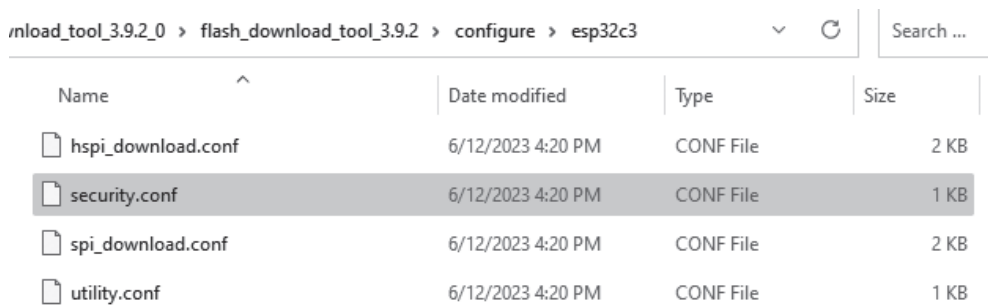


Рисунок 13.18. Файл конфигурации безопасности в инструменте загрузки флеш-памяти

Примечание

Если файл конфигурации безопасности `security.conf` не отображается при первом открытии каталога, закройте программу и снова запустите ее, тогда файл появится.

Настройки безопасности в файле `security.conf` по умолчанию следующие:

```
1. [SECURE BOOT]
2. secure_boot_en = False //Enable secure boot?
3.
4. [FLASH ENCRYPTION]
5. flash_encryption_en = False //Enable flash encryption?
6. reserved_burn_times = 0 //Reserve flash encryption in development mode?
7. //The number of times control bit SPI_BOOT_CRYPT_CNT can be burnt
8.
9. [ENCRYPTION KEYS SAVE]
10. keys_save_enable = False //Save the key for flash encryption locally?
11. encrypt_keys_enable = False //Encrypt the key saved locally?
12. encrypt_keys_aeskey_path = //Key path
13.
14. [DISABLE FUNC]
15. jtag_disable = False
16. dl_encrypt_disable = False
17. dl_decrypt_disable = False
18. dl_cache_disable = False
```

Для получения дополнительной информации обратитесь к руководству пользователя инструмента загрузки флеш-памяти.

Примечание

На этапе производства, когда на устройстве включены флеш-шифрование и безопасная загрузка, важно использовать стандартный и стабильный **источник питания**, иначе устройство может необратимо выйти из строя.

13.5.3. Включение флеш-шифрования и безопасной загрузки в проекте Smart Light

В отличие от других решений, представленных в этой книге, флеш-шифрование и безопасная загрузка могут использоваться практически «из коробки», без необходимости дополнительного кодирования. Эти системы безопасности удобно включать настройкой параметров меню конфигурации. Для системы Smart Light мы рекомендуем использовать одновременно флеш-шифрование, шифрование NVS и аппаратную защиту, чтобы максимально повысить общую безопасность устройства.

13.6. Резюме

В этой главе сначала представлены два ключевых аспекта безопасности устройств интернета вещей: хранение и передача данных. Эти операции имеют особые требования к целостности, конфиденциальности и легитимности данных. Затем в главе обсуждается несколько решений, направленных на решение этих проблем безопасности, в том числе:

- **алгоритм проверки целостности** данных проверяет целостность данных во время процесса загрузки прошивки или OTA-обновления;
- **флеш-шифрование** и **NVS-шифрование** обеспечивают конфиденциальность данных, хранящихся во флеш-памяти, и предотвращают несанкцио-

нированный доступ к ключевым данным. Зашифрованные данные могут быть загружены только после расшифровки с использованием заданного ключа. Включение флеш-шифрования гарантирует, что даже если данные получены с флешки, их нельзя скопировать на другое устройство для загрузки, нарушив права интеллектуальной собственности разработчиков программного обеспечения;

- **система безопасной загрузки** проверяет подлинность и легитимность данных прошивки, гарантируя, что на устройстве будут запускаться только прошивки из проверенных источников.

Наконец, в главе представлен краткий обзор того, как включить флеш-шифрование и обеспечить безопасность загрузки для серийного производства с помощью инструментов загрузки.

Глава 14

Запись и тестирование прошивки для серийного производства

После этапа разработки наступает время пилотных проверочных испытаний и серийного производства.

Пилотные проверочные испытания включают в себя следующее.

EVT (инженерное проверочное испытание, Engineering Verification Test)

EVT выполняется при первой сборке печатной платы (PCBA⁴⁷), чтобы гарантировать, что базовый дизайн соответствует требованиям и спецификациям, включая основные функции оборудования, радиочастотные характеристики, радиочастотные помехи и энергопотребление. Процесс EVT может повторяться столько раз, сколько необходимо для выявления и устранения всех проблем.

DVT (проверочное испытание конструкции, Design Verification Test)

DVT выполняется для всего продукта, чтобы убедиться, что продукт соответствует требованиям и спецификациям, прежде чем переходить к серийному производству. DVT включает температурные испытания, испытание на электростатический разряд (electrostatic discharge, ESD) и испытание на падение.

Сертификация продукции

Как только продукт пройдет EVT и DVT, прототип может быть подготовлен для национального или отраслевого тестирования на сертификаты, такие как SRRC, FCC, CE и т. д.

После пробного производства продукт готов к серийному производству. Серийное производство предполагает множество этапов, таких как подготовка материала, монтаж, прошивка, тестирование, упаковка и т. д. Эта глава посвящена только двум этапам, которые тесно связаны с продуктами Espressif, – прошивке и тестированию продукции.

14.1. Загрузка прошивки при серийном производстве

Прошивка для серийного производства в основном состоит из двух частей: прошивки приложения и разделов данных. В этом разделе основное внима-

⁴⁷ Printed Circuit Board Assembly, сборка (монтаж) печатной платы.

ние уделяется подготовке микропрограммы приложения и разделов данных, а также способам их прошивки.

14.1.1. Определение разделов данных

Чтобы идентифицировать различные умные продукты на рынке и связать их с пользователями, в каждом интеллектуальном продукте часто необходимо хранить уникальные данные. Например, для эффективного подключения IoT-продуктов к облачной платформе поставщика уникальная аутентификационная информация (например, сертификат устройства, идентификатор и пароль) должна быть сгенерирована и сохранена в каждом устройстве. Она будет использоваться на стороне сервера при подключении и аутентификации интеллектуального продукта.

На этапе разработки мы можем легко сохранить в устройстве информацию для аутентификации, определяя константы и сохраняя их в прошивке или записывая во флеш-память. Но в серийном производстве эти методы становятся неуклюжими и неэффективными. Поэтому для записи разделов данных в реальном производстве необходим более удобный метод.

В части II «Разработка оборудования и драйверов», раздел 6.4.1, мы представили библиотеку NVS, которая при серийном производстве интеллектуальных продуктов может быть вариантом хранения уникальных данных, а также любых других пользовательских данных, связанных с приложением. Пользовательские данные в умных продуктах часто считываются и изменяются и будут стерты при возврате к заводским настройкам, в то время как уникальные производственные данные в серийных продуктах можно только прочитать. Поэтому производственным данным и пользовательским данным будут назначены разные пространства имен, например `mass_prod` (для производственных данных) и `user_data` (для пользовательских данных). Это позволяет напрямую стереть пользовательские данные, в то же время сохраняя производственные данные без изменений во время сброса к заводским настройкам. Кроме того, производственные данные и пользовательские данные также могут храниться отдельно в разных разделах NVS.

Следующий код показывает, как сохранить сертификат продукта в `mass_prod` и SSID Wi-Fi в `user_data`:

```
1. nvs_handle_t mass_prod_handle = NULL;
2. nvs_handle_t user_data_handle = NULL;
3. //Initialize NVS Flash Storage
4. nvs_flash_init_partition(partition_label);
5.
6. //Open non-volatile storage with mass_prod namespace
7. nvs_open("mass_prod", NVS_READONLY, &mass_prod_handle);
8.
9. //Open non-volatile storage with user_data namespace
10. nvs_open("user_data", NVS_READWRITE, &user_data_handle);
11.
12. uint8_t *product_cert = malloc(2048);
13. //read operation in mass_prod namespace
14. nvs_get_blob(mass_prod_handle, "product_cert", &product_cert);
15.
```

```

16. char ssid[36] = {0};
17. //read operation in user_data namespace
18. nvs_get_str(user_data_handle, "ssid", &ssid);
19. //write operation in user_data namespace
20. nvs_set_str(user_data_handle, "ssid", &ssid);
21.
22. //Erase user_date namespace when reset to factory
23. nvs_erase_all(user_data_handle);

```

Теперь, когда мы знаем, как хранить производственные данные при серийном производстве, нам нужно перед прошивкой на устройство преобразовать их в необходимый формат. Основные этапы создания производственных данных показаны на рис. 14.1.



Рисунок 14.1. Основные этапы создания производственных данных

Сначала создайте CSV-файл для хранения пар ключ–значение; запишите необходимые данные в файл. Для серийного производства на основе этого CSV-файла необходимо создать раздел NVS для двоичного файла, а затем прошить его на устройство. Для каждого произведенного устройства в раздел NVS будет прошит уникальный двоичный файл.

Например:

```

1. key,          type,          encoding,  value
2. mass_prod,   namespace, ,
3. ProductID,  data,         string,   12345
4. DeviceSecret, data,         string,   12345678901234567890123456789012
5. DeviceName, data,         string,   123456789012

```

Затем используйте `esp-idf/components/nvs_flash/nvs_partition_generator/nvs_partition_gen.py` для создания раздела NVS на хосте разработки с помощью следующей команды:

```

$ python $IDF_PATH/components/nvs_flash/nvs_partition_generator/nvs_partition_gen.py --input mass_prod.csv --output mass_prod.bin --size NVS_PARTITION_SIZE

```

Примечание

Замените параметр `NVS_PARTITION_SIZE` фактическим размером соответствующего раздела NVS в таблице разделов.

После выполнения вышеуказанной команды полученный файл `mass_prod.bin` – это двоичный файл для серийного производства. Запустите следующую команду для записи этого файла на флеш-память устройства:

```

$ python $IDF_PATH/components/esptool_py/esptool/esptool.py --port $ESPPORT write flash NVS_PARTITION_ADDRESS mass_prod.bin

```

Примечание

Замените параметр NVS_PARTITION_ADDRESS фактическим адресом соответствующего раздела NVS в таблице разделов.

14.1.2. Запись прошивки

При серийном производстве двоичные файлы, которые необходимо записать на устройство, включают:

- прошивки для устройств серийного производства;
- двоичные файлы данных серийного производства.

Во время записи необходимо следить за тем, чтобы на каждое устройство записывался уникальный бинарный файл с данными, при этом прошивка приложения обычно одинакова для всех устройств. Чтобы добиться этого, мы можем написать скрипт для создания уникального двоичного файла с производственными данными для каждого устройства на основе его MAC-адреса. Затем этот файл можно записать в устройство вместе с прикладной прошивкой. В этом процессе также может быть создана таблица, связывающая MAC-адрес каждого устройства с его производственными данными для запросов, отладки и отслеживания.

Espressif может настроить прошивку для наших модульных продуктов в соответствии с требованиями клиентов. Например, мы можем выполнить настройку безопасности для модулей серии ESP32-C3, включая уникальные производственные данные для каждого устройства. Таким образом, модули Espressif могут быть установлены непосредственно в пользовательскую аппаратуру, что экономит затраты производителей и снимает проблему вторичной прошивки. Espressif также предоставляет инструмент прошивки флеш-памяти Flash Download Tool, который можно использовать для одновременной записи нескольких устройств в специальном фабричном режиме. Подробности о Flash Download Tool можно найти в документации на нашем сайте.

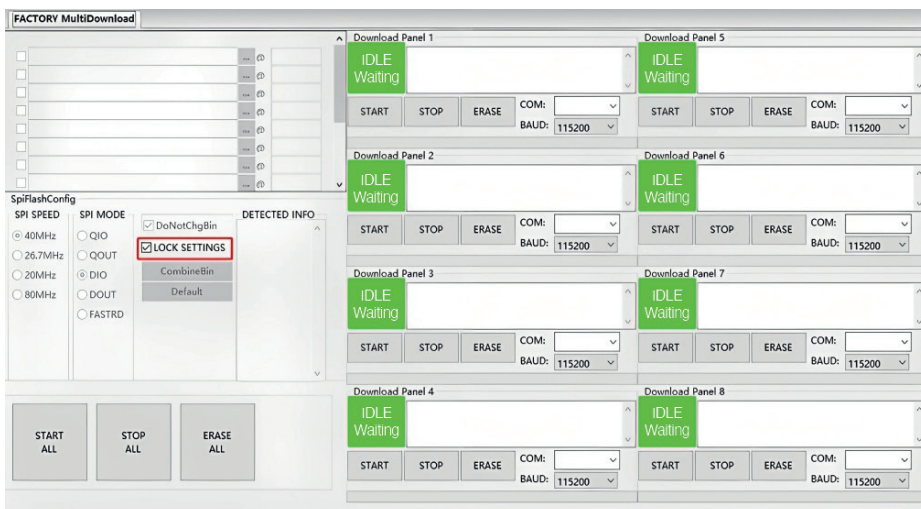


Рисунок 14.2. Интерфейс фабричного режима в Flash Download Tool

Интерфейс фабричного режима Flash Download Tool показан на рис. 14.2. На фабрике Flash Download Tool загружает прошивку по относительным путям, и по умолчанию она хранится в каталоге *bin* на устройстве. Пользователи могут хранить свою прошивку в каталоге инструментов папки *bin*, чтобы они могли копировать проект на разные компьютеры без ошибок, связанных с указанием пути.

14.2. Тестирование серийной продукции

Цель серийного производственного тестирования – гарантия, что функциональность и эффективность всей продукции соответствуют стандартам, поэтому необходимо полностью проверить функции и эффективность каждого продукта на этом этапе. В зависимости от характеристик продукта схемы испытаний массового производства могут немного отличаться. Для продуктов с радиосвязью необходимо убедиться, что характеристики RF-компонента соответствуют стандартам и каждый компонент функционирует так, как ожидалось. Обычно это включает в себя проверку радиочастотных характеристик, проверку энергопотребления и функциональное тестирование различных периферийных устройств.

Для беспроводных продуктов общие испытания при серийном производстве включают электромагнитную совместимость (EMC), тест RF-эффективности, тест на соответствие производственному стандарту, тест на безопасность, SAR⁴⁸ и т. д., среди которых большое значение имеет тест на RF-эффективность, требующий большого количества испытаний⁴⁹. Он проводится для проверки работоспособности радиочастотного продукта и соответствия его установленным требованиям и соответствующим стандартам. Тест включает в себя два набора измерений – характеристики передатчика (TX) и приемника (RX).

В этом разделе в основном представлены схемы серийного тестирования модулей и чипов Espressif с наличием компонентов Wi-Fi/Bluetooth Low Energy (BLE), которые можно использовать в качестве эталона для разработки схем тестирования аналогичной продукции. Для получения подробной информации, пожалуйста, обратитесь к руководству по тестированию продукции Espressif Production Testing Guide на нашем официальном сайте. Обычно для проверки радиочастотных характеристик продукции применяются два метода тестирования: метод универсального ВЧ-тестера **RF General-purpose Tester** (широко распространенный в отрасли) и метод сигнальной платы **Signal Board** (разработан Espressif).

Метод универсального ВЧ-тестера

Метод ВЧ-тестера широко используется для производственного тестирования Wi-Fi/BLE-продуктов. Espressif предоставляет необходимые команды последовательного порта и встроенное ПО, поэтому клиенты могут легко использовать эту схему для тестирования. Этапы тестирования показаны на рис. 14.3.

⁴⁸ Specific Absorption Rate – коэффициент удельного поглощения, выражается в Вт/кг.

⁴⁹ По поводу RF-теста см. также раздел 5.2.5.

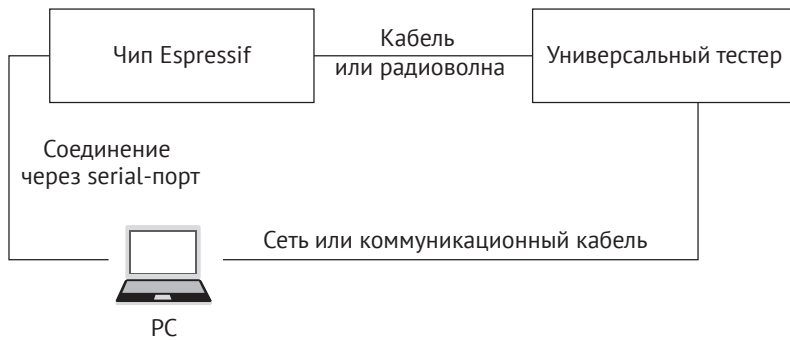


Рисунок 14.3. Схема метода тестера

Схема сигнальной платы

Схема сигнальной платы специально разработана компанией Espressif, которая может эффективно тестировать радиочастотные характеристики серийно выпускаемых продуктов Wi-Fi/BLE, что гарантирует качество RF-компонентов. Эта схема отличается низкой стоимостью оборудования и простой настройкой среды для использования на фабрике.

Как показано на рис. 14.4, сигнальную плату можно использовать в качестве стандартного устройства для связи с тестируемым устройством (DUT) и протестировать его, проанализировав обмен данными. Схема аппаратного подключения сигнальной платы представлена на рис. 14.5.

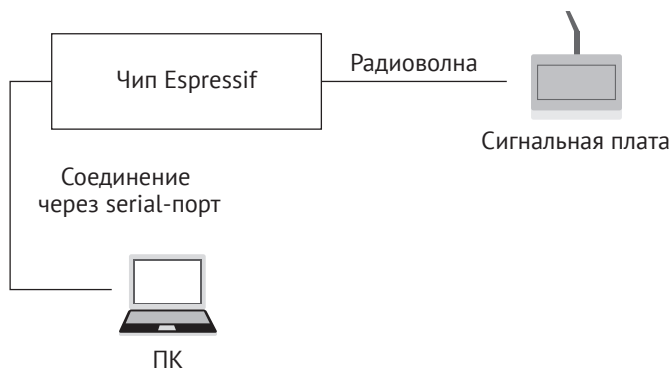


Рисунок 14.4. Схема метода сигнальной платы



Рисунок 14.5. Подключение аппаратуры для схемы сигнальной платы

14.3. Практика: производственные данные в проекте Smart Light

В проекте Smart Light, представленном в этой книге, каждое устройство должно хранить уникальную информацию для идентификации, а также некоторую общую рабочую конфигурацию. Общедоступный ESP RainMaker, используемый в проекте, предназначен для прототипирования и оценки качества максимум пяти устройств. Вы можете создавать двоичные файлы производственных данных на основе полученных учетных данных и изменять соответствующие исходные коды для устройств массового производства. Чтобы получить учетные данные для подключения к облаку в большем масштабе, свяжитесь с Espressif для частных развертываний ESP RainMaker.

14.4. Резюме

В этой главе мы представили массовое тестирование продуктов на основе модулей и чипов Espressif, а также запись производственной прошивки, чтобы читатели могли иметь предварительное представление о массовом производстве. Для идентификации и привязки устройств к пользователям уникальные производственные данные должны быть записаны на каждое устройство. Чтобы спасти производителей от проблемы с записью прошивки при массовом производстве, Espressif предоставляет модули с индивидуальным встроенным ПО, которые можно устанавливать непосредственно в аппаратную схему продукта.

ESP Insights: платформа удаленного мониторинга

В предыдущих главах была представлена облачная платформа ESP RainMaker IoT Cloud, решение для передачи данных между устройством и облаком от Espressif. Используя компоненты облачной платформы ESP RainMaker IoT Cloud, пользователи могут легко подключаться к ESP RainMaker, реализуя удаленное управление устройствами. С помощью ESP RainMaker IoT Cloud пользователи могут разрабатывать умные светодиодные продукты на основе ESP32-C3 с легкостью. Но мы все знаем, что от утверждения проекта до массового производства продукт должен пройти основные процессы, включая функциональную оценку, реализацию и проверку.

На основе введения о функциях интеллектуальных светодиодных продуктов и о том, как они работают, изложенного в предыдущих главах, вы можете задать вопросом, как проводить систематическую функциональную проверку и проверку в подключенном состоянии.

После разработки функционального кода проекта требуется функциональная проверка. В это время вы можете установить уровень лога в режим отладки и отслеживать лог на последовательном порту, чтобы получать результаты отладки. Это необходимая проверка перед выпуском программного обеспечения. Завершив базовую функциональную проверку, такие функции, как вывод лога и командная строка, обычно следует отключить. Затем пришло время выпустить рабочий релиз. После этого, даже если программное обеспечение ведет себя ненормально во время теста качества (QA) или во время использования, разработчикам будет сложно быстро обнаружить и устранить проблемы с помощью логов устройства. Иногда разработчикам может даже потребоваться разобрать устройство для анализа причины неисправности. Чтобы решить эту проблему, Espressif Systems разработал ESP Insights, который помогает разработчикам проверять текущий статус и логи прошивки удаленно, чтобы вовремя обнаружить и решить проблемы с прошивкой, ускоряя процесс разработки программного обеспечения.

15.1. Введение в ESP Insights

ESP Insights (ссылка на проект: <https://github.com/espressif/esp-insights>) – средство удаленного мониторинга, платформа, позволяющая пользователям удаленно следить за состоянием устройства, включая предупреждения и логи ошибок, метрики рабочих параметров устройства, информацию о дампе ядра устройства, а также пользовательские данные и события.

В этой главе мы познакомим вас с функциями и приложениями ESP Insights на основе проекта *esp-insights*. Идентификатор проекта (commit ID⁵⁰): afd70855eb4f456e7ef7dc233bf082ec7892d9df.

ESP Insights включает в себя агент прошивки, агент Insights, собирающий важные фрагменты диагностической информации с устройства во время выполнения и загружающий ее в облако ESP Insights Cloud. Затем облако обрабатывает эти данные для визуализации. Разработчики могут войти в систему на веб-панель мониторинга, чтобы просмотреть состояние устройств и проблемы, о которых сообщают их устройства во время работы. В настоящее время мы поддерживаем только обработку диагностической информации и отчетов по облачной платформе ESP RainMaker IoT Cloud. Поддержка других облачных платформ будет доступна в более поздних релизах. На рис. 15.1 представлен обзорный отчет облачной платформы ESP RainMaker. На рис. 15.2 дан отчет облачной платформы ESP RainMaker о метриках. Рисунок 15.3 представляет отчет облачной платформы ESP RainMaker о значениях переменных.

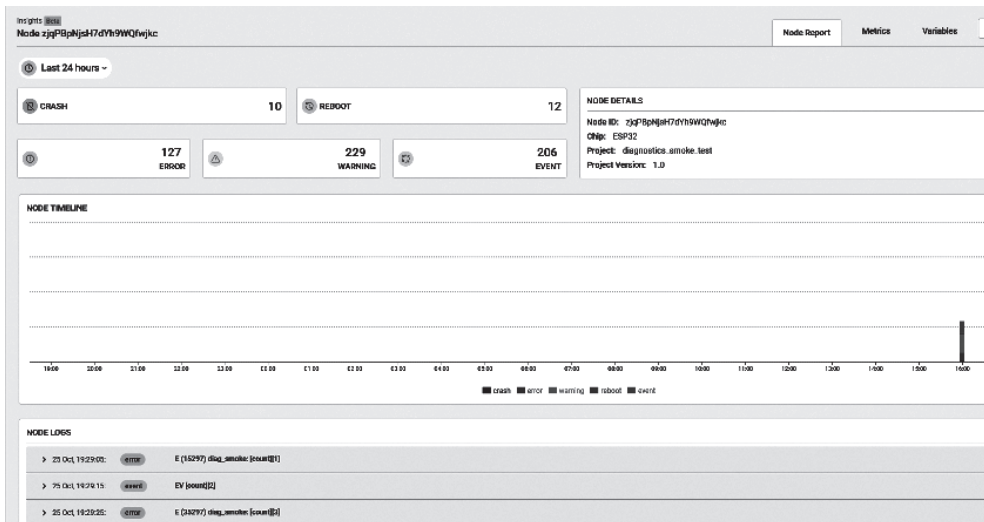


Рисунок 15.1. Обзорный отчет от ESP RainMaker

В настоящее время разработчики могут отслеживать следующую информацию на веб-панели:

Логи ошибок

Выводятся через последовательный порт, когда функция вывода лога `ESP_LOGE()` вызывается компонентами или пользовательскими приложениями.

Логи предупреждений

Выводятся через последовательный порт, когда функция вывода лога `ESP_LOGW()` вызывается компонентами или пользовательскими приложениями.

⁵⁰ Commit – так в терминологии среды разработки проектов Git называются изменения, отправленные в репозиторий Git. Идентификатор commit ID фиксирует группу изменений на определенный момент.

Пользовательские события

Выводятся через последовательный порт, когда ESP_DIAG_EVENT() вызывается пользовательскими приложениями. Обычно события применяются для пользовательских данных.

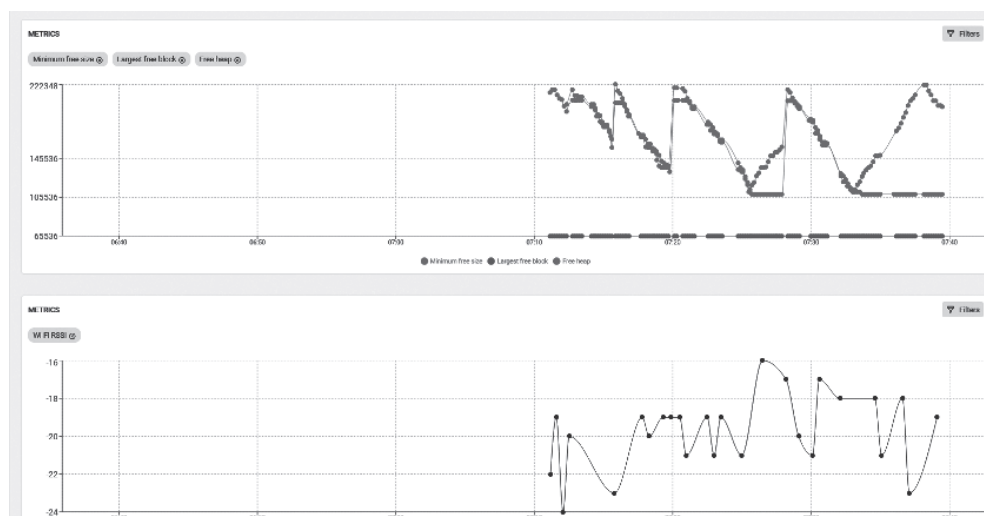


Рисунок 15.2. Отчет ESP RainMaker по метрикам

The figure shows a screenshot of the ESP RainMaker Variables report. It is divided into two main sections: 'IP' and 'Wi-Fi'. Each section contains a table of variables and their values, along with the last update time.

IP		
Station		
Gateway:	192.168.2.1	Last updated Today at 19:28
IPv4:	192.168.2.5	Last updated Today at 19:28
Netmask:	255.255.255.0	Last updated Today at 19:28

Wi-Fi		
Station		
Auto Mode:	4	Last updated Today at 19:28
BSSID:	84:29:12:42:C8:40	Last updated Today at 19:28
Channel:	6	Last updated Today at 19:28
Connected:	true	Last updated Today at 19:28
Last Wi-Fi disconnect reason:	-	-
SSID:	HUAWEL_WSN200_XM.1	Last updated Today at 19:28

Рисунок 15.3. Отчет ESP RainMaker о значениях переменных

Причина сброса

Причины, по которым происходит сброс устройства, например включение питания, программный сброс, отключение питания и т. д.

Сводка дампа памяти

Зарегистрируйте содержимое и отслеживайте стек проблемного потока на случай сбоя.

Метрики

Изменяющиеся во времени данные, например размер свободной кучи, изменения уровня сигнала Wi-Fi с течением времени и т. д.

Переменные

Значения переменных, например IP-адрес устройства, адрес шлюза, информация о подключении Wi-Fi и т. д.

1. Особенности ESP Insights

- (1) Проверьте свойства устройства (например, имя, идентификатор, версию прошивки и т. д.) и состояние устройства (например, использование памяти, максимальный свободный блок, значение свободной кучи, уровень сигнала Wi-Fi и т. д.).
- (2) Проверьте логи, созданные во время работы прошивки устройства, такие как логи ошибок и предупреждений, информацию трассировки сбоев, перезагрузки и другие пользовательские события.
- (3) Проверьте текущие данные, сообщаемые устройством, и создайте таблицы данных в соответствии со временем.
- (4) Поддержка настраиваемых показателей и переменных в зависимости от потребностей пользователей.

2. Преимущества ESP Insights

- (1) Ускорение разработки и выпуска программных продуктов.

Перед официальным выпуском каких-либо программных продуктов обычно требуется бета-тестирование. Во время бета-тестирования пользователи предоставят отзывы о производительности, стабильности, надежности и других проблемах продукта в реальных вариантах использования, которые затем будут обработаны и исправлены разработчиками. Этот процесс часто обходится разработчикам в большую сумму времени и усилий на обнаружение проблем и анализ причин. Благодаря ESP Insights разработчики могут удаленно проверять состояние работы устройства и получать подробную информацию об аномальных событиях своевременно, что значительно экономит время на решение проблем и ускоряет процесс разработки и выпуска программного обеспечения. ESP Insights также сохраняет записи аномальных событий, произошедших до сбоя прошивки устройства. После того как устройство перезагрузится, оно загружает данные в облако, избегая таким образом потери информации о сбое.

- (2) Своевременное решение различных проблем с прошивкой. Например:
 - А) разработчики могут использовать ESP Insights для проверки состояния устройства (например, объема доступной памяти, максимального свободного блока, мощности сигнала Wi-Fi и т. д.), проанализировать пиковое значение каждого показателя и внедрить оптимизацию в будущих версиях прошивки;

Б) в логах ESP Insights фиксируются подробности всех аномальных событий, чтобы разработчики могли своевременно обработать ошибку до того, как она будет обнаружена пользователем, предотвращая любое влияние неисправности на фактическое использование устройства.

(3) Передача данных: легкая, простая, безопасная и надежная.

ESP Insights способен передавать диагностические данные по протоколам HTTPS и MQTT. При работе с облачной платформой ESP RainMaker IoT Cloud ESP Insights поддерживает обмен диагностическими данными, передаваемыми по зашифрованному каналу через протокол MQTT, что значительно сокращает использование памяти устройства при обеспечении информационной безопасности. Если вы не используете облачную платформу ESP RainMaker, то можете применять только протокол HTTPS для передачи диагностических данных. Однако по сравнению с ESP RainMaker IoT Cloud использование только протокола HTTPS требует добавления TLS-соединения, которое приведет к увеличению использования памяти. Данные, передаваемые между устройством и облачной платформой, оптимизированы за счет кодировки CBOR, что существенно экономит полосу пропускания передачи данных. В будущем ESP Insights также будет интегрировать данные устройств с данными управления и контроля из облака и упаковывать их в одно и то же сообщение MQTT, что еще больше снизит затраты за счет меньшего количества сообщений MQTT.

15.2. Начало работы с ESP Insights

В этом разделе мы расскажем, как начать работу с ESP Insights на основе проекта *esp-insights*, учитывая особенности и преимущества ESP Insights, и узнаем, как проверить информацию об устройстве на удаленной приборной панели.

15.2.1. Начало работы с ESP Insights в проекте *esp-insights*

Чтобы начать работу с ESP Insights в проекте *esp-insights*, выполните следующие действия:

1. Клонировать последнюю версию *esp-RainMaker*.

Основываясь на предыдущем знакомстве с облачной платформой ESP RainMaker IoT Cloud, извлеките код проекта *esp-RainMaker*, при этом *esp-insights* будет находиться в каталоге проекта *esp-RainMaker/Components* в качестве подмодуля.

```
$ git clone --recursive https://github.com/espressif/esp-RainMaker.git
```

2. Измените *CMakeLists.txt* в *esp-RainMaker*.

Добавьте *esp-insight* в качестве компонента в проект *esp-RainMaker*, гарантируя, что функции *esp-insight* можно вызывать в проекте *esp-RainMaker*. В текущем каталоге сборки проекта измените следующую команду в *CMakeLists.txt*:

```
1. set(EXTRA_COMPONENT_DIRS ${RMAKER_PATH}/components ${RMAKER_PATH}/examples/  
common)
```

на:


```
1. set(EXTRA_COMPONENT_DIRS ${RMAKER_PATH}/components ${RMAKER_PATH}/examples/  
common ${RMAKER_PATH}/components/esp-insights/components)
```

3. Обеспечьте выполнение функций ESP Insights.

Код ESP Insights уже содержится в компоненте *example/common/app_insights*.

Пользователям нужно только включить *app_insights.h* в свой код и вызвать функцию *app_insights_enable()* перед вызовом *esp_rmaker_start()*. Однако этот компонент контролируется макросом *CONFIG_ESP_INSIGHTS_ENABLED*, который по умолчанию отключен. Пользователи могут включить эту функцию в конфигурации по умолчанию или в интерфейсе конфигурации образа (используйте инструмент *idf.py*, чтобы открыть в меню конфигурации *menuconfig* → **Component config** (Конфигурация компонента) → **ESP Insights** → **Enable ESP Insights** → (Включить ESP Insights).

4. Сборка и прошивка

Запустите следующую команду для сборки и прошивки:

```
$ idf.py build flash monitor
```

Когда сборка завершится, следующий лог будет создан в виде файла *led_light-v1.0.zip* в каталоге сборки *build* для будущего использования:

```
=====  
Generating insights firmware package build/led_light-v1.0.zip  
=====  
led_light-v1.0  
led_light-v1.0/led_light.bin  
led_light-v1.0/sdkconfig  
led_light-v1.0/partition_table  
led_light-v1.0/partition_table/partition-table.bin  
led_light-v1.0/bootloader  
led_light-v1.0/bootloader/bootloader.bin  
led_light-v1.0/partitions.csv  
led_light-v1.0/project_build_config.json  
led_light-v1.0/led_light.map  
led_light-v1.0/led_light.elf  
led_light-v1.0/project_description.json
```

5. Доступ к облачной платформе ESP RainMaker IoT Cloud

Поскольку разработчикам необходим доступ администратора к облаку ESP Insights, необходимо подать заявку. Подробную информацию о доступе можно найти в главе 3.

6. Войдите в панель управления ESP RainMaker.

После завершения прошивки и оформления заявки устройство готово к подключению к ESP RainMaker. Теперь пользователи могут войти в интерфейс ESP RainMaker (<https://dashboard.RainMaker.espressif.com/>) и открыть панель управления устройством, щелкнув на соответствующий узел.

7. Загрузите сгенерированный zip-файл.

Чтобы лучше понять диагностическую информацию, пользователям также необходимо загрузить ранее сгенерированный zip-файл *led_light-v1.0.zip* в образы прошивки на левой панели навигации интерфейса ESP RainMaker, так как zip-файл содержит двоичные файлы, файлы elf, файлы соответствий (mapping) и другую полезную информацию для анализа.

Даже без изменений в коде такие команды, как `idf.py build` и `idf.py flash`, проведут пересборку для создания новой прошивки. Таким образом, важно обеспечить, чтобы прошивка, работающая на устройстве, соответствовала zip-архиву, загруженному в платформу ESP RainMaker. В противном случае ESP RainMaker может сообщить об ошибках при обработке и анализе информации, поступающей с устройства.

15.2.2. Пример выполнения в проекте esp-insights

1. Клонировать ESP Insights.

Клонировать код проекта для ESP Insights, используя следующую команду:

```
$ git clone --recursive https://github.com/espressif/esp-insights.git
```

2. Настройте ESP-IDF.

ESP Insights в настоящее время поддерживает основную ветку `master` и версии `v4.3.x`, `v4.2.x` и `v4.1.x`.

Чтобы получить поддержку версии `4.3.2`, вам необходимо запустить следующую команду для установки патча:

```
$ cd $IDF_PATH
$ git apply -v <path/to/esp-insights>/idf-patches/Diagnostics-support-in-espidf-tag-v4.3.2.patch
```

Чтобы получить поддержку `v4.2.2` и `v4.0.0`, для установки патча необходима следующая команда:

```
$ cd $IDF_PATH
$ git apply -v <path/to/esp-insights>/idf-patches/Diagnostics-support-in-espidf-tag-v4.1.1-and-tag-v4.2.2.patch
```

Пользователи в соответствии со своими потребностями могут выбрать протокол HTTPS или протокол MQTT для передачи диагностических данных. Для конкретных конфигураций используйте следующую команду:

```
$ idf.py menuconfig
```

Перейдите в **Component config** (Конфигурация компонента) → **ESP Insights** → **Insights default transports** (Передача Insights по умолчанию).

Если для передачи диагностических данных выбран протокол HTTPS, пользователям необходимо войти в систему <https://dashboard.insights.espressif.com/home/insights>, чтобы проверить лог диагностики устройства.

3. Сборка и прошивка.

См. шаги 4–7 в разделе 15.2.1.

15.2.3. Отчетность об информации дампа памяти

В случае сбоя встроенного ПО агент Insights записывает информацию дампа памяти при сбое (`coredump`) во флеш-память и сообщает об этом облаку ESP Insights Cloud при последующей загрузке. Это позволяет вам при необходи-

мости просмотреть все логи сбоев, которые устройства могут создавать в полевых условиях.

Вся обратная трассировка стека, приведшая к сбою, также фиксируется и выводится. Для оптимизации связи между устройством и облаком прошивка отображает только сводку дампа памяти. Сводка содержит наиболее полезное содержимое дампа памяти: счетчик программ, причину исключения, адрес исключения, регистры общего назначения и обратную трассировку. На рис. 15.4 показана часть информации дампа.



```
> 16 Jan, 11:45:56: error E (25697) diag_smoke: [count][0] Crashing...
> 16 Jan, 11:45:57: crash Exception StoreProhibitedCause has occurred in task smoke_test.

Register Values
PC: 0x40087109      A0: 0x00087520      A1: 0x3ffdc630      A2: 0x3ff48008
A3: 0x2f000000      A4: 0xc0000000      A5: 0xffffffff      A6: 0x00000000
A7: 0xffffffffe     A8: 0x00000000      A9: 0x3ffdc610      A10: 0x00000000
A11: 0x10000000     A12: 0x3ffdc56f     A13: 0x30000000     A14: 0x00000000
A15: 0x3ffdc579     EXCCAUSE: 0x0000001d EXCVADDR: 0x00000000

Backtrace
0x40087109: invoke_abort at /Users/kedars/work/idf_v4.1/components/esp32/panic.c : 157
0x4008751d: abort at /Users/kedars/work/idf_v4.1/components/esp32/panic.c : 174
0x4014d7e3: __assert_func at /builds/idf/crosstool-NG/.build/HOST-x86_64-apple-darwin12/xtensa-esp32-elf/src/newlib/newlib/libc/stdlib/assert.c : 62
0x400d6571: smoke_test at /Users/kedars/work/rainmaker/esp-rainmaker-diagnostics/examples/diagnostics_smoke_test/main/app_main.c : 60
0x40087bfa: vPortTaskWrapper at /Users/kedars/work/idf_v4.1/components/freertos/port.c : 143

16 Jan, 11:45:57: reboot Software reset due to exception/panic (ESP_RST_PANIC)
```

Рисунок 15.4. Информация дампа памяти

Для этой функции требуются следующие настройки, которые следует добавить в файл конфигурации проекта по умолчанию *sdkconfig.defaults*:

1. CONFIG_ESP32_ENABLE_COREDUMP=y
2. CONFIG_ESP32_ENABLE_COREDUMP_TO_FLASH=y
3. CONFIG_ESP32_COREDUMP_DATA_FORMAT_ELF=y
4. CONFIG_ESP32_COREDUMP_CHECKSUM_CRC32=y
5. CONFIG_ESP32_CORE_DUMP_MAX_TASKS_NUM=64

Для сохранения дампа во флеш-памяти требуется дополнительный раздел. Необходимо добавить следующую строку в файл *parts.csv* проекта:

1. coredump, data, coredump, , 64K

15.2.4. Настройка интересных логов

esp_log – это компонент ведения журнала по умолчанию в ESP-IDF. Обычно команды *ESP_LOGE* и *ESP_LOGW* используются для регистрации ошибок и предупреждений в прошивке. Все логи, записанные с использованием компонента *esp_log*, отслеживаются агентом Insights и передаются в облако ESP Insights. Это позволяет разработчикам просматривать ошибки через панель мониторинга ESP Insights, предоставляющую подробную информацию о происходящем.

Разработчики могут настроить уровень логов, вызвав функции *esp_diag_log_hook_enable()* и *esp_diag_log_hook_disable()*.

```
1. /*enable tracking error logs*/
2. esp_diag_log_hook_enable(ESP_DIAG_LOG_TYPE_ERROR);
3.
4. /*enable tracking all log levels*/
5. esp_diag_log_hook_enable(ESP_DIAG_LOG_TYPE_ERROR|ESP_DIAG_LOG_TYPE_WARNING|
6. ESP_DIAG_LOG_TYPE_EVENT);
7.
8. /*disable tracking custom events*/
9. esp_diag_log_hook_disable(ESP_DIAG_LOG_TYPE_EVENT);
```

Обычно перед сбоем устройства выводятся какие-то логи ошибок или предупреждений, что сложно для передачи в облако. Агент ESP Insights предоставляет возможность выводить логи и отчеты в облако ESP Insights Cloud после перезагрузки. ESP32-C3 для подобных целей оснащен памятью RTC⁵¹. Агент Insights использует эту память для хранения критических ошибок, произошедших в системе. При загрузке чипа агент Insights проверяет наличие незарегистрированных ошибок из предыдущей загрузки в памяти RTC и сообщает об этом в облако ESP Insights.

15.2.5. Сообщение о причине перезагрузки

По умолчанию агент Insights поддерживает сообщение о причине перезагрузки устройства при каждой загрузке в облако. Это позволяет разработчикам определить, перезагрузилось ли устройство из-за сбоя, срабатывания сторожевого таймера, программного сброса или сброса питания конечным пользователем.

15.2.6. Отчетность по заданным показателям

Агент Insights поддерживает запись показателей и отправку отчетов в облако. Вы можете просматривать графики через панель мониторинга Insights, где отображаются изменения этих показателей в течение определенного периода времени.

Установите `CONFIG_DIAG_ENABLE_METRICS=y`, чтобы включить поддержку вывода показателей. Агент Insights может записывать набор заранее определенных системных показателей, таких как объем памяти и уровень сигнала Wi-Fi. Кроме того, вы можете добавить свои собственные показатели. На рис. 15.5 представлен вывод некоторых показателей.

1. Показатели кучи

Агент Insights поддерживает отчеты о свободной памяти, наибольшем свободном блоке и минимальном свободном блоке памяти в любой момент времени. Эти параметры отслеживаются и сообщаются для кучи во внутренней оперативной памяти, а также для кучи во внешней оперативной памяти (в случае если устройство имеет PSRAM). Агент Insights также записывает неудачные выделения памяти, которые доступны в версиях ESP-IDF v4.2 и выше.

Установите `CONFIG_DIAG_ENABLE_HEAP_METRICS=y`, чтобы включить показатели кучи.

⁵¹ См. сноску на стр. 97. Модуль RTC в ESP32 оснащен отдельной быстрой памятью, с которой питание не снимается даже в режиме глубокого сна Deep-sleep mode.

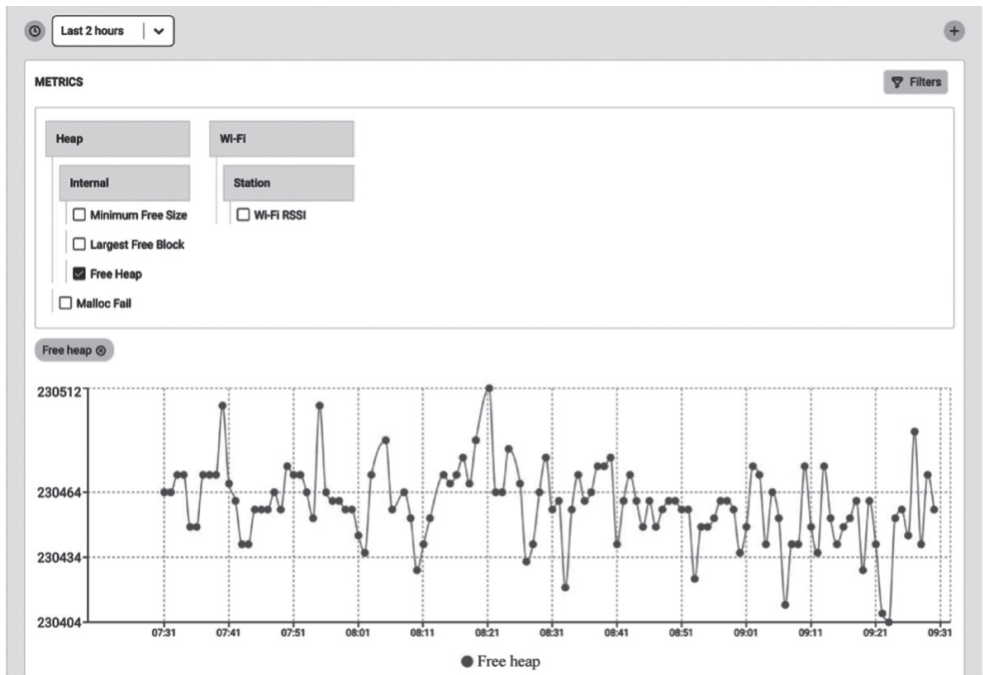


Рисунок 15.5. Информация о показателях

2. Показатели Wi-Fi

Агент ESP Insights поддерживает и показатели Wi-Fi. Он собирает уровень сигнала Wi-Fi (RSSI⁵²) и информацию о минимальном значении RSSI. RSSI замеряется каждые 30 секунд, и если есть 5 дБ разницы между предыдущим и текущим подсчетами, об этом будет сообщено в облако ESP Insights Cloud. Начиная с ESP-IDF v4.3 также записывается минимальный уровень RSSI, когда его значение падает ниже предварительно настроенного порога. Порог можно настроить с помощью вызова `esp_wifi_set_rssi_threshold()`. Также есть функция, которая может собирать и сообщать показатели Wi-Fi в любой заданный момент времени:

1. /*Отправляет RSSI в облако, а также выводит на консоль*/
2. `esp_diag_wifi_metrics_dump();`

3. Пользовательские показатели

Разработчики могут добавлять собственные показатели с помощью следующих функций:

1. /*Зарегистрировать метрику для отслеживания температуры в помещении*/
2. `esp_diag_metrics_register("temp", "temp1", "Room temperature", "room",`
3. `ESP_DIAG_DATA_TYPE_UINT);`
- 4.
5. /*Записать отсчет для комнатной температуры*/

⁵² Received Signal Strength Indicator, значение мощности сигнала, поступающего на антенны устройства.

```
6. uint32_t room_temp = get_room_temperature();
7. esp_diag_metrics_add_uint("temp1", &room_temp);
```

Прототип функции `esp_diag_metrics_register()` выглядит следующим образом:

```
1. esp_err_t esp_diag_metrics_register(const char *tag, const char *key,
2.
const char *label, const char *path,
3.
esp_diag_data_type_t type);
```

В прототипе функции `esp_diag_metrics_register()` тег параметра `tag` указывает метку показателя, которую могут определить пользователи. Ключ параметра `key` указывает уникальный идентификатор показателя, используемый для поиска и установки идентификатора показателя. Метка параметра `label` – это метка, отображаемая на панели мониторинга ESP Insights, параметр `path` указывает иерархический путь к ключу `key`, который должен разделяться знаком «.», например `wifi`, `heap.internal` или `heap.external`. Тип параметра `type` представляет тип данных, для которого поддерживаются следующие перечисляемые значения:

```
1. typedef enum {
2. ESP_DIAG_DATA_TYPE_BOOL, /*! < Data type boolean*/
3. ESP_DIAG_DATA_TYPE_BOOL, /*! < Data type boolean*/
4. ESP_DIAG_DATA_TYPE_UINT, /*! < Data type unsigned integer*/
5. ESP_DIAG_DATA_TYPE_FLOAT, /*! < Data type float*/
6. ESP_DIAG_DATA_TYPE_STR, /*! < Data type string*/
7. ESP_DIAG_DATA_TYPE_IPv4, /*! < Data type IPv4 address*/
8. ESP_DIAG_DATA_TYPE_MAC, /*! < Data type MAC address*/
9. } esp_diag_data_type_t;
```

4. Переменные

Переменные аналогичны показателям, но не нуждаются в отслеживании во времени, поскольку обычно они представляют информацию об устройстве (например, IP-адрес устройства). Чтобы включить поддержку переменных, вы можете установить `CONFIG_DIAG_ENABLE_VARIABLES=y`. Как и для показателей, поддерживается ряд predefined переменных, таких как IP и Wi-Fi. Кроме того, вы можете добавить свои собственные пользовательские переменные. Информация о переменных показана на рис. 15.6.

• Сетевые переменные

Как показано на рис. 15.6, ESP Insights в настоящее время поддерживает переменные Wi-Fi и IP. Для Wi-Fi поддерживаемые переменные включают BSSID, SSID, причину отключения Wi-Fi, текущий канал, режим аутентификации при подключении Wi-Fi и состояние подключения. Для IP поддерживаемые переменные включают адрес шлюза, адрес IPv4 и параметры маски сети.

• Пользовательские переменные

Разработчики могут добавлять пользовательские переменные с помощью следующих функций:

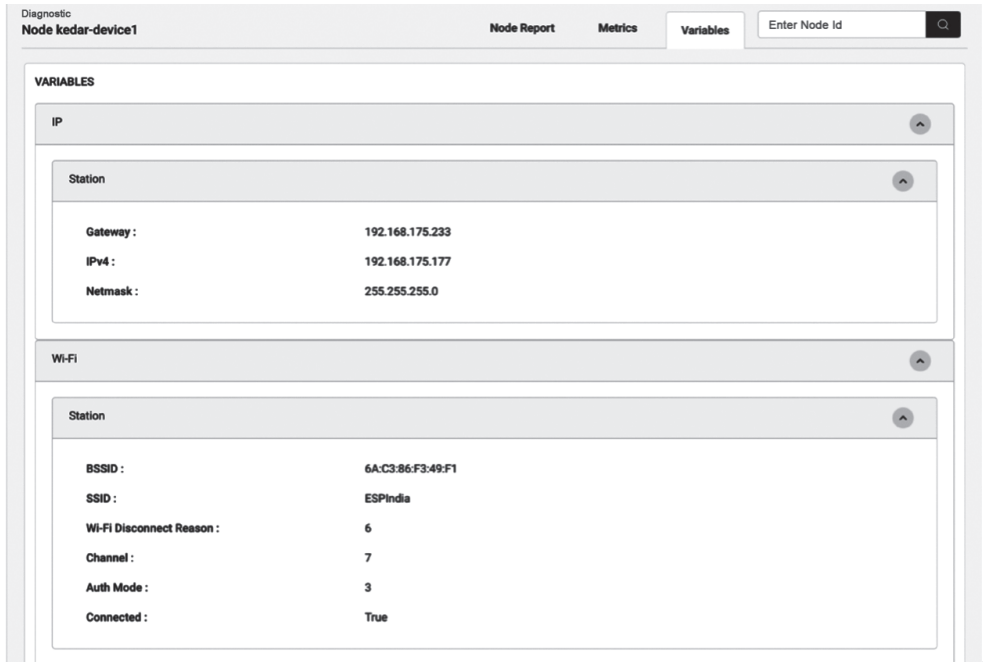


Рисунок 15.6. Информация о переменных

```

1. /*Register a variable to track stations associated with ESP32 AP*/
2. esp_diag_variable_register("wifi", "sta_cnt", "STAs associated",
3. "wifi.sta", ESP_DIAG_DATA_TYPE_UINT);
4.
5. /*Assuming WIFI_EVENT_AP_STACONNECTED and WIFI_EVENT_AP_STADISCONNECTED
6. events track the number of associated stations*/
7. esp_diag_variable_add_uint("sta_cnt", &sta_cnt);

```

Прототип функции `esp_diag_metrics_register()` выглядит таким образом:

```

1. esp_err_t esp_diag_variable_register(const char *tag, const char *key,
2.                                     const char *label, const char *path,
3.                                     esp_diag_data_type_t type);

```

Параметры функции `esp_diag_variable_register()` имеют то же значение, что и для функции `esp_diag_metrics_register()`.

15.3. Практика: использование ESP Insights в проекте Smart Light

В разделе 15.2 мы представили использование ESP Insights. В главе 9 мы рассказали, как реализовать удаленное управление устройствами через облачную IoT-платформу ESP RainMaker. В этом разделе добавим ESP Insights в проект Smart Light на основе разработки из раздела 9.4 для реализации отчета о диагностических данных.

```

1. #define APP_INSIGHTS_LOG_TYPE ESP_DIAG_LOG_TYPE_ERROR
2.                               | ESP_DIAG_LOG_TYPE_WARNING
3.                               | ESP_DIAG_LOG_TYPE_EVENT
4. esp_err_t app_insights_enable(void)
5. {
6.     esp_rmaker_mqtt_config_t mqtt_config = {
7.         .init = NULL,
8.         .connect = NULL,
9.         .disconnect = NULL,
10.        .publish = esp_rmaker_mqtt_publish,
11.        .subscribe = esp_rmaker_mqtt_subscribe,
12.        .unsubscribe = esp_rmaker_mqtt_unsubscribe,
13.    };
14.    esp_insights_mqtt_setup(mqtt_config);
15.
16.    esp_insights_config_t config = {
17.        .log_type = APP_INSIGHTS_LOG_TYPE,
18.    };
19.    esp_insights_enable(&config);
20.    return ESP_OK;
21. }
22.
23. void app_main()
24. {
25.     .....
26.     /*Enable Schedule*/
27.     esp_rmaker_schedule_enable();
28.
29.     /*Use Insights*/
30.     app_insights_enable();
31.
32.     /*Launch the ESP RainMaker IoT Cloud platform server*/
33.     esp_rmaker_start();
34.     .....
35. }

```

В приведенном коде показано, как использовать ESP Insights в примере ESP-RainMaker. Функция `esp_insights_mqtt_setup()` устанавливает интерфейс для предоставления диагностических данных. В этом случае ESP Insights использует тот же канал MQTT, что и ESP RainMaker IoT Cloud, что значительно экономит память. `APP_INSIGHTS_LOG_TYPE` определяет тип сообщений лога. Текущий пример поддерживает логи отчетов об ошибках, предупреждениях и событиях. По умолчанию агент Insights поддерживает загрузку логов сбоя устройств, поэтому пользователям нет необходимости специально настраивать этот тип. Пользователи могут включить следующие параметры в конфигурацию по умолчанию для записи затрат памяти, сигнала Wi-Fi и сетевых переменных устройства:

```

1. CONFIG_DIAG_ENABLE_METRICS=y
2. CONFIG_DIAG_ENABLE_HEAP_METRICS=y
3. CONFIG_DIAG_ENABLE_WIFI_METRICS=y
4. CONFIG_DIAG_ENABLE_VARIABLES=y
5. CONFIG_DIAG_ENABLE_NETWORK_VARIABLES=y

```


15.4. Резюме

В этой главе описывается ESP Insights, который включает в себя агент Insights, запускающийся на устройстве пользователя для регистрации рабочего состояния и неисправностей устройства и отправки сообщений о них в облако ESP Insights Cloud. При проверке функций продукта в реальных условиях пользователи могут войти в панель управления облачной платформы ESP RainMaker IoT Cloud для просмотра состояния каждого устройства и наличия отклонений. Вместо того чтобы перехватывать логи работы устройства при каждом запуске устройства, логи неисправностей будут передаваться в облако ESP Insights Cloud. Пользователи могут четко видеть причины неисправности устройства с помощью интерфейса ESP Insights Cloud, что значительно упрощает отладку.

В настоящее время агент Insights по умолчанию отправляет данные в облачную платформу ESP RainMaker IoT Cloud. В будущем Espressif выпустит решения для поддержки большего количества облачных платформ для получения и обработки информации об устройстве, сообщаемой агентом Insights. Благодаря этим решениям функциональная проверка и отладка устройства станут намного проще, что приведет к ускорению выпуска пользовательских продуктов.

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@aliens-kniga.ru.

ESP32-C3: Беспроводное приключение

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. 35,83. Тираж 100 экз.

Веб-сайт издательства: www.dmkpress.com

ESP32-C3 — это одноядерный микроконтроллер, представляющий собой «систему-на-кристалле» (SoC) с интегрированными Wi-Fi и Bluetooth 5 (LE), основанный на архитектуре RISC-V с открытым исходным кодом. Он обеспечивает необходимый баланс мощности, возможностей ввода-вывода и безопасности, предлагая таким образом оптимальное экономичное решение для подключаемых устройств.

Эта книга от компании Espressif демонстрирует различные приложения семейства ESP32-C3, начиная с основ разработки проектов интернета вещей (IoT) и настройки среды и заканчивая практическими примерами реализации готовых решений. Сначала рассказано об IoT, ESP RainMaker и ESP-IDF. Далее излагается проектирование оборудования и разработка драйверов. По мере чтения вы узнаете, как настроить свой проект через сети Wi-Fi и мобильные приложения. Наконец, вы научитесь оптимизировать свой проект и запускать его в серийное производство.

Примеры кода, использованные в этой книге, вы можете скачать с сайта Espressif на GitHub.

Если вы инженер-схемотехник, разработчик программного обеспечения, преподаватель, студент или просто радиолюбитель, интересующийся IoT, — это издание для вас.

Espressif Systems — публичная транснациональная компания по производству полупроводников, основанная в 2008 году, с офисами в Китае, Чехии, Индии, Сингапуре и Бразилии. Создатель популярных серий чипов, модулей и плат разработки ESP8266, ESP32, ESP32-S, ESP32-C и ESP32-H.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК “Галактика”
books@aliants-kniga.ru

 **ESPRESSIF**


ИЗДАТЕЛЬСТВО
www.dmk.pф

ISBN 978-5-93700-248-8



9 785937 002488 >