Муниципальное общеобразовательное бюджетное учреждение
«Средняя общеобразовательная школа «Центр образования «Кудрово»
Отделение дополнительного образования детей

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

ESP32-S3

Практический курс программирования микроконтроллеров



от первых сигналов до самостоятельных устройств

Авторы-составители
Корягин Сергей Андреевич, **Полякова Полина Алексеевна**
педагоги отделения дополнительного образования детей

Кудрово · 2026

Корягин С. А., Полякова П. А. ESP32-S3: практический курс программирования микроконтроллеров : учебно-методическое пособие / С. А. Корягин, П. А. Полякова. — Кудрово : МОБУ «СОШ «ЦО «Кудрово», ОДОД, 2026. — 296 с.

Пособие содержит вводные главы и сорок четыре урока (с нулевого по сорок третий) по программированию микроконтроллера ESP32-S3 в среде Arduino: от техники безопасности, цифровых выводов и последовательного порта до FreeRTOS, беспроводных сетей, защищённых обновлений, диагностики сбоев, основ HTML и CSS, датчиков, моторов, звука, USB, камеры, автономного питания, I^2S, RMT, PCNT/MCPWM, TWAI, графики, USB Host, JTAG, ULP, распознавания, ESPUI, настройки Wi-Fi через портал, сборки в PlatformIO, TinyML и итогового проекта. Каждый урок включает объяснение идеи, готовые примеры, самостоятельный опыт, разбор типичных ошибок, вопросы для самопроверки и шесть заданий четырёх уровней сложности. В приложениях — справочник, словарь, ответы на вопросы и примерное тематическое планирование для педагога.
Пособие адресовано обучающимся объединений технической направленности и педагогам дополнительного образования.

Примеры рассчитаны на работу с Arduino-ESP32 версии 3.x (на базе ESP-IDF 5.x) и отладочной платой ESP32-S3-DevKitC-1 (или совместимой).

© Корягин С. А., Полякова П. А., 2026

Содержание

Техника безопасности и рабочее место **10**

Макетная плата, схема и мультиметр **11**

Сигналы: аналоговые и цифровые **13**

Выводы платы: что есть что **20**

Перед первым уроком: сигнал, программа и безопасность **24**

0 Знакомство с ESP32-S3 и настройка среды **26**
0.1 Сначала — что именно мы делаем . . . 26
0.2 Что такое ESP32-S3 . . . 26
0.3 Сравнение с «родственниками» . . . 27
0.4 Модули и маркировка . . . 27
0.5 Отладочная плата ESP32-S3-DevKitC-1 . . . 28
0.6 Установка среды . . . 28
0.6.1 Если прошивка не начинается . . . 29
0.7 Структура Arduino-скетча . . . 29
0.8 Прочитайте пример как инструкцию . . . 30
0.9 Учимся программировать: как записывается программа . . . 30
0.10 Учимся программировать: переменные . . . 31
0.11 Как устроены уроки . . . 33

1 Цифровой выход: светодиод **34**
1.1 Идея урока коротко . . . 34
1.2 Зачем нужен резистор: закон Ома . . . 34
1.2.1 Почему светодиод сгорает без резистора . . . 35
1.2.2 Рассчитываем резистор . . . 35
1.2.3 Как узнать сопротивление резистора . . . 36
1.3 Как устроен выход GPIO . . . 36
1.4 Схема подключения . . . 37
1.5 Подключаем светодиод . . . 38
1.6 Классический Blink . . . 38
1.7 Разберём Blink по строкам . . . 39
1.8 Мигание без `delay()` . . . 39
1.9 Учимся программировать: условие `if` . . . 40
1.10 Практика: бегущий огонь . . . 41
1.11 Учимся программировать: массивы и цикл `for` . . . 42
1.12 Встроенный RGB-светодиод . . . 44
1.13 Самостоятельный опыт: внешний и встроенный светодиоды . . . 44

2 Последовательный порт: разговор с компьютером **46**
2.1 Кадр UART . . . 46
2.2 Serial и Serial Monitor . . . 46
2.3 Самостоятельный опыт: команды с компьютера . . . 47
2.4 Учимся программировать: строки и цепочка условий . . . 47
2.5 Практика: игра «Угадай число» . . . 48
2.6 Второй UART: связь с другими устройствами . . . 49

3 Цифровой вход: кнопки и прерывания 52
3.1 Сначала: что делает кнопка . . . 52
3.2 Проблема «висящего» входа . . . 52
3.3 Схема подключения . . . 52
3.4 Опрос кнопки . . . 53
3.5 Проверка до следующего шага . . . 54
3.6 Дребезг контактов . . . 54
3.7 Учимся программировать: логические операции и `static` . . . 55
3.8 Практика: кнопка-переключатель яркости . . . 55
3.9 Самостоятельный опыт: кнопка-переключатель . . . 56
3.10 Прерывания . . . 56
3.11 Strapping-выводы . . . 57

4 ШИМ (LEDC): плавная яркость 59
4.1 Сначала: откуда берётся промежуточная яркость . . . 59
4.2 Что такое ШИМ . . . 59
4.3 Периферия LEDC . . . 59
4.4 Что наблюдать в примере . . . 60
4.5 Яркость и восприятие: гамма-коррекция . . . 60
4.6 Самостоятельный опыт: яркость LED . . . 60
4.7 Учимся программировать: свои функции . . . 61
4.8 Какую частоту ШИМ выбрать . . . 62
4.9 Практика: плавная волна . . . 62
4.10 Другое применение ШИМ: сервопривод . . . 63
4.11 То же на ESPUI: ползунок яркости на телефоне . . . 64

5 АЦП: читаем аналоговые сигналы 67
5.1 Сначала: сигнал не всегда только 0 или 1 . . . 67
5.2 Как работает АЦП . . . 67
5.2.1 Особенности АЦП ESP32-S3 . . . 67
5.3 Потенциометр — «крутилка» . . . 68
5.4 Проверка чтения ручки . . . 69
5.5 Самостоятельный опыт: ручка яркости . . . 69
5.6 Измерение напряжения выше 3,3 В . . . 69
5.7 Практика: ручка поворачивает сервопривод . . . 70
5.8 То же на ESPUI: вольтметр со стрелкой и графиком . . . 72

6 Шина I^2C: дисплей 74
6.1 Топология шины . . . 74
6.2 Временная диаграмма . . . 75
6.3 Сканер шины . . . 75
6.4 Первый вывод на экран . . . 75
6.5 Самостоятельный опыт: экран состояния . . . 76
6.6 Практика: вольтметр с экраном . . . 77

7 FreeRTOS: задачи и синхронизация 81
7.1 Зачем нужна операционная система реального времени . . . 81
7.2 Первые задачи . . . 81
7.3 Общие данные: очередь и мьютекс . . . 82
7.4 Практика: мьютекс — «ключ» от общего ресурса . . . 83
7.5 Самостоятельная работа: устройство на задачах . . . 84

8 Wi-Fi и веб-сервер: HTTP, HTML и CSS 86
8.1 Что такое радиоволны . . . 86
8.2 Wi-Fi и его каналы . . . 87
8.3 Bluetooth . . . 88
8.4 Режимы Wi-Fi . . . 89
8.5 Подключение к сети . . . 89
8.5.1 Учимся программировать: цикл `while` и логическое «и» . . . 89
8.6 MAC-адрес и IP-адрес . . . 90
8.6.1 Кто раздаёт IP-адреса . . . 91
8.6.2 Как задать IP, MAC и имя в программе . . . 91
8.7 Имя вместо адреса: mDNS . . . 92
8.8 Клиент и сервер . . . 93
8.9 Из чего состоит адрес . . . 94
8.10 Методы GET и POST . . . 94
8.10.1 Пример: управление через GET . . . 95
8.10.2 Пример: отправка данных через POST . . . 97
8.11 HTML: из чего сделана страница . . . 98
8.12 CSS: делаем красиво . . . 99
8.13 Практика: панель управления светодиодом . . . 99
8.14 Самостоятельный опыт: кнопка и браузер вместе . . . 102
8.15 Если роутера нет: собственная точка доступа . . . 102

9 Сон и энергосбережение 104
9.1 Режимы питания . . . 104
9.2 Сколько проработает устройство от батареи . . . 105
9.3 Deep-sleep по таймеру . . . 105
9.4 Пробуждение кнопкой . . . 106
9.5 Самостоятельная работа: автономный датчик . . . 106
9.6 Практика: датчик на батарейке . . . 106

10 SPI: быстрая периферия и SD-карты 109
10.1 Чем SPI отличается от I^2C . . . 109
10.2 Одна шина, несколько устройств . . . 109
10.3 Первый журнал на SD-карте . . . 110
10.4 Разбор ошибки: «карта то работает, то нет» . . . 111

11 Память: RAM, Flash, PSRAM и NVS 112
11.1 У каждой памяти своя роль . . . 112
11.2 Настройка в NVS . . . 112
11.3 PSRAM и большие буферы . . . 113
11.4 Разбор ошибки: настройки «забываются» . . . 113

12 IoT: ESP-NOW, MQTT и Bluetooth LE 114
12.1 Три задачи — три протокола . . . 114
12.2 ESP-NOW: пакет по MAC-адресу . . . 114
12.3 MQTT: темы вместо прямых соединений . . . 115
12.4 BLE: сервисы и характеристики . . . 117
12.5 Разбор ошибки: сеть «подвисает» . . . 118

13 HTTPS и OTA: защищённое устройство 119
13.1 TLS не равен «порт 443» . . . 119
13.2 OTA: два слота вместо одной ставки . . . 120
13.3 Разбор ошибки: HTTPS или OTA не запускается . . . 121

14 Профессиональная диагностика: Guru Meditation 122
14.1 Паника — это полезный отчёт . . . 122
14.2 Как читать backtrace . . . 122
14.3 Разбор настоящего сбоя . . . 122
14.4 Watchdog: защита от «всё зависло» . . . 123
14.5 Питание — тоже часть отладки . . . 123
14.6 Минимальный журнал здоровья . . . 124

15 Датчики: температура, влажность, расстояние 125
15.1 Сначала: датчик — это собеседник со своими правилами . . . 125
15.2 Как читать datasheet за пять минут . . . 125
15.3 DS18B20: температура по шине 1-Wire . . . 125
15.4 DHT22: температура и влажность . . . 127
15.5 HC-SR04: расстояние по времени эха . . . 128
15.6 Самостоятельный опыт: парковочный радар . . . 130
15.7 То же на ESPUI: дальномер на экране телефона . . . 130

16 Силовая нагрузка: транзистор, реле, мотор 132
16.1 Сначала: вывод платы — это «кнопка», а не «розетка» . . . 132
16.2 Ключ на транзисторе . . . 132
16.3 Индуктивная нагрузка и защитный диод . . . 134
16.4 Модуль реле . . . 134
16.5 Мотор постоянного тока и драйвер . . . 135
16.6 Питание: раздельное, но с общей землёй . . . 137
16.7 Самостоятельный опыт: вентилятор с регулятором . . . 137
16.8 То же на ESPUI: пульт для мотора . . . 138

17 Адресные светодиоды WS2812 140
17.1 Сначала: цепочка светодиодов, которые слушают по очереди . . . 140
17.2 Протокол: нули и единицы разной длины . . . 140
17.3 Почему не `digitalWrite()` . . . 141
17.4 Питание: считаем ток . . . 141
17.5 Цвет через HSV . . . 142
17.6 Встроенный светодиод как лента из одного пикселя . . . 142
17.7 Эффекты без `delay()` . . . 143
17.8 Самостоятельный опыт: шкала уровня . . . 144
17.9 То же на ESPUI: выбираем цвет ползунком . . . 144

18 Плата как клиент: HTTP, JSON, время и файлы 147
18.1 Сначала: плата ходит в интернет, как браузер . . . 147
18.2 Путь запроса . . . 147
18.3 HTTP-запрос: GET . . . 147
18.4 JSON и ArduinoJson . . . 148
18.5 Точное время по NTP . . . 149
18.6 LittleFS: файлы во Flash . . . 150
18.7 Самостоятельный опыт: сетевые часы . . . 151

19 Нативный USB: плата как клавиатура и мышь 152
19.1 Сначала: компьютер не знает, что вы ему подключили . . . 152
19.2 Два разъёма — два разных пути . . . 153
19.3 Настройка Arduino IDE . . . 153
19.4 Клавиатура по кнопке . . . 153
19.5 Громкость потенциометром . . . 155

19.6 Мышь . . . 156
19.7 Самостоятельный опыт: пульт для презентаций . . . 157

20 ESP32-S3 CAM: камера 158
20.1 Сначала: камера, которая сама сжимает картинку . . . 158
20.2 Что такое плата ESP32-S3-CAM . . . 158
20.3 Сенсоры OV2640 и OV5640 . . . 158
20.4 Интерфейс DVP и путь кадра . . . 159
20.5 Распиновка плат типа ESP32-S3-EYE . . . 160
20.6 Настройки Arduino IDE . . . 160
20.7 Пример 1: запуск камеры . . . 161
20.8 Параметры качества и буферы кадра . . . 162
20.9 Пример 2: снимок и видеопоток в браузере . . . 163
20.10 Пример 3: снимок на microSD . . . 165
20.11 Питание, нагрев, антенна . . . 166
20.12 Типичные ошибки . . . 167
20.13 Что дальше . . . 167
20.14 Самостоятельный опыт: фотоловушка . . . 167

21 Таймеры и конечные автоматы 168
21.1 Сначала: будильник и расписание . . . 168
21.2 Аппаратный таймер . . . 169
21.3 Конечный автомат . . . 170
21.4 Самостоятельный опыт: ночной режим . . . 172

22 Биты, байты и регистры 173
22.1 Сначала: восемь выключателей в одном числе . . . 173
22.2 Двоичная и шестнадцатеричная запись . . . 173
22.3 Битовые операции . . . 173
22.4 Регистры: память, которая управляет железом . . . 174
22.5 Сдвиговый регистр 74HC595 . . . 175
22.6 Самостоятельный опыт: счётчик нажатий . . . 177

23 Органы управления: энкодер, клавиатура, сенсорные кнопки 178
23.1 Сначала: как человек разговаривает с устройством . . . 178
23.2 Энкодер . . . 178
23.3 Матричная клавиатура . . . 180
23.4 Сенсорные входы . . . 181
23.5 Самостоятельный опыт: меню на энкодере . . . 182

24 Звук: пищалка и мелодии 184
24.1 Сначала: звук — это быстрое мигание . . . 184
24.2 Частота и нота . . . 184
24.3 Какая бывает пищалка . . . 184
24.4 Функция `tone()` . . . 185
24.5 Мелодия без остановки программы . . . 185
24.6 Что дальше: настоящий звук . . . 186
24.7 Самостоятельный опыт: музыкальная клавиатура . . . 186
24.8 То же на ESPUI: ноты в списке на телефоне . . . 186

25 Шаговый двигатель **189**
25.1 Сначала: мотор, который умеет считать . . . 189
25.2 Как он устроен . . . 189
25.3 Подключение . . . 190
25.4 Программа . . . 190
25.5 Самостоятельный опыт: стрелочный указатель . . . 191
25.6 То же на ESPUI: поворот на заданный угол с телефона . . . 191

26 Акселерометр и гироскоп: датчик движения **194**
26.1 Сначала: два датчика, которые ошибаются по-разному . . . 194
26.2 Что измеряет датчик . . . 194
26.3 Разговор с датчиком через регистры . . . 194
26.4 Комплементарный фильтр . . . 195
26.5 Самостоятельный опыт: электронный уровень . . . 197
26.6 То же на ESPUI: график наклона . . . 197

27 Автономное питание: батареи и аккумуляторы **200**
27.1 Сначала: откуда брать энергию . . . 200
27.2 Источники питания . . . 200
27.3 Цепочка питания . . . 201
27.4 Сколько осталось заряда . . . 201
27.5 Сколько проработает устройство . . . 203
27.6 Самостоятельный опыт: индикатор заряда . . . 203

28 Порядок в коде и инструменты разработчика **204**
28.1 Сначала: программа растёт быстрее, чем кажется . . . 204
28.2 От функций к классам . . . 204
28.3 Несколько файлов . . . 205
28.4 Git: машина времени для проекта . . . 206
28.5 Что ещё пригодится . . . 207
28.6 Самостоятельный опыт: наведите порядок . . . 207

29 Цифровой звук: I²S и микрофон **208**
29.1 Звук — поток чисел . . . 208
29.2 Подключение и первый поток . . . 209
29.3 Что именно измеряет пример . . . 210
29.4 От измерения к воспроизведению . . . 210

30 Точные импульсы: периферия RMT **212**
30.1 Когда программная задержка мешает . . . 212
30.2 Пакет, который можно проверить . . . 212
30.3 ИК: несущая, огибающая и команда . . . 213
30.4 Асинхронность и память . . . 213

31 Счётчики и движение: PCNT и MCPWM **215**
31.1 Периферия считает вместо ISR . . . 215
31.2 Частотомер на одной плате . . . 215
31.3 MCPWM и форма управляющего сигнала . . . 216
31.4 Первый контур обратной связи . . . 217

32 Проводная сеть: TWAI/CAN и RS-485 **218**
32.1 Контроллер, трансивер и протокол . . . 218
32.2 Два узла CAN . . . 219
32.3 Два узла RS-485 . . . 220

33 Цветной экран: SPI, RGB, память и DMA **223**
33.1 Цена одного кадра . . . 223
33.2 Подключение и библиотеки . . . 223
33.3 DMA, PSRAM и время жизни кадра . . . 224
33.4 От рисунков к LVGL . . . 224

34 USB глубже: CDC, накопитель и роль Host **226**
34.1 Класс и роль — разные понятия . . . 226
34.2 Своя консоль TinyUSB CDC . . . 226
34.3 MSC: кто владеет файловой системой . . . 227
34.4 Host: переходника мало . . . 227

35 ESP-IDF и JTAG: увидеть выполнение **228**
35.1 Когда печати недостаточно . . . 228
35.2 Минимальная сборка ESP-IDF . . . 228
35.3 Ошибка с понятным ожиданием . . . 229
35.4 Подключение встроенного USB Serial/JTAG . . . 229

36 ULP: наблюдение, пока основные ядра спят **230**
36.1 Два домена исполнения . . . 230
36.2 Проект с двумя образами . . . 231
36.3 Что делает основная программа . . . 231
36.4 Программа сопроцессора . . . 232
36.5 Проверка экономии . . . 232

37 Распознавание на устройстве: данные, модель и ESP-DL **233**
37.1 Что даёт именно ESP32-S3 . . . 233
37.2 Обучение на своих измерениях . . . 233
37.3 Как проверить честно . . . 234
37.4 Переход к IMU, камере и ESP-DL . . . 234

38 От макетки к устройству: запуск, восстановление и защита **236**
38.1 Запуск без компьютера . . . 236
38.2 Flash и OTA планируют вместе . . . 236
38.3 Provisioning: как выдать учётные данные . . . 237
38.4 Автомат сети и повтор команд . . . 237
38.5 Механизмы защиты решают разные задачи . . . 237
38.6 Приёмочные испытания . . . 237

39 ESPUI: веб-панель из программы C++ **239**
39.1 Зачем библиотека, если уже знаем HTML . . . 239
39.2 Установка: проверяем комплект зависимостей . . . 239
39.3 Первая страница без роутера и файловой системы . . . 239
39.4 Элемент, значение и событие . . . 240
39.5 Каталог виджетов . . . 241
39.6 Как выглядит панель . . . 241
39.7 Gauge и Graph: вольтметр на телефоне . . . 242
39.8 Ввод: числа, текст, списки и крестовина . . . 243

39.9 Одна модель состояния для кнопок и браузера . . . 245
39.10 Полная панель: LED, вкладки и телеметрия . . . 245
39.11 Что проверять после запуска . . . 247
39.12 Оформление, файлы и доступ . . . 247
39.13 Те же панели в других уроках . . . 248
39.14 Какие ещё средства стоит знать . . . 248

40 WiFiManager: настройка сети без перепрошивки 250
40.1 Панель управления и портал настройки . . . 250
40.2 Когда открывать доступ . . . 250
40.3 Неблокирующий режим и явное окно времени . . . 250
40.4 Как совместить с ESPUI . . . 252
40.5 План проверки владельцем устройства . . . 252

41 PlatformIO: профессиональная сборка проекта 253
41.1 Сначала: чего не хватает Arduino IDE . . . 253
41.2 Установка . . . 254
41.3 Из чего состоит проект . . . 254
41.4 Файл platformio.ini: разбор по строкам . . . 255
41.5 Первый проект . . . 256
41.6 Команды . . . 257
41.7 Свои библиотеки и папка include . . . 257
41.8 Тесты без платы . . . 257
41.9 Отладка и работа по сети . . . 258
41.10 PlatformIO в автоматической проверке . . . 258
41.11 Самостоятельный опыт: проект с тестом . . . 259

42 TinyML: нейросеть внутри микроконтроллера 261
42.1 Сначала: что такое TinyML . . . 261
42.2 Когда нейросеть не нужна . . . 261
42.3 Рабочий цикл TinyML . . . 262
42.4 Нейрон и слой . . . 262
42.5 Как сеть учится . . . 262
42.6 Квантование: как уместить модель . . . 263
42.7 Опыт 1: сеть повторяет синус . . . 263
42.8 Опыт 2: распознавание жестов по акселерометру . . . 266
42.9 Профессиональные инструменты . . . 271
42.10 Что измерять и как не обмануться . . . 272

43 Итоговый проект: от идеи до защиты 274
43.1 Сначала: проект — это история с началом и концом . . . 274
43.2 Как выбрать тему . . . 274
43.3 Проекты на возможностях ESP32-S3 . . . 275
43.4 Техническое задание на полстраницы . . . 275
43.5 Блок-схема устройства . . . 275
43.6 План по неделям . . . 276
43.7 Чек-лист готовности . . . 276
43.8 Защита за пять минут . . . 276
43.9 Критерии оценки . . . 277

A Справочник 279

B Компоненты для курса 281

C Словарь терминов **282**

D Шпаргалка по C++ для Arduino **285**

E Ответы на вопросы «Проверь себя» **288**

F Материалы для педагога **294**

Техника безопасности и рабочее место

Опыты этого курса идут на безопасном напряжении $3{,}3$ – 5 V. Ударить током оно не может, но испортить плату, обжечь палец о перегретую деталь или повредить аккумулятор можно за секунду. Правила ниже короткие. Прочитайте их до первой сборки и возвращайтесь к ним перед каждым новым уроком.

Десять правил

1. **Сначала отключите питание, потом меняйте схему.** Выньте USB-кабель, пересоберите, проверьте по рисунку и только после этого включайте.
2. **На выводы GPIO — не больше** $3{,}3$ **V.** Контакт 5V — это питание, а не сигнал.
3. **Не замыкайте 3V3 и 5V на GND.** Оголённые концы проводов и металлические предметы на столе — частая причина короткого замыкания.
4. **Деталь греется или пахнет — сразу отключите питание.** Потом ищите ошибку: чаще всего перепутаны плюс и минус.
5. **Светодиод подключайте через резистор, а мотор и реле — через ключ или драйвер** (уроки 1 и 16).
6. **Сетевое напряжение** 220 V **в учебных опытах не используется.** Никогда не подключайте к макетной плате провода от розетки и не разбирайте блоки питания.
7. **Литиевые аккумуляторы требуют особой осторожности:** не замыкать, не прокалывать, не заряжать без платы защиты и без присмотра (урок 27). Вздувшийся аккумулятор отдайте педагогу.
8. **Паяльник — только с разрешения педагога.** Жало нагрето выше 300 °C: кладите паяльник только на подставку, работайте в проветриваемом помещении, после пайки мойте руки.
9. **Берегите глаза.** Откусывая выводы деталей, придерживайте их и направляйте вниз, от себя и от соседей. Не смотрите на яркие светодиоды в упор.
10. **Порядок на столе.** Никакой еды и напитков рядом с электроникой; в конце занятия отключите питание и разложите детали по ячейкам.

Осторожно

Статическое электричество способно повредить микросхему незаметно для вас. Перед работой коснитесь заземлённого металлического предмета, не трогайте контакты модуля камеры и чипа пальцами и держите платы за края.

Если что-то пошло не так

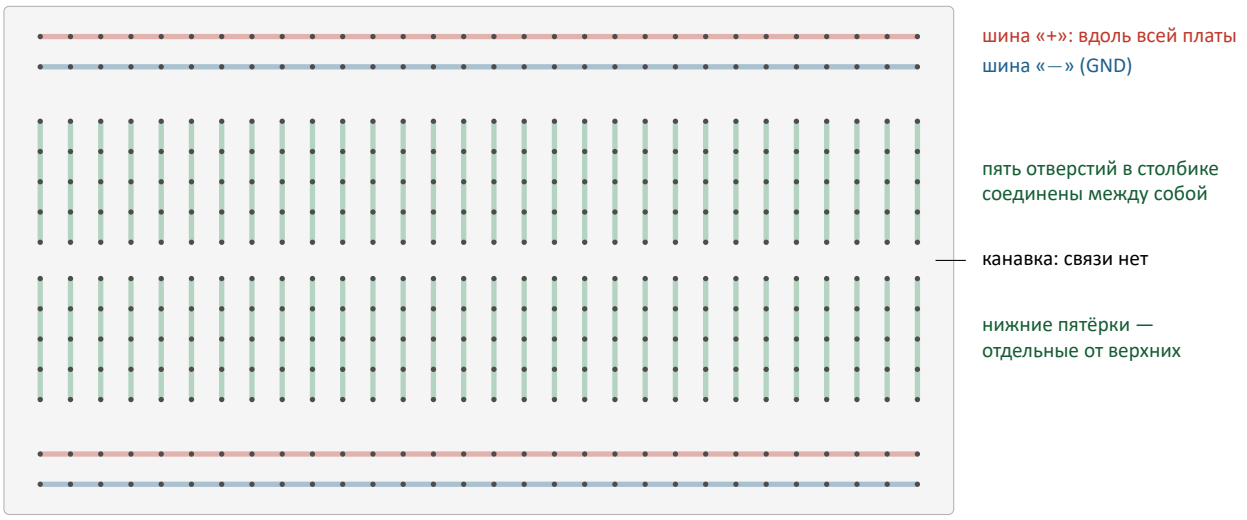
- **Дым, искры, запах гари** — отключите питание, отойдите, позовите педагога. Не пытайтесь потушить или разобрать устройство самостоятельно.
- **Ожог** — подержите обожжённое место под прохладной водой и сообщите педагогу.
- **Плата перестала определяться компьютером** — отключите всё от её выводов и проверьте кабель; чаще всего виновато короткое замыкание в схеме, а не «сгоревшая» плата.

Макетная плата, схема и мультиметр

Три инструмента понадобятся в каждом уроке: макетная плата, на которой собирают схему без пайки, принципиальная схема, по которой её собирают, и мультиметр, которым проверяют результат.

Как устроена макетная плата

Под отверстиями макетной платы спрятаны металлические полоски. Они соединяют отверстия группами, и это нужно запомнить один раз.



- **Шины питания** по краям соединены *вдоль* платы. К красной подключают 3V3, к синей — GND.
- **Рабочее поле** соединено *поперёк*, столбиками по пять отверстий. Если вставить две ножки детали в один столбик, они окажутся замкнуты.
- **Канавка** посередине разделяет верхние и нижние пятёрки. Плату ESP32-S3 и микросхемы ставят над канавкой, чтобы ножки левого и правого ряда не соединялись.

Осторожно

На многих длинных макетных платах шины питания разорваны посередине: левая и правая половины не соединены. Если половина схемы «не получает питание», поставьте перемычки между половинами шин или проверьте их мультиметром в режиме прозвонки.

Как читать схему

Принципиальная схема показывает не внешний вид деталей, а их соединения. Линия — это провод, точка на пересечении линий — соединение. Чаще всего в книге встречаются такие обозначения:



Все значки «земля» на схеме соединены между собой, даже если линия между ними не нарисована. То же относится к одинаково подписанным питаниям (3,3 В). У светодиода треугольник показывает направление тока: от анода (длинная ножка) к катоду.

Номинал резистора можно прочитать по цветным полоскам: первые две — цифры, третья — число нулей.

чёрный — 0	коричневый — 1	красный — 2	оранжевый — 3	жёлтый — 4
зелёный — 5	синий — 6	фиолетовый — 7	серый — 8	белый — 9

Например, «красный, красный, коричневый» — это 2, 2 и один ноль: $220\,\Omega$. «Коричневый, чёрный, оранжевый» — 1, 0 и три нуля: $10\,\text{k}\Omega$. Золотая или серебряная полоска в конце показывает точность. Если сомневаетесь, измерьте резистор мультиметром.

Мультиметр: четыре измерения

Что измеряем	Режим	Как подключать щупы
Напряжение	V с прямой чертой (DC), предел 20 В	*параллельно*: чёрный щуп на GND, красный — в точку схемы; питание включено
Целостность цепи	прозвонка (значок звука или диода)	к двум концам провода или дорожки; **питание выключено**
Сопротивление	Ω	к выводам детали, вынутой из схемы; **питание выключено**
Ток	mA или A (DC)	*последовательно*, в разрыв цепи; красный щуп иногда переставляют в другое гнездо

Осторожно

В режиме измерения тока мультиметр ведёт себя как кусок провода. Если в этом режиме коснуться щупами плюса и минуса источника, получится короткое замыкание. После измерения тока сразу возвращайте переключатель и красный щуп в положение для напряжения.

Первое, что делают при любой неполадке, — измеряют питание: между 3V3 и GND должно быть около $3{,}3\,\text{V}$, между 5V и GND — около $5\,\text{V}$. Если напряжения нет, искать ошибку в программе бесполезно.

Совет

Нет платы или деталей под рукой? Многие опыты первых уроков можно собрать в браузерном симуляторе Wokwi (`https://wokwi.com`): там есть ESP32-S3, светодиоды, кнопки, дисплеи и Serial Monitor. Симулятор не заменит настоящую сборку, но помогает проверить программу и схему до занятия.

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Сформулируйте вслух три правила безопасности, которые нарушать нельзя, и объясните соседу, почему.
2. ★ Найдите на макетной плате ряды питания и покажите, какие отверстия в одном столбце соединены между собой.
3. ★★ Определите по полоскам номиналы трёх резисторов и проверьте их мультиметром. Совпали ли значения? Почему допустимо небольшое расхождение?
4. ★★ Измерьте напряжения 3V3–GND и 5V–GND при питании от USB и запишите результаты: они пригодятся при поиске неисправностей.
5. ★★★ Соберите на макетной плате цепь «3V3 — резистор — светодиод — GND», нарисуй-

те её схему и объясните, в какую сторону включён светодиод.

6. ★★★★ Мультиметром в режиме прозвонки найдите, не разорваны ли шины питания посередине вашей платы, и составьте карту: какие участки соединены, а какие нет.

Сигналы: аналоговые и цифровые

Цель	понять, что такое сигнал, чем аналоговый сигнал отличается от цифрового и как один превращается в другой
Опирается на	техника безопасности и рабочее место; предварительных знаний не требуется
Результат	учимся читать графики сигналов и переводить числа в двоичную систему
Понадобится	карандаш, бумага и калькулятор — паяльник сегодня не нужен

Посмотрите на плату: у её выводов разные подписи: одни умеют работать только с «включено/выключено», другие — ещё и измерять напряжение (ADC). Чтобы понять, в чём разница, разберёмся, *с чем* вообще работает микроконтроллер. Он не видит, не слышит и не чувствует — он умеет только измерять **напряжение** на своих ножках и выдавать напряжение сам. Всё, что происходит вокруг — свет, звук, нажатие кнопки, — должно превратиться в напряжение, то есть в **электрический сигнал**.

Что такое сигнал

Сигнал — это величина, которая меняется и этим что-то сообщает. Светофор сигналит цветом, учитель — голосом, градусник — высотой столбика. В электронике сигнал — это почти всегда напряжение, которое меняется со временем.
Сигналы удобно рисовать графиками. По горизонтали откладываем **время** t, по вертикали — **напряжение** U. Каждая точка линии отвечает на вопрос: «какое напряжение было в этот момент?»

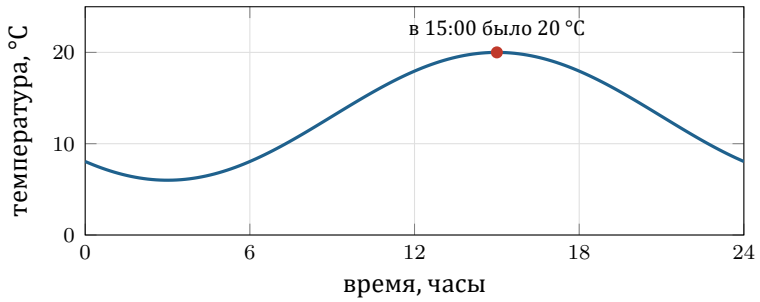


Рис. С.1: Температура на улице в течение суток. Это тоже сигнал: величина, которая меняется со временем

Аналоговый сигнал

Аналоговый сигнал меняется *плавно* и может принимать *любое* значение: не только 1 или 2 вольта, но и 1,5, и 1,537, и 1,53712… Как температура: она не прыгает с 15 °C сразу на 16 °C, а проходит все промежуточные значения.

Представь горку и лестницу. По горке можно съехать и остановиться на любой высоте — это аналоговый сигнал. По лестнице можно стоять только на ступеньках — это цифровой сигнал, о нём чуть ниже.

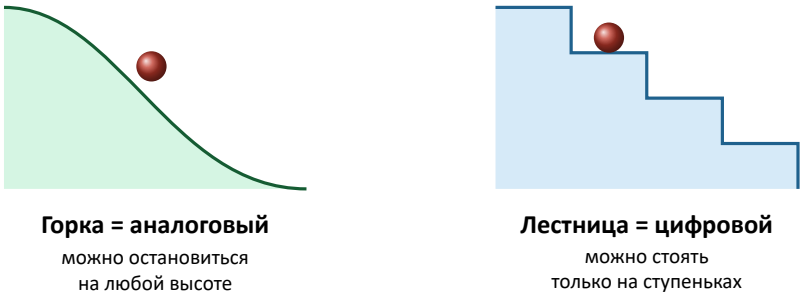


Рис. С.2: Аналоговый и цифровой сигнал — как горка и лестница

Самый известный аналоговый сигнал — **синусоида**. Такой сигнал выдаёт микрофон, когда рядом звучит чистая нота. У неё есть два главных параметра:

- **амплитуда** U_m — насколько высоко поднимается волна (у звука — громкость);
- **период** T — через сколько времени волна повторяется. Сколько раз волна повторяется за секунду, называют **частотой**: $f = 1/T$. Частоту измеряют в герцах (Гц): 1 Гц = одно повторение в секунду.

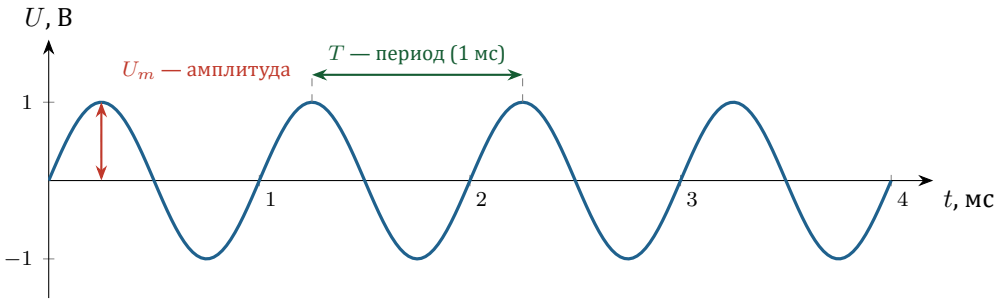


Рис. С.3: Синусоида. Период 1 мс значит, что за секунду волна повторяется 1000 раз: частота $f = 1000$ Гц = 1 кГц. Такой звук похож на писк старого телефона

Это интересно

Человек слышит звуки с частотой примерно от 20 Гц (очень низкий гул) до 20 000 Гц (тонкий писк). С возрастом верхняя граница снижается, поэтому некоторые высокие звуки слышат только дети и подростки.

Цифровой сигнал

Цифровой сигнал может принимать только несколько заранее договорённых значений. Чаще всего значений всего два:

- **0** — «нет», низкий уровень, LOW, напряжение около 0 В;
- **1** — «да», высокий уровень, HIGH, для ESP32-S3 это около 3,3 В.

Одна такая «ячейка» со значением 0 или 1 называется **бит**. Выключатель света — цифровой: свет либо горит, либо нет. А регулятор громкости у колонки — аналоговый.
Чтобы передать много информации, биты посылают по очереди, один за другим. Каждому биту отводится одинаковое время — *длительность бита*:

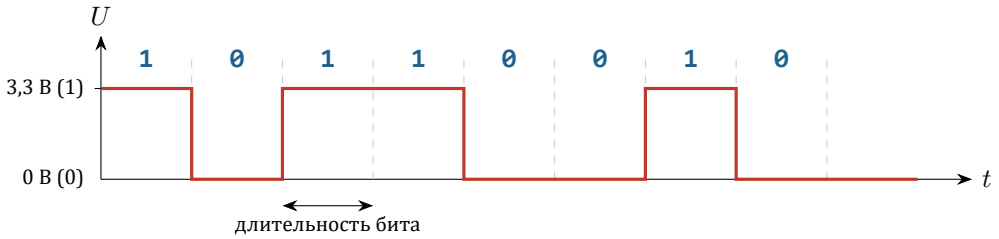


Рис. С.4: Цифровой сигнал: восемь бит `10110010` передаются по одному проводу друг за другом

Аналоговые источники сигнала	Цифровые источники сигнала
микрофон (звук)	кнопка (нажата / не нажата)
фоторезистор (освещённость)	датчик движения (есть движение / нет)
термистор (температура)	геркон, концевой выключатель
потенциометр — «крутилка» (урок 5)	линия связи с компьютером (урок 2)

Таблица С.1: Примеры аналоговых и цифровых сигналов, с которыми мы встретимся

Логические уровни: где кончается ноль и начинается единица

Настоящее напряжение никогда не бывает ровно 0 или ровно 3,3 В: провода длинные, детали неидеальные. Поэтому микросхема договаривается: всё, что *достаточно низко*, — это 0, всё, что *достаточно высоко*, — это 1. Посередине остаётся «серая зона», где микросхема может ошибиться. Хорошая схема в эту зону никогда не попадает.



Рис. С.5: Как ESP32-S3 (питание 3,3 В) понимает напряжение на входе. Граница нуля — четверть напряжения питания, граница единицы — три четверти

Почему цифровой сигнал надёжнее

По дороге к приёмнику любой сигнал обрастает **помехами**: рядом работает телефон, мигает лампа, моторчик создаёт искры. Помехи — это маленькие случайные добавки к напряжению. Аналоговому сигналу от них не спастись: приёмник не знает, какая часть напряжения — полезный сигнал, а какая — помеха. А у цифрового сигнала всего два значения, и достаточно спросить: «напряжение выше середины или ниже?» Если помеха не слишком большая, ответ будет правильным, и сигнал восстановится *в точности*.

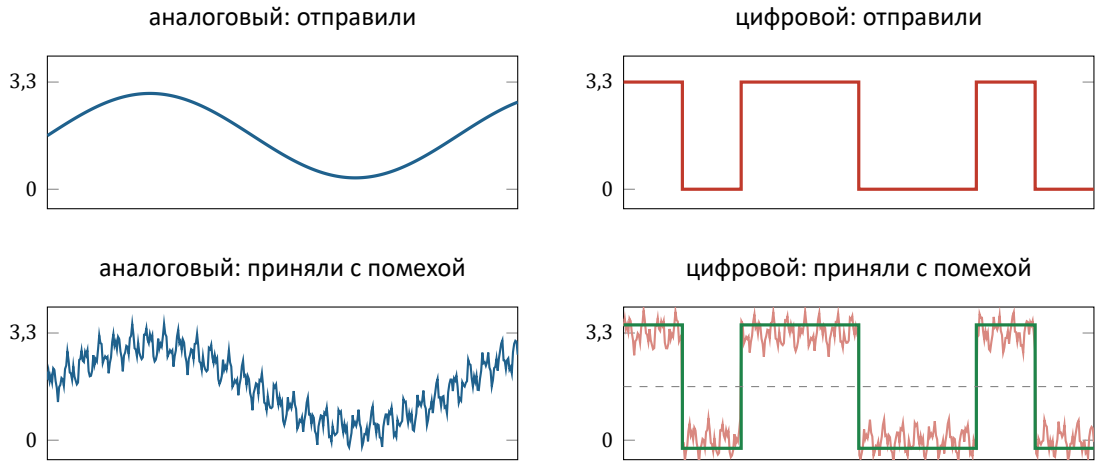


Рис. С.6: Одна и та же помеха портит аналоговый сигнал навсегда, а цифровой легко восстановить: всё, что выше пунктирной линии, — единица, ниже — ноль (зелёная линия)

Именно поэтому музыку, фотографии и фильмы сегодня хранят и передают в цифровом виде: копия цифрового файла ничем не отличается от оригинала, сколько бы раз его ни копировали.

Двоичные числа

Один бит хранит всего два варианта: 0 или 1. Два бита — уже четыре варианта (`00`, `01`, `10`, `11`), три бита — восемь. Каждый новый бит удваивает количество вариантов:

$$\text{количество вариантов} = 2^N, \quad N \text{ — число бит.}$$

Восемь бит называют **байтом**. Байт хранит числа от 0 до 255. Чтобы перевести двоичное число в обычное, подпишем над каждым битом его «вес» — степень двойки — и сложим веса тех битов, где стоит единица:

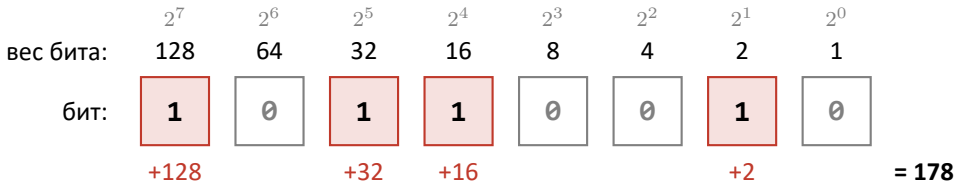


Рис. С.7: Двоичное число `10110010` равно $128 + 32 + 16 + 2 = 178$

Число бит	Вариантов	Диапазон	Число бит	Вариантов	Диапазон
1	2	0–1	8 (байт)	256	0–255
2	4	0–3	10	1024	0–1023
3	8	0–7	12	4096	0–4095
4	16	0–15	16	65 536	0–65 535

Таблица С.2: Сколько разных чисел помещается в N бит. Эти числа ещё не раз встретятся нам в курсе

Из аналогового в цифровой: АЦП

Процессор умеет работать только с числами. Чтобы он «услышал» микрофон или «узнал» положение ручки, аналоговый сигнал нужно превратить в числа. Этим занимается **АЦП** — *аналого-цифровой преобразователь*. Он делает два шага.

Шаг 1. Дискретизация — измеряем сигнал не всё время, а через одинаковые промежутки времени. Так работает кино: плавное движение снимают отдельными кадрами, 24 кадра в секунду, и глаз видит непрерывное движение. Сколько измерений делается за секунду, называют **частотой дискретизации**.
Шаг 2. Квантование — каждое измерение округляем до ближайшей «ступеньки». Это как измерять рост линейкой, на которой есть только сантиметровые деления: 152,7 см запишем как 153 см. Число ступенек зависит от числа бит АЦП: 3 бита — 8 ступенек, 12 бит — 4096 ступенек.

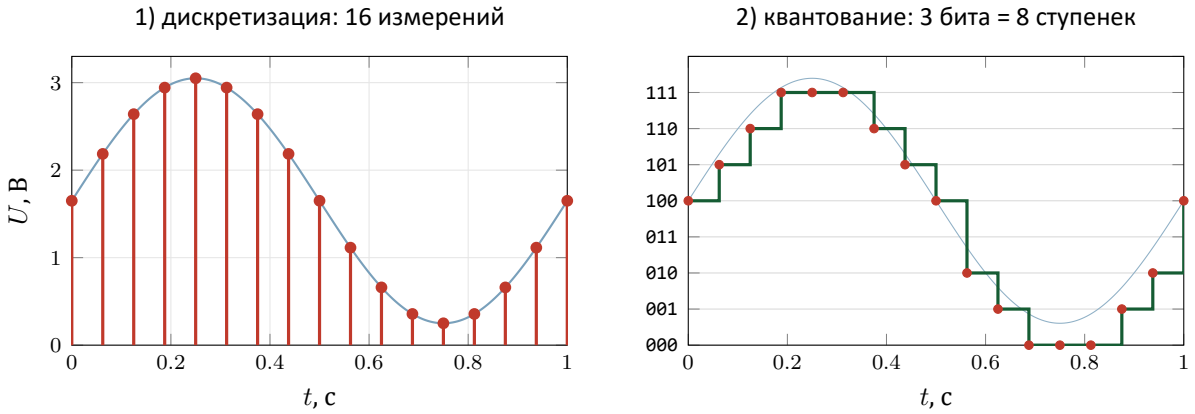


Рис. С.8: Как АЦП превращает плавную волну в числа. Слева: измеряем 16 раз за период. Справа: каждое измерение округляем до одной из 8 ступенек и записываем номер ступеньки двоичным числом

Высота одной ступеньки называется **шагом квантования**:

$$q = \frac{U_{\max}}{2^N}, \qquad \text{для 12-битного АЦП и 3,3 В:} \quad q = \frac{3{,}3\ \text{В}}{4096} \approx 0{,}8\ \text{мВ}.$$

Чем больше бит, тем мельче ступеньки и тем точнее цифровая копия. В ESP32-S3 стоит 12-битный АЦП — с ним мы поработаем в уроке 5.

Как часто нужно измерять?

Если измерять слишком редко, можно получить совсем не тот сигнал. Посмотри на рисунок: быстрая волна (9 колебаний в секунду) измерена всего 10 раз в секунду, и точки измерений легли на медленную волну, которой на самом деле нет!

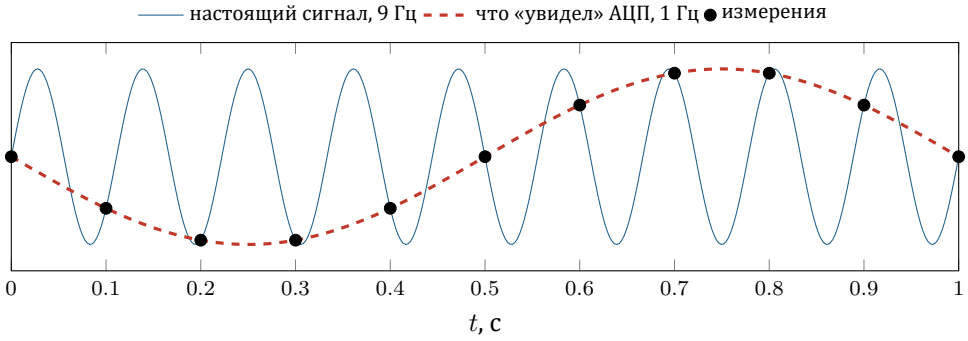


Рис. С.9: Слишком редкие измерения обманывают. Поэтому в кино колёса машин иногда кажутся крутящимися медленно или даже назад

Правило (его называют **теоремой Котельникова**): измерять нужно *больше чем в два раза чаще*, чем самая высокая частота в сигнале. Человек слышит до 20 000 Гц, поэтому музыку записывают с частотой дискретизации 44 100 измерений в секунду.

Из цифрового в аналоговый: ЦАП и ШИМ

Обратная задача — из чисел получить плавное напряжение, например чтобы динамик заиграл музыку. Это делает **ЦАП** — цифро-аналоговый преобразователь.
В ESP32-S3 ЦАПа нет, но есть хитрый приём — **ШИМ** (широтно-импульсная модуляция). Вывод очень быстро включается и выключается, тысячи раз в секунду. Если он половину времени включён, а половину выключен, то «в среднем» на нём половина напряжения. Светодиод при этом светит вполовину яркости: глаз не успевает заметить мигание. Подробно ШИМ разберём в уроке 4.

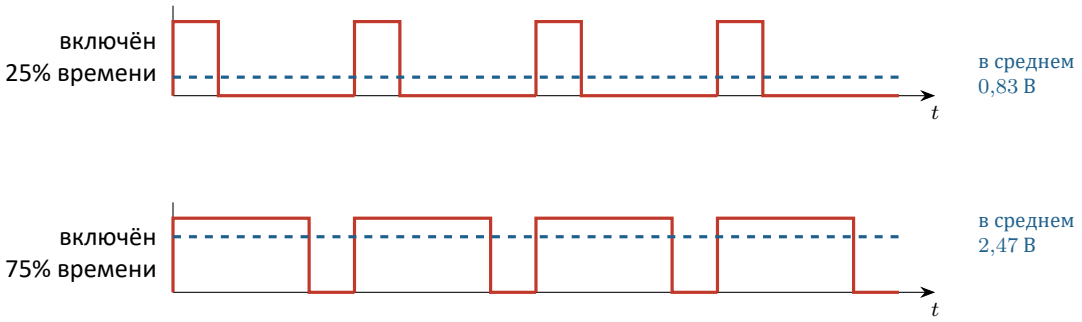


Рис. С.10: ШИМ: чем дольше вывод включён, тем больше среднее напряжение (пунктир)

Путь сигнала внутри устройства

Почти любое «умное» устройство устроено одинаково: *аналоговый мир* → *цифровая обработка* → *аналоговый мир*. Вот, например, как работает цифровой диктофон:

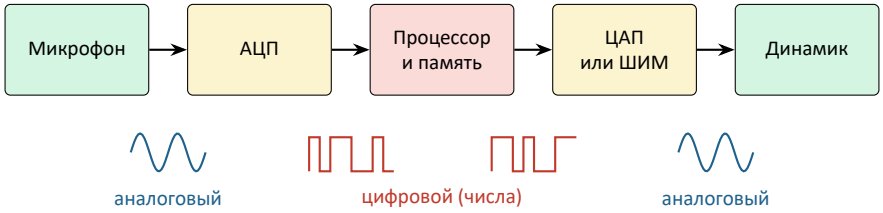


Рис. С.11: Путь звука в цифровом диктофоне

В нашем курсе ESP32-S3 будет работать с сигналами обоих видов:

Что делаем	Какой сигнал	Урок
Включаем и выключаем светодиод	цифровой выход (0 или 1)	1
Разговариваем с компьютером	цифровой, биты друг за другом	2
Читаем кнопку	цифровой вход	3
Плавно меняем яркость	цифровой ШИМ, который «притворяется» аналоговым	4
Измеряем положение ручки	аналоговый вход → АЦП → число	5
Выводим картинку на экран	цифровой, два провода	6
Передаём данные по Wi-Fi	цифровые данные внутри радиоволны	8

Таблица С.3: Какие сигналы встретятся в следующих уроках

Итоги урока

- Сигнал — это величина, которая меняется со временем и несёт информацию. В электронике это напряжение.
- Аналоговый сигнал плавный и принимает любые значения; цифровой — только догово-

рённые (обычно 0 и 1).

- Цифровой сигнал устойчив к помехам: его легко восстановить по порогу.
- АЦП превращает аналоговый сигнал в числа: измеряет (дискретизация) и округляет (квантование).
- N бит дают 2^N вариантов: 8 бит — 256, 12 бит — 4096.
- ESP32-S3 понимает уровни 0 и 3,3 В. Напряжение 5 В для его выводов опасно.

Проверь себя

1. Какой сигнал у обычного выключателя света, а какой — у ручки громкости? Почему?
2. На вход ESP32-S3 подали 1,5 В. Это 0, 1 или «серая зона»?
3. Сколько вариантов можно закодировать 5 битами?
4. Переведи в обычное число двоичное `00101101`.

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Найдите по три аналоговых и цифровых источника сигнала. Запишите передаваемую величину.
2. ★ Переведите число 13 в двоичную запись из восьми бит и проверьте обратным переводом.
3. ★★ Нарисуйте сигнал для буквы A (код 65): восемь бит, старший первым, один бит длится 1 мс.
4. ★★ Вычислите шаг идеального 10-битного и 12-битного АЦП для диапазона 3,3 В. Сравните результаты.
5. ★★★ Нарисуйте исходную синусоиду, точки измерений и округлённые уровни 3-битного АЦП. Покажите две разные причины потери информации.
6. ★★★★ Для сигнала до 3000 Гц выберите частоту дискретизации с запасом. Объясните, зачем перед АЦП нужен фильтр, и приведите пример наложения частот.

Выводы платы: что есть что

Цель	узнать назначение выводов платы: питание, земля, цифровые и аналоговые входы, особые выводы
Опирается на	вводная глава «Сигналы» (аналоговые и цифровые сигналы)
Результат	находим на плате выводы, которые понадобятся в курсе, и запоминаем, какие трогать нельзя
Понадобится	плата ESP32-S3-DevKitC-1

Теперь, когда мы знаем, чем аналоговый сигнал отличается от цифрового, вернёмся к выводам платы. Вот полная карта — её стоит держать под рукой все уроки:

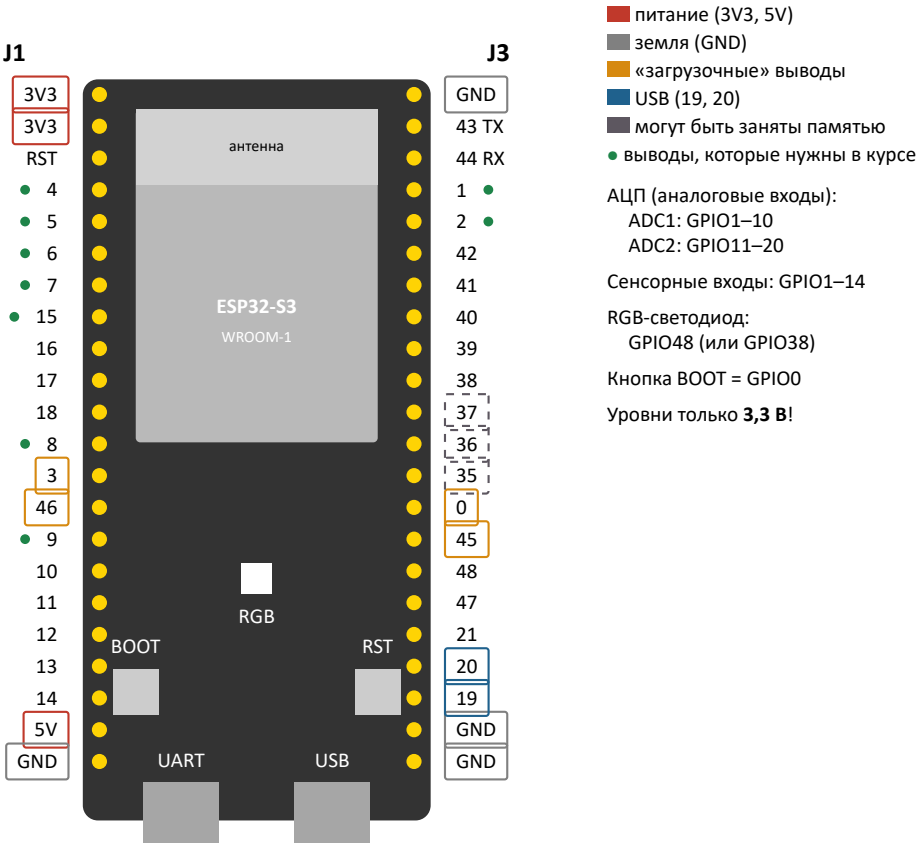


Рис. П.1: Карта выводов ESP32-S3-DevKitC-1 (вид сверху). Числа — номера GPIO. У разных версий платы бывают отличия — сверяйся с документацией своей платы

Питание и земля

Электрический ток течёт только по **замкнутому кругу**: от «плюса» источника через нагрузку (например, светодиод) и обратно к «минусу». Если круг где-то разорван — тока нет. Это похоже на воду в трубах: насос гонит воду по кругу, а если перекрыть трубу, течение прекратится.

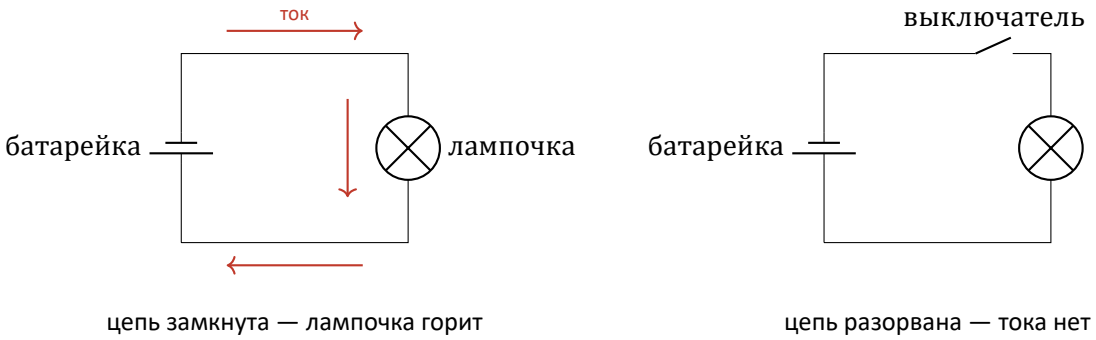


Рис. П.2: Электрическая цепь: ток течёт только по замкнутому кругу

На плате для этого есть выводы питания:

- **GND** (англ. *ground* — земля) — общий «минус». От него отсчитывают все напряжения, как высоту гор отсчитывают от уровня моря. На плате несколько выводов GND, все они соединены.
- **3V3** — «плюс» 3,3 В. Отсюда будем питать датчики и экраны.
- **5V** — 5 В прямо от USB. Подходит для питания моторчиков и модулей на 5 В, но *никогда* не подключай 5 В к GPIO!

Осторожно

Никогда не соединяй проводом 3V3 или 5V с GND напрямую. Это **короткое замыкание**: ток станет огромным, плата нагреется и может сгореть, а USB-порт компьютера — отключиться.

Цифровые выводы GPIO

Свободный GPIO, доступный на вашей плате, может быть **цифровым**:

- **выходом** — программа выдаёт на нём 0 (0 В) или 1 (3,3 В). Так мы будем зажигать светодиоды (урок 1);
- **входом** — программа узнаёт, что на выводе: 0 или 1. Так мы будем читать кнопки (урок 3).

Вывод-выход похож на переключатель, который подсоединяет ножку либо к 3,3 В, либо к земле:

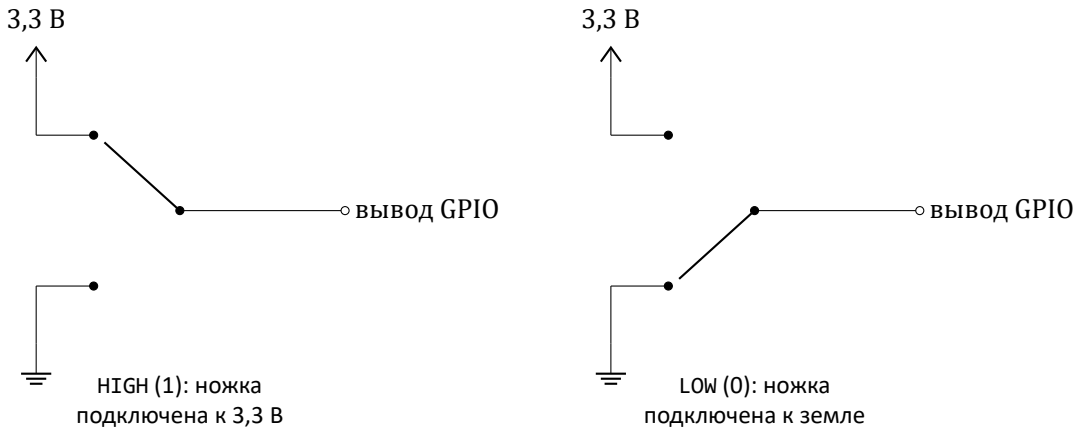


Рис. П.3: Цифровой выход можно представить как переключатель. На самом деле внутри чипа стоят два крошечных транзистора, и переключаются они миллионы раз в секунду

Осторожно

В опытах ограничиваем ток светодиода резистором до нескольких миллиампер. Допустимый ток зависит от режима вывода и общей нагрузки чипа; но мотор, реле или мощную лампу подключать напрямую **нельзя** — только через транзистор или специальный модуль-драйвер.

Аналоговые входы (АЦП)

Выводы **GPIO1–GPIO20** умеют не только отличать 0 от 1, но и **измерять напряжение**: внутри чипа к ним можно подключить АЦП, о котором мы говорили в вводной главе «Сигналы». АЦП в ESP32-S3 12-битный: он превращает напряжение от 0 до ≈3,1 В в число от 0 до 4095.
АЦП в чипе два:

- **ADC1** — выводы **GPIO1–GPIO10**. рекомендуется для опытов курса;
- **ADC2** — выводы **GPIO11–GPIO20**. Используется также Wi-Fi; доступ к нему зависит от режима и драйвера.

Поэтому для аналоговых датчиков лучше сразу брать выводы ADC1. Мы будем использовать **GPIO1** и **GPIO2**.

Сенсорные входы

Выводы **GPIO1–GPIO14** умеют чувствовать **прикосновение пальца**. Человеческое тело немного проводит ток и работает как маленький конденсатор, поэтому, если коснуться провода, подключённого к такому выводу, чип это заметит. Так устроены сенсорные кнопки на плитах и лампах.

Выводы для связи и особые выводы

Выводы	Для чего нужны
GPIO43 (TX), **GPIO44** (RX)	связь с компьютером через разъём UART (урок 2)
GPIO19, **GPIO20**	разъём USB — лучше не занимать
GPIO0, **GPIO3**, **GPIO45**, **GPIO46**	**«загрузочные»**: чип смотрит на них при включении, чтобы понять, запускать программу или ждать прошивку. **GPIO0** — это кнопка BOOT. Не подключай к ним кнопки и датчики
GPIO35, **GPIO36**, **GPIO37**	на многих модулях заняты памятью PSRAM — не использовать
GPIO48 (или **GPIO38**)	встроенный RGB-светодиод
GPIO26–GPIO32	заняты памятью программ, на гребёнку не выведены

Таблица П.1: Выводы с особыми обязанностями

А где же выводы для экранов (I^2C) и других модулей? У ESP32-S3 есть удобная особенность — **матрица GPIO**. Это как старинный телефонный коммутатор: внутренний блок связи можно «соединить» почти с любым выводом. Поэтому мы сами выбираем, какие выводы использовать, и просто сообщаем об этом программе.

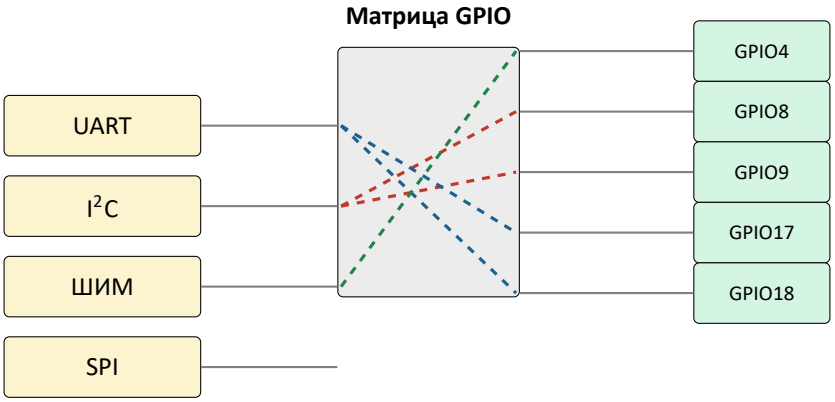


Рис. П.4: Матрица GPIO соединяет внутренние блоки с выводами, которые мы выбрали

Выводы, которые мы будем использовать

Чтобы не путаться, во всех уроках одни и те же детали подключаются к одним и тем же выводам:

Вывод	Что подключаем	Уроки
GPIO4	светодиод	1, 4, 8, 2, 3, 9
GPIO5, **GPIO6**, **GPIO7**	ещё три светодиода для «бегущего огня»	1, 4
GPIO5	кнопка	3, 9
GPIO6	сервопривод	5
GPIO15	пьезопищалка	4
GPIO1	потенциометр (аналоговый вход)	5, 6, 9
GPIO2	делитель напряжения (аналоговый вход)	5
GPIO8, **GPIO9**	экран: SDA и SCL	6
GPIO17, **GPIO18**	второй UART	2

Таблица П.2: Кто куда подключается в этом курсе

Итоги урока

- Ток течёт по замкнутой цепи. GND — общий «минус», 3V3 — «плюс» 3,3 В, 5V — питание от USB.
- Любой GPIO может быть цифровым входом или выходом; **GPIO1**–**GPIO20** ещё и измеряют напряжение (АЦП).
- Для аналоговых датчиков в проектах с Wi-Fi выбираем ADC1 (**GPIO1**–**GPIO10**).
- Загрузочные выводы, выводы USB и памяти лучше не занимать.

Проверь себя

1. Почему нельзя соединять 3V3 и GND проводом?
2. Какие выводы подойдут для измерения напряжения, если в проекте нужен Wi-Fi?
3. Можно ли подключить мотор прямо к выводу GPIO? Почему?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Найдите на обесточенной плате 3V3, GND, GPIO4 и оба разъёма USB.

2. ★ Отметьте на бумажной карте выводы питания, USB, памяти и загрузки разными цветами.
3. ★★ Выберите выводы для светодиода, кнопки и аналогового датчика. Объясните выбор.
4. ★★ Нарисуйте цепь светодиода с резистором и общий GND платы и внешнего источника. Собирать пока не нужно.
5. ★★★ Спланируйте подключение OLED, потенциометра, кнопки и светодиода без конфликтов выводов. Сверьте со справочником курса.
6. ★★★★ Сравните две ревизии DevKitC-1 по документации: найдите вывод RGB и ограничения памяти. Составьте собственную карту для своей платы.

Перед первым уроком: сигнал, программа и безопасность

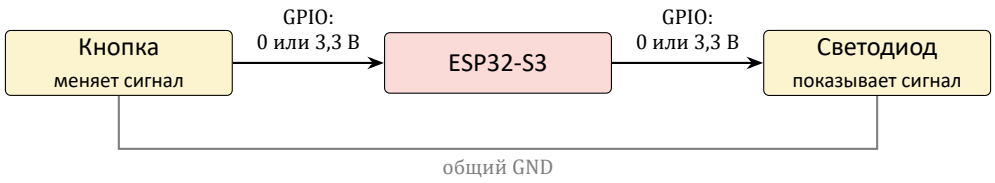
Эта страница — фундамент всей книги. Не нужно заранее знать физику или C++. Сейчас достаточно понять одну идею: плата получает информацию, *измеряя напряжение*, и действует, *меняя напряжение* на своих выводах.

Совет

Примеры этой главы запускаются после установки среды в уроке 0. Если Arduino IDE ещё не установлена, прочитайте главу сейчас, а примеры выполните сразу после урока 0.

Что такое электрический сигнал

У ESP32-S3 есть выводы GPIO — маленькие контакты, к которым подключают кнопку, светодиод или датчик. Напряжение на выводе всегда сравнивают с **GND** (землёй, нулём схемы). Поэтому у двух соединённых устройств обязательно должна быть общая земля.



Цифровой сигнал имеет два понятных состояния: LOW — около 0 В, HIGH — около 3,3 В. Позже потенциометр покажет **аналоговый** сигнал: там напряжение может быть любым между этими границами. GPIO ESP32-S3 рассчитаны только на 3,3 В: 5 В на GPIO может необратимо повредить чип.

Как программа управляет платой

Скетч Arduino — это текст с двумя обязательными функциями. `void` означает: функция ничего не возвращает. `setup()` запускается один раз после включения. `loop()` затем повторяется снова и снова, пока плата получает питание.

Листинг B.1: Самая маленькая программа: увидеть работу `setup()` и `loop()`

```
void setup() {
  Serial.begin(115200);              // открыть связь с компьютером
  Serial.println("Старт: setup() был вызван один раз");
}

void loop() {
  Serial.println("Сейчас выполняется loop()");
  delay(500);                        // подождать полсекунды
}
```

Откройте Serial Monitor на скорости 115200. Слово «Старт» появится один раз, а строка из `loop()` — много раз. Это и есть жизненный цикл простой программы.

Первый сигнал без подключения деталей

Команда ниже переводит GPIO4 в режим выхода, затем задаёт на нём HIGH и LOW. Светодиод пока не нужен: мы проверяем логику в Serial Monitor. В уроке 1 этот же сигнал зажжёт настоящий LED через резистор.

Листинг B.2: Вывод GPIO меняет цифровой сигнал

```
const int PIN = 4;                   // имя для номера вывода

void setup() {
  Serial.begin(115200);
  pinMode(PIN, OUTPUT);              // теперь GPIO4 создаёт сигнал
}

void loop() {
  digitalWrite(PIN, HIGH);           // GPIO4: около 3,3 В относительно GND
  Serial.println("HIGH: на выводе есть сигнал");
  delay(1000);

  digitalWrite(PIN, LOW);            // GPIO4: около 0 В относительно GND
  Serial.println("LOW: сигнал выключен");
  delay(1000);
}
```

Осторожно

Никогда не соединяйте GPIO напрямую с 5 В, с мотором, реле или светодиодной лентой. GPIO — это сигнальный вывод, а не источник питания. Светодиоду нужен резистор; мощной нагрузке — драйвер или транзисторный ключ. До любой новой сборки отключайте USB и сначала сверяйте схему.

Итоги урока

- Сигнал — это напряжение относительно GND; цифровой сигнал ESP32-S3: LOW ≈ 0 В или HIGH ≈ 3,3 В.
- `setup()` работает один раз, `loop()` повторяется; сначала учимся наблюдать результат через Serial Monitor.
- Пины GPIO безопасны только для 3,3 В и малых токов.

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Измените сообщение в `setup()` на своё имя, а сообщение в `loop()` — на номер текущего шага.
2. ★ На схеме платы найдите GND, 3V3, GPIO4, BOOT и RST (EN). Пока ничего к ним не подключайте.
3. ★★ Поменяйте задержку в `loop()` на 1000 мс. Что изменится в Serial Monitor? Сколько строк придёт за 10 секунд?
4. ★★ Добавьте в `loop()` счётчик и выводите «Шаг 1», «Шаг 2» и так далее. Почему переменная должна быть объявлена вне функции?
5. ★★★ Выведите в Serial Monitor «таблицу» из трёх строк, повторяющуюся каждые 2 секунды: имя, номер шага и время работы платы в секундах (`millis() / 1000`).
6. ★★★★ Сделайте так, чтобы программа раз в секунду печатала «HIGH», а в следующую секунду «LOW», имитируя сигнал на выводе, и нарисуйте его график на бумаге.

Урок 0. Знакомство с ESP32-S3 и настройка среды

Цель	понять, из чего состоит ESP32-S3, как устроена отладочная плата, установить среду и залить первую программу
Опирается на	ничего — это стартовый урок
Результат	проверяем, что плата готова к самостоятельным опытам
Понадобится	плата ESP32-S3-DevKitC-1, USB-кабель с передачей данных

0.1 Сначала — что именно мы делаем

Мы пишем программу (скетч), компьютер превращает её в прошивку и записывает в плату по USB. Успех урока: сообщение из программы видно в Serial Monitor. Это проверяет кабель, IDE, порт и плату сразу. Названия ядер, памяти и шин ниже пока не нужно запоминать: сейчас важны USB, BOOT/EN, GND, 3V3 и безопасные GPIO.

0.2 Что такое ESP32-S3

ESP32-S3 — система на кристалле (SoC) компании Espressif. Главное:

- **Два ядра Xtensa LX7** до 240 MHz с векторными инструкциями (ускорение DSP и нейросетей).
- 512 KB **SRAM** на кристалле, 384 KB ROM, до 16 MB внешней Flash и до 16 MB внешней PSRAM (в том числе в режиме Octal SPI).
- **Wi-Fi 802.11 b/g/n** (2,4 GHz) и **Bluetooth 5 LE**. Классического Bluetooth (BR/EDR) *нет* — в отличие от первого ESP32.
- **Нативный USB OTG** и встроенный **USB-Serial/JTAG**: прошивать и отлаживать можно без отдельного преобразователя.

- **45 GPIO**, 2 АЦП по 12 бит (20 каналов), 14 сенсорных (touch) входов, 3 UART, 2 I^2C, 2 I^2S, 4 SPI (два доступны пользователю), LEDC, MCPWM, RMT, TWAI (CAN), интерфейс камеры и LCD.
- **Сопроцессоры ULP** (RISC-V и FSM), которые работают, пока основные ядра спят.

Не нужно запоминать всё сразу: каждый блок схемы ниже мы по очереди задействуем в уроках курса.

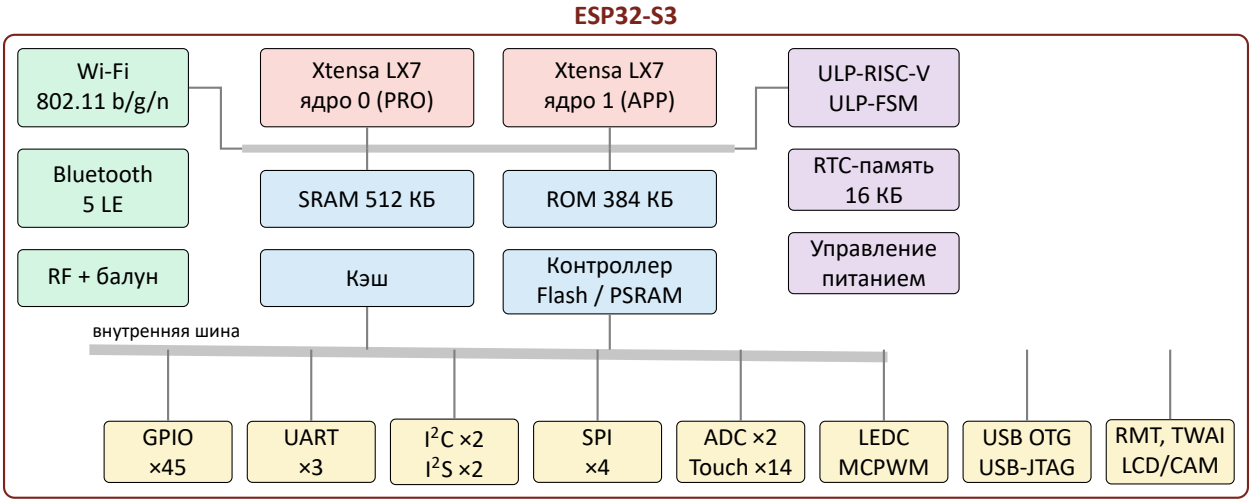


Рис. 0.1: Упрощённая блок-схема ESP32-S3: оба ядра, память, радио и RTC-домен связаны общими шинами

Совет

Цвета на схемах книги постоянны:

микроконтроллер внешние устройства и действия человека
программа, данные и память
радио и сеть питание, сброс и сон

0.3 Сравнение с «родственниками»

	ESP32	ESP32-S3	ESP32-C3
Ядро	2× Xtensa LX6, 240 МГц	2× Xtensa LX7, 240 МГц	1× RISC-V, 160 МГц
SRAM	520 КБ	512 КБ	400 КБ
Bluetooth	Classic + BLE 4.2	BLE 5	BLE 5
USB	нет	OTG + Serial/JTAG	Serial/JTAG
GPIO	34	45	22
Особенности	ЦАП, Ethernet MAC	векторные инструкции, камера, LCD	дешёвый, малый

Таблица 0.1: Сравнение популярных чипов Espressif

0.4 Модули и маркировка

Чип почти всегда используется в составе *модуля* (ESP32-S3-WROOM-1, WROOM-1U, MINI-1), где уже есть Flash, кварц и антенна. Суффикс модуля кодирует объём памяти:

ESP32-S3-WROOM-1-N16R8

Flash 16 МБ — PSRAM 8 МБ (Octal)

Осторожно

В модулях с **Octal PSRAM** (R8, R16V) выводы **GPIO35**, **GPIO36**, **GPIO37** заняты памятью — не используйте их. Выводы **GPIO26–GPIO32** всегда заняты SPI-Flash и на плату обычно не выведены.

0.5 Отладочная плата ESP32-S3-DevKitC-1

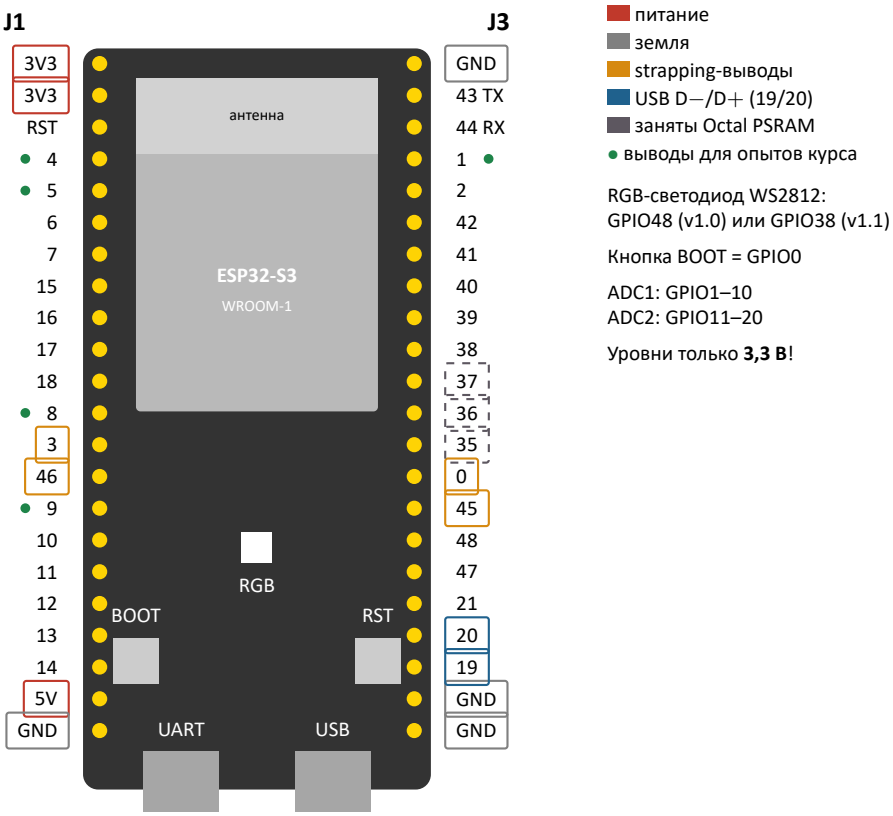


Рис. 0.2: Разводка выводов ESP32-S3-DevKitC-1 (вид сверху). Числа — номера GPIO. Сверьтесь с документацией именно вашей ревизии платы.

У платы **два USB-разъёма**:

- **UART** — через микросхему-преобразователь (CP2102N/CH343) на UART0 (**GPIO43**/**GPIO44**). Самый надёжный вариант для прошивки и вывода сообщений.
- **USB** — напрямую к выводам нативного USB чипа (**GPIO19**/**GPIO20**). Позволяет прошивать, отлаживать через JTAG и превращать плату в USB-клавиатуру, MIDI-устройство и т. п.

0.6 Установка среды

Для уроков используем Arduino IDE 2.x (подойдёт и PlatformIO — код не меняется).

1. Установите Arduino IDE 2 с сайта `https://www.arduino.cc/en/software`.

2. File → Preferences → Additional boards manager URLs, добавьте:
   `https://espressif.github.io/arduino-esp32/package_esp32_index.json`
3. Tools → Board → Boards Manager, найдите **esp32 by Espressif Systems**, установите версию 3.x.
4. Выберите плату **ESP32S3 Dev Module** и порт.
5. Если подключены через разъём **USB** (нативный), включите Tools → USB CDC On Boot: Enabled, иначе сообщения от платы не дойдут до компьютера.
6. Для модулей с PSRAM выберите Tools → PSRAM: OPI PSRAM (для R8) или QSPI PSRAM (для R2).

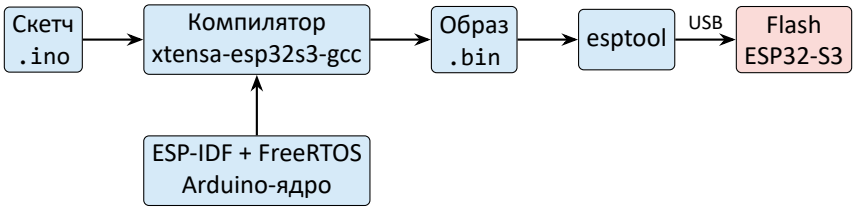


Рис. 0.3: Путь от исходного кода до прошивки

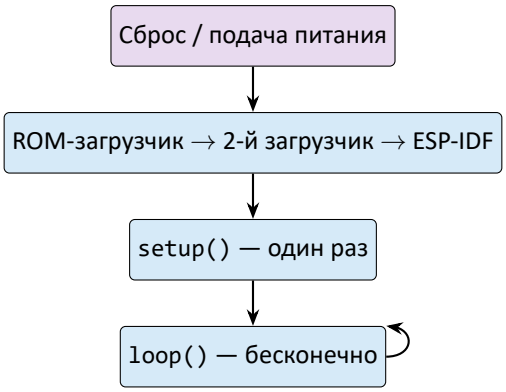
Если прошивка не начинается

Чип входит в режим загрузчика, если при сбросе **GPIO0** притянут к земле. Обычно IDE делает это автоматически, но если видите `Failed to connect`:



0.7 Структура Arduino-скетча

Функции `setup()` и `loop()` уже знакомы по вводной главе. Схема ниже показывает, что происходит до их вызова. (Под ними работает операционная система FreeRTOS; она понадобится в уроке 7.)



Первая программа ничего не подключает: она только сообщает о себе через USB. Как именно работает этот канал связи, разберём в уроке 2, а пока просто проверим, что всё настроено.

Листинг 0.1: Первый скетч: «Hello, ESP32-S3»

```
void setup() {
  Serial.begin(115200);              // открыть канал связи с компьютером
  delay(500);                        // дать USB время подняться
  Serial.println("Привет, ESP32-S3!");
  Serial.printf("Частота CPU: %lu МГц\n", getCpuFrequencyMhz());
  Serial.printf("Свободно SRAM: %lu байт\n", ESP.getFreeHeap());
  Serial.printf("PSRAM: %lu байт\n", ESP.getPsramSize());
  Serial.printf("Flash: %lu байт\n", ESP.getFlashChipSize());
}

void loop() {
  Serial.printf("Прошло %lu мс\n", millis());
  delay(1000);
}
```

0.8 Прочитайте пример как инструкцию

`Serial.begin(115200)` открывает разговор с компьютером; число 115200 должно совпасть со скоростью в Serial Monitor. `Serial.println(...)` печатает строку и переходит на новую строку. Если вместо текста видны непонятные символы, почти всегда выбрана неверная скорость порта.
Откройте Tools → Serial Monitor, установите скорость 115200 — должны появиться сообщения.

0.9 Учимся программировать: как записывается программа

Программа похожа на рецепт: список команд, которые выполняются *по порядку*, сверху вниз. Компьютер выполняет их очень точно и ничего не додумывает. Если в рецепте написано «положить 100 кг соли», компьютер положит 100 кг. Поэтому у языка C++ (читается «си-плюс-плюс»), на котором пишут программы для Arduino, есть строгие правила записи. Посмотри на них на примере строчки из следующей программы:

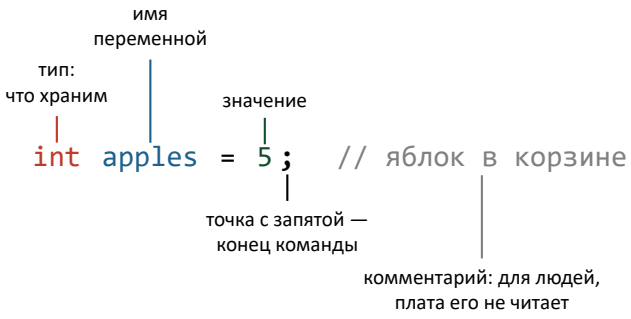


Рис. 0.4: Из чего состоит строчка программы

- каждая команда заканчивается **точкой с запятой** `;`;
- после `//` до конца строки идёт **комментарий** — пояснение для людей;
- фигурные скобки `{ }` объединяют несколько команд в **блок**, как абзац в тексте. Всё, что между скобками после `void setup()`, — это содержимое функции `setup()`;
- большие и маленькие буквы различаются: `apples` и `Apples` — разные имена;
- в дробных числах пишут **точку**, а не запятую: `23.5`;

- текст пишут в двойных кавычках: `"Привет"`.

Совет

Если в программе ошибка, при компиляции внизу Arduino IDE появится красное сообщение с номером строки. Чаще всего виноваты забытая `;`, незакрытая скобка или опечатка в имени. Ошибки — это нормально: их делают все программисты, даже самые опытные.

0.10 Учимся программировать: переменные

Программе нужно что-то помнить: сколько раз нажали кнопку, какая яркость у светодиода, какая температура. Для этого есть **переменные**. Переменная — это коробка с названием, в которой хранится значение. Значение можно положить, посмотреть и заменить другим. На коробке написано не только имя, но и **тип** — что в неё можно класть: целые числа, дробные, текст.

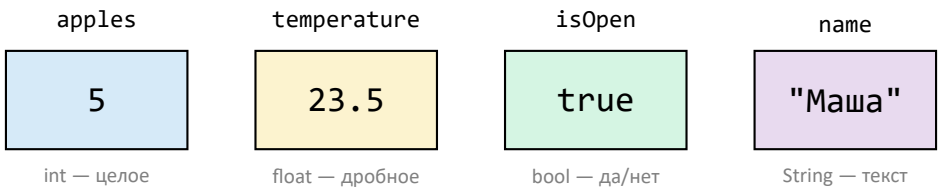


Рис. 0.5: Переменные — подписанные коробки. Тип говорит, что в коробку можно положить

Тип	Что хранит	Пример
`int`	целое число от $-2\,147\,483\,648$ до $2\,147\,483\,647$ (32 бита)	`int score = 10;`
`float`	дробное число (около 7 точных цифр)	`float t = 36.6;`
`bool`	логическое значение: `true` (истина) или `false` (ложь)	`bool on = false;`
`String`	строку текста	`String s = "Привет";`
`uint8_t`, `byte`	целое от 0 до 255 (8 бит, как в вводной главе «Сигналы»)	`byte r = 200;`
`uint32_t`	целое от 0 до 4 294 967 295 — для времени `millis()`	`uint32_t t = millis();`

Таблица 0.2: Типы переменных, которые встретятся в курсе

Знак `=` в программировании означает не «равно», а «**положить в коробку**». Запись `apples = apples + 3` читается так: «возьми, что лежит в `apples`, прибавь 3 и положи результат обратно в `apples`». В математике это бессмыслица, а для программы — обычное дело.

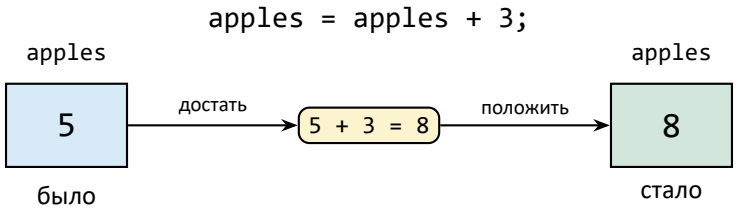


Рис. 0.6: Присваивание: сначала считается правая часть, потом результат кладётся в переменную

Запись	Что делает	Пример ($x = 7$)
`+ - * /`	сложение, вычитание, умножение, деление	`x * 2` $\to 14$
`%`	остаток от деления	`x % 2` $\to 1$
`x++`, `x--`	увеличить или уменьшить на 1	x станет 8 или 6
`x += 5`	то же, что `x = x + 5`	x станет 12

Таблица 0.3: Арифметические операции

Листинг 0.2: Переменные и операции

```ino
// Переменные: коробки с названиями
int apples = 5;                    // целое число
float temperature = 23.5;          // дробное число (точка вместо запятой!)
bool isOpen = true;                // логическое: true (да) или false (нет)
String name = "Маша";              // строка текста
const int DAYS_IN_WEEK = 7;        // константа: менять нельзя

void setup() {
  Serial.begin(115200);
  delay(500);

  Serial.print("Яблок: ");
  Serial.println(apples);          // 5

  apples = apples + 3;             // взять старое значение, прибавить 3, положить обратно
  Serial.print("Стало яблок: ");
  Serial.println(apples);          // 8

  apples++;                        // то же, что apples = apples + 1
  Serial.println(apples);          // 9

  Serial.println(7 / 2);           // 3   -- целые числа делятся нацело!
  Serial.println(7 % 2);           // 1   -- остаток от деления
  Serial.println(7.0 / 2);         // 3.50 -- а дробные -- как обычно

  Serial.println("Привет, " + name + "!");
  Serial.printf("На улице %.1f градусов\n", temperature);
}

void loop() {}
```

Осторожно

Если оба числа целые, деление тоже будет целым: `7 / 2` даёт **3**, а не 3,5 — дробная часть просто отбрасывается. Чтобы получить 3,5, хотя бы одно число должно быть дробным: `7.0 / 2`.

Переменная, объявленная *вне* функций (в начале программы), — **глобальная**: её видят все части программы. Переменная, объявленная *внутри* фигурных скобок, — **локальная**: она существует только внутри этого блока. Как записка на доске в классе (видят все) и записка в твоём кармане (видишь только ты).

Теперь понятно, что было в первой программе: `setup()` и `loop()` — это **функции**, наборы команд под одним именем; `Serial.println()` и `delay()` — готовые функции, которые мы *вызываем*, передавая им в скобках данные; `millis()` возвращает число — его можно положить в переменную.

0.11 Как устроены уроки

Каждый урок построен одинаково: цель и список деталей, короткое объяснение идеи, устройство периферии, готовый пример, *самостоятельный опыт*, итоги и задания. Опыты не связаны общим проектом: каждый собирается отдельным скетчем, поэтому урок можно повторить или пропустить, не ломая остальные. В конце урока — шесть заданий, от лёгких к трудным, с отметкой уровня: ★ — повторить пример, ★★ — изменить и дополнить, ★★★ — создать своё устройство, ★★★★ — вызов для тех, кто хочет большего. Выполняйте задания по порядку: каждое опирается на предыдущее.
Чтобы не пересобирать макетную плату, детали закрепим за одними и теми же выводами:

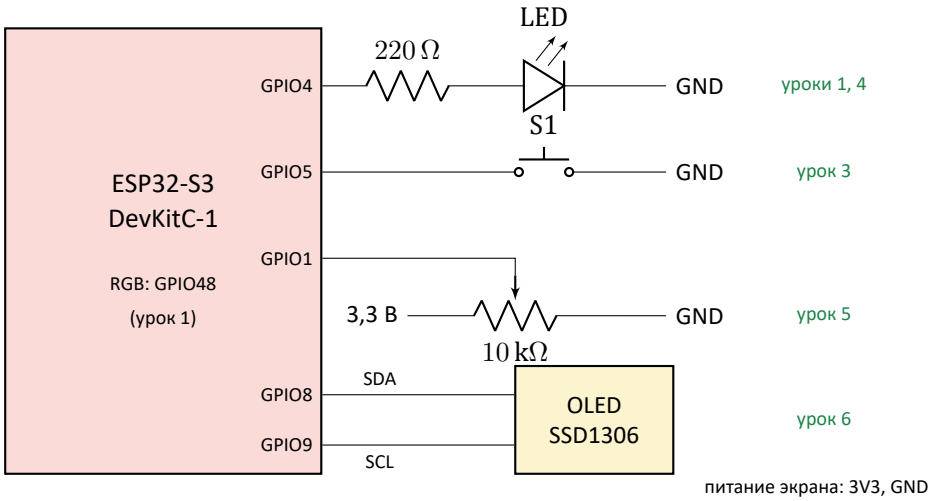


Рис. 0.7: Учебный стенд: какие детали к каким выводам подключаются в опытах курса

Итоги урока

- ESP32-S3 — двухъядерный чип с Wi-Fi, BLE, USB и богатой периферией; работает на 3,3 V.
- Прошивка заливается из Arduino IDE; при проблемах помогает ручной вход в загрузчик (BOOT + RST).
- Скетч — это `setup()` (один раз) и `loop()` (бесконечно).

Проверь себя

1. Чем модуль отличается от чипа и что означает суффикс `N16R8`?
2. Зачем на плате два USB-разъёма и чем они различаются?
3. Что делать, если IDE пишет `Failed to connect`?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выясните, какой модуль стоит на вашей плате (надпись на экране модуля) и сколько у него Flash и PSRAM.
2. ★ Выведите в Serial Monitor номер версии вашей Arduino IDE и название выбранной платы (из меню Tools).
3. ★★ Выведите MAC-адрес чипа: `ESP.getEfuseMac()`. Зачем устройствам нужен уникальный адрес?

4. ★★ Выведите размер Flash (`ESP.getFlashChipSize()`) и свободную кучу (`ESP.getFreeHeap()`) и сравните с данными из маркировки модуля.
5. ★★★ Измените частоту CPU функцией `setCpuFrequencyMhz(80)` и проверьте результат через `getCpuFrequencyMhz()`. Как это повлияет на скорость простого цикла? Измерьте через `micros()`.
6. ★★★★ Составьте «паспорт платы»: программа печатает модель чипа, число ядер, частоту, Flash, PSRAM, MAC и причину последнего сброса в виде аккуратной таблицы.

Урок 1. Цифровой выход: светодиод

Цель	научиться управлять выводом GPIO как выходом, правильно подключать светодиод, отмерять время без остановки программы
Опирается на	урок 0: установка среды, `setup()` и `loop()`
Результат	светодиод на **GPIO4** мигает, не останавливая программу; встроенный RGB-светодиод меняет цвет
Понадобится	макетная плата, светодиод, резистор $220 – 330\,\Omega$, провода

1.1 Идея урока коротко

GPIO4 создаёт HIGH (около 3,3 В) или LOW (около 0 В), а светодиод показывает этот сигнал светом. Току нужен замкнутый путь: GPIO4 → резистор → LED → GND. `const int` даёт имя номеру пина; `pinMode` назначает OUTPUT; `digitalWrite` выбирает HIGH или LOW; `delay` ждёт миллисекунды.

1.2 Зачем нужен резистор: закон Ома

Электричество удобно сравнивать с водой в трубах:

- **напряжение** U (вольты, В) — это «давление», которое толкает воду;
- **ток** I (амперы, А) — сколько воды протекает за секунду;
- **сопротивление** R (омы, Ω) — насколько узкая труба: чем уже, тем меньше воды пройдёт.

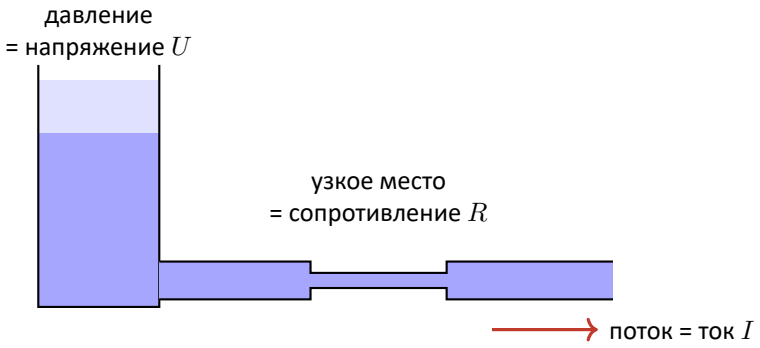


Рис. 1.1: «Водяная» модель электрической цепи

Эти три величины связаны **законом Ома** — главным законом электроники:

$$I = \frac{U}{R} \qquad \text{или} \qquad R = \frac{U}{I}.$$

Больше давление — больше поток. Уже труба — меньше поток.

Почему светодиод сгорает без резистора

Светодиод ведёт себя необычно: пока напряжение на нём маленькое, ток почти не течёт. Но стоит напряжению превысить определённый порог — **прямое напряжение** U_F, для красного светодиода около 2 В, — ток начинает расти *очень* быстро:

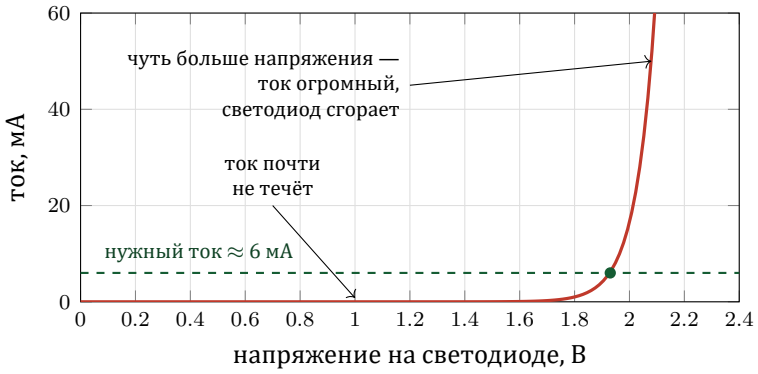


Рис. 1.2: Как ток через красный светодиод зависит от напряжения на нём (вольт-амперная характеристика)

Вывод платы выдаёт 3,3 В — намного больше, чем нужно светодиоду. Если подключить его напрямую, ток окажется огромным, и светодиод (а может, и вывод платы) сгорит. **Резистор** забирает на себя «лишнее» напряжение и ограничивает ток. Поэтому правило:

Светодиод подключается к выводу только через резистор!

Рассчитываем резистор

Из 3,3 В на светодиод придётся $U_F \approx 2{,}0$ В (для красного), значит, на резистор — оставшиеся $3{,}3 - 2{,}0 = 1{,}3$ В. Светодиоду достаточно тока около 6 мА = 0,006 А. По закону Ома:

$$R = \frac{U_{\text{питания}} - U_F}{I} = \frac{3{,}3\ \text{В} - 2{,}0\ \text{В}}{0{,}006\ \text{А}} \approx 217\ \Omega.$$

Резисторы продаются только определённых номиналов, поэтому берём ближайший: $220\,\Omega$. Разные светодиоды требуют разного напряжения: красному хватает около 2 В, а синему нужно почти 3 В.[1] Поэтому и резистор для них нужен разный:

[1] Всё дело в энергии света. Свет состоит из крошечных порций — фотонов, и у синего света каждый фотон несёт больше энергии, чем у красного. Чтобы «выбить» из светодиода синий фотон, электрону нужно больше энергии — значит, нужно и большее напряжение. Поэтому ультрафиолетовые светодиоды требуют ещё больше, чем синие, а инфракрасные (в пульте от телевизора) — меньше всего, около 1,2 В.

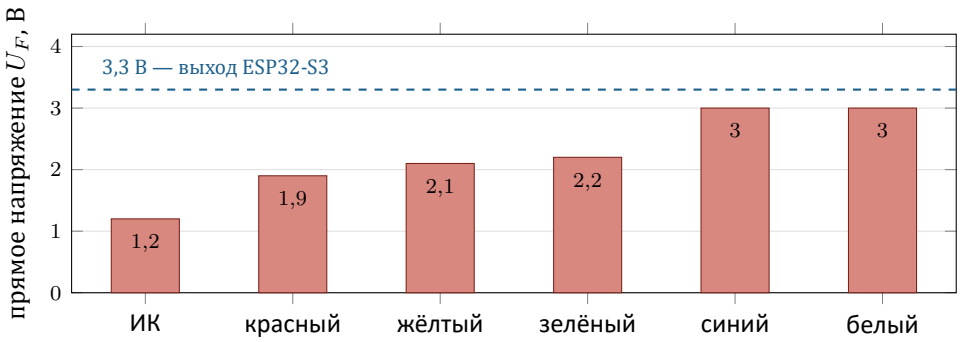


Рис. 1.3: Прямое напряжение светодиодов разных цветов (примерные значения)

Цвет светодиода	U_F, В	На резисторе, В	Резистор для 6 мА
Красный	1,9	$3{,}3 - 1{,}9 = 1{,}4$	$1{,}4/0{,}006 \approx 230\,\Omega \to 220\,\Omega$
Жёлтый, зелёный	2,1–2,2	1,1–1,2	$\approx 190\,\Omega \to 180\,\Omega$–$220\,\Omega$
Синий, белый	3,0	0,3	$0{,}3/0{,}006 = 50\,\Omega \to 47\,\Omega$

Таблица 1.1: Пример расчёта резисторов при питании от вывода 3,3 В

Синему и белому светодиоду от 3,3 В достаётся совсем небольшой запас, поэтому они светят чуть слабее. Для опытов в курсе лучше всего подходят красные, жёлтые и зелёные светодиоды с резистором $220\,\Omega$.

Как узнать сопротивление резистора

Номинал резистора закодирован цветными полосками. Первые две — цифры, третья — сколько нулей дописать, а золотая полоска с краю — точность:

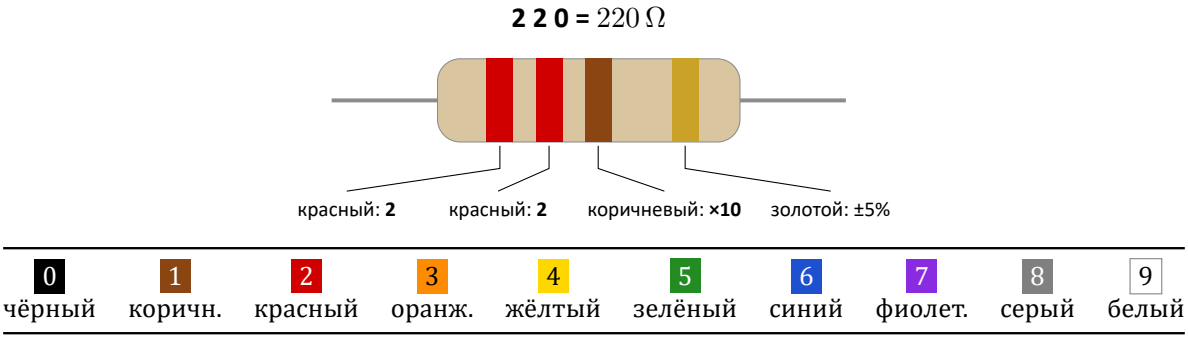


Рис. 1.4: Цветовая маркировка резисторов. Например, коричневый–чёрный–оранжевый = 1, 0 и три нуля = $10\,000\,\Omega$ = $10\,\mathrm{k}\Omega$

1.3 Как устроен выход GPIO

Выход GPIO — это пара транзисторов (двухтактная схема, push-pull). Логическая `1` подключает вывод к $3{,}3\,\mathrm{V}$, логический `0` — к земле.

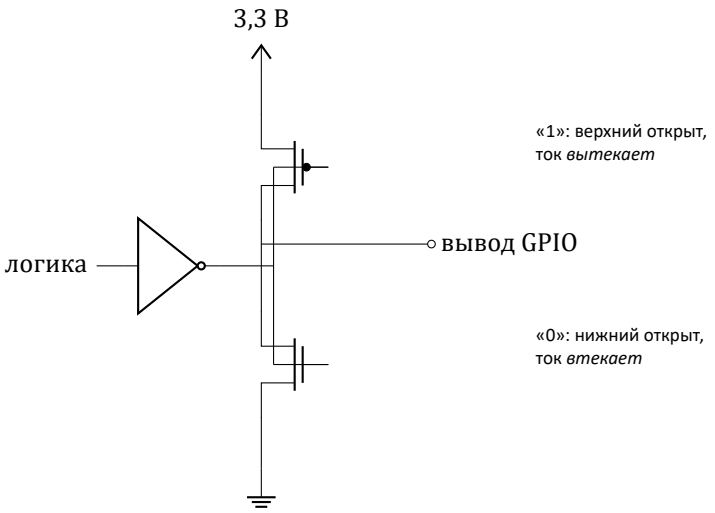


Рис. 1.5: Упрощённая схема выходного каскада GPIO

Осторожно

- ESP32-S3 работает с логикой **3,3 В**. Подача $5\,\text{V}$ на вывод может вывести чип из строя.
- Допустимый ток вывода — до $\approx 40\,\text{mA}$ (по умолчанию сила выхода ограничена $\approx 20\,\text{mA}$). Моторы, реле и мощные светодиоды подключаются только через транзистор или драйвер.

1.4 Схема подключения

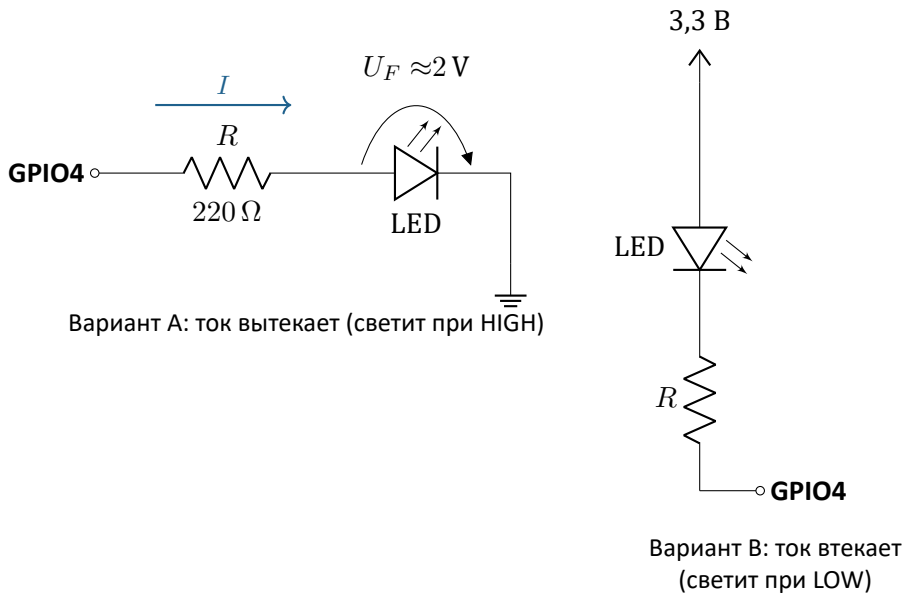


Рис. 1.6: Два способа подключить светодиод. В опытах используем вариант A

Резистор ограничивает ток. По закону Ома:

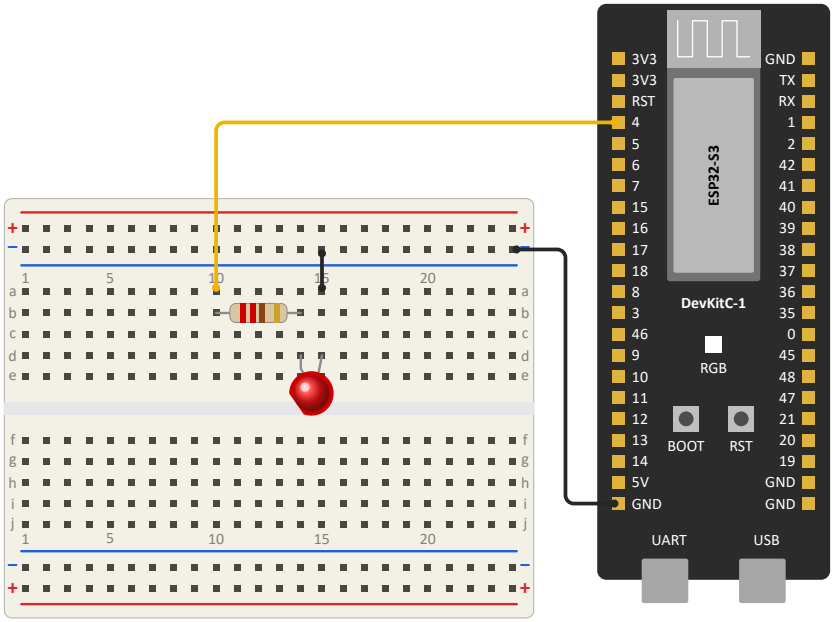
$$R = \frac{U_{\text{вых}} - U_F}{I} = \frac{3{,}3 - 2{,}0}{0{,}006} \approx 217\,\Omega \quad \Rightarrow \quad \text{берём } 220\,\Omega.$$

Цвет светодиода	U_F, В	R для 5 mA	R для 10 mA
Красный	1,8–2,0	270 Ω	150 Ω
Жёлтый/зелёный	2,0–2,2	220 Ω	120 Ω
Синий/белый	2,8–3,1	68 Ω	33 Ω

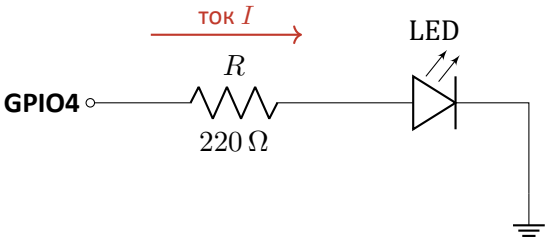
Таблица 1.2: Ориентировочные номиналы резисторов при питании 3,3 В

1.5 Подключаем светодиод

Соберём схему: вывод **GPIO4** → резистор 220 Ω → анод светодиода, катод → земля (GND).



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 1.7: Подключение светодиода к выводу GPIO4

Сравни две картинки: жёлтый провод на макетной плате — это линия от **GPIO4** на схеме, чёрные провода — значок земли. Проследи путь тока пальцем: от вывода платы через резистор и светодиод к земле и обратно в плату — цепь замкнута.

1.6 Классический Blink

Листинг 1.1: Мигание светодиодом

```
const int LED_PIN = 4;

void setup() {
  pinMode(LED_PIN, OUTPUT);     // настроить вывод как выход
}

void loop() {
  digitalWrite(LED_PIN, HIGH);  // 3,3 В -> светодиод горит
  delay(500);
  digitalWrite(LED_PIN, LOW);   // 0 В -> гаснет
  delay(500);
}
```

1.7 Разберём Blink по строкам

- `const int LED_PIN = 4;` не включает светодиод: это только удобное имя для GPIO4.
- `setup()` выполняется один раз и сообщает плате, что GPIO4 теперь `OUTPUT`.
- В `loop()` строка `digitalWrite(LED_PIN, HIGH)` даёт на GPIO4 3,3 В; LED загорается.
- `delay(500)` ждёт полсекунды. Потом LOW даёт 0 В, LED гаснет, и цикл начинается снова.

Если LED не мигает, сначала проверьте длинную ножку LED (анод), резистор и провод GND. Не меняйте программу, пока не убедились в схеме.

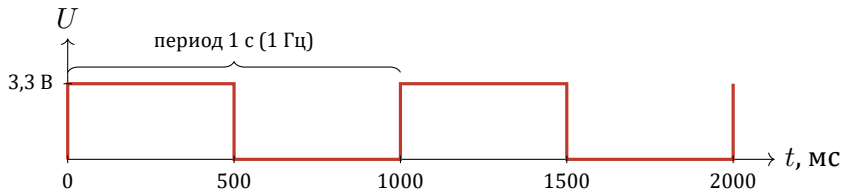


Рис. 1.8: Напряжение на выводе при работе Blink

1.8 Мигание без `delay()`

`delay()` останавливает всю программу: пока идёт ожидание, она не может делать ничего другого. Настоящее устройство одновременно следит за кнопкой, датчиками, экраном и сетью, поэтому сразу привыкаем к другому приёму — сравнивать текущее время `millis()` с моментом последнего действия:

Листинг 1.2: Неблокирующее мигание

```
const int LED_PIN = 4;
const uint32_t INTERVAL = 500;  // мс
uint32_t lastToggle = 0;
bool ledState = false;

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

void loop() {
  uint32_t now = millis();
  if (now - lastToggle >= INTERVAL) {  // корректно и при переполнении millis()
```

```
    lastToggle = now;
    ledState = !ledState;
    digitalWrite(LED_PIN, ledState);
  }
  // здесь можно делать что-то ещё, не дожидаясь светодиода
}
```

1.9 Учимся программировать: условие `if`

Часто нужно сделать что-то *только если* выполняется условие: «если прошло полсекунды — переключи светодиод». Для этого есть команда `if` («если»). Команды внутри фигурных скобок выполняются, только когда условие в круглых скобках верно. А после `else` («иначе») пишут, что делать, если условие неверно.

Алгоритмы удобно рисовать **блок-схемами**. Овал — начало и конец, прямоугольник — действие, ромб — вопрос с ответами «да» и «нет». Вот блок-схема функции `loop()` из программы ниже:

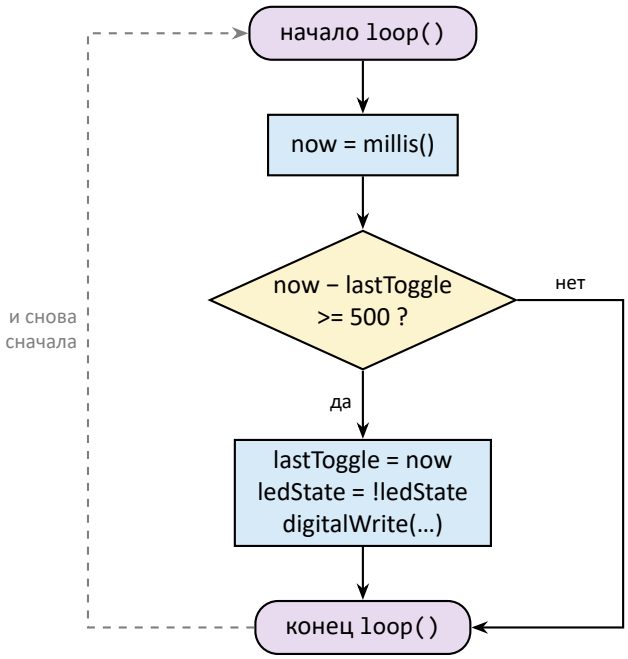


Рис. 1.9: Блок-схема неблокирующего мигания: `loop()` крутится тысячи раз в секунду, но светодиод переключается, только когда ответ на вопрос — «да»

Для сравнения чисел используются такие знаки:

Знак	Смысл	Пример	Знак	Смысл	Пример
`==`	равно	`x == 5`	`!=`	не равно	`x != 0`
`<`	меньше	`t < 0`	`>`	больше	`t > 30`
`<=`	меньше или равно	`n <= 10`	`>=`	больше или равно	`now - last >= 500`

Таблица 1.3: Операции сравнения. Результат сравнения — `true` (верно) или `false` (неверно)

Осторожно

Не путай `=` и `==`! Одно равно — «положить в коробку», два равно — «сравнить». Запись `if (x = 5)` не проверит, равен ли `x` пяти, а *положит* в него 5. Это одна из самых частых ошибок.

В программе есть ещё два новых приёма. Переменная `ledState` типа `bool` помнит, горит ли светодиод. Восклицательный знак `!` означает «наоборот»: `ledState = !ledState` превращает `true` в `false` и обратно. А время хранится в `uint32_t` — `millis()` быстро вырастает до огромных чисел.

Листинг 1.3: Мигание без остановки программы

```
const int LED_PIN = 4;
const uint32_t INTERVAL = 500;  // мс
uint32_t lastToggle = 0;
bool ledState = false;

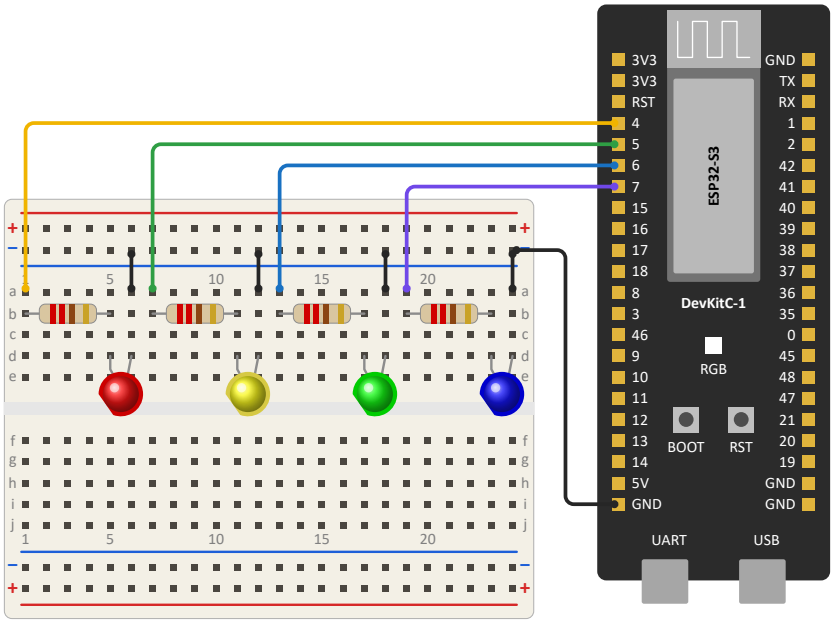
void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  uint32_t now = millis();
  if (now - lastToggle >= INTERVAL) {  // корректно и при переполнении millis()
    lastToggle = now;
    ledState = !ledState;
    digitalWrite(LED_PIN, ledState);
  }
  // здесь можно делать что-то ещё, не дожидаясь светодиода
}
```

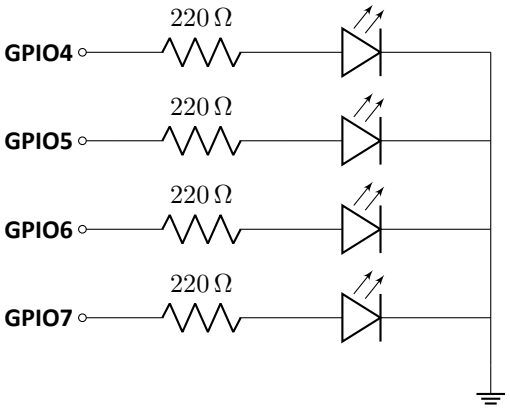
Запомни этот приём: `if (millis() - last >= INTERVAL)`. Он будет встречаться почти в каждом уроке.

1.10 Практика: бегущий огонь

Подключим четыре светодиода к выводам **GPIO4**–**GPIO7**, каждый со своим резистором, и будем зажигать их по очереди. Получится огонёк, который бежит по кругу, как на новогодней гирлянде.



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 1.10: Четыре светодиода на выводах GPIO4–GPIO7

Обрати внимание: на принципиальной схеме земли всех светодиодов соединены одной линией. На макетной плате это сделано с помощью шины «−»: все катоды подключены к ней, а она — к GND платы.

1.11 Учимся программировать: массивы и цикл for

В бегущем огне четыре светодиода, и для каждого нужно вызвать `pinMode()`. Можно написать четыре одинаковые строчки, но что, если светодиодов будет двадцать? На помощь приходят два инструмента.

Массив — это ряд пронумерованных коробок под одним именем. Номер коробки называется **индексом**, и счёт начинается **с нуля**!

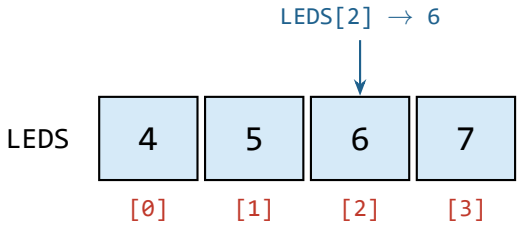


Рис. 1.11: Массив `const int LEDS[] = {4, 5, 6, 7};` — четыре ячейки с индексами от 0 до 3

Цикл `for` повторяет команды нужное число раз. У него четыре части:

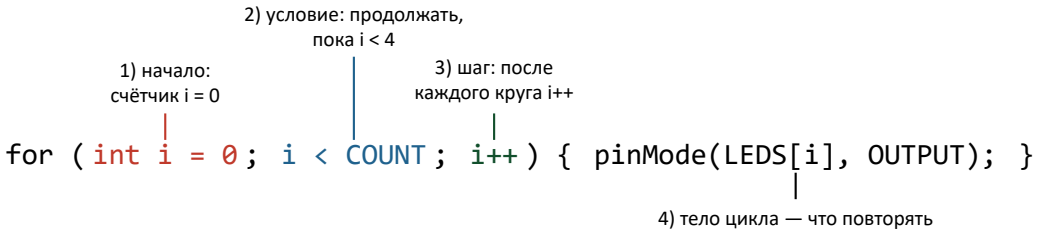
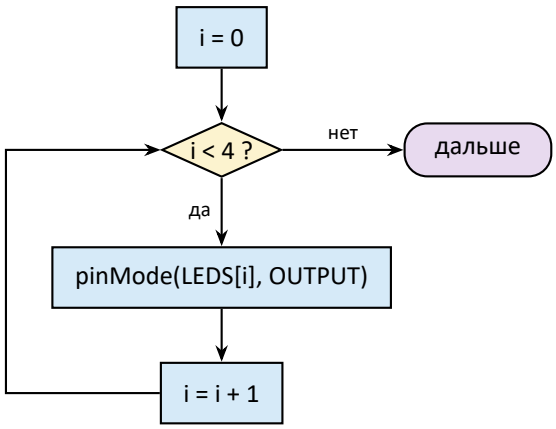


Рис. 1.12: Четыре части цикла `for`



Круг	i	LEDS[i]	Что выполнилось
1	0	4	`pinMode(4, OUTPUT)`
2	1	5	`pinMode(5, OUTPUT)`
3	2	6	`pinMode(6, OUTPUT)`
4	3	7	`pinMode(7, OUTPUT)`
—	4	—	$4 < 4$ неверно — выход

Рис. 1.13: Блок-схема цикла и «трассировка» — как меняется счётчик на каждом круге

Листинг 1.4: Бегущий огонь

```
// Бегущий огонь: четыре светодиода загораются по очереди
const int LEDS[] = {4, 5, 6, 7};            // выводы светодиодов
const int COUNT = 4;                        // сколько их
const uint32_t STEP_MS = 150;               // время горения одного, мс

int current = 0;                            // какой светодиод горит сейчас
uint32_t lastStep = 0;                      // когда было последнее переключение

void setup() {
  for (int i = 0; i < COUNT; i++) {
    pinMode(LEDS[i], OUTPUT);
  }
}

void loop() {
  if (millis() - lastStep >= STEP_MS) {     // пора переключаться?
```

```
    lastStep = millis();
    digitalWrite(LEDS[current], LOW);        // гасим текущий
    current = (current + 1) % COUNT;         // следующий: 0,1,2,3,0,1...
    digitalWrite(LEDS[current], HIGH);       // зажигаем его
  }
}
```

Цикл `for` в `setup()` настраивает все четыре вывода. В `loop()` переменная `current` хранит индекс горящего светодиода. Выражение `(current + 1) % COUNT` даёт по очереди 1, 2, 3, 0, 1, 2...: знак `%` означает остаток от деления, и после последнего светодиода счёт начинается снова с нулевого.

1.12 Встроенный RGB-светодиод

На DevKitC-1 распаян адресный светодиод WS2812 (**GPIO48** или **GPIO38** в зависимости от ревизии). Для него в Arduino-ESP32 3.x есть готовая функция `rgbLedWrite(pin, r, g, b)`:

Листинг 1.5: Смена цветов встроенного RGB-светодиода

```
const int RGB_PIN = 48;   // на ревизии v1.1 -- 38

void setup() {}

void loop() {
  rgbLedWrite(RGB_PIN, 40, 0, 0);  delay(400);  // красный (яркость 0..255)
  rgbLedWrite(RGB_PIN, 0, 40, 0);  delay(400);  // зелёный
  rgbLedWrite(RGB_PIN, 0, 0, 40);  delay(400);  // синий
  rgbLedWrite(RGB_PIN, 0, 0, 0);   delay(400);  // выключен
}
```

1.13 Самостоятельный опыт: внешний и встроенный светодиоды

Подключите LED к **GPIO4** через резистор. Затем запустите пример RGB-светодиода и измените цвета. Это два отдельных опыта: внешний LED показывает обычный GPIO, а встроенный RGB показывает адресный LED.

Если не работает

- **LED не горит** — проверьте полярность: длинная ножка (анод) — к резистору и GPIO, короткая — к GND.
- **LED горит постоянно** — номер вывода в программе не совпадает с проводом; сверьте `LED_PIN` и pinout.
- **Плата не видна в IDE** — кабель только для зарядки или выбран не тот порт; попробуйте другой кабель и разъём.

Итоги урока

- `pinMode(pin, OUTPUT)` и `digitalWrite(pin, HIGH/LOW)` — управление выходом.
- Светодиод всегда подключается через резистор; ток вывода ограничен.
- Для отсчёта времени без остановки программы используем `millis()`, а не `delay()`.

Проверь себя

1. Зачем светодиоду резистор и как рассчитать его номинал?
2. Почему `delay()` мешает делать несколько дел одновременно?

3. Почему проверка `now - last >= INTERVAL` работает и после переполнения `millis()`?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Соберите схему из урока и запустите Blink. Измените паузу так, чтобы светодиод мигал в два раза быстрее.
2. ★ Измените программу так, чтобы светодиод горел 100 мс, а гас 900 мс. Как называется отношение этих времён?
3. ★★ Замените `delay()` на проверку `millis()` по образцу урока и убедитесь, что светодиод мигает так же.
4. ★★ Включайте встроенный RGB-светодиод по очереди красным, зелёным и синим цветом, по секунде на каждый.
5. ★★★ Соберите «светофор» из трёх светодиодов с правильной последовательностью и длительностями, без `delay()`.
6. ★★★★ Сделайте мигание «сердцебиением»: две короткие вспышки и пауза, а второй светодиод одновременно мигает с частотой 3 Hz — оба без `delay()`.

Урок 2. Последовательный порт: разговор с компьютером

Цель	понять, как устроен UART, научиться выводить отладочные сообщения и принимать команды с компьютера
Опирается на	урок 1: неблокирующий цикл, светодиод на **GPIO4**
Результат	светодиод включается и выключается командами `on` / `off` из Serial Monitor
Понадобится	то же, что в уроке 1; для последнего раздела — второе устройство с UART (по желанию)

Прежде чем подключать кнопки и датчики, нужен способ видеть, что происходит внутри программы. Этим способом будет последовательный порт: через него мы будем проверять результат в каждом следующем уроке.

2.1 Кадр UART

UART — асинхронный протокол: у приёмника и передатчика нет общего тактового сигнала, поэтому обе стороны должны заранее договориться о скорости (бодах). Линия в покое находится в состоянии HIGH.



Рис. 2.1: Передача символа «A» в формате 8N1 (8 бит данных, без контроля чётности, 1 стоп-бит)

2.2 Serial и Serial Monitor

У ESP32-S3 три аппаратных UART, а в Arduino им соответствуют объекты `Serial`:

Объект Arduino	Контроллер	Выводы по умолчанию	Назначение
`Serial`	USB CDC или UART0	USB / 43, 44	связь с компьютером
`Serial0`	UART0	TX 43, RX 44	разъём «UART» на плате
`Serial1`	UART1	есть по умолчанию; лучше явно задать в `begin()`	свободен
`Serial2`	UART2	есть по умолчанию; лучше явно задать в `begin()`	свободен

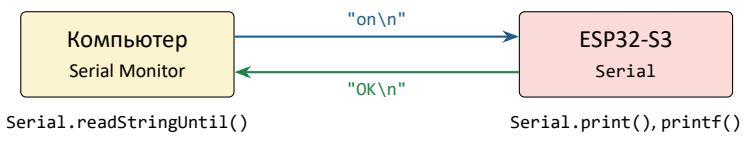


Рис. 2.2: Обмен строками между компьютером и платой

Основные функции:

- `Serial.begin(115200)` — открыть порт; скорость в Serial Monitor должна совпадать.
- `Serial.print()`, `println()`, `printf()` — отправить текст.
- `Serial.available()` — сколько байт пришло и ждёт чтения.
- `Serial.readStringUntil('\n')` — прочитать строку до символа перевода строки.

Совет

В Serial Monitor выберите окончание строки New Line: тогда каждая отправленная команда будет заканчиваться символом `\n`, и `readStringUntil()` сразу поймёт, где конец команды.

2.3 Самостоятельный опыт: команды с компьютера

Сделайте отдельный скетч с LED на GPIO4: команда on включает его, `off` выключает. Затем добавьте `status`, которая печатает текущее состояние. Главное правило: если данных нет, функция чтения сразу возвращается и `loop()` продолжает работу.

2.4 Учимся программировать: строки и цепочка условий

Команда из Serial Monitor приходит как **строка** — переменная типа `String`. Со строками можно делать много полезного:

Запись	Что делает	Пример для `cmd = "level 30"`
`cmd == "on"`	сравнить строки	`false`
`cmd.startsWith("level ")`	начинается ли с этих букв	`true`
`cmd.substring(6)`	вырезать часть, начиная с символа № 6	`"30"`
`cmd.substring(6).toInt()`	превратить текст в число	`30`
`cmd.length()`	длина строки	`8`
`cmd.trim()`	убрать пробелы и переводы строки по краям	—
`"уровень " + String(30)`	склеить строки	`"уровень 30"`

Таблица 2.1: Полезные действия со строками `String`. Символы нумеруются с нуля, как ячейки массива

Чтобы выбрать одно действие из многих, условия выстраивают в **цепочку** `if` — `else if` — `else`. Они проверяются по очереди сверху вниз, и выполняется только *первая* подходящая ветка. Последний `else` ловит всё остальное — например, опечатки.

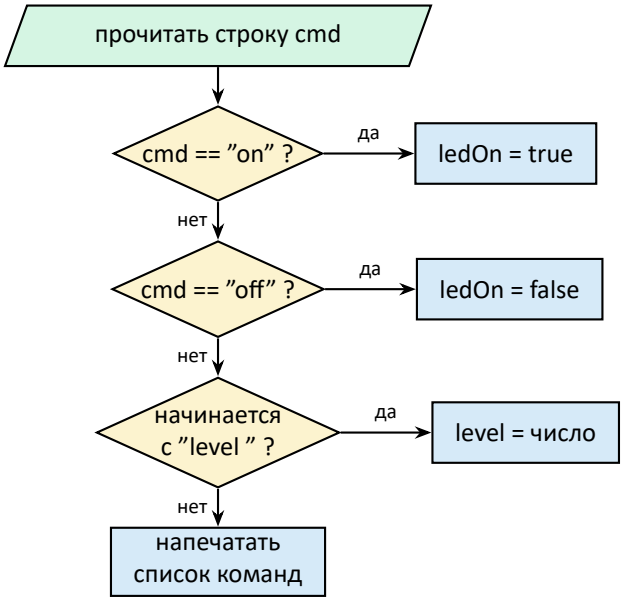


Рис. 2.3: Цепочка условий в функции `handleSerial()`: проверяем команды по очереди

Обрати внимание на `return` в начале функции: `if (!Serial.available()) return;` — «если данных нет, сразу выйти из функции». Такой ранний выход избавляет от лишних фигурных скобок.

2.5 Практика: игра «Угадай число»

Теперь, когда мы умеем читать ввод с компьютера, сделаем игру. Плата загадывает случайное число от 1 до 100, а ты угадываешь, вводя числа в поле Serial Monitor (окончание строки — New Line). Плата подсказывает «мало» или «много». Светодиод для игры не нужен.

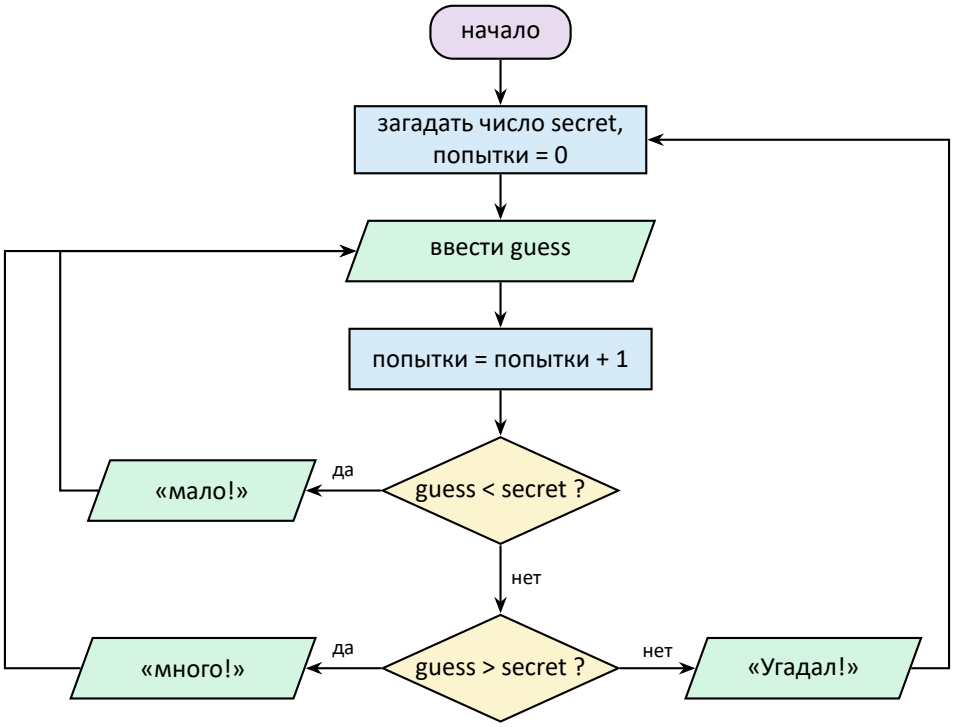


Рис. 2.4: Блок-схема игры «Угадай число»

Листинг 2.1: Игра «Угадай число»

```
// Игра «Угадай число»: плата загадывает число от 1 до 100
int secret;                                  // загаданное число
int tries;                                   // сколько было попыток

void newGame() {
  secret = random(1, 101);                   // случайное от 1 до 100
  tries = 0;
  Serial.println("Я загадала число от 1 до 100. Угадай!");
}

int readNumber() {                           // ждём, пока человек введёт число
  while (Serial.available() == 0) {
    delay(10);
  }
  int n = Serial.parseInt();                 // прочитать число из строки
  while (Serial.available() > 0) Serial.read();   // выбросить остаток строки
  return n;
}

void setup() {
  Serial.begin(115200);
  delay(500);
  randomSeed(esp_random());                  // чтобы числа каждый раз были разными
  newGame();
}

void loop() {
  int guess = readNumber();
  tries++;
  if (guess < secret) {
    Serial.printf("%d -- мало!\n", guess);
  } else if (guess > secret) {
    Serial.printf("%d -- много!\n", guess);
  } else {
    Serial.printf("Угадал за %d попыток! Сыграем ещё.\n", tries);
    newGame();
  }
}
```

Здесь встретилось всё, что мы уже знаем: глобальные **переменные** `secret` и `tries`, **цепочка условий**, цикл `while`, который ждёт ввода, и **свои функции** `newGame()` и `readNumber()`. Новые готовые функции: `random(1, 101)` даёт случайное число от 1 до 100, а `Serial.parseInt()` читает из пришедших символов целое число.

Это интересно

Если угадывать с умом — каждый раз называть середину оставшегося промежутка (50, потом 25 или 75…), — любое число от 1 до 100 угадывается не больше чем за 7 попыток, ведь $2^7 = 128 > 100$. Точно так же работает АЦП из урока «Сигналы»: он «угадывает» напряжение за 12 шагов. Этот способ называется **двоичным поиском**.

2.6 Второй UART: связь с другими устройствами

Тот же протокол используют GPS-модули, модемы и другие микроконтроллеры. Благодаря матрице GPIO UART1 и UART2 можно вывести почти на любые выводы.

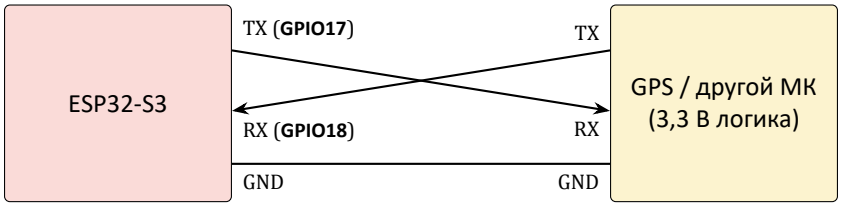


Рис. 2.5: TX одного устройства соединяется с RX другого; земли объединяются

Листинг 2.2: Мост: всё, что пришло с модуля, пересылается на компьютер, и наоборот

```
void setup() {
  Serial.begin(115200);                          // к компьютеру
  Serial1.begin(9600, SERIAL_8N1, 18, 17);       // RX=18, TX=17 к модулю
}

void loop() {
  while (Serial1.available()) Serial.write(Serial1.read());
  while (Serial.available())  Serial1.write(Serial.read());
}
```

Осторожно

Модули с 5-вольтовой логикой (часть Arduino, некоторые GPS) нельзя подключать к RX ESP32-S3 напрямую — используйте делитель или преобразователь уровней.

Если не работает

- **Вместо текста «кракозябры»** — скорость Serial Monitor не 115200.
- **Ничего не выводится через разъём USB** — включите Tools → USB CDC On Boot: Enabled.
- **Команда не срабатывает** — выберите окончание строки New Line и уберите лишние пробелы через `trim()`.

Итоги урока

- UART передаёт байты кадрами: старт-бит, 8 бит данных, стоп-бит; скорости сторон должны совпадать.
- `Serial` — главный инструмент отладки: будем пользоваться им во всех следующих уроках.
- Команды читаем неблокирующе: проверяем `available()` и только тогда читаем строку.

Проверь себя

1. Почему UART называют асинхронным и что будет, если скорости сторон разные?
2. Как соединяют TX и RX двух устройств и зачем нужна общая земля?
3. Зачем перед чтением проверяют `Serial.available()`?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выведите в Serial Monitor строку «Привет, ESP32-S3!» и время работы платы в секундах.

2. ★ Переключите скорость порта на 9600 в программе, но оставьте 115200 в мониторе. Что вы увидите? Верните настройки.
3. ★★ Добавьте команды `on` и `off` для светодиода и ответ «OK» на каждую.
4. ★★ Добавьте команду `blink N`: мигнуть светодиодом N раз без `delay()`.
5. ★★★ Выводите раз в 10 секунд строку со временем работы, не мешая приёму команд, и добавьте команду `help` со списком команд.
6. ★★★★ Соедините две платы по UART (TX–RX крест-накрест, общая земля) и управляйте светодиодом одной платы командами с другой.

Урок 3. Цифровой вход: кнопки и прерывания

Цель	научиться читать состояние кнопки, понять роль подтягивающих резисторов, побороть дребезг, познакомиться с прерываниями
Опирается на	урок 1 (выходы, `millis()`), урок 2 (вывод в Serial для проверки)
Результат	кнопка на **GPIO5** включает и выключает светодиод
Понадобится	тактовая кнопка

3.1 Сначала: что делает кнопка

Кнопка не посылает команду: она соединяет контакты. Один контакт идёт в GND, другой — в GPIO5. Встроенная подтяжка держит GPIO5 в HIGH, пока кнопка отпущена; нажатие соединяет GPIO5 с GND, и программа читает LOW. `bool` хранит true/false, `if` проверяет условие, `digitalRead` считывает сигнал.

3.2 Проблема «висящего» входа

Вход GPIO имеет очень высокое сопротивление. Если к нему подключить только кнопку на землю, то в отпущенном состоянии вывод никуда не подключён и ловит наводки — читается случайно `0` или `1`. Решение — **подтягивающий резистор**, задающий уровень по умолчанию.

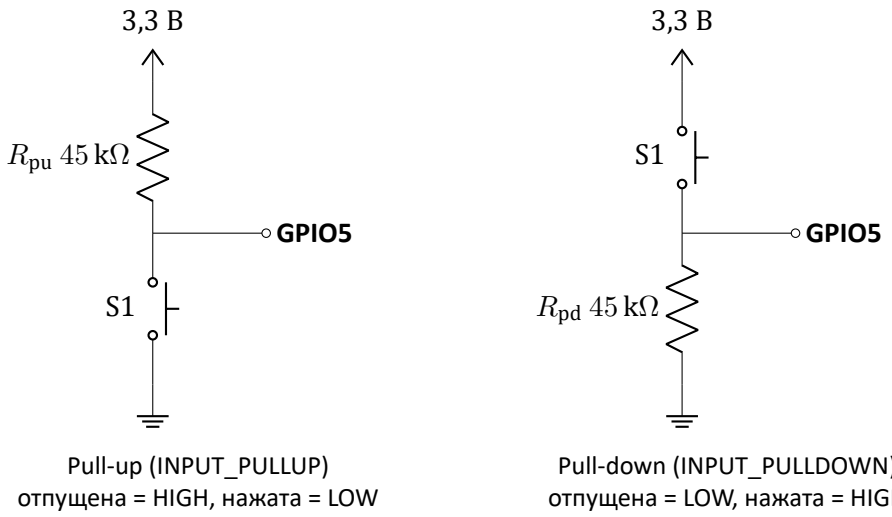


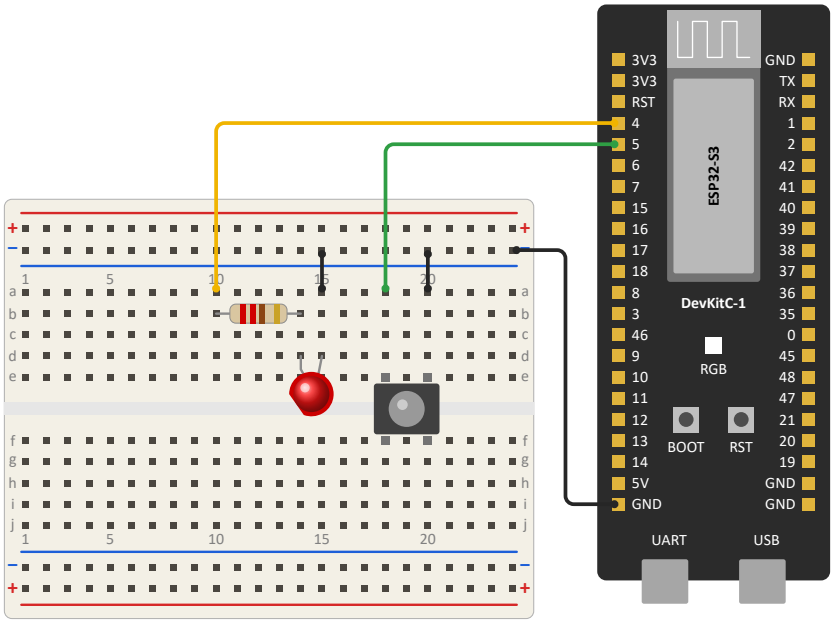
Рис. 3.1: Подтяжка к питанию и к земле. У ESP32-S3 оба резистора встроены ($\approx 45\,\text{k}\Omega$). В опытах используем pull-up: кнопка замыкает **GPIO5** на землю

Совет

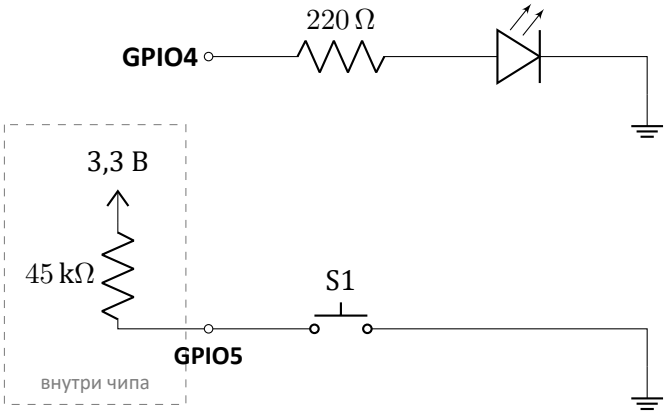
Встроенная подтяжка слабая. Для длинных проводов или в условиях помех добавьте внешний резистор $4{,}7 – 10\,\text{k}\Omega$.

3.3 Схема подключения

Добавим к светодиоду на **GPIO4** кнопку на **GPIO5**. Внешний резистор для кнопки не нужен: включим встроенную подтяжку командой `pinMode(5, INPUT_PULLUP)`.



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 3.2: Кнопка на GPIO5 и светодиод на GPIO4

На макетной плате кнопка стоит поперёк канавки. Зелёный провод от **GPIO5** подключён к одной её стороне, чёрный провод от другой стороны ведёт на шину земли.

3.4 Опрос кнопки

Листинг 3.1: Состояние кнопки в Serial Monitor

```
const int BTN_PIN = 5;

void setup() {
  Serial.begin(115200);
  pinMode(BTN_PIN, INPUT_PULLUP);   // включаем встроенную подтяжку
}

void loop() {
  bool pressed = (digitalRead(BTN_PIN) == LOW);  // pull-up: нажатие = LOW
  Serial.println(pressed ? "нажата" : "отпущена");
  delay(200);
}
```

3.5 Проверка до следующего шага

В Serial Monitor отпущенная кнопка должна давать `HIGH`, нажатая — `LOW`. Если числа меняются сами, это не «плохая программа»: вероятнее всего, нет общего GND или не включена подтяжка. Сначала добейтесь стабильного чтения, и лишь потом добавляйте переключение светодиода. Если просто написать «при `LOW` переключить светодиод», он будет мигать всё время, пока кнопка удерживается, и срабатывать по нескольку раз на одно нажатие. Нам нужно ловить именно *момент нажатия* — и при этом учесть дребезг.

3.6 Дребезг контактов

Механические контакты при замыкании несколько раз «подпрыгивают». Микроконтроллер, опрашивающий вход миллионы раз в секунду, видит это как серию нажатий.

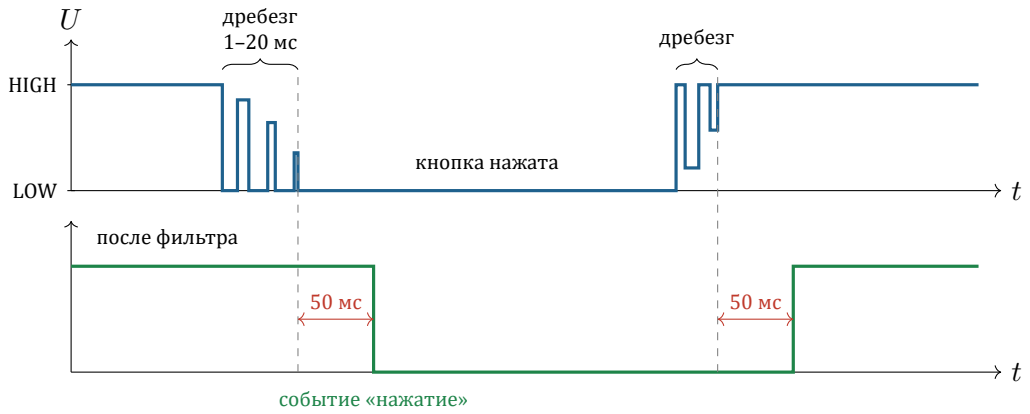


Рис. 3.3: Дребезг контактов (сверху) и сигнал после программного фильтра (снизу)

Программный фильтр принимает новый уровень, только если он стабилен дольше 50 ms. Оформим его в функцию, которая возвращает `true` ровно один раз на каждое нажатие:

Листинг 3.2: Функция buttonPressed(): нажатие с подавлением дребезга

```
const int BTN_PIN = 5;
const uint32_t DEBOUNCE_MS = 50;

bool buttonPressed() {
  static bool stable = digitalRead(BTN_PIN);  // подтверждённое состояние
  static bool last = stable;                  // последнее прочитанное
  static uint32_t changed = 0;                // когда уровень менялся
  bool r = digitalRead(BTN_PIN);
  if (r != last) {                            // уровень дёрнулся --
    last = r;                                 // перезапускаем отсчёт
    changed = millis();
  }
  if (millis() - changed > DEBOUNCE_MS && r != stable) {
    stable = r;                               // уровень устоялся
    return stable == LOW;                     // событие -- только нажатие
  }
  return false;
}

void setup() {
  Serial.begin(115200);
  pinMode(BTN_PIN, INPUT_PULLUP);
}
```

```
24
25 void loop() {
26   if (buttonPressed()) Serial.println("Нажатие!");
27 }
```

3.7 Учимся программировать: логические операции и `static`

В функции `buttonPressed()` встретилось условие посложнее: `millis() - changed > DEBOUNCE_MS && r != stable`. Оно состоит из двух вопросов, соединённых **логической операцией**. Таких операций три:

Запись	Название	Когда верно
`A && B`	«и»	только если верны *оба*: «прошло 50 мс *и* уровень изменился»
`A || B`	«или»	если верно *хотя бы одно*: «слишком холодно *или* слишком жарко»
`!A`	«не»	если `A` неверно: `!ledOn` — «светодиод не горит»

Таблица 3.1: Логические операции

A && B

A	B	результат
false	false	false
false	true	false
true	false	false
true	true	true

A || B

A	B	результат
false	false	false
false	true	true
true	false	true
true	true	true

Рис. 3.4: Таблицы истинности: результат «и» и «или» для всех сочетаний

Второе новое слово — `static`. Обычная локальная переменная создаётся заново при каждом вызове функции и забывает старое значение. А переменная со словом `static` создаётся один раз и *помнит* значение между вызовами — как блокнот, который функция носит с собой. Именно поэтому `buttonPressed()` помнит, каким был уровень кнопки в прошлый раз.

3.8 Практика: кнопка-переключатель яркости

Каждое нажатие переключает светодиод на следующую ступень: выключен → 30 % → 100 % → выключен. Так работают многие настольные лампы.

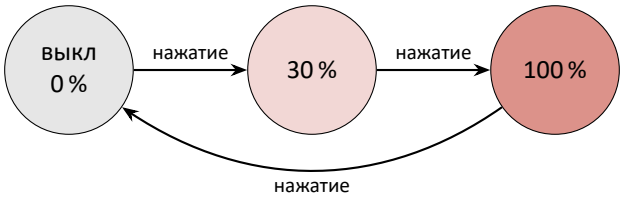


Рис. 3.5: Диаграмма состояний: кружки — состояния, стрелки — что их меняет

Листинг 3.3: Кнопка переключает яркость

```
// Кнопка переключает яркость: выкл -> 30 % -> 100 % -> выкл ...
const int LED_PIN = 4;
const int BTN_PIN = 5;
const int LEVELS[] = {0, 30, 100};           // ступени яркости, %

int mode = 0;                                // номер текущей ступени

uint32_t gammaDuty(int percent) {
  return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
}

bool buttonPressed() {                       // см. выше: с защитой от дребезга
  static bool stable = digitalRead(BTN_PIN);
  static bool last = stable;
  static uint32_t changed = 0;
  bool r = digitalRead(BTN_PIN);
  if (r != last) { last = r; changed = millis(); }
  if (millis() - changed > 50 && r != stable) {
    stable = r;
    return stable == LOW;
  }
  return false;
}

void setup() {
  Serial.begin(115200);
  pinMode(BTN_PIN, INPUT_PULLUP);
  ledcAttach(LED_PIN, 5000, 8);
  ledcWrite(LED_PIN, 0);
}

void loop() {
  if (buttonPressed()) {
    mode = (mode + 1) % 3;                   // 0 -> 1 -> 2 -> 0
    ledcWrite(LED_PIN, gammaDuty(LEVELS[mode]));
    Serial.printf("Яркость: %d%%\n", LEVELS[mode]);
  }
}
```

3.9 Самостоятельный опыт: кнопка-переключатель

Подключите кнопку между **GPIO5** и GND. Используйте `buttonPressed()` из примера выше, чтобы каждое нажатие меняло состояние LED на GPIO4 ровно один раз.
Добавьте сюда команды `on`/`off` из урока 2: светодиодом можно управлять и кнопкой, и командами — оба способа работают одновременно, потому что ни одна функция в `loop()` не останавливает программу.

3.10 Прерывания

Опрос в `loop()` работает, пока цикл крутится быстро. Если программа занята, короткое нажатие можно пропустить. Альтернатива — попросить процессор *самого* вызвать функцию в момент изменения уровня.

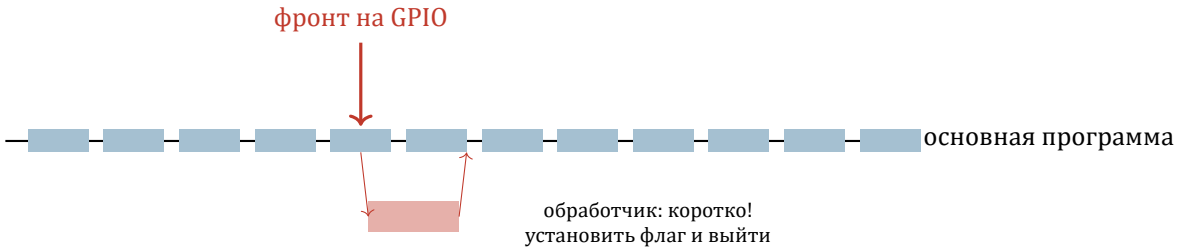


Рис. 3.6: Прерывание приостанавливает основную программу на время обработчика

Листинг 3.4: Подсчёт нажатий через прерывание

```
const int BTN_PIN = 5;

volatile uint32_t pressCount = 0;   // volatile: меняется в обработчике
volatile uint32_t lastIsrTime = 0;

void IRAM_ATTR onButton() {         // IRAM_ATTR: код в быстрой памяти
  uint32_t now = millis();
  if (now - lastIsrTime > 50) {     // грубое подавление дребезга
    pressCount++;
    lastIsrTime = now;
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(BTN_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BTN_PIN), onButton, FALLING);
}

void loop() {
  static uint32_t shown = 0;
  if (pressCount != shown) {        // печатаем здесь, а не в обработчике
    shown = pressCount;
    Serial.printf("Нажатий: %lu\n", shown);
  }
}
```

Осторожно

Внутри обработчика прерывания (ISR) нельзя вызывать `delay()`, `Serial.print()` и другие медленные/блокирующие функции. Правило: изменить переменную — и выйти. Всю работу делает основная программа.

Для обычной кнопки опроса достаточно, поэтому в опытах используем `buttonPressed()`. К прерываниям вернёмся в уроке 7, когда научимся будить из них задачи.

3.11 Strapping-выводы

Уровни на **GPIO0**, **GPIO3**, **GPIO45**, **GPIO46** считываются в момент сброса и определяют режим загрузки и напряжение Flash. Не вешайте на них кнопки или нагрузку, которые могут изменить уровень при старте. Именно поэтому кнопку в опытах подключаем к **GPIO5**.

Если не работает

- **Значения прыгают сами** — нет подтяжки (`INPUT_PULLUP`) или общего GND.
- **Одно нажатие считается несколько раз** — увеличьте `DEBOUNCE_MS` до 50–80 мс.
- **Плата не стартует с подключённой кнопкой** — кнопка висит на strapping-выводе; перенесите её на **GPIO5**.

Итоги урока

- Вход без подтяжки «висит»; используем `INPUT_PULLUP` и кнопку на землю.
- Нажатие нужно ловить как *событие* и фильтровать дребезг ($20 - 50$ ms).
- Прерывание реагирует мгновенно, но обработчик должен быть минимальным.

Проверь себя

1. Что такое «висящий» вход и как его исправляет подтяжка?
2. Почему одно нажатие может засчитаться несколько раз?
3. Что можно и чего нельзя делать в обработчике прерывания?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Соберите кнопку с `INPUT_PULLUP` и выведите в Serial Monitor «нажата» и «отпущена».
2. ★ Используйте кнопку BOOT (**GPIO0**) как вторую кнопку после загрузки: выводите её состояние.
3. ★★ Сделайте кнопку-переключатель: каждое нажатие меняет состояние светодиода. Добавьте защиту от дребезга.
4. ★★ Посчитайте число нажатий и выводите его в Serial Monitor. Проверьте, что за одно нажатие счётчик растёт ровно на единицу.
5. ★★★ Различайте короткое (меньше 0,5 с) и длинное (больше 1 с) нажатие: короткое переключает светодиод, длинное печатает время работы платы.
6. ★★★★ Замените кнопку сенсорным входом (`touchRead(2)`), подберите порог и сделайте то же самое переключение прикосновением.

Урок 4. ШИМ (LEDC): плавная яркость

Цель	понять принцип широтно-импульсной модуляции и научиться задавать «промежуточные» уровни
Опирается на	урок 1 (светодиод), урок 2 (команды), урок 3 (кнопка)
Результат	яркость светодиода задаётся в процентах 0–100 %, и её изменение выглядит равномерным
Понадобится	то же, что раньше; для раздела про серво — сервопривод SG90 (по желанию)

4.1 Сначала: откуда берётся промежуточная яркость

GPIO знает только HIGH и LOW. Если включать LED очень быстро, глаз усредняет вспышки: короткий HIGH кажется тусклым, длинный — ярким. Это ШИМ. Мы меняем не напряжение, а долю включённого времени. Частота — число повторов в секунду; duty — доля включённого времени; `ledcWrite` задаёт её.

4.2 Что такое ШИМ

`digitalWrite()` умеет только «полностью включено» и «полностью выключено». ESP32-S3 не имеет ЦАП, но может «имитировать» промежуточное напряжение, быстро включая и выключая вывод. Доля времени во включённом состоянии называется **коэффициентом заполнения** (duty cycle):

$$D = \frac{t_{\text{вкл}}}{T}, \qquad U_{\text{ср}} = D \cdot 3{,}3\,\text{V}.$$

При частоте в тысячи герц глаз не замечает мигания и видит среднюю яркость.

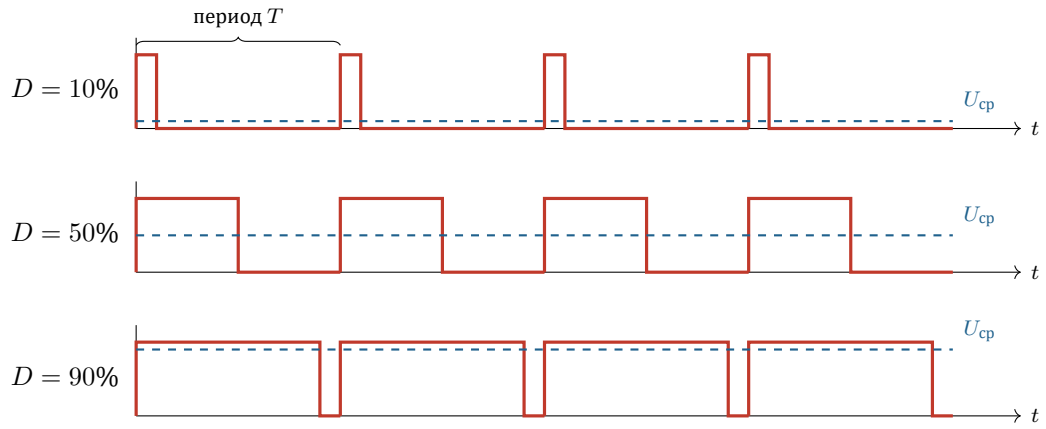


Рис. 4.1: ШИМ с разным коэффициентом заполнения

4.3 Периферия LEDC

В ESP32-S3 ШИМ формирует модуль **LEDC**: 8 каналов, 4 таймера, разрешение до 14 бит. Частота и разрешение связаны: чем выше частота, тем меньше доступное разрешение ($f_{\text{max}} \approx 80\,\text{MHz}/2^{\text{bits}}$).

Применение	Частота	Разрешение
Светодиоды	5 kHz	8–13 бит
Сервопривод	50 Hz	14 бит
Двигатель через драйвер	10 – 20 kHz	10 бит
Звук (пищалка)	100 – 5000 Hz	8 бит

В Arduino-ESP32 3.x канал выбирается автоматически, а обращаются к нему по номеру вывода: `ledcAttach(pin, freq, bits)` заменяет `pinMode()`, `ledcWrite(pin, duty)` — `digitalWrite()`.

Листинг 4.1: Плавное «дыхание» светодиода

```
const int LED_PIN = 4;
const uint32_t FREQ = 5000;   // Гц
const uint8_t  RES  = 8;      // бит -> значения 0..255

void setup() {
  ledcAttach(LED_PIN, FREQ, RES);
}

void loop() {
  for (int d = 0; d <= 255; d++) { ledcWrite(LED_PIN, d); delay(4); }
  for (int d = 255; d >= 0; d--) { ledcWrite(LED_PIN, d); delay(4); }
}
```

4.4 Что наблюдать в примере

Число в `ledcWrite` не является «вольтами». Это доля включения: 0 означает всегда LOW, большое число — почти всегда HIGH. Меняйте шаг изменения яркости с 1 на 10: дыхание станет ступенчатым. Так можно увидеть разницу между плавной регулировкой и грубым переключением.

4.5 Яркость и восприятие: гамма-коррекция

Запустив пример, вы заметите: светодиод быстро становится ярким, а дальше почти не меняется. Глаз воспринимает яркость нелинейно, поэтому «50 % для человека» соответствует гораздо меньшему коэффициенту заполнения. Исправляет это гамма-коррекция: $D = D_{\max} \cdot (x/100)^{2{,}2}$.

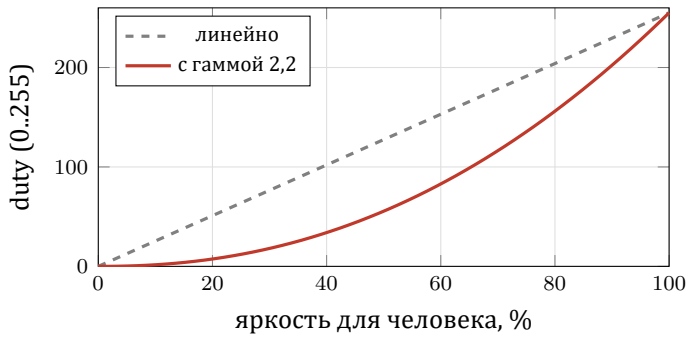


Рис. 4.2: Перевод «человеческой» яркости в коэффициент заполнения

4.6 Самостоятельный опыт: яркость LED

Сделайте отдельный скетч: Serial Monitor принимает `level N`, где N от 0 до 100, и меняет яркость LED. Проверьте 10, 50 и 100 процентов; затем добавьте проверку неверного числа.

4.7 Учимся программировать: свои функции

Формулу гамма-коррекции придётся использовать много раз: для каждого значения яркости. Чтобы не переписывать её снова и снова, оформим её как **свою функцию**. Функция — это «машинка»: у неё есть имя, на вход ей передают данные (**параметры**), а на выходе она **возвращает** результат. Написал один раз — пользуешься сколько угодно, как новым словом в языке.

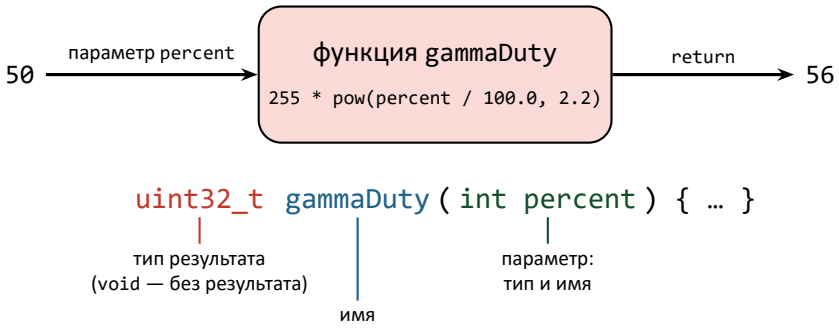


Рис. 4.3: Функция получает яркость в процентах и возвращает значение duty: 50 % → 56

- Слово `return` («вернуть») отдаёт результат туда, откуда функцию вызвали. Поэтому `ledcWrite(LED_PIN, gammaDuty(p))` сначала вычислит `gammaDuty(p)`, а потом подставит результат.
- Функции `setup()` и `loop()` имеют тип `void` («пустота»): они ничего не возвращают.
- Почему в формуле `100.0`, а не `100`? Помнишь целочисленное деление из урока 0? `50 / 100` даст 0, а `50 / 100.0` — правильные 0,5.

Совет

Хорошая функция делает *одно* дело, и её имя говорит, какое: `gammaDuty()`, `blinkTwice()`, `isButtonPressed()`. Тогда программа читается почти как обычный текст.

Листинг 4.2: «Дыхание» с гамма-коррекцией

```
// «Дыхание» с поправкой на глаз (гамма-коррекция)
const int LED_PIN = 4;

uint32_t gammaDuty(int percent) {            // 0..100 % -> 0..255
  return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
}

void setup() {
  ledcAttach(LED_PIN, 5000, 8);
}

void loop() {
  for (int p = 0; p <= 100; p++)  { ledcWrite(LED_PIN, gammaDuty(p)); delay(10); }
  for (int p = 100; p >= 0; p--)  { ledcWrite(LED_PIN, gammaDuty(p)); delay(10); }
}
```

Функция `pow(x, y)` возводит x в степень y. Сравни два примера: теперь светодиод разгорается и гаснет плавно по всей длине.

Это интересно

Глаз способен видеть и при свете луны, и на солнечном пляже, где света в миллион раз больше. Такой огромный диапазон возможен как раз потому, что зрение реагирует на *отношения*, а не на разницу. Слух устроен так же, поэтому громкость звука измеряют в децибелах — логарифмической шкале. Этот закон восприятия называют законом Вебера–Фехнера.

4.8 Какую частоту ШИМ выбрать

Частота и разрешение ШИМ связаны: блок LEDC работает от внутреннего «метронома» 80 МГц, и за один период ШИМ он должен успеть отсчитать все ступеньки. Чем выше частота, тем меньше ступенек помещается в период.

Применение	Частота	Разрешение
Светодиоды	5 kHz	8–13 бит
Сервопривод (урок 5)	50 Hz	14 бит
Моторчик через драйвер	10 – 20 kHz	10 бит
Звук (пищалка)	100 – 5000 Hz	8 бит

Таблица 4.1: Типичные настройки ШИМ

Совет

Если выбрать для светодиода частоту всего 50–100 Гц, глаз не заметит мерцания, если смотреть прямо. Но стоит быстро перевести взгляд или снять светодиод на камеру телефона — появятся полосы и «пунктир». Поэтому для света берут тысячи герц.

4.9 Практика: плавная волна

Вернёмся к схеме из четырёх светодиодов (рис. 1.10). Теперь у каждого будет свой ШИМ, и яркость будет меняться по синусоиде — той самой волне из урока «Сигналы». Если сдвинуть волну каждого следующего светодиода на четверть периода, по ряду побежит мягкая световая волна.

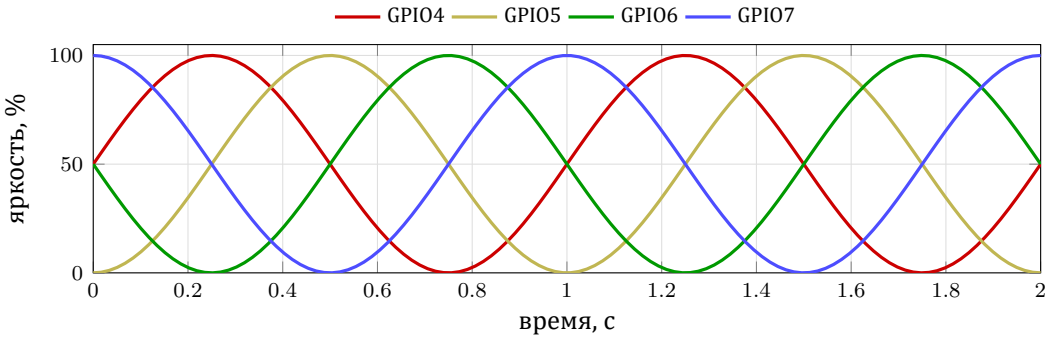


Рис. 4.4: Яркость четырёх светодиодов: одна и та же волна, сдвинутая на четверть периода

Листинг 4.3: Световая волна

```
// Плавная «волна» на четырёх светодиодах
const int LEDS[] = {4, 5, 6, 7};
const int COUNT = 4;

uint32_t gammaDuty(int percent) {
  return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
}

void setup() {
  for (int i = 0; i < COUNT; i++) {
    ledcAttach(LEDS[i], 5000, 8);            // каждому светодиоду -- свой ШИМ
  }
}

void loop() {
  float t = millis() / 1000.0;               // время в секундах
  for (int i = 0; i < COUNT; i++) {
    // синусоида, сдвинутая для каждого светодиода на четверть периода
    float s = sin(2 * PI * (t - i * 0.25));  // от -1 до 1
    int percent = (s + 1) * 50;              // от 0 до 100
    ledcWrite(LEDS[i], gammaDuty(percent));
  }
  delay(10);
}
```

4.10 Другое применение ШИМ: сервопривод

Этот раздел показывает, что тот же LEDC умеет больше, чем яркость. Хобби-сервопривод ожидает импульсы с периодом 20 ms (50 Hz); длительность импульса задаёт угол.

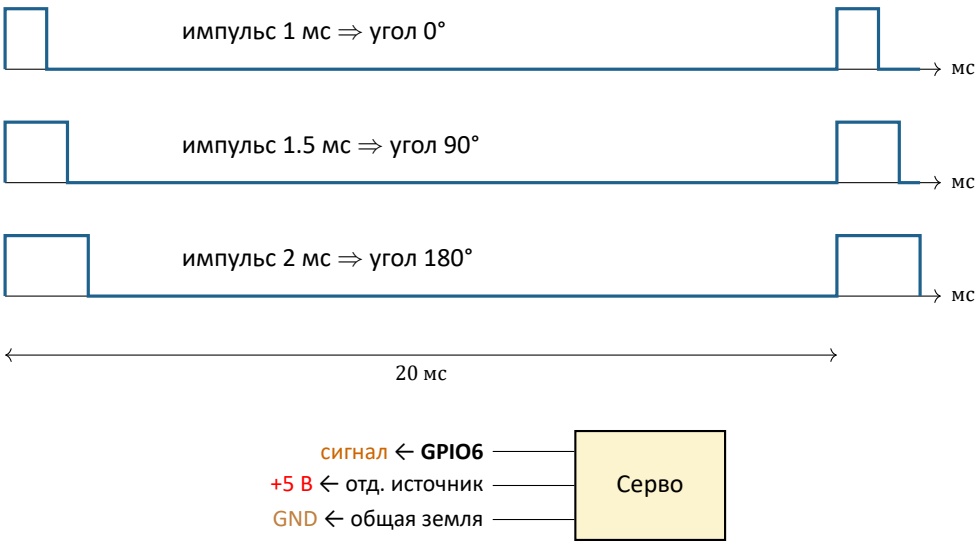


Рис. 4.5: Управляющие импульсы и подключение сервопривода

Листинг 4.4: Управление сервоприводом через LEDC

```
const int SERVO_PIN = 6;
const uint8_t RES = 14;                    // 0..16383 на период 20 мс

uint32_t usToDuty(uint32_t us) {           // микросекунды -> значение duty
  return (uint32_t)((uint64_t)us * ((1 << RES) - 1) / 20000);
}

void servoWrite(int angle) {               // 0..180 градусов
  uint32_t us = map(angle, 0, 180, 1000, 2000);  // у многих серво 500..2500
  ledcWrite(SERVO_PIN, usToDuty(us));
}

void setup() {
  ledcAttach(SERVO_PIN, 50, RES);
}

void loop() {
  for (int a = 0; a <= 180; a += 2) { servoWrite(a); delay(20); }
  for (int a = 180; a >= 0; a -= 2) { servoWrite(a); delay(20); }
}
```

Осторожно

Не питайте сервопривод от вывода 3V3 платы: пусковые токи достигают сотен миллиампер и вызывают перезагрузку ESP32. Используйте отдельный источник 5 V, объединив земли.

4.11 То же на ESPUI: ползунок яркости на телефоне

Гамма-коррекцию лучше всего «почувствовать» руками: подвигать ползунок и сравнить, как светит светодиод с поправкой и без неё. Для этого не нужна кнопка и Serial Monitor: плата сама создаёт панель, а телефон показывает ползунок, переключатель и текущий коэффициент заполнения. Как собирать такие панели, объясняет урок 39; сейчас достаточно прошить скетч и открыть страницу.

Листинг 4.5: ESPUI: яркость светодиода с переключаемой гаммой

```
// LED: GPIO4 -> 220 Ом -> LED -> GND. Панель: ползунок яркости и переключатель гаммы.
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

const int LED_PIN = 4;
std::atomic<int> wantLevel{0};             // сюда пишет callback, читает loop()
std::atomic<bool> gammaOn{true};
uint16_t dutyId;

uint32_t gammaDuty(int percent) {          // как в уроке 4: 0..100 % -> 0..255
  return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
}

void levelCallback(Control* sender, int event) {
  if (event != SL_VALUE) return;
  wantLevel = constrain(sender->value.toInt(), 0, 100);   // ползунок и так 0..100, но
↪  проверяем
```

```
}

void gammaCallback(Control*, int event) {
  if (event == S_ACTIVE) gammaOn = true;
  if (event == S_INACTIVE) gammaOn = false;
}

void setup() {
  Serial.begin(115200);
  if (!ledcAttach(LED_PIN, 5000, 8)) { Serial.println("PWM init failed"); return; }
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Gamma", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  ESPUI.sliderContinuous = false;
  auto level = ESPUI.slider("Яркость для глаза, %", levelCallback, ControlColor::Peterriver,
↪  0, 0, 100);
  ESPUI.switcher("Гамма-коррекция", gammaCallback, ControlColor::Emerald, true);
  dutyId = ESPUI.label("Коэффициент заполнения (0..255)", ControlColor::Turquoise, "0");
  (void)level;
  ESPUI.begin("ESP32-S3 gamma");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  int percent = wantLevel;
  uint32_t duty = gammaOn ? gammaDuty(percent) : percent * 255 / 100;
  ledcWrite(LED_PIN, duty);               // аппаратную часть трогает только loop()

  static uint32_t last = 0;
  if (millis() - last >= 300) {           // показываем применённое значение
    last = millis();
    ESPUI.updateLabel(dutyId, String(duty));
  }
  delay(5);
}
```

Подключите телефон к сети `S3-Gamma` (пароль `lesson-39-demo`) и откройте адрес из Serial Monitor. Установите ползунок на 50 % и переключайте «Гамма-коррекция»: коэффициент заполнения упадёт примерно с 127 до 28, а светодиод при этом светит заметно похоже на «половину». Обратите внимание на устройство программы: callback только записывает число в переменную, а `ledcWrite()` вызывает `loop()`.

Если не работает

- **Яркость не меняется** — после `ledcAttach()` не вызывайте для этого вывода `pinMode()` и `digitalWrite()`.
- **Заметно мерцание (особенно на камере телефона)** — поднимите частоту ШИМ до 5–20 кГц.
- **Сервопривод дёргается, плата перезагружается** — питайте серво от отдельного источника 5 В с общей землёй.

Итоги урока

- ШИМ задаёт среднее напряжение долей времени включения; на ESP32-S3 его формирует LEDC.

- `ledcAttach()` + `ledcWrite()` вместо `pinMode()` + `digitalWrite()`.
- Для света используем гамма-коррекцию, иначе регулировка кажется неравномерной.

Проверь себя

1. Что такое коэффициент заполнения и как он связан со средним напряжением?
2. Как связаны частота ШИМ и разрешение LEDC?
3. Зачем нужна гамма-коррекция?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите пример и убедитесь, что яркость светодиода меняется плавно. Измените шаг так, чтобы нарастание шло вдвое медленнее.
2. ★ Установите яркость 25, 50 и 100% и проверьте мультиметром среднее напряжение на выводе. Совпало ли оно с расчётом?
3. ★★ Сделайте включение светодиода плавным: `ledcFade(pin, from, to, ms)` выполняет переход аппаратно.
4. ★★ Добавьте короткий «щелчок» пищалки при нажатии кнопки: `ledcWriteTone(pin, 2000)` на 30 ms.
5. ★★★ Подключите внешний RGB-светодиод к трём каналам LEDC и управляйте им командой `color R G B`.
6. ★★★★ Подключите сервопривод (с отдельным питанием 5 V) и управляйте углом от 0 до 180° командами из Serial Monitor, ограничив диапазон безопасными импульсами.

Урок 5. АЦП: читаем аналоговые сигналы

Цель	научиться измерять напряжение, понять ограничения АЦП ESP32-S3, измерять напряжения выше 3,3 В
Опирается на	урок 2 (вывод результатов в Serial), урок 4 (яркость через ШИМ)
Результат	потенциометр управляет яркостью светодиода
Понадобится	потенциометр $10\,\text{k}\Omega$; для раздела про делитель — резисторы $100\,\text{k}\Omega$ и $33\,\text{k}\Omega$

5.1 Сначала: сигнал не всегда только 0 или 1

Потенциометр делит 3,3 В между GND и GPIO: в середине получается около 1,65 В. АЦП измеряет это напряжение и возвращает число. Так программа понимает, насколько повернули ручку. `int` хранит целое число; `analogReadMilliVolts` читает милливольты; 1000 мВ = 1 В.

5.2 Как работает АЦП

АЦП последовательного приближения (SAR) преобразует напряжение в число. При разрешении 12 бит диапазон разбивается на $2^{12} = 4096$ уровней:

$$N = \left\lfloor \frac{U_{\text{вх}}}{U_{\text{полн}}} \cdot 4095 \right\rfloor .$$

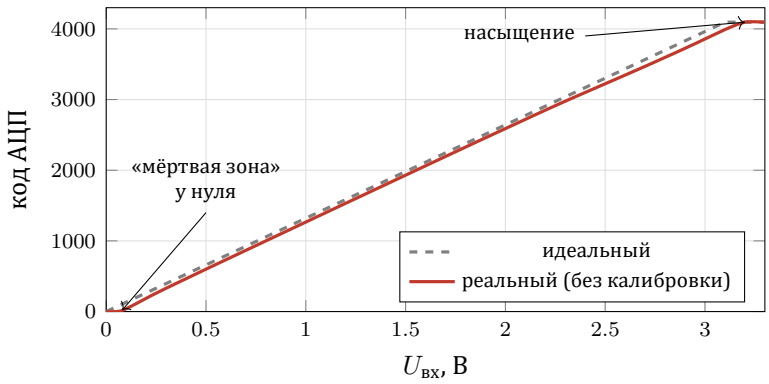


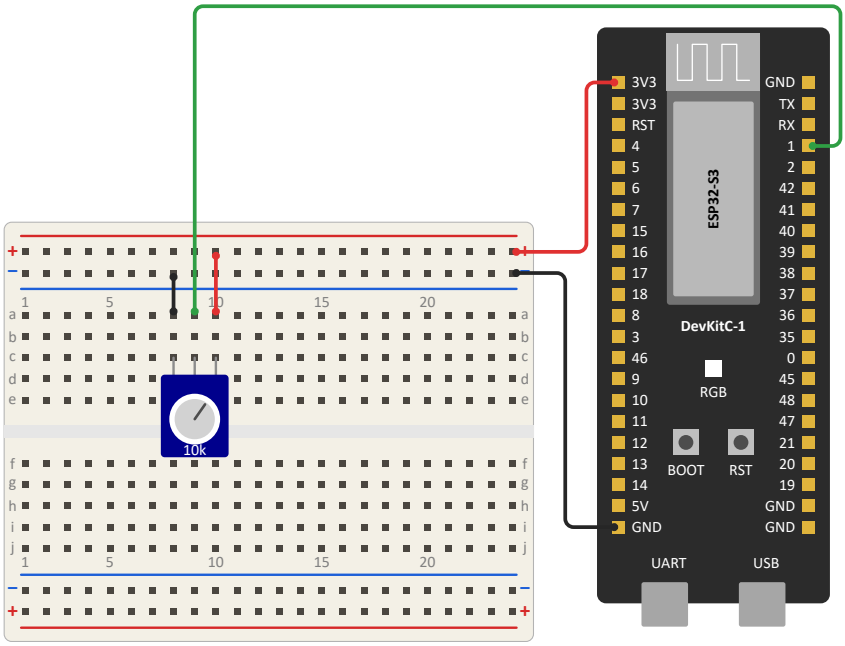
Рис. 5.1: Характеристика АЦП (иллюстрация). Края диапазона нелинейны — используйте калиброванное чтение

Особенности АЦП ESP32-S3

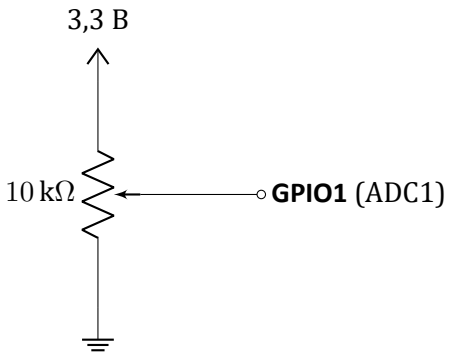
- Два блока: **ADC1** (**GPIO1–GPIO10**) и **ADC2** (**GPIO11–GPIO20**).
- **ADC2 используется также Wi-Fi.** Доступ зависит от режима и драйвера. В уроке 8 плата будет подключена к сети, поэтому потенциометр ставим на ADC1 — **GPIO1**.
- Внутреннее опорное напряжение $\approx 1{,}1\,\text{V}$, поэтому на входе стоит *аттенюатор*. По умолчанию в Arduino выбран $11\,\text{dB}$ — рабочий диапазон примерно $0 - 3{,}1\,\text{V}$.
- Каждый чип откалиброван на заводе (eFuse). Функция `analogReadMilliVolts()` учитывает калибровку и возвращает милливольты — предпочитайте её «сырому» `analogRead()`.

5.3 Потенциометр — «крутилка»

Потенциометр — это резистор с третьим, подвижным контактом (движком). Крайние выводы подключают к 0 и 3,3 В, а движок, скользя по резистивной дорожке, выдаёт любое напряжение между ними. Поворот ручки — самый наглядный аналоговый сигнал.



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 5.2: Потенциометр на выводе GPIO1

Листинг 5.1: Чтение потенциометра

```
const int POT_PIN = 1;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12);                  // 0..4095
}

void loop() {
  int raw = analogRead(POT_PIN);
  int mv  = analogReadMilliVolts(POT_PIN);    // с учётом калибровки
  Serial.printf("raw:%d mV:%d\n", raw, mv);   // формат для Serial Plotter
  delay(100);
}
```

Совет

Открой Tools → Serial Plotter: пары «имя:значение» Arduino IDE нарисует графиками. Покрути ручку — увидишь свой аналоговый сигнал вживую!

5.4 Проверка чтения ручки

До подключения LED откройте Serial Monitor и медленно поверните ручку. Ожидается: около 0 мВ у GND, около 3100 мВ у 3V3 (дальше АЦП упирается в предел диапазона) и промежуточные числа между ними. Если всегда 0, проверьте провод с ползунка потенциометра до ADC. Если всегда максимум, проверьте, что крайние выводы действительно идут к GND и 3V3.

Совет

Откройте Tools → Serial Plotter: пары «имя:значение» IDE нарисует графиками. Покрутите ручку и обратите внимание на шум — соседние отсчёты отличаются на единицы-десятки милливольт.

5.5 Самостоятельный опыт: ручка яркости

Переведите милливольты в проценты яркости и поборите шум двумя приёмами:

- **усреднение** — 8 измерений подряд;
- **порог** — яркость меняется, только если ручку повернули больше чем на 3 %, чтобы LED не дрожал при неподвижной ручке.

Свяжите уровень потенциометра с `ledcWrite()` из урока 4 в новом самостоятельном скетче.

5.6 Измерение напряжения выше 3,3 В

Чтобы измерить, например, батарею 12 V, понижаем напряжение делителем:

$$U_{\text{АЦП}} = U_{\text{вх}} \cdot \frac{R_2}{R_1 + R_2} = 12 \cdot \frac{33}{100 + 33} \approx 2{,}98\,\text{V}.$$

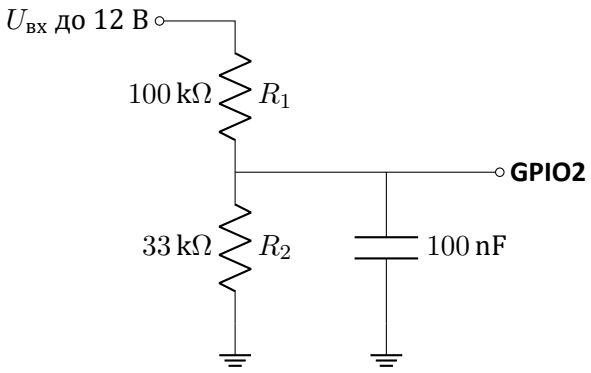


Рис. 5.3: Делитель напряжения с фильтрующим конденсатором

Листинг 5.2: Вольтметр с усреднением

```
const int VIN_PIN = 2;
const float K = (100.0 + 33.0) / 33.0;     // коэффициент делителя

float readVoltage() {
  uint32_t sum = 0;
  for (int i = 0; i < 32; i++) sum += analogReadMilliVolts(VIN_PIN);
  return (sum / 32.0) / 1000.0 * K;        // вольты на входе делителя
}

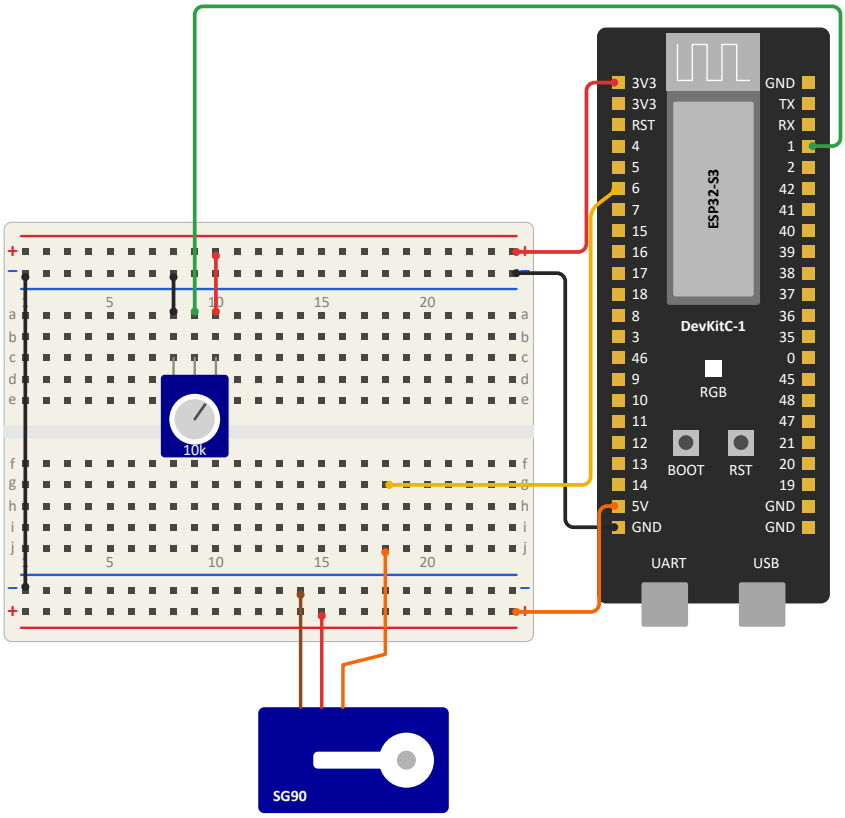
void setup() { Serial.begin(115200); }

void loop() {
  Serial.printf("U = %.2f В\n", readVoltage());
  delay(500);
}
```

5.7 Практика: ручка поворачивает сервопривод

Соединим «крутилку» и сервопривод: повернул ручку — повернулся моторчик. Так устроены многие манипуляторы роботов.

У сервопривода три провода: **коричневый** — земля, **красный** — питание 5 В, **оранжевый** — сигнал. Питание сервопривода возьмём с вывода 5V платы, поэтому на макетной плате у нас будут *две разные шины «+»*: сверху 3,3 В для потенциометра, снизу 5 В для мотора. Будь внимателен и не перепутай их! Шины «−» соединены перемычкой слева.



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 5.4: Потенциометр управляет сервоприводом

Листинг 5.3: Ручка поворачивает сервопривод

```
// Ручка-потенциометр поворачивает сервопривод
const int POT_PIN   = 1;                    // ADC1
const int SERVO_PIN = 6;
const uint8_t RES = 14;                     // 0..16383 на период 20 мс

void servoWrite(int angle) {                // 0..180 градусов
  uint32_t us = map(angle, 0, 180, 1000, 2000);        // длина импульса, мкс
  ledcWrite(SERVO_PIN, us * ((1 << RES) - 1) / 20000);
}

int readPot() {                             // среднее из 8 измерений, мВ
  uint32_t sum = 0;
  for (int i = 0; i < 8; i++) sum += analogReadMilliVolts(POT_PIN);
  return sum / 8;
```

```
}

int lastAngle = -100;

void setup() {
  Serial.begin(115200);
  ledcAttach(SERVO_PIN, 50, RES);            // 50 Гц -- как любит серво
}

void loop() {
  int angle = constrain(map(readPot(), 0, 3100, 0, 180), 0, 180);
  if (abs(angle - lastAngle) >= 2) {         // не дёргаемся из-за шума
    lastAngle = angle;
    servoWrite(angle);
    Serial.printf("угол: %d\n", angle);
  }
  delay(20);
}
```

Функция `map(x, 0, 3100, 0, 180)` пересчитывает число из одного диапазона в другой: 0 мВ → 0°, 3100 мВ → 180°. А проверка `abs(angle - lastAngle) >= 2` не даёт мотору дёргаться из-за шума.

Осторожно

Под нагрузкой сервопривод потребляет сотни миллиампер. Если при его движении плата перезагружается, питай сервопривод от отдельного источника 5 В (например, 4 батарейки AA), соединив его «минус» с GND платы.

5.8 То же на ESPUI: вольтметр со стрелкой и графиком

Число в Serial Monitor удобно для отладки, но смотреть на ручку потенциометра и на график одновременно приятнее на экране телефона. Если у вас уже собрана схема этого урока, дополнительных деталей не нужно: прошейте скетч из урока 39 и подключитесь к сети `S3-Voltmeter` (пароль `lesson-39-demo`).

Листинг 5.4: ESPUI: Gauge и Graph для потенциометра на GPIO1

```
// Потенциометр: крайние выводы к 3V3 и GND, движок -> GPIO1 (ADC1).
#include <WiFi.h>
#include <ESPUI.h>

const int POT_PIN = 1;
uint16_t gaugeId, graphId, valueId;

int readMilliVolts() {                          // среднее из 16 замеров гасит шум
  uint32_t sum = 0;
  for (int i = 0; i < 16; i++) sum += analogReadMilliVolts(POT_PIN);
  return sum / 16;
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Voltmeter", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  gaugeId = ESPUI.addControl(ControlType::Gauge, "Напряжение, мВ", "0",
↪  ControlColor::Wetasphalt);
```

```
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, gaugeId);
  ESPUI.addControl(ControlType::Max, "", "3300", ControlColor::None, gaugeId);
  valueId = ESPUI.label("Значение", ControlColor::Peterriver, "0 мВ");
  graphId = ESPUI.graph("Последние отсчёты", ControlColor::Emerald);
  ESPUI.begin("ESP32-S3 voltmeter");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static uint32_t last = 0;
  if (millis() - last >= 500) {                    // два обновления в секунду достаточно глазу
    last = millis();
    int mv = readMilliVolts();
    ESPUI.updateControlValue(gaugeId, String(mv));
    ESPUI.updateLabel(valueId, String(mv) + " мВ");
    ESPUI.addGraphPoint(graphId, mv);
  }
  delay(5);
}
```

Крутите ручку: стрелка Gauge показывает милливольты, а на графике остаётся след последних отсчётов. Усреднение по 16 замерам из этого урока работает прежним образом, а страница получает новое значение два раза в секунду.

Если не работает

- **Всегда 0 или всегда максимум** — ползунок потенциометра не дошёл до GPIO или крайние выводы подключены неверно.
- **Показания сильно скачут** — усредняйте 8–32 отсчёта и поставьте конденсатор 100 нФ на вход.
- **С Wi-Fi чтение перестало работать** — датчик стоит на ADC2; перенесите его на ADC1 (**GPIO1–GPIO10**).

Итоги урока

- `analogReadMilliVolts()` даёт калиброванный результат; диапазон — примерно $0 - 3{,}1$ V.
- Для совместной работы с Wi-Fi выбираем ADC1; для сетевых проектов берите ADC1 (**GPIO1–GPIO10**).
- Шум убираем усреднением, а дрожание управления — порогом (гистерезисом).

Проверь себя

1. Почему `analogReadMilliVolts()` лучше, чем `analogRead()`?
2. Почему в сетевых опытах аналоговый датчик подключают к ADC1?
3. Как измерить напряжение 12 В и что делает конденсатор в делителе?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выведите «сырое» значение АЦП потенциометра и убедитесь, что оно меняется от 0 до 4095.

2. ★ Выведите напряжение в милливольтах через `analogReadMilliVolts()` и сравните с мультиметром в трёх положениях ручки.
3. ★★ Добавьте усреднение по 8 отсчётам. Измерьте, на сколько уменьшился разброс показаний.
4. ★★ Управляйте яркостью светодиода (урок 4) потенциометром и сделайте «мёртвую зону» в крайних положениях.
5. ★★★ Добавьте фоторезистор в делитель и режим «авто»: светодиод ярче, когда в комнате темнее, с гистерезисом.
6. ★★★★ Измерьте `micros()`, сколько длится чтение с усреднением по 8 и по 64 отсчётам, и постройте график «точность против скорости».

Урок 6. Шина I²C: дисплей

Цель	разобраться в устройстве шины I²C, найти устройства сканером, выводить информацию на OLED
Опирается на	урок 2 (Serial для сканера), урок 5 (значение потенциометра, которое будем показывать)
Результат	экран показывает текст и значение датчика
Понадобится	OLED-дисплей SSD1306 128×64 с интерфейсом I²C

6.1 Топология шины

I²C использует всего два провода: **SDA** (данные) и **SCL** (такт). Выходы устройств — с *открытым стоком*: они могут только притянуть линию к земле, а в HIGH её возвращают подтягивающие резисторы (тот же принцип, что у кнопки в уроке 3). Каждое ведомое устройство имеет 7-битный адрес.

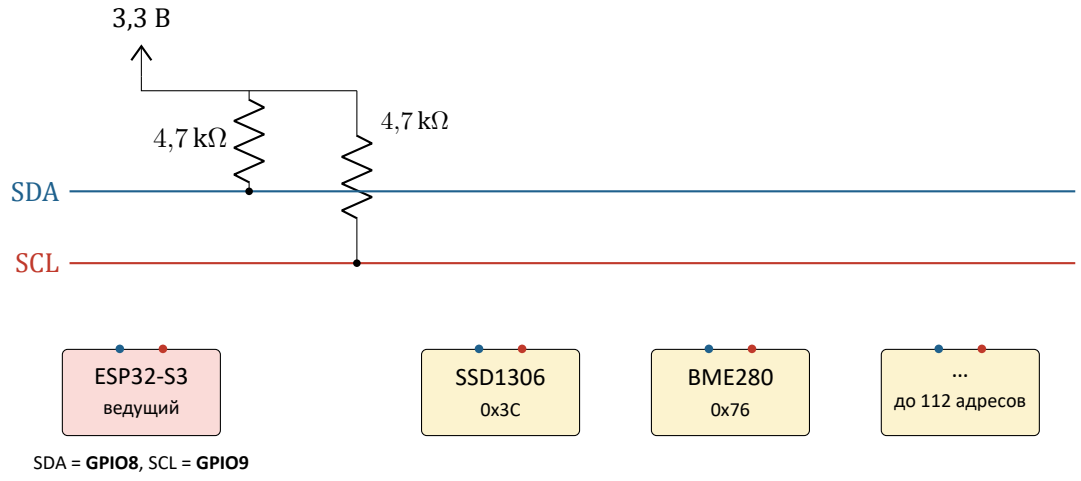


Рис. 6.1: Шина I²C: один ведущий, несколько ведомых на общих линиях

Совет

Большинство модулей (дисплеи, датчики на платах-переходниках) уже имеют подтягивающие резисторы. Если на шине много модулей, суммарное сопротивление подтяжки может оказаться слишком малым — иногда лишние резисторы приходится выпаивать.

6.2 Временная диаграмма

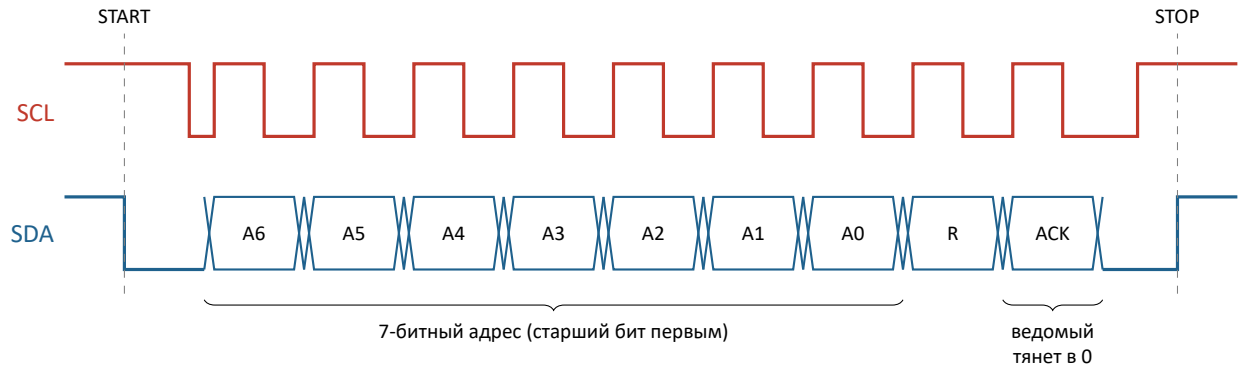


Рис. 6.2: Начало транзакции: START, адрес, бит чтения/записи, подтверждение ACK, STOP. Данные меняются, пока SCL = LOW, и считываются, пока SCL = HIGH

6.3 Сканер шины

Подключите дисплей: SDA → **GPIO8**, SCL → **GPIO9**, питание 3V3 и GND. Первое, что стоит сделать с новым модулем, — убедиться, что он отвечает, и узнать его адрес.

Листинг 6.1: Сканер I²C

```
#include <Wire.h>

void setup() {
  Serial.begin(115200);
  delay(500);
  Wire.begin(8, 9);                    // SDA, SCL
  Serial.println("Сканирование...");
  int found = 0;
  for (uint8_t addr = 0x08; addr < 0x78; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {   // 0 = устройство ответило ACK
      Serial.printf("  найдено устройство: 0x%02X\n", addr);
      found++;
    }
  }
  Serial.printf("Всего: %d\n", found);
}

void loop() {}
```

6.4 Первый вывод на экран

Установите через Library Manager библиотеки **Adafruit SSD1306** и **Adafruit GFX**. Библиотека рисует в буфере в памяти, а `display()` отправляет весь буфер (1 КБ) по шине.

Листинг 6.2: Hello на OLED-дисплее

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(128, 64, &Wire, -1);   // -1: без вывода сброса

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);
  Wire.setClock(400000);                        // Fast mode, 400 кГц
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("Дисплей не найден");
    while (true) delay(1000);
  }
  display.setTextColor(SSD1306_WHITE);
}

void loop() {
  display.clearDisplay();                       // очистить буфер
  display.setTextSize(2);
  display.setCursor(0, 0);
  display.print("ESP32-S3");
  display.setTextSize(1);
  display.setCursor(0, 30);
  display.printf("uptime: %lu s", millis() / 1000);
  display.display();                            // отправить буфер на экран
  delay(200);
}
```

Совет

Стандартные шрифты Adafruit GFX не содержат кириллицы — для русского текста нужны сторонние шрифты или библиотека U8g2 с кириллическими наборами. В примерах обойдёмся латиницей.

6.5 Самостоятельный опыт: экран состояния



Рис. 6.3: Пример экрана: заголовок, значение крупно и полоска-индикатор

Рисовать экран в каждом проходе `loop()` не нужно: отправка буфера занимает десятки миллисекунд. Обновляем его 10 раз в секунду — тем же приёмом с `millis()`, что и неблокирующее мигание в уроке 1.

Сделайте отдельный скетч, который выводит на OLED значение потенциометра или датчика раз в секунду. Затем добавьте в него кнопку (урок 3) и команды Serial (урок 2). Посмотрите на

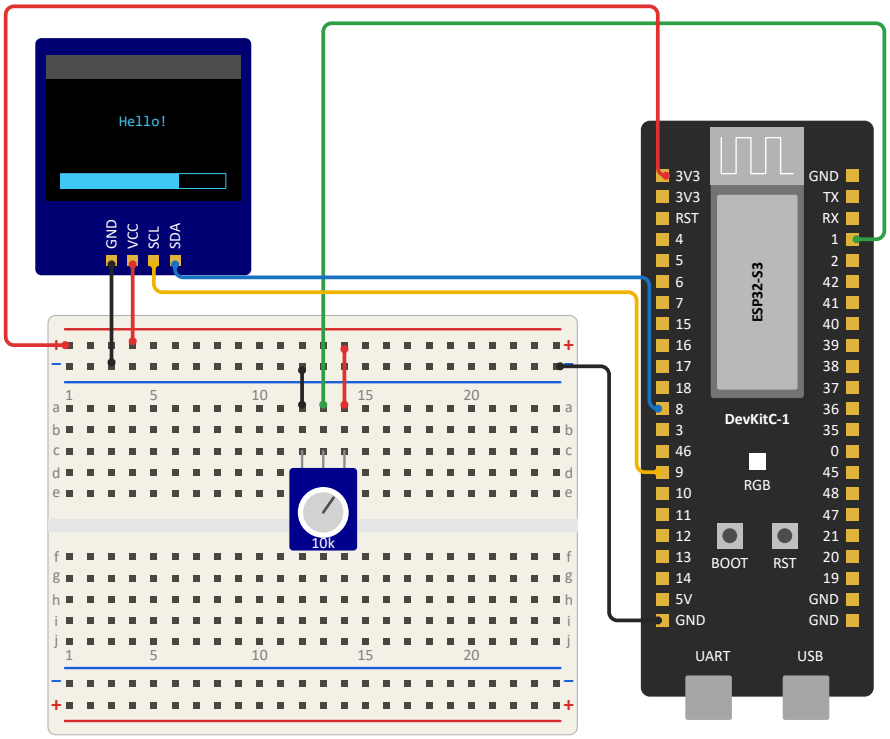
`loop()`: в нём уже несколько разных дел, и каждое вынуждено «уступать» остальным. Пока отправляется буфер экрана, кнопка не опрашивается; добавьте сюда ещё и сеть — и разобраться, кто кому мешает, станет трудно. Эту проблему решает следующий урок.

6.6 Практика: вольтметр с экраном

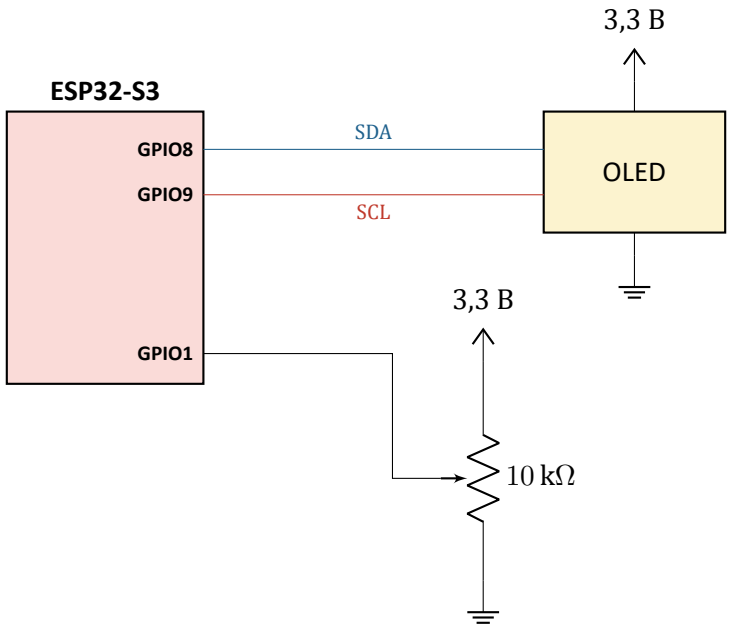
Соединим экран с потенциометром из урока 5: на экране будет напряжение на движке крупными цифрами и полоска-шкала, как у настоящего прибора.



Рис. 6.4: Экран вольтметра: заголовок, напряжение крупно и полоска



а) Монтажная схема: как собрать на макетной плате



б) Принципиальная электрическая схема

Рис. 6.5: Вольтметр: потенциометр на GPIO1 и экран на I²C

Листинг 6.3: Вольтметр с экраном

```
// Вольтметр с экраном: напряжение на ручке крупно и полоска-шкала
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

const int POT_PIN = 1;
Adafruit_SSD1306 display(128, 64, &Wire, -1);

```

```
int readMilliVolts() {
  uint32_t sum = 0;
  for (int i = 0; i < 16; i++) sum += analogReadMilliVolts(POT_PIN);
  return sum / 16;
}

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);                          // SDA, SCL
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("Дисплей не найден");
    while (true) delay(1000);
  }
  display.setTextColor(SSD1306_WHITE);
}

void loop() {
  int mv = readMilliVolts();
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("Voltmeter");
  display.setTextSize(3);
  display.setCursor(0, 18);
  display.printf("%.2fV", mv / 1000.0);
  display.drawRect(0, 54, 128, 10, SSD1306_WHITE);        // рамка шкалы
  display.fillRect(2, 56, mv * 124 / 3300, 6, SSD1306_WHITE);
  display.display();                           // отправить картинку на экран
  delay(100);                                  // 10 кадров в секунду хватит
}
```

Экран обновляется 10 раз в секунду — чаще не нужно: глаз всё равно не заметит разницы (вспомни инерцию зрения из урока 4), а отправка картинки по шине занимает время.

Если не работает

- **Сканер ничего не находит** — перепутаны SDA и SCL, нет питания 3V3 или в `Wire.begin()` другие выводы.
- **Дисплей найден по адресу 0x3D** — укажите этот адрес в `display.begin()`.
- **Экран пустой, ошибок нет** — после рисования не вызван `display.display()`.

Итоги урока

- I²C — два провода, подтяжки, адреса; первым делом запускайте сканер.
- Библиотека дисплея рисует в буфере; `display()` отправляет его целиком — это небыстро.
- Медленные действия выполняем периодически, а не на каждом проходе `loop()`.

Проверь себя

1. Зачем шине I²C подтягивающие резисторы?
2. Что показывает сканер шины и о чём говорит отсутствие ответа?
3. Почему экран не перерисовывают на каждом проходе `loop()`?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите сканер шины и запишите адрес вашего экрана. Что покажет сканер, если отключить SCL?
2. ★ Выведите на экран своё имя крупным шрифтом.
3. ★★ Показывайте на экране время работы платы в секундах, обновляя его раз в секунду.
4. ★★ Покажите на экране значение потенциометра и шкалу-полоску под ним.
5. ★★★ Добавьте датчик BME280/AHT20 на ту же шину и показывайте температуру в заголовке экрана.
6. ★★★★ Нарисуйте график изменения яркости за последние 128 обновлений экрана, а через 10 секунд бездействия приглушайте экран (`display.dim(true)`).

Урок 7. FreeRTOS: задачи и синхронизация

Цель	разделить программу на независимые задачи, организовать безопасный обмен данными, задействовать оба ядра
Опирается на	уроки 1–6 (неблокирующий `loop()`, кнопка, АЦП, экран), прерывания из урока 3
Результат	кнопка, экран и Serial работают как отдельные задачи
Понадобится	ничего нового из «железа»

7.1 Зачем нужна операционная система реального времени

В конце урока 6 `loop()` совмещал пять дел, и каждое новое дело замедляло остальные. FreeRTOS позволяет написать каждое дело как отдельную «программу» — **задачу** — со своим бесконечным циклом и приоритетом. Планировщик переключает задачи каждую миллисекунду, а на ESP32-S3 задачи могут выполняться на двух ядрах по-настоящему параллельно.
Вы уже пользовались FreeRTOS, не зная об этом: `setup()` и `loop()` выполняются внутри задачи `loopTask` на ядре 1, а `delay()` — это вызов FreeRTOS `vTaskDelay()`, который отдаёт процессор другим задачам.

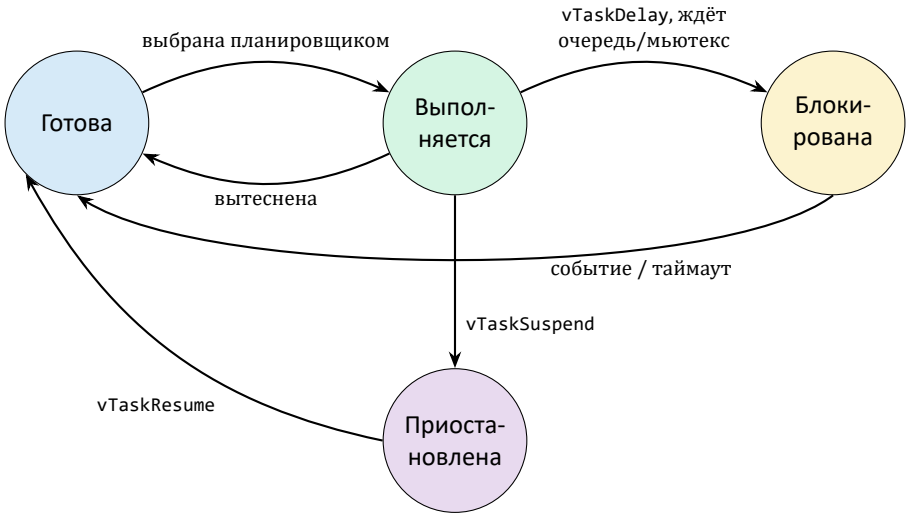


Рис. 7.1: Состояния задачи FreeRTOS. Задача, ждущая в `vTaskDelay()`, не тратит процессорное время

7.2 Первые задачи

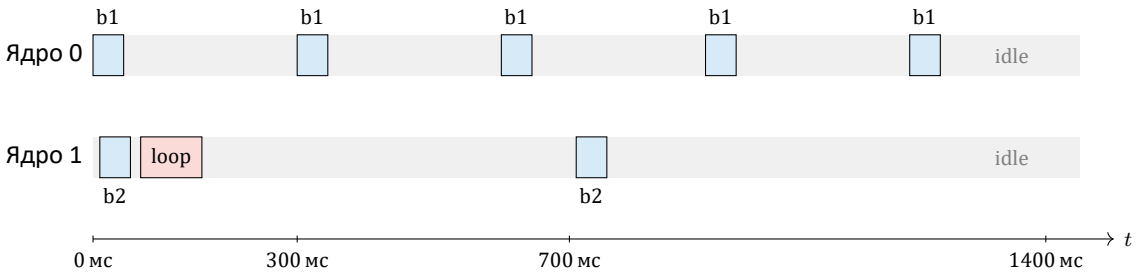


Рис. 7.2: Задачи из листинга ниже: `b1` (GPIO4, каждые 300 мс) на ядре 0, `b2` (GPIO7, каждые 700 мс) и `loop()` на ядре 1. Каждая работает доли миллисекунды, остальное время ядро свободно (idle)

Листинг 7.1: Две независимые задачи (отдельный эксперимент, светодиоды на GPIO4 и GPIO7)

```
void blinkTask(void* param) {
  int pin = (int)(intptr_t)param;          // номер вывода передан параметром
  int period = (pin == 4) ? 300 : 700;
  pinMode(pin, OUTPUT);
  for (;;) {                               // задача никогда не завершается сама
    digitalWrite(pin, !digitalRead(pin));
    vTaskDelay(pdMS_TO_TICKS(period));     // отдаём процессор другим задачам
  }
}

void setup() {
  Serial.begin(115200);
  //                    функция    имя   стек  параметр        приор. дескр. ядро
  xTaskCreatePinnedToCore(blinkTask, "b1", 2048, (void*)(intptr_t)4, 1, NULL, 0);
  xTaskCreatePinnedToCore(blinkTask, "b2", 2048, (void*)(intptr_t)7, 1, NULL, 1);
}

void loop() {
  Serial.printf("loop() на ядре %d\n", xPortGetCoreID());
  delay(2000);
}
```

Сравните с уроком 1: там для двух частот мигания пришлось вести отдельные таймеры на `millis()`, а здесь каждая задача просто «спит» столько, сколько ей нужно.

7.3 Общие данные: очередь и мьютекс

Когда несколько задач работают с одними данными, возникает опасность: одна задача может прочитать структуру в тот момент, когда другая записала её лишь наполовину. Для передачи данных между задачами используется **очередь** — потокобезопасный FIFO-буфер.



Рис. 7.3: Очередь развязывает задачи по времени

Листинг 7.2: Производитель и потребитель

```
QueueHandle_t q;

void sensorTask(void*) {
  for (;;) {
    int mv = analogReadMilliVolts(1);
    xQueueSend(q, &mv, 0);                      // не ждать, если очередь полна
    vTaskDelay(pdMS_TO_TICKS(100));
  }
}

void printTask(void*) {
  int mv;
  for (;;) {
    if (xQueueReceive(q, &mv, portMAX_DELAY) == pdTRUE) {  // спим до прихода данных
      Serial.printf("U = %d мВ\n", mv);
```

```
    }
  }
}

void setup() {
  Serial.begin(115200);
  q = xQueueCreate(5, sizeof(int));
  xTaskCreatePinnedToCore(sensorTask, "sensor", 3072, NULL, 2, NULL, 1);
  xTaskCreatePinnedToCore(printTask,  "print",  4096, NULL, 1, NULL, 1);
}

void loop() { vTaskDelete(NULL); }              // loopTask больше не нужен
```

Если же данные — это общее *состояние*, которое все читают и меняют (например, «светодиод включён, яркость 60 %»), удобнее **мьютекс**: «ключ», который задача берёт перед работой с данными и возвращает после. Пока ключ у одной задачи, остальные ждут.

Примитив	Когда использовать
Очередь (`xQueue*`)	передача данных между задачами или из прерывания (`xQueueSendFromISR`)
Мьютекс (`xSemaphoreCreateMutex`)	защита общего ресурса: переменных, `Serial`, шины I^2C
Двоичный семафор	сигнал «событие произошло» (например, из прерывания в задачу)
Уведомления (`xTaskNotify`)	самый быстрый и лёгкий сигнал конкретной задаче
Программный таймер (`xTimerCreate`)	периодический вызов короткой функции без отдельной задачи

Таблица 7.1: Основные средства синхронизации FreeRTOS

7.4 Практика: мьютекс — «ключ» от общего ресурса

Когда две задачи пользуются одной вещью — например, `Serial`, — они могут мешать друг другу. Проверим! Две задачи печатают свои фразы по одной букве. Сначала поставь в программе `USE_MUTEX = false`:

Листинг 7.3: Две задачи и мьютекс

```
// Две задачи печатают в Serial. Мьютекс не даёт им перебивать друг друга
SemaphoreHandle_t serialMutex;              // «ключ» от Serial
const bool USE_MUTEX = true;                // поставь false и сравни!

void say(const char* text) {
  if (USE_MUTEX) xSemaphoreTake(serialMutex, portMAX_DELAY);  // взять ключ
  for (const char* c = text; *c; c++) {     // печатаем по одной букве,
    Serial.print(*c);                       // как будто это долгая работа
    delayMicroseconds(300);
  }
  Serial.println();
  if (USE_MUTEX) xSemaphoreGive(serialMutex);               // вернуть ключ
}

void catTask(void*) {
  for (;;) { say("Кошка говорит: мяу-мяу-мяу"); vTaskDelay(pdMS_TO_TICKS(7)); }
}

void dogTask(void*) {
  for (;;) { say("Собака говорит: гав-гав-гав"); vTaskDelay(pdMS_TO_TICKS(5)); }
}
```

```
22
23 void setup() {
24   Serial.begin(115200);
25   serialMutex = xSemaphoreCreateMutex();     // создать до запуска задач!
26   xTaskCreatePinnedToCore(catTask, "cat", 3072, NULL, 1, NULL, 0);
27   xTaskCreatePinnedToCore(dogTask, "dog", 3072, NULL, 1, NULL, 1);
28 }
29
30 void loop() { vTaskDelete(NULL); }          // loop() нам не нужен
```

Без мьютекса в Serial Monitor получится каша: «Кошка говСобака говорит: гаорит: мяу...». Задачи перебивают друг друга на полуслове. Теперь поставь `true`: фразы идут ровно, по очереди. **Мьютекс** (от англ. *mutual exclusion* — взаимное исключение) работает как ключ от единственной комнаты: задача берёт ключ (`xSemaphoreTake`), делает свою работу и возвращает ключ (`xSemaphoreGive`). Пока ключ у одной задачи, другая ждёт.



Рис. 7.4: Мьютекс не даёт задачам перебивать друг друга

Инструмент FreeRTOS	Когда пригодится
Задача (`xTaskCreate`)	отдельное дело со своим циклом
Очередь (`xQueue...`)	передать данные из одной задачи в другую (или из прерывания)
Мьютекс (`xSemaphoreCreateMutex`)	не дать двум задачам одновременно пользоваться `Serial`, экраном, общей переменной
Уведомление (`xTaskNotify`)	разбудить конкретную задачу: «событие произошло»
Программный таймер (`xTimerCreate`)	вызывать короткую функцию периодически

Таблица 7.2: Главные инструменты FreeRTOS

Осторожно

Если задаче не хватает памяти стека, плата перезагружается с сообщением `Stack canary watchpoint triggered`. Тогда увеличь третий параметр `xTaskCreatePinnedToCore()` — например, с 2048 до 4096.

7.5 Самостоятельная работа: устройство на задачах

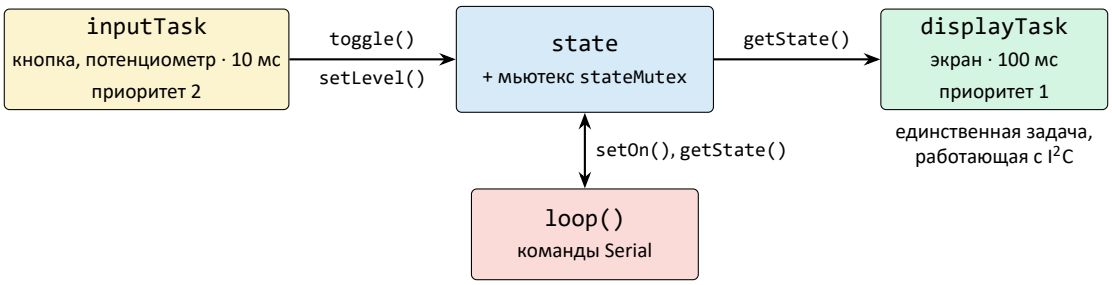


Рис. 7.5: Устройство на задачах: задачи обращаются к общему состоянию только через функции под мьютексом

Главное правило: общее состояние меняется только через функции `toggle()`, `setLevel()`, `setOn()` и читается через `getState()`. Мьютекс берётся внутри этих функций, поэтому защищать данные нужно в одном месте, а не в каждой задаче.
Соберите по этой схеме отдельный скетч: кнопка и потенциометр (уроки 3 и 5), OLED (урок 6), команды Serial (урок 2) и светодиод на ШИМ (урок 4). Экран должен обновляться независимо от нажатий.
Кнопка теперь опрашивается каждые 10 ms независимо от того, чем занят экран: у задачи ввода выше приоритет, и планировщик прервёт рисование, как только ей пора работать.

Осторожно

Переполнение стека — частая причина перезагрузок с сообщением `Stack canary watchpoint triggered`. Проверяйте запас функцией `uxTaskGetStackHighWaterMark(NULL)` и увеличивайте размер стека при использовании `printf`, `String` и больших локальных массивов.

Если не работает

- **Перезагрузка с `Stack canary`** — увеличьте размер стека задачи в `xTaskCreatePinnedToCore()`.
- **Срабатывает watchdog** — в бесконечном цикле задачи нет `vTaskDelay()` или ожидания очереди.
- **Данные «перемешиваются»** — общая переменная меняется без мьютекса или сразу из нескольких задач.

Итоги урока

- Задача — функция с бесконечным циклом; `vTaskDelay()` отдаёт процессор другим.
- Данные между задачами передаём очередью, общее состояние защищаем мьютексом.
- Каждое периферийное устройство (экран, Serial) лучше закрепить за одной задачей.

Проверь себя

1. Чем задача отличается от функции `loop()`?
2. Когда нужна очередь, а когда мьютекс?
3. Как заметить переполнение стека задачи?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите пример с двумя задачами и убедитесь, что они работают одновременно. Измените период одной из них.
2. ★ Раз в 5 секунд печатайте в Serial запас стека каждой задачи (`uxTaskGetStackHighWaterMark`).
3. ★★ Вынесите мигание встроенного RGB-светодиода в отдельную задачу. Как упростился `loop()`?
4. ★★ Передайте число из одной задачи в другую через очередь и выведите его на экран.
5. ★★★ Переведите кнопку на прерывание (урок 3): обработчик будит задачу через

`vTaskNotifyGiveFromISR`.

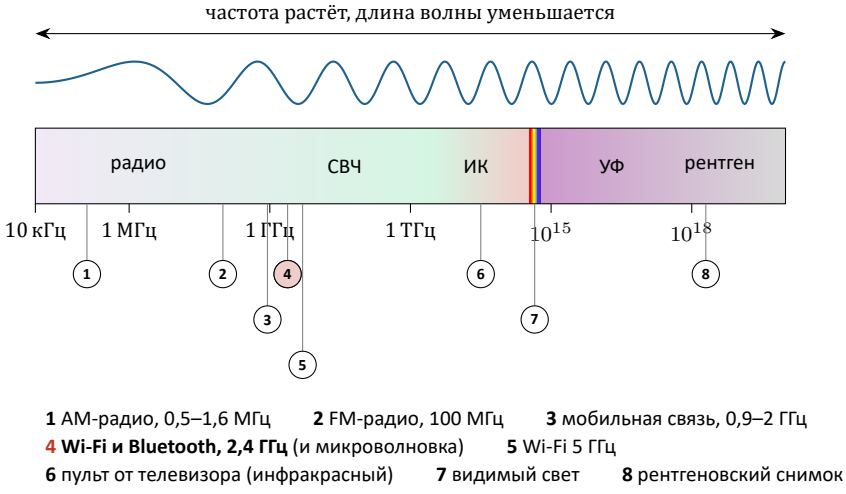
6. ★★★★ Защитите общий ресурс (например, Serial) мьютексом, а затем намеренно уберите его и найдите, когда сообщения перемешиваются.

Урок 8. Wi-Fi и веб-сервер: HTTP, HTML и CSS

Цель	подключить ESP32-S3 к Wi-Fi и сделать веб-страницу для управления светодиодом
Опирается на	урок 4 (ШИМ), урок 7 (общее состояние под мьютексом), урок 5 (почему АЦП на ADC1)
Результат	светодиодом можно управлять с телефона: кнопка вкл/выкл и ползунок яркости
Понадобится	Wi-Fi-сеть на 2,4 GHz

8.1 Что такое радиоволны

Радиоволны — это **электромагнитные волны**, как и свет. Отличаются они только *частотой*: сколько раз в секунду колеблется волна. Мы уже знаем, что частоту измеряют в герцах. Для радио получаются огромные числа, поэтому используют приставки: 1 килогерц (кГц) = 1000 Гц, 1 мегагерц (МГц) = миллион герц, 1 гигагерц (ГГц) = миллиард герц.



1 AM-радио, 0,5–1,6 МГц **2** FM-радио, 100 МГц **3** мобильная связь, 0,9–2 ГГц
4 **Wi-Fi и Bluetooth, 2,4 ГГц** (и микроволновка) **5** Wi-Fi 5 ГГц
6 пульт от телевизора (инфракрасный) **7** видимый свет **8** рентгеновский снимок

Рис. 8.1: Шкала электромагнитных волн (частоты отложены так, что каждое деление больше предыдущего в тысячу раз). Видимый свет занимает крошечную полоску, а Wi-Fi — соседи микроволновки

Все электромагнитные волны летят со скоростью света $c = 300\,000$ км/с. Зная частоту, можно найти **длину волны** — расстояние между соседними «гребнями»:

$$\lambda = \frac{c}{f} = \frac{300\,000\,000 \text{ м/с}}{2\,400\,000\,000 \text{ Гц}} = 0{,}125 \text{ м} = 12{,}5 \text{ см}.$$

Антенна обычно делается длиной в четверть волны — около 3 см. Посмотри на зигзаг на модуле ESP32-S3: если его «распрямить», получится примерно такая длина!

Это интересно

Микроволновая печь греет еду волнами почти той же частоты, что и Wi-Fi, — 2,45 ГГц. Только мощность у неё около 800 Вт, а у Wi-Fi платы — меньше 0,1 Вт, в десять тысяч раз слабее. Поэтому работающая микроволновка может мешать Wi-Fi, но Wi-Fi никого не нагреет.

8.2 Wi-Fi и его каналы

Wi-Fi — это способ передавать цифровые данные по радио. Биты «прячутся» в радиоволне: передатчик очень быстро меняет её параметры, а приёмник расшифровывает изменения обратно в нули и единицы.

Диапазон 2,4 ГГц поделён на **каналы**, как дорога — на полосы. Если две сети работают на одном канале, они мешают друг другу. Каналы шириной 20 МГц частично перекрываются, поэтому не мешают друг другу только каналы 1, 6 и 11:

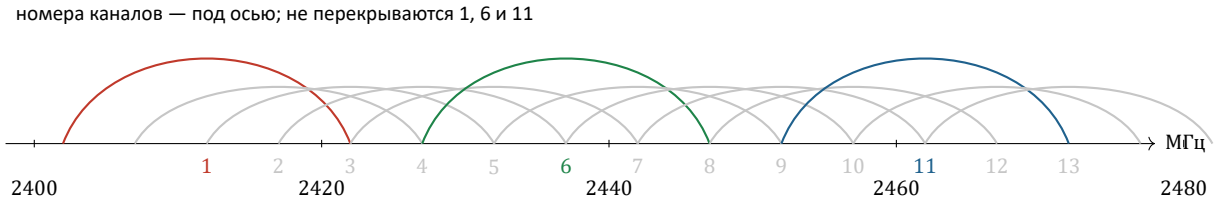


Рис. 8.2: Каналы Wi-Fi в диапазоне 2,4 ГГц

Начнём с того, что посмотрим, какие сети «видит» плата. Эта программа выводит имя каждой сети, её канал и силу сигнала:

Листинг 8.1: Поиск Wi-Fi сетей

```
// Какие Wi-Fi сети видит плата?
#include <WiFi.h>

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);                        // режим «клиент»
  Serial.print("MAC-адрес платы: ");
  Serial.println(WiFi.macAddress());
}

void loop() {
  Serial.println("Ищу сети...");
  int n = WiFi.scanNetworks();                // занимает пару секунд
  for (int i = 0; i < n; i++) {
    Serial.printf("%2d) %-20s канал %2d, сигнал %4d дБм, %s\n",
                  i + 1, WiFi.SSID(i).c_str(), WiFi.channel(i), WiFi.RSSI(i),
                  WiFi.encryptionType(i) == WIFI_AUTH_OPEN ? "открытая" : "с паролем");
  }
  WiFi.scanDelete();                          // освободить память
  delay(10000);
}
```

Сила сигнала (RSSI) измеряется в **дБм** (децибел-милливаттах) и всегда отрицательна: чем ближе к нулю, тем сильнее сигнал.

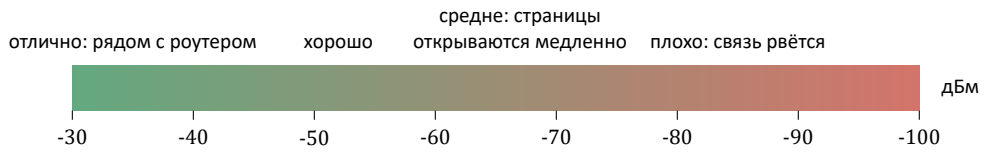


Рис. 8.3: Как понимать силу сигнала RSSI

8.3 Bluetooth

Bluetooth работает в том же диапазоне 2,4 ГГц, но создан для другого: связывать *близкие* устройства (наушники, часы, телефон) и при этом тратить очень мало энергии. ESP32-S3 поддерживает **Bluetooth Low Energy** (BLE, «Bluetooth с низким энергопотреблением»).

	Wi-Fi	**Bluetooth LE**
Дальность	десятки метров	10–30 метров
Скорость	миллионы бит в секунду	сотни тысяч бит в секунду
Потребление	большое (до 300 мА в пике)	маленькое (можно годами работать от батарейки)
Нужен роутер?	обычно да	нет, устройства связываются напрямую
Для чего	интернет, веб-страницы, видео	датчики, фитнес-браслеты, умные замки, наушники

Таблица 8.1: Wi-Fi и Bluetooth LE: одна частота — разные задачи

Устройства BLE постоянно «кричат» в эфир короткие сообщения: «Я тут, меня зовут так-то». Послушаем их:

Листинг 8.2: Поиск Bluetooth-устройств поблизости

```
// Какие Bluetooth-устройства рядом?
#include <BLEDevice.h>

void setup() {
  Serial.begin(115200);
  BLEDevice::init("");                      // включить Bluetooth
}

void loop() {
  Serial.println("Слушаю эфир 5 секунд...");
  BLEScan* scan = BLEDevice::getScan();
  scan->setActiveScan(true);                // просить устройства назвать себя
  BLEScanResults* found = scan->start(5, false);
  for (int i = 0; i < found->getCount(); i++) {
    BLEAdvertisedDevice d = found->getDevice(i);
    Serial.printf("%s  сигнал %4d дБм  %s\n", d.getAddress().toString().c_str(),
                  d.getRSSI(), d.getName().c_str());
  }
  scan->clearResults();
  delay(5000);
}
```

Сканер показывает только устройства, которые передают объявления BLE. Подключение и обмен данными разобраны в уроке 12.

8.4 Режимы Wi-Fi

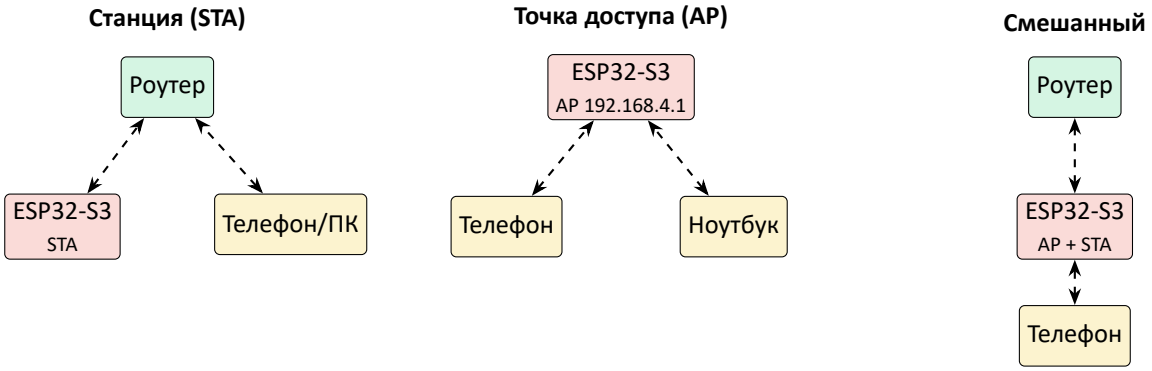


Рис. 8.4: Три режима работы Wi-Fi. В опытах используем режим станции

Стек Wi-Fi — это несколько задач FreeRTOS, которые по умолчанию работают на ядре 0. Наши задачи закреплены за ядром 1, поэтому сеть не мешает кнопке и экрану.

8.5 Подключение к сети

Листинг 8.3: Подключение к Wi-Fi

```
#include <WiFi.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PASS);
  Serial.print("Подключение");
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) {
    delay(300);
    Serial.print(".");
  }
  if (WiFi.status() == WL_CONNECTED) {
    Serial.printf("\nIP: %s, RSSI: %d дБм\n",
                  WiFi.localIP().toString().c_str(), WiFi.RSSI());
  } else {
    Serial.println("\nНе удалось подключиться");
  }
}

void loop() {}
```

Осторожно

ESP32-S3 работает только в диапазоне 2,4 GHz. Сети 5 GHz он не увидит.

Учимся программировать: цикл `while` и логическое «и»

Подключение к сети занимает несколько секунд, и заранее неизвестно, сколько именно. Цикл `for` здесь не подходит — он для случаев, когда число повторений известно. Нужен цикл `while` («пока»): он повторяет команды, *пока* условие верно.

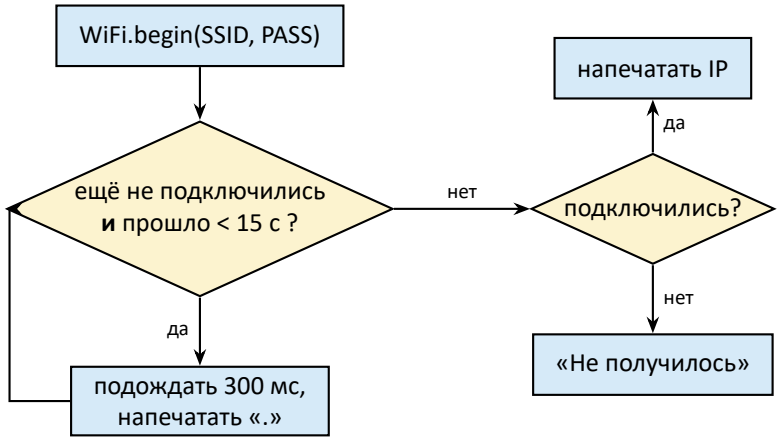


Рис. 8.5: Блок-схема подключения: ждём в цикле, но не дольше 15 секунд

В условии цикла два вопроса соединены знаком `&&` — логическим «**и**»: цикл продолжается, только если верны *оба* условия. Как только плата подключилась *или* время вышло, цикл заканчивается. Без второго условия плата могла бы ждать вечно, если пароль неверный.

Осторожно

Если условие `while` никогда не станет ложным, программа застрянет в цикле навсегда. Проверяй, что рано или поздно цикл закончится. Выйти из любого цикла досрочно можно командой `break`, а из функции — командой `return` (так сделано в программе, если подключиться не удалось).

Осторожно

ESP32-S3 работает только в диапазоне 2,4 ГГц. Сеть 5 ГГц он не увидит. Если роутер раздаёт обе сети с одинаковым именем, иногда помогает включить на телефоне точку доступа и подключить плату к нему.

8.6 MAC-адрес и IP-адрес

Когда в сети много устройств, данные нужно доставлять точно по адресу. Для этого у каждого устройства есть *два* адреса.

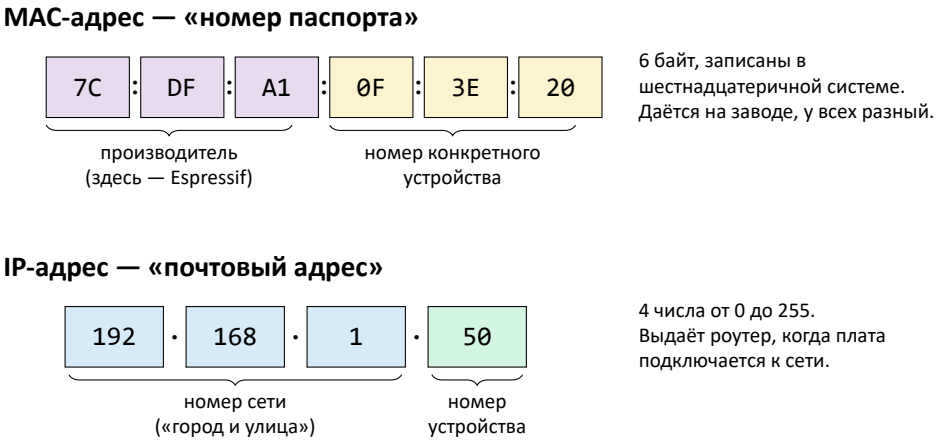


Рис. 8.6: Два адреса каждого сетевого устройства

- **MAC-адрес** — постоянный «номер паспорта» сетевого устройства. Его записывают на за-

воде, и он не меняется при переезде в другую сеть. Роутер по MAC-адресу узнаёт устройство «в лицо»: например, можно запретить подключение незнакомым устройствам или выдать своему всегда один и тот же IP.

- **IP-адрес** — «почтовый адрес» в конкретной сети. В домашней сети он обычно выглядит как `192.168.1.`*xx* или `192.168.0.`*xx*. Переедешь в другую сеть (например, в школьную) — адрес будет другим, как у человека при переезде.

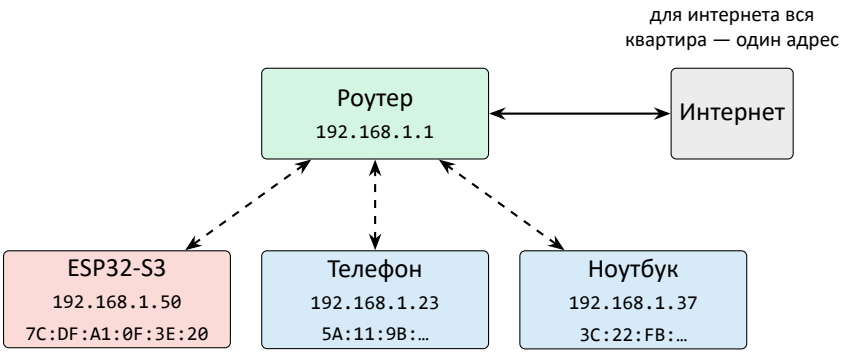


Рис. 8.7: Домашняя сеть: у каждого устройства свой IP-адрес внутри сети и свой MAC-адрес

Кто раздаёт IP-адреса

Когда плата подключается, она не знает своего IP. Она громко спрашивает всю сеть, и роутер выдаёт ей свободный адрес. Этот механизм называется **DHCP**:

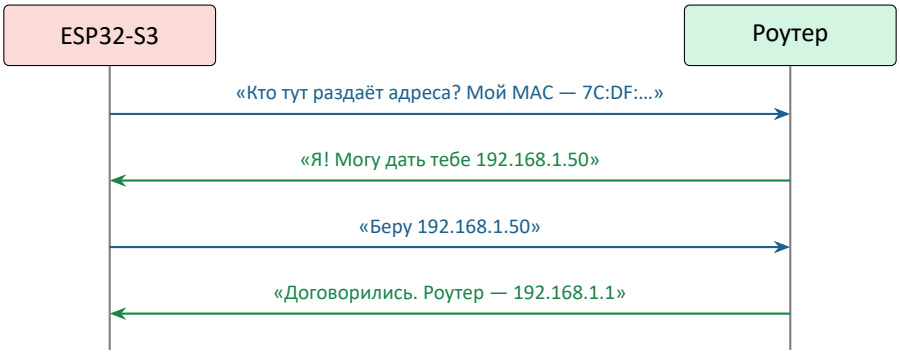


Рис. 8.8: Как плата получает IP-адрес (DHCP)

Неудобство в том, что при следующем включении роутер может выдать *другой* адрес. Решений три:

1. попросить роутер всегда выдавать этому MAC-адресу один и тот же IP (в настройках роутера);
2. задать **постоянный (статический) IP** прямо в программе;
3. вообще не запоминать IP, а обращаться к плате **по имени** — об этом ниже.

Как задать IP, MAC и имя в программе

Листинг 8.4: Постоянный IP-адрес, свой MAC и имя в сети

```
// Постоянный IP-адрес, свой MAC-адрес и имя esp32-led.local
#include <WiFi.h>
#include <ESPmDNS.h>
#include <esp_wifi.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

IPAddress ip(192, 168, 1, 50);              // адрес, который мы хотим получить
IPAddress gateway(192, 168, 1, 1);          // адрес роутера
IPAddress subnet(255, 255, 255, 0);         // маска: «своя» сеть -- 192.168.1.*
IPAddress dns(192, 168, 1, 1);              // кто переводит имена сайтов в адреса

uint8_t myMac[6] = {0x02, 0x12, 0x34, 0x56, 0x78, 0x9A};  // «самодельный» MAC

void setup() {
  Serial.begin(115200);
  WiFi.setHostname("esp32-led");            // имя в списке устройств роутера
  WiFi.mode(WIFI_STA);
  esp_wifi_set_mac(WIFI_IF_STA, myMac);     // по желанию: сменить MAC
  WiFi.config(ip, gateway, subnet, dns);    // постоянный IP вместо выданного роутером
  WiFi.begin(SSID, PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
  }
  Serial.printf("\nMAC: %s, IP: %s\n",
                WiFi.macAddress().c_str(), WiFi.localIP().toString().c_str());

  if (MDNS.begin("esp32-led")) {            // включить имя esp32-led.local
    Serial.println("Плата отзывается на имя esp32-led.local");
  }
}

void loop() {}
```

- `WiFi.config(ip, gateway, subnet, dns)` задаёт IP вручную. Выбирай адрес из своей сети (первые три числа — как у роутера) и такой, который точно никем не занят. Если адрес совпадёт с чужим, оба устройства будут работать с ошибками.
- `esp_wifi_set_mac()` меняет MAC-адрес. Обычно этого делать не нужно; первый байт `0x02` означает «адрес назначен вручную», это локально назначенный адрес; его уникальность внутри сети нужно обеспечить самостоятельно.
- `WiFi.setHostname()` — имя, которое будет видно в списке устройств на странице роутера.

8.7 Имя вместо адреса: mDNS

Запоминать `192.168.1.50` неудобно. В интернете эту задачу решают имена сайтов: мы пишем `wikipedia.org`, а специальный сервер DNS переводит имя в IP-адрес. В домашней сети есть похожий механизм — **mDNS** (англ. *multicast DNS*). Устройство само отвечает на своё имя, которое всегда оканчивается на `.local`. Достаточно одной строки `MDNS.begin("esp32-led")`, и плата станет доступна как `esp32-led.local`.

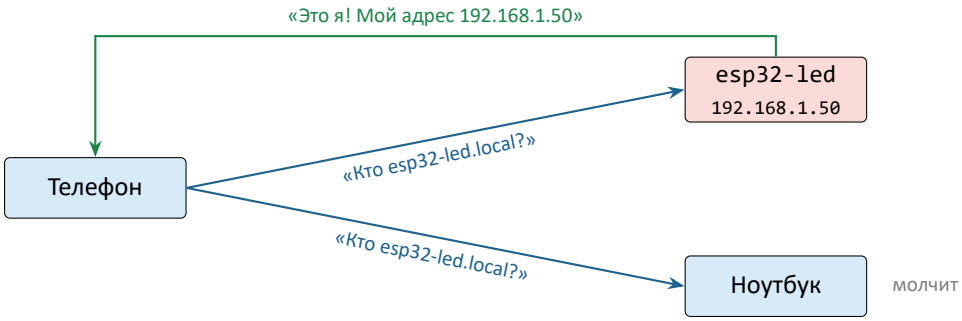


Рис. 8.9: mDNS: вопрос отправляется всем устройствам сразу, отвечает только то, чьё это имя

Совет

Имена `.local` понимают Windows 10/11, macOS, iPhone и Linux. Некоторые старые Android-телефоны mDNS не поддерживают — тогда придётся ввести IP-адрес.

8.8 Клиент и сервер

Когда ты открываешь сайт, твой браузер (**клиент**) отправляет **запрос** другому компьютеру (**серверу**), а тот присылает **ответ** — страницу. Это похоже на кафе: посетитель делает заказ, официант передаёт его на кухню и приносит блюдо. Сегодня кухней будет наша плата: ESP32-S3 станет **веб-сервером**, а браузер телефона — клиентом.

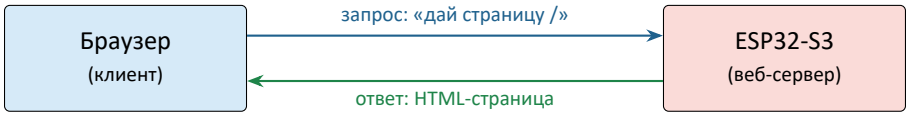


Рис. 8.10: Клиент спрашивает — сервер отвечает. Сам сервер никогда не начинает разговор первым

Язык, на котором они разговаривают, называется **HTTP** (англ. *HyperText Transfer Protocol* — протокол передачи гипертекста). Начнём с самой простой страницы:

Листинг 8.5: Первый веб-сервер

```
// Самый простой веб-сервер: страница «Привет!»
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

WebServer server(80);                       // 80 -- стандартный «порт» для HTTP

void handleRoot() {                         // кто-то открыл адрес "/"
  String html = "<h1>Привет! Я ESP32-S3</h1>";
  html += "<p>Я работаю уже " + String(millis() / 1000) + " секунд.</p>";
  server.send(200, "text/html; charset=utf-8", html);  // 200 = «всё хорошо»
}

void handleNotFound() {                     // адрес, которого у нас нет
  server.send(404, "text/plain; charset=utf-8", "Такой страницы нет");
}

void setup() {
```

```
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PASS);
  const uint32_t started = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - started < 15000) {
    delay(300); Serial.print(".");
  }
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nWi-Fi: проверьте сеть и перезапустите плату");
    return;
  }
  Serial.printf("\nОткрой http://%s или http://esp32.local\n",
                WiFi.localIP().toString().c_str());
  MDNS.begin("esp32");

  server.on("/", handleRoot);               // адрес -> функция-обработчик
  server.onNotFound(handleNotFound);
  server.begin();
}

void loop() {
  server.handleClient();                    // ответить браузеру, если он что-то спросил
}
```

Загрузи программу, дождись сообщения в Serial Monitor и открой адрес в браузере телефона, подключённого к той же сети. Обнови страницу несколько раз — число секунд будет расти: страницу каждый раз заново «готовит» плата.

- `WebServer server(80)` — создать сервер. 80 — номер «двери» (порта), в которую браузер стучится по умолчанию;
- `server.on("/", handleRoot)` — «если спросят адрес `/`, вызови `handleRoot`»;
- `server.send(200, тип, текст)` — отправить ответ. 200 — код «всё хорошо»;
- `server.handleClient()` — проверить, не пришёл ли запрос. Вызываем постоянно в `loop()`.

8.9 Из чего состоит адрес

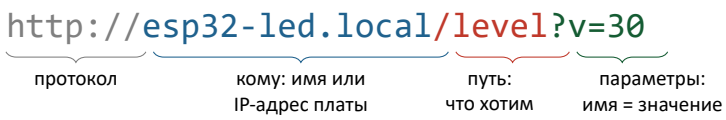


Рис. 8.11: Адрес (URL) состоит из частей. Параметров может быть несколько: `?v=30&color=red`

8.10 Методы GET и POST

Каждый запрос начинается со слова-**метода**, который говорит, чего хочет клиент. Самые важные:

- **GET** («получить») — «дай мне страницу». Параметры записываются прямо в адресе, после знака `?`. Так работают обычные ссылки и адресная строка браузера;
- **POST** («отправить») — «прими от меня данные». Данные передаются в *теле* запроса, отдельно от адреса. Так отправляют формы: логин и пароль, сообщение, заказ.

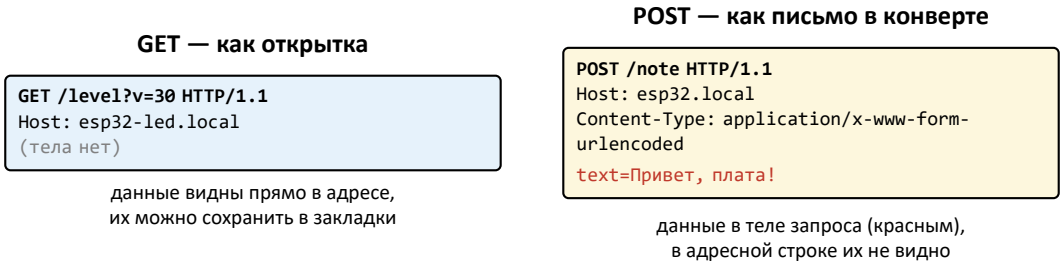


Рис. 8.12: Так выглядят запросы на самом деле — это обычный текст

	GET	POST
Для чего	получить страницу или данные	отправить данные, что-то изменить
Где данные	в адресе: `/level?v=30`	в теле запроса
Видно в адресной строке	да	нет
Можно добавить в закладки	да	нет
Повтор при обновлении страницы	безопасен	браузер спросит «отправить повторно?»
Объём данных	небольшой (адрес ограничен по длине)	большой (хоть целый файл)

Таблица 8.2: Сравнение GET и POST

Совет

Правило хорошего тона: **GET** — только чтобы что-то *узнать*, **POST** — чтобы что-то *изменить*. Иначе поисковый робот или «ускоритель» браузера, заранее открывающий ссылки, может случайно выключить тебе свет!

Сервер отвечает **кодом ответа** — трёхзначным числом:

Код	Значение	Когда встречается
200	OK — всё хорошо	страница найдена и отправлена
303	See Other — смотри другую	«я всё сделал, теперь открой главную»
404	Not Found — не найдено	адреса нет — знаменитая «ошибка 404»
500	Internal Server Error	ошибка в программе сервера

Таблица 8.3: Самые частые коды ответа HTTP

Пример: управление через GET

Самый простой способ управлять светодиодом — ссылки. Нажатие на ссылку `/toggle` переключает светодиод, а форма с ползунком отправляет `/level?v=...`:

Листинг 8.6: Светодиод управляется GET-запросами

```
// Управляем светодиодом через GET-запросы: /toggle и /level?v=...
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

const char* SSID = "MyNetwork";              // имя твоей Wi-Fi сети
const char* PASS = "MyPassword";             // и пароль
const int LED_PIN = 4;
```

```
9
10 WebServer server(80);
11 bool ledOn = false;
12 int level = 50;                              // яркость, %
13
14 uint32_t gammaDuty(int percent) {
15   return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
16 }
17
18 void applyLed() { ledcWrite(LED_PIN, ledOn ? gammaDuty(level) : 0); }
19
20 String page() {                              // собираем HTML-страницу
21   String html = "<!DOCTYPE html><html><head><meta charset='utf-8'>"
22                 "<meta name='viewport' content='width=device-width'>"
23                 "<title>Светодиод</title></head><body style='font-family:sans-serif'>";
24   html += "<h1>Светодиод: ";
25   html += ledOn ? "ВКЛ" : "ВЫКЛ";
26   html += "</h1><p><a href='/toggle'><button>Вкл/выкл</button></a></p>"
27           "<form action='/level'>Яркость: <input type='range' name='v' "
28           "min='0' max='100' value='" + String(level) + "' "
29           "onchange='this.form.submit()'></form></body></html>";
30   return html;
31 }
32
33 void goHome() {                              // «перейди на главную страницу»
34   server.sendHeader("Location", "/");
35   server.send(303);
36 }
37
38 void setup() {
39   Serial.begin(115200);
40   ledcAttach(LED_PIN, 5000, 8);
41   applyLed();
42
43   WiFi.mode(WIFI_STA);
44   WiFi.begin(SSID, PASS);
45   const uint32_t started = millis();
46   while (WiFi.status() != WL_CONNECTED && millis() - started < 15000) {
47     delay(300); Serial.print(".");
48   }
49   if (WiFi.status() != WL_CONNECTED) {
50     Serial.println("\nWi-Fi: проверьте сеть и перезапустите плату");
51     return;
52   }
53   Serial.printf("\nОткрой http://%s или http://esp32-led.local\n",
54                 WiFi.localIP().toString().c_str());
55   MDNS.begin("esp32-led");
56
57   server.on("/",       []() { server.send(200, "text/html; charset=utf-8", page()); });
58   server.on("/toggle", []() { ledOn = !ledOn; applyLed(); goHome(); });
59   server.on("/level",  []() {
60     level = constrain(server.arg("v").toInt(), 0, 100);
61     applyLed();
62     goHome();
63   });
64   server.begin();
65 }
66
67 void loop() {
```

```
  server.handleClient();                    // ответить браузеру, если он что-то спросил
}
```

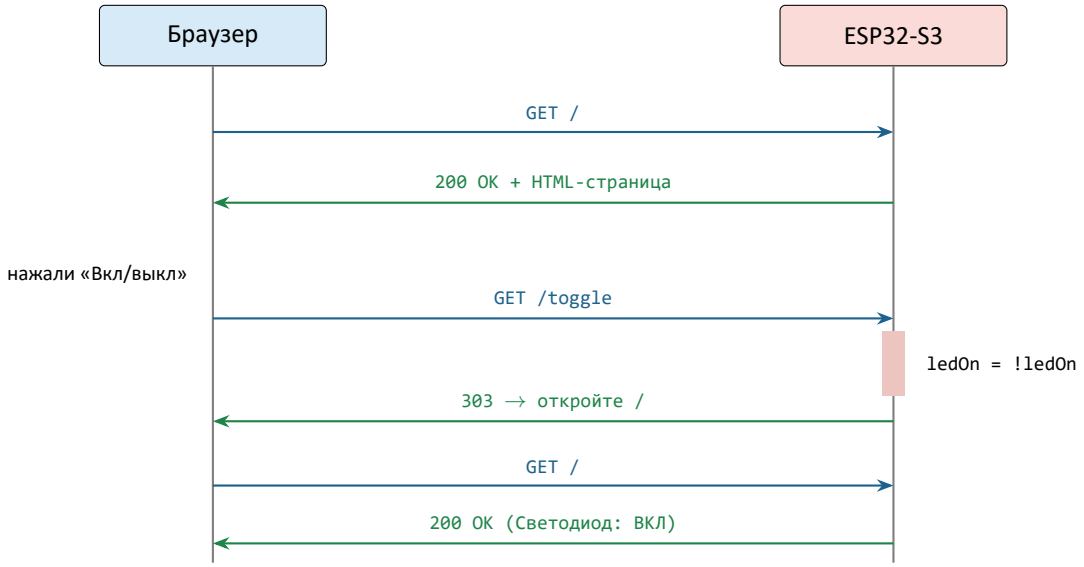


Рис. 8.13: Что происходит при нажатии кнопки на странице

Работает, но у этого способа два минуса: команды меняют состояние через GET (см. совет выше), а страница каждый раз полностью перезагружается и мигает. Исправим оба.

Пример: отправка данных через POST

Сделаем «записку для платы»: форма отправляет текст методом POST, плата печатает его в Serial Monitor и показывает на странице.

Листинг 8.7: Приём данных методом POST

```
// Записка для платы: форма отправляет текст методом POST
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

WebServer server(80);
String note = "(записок пока нет)";          // последняя полученная записка

void handleRoot() {                          // GET /  -- показать страницу с формой
  String html = "<!DOCTYPE html><html><head><meta charset='utf-8'></head><body>"
                "<h1>Записка для ESP32</h1>"
                "<p>Последняя записка: <b>" + note + "</b></p>"
                "<form method='POST' action='/note'>"
                "<input name='text' placeholder='Напиши что-нибудь'>"
                "<button>Отправить</button></form></body></html>";
  server.send(200, "text/html; charset=utf-8", html);
}

void handleNote() {                          // POST /note -- принять записку
  note = server.arg("text");                 // данные из тела запроса
  Serial.println("Новая записка: " + note);
```

```
  server.sendHeader("Location", "/");        // вернуться на главную
  server.send(303);
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PASS);
  const uint32_t started = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - started < 15000) {
    delay(300); Serial.print(".");
  }
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nWi-Fi: проверьте сеть и перезапустите плату");
    return;
  }
  Serial.printf("\nОткрой http://%s или http://esp32.local\n",
                WiFi.localIP().toString().c_str());
  MDNS.begin("esp32");

  server.on("/", HTTP_GET, handleRoot);
  server.on("/note", HTTP_POST, handleNote); // этот адрес принимает только POST
  server.begin();
}

void loop() {
  server.handleClient();
}
```

В форме указано `method='POST'`, а на плате обработчик зарегистрирован с `HTTP_POST`: на обычный GET-запрос по адресу `/note` он не ответит. Текст записки достаётся функцией `server.arg("text")` — так же, как параметры GET, только данные пришли из тела запроса.

8.11 HTML: из чего сделана страница

Страница, которую отправляет сервер, написана на языке **HTML** (англ. *HyperText Markup Language* — язык разметки гипертекста). Это не язык программирования: он описывает, *что* есть на странице. Каждый элемент обозначается **тегами** в угловых скобках: открывающим `<p>` и закрывающим `</p>`. Теги вкладываются друг в друга, как матрёшки:



Рис. 8.14: Структура HTML-страницы (слева) и как браузер её показывает (справа)

Тег	Что означает
`<h1>…</h1>`	крупный заголовок (есть ещё `h2`–`h6` поменьше)
`<p>…</p>`	абзац текста
`<b>…</b>`	жирный текст
`<a href="/toggle">…</a>`	ссылка
`<button>…</button>`	кнопка
`<input type="range">`	ползунок (`type="text"` — поле для текста)
`<form>…</form>`	форма: собирает поля и отправляет на сервер
`<div>…</div>`	«коробка» для группировки элементов

Таблица 8.4: Теги, которых хватит для первых страниц

8.12 CSS: делаем красиво

HTML отвечает за содержание, а за *внешний вид* — цвета, шрифты, отступы, скругления, тени — отвечает язык **CSS** (англ. *Cascading Style Sheets* — каскадные таблицы стилей). Стили пишут внутри тега `<style>`. Каждое правило состоит из *селектора* (к чему применить) и списка свойств:

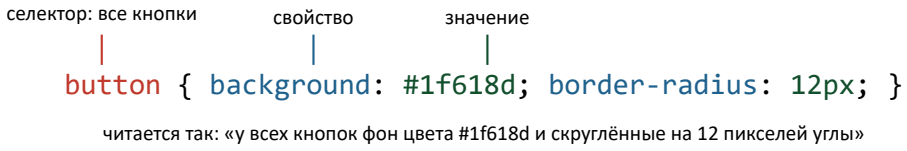


Рис. 8.15: Устройство правила CSS

Цвета в CSS часто записывают шестнадцатеричным кодом `#RRGGBB`: две цифры на красный, зелёный и синий — та же смесь трёх цветов, что в RGB-светодиоде из урока 1! Например, `#ff0000` — красный, `#ffffff` — белый.

Свойство CSS	Что меняет
`color`, `background`	цвет текста и фона
`font-family`, `font-size`	шрифт и его размер
`padding`, `margin`	внутренний и внешний отступ
`border-radius`	скругление углов (`50%` — круг)
`box-shadow`	тень или свечение вокруг элемента
`width`, `height`	ширина и высота
`transition`	плавная смена стиля (анимация)

Таблица 8.5: Свойства CSS для первых экспериментов

8.13 Практика: панель управления светодиодом

Соберём всё вместе: HTML-страницу с CSS-стилями и маленькой программой на **JavaScript** — языке, который выполняется прямо в браузере. Он будет отправлять POST-запросы *без перезагрузки страницы* и перерисовывать «лампочку» на экране.



Рис. 8.16: Как будет выглядеть панель на телефоне

Листинг 8.8: Панель управления: HTML + CSS + JavaScript

```
// Красивая панель управления светодиодом: HTML + CSS + немного JavaScript
#include <WiFi.h>
#include <WebServer.h>
#include <ESPmDNS.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";
const int LED_PIN = 4;

WebServer server(80);
bool ledOn = false;
int level = 50;                                  // яркость, %

// Вся страница -- одна большая строка. R"rawliteral( ... )rawliteral"
// позволяет писать кавычки и переносы строк без лишних значков.
const char PAGE[] = R"rawliteral(
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Светодиод</title>
<style>
  body  { font-family: sans-serif; background: #eef2f7;
          display: flex; justify-content: center; padding-top: 40px; }
  .card { background: white; border-radius: 20px; padding: 24px 32px; width: 260px;
          box-shadow: 0 6px 20px rgba(0,0,0,.12); text-align: center; }
  h1    { color: #7b241c; font-size: 24px; }
  .led { width: 90px; height: 90px; border-radius: 50%; margin: 10px auto 20px;
         background: #ccc; transition: .3s; }
  .led.on { background: #ff5a4f; box-shadow: 0 0 30px #ff5a4f; }
  button { background: #1f618d; color: white; border: none; border-radius: 12px;
           font-size: 18px; padding: 10px 28px; }
  input  { width: 100%; margin-top: 20px; }
</style>
```

```
</head>
<body>
<div class="card">
  <h1>Мой светодиод</h1>
  <div id="led" class="led"></div>
  <button onclick="send('toggle')">Вкл / выкл</button>
  <input id="lvl" type="range" min="0" max="100" onchange="send('level=' + this.value)">
  <p>Яркость: <b id="txt">?</b> %</p>
</div>
<script>
function show(s) {                    // нарисовать состояние светодиода
  document.getElementById('led').className = s.on ? 'led on' : 'led';
  document.getElementById('lvl').value = s.level;
  document.getElementById('txt').textContent = s.level;
}
function send(cmd) {                  // POST-запрос без перезагрузки страницы
  fetch('/api', { method: 'POST', body: new URLSearchParams(cmd) })
    .then(r => r.json()).then(show);
}
fetch('/api').then(r => r.json()).then(show);   // GET: узнать состояние при открытии
</script>
</body>
</html>
)rawliteral";

uint32_t gammaDuty(int percent) {
  return (uint32_t)(255 * pow(percent / 100.0, 2.2) + 0.5);
}

void applyLed() { ledcWrite(LED_PIN, ledOn ? gammaDuty(level) : 0); }

void handleApi() {
  if (server.method() == HTTP_POST) {        // POST -- команда изменить состояние
    if (server.hasArg("toggle")) ledOn = !ledOn;
    if (server.hasArg("level")) {
      level = constrain(server.arg("level").toInt(), 0, 100);
      ledOn = true;
    }
    applyLed();
  }
  // и на GET, и на POST отвечаем текущим состоянием в формате JSON
  String json = String("{\"on\":") + (ledOn ? "true" : "false") +
                ",\"level\":" + level + "}";
  server.send(200, "application/json", json);
}

void setup() {
  Serial.begin(115200);
  ledcAttach(LED_PIN, 5000, 8);
  applyLed();
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PASS);
  const uint32_t started = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - started < 15000) {
    delay(300); Serial.print(".");
  }
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nWi-Fi: проверьте сеть и перезапустите плату");
    return;
```

```
  }
  Serial.printf("\nОткрой http://%s или http://esp32-led.local\n",
                WiFi.localIP().toString().c_str());
  MDNS.begin("esp32-led");

  server.on("/", []() { server.send(200, "text/html; charset=utf-8", PAGE); });
  server.on("/api", handleApi);
  server.begin();
}

void loop() {
  server.handleClient();
}
```

Как это работает:

1. При открытии `http://esp32-led.local` браузер делает `GET /` и получает страницу `PAGE`.
2. Скрипт на странице сразу делает `GET /api`. Плата отвечает состоянием в формате **JSON** — `{"on":true,"level":60}`. Это текст, который легко читают и программы, и люди.
3. При нажатии кнопки функция `fetch()` отправляет `POST /api` с телом `toggle=`, а при движении ползунка — `level=60`. Плата меняет яркость и снова отвечает JSON.
4. Функция `show()` перекрашивает «лампочку» и обновляет число. Страница не перезагружается.

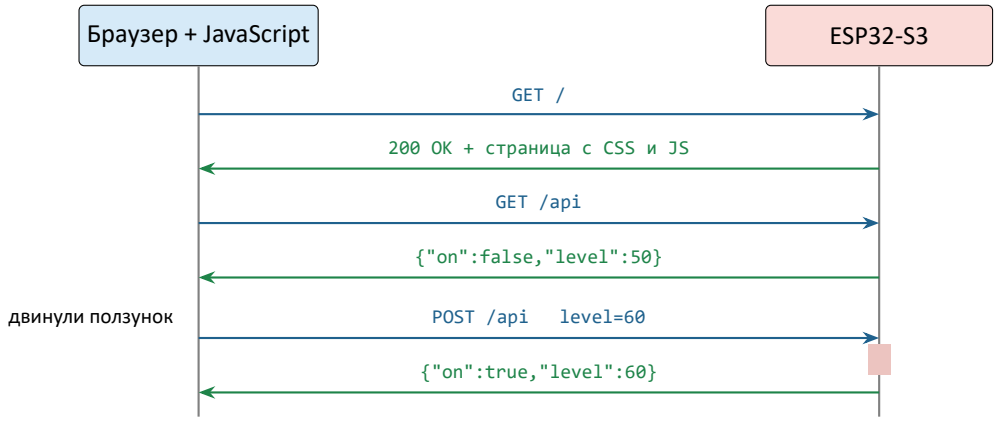


Рис. 8.17: Страница загружается один раз, дальше идут только маленькие запросы к `/api`

Совет

Хранить страницу в строке `PAGE` удобно, пока она небольшая. Для больших сайтов с картинками файлы записывают в файловую систему платы (LittleFS); работа с ней разобрана в уроке 18.

8.14 Самостоятельный опыт: кнопка и браузер вместе

Добавьте в скетч кнопку из урока 3: нажатие и ссылка `/toggle` должны менять одно и то же состояние. Если кнопку читает отдельная задача (урок 7), защитите `ledOn` и `level` мьютексом — обработчики WebServer выполняются из loop(), а задача кнопки может работать параллельно.

8.15 Если роутера нет: собственная точка доступа

ESP32-S3 может сам создать сеть. Замените в примере веб-сервера подключение к роутеру (`WiFi.mode` и `WiFi.begin`) на:

Листинг 8.9: Фрагмент: собственная точка доступа

```
WiFi.softAP("ESP32-S3-Lab", "12345678");     // пароль не короче 8 символов
Serial.println(WiFi.softAPIP());             // обычно 192.168.4.1
```

Совет

Чтобы не запоминать IP-адрес, подключите mDNS: `#include <ESPmDNS.h>` и `MDNS.begin("esp32s3");` — страница откроется по адресу `http://esp32s3.local/`.

Если не работает

- **Не подключается к сети** — сеть 5 ГГц, ошибка в пароле или слабый сигнал (`WiFi.RSSI()` ниже -80 дБм).
- **Страница не открывается** — телефон в другой (например, гостевой) сети или адрес набран с ошибкой.
- **Страница открывается очень долго** — в `loop()` есть долгие `delay()`, и `handleClient()` вызывается редко.

Итоги урока

HTML описывает содержание, CSS задаёт оформление, JavaScript отправляет запросы и обновляет страницу. GET читает состояние, POST передаёт команду; JSON связывает браузер с программой платы.

- Режимы Wi-Fi: станция, точка доступа, смешанный; только $2{,}4$ GHz.
- `WebServer`: адрес $\rightarrow$ обработчик; `handleClient()` вызываем постоянно.
- Сеть — ещё один источник команд; устройство обрабатывает их теми же функциями, что и нажатие кнопки.

Проверь себя

1. Чем режим станции отличается от режима точки доступа?
2. Что происходит между браузером и платой при переходе по ссылке `/toggle`?
3. Почему ожидание подключения к Wi-Fi ограничивают по времени?
4. Чем отличаются GET и POST? Почему POST сам по себе не шифрует данные?
5. За что отвечают HTML, CSS и JavaScript?
6. Какое CSS-свойство превращает квадрат в круг?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Подключитесь к сети, выведите IP-адрес и откройте панель. Найдите в коде HTML, CSS и JavaScript.
2. ★ Измените заголовок, подписи и цвета панели. Сделайте индикатор зелёным; объясните, какие CSS-селекторы изменили.
3. ★★ Добавьте кнопки яркости 30, 60 и 100 процентов. Каждая отправляет POST и отоб-

ражает ответ платы.

4. ★★ Добавьте GET /api/uptime с временем работы в JSON и обновляйте число на странице каждые две секунды без перезагрузки.
5. ★★★ Если за 15 секунд подключение не удалось, запустите точку доступа и ту же панель. Проверьте оба сценария.
6. ★★★★ Добавьте физическую кнопку и синхронизацию двух браузеров: периодически читайте состояние, показывайте ошибку сети и восстанавливайте связь без повторного переключения светодиода.

Урок 9. Сон и энергосбережение

Цель	разобраться в режимах сна ESP32-S3 и научиться будить чип таймером и кнопкой
Опирается на	урок 3 (кнопка), урок 7 (управление задачами), урок 8 (Wi-Fi как главный потребитель)
Результат	автономный датчик засыпает и просыпается по таймеру или кнопке
Понадобится	мультиметр для измерения тока (по желанию)

9.1 Режимы питания

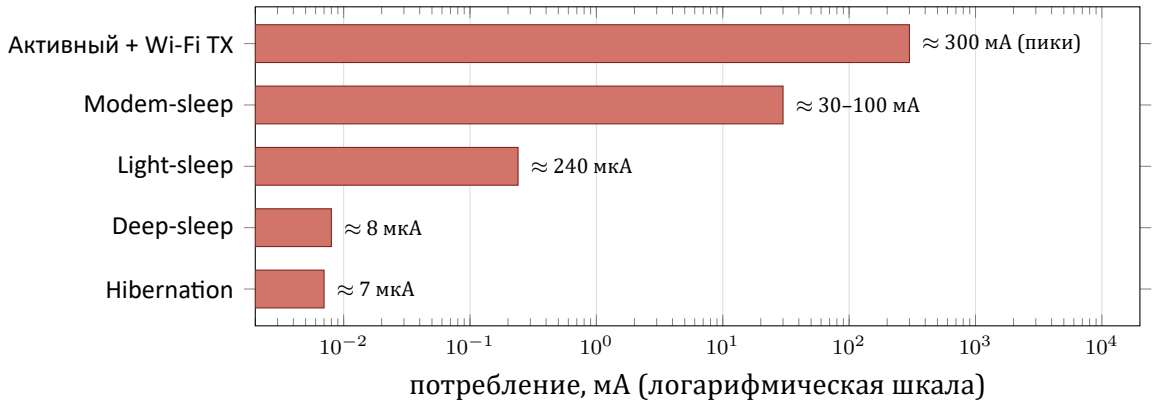


Рис. 9.1: Типичное потребление самого чипа в разных режимах (ориентировочно, по datasheet). Отладочная плата со стабилизатором и светодиодом питания потребляет заметно больше

Режим	Что работает	Что происходит при пробуждении
Modem-sleep	CPU, радио выключается между пакетами	— (обычная работа)
Light-sleep	всё заморожено, память сохранена	программа продолжается со следующей строки
Deep-sleep	только RTC, RTC-память и ULP	**полная перезагрузка**, `setup()` заново
Hibernation	только RTC-таймер и некоторые выводы	полная перезагрузка

Главное про deep-sleep: при пробуждении программа начинается с начала, а обычные переменные теряются. Сохранить данные можно в RTC-памяти — переменные с атрибутом `RTC_DATA_ATTR`.

9.2 Сколько проработает устройство от батареи

Типичный батарейный датчик просыпается, делает работу и снова засыпает:

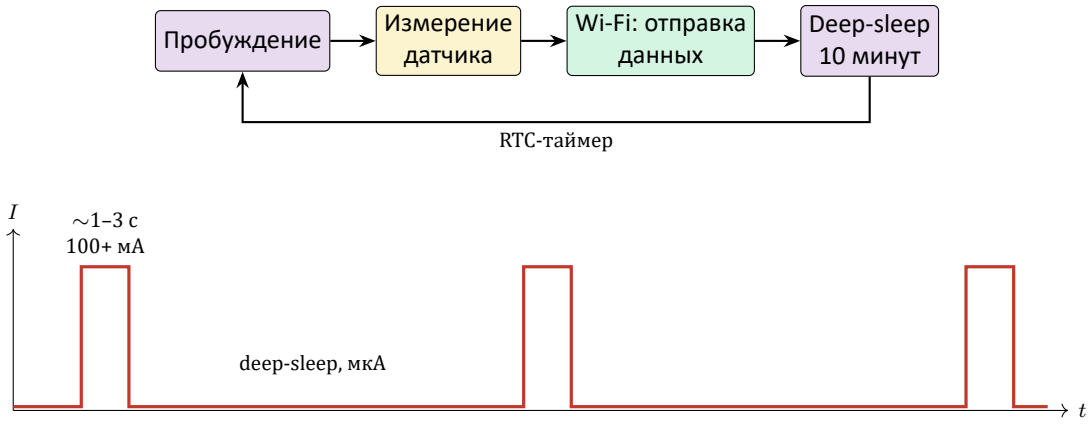


Рис. 9.2: Устройство почти всё время спит; средний ток определяется длительностью активной фазы

$$I_{\text{ср}} = \frac{I_{\text{акт}}\, t_{\text{акт}} + I_{\text{сон}}\, t_{\text{сон}}}{t_{\text{акт}} + t_{\text{сон}}} = \frac{120\,\text{mA} \cdot 2\,\text{s} + 0{,}01\,\text{mA} \cdot 598\,\text{s}}{600\,\text{s}} \approx 0{,}41\,\text{mA}.$$

Аккумулятор $2000\,\text{mAh}$ прослужит $\approx 2000/0{,}41 \approx 4900$ часов, то есть около полугода.

9.3 Deep-sleep по таймеру

Листинг 9.1: Пробуждение по таймеру с сохранением счётчика

```
#include "esp_sleep.h"

RTC_DATA_ATTR int bootCount = 0;          // переживает deep-sleep

const uint64_t SLEEP_US = 10ULL * 1000000; // 10 секунд

void setup() {
  Serial.begin(115200);
  delay(500);
  bootCount++;
  Serial.printf("Пробуждение №%d, причина: %d\n",
                bootCount, esp_sleep_get_wakeup_cause());

  // ... здесь измерения и отправка данных ...

  esp_sleep_enable_timer_wakeup(SLEEP_US);
  Serial.println("Засыпаю");
  Serial.flush();
  esp_deep_sleep_start();                 // сюда программа уже не вернётся
}

void loop() {}                            // не вызывается
```

Осторожно

При питании через нативный USB-порт после засыпания USB-устройство исчезает из системы, и Serial Monitor «отваливается». Это нормально. Если плата не даёт залить новую прошивку, войдите в загрузчик вручную (BOOT + RST, урок 0).

9.4 Пробуждение кнопкой

Проснуться можно по уровню на RTC-выводе (**GPIO0–GPIO21**) — кнопка на **GPIO5** из урока 3 подходит (схема — на рис. 3.2: кнопка между **GPIO5** и GND, внешний резистор не нужен). Во сне обычная подтяжка из урока 3 отключается, поэтому включаем подтяжку RTC-домена:

Листинг 9.2: Фрагмент: пробуждение кнопкой на **GPIO5** (вставляется перед `esp_deep_sleep_start()`)

```
#include "driver/rtc_io.h"                  // функции rtc_gpio_*

rtc_gpio_pullup_en(GPIO_NUM_5);             // подтяжка, работающая во сне
rtc_gpio_pulldown_dis(GPIO_NUM_5);
esp_sleep_enable_ext0_wakeup(GPIO_NUM_5, 0); // проснуться при LOW (кнопка нажата)
esp_deep_sleep_start();
```

9.5 Самостоятельная работа: автономный датчик

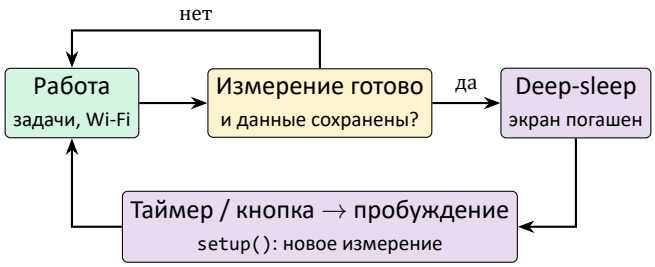


Рис. 9.3: Цикл работы автономного датчика

Соберите отдельный скетч автономного датчика. Что нужно сделать:

- прочитать потенциометр или датчик и напечатать результат в Serial;
- сохранить в RTC-памяти число пробуждений;
- перед сном погасить экран, если он подключён, и завершить запись в файл;
- настроить пробуждение таймером, а затем добавить кнопку как второй источник.

Начните с примера пробуждения по таймеру выше, а затем добавьте чтение датчика перед `esp_deep_sleep_start()`.

Совет

После пробуждения программа запускается заново. Поэтому сначала печатайте причину пробуждения, а уже затем читайте кнопку или датчик: так легко отличить таймер от внешнего сигнала.

9.6 Практика: датчик на батарейке

Соберём всё вместе. Используем кнопку и светодиод (рис. 3.2) и потенциометр вместо «настоящего» датчика (рис. 5.2). Плата просыпается раз в 30 секунд *или* от кнопки, измеряет напряжение, коротко мигает светодиодом и снова засыпает. Счётчики пробуждений хранятся в RTC-памяти.

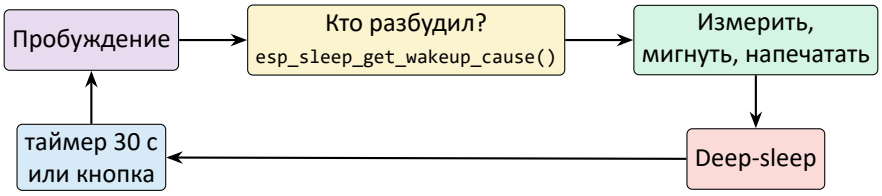


Рис. 9.4: Жизненный цикл «датчика на батарейке»

Листинг 9.3: Датчик на батарейке

```
// «Датчик на батарейке»: просыпается раз в 30 с или от кнопки,
// измеряет напряжение, мигает светодиодом и снова засыпает
#include "driver/rtc_io.h"

const int LED_PIN = 4;
const int BTN_PIN = 5;
const int POT_PIN = 1;                          // «датчик» -- потенциометр
const uint64_t SLEEP_US = 30ULL * 1000000;      // 30 секунд

RTC_DATA_ATTR int wakeCount = 0;                // эти переменные
RTC_DATA_ATTR int buttonCount = 0;              // переживают сон

void setup() {
  Serial.begin(115200);
  delay(300);
  rtc_gpio_deinit((gpio_num_t)BTN_PIN);         // вернуть кнопку в обычный режим
  pinMode(LED_PIN, OUTPUT);
  wakeCount++;

  if (esp_sleep_get_wakeup_cause() == ESP_SLEEP_WAKEUP_EXT0) {
    buttonCount++;                              // разбудили кнопкой
    Serial.println("Меня разбудила кнопка!");
  }
  int mv = analogReadMilliVolts(POT_PIN);
  Serial.printf("Пробуждение %d (кнопкой: %d), U = %d мВ\n", wakeCount, buttonCount, mv);

  digitalWrite(LED_PIN, HIGH);                  // короткая вспышка: «я работаю»
  delay(100);
  digitalWrite(LED_PIN, LOW);

  esp_sleep_enable_timer_wakeup(SLEEP_US);      // будильник 1: таймер
  rtc_gpio_pullup_en((gpio_num_t)BTN_PIN);      // будильник 2: кнопка
  rtc_gpio_pulldown_dis((gpio_num_t)BTN_PIN);
  esp_sleep_enable_ext0_wakeup((gpio_num_t)BTN_PIN, 0);
  Serial.println("Сплю...");
  Serial.flush();
  esp_deep_sleep_start();
}

void loop() {}                                  // сюда программа не доходит
```

Если не работает

- **Кнопка не будит плату** — кнопка не на RTC-выводе (**GPIO0–GPIO21**) или не включена RTC-подтяжка.
- **После сна пропал Serial Monitor** — это нормально для нативного USB: порт исчезает, пока чип спит.

- **Плату не получается прошить** — она спит; войдите в загрузчик кнопками BOOT и RST.

Итоги урока

- Deep-sleep снижает ток до микроампер, но пробуждение — это перезапуск программы.
- Нужные данные храним в `RTC_DATA_ATTR`, причину пробуждения узнаём через `esp_sleep_get_wakeup_cause()`.
- Будить можно таймером, уровнем на RTC-выводе, сенсорным входом.

Проверь себя

1. Чем пробуждение из deep-sleep отличается от выхода из light-sleep?
2. Какие данные переживают deep-sleep и как их объявить?
3. От чего сильнее всего зависит время работы от батареи?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Уложите плату в deep-sleep на 10 секунд и выведите сообщение после пробуждения.
2. ★ Сохраните в RTC-памяти счётчик пробуждений и выводите его после каждого сна.
3. ★★ Добавьте пробуждение по кнопке и выводите причину пробуждения (таймер или кнопка).
4. ★★ Измерьте ток платы мультиметром в работе и во сне и сравните с расчётом из урока.
5. ★★★ Добавьте второй источник пробуждения — таймер на 1 час — и при таком пробуждении только печатайте статистику и засыпайте снова.
6. ★★★★ Рассчитайте время работы от аккумулятора 1000 mAh для вашего режима «5 секунд работы, 5 минут сна» и проверьте расчёт по измеренным токам.

Урок 10. SPI: быстрая периферия и SD-карты

Цель	подключать быстрые устройства по SPI и безопасно делить одну шину между несколькими микросхемами
Опирается на	урок 6 (I^2C), урок 2 (провода и логические уровни)
Результат	плата сохраняет журнал событий на SD-карту
Понадобится	модуль microSD с логикой 3,3 В, карта памяти, короткие провода

10.1 Чем SPI отличается от I^2C

SPI — синхронная шина без адресов. Ведущий (ESP32-S3) создаёт тактирование `SCK`; линии `MOSI` и `MISO` несут данные к устройству и от него. Для каждого ведомого нужен свой **CS** (Chip Select): только устройство с активным CS имеет право отвечать.

	I^2C	SPI
Провода	SDA, SCL; адрес в пакете	SCK, MOSI, MISO + отдельный CS на устройство
Скорость	обычно 100 кГц–1 МГц	единицы–десятки МГц
Сильная сторона	много простых датчиков	дисплеи, SD-карты, АЦП с быстрым потоком данных

Осторожно

SPI не задаёт единой распиновки. В Arduino-ESP32 сигналы можно назначить программно, но у модуля SD и платы должны совпадать *все четыре* названия. Питание и логика также только 3,3 В.

10.2 Одна шина, несколько устройств

К линиям SCK, MOSI и MISO параллельно подключают все устройства. CS у каждого свой. Перед обменом библиотека опускает нужный CS в LOW, после — возвращает HIGH. Если два CS активны одновременно, два выхода MISO могут физически спорить друг с другом: результатом будут ошибки или повреждение.

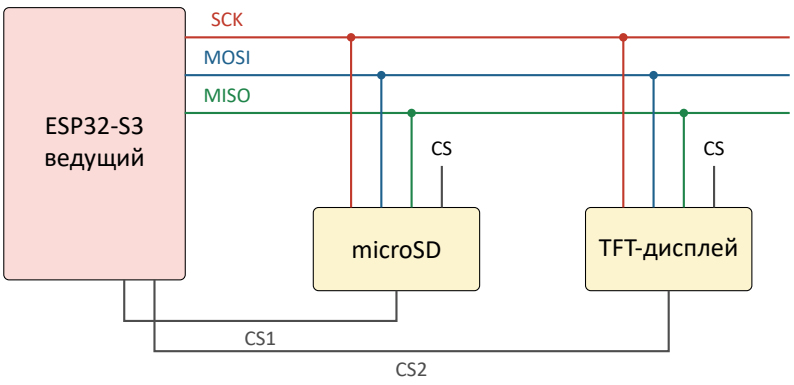


Рис. 10.1: Общая шина SPI: линии SCK, MOSI и MISO идут ко всем устройствам, а линия выбора CS у каждого устройства своя. В каждый момент активен (LOW) только один CS

Выберите, например, GPIO12 (SCK), GPIO11 (MOSI), GPIO13 (MISO), GPIO10 (CS SD), если они свободны на вашей ревизии. Перед подключением всегда сверяйтесь с pinout именно своего моду-

ля: у плат с PSRAM часть выводов занята памятью.

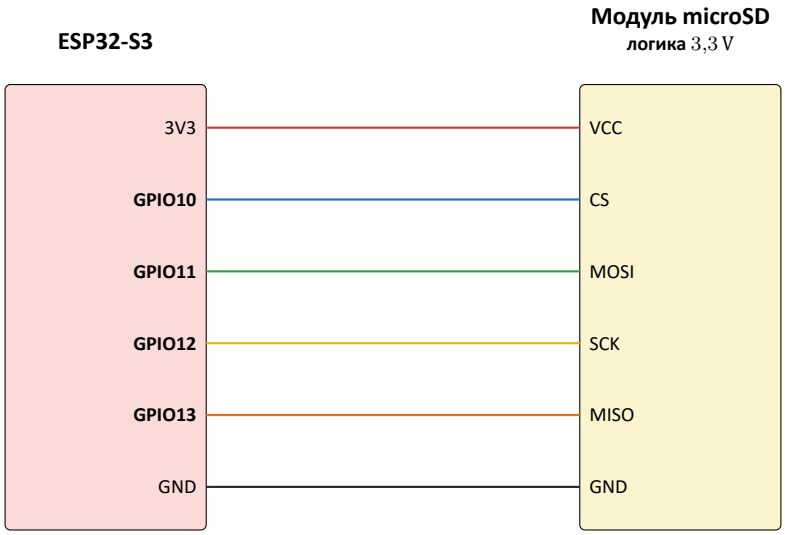


Рис. 10.2: Подключение модуля microSD к SPI-шине: **GPIO12** — SCK, **GPIO11** — MOSI, **GPIO13** — MISO, **GPIO10** — CS. Питание 3,3 V и общая земля. Порядок контактов на вашем модуле может отличаться: ориентируйтесь на надписи

10.3 Первый журнал на SD-карте

Листинг 10.1: Инициализация SPI и запись строки в файл

```
#include <SPI.h>
#include <SD.h>

constexpr int PIN_SCK  = 12;
constexpr int PIN_MISO = 13;
constexpr int PIN_MOSI = 11;
constexpr int PIN_CS   = 10;  // отдельный CS только для SD-карты

SPIClass spi(FSPI);

void setup() {
  Serial.begin(115200);
  spi.begin(PIN_SCK, PIN_MISO, PIN_MOSI, PIN_CS);

  // Начинаем с 4 МГц: это надёжнее для макетной платы и длинных проводов.
  if (!SD.begin(PIN_CS, spi, 4000000)) {
    Serial.println("SD-карта не найдена: проверьте питание, CS и общую землю");
    return;
  }

  File log = SD.open("/log.txt", FILE_APPEND);
  if (log) {
    log.println("ESP32-S3 запустился");
    log.close();                // обязательно: данные уйдут из буфера в карту
    Serial.println("Строка записана в /log.txt");
  }
}

void loop() {}
```

Начинайте с 4 MHz. Макетная плата, длинные провода и плохое питание добавляют паразитную

ёмкость и искажают фронты. Лишь после стабильной работы поднимайте частоту небольшими шагами. Частота 20 – 40 MHz, которую часто обещает модуль, обычно достижима только на короткой разводке платы.

10.4 Разбор ошибки: «карта то работает, то нет»

Проверьте в таком порядке:

1. Общая земля, питание 3,3 В и настоящий преобразователь уровней на модуле SD. Модули, рассчитанные на Arduino Uno, иногда требуют 5 В питания, но их входы всё равно нельзя считать безопасными для ESP32 без проверки схемы.
2. CS: он не должен быть оставлен «в воздухе»; при старте задайте его OUTPUT и HIGH.
3. Распиновка MOSI/MISO: они легко меняются местами. MISO идёт *к* ESP32.
4. Понизьте SPI-частоту до 1 MHz, укоротите соединения и добавьте развязывающий конденсатор $0{,}1$ µF около модуля.
5. Закрывайте файл после записи. Иначе сообщение остаётся в буфере и пропадёт при отключении питания.

Итоги урока

- SPI быстр, но требует отдельного CS для каждого ведомого устройства.
- На макетной плате сначала выбираем низкую частоту и короткие провода.
- Ошибка SD чаще всего означает питание, CS, перепутанные линии или незакрытый файл.

Проверь себя

1. Зачем каждому SPI-устройству своя линия CS?
2. Почему на макетной плате начинают с низкой частоты SPI?
3. Почему файл обязательно закрывают после записи?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Нарисуйте таблицу выводов SPI вашей платы: MOSI, MISO, SCK, CS.
2. ★ Запустите пример с SD-картой и выведите её размер и тип.
3. ★★ Записывайте в `/log.csv` время работы и показания потенциометра, добавляя заголовок только в новый файл.
4. ★★ Прочитайте файл обратно и выведите его в Serial Monitor построчно.
5. ★★★ Подключите TFT-дисплей и SD-карту к общей шине: нарисуйте таблицу CS и проверьте, что они не мешают друг другу.
6. ★★★★ Найдите максимальную надёжную частоту SPI для своей сборки, увеличивая её и записывая результаты в таблицу.

Урок 11. Память: RAM, Flash, PSRAM и NVS

Цель	выбирать место для данных и сохранять настройки так, чтобы не изнашивать Flash
Опирается на	урок 7 (задачи и общий доступ), урок 9 (данные после сна)
Результат	яркость и имя устройства переживают отключение питания
Понадобится	ничего нового

11.1 У каждой памяти своя роль

RAM хранит переменные и стек задач: она быстрая, но очищается при сбросе. Внешняя PSRAM даёт больше места для больших буферов, кадров камеры и веб-страниц, но медленнее и есть не на каждой плате. Flash содержит прошивку и переживает отключение питания. Внутри Flash выделен раздел NVS — маленькое журналируемое хранилище «ключ–значение» для настроек.

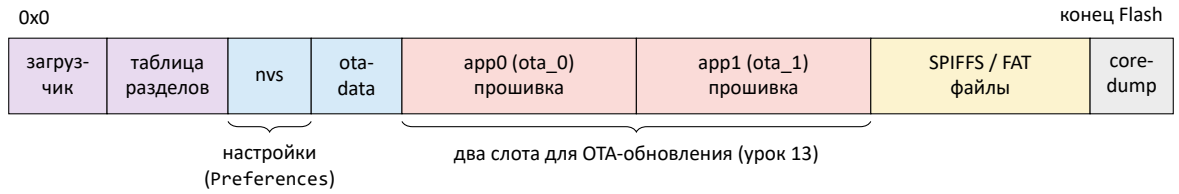


Рис. 11.1: Разметка Flash по умолчанию в Arduino-ESP32 (схема, не в масштабе). Выбрать другую таблицу разделов можно в Tools → Partition Scheme

Где	Подходит для	Не подходит для
RAM	текущие измерения, очереди, стек	сохранения после питания
PSRAM	большой временный буфер	критичного к задержкам кода
RTC RAM	счётчик deep-sleep, короткое состояние	сохранения после обычного сброса
NVS во Flash	настройки, калибровка, пароль Wi-Fi	частой телеметрии и журнала
SD / сервер	большой журнал и файлы	мгновенного доступа без питания

11.2 Настройка в NVS

Класс `Preferences` в Arduino-ESP32 безопасно работает с NVS. Пространство имён `"app"` отделяет ваши ключи от других частей прошивки; ключи делайте короткими и постоянными.

Листинг 11.1: Чтение значения по умолчанию и редкое сохранение

```
#include <Preferences.h>

Preferences prefs;
uint8_t brightness;

void setup() {
  Serial.begin(115200);
  prefs.begin("app", false);                  // пространство имён, запись разрешена
  brightness = prefs.getUChar("level", 50);   // 50 -- значение по умолчанию
  Serial.printf("Сохранённая яркость: %u%%\n", brightness);
}

void loop() {
  // Вызывайте только когда пользователь закончил менять настройку,
```

```
  // а не на каждом проходе loop().
  if (Serial.available()) {
    int newLevel = Serial.parseInt();
    if (newLevel >= 0 && newLevel <= 100) {
      brightness = newLevel;
      prefs.putUChar("level", brightness);
      Serial.println("Яркость сохранена в NVS");
    }
  }
}
```

Осторожно

Flash нельзя записывать на каждом проходе `loop()`. Ресурс ячеек ограничен: запись уровня потенциометра сотни раз в секунду быстро превратит полезный раздел в источник ошибок. Сохраняйте настройку по действию пользователя, после тайм-аута «ручка перестала двигаться» или с ограничением, например, не чаще раза в минуту.

11.3 PSRAM и большие буферы

Наличие PSRAM нужно проверить: `psramFound()` возвращает `true`, а размер — `ESP.getPsramSize()`. Большой буфер выделяют через `ps_malloc()`, проверяют на `nullptr` и освобождают `free()`. Не переносите туда бездумно всё подряд: стек задач и маленькие горячие структуры лучше оставить в обычной RAM. Утечки в долгоживущем веб-сервере удобно ловить сравнением `ESP.getFreeHeap()` до и после сотен запросов.

11.4 Разбор ошибки: настройки «забываются»

Сначала убедитесь, что `prefs.begin("app", false)` открыт в режиме записи, а ключ при чтении и записи совпадает. Затем посмотрите на результат `put...`: он возвращает число записанных байтов. Не вызывайте `clear()` при обычном старте — это функция сброса настроек. Наконец, не путайте NVS с RTC-памятью: `RTC_DATA_ATTR` переживает deep-sleep, но не отключение питания.

Итоги урока

- RAM — для работы сейчас, NVS — для редких постоянных настроек, RTC RAM — между снами.
- PSRAM полезна для больших буферов, но сначала проверяем её наличие и результат выделения.
- Редкая осмысленная запись бережёт Flash и делает поведение устройства предсказуемым.

Проверь себя

1. Где хранить счётчик пробуждений, пароль Wi-Fi и большой буфер изображения?
2. Почему Flash нельзя записывать на каждом проходе `loop()`?
3. Как проверить, что PSRAM есть и память действительно выделилась?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Сохраните в NVS число запусков и выведите его после перезагрузки.

2. ★ Сбросьте плату питанием и убедитесь, что значение сохранилось. Чем NVS отличается от обычной переменной?
3. ★★ Сохраните последнее состояние светодиода (вкл/выкл и яркость) и восстановите его после включения.
4. ★★ Реализуйте команду `factory`: удалите пространство имён `app` после подтверждения.
5. ★★★ Добавьте отложенную запись яркости через 3 секунды после последнего поворота ручки, чтобы не изнашивать Flash.
6. ★★★★ Выделите в PSRAM буфер на 1 МБ, заполните его и проверьте содержимое; сравните скорость с обычной кучей.

Урок 12. IoT: ESP-NOW, MQTT и Bluetooth LE

Цель	выбрать подходящий беспроводной протокол и обмениваться данными без блокировки устройства
Опирается на	урок 8 (Wi-Fi), урок 7 (задачи и очереди)
Результат	плата становится беспроводным датчиком, частью умного дома или устройством для настройки с телефона
Понадобится	вторая ESP32-S3; для MQTT — брокер; для BLE — телефон

12.1 Три задачи — три протокола

ESP-NOW нужен для коротких сообщений между ESP без роутера: пульт, кнопка, распределённый датчик. MQTT нужен, когда независимые устройства должны общаться через брокер и Home Assistant. BLE удобен для настройки телефоном рядом с устройством и экономит энергию. Это не взаимозаменяемые технологии: выбирайте протокол по пути данных, а не по модному названию.

Протокол	Хорош для	Главное ограничение
ESP-NOW	коротких команд, малой задержки, без роутера	обе платы на одном Wi-Fi-канале
MQTT	умного дома, журналов, нескольких клиентов	нужен Wi-Fi и брокер
BLE GATT	настройки и данные рядом со смартфоном	небольшие пакеты, не «обычный Bluetooth Serial»

12.2 ESP-NOW: пакет по MAC-адресу

Отправитель добавляет приёмник как peer и посылает структуру фиксированного размера. В реальном проекте добавьте версию протокола, счётчик `sequence` и контроль диапазонов на приёмнике. Коллбэк «доставлено» означает подтверждение на уровне радио, но не заменяет подтверждение, что приложение действительно обработало команду.

Листинг 12.1: Отправка структуры раз в секунду

```
#include <WiFi.h>
#include <esp_now.h>

struct __attribute__((packed)) SensorPacket {
  uint32_t sequence;
  float temperature;
};

SensorPacket packet {0, 23.5f};
uint8_t receiver[] = {0x24, 0x6F, 0x28, 0x00, 0x00, 0x00}; // замените MAC адресом приёмника

// В ядре 3.3 изменился первый аргумент колбэка отправки
#if ESP_ARDUINO_VERSION >= ESP_ARDUINO_VERSION_VAL(3, 3, 0)
void onSent(const esp_now_send_info_t*, esp_now_send_status_t status) {
#else
void onSent(const uint8_t*, esp_now_send_status_t status) {
#endif
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "приёмник подтвердил (ACK)" : "нет ACK");
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) { Serial.println("ESP-NOW не запустился"); return; }
  esp_now_register_send_cb(onSent);

  esp_now_peer_info_t peer{};
  memcpy(peer.peer_addr, receiver, 6);
  peer.channel = 0;                  // текущий канал; у обеих плат он должен совпадать
  peer.encrypt = false;
  esp_now_add_peer(&peer);
}

void loop() {
  packet.sequence++;
  esp_now_send(receiver, reinterpret_cast<uint8_t*>(&packet), sizeof(packet));
  delay(1000);
}
```

Осторожно

ESP-NOW и обычный Wi-Fi используют один радиоканал. Если плата подключилась к роутеру на канале 6, вторая плата с ESP-NOW должна оказаться на том же канале. MAC-адрес печатается через `WiFi.macAddress()`; не копируйте адрес из примера. В ядре 3.3 изменился первый аргумент колбэка отправки, поэтому пример выбирает нужный вариант через `ESP_ARDUINO_VERSION`.

12.3 MQTT: темы вместо прямых соединений

В MQTT устройство подключается к брокеру и публикует сообщения в *topics*, например `home/esp-01/state`. Другой клиент подписывается на эту тему; команда приходит в `home/esp-01/set`. Для состояния удобно оставить retained-сообщение, а для команд — нет. Используйте компактный JSON, например `{"on":true,"level":75}`, и проверяйте каждое поле до применения. Для Arduino обычно устанавливают библиотеку PubSubClient через Library Manager.

Подключение к брокеру не помещают в бесконечный цикл: выполняйте одну попытку по таймеру, увеличивайте интервал после ошибок и давайте работать кнопке, экрану и `client.loop()`. Логин/пароль и TLS не делают публичный брокер бесконечным ресурсом: соблюдайте его ограничения и не публикуйте данные десятки раз в секунду без причины.

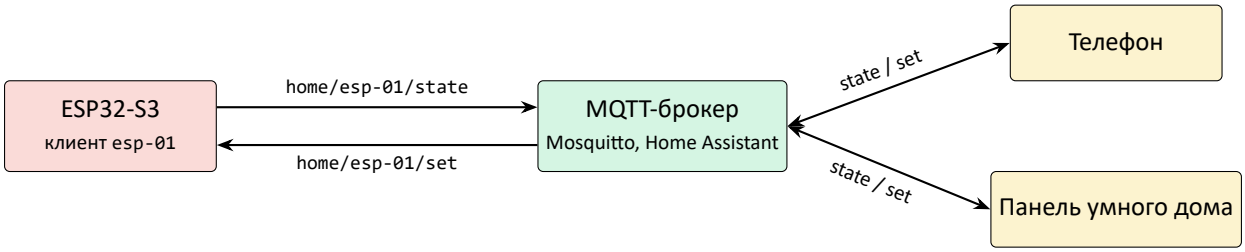


Рис. 12.1: Клиенты MQTT не соединяются друг с другом напрямую: они публикуют сообщения в темы брокера и подписываются на нужные темы

Листинг 12.2: MQTT-клиент: неблокирующее переподключение, подписка и публикация состояния

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* SSID   = "MyNetwork";
const char* PASS   = "MyPassword";
const char* BROKER = "192.168.1.10";            // адрес вашего брокера

WiFiClient net;
PubSubClient mqtt(net);
uint32_t lastTry = 0, lastPub = 0;

void onMessage(char* topic, byte* payload, unsigned int len) {
  String msg;
  for (unsigned int i = 0; i < len; i++) msg += (char)payload[i];
  Serial.printf("%s -> %s\n", topic, msg.c_str());  // здесь проверяем и применяем команду
}

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);                       // подключение идёт в фоне
  mqtt.setServer(BROKER, 1883);
  mqtt.setCallback(onMessage);
}

void loop() {
  if (!mqtt.connected()) {
    // одна попытка раз в 5 секунд, а не бесконечный цикл ожидания
    if (WiFi.status() == WL_CONNECTED && millis() - lastTry > 5000) {
      lastTry = millis();
      if (mqtt.connect("esp-01")) mqtt.subscribe("home/esp-01/set");
    }
    return;
  }
  mqtt.loop();                                  // приём сообщений и keep-alive
  if (millis() - lastPub > 10000) {
    lastPub = millis();
    String state = String("{\"uptime\":") + millis() / 1000 + "}";
    mqtt.publish("home/esp-01/state", state.c_str(), true);  // true = retained
  }
}
```

12.4 BLE: сервисы и характеристики

BLE-сервер публикует сервис (например, «настройки устройства»), а внутри — характеристики: яркость, состояние, имя. У каждой есть UUID и права read/write/notify. Стандартный минимальный MTU даёт всего 20 байтов полезных данных в одном уведомлении, поэтому длинный JSON нужно фрагментировать или согласовать увеличенный MTU. Для нового проекта предпочтительна библиотека NimBLE: она экономнее старого BLE-стека. В примере используется встроенная в Arduino-ESP32 библиотека `BLEDevice`: ей не нужна отдельная установка.

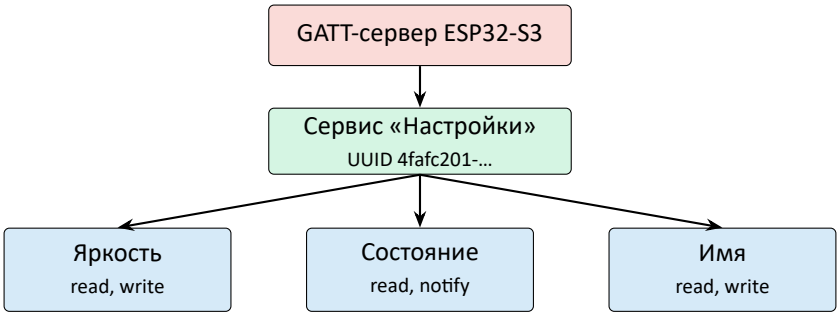


Рис. 12.2: Устройство BLE описывает себя деревом: сервисы, а в них — характеристики со своими правами

Пример ниже создаёт сервис с одной характеристикой «Яркость». Запишите в неё число 0–100 из приложения nRF Connect на телефоне — светодиод на **GPIO4** изменит яркость.

Листинг 12.3: BLE-сервер с записываемой характеристикой яркости

```
#include <BLEDevice.h>
#include <BLEServer.h>

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define LEVEL_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"

const int LED_PIN = 4;

class LevelCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* c) override {
    String v = c->getValue();                    // текст от телефона, например "75"
    int level = constrain(v.toInt(), 0, 100);    // проверяем диапазон до применения
    ledcWrite(LED_PIN, level * 255 / 100);
    Serial.printf("Яркость: %d%%\n", level);
  }
};

void setup() {
  Serial.begin(115200);
  ledcAttach(LED_PIN, 5000, 8);

  BLEDevice::init("ESP32-S3-Lab");               // имя, которое увидит телефон
  BLEServer* server = BLEDevice::createServer();
  BLEService* service = server->createService(SERVICE_UUID);
  BLECharacteristic* level = service->createCharacteristic(
      LEVEL_UUID, BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE);
  level->setCallbacks(new LevelCallbacks());
  level->setValue("0");
  service->start();

  BLEDevice::getAdvertising()->addServiceUUID(SERVICE_UUID);
  BLEDevice::startAdvertising();                 // плата видна в списке устройств
```

```
}

void loop() { delay(1000); }
```

12.5 Разбор ошибки: сеть «подвисает»

Не вызывайте сетевой код из ISR и не держите мьютекс во время Wi-Fi/MQTT-запроса. Обработчик кнопки кладёт событие в очередь, а сетевая задача публикует его позднее. Для MQTT храните последнее состояние и повторяйте подключение с задержкой. Для BLE проверяйте длину входного значения до копирования в буфер. Так один неверный пакет не заблокирует всё устройство.

Итоги урока

- ESP-NOW — прямая короткая связь ESP; MQTT — обмен через брокер; BLE — близкая настройка.
- Радиоканал у ESP-NOW должен совпадать, а MAC-адреса и входные данные нужно проверять.
- Сетевые операции выполняются задачей с тайм-аутами, а не в ISR или бесконечном ожидании.

Проверь себя

1. Какой протокол выбрать для пульта без роутера, для умного дома и для настройки с телефона?
2. Что означает retained-сообщение в MQTT?
3. Почему сетевой код нельзя вызывать из обработчика прерывания?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите ESP-NOW между двумя платами и убедитесь, что сообщение доходит.
2. ★ Найдите MAC-адрес приёмника и объясните, зачем он нужен передатчику.
3. ★★ Сделайте ESP-NOW-пульт: короткое нажатие отправляет переключение, длинное — яркость 100%.
4. ★★ Опишите четыре MQTT-темы устройства: state, set, availability и diagnostics, и решите, какие из них retained.
5. ★★★ Подключитесь к MQTT-брокеру: публикуйте значение потенциометра раз в 5 секунд и принимайте команду включения светодиода.
6. ★★★★ Спроектируйте BLE-сервис с характеристиками имени, яркости и версии прошивки; назначьте UUID и права доступа.

Урок 13. HTTPS и OTA: защищённое устройство

Цель	проверять подлинность сервера и обновлять прошивку без USB, не оставляя устройство «кирпичом»
Опирается на	урок 8 (Wi-Fi), урок 11 (Flash и NVS)
Результат	устройство безопасно получает данные и может обновляться по сети
Понадобится	доступ к защищённой Wi-Fi-сети; для OTA — стабильное питание

13.1 TLS не равен «порт 443»

HTTPS защищает содержимое канала и проверяет, что устройство разговаривает с тем сервером. Для проверки ESP32-S3 нужна корневая цепочка сертификата (Root CA) и правильное время: сертификат имеет срок действия. Если просто вызвать `client.setInsecure()`, соединение заработает, но устройство перестанет отличать ваш сервер от подмены — это допустимо только для краткого лабораторного опыта.

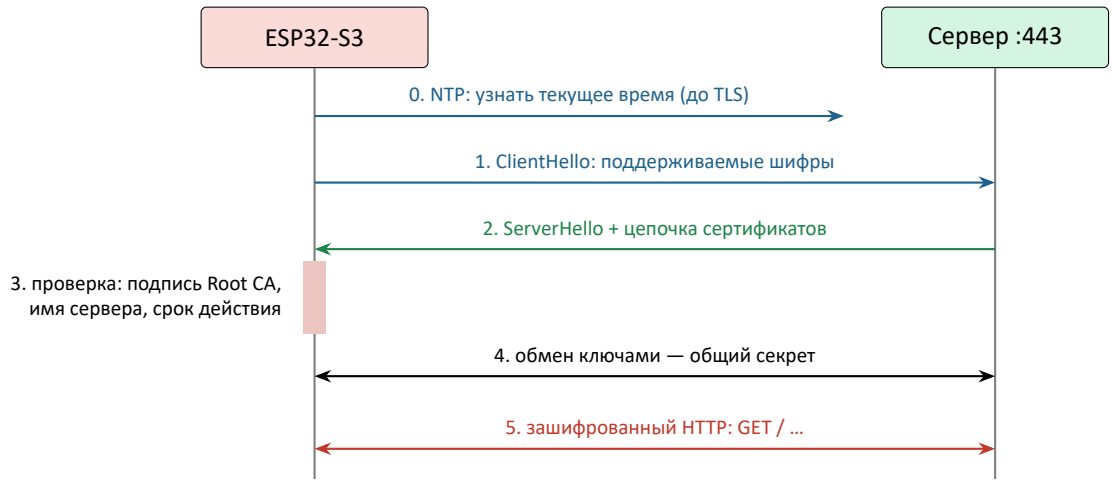


Рис. 13.1: Упрощённая схема TLS-соединения. Без точного времени шаг 3 не проверит срок действия сертификата, а без Root CA — подлинность сервера

Листинг 13.1: Синхронизация времени и проверка Root CA

```
#include <WiFi.h>
#include <WiFiClientSecure.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

const char* rootCA = R"EOF(
-----BEGIN CERTIFICATE-----
... корневой сертификат центра сертификации вашего сервера ...
-----END CERTIFICATE-----
)EOF";

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
```

```
  if (WiFi.status() != WL_CONNECTED) { Serial.println("Нет Wi-Fi"); return; }

  configTime(0, 0, "pool.ntp.org", "time.nist.gov");
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo, 10000)) {
    Serial.println("Нет точного времени: TLS-проверка сертификата невозможна");
    return;
  }

  WiFiClientSecure client;
  client.setCACert(rootCA);              // не используйте setInsecure() в готовом устройстве
  if (!client.connect("example.com", 443)) {
    Serial.println("TLS-соединение не установлено");
    return;
  }
  client.print("GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n");
  uint32_t t0 = millis();
  while ((client.connected() || client.available()) && millis() - t0 < 10000) {
    if (client.available()) Serial.write(client.read());  // печатаем ответ сервера
  }
  client.stop();
}

void loop() {}
```

Сертификат берите у центра сертификации сервера и обновляйте с запасом до истечения. Не кладите в прошивку пароль Wi-Fi, API-токен и приватный ключ публичного проекта. Для устройства, которое будет продаваться, включают Secure Boot, Flash Encryption и подписанные образы в ESP-IDF.

13.2 OTA: два слота вместо одной ставки

OTA записывает новый образ в неиспользуемый раздел Flash. После перезапуска загрузчик запускает новый слот. Если в загрузчике включён откат (rollback), слот становится постоянным только после того, как новая прошивка подтвердила работоспособность. Это требует таблицы разделов с двумя OTA-слотами и разделом `otadata`. Нельзя перезаписывать текущую работающую прошивку.

Листинг 13.2: Минимальный OTA-сервис Arduino для локальной сети

```
#include <WiFi.h>
#include <ArduinoOTA.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
  Serial.println(WiFi.localIP());

  ArduinoOTA.setHostname("esp32s3-lab");
  ArduinoOTA.setPassword("change-me");  // без пароля прошить плату может любой в сети
  ArduinoOTA.onStart([] { Serial.println("Начато OTA-обновление"); });
  ArduinoOTA.onError([](ota_error_t e) { Serial.printf("OTA ошибка: %u\n", e); });
  ArduinoOTA.begin();
```

```
}

void loop() {
  ArduinoOTA.handle();  // вызывается часто; не блокируйте loop() длинными delay()
}
```

Этот вариант удобен для лаборатории, но не является удалённым промышленным обновлением: добавьте пароль, проверку подписи/хэша образа, HTTPS-скачивание и журнал версии. После прошивки устройство должно самостоятельно проверить датчики, сеть и ключевые задачи, а затем подтвердить загрузчику новую версию. Если прошивка падает до подтверждения, загрузчик возвращается к прошлому слоту — но только при включённой опции `CONFIG_BOOTLOADER_APP_ROLLBACK_ENABLE`. В стандартной сборке Arduino-ESP32 она выключена, поэтому для настоящего изделия откат настраивают в ESP-IDF.

13.3 Разбор ошибки: HTTPS или OTA не запускается

При ошибке TLS сначала выведите текущую дату: почти всегда причина — устройство не дождалось NTP, не видит DNS или имеет старый Root CA. Не заменяйте диагностику `setInsecure()`. При обрыве OTA не выключайте питание вручную: двухслотовая разметка сохраняет старую прошивку. Если устройство больше не отвечает после эксперимента, зайдите в загрузчик кнопками BOOT и RST из урока 0 и прошейте USB-кабелем.

Итоги урока

- TLS требует Root CA и корректных часов; `setInsecure()` не решает задачу безопасности.
- Безопасное OTA — это два слота, проверка образа и подтверждение успешного запуска.
- Обновление не должно останавливать контроль нагрузки и не должно зависеть от бесконечного ожидания сети.

Проверь себя

1. Зачем устройству точное время для HTTPS?
2. Почему `setInsecure()` не годится для готового устройства?
3. Зачем OTA-обновлению два слота прошивки?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Объясните своими словами, чем HTTPS отличается от HTTP и что проверяет сертификат.
2. ★ Выведите время UTC после NTP: без него HTTPS не должен запускаться.
3. ★★ Добавьте в веб-страницу из урока 8 номер версии, дату сборки и свободное место Flash.
4. ★★ Выполните запрос HTTPS с проверкой сертификата и проверьте, что при неверном времени он завершается ошибкой.
5. ★★★ Составьте checklist OTA: питание, версия, хэш, тест после перезапуска, откат и журнал.
6. ★★★★ Сделайте обновление по сети: плата скачивает прошивку, проверяет и подтверждает её после перезагрузки, а при сбое откатывается.

Урок 14. Профессиональная диагностика: Guru Meditation

Цель	превратить пугающий лог падения в конкретную строку и предотвратить повторение ошибки
Опирается на	все предыдущие уроки, особенно задачи, память и сеть
Результат	у устройства есть понятный журнал состояния и путь к поиску редких сбоев
Понадобится	Serial Monitor; файл собранной прошивки ELF для декодирования адресов

14.1 Паника — это полезный отчёт

Когда ESP-IDF обнаруживает критическую ошибку, он печатает `Guru Meditation Error`, причину, регистры и backtrace, а затем обычно перезапускает чип. Не удаляйте этот лог: скопируйте его целиком вместе со строкой версии прошивки. Самые частые причины для начинающего проекта — обращение по нулевому указателю, выход за границы массива, переполнение стека задачи и использование объекта после `free()`.

Симптом	Вероятная причина	Первое действие
`LoadProhibited`	нулевой/испорченный указатель	проверить указатель и место создания объекта
`Stack canary`	стек задачи мал	измерить High Water Mark, увеличить стек
WDT timeout	задача/ISR не отдаёт CPU	убрать долгую работу, добавить ожидание
Brownout detector	питание просело	проверить источник, конденсаторы и нагрузку

14.2 Как читать backtrace

Адреса в backtrace сами по себе не говорят номер строки. После каждой сборки IDE создаёт файл `.elf`, в котором есть соответствие адресов исходному коду. ESP Exception Decoder или утилита `xtensa-esp32s3-elf-addr2line` переводят адреса в имена функций и номера строк *только для той же сборки*. Не декодируйте лог от старой прошивки новым ELF: результат будет правдоподобным, но ложным.
Рабочий порядок такой:

1. сохраните полный лог, версию, причину сброса и шаги, после которых он случился;
2. найдите первую пользовательскую функцию в декодированном стеке, а не последнюю системную;
3. добавьте проверку входных данных и краткие диагностические сообщения вокруг места ошибки;
4. повторите сценарий много раз и только затем считайте исправление подтверждённым.

14.3 Разбор настоящего сбоя

Программа ниже обращается к объекту, который так и не был создан:

Листинг 14.1: Скетч с ошибкой: указатель `sensor` равен `nullptr`

```
struct Sensor { int pin; int value; };
Sensor* sensor = nullptr;               // ошибка: объект так и не создан

int readSensor(Sensor* s) {
  s->value = analogRead(s->pin);        // s == nullptr -> LoadProhibited
  return s->value;
}

void setup() { Serial.begin(115200); }

void loop() {
  Serial.println(readSensor(sensor));
  delay(1000);
}
```

Через секунду после запуска в Serial Monitor появится отчёт (сокращён):

```
Guru Meditation Error: Core  1 panic'ed (LoadProhibited). Exception was unhandled.
Core  1 register dump:
PC      : 0x42002a1c  PS      : 0x00060830  A0      : 0x82002b45  A1      : 0x3fcebd80
...
EXCVADDR: 0x00000000  LBEG    : 0x40056f5c  LEND    : 0x40056f72  LCOUNT  : 0x00000000

Backtrace: 0x42002a19:0x3fcebd80 0x42002b42:0x3fcebda0 0x4200593e:0x3fcebdc0
Rebooting...
```

- `LoadProhibited` — чтение из запрещённого адреса; `EXCVADDR: 0x00000000` уточняет: читали по нулевому адресу, то есть через нулевой указатель.
- `Core 1` — сбой случился на ядре 1, где работает `loop()`.
- `Backtrace` после декодирования ELF-файлом той же сборки превращается в строки кода:

```
0x42002a19: readSensor(Sensor*) at sketch.ino:5
0x42002b42: loop() at sketch.ino:11
0x4200593e: loopTask(void*) at cores/esp32/main.cpp:74
```

Первая пользовательская функция в стеке — `readSensor()`, строка 5. Исправление: создать объект (`Sensor s{1, 0}; sensor = &s;`) и проверять указатель перед использованием.

14.4 Watchdog: защита от «всё зависло»

Watchdog перезагружает ядро, если задача не даёт работать idle-задаче. В `loop()` и задачах должны быть точки, где вы отдаёте процессор: `vTaskDelay()`, ожидание очереди с тайм-аутом или короткая неблокирующая работа. В ISR допустимы лишь установка флага, счётчик или отправка события через API `...FromISR`. `delay()`, Serial, сеть и выделение памяти внутри ISR — типичный путь к WDT.

14.5 Питание — тоже часть отладки

Если лог сообщает Brownout, код может быть совершенно исправен. Wi-Fi при передаче создаёт короткие пики тока; слабый USB-кабель, маленькая батарея или питание сервопривода от 3V3 платы дают провал. Сначала отключите мощную нагрузку, измерьте напряжение возле ESP32 и объедините земли источников. Мотор, реле и сервопривод питаются от отдельного подходящего источника через ключ/драйвер, а не GPIO.

14.6 Минимальный журнал здоровья

Раз в минуту полезно печатать: время работы, причину последнего сброса, свободную RAM, `WiFi.RSSI()` и запас стека важных задач. Не печатайте эти значения из ISR и не превращайте диагностику в бесконечный поток: один компактный отчёт помогает увидеть медленную утечку памяти и падение качества сети без вреда для основной функции устройства.

Итоги урока

- Полный лог + ELF той же сборки превращают адреса backtrace в строку исходного кода.
- WDT означает блокировку CPU, а Brownout часто означает проблему питания, не алгоритма.
- Диагностируйте гипотезу воспроизводимым сценарием и проверяйте исправление повторными запусками.

Проверь себя

1. Почему для декодирования backtrace нужен ELF-файл именно той же сборки?
2. О чём говорит Brownout и где искать его причину?
3. Что нужно делать в задачах, чтобы watchdog не перезагружал плату?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выведите причину последнего сброса и расшифруйте её своими словами.
2. ★ Составьте шаблон отчёта об ошибке: версия, причина сброса, лог, шаги, питание и периферия.
3. ★★ Намеренно вызовите ошибку (деление на нуль указателя) и разберите backtrace по строкам.
4. ★★ Включите предупреждения компилятора и найдите хотя бы одно предупреждение в своём скетче.
5. ★★★ Добавьте в скетч урока 8 раз в минуту строку здоровья: время, куча, RSSI и причина сброса.
6. ★★★★ Добавьте сторожевой таймер и проверьте, что намеренно зависшая задача приводит к перезагрузке, а причина сохраняется в журнал.

Урок 15. Датчики: температура, влажность, расстояние

Цель	подключать цифровые датчики температуры, влажности и расстояния и уметь читать описание (datasheet) незнакомого датчика
Опирается на	урок 3 (подтяжка), урок 5 (делитель напряжения), урок 7 (опрос по `millis()` без пауз)
Результат	плата измеряет температуру, влажность и расстояние и печатает результаты в Serial Monitor
Понадобится	DS18B20 и резистор 4,7 kΩ; DHT22; HC-SR04 и резисторы 1 kΩ и 2 kΩ

15.1 Сначала: датчик — это собеседник со своими правилами

Цифровой датчик сам измеряет температуру или расстояние и присылает плате готовое число. Но у каждого датчика свои правила разговора: какое напряжение ему нужно, по какому проводу и в каком порядке передавать сигналы, как часто можно спрашивать. Если нарушить правила, датчик либо промолчит, либо ответит ерунду, а иногда и выйдет из строя. Все правила записаны в datasheet — паспорте датчика. Этот урок учит находить в нём главное на примере трёх популярных датчиков.

15.2 Как читать datasheet за пять минут

Не нужно читать документ целиком. Для первого подключения достаточно найти пять вещей.

1. **Питание.** Допустимый диапазон напряжения (например, 3 – 5,5 V) и потребляемый ток.
2. **Логические уровни.** Каким напряжением датчик «говорит». Если выходной сигнал выше 3,3 V, вход ESP32-S3 нужно защищать.
3. **Интерфейс.** Один провод, I²C, SPI или импульсы; есть ли подтяжка.
4. **Время и период опроса.** Сколько длится замер и как часто можно его повторять.
5. **Предельные значения** (Absolute Maximum Ratings). Выход за них — и датчик необратимо портится.

	DS18B20	DHT22	HC-SR04
Что измеряет	температура	температура и влажность	расстояние
Питание	3 – 5,5 V	3,3 – 6 V	5 V
Сигнал	по уровню питания	по уровню питания	ECHO даёт 5 V
Интерфейс	1-Wire	свой однопроводной	импульсы TRIG и ECHO
Период опроса	замер до 750 ms	не чаще раза в 2 s	не чаще раза в 60 ms
Диапазон	от −55 до +125 °C	0 – 100 % влажности	2 – 400 cm

Значения в таблице ориентировочные: у разных производителей и версий модулей они немного отличаются. Всегда сверяйтесь с документом на свой экземпляр.

15.3 DS18B20: температура по шине 1-Wire

DS18B20 — маленький датчик температуры в корпусе, похожем на транзистор, или в герметичном щупе. Он использует шину **1-Wire**: один провод данных плюс питание и земля. Выход датчика — с открытым стоком, как у I²C (урок 6), поэтому линию данных к питанию притягивает *подтягивающий резистор* 4,7 kΩ. Без него шина не работает.

У каждого DS18B20 есть уникальный 64-битный адрес, записанный на заводе. Поэтому несколько датчиков подключают к одному и тому же выводу платы и различают по адресам.

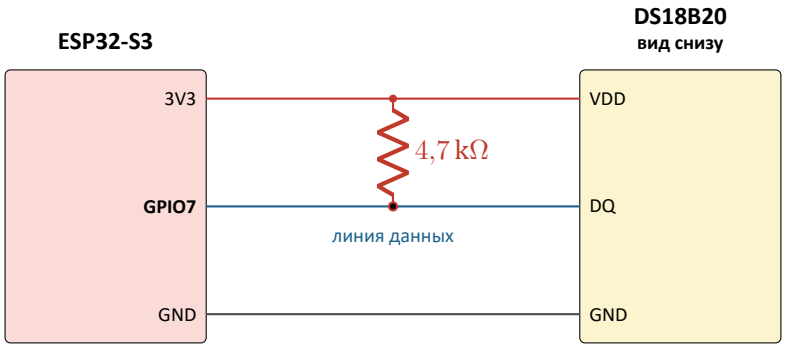


Рис. 15.1: Подключение DS18B20: линия данных одним проводом, подтяжка $4{,}7\,\text{k}\Omega$ к 3V3. Все датчики шины включаются параллельно

Листинг 15.1: DS18B20: чтение температуры

```
#include <OneWire.h>
#include <DallasTemperature.h>

const int ONE_WIRE_PIN = 7;              // DATA датчика + подтяжка 4,7 кОм к 3V3

OneWire oneWire(ONE_WIRE_PIN);
DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(115200);
  sensors.begin();
  sensors.setResolution(12);             // 12 бит: шаг 0,0625 °C, ~750 мс на замер
  Serial.printf("Найдено датчиков: %d\n", sensors.getDeviceCount());
}

void loop() {
  sensors.requestTemperatures();         // запустить замер (ждёт окончания)
  float t = sensors.getTempCByIndex(0);  // первый датчик на шине
  if (t == DEVICE_DISCONNECTED_C) {      // -127: датчик не отвечает
    Serial.println("DS18B20 не отвечает: проверьте подтяжку и провода");
  } else {
    Serial.printf("Температура: %.2f °C\n", t);
  }
  delay(2000);
}
```

Установите через Library Manager библиотеки **OneWire** и **DallasTemperature**. Значение `DEVICE_DISCONNECTED_C` (в библиотеке это -127) означает, что датчик не ответил. Не принимайте его за температуру: поставьте проверку, как в примере.

Совет

Щуп на проводах обычно имеет красный провод (VDD), чёрный (GND) и жёлтый или белый (DQ). Цвета не стандартизованы, поэтому проверьте маркировку у продавца. Если щуп сильно греется в первые секунды, то питание и земля перепутаны: сразу отключите его.

15.4 DHT22: температура и влажность

DHT22 (AM2302) измеряет температуру и относительную влажность. Он тоже общается по одному проводу, но *это не 1-Wire*: у него собственный протокол из коротких импульсов, где ноль и единица различаются длительностью, измеряемой десятками микросекунд. Поэтому библиотека на время чтения занимает процессор и не терпит пауз.
Датчик внутри медленный: новые данные у него появляются примерно раз в две секунды, а каждый обмен слегка его нагревает. Поэтому опрашивать DHT22 чаще раза в 2 s нельзя — вы получите повтор старого значения или ошибку. В примере для этого стоит проверка по `millis()`, а не `delay()`, как в уроке 1.

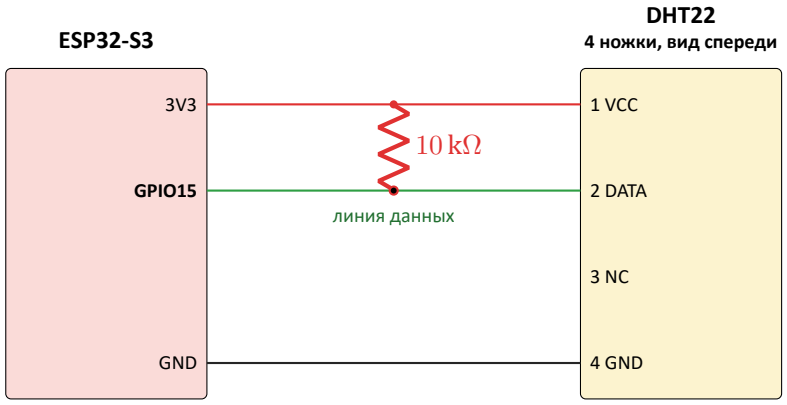


Рис. 15.2: Подключение DHT22 (датчик без платы): данные одним проводом на **GPIO15**, подтяжка 10 kΩ к 3V3. В модуле с тремя контактами (VCC, DATA, GND) резистор уже установлен

Листинг 15.2: DHT22: температура и влажность раз в 2,5 секунды

```
#include <DHT.h>

const int DHT_PIN = 15;                  // DATA датчика + подтяжка 10 кОм к 3V3
const uint32_t PERIOD_MS = 2500;         // DHT22 нельзя опрашивать чаще раза в 2 с

DHT dht(DHT_PIN, DHT22);
uint32_t lastRead = 0;

void setup() {
  Serial.begin(115200);
  dht.begin();
}

void loop() {
  if (millis() - lastRead >= PERIOD_MS) {
    lastRead = millis();
    float h = dht.readHumidity();        // влажность, %
    float t = dht.readTemperature();     // температура, °C
    if (isnan(h) || isnan(t)) {          // NaN -- чтение не удалось
      Serial.println("DHT22: ошибка чтения");
    } else {
      Serial.printf("T = %.1f °C, влажность = %.1f %%\n", t, h);
    }
  }
}
```

В Library Manager нужны **DHT sensor library** и **Adafruit Unified Sensor**. Если у вас модуль на плате с тремя контактами, подтяжка на нём уже есть; для «голого» датчика с четырьмя ногами поставьте резистор 10 kΩ между DATA и 3V3.

15.5 HC-SR04: расстояние по времени эха

Ультразвуковой дальномер HC-SR04 посылает звуковой щелчок частотой около 40 kHz и слушает эхо. Чем дальше предмет, тем дольше звук идёт туда и обратно. Плата не получает «число сантиметров» — она измеряет *длительность* импульса на выводе ECHO.

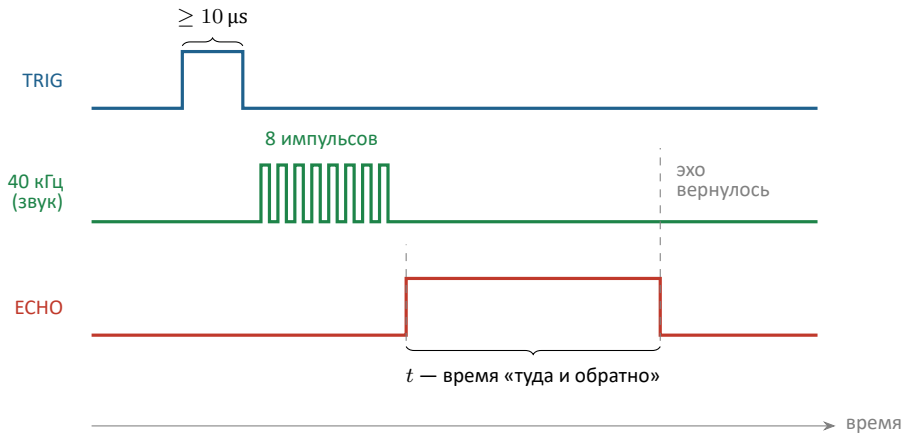


Рис. 15.3: Измерение расстояния HC-SR04. Импульс на TRIG запускает измерение, датчик излучает пачку звука, и на ECHO появляется HIGH, длительность которого равна времени пути звука туда и обратно

Скорость звука в воздухе около 343 m/s, то есть $0{,}0343$ cm/µs. Звук проходит расстояние дважды, поэтому

$$L = \frac{t \cdot 0{,}0343}{2} \ [\text{см}],$$

где t — длительность ECHO в микросекундах. Например, при $t = 1166$ µs получаем $L \approx 20$ cm. Измерить длительность импульса можно функцией `pulseIn(pin, HIGH, timeout)`. Третий аргумент — тайм-аут в микросекундах: если эхо не вернулось (предмет слишком далеко или слишком мягкий), функция вернёт 0, а не зависнет. Без тайм-аута по умолчанию ждёт целую секунду.

Осторожно

Вывод ECHO даёт сигнал 5 V, а входы ESP32-S3 рассчитаны на $3{,}3$ V. Подключать ECHO к плате **напрямую нельзя**: вход может выйти из строя. Поставьте делитель напряжения из резисторов 1 kΩ и 2 kΩ, как на рисунке ниже.

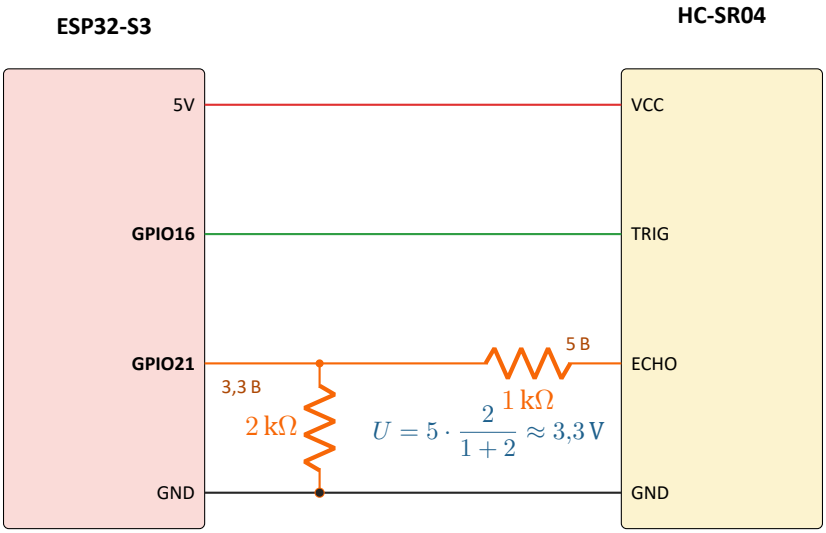


Рис. 15.4: Подключение HC-SR04. Питание 5 V, TRIG идёт на **GPIO16** напрямую, а ECHO попадает на **GPIO21** только через делитель: нижний резистор вдвое больше верхнего, поэтому на вход платы приходит две трети от 5 V

Питание самого HC-SR04 — 5 V (вывод 5V платы, который при питании от USB даёт напряжение шины). TRIG подключается напрямую: плата выдаёт $3{,}3$ V, а датчик такой уровень воспринимает как единицу.

Листинг 15.3: HC-SR04: расстояние с тайм-аутом

```
const int TRIG_PIN = 16;                  // выход: запускающий импульс
const int ECHO_PIN = 21;                  // вход: ECHO через делитель 5 В -> 3,3 В
const uint32_t ECHO_TIMEOUT_US = 30000;   // ~5 м туда и обратно; дольше не ждём

void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  digitalWrite(TRIG_PIN, LOW);
}

float readDistanceCm() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);           // импульс не короче 10 мкс запускает измерение
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  uint32_t us = pulseIn(ECHO_PIN, HIGH, ECHO_TIMEOUT_US);  // длительность эха
  if (us == 0) return -1;                 // тайм-аут: эхо не вернулось
  return us * 0.0343f / 2.0f;             // скорость звука 343 м/с = 0,0343 см/мкс
}

void loop() {
  float d = readDistanceCm();
  if (d < 0) Serial.println("Нет эха: объект далеко или нет питания");
  else       Serial.printf("Расстояние: %.1f см\n", d);
  delay(100);                             // пауза между замерами не меньше 60 мс
}
```

15.6 Самостоятельный опыт: парковочный радар

Соберите сигнализатор, как в автомобиле: чем ближе предмет, тем чаще мигает светодиод на **GPIO4**. Сделайте отдельный скетч и разберите задачу на шаги:

1. Измеряйте расстояние раз в 100 ms функцией из примера.
2. Переведите расстояние в период мигания: 100 cm и дальше — без мигания, 5 cm — период 50 ms (подсказка: `map()` и `constrain()`).
3. Мигайте через `millis()`, а не через `delay()`, иначе измерения будут «залипать».

Проверьте работу на плоском твёрдом предмете (книга, ладонь) и на мягком (свитер). У мягких предметов эхо слабее, поэтому дальность падает.

15.7 То же на ESPUI: дальномер на экране телефона

Парковочный радар из самостоятельного опыта удобно смотреть на экране: круговая шкала даёт мгновенное «далеко/близко», график показывает, как менялось расстояние. Схема подключения HC-SR04 прежняя (ECHO через делитель!), добавляется только телефон.

Листинг 15.4: ESPUI: расстояние HC-SR04 как Gauge и Graph

```
// HC-SR04: TRIG -> GPIO16, ECHO -> GPIO21 через делитель 1 кОм / 2 кОм (урок 15).
#include <WiFi.h>
#include <ESPUI.h>

const int TRIG_PIN = 16;
const int ECHO_PIN = 21;
const uint32_t ECHO_TIMEOUT_US = 30000;
uint16_t gaugeId, graphId, stateId;

float readDistanceCm() {                  // тот же код, что в листинге урока
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  uint32_t us = pulseIn(ECHO_PIN, HIGH, ECHO_TIMEOUT_US);
  if (us == 0) return -1;
  return us * 0.0343f / 2.0f;
}

void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Radar", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  gaugeId = ESPUI.addControl(ControlType::Gauge, "Расстояние, см", "0",
  ↪  ControlColor::Wetasphalt);
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, gaugeId);
  ESPUI.addControl(ControlType::Max, "", "200", ControlColor::None, gaugeId);
  stateId = ESPUI.label("Состояние", ControlColor::Peterriver, "ждём измерение");
  graphId = ESPUI.graph("История расстояния", ControlColor::Emerald);
  ESPUI.begin("ESP32-S3 radar");
  Serial.println(WiFi.softAPIP());
}

```

```
void loop() {
  static uint32_t last = 0;
  if (millis() - last >= 200) {           // не чаще 5 замеров в секунду
    last = millis();
    float d = readDistanceCm();
    if (d < 0) {
      ESPUI.updateLabel(stateId, "нет эха: далеко или нет питания");   // ошибка, а не ложный
↪   ноль
    } else {
      int cm = constrain((int)d, 0, 200);
      ESPUI.updateControlValue(gaugeId, String(cm));
      ESPUI.updateLabel(stateId, String(d, 1) + " см");
      ESPUI.addGraphPoint(graphId, cm);
    }
  }
  delay(5);
}
```

Подключитесь к сети `S3-Radar` (пароль `lesson-39-demo`). Поднесите ладонь к датчику: стрелка уйдёт к нулю, а в графике появится «ступенька». Закройте датчик мягкой тканью или отведите далеко: вместо нулевого значения на странице появится сообщение «нет эха», потому что ошибка датчика не должна выглядеть как «расстояние 0 см».

Если не работает

- **DS18B20 показывает** -127 **°C** — нет подтяжки $4{,}7\,\text{k}\Omega$, оборван DQ или перепутаны питание и земля.
- **DHT22 выдаёт NaN или повторяет значения** — опрос чаще раза в 2 с, нет подтяжки или выбран не тот тип датчика в конструкторе.
- **HC-SR04 всегда «нет эха» или показывает случайные числа** — питание не $5\,\text{V}$, перепутаны TRIG и ECHO или нет общей земли.

Итоги урока

- Перед подключением датчика найдите в datasheet питание, логические уровни, интерфейс и период опроса.
- Для DS18B20 и DHT22 нужна подтяжка линии данных, а опрашивать их надо не чаще, чем разрешает документ.
- Выход $5\,\text{V}$ (ECHO у HC-SR04) подключают к ESP32-S3 только через делитель, а время ожидания эха ограничивают тайм-аутом.

Проверь себя

1. Какие пять вещей надо найти в datasheet нового датчика?
2. Почему нельзя опрашивать DHT22 чаще одного раза в две секунды?
3. Зачем нужен делитель напряжения для ECHO и как посчитать расстояние по времени эха?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Прочитайте datasheet своего датчика и выпишите рабочее напряжение, диапазон и

точность.

2. ★ Выводите температуру DS18B20 раз в секунду в Serial Monitor.
3. ★★ Включайте светодиод на **GPIO4**, когда температура выше заданной, с гистерезисом в 1°C.
4. ★★ Измерьте расстояние HC-SR04 пять раз на одной дистанции и найдите разброс показаний.
5. ★★★ Сделайте «электронную линейку»: расстояние выводится на OLED, кнопка запоминает ноль.
6. ★★★★ Подключите к одной линии два DS18B20 и выведите температуру каждого с его адресом.

Урок 16. Силовая нагрузка: транзистор, реле, мотор

Цель	управлять нагрузкой, которой не хватает тока вывода GPIO: лампой, мотором, реле
Опирается на	урок 1 (цифровой выход), урок 4 (ШИМ), урок 5 (напряжение и ток)
Результат	плата плавно меняет яркость мощной нагрузки через транзистор, щёлкает реле и вращает мотор в обе стороны
Понадобится	N-MOSFET логического уровня, диод 1N4007, резисторы, модуль реле, драйвер TB6612 или DRV8833, мотор 3 – 6 V, источник питания

16.1 Сначала: вывод платы — это «кнопка», а не «розетка»

Вывод GPIO умеет сообщить «включи», но не способен сам дать много энергии. Он выдаёт считанные миллиамперы: для светодиода хватает, для мотора — нет. Поэтому между платой и мощной нагрузкой ставят *ключ*: электронный (транзистор) или механический (реле). Плата лишь нажимает на «кнопку» ключа, а сила приходит из отдельного источника питания. Земли у них соединяют вместе, чтобы сигнал был понятен обеим сторонам.

	Вывод GPIO	Мотор, лампа, реле
Ток	до $\approx$ 20 mA (безопасный предел для опытов)	от сотен миллиампер до нескольких ампер
Напряжение	только 3,3 V	5 – 12 V и выше
Помехи	сам почти не создаёт	мотор и реле создают броски напряжения

16.2 Ключ на транзисторе

Транзистор работает как кран, который открывают слабым сигналом. Есть два семейства, которые нужны нам.

Биполярный NPN (например, 2N2222, BC337). Управляется *током* базы: базовый ток I_B открывает путь для большого тока коллектора I_C. Резистор в цепи базы обязателен, он задаёт этот ток:

$$R_B = \frac{U_{\text{GPIO}} - U_{BE}}{I_B}, \qquad I_B \approx \frac{I_C}{20}, \qquad U_{BE} \approx 0{,}7\,\text{V}.$$

Делитель на 20 — это запас, чтобы транзистор открылся полностью (насытился) и не грелся. Пример: нагрузка $I_C = 100$ mA. Тогда $I_B \approx 5$ mA и $R_B = (3{,}3 - 0{,}7)/0{,}005 \approx 520\,\Omega$. Берём ближайший стандартный номинал $470\,\Omega$.

N-MOSFET логического уровня (например, IRLZ44N, IRLB8721, AO3400). Управляется *напряжением* на затворе и почти не потребляет ток от платы. Именно он удобнее для ESP32-S3, если нагрузка больше $0{,}5$ А. Пара требований:

- Ключевое слово «логического уровня» (logic level). Обычные MOSFET вроде IRF540 открываются только при 10 V на затворе и от $3{,}3$ V почти не откроются.
- В datasheet посмотрите график сопротивления открытого канала $R_{DS(on)}$ при $U_{GS} = 3{,}3$ V (или ближайшем напряжении): оно должно быть мало, иначе транзистор будет греться.

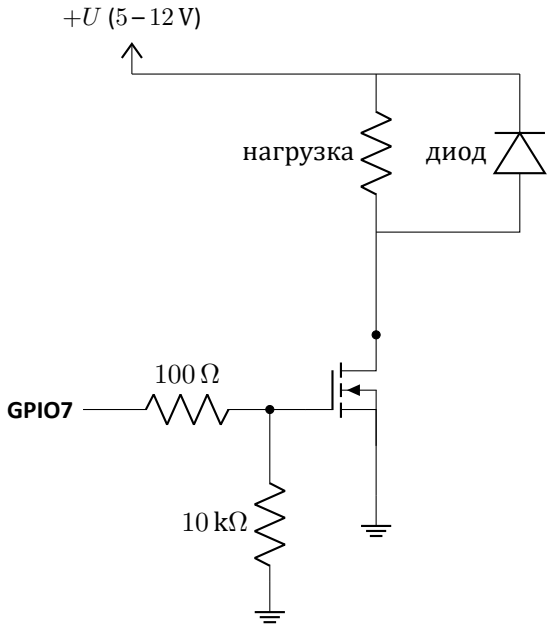


Рис. 16.1: Ключ на N-MOSFET. Резистор $100\,\Omega$ ограничивает бросок тока затвора, $10\,\text{k}\Omega$ держит затвор закрытым при загрузке платы, диод защищает ключ от выброса индуктивной нагрузки

Режим «включено–выключено» можно заменить ШИМом из урока 4: на частоте 20 kHz лампа или мотор получат среднюю мощность, пропорциональную коэффициенту заполнения.

Листинг 16.1: Плавная регулировка мощности через MOSFET

```
const int GATE_PIN = 7;                  // затвор N-MOSFET через резистор 100 Ом

void setup() {
  ledcAttach(GATE_PIN, 20000, 8);        // 20 кГц: выше слышимого диапазона
}

void loop() {
  for (int d = 0; d <= 255; d += 5) {    // плавный разгон нагрузки
    ledcWrite(GATE_PIN, d);
    delay(30);
  }
  for (int d = 255; d >= 0; d -= 5) {    // и плавное затухание
    ledcWrite(GATE_PIN, d);
    delay(30);
  }
}
```

Совет

Готовые «MOSFET-модули» на IRF520 продаются очень часто, но этот транзистор *не* логического уровня. Он будет работать вполсилы и греться. Ищите модули на IRLZ44N, AO3400 или смотрите характеристики конкретного модуля.

16.3 Индуктивная нагрузка и защитный диод

Мотор, реле и электромагнит — это катушки. Если резко отключить ток катушки, она создаёт выброс напряжения противоположной полярности, в десятки раз выше напряжения питания. Он легко убивает транзистор или перезагружает плату. Против этого параллельно катушке ставят **защитный диод** *катодом к плюсу питания*, как на рисунке. Пока ключ включён, диод закрыт и не мешает. После выключения он даёт выбросу замкнутый путь, и энергия безопасно гаснет. Для небольших нагрузок подойдёт 1N4007 или 1N4148 (у 1N4148 ток ниже — для маленьких реле); для быстрых ШИМ-ключей лучше диод Шоттки 1N5819.

16.4 Модуль реле

Реле — электромагнит, который механически замыкает контакты. Оно полностью отделяет цепь нагрузки от платы, поэтому им можно включать то, что не получается включить транзистором: переменный ток, другое напряжение. Недостатки: реле щёлкает, медленное (около 10 ms), изнашивается и не работает с ШИМ.
Готовый модуль уже содержит транзистор, защитный диод, светодиод и часто **оптопару** — прибор, передающий сигнал светом, чтобы схемы не были связаны электрически. У большинства модулей управление **активным LOW**: реле включается, когда на входе IN уровень LOW, и выключается при HIGH. Контакты обозначают COM (общий), NO (нормально разомкнутый) и NC (нормально замкнутый).

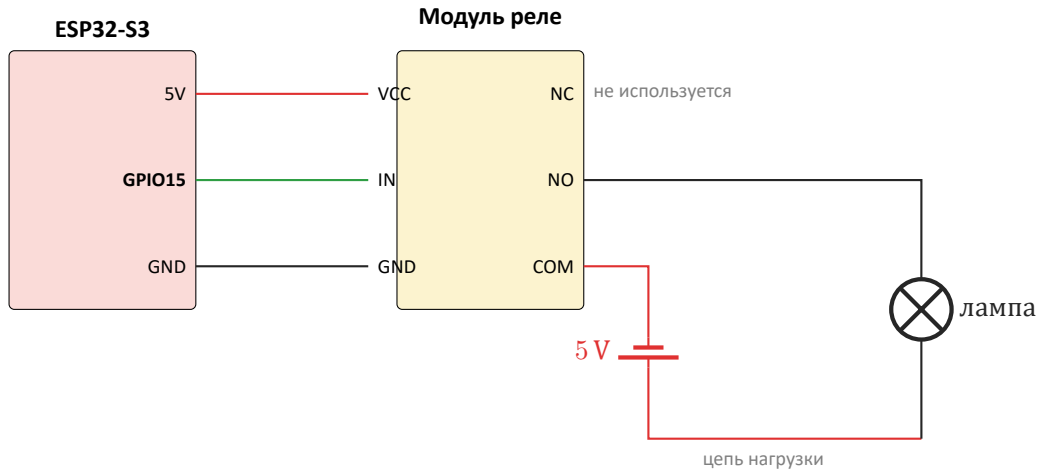


Рис. 16.2: Подключение модуля реле. Слева — управление: питание, общая земля и вход IN на **GPIO15**. Справа — отдельная цепь нагрузки: безопасный источник 5 V и лампа включаются между COM и NO. Эта цепь не связана с платой электрически. К сети 220 V её подключать нельзя

Листинг 16.2: Модуль реле: включение на 2 секунды

```
const int RELAY_PIN = 15;                // вход IN модуля реле
const bool ACTIVE_LOW = true;            // у большинства модулей включает уровень LOW

void relay(bool on) {
  digitalWrite(RELAY_PIN, (on == ACTIVE_LOW) ? LOW : HIGH);
}

void setup() {
  pinMode(RELAY_PIN, OUTPUT);
  relay(false);                          // сразу выключить: для модуля LOW -- это «включено»
}

void loop() {
  relay(true);                           // щелчок: контакты замкнулись
  delay(2000);
  relay(false);
  delay(3000);                           // реле не переключают чаще раза в секунду
}
```

Осторожно

Сетевое напряжение 220 V **смертельно опасно.** В учебных опытах *не подключайте* к контактам реле питание от розетки. Тренируйтесь на безопасных нагрузках: светодиод с резистором, лампа от батарейки, мотор 5 V. Работать с сетевым напряжением можно только после специального обучения, в корпусе с изоляцией и под контролем специалиста.

Если при загрузке платы реле на мгновение щёлкает, это нормально: до выполнения `setup()` состояние вывода не определено. Для ответственных устройств выбирают вывод, который при старте спокойно подтянут к нужному уровню.

16.5 Мотор постоянного тока и драйвер

Коллекторный мотор вращается в одну сторону, если подать ток в одну сторону, и в другую, если поменять полярность. Для смены полярности используют **H-мост**: четыре ключа, которые подают напряжение на мотор то так, то иначе. Мосты уже собраны в драйверах **TB6612FNG** и **DRV8833**. Каждый драйвер управляет мотором тремя сигналами с платы.

AIN1	AIN2	Движение мотора
HIGH	LOW	вперёд (скорость задаёт ШИМ)
LOW	HIGH	назад
LOW	LOW	остановка выбегом
HIGH	HIGH	торможение (у TB6612)

Скорость задаёт сигнал ШИМ на выводе PWMA. Вывод STBY должен быть HIGH, иначе драйвер «спит»: соедините его с 3V3. Ориентировочные характеристики: TB6612FNG — питание мотора до 13,5 V, около 1,2 A на канал; DRV8833 — до 10,8 V, около 1,5 A на канал. Точные цифры смотрите в datasheet своего модуля.

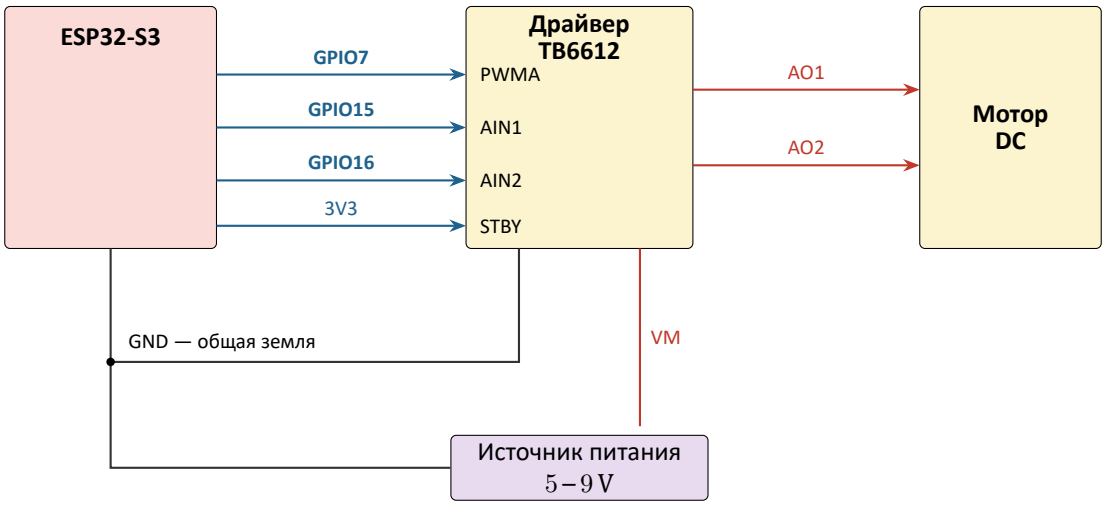


Рис. 16.3: Подключение мотора через драйвер. Управляющие сигналы приходят с платы, мощность мотора — из отдельного источника. Земля у платы, драйвера и источника общая

Листинг 16.3: Мотор вперёд и назад с драйвером TB6612

```
const int IN1_PIN = 15;                  // направление (вход AIN1 драйвера)
const int IN2_PIN = 16;                  // направление (вход AIN2 драйвера)
const int PWM_PIN = 7;                   // скорость (вход PWMA драйвера)

void motor(int speed) {                  // -255..255: знак задаёт направление
  speed = constrain(speed, -255, 255);
  digitalWrite(IN1_PIN, speed > 0);
  digitalWrite(IN2_PIN, speed < 0);
  ledcWrite(PWM_PIN, abs(speed));        // 0 -- мотор останавливается
}

void setup() {
  pinMode(IN1_PIN, OUTPUT);
  pinMode(IN2_PIN, OUTPUT);
  ledcAttach(PWM_PIN, 20000, 8);
  motor(0);
}

void loop() {
  motor(150);  delay(2000);              // вперёд, примерно 60 %
  motor(0);    delay(1000);              // остановка
  motor(-150); delay(2000);              // назад
  motor(0);    delay(1000);
}
```

16.6 Питание: раздельное, но с общей землёй

Мотор при запуске потребляет ток в разы больше, чем в работе, и вызывает просадку напряжения. Если питать мотор и плату от одного порта USB, плата может перезагрузиться (brownout, урок 14). Поэтому мощную часть питают от отдельного источника, а чтобы сигналы платы были понятны драйверу, земли (GND) источников соединяют.

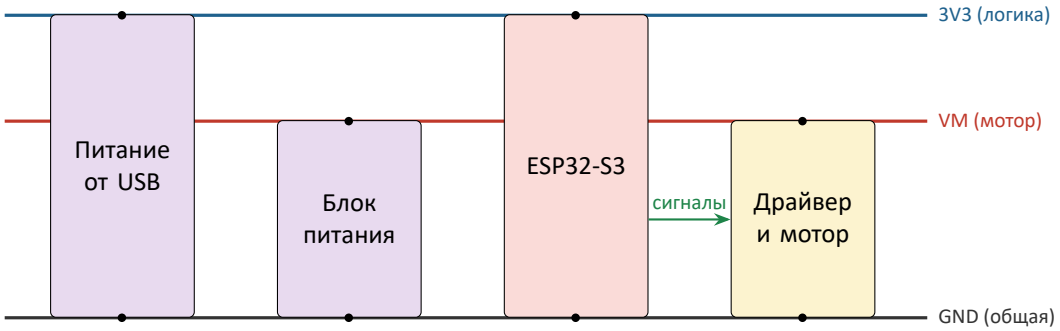


Рис. 16.4: Две линии питания и одна общая земля. Сигналы идут от платы к драйверу, а нагрузка получает энергию только от своего источника

Для сглаживания бросков параллельно питанию драйвера ставят электролитический конденсатор 100 – 470 µF, а на выводы мотора — керамический 100 nF: он гасит помехи от щёток.

16.7 Самостоятельный опыт: вентилятор с регулятором

Соберите отдельный скетч, который управляет мотором с помощью потенциометра на **GPIO1** (урок 5) и кнопки на **GPIO5** (урок 3). Подсказка по логике: потенциометр задаёт скорость от

0 до 255 (`map()`), а каждое нажатие кнопки меняет направление. Не меняйте направление на полном ходу: сначала остановите мотор на 200 ms, иначе пик тока при реверсе нагрузит драйвер и источник.

16.8 То же на ESPUI: пульт для мотора

Мотор с драйвером удобно проверять с телефона, но именно здесь особенно важна осторожность. Управлять нужно так, чтобы потеря связи не оставила мотор включённым. В скетче ниже кнопка Pad «вперёд» едет, пока она нажата, а отпускание останавливает мотор. Если сигнал отпускания не пришёл (связь пропала, телефон погас), то через четыре секунды мотор всё равно остановится. Реверс выполняется с паузой, как рекомендовано в самостоятельном опыте.

Листинг 16.4: ESPUI: Pad «вперёд/назад», ползунок скорости и ограничение времени пуска

```
// Драйвер TB6612: AIN1 -> GPIO15, AIN2 -> GPIO16, PWMA -> GPIO7, STBY -> 3V3 (урок 16).
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

const int IN1_PIN = 15;
const int IN2_PIN = 16;
const int PWM_PIN = 7;
const uint32_t MAX_RUN_MS = 4000;        // предел одного непрерывного пуска: страховка при потере связи

std::atomic<int> direction{0};           // -1 назад, 0 стоп, +1 вперёд
std::atomic<int> speedPercent{40};
std::atomic<uint32_t> runStart{0};
uint16_t stateId;

void motor(int speed) {                  // -255..255, как в листинге урока
  speed = constrain(speed, -255, 255);
  digitalWrite(IN1_PIN, speed > 0);
  digitalWrite(IN2_PIN, speed < 0);
  ledcWrite(PWM_PIN, abs(speed));
}

void speedCallback(Control* sender, int event) {
  if (event == SL_VALUE) speedPercent = constrain(sender->value.toInt(), 0, 100);
}

void padCallback(Control*, int event) {
  switch (event) {
    case P_FOR_DOWN:  direction = 1;  runStart = millis(); break;
    case P_BACK_DOWN: direction = -1; runStart = millis(); break;
    case P_FOR_UP:                    // отпускание кнопки = остановка
    case P_BACK_UP:
    case P_CENTER_DOWN: direction = 0; break;
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(IN1_PIN, OUTPUT);
  pinMode(IN2_PIN, OUTPUT);
  if (!ledcAttach(PWM_PIN, 20000, 8)) { Serial.println("PWM init failed"); return; }
  motor(0);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Motor", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  ESPUI.sliderContinuous = false;
  ESPUI.slider("Скорость, %", speedCallback, ControlColor::Peterriver, 40, 0, 100);
  ESPUI.pad("Вперёд / назад (держите)", padCallback, ControlColor::Carrot);
```

```
  stateId = ESPUI.label("Применённое состояние", ControlColor::Turquoise, "СТОП");
  ESPUI.begin("ESP32-S3 motor");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static int applied = 0;
  int dir = direction;
  if (dir != 0 && millis() - runStart > MAX_RUN_MS) {     // время вышло: остановка
    direction = 0;
    dir = 0;
  }
  int target = dir * speedPercent * 255 / 100;
  if (target != applied) {
    if ((target > 0) != (applied > 0) && applied != 0) {   // реверс: сначала пауза (урок 16)
      motor(0);
      delay(200);
    }
    motor(target);
    applied = target;
    ESPUI.updateLabel(stateId, dir == 0 ? "СТОП" : (dir > 0 ? "ВПЕРЁД" : "НАЗАД"));
  }
  delay(5);
}
```

Схема остаётся прежней: TB6612, питание мотора от отдельного источника, общая земля. Сеть панели называется `S3-Motor`. Для первой проверки отсоедините нагрузку или поднимите колесо над столом.

Осторожно

Пока вы не убедились, что отпускание кнопки останавливает мотор, не подключайте механизм, который может что-то повредить. Программная остановка не заменяет физический выключатель питания.

Если не работает

- **Мотор не вращается, драйвер молчит** — вывод STBY не поднят в HIGH, не подан VM или у платы и драйвера не общая земля.
- **Плата перезагружается при пуске мотора** — просадка питания: используйте отдельный источник, добавьте конденсатор 100 – 470 µF.
- **MOSFET горячий, а нагрузка тусклая** — транзистор не логического уровня или затвор подключён без резистора; проверьте $R_{DS(on)}$ при $3{,}3$ V.

Итоги урока

- Мощную нагрузку включают через ключ (транзистор или реле) с отдельным питанием и общей землёй.
- Индуктивной нагрузке нужен защитный диод; мотору — драйвер с H-мостом, а ШИМ из урока 4 регулирует скорость.
- Реле модуля обычно активное LOW, а сетевое напряжение 220 V в учебных опытах не коммутируют.

Проверь себя

1. Почему нельзя подключить мотор прямо к выводу GPIO?
2. Зачем нужен диод параллельно катушке и как его включают?
3. Почему у платы и источника питания мотора должна быть общая земля?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Нарисуйте схему включения нагрузки через N-канальный MOSFET и подпишите назначение каждого элемента.
2. ★ Рассчитайте ток через светодиодную ленту $5\,\text{V}$ из 30 светодиодов, если один потребляет $20\,\text{mA}$.
3. ★★ Включайте лампу $5\,\text{V}$ через MOSFET кнопкой на **GPIO5**.
4. ★★ Рассчитайте резистор базы для NPN-транзистора и нагрузки $200\,\text{mA}$ при $h_{21E} = 100$ и соберите ключ.
5. ★★★ Добавьте плавный разгон нагрузки ШИМ и защитный диод при индуктивной нагрузке.
6. ★★★★ Сделайте автоматическую вентиляцию: мотор включается, когда DS18B20 показывает температуру выше порога, а скорость растёт вместе с ней.

Урок 17. Адресные светодиоды WS2812

Цель	управлять лентой адресных светодиодов WS2812 по одному проводу и безопасно рассчитывать её питание
Опирается на	урок 1 (неблокирующая работа по `millis()`), урок 4 (гамма-коррекция), урок 16 (отдельное питание)
Результат	на ленте из 8 пикселей идут радуга и «бегущий огонь», а плата остаётся свободной
Понадобится	лента, кольцо или модуль WS2812B на 8 пикселей; резистор $330\,\Omega$; конденсатор $1000\,\mu\text{F}$; для длинных лент — блок питания $5\,\text{V}$

17.1 Сначала: цепочка светодиодов, которые слушают по очереди

Обычная RGB-лента требует по проводу на каждый цвет, и все её светодиоды горят одинаково. В ленте WS2812 всё иначе: в каждом пикселе есть крошечная микросхема. Вы шлёте по одному проводу длинную очередь цветов. Первый пиксель забирает первые три числа (красный, зелёный, синий), остальное передаёт дальше, второй забирает свои три — и так по цепочке. В итоге каждый пиксель горит своим цветом, а плате нужен всего один вывод.

17.2 Протокол: нули и единицы разной длины

Данные передаются одним проводом без отдельного такта. Каждый бит — это импульс HIGH, за которым следует LOW. Бит определяет *длительность HIGH*: короткая — «0», длинная — «1». Весь бит занимает около $1{,}25\,\mu\text{s}$, то есть скорость примерно $800\,\text{kHz}$.

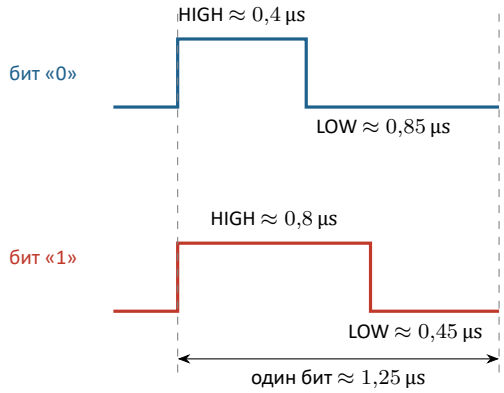


Рис. 17.1: Кодирование битов в WS2812. Значения ориентировочные: допуск около $\pm 0{,}15$ µs, точные цифры смотрите в datasheet своей микросхемы

На каждый пиксель приходится 24 бита: по 8 бит на зелёный, красный и синий (именно в этом порядке у большинства лент). После всех данных нужна пауза LOW не короче ≈ 50 µs — сигнал сброса. Он говорит ленте «кадр закончился, показывайте»: поэтому `strip.show()` вызывают после изменений.

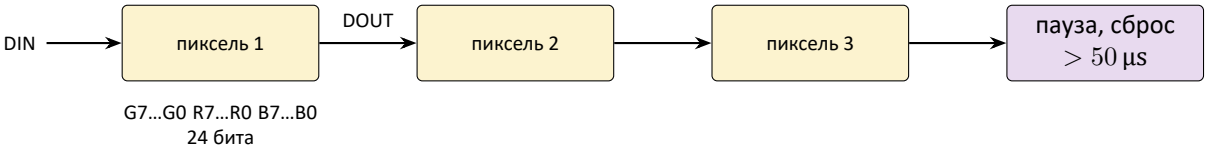


Рис. 17.2: Кадр данных. Каждый пиксель отрезает первые 24 бита, а остальное передаёт дальше через DOUT

17.3 Почему не `digitalWrite()`

Импульсы длиной меньше микросекунды нужно формировать с точностью в десятки наносекунд. Вызов `digitalWrite()` выполняется долго и неравномерно, а прерывания (Wi-Fi, USB, таймеры) могут вклиниться посреди передачи, и цвета «поплывут». На ESP32-S3 для этого есть периферия **RMT** (Remote Control): процессор записывает в её память список длительностей, а аппаратура выдаёт импульсы сама. Библиотека **Adafruit NeoPixel** использует RMT, поэтому цвета остаются точными даже при работе Wi-Fi.

17.4 Питание: считаем ток

Каждый пиксель — это три светодиода. Один пиксель на полной яркости белым цветом потребляет до ≈ 60 mA (3 цвета по 20 mA). Лента из 30 пикселей — уже до $1{,}8$ A. Верхняя оценка тока:

$$I \approx N \cdot 60\,\text{mA} \cdot \frac{\text{BRIGHTNESS}}{255},$$

где N — число пикселей, а BRIGHTNESS — значение в `setBrightness()`.
Для 8 пикселей на яркости 40 из 255 получаем около $8 \cdot 60 \cdot 0{,}16 \approx 75$ mA, и порт USB это выдержит. Для 60 пикселей на полной яркости нужен блок питания 5 V на 3 A.

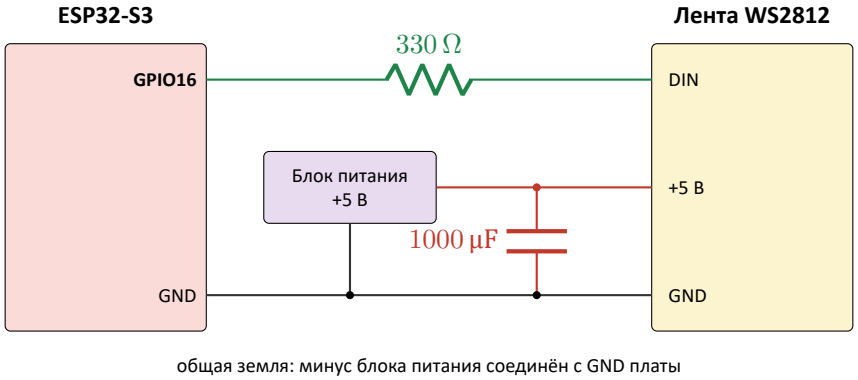


Рис. 17.3: Подключение ленты. Резистор в линии данных гасит выбросы, конденсатор на питании смягчает бросок тока при включении, а земля платы и блока питания общая

Осторожно

Сначала подключайте GND, затем питание, а сигнал — в последнюю очередь. Подача данных на ленту без общей земли может вывести из строя первый пиксель. Конденсатор должен быть рассчитан хотя бы на $6{,}3$ V, его **полярность** (минус на полоске корпуса) нельзя путать.

Резистор 330 – $470\,\Omega$ ставьте как можно ближе к первому пикселю. Лента питается от 5 V, и по datasheet ей нужен вход выше $0{,}7 \cdot 5\,\text{V} = 3{,}5$ V, а плата даёт $3{,}3$ V. Часто это работает, но на длинных проводах может сбоить. Надёжный вариант — микросхема-преобразователь уровня (например, 74AHCT125) в линии данных; она превращает $3{,}3$ V в 5 V.

17.5 Цвет через HSV

Смешивать красный, зелёный и синий для радуги неудобно. Модель **HSV** описывает цвет понятнее: *Hue* (оттенок) — положение на цветовом круге от 0° до 360°, *Saturation* (насыщенность) — насколько цвет яркий, а не бледный, *Value* (яркость). Чтобы получить радугу, достаточно менять один Hue.

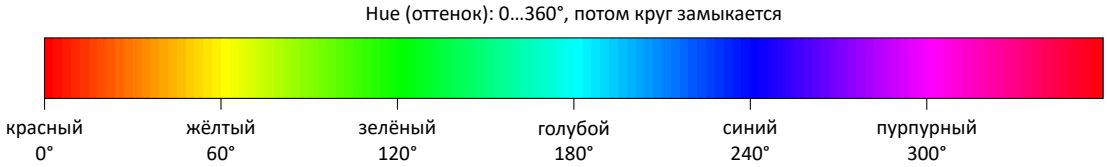


Рис. 17.4: Шкала оттенков HSV. Ползунок Hue проходит все цвета радуги

17.6 Встроенный светодиод как лента из одного пикселя

На многих платах ESP32-S3-DevKitC-1 есть светодиод WS2812: на ревизии v1.0 он подключён к **GPIO48**, а на v1.1 — к **GPIO38** (мы использовали его в уроке 1). Это та же «лента», только из одного пикселя. В ядре Arduino-ESP32 для неё есть готовая функция `rgbLedWrite(pin, r, g, b)`. Пример ниже без всякой ленты показывает радугу из HSV.

Листинг 17.1: Радуга на встроенном светодиоде: HSV вручную

```
const int RGB_PIN = 48;                  // на ревизии v1.1 -- 38

// Перевод цвета из HSV в RGB: hue 0..359, sat и val 0..255.
void hsvToRgb(int hue, uint8_t sat, uint8_t val, uint8_t &r, uint8_t &g, uint8_t &b) {
  int region = hue / 60;                 // шесть секторов цветового круга
  int rem = (hue % 60) * 255 / 60;       // положение внутри сектора
  uint8_t p = val * (255 - sat) / 255;
  uint8_t q = val * (255 - sat * rem / 255) / 255;
  uint8_t t = val * (255 - sat * (255 - rem) / 255) / 255;
  switch (region) {
    case 0:  r = val; g = t;   b = p;   break;
    case 1:  r = q;   g = val; b = p;   break;
    case 2:  r = p;   g = val; b = t;   break;
    case 3:  r = p;   g = q;   b = val; break;
    case 4:  r = t;   g = p;   b = val; break;
    default: r = val; g = p;   b = q;   break;
  }
}

void setup() {}

void loop() {
  static int hue = 0;
  uint8_t r, g, b;
  hsvToRgb(hue, 255, 40, r, g, b);       // яркость 40 из 255: не слепит глаза
  rgbLedWrite(RGB_PIN, r, g, b);
  hue = (hue + 1) % 360;                 // за один оборот проходят все цвета радуги
  delay(15);
}
```

17.7 Эффекты без delay()

В примере с лентой каждый эффект — это функция одного шага: она сдвигает картинку и вызывает `show()`. Шаги запускаются по таймеру `millis()`, поэтому `loop()` не блокируется и плате остаётся время на кнопки, датчики и сеть (урок 7). Радуга использует Hue пикселя $=$ сдвиг $+ i \cdot 65536/N$; бегущий огонь гасит ленту и зажигает один пиксель, сдвигая его на шаг.

Листинг 17.2: Радуга и бегущий огонь на ленте из 8 пикселей

```
#include <Adafruit_NeoPixel.h>

const int DATA_PIN = 16;                 // через резистор 330 Ом
const int NUM_PIXELS = 8;
const uint8_t BRIGHTNESS = 40;           // 0..255; ограничивает ток всей ленты

Adafruit_NeoPixel strip(NUM_PIXELS, DATA_PIN, NEO_GRB + NEO_KHZ800);

uint32_t lastStep = 0;
uint32_t lastSwitch = 0;
uint16_t hueOffset = 0;                  // сдвиг радуги, 0..65535
int head = 0;                            // положение «бегущего огня»
bool rainbowMode = true;

void rainbowStep() {
  for (int i = 0; i < NUM_PIXELS; i++) {
```

```
    uint16_t hue = hueOffset + i * 65536UL / NUM_PIXELS;
    strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(hue)));
  }
  hueOffset += 256;
}

void chaseStep() {
  strip.clear();
  strip.setPixelColor(head, strip.Color(255, 60, 0));  // оранжевая «искра»
  head = (head + 1) % NUM_PIXELS;
}

void setup() {
  strip.begin();
  strip.setBrightness(BRIGHTNESS);
  strip.show();                            // сразу погасить все пиксели
}

void loop() {
  uint32_t now = millis();
  if (now - lastSwitch >= 8000) {          // каждые 8 с меняем эффект
    lastSwitch = now;
    rainbowMode = !rainbowMode;
  }
  if (now - lastStep >= 40) {              // шаг анимации без delay()
    lastStep = now;
    if (rainbowMode) rainbowStep(); else chaseStep();
    strip.show();
  }
}
```

Установите библиотеку **Adafruit NeoPixel**. Функция `gamma32()` применяет ту же гамма-коррекцию, что и урок 4: без неё середина радуги выглядит блёклой.

17.8 Самостоятельный опыт: шкала уровня

Сделайте отдельный скетч, в котором потенциометр на **GPIO1** задаёт, сколько пикселей горит: от 0 до 8. Первые пиксели зелёные, средние жёлтые, последний красный. Подсказки: `map(analogRead(...), 0, 4095, 0, NUM_PIXELS + 1)` и перебор пикселей циклом `for`. Обновляйте ленту только когда число пикселей изменилось, а не на каждом проходе `loop()`.

17.9 То же на ESPUI: выбираем цвет ползунком

Радугу из HSV мы раньше только смотрели, а теперь можно выбирать цвет самому. Два ползунка в панели — «Hue» от 0° до 359° и «Яркость» — превращаются в тот же перевод HSV → RGB, который уже есть в этом уроке. Никакой ленты не нужно: используем встроенный светодиод.

Листинг 17.3: ESPUI: оттенок и яркость встроенного RGB

```
// Встроенный RGB-светодиод (GPIO48, на ревизии v1.1 -- GPIO38): оттенок и яркость с телефона.
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

const int RGB_PIN = 48;
std::atomic<int> hueValue{0};            // 0..359
std::atomic<int> brightness{40};         // 0..255, для глаз берём небольшую
uint16_t colorId;
```

```

void hsvToRgb(int hue, uint8_t sat, uint8_t val, uint8_t &r, uint8_t &g, uint8_t &b) {
  int region = hue / 60;                  // тот же перевод HSV -> RGB, что в уроке
  int rem = (hue % 60) * 255 / 60;
  uint8_t p = val * (255 - sat) / 255;
  uint8_t q = val * (255 - sat * rem / 255) / 255;
  uint8_t t = val * (255 - sat * (255 - rem) / 255) / 255;
  switch (region) {
    case 0:  r = val; g = t;   b = p;   break;
    case 1:  r = q;   g = val; b = p;   break;
    case 2:  r = p;   g = val; b = t;   break;
    case 3:  r = p;   g = q;   b = val; break;
    case 4:  r = t;   g = p;   b = val; break;
    default: r = val; g = p;   b = q;   break;
  }
}

void hueCallback(Control* sender, int event) {
  if (event == SL_VALUE) hueValue = constrain(sender->value.toInt(), 0, 359);
}
void brightCallback(Control* sender, int event) {
  if (event == SL_VALUE) brightness = constrain(sender->value.toInt(), 0, 255);
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Hue", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  ESPUI.sliderContinuous = true;          // цвет должен меняться, пока двигаем ползунок
  ESPUI.slider("Hue, °", hueCallback, ControlColor::Carrot, 0, 0, 359);
  ESPUI.slider("Яркость (0..255)", brightCallback, ControlColor::Wetasphalt, 40, 0, 255);
  colorId = ESPUI.label("Цвет RGB", ControlColor::Turquoise, "0, 0, 0");
  ESPUI.begin("ESP32-S3 hue");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  uint8_t r, g, b;
  hsvToRgb(hueValue, 255, brightness, r, g, b);
  rgbLedWrite(RGB_PIN, r, g, b);

  static uint32_t last = 0;
  if (millis() - last >= 300) {           // подпись обновляем редко, цвет -- каждый проход
    last = millis();
    ESPUI.updateLabel(colorId, String(r) + ", " + String(g) + ", " + String(b));
  }
  delay(10);
}
```

Подключитесь к сети `S3-Hue` (пароль `lesson-39-demo`). Здесь режим `sliderContinuous` включён намеренно: цвет должен меняться, пока вы двигаете палец. Если плата и браузер не успевают, верните `false`: событие придёт после отпускания. На ревизии v1.1 замените `RGB_PIN` на 38.

Если не работает

- **Лента не горит или горит случайными цветами** — сигнал подан на DOUT вместо DIN, нет общей земли или не тот вывод в программе.
- **Цвета не те (вместо красного зелёный)** — у ленты другой порядок цветов: замените `NEO_GRB` на `NEO_RGB` в конструкторе.
- **Плата перезагружается при яркости выше нескольких пикселей** — просадка питания:

снизьте `setBrightness()` или питайте ленту от блока 5 V.

Итоги урока

- WS2812 принимает по одному проводу 24 бита на пиксель; бит «0» и «1» отличаются длиной HIGH около 0,4 µs и 0,8 µs.
- Точные импульсы формирует RMT, поэтому используйте библиотеку, а не `digitalWrite()`; ток считайте как $N \cdot 60$ mA на полной яркости.
- Для эффектов используйте HSV и шаги по `millis()`, а питание ленты и общую землю соединяйте по правилам.

Проверь себя

1. Чем отличаются в протоколе WS2812 нулевой и единичный биты?
2. Какой ток потребляет лента из 30 пикселей на полной яркости белого и что с этим делать?
3. Зачем нужны резистор в линии данных и конденсатор на питании?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Зажгите первые три пикселя красным, зелёным и синим цветом.
2. ★ Измените яркость всей ленты и посчитайте потребляемый ток для вашего числа светодиодов.
3. ★★ Сделайте «бегущий огонёк», который перемещается по ленте без `delay()`.
4. ★★ Измените скорость радуги потенциометром и направление движения кнопкой.
5. ★★★ Покажите на ленте температуру DS18B20: синий — холодно, зелёный — комфортно, красный — жарко.
6. ★★★★ Сделайте эффект «дыхание» с гамма-коррекцией, не используя `delay()`.

Урок 18. Плата как клиент: HTTP, JSON, время и файлы

Цель	получать данные из интернета, разбирать ответ в формате JSON, узнавать точное время и хранить файлы настроек во Flash
Опирается на	урок 8 (Wi-Fi), урок 11 (NVS и Flash), урок 13 (время для TLS)
Результат	плата запрашивает данные у сервера, показывает московское время и сохраняет файл настроек
Понадобится	доступ к Wi-Fi с выходом в интернет; библиотека ArduinoJson версии 7

18.1 Сначала: плата ходит в интернет, как браузер

В уроке 8 плата была сервером: к ней приходили запросы. Теперь поменяем роли — плата сама отправляет запрос и ждёт ответ. Для человека сервер вернул бы красивую страницу, а для программы — короткий текст с данными в формате JSON: «город, температура, влажность». Дальше плата вытаскивает из текста нужные числа. А чтобы знать, какой сейчас час, она спрашивает точное время у специальных серверов в сети.

18.2 Путь запроса

Перед запросом нужно превратить имя сайта в IP-адрес. Это делает DNS-сервер, адрес которого плата получает от роутера. Затем плата устанавливает соединение с сервером и отправляет HTTP-запрос `GET`. Сервер отвечает кодом состояния (200 означает «всё хорошо») и телом ответа.

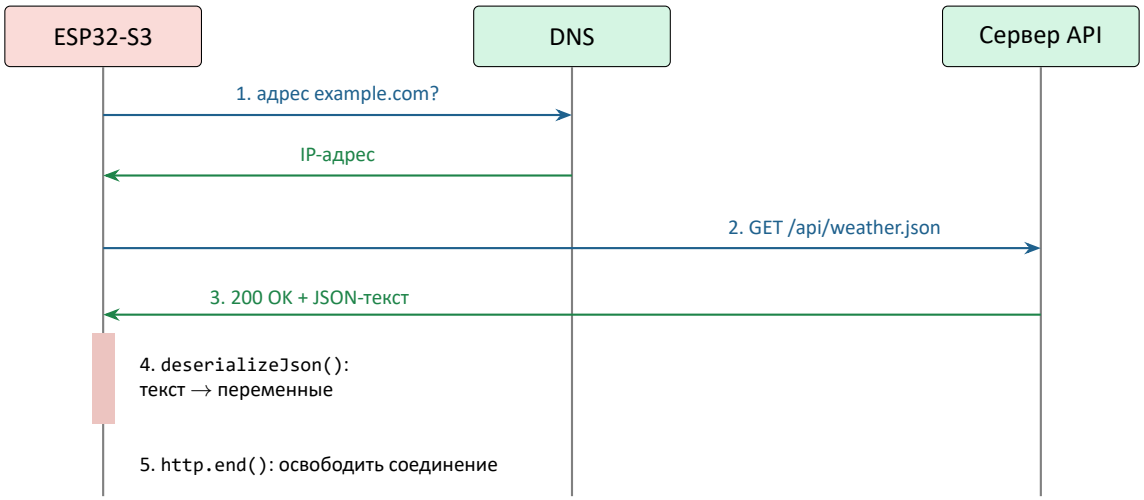


Рис. 18.1: Путь запроса: имя превращается в IP-адрес, затем идёт запрос, ответ и разбор

18.3 HTTP-запрос: GET

Клиент `HTTPClient` входит в ядро Arduino-ESP32, ставить библиотеку не нужно. Внизу пример выводит в Serial текст ответа. Адрес в примере *условный*: сервиса по нему может не быть. Замените его на адрес своего сервиса, у которого есть открытый API; многие метеосервисы выдают JSON бесплатно, но часто требуют регистрации и ключ.

Листинг 18.1: GET-запрос и вывод ответа

```
#include <WiFi.h>
#include <HTTPClient.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";
// Замените адрес на свой: это условный пример, такого сервиса может не быть.
const char* URL = "http://example.com/api/weather.json";

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
  if (WiFi.status() != WL_CONNECTED) { Serial.println("Нет Wi-Fi"); return; }

  HTTPClient http;
  http.setTimeout(8000);                  // не ждать ответ дольше 8 с
  http.begin(URL);
  int code = http.GET();                  // 200 -- успех, отрицательное число -- ошибка сети
  if (code == 200) {
    String body = http.getString();
    Serial.println(body);
  } else {
    Serial.printf("Ошибка запроса: %d (%s)\n", code, HTTPClient::errorToString(code).c_str());
  }
  http.end();                             // освободить соединение
}

void loop() {}
```

Положительный код — ответ сервера (200 успех, 404 страницы нет, 500 сбой на сервере). Отрицательный код — ошибка на самой плате: нет соединения, истёк тайм-аут. Функция `HTTPClient::errorToString()` превратит код в понятное сообщение.

Осторожно

Адрес вида `http://` передаёт всё открытым текстом. Не отправляйте так пароли и ключи доступа. Для защищённого канала нужен HTTPS с проверкой сертификата (урок 13). Кроме того, не отправляйте запросы чаще, чем разрешено сервису: обычно хватает раза в минуту или реже, иначе ключ заблокируют.

18.4 JSON и ArduinoJson

JSON — текстовый формат для данных: пары «имя: значение» в фигурных скобках. Он понятен и людям, и программам.

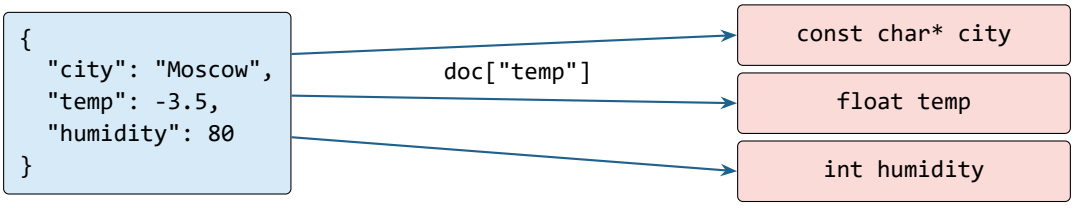


Рис. 18.2: Разбор JSON: текст ответа превращается в обычные переменные программы

Библиотека **ArduinoJson** (версия 7, ставится через Library Manager) умеет разобрать такой текст. Создайте объект `JsonDocument`, вызовите `deserializeJson()` и *обязательно* проверьте результат: сервер может вернуть страницу с ошибкой вместо JSON. Оператор `|` подставляет значение по умолчанию, если поля в ответе нет.

Листинг 18.2: Разбор JSON-ответа

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";
// Замените адрес на свой. Ожидаемый ответ: {"city":"Moscow","temp":-3.5,"humidity":80}
const char* URL = "http://example.com/api/weather.json";

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
  if (WiFi.status() != WL_CONNECTED) { Serial.println("Нет Wi-Fi"); return; }

  HTTPClient http;
  http.setTimeout(8000);
  http.begin(URL);
  if (http.GET() == 200) {
    JsonDocument doc;                                   // ArduinoJson 7
    DeserializationError err = deserializeJson(doc, http.getString());
    if (err) {                                          // ответ не похож на JSON
      Serial.printf("Ошибка разбора: %s\n", err.c_str());
    } else {
      const char* city = doc["city"] | "?";             // "?" -- значение по умолчанию
      float temp = doc["temp"] | 0.0f;
      int humidity = doc["humidity"] | -1;
      Serial.printf("%s: %.1f °C, влажность %d %%\n", city, temp, humidity);
    }
  } else {
    Serial.println("Сервер не ответил кодом 200");
  }
  http.end();
}

void loop() {}
```

Совет

Если разбор не удаётся, сначала выведите в Serial сырой ответ `http.getString()`. Чаще всего вместо JSON приходит HTML-страница ошибки или перенаправление.

18.5 Точное время по NTP

В плате нет часов с батарейкой: после включения она «не знает», какое сейчас число. Узнать его можно у серверов времени по протоколу **NTP**. Функция `configTzTime()` задаёт часовой пояс и серверы, после чего время синхронизируется само в фоне. Часовой пояс записывается в формате POSIX, где знак *обратный* привычному: Москва (UTC+3, без перехода на летнее время) записывается как `"MSK-3"`.

Функция `getLocalTime(&t, timeout)` ждёт синхронизации не дольше заданного времени и возвращает `false`, если время ещё не получено. Это нужно и для HTTPS (урок 13): без правильной даты проверка срока сертификата невозможна.

Листинг 18.3: Московское время в Serial Monitor

```
#include <WiFi.h>
#include <time.h>

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";

void setup() {
  Serial.begin(115200);
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
  if (WiFi.status() != WL_CONNECTED) { Serial.println("Нет Wi-Fi"); return; }

  // Москва: UTC+3 без перехода на летнее время. В записи POSIX знак обратный: "MSK-3".
  configTzTime("MSK-3", "pool.ntp.org", "time.nist.gov");
}

void loop() {
  struct tm t;
  if (getLocalTime(&t, 5000)) {           // ждать синхронизации не дольше 5 с
    Serial.println(&t, "%d.%m.%Y %H:%M:%S");
  } else {
    Serial.println("Время ещё не получено");
  }
  delay(1000);
}
```

18.6 LittleFS: файлы во Flash

В уроке 11 мы сохраняли настройки в NVS, в уроке 10 — журнал на SD-карту. **LittleFS** занимает промежуточное место: это настоящая файловая система внутри Flash платы. Файлы лежат в своём разделе памяти, их можно открывать, читать и дописывать, как на карте, и никакие дополнительные детали не нужны.

	NVS (урок 11)	LittleFS	SD-карта (урок 10)
Что хранит	пары «ключ–значение»	файлы и папки	файлы и папки
Объём	десятки килобайт	около 1 – 1,5 MB (зависит от схемы разделов)	гигабайты
Нужны ли детали	нет	нет	модуль и карта
Хорош для	небольшие настройки, счётчики	страницы, конфиги, небольшие журналы	большие журналы, фото

Листинг 18.4: Чтение и запись файла настроек

```
#include <LittleFS.h>

const char* PATH = "/settings.txt";

void setup() {
  Serial.begin(115200);
  if (!LittleFS.begin(true)) {            // true: отформатировать раздел при первом запуске
    Serial.println("LittleFS не запустилась");
    return;
  }

  File f = LittleFS.open(PATH, "r");      // читаем, если файл уже есть
  if (f) {
    int level = f.readStringUntil('\n').toInt();
    f.close();
    Serial.printf("Прочитано: уровень = %d\n", level);
  } else {
    Serial.println("Файла настроек ещё нет");
  }

  File w = LittleFS.open(PATH, "w");      // "w" стирает старое содержимое
  if (w) {
    w.println(75);                        // новое значение настройки
    w.close();                            // закрыть, чтобы данные записались
    Serial.println("Настройка записана");
  }
}

void loop() {}
```

Параметр `true` в `LittleFS.begin()` форматирует раздел при первом запуске. Так же, как и на SD, файл нужно закрывать после записи. Помните про износ Flash: не пишите в файл на каждом проходе `loop()`, как и с NVS.

18.7 Самостоятельный опыт: сетевые часы

Соберите отдельный скетч: плата подключается к Wi-Fi, получает московское время и печатает его раз в секунду. Затем добавьте в LittleFS файл `/boot.txt`, куда при каждом запуске дописывается строка с датой и временем старта. Перезагрузите плату несколько раз и прочитайте файл целиком: вы увидите историю запусков.

Если не работает

- **Код ответа отрицательный** — нет Wi-Fi или интернета, неверный адрес, либо сайт требует HTTPS, а в адресе стоит `http://`.
- **`deserializeJson()` возвращает ошибку** — сервер прислал не JSON (страница ошибки, пустой ответ); выведите сырой текст и проверьте адрес и ключ доступа.
- **Время не приходит или показывает 1970 год; `LittleFS.begin()` не работает** — закрыт доступ к NTP или нет интернета; для LittleFS выберите схему разделов с файловой системой.

Итоги урока

- `HTTPClient` отправляет GET-запрос; положительный код — ответ сервера, отрицательный — ошибка сети.
- JSON разбирают через `deserializeJson()`, обязательно проверяя ошибку и подставляя значения по умолчанию.
- Время приходит по NTP через `configTzTime()` (для Москвы `"MSK-3"`), а файлы настроек удобно хранить в LittleFS.

Проверь себя

1. Что происходит между вызовом `http.GET()` и получением ответа?
2. Почему результат `deserializeJson()` нужно проверять?
3. Чем LittleFS отличается от NVS и от SD-карты?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выполните GET-запрос и выведите код ответа и размер данных.
2. ★ Разберите готовый JSON и выведите одно из его полей.
3. ★★ Выведите московское время на OLED в формате ЧЧ:ММ:СС с обновлением раз в секунду.
4. ★★ Показывайте температуру из интернета на OLED вместе со временем получения.
5. ★★★ Раз в десять минут запрашивайте данные и сохраняйте последнее значение в LittleFS, чтобы показывать его без сети.
6. ★★★★ Сделайте журнал с отметками времени в LittleFS, который не растёт бесконечно: при превышении размера старые записи удаляются.

Урок 19. Нативный USB: плата как клавиатура и мышь

Цель	превратить плату в USB-устройство ввода: клавиатуру, мышь и регулятор громкости
Опирается на	урок 2 (последовательный порт), урок 3 (кнопка и дребезг), урок 5 (потенциометр)
Результат	нажатие кнопки печатает текст на компьютере, а ручка потенциометра меняет громкость
Понадобится	плата с двумя разъёмами USB (на DevKitC-1 это «UART» и «USB»), кнопка на **GPIO5**, потенциометр на **GPIO1**

19.1 Сначала: компьютер не знает, что вы ему подключили

Когда вы вставляете в компьютер клавиатуру, она представляется: «я клавиатура». После этого компьютер принимает от неё нажатия, не разбираясь, какой микросхемой они формируются. ESP32-S3 умеет представляться по USB сама — как клавиатура, мышь или регулятор громкости.

Это называется HID (Human Interface Device, «устройство взаимодействия с человеком»). Нужно только правильно подключить плату и выбрать в Arduino IDE особый режим работы USB.

19.2 Два разъёма — два разных пути

Плата ESP32-S3-DevKitC-1 имеет два разъёма Type-C. Они ведут к разным частям платы.

- **«UART»** подключён к микросхеме-мосту USB–UART. Она переводит USB в обычный последовательный порт (UART0, выводы **GPIO43**/**GPIO44**). Именно этим путём мы прошивали плату и печатали в Serial Monitor в уроках 0–2.
- **«USB»** идёт напрямую к самому ESP32-S3 (выводы **GPIO19** — D− и **GPIO20** — D+). Чип сам реализует USB, поэтому может быть чем угодно: виртуальным портом, клавиатурой, накопителем.

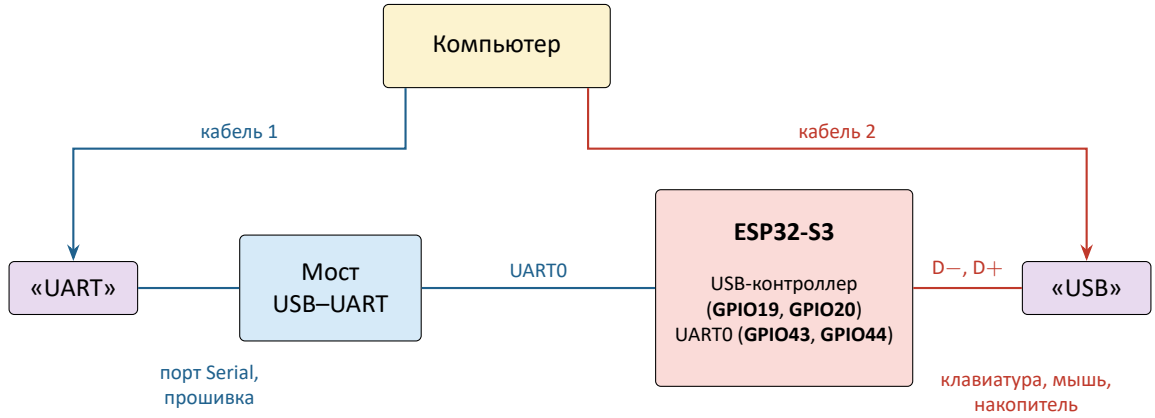


Рис. 19.1: Два пути USB. Левый проходит через мост и даёт обычный последовательный порт, правый ведёт напрямую к чипу и позволяет ему быть HID-устройством

19.3 Настройка Arduino IDE

По умолчанию нативный USB работает в режиме «Hardware CDC and JTAG»: чип выглядит как порт и больше ничем. Чтобы он стал клавиатурой, режим нужно изменить:

1. Tools → USB Mode → USB-OTG (TinyUSB) — включает полноценный USB-стек.
2. Tools → USB CDC On Boot оставьте Disabled, если хотите смотреть Serial через порт «UART».
3. Прошивайте плату через порт «UART», а HID-устройство появится на порту «USB». Проверьте это, подключив оба кабеля.

Совет

В режиме TinyUSB `Serial.println()` выводится в порт «UART» (UART0), а не в «USB». Если Serial Monitor молчит, проверьте, что выбран COM-порт от моста.

19.4 Клавиатура по кнопке

Библиотеки `USB.h` и `USBHIDKeyboard.h` входят в ядро Arduino-ESP32. Порядок работы такой: создать объект `USBHIDKeyboard`, вызвать `Keyboard.begin()` и затем `USB.begin()`, которая запускает USB. Дальше работают привычные `print()` и `write()`: они «нажимают» нужные клавиши.

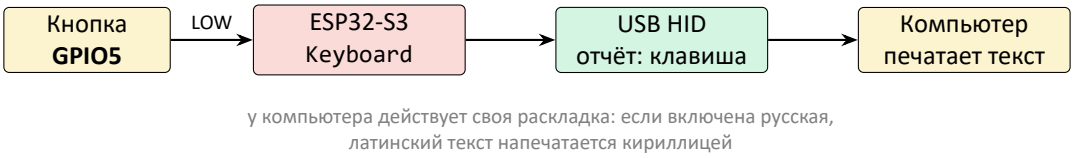


Рис. 19.2: Цепочка передачи: нажатие кнопки превращается в HID-отчёт, который компьютер считает нажатием клавиши

Листинг 19.1: Кнопка печатает текст

```
#include "USB.h"
#include "USBHIDKeyboard.h"

const int BUTTON_PIN = 5;                 // кнопка между GPIO5 и GND
USBHIDKeyboard Keyboard;

bool wasPressed = false;
uint32_t lastChange = 0;

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  Keyboard.begin();
  USB.begin();
}

void loop() {
  bool pressed = (digitalRead(BUTTON_PIN) == LOW);
  if (pressed != wasPressed && millis() - lastChange > 30) {   // защита от дребезга
    lastChange = millis();
    wasPressed = pressed;
    if (pressed) {                        // печатаем один раз на нажатие
      Keyboard.print("Hello from ESP32-S3");  // только латиница и цифры
      Keyboard.write(KEY_RETURN);         // клавиша Enter
    }
  }
}
```

Печатайте только латиницу и цифры: компьютер сам переводит коды клавиш в буквы по своей раскладке. Для сочетаний используйте `Keyboard.press(KEY_LEFT_CTRL); Keyboard.press('c'); Keyboard.releaseAll();` — так выглядит Ctrl+C. Клавиши-стрелки и Enter называются `KEY_RIGHT_ARROW`, `KEY_RETURN` и так далее.

Осторожно

Клавиатура, которая «печатает без остановки», может превратить компьютер в хаос: удалять файлы, закрывать окна, переполнять поля ввода. Поэтому:

- всегда оставляйте в коде условие по кнопке или паузу, а не бесконечную печать в `loop()`;
- знайте, как войти в загрузчик: нажмите и держите BOOT, коротко нажмите RST, затем отпустите BOOT. Плата перестанет выполнять прошивку и будет готова к перепрошивке через «UART»;
- проверяйте опыты только на своём компьютере и никогда не имитируйте ввод без ведома владельца.

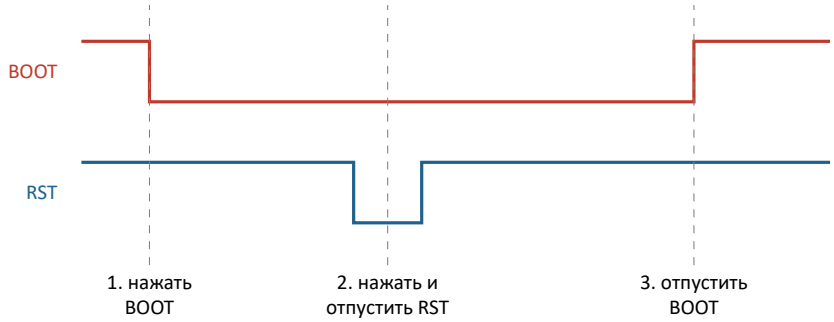


Рис. 19.3: Вход в загрузчик: BOOT удерживается, пока вы нажимаете и отпускаете RST. После этого плата ждёт прошивку, а выполнение вашего кода остановлено

19.5 Громкость потенциометром

Кроме клавиатуры, есть класс «потребительское управление» (Consumer Control): мультимедийные клавиши, громкость, пауза. Класс `USBHIDConsumerControl` умеет нажимать такие клавиши. Ручку потенциометра (урок 5) мы превращаем в нажатия «громче» и «тише»: при повороте вправо отправляем столько нажатий `CONSUMER_CONTROL_VOLUME_INCREMENT`, на сколько делений сдвинулась ручка. Абсолютного уровня HID не передаёт, только шаги.
Потенциометр и кнопка подключены так же, как в уроках 3 и 5: движок на **GPIO1**, кнопка между **GPIO5** и GND (подтяжка внутренняя).

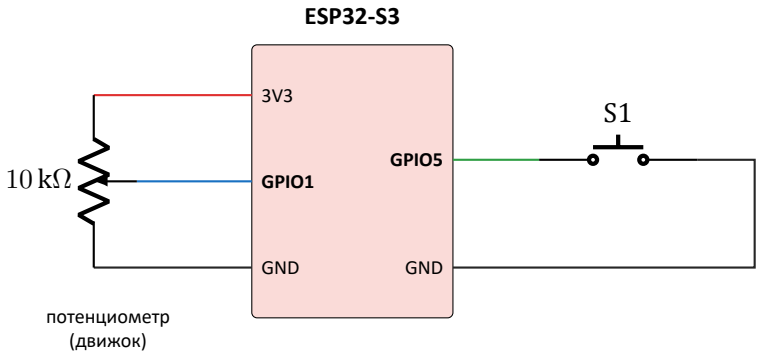


Рис. 19.4: Подключение потенциометра (**GPIO1**) и кнопки (**GPIO5**) для регулятора громкости

Листинг 19.2: Потенциометр как регулятор громкости

```
#include "USB.h"
#include "USBHIDConsumerControl.h"

const int POT_PIN = 1;                  // потенциометр на GPIO1
const int BUTTON_PIN = 5;               // кнопка: включить/выключить звук
const int STEP = 128;                   // 4096 / 128 = 32 положения ручки

USBHIDConsumerControl Consumer;
int lastPos = 0;
bool wasPressed = false;

void tap(uint16_t key) {                // нажать и сразу отпустить мультимедийную клавишу
  Consumer.press(key);
  Consumer.release();
}

void setup() {
```

```
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  analogReadResolution(12);
  Consumer.begin();
  USB.begin();
  lastPos = analogRead(POT_PIN) / STEP;   // запоминаем стартовое положение
}

void loop() {
  int pos = analogRead(POT_PIN) / STEP;
  while (pos > lastPos) { tap(CONSUMER_CONTROL_VOLUME_INCREMENT); lastPos++; }  // громче
  while (pos < lastPos) { tap(CONSUMER_CONTROL_VOLUME_DECREMENT); lastPos--; }  // тише

  bool pressed = (digitalRead(BUTTON_PIN) == LOW);
  if (pressed && !wasPressed) tap(CONSUMER_CONTROL_MUTE);   // нажатие: звук вкл/выкл
  wasPressed = pressed;
  delay(20);
}
```

Деление на `STEP` создаёт «зону нечувствительности»: шум АЦП не вызывает ложных нажатий. Если громкость сдвигается сама, увеличьте `STEP`.

19.6 Мышь

Класс `USBHIDMouse` двигает курсор и нажимает кнопки: `Mouse.move(dx, dy)` — относительный сдвиг, `Mouse.click()` — щелчок. Пример ниже обводит курсор по квадрату, пока удерживается кнопка. Условие по кнопке стоит и на внешнем уровне, и внутри цикла, чтобы остановка была мгновенной.

Листинг 19.3: Мышь рисует квадрат, пока нажата кнопка

```
#include "USB.h"
#include "USBHIDMouse.h"

const int BUTTON_PIN = 5;                 // пока кнопка нажата, курсор «рисует» квадрат
USBHIDMouse Mouse;

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  Mouse.begin();
  USB.begin();
}

void loop() {
  if (digitalRead(BUTTON_PIN) != LOW) {   // кнопка отпущена -- ничего не делаем
    delay(20);
    return;
  }
  const int8_t dx[4] = { 5, 0, -5, 0 };   // вправо, вниз, влево, вверх
  const int8_t dy[4] = { 0, 5, 0, -5 };
  for (int side = 0; side < 4; side++) {
    for (int i = 0; i < 20; i++) {
      if (digitalRead(BUTTON_PIN) != LOW) return;   // отпустили -- сразу остановиться
      Mouse.move(dx[side], dy[side]);
      delay(10);
    }
  }
}
```

19.7 Самостоятельный опыт: пульт для презентаций

Сделайте отдельный скетч с двумя кнопками: **GPIO5** посылает `KEY_RIGHT_ARROW` (следующий слайд), **GPIO6** — `KEY_LEFT_ARROW` (предыдущий). Каждый вызов должен происходить один раз на нажатие, а не непрерывно, пока кнопка удерживается. Проверьте пульт в слайд-шоу на своём компьютере.

Если не работает

- **Компьютер не видит клавиатуру** — кабель вставлен в порт «UART», а не «USB», либо не выбран USB Mode: USB-OTG (TinyUSB).
- **Не прошивается после включения TinyUSB** — прошивайте через порт «UART»; если не получается, войдите в загрузчик (BOOT + RST) и повторите.
- **Вместо английских букв печатаются русские** — на компьютере включена русская раскладка; переключите её на английскую.

Итоги урока

- Порт «UART» работает через мост и даёт последовательный порт, а порт «USB» (**GPIO19**/**GPIO20**) подключён прямо к чипу.
- Для HID выберите USB Mode: USB-OTG (TinyUSB); клавиатуру, мышь и громкость дают классы `USBHIDKeyboard`, `USBHIDMouse`, `USBHIDConsumerControl`.
- Всегда ограничивайте ввод условием по кнопке и помните вход в загрузчик (BOOT + RST).

Проверь себя

1. Чем нативный USB отличается от порта «UART»?
2. Какую настройку в Arduino IDE нужно изменить, чтобы плата стала клавиатурой?
3. Как выйти из ситуации, если плата «печатает» без остановки?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Настройте режим USB-устройства и убедитесь, что компьютер видит плату как клавиатуру.
2. ★ По нажатию кнопки вводите ваше имя в текстовом редакторе.
3. ★★ Сделайте кнопку, вводящую сочетание Ctrl+C, а по долгому нажатию — Ctrl+V.
4. ★★ Регулируйте громкость потенциометром, как в примере, и добавьте кнопку выключения звука.
5. ★★★ Добавьте кнопки «пауза/плей» и «следующий трек» (медиаклавиши).
6. ★★★★ Сделайте «защитный» режим: HID работает только после удержания кнопки 2 секунды, а светодиод показывает состояние.

Урок 20. ESP32-S3 CAM: камера

Цель	запустить камеру на плате ESP32-S3-CAM, показать видео в браузере и сохранить снимок на microSD
Опирается на	урок 8 (Wi-Fi и веб-сервер), урок 10 (карта памяти), урок 11 (PSRAM)
Результат	плата показывает поток MJPEG на странице в браузере, отдаёт снимок по адресу `/capture` и сохраняет фото на карту
Понадобится	плата ESP32-S3-CAM (например, ESP32-S3 WROOM CAM) с камерой OV2640, кабель USB, microSD-карта, питание 5 V от порта с током не менее 1 A

20.1 Сначала: камера, которая сама сжимает картинку

Камера платы — это крошечная микросхема-сенсор. Она смотрит на мир через линзу, превращает свет в цифровые числа и сразу упаковывает снимок в формат JPEG — такой же, как фотографии в телефоне. Плата получает не миллионы отдельных точек, а готовый небольшой файл. Его можно сохранить на карту или передать по Wi-Fi в браузер. Если слать такие кадры один за другим, получится видео — MJPEG. Поэтому для работы нужны три вещи: правильно подключённая камера, память для кадров (PSRAM) и аккуратное обращение с буферами — об этом ниже.

20.2 Что такое плата ESP32-S3-CAM

ESP32-S3-CAM (в том числе ESP32-S3 WROOM CAM и совместимые платы разных производителей) — это ESP32-S3 с разъёмом камеры, слотом microSD и памятью PSRAM на одной плате. Такая плата во многом похожа на знаменитую ESP32-CAM, но удобнее.

	ESP32-CAM (старая)	ESP32-S3-CAM
Чип	ESP32, 2 ядра Xtensa LX6	ESP32-S3, 2 ядра Xtensa LX7 с инструкциями для ИИ
Память PSRAM	около 4 MB, шина Quad	обычно 8 MB, шина Octal (OPI)
USB и прошивка	нет USB, нужен внешний USB–UART-программатор	нативный USB: прошивка и порт по одному кабелю
Камера	OV2640	OV2640 или OV5640 (зависит от комплекта)
microSD	есть	есть
Bluetooth	Classic и BLE	только BLE 5

Конкретные характеристики зависят от производителя: сверяйтесь с описанием своей платы.

20.3 Сенсоры OV2640 и OV5640

В комплекте обычно идёт сенсор OV2640 (2 Мп). Более новый OV5640 (5 Мп) даёт больше деталей, но греется сильнее и требует больше памяти. Оба умеют сами сжимать кадр в JPEG, и драйвер `esp32-camera`, который входит в ядро Arduino-ESP32, поддерживает оба.

	OV2640	OV5640
Матрица	2 Мп, до 1600×1200 (UXGA)	5 Мп, до 2592×1944
JPEG в самом датчике	есть	есть
Автофокус	нет, объектив с фиксированным фокусом	есть у вариантов с механизмом фокусировки (AF)
Нагрев и потребление	ниже	выше
Для чего подходит	видеопоток, фото, учебные проекты	подробные фото, но нагрузка на память и Wi-Fi выше

Возможности указаны ориентировочно: они зависят от модуля, объектива и версии драйвера.

20.4 Интерфейс DVP и путь кадра

Камера соединяется с ESP32-S3 по **параллельному интерфейсу DVP** (Digital Video Port). Данные идут по восьми линиям D0–D7 сразу целым байтом, а ритм задают несколько служебных сигналов:

- **XCLK** — тактовая частота, которую плата *подаёт* камере (около 20 MHz);
- **PCLK** — тактовая частота, с которой камера *выдаёт* байты данных;
- **VSYNC** — импульс «начался новый кадр»;
- **HREF** — уровень HIGH, пока идёт строка кадра.

Отдельно, по двум проводам, плата настраивает камеру (разрешение, яркость, качество JPEG). Эта шина называется **SCCB**, фактически она совместима с I^2C и называется SIOD/SIOC (данные/такт).

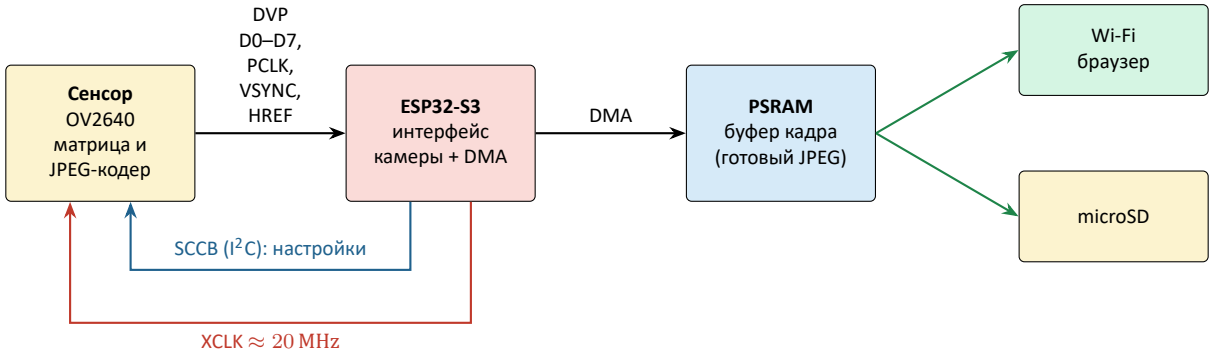


Рис. 20.1: Путь кадра: сенсор формирует JPEG, интерфейс DVP передаёт байты в ESP32-S3, DMA кладёт кадр в PSRAM, а программа отправляет его по Wi-Fi или записывает на карту

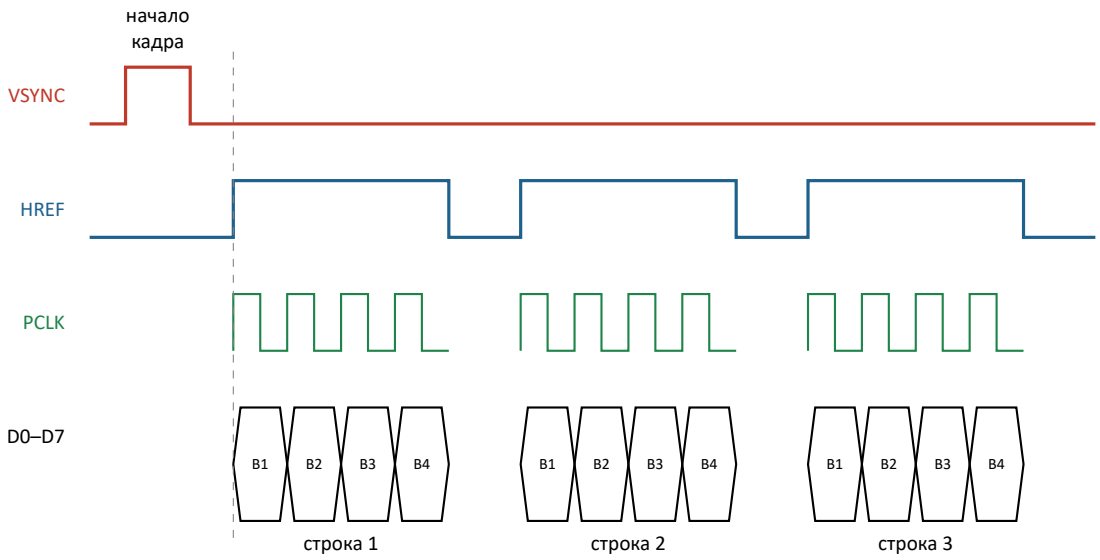


Рис. 20.2: Временная диаграмма DVP (упрощённо). VSYNC открывает кадр, HREF охватывает строку, а на каждый такт PCLK приходит один байт данных. Для JPEG-режима байты — это уже сжатый поток, поэтому «строка» условна

20.5 Распиновка плат типа ESP32-S3-EYE

Какой GPIO соединён с каким выводом камеры, определяется разводкой платы. Большинство плат ESP32-S3-CAM повторяет схему *Espressif ESP32-S3-EYE*; в примере CameraWebServer из ядра она называется `CAMERA_MODEL_ESP32S3_EYE`.

Сигнал	GPIO	Сигнал	GPIO	Сигнал	GPIO	Сигнал	GPIO
XCLK	15	SIOD (SDA)	4	SIOC (SCL)	5	PCLK	13
VSYNC	6	HREF	7	PWDN	−1	RESET	−1
D0	11	D1	9	D2	8	D3	10
D4	12	D5	18	D6	17	D7	16

Значение −1 означает, что вывод не используется. **Эта таблица верна для платы типа ESP32-S3-EYE.** Сверьте её со схемой (schematic) именно вашей платы: производитель мог выбрать другие выводы.

Осторожно

Камера занимает **GPIO4**, **GPIO5**, **GPIO6**, **GPIO7**, **GPIO8**, **GPIO9**, **GPIO10**, **GPIO11**, **GPIO12**, **GPIO13**, **GPIO15**, **GPIO16**, **GPIO17** и **GPIO18**. Поэтому на плате с камерой светодиод, кнопку и I^2C из прошлых уроков (**GPIO4**, **GPIO5**, **GPIO8**, **GPIO9**) **придётся перенести** на свободные выводы, например **GPIO1**, **GPIO2**, **GPIO14**, **GPIO21**, **GPIO41**, **GPIO42** (проверьте по схеме, какие из них выведены на разъём). Выводы **GPIO19** и **GPIO20** заняты USB.

20.6 Настройки Arduino IDE

Выберите плату ESP32S3 Dev Module. Затем в меню Tools задайте:

- PSRAM → OPI PSRAM для модулей с Octal PSRAM (например, N16R8, N8R8). Для модулей с Quad PSRAM (N4R2, N8R2) выбирайте QSPI PSRAM. Неверный выбор приведёт к ошибке запуска PSRAM.
- Partition Scheme → Huge APP (3MB No OTA/1MB SPIFFS): программа с камерой и Wi-Fi большая и в стандартный раздел не помещается.

- USB CDC On Boot → Enabled, если Serial Monitor подключён через нативный порт «USB».
- Flash Size — по вашей плате (чаще всего 8 или 16 МБ).

20.7 Пример 1: запуск камеры

Минимальная программа: проверить PSRAM, инициализировать камеру, взять один кадр, вывести его размер и вернуть кадр драйверу. Значения в структуре `camera_config_t` задают выводы (таблица выше) и параметры кадра.

Листинг 20.1: Инициализация камеры и размер кадра

```
#include "esp_camera.h"

// Выводы камеры для плат типа ESP32-S3-EYE (CAMERA_MODEL_ESP32S3_EYE).
// Сверьте их со схемой своей платы!
#define PWDN_GPIO_NUM  -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM  15
#define SIOD_GPIO_NUM  4
#define SIOC_GPIO_NUM  5
#define Y9_GPIO_NUM    16   // D7
#define Y8_GPIO_NUM    17   // D6
#define Y7_GPIO_NUM    18   // D5
#define Y6_GPIO_NUM    12   // D4
#define Y5_GPIO_NUM    10   // D3
#define Y4_GPIO_NUM    8    // D2
#define Y3_GPIO_NUM    9    // D1
#define Y2_GPIO_NUM    11   // D0
#define VSYNC_GPIO_NUM 6
#define HREF_GPIO_NUM  7
#define PCLK_GPIO_NUM  13

bool startCamera() {
  camera_config_t config = {};
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;         // тактовая частота датчика, 20 МГц
  config.pixel_format = PIXFORMAT_JPEG;   // сжатие в JPEG делает сам датчик
  config.frame_size = FRAMESIZE_VGA;      // 640x480
  config.jpeg_quality = 12;               // 0..63: меньше число -- лучше качество
  config.fb_count = 2;                    // два буфера кадра в PSRAM
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.grab_mode = CAMERA_GRAB_LATEST;  // при отставании брать самый свежий кадр

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
```

```
    Serial.printf("Камера не запустилась, код 0x%x\n", err);
    return false;
  }
  return true;
}

void setup() {
  Serial.begin(115200);
  delay(1000);                              // дать USB-порту время подключиться
  if (!psramFound()) {
    Serial.println("PSRAM не найдена: выберите Tools -> PSRAM -> OPI PSRAM");
    return;
  }
  Serial.printf("PSRAM: %u байт\n", (unsigned)ESP.getPsramSize());
  if (!startCamera()) return;

  camera_fb_t *fb = esp_camera_fb_get();  // взять кадр из очереди драйвера
  if (!fb) {
    Serial.println("Кадр не получен");
    return;
  }
  Serial.printf("Кадр: %ux%u, %u байт JPEG\n",
                (unsigned)fb->width, (unsigned)fb->height, (unsigned)fb->len);
  esp_camera_fb_return(fb);                 // обязательно вернуть буфер драйверу
}

void loop() {}
```

20.8 Параметры качества и буферы кадра

Четыре поля структуры задают баланс между качеством, скоростью и памятью.

Поле	Что делает
`frame_size`	разрешение: `FRAMESIZE_QVGA` (320×240), `VGA` (640×480), `SVGA` (800×600), `XGA`, `UXGA` (1600×1200)
`jpeg_quality`	0–63: *меньше число — лучше качество и больше файл*; обычно 10–12
`fb_count`	число буферов кадра в PSRAM; два буфера позволяют снимать новый кадр, пока предыдущий отправляется
`grab_mode`	`CAMERA_GRAB_WHEN_EMPTY` — новый кадр, когда буфер свободен; `CAMERA_GRAB_LATEST` — всегда самый свежий кадр

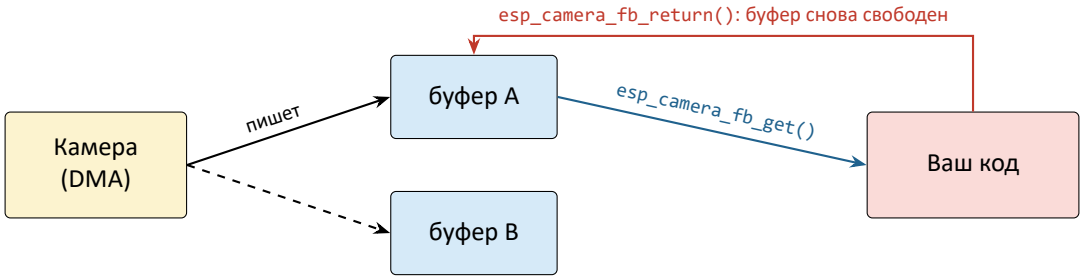


Рис. 20.3: Кадровые буферы. Драйвер заполняет буферы по очереди, а программа берёт готовый кадр и обязательно возвращает его. Если не вернуть, свободные буферы закончатся и `fb_get()` начнёт возвращать `NULL`

Осторожно

Каждый вызов `esp_camera_fb_get()` должен закончиться вызовом `esp_camera_fb_return()` с этим же кадром — в том числе при ошибке записи или отключении клиента. Забытый кадр навсегда занимает буфер, и через несколько секунд видеопоток остановится. После возврата указатель `fb` использовать нельзя.

Ориентировочный размер JPEG-кадра и скорость при качестве 10–12. Реальные цифры зависят от сцены, освещения и качества Wi-Fi.

Разрешение	Размер кадра	Размер JPEG (ориентировочно)	Кадров в секунду
QVGA 320 × 240	76 800 точек	5 – 15 kB	20–30
VGA 640 × 480	307 200 точек	15 – 40 kB	10–25
SVGA 800 × 600	480 000 точек	25 – 60 kB	8–20
UXGA 1600 × 1200	1 920 000 точек	100 – 250 kB	3–10

20.9 Пример 2: снимок и видеопоток в браузере

Во втором примере плата работает как веб-сервер (урок 8). Адрес `/capture` возвращает один JPEG, а адрес `/stream` — поток **MJPEG**: ответ сервера не заканчивается, а превращается в бесконечную последовательность JPEG-кадров, которую браузер показывает как видео внутри обычного тега `<img>`. Простой `WebServer` обслуживает одного клиента за раз, и пока идёт поток, остальные ждут: для урока этого достаточно. Поток сам заканчивается через 10 минут (`MAX_STREAM_MS`).

Листинг 20.2: Снимок `/capture` и поток `/stream`

```
#include <WiFi.h>
#include <WebServer.h>
#include "esp_camera.h"

const char* SSID = "MyNetwork";
const char* PASS = "MyPassword";
const uint32_t MAX_STREAM_MS = 10UL * 60 * 1000;   // поток сам закончится через 10 минут

// Выводы камеры (ESP32-S3-EYE). Сверьте со схемой своей платы!
#define XCLK_GPIO_NUM  15
#define SIOD_GPIO_NUM  4
#define SIOC_GPIO_NUM  5
#define VSYNC_GPIO_NUM 6
#define HREF_GPIO_NUM  7
#define PCLK_GPIO_NUM  13

WebServer server(80);

bool startCamera() {
  camera_config_t config = {};
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = 11;  config.pin_d1 = 9;   config.pin_d2 = 8;   config.pin_d3 = 10;
  config.pin_d4 = 12;  config.pin_d5 = 18;  config.pin_d6 = 17;  config.pin_d7 = 16;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
```

```
  config.pin_pwdn = -1;
  config.pin_reset = -1;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;      // для потока хватает 640x480
  config.jpeg_quality = 12;
  config.fb_count = 2;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.grab_mode = CAMERA_GRAB_LATEST;
  return esp_camera_init(&config) == ESP_OK;
}

void handleRoot() {                       // страница с картинкой, подключённой к потоку
  server.send(200, "text/html",
              "<!DOCTYPE html><html><head><meta charset='utf-8'></head><body>"
              "<h3>ESP32-S3 CAM</h3><img src='/stream' width='640'>"
              "<p><a href='/capture'>Один снимок</a></p></body></html>");
}

void handleCapture() {                    // один JPEG-кадр по адресу /capture
  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) { server.send(500, "text/plain", "Кадр не получен"); return; }
  server.sendHeader("Content-Disposition", "inline; filename=capture.jpg");
  server.send_P(200, "image/jpeg", (const char*)fb->buf, fb->len);
  esp_camera_fb_return(fb);               // вернуть буфер, иначе кадры закончатся
}

void handleStream() {                     // MJPEG: непрерывная череда JPEG-кадров
  WiFiClient client = server.client();
  client.print("HTTP/1.1 200 OK\r\n"
               "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n");
  uint32_t start = millis();
  while (client.connected() && millis() - start < MAX_STREAM_MS) {
    camera_fb_t *fb = esp_camera_fb_get();
    if (!fb) break;
    size_t len = fb->len;
    client.printf("--frame\r\nContent-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n",
                  (unsigned)len);
    size_t sent = client.write(fb->buf, len);
    client.print("\r\n");
    esp_camera_fb_return(fb);             // вернуть кадр до любого выхода из цикла
    if (sent != len) break;               // клиент отключился -- выходим
  }
}

void setup() {
  Serial.begin(115200);
  if (!psramFound() || !startCamera()) {
    Serial.println("Камера не запустилась: проверьте PSRAM, шлейф и выводы");
    return;
  }
  WiFi.begin(SSID, PASS);
  uint32_t start = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) delay(300);
  if (WiFi.status() != WL_CONNECTED) { Serial.println("Нет Wi-Fi"); return; }
  Serial.print("Откройте в браузере: http://");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);
```

```
  server.on("/capture", handleCapture);
  server.on("/stream", handleStream);
  server.begin();
}

void loop() {
  server.handleClient();
}
```

Запустите программу, откройте Serial Monitor, скопируйте IP-адрес в браузер. Если картинка вверх ногами, поверните плату или вызовите после инициализации `s->set_vflip(s, 1)`, где `s` — это `esp_camera_sensor_get()`.

20.10 Пример 3: снимок на microSD

На платах типа ESP32-S3-EYE слот microSD подключён к периферии SDMMC, а не к SPI из урока 10. Используется библиотека `SD_MMC`, выводы: CLK — **GPIO39**, CMD — **GPIO38**, D0 — **GPIO40**. Режим «1-bit» использует одну линию данных, этого хватает для фотографий. Выводы слота, как и камеры, **сверяйте со схемой своей платы**. Карта должна быть отформатирована в FAT32.

Листинг 20.3: Фото на карту раз в 5 секунд

```
#include "esp_camera.h"
#include "SD_MMC.h"

// Слот microSD (ESP32-S3-EYE): CLK 39, CMD 38, D0 40. Сверьте со своей платой!
const int SD_CLK = 39;
const int SD_CMD = 38;
const int SD_D0  = 40;
const uint32_t SHOT_PERIOD_MS = 5000;     // один снимок раз в 5 секунд
const int MAX_SHOTS = 10;                 // и не больше 10 штук

int shotNo = 0;
uint32_t lastShot = 0;

bool startCamera() {
  camera_config_t config = {};
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = 11;  config.pin_d1 = 9;   config.pin_d2 = 8;   config.pin_d3 = 10;
  config.pin_d4 = 12;  config.pin_d5 = 18;  config.pin_d6 = 17;  config.pin_d7 = 16;
  config.pin_xclk = 15;
  config.pin_pclk = 13;
  config.pin_vsync = 6;
  config.pin_href = 7;
  config.pin_sccb_sda = 4;
  config.pin_sccb_scl = 5;
  config.pin_pwdn = -1;
  config.pin_reset = -1;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_SVGA;     // 800x600: для фото чуть больше, чем для потока
  config.jpeg_quality = 10;
  config.fb_count = 1;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  return esp_camera_init(&config) == ESP_OK;
}
```

```
37
38 void setup() {
39   Serial.begin(115200);
40   delay(1000);
41   if (!psramFound() || !startCamera()) {
42     Serial.println("Камера не запустилась");
43     return;
44   }
45   SD_MMC.setPins(SD_CLK, SD_CMD, SD_D0);  // режим с одной линией данных (1-bit)
46   if (!SD_MMC.begin("/sdcard", true)) {
47     Serial.println("microSD не найдена: проверьте карту (FAT32) и выводы слота");
48     return;
49   }
50   Serial.println("Готово: снимки будут сохраняться в /photo_N.jpg");
51 }
52
53 void loop() {
54   if (shotNo >= MAX_SHOTS || millis() - lastShot < SHOT_PERIOD_MS) return;
55   lastShot = millis();
56
57   camera_fb_t *fb = esp_camera_fb_get();
58   if (!fb) { Serial.println("Кадр не получен"); return; }
59
60   String path = "/photo_" + String(shotNo) + ".jpg";
61   File f = SD_MMC.open(path, FILE_WRITE);
62   if (f) {
63     f.write(fb->buf, fb->len);            // JPEG уже готов -- пишем как есть
64     f.close();
65     Serial.printf("Сохранено %s (%u байт)\n", path.c_str(), (unsigned)fb->len);
66     shotNo++;
67   } else {
68     Serial.println("Не удалось открыть файл на карте");
69   }
70   esp_camera_fb_return(fb);               // кадр возвращаем в любом случае
71 }
```

20.11 Питание, нагрев, антенна

- **Питание.** Камера, Wi-Fi и карта вместе потребляют ориентировочно до $300 – 500$ mA в пике. Используйте хороший USB-порт и короткий качественный кабель: слабый порт или длинный тонкий провод приводят к перезагрузкам из-за brownout (урок 14).
- **Нагрев.** Во время потока модуль заметно тёплый: это нормально. Если корпус обжигает палец, снизьте разрешение, частоту кадров или добавьте небольшой радиатор.
- **Антенна.** Плата с печатной антенной не любит металлических корпусов и руки у края модуля. У плат с разъёмом для внешней антенны нужно проверить, какая антенна выбрана перемычкой.
- **Шлейф камеры** вставляйте и вынимайте только при отключённом питании, контактами в нужную сторону.

20.12 Типичные ошибки

Сообщение или признак	Что проверить
`Camera probe failed with error 0x105`	шлейф вставлен не до конца или не той стороной; неверные **GPIO4**/**GPIO5** (SCCB); камера не получает питание
`0x20001` (камера не найдена)	то же: проблемы с шлейфом или SCCB; неверный тип сенсора
`PSRAM ID read error` или `psramFound() == false`	в меню выбран не тот тип PSRAM (OPI/QSPI)
Полосы, шум и искажения на кадре	помехи по питанию или длинному шлейфу; снизьте `xclk_freq_hz` до 10 MHz
Перезагрузки и `Brownout detector`	недостаточное питание; другой кабель или порт

20.13 Что дальше

Когда видеопоток работает, камеру можно «научить» понимать картинку. Библиотеки Espressif **ESP-DL** (нейросети на ESP32-S3) и **ESP-WHO** (готовые примеры: обнаружение лиц, распознавание объектов, чтение QR-кодов) используют те же буферы кадров, что и этот урок. Это отдельная большая тема, с которой удобно начать после освоения базового курса.

20.14 Самостоятельный опыт: фотоловушка

Сделайте отдельный скетч. Подключите кнопку между свободным выводом (например, **GPIO21**) и GND, подтяжка внутренняя. Когда кнопка нажата, плата делает снимок и сохраняет его на microSD с очередным номером в имени. Добавьте светодиод, мигающий на время съёмки. Не забывайте про дребезг (урок 3) и про `esp_camera_fb_return()`.

Осторожно

Этика и безопасность съёмки. Камера — не игрушка.

- Снимать людей можно только с их согласия; в школе, спортзале и других местах, где есть дети, — только с разрешения педагога.
- Не выкладывайте видеопоток в открытую сеть и не открывайте порт камеры в интернет. Без пароля на вашу камеру сможет смотреть любой.
- Если используете собственную точку доступа (урок 8, SoftAP), обязательно задайте пароль не короче 8 символов; не оставляйте камеру включённой без присмотра.

Если не работает

- **«Camera probe failed»** — проверьте шлейф, выводы SCCB (**GPIO4**, **GPIO5**) и питание; убедитесь, что распиновка соответствует вашей плате.
- **Нет PSRAM или программа не помещается** — выберите OPI PSRAM (или QSPI по модулю) и схему Huge APP.
- **Поток зависает через несколько секунд или плата перезагружается** — не возвращается кадр (`esp_camera_fb_return()`) либо слабое питание.

Итоги урока

- Камера по DVP передаёт байты в PSRAM; сенсор сам формирует JPEG, а настраивается по SCCB.
- Распиновку камеры сверяйте со схемой платы; камера занимает много GPIO, и прежние

выводы нужно переносить.

- Каждый кадр обязательно возвращают драйверу; качество и скорость задают `frame_size`, `jpeg_quality`, `fb_count`.

Проверь себя

1. Какие сигналы входят в интерфейс DVP и для чего нужна шина SCCB?
2. Зачем после `esp_camera_fb_get()` нужен вызов `esp_camera_fb_return()`?
3. Какие настройки Arduino IDE нужны для камеры и почему?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите камеру и выведите в Serial Monitor размер кадра.
2. ★ Измените `frame_size` на QVGA и на UXGA и сравните размер кадра.
3. ★★ Измените качество JPEG и постройте таблицу «качество — размер кадра».
4. ★★ Добавьте на страницу кнопку «Сохранить на карту» (обработчик `/save`).
5. ★★★ Сделайте покадровую съёмку: снимок раз в 10 секунд с именем файла по времени NTP.
6. ★★★★ Сделайте фотоловушку: кадр сохраняется, когда датчик движения срабатывает, а между снимками плата спит.

Урок 21. Таймеры и конечные автоматы

Цель	отмерять точные интервалы аппаратным таймером и строить программу как набор состояний с понятными переходами
Опирается на	урок 1 (`millis()` без `delay()`), урок 3 (прерывания и правила для обработчика)
Результат	светодиод мигает строго по таймеру; три светодиода работают как светофор с кнопкой пешехода
Понадобится	три светодиода (красный, жёлтый, зелёный), три резистора 220 Ω, кнопка

21.1 Сначала: будильник и расписание

До сих пор время отмеряла сама программа: в каждом проходе `loop()` она смотрела на часы `millis()` и решала, не пора ли что-то сделать. Это работает, пока `loop()` крутится быстро. Аппаратный таймер — это будильник внутри чипа: он считает такты кварца и в нужный момент сам прерывает программу, чем бы она ни была занята. Вторая идея урока — конечный автомат. Это способ записать поведение устройства как расписание: в каком состоянии оно находится и при каком условии переходит в следующее. Так устроены светофор, стиральная машина и меню любого прибора.

21.2 Аппаратный таймер

Таймер — это счётчик, который увеличивается с заданной частотой. Когда он досчитывает до установленного значения, срабатывает прерывание, а счётчик начинает заново. У ESP32-S3 четыре таких таймера общего назначения.

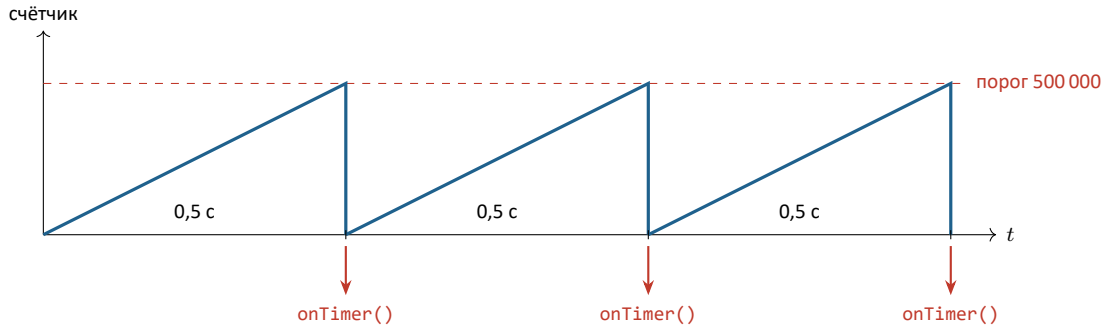


Рис. 21.1: Таймер считает до порога, вызывает обработчик и начинает счёт заново. Интервал не зависит от того, чем занята основная программа

В Arduino-ESP32 3.x таймер настраивается тремя вызовами: `timerBegin(частота)` запускает счётчик, `timerAttachInterrupt()` назначает обработчик, `timerAlarm(таймер, порог, повтор, число)` задаёт порог срабатывания.

Листинг 21.1: Мигание по аппаратному таймеру

```
const int LED_PIN = 4;

hw_timer_t* timer = nullptr;
volatile uint32_t ticks = 0;                  // меняется в обработчике

void ARDUINO_ISR_ATTR onTimer() {             // вызывается ровно раз в 0,5 с
  ticks++;
  digitalWrite(LED_PIN, !digitalRead(LED_PIN));
}

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  timer = timerBegin(1000000);                // счётчик: 1 000 000 тиков в секунду
  timerAttachInterrupt(timer, &onTimer);
  timerAlarm(timer, 500000, true, 0);         // сигнал каждые 500 000 тиков, без конца
}

void loop() {
  static uint32_t shown = 0;
  uint32_t now = ticks;
  if (now != shown) {                         // печатаем здесь, а не в обработчике
    shown = now;
    Serial.printf("Срабатываний: %lu\n", now);
  }
}
```

Осторожно

Обработчик таймера — такое же прерывание, как в уроке 3. В нём нельзя вызывать `delay()`, `Serial.print()`, работать с Wi-Fi и дисплеем. Изменили переменную или вывод — и вышли. Общие с основной программой переменные объявляйте `volatile`.

Способ отсчёта	Когда выбирать	Ограничение
`millis()` в `loop()`	мигание, опрос датчиков, таймауты	точность зависит от скорости `loop()`
Аппаратный таймер	строгий период: измерения, генерация сигнала	обработчик должен быть очень коротким
Задача FreeRTOS с `vTaskDelay()`	долгая работа по расписанию (урок 7)	шаг времени — 1 мс

21.3 Конечный автомат

Попробуйте описать светофор словами: «горит зелёный; если пешеход нажал кнопку и прошло не меньше пяти секунд, включить жёлтый; через две секунды — красный…». В этом описании есть всё, что нужно программе: **состояния** (что сейчас горит), **события** (нажатие, истёкшее время) и **переходы** между состояниями.

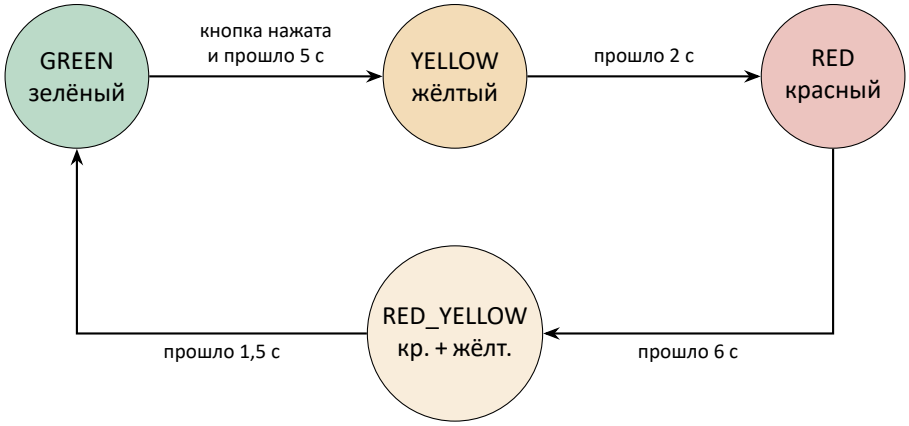


Рис. 21.2: Диаграмма состояний светофора. Кружок — состояние, стрелка — переход, подпись — условие перехода

В программе состояние хранит одна переменная. Тип `enum class` даёт состояниям имена вместо загадочных чисел, а оператор `switch` для каждого состояния проверяет только его переходы. Светодиоды подключены к **GPIO4**, **GPIO6** и **GPIO7** через резисторы 220 Ω, а кнопка пешехода стоит между **GPIO5** и GND: внешний резистор не нужен, включена внутренняя подтяжка.

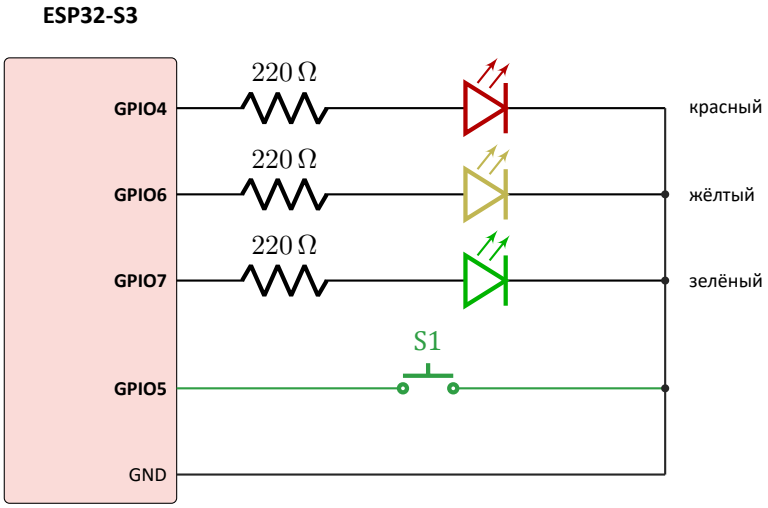


Рис. 21.3: Подключение светофора: три светодиода через резисторы 220 Ω и кнопка пешехода. Аноды светодиодов (длинные ножки) смотрят в сторону платы, катоды идут на общую землю

Листинг 21.2: Светофор с кнопкой пешехода как конечный автомат

```
const int RED_PIN = 4;
const int YELLOW_PIN = 6;
const int GREEN_PIN = 7;
const int BTN_PIN = 5;                          // кнопка пешехода

enum class State { GREEN, YELLOW, RED, RED_YELLOW };

State state = State::GREEN;
uint32_t enteredAt = 0;                         // когда вошли в текущее состояние
bool request = false;                           // пешеход нажал кнопку

void enter(State s) {                           // единственное место смены состояния
  state = s;
  enteredAt = millis();
  digitalWrite(RED_PIN,    s == State::RED || s == State::RED_YELLOW);
  digitalWrite(YELLOW_PIN, s == State::YELLOW || s == State::RED_YELLOW);
  digitalWrite(GREEN_PIN,  s == State::GREEN);
}

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BTN_PIN, INPUT_PULLUP);
  enter(State::GREEN);
}

void loop() {
  if (digitalRead(BTN_PIN) == LOW) request = true;   // событие запоминаем
  uint32_t t = millis() - enteredAt;                 // время в текущем состоянии

  switch (state) {
    case State::GREEN:                              // машины едут, пока нет запроса
      if (request && t > 5000) enter(State::YELLOW);
      break;
    case State::YELLOW:
      if (t > 2000) enter(State::RED);
      break;
    case State::RED:                                // пешеходы идут
      if (t > 6000) { request = false; enter(State::RED_YELLOW); }
      break;
    case State::RED_YELLOW:
      if (t > 1500) enter(State::GREEN);
      break;
  }
}
```

Обратите внимание на три приёма. Состояние меняется только в функции `enter()`: там же запоминается время входа и переключаются светодиоды, поэтому забыть что-то одно невозможно. Нажатие кнопки не обрабатывается сразу, а запоминается во флаге `request`. И в программе нет ни одного `delay()`: в любой момент она готова реагировать на новые события.

Совет

Сначала рисуйте диаграмму на бумаге, потом пишите код. Если на диаграмме трудно объяснить какую-то стрелку, в программе в этом месте будет ошибка.

21.4 Самостоятельный опыт: ночной режим

Добавьте светофору пятое состояние `NIGHT`: жёлтый мигает раз в секунду. Переход в ночной режим и обратно — по команде `night` и `day` из Serial Monitor (урок 2). Сначала дорисуйте диаграмму: из каких состояний можно попасть в `NIGHT` и куда из него возвращаться?

Если не работает

- **Плата перезагружается после запуска таймера** — в обработчике вызвано что-то медленное (`Serial`, `delay()`) или обработчик срабатывает слишком часто.
- **Автомат «застрял» в одном состоянии** — время входа не обновляется: состояние меняется в обход функции `enter()`.
- **Кнопка срабатывает не всегда** — нажатие проверяется только в одном состоянии; запоминайте его во флаге в начале `loop()`.

Итоги урока

- Аппаратный таймер вызывает обработчик со строгим периодом, независимо от основной программы.
- Конечный автомат — это состояния, события и переходы; в коде это `enum class` и `switch`.
- Состояние меняем в одном месте, события запоминаем во флагах, `delay()` не используем.

Проверь себя

1. Чем отсчёт времени аппаратным таймером отличается от проверки `millis()` в `loop()`?
2. Что такое состояние, событие и переход? Приведите пример из светофора.
3. Почему состояние автомата удобно менять только в одной функции?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите таймер с периодом 1 s и мигайте светодиодом в обработчике.
2. ★ Нарисуйте граф состояний светофора: кружки — состояния, стрелки — переходы.
3. ★★ Измените период таймера так, чтобы светодиод мигал 5 раз в секунду, и проверьте по счёту.
4. ★★ Добавьте светофору зелёный мигающий сигнал (3 вспышки) перед жёлтым как отдельное состояние.
5. ★★★ Добавьте кнопку для пешехода: нажатие запрашивает красный свет в конце текущего цикла.
6. ★★★★ Сделайте кодовый замок: автомат ждёт три нажатия двух кнопок в правильном порядке и зажигает зелёный светодиод; при ошибке — красный и возврат в начало.

Урок 22. Биты, байты и регистры

Цель	читать двоичные и шестнадцатеричные числа, управлять отдельными битами и выводить цифры на семисегментный индикатор через сдвиговый регистр
Опирается на	урок 1 (цифровой выход), урок 10 (последовательная передача по такту)
Результат	программа печатает биты числа в Serial Monitor; индикатор считает от 0 до 9, занимая всего три вывода платы
Понадобится	микросхема 74HC595, семисегментный индикатор с общим катодом, восемь резисторов 220 Ω

22.1 Сначала: восемь выключателей в одном числе

Внутри микроконтроллера нет «чисел» в привычном смысле — есть ряды выключателей, каждый из которых либо включён (1), либо выключен (0). Один выключатель — это бит, восемь подряд — байт. Число 91 и набор «выключен, включён, выключен, включён, включён, выключен, включён, включён» — одно и то же. Умея обращаться с отдельными битами, можно одним байтом зажечь нужные сегменты индикатора, прочитать настройки датчика и понять любую таблицу регистров в документации.

22.2 Двоичная и шестнадцатеричная запись

У каждого бита свой «вес» — степень двойки. Значение байта равно сумме весов включённых битов. Биты нумеруют справа налево, начиная с нуля: бит 0 — младший, бит 7 — старший.

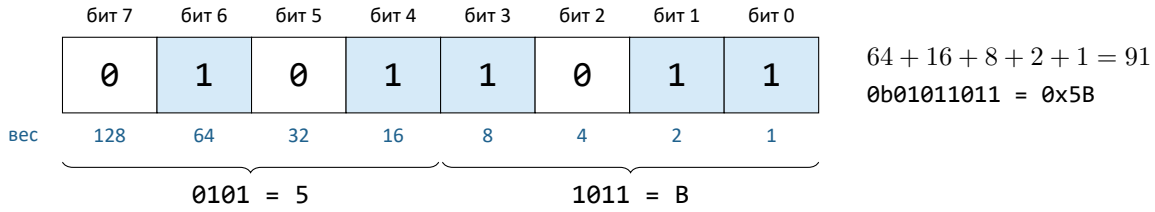


Рис. 22.1: Один байт в трёх записях: двоичной, десятичной и шестнадцатеричной. Каждая шестнадцатеричная цифра заменяет ровно четыре бита

Шестнадцатеричная запись (hex) — просто короткий способ записать биты: четыре бита превращаются в одну цифру от 0 до F (A = 10, B = 11, …, F = 15). В программе двоичное число пишут с приставкой `0b`, шестнадцатеричное — с `0x`. Адрес дисплея `0x3C` из урока 6 — это `0b00111100`, или 60.

22.3 Битовые операции

Операция	Запись	Что делает
И (AND)	`a & b`	бит равен 1, только если он 1 в обоих числах; так *проверяют* и *сбрасывают* биты
ИЛИ (OR)	`a | b`	бит равен 1, если он 1 хотя бы в одном числе; так *устанавливают* биты
Исключающее ИЛИ (XOR)	`a ^ b`	бит равен 1, если биты различаются; так *переключают* биты
НЕ (NOT)	`~a`	меняет каждый бит на противоположный
Сдвиг	`a << n`, `a >> n`	сдвигает биты влево или вправо на n позиций

Выражение `(1 << n)` — это «маска»: число, в котором включён только бит n. С ней записываются четыре приёма, которые встречаются в любой программе для микроконтроллера:

```
x |=  (1 << n);      // установить бит n
x &= ~(1 << n);      // сбросить бит n
x ^=  (1 << n);      // переключить бит n
if (x & (1 << n))    // проверить бит n
```

Листинг 22.1: Битовые операции: результат каждого шага виден в Serial Monitor

```
void printBits(const char* label, uint8_t v) {
  Serial.printf("%-22s", label);
  for (int i = 7; i >= 0; i--) Serial.print((v >> i) & 1);   // от старшего бита к младшему
  Serial.printf("  = 0x%02X = %u\n", v, v);
}

void setup() {
  Serial.begin(115200);
  delay(500);

  uint8_t flags = 0;
  printBits("начало", flags);

  flags |= (1 << 3);                // установить бит 3
  printBits("установили бит 3", flags);

  flags |= (1 << 0) | (1 << 6);     // установить сразу два бита
  printBits("установили 0 и 6", flags);

  flags &= ~(1 << 3);               // сбросить бит 3
  printBits("сбросили бит 3", flags);

  flags ^= (1 << 7);                // переключить бит 7
  printBits("переключили бит 7", flags);

  if (flags & (1 << 6)) Serial.println("бит 6 установлен");   // проверить бит

  uint16_t value = 0x1234;
  uint8_t high = value >> 8;        // старший байт: 0x12
  uint8_t low  = value & 0xFF;      // младший байт: 0x34
  Serial.printf("0x%04X -> старший 0x%02X, младший 0x%02X\n", value, high, low);
}

void loop() {}
```

Осторожно

Не путайте `&` и `&&`, `|` и `||`. Одиночные знаки работают с каждым битом числа, двойные — с условиями «истина/ложь». `if (a & b)` и `if (a && b)` — разные проверки.

22.4 Регистры: память, которая управляет железом

Регистр — это ячейка памяти, биты которой напрямую связаны с аппаратурой. Запись числа в регистр немедленно меняет поведение чипа. Функция `digitalWrite(4, HIGH)` в конце концов делает именно это: записывает единицу в бит 4 регистра выходов. То же можно сделать вручную.

Листинг 22.2: Мигание записью в регистры GPIO напрямую

```
#include "soc/gpio_struct.h"

const int LED_PIN = 4;

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

void loop() {
  GPIO.out_w1ts = (1UL << LED_PIN);   // регистр «установить»: бит 4 -> вывод GPIO4 в HIGH
  delay(500);
  GPIO.out_w1tc = (1UL << LED_PIN);   // регистр «сбросить»: бит 4 -> вывод GPIO4 в LOW
  delay(500);
}
```

В обычных программах пользуйтесь `digitalWrite()`: она понятнее и не зависит от модели чипа. Но знать о регистрах нужно, чтобы читать документацию. Настройки любого датчика — это тоже регистры, только расположенные внутри датчика (урок 26).

22.5 Сдвиговый регистр 74HC595

Семисегментный индикатор — это восемь светодиодов (семь сегментов и точка). Если подключать каждый к своему выводу, на четыре цифры уйдёт вся плата. Микросхема 74HC595 решает задачу: она принимает восемь битов по одному проводу и выставляет их на восьми своих выходах. Плате нужно всего три вывода.

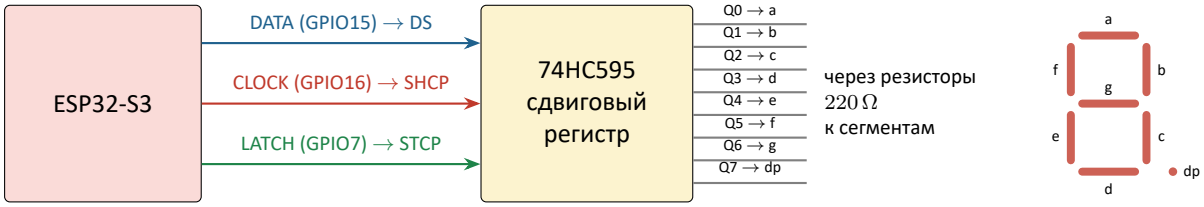


Рис. 22.2: Три вывода платы управляют восемью сегментами. Справа — принятые названия сегментов индикатора

Работа идёт в три шага. Плата опускает LATCH. Затем восемь раз выставляет очередной бит на DATA и «щёлкает» CLOCK — биты один за другим вдвигаются в микросхему. Наконец LATCH поднимается, и все восемь выходов обновляются одновременно. Всю передачу выполняет готовая функция `shiftOut()`.

Остальные выводы микросхемы: питание VCC (16) — к 3V3, GND (8) — к земле, $\overline{\text{OE}}$ (13) — к земле (выходы включены), $\overline{\text{MR}}$ (10) — к 3V3 (сброс не активен). Общий катод индикатора — к GND.

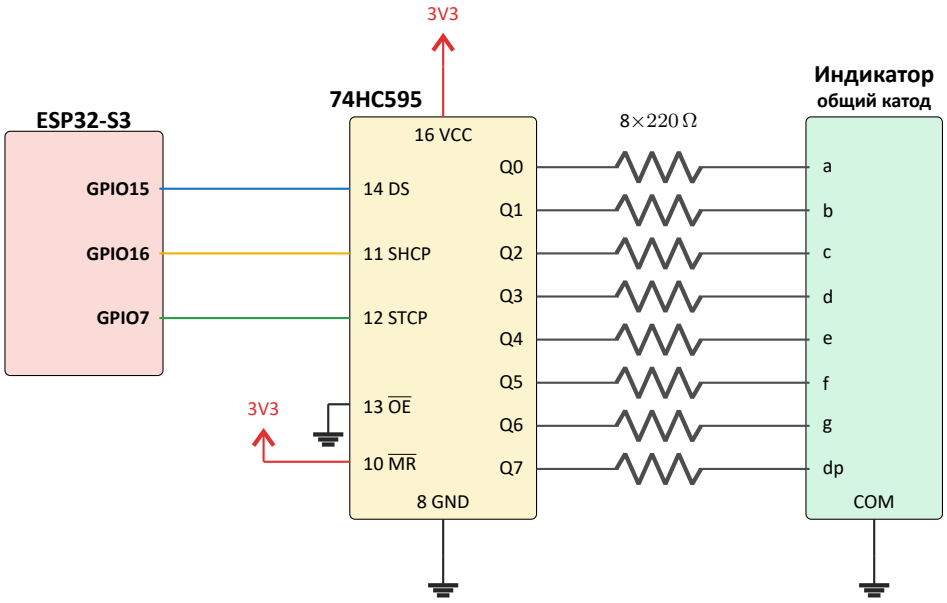


Рис. 22.3: Полная схема подключения 74HC595 и индикатора. Три сигнала идут с платы на DS, SHCP и STCP, выходы Q0–Q7 через резисторы 220 Ω подключены к сегментам a–g и точке, общий катод — на GND

Листинг 22.3: Счёт от 0 до 9 на семисегментном индикаторе

```
const int DATA_PIN  = 15;   // DS   (вывод 14 микросхемы 74HC595)
const int CLOCK_PIN = 16;   // SHCP (вывод 11)
const int LATCH_PIN = 7;    // STCP (вывод 12)

// Сегменты: бит 0 = a, бит 1 = b, ..., бит 6 = g, бит 7 = точка
const uint8_t DIGITS[10] = {
  0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110,
  0b01101101, 0b01111101, 0b00000111, 0b01111111, 0b01101111};

void show(uint8_t pattern) {
  digitalWrite(LATCH_PIN, LOW);                     // защёлка закрыта: выходы не меняются
  shiftOut(DATA_PIN, CLOCK_PIN, MSBFIRST, pattern);  // 8 бит по одному, старший первым
  digitalWrite(LATCH_PIN, HIGH);                    // все 8 выходов обновились разом
}

void setup() {
  pinMode(DATA_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(LATCH_PIN, OUTPUT);
}

void loop() {
  for (int d = 0; d <= 9; d++) {
    show(DIGITS[d]);
    delay(600);
    show(DIGITS[d] | 0b10000000);                   // та же цифра с точкой
    delay(200);
  }
}
```

Каждое число в массиве `DIGITS` — это рисунок цифры: единица в бите означает горящий сегмент. Для цифры 2 горят a, b, d, e, g — биты 0, 1, 3, 4, 6, то есть `0b01011011`. Это то самое число с первого рисунка урока.

Совет

Микросхемы 74HC595 соединяют цепочкой: выход Q7S (вывод 9) первой подключают к входу DS второй, а CLOCK и LATCH делают общими. Два вызова `shiftOut()` подряд — и те же три вывода платы управляют уже шестнадцатью выходами.

22.6 Самостоятельный опыт: счётчик нажатий

Добавьте кнопку из урока 3: каждое нажатие увеличивает цифру на индикаторе, после 9 идёт 0. Затем сделайте так, чтобы точка мигала раз в секунду, не мешая счёту. Подсказка: точку включает операция `| 0b10000000`, а выключает — её отсутствие.

Если не работает

- **Индикатор не светится совсем** — общий вывод индикатора не подключён к GND или вывод $\overline{\text{OE}}$ микросхемы «висит в воздухе».
- **Горят не те сегменты** — перепутан порядок выходов Q0–Q7 или у вас индикатор с общим анодом: для него нужно инвертировать рисунок (`~pattern`) и подключить общий вывод к 3V3.
- **Цифры мелькают случайным образом** — не подключён LATCH или перепутаны CLOCK и LATCH.

Итоги урока

- Байт — это восемь битов с весами 128…1; hex-цифра заменяет четыре бита.
- Маска `(1 << n)` с операциями `|=`, `&=~`, `^=` и `&` устанавливает, сбрасывает, переключает и проверяет бит.
- Сдвиговый регистр превращает три вывода платы в восемь (и больше) выходов.

Проверь себя

1. Как перевести `0b00101101` в десятичную и шестнадцатеричную запись?
2. Какой строкой кода сбросить бит 5 переменной `x`, не меняя остальные биты?
3. Зачем сдвиговому регистру вывод LATCH, если данные уже переданы?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Переведите числа 13, 42 и 255 в двоичную и шестнадцатеричную запись и проверьте в программе.
2. ★ Выведите состояние одного бита числа с помощью сдвига и маски.
3. ★★ Допишите в массив рисунки букв A, b, C, d, E, F и выведите шестнадцатеричный счёт на индикатор.
4. ★★ Включайте, выключайте и переключайте отдельный бит числа функциями-обёртками.
5. ★★★ Подключите восемь светодиодов к 74HC595 и сделайте «бегущий огонь» одной операцией сдвига.

6. ★★★★ Соедините две 74HC595 цепочкой и выведите двузначное число на два индикатора.

Урок 23. Органы управления: энкодер, клавиатура, сенсорные кнопки

Цель	подключать три распространённых способа ввода и выбирать подходящий для своего устройства
Опирается на	урок 3 (кнопка, подтяжка, дребезг, прерывания), урок 5 (потенциометр)
Результат	поворот ручки меняет счётчик, клавиатура печатает нажатую клавишу, касание провода зажигает светодиод
Понадобится	модуль энкодера KY-040, матричная клавиатура 4×4, отрезок провода или фольги

23.1 Сначала: как человек разговаривает с устройством

Одной кнопки хватает, чтобы включить и выключить. Но чтобы выбрать пункт меню, набрать код или плавно настроить громкость, нужны другие органы управления. В этом уроке их три. Энкодер — ручка, которая крутится бесконечно и сообщает, в какую сторону её повернули. Матричная клавиатура — шестнадцать кнопок, которым хватает восьми выводов. Сенсорная кнопка — контакт без механики, который чувствует прикосновение пальца.

Орган управления	Удобен для	Особенность
Кнопка	включить, подтвердить	один вывод на кнопку
Потенциометр	плавная настройка с упорами	помнит положение; нужен АЦП
Энкодер	меню, громкость, счёт	крутится бесконечно; положение считает программа
Клавиатура 4×4	ввод чисел и кодов	16 кнопок на 8 выводах
Сенсорная кнопка	панель под стеклом или пластиком	нет движущихся частей; нужна калибровка

23.2 Энкодер

Внутри энкодера два контакта, A и B, которые при вращении замыкаются со сдвигом. По тому, какой из них сработал первым, определяют направление: в момент спада сигнала A смотрят на уровень B.

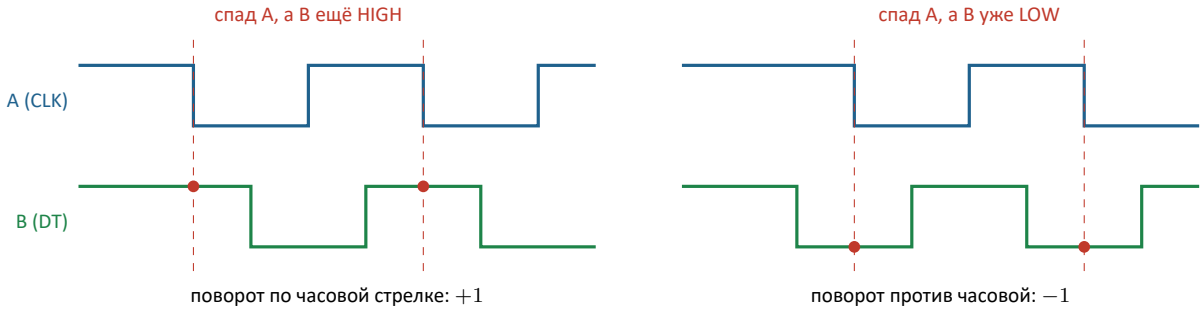


Рис. 23.1: Сигналы энкодера. Направление вращения определяет уровень B в момент спада A

Модуль KY-040 подключают так: «+» к 3V3, GND к земле, CLK к **GPIO6**, DT к **GPIO7**, SW (кнопка на оси) к **GPIO15**. Спад сигнала A ловит прерывание, поэтому ни один щелчок не теряется, даже если основная программа занята.

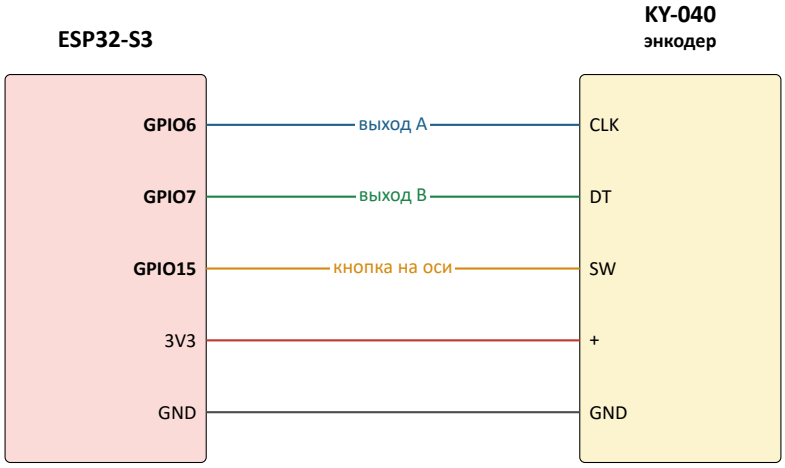


Рис. 23.2: Подключение энкодера KY-040: три сигнала на **GPIO6**, **GPIO7**, **GPIO15**, питание 3,3 V. Подтяжки на плате модуля уже есть, а программа включает ещё и внутренние

Листинг 23.1: Энкодер: счёт поворотов и сброс нажатием на ось

```
const int CLK_PIN = 6;      // выход A энкодера
const int DT_PIN  = 7;      // выход B энкодера
const int SW_PIN  = 15;     // кнопка на оси

volatile int position = 0;
volatile uint32_t lastEdge = 0;

void IRAM_ATTR onClk() {
  uint32_t now = micros();
  if (now - lastEdge < 2000) return;                    // дребезг: игнорируем 2 мс
  lastEdge = now;
  position += (digitalRead(DT_PIN) == HIGH) ? 1 : -1;   // DT показывает направление
}

void setup() {
  Serial.begin(115200);
  pinMode(CLK_PIN, INPUT_PULLUP);
  pinMode(DT_PIN, INPUT_PULLUP);
  pinMode(SW_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(CLK_PIN), onClk, FALLING);
}

void loop() {
  static int shown = 0;
  int p = position;
  if (p != shown) {
    shown = p;
    Serial.printf("Позиция: %d\n", p);
  }
  if (digitalRead(SW_PIN) == LOW) {                     // нажатие оси обнуляет счёт
    position = 0;
    delay(200);
  }
}
```

23.3 Матричная клавиатура

Шестнадцать кнопок клавиатуры расположены на пересечениях четырёх строк и четырёх столбцов. Кнопка соединяет «свою» строку со «своим» столбцом.

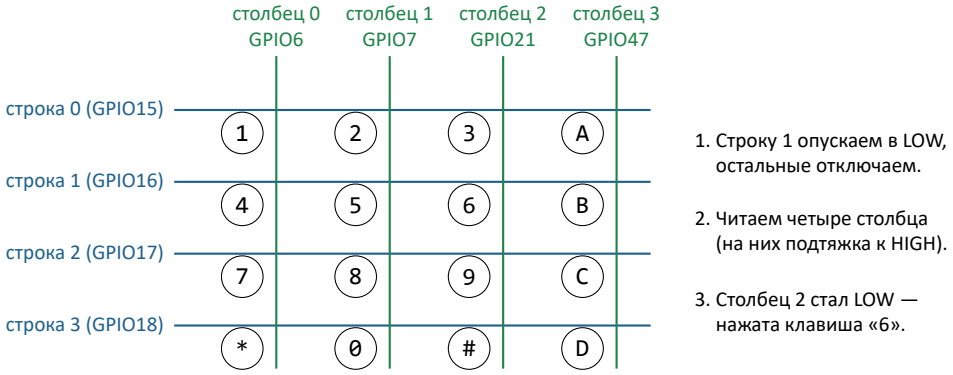


Рис. 23.3: Матрица 4×4: кнопка стоит на пересечении строки и столбца. Строки опрашиваются по очереди

Программа по очереди делает активной (LOW) одну строку и читает столбцы. Неактивные строки переводятся в режим входа, то есть как бы отключаются: иначе при двух нажатых кнопках два выхода с разными уровнями окажутся соединены накоротко.
Клавиатура подключается восемью проводами: четыре строки к **GPIO15**–**GPIO18**, четыре столбца к **GPIO6**, **GPIO7**, **GPIO21** и **GPIO47**. Резисторы не нужны: подтяжки столбцов включает программа.

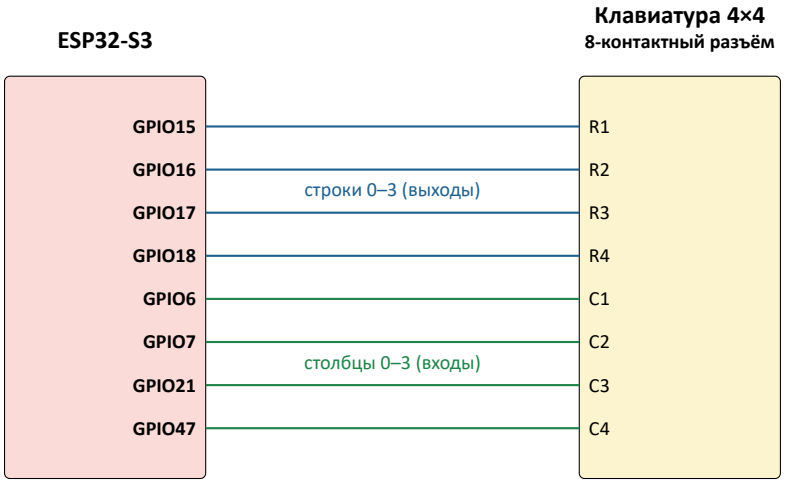


Рис. 23.4: Подключение клавиатуры 4×4. Порядок выводов на разъёме у разных моделей разный: найдите первую строку и первый столбец по прозвонке

Листинг 23.2: Опрос матричной клавиатуры 4×4

```
const int ROWS[4] = {15, 16, 17, 18};        // строки клавиатуры
const int COLS[4] = {6, 7, 21, 47};          // столбцы клавиатуры
const char KEYS[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}};

char scanKeypad() {                          // 0, если ничего не нажато
  char found = 0;
```

```
  for (int r = 0; r < 4; r++) {
    pinMode(ROWS[r], OUTPUT);               // активна только одна строка
    digitalWrite(ROWS[r], LOW);
    delayMicroseconds(50);                  // дать уровню установиться
    for (int c = 0; c < 4; c++) {
      if (digitalRead(COLS[c]) == LOW) found = KEYS[r][c];
    }
    pinMode(ROWS[r], INPUT);                // остальные строки «отключены»
  }
  return found;
}

void setup() {
  Serial.begin(115200);
  for (int c = 0; c < 4; c++) pinMode(COLS[c], INPUT_PULLUP);
  for (int r = 0; r < 4; r++) pinMode(ROWS[r], INPUT);
}

void loop() {
  static char last = 0;
  char key = scanKeypad();
  if (key && key != last) Serial.printf("Клавиша: %c\n", key);  // только новое нажатие
  last = key;
  delay(20);                                // опрос 50 раз в секунду гасит дребезг
}
```

23.4 Сенсорные входы

Выводы **GPIO1–GPIO14** умеют измерять электрическую ёмкость. Палец увеличивает ёмкость контакта, и число, которое возвращает `touchRead()`, растёт. Достаточно вставить в **GPIO14** отрезок провода или приклеить к нему кусочек фольги. Для индикации к **GPIO4** подключён светодиод через резистор 220 Ω, катодом к земле.

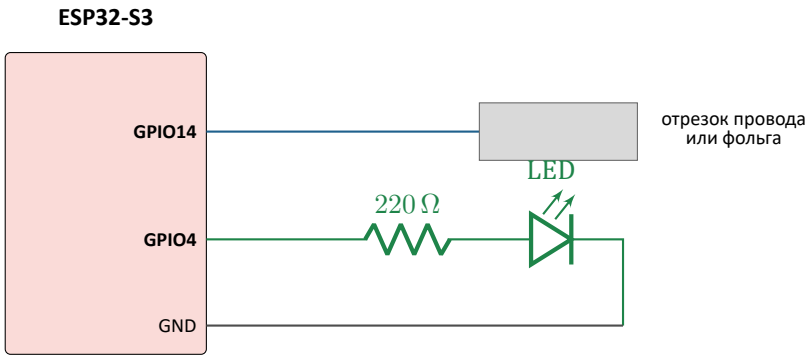


Рис. 23.5: Сенсорная кнопка: на **GPIO14** — только кусочек проводника, никаких деталей. Светодиод со своим резистором подключён к **GPIO4**

Листинг 23.3: Сенсорная кнопка с калибровкой при запуске

```
const int TOUCH_PIN = 14;               // сенсорные входы ESP32-S3: GPIO1..GPIO14
const int LED_PIN = 4;

uint32_t threshold = 0;

void setup() {
  Serial.begin(115200);
```

```
  pinMode(LED_PIN, OUTPUT);
  delay(500);

  uint32_t base = 0;                      // калибровка: не трогайте провод при запуске
  for (int i = 0; i < 16; i++) {
    base += touchRead(TOUCH_PIN);
    delay(10);
  }
  base /= 16;
  threshold = base + base / 10;           // порог на 10 % выше «спокойного» уровня
  Serial.printf("Базовый уровень: %lu, порог: %lu\n", base, threshold);
}

void loop() {
  uint32_t value = touchRead(TOUCH_PIN);  // на ESP32-S3 значение РАСТЁТ при касании
  digitalWrite(LED_PIN, value > threshold);
  Serial.printf("touch:%lu порог:%lu\n", value, threshold);
  delay(100);
}
```

Порог нельзя задать раз и навсегда: уровень зависит от длины провода, влажности и даже от того, питается ли плата от ноутбука или от батареи. Поэтому программа измеряет «спокойный» уровень при запуске и ставит порог чуть выше.

Осторожно

На первом ESP32 значение `touchRead()` при касании *уменьшается*, а на ESP32-S3 — *увеличивается*. Примеры из интернета для старого чипа нужно переписывать с обратным условием.

23.5 Самостоятельный опыт: меню на энкодере

Соедините энкодер с OLED-дисплеем из урока 6. На экране — три строки меню («Яркость», «Звук», «Выход»), поворот ручки перемещает указатель, нажатие на ось печатает выбранный пункт в Serial Monitor. Указатель не должен уходить за первую и последнюю строку.

Если не работает

- **Энкодер считает через раз или «скачет»** — дребезг контактов: увеличьте паузу в обработчике или поставьте конденсаторы 100 nF между CLK, DT и GND.
- **Клавиатура выдаёт не те символы** — шлейф подключён в обратном порядке; поменяйте порядок в массивах `ROWS` и `COLS`.
- **Сенсор срабатывает сам** — провод слишком длинный или лежит рядом с источником помех; укоротите его и увеличьте запас порога.

Итоги урока

- Энкодер сообщает направление: в момент спада A читаем уровень B.
- Матричная клавиатура опрашивается по строкам; неактивные строки переводим в режим входа.
- Сенсорный вход измеряет ёмкость; порог срабатывания калибруют при запуске.

Проверь себя

1. Как по двум сигналам энкодера определить направление вращения?
2. Почему 16 кнопкам матричной клавиатуры хватает восьми выводов?
3. Зачем сенсорной кнопке калибровка при запуске?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите пример с энкодером и выведите в Serial Monitor направление вращения.
2. ★ Определите нажатую клавишу матричной клавиатуры и выведите её в Serial Monitor.
3. ★★ Управляйте яркостью светодиода поворотом энкодера: один щелчок — 5%.
4. ★★ Выведите на экран вводимые цифры с клавиатуры, клавиша «#» подтверждает ввод.
5. ★★★ Сделайте кодовый замок: четыре цифры и «#»; верный код зажигает зелёный светодиод, неверный — красный.
6. ★★★★ Сделайте сенсорный «слайдер» из трёх контактов и определяйте направление движения пальца.

Урок 24. Звук: пищалка и мелодии

Цель	извлекать звук заданной высоты, понимать связь частоты и ноты и проигрывать мелодию, не останавливая программу
Опирается на	урок 4 (ШИМ и частота), урок 21 (отсчёт времени без `delay()`)
Результат	плата подаёт звуковой сигнал по кнопке и играет мелодию, оставаясь готовой к другим делам
Понадобится	пассивная пьезопищалка (пьезоизлучатель)

24.1 Сначала: звук — это быстрое мигание

Светодиод, мигающий 440 раз в секунду, глаз видит как ровный свет. Пищалка, на которую подали такой же сигнал, 440 раз в секунду толкает воздух — и ухо слышит ноту «ля». Чем чаще колебания, тем выше звук. Значит, для звука не нужно ничего нового: достаточно менять уровень на выводе с нужной частотой.

24.2 Частота и нота

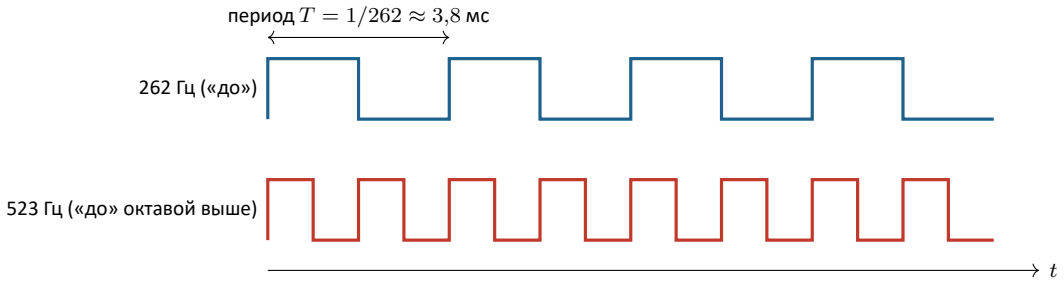


Рис. 24.1: Прямоугольный сигнал двух частот. Удвоение частоты повышает звук ровно на октаву

Соседние ноты отличаются по частоте в одно и то же число раз — примерно в 1,06. Двенадцать таких шагов дают удвоение частоты, то есть октаву. За точку отсчёта принята нота «ля» первой октавы: 440 Hz.

Нота	до	ре	ми	фа	соль	ля	си	до
Обозначение	C4	D4	E4	F4	G4	A4	B4	C5
Частота, Гц	262	294	330	349	392	440	494	523

24.3 Какая бывает пищалка

	Пассивная (пьезоизлучатель)	Активная (со встроенным генератором)
Что подавать	сигнал нужной частоты	просто HIGH
Высота звука	любая, задаёт программа	одна, задана на заводе
Как отличить	при подаче 3,3 В только щёлкает	при подаче 3,3 В пищит

Для мелодий нужна **пассивная** пищалка. Пьезоизлучатель потребляет очень малый ток, поэтому его подключают прямо между **GPIO21** и GND. Электромагнитную пищалку и тем более динамик так подключать нельзя: им нужен транзисторный ключ из урока 16.
Для опытов урока понадобится ещё кнопка из урока 3: она включена между **GPIO5** и GND, подтяжку к питанию включает программа.

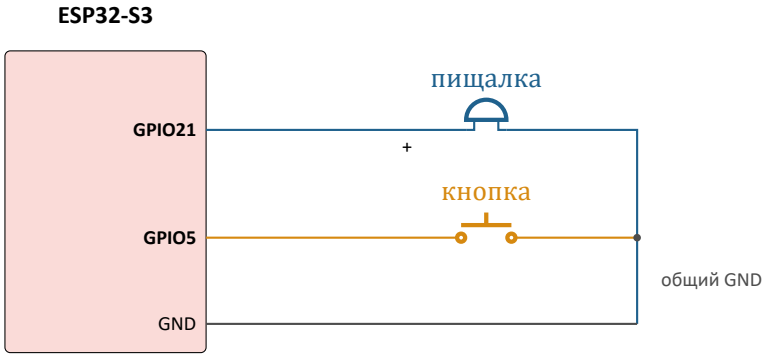


Рис. 24.2: Подключение пассивной пьезопищалки к **GPIO21** и кнопки к **GPIO5**. Резисторы не нужны: пищалка потребляет микроамперы, а подтяжку кнопки включает `INPUT_PULLUP`

24.4 Функция `tone()`

`tone(вывод, частота, длительность)` запускает звук и сразу возвращается: программа продолжает работать, а звук выключится сам. Без третьего аргумента звук длится, пока не вызвать `noTone(вывод)`.

Листинг 24.1: Звуковой сигнал при нажатии кнопки

```
const int BUZZER_PIN = 21;   // пассивная пьезопищалка между GPIO21 и GND
const int BTN_PIN = 5;

void setup() {
  pinMode(BTN_PIN, INPUT_PULLUP);
}

void loop() {
  static bool wasPressed = false;
  bool pressed = (digitalRead(BTN_PIN) == LOW);
  if (pressed && !wasPressed) {
    tone(BUZZER_PIN, 1000, 150);         // 1000 Гц на 150 мс: короткий «бип»
  }
  wasPressed = pressed;
  delay(20);
}
```

24.5 Мелодия без остановки программы

Простейший способ сыграть мелодию — цикл с `delay()` после каждой ноты. Но пока играет музыка, такая программа глуха к кнопкам. Правильный способ уже знаком по уроку 21: мелодия — это автомат, который помнит номер текущей ноты и время её начала.

Листинг 24.2: Проигрыватель мелодии на `millis()`

```
const int BUZZER_PIN = 21;

struct Note {
  uint16_t freq;                         // частота в герцах; 0 -- пауза
  uint16_t ms;                           // длительность
};

const Note MELODY[] = {
  {262, 300}, {294, 300}, {330, 300}, {349, 300}, {392, 600}, {0, 200},
  {392, 300}, {349, 300}, {330, 300}, {294, 300}, {262, 600}, {0, 800}};
```

```
const int COUNT = sizeof(MELODY) / sizeof(MELODY[0]);
const uint16_t GAP_MS = 30;               // короткая тишина между нотами

int noteIndex = 0;
uint32_t noteStart = 0;
bool sounding = false;

void playMelody() {                       // вызывается из loop() и сразу возвращается
  const Note& n = MELODY[noteIndex];
  uint32_t t = millis() - noteStart;
  if (!sounding) {                        // начало ноты
    if (n.freq) tone(BUZZER_PIN, n.freq); else noTone(BUZZER_PIN);
    sounding = true;
    noteStart = millis();
  } else if (t >= n.ms) {                 // нота закончилась -- переходим к следующей
    noTone(BUZZER_PIN);
    if (t >= (uint32_t)n.ms + GAP_MS) {
      noteIndex = (noteIndex + 1) % COUNT;
      sounding = false;
    }
  }
}

void setup() {}

void loop() {
  playMelody();
  // здесь программа свободна: можно читать кнопки и датчики
}
```

Мелодия записана таблицей «частота — длительность», нулевая частота означает паузу. Между нотами программа делает тишину в 30 ms: без неё две одинаковые ноты подряд сольются в одну длинную.

Совет

Громкость пьезопищалки заметно зависит от частоты: у большинства она максимальна в районе 2 – 4 kHz. Для сигналов тревоги выбирайте частоты из этого диапазона, а для мелодий мириться с тихим звуком помогает корпус-резонатор — положите пищалку на пустую коробку.

24.6 Что дальше: настоящий звук

Пищалка выдаёт только прямоугольный сигнал. Чтобы воспроизводить речь и музыку, у ESP32-S3 есть интерфейс I^2S: по трём проводам он передаёт цифровой звук на усилитель (например, MAX98357A) или принимает его с цифрового микрофона (INMP441). Это тема для итогового проекта.

24.7 Самостоятельный опыт: музыкальная клавиатура

Подключите матричную клавиатуру из урока 23. Клавиши «1»–«8» играют ноты от C4 до C5, пока клавиша нажата; при отпускании звук прекращается.

24.8 То же на ESPUI: ноты в списке на телефоне

Вместо клавиатуры можно взять телефон: список нот и кнопка «Играть». Это хороший способ проверить таблицу частот урока на слух и увидеть, как выбор из списка превращается в частоту.

Листинг 24.3: ESPUI: Select ноты и Button «Играть»

```
// Пассивная пищалка между GPIO21 и GND (урок 24). Выбираем ноту и нажимаем «Играть».
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

const int BUZZER_PIN = 21;
const uint16_t NOTES[] = {262, 294, 330, 349, 392, 440, 494, 523};   // до ... до
const uint32_t BEEP_MS = 400;

std::atomic<int> noteIndex{0};
std::atomic<bool> playRequest{false};
uint16_t freqId;

void noteCallback(Control* sender, int event) {
  if (event != S_VALUE) return;
  int i = sender->value.toInt();          // в value хранится номер ноты как строка
  if (i >= 0 && i < 8) noteIndex = i;
}

void playCallback(Control*, int event) {
  if (event == B_UP) playRequest = true;  // один раз на щелчок
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Notes", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  auto select = ESPUI.addControl(ControlType::Select, "Нота", "0", ControlColor::Sunflower,
↪  Control::noParent, noteCallback);
  const char* names[] = {"до (C4)", "ре (D4)", "ми (E4)", "фа (F4)", "соль (G4)", "ля (A4)", "си (B4)",
↪  "до (C5)"};
  for (int i = 0; i < 8; i++)
    ESPUI.addControl(ControlType::Option, names[i], String(i), ControlColor::None, select);
  ESPUI.button("Звук", playCallback, ControlColor::Emerald, "Играть");
  freqId = ESPUI.label("Частота, Гц", ControlColor::Peterriver, String(NOTES[0]));
  ESPUI.begin("ESP32-S3 notes");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static uint32_t stopAt = 0;
  if (playRequest.exchange(false)) {
    uint16_t f = NOTES[noteIndex];
    tone(BUZZER_PIN, f);
    stopAt = millis() + BEEP_MS;          // выключим позже, не блокируя программу
    ESPUI.updateLabel(freqId, String(f));
  }
  if (stopAt && (int32_t)(millis() - stopAt) >= 0) {
    noTone(BUZZER_PIN);
    stopAt = 0;
  }
  delay(2);
}
```

Сеть называется `S3-Notes` (пароль `lesson-39-demo`), пищалка по-прежнему между GPIO21 и GND. Звук включается `tone()`, а выключается позже, когда `loop()` видит, что время прошло: программа ни разу не останавливается на `delay()`.

Если не работает

- **Вместо ноты слышны щелчки** — пищалка активная: ей нужен постоянный уровень, а не частота.
- **Звук очень тихий** — частота далека от резонанса пищалки; попробуйте 2 – 4 kHz.
- **Перестал работать ШИМ на другом выводе** — `tone()` использует ту же периферию LEDC; задавайте звук и яркость на разных выводах и не вызывайте для вывода пищалки `ledcAttach()`.

Итоги урока

- Высота звука — это частота сигнала; удвоение частоты даёт октаву.
- Для мелодий нужна пассивная пищалка и функции `tone()` и `noTone()`.
- Мелодию играет автомат на `millis()`, а не цикл с `delay()`.

Проверь себя

1. Чем пассивная пищалка отличается от активной и как их различить?
2. Во сколько раз отличаются частоты одной и той же ноты в соседних октавах?
3. Почему мелодию не стоит играть циклом с `delay()`?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Сыграйте ноту ля (440 Hz) и измерьте её длительность по секундомеру.
2. ★ Составьте таблицу частот для семи нот одной октавы и проиграйте гамму.
3. ★★ Запишите в таблицу свою мелодию из 10–15 нот и проиграйте её.
4. ★★ Проиграйте мелодию без остановки программы: кнопка должна реагировать во время звучания.
5. ★★★ Сделайте сигнализацию: при нажатии кнопки звучит сирена — частота плавно растёт от 800 до 1800 Hz и падает обратно.
6. ★★★★ Сделайте метроном: потенциометр задаёт темп от 40 до 200 ударов в минуту, каждый четвёртый удар выше остальных.

Урок 25. Шаговый двигатель

Цель	понять принцип шагового двигателя, подключить его через драйвер и поворачивать вал на заданный угол
Опирается на	урок 16 (нагрузка через драйвер, отдельное питание), урок 21 (шаги по времени без `delay()`), урок 22 (биты)
Результат	вал двигателя поворачивается на угол, введённый в Serial Monitor, и останавливается точно в нужном месте
Понадобится	шаговый двигатель 28BYJ-48 с платой драйвера ULN2003, источник 5 V

25.1 Сначала: мотор, который умеет считать

Обычный мотор из урока 16 просто крутится, и программа не знает, на сколько он повернулся. Сервопривод из урока 4 знает свой угол, но поворачивается только на пол-оборота. Шаговый двигатель сочетает оба свойства: он вращается сколько угодно, но делает это маленькими одинаковыми шагами. Сосчитав шаги, программа точно знает положение вала. Так работают 3D-принтеры, станки и механизмы камер.

	Коллекторный мотор	Сервопривод	Шаговый двигатель
Движение	непрерывное вращение	поворот на угол 0–180°	вращение шагами
Что задаёт программа	скорость и направление	угол	число шагов и их темп
Знает ли положение	нет	да (внутренний датчик)	да, если не пропускал шаги
Где применяют	колёса, вентиляторы	рули, захваты	принтеры, станки, поворотные столы

25.2 Как он устроен

Внутри двигателя — несколько обмоток (электромагнитов) вокруг ротора с магнитом. Если включать обмотки по очереди, ротор каждый раз поворачивается к ближайшей включённой обмотке. Один такой поворот — шаг. Порядок включения определяет направление, темп — скорость.

У двигателя 28BYJ-48 четыре обмотки. Если включать то одну обмотку, то две соседние, шаг получается вдвое мельче. Этот режим называется полушаговым, и в нём восемь фаз.

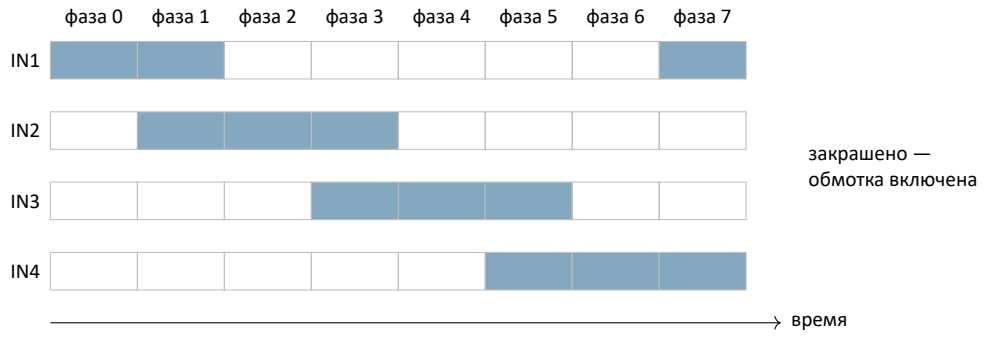


Рис. 25.1: Полушаговый режим: восемь фаз. Включена то одна обмотка, то две соседние; после фазы 7 снова идёт фаза 0

Между валом и ротором 28BYJ-48 стоит редуктор примерно 1:64, поэтому на один оборот вала

приходится около 4096 полушагов — меньше десятой доли градуса на шаг. Двигатель получается медленным (до 10–15 оборотов в минуту), но точным.

25.3 Подключение

Обмотка потребляет сотни миллиампер, поэтому между платой и двигателем стоит драйвер ULN2003 — семь транзисторных ключей с защитными диодами в одном корпусе (тот же ключ, что в уроке 16).

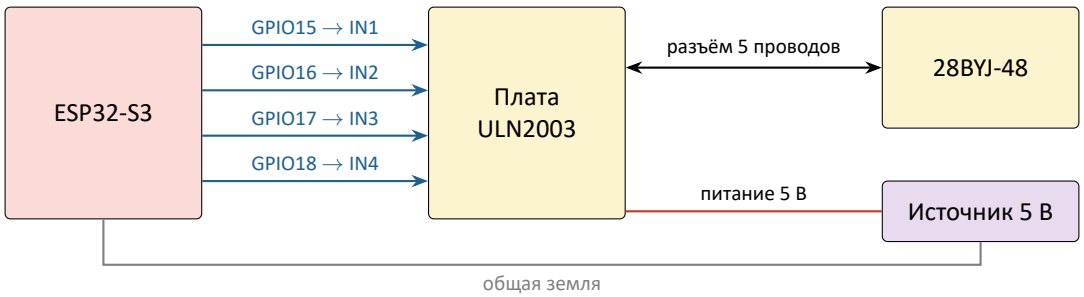


Рис. 25.2: Подключение шагового двигателя. Сигналы идут от платы, ток обмоток — от отдельного источника

Осторожно

Не питайте двигатель от вывода 3V3. От вывода 5V платы один 28BYJ-48 обычно работает, но при слабом USB-порте плата может перезагружаться. Надёжнее отдельный источник 5 V с общей землёй.

25.4 Программа

Восемь фаз удобно хранить как восемь чисел: бит 0 отвечает за IN1, бит 1 — за IN2 и так далее. Это в точности таблица с рисунка, записанная по столбцам.

Листинг 25.1: Поворот вала на заданный угол без `delay()`

```
const int PINS[4] = {15, 16, 17, 18};      // входы IN1..IN4 платы ULN2003

// Полушаговый режим: 8 фаз. Бит 0 -> IN1, бит 1 -> IN2, бит 2 -> IN3, бит 3 -> IN4
const uint8_t PHASES[8] = {0b0001, 0b0011, 0b0010, 0b0110, 0b0100, 0b1100, 0b1000, 0b1001};

const long STEPS_PER_REV = 4096;           // полушагов на оборот вала 28BYJ-48 (примерно)
const uint32_t STEP_US = 1200;             // пауза между шагами: меньше -- быстрее, но слабее

long current = 0;                          // где находимся, в шагах
long target = 0;                           // куда нужно прийти
uint32_t lastStep = 0;

void applyPhase(int phase) {
  for (int i = 0; i < 4; i++) digitalWrite(PINS[i], (PHASES[phase] >> i) & 1);
}

void release() {                           // снять ток с обмоток: мотор не греется
  for (int i = 0; i < 4; i++) digitalWrite(PINS[i], LOW);
}

void stepperRun() {                        // один шаг за вызов, без delay()
  if (current == target) return;
```

```
  if (micros() - lastStep < STEP_US) return;
  lastStep = micros();
  current += (target > current) ? 1 : -1;
  applyPhase(((current % 8) + 8) % 8);
  if (current == target) release();
}

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(50);
  for (int i = 0; i < 4; i++) pinMode(PINS[i], OUTPUT);
  Serial.println("Введите угол в градусах, например 90 или -180");
}

void loop() {
  stepperRun();
  if (Serial.available()) {
    long degrees = Serial.parseInt();
    while (Serial.available()) Serial.read();          // убрать остаток строки
    target = current + degrees * STEPS_PER_REV / 360;
    Serial.printf("Поворот на %ld градусов\n", degrees);
  }
}
```

Функция `stepperRun()` устроена как автомат из урока 21: за один вызов она делает не больше одного шага и сразу возвращается. Программа хранит текущее положение `current` и цель `target`; пока они различаются, двигатель шагает. После прихода в цель ток с обмоток снимается, чтобы двигатель не грелся.

Совет

Если шаги идут слишком часто, ротор не успевает за магнитным полем: двигатель жужжит, но не вращается, а программа продолжает считать шаги и теряет положение. Уменьшайте `STEP_US` постепенно и найдите значение, при котором вал ещё уверенно вращается под нагрузкой.

25.5 Самостоятельный опыт: стрелочный указатель

Наденьте на вал бумажную стрелку и нарисуйте шкалу. Потенциометр (урок 5) задаёт положение стрелки от 0° до 270°. Двигатель не знает, где он находится при включении, поэтому при запуске поставьте стрелку на ноль вручную или добавьте кнопку «установить ноль».

25.6 То же на ESPUI: поворот на заданный угол с телефона

Вместо ввода угла в Serial Monitor сделаем пульт: поле «Шаг поворота» в градусах и три кнопки, «вправо», «влево» и «в начало». Нажатия накапливаются, а шаги выполняются без `delay()`, как в основной программе урока.

Листинг 25.2: ESPUI: Number шага и кнопки поворота

```
// ULN2003: IN1..IN4 -> GPIO15..GPIO18 (урок 25). Панель задаёт угол в градусах.
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

const int PINS[4] = {15, 16, 17, 18};
const uint8_t PHASES[8] = {0b0001, 0b0011, 0b0010, 0b0110, 0b0100, 0b1100, 0b1000, 0b1001};
const long STEPS_PER_REV = 4096;
const uint32_t STEP_US = 1200;

std::atomic<long> requestDeg{0};          // накопленная просьба: «повернись ещё на столько»
std::atomic<int> stepSize{90};
long current = 0, target = 0;
uint32_t lastStep = 0;
uint16_t angleId;

void applyPhase(int phase) {
  for (int i = 0; i < 4; i++) digitalWrite(PINS[i], (PHASES[phase] >> i) & 1);
}
void release() {
  for (int i = 0; i < 4; i++) digitalWrite(PINS[i], LOW);
}
void stepperRun() {
  if (current == target) return;
  if (micros() - lastStep < STEP_US) return;
  lastStep = micros();
  current += (target > current) ? 1 : -1;
  applyPhase(((current % 8) + 8) % 8);
  if (current == target) release();
}

void stepCallback(Control* sender, int event) {
  if (event == N_VALUE) stepSize = constrain(sender->value.toInt(), 1, 360);
}
void plusCallback(Control*, int event)  { if (event == B_UP) requestDeg += stepSize; }
void minusCallback(Control*, int event) { if (event == B_UP) requestDeg -= stepSize; }
void zeroCallback(Control*, int event)  { if (event == B_UP) requestDeg += -(current * 360 /
↪  STEPS_PER_REV); }

void setup() {
  Serial.begin(115200);
  for (int i = 0; i < 4; i++) pinMode(PINS[i], OUTPUT);
  release();
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Stepper", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  ESPUI.number("Шаг поворота, °", stepCallback, ControlColor::Peterriver, 90, 1, 360);
  ESPUI.button("Вправо", plusCallback, ControlColor::Emerald, "+ шаг");
  ESPUI.button("Влево", minusCallback, ControlColor::Carrot, "− шаг");
  ESPUI.button("К нулю", zeroCallback, ControlColor::Alizarin, "В начало");
  angleId = ESPUI.label("Текущий угол, °", ControlColor::Turquoise, "0");
  ESPUI.begin("ESP32-S3 stepper");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  stepperRun();
  long deg = requestDeg.exchange(0);      // забираем накопленные просьбы одним действием
  if (deg != 0) target += deg * STEPS_PER_REV / 360;

  static uint32_t last = 0;
```

```
  if (millis() - last >= 300) {
    last = millis();
    ESPUI.updateLabel(angleId, String(current * 360 / STEPS_PER_REV));
  }
}
```

Подключите ULN2003 к GPIO15–GPIO18, как в этом уроке, и подсоедините телефон к сети `S3-Stepper` (пароль `lesson-39-demo`). Подпись «Текущий угол» показывает, где по мнению платы стоит вал; при включении это всегда 0°, а настоящее положение вала плата не знает.

Если не работает

- **Двигатель жужжит и дрожит на месте** — перепутан порядок проводов IN1–IN4 или шаги слишком частые.
- **Вал вращается не в ту сторону** — поменяйте порядок выводов в массиве `PINS` на обратный.
- **Двигатель сильно греется** — обмотки остаются под током на стоянке; вызывайте `release()`.

Итоги урока

- Шаговый двигатель поворачивается одинаковыми шагами; положение — это счётчик шагов.
- Порядок включения обмоток задаёт направление, пауза между шагами — скорость.
- Двигатель подключают через драйвер и питают отдельно; на стоянке ток с обмоток снимают.

Проверь себя

1. Чем шаговый двигатель отличается от сервопривода и от обычного мотора?
2. Сколько полушагов нужно сделать 28BYJ-48, чтобы повернуть вал на 45°?
3. Что произойдёт, если подавать шаги слишком часто, и почему это опасно для программы?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Сделайте один оборот в одну сторону и один в обратную. Сколько шагов в обороте вашего мотора?
2. ★ Запишите по датасету мотора ток обмотки и сверьте с током драйвера.
3. ★★ Добавьте команду `speed N`, которая меняет паузу между шагами.
4. ★★ Добавьте команду `angle N`: повернуться на N градусов.
5. ★★★ Сделайте плавный разгон и торможение: первые и последние 200 шагов идут с увеличенной паузой.
6. ★★★★ Добавьте концевой выключатель: при включении мотор медленно вращается до него и принимает это положение за ноль.

Урок 26. Акселерометр и гироскоп: датчик движения

Цель	читать датчик напрямую через его регистры, измерять наклон и убирать шум и дрейф фильтром
Опирается на	урок 6 (шина I²C), урок 22 (регистры, старший и младший байты), урок 5 (шум и усреднение)
Результат	плата измеряет угол наклона и рисует в Serial Plotter «сырой» и отфильтрованный графики
Понадобится	модуль MPU6050 (плата GY-521)

26.1 Сначала: два датчика, которые ошибаются по-разному

В модуле MPU6050 два датчика. Акселерометр чувствует ускорение, в том числе силу тяжести, поэтому по нему видно, куда направлен «низ». Гироскоп чувствует скорость поворота. Каждый из них по отдельности плох: акселерометр дрожит от любой вибрации, а угол, вычисленный по гироскопу, постепенно «уплывает». Но ошибки у них разные, и если правильно смешать показания, получится точный и спокойный угол. Тот же приём стоит в каждом телефоне, квадрокоптере и самобалансирующемся роботе.

26.2 Что измеряет датчик



Рис. 26.1: Оси датчика и измерение наклона. Сила тяжести раскладывается по осям Y и Z, и по их отношению находят угол

	Акселерометр	Гироскоп
Что измеряет	ускорение по трём осям (в долях g)	скорость поворота вокруг трёх осей (°/с)
Как получить угол	сразу, по направлению силы тяжести	накапливая: угол += скорость × время
Сильная сторона	не накапливает ошибку	не чувствует вибрацию, быстрый
Слабая сторона	шумит и «врёт» при рывках	угол постепенно уплывает (дрейф)

26.3 Разговор с датчиком через регистры

В уроке 6 всю работу с дисплеем делала библиотека. Здесь обойдёмся без неё, чтобы увидеть, что происходит на самом деле. Датчик — это набор регистров (урок 22), а шина I²C позволяет записать число в регистр или прочитать несколько регистров подряд.

Адрес регистра	Название	Назначение
`0x75`	WHO_AM_I	«паспорт»: исправный датчик отвечает `0x68`
`0x6B`	PWR_MGMT_1	питание: запись 0 выводит датчик из сна
`0x3B–0x40`	ACCEL_X, Y, Z	ускорение: по два байта на ось, старший первым
`0x43–0x48`	GYRO_X, Y, Z	скорость поворота: по два байта на ось

Каждое измерение — число со знаком из 16 бит, которое приходит двумя байтами. Собрать его обратно помогает сдвиг: `(старший << 8) | младший`. В диапазоне по умолчанию 16384 единицы акселерометра соответствуют $1g$, а 131 единица гироскопа — одному градусу в секунду. Модуль GY-521 подключается как любое устройство I²C: VCC к 3V3, GND к земле, SDA к **GPIO8**, SCL к **GPIO9**. Его адрес — `0x68`; проверьте это сканером из урока 6.

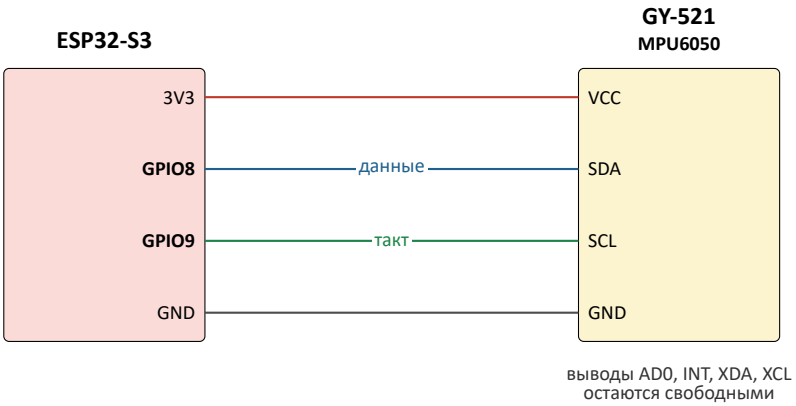


Рис. 26.2: Подключение GY-521 по I²C. Подтяжки шины и стабилизатор уже есть на модуле, поэтому внешних деталей не нужно. При свободном AD0 адрес равен `0x68`

26.4 Комплементарный фильтр

Идея фильтра умещается в одну строку. Новый угол на 98 % берётся из гироскопа (прошлый угол плюс поворот за прошедшее время) и на 2 % — из акселерометра:

$$\varphi = 0{,}98 \cdot (\varphi + \omega\,\Delta t) + 0{,}02 \cdot \varphi_{\text{акс}}.$$

Гироскоп отвечает за быстрые изменения, а маленькая постоянная добавка от акселерометра не даёт углу уплыть.

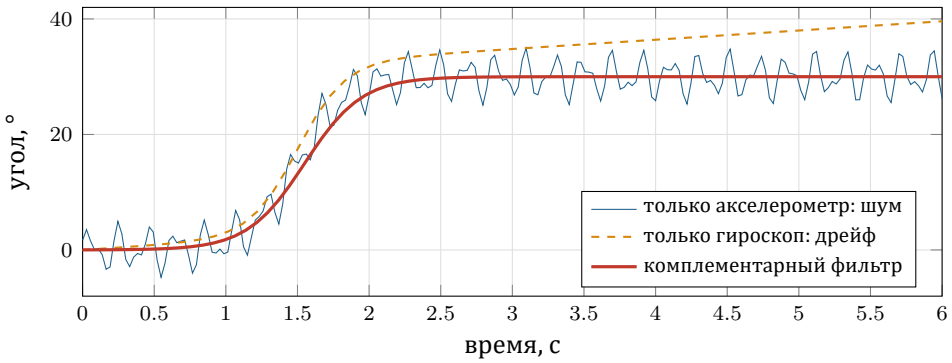


Рис. 26.3: Плату наклонили на 30° (иллюстрация). Акселерометр шумит, угол по гироскопу уплывает, фильтр берёт лучшее от обоих

Листинг 26.1: Чтение MPU6050 через регистры и комплементарный фильтр

```
#include <Wire.h>

const uint8_t MPU_ADDR = 0x68;
const uint8_t REG_PWR_MGMT_1 = 0x6B;      // управление питанием
const uint8_t REG_ACCEL_XOUT = 0x3B;      // отсюда подряд идут 14 байт измерений
const uint8_t REG_WHO_AM_I = 0x75;        // «паспорт» микросхемы

float angle = 0;                          // отфильтрованный угол наклона, градусы
uint32_t lastUs = 0;

void writeReg(uint8_t reg, uint8_t value) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(reg);
  Wire.write(value);
  Wire.endTransmission();
}

bool readRegs(uint8_t reg, uint8_t* buf, size_t n) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(reg);                                    // с какого регистра читать
  if (Wire.endTransmission(false) != 0) return false; // датчик не ответил
  if (Wire.requestFrom(MPU_ADDR, n) != n) return false;
  for (size_t i = 0; i < n; i++) buf[i] = Wire.read();
  return true;
}

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);
  uint8_t id = 0;
  if (!readRegs(REG_WHO_AM_I, &id, 1)) {
    Serial.println("MPU6050 не отвечает: проверьте провода и адрес");
  } else {
    Serial.printf("WHO_AM_I = 0x%02X\n", id);
  }
  writeReg(REG_PWR_MGMT_1, 0);            // после включения датчик спит -- будим
  lastUs = micros();
}

void loop() {
  uint8_t b[14];
  if (!readRegs(REG_ACCEL_XOUT, b, 14)) { delay(500); return; }

  int16_t ay = (int16_t)((b[2] << 8) | b[3]);         // два байта -> число со знаком
  int16_t az = (int16_t)((b[4] << 8) | b[5]);
  int16_t gx = (int16_t)((b[8] << 8) | b[9]);

  uint32_t now = micros();
  float dt = (now - lastUs) / 1000000.0;              // секунд с прошлого замера
  lastUs = now;

  float accAngle = atan2(ay, az) * 57.2958;           // наклон по акселерометру
  float gyroRate = gx / 131.0;                        // скорость поворота, град/с
  angle = 0.98 * (angle + gyroRate * dt) + 0.02 * accAngle;   // комплементарный фильтр

  Serial.printf("acc:%.1f angle:%.1f\n", accAngle, angle);    // формат для Serial Plotter
```

```
  delay(20);
}
```

Откройте Tools → Serial Plotter, наклоняйте плату и постучите по столу: линия `acc` будет вздрагивать, а `angle` останется спокойной.

Совет

Коэффициенты 0,98 и 0,02 — не закон. Чем больше доля гироскопа, тем спокойнее, но медленнее исправляется дрейф. Попробуйте 0,90 и 0,995 и сравните графики.

26.5 Самостоятельный опыт: электронный уровень

Соедините датчик с OLED-дисплеем (оба устройства на одной шине I^2C). Покажите угол крупными цифрами и нарисуйте «пузырёк», который смещается при наклоне. Если угол меньше 1°, зажигайте зелёный светодиод.

26.6 То же на ESPUI: график наклона

График комплементарного фильтра в этом уроке был иллюстрацией; теперь можно получить настоящий. Плата считает угол комплементарным фильтром, а телефон показывает число по акселерометру, число после фильтра и ленту последних значений. Покачайте плату вокруг оси X: график плавный, а «сырое» число заметно дрожит.

Листинг 26.2: ESPUI: угол по акселерометру и после фильтра

```
// MPU6050 (GY-521): SDA -> GPIO8, SCL -> GPIO9, адрес 0x68 (урок 26). График наклона на телефоне.
#include <Wire.h>
#include <WiFi.h>
#include <ESPUI.h>

const uint8_t MPU_ADDR = 0x68;
float angle = 0;
uint32_t lastUs = 0;
bool sensorOk = false;
uint16_t graphId, accId, filtId, stateId;

void writeReg(uint8_t reg, uint8_t value) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(reg);
  Wire.write(value);
  Wire.endTransmission();
}

bool readRegs(uint8_t reg, uint8_t* buf, size_t n) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(reg);
  if (Wire.endTransmission(false) != 0) return false;
  if (Wire.requestFrom(MPU_ADDR, n) != n) return false;
  for (size_t i = 0; i < n; i++) buf[i] = Wire.read();
  return true;
}

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);
  writeReg(0x6B, 0);                      // разбудить датчик
  uint8_t id = 0;
  sensorOk = readRegs(0x75, &id, 1);      // WHO_AM_I: ответ вообще есть?
  lastUs = micros();
```

```
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Tilt", "lesson-39-demo")) { Serial.println("AP failed"); return; }
  stateId = ESPUI.label("Датчик", ControlColor::Peterriver, sensorOk ? "отвечает" : "НЕ ОТВЕЧАЕТ");
  accId  = ESPUI.label("Угол по акселерометру, °", ControlColor::Alizarin, "0");
  filtId = ESPUI.label("Угол после фильтра, °", ControlColor::Emerald, "0");
  graphId = ESPUI.graph("Фильтрованный угол", ControlColor::Turquoise);
  ESPUI.begin("ESP32-S3 tilt");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static uint32_t lastShow = 0;
  uint8_t b[14];
  if (!readRegs(0x3B, b, 14)) {           // нет ответа: честно говорим об этом
    ESPUI.updateLabel(stateId, "НЕ ОТВЕЧАЕТ");
    delay(500);
    return;
  }
  int16_t ay = (int16_t)((b[2] << 8) | b[3]);
  int16_t az = (int16_t)((b[4] << 8) | b[5]);
  int16_t gx = (int16_t)((b[8] << 8) | b[9]);

  uint32_t now = micros();
  float dt = (now - lastUs) / 1000000.0f;
  lastUs = now;
  float accAngle = atan2(ay, az) * 57.2958f;
  angle = 0.98f * (angle + (gx / 131.0f) * dt) + 0.02f * accAngle;

  if (millis() - lastShow >= 250) {       // фильтр считаем часто, показываем редко
    lastShow = millis();
    ESPUI.updateLabel(stateId, "отвечает");
    ESPUI.updateLabel(accId, String(accAngle, 1));
    ESPUI.updateLabel(filtId, String(angle, 1));
    ESPUI.addGraphPoint(graphId, (int)angle);
  }
  delay(10);
}
```

Датчик подключён так же, как раньше: SDA к GPIO8, SCL к GPIO9. Сеть называется `S3-Tilt` (пароль `lesson-39-demo`). Если датчик не отвечает, на странице появляется «НЕ ОТВЕЧАЕТ», а программа не выдумывает нулевые углы.

Если не работает

- **Датчик не отвечает** — проверьте адрес сканером: при выводе AD0, подключённом к питанию, адрес равен `0x69`.
- **Все значения равны нулю** — датчик спит: не записан 0 в регистр `PWR_MGMT_1`.
- **Угол на ровном столе не равен нулю** — у каждого экземпляра свой сдвиг; измерьте его при запуске и вычитайте (калибровка нуля).

Итоги урока

- Любой датчик I^2C — это регистры: в одни пишут настройки, из других читают результат.
- Акселерометр даёт наклон по силе тяжести, гироскоп — скорость поворота.
- Комплементарный фильтр смешивает оба показания и убирает и шум, и дрейф.

Проверь себя

1. Как из двух прочитанных байтов получить 16-битное число со знаком?
2. Почему по одному гироскопу нельзя долго измерять угол?
3. Что изменится, если в фильтре заменить коэффициенты 0,98 и 0,02 на 0,5 и 0,5?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выведите в Serial Plotter ускорения по трём осям и найдите, какая ось куда направлена.
2. ★ Положите плату на стол и проверьте, что суммарное ускорение близко к $1g$.
3. ★★ Добавьте калибровку: первые две секунды плата неподвижна, программа измеряет сдвиг гироскопа и вычитает его.
4. ★★ Рассчитайте угол наклона только по акселерометру и выведите его в градусах.
5. ★★★ Сделайте детектор встряхивания: если полное ускорение превышает $2g$, плата подаёт звуковой сигнал.
6. ★★★★ Сравните угол по акселерометру, по гироскопу и по комплементарному фильтру на графике при быстром покачивании платы.

Урок 27. Автономное питание: батареи и аккумуляторы

Цель	выбрать источник питания для устройства без USB, безопасно обращаться с литиевым аккумулятором и измерять его заряд
Опирается на	урок 5 (делитель напряжения и АЦП), урок 9 (сон и расчёт времени работы), урок 16 (раздельное питание)
Результат	устройство работает без компьютера, показывает напряжение батареи и засыпает при разряде
Понадобится	держатель на три батарейки AA или powerbank; для опытов с Li-ion — аккумулятор с платой защиты и модуль зарядки TP4056

27.1 Сначала: откуда брать энергию

Пока плата подключена к компьютеру, о питании можно не думать. Но готовое устройство — метеостанция на балконе, робот, ночник — должно работать само. Для этого нужно ответить на три вопроса: какой источник выбрать, как получить из его напряжения нужные плате $5\,\text{V}$ или $3{,}3\,\text{V}$ и как узнать, что он скоро разрядится.

27.2 Источники питания

Источник	Напряжение	Плюсы	Минусы
Powerbank	5 В (USB)	безопасен, готовая зарядка	может отключаться при малом токе спящей платы
3 × AA (щелочные)	4,5 В → 3,3 В	просты, безопасны	не перезаряжаются; напряжение падает по мере разряда
4 × AA (NiMH)	4,8 В	перезаряжаются, безопасны	нужно отдельное зарядное устройство
Li-ion / Li-Po	4,2 В → 3,0 В	большая ёмкость при малом размере	требует защиты и аккуратности

Для первых автономных опытов возьмите powerbank или три батарейки AA, подключённые к выводу 5V и GND: на плате есть стабилизатор, который сделает из них $3{,}3\,\text{V}$.

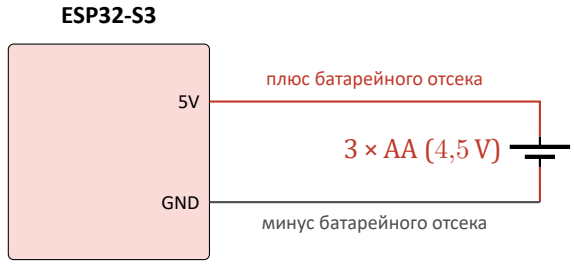


Рис. 27.1: Питание платы от трёх батареек AA через вывод 5V. Во время работы от батареек USB-кабель лучше отключить

Осторожно

Литиевый аккумулятор при коротком замыкании, перезаряде или повреждении корпуса может загореться. Используйте только аккумуляторы с платой защиты, заряжайте их только специальным модулем и только под присмотром, не носите в кармане без чехла, не паяйте провода прямо к корпусу. Вздувшийся или помятый аккумулятор использовать нельзя.

В учебных опытах Li-ion применяется только с разрешения педагога.

27.3 Цепочка питания

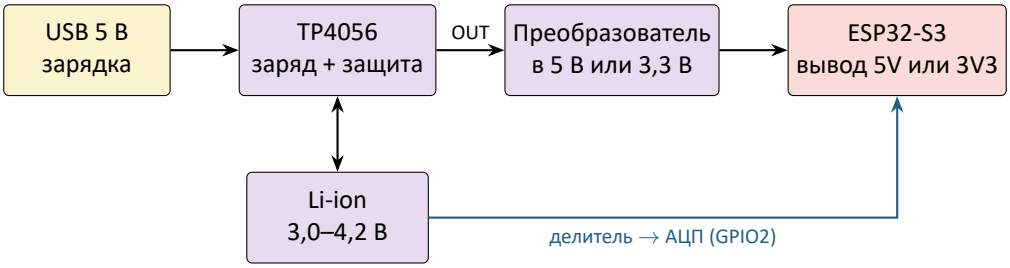


Рис. 27.2: Питание от литиевого аккумулятора. Модуль TP4056 заряжает аккумулятор и защищает его от перезаряда, глубокого разряда и короткого замыкания; преобразователь даёт плате стабильное напряжение

Напряжение аккумулятора меняется от $4{,}2\,\text{V}$ до $3{,}0\,\text{V}$, а плате нужно ровно $3{,}3\,\text{V}$. Есть два пути. Повышающий преобразователь делает из напряжения аккумулятора $5\,\text{V}$, которые подают на вывод 5V платы. Либо стабилизатор с малым падением напряжения выдаёт $3{,}3\,\text{V}$ прямо на вывод 3V3; стабилизатор на самой отладочной плате для этого обычно не подходит: ему нужен запас около вольта.

Совет

Отладочная плата не рассчитана на экономию: её светодиод питания и микросхема USB потребляют несколько миллиампер даже тогда, когда чип спит. Для устройства, которое должно работать месяцами, берут плату без лишних деталей или отдельный модуль ESP32-S3.

27.4 Сколько осталось заряда

Напряжение литиевого аккумулятора падает по мере разряда, поэтому заряд можно оценить вольтметром из урока 5. Напряжение до $4{,}2\,\text{V}$ выше предела АЦП, и его делят пополам двумя резисторами по $100\,\text{k}\Omega$: большие номиналы почти не разряжают аккумулятор.

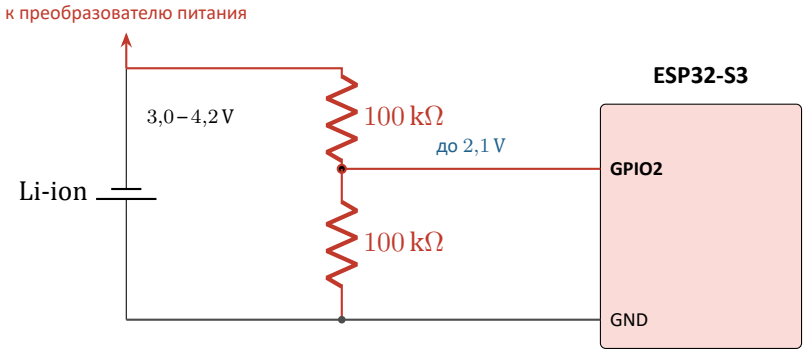


Рис. 27.3: Делитель напряжения для измерения заряда: два резистора по $100\,\text{k}\Omega$ уменьшают напряжение вдвое, и на **GPIO2** попадает не более $2{,}1\,\text{V}$. «Минус» аккумулятора обязательно соединяют с GND платы

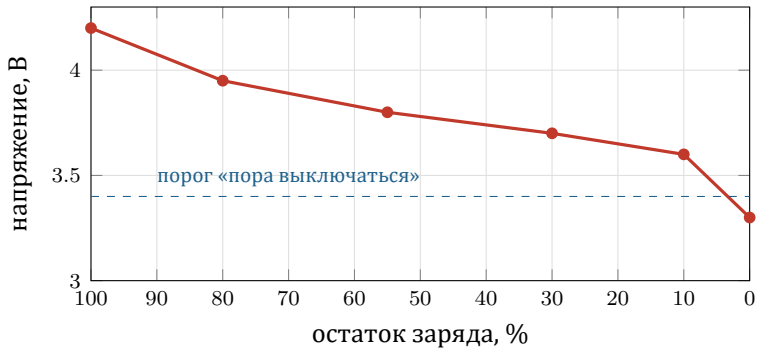


Рис. 27.4: Примерная зависимость напряжения Li-ion аккумулятора от остатка заряда. В середине кривая почти плоская, поэтому оценка в процентах всегда грубая

Листинг 27.1: Измерение напряжения батареи и защита от глубокого разряда

```
const int BAT_PIN = 2;                    // ADC1; делитель 100 кОм + 100 кОм
const float DIVIDER = 2.0;                // делитель уменьшает напряжение вдвое
const float LOW_BATTERY = 3.40;           // ниже -- пора выключаться

float readBattery() {                     // напряжение аккумулятора в вольтах
  uint32_t sum = 0;
  for (int i = 0; i < 32; i++) sum += analogReadMilliVolts(BAT_PIN);
  return (sum / 32.0) / 1000.0 * DIVIDER;
}

int batteryPercent(float v) {             // грубая оценка для Li-ion по напряжению
  const float V[] = {3.30, 3.60, 3.70, 3.80, 3.95, 4.20};
  const int   P[] = {0,    10,   30,   55,   80,   100};
  if (v <= V[0]) return 0;
  if (v >= V[5]) return 100;
  for (int i = 1; i < 6; i++) {
    if (v < V[i]) {                       // линейная интерполяция между точками таблицы
      return P[i - 1] + (int)((v - V[i - 1]) / (V[i] - V[i - 1]) * (P[i] - P[i - 1]));
    }
  }
  return 100;
}

void setup() {
  Serial.begin(115200);
}

void loop() {
  float v = readBattery();
  Serial.printf("Батарея: %.2f В, примерно %d %%\n", v, batteryPercent(v));
  if (v < LOW_BATTERY) {
    Serial.println("Батарея разряжена: сохраняем данные и засыпаем");
    Serial.flush();
    esp_sleep_enable_timer_wakeup(600ULL * 1000000);   // проверить снова через 10 минут
    esp_deep_sleep_start();
  }
  delay(2000);
}
```

Программа оценивает проценты по таблице из шести точек, а между точками проводит прямые. Когда напряжение падает ниже $3{,}4\,\text{V}$, устройство сохраняет данные и уходит в глубокий сон (урок 9): так оно не разрядит аккумулятор «в ноль».

Осторожно

Измеряйте батарею без нагрузки или всегда в одинаковых условиях. Во время передачи по Wi-Fi напряжение проседает, и программа может решить, что аккумулятор пуст, хотя заряда ещё достаточно.

27.5 Сколько проработает устройство

Время работы оценивают делением ёмкости на средний ток: аккумулятор 2000 mAh при токе 100 mA проработает около 20 часов. Формула среднего тока для устройства, которое спит и просыпается, разобрана в уроке 9. Реальное время на 20–30 % меньше расчётного: часть энергии теряется в преобразователе, а полностью разряжать аккумулятор нельзя.

27.6 Самостоятельный опыт: индикатор заряда

Запитайте плату от трёх батареек AA через вывод 5V и измеряйте их напряжение через тот же делитель. Для батареек пороги другие: около $4{,}5$ V — новые, ниже $3{,}6$ V — пора менять. Покажите заряд тремя светодиодами (или полоской на OLED) и убедитесь, что устройство работает без компьютера.

Если не работает

- **От батареи плата перезагружается при включении Wi-Fi** — источник не выдерживает пики тока; добавьте конденсатор 470 – 1000 µF у выводов питания или возьмите источник мощнее.
- **Powerbank отключается через полминуты** — он считает малый ток признаком отсутствия нагрузки; нужен powerbank с режимом малых токов или другой источник.
- **Показания напряжения занижены** — проверьте коэффициент делителя: измерьте батарею мультиметром и поправьте `DIVIDER`.

Итоги урока

- Источник выбирают по ёмкости, безопасности и способу зарядки; для первых опытов — powerbank или батарейки AA.
- Литиевый аккумулятор используют только с платой защиты и модулем зарядки.
- Заряд оценивают по напряжению через делитель и АЦП; при разряде устройство засыпает само.

Проверь себя

1. Почему литиевый аккумулятор нельзя использовать без платы защиты?
2. Зачем перед АЦП ставят делитель и почему его резисторы берут по $100\,\text{k}\Omega$, а не по $100\,\Omega$?
3. Сколько примерно проработает устройство со средним током 20 mA от аккумулятора 1000 mAh?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Измерьте напряжение батареи мультиметром и запишите его.

2. ★ Рассчитайте время работы платы от батареи по её ёмкости и измеренному току.
3. ★★ Измеряйте напряжение батареи через делитель и АЦП и сравните с мультиметром.
4. ★★ Покажите процент заряда на экране по таблице напряжений.
5. ★★★ Записывайте напряжение батареи раз в минуту в NVS или на SD-карту и постройте график разряда.
6. ★★★★ Сделайте автономный датчик температуры, который просыпается раз в 10 минут, и оцените время его работы от батареи.

Урок 28. Порядок в коде и инструменты разработчика

Цель	разделять программу на файлы и классы, хранить историю изменений в Git и знать инструменты, которые понадобятся в большом проекте
Опирается на	урок 3 (кнопка с подавлением дребезга), урок 14 (диагностика), приложение «Шпаргалка по C++»
Результат	кнопка и светодиод оформлены как классы в отдельных файлах; у проекта есть история изменений
Понадобится	компьютер с установленным Git; новых деталей не нужно

28.1 Сначала: программа растёт быстрее, чем кажется

Скетч на сорок строк помещается в голове целиком. Итоговый проект — это несколько сотен строк, три датчика, экран и сеть. Если всё это лежит в одном файле с десятком глобальных переменных, любая правка что-нибудь ломает, а вчерашнюю рабочую версию уже не вернуть. Этот урок не про новое «железо», а про привычки, которые отличают законченный проект от брошенного: части программы лежат отдельно, у каждой одна задача, а история изменений сохраняется.

28.2 От функций к классам

В уроке 3 кнопка состояла из функции `buttonPressed()` и трёх переменных, спрятанных внутри неё. Для второй кнопки пришлось бы копировать всё целиком. Класс решает эту задачу: он объединяет данные (номер вывода, время последнего изменения) и действия над ними (`begin()`, `pressed()`) в один новый тип. Сколько нужно кнопок — столько создаём объектов.

Листинг 28.1: Кнопка и светодиод как классы

```
class Button {                          // кнопка с подавлением дребезга (урок 3)
 public:
  explicit Button(int pin) : pin_(pin) {}

  void begin() {
    pinMode(pin_, INPUT_PULLUP);
    stable_ = last_ = digitalRead(pin_);
  }

  bool pressed() {                      // true ровно один раз на нажатие
    bool r = digitalRead(pin_);
    if (r != last_) { last_ = r; changed_ = millis(); }
```

```
    if (millis() - changed_ > 50 && r != stable_) {
      stable_ = r;
      return stable_ == LOW;
    }
    return false;
  }

 private:                                  // снаружи эти переменные недоступны
  int pin_;
  bool stable_ = HIGH;
  bool last_ = HIGH;
  uint32_t changed_ = 0;
};

class Led {
 public:
  explicit Led(int pin) : pin_(pin) {}
  void begin() { pinMode(pin_, OUTPUT); }
  void set(bool on) { on_ = on; digitalWrite(pin_, on_); }
  void toggle() { set(!on_); }

 private:
  int pin_;
  bool on_ = false;
};

Button button(5);                         // два объекта -- и никаких глобальных флагов
Led led(4);

void setup() {
  button.begin();
  led.begin();
}

void loop() {
  if (button.pressed()) led.toggle();
}
```

Раздел `private` — это «внутренности»: к ним нельзя обратиться снаружи, а значит, нельзя и случайно испортить. Всё общение с объектом идёт через несколько функций из раздела `public`. Посмотрите на `loop()`: одна строка читается как фраза на человеческом языке.

28.3 Несколько файлов

Когда класс готов и отлажен, его выносят из основного скетча. В Arduino IDE новая вкладка создаётся кнопкой с тремя точками справа от названия скетча (New Tab).

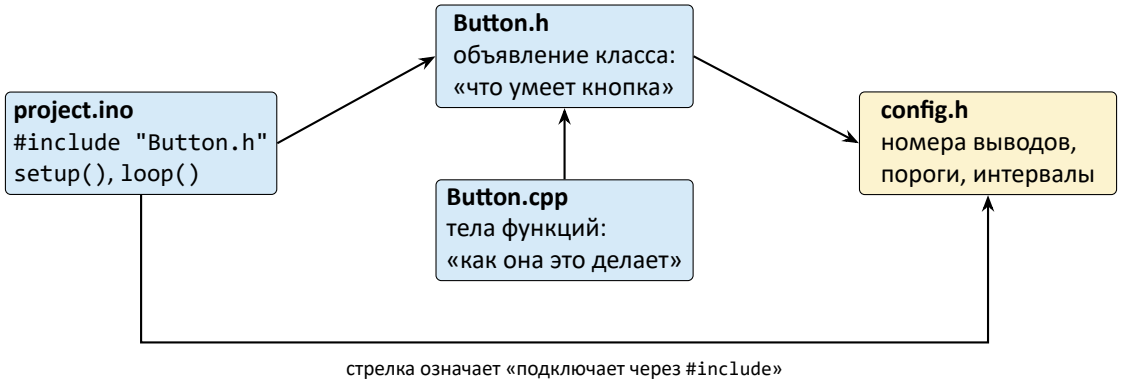


Рис. 28.1: Типичная структура проекта: основной файл остаётся коротким, каждый класс живёт в паре файлов, настройки собраны в одном месте

```
// Button.h
#pragma once                    // защита от повторного подключения файла
#include <Arduino.h>

class Button {
 public:
  explicit Button(int pin);
  void begin();
  bool pressed();
 private:
  int pin_;
  bool stable_ = HIGH, last_ = HIGH;
  uint32_t changed_ = 0;
};
```

Три правила порядка, которые стоит принять сразу:

- **Числа получают имена.** Номера выводов, пороги и интервалы — константы в `config.h`, а не «магические» числа, разбросанные по коду.
- **Одна функция — одно дело.** Если функцию трудно назвать коротко, она делает слишком много.
- **Пароли не хранятся рядом с кодом.** Имя и пароль сети выносят в отдельный файл `secrets.h`, который не публикуют.

28.4 Git: машина времени для проекта

Git хранит снимки проекта (коммиты). К любому снимку можно вернуться, сравнить его с текущей версией и увидеть, какая именно правка всё сломала.

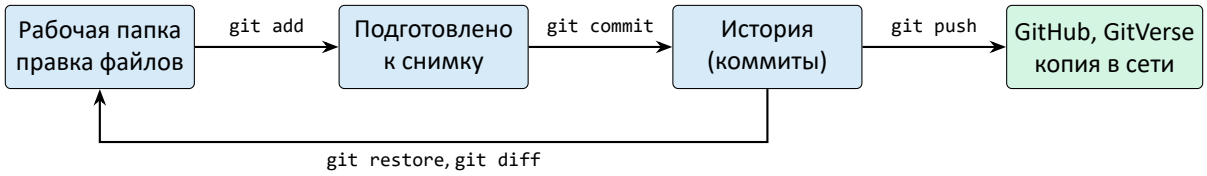


Рис. 28.2: Путь изменений в Git: от правки файла до копии проекта в сети

```
git init                          # один раз: сделать папку проекта репозиторием
git add .                         # подготовить все изменения
git commit -m "Кнопка включает светодиод"    # сохранить снимок с описанием
git status                        # что изменилось с последнего снимка
```

```
git log --oneline                # список снимков
git diff                         # что именно изменено в файлах
```

Делайте коммит каждый раз, когда что-то заработало, и пишите в описании, *что* теперь умеет программа. Перед рискованной правкой коммит обязателен: вернуться к нему — дело одной команды.

Осторожно

Не отправляйте в открытый репозиторий пароли Wi-Fi, ключи API и сертификаты. Добавьте `secrets.h` в файл `.gitignore` до первого коммита: удалить пароль из уже опубликованной истории очень трудно.

28.5 Что ещё пригодится

Инструмент	Зачем нужен
PlatformIO (расширение для VS Code)	проекты из многих файлов, автодополнение, версии библиотек записаны в файле проекта
Симулятор Wokwi	проверить схему и программу без платы; удобно для домашней работы
Логический анализатор	увидеть сигналы UART, I^2C и SPI на экране компьютера и расшифровать их
Встроенный USB-JTAG	пошаговая отладка: остановить программу на строке и посмотреть переменные
Мультиметр и осциллограф	измерить питание и форму сигнала, когда «программа верная, а не работает»

В PlatformIO (подробно — в уроке 41) настройки проекта записаны в файле `platformio.ini`. Достаточно скопировать папку проекта на другой компьютер, и нужные библиотеки установятся сами:

```
[env:esp32s3]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
monitor_speed = 115200
lib_deps =
    adafruit/Adafruit SSD1306
    bblanchon/ArduinoJson
```

28.6 Самостоятельный опыт: наведите порядок

Возьмите свой самый большой скетч из прошлых уроков. Создайте для него репозиторий и сделайте первый коммит. Затем вынесите номера выводов в `config.h`, а кнопку и светодиод — в классы в отдельных файлах. После каждого шага проверяйте, что программа работает, и делайте коммит.

Если не работает

- **Ошибка `redefinition of class`** — в заголовочном файле нет строки `#pragma once`.
- **Ошибка `was not declared in this scope`** — в файле `.cpp` забыт `#include <Arduino.h>` или заголовок своего класса.
- **Git «не видит» изменения** — файл не сохранён в редакторе или указан в `.gitignore`.

Итоги урока

- Класс объединяет данные и действия; объектов можно создать сколько угодно.
- Большой проект делят на файлы, а настройки и пароли хранят отдельно от логики.
- Git сохраняет историю: коммит после каждого рабочего шага и перед каждой рискованной правкой.

Проверь себя

1. Чем класс удобнее набора функций и глобальных переменных?
2. Что пишут в файле `.h` и что — в файле `.cpp`?
3. Когда нужно делать коммит и что писать в его описании?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Разбейте свой скетч на функции с понятными названиями.
2. ★ Добавьте комментарии: что делает каждая функция и какие у неё параметры.
3. ★★ Добавьте классу `Led` функцию `blink(период)`, которая мигает без `delay()`.
4. ★★ Выделите работу с кнопкой в отдельный класс с защитой от дребезга.
5. ★★★ Напишите класс `Potentiometer` с усреднением и порогом из урока 5 и вынесите его в отдельные файлы.
6. ★★★★ Перенесите один из своих скетчей в PlatformIO, добавьте `.gitignore` и опубликуйте проект в сети.

Урок 29. Цифровой звук: I²S и микрофон

Цель	получать звуковые отсчёты и измерять уровень сигнала
Опирается на	уроки 5, 7 и 24
Результат	график относительной громкости речи
Понадобится	INMP441, провода; для расширения — I²S-усилитель

29.1 Звук — поток чисел

Пищалка из урока 24 воспроизводила одну частоту. Для речи нужен поток отсчётов: каждое число задаёт амплитуду в определённый момент. Частота дискретизации 16000 Гц означает 16000 отсчётов в секунду. Разрядность задаёт точность отсчёта, число каналов — количество потоков. I²S передаёт такой поток между чипом и цифровым микрофоном, кодеком или усилителем. I²S и I²C — разные интерфейсы. У стандартного I²S есть BCLK (такты битов), WS (выбор канала) и линия данных. В нашем опыте ESP32-S3 формирует такты, микрофон выдаёт данные. Согласуйте частоту, длину слота, канал и формат Philips I²S. Для 16000 кадров/с, двух слотов по 32 бита получаем BCLK $16000 \cdot 2 \cdot 32 = 1024000$ Гц.

29.2 Подключение и первый поток

Отключите предыдущую схему. INMP441: VDD к 3V3, GND к GND, SCK к GPIO16, WS к GPIO17, SD к GPIO18, L/R к GND. Последняя перемычка выбирает левый слот. Проверьте питание и маркировку конкретного модуля.

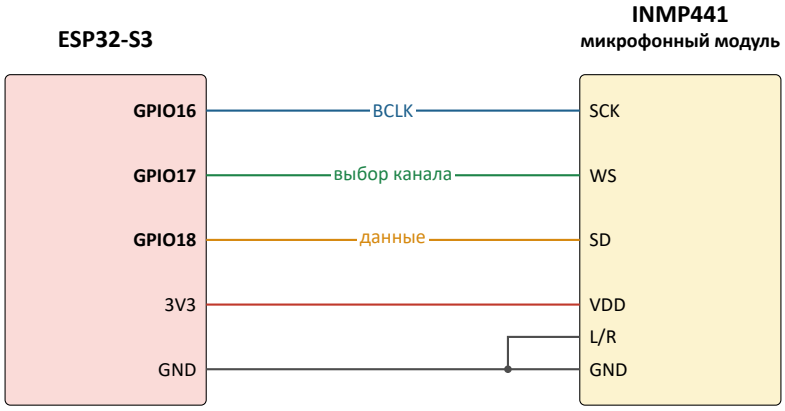


Рис. 29.1: Подключение микрофона INMP441: три сигнальные линии, питание 3,3 V и перемычка L/R на землю (левый слот). Порядок выводов на модуле зависит от производителя

В Arduino-ESP32 3.2.0 используем `ESP_I2S.h`, а не примеры старого драйвера `driver/i2s.h`.

Листинг 29.1: Отсчёты INMP441 и среднеквадратичный уровень

```
// INMP441: VDD=3V3, GND=GND, L/R=GND (left slot).
#include <ESP_I2S.h>
#include <math.h>
I2SClass mic;
bool ready = false;
int32_t samples[256];
void setup() {
  Serial.begin(115200);
  mic.setPins(16, 17, -1, 18); // BCLK, WS, DOUT unused, DIN
  ready = mic.begin(I2S_MODE_STD, 16000,
                    I2S_DATA_BIT_WIDTH_32BIT, I2S_SLOT_MODE_MONO,
                    I2S_STD_SLOT_LEFT);
  if (!ready) Serial.println("I2S init failed");
}
void loop() {
  if (!ready) { delay(100); return; }
  size_t bytes = mic.readBytes((char*)samples, sizeof(samples));
  size_t n = bytes / sizeof(samples[0]);
  if (!n) return;
  double mean = 0, energy = 0;
  // INMP441: signed sample is left-aligned in the 32-bit slot.
  for (size_t i = 0; i < n; ++i) mean += samples[i] / 256.0;
  mean /= n;
  for (size_t i = 0; i < n; ++i) {
    double x = samples[i] / 256.0 - mean;
    energy += x * x;
  }
  Serial.printf("rms:%.1f\n", sqrt(energy / n));
  delay(20);
}
```

29.3 Что именно измеряет пример

У микрофона полезные данные занимают 24 бита внутри 32-битного слота. Деление на 256 убирает младшие биты, сохраняя знак. Вычитаем среднее окна, чтобы постоянное смещение не считалось звуком. Вычисляем RMS $= \sqrt{\frac{1}{N}\sum(x_i - \bar{x})^2}$. Это относительный уровень, а не дБ SPL: для звукового давления нужна акустическая калибровка.
DMA передаёт блоки между периферией и памятью: CPU не обслуживает каждый бит. Чтение блока может ждать данные. Для устройства с экраном и кнопками вынесите чтение аудио в отдельную задачу, а результаты передавайте очередью. Буфер должен оставаться доступным, пока им пользуется драйвер. Не печатайте каждый отсчёт текстом на 115200 бод: каналу может не хватить пропускной способности.

29.4 От измерения к воспроизведению

Сравните фон, речь и хлопок на одинаковом расстоянии. Добавьте гистерезис индикатора «громко». Для воспроизведения используют I²S-усилитель, например MAX98357A: к нему идут BCLK, WS и DOUT, динамик подключается к выходам усилителя. Питание выбирают по документации модуля. У ESP32-S3 нет встроенного аналогового DAC: соответствующие примеры первого ESP32 нельзя переносить напрямую. Цифровой I²S, внешний DAC и PWM решают разные задачи.

Если не работает

Нули: выбран не тот слот или отсутствуют такты. Шум без звука: формат, выравнивание или питание. Пропуски: обработка и печать занимают слишком много времени. Проверяйте фактическое число прочитанных байтов, а не предполагаемую длину массива.

Итоги урока

Согласуем такты, слоты и формат, обрабатываем блоки. Различаем относительную амплитуду, цифровой уровень и откалиброванное звуковое давление.

Проверь себя

1. Чем I²S отличается от I²C?
2. Почему 24-битный микрофон читаем в 32-битный массив?
3. Почему RMS нельзя подписать «дБ SPL»?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Составьте таблицу соединений BCLK, WS, SD и питания.
2. ★ Запишите фон и максимум хлопка на расстоянии 30 см.
3. ★★ Сравните окна 256 и 512 отсчётов по задержке обновления.
4. ★★ Добавьте индикатор «громко» с разными порогами включения и выключения.
5. ★★★ Вынесите чтение в задачу и измерьте число пропущенных блоков при работе экрана.
6. ★★★★ Запишите PCM/WAV на SD: рассчитайте размер и проверьте длительность на компьютере.

Документация для продолжения Официальное руководство по теме.

Урок 30. Точные импульсы: периферия RMT

Цель	формировать пакеты уровней аппаратно и выбирать точность времени
Опирается на	уроки 17 и 21
Результат	пакет из двух импульсов с проверенной длительностью
Понадобится	логический анализатор или вторая плата

30.1 Когда программная задержка мешает

Для адресного светодиода или ИК-пульта важна длительность каждого уровня. Цикл с задержками зависит от прерываний и планировщика. RMT получает массив уровней и длительностей, затем формирует или измеряет фронты аппаратно. Периодический PWM удобнее получать LEDC, а переменный пакет — RMT.

Один символ содержит две части: `level0/duration0` и `level1/duration1`. При 1 МГц один такт равен 1 мкс, при 10 МГц — 0,1 мкс. Более высокая точность уменьшает максимальный интервал при той же разрядности поля. У ESP32-S3 TX и RX имеют отдельные ресурсы; библиотека NeoPixel уже может занимать RMT, поэтому учитывайте соседние драйверы.

30.2 Пакет, который можно проверить

Соедините GPIO4 с входом анализатора, земли объедините. Без анализатора можно соединить GPIO4 с GPIO5 опыта PCNT из урока 31. Счётчик увидит два положительных фронта на пакет, но не проверит их длительность. На выход подаётся логика 3,3 В.

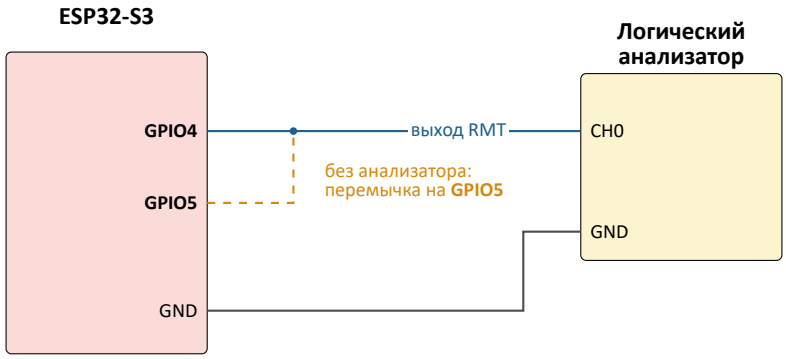


Рис. 30.1: Проверка выхода RMT: **GPIO4** подключается к входу логического анализатора, земли объединяются. Пунктиром показана замена анализатору — перемычка на вход PCNT из урока 31

Листинг 30.1: Две пары уровней: 100/200 и 300/400 мкс

```
// GPIO4 -> logic analyser, or GPIO5 of the PCNT experiment.
#include <Arduino.h>
rmt_data_t pulse[2];
bool ready = false;
void setup() {
  Serial.begin(115200);
  ready = rmtInit(4, RMT_TX_MODE, RMT_MEM_NUM_BLOCKS_1, 1000000);
  pulse[0].duration0 = 100; pulse[0].level0 = 1;
  pulse[0].duration1 = 200; pulse[0].level1 = 0;
  pulse[1].duration0 = 300; pulse[1].level0 = 1;
  pulse[1].duration1 = 400; pulse[1].level1 = 0;
```

```
  if (!ready) Serial.println("RMT init failed");
}
void loop() {
  if (ready && !rmtWrite(4, pulse, 2, 100))
    Serial.println("RMT timeout");
  delay(1000);
}
```

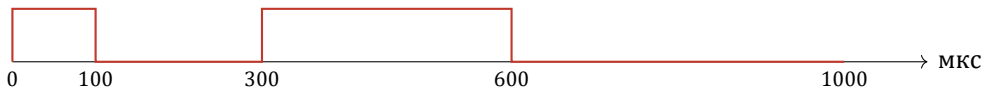


Рис. 30.2: Один пакет RMT, общая длительность 1000 мкс

30.3 ИК: несущая, огибающая и команда

ИК-пульт обычно передаёт внутри отметки колебания несущей, например 38 кГц. Демодулирующий приёмник убирает несущую и выдаёт огибающую, нередко активным LOW. Передатчику нужны согласованная несущая, драйвер светодиода и ограничение тока. Наш пример создаёт логический пакет, а не готовую команду телевизору.
Для декодирования сначала выделяют старт, затем длительности битов и паузы; конкретные числа берут из описания выбранного протокола. Начните с записи собственного пульта: сравните одиночное нажатие и удержание, найдите повторяющиеся участки. Ошибка приёма не всегда означает плохую библиотеку: источником может быть световая помеха или питание.

30.4 Асинхронность и память

`rmtWrite` ждёт завершения с ограниченным тайм-аутом. При асинхронном вызове массив должен жить до окончания передачи. Локальный массив завершившейся функции для этого не подходит; также нельзя менять буфер, который ещё читает периферия. При RX конец пакета определяют паузой: слишком короткий порог разрежет сообщение, слишком длинный задержит его обработку. Учитывайте ёмкость буфера и число символов.

Если не работает

Нет сигнала: неверный GPIO или занят канал. Пакет обрезан: не хватает буфера либо тайм-аута. ИК-приёмник молчит: нет нужной несущей или перепутаны питание и выход. Проверяйте сначала форму сигнала, затем смысл команды.

Итоги урока

RMT задаёт аппаратные длительности, частота счётчика определяет цену такта. Несущая, огибающая и содержание команды — разные уровни описания.

Проверь себя

1. Сколько длятся 250 тактов при 1 МГц?
2. Почему PCNT недостаточно для проверки пакета?
3. Как долго должен жить буфер асинхронной передачи?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Пересчитайте четыре длительности примера в микросекунды.
2. ★ Запишите пакет и сравните с рисунком.
3. ★★ Добавьте третий импульс 500/500 мкс и измерьте общую длительность.
4. ★★ Сравните точность и предельный интервал при 1 и 10 МГц.
5. ★★★ Передавайте асинхронно, продолжая читать кнопку; обеспечьте время жизни буфера.
6. ★★★★ Запишите огибающую собственного ИК-пульта, выделите старт, биты и повтор удержания.

Документация для продолжения Официальное руководство по теме.

Урок 31. Счётчики и движение: PCNT и MCPWM

Цель	считать фронты аппаратно и формировать управляющий PWM
Опирается на	уроки 4, 16, 23 и 25
Результат	измеренная частота и стабильный импульс сервопривода
Понадобится	перемычка GPIO4–GPIO5; сервопривод с отдельным питанием по желанию

31.1 Периферия считает вместо ISR

При частых фронтах обработчик на каждое событие отнимает CPU и может пропустить импульс. PCNT считает фронты аппаратно; процессор периодически читает результат. Канал умеет учитывать направление по второй линии, поэтому квадратурный энкодер можно декодировать аппаратно. Фильтр коротких импульсов помогает с помехами, но не заменяет устранение дребезга механической кнопки: полезный сигнал нельзя удалить вместе с шумом.

31.2 Частотомер на одной плате

Отключите LED и кнопку от GPIO4/5 и соедините GPIO перемычкой. LEDC формирует 1000 Гц, PCNT считает положительные фронты. За секунду ожидаем около 1000 событий; точное время окна измеряем через `millis()`. Все логические сигналы в опыте имеют уровень 3,3 В.

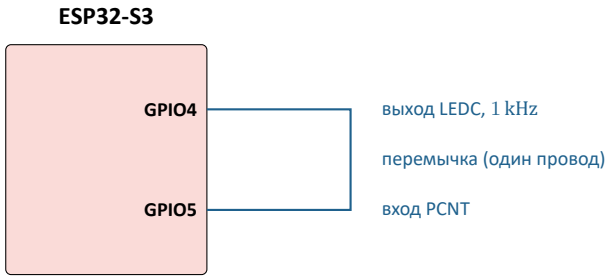


Рис. 31.1: Частотомер на одной плате: перемычка соединяет выход генератора **GPIO4** со входом счётчика **GPIO5**. Другие детали к этим выводам не подключайте

Листинг 31.1: PCNT: аппаратный подсчёт фронтов генератора

```
// GPIO4 (LEDC generator) -> jumper -> GPIO5 (PCNT input).
#include <Arduino.h>
#include "driver/pulse_cnt.h"
pcnt_unit_handle_t counter = nullptr;
void setup() {
  Serial.begin(115200);
  pcnt_unit_config_t unit = {};
  unit.low_limit = -30000; unit.high_limit = 30000;
  ESP_ERROR_CHECK(pcnt_new_unit(&unit, &counter));
  pcnt_glitch_filter_config_t filter = {};
  filter.max_glitch_ns = 1000;
  ESP_ERROR_CHECK(pcnt_unit_set_glitch_filter(counter, &filter));
  pcnt_chan_config_t input = {};
  input.edge_gpio_num = 5; input.level_gpio_num = -1;
  pcnt_channel_handle_t channel = nullptr;
  ESP_ERROR_CHECK(pcnt_new_channel(counter, &input, &channel));
```

```
  ESP_ERROR_CHECK(pcnt_channel_set_edge_action(channel,
      PCNT_CHANNEL_EDGE_ACTION_INCREASE, PCNT_CHANNEL_EDGE_ACTION_HOLD));
  ESP_ERROR_CHECK(pcnt_unit_enable(counter));
  ESP_ERROR_CHECK(pcnt_unit_clear_count(counter));
  ESP_ERROR_CHECK(pcnt_unit_start(counter));
  if (!ledcAttach(4, 1000, 8)) {
    Serial.println("LEDC init failed"); return;
  }
  ledcWrite(4, 128);
}
void loop() {
  static int previous = 0;
  static uint32_t last = millis();
  uint32_t now = millis();
  if (now - last < 1000) return;
  int current = 0;
  ESP_ERROR_CHECK(pcnt_unit_get_count(counter, &current));
  // Modular difference; valid if fewer than 30000 pulses per interval.
  int delta = current - previous;
  if (delta < 0) delta += 30000;
  Serial.printf("count:%d frequency:%.1f Hz\n", current,
                delta * 1000.0 / (now - last));
  previous = current; last = now;
}
```

Граница счётчика конечна. При достижении верхнего предела 30000 аппаратный счётчик обнуляется. В примере разность корректируется по модулю 30000: за окно должно прийти меньше 30000 событий, а счёт идёт только вверх. Для реального двунаправленного энкодера нужны watch points и расширение счётчика; этот частный приём туда напрямую не переносится. Чтение и очистка отдельными командами могли бы потерять фронт, поэтому каждую секунду не очищаем.

31.3 MCPWM и форма управляющего сигнала

LEDC подходит для яркости и простого PWM. MCPWM добавляет связанные генераторы, компараторы, синхронизацию, захват и обработку аварий. Dead time — пауза между выключением одного плеча моста и включением другого. Она снижает риск одновременного открытия плеч. GPIO всё равно управляет входом драйвера, а не обмоткой или силовым ключом напрямую.

Листинг 31.2: MCPWM: период 20 мс, высокий уровень 1500 мкс

```
// Signal only: GPIO6. Power servo from an appropriate external supply.
#include <Arduino.h>
#include "driver/mcpwm_prelude.h"
void setup() {
  Serial.begin(115200);
  mcpwm_timer_handle_t timer = nullptr;
  mcpwm_timer_config_t tc = {};
  tc.group_id = 0; tc.clk_src = MCPWM_TIMER_CLK_SRC_DEFAULT;
  tc.resolution_hz = 1000000; tc.period_ticks = 20000;
  tc.count_mode = MCPWM_TIMER_COUNT_MODE_UP;
  ESP_ERROR_CHECK(mcpwm_new_timer(&tc, &timer));
  mcpwm_oper_handle_t oper = nullptr;
  mcpwm_operator_config_t oc = {}; oc.group_id = 0;
  ESP_ERROR_CHECK(mcpwm_new_operator(&oc, &oper));
  ESP_ERROR_CHECK(mcpwm_operator_connect_timer(oper, timer));
  mcpwm_cmpr_handle_t cmp = nullptr;
  mcpwm_comparator_config_t cc = {}; cc.flags.update_cmp_on_tez = true;
```

```
  ESP_ERROR_CHECK(mcpwm_new_comparator(oper, &cc, &cmp));
  ESP_ERROR_CHECK(mcpwm_comparator_set_compare_value(cmp, 1500));
  mcpwm_gen_handle_t gen = nullptr;
  mcpwm_generator_config_t gc = {}; gc.gen_gpio_num = 6;
  ESP_ERROR_CHECK(mcpwm_new_generator(oper, &gc, &gen));
  ESP_ERROR_CHECK(mcpwm_generator_set_action_on_timer_event(gen,
      MCPWM_GEN_TIMER_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP,
      MCPWM_TIMER_EVENT_EMPTY, MCPWM_GEN_ACTION_HIGH)));
  ESP_ERROR_CHECK(mcpwm_generator_set_action_on_compare_event(gen,
      MCPWM_GEN_COMPARE_EVENT_ACTION(MCPWM_TIMER_DIRECTION_UP,
      cmp, MCPWM_GEN_ACTION_LOW)));
  ESP_ERROR_CHECK(mcpwm_timer_enable(timer));
  ESP_ERROR_CHECK(mcpwm_timer_start_stop(timer, MCPWM_TIMER_START_NO_STOP));
}
void loop() { delay(100); }
```

Таймер задаёт шкалу 1 мкс и период 20000 отсчётов. Событие начала периода включает выход, совпадение с компаратором 1500 выключает его. Получается команда среднего положения на GPIO6. Питание сервопривода отдельное, GND объединён. Допустимый диапазон импульсов проверьте для модели: начинайте с середины, не с механических упоров.

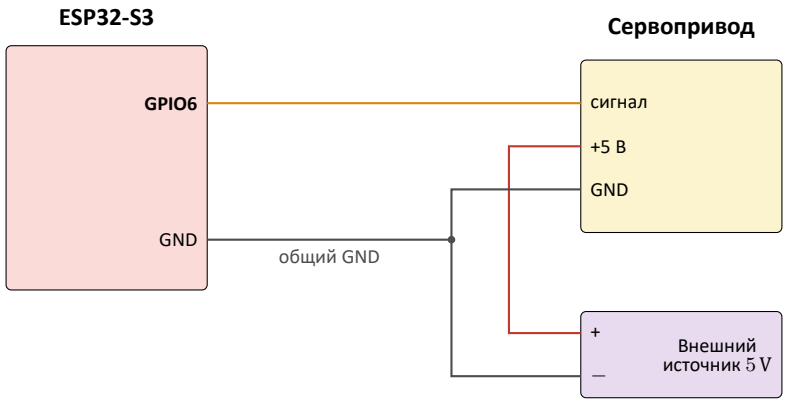


Рис. 31.2: Подключение сервопривода: сигнал с **GPIO6**, питание 5 V от отдельного источника, земли платы и источника соединены. Провод питания сервопривода к 3V3 платы не подключайте

31.4 Первый контур обратной связи

Управление скоростью состоит из задания, измерения PCNT, ошибки и изменения PWM. Проверьте каждый блок отдельно. Перевод числа событий в обороты зависит от импульсов на оборот и выбранных фронтов: квадратурный режим x4 даёт иной коэффициент, чем одна линия и один фронт. Период регулирования должен быть известен; случайные задержки ухудшают поведение контура.

Если не работает

Ноль событий: нет перемычки или выбран другой GPIO. Удвоенная частота: считаются оба фронта. Потеря положения: неверный коэффициент энкодера или не обработана граница счётчика.

Итоги урока

PCNT считает события, MCPWM задаёт связанные выходы. Для измерения нужны время окна, обработка границ и правильное число фронтов на оборот.

Проверь себя

1. Сколько положительных фронтов даёт 1000 Гц за 250 мс?
2. Зачем обрабатывать границы PCNT?
3. Что означает dead time?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Нарисуйте соединение генератора и PCNT.
2. ★ Запишите пять измерений около 1000 Гц.
3. ★★ Сравните 100, 1000 и 5000 Гц при двух длинах окна.
4. ★★ Измените компаратор на 1400 мкс и сначала проверьте анализатором.
5. ★★★ Подключите энкодер с аппаратным определением направления и проверьте возврат к нулю.
6. ★★★★ Спроектируйте регулирование скорости низковольтного мотора с ограничением PWM и аварийной остановкой.

Документация для продолжения Официальное руководство по теме. Архитектура MCPWM.

Урок 32. Проводная сеть: TWAI/CAN и RS-485

Цель	разделять контроллер и физический интерфейс и связать два узла
Опирается на	уроки 2 и 15
Результат	сообщения между двумя ESP32-S3 по кабелю
Понадобится	две платы и два CAN- либо RS-485-трансивера с логикой 3,3 В

32.1 Контроллер, трансивер и протокол

Дифференциальный приёмник сравнивает две линии: одинаково наведённая помеха меньше влияет на результат. Это не отменяет ограничений по длине, скорости, земле и согласованию. При разных источниках питания проверьте допустимую разность потенциалов; удалённым узлам могут понадобиться развязка и защита, которых нет на простой макетке.

	TWAI / Classical CAN	UART через RS-485
В чипе	контроллер кадров и арбитража	UART, направление задаёт программа
Снаружи	CAN-трансивер	RS-485-трансивер
Данные	ID и до 8 байтов	поток байтов, границы задаёт приложение

TWAI ESP32-S3 поддерживает Classical CAN, а не CAN FD. RS-485 — электрический интерфейс, он сам не добавляет адреса, контрольную сумму и команды Modbus.

32.2 Два узла CAN

Используйте отдельную учебную шину. GPIO16 идёт на TXD трансивера, RXD на GPIO17. Соедините CANH с CANH, CANL с CANL, сигнальные земли в пределах стенда объедините. На двух физических концах ставят по 120 Ом между CANH и CANL, а не на каждом узле. У модулей резисторы могут быть уже установлены. Проверьте питание модели и допустимые уровни RXD.

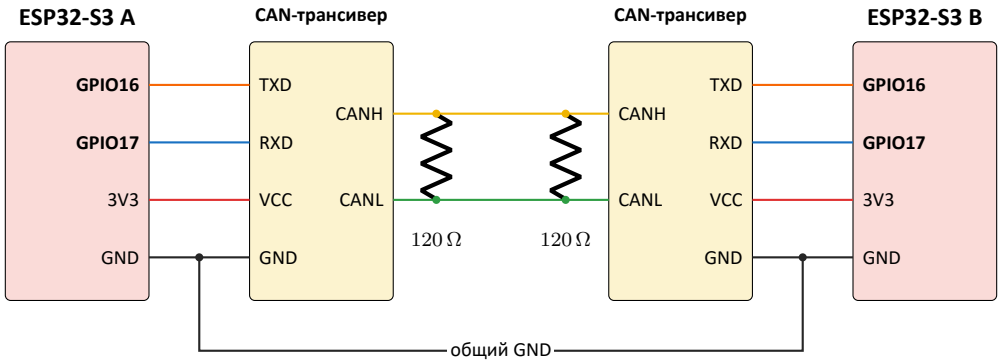


Рис. 32.1: Две платы по CAN: TXD трансивера на **GPIO16**, RXD на **GPIO17**, CANH с CANH, CANL с CANL; резисторы 120 Ω стоят только на двух концах шины. Трансиверы питаются от 3V3 (модель с логикой 3,3 В)

Листинг 32.1: Кадры Classical CAN со счётчиком

```
// Two external 3.3V-logic CAN transceivers; 500 kbit/s on both boards.
#include <Arduino.h>
#include "driver/twai.h"
const uint32_t NODE_ID = 0x101; // second board: 0x102
bool ready = false;
void setup() {
  Serial.begin(115200);
  twai_general_config_t general = TWAI_GENERAL_CONFIG_DEFAULT(GPIO_NUM_16, GPIO_NUM_17,
                                  TWAI_MODE_NORMAL);
  twai_timing_config_t timing = TWAI_TIMING_CONFIG_500KBITS();
  twai_filter_config_t filter = TWAI_FILTER_CONFIG_ACCEPT_ALL();
  ESP_ERROR_CHECK(twai_driver_install(&general, &timing, &filter));
  ESP_ERROR_CHECK(twai_start()); ready = true;
}
void loop() {
  if (!ready) return;
  static uint32_t last = 0, sequence = 0;
  if (millis() - last >= 1000) {
    last = millis();
    twai_message_t tx = {}; tx.identifier = NODE_ID;
    tx.data_length_code = 4;
    for (int i = 0; i < 4; ++i) tx.data[i] = (sequence >> (8 * i)) & 255;
    esp_err_t err = twai_transmit(&tx, pdMS_TO_TICKS(20));
    Serial.printf("queue result=%s seq=%lu\n", esp_err_to_name(err),
                  (unsigned long)sequence++);
  }
  twai_message_t rx;
  if (twai_receive(&rx, pdMS_TO_TICKS(10)) == ESP_OK)
    Serial.printf("RX id=%03lx bytes=%d first=%d\n",
        (unsigned long)rx.identifier, rx.data_length_code, rx.data[0]);
  twai_status_info_t status;
  if (twai_get_status_info(&status) == ESP_OK &&
      status.state == TWAI_STATE_BUS_OFF) {
    Serial.println("BUS OFF: check wiring, then reset");
```

```
    ready = false;
  }
}
```

На второй плате измените ID на `0x102`. Скорость обоих узлов 500 кбит/с. Успех `twai_transmit` означает постановку в очередь, а не доставку приложению. Один узел в нормальном режиме не получит ACK от самого себя. Подтвердите приём другой платой. Для гарантии обработки команды добавьте ответ приложения с номером сообщения. При BUS OFF сначала ищут неисправность линии, затем выполняют восстановление по драйверу.

32.3 Два узла RS-485

У полудуплексного модуля соедините DE и активный LOW вход /RE с GPIO4. DI к GPIO16, RO к GPIO17. HIGH включает передатчик, LOW — приёмник. Линии соединяйте по документации: обозначения A/B у производителей могут различаться. Согласование ставят на концах; смещение задают в одном рассчитанном месте либо используют трансивер с нужным failsafe.

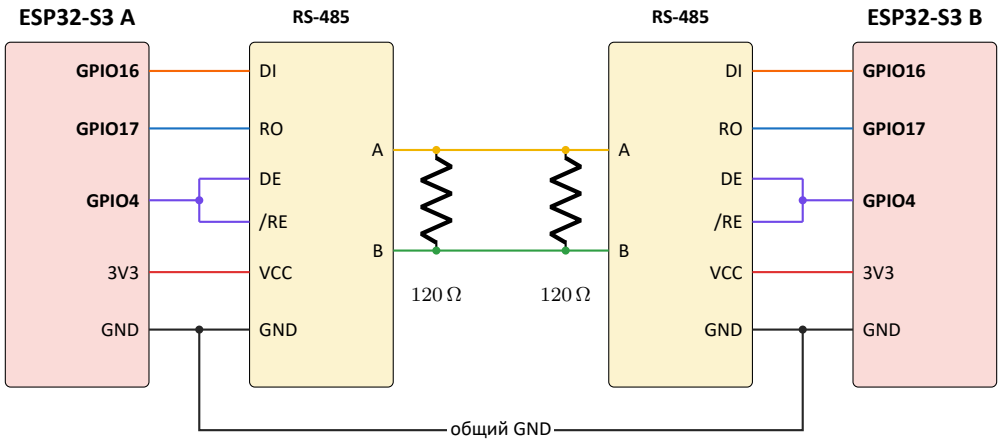


Рис. 32.2: Две платы по RS-485 (модули с логикой 3,3 В): DI на **GPIO16**, RO на **GPIO17**, выводы DE и /RE соединены между собой и идут на **GPIO4**; A с A, B с B (обозначения у модулей могут отличаться). Согласующие 120 Ω стоят на двух концах линии

Листинг 32.2: Учебные запрос PING и ответ PONG

```
// 3.3V RS-485 module: DI=16, RO=17, DE and /RE joined on GPIO4.
#include <Arduino.h>
const bool MASTER = true; // second board: false
const int DIR = 4;
void sendFrame(const char* frame) {
  digitalWrite(DIR, HIGH);
  delayMicroseconds(10);
  Serial1.print(frame);
  Serial1.flush(); // wait for the final stop bit before releasing the bus
  digitalWrite(DIR, LOW);
}
void setup() {
  Serial.begin(115200);
  pinMode(DIR, OUTPUT); digitalWrite(DIR, LOW);
  Serial1.begin(9600, SERIAL_8N1, 17, 16);
}
void loop() {
  static uint32_t last = 0;
  if (MASTER && millis() - last >= 1000) {
```

```
    last = millis(); sendFrame("1,PING\n");
  }
  static char line[48]; static size_t used = 0;
  while (Serial1.available()) {
    char c = Serial1.read();
    if (c == '\n') {
      line[used] = 0; Serial.println(line);
      if (!MASTER && strcmp(line, "1,PING") == 0) {
        delay(2); sendFrame("0,PONG\n");
      }
      used = 0;
    } else if (used < sizeof(line) - 1) line[used++] = c;
    else used = 0;
  }
}
```

На второй плате `MASTER=false`. Только ведущий начинает обмен, второй отвечает после запроса. `flush()` ждёт передачи последнего стоп-бита до выключения DE. Это учебный текстовый протокол, не Modbus RTU. В реальной сети нужны адреса, контроль целостности, тайм-аут, ограниченный повтор и правило предотвращения коллизий. На длинной линии важны топология и длина ответвлений: набор соединённых перемычек не моделирует любой кабель.

Если не работает

CAN без ACK: нет второго рабочего узла, неверна скорость или проводка. RS-485 молчит: неверна полярность пары, DE остался включён либо /RE не активен. Проверяйте физическую линию до разбора содержимого кадров.

Итоги урока

Контроллер формирует данные, трансивер работает с кабелем. CAN имеет арбитраж и кадры, а поверх RS-485 протокол выбирают отдельно. Очередь и доставка — разные события.

Проверь себя

1. Почему нельзя соединить GPIO с CANH напрямую?
2. Где ставят согласующие резисторы?
3. Почему успешный `twai_transmit` не доказывает доставку?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Найдите контроллер, трансивер и кабель на схеме.
2. ★ Настройте два CAN-узла с разными ID и подтвердите приём.
3. ★★ Передайте 16-битную температуру с масштабом 0,01 градуса и опишите порядок байтов.
4. ★★ Повторите PING/PONG через RS-485 и измерьте время ответа.
5. ★★★ Добавьте номер запроса, контрольную сумму и ограниченные повторы; проверьте отключение второго узла.
6. ★★★★ Спроектируйте сеть трёх датчиков: адреса, опрос, согласование, тайм-ауты и реакции на неисправности.

Документация для продолжения Официальное руководство по теме. UART и RS-485.

Урок 33. Цветной экран: SPI, RGB, память и DMA

Цель	рассчитать память кадра и обновлять экран без лишней перерисовки
Опирается на	уроки 6, 10 и 11
Результат	цветная полоса потенциометра с измерением времени рисования
Понадобится	ST7789 240×240 с CS и логикой 3,3 В, потенциометр

33.1 Цена одного кадра

OLED хранил один бит на пиксель. RGB565 занимает два байта: 5 бит красного, 6 зелёного и 5 синего. Кадр 240×240 требует 115200 байт, а 800×480 — 768000 байт. Двойная буферизация удваивает эти объёмы. Дополнительно нужны память программы, задачи, сеть и служебные буферы.

По SPI передаём команды и пиксели в контроллер дисплея, обычно имеющий собственную память. Параллельная RGB-панель требует постоянного потока пикселей и синхросигналов. ESP32-S3 имеет аппаратную поддержку LCD, но подключение RGB-панели не выводится из SPI-примера. Первый опыт делаем на SPI-модуле, а RGB изучаем на готовой плате с экраном.

33.2 Подключение и библиотеки

У модуля должны быть указанное разрешение и выведенный CS. SCK соедините с GPIO12, MOSI с GPIO11, CS с GPIO10, DC с GPIO9, RST с GPIO8; MISO не используется. Потенциометр: края к 3V3/GND, движок к GPIO1. OLED отключите от GPIO8/9. Питание и подсветка зависят от модуля: LED/BL может требовать драйвера. Не подключайте ток подсветки к сигнальному GPIO без проверки документации.

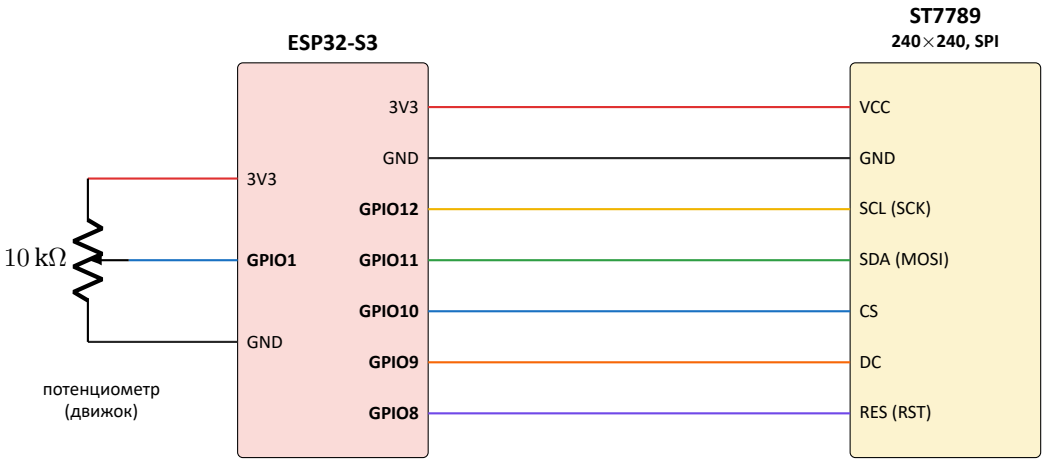


Рис. 33.1: Подключение ST7789 по SPI (**GPIO12**, **GPIO11**, **GPIO10**, **GPIO9**, **GPIO8**) и потенциометра на **GPIO1**. Вывод MISO не используется. Подсветку (BLK) подключайте по документации модуля

Установите Adafruit GFX Library и Adafruit ST7735 and ST7789 Library. Базовый шрифт GFX не содержит кириллицу, поэтому пример использует латинские подписи.

Листинг 33.1: Рисуем только изменившуюся часть полосы

```
// 240x240 SPI ST7789 module with CS pin and 3.3V logic.
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ST7789.h>
Adafruit_ST7789 screen(10, 9, 8); // CS, DC, RST
void setup() {
  Serial.begin(115200);
  SPI.begin(12, -1, 11, 10); // SCK, MISO unused, MOSI, CS
  screen.init(240, 240); screen.setRotation(0);
  screen.fillScreen(ST77XX_BLACK);
  screen.setTextColor(ST77XX_WHITE); screen.setTextSize(2);
  screen.setCursor(12, 20); screen.print("ESP32-S3");
  screen.drawRect(19, 99, 202, 22, ST77XX_WHITE);
}
void loop() {
  static uint32_t last = 0;
  static int oldWidth = 0;
  if (millis() - last < 100) return;
  last = millis();
  int width = analogRead(1) * 200L / 4095;
  uint32_t start = micros();
  if (width > oldWidth)
    screen.fillRect(20 + oldWidth, 100, width - oldWidth, 20, ST77XX_GREEN);
  if (width < oldWidth)
    screen.fillRect(20 + width, 100, oldWidth - width, 20, ST77XX_BLACK);
  oldWidth = width;
  Serial.printf("draw_us:%lu\n", (unsigned long)(micros() - start));
}
```

Сохранённая ширина позволяет закрашивать только изменившийся прямоугольник. Сравните это с `fillScreen()` на каждом шаге: уменьшатся трафик и мерцание. При 40 МГц идеальная передача полного кадра требует не меньше $115200 \cdot 8/40000000 \approx 23$ мс; команды и паузы увеличат время. Нельзя требовать 60 полных кадров/с от такого канала только потому, что CPU работает на 240 МГц.

33.3 DMA, PSRAM и время жизни кадра

DMA разгружает CPU при передаче блоков. Пригодность внешней памяти зависит от конкретного драйвера и режима. PSRAM не делает любой указатель подходящим для любой DMA-передачи. Проверьте требования к размещению и выравниванию; иногда нужен внутренний передаточный буфер, даже когда кадр хранится в PSRAM. Драйвер RGB LCD ESP-IDF предусматривает кадровые и промежуточные bounce-буферы. Пока DMA отправляет область, её нельзя менять без синхронизации. Наш простой GFX-пример не демонстрирует все эти режимы: он сначала даёт измеримый базовый результат.

33.4 От рисунков к LVGL

LVGL добавляет виджеты, стили и ввод. Ей нужны работающий драйвер экрана, функция передачи области и обслуживание таймеров. Сначала получите правильные цвета и геометрию, затем подключайте LVGL выбранной версии: API разных основных версий отличается. Все вызовы интерфейса выполняйте из одной задачи либо под общей блокировкой. Задача датчика передаёт данные очередью. Для RGB-проекта заранее составьте карту выводов: экран занимает много GPIO и может конфликтовать с камерой, памятью и другими периферийными устройствами.

Если не работает

Белый экран: питание, RESET, CS или не тот контроллер. Сдвиг: размеры и смещения матрицы. Неверные цвета: RGB/BGR или порядок байтов. Полосы: питание, скорость либо меняющийся буфер.

Итоги урока

Экран выбирают вместе с интерфейсом и бюджетом памяти. Частичное обновление часто полезнее увеличения частоты; доступность PSRAM и пригодность буфера для DMA проверяют отдельно.

Проверь себя

1. Сколько занимает кадр 320×240 RGB565?
2. Чем SPI-дисплей отличается от RGB-панели?
3. Почему PSRAM не решает все вопросы DMA?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Рассчитайте память двух разрешений RGB565.
2. ★ Проверьте крайние положения полосы.
3. ★★ Измерьте полное и частичное обновление в одинаковых условиях.
4. ★★ Добавьте число и обновляйте только его область.
5. ★★★ Организуйте задачи датчика и экрана с очередью; проверьте отзывчивость.
6. ★★★★ На готовой RGB-плате соберите два виджета LVGL и составьте бюджет памяти, кадра и буферов.

Документация для продолжения Официальное руководство по теме.

Урок 34. USB глубже: CDC, накопитель и роль Host

Цель	различать классы и роли USB и учитывать общий PHY
Опирается на	уроки 19 и 11
Результат	канал команд ping/pong через нативный USB CDC
Понадобится	USB-кабель; для дополнительного опыта — подготовленная host-плата

34.1 Класс и роль — разные понятия

Класс USB описывает данные: HID — ввод, CDC — последовательный канал, MSC — блочное хранилище. Composite-устройство объединяет интерфейсы в одном наборе дескрипторов. VID/PID и строки описания — часть интерфейса устройства, а не случайные числа. В роли Device плата отвечает компьютеру. В роли Host сама перечисляет подключённое устройство и управляет обменом. Поддержка одного класса не означает поддержку любого USB-устройства.
ESP32-S3 поддерживает Full Speed: 12 Мбит/с на линии, а не обещанную скорость файлов. Внутренний PHY общий для USB-OTG и фиксированного USB Serial/JTAG. На обычной плате нельзя получить одновременно независимый TinyUSB и встроенную USB-JTAG-отладку через те же D+/D−. Мост UART остаётся независимым путём прошивки и диагностики.

34.2 Своя консоль TinyUSB CDC

Выберите USB Mode: USB-OTG (TinyUSB), USB CDC On Boot: Disabled. Объект CDC создаёт сам скетч, поэтому автоматический CDC отключён. Подключите нативный USB, откройте появившийся порт и отправьте `ping` с переводом строки. При этих настройках диагностический `Serial` направлен в мост UART. Поэтому в этом опыте к компьютеру подключают *оба* разъёма DevKitC-1.

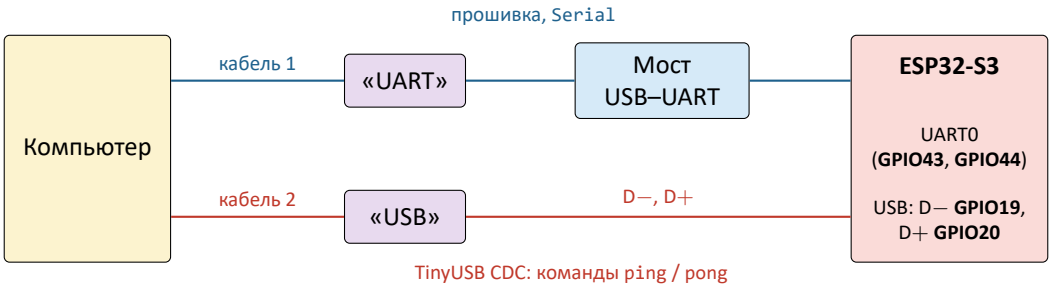


Рис. 34.1: Какой разъём для чего: «UART» через мост даёт прошивку и диагностический `Serial`, «USB» идёт прямо на выводы **GPIO19**/**GPIO20** и несёт TinyUSB CDC

Листинг 34.1: Одна команда на строку через USB CDC

```
// USB Mode: USB-OTG (TinyUSB); USB CDC On Boot: Disabled.
#include <USB.h>
#include <USBCDC.h>
USBCDC console;
void setup() {
  Serial.begin(115200); // bridge UART for diagnostics with CDC on boot disabled
  console.begin();
  USB.begin();
}
void loop() {
```

```
  static char line[32]; static size_t used = 0;
  while (console.available()) {
    char c = console.read();
    if (c == '\n') {
      line[used] = 0;
      if (strcmp(line, "ping") == 0) console.println("pong");
      else console.println("unknown command");
      used = 0;
    } else if (c != '\r') {
      if (used < sizeof(line)-1) line[used++] = c;
      else { used = 0; console.println("line too long"); }
    }
  }
  delay(1);
}
```

CDC передаёт поток байтов: команда может прийти частями, несколько команд — одним чтением. Поэтому выделяем строку самостоятельно и ограничиваем её длину. Значение baud rate виртуального CDC не задаёт физическую скорость USB так же, как у UART. Опишите кодировку, завершение строки, максимальную длину и ответы на неизвестные команды. Проверьте отключение и повторное подключение: программа не должна бесконечно ждать открытого монитора.

34.3 MSC: кто владеет файловой системой

Накопитель предъявляет хосту сектора, а не готовые файлы. FAT и каталоги обслуживает хост или файловая библиотека. Если компьютер и ESP одновременно изменяют одну файловую систему, их кэши могут разойтись и повредить данные. Назначьте одного владельца: пока диск смонтирован на компьютере, приложение туда не пишет; после извлечения владельцем может стать плата. Первый дополнительный опыт делайте с отдельным тестовым RAM-диском, не с картой ученических работ. Пример CDC диски не создаёт и файлы не изменяет.

34.4 Host: переходника мало

Хост подаёт устройству 5 В VBUS через рассчитанную цепь с ограничением тока и не конфликтует с питанием компьютера. USB-переходник сам не добавляет эту схему в DevKitC-1. Используйте host-плату или документированную схему с управляемым питанием. Далее нужны host-стек и драйвер класса: ESP-IDF содержит примеры HID, CDC и MSC. Первый опыт — чтение дескрипторов собственной клавиатуры и вывод событий в UART. Снятие устройства обрабатывается как штатное событие, а не бесконечное ожидание данных.

Если не работает

Порт не появился: разъём, режим или кабель. После прошивки порт изменился: обновите выбор. JTAG пропал при TinyUSB: контроллеры используют общий PHY. Восстанавливайте доступ через мост и BOOT/RST; eFuse для этих опытов не меняют.

Итоги урока

Класс описывает данные, роль — инициатора обмена. CDC требует выделения команд, MSC — одного владельца диска, Host — корректного VBUS и драйвера класса.

Проверь себя

1. Может ли одна команда CDC прийти несколькими чтениями?
2. Почему нельзя одновременно писать на диск с двух сторон?
3. Почему переходника недостаточно для Host?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Укажите роли в парах «компьютер — плата» и «плата — клавиатура».
2. ★ Получите pong на команду ping.
3. ★★ Добавьте uptime с временем в миллисекундах.
4. ★★ Проверьте команды частями и две команды вместе, задайте реакцию на переполнение.
5. ★★★ Составьте автомат владения MSC-диском: приложение, хост, извлечение, возврат.
6. ★★★★ На host-плате прочитайте дескрипторы своей клавиатуры и обработайте отключение без зависания.

Документация для продолжения Официальное руководство по теме. USB Host и драйверы классов.

Урок 35. ESP-IDF и JTAG: увидеть выполнение

Цель	использовать точки останова и воспроизводимую сборку
Опирается на	уроки 14 и 28
Результат	найденная причина целочисленного деления и исправленная функция
Понадобится	ESP-IDF 5.4.x, OpenOCD Espressif, нативный USB

35.1 Когда печати недостаточно

Отладчик позволяет остановиться перед строкой, посмотреть локальные переменные, стек вызовов и пройти код по шагам. Breakpoint задаёт место остановки, watch показывает значение, backtrace — цепочку вызовов. Остановка меняет время работы: сеть может потерять соединение, а часть периферии продолжит работать. Учебный опыт проводим на плате без нагрузки.

35.2 Минимальная сборка ESP-IDF

Arduino вызывает `setup()` и `loop()`, а ESP-IDF начинает код с `app_main()`. Проект имеет корневой CMakeLists, каталог main с регистрацией исходников и sdkconfig. Хотя Arduino-ESP32 использует ESP-IDF, это разные модели сборки: нельзя просто переопределить опцию уже собранного загрузчика строкой в скетче.
В активированном терминале ESP-IDF откройте `examples/debug_mean` из репозитория:

```
idf.py set-target esp32s3
idf.py menuconfig
```

```
idf.py build
idf.py -p COM5 flash monitor
```

Замените COM5 своим портом. Фиксируйте версию среды и sdkconfig рядом с проектом. Файл ELF каждой прошивки нужен для сопоставления машинных адресов и исходника.

35.3 Ошибка с понятным ожиданием

Листинг 35.1: Arduino-вариант: ожидаем 2,5, получаем 2,0

```
// Deterministic logic bug for a debugger: expected 2.5, printed 2.0.
#include <Arduino.h>
volatile float observed = 0;
float meanWrong(int total, int count) {
  return total / count; // set a breakpoint here; inspect operand types
}
void setup() { Serial.begin(115200); }
void loop() {
  observed = meanWrong(5, 2);
  Serial.printf("mean=%.2f\n", (double)observed);
  delay(1000);
}
```

Полный IDF-проект в `examples/debug_mean` содержит ту же функцию в main.c. Вместо Arduino delay используется `vTaskDelay`, вместо Serial — `printf`. При `total=5`, `count=2` возвращается 2,0: оба операнда целые, дробь отбрасывается до преобразования результата в float. Исправьте выражение на `(float)total / count`; отдельно определите поведение при нулевом числе элементов.

35.4 Подключение встроенного USB Serial/JTAG

GPIO19/20 оставьте свободными, в этой прошивке TinyUSB не используется. Подключите нативный USB. Используйте OpenOCD Espressif и настройку доступа/USB-драйвер для своей ОС по официальной инструкции. Обычный системный OpenOCD может не подходить. Запустите `idf.py openocd`; в другом активированном терминале того же проекта — `idf.py gdb`. В GDB выполните `break meanWrong`, `continue`, `print total`, `print count`, `next`, `bt`. Проверьте, на каком этапе теряется дробная часть.

Для отладочного опыта выберите Debug optimization в menuconfig. Оптимизатор может встроить функцию или убрать переменную; отсутствие переменной в watch не доказывает, что её нет в исходнике. После исправления пересоберите, прошейте и отлаживайте именно новый ELF. Не путайте логическую ошибку этого примера с падением: здесь программа исполняется, но результат неверен.

Если не работает

Точка не ставится: другой ELF или оптимизация. Нет подключения: выбран мост вместо нативного USB, не установлен драйвер либо прошивка переключила PHY на USB-OTG. Сначала проверьте устройство и конфигурацию адаптера, затем команды GDB.

Итоги урока

Отладчик показывает причину ошибки, ELF связывает адреса со строками. Остановка меняет временное поведение; воспроизводимый сценарий важнее случайных наблюдений.

Проверь себя

1. Почему float-результат не спасает целочисленное деление?
2. Зачем нужен ELF этой прошивки?
3. Почему breakpoint мешает сетевому обмену?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Подпишите назначение моста и нативного USB.
2. ★ Воспроизведите 2,00 и запишите ожидаемый результат.
3. ★★ Остановитесь в функции и посмотрите аргументы и их типы.
4. ★★ Исправьте деление и задайте поведение при count=0.
5. ★★★ Оформите отчёт: вход, ожидание, факт, стек, причина, правка.
6. ★★★★ Перенесите собственный опыт в ESP-IDF и сохраните версии, sdkconfig, исходники и ELF.

Документация для продолжения Официальное руководство по теме.

Урок 36. ULP: наблюдение, пока основные ядра спят

Цель	собрать проект с программой CPU и программой сопроцессора
Опирается на	уроки 9, 27 и 35
Результат	пробуждение после десяти последовательных нажатых отсчётов
Понадобится	кнопка GPIO4–GND, резистор 10 кОм GPIO4–3V3, ESP-IDF 5.4.x

36.1 Два домена исполнения

Deep-sleep из урока 9 будил CPU по таймеру или внешнему сигналу. ULP может опрашивать поддерживаемые RTC-входы и будить основную программу при выполнении условия. У ESP32-S3 есть ULP FSM и ULP RISC-V; здесь выбираем RISC-V и язык C. Это отдельный маломощный сопроцессор, а не третье ядро LX7. Он не запускает Arduino, Wi-Fi и обычные задачи FreeRTOS. Набор доступной периферии проверяют отдельно.

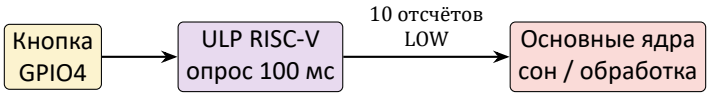


Рис. 36.1: Простой частый опрос отделён от сложного редкого действия

Кнопка подключена к RTC-входу **GPIO4**: внешний резистор 10 kΩ подтягивает вход к 3V3, кнопка замыкает его на GND. Встроенную подтяжку для этого входа в коде отключают, поэтому внешний резистор обязателен.

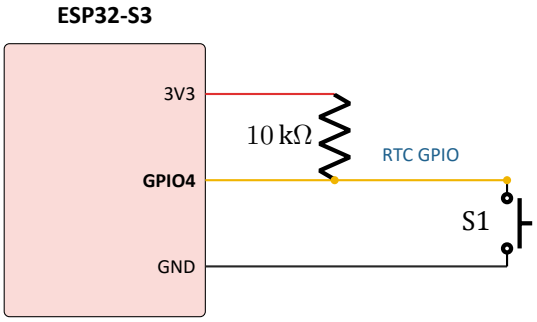


Рис. 36.2: Подключение кнопки к **GPIO4**: внешняя подтяжка 10 kΩ к 3V3 и кнопка на GND. Светодиод с этого вывода нужно снять

36.2 Проект с двумя образами

Основной код компилируется для Xtensa, ULP — для RISC-V. CMake встраивает маленький образ в основную прошивку и создаёт заголовок общих символов. Полный проект `examples/ulp_button` содержит корневой CMakeLists, `main/CMakeLists.txt`, `main/main.c`, `main/ulp/main.c` и sdkconfig.defaults. В defaults включены ULP RISC-V и резерв RTC-памяти. Открывать эти файлы как скетч .ino нельзя. Сборка: `idf.py set-target esp32s3`, `idf.py build`, `idf.py -p COM5 flash monitor`; замените порт своим.

36.3 Что делает основная программа

GPIO4 настраиваем как RTC-вход, LED с него снимаем. Внешний резистор задаёт HIGH, кнопка замыкает на GND. Программа ждёт секунду для наблюдения лога, загружает ULP, задаёт период 100000 мкс, разрешает ULP-пробуждение и засыпает.

Листинг 36.1: Основные ядра: загрузить ULP и уснуть

```
#include <stdio.h>
#include "esp_sleep.h"
#include "driver/rtc_io.h"
#include "ulp_riscv.h"
#include "ulp_main.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
extern const uint8_t ulp_main_bin_start[] asm("_binary_ulp_main_bin_start");
extern const uint8_t ulp_main_bin_end[] asm("_binary_ulp_main_bin_end");
void app_main(void) {
    vTaskDelay(pdMS_TO_TICKS(1000));
    printf("wake=%d\n", (int)esp_sleep_get_wakeup_cause());
    ESP_ERROR_CHECK(rtc_gpio_init(GPIO_NUM_4));
    ESP_ERROR_CHECK(rtc_gpio_set_direction(GPIO_NUM_4, RTC_GPIO_MODE_INPUT_ONLY));
    ESP_ERROR_CHECK(rtc_gpio_pullup_dis(GPIO_NUM_4));
    ESP_ERROR_CHECK(rtc_gpio_pulldown_dis(GPIO_NUM_4));
    // External 10k pull-up: GPIO4 -> resistor -> 3V3; button GPIO4 -> GND.
    ESP_ERROR_CHECK(ulp_riscv_load_binary(ulp_main_bin_start,
                        ulp_main_bin_end - ulp_main_bin_start));
    ESP_ERROR_CHECK(ulp_set_wakeup_period(0, 100000));
    ESP_ERROR_CHECK(esp_sleep_enable_ulp_wakeup());
    ESP_ERROR_CHECK(ulp_riscv_run());
    printf("Hold button about one second to wake\n");
    fflush(stdout);
    vTaskDelay(pdMS_TO_TICKS(100));
    esp_deep_sleep_start();
}
```

36.4 Программа сопроцессора

Листинг 36.2: ULP: десять последовательных отсчётов LOW

```c
#include <stdint.h>
#include "ulp_riscv.h"
#include "ulp_riscv_utils.h"
#include "ulp_riscv_gpio.h"
uint32_t consecutive = 0; // RTC slow memory, retained between timer starts
int main(void) {
    if (ulp_riscv_gpio_get_level(GPIO_NUM_4) == 0) ++consecutive;
    else consecutive = 0;
    if (consecutive >= 10) {
        consecutive = 0;
        ulp_riscv_wakeup_main_processor();
    }
    return 0; // automatic halt; next timer period starts another scan
}
```

Счётчик в RTC slow memory сохраняется между запусками ULP по таймеру. HIGH сбрасывает его, LOW увеличивает; на десятом отсчёте запрашиваем пробуждение. Выход из main завершает текущий опрос до следующего периода. Удержание длится примерно секунду, но результат зависит от фазы опроса: это не точное измерение времени. После пробуждения отпустите кнопку в течение секундной паузы, иначе следующий цикл снова обнаружит нажатие.

36.5 Проверка экономии

ULP полезен, если предотвращает достаточно дорогие пробуждения CPU. Сравните активный опрос, пробуждение CPU каждые 100 мс и ULP с редким wakeup. Измеряйте ток и длительность каждого режима. На DevKit стабилизатор, индикатор питания и USB-мост могут перекрыть выгоду чипа. Для оценки автономности нужен ток всей целевой платы, включая датчики, а не только значение deep-sleep из datasheet.

Если не работает

Не создан заголовок ULP: проверьте CMake и конфигурацию. Нет wakeup: не тот RTC-вход или не разрешён источник. Постоянное пробуждение: удерживается кнопка либо плавает вход. Полные исходники приведены, но сборка ESP-IDF и измерения на плате в этой среде не выполнялись.

Итоги урока

CPU и ULP имеют отдельные программы и ресурсы. Начинайте с условия пробуждения, периода опроса и измеримого бюджета энергии.

Проверь себя

1. Почему ULP не собирается обычным Arduino-скетчем?
2. Равны ли десять отсчётов точной секунде?
3. Почему ULP почти не меняет ток некоторых DevKit?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выделите на схеме работающие во сне блоки.
2. ★ Соберите проект и подтвердите источник wakeup в логе.
3. ★★ Сравните пороги 10 и 5 при одном периоде.
4. ★★ Добавьте общий счётчик опросов и прочитайте после wakeup.
5. ★★★ Сравните заряд за час в трёх режимах при одинаковых событиях.
6. ★★★★ Замените кнопку поддерживаемым ULP-датчиком и добавьте порог с гистерезисом.

Документация для продолжения Официальное руководство по теме.

Урок 37. Распознавание на устройстве: данные, модель и ESP-DL

Цель	различать обучение и инференс и измерять качество
Опирается на	уроки 5, 20, 26 и 35
Результат	классификатор двух положений ручки на собственных данных
Понадобится	потенциометр GPIO1; для расширения — IMU или камера

37.1 Что даёт именно ESP32-S3

LX7 имеет векторные инструкции для ускорения некоторых вычислений; PSRAM полезна для моделей и изображений. Но произвольный цикл C++ не ускорится от установки библиотеки: нужны поддерживаемые операции и оптимизированные ядра. ESP-DL — библиотека Espressif для инференса. Проверяйте совместимые модели, конвертер и версию ESP-IDF по конкретному выпуску.

Обучение настраивает параметры по примерам, инференс применяет их к новому входу. Начнём с маленькой модели: среднее измерение для каждого из двух классов. Она не является нейросетью и не использует векторное ускорение, но показывает весь путь от данных к предсказанию, включая ошибки и неизвестный класс.

37.2 Обучение на своих измерениях

Определите классы 0 и 1 как два диапазона положения ручки. Подключите потенциометр как в уроке 5. Поставьте ручку в первый диапазон и отправьте 0, затем во второй и отправьте 1. На класс собирается 100 отсчётов; слегка меняйте положение внутри выбранного диапазона.

Листинг 37.1: Обучение двух центров и отказ от далёкого входа

```
// Train two one-dimensional centroids using your own potentiometer samples.
#include <Arduino.h>
float center[2] = {0, 0};
bool trained[2] = {false, false};
void train(int label) {
  uint32_t sum = 0;
  for (int i = 0; i < 100; ++i) { sum += analogRead(1); delay(5); }
  center[label] = sum / (100.0f * 4095.0f);
  trained[label] = true;
  Serial.printf("class %d center=%.3f\n", label, center[label]);
}
void setup() {
  Serial.begin(115200);
  Serial.println("Place knob at class 0, send 0; then at class 1, send 1");
}
void loop() {
  if (Serial.available()) {
    char command = Serial.read();
    if (command == '0' || command == '1') train(command - '0');
  }
  if (!trained[0] || !trained[1]) { delay(20); return; }
  float x = analogRead(1) / 4095.0f;
  float d0 = fabsf(x - center[0]), d1 = fabsf(x - center[1]);
  // Reject samples far from both learned classes; this is not a probability.
  int label = min(d0, d1) > 0.12f ? -1 : (d0 < d1 ? 0 : 1);
  Serial.printf("x:%.3f class:%d\n", x, label);
  delay(200);
}
```

Нормализация приводит отсчёт к масштабу 0–1. Модель выбирает ближайший центр. Если оба далеко, возвращает −1 («неизвестно»). Порог 0,12 учебный, подбирайте его на отдельных данных. Расстояние — не вероятность. После сброса центры исчезают: сохранение обученной модели в NVS — отдельная задача с проверкой версии формата.

37.3 Как проверить честно

После обучения заново установите ручку в тестовые положения: те же 100 отсчётов не являются независимым тестом. Для жестов разделяйте целые записи и сеансы, а не соседние отсчёты движения: иначе оценка будет завышена из-за похожих данных. Включите промежуточные положения и фон. Составьте таблицу истинного и предсказанного класса, включая «неизвестно». Считайте не только долю успехов, но и ложные срабатывания. Сравните с простым порогом: модель оправдана, если улучшает решение конкретной задачи.

37.4 Переход к IMU, камере и ESP-DL

Для IMU выделяют окно, затем признаки: среднее, разброс, пики, энергию. Для камеры фиксируют размер входа, порядок каналов и нормализацию, одинаковые при обучении и применении. Квантование int8 может уменьшить память, но качество проверяют после преобразования. Сквозной путь с настоящей нейросетью, квантованием и измерениями разобран в уроке 42. Выберите официальный пример ESP-DL для ESP32-S3, зафиксируйте версию и рекомендованный конвертер. Не смешивайте произвольную модель с несовместимым форматом исполнения.
Память нужна не только весам: входы, промежуточные тензоры, кадры, стек и сеть тоже занимают место. Измеряйте отдельно сбор данных, предобработку, инференс и реакцию. Команда

мотору требует подтверждения или нескольких устойчивых совпадений: единичная ошибка распознавания не должна немедленно запускать движение.

Если не работает

Один класс на всех входах: центры обучены рядом. Качество только дома: нет других условий в данных. Тормозит камера: причина может быть в копировании и предобработке, а не модели.

Итоги урока

Качество измеряют на новых данных вместе с ошибками, задержкой и памятью. Аппаратная возможность ускорения не заменяет измерения конкретного приложения.

Проверь себя

1. Чем обучение отличается от инференса?
2. Почему соседние отсчёты жеста нельзя случайно делить между обучением и тестом?
3. Почему размер модели меньше памяти приложения?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Опишите два класса так, чтобы другой ученик повторил опыт.
2. ★ Обучите центры и проверьте по пять новых положений.
3. ★★ Проверьте десять промежуточных положений и настройте отказ.
4. ★★ Сохраните модель в NVS с номером формата.
5. ★★★ Соберите жесты IMU из разных сеансов и сравните модель с порогами.
6. ★★★★ Запустите официальный пример ESP-DL и опишите версии, вход, память, задержку и матрицу ошибок.

Документация для продолжения Официальное руководство по теме.

Урок 38. От макетки к устройству: запуск, восстановление и защита

Цель	проверить разметку Flash и предусмотреть потерю сети и сбои
Опирается на	уроки 11, 13, 14, 27 и 35
Результат	паспорт прошивки и проверяемый сценарий восстановления
Понадобится	плата, тестовая сеть, резервная прошивка

38.1 Запуск без компьютера

Устройство должно стартовать самостоятельно, переживать потерю сети и объяснять состояние. Определите безопасные уровни выходов до настройки периферии. Отсутствие Wi-Fi не должно блокировать кнопку выключения или контроль температуры. Сначала локальное управление, затем сетевые возможности и интерфейс наблюдения.

38.2 Flash и OTA планируют вместе

Для стандартной двухслотовой схемы нужны `ota_0`, `ota_1` и `otadata`. Каждый слот должен вмещать новую прошивку. Большой слот Huge APP может означать отсутствие OTA: проверяйте реальную таблицу, не только название меню. Изменение разметки затрагивает данные; сначала сохраните настройки и рабочий образ восстановления.

Листинг 38.1: Разделы и здоровье памяти: только чтение

```
// Read-only inspection: no flash erase and no eFuse programming.
#include <Arduino.h>
#include "esp_ota_ops.h"
#include "esp_heap_caps.h"
#include "esp_system.h"
void setup() {
  Serial.begin(115200); delay(1000);
  const esp_partition_t* running = esp_ota_get_running_partition();
  const esp_partition_t* next = esp_ota_get_next_update_partition(nullptr);
  Serial.printf("reset=%d app=%s size=%lu next_OTA=%s\n",
    (int)esp_reset_reason(), running ? running->label : "?",
    running ? (unsigned long)running->size : 0UL,
    next ? next->label : "none");
  Serial.printf("Flash=%lu PSRAM=%lu IDF=%s\n",
    (unsigned long)ESP.getFlashChipSize(),
    (unsigned long)ESP.getPsramSize(), esp_get_idf_version());
}
void loop() {
  Serial.printf("heap=%u largest_internal=%u minimum=%u\n",
    (unsigned)esp_get_free_heap_size(),
    (unsigned)heap_caps_get_largest_free_block(MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT),
    (unsigned)esp_get_minimum_free_heap_size());
  delay(5000);
}
```

Следующий OTA-слот сам не включает rollback. Для отката нужен загрузчик с соответствующей опцией и самопроверка новой прошивки с подтверждением. В опыте конфигурацию загрузчика и eFuse не меняем. Паспорт включает версии, разметку, размер Flash/PSRAM и причину сброса.

38.3 Provisioning: как выдать учётные данные

Первичная настройка возможна через USB, временную точку доступа или BLE. У временного канала нужны явное включение кнопкой, ограниченное время жизни и способ подтвердить владельца, например индивидуальный секрет. Сохраняйте параметры через Preferences, пароль в журнал не печатайте. Полный возврат к настройке запускайте осмысленным действием, а не обычной потерей Wi-Fi: временный сбой не должен стирать рабочие настройки.

38.4 Автомат сети и повтор команд

Задайте состояния «подключение», «работа», «пауза перед повтором», ограничьте тайм-ауты и буферы. Пауза с увеличением интервала и случайной добавкой предотвращает одновременные запросы всех узлов после восстановления роутера. Не накапливайте сообщения без ограничения памяти. Повтор «установить яркость 30» сохраняет результат, а повтор «переключить» отменяет предыдущее действие. Номер команды и подтверждение помогают обнаружить дубли. Возвращение Wi-Fi ещё не означает восстановление MQTT-подписки: проверяйте каждый уровень связи.

38.5 Механизмы защиты решают разные задачи

HTTPS защищает канал и проверку сервера, аутентификация приложения — право дать команду. Secure Boot проверяет подпись запускаемого кода, Flash Encryption защищает содержимое внешней Flash в рамках своей модели угроз. Они не заменяют хранение ключей и план обновления. Часть eFuse необратима: сначала изучите процесс по документации, а внедрение выполняйте на выделенной плате с подготовленными ключами и планом восстановления.

38.6 Приёмочные испытания

Таблица испытания содержит исходное состояние, действие, ожидание, факт и время восстановления. Проверьте старт без USB, пропажу сети, неверный адрес сервера, отсутствие датчика, заполненную очередь и повторную загрузку настроек. За час сравните кучу и крупнейший свободный блок: достаточная сумма памяти не исключает фрагментацию. Сбой питания моделируют на подготовленном стенде с копией данных и штатным способом восстановления. Сначала убедитесь, что известны рабочий образ, порт, настройки загрузчика и порядок возврата.

Если не работает

После смены разметки нет настроек: раздел перенесён или стёрт. Цикл перезагрузок: проверьте причину сброса и минимальную самопроверку. Сеть вернулась, устройство молчит: не восстановлены соединение приложения или подписки.

Итоги урока

Надёжность описывается реакцией на отказ и временем восстановления. Память, сеть, ключи, выходы и журнал проектируют вместе, проверяя реальную работу без компьютера.

Проверь себя

1. Почему OTA-слота недостаточно для отката?
2. Чем Secure Boot отличается от Flash Encryption?
3. Чем повтор «переключить» отличается от «установить 30»?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выпишите версию ядра, Flash, PSRAM и схему разделов.
2. ★ Найдите в паспорте текущий раздел и следующий OTA-слот.
3. ★★ Составьте автомат подключения с тайм-аутом и локальным управлением.
4. ★★ Трижды восстановите Wi-Fi и измерьте худшую задержку.
5. ★★★ Спроектируйте настройку через USB/BLE/AP с подтверждением владельца и сроком действия.
6. ★★★★ Подготовьте протокол десяти отказов и стратегию обновления: подпись, самопроверка, откат, копия и восстановление.

Документация для продолжения Официальное руководство по теме. Secure Boot v2. Flash Encryption.

Урок 39. ESPUI: веб-панель из программы C++

Цель	собрать веб-интерфейс из готовых элементов и связать его с состоянием устройства
Опирается на	уроки 7 (очереди), 8 (HTTP, HTML, CSS), 18 (JSON), 38 (надёжность)
Результат	панель с переключателем LED, ползунком яркости, вкладками и показаниями АЦП
Понадобится	ESPUI и её зависимости; LED с резистором на GPIO4, потенциометр на GPIO1

39.1 Зачем библиотека, если уже знаем HTML

В уроке 8 мы сами описывали страницу, стили и JavaScript. ESPUI позволяет создавать типовые элементы из C++: подпись, кнопку, переключатель, ползунок, поле ввода и вкладку. Библиотека обслуживает страницу и обмен с браузером. Так удобно быстро сделать учебный стенд, настройку датчика или диагностическую панель. Для нестандартного дизайна и собственного протокола полезно сохранить подход урока 8: знание HTML/CSS не становится лишним.
Первую страницу браузер получает по HTTP, дальнейшие события и обновления идут через WebSocket. Соединение двустороннее: браузер отправляет команду, плата может сообщить новое показание, не дожидаясь перезагрузки страницы. Наличие страницы ещё не доказывает работу WebSocket: если кнопки не действуют, проверяйте отдельно канал событий.

39.2 Установка: проверяем комплект зависимостей

Примеры главы скомпилированы для ESP32-S3 с Arduino-ESP32 3.2.0. Проверяемые версии зависимостей приведены ниже; это комплект для воспроизводимого опыта, а не требование всегда использовать только эти выпуски.

Библиотека	Версия	Назначение
ESPUI	2.2.4	элементы интерфейса и связь с браузером
ESPAsyncWebServer (ESP32Async)	3.12.1	HTTP, WebSocket, обслуживание клиентов
AsyncTCP (ESP32Async)	3.5.0	асинхронный TCP для ESP32
ArduinoJson	7.4.2	представление сообщений

В менеджере Arduino IDE установите ESPUI, ArduinoJson и зависимости из семейства ESP32Async. У сетевых библиотек встречаются старые копии с теми же заголовками: смотрите в журнале сборки, какой каталог действительно выбран. Не держите несколько конкурирующих версий. На ESP32-S3 нужен AsyncTCP, а не ESPAsyncTCP для ESP8266. LittleFS уже входит в используемое ядро; отдельная старая LittleFS-библиотека здесь не требуется. Исходники и точные Git-коммиты проверенного набора записаны в `examples/web-panels/README.md`.

39.3 Первая страница без роутера и файловой системы

Начнём с точки доступа: так ошибка пароля домашней сети не мешает изучению интерфейса. Загрузите скетч, подключите телефон к сети `S3-ESPUI` с учебным паролем `lesson-39-demo`. Откройте в обычном браузере адрес, напечатанный в Serial Monitor, например `http://192.168.4.1/`. Если телефон пишет «без интернета», останьтесь в этой сети. Для собственной платы задайте индивидуальный пароль.

Листинг 39.1: ESPUI: счётчик времени работы

```
// ESPUI 2.2.4: resources served from firmware; no filesystem upload.
#include <WiFi.h>
#include <ESPUI.h>
uint16_t uptimeId;
bool ready = false;
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  // Classroom demonstration password. Change for your own board.
  if (!WiFi.softAP("S3-ESPUI", "lesson-39-demo")) {
    Serial.println("AP failed"); return;
  }
  uptimeId = ESPUI.label("Uptime, s", ControlColor::Peterriver, "0");
  ESPUI.begin("My first ESPUI");
  ready = true;
  Serial.println(WiFi.softAPIP());
}
void loop() {
  static uint32_t last = 0;
  if (ready && millis() - last >= 1000) {
    last = millis();
    ESPUI.updateLabel(uptimeId, String(millis() / 1000));
  }
  delay(1);
}
```

`ESPUI.label()` создаёт элемент и возвращает его **ID**. Сохраняем ID в `uint16_t`, чтобы затем обновлять именно этот элемент. `ESPUI.begin()` запускает обслуживание страницы; её ресурсы берутся из прошивки. Загрузка папки `data` во Flash для этого примера не нужна. В `loop()` раз в секунду передаём новое значение. В отличие от обычного WebServer здесь не нужен вызов `handleClient()`: сетевой стек асинхронный. Но собственная программа всё равно должна уступать CPU.

39.4 Элемент, значение и событие

У элемента есть ID, подпись, цвет, значение и при необходимости обработчик события. ID не является номером GPIO: он обозначает виджет, который может управлять любым объектом. Обработчик вида `void callback(Control* sender, int event)` получает источник события и его тип. Значение доступно как `sender->value`, то есть строка.

Элемент	Что происходит	Как обновляем из программы
Label	показывает состояние или измерение	`updateLabel(id, text)`
Button	`B_DOWN` при нажатии, `B_UP` при отпускании	обработать выбранное событие один раз
Switcher	`S_ACTIVE` или `S_INACTIVE`	`updateSwitcher(id, on)`
Slider	`SL_VALUE`, значение читается из строки	`updateSlider(id, value)`
Tab	группирует дочерние элементы	задаём родителя при создании

Для действия «выключить» достаточно обработать одно событие кнопки, например отпускание. Если выполнять переключение и на нажатии, и на отпускании, один щелчок вернёт исходное состояние. Для удерживаемого управления нужна другая модель: команда старта, команда остановки и тайм-аут на случай потери клиента. Отпускание в браузере не является аппаратной аварийной остановкой.

39.5 Каталог виджетов

Сначала посмотрим, что вообще умеет библиотека. Каждый виджет — это либо *вход* (пользователь что-то сообщает плате), либо *выход* (плата что-то показывает), либо украшение. Заголовок и цвет у всех одинаковые, различаются значение и событие.

Виджет	Тип события	Что приходит и как показывать	Для чего
Label	—	`updateLabel(id, text)`	число, состояние, ошибка
Button	`B_DOWN`, `B_UP`	нажали / отпустили; текст кнопки в `value`	действие один раз
Switcher	`S_ACTIVE`, `S_INACTIVE`	`updateSwitcher(id, on)`	вкл/выкл режима
Slider	`SL_VALUE`	число в строке; границы задают `Min`, `Max`	яркость, скорость
Number	`N_VALUE`	число; границы тоже `Min`, `Max`	угол, порог, период
Text	`T_VALUE`	произвольная строка — её надо проверять	имя, пароль, команда
Select	`S_VALUE`	в `value` лежит значение выбранного `Option`	режим, нота, профиль
Pad / PadWithCenter	`P_FOR_DOWN`, `P_FOR_UP`, …	пять кнопок со стрелками (с центром — шесть)	движение, меню
Gauge	—	`updateControlValue(id, строка)`, `Min`, `Max`	круговая шкала
Graph	—	`addGraphPoint(id, число)`, `clearGraph(id)`	показания за время
Separator	—	только подпись	разделить группы
Accelerometer	—	датчик наклона самого телефона	управление наклоном

Цвет задаётся одним из восьми имён: `Turquoise`, `Emerald`, `Peterriver`, `Wetasphalt`, `Sunflower`, `Carrot`, `Alizarin`, `Dark`. Они не несут смысла сами по себе, поэтому договоритесь заранее: например, красный (`Alizarin`) — только для опасных действий и ошибок, зелёный (`Emerald`) — для включения, синий (`Peterriver`) — для чисел и измерений. Тогда по цвету видно, что делает элемент, как на цветовой схеме книги.

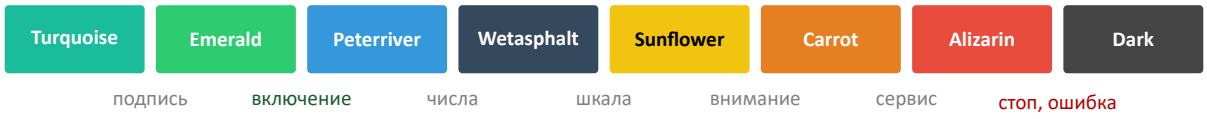


Рис. 39.1: Восемь цветов ESPUI. Закрепите за ними роли, например: красный — «стоп» и ошибки, зелёный — включение, синий — числа

39.6 Как выглядит панель

Программа создаёт элементы по порядку, и страница выстраивает их в колонку карточек. Вкладки (`Tab`) становятся кнопками вверху, а у каждого элемента `parentControl` говорит, на какой вкладке он живёт. Ниже — схематичный вид панели из примера `S3-Widgets`: слева вкладка «Input» с полями ввода, справа вкладка «Output» с показаниями.

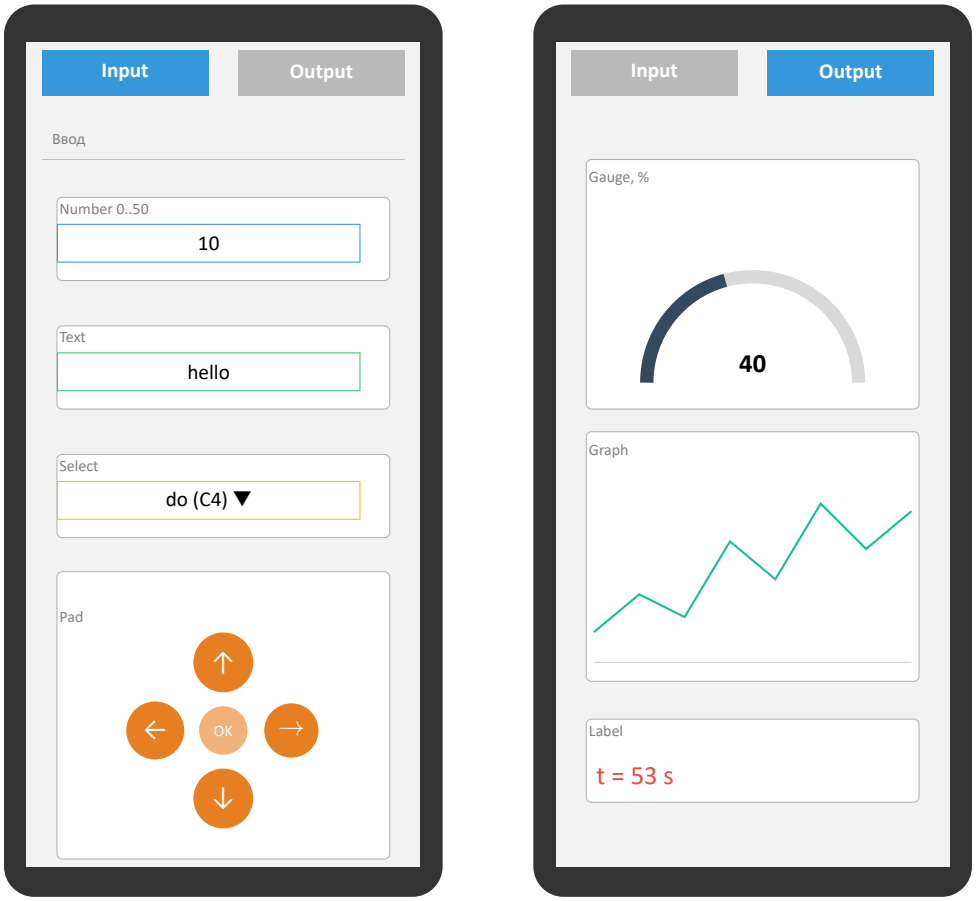


Рис. 39.2: Схема панели `S3-Widgets`: у каждого элемента подпись сверху и значение под ней; вкладки переключаются нажатием

Размеры карточек и порядок задаёт библиотека, поэтому в интерфейсе не нужно думать о координатах. Если на странице не хватает места, широкую карточку можно сделать функцией `setPanelWide(id, true)`, а набор кнопок разложить в строку через `setVertical(id, false)`.

39.7 Gauge и Graph: вольтметр на телефоне

В уроке 5 мы читали потенциометр и выводили число в Serial Monitor. Покажем то же самое стрелкой и графиком. Подключение прежнее: крайние выводы потенциометра к 3V3 и GND, движок к GPIO1.

Листинг 39.2: ESPUI: вольтметр со шкалой и графиком

```
// Потенциометр: крайние выводы к 3V3 и GND, движок -> GPIO1 (ADC1).
#include <WiFi.h>
#include <ESPUI.h>

const int POT_PIN = 1;
uint16_t gaugeId, graphId, valueId;

int readMilliVolts() {                          // среднее из 16 замеров гасит шум
  uint32_t sum = 0;
  for (int i = 0; i < 16; i++) sum += analogReadMilliVolts(POT_PIN);
  return sum / 16;
}

void setup() {
```

```
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Voltmeter", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  gaugeId = ESPUI.addControl(ControlType::Gauge, "Напряжение, мВ", "0",
↪   ControlColor::Wetasphalt);
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, gaugeId);
  ESPUI.addControl(ControlType::Max, "", "3300", ControlColor::None, gaugeId);
  valueId = ESPUI.label("Значение", ControlColor::Peterriver, "0 мВ");
  graphId = ESPUI.graph("Последние отсчёты", ControlColor::Emerald);
  ESPUI.begin("ESP32-S3 voltmeter");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static uint32_t last = 0;
  if (millis() - last >= 500) {                    // два обновления в секунду достаточно глазу
    last = millis();
    int mv = readMilliVolts();
    ESPUI.updateControlValue(gaugeId, String(mv));
    ESPUI.updateLabel(valueId, String(mv) + " мВ");
    ESPUI.addGraphPoint(graphId, mv);
  }
  delay(5);
}
```

Что здесь нового:

- у Gauge границы шкалы создаются дочерними элементами Min и Max; значение обновляется через updateControlValue — это строка, поэтому число оборачиваем в String();
- точки Graph добавляет addGraphPoint; график хранит ограниченное число последних отсчётов. Это окно просмотра, а не журнал: историю сохраняйте отдельно;
- чтение АЦП усредняется по 16 замерам, как в уроке 5, и выполняется два раза в секунду. Страница не должна обновляться чаще, чем глазу нужно.

Совет

Если стрелка Gauge «упирается» в край, проверьте Min и Max: библиотека не отсекает значения сама. Напряжение выше 3,3 В на вывод подавать нельзя ни при какой шкале.

39.8 Ввод: числа, текст, списки и крестовина

Скетч S3-Widgets не управляет внешними деталями, он нужен, чтобы увидеть все входные виджеты и их события. Нажимайте и смотрите Serial Monitor: для каждого события печатается строка.

Листинг 39.3: ESPUI: все входные виджеты в одном скетче

```
// Каталог виджетов ESPUI: только панель, без внешних деталей (нужен лишь Serial Monitor).
#include <WiFi.h>
#include <ESPUI.h>

uint16_t numberId, textId, selectId, gaugeId, graphId, padId, echoId;

void numberCallback(Control* sender, int event) {
  if (event != N_VALUE) return;
  Serial.printf("Number = %d\n", sender->value.toInt());
}
```

```
void textCallback(Control* sender, int event) {
  if (event != T_VALUE) return;
  Serial.printf("Text = '%s'\n", sender->value.c_str());
}
void selectCallback(Control* sender, int event) {
  if (event != S_VALUE) return;
  Serial.printf("Select = %s\n", sender->value.c_str());
}
void padCallback(Control*, int event) {
  switch (event) {                                  // пары «нажали / отпустили»
    case P_FOR_DOWN:   Serial.println("вперёд: нажата");  break;
    case P_FOR_UP:     Serial.println("вперёд: отпущена"); break;
    case P_BACK_DOWN:  Serial.println("назад: нажата");   break;
    case P_LEFT_DOWN:  Serial.println("влево: нажата");   break;
    case P_RIGHT_DOWN: Serial.println("вправо: нажата");  break;
    case P_CENTER_DOWN: Serial.println("центр: нажата");  break;
  }
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Widgets", "lesson-39-demo")) { Serial.println("AP failed"); return; }

  ESPUI.sliderContinuous = false;
  auto inputs  = ESPUI.addControl(ControlType::Tab, "", "Input");
  auto outputs = ESPUI.addControl(ControlType::Tab, "", "Output");

  ESPUI.addControl(ControlType::Separator, "Ввод", "", ControlColor::None, inputs);
  numberId = ESPUI.addControl(ControlType::Number, "Number 0..50", "10", ControlColor::Peterriver,
↪  inputs, numberCallback);
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, numberId);
  ESPUI.addControl(ControlType::Max, "", "50", ControlColor::None, numberId);
  textId = ESPUI.addControl(ControlType::Text, "Text", "hello", ControlColor::Emerald, inputs,
↪  textCallback);
  selectId = ESPUI.addControl(ControlType::Select, "Select", "do", ControlColor::Sunflower, inputs,
↪  selectCallback);
  ESPUI.addControl(ControlType::Option, "do (C4)", "do", ControlColor::None, selectId);
  ESPUI.addControl(ControlType::Option, "re (D4)", "re", ControlColor::None, selectId);
  ESPUI.addControl(ControlType::Option, "mi (E4)", "mi", ControlColor::None, selectId);
  padId = ESPUI.addControl(ControlType::PadWithCenter, "Pad", "", ControlColor::Carrot, inputs,
↪  padCallback);

  gaugeId = ESPUI.addControl(ControlType::Gauge, "Gauge, %", "0", ControlColor::Wetasphalt, outputs);
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, gaugeId);
  ESPUI.addControl(ControlType::Max, "", "100", ControlColor::None, gaugeId);
  graphId = ESPUI.addControl(ControlType::Graph, "Graph", "", ControlColor::Turquoise, outputs);
  echoId  = ESPUI.addControl(ControlType::Label, "Label", "—", ControlColor::Alizarin, outputs);
  ESPUI.begin("ESP32-S3 widgets");
  Serial.println(WiFi.softAPIP());
}

void loop() {
  static uint32_t last = 0;
  static int phase = 0;
  if (millis() - last >= 1000) {
    last = millis();
    phase = (phase + 10) % 110;                     // «пила» 0..100 вместо датчика
    ESPUI.updateControlValue(gaugeId, String(phase));
    ESPUI.addGraphPoint(graphId, phase);
    ESPUI.updateLabel(echoId, "t = " + String(millis() / 1000) + " s");
  }
  delay(5);
}
```

Обратите внимание на четыре правила:

1. **Text хранит строку, а не число.** Поле можно очистить или вставить в него что угодно. Всё, что приходит из `Text`, проверяйте так же строго, как команды из Serial в уроке 2.
2. **Number и Slider отдают число строкой**, поэтому `toInt()` вернёт 0 на мусоре. Для настоящих «нулей» проверяйте длину и символы, как в панели урока.
3. **Select**: сначала создаём сам список, затем `Option` с `parentControl`, равным его ID. Значение, которое придёт в `sender->value`, — второй аргумент `Option`.
4. **Pad** шлёт пары «нажали–отпустили». Если по нажатию мотор едет, то по отпусканию он обязан остановиться; при потере связи отпускание может не прийти, поэтому нужен таймер.

Это интересно

Раскладка панели не хранится в Flash отдельно: библиотека строит её заново из ваших вызовов `addControl` при каждом запуске. Поэтому порядок создания в `setup()` — это порядок элементов на странице.

39.9 Одна модель состояния для кнопок и браузера

Не поручайте сетевому callback долгие измерения, запись Flash или управление мотором. У асинхронного сервера обработчик может выполняться вне `loop()`. В нашем примере callback проверяет вход и ставит короткую команду в очередь; `loop()` единолично меняет состояние LED и GPIO. Это защищает модель приложения от одновременного изменения разными источниками команд.

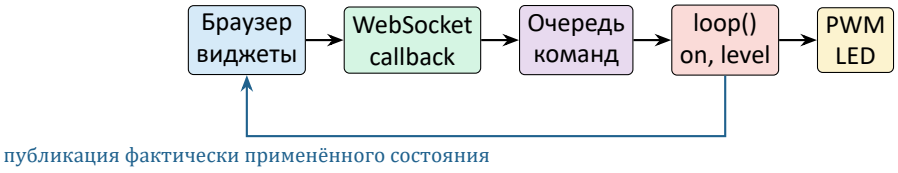


Рис. 39.3: Команда интерфейса и состояние устройства проходят через общую модель

Ползунок ограничен 0–100 и в браузере, и на плате. Строка сначала проверяется на цифры и длину: одного `toInt()` недостаточно, иначе ошибочный текст можно принять за ноль. В очередь копируется число, а не указатель `Control*`, срок жизни которого не стоит переносить за пределы обработчика.

39.10 Полная панель: LED, вкладки и телеметрия

GPIO4 подключите к LED через резистор $220\,\Omega$, катод — к GND. Края потенциометра соедините с 3V3 и GND, движок — с GPIO1. Другие опыты с этими GPIO отключите. После прошивки подключитесь к `S3-Panel`, пароль тот же учебный. Панель имеет вкладки управления и измерений; никаких реальных градусов из потенциометра она не выдумывает: отображаются милливольты АЦП.

Листинг 39.4: ESPUI: панель с очередью команд и двумя вкладками

```
// LED: GPIO4 -> 220 ohm -> LED -> GND. Potentiometer wiper: GPIO1.
#include <WiFi.h>
#include <ESPUI.h>
#include <atomic>

enum class Kind : uint8_t { Power, Level, Stop };
struct Command { Kind kind; int value; };
QueueHandle_t commands;
std::atomic<unsigned> dropped{0};
uint16_t powerId, levelId, voltageId, stateId, queueId;
bool on = false, ready = false;
int level = 40;

void enqueue(Kind kind, int value) {
  Command cmd{kind, value};
  if (xQueueSend(commands, &cmd, 0) != pdTRUE) ++dropped;
}
void powerCallback(Control*, int event) {
  if (event == S_ACTIVE) enqueue(Kind::Power, 1);
  if (event == S_INACTIVE) enqueue(Kind::Power, 0);
}
void levelCallback(Control* sender, int event) {
  if (event != SL_VALUE) return;
  const String text = sender->value;
  if (text.length() == 0 || text.length() > 3) return;
  for (size_t i = 0; i < text.length(); ++i)
    if (text[i] < '0' || text[i] > '9') return;
  int value = text.toInt();
  if (value >= 0 && value <= 100) enqueue(Kind::Level, value);
}
void stopCallback(Control*, int event) {
  if (event == B_UP) enqueue(Kind::Stop, 0); // once per completed click
}
void setup() {
  Serial.begin(115200);
  commands = xQueueCreate(12, sizeof(Command));
  if (!commands || !ledcAttach(4, 5000, 8)) {
    Serial.println("Queue or PWM init failed"); return;
  }
  ledcWrite(4, 0);
  WiFi.mode(WIFI_AP);
  if (!WiFi.softAP("S3-Panel", "lesson-39-demo")) {
    Serial.println("AP failed"); return;
  }
  ESPUI.sliderContinuous = false;
  auto mainTab = ESPUI.addControl(ControlType::Tab, "", "Control");
  auto dataTab = ESPUI.addControl(ControlType::Tab, "", "Measurements");
  powerId = ESPUI.addControl(ControlType::Switcher, "LED enabled", "0",
      ControlColor::Emerald, mainTab, powerCallback);
  levelId = ESPUI.addControl(ControlType::Slider, "Brightness, %",
      String(level), ControlColor::Peterriver, mainTab, levelCallback);
  ESPUI.addControl(ControlType::Min, "", "0", ControlColor::None, levelId);
  ESPUI.addControl(ControlType::Max, "", "100", ControlColor::None, levelId);
  ESPUI.addControl(ControlType::Button, "Output", "All off",
      ControlColor::Alizarin, mainTab, stopCallback);
  stateId = ESPUI.addControl(ControlType::Label, "Applied state", "OFF",
      ControlColor::Turquoise, mainTab);
  voltageId = ESPUI.addControl(ControlType::Label, "ADC1, mV", "0",
      ControlColor::Peterriver, dataTab);
  queueId = ESPUI.addControl(ControlType::Label, "Dropped commands", "0",
      ControlColor::Carrot, dataTab);
```

```
  ESPUI.begin("ESP32-S3 control panel");
  ready = true;
  Serial.println(WiFi.softAPIP());
}
void loop() {
  if (!ready) { delay(10); return; }
  Command cmd;
  // All changes to our application model and GPIO happen here.
  while (xQueueReceive(commands, &cmd, 0) == pdTRUE) {
    if (cmd.kind == Kind::Power) on = cmd.value != 0;
    if (cmd.kind == Kind::Level) level = cmd.value;
    if (cmd.kind == Kind::Stop) on = false;
  }
  ledcWrite(4, on ? level * 255 / 100 : 0);
  static uint32_t last = 0;
  if (millis() - last >= 500) {
    last = millis();
    ESPUI.updateSwitcher(powerId, on);
    ESPUI.updateSlider(levelId, level);
    ESPUI.updateLabel(stateId, on ? "ON" : "OFF");
    ESPUI.updateLabel(voltageId, String(analogReadMilliVolts(1)));
    ESPUI.updateLabel(queueId, String(dropped.load()));
  }
  delay(1);
}
```

39.11 Что проверять после запуска

Включите LED и измените яркость, затем нажмите All off: переключатель и физический выход должны показывать один результат. Откройте страницу во втором браузере. Значения публикуются из общей модели каждые 500 мс, поэтому это не два независимых устройства. При одновременных командах побеждает последняя обработанная команда; права владельца или блокировку второго оператора при необходимости задаёт приложение.

`sliderContinuous=false` уменьшает поток событий: значение отправляется после окончания перемещения. При непрерывном режиме задайте ограничение частоты и правило обработки переполнения. Очередь примера конечна: потерянные команды считаются и видны на вкладке измерений. Периодическая публикация возвращает браузеру применённое состояние, но не превращает потерянную команду в выполненную. Для ответственных операций добавляют номер запроса и подтверждение.

Обновление 2 раза в секунду подходит для учебного АЦП. Не отправляйте новый JSON в каждом проходе `loop()`: это тратит память и канал. Для датчика температуры обычно достаточно более редких обновлений. Сохранение настроек в NVS делайте отдельной кнопкой «Сохранить» или после паузы, а не на каждое движение ползунка. Подпись состояния и признак отсутствующего датчика полезнее ложного нуля.

39.12 Оформление, файлы и доступ

Вкладки и группы создаются через `addControl` с ID родителя. Для небольших изменений оформления есть `setPanelStyle` и `setElementStyle`; внешний вид настраивают после того, как работают значения, ошибки и синхронизация. График — виджет интерфейса, не долговременный журнал: историю измерений храните отдельно, например на SD, в файле или внешней системе. Для ресурсов из файловой системы есть `beginLITTLEFS()`. Подготовка раздела — отдельная операция: `prepareFileSystem()` по умолчанию допускает форматирование. Не вставляйте её бездумно в каждый старт и не проверяйте на разделе с важными файлами. В учебных скетчах главы используется обычный `begin()`, файловая система не меняется.

Пароль точки доступа ограничивает подключение к сети, но не является полной авторизацией команд. Логин страницы также не доказывает защиту всех каналов: в проверенном ESPUI 2.2.4

флаг `WS_AUTHENTICATION` по умолчанию выключен, а установка аутентификации WebSocket зависит от этого флага. Перед применением в изделии отдельно проверьте HTTP, WebSocket, шифрование и права команд. Примеры главы предназначены для локального учебного стенда.

39.13 Те же панели в других уроках

ESPUI особенно хорош как «пульт» для опытов, которые вы уже собрали. В нескольких уроках книги есть врезка «То же на ESPUI»: к знакомой схеме добавляется телефон, а программа датчика или мотора остаётся прежней. Все скетчи следуют правилу этого урока: callback только записывает просьбу, а железо трогает `loop()`.

Урок	Скетч	Что показывает или чем управляет
4 (ШИМ)	`l04_espui_gamma`	Slider яркости, Switcher гаммы, Label с коэффициентом заполнения
5 (АЦП)	`l39_espui_voltmeter`	Gauge и Graph напряжения потенциометра
15 (датчики)	`l15_espui_radar`	Gauge и Graph расстояния HC-SR04, сообщение «нет эха»
16 (мощная нагрузка)	`l16_espui_motor`	Slider скорости и Pad «вперёд/назад» с ограничением времени пуска
17 (адресные светодиоды)	`l17_espui_hue`	два Slider: оттенок и яркость встроенного RGB
24 (звук)	`l24_espui_notes`	Select ноты и Button «Играть»
25 (шаговый двигатель)	`l25_espui_stepper`	Number шага, кнопки «вправо/влево/в начало», Label угла
26 (акселерометр)	`l26_espui_tilt`	Label с «сырым» и фильтрованным углом, Graph наклона

39.14 Какие ещё средства стоит знать

Средство	Подходящая учебная задача	Что проверить перед выбором
ESPUI	управление и диагностика готовыми виджетами	зависимости, поток событий, доступ
ESP-DASH	панель показаний, карточки и графики	различия открытой и Pro-версий
GyverLibs Settings	страница настроек, привязка данных и сохранение	комплект зависимостей, выбранный сервер
ESPAsyncWebServer	собственный HTML/CSS/JS и протокол	самостоятельно реализуем состояние и ввод
WiFiManager	первичная настройка Wi-Fi	это портал настройки, а не замена панели управления

ESP-DASH и Settings дают другой набор готовых компонентов; сравнивайте их на одной и той же маленькой задаче, а не по числу картинок в README. В открытой и коммерческой редакциях могут отличаться возможности и условия использования. Зависимости и API фиксируйте вместе со скетчем. Эти альтернативы здесь рассмотрены для выбора; их отдельные программы не объявляются проверенными.

Если страницу пишете сами, выбирайте канал по задаче: обычный запрос удобен для разовой операции, SSE — для событий от сервера к браузеру, WebSocket — для двустороннего обмена. Любой канал требует понятного поведения при потере связи. WiFiManager разберём следующим уроком.

Если не работает

- Не компилируется сеть — проверьте версии, автора зависимостей и выбранные каталоги.
- Страница есть, кнопки молчат — проверьте WebSocket, события и сохранённые ID.

- Одно нажатие действует дважды — обработаны и B_DOWN, и B_UP.
- Значение «отскакивает» — проверьте очередь, валидацию и источник истинного состояния.
- Телефон уходит из AP — подтвердите работу без интернета и откройте адрес вручную.

Итоги урока

ESPUI ускоряет создание типовой панели. ID обозначает виджет, callback сообщает событие, а модель приложения хранит применённое состояние. Обновления, очередь, сохранение и обработка разрыва связи остаются частью программы устройства.

Проверь себя

1. Чем ID элемента ESPUI отличается от номера GPIO?
2. Почему долгое действие лучше вынести из callback в loop() или отдельную задачу?
3. Почему успешная загрузка страницы и пароль HTTP ещё не доказывают защиту команд?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите минимальную страницу, измените заголовок и найдите ID подписи.
2. ★ Подключите панель и подтвердите, что All off выключает LED и переключатель.
3. ★★ Добавьте кнопку установки яркости 25 процентов через ту же очередь команд.
4. ★★ Подключите датчик температуры из урока 15; показывайте ошибку чтения вместо нуля.
5. ★★★ Сохраните настройки по отдельной кнопке, загрузите после перезапуска и проверьте два браузера.
6. ★★★★ Проверьте потерю связи, переполнение очереди и повтор команд; добавьте номер команды, подтверждение и безопасное состояние для выбранного устройства.

Документация для продолжения ESPUI: API и примеры, ESP32Async, ESP-DASH, Settings.

Урок 40. WiFiManager: настройка сети без перепрошивки

Цель	дать пользователю ограниченный по времени способ настроить Wi-Fi и сохранить локальную работу
Опирается на	уроки 8 (STA/AP), 21 (автоматы), 38 (первичная настройка), 39 (ESPUI)
Результат	портал по удержанию кнопки и подключение к выбранной пользователем сети
Понадобится	WiFiManager 2.0.17, кнопка GPIO5–GND, LED через резистор на GPIO4, сеть 2,4 ГГц

40.1 Панель управления и портал настройки

ESPUI отвечает на вопрос «как управлять устройством». WiFiManager — «как сообщить ему имя и пароль сети, не редактируя исходники». Это разные этапы жизненного цикла. Provisioning нужен при первом запуске, переезде в другую сеть или осмысленном запросе владельца, а обычная панель нужна во время работы.

Портал создаёт временную точку доступа и страницу выбора сети. Captive portal пытается направить подключённый браузер на страницу настройки, но автоматическое окно зависит от телефона и ОС. Всегда оставляйте способ открыть IP вручную. У ESP32-S3 поддерживается Wi-Fi 2,4 ГГц; наличие сети только 5 ГГц не исправляется библиотекой.

40.2 Когда открывать доступ

Не стирайте учётные данные при каждой потере Wi-Fi и не оставляйте открытую AP навсегда. В нашем опыте плата пробует ранее сохранённую сеть. Для настройки пользователь удерживает физическую кнопку 2 секунды; открывается окно с целевым временем 120 секунд. После отпускания кнопки можно запросить новую попытку. Пароль `setup-40-demo` учебный: на собственных платах задавайте индивидуальный и сообщайте владельцу отдельно.

Пароль AP и пароль выбранной Wi-Fi-сети — разные секреты. Временная сеть `S3-Setup` служит для настройки, а не для выдачи доступа в школьную административную сеть. Учётные данные выбранного роутера сохраняет стек Wi-Fi по используемому библиотекой пути. Новые поля приложения автоматически такими данными не становятся: для них отдельно проектируют валидацию, версию формата и запись в Preferences.

40.3 Неблокирующий режим и явное окно времени

Установите WiFiManager через менеджер библиотек. В главе проверена версия 2.0.17. Объект объявлен глобально: при неблокирующем режиме он должен существовать и после завершения `setup()`. Функция `process()` обслуживается из основного цикла. Закрытие окна реализуем своим счётчиком `millis()`, а не полагаемся на действие настроек тайм-аута из другого режима библиотеки. Закрытие проверяется в цикле: сетевой вызов может задержать его, поэтому это не жёсткая гарантия времени.

Листинг 40.1: Портал по кнопке, собственный тайм-аут и независимое мигание

```
// Button GPIO5 -> GND. Hold for 2 s to open a 120 s setup window.
// LED GPIO4 through a resistor; standalone example, no ESPUI server.
#include <WiFi.h>
#include <WiFiManager.h>
WiFiManager wm; // must outlive setup() in nonblocking mode
uint32_t portalStarted = 0;
bool buttonHeld = false, handled = false;
uint32_t pressedAt = 0;

void openPortal() {
  wm.startConfigPortal("S3-Setup", "setup-40-demo");
  portalStarted = millis();
  Serial.println("Setup window requested; see AP address below");
  Serial.println(WiFi.softAPIP());
}
void setup() {
  Serial.begin(115200);
  pinMode(4, OUTPUT); digitalWrite(4, LOW);
  pinMode(5, INPUT_PULLUP);
  WiFi.mode(WIFI_STA);
  wm.setDebugOutput(false); // do not print network secrets
  wm.setConfigPortalBlocking(false);
  wm.setConnectTimeout(10);
  WiFi.begin(); // try credentials already stored on the board
  Serial.println("Hold GPIO5 button for 2 s to configure Wi-Fi");
}
void loop() {
  wm.process();
  uint32_t now = millis();
  if (wm.getConfigPortalActive() && now - portalStarted >= 120000) {
    wm.stopConfigPortal();
    Serial.println("Setup window closed");
  }
  if (digitalRead(5) == LOW) {
    if (!buttonHeld) {
      buttonHeld = true; handled = false; pressedAt = now;
    }
    if (!handled && now - pressedAt >= 2000) {
      handled = true;
      if (!wm.getConfigPortalActive()) openPortal();
    }
  } else {
    buttonHeld = false; handled = false;
  }
  static uint32_t last = 0;
  static bool led = false;
  if (now - last >= 500) {
    last = now; led = !led; digitalWrite(4, led);
  }
  static bool reported = false;
  if (WiFi.status() == WL_CONNECTED && !reported) {
    Serial.print("Connected: "); Serial.println(WiFi.localIP());
    reported = true;
  }
  if (WiFi.status() != WL_CONNECTED) reported = false;
  delay(1);
}
```

Подключите LED к GPIO4 через резистор, кнопку — к GPIO5 и GND. После старта удержите кнопку, подключитесь к `S3-Setup` и откройте адрес из лога. Выберите тестовую сеть 2,4 ГГц и введи-

те её пароль. Затем вернитесь на эту сеть и проверьте выданный устройству IP. Перезапустите плату: ранее сохранённые данные должны позволить подключение без повторного ввода. Имя «неблокирующий» не обещает жёсткого времени каждого вызова. Сканирование, подключение и обработка HTTP имеют собственную стоимость; в примере задан предел попытки подключения. Мигание показывает, что программа в целом продолжает работать, но не служит доказательством точного периода во время всех сетевых операций. Для аппаратно критичного действия нужны соответствующие таймер, задача и независимая защита.

40.4 Как совместить с ESPUI

Оба средства по умолчанию хотят обслуживать HTTP на порту 80. Нельзя просто объединить два скетча и дважды открыть сервер на одном адресе и порту. Для первого проекта используйте последовательные этапы:

1. устройство принимает запрос настройки и запускает портал WiFiManager;
2. после успешной настройки портал закрывается и его сетевые ресурсы освобождаются;
3. выполняется перезапуск либо явный переход режима, затем запускается ESPUI;
4. повторная настройка снова переводит устройство в отдельный режим.

Состояние режима храните отдельно от переключателя LED. Перезапуск после сохранения — простой учебный способ разделить серверы; запускайте его после подтверждения сохранения, а не в середине HTTP-обработчика. При потере сети в рабочем режиме устройство сохраняет локальное управление и пробует восстановить связь, а не немедленно возвращается к настройке. Если нужен одновременный доступ, придётся осознанно спроектировать владение серверами, портами, DNS и ресурсами: это отдельная архитектура. Captive portal не заменяет авторизацию рабочей панели и не добавляет HTTPS. Сначала проверьте каждый пример отдельно.

40.5 План проверки владельцем устройства

Проверьте первое включение без настроек; неверный пароль; закрытие 120-секундного окна; выключенный роутер; повторный запрос по кнопке; перезапуск после успешной настройки. Зафиксируйте ожидаемое безопасное состояние нагрузки и максимальное допустимое время ожидания. Не выводите пароли в Serial, веб-журнал или публичный репозиторий. Стирание всех настроек — отдельная команда с подтверждением, а не штатное действие при старте.

Если не работает

- Не открылось автоматическое окно — введите IP вручную; проверьте, что телефон остался в AP.
- Нет нужной сети — убедитесь в диапазоне 2,4 ГГц и уровне сигнала.
- Программа перестала обслуживать портал — объект уничтожен или не вызывается process().
- ESPUI не запускается — проверьте, остановлен ли сервер настройки и освобождён ли порт.

Итоги урока

WiFiManager упрощает выдачу параметров сети. Приложение определяет, кто и когда может открыть портал, сколько он работает и что происходит при ошибке. Настройка, рабочая панель и восстановление связи — отдельные состояния.

Проверь себя

1. Чем пароль временной AP отличается от пароля роутера?
2. Почему объект WiFiManager в этом примере не локален для setup()?
3. Почему два готовых веб-скетча нельзя просто запустить одновременно на порту 80?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Настройте подключение к тестовой сети через портал и найдите IP в логе.
2. ★ Перезапустите плату и подтвердите подключение без повторного ввода.
3. ★★ Проверьте неверный пароль и закрытие окна через 120 секунд; запишите фактическое время.
4. ★★ Добавьте понятный индикатор режимов «настройка», «подключение», «работа» без бесконечного ожидания.
5. ★★★ Добавьте проверяемое поле имени устройства и сохраняйте его отдельно от учётных данных Wi-Fi.
6. ★★★★ Объедините настройку и ESPUI в автомат с отдельными этапами, проверьте повторную настройку, потерю сети и восстановление без конфликта серверов.

Документация для продолжения WiFiManager: API и примеры, неблокирующие сценарии.

Урок 41. PlatformIO: профессиональная сборка проекта

Цель	собрать, прошить и проверить проект из нескольких файлов в PlatformIO и понимать, что записано в `platformio.ini`
Опирается на	урок 0 (прошивка), урок 14 (диагностика), урок 28 (классы и файлы, Git), урок 35 (отладка)
Результат	проект «кнопка и светодиод» с библиотекой, тестами на компьютере и отдельной сборкой для отладки
Понадобится	компьютер с VS Code и доступом в интернет; плата, кнопка и светодиод из урока 28; ESP-IDF не нужна

41.1 Сначала: чего не хватает Arduino IDE

Arduino IDE создана для первых шагов: открыл скетч, нажал «Загрузить». Пока в проекте один файл и две библиотеки, этого достаточно. Когда проект вырастает до десятка файлов, а над ним работают несколько человек, начинают мешать мелочи: нет записи, *какие именно* версии библиотек и ядра использовались; чтобы собрать проект на другом компьютере, нужно вручную повторить десяток действий в менеджере плат; нельзя одним файлом описать две сборки (рабочую и отладочную); нельзя автоматически прогнать проверки перед отправкой в Git.
PlatformIO решает эти задачи так: *весь проект описан одним текстовым файлом* `platformio.ini`. Достаточно скопировать папку проекта на новый компьютер и выполнить

сборку: PlatformIO сам скачает нужные платформу, компилятор, ядро Arduino и библиотеки нужных версий.

	Arduino IDE	PlatformIO
Описание проекта	выбор в меню, хранится в настройках программы	текстовый файл `platformio.ini` внутри проекта
Версии библиотек	«последняя установленная»	записаны в проекте, можно закрепить
Несколько сборок	переключать платы и параметры вручную	несколько разделов `[env:…]`
Структура	папка скетча с вкладками	`src/`, `include/`, `lib/`, `test/`
Тесты и проверка кода	нет	`pio test`, `pio check`
Отладка	ограниченно	пошагово, через встроенный USB-JTAG
Автоматическая проверка в Git	сложно	одна команда `pio run` в CI
Порог входа	очень низкий	выше: нужно понимать структуру проекта

Это не «лучшая» и «худшая» программа, а два инструмента для разных задач. Для уроков 0–27 Arduino IDE удобнее; для итогового проекта и командной работы PlatformIO выгоднее.

41.2 Установка

Есть два способа, и они ставят один и тот же PlatformIO Core:

1. **Расширение для VS Code.** Установите редактор Visual Studio Code, откройте вкладку расширений и найдите PlatformIO IDE. Первый запуск скачивает ядро PlatformIO и занимает несколько минут: не закрывайте окно, пока внизу не появится панель с кнопками.
2. **Командная строка.** Установите Python и выполните `pip install platformio`. Появится команда `pio`. Так PlatformIO запускают на сервере и в автоматических проверках.

Платы и компиляторы скачиваются по мере надобности при первой сборке и занимают порядка гигабайта. Подготовьте их заранее на всех компьютерах класса: скачивание во время занятия отнимает много времени.

Осторожно

Версия ядра Arduino. Эта книга написана для Arduino-ESP32 3.x. Официальная платформа `espressif32` в PlatformIO долго оставалась на ядре 2.x, а в нём иначе устроены некоторые функции (например, `ledcAttach` из урока 4 появилась именно в 3.x). Чтобы собирать примеры книги без правок, используйте платформу сообщества **pioarduino**: вместо имени `espressif32` в поле `platform` указывается адрес её архива (см. пример ниже). Какой вариант актуален сегодня, проверьте в документации PlatformIO и pioarduino.

41.3 Из чего состоит проект

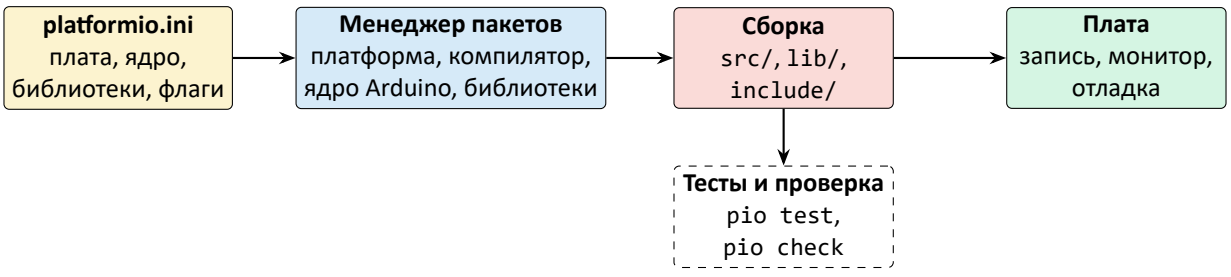


Рис. 41.1: Путь проекта в PlatformIO: всё начинается с файла настроек, остальное делают команды

Папка проекта выглядит так (в скобках указано, что попадает в Git):

```
lamp/
├── platformio.ini        настройки проекта (в Git)
├── src/
│   └── main.cpp          основная программа вместо .ino (в Git)
├── include/
│   └── config.h          общие заголовочные файлы: выводы, пороги (в Git)
├── lib/
│   └── Debounce/         собственные библиотеки проекта (в Git)
│       ├── library.json
│       └── src/Debounce.h, Debounce.cpp
├── test/
│   └── test_debounce/    модульные тесты (в Git)
└── .pio/                 результат сборки и скачанные пакеты (НЕ в Git)
```

Всё, что лежит в `.pio/`, PlatformIO создаёт сам и может создать заново. Там же находятся готовая прошивка `.pio/build/<окружение>/firmware.bin` и файл `firmware.elf`, который понадобится для разбора сбоев (урок 14) и отладки (урок 35). Папку `.pio` добавляют в `.gitignore`.

41.4 Файл platformio.ini: разбор по строкам

Файл состоит из разделов в квадратных скобках и пар «ключ = значение». Раздел `[env:имя]` описывает одну сборку (окружение); имя придумываете вы.

```
[platformio]
default_envs = esp32s3                ; что собирать, если окружение не указано

[env:esp32s3]
platform = https://github.com/pioarduino/platform-espressif32/releases/download/stable/platform-espressif32.zip
board = esp32-s3-devkitc-1            ; плата из списка: pio boards esp32-s3
framework = arduino                   ; Arduino-совместимые функции (setup, loop)
monitor_speed = 115200                ; скорость монитора порта
monitor_filters = esp32_exception_decoder, time
build_flags =
    -Wall                             ; показывать больше предупреждений
    -DCORE_DEBUG_LEVEL=1              ; уровень сообщений ядра (как в меню Arduino IDE)
lib_deps =
    adafruit/Adafruit SSD1306 @ ^2.5.0 ; библиотека и допустимые версии
    bblanchon/ArduinoJson @ ^7.0.0
```

Ключ	Что означает
`platform`	набор «платы, компилятор, ядро»: для ESP32 — `espressif32` или pioarduino
`board`	конкретная плата; от неё зависят размер Flash, частота, схема выводов. Список: `pio boards esp32-s3`
`framework`	`arduino` (эта книга) или `espidf` (урок 35)
`monitor_speed`	скорость порта; должна совпадать с `Serial.begin()`
`monitor_filters`	`esp32_exception_decoder` превращает адреса из Guru Meditation в имена функций и строки (урок 14); `time` добавляет время к строкам
`build_flags`	параметры компилятора; `-DИМЯ=значение` то же, что `#define` в коде
`lib_deps`	библиотеки проекта; PlatformIO скачивает их сам
`upload_port`, `monitor_port`	порт, если автоматический выбор ошибается: `COM5` или `/dev/ttyACM0`

Версии библиотек

После имени библиотеки указывают версию. Запись `@ ^7.0.0` означает «любая версия 7.x.y, не ниже 7.0.0»; `@ ~7.0.4` — «7.0.x, не ниже 7.0.4»; `@ 7.0.4` — ровно эта версия. Для учебного проекта закрепляйте версию: тогда через год проект соберётся так же, как сегодня. Библиотеку можно взять и прямо из Git: `https://github.com/автор/проект.git#v1.2.3`.

Несколько окружений

Раздел `[env:…]` можно расширять другим: ключ `extends` копирует все настройки и позволяет изменить лишь нужные. Общие строки выносят в обычный раздел и подставляют как `${common.build_flags}`. Так в одном файле живут рабочая сборка, отладочная и сборка для проверки на компьютере (пример в папке `examples/platformio-lamp`).

41.5 Первый проект

Откройте вкладку PlatformIO в VS Code, выберите PIO Home → New Project, плату Espressif ESP32-S3-DevKitC-1 и фреймворк Arduino. Или в командной строке:

```
pio project init --board esp32-s3-devkitc-1 --project-option "framework=arduino"
```

Схема подключения такая же, как в уроках 3 и 4: кнопка между **GPIO5** и GND (подтяжка внутренняя, внешний резистор не нужен), светодиод на **GPIO4** через резистор $220\,\Omega$.

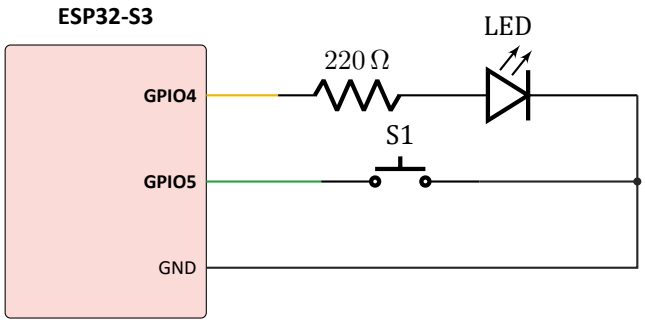


Рис. 41.2: Подключение примера `platformio-lamp`: кнопка **GPIO5**–GND, светодиод **GPIO4** через $220\,\Omega$ на GND

Положите в `src/main.cpp` программу кнопки и светодиода из урока 28. Её отличие от скетча в Arduino IDE — всего две строки: подключается `Arduino.h`, а время для проверки дребезга передаётся в класс снаружи.

```
#include <Arduino.h>                    // в .ino эту строку добавляла сама среда
#include <Debounce.h>                   // наша библиотека из папки lib/
#include "config.h"                     // выводы и интервалы из папки include/

Debounce button(DEBOUNCE_MS);
bool ledOn = false;

void setup() {
  Serial.begin(115200);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  if (button.update(digitalRead(BUTTON_PIN), millis())) {
    ledOn = !ledOn;
    digitalWrite(LED_PIN, ledOn);
    Serial.printf("lamp %s\n", ledOn ? "ON" : "OFF");
  }
}
```

Совет

Полный проект лежит в папке `examples/platformio-lamp` репозитория книги. Откройте её как папку в VS Code: PlatformIO сам найдёт `platformio.ini`.

Как перенести скетч из Arduino IDE

1. Переименуйте файл `.ino` в `src/main.cpp`.
2. Добавьте первой строкой `#include <Arduino.h>`.
3. Среда Arduino сама дописывала объявления функций. В `.cpp` функцию нужно объявить *раньше*, чем вызвать: поставьте определения выше места использования или добавьте прототипы вроде `void showTemp(float t);` в начало файла.
4. Библиотеки, которые раньше ставились через менеджер, впишите в `lib_deps`.

41.6 Команды

Кнопки внизу окна VS Code (галочка — сборка, стрелка — запись, вилка — монитор порта, колба — тесты) запускают те же команды, что и терминал:

Команда	Что делает
`pio run`	собрать окружение по умолчанию
`pio run -e esp32s3_debug`	собрать указанное окружение
`pio run -t upload`	собрать и записать в плату
`pio run -t clean`	удалить результаты сборки (помогает при странных ошибках)
`pio device list`	показать подключённые порты
`pio device monitor`	монитор порта с фильтрами из `platformio.ini`
`pio pkg install -l "имя"`	добавить библиотеку в проект
`pio pkg update`	обновить пакеты в рамках разрешённых версий
`pio test -e native`	запустить тесты на компьютере
`pio check`	статический анализ: поиск подозрительных мест без запуска

Запись по USB работает так же, как в уроке 0: у DevKitC-1 два разъёма. Порт с пометкой UART (через преобразователь) проще всего; на порту USB программа может не получить `Serial` без флагов `-DARDUINO_USB_MODE=1 -DARDUINO_USB_CDC_ON_BOOT=1` в `build_flags`. Если запись не начинается, удерживайте BOOT, нажмите и отпустите RESET, затем отпустите BOOT.

41.7 Свои библиотеки и папка include

Код, который нужен в нескольких проектах или должен проверяться отдельно, оформляют как библиотеку в `lib/<имя>/`. Минимум — заголовок и файл с реализацией в подпапке `src`. Файл `library.json` с именем и версией необязателен, но полезен: в нём можно перечислить зависимости.
Заголовки, общие для всего проекта (`config.h`), кладут в `include/`. Пароль сети и ключи — в `include/secrets.h`, который внесён в `.gitignore`. В Git добавляют образец `secrets.example.h` с пустыми значениями: так напарник сразу видит, какие поля нужно заполнить (напоминание из урока 28).

41.8 Тесты без платы

Самое ценное новшество PlatformIO — проверка логики программы на самом компьютере. Чтобы это работало, класс не должен напрямую вызывать `millis()` или `digitalRead()`: время и уровень ему передают снаружи, как в классе `Debounce`. Тогда тест подаёт в него придуманные последовательности и проверяет ответы.

```
void test_bounce_then_stable_press_gives_one_event() {
  Debounce d(50);
  const bool bounce[] = {false, true, false, true, false};   // дребезг в первые 5 мс
  for (uint32_t t = 0; t < 5; ++t) TEST_ASSERT_FALSE(d.update(bounce[t], t));
  TEST_ASSERT_FALSE(d.update(false, 53));    // ещё 49 мс -- рано
  TEST_ASSERT_TRUE(d.update(false, 54));     // 50 мс устойчивого уровня: событие
  TEST_ASSERT_FALSE(d.update(false, 60));    // оно не повторяется
}
```

Тесты лежат в папке `test/`, а для запуска на компьютере в `platformio.ini` нужно окружение `[env:native]` с `platform = native` (на компьютере должен быть установлен компилятор C++, на Windows — например, MinGW). Команда `pio test -e native` собирает и запускает все тесты и печатает итоговую таблицу. Те же тесты можно запустить на плате: `pio test -e esp32s3`. Проверять так стоит всё, что считает и решает: фильтры, конечные автоматы (урок 21), разбор команд, битовые операции (урок 22). Чтение датчиков и управление выводами тестируют уже на плате.

41.9 Отладка и работа по сети

Пошаговая отладка. В уроке 35 мы запускали GDB вручную. В PlatformIO то же самое описывается тремя строками, а запускается кнопкой Debug:

```
[env:esp32s3_debug]
extends = env:esp32s3
build_type = debug                  ; без оптимизации, с отладочной информацией
debug_tool = esp-builtin            ; встроенный USB-JTAG ESP32-S3
debug_init_break = tbreak setup     ; остановиться в начале setup()
```

Плата подключается к порту USB (а не UART). В Windows для интерфейса JTAG может потребоваться отдельный драйвер; ищите инструкцию в документации Espressif по JTAG.

Файлы LittleFS. Положите файлы в папку `data/` рядом с `platformio.ini`, добавьте строку `board_build.filesystem = littlefs` и выполните `pio run -t uploadfs`: образ файловой системы (урок 18) запишется в плату отдельно от прошивки.

Обновление по Wi-Fi. Если в прошивке есть ArduinoOTA (урок 13), запись можно перенаправить в сеть: `upload_protocol = espota`, `upload_port = lamp.local`. Первый раз прошивку всё равно записывают по кабелю.

Платы с PSRAM и большим Flash. Для модулей вроде ESP32-S3-WROOM-1 N16R8 (урок 11) добавляют `board_build.arduino.memory_type = qio_opi`, `board_upload.flash_size = 16MB`, `board_build.partitions = default_16MB.csv` и флаг `-DBOARD_HAS_PSRAM`. Сочетание параметров зависит от конкретного модуля: сверяйтесь с его маркировкой. Ошибочный выбор приводит к циклической перезагрузке на запуске.

41.10 PlatformIO в автоматической проверке

Раз всё описано файлом, проверку сборки легко поручить серверу. Для GitHub достаточно такого шага (так же устроены проверки примеров этой книги):

```
- uses: actions/setup-python@v5
  with: { python-version: '3.12' }
- run: pip install platformio
- run: pio test -e native          # быстрые тесты логики
- run: pio run -e esp32s3          # проект собирается
```

Каждая правка в репозитории теперь проверяется без участия человека: сломанная сборка видна сразу, а не в день защиты проекта.

41.11 Самостоятельный опыт: проект с тестом

1. Создайте проект для платы ESP32-S3-DevKitC-1 и перенесите в него кнопку и светодиод из урока 28.
2. Вынесите класс подавления дребезга в `lib/Debounce` так, чтобы в нём не было вызовов Arduino.
3. Напишите тест, подающий последовательность с дребезгом, и запустите `pio test -e native`. Затем сломайте программу (например, замените `>=` на `<`) и убедитесь, что тест это заметил.
4. Добавьте окружение `esp32s3_debug`, остановитесь на входе в `setup()` и посмотрите значение переменной.
5. Сделайте коммит. В Git должны попасть `platformio.ini`, `src`, `include`, `lib`, `test`, но не `.pio`.

Если не работает

- **«Плата не найдена»** — порт занят монитором другой программы, кабель только для зарядки или выбран не тот разъём. Проверьте `pio device list`.
- **«... was not declared in this scope»** после переноса скетча — функция вызвана раньше определения: добавьте прототипы или переставьте функции.
- **Ошибка при загрузке пакетов** — нет доступа к интернету или к GitHub; PlatformIO качает платформу при первой сборке. Подготовьте пакеты заранее.
- **Странные ошибки после правки настроек** — выполните `pio run -t clean` и соберите заново.
- **`Serial` молчит на порту USB** — добавьте флаги `-DARDUINO_USB_MODE=1` и `-DARDUINO_USB_CDC_ON_BOOT=1` или подключитесь к порту UART.
- **Не собираются функции 3.x (например, `ledcAttach`)** — выбрана платформа с ядром 2.x; используйте платформу pioarduino.
- **`pio test -e native` не запускается** — на компьютере нет компилятора C++ или в тестируемой библиотеке остались вызовы Arduino.

Итоги урока

- Файл `platformio.ini` полностью описывает проект: плату, ядро, библиотеки и их версии.
- Структура `src`, `include`, `lib`, `test` отделяет программу, общие заголовки, библиотеки и проверки; папка `.pio` в Git не попадает.
- Логику, не зависящую от платы, проверяют тестами на компьютере; вызовы `millis()` и `digitalRead()` передают в неё снаружи.
- Для книги нужно ядро Arduino-ESP32 3.x: смотрите, какую платформу вы указали.

Проверь себя

1. Какие преимущества даёт хранение версий библиотек в `platformio.ini`?
2. Почему класс подавления дребезга не вызывает `millis()` сам и как это помогает те-

стам?

3. Какие папки проекта попадают в Git, а какие нет и почему?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Установите PlatformIO, соберите и прошейте программу мигания светодиода.
2. ★ Найдите в папке `.pio` файлы `firmware.bin` и `firmware.elf` и сравните их размеры.
3. ★★ Добавьте библиотеку из `lib_deps` с закреплённой версией и выведите на OLED показание потенциометра.
4. ★★ Добавьте окружение `esp32s3_debug`, поставьте точку останова в `loop()` и покажите значение переменной.
5. ★★★ Напишите тесты для своего фильтра усреднения и конечного автомата и запустите их командой `pio test -e native`.
6. ★★★★ Настройте автоматическую сборку и тесты вашего проекта в GitHub Actions; покажите, как ломающая сборку правка отражается в проверке.

Документация для продолжения Плата ESP32-S3-DevKitC-1 в PlatformIO, описание platformio.ini, платформа pioarduino.

Урок 42. TinyML: нейросеть внутри микроконтроллера

Цель	пройти весь путь TinyML — от данных до работающей нейросети на плате — и измерить качество, время и память
Опирается на	урок 4 (ШИМ), урок 5 (АЦП), урок 26 (акселерометр), урок 37 (обучение, инференс, честная проверка)
Результат	сеть, которая повторяет синус и управляет яркостью светодиода, и классификатор жестов по акселерометру
Понадобится	плата, светодиод, MPU6050 (для опыта 2); компьютер с Python и NumPy; PlatformIO по желанию (урок 41)

42.1 Сначала: что такое TinyML

В уроке 37 мы научили программу различать два положения ручки, запомнив «центры» двух классов. Это уже машинное обучение: правило получено не от программиста, а из примеров. **TinyML** — направление, в котором такие модели, в том числе нейронные сети, запускают на микроконтроллерах, где нет операционной системы, гигабайт памяти и розетки. «Tiny» (крошечный) относится ко всему сразу: к модели, к памяти, к потреблению энергии.
Зачем это нужно, если есть облако? Сравните:

	Модель в облаке	Модель на микроконтроллере
Задержка	сотни миллисекунд и зависит от сети	единицы или десятки миллисекунд, стабильно
Без интернета	не работает	работает
Приватность	звук и кадры уходят с устройства	наружу уходит только решение («жест: тряска»)
Энергия	радиомодуль включён постоянно	радио спит, пока нет события (урок 9)
Размер модели	сколько угодно	от сотен байт до сотен килобайт
Сложность задачи	речь, текст, сложные изображения	жесты, шаги, вибрация, ключевое слово, простая картинка

Типичные задачи TinyML: «проснись по слову», распознавание жестов и активности по акселерометру, обнаружение износа подшипника по вибрации, подсчёт шагов, определение, что в кадре есть человек. Во всех случаях данные приходят от датчика *на самом устройстве*, а решение нужно сразу.

42.2 Когда нейросеть не нужна

Нейросеть — не волшебство, а ещё один способ написать функцию «признаки → решение». Если задачу решает порог (температура выше 30 градусов) или простая формула, нейросеть лишь усложнит проект: нужны данные, обучение, проверка и больше памяти. Нейросеть оправданна, когда правило трудно записать вручную: чем жест «тряска» отличается от жеста «встряхнул и положил» словами не скажешь, а примеры собрать легко. Всегда сравнивайте с простым решением, как мы делали в уроке 37.

42.3 Рабочий цикл TinyML

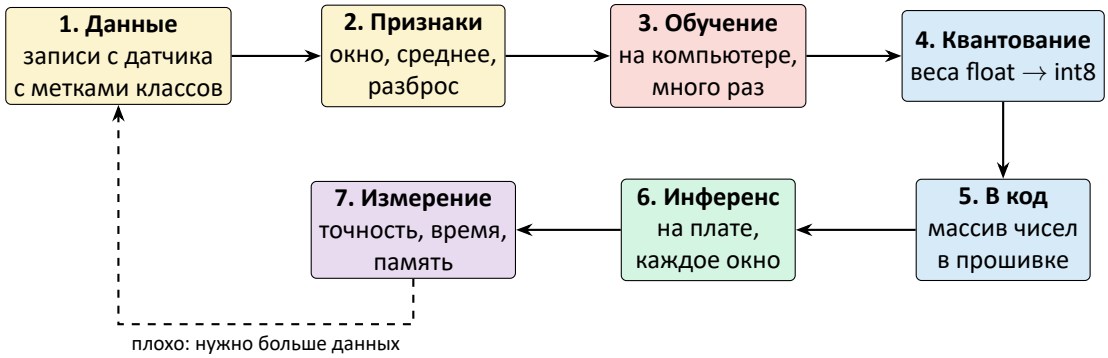


Рис. 42.1: Цикл TinyML. Обучение идёт на компьютере, на плате остаётся только прямой расчёт. Результат измерений возвращает нас к данным

Главное отличие от обычной программы: ошибка чаще прячется в *данных*, а не в коде. Поэтому шаги 1 и 7 важнее всего остального.

42.4 Нейрон и слой

Искусственный нейрон делает три вещи: умножает каждый вход на свой вес, складывает и добавляет смещение, затем пропускает результат через «функцию активации». В этой главе — функция ReLU, которая заменяет отрицательное число нулём:

$$y = \mathrm{ReLU}(w_1 x_1 + w_2 x_2 + b), \qquad \mathrm{ReLU}(z) = \max(0, z).$$

Пример. Пусть $x = (0{,}5;\ -1)$, веса $w = (2;\ 1)$, смещение $b = 0{,}3$. Тогда $2 \cdot 0{,}5 + 1 \cdot (-1) + 0{,}3 = 0{,}3$, и $y = 0{,}3$. При $b = -0{,}5$ сумма равна $-0{,}5$, ReLU даёт $y = 0$: нейрон «не сработал».
Слой — несколько таких нейронов, получающих одни и те же входы. Для слоя с n_{in} входами и n_{out} нейронами нужна таблица весов $n_{\text{in}} \times n_{\text{out}}$ и n_{out} смещений. **Сеть** — цепочка слоёв. Число параметров считают просто:

$$N = \sum_{\text{слои}} (n_{\text{in}} \cdot n_{\text{out}} + n_{\text{out}}).$$

У сети «1–8–8–1» из первого опыта $N = (8{+}8){+}(64{+}8){+}(8{+}1) = 97$ параметров. Это немногим больше сотни чисел — такая сеть помещается в обычную переменную.
Для классификации последний слой имеет по нейрону на класс, а ответы превращают в доли функцией **softmax**: если слой выдал $z = (2{,}0;\ 1{,}0;\ 0{,}1)$, то доли равны $e^{z_k} / \sum e^{z}$, то есть примерно $(0{,}66;\ 0{,}24;\ 0{,}10)$. Они складываются в единицу.

Осторожно

Доля после softmax — не «вероятность правоты». Сеть *всегда* распределит единицу между известными классами, даже если на вход пришло то, чего она не видела. Поэтому в классификаторе нужны порог уверенности, класс «неизвестно» и независимая проверка.

42.5 Как сеть учится

Обучение подбирает числа весов так, чтобы ответы на примерах были близки к правильным. Для этого вводят **функцию потерь** — число, показывающее, насколько ответ плох (для синуса — средний квадрат ошибки). Затем многократно вычисляют, в какую сторону изменить каждый вес, чтобы потери уменьшились, и делают небольшой шаг; размер шага называют **скоростью обучения**. Один проход по всем примерам — **эпоха**. Вычисления ведутся на компьютере, и вам не придётся писать их формулы: в файле `tinyml_nn.py` примерно сорок строк.

Главная опасность — **переобучение**: сеть запоминает примеры вместо общего правила. Поэтому данные делят на части: по одной сеть учится, по другой выбирают настройки, третью не трогают до самого конца и используют только для итоговой оценки.

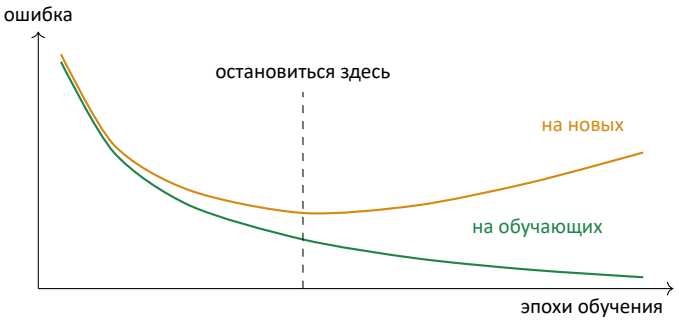


Рис. 42.2: Переобучение: ошибка на обучающих данных падает всё время, а на новых после какого-то момента растёт

42.6 Квантование: как уместить модель

Обычно вес — число с плавающей запятой, 4 байта. Но для маленьких сетей точность в семь знаков не нужна. **Квантование** хранит вес как целое число q в диапазоне от -127 до 127 (1 байт) и общий для слоя множитель **scale**:

$$w \approx q \cdot \text{scale}, \qquad \text{scale} = \frac{\max |w|}{127}, \qquad q = \text{round}\left(\frac{w}{\text{scale}}\right).$$

Пример: веса $(0{,}30;\ -1{,}27;\ 0{,}64)$, максимум по модулю $1{,}27$, scale $= 0{,}01$, получаем $q = (30;\ -127;\ 64)$. Ошибка каждого веса не превышает $\text{scale}/2 = 0{,}005$. Память весов уменьшилась в четыре раза, а сеть, как правило, почти не изменила своих ответов.

В нашем коде квантуются только веса (смещения и расчёт остаются во float), а в промышленных инструментах, таких как TensorFlow Lite Micro, квантуются также промежуточные значения. Тогда расчёт идёт целыми числами — быстрее и экономнее по энергии. Для этого требуется *калибровка*: модели показывают типичные данные, чтобы подобрать диапазоны. Но принцип остаётся тем же: меньше битов $\rightarrow$ меньше памяти и быстрее счёт, цена — небольшая потеря точности, *которую нужно измерить* после преобразования.

42.7 Опыт 1: сеть повторяет синус

Начнём с задачи, у которой есть готовый ответ, — $y = \sin x$ на отрезке от 0 до 2π. Её удобно проверять: правильное значение известно в любой точке. Значит, качество модели можно *измерить* числом.

Для опыта нужен только светодиод на **GPIO4** через резистор $220\,\Omega$ (как в уроке 4):

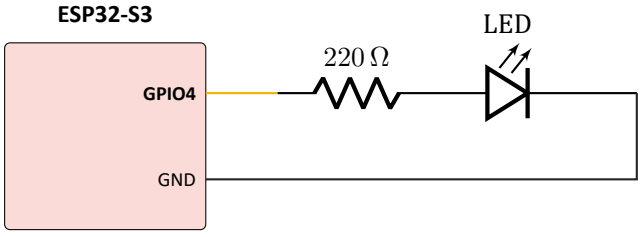


Рис. 42.3: Подключение светодиода для опыта 1: **GPIO4** через $220\,\Omega$ на GND

Шаг 1. Обучение на компьютере. В папке `examples/tinyml` лежат три файла: `tinyml_nn.py` (обучение, квантование, экспорт), `train_sine.py` и `train_gesture.py`. Нужен Python и библиотека NumPy (`pip install numpy`).

```
cd examples/tinyml
python train_sine.py > sine_model.txt
```

Сообщения о качестве выводятся в консоль, а готовый блок на C++ (массивы весов, множители scale, смещения) — в файл. Скрипт сам себя проверяет на 400 *новых* точках, не использованных при обучении:

```
макс. ошибка float32: 0.026
макс. ошибка int8:    0.044
средняя ошибка int8:  0.010
```

Видно цену квантования: максимальная ошибка выросла с 0,026 до 0,044, но на 80 байтах весов. Для яркости светодиода, где глаз различает единицы процентов, этого достаточно; для измерительного прибора — нет. Решение о том, «достаточно ли», зависит от задачи, а не от модели.

Шаг 2. Сеть в прошивке. Содержимое сгенерированного блока вставляют между строками `НАЧАЛО/КОНЕЦ СГЕНЕРИРОВАННОЙ МОДЕЛИ`. Весь расчёт сети — это одна функция `dense()` (слой), вызванная три раза. Она записана так, как слой выглядит в формуле: для каждого нейрона суммируем «вход × вес», умножаем на scale и добавляем смещение.

Листинг 42.1: Нейросеть 1–8–8–1 с весами int8; светодиод повторяет синус

```
// Урок 42. Сеть 1-8-8-1 повторяет sin(x); яркость светодиода следует за её ответом.
// Веса получены скриптом examples/tinyml/train_sine.py и хранятся как int8.
#include <Arduino.h>

const int LED_PIN = 4;

// ---- НАЧАЛО СГЕНЕРИРОВАННОЙ МОДЕЛИ ----
// слой 1: 1 -> 8
const int8_t W1[8] = {
  40, 124, 100, -127, 99, 56, -52, 80
};
const float S1 = 0.01451136f;
const float B1[8] = {
  0.59202f, -0.72172f, -0.37317f, 0.00000f, -0.89024f, -0.00234f, 0.00000f, -0.00330f
};
// слой 2: 8 -> 8
const int8_t W2[64] = {
  3, 36, 30, 38, -25, -9, -64, 21,
  0, -2, -41, -14, 0, -12, 102, 28,
  -118, -123, -72, -59, 7, 7, 123, -41,
  -13, 68, 22, 22, -17, -55, 6, 4,
  -45, -21, -37, -47, -3, 0, 68, -22,
  52, 42, 20, -23, 24, -17, 46, -127,
  31, -1, -42, -10, 2, 9, -33, -37,
  36, 7, 25, 37, -55, 5, 59, -49
};
const float S2 = 0.01495892f;
const float B2[8] = {
  -0.26971f, 0.27132f, 0.38404f, 0.29878f, 0.00000f, -0.05366f, -1.61403f, 0.05801f
};
// слой 3: 8 -> 1
```

```
const int8_t W3[8] = {
  -127, 101, 87, 93, -12, -51, 50, -76
};
const float S3 = 0.01427681f;
const float B3[1] = {
  -2.20406f
};
// всего весов: 80 (int8: 80 байт; во float было бы 320)

// ---- КОНЕЦ СГЕНЕРИРОВАННОЙ МОДЕЛИ ----

// Один полносвязный слой: out[j] = sum_i(in[i] * w[i][j]) * scale + b[j]
void dense(const float* in, int nIn, const int8_t* w, float scale,
           const float* b, float* out, int nOut, bool relu) {
  for (int j = 0; j < nOut; ++j) {
    float sum = 0;
    for (int i = 0; i < nIn; ++i) sum += in[i] * w[i * nOut + j];
    sum = sum * scale + b[j];
    out[j] = (relu && sum < 0) ? 0 : sum;
  }
}

float predictSin(float x) {                 // x в радианах, от 0 до 2π
  float in[1] = { x / (2 * PI) };           // та же нормализация, что при обучении
  float h1[8], h2[8], y[1];
  dense(in, 1, W1, S1, B1, h1, 8, true);
  dense(h1, 8, W2, S2, B2, h2, 8, true);
  dense(h2, 8, W3, S3, B3, y, 1, false);
  return y[0];
}

void setup() {
  Serial.begin(115200);
  if (!ledcAttach(LED_PIN, 5000, 8)) { Serial.println("PWM init failed"); return; }

  // Проверка качества: максимальное отличие от настоящего sinf() на 100 точках
  float worst = 0;
  for (int k = 0; k < 100; ++k) {
    float x = 2 * PI * k / 99.0f;
    worst = max(worst, fabsf(predictSin(x) - sinf(x)));
  }
  Serial.printf("max error = %.3f\n", worst);

  // Время одного предсказания: усредняем по 1000 запусков
  volatile float sink = 0;                  // чтобы компилятор не выбросил цикл
  uint32_t t0 = micros();
  for (int k = 0; k < 1000; ++k) sink = predictSin(k * 0.00628f);
  Serial.printf("one inference = %.2f us\n", (micros() - t0) / 1000.0f);
}

void loop() {
  float x = 2 * PI * (millis() % 4000) / 4000.0f;   // один оборот за 4 секунды
  float y = predictSin(x);                          // от -1 до +1
  int duty = constrain((int)((y + 1) * 127.5f), 0, 255);
  ledcWrite(LED_PIN, duty);
  Serial.printf("x:%.2f y:%.3f\n", x, y);           // формат Serial Plotter
  delay(20);
}
```

Разберём главное.

- `W[i * nOut + j]` — вес от входа `i` к нейрону `j`. Скрипт сохраняет таблицу «по строкам», поэтому порядок в Python и C++ совпадает; перепутаете оси — ответы станут случайными.
- `x / (2 * PI)` — та же *нормализация* входа, что при обучении. Это самая частая причина ошибки: данные в прошивке подготовлены не так, как во время обучения.
- Результат $[-1;\ 1]$ переведён в ШИМ $0 \ldots 255$ формулой $(y + 1) \cdot 127{,}5$ (урок 4).
- В `setup()` прошивка сама сравнивает сеть со стандартной `sinf()` и измеряет время одного предсказания. Эти два числа — ваши первые «измерения TinyML».

Откройте Serial Plotter: график y должен быть похож на синусоиду с периодом 4 секунды, а светодиод плавно «дышит». В Serial Monitor выводятся максимальная ошибка и время инференса: запишите оба числа в тетрадь, они понадобятся в задании 2.

42.8 Опыт 2: распознавание жестов по акселерометру

Теперь задача, для которой нет формулы: различить три состояния платы с MPU6050 (урок 26) — *покой*, *тряска* и *наклон*. Можно придумать свои классы, например «круг» и «удар», но их должно быть заметно меньше десяти.
Модуль подключается по I^2C так же, как в уроке 26: SDA к **GPIO8**, SCL к **GPIO9**, вывод AD0 на землю (адрес `0x68`, как в коде).

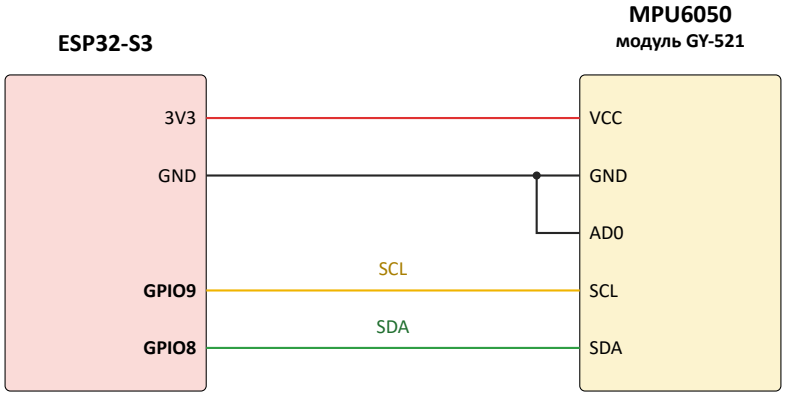


Рис. 42.4: Подключение MPU6050 (GY-521): питание 3,3 V, SDA на **GPIO8**, SCL на **GPIO9**. Подтяжки шины уже стоят на модуле; AD0 на GND задаёт адрес `0x68`

Шаг 1. Данные. Прошивка сбора берёт окно из 50 отсчётов (1 секунда) и сразу считает *признаки*: среднее и разброс (стандартное отклонение) по каждой из осей. Вместо 150 чисел сети нужны всего шесть — это уменьшает модель и ускоряет расчёт.

Листинг 42.2: Сбор данных: окно 1 секунда → строка CSV с шестью признаками

```
// Урок 42. Сбор данных: окно 1 с с MPU6050 (SDA GPIO8, SCL GPIO9) -> строка CSV.
// Команды из Serial Monitor: цифра 0..9 -- номер класса, n -- новый сеанс записи.
#include <Wire.h>

const uint8_t MPU_ADDR = 0x68;
const int WINDOW = 50;                      // 50 отсчётов по 20 мс = 1 секунда
const uint32_t PERIOD_MS = 20;

float samples[WINDOW][3];                   // ускорения по трём осям, в g
int count = 0;
int label = 0;
```

```
int session = 0;
uint32_t nextMs = 0;

bool readAccel(float* g) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);                          // регистр ACCEL_XOUT_H
  if (Wire.endTransmission(false) != 0) return false;
  if (Wire.requestFrom(MPU_ADDR, (size_t)6) != 6) return false;
  for (int i = 0; i < 3; ++i) {
    uint8_t hi = Wire.read();                  // порядок чтения важен: сначала старший байт
    uint8_t lo = Wire.read();
    int16_t raw = (int16_t)((hi << 8) | lo);
    g[i] = raw / 16384.0f;                     // диапазон ±2 g: 16384 отсчёта на 1 g
  }
  return true;
}

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);                             // разбудить датчик
  Wire.endTransmission();
  Serial.println("# session,label,mx,my,mz,sx,sy,sz");
  Serial.println("# digit = class, n = new session");
}

void loop() {
  while (Serial.available()) {
    char c = Serial.read();
    if (c >= '0' && c <= '9') { label = c - '0'; Serial.printf("# class %d\n", label); }
    if (c == 'n') { ++session; Serial.printf("# session %d\n", session); }
  }
  if (millis() < nextMs) return;
  nextMs += PERIOD_MS;
  if (nextMs < millis()) nextMs = millis() + PERIOD_MS;   // отстали -- не догоняем

  if (!readAccel(samples[count])) { Serial.println("# sensor error"); return; }
  if (++count < WINDOW) return;
  count = 0;

  float mean[3] = {0, 0, 0}, sd[3] = {0, 0, 0};
  for (int a = 0; a < 3; ++a) {
    for (int i = 0; i < WINDOW; ++i) mean[a] += samples[i][a];
    mean[a] /= WINDOW;
    for (int i = 0; i < WINDOW; ++i) sd[a] += sq(samples[i][a] - mean[a]);
    sd[a] = sqrtf(sd[a] / WINDOW);
  }
  Serial.printf("%d,%d,%.4f,%.4f,%.4f,%.4f,%.4f,%.4f\n", session, label,
                mean[0], mean[1], mean[2], sd[0], sd[1], sd[2]);
}
```

Откройте монитор порта. Цифрой выберите класс, затем делайте жест и наблюдайте строки вида `0,2,0.0123,0.7012,0.7021,0.0210,0.0198,0.0205`. Команда `n` начинает новый **сеанс**: новый запуск, новая поза платы, другой человек. Скопируйте строки в файл `gestures.csv` (строки с `#` скрипт пропускает).

Правило сбора	Почему
Не менее 3 сеансов на каждый класс, в каждом 40 и более окон	модель должна увидеть разброс условий, а не один раз
Меняйте позу платы, руку, темп	иначе сеть выучит «как лежала плата», а не «что такое тряска»
Включите класс «ничего» (покой, переход между жестами)	иначе любое движение станет одним из жестов
Записывайте отдельно сеансы для итоговой проверки	см. урок 37: соседние окна почти одинаковы

Шаг 2. Обучение. Скрипт делит данные *по сеансам*: по последнему сеансу каждого класса откладывается для теста и не участвует в обучении. Это тот самый честный тест из урока 37.

```
python train_gesture.py --csv gestures.csv > gesture_model.txt
```

Для проверки цепочки без платы есть режим `--demo` на искусственных данных. Он печатает **матрицу ошибок**: строка — настоящий класс, столбец — ответ сети.

```
Тест (int8), точность 100.0%, окон: 180
[[60  0  0]
 [ 0 60  0]
 [ 0  0 60]]
```

Идеальные 100% здесь получены на искусственных данных, где классы разведены нарочно далеко. Это проверка программы, а не жестов. На ваших настоящих записях цифра будет ниже, и именно по внедиагональным числам видно, *какие* жесты путаются. Не расстраивайтесь, а добавляйте данные.

Шаг 3. Инференс на плате. В прошивке есть те же шесть признаков, нормализация (`FMEAN`, `FSTD` — среднее и разброс признаков из обучения), сеть 6–8–3 и softmax.

Листинг 42.3: Распознавание жестов на плате: порог уверенности и подтверждение

```
// Урок 42. Распознавание жестов на устройстве: окно 1 с -> 6 признаков -> сеть 6-8-3.
// Блок модели создаёт examples/tinyml/train_gesture.py; здесь стоит модель, обученная на
↪  демо-данных.
#include <Wire.h>

const uint8_t MPU_ADDR = 0x68;
const int WINDOW = 50;
const uint32_t PERIOD_MS = 20;
const float MIN_CONFIDENCE = 0.80f;          // ниже -- "не уверен", класс не объявляем
const int CONFIRM = 3;                       // сколько окон подряд должны согласиться

// ---- НАЧАЛО СГЕНЕРИРОВАННОЙ МОДЕЛИ ----
// классы: 0=покой, 1=тряска, 2=наклон
const int NC = 3;
// слой 1: 6 -> 8
const int8_t W1[48] = {
  41, 10, -9, -72, -14, -9, 15, -12,
  -36, 56, 32, 65, -77, 23, -60, -27,
  48, -54, -58, -61, 48, -39, 89, 77,
  -111, -53, -28, -42, -35, 17, 127, -47,
  -29, 89, 2, -4, -60, -50, 59, -4,
  -60, -9, -24, -61, -46, 13, 55, -27
};
const float S1 = 0.01629021f;
```

```
const float B1[8] = {
  0.18936f, 0.16731f, 0.57299f, 0.46070f, 0.83547f, -0.11008f, 0.74104f, 0.07001f
};
// слой 2: 8 -> 3
const int8_t W2[24] = {
  92, -3, 21, -100, 18, 39, -16, -100,
  99, -41, -102, 27, 80, -59, -127, -64,
  -16, 35, -57, 102, -116, 63, 14, -90
};
const float S2 = 0.01073441f;
const float B2[3] = {
  0.36822f, -0.36463f, 0.23739f
};
// всего весов: 72 (int8: 72 байт; во float было бы 288)

const float FMEAN[6] = { 0.00933f, 0.21500f, 0.87735f, 0.16119f, 0.15984f, 0.12908f };
const float FSTD[6] = { 0.04454f, 0.34230f, 0.16016f, 0.20487f, 0.20293f, 0.15954f };
// ---- КОНЕЦ СГЕНЕРИРОВАННОЙ МОДЕЛИ ----

const char* NAMES[] = {"покой", "тряска", "наклон"};

float samples[WINDOW][3];
int count = 0;
uint32_t nextMs = 0;

bool readAccel(float* g) {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);
  if (Wire.endTransmission(false) != 0) return false;
  if (Wire.requestFrom(MPU_ADDR, (size_t)6) != 6) return false;
  for (int i = 0; i < 3; ++i) {
    uint8_t hi = Wire.read();                 // порядок чтения важен: сначала старший байт
    uint8_t lo = Wire.read();
    int16_t raw = (int16_t)((hi << 8) | lo);
    g[i] = raw / 16384.0f;
  }
  return true;
}

void dense(const float* in, int nIn, const int8_t* w, float scale,
           const float* b, float* out, int nOut, bool relu) {
  for (int j = 0; j < nOut; ++j) {
    float sum = 0;
    for (int i = 0; i < nIn; ++i) sum += in[i] * w[i * nOut + j];
    sum = sum * scale + b[j];
    out[j] = (relu && sum < 0) ? 0 : sum;
  }
}

// Возвращает номер класса и уверенность (0..1); softmax превращает ответы сети в доли.
int classify(const float* feat, float* confidence) {
  float in[6], h[8], z[NC];
  for (int i = 0; i < 6; ++i) in[i] = (feat[i] - FMEAN[i]) / FSTD[i];   // как при обучении
  dense(in, 6, W1, S1, B1, h, 8, true);
  dense(h, 8, W2, S2, B2, z, NC, false);
  int best = 0;
  for (int k = 1; k < NC; ++k) if (z[k] > z[best]) best = k;
  float sum = 0;
```

```
  for (int k = 0; k < NC; ++k) sum += expf(z[k] - z[best]);   // вычитаем максимум -- так
↪  надёжнее
  *confidence = 1.0f / sum;
  return best;
}

void setup() {
  Serial.begin(115200);
  Wire.begin(8, 9);
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission();
}

void loop() {
  if (millis() < nextMs) return;
  nextMs += PERIOD_MS;
  if (nextMs < millis()) nextMs = millis() + PERIOD_MS;

  if (!readAccel(samples[count])) { Serial.println("sensor error"); return; }
  if (++count < WINDOW) return;
  count = 0;

  float feat[6] = {0, 0, 0, 0, 0, 0};       // 0..2 -- средние, 3..5 -- разброс
  for (int a = 0; a < 3; ++a) {
    for (int i = 0; i < WINDOW; ++i) feat[a] += samples[i][a];
    feat[a] /= WINDOW;
    for (int i = 0; i < WINDOW; ++i) feat[3 + a] += sq(samples[i][a] - feat[a]);
    feat[3 + a] = sqrtf(feat[3 + a] / WINDOW);
  }

  uint32_t t0 = micros();
  float conf;
  int cls = classify(feat, &conf);
  uint32_t us = micros() - t0;

  static int lastCls = -1, streak = 0;
  int shown = (conf >= MIN_CONFIDENCE) ? cls : -1;      // -1 -- «неизвестно»
  streak = (shown == lastCls) ? streak + 1 : 1;
  lastCls = shown;

  Serial.printf("class:%s conf:%.2f time:%u us%s\n",
                shown < 0 ? "?" : NAMES[shown], conf, (unsigned)us,
                (shown >= 0 && streak >= CONFIRM) ? "  <-- ПОДТВЕРЖДЕНО" : "");
}
```

Обратите внимание на три приёма, которые отличают игрушку от рабочего устройства:

- **Порог уверенности** `MIN_CONFIDENCE`: если лучший класс получил меньше 80%, ответ — «?» (неизвестно). Порог подбирают на тестовых сеансах: ниже — больше ложных срабатываний, выше — больше пропусков.
- **Подтверждение** `CONFIRM`: действие запускается, только когда три окна подряд дали один и тот же класс. Единичная ошибка не включит мотор (мысль из урока 37).
- **Измерение времени**: в каждой строке вывода — время расчёта в микросекундах. Оно складывается из считывания окна (1 с) и самого инференса; на фоне секунды расчёт сети ничтожен. Оценивайте задержку *всей* цепочки, а не одной функции.

Осторожно

Модель в `l42_imu_infer` обучена на демонстрационных данных и на реальной плате будет ошибаться. Замените блок между строками `НАЧАЛО/КОНЕЦ СГЕНЕРИРОВАННОЙ МОДЕЛИ` результатом вашего `train_gesture.py`, иначе вы проверяете чужие данные. Если вы меняете число классов или признаков, измените также `NAMES`, размеры массивов `in[]` и `h[]` и параметры вызовов `dense()`.

42.9 Профессиональные инструменты

Собственный код на NumPy нужен, чтобы понять принцип: за рамками учебных моделей разумнее использовать готовые инструменты. Они делают то же самое, но умеют сложные слои (свёртки для изображений и звука), полное квантование и оптимизированные ядра для процессора ESP32-S3.

Инструмент	Для чего
TensorFlow Lite for Microcontrollers (TFLM) и его порт для Espressif, `esp-tflite-micro`	запуск обученных в Python моделей на микроконтроллере; ядра ESP-NN используют возможности ESP32-S3
ESP-DL (с преобразователем ESP-PPQ)	библиотека Espressif для инференса и квантованных моделей на ESP32; нужна версия ESP-IDF, указанная в документации
Edge Impulse	сайт, где собирают данные, обучают и экспортируют готовую библиотеку для Arduino или ESP-IDF
ESP-SR, ESP-WHO	готовые решения Espressif: распознавание ключевых слов и речи, лица и объекты

Типовой маршрут с TFLM состоит из тех же шагов цикла TinyML. Модель описывают в Keras и обучают на компьютере, затем преобразуют в формат `.tflite` с полным квантованием в int8:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data   # типичные входы для калибровки
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
open("model.tflite", "wb").write(converter.convert())
```

Файл превращают в массив C++ (командой `xxd -i` или коротким скриптом на Python) и подключают к прошивке. Схема кода на плате выглядит так (имена зависят от версии библиотеки, сверяйтесь с её примерами):

```
const tflite::Model* model = tflite::GetModel(model_data);   // массив из .tflite
static tflite::MicroMutableOpResolver<3> ops;                // только нужные операции
ops.AddFullyConnected();  ops.AddRelu();  ops.AddSoftmax();

alignas(16) static uint8_t arena[20 * 1024];                 // рабочая память модели
static tflite::MicroInterpreter interp(model, ops, arena, sizeof(arena));
interp.AllocateTensors();                                    // не хватило арены -- ошибка

interp.input(0)->data.int8[0] = quantizedInput;              // положить вход
interp.Invoke();                                             // посчитать
int8_t result = interp.output(0)->data.int8[0];              // забрать ответ
```

Здесь «арена» (*tensor arena*) — заранее выделенный кусок RAM для всех промежуточных значений. Её размер подбирают опытом: начинают с запаса и уменьшают, пока `AllocateTensors()` успешен. В PlatformIO (урок 41) такие библиотеки подключают строкой в `lib_deps`. Разберитесь

сначала с приведённым выше «ручным» кодом: он делает то же, что `Invoke()`, только слои у вас на виду.

Это интересно

ESP32-S3 имеет векторные инструкции, ускоряющие операции сразу над несколькими значениями. Но ускорение получают только оптимизированные ядра (ESP-NN, ESP-DL) и только на поддерживаемых операциях. Наш цикл на C++ из опыта 1 выполняется обычными командами процессора. Это не недостаток, а повод измерять, а не верить рекламе.

42.10 Что измерять и как не обмануться

Что	Как измерить	Зачем
Размер модели	размер массивов весов в байтах (у нас 80 и 72)	хватит ли Flash и сколько останется приложению
Память во время работы	`ESP.getFreeHeap()` до и после создания модели, размер арены	модель не должна вытеснить Wi-Fi и стек
Время инференса	`micros()` до и после, усреднить по 1000 запускам (как в опыте 1)	уложится ли расчёт в период опроса датчика
Качество	матрица ошибок на отдельных сеансах, доля ложных срабатываний	модель лучше простого порога или нет
Энергия	$E = U \cdot I \cdot t$: напряжение, ток и время расчёта	хватит ли аккумулятора (урок 27)

Пример расчёта энергии с условными числами: инференс длится 2 мс при токе 80 мА и напряжении 3,3 В. Тогда $E = 3{,}3 \cdot 0{,}08 \cdot 0{,}002 \approx 0{,}00053$ Дж, то есть 0,53 мДж на одно решение. Если решение нужно раз в секунду, расчёт потребляет в среднем 0,53 мВт — но датчик, радио и стабилизатор могут расходовать на два порядка больше (урок 36). Поэтому энергию меряют у *всего устройства*.

Три источника самообмана в TinyML:

- **Тест на обучающих данных.** Почти любая модель «хороша» на тех примерах, на которых училась.
- **Данные одной обстановки.** Сеть выучила стол и руку одного человека, а не жест.
- **Среднее скрывает редкое.** 95% точности могут означать, что один класс не распознаётся никогда. Смотрите на матрицу целиком.

Модель — часть прошивки и должна иметь версию. Если вы меняете признаки (длину окна, ось), старые веса теряют смысл: храните номер формата рядом с моделью, как в NVS (урок 37). Не записывайте чужие звук и видео без разрешения: даже локальное устройство, которое «слышит», вызывает вопросы приватности. Пока решение принимает не человек, действие должно быть безопасным при ошибке.

Если не работает

- **Сеть выдаёт одно и то же число.** Проверьте нормализацию входа и порядок осей весов: самый частый виновник — данные при обучении и на плате подготовлены по-разному.
- **Модель идеальна на компьютере и плоха на плате.** Признаки в прошивке посчитаны иначе (другие единицы, другая длина окна, `int` вместо `float`). Выведите признаки на плате и сравните с теми, что видел скрипт.
- **Все жесты распознаются как один класс.** Классы слишком похожи или данных мало; посмотрите на матрицу ошибок и графики признаков.
- **Ошибка компиляции в сгенерированном блоке.** Скрипт запущен с другим числом клас-

сов, чем ожидает прошивка (`NC`, `NAMES`, размер `z[]`).

- **Данные собираются «рывками».** Опрос датчика выполняется реже 50 Гц: избегайте длинных `delay()` и тяжёлого вывода в цикле.

Итоги урока

- TinyML обучает модель на компьютере, а на плате остаётся только расчёт; для маленьких задач это десятки или сотни байт весов.
- Квантование уменьшает вес с 4 байт до 1 за небольшую потерю точности, которую нужно измерить.
- Качество проверяют на отдельных сеансах, смотрят на матрицу ошибок, а в устройстве добавляют порог уверенности и подтверждение, потому что softmax всегда выбирает какой-то класс.
- Готовые инструменты (TFLM, ESP-DL, Edge Impulse) устроены по тому же циклу, но умеют больше слоёв и используют ускорение процессора ESP32-S3.

Проверь себя

1. Что выигрывают и чем рискуют при квантовании весов из float32 в int8?
2. Почему высокая точность на обучающих данных ничего не говорит о качестве модели?
3. Зачем в классификаторе жестов нужны порог уверенности и несколько подтверждений подряд?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Запустите `train_sine.py` и сравните ошибки float32 и int8 в консоли.
2. ★ Прошейте опыт 1 и запишите максимальную ошибку и время одного предсказания.
3. ★★ Замените размеры слоёв на 1–4–4–1 и 1–16–16–1; сравните число весов, ошибку и время (размеры в вызовах `dense()` в прошивке тоже изменятся).
4. ★★ Научите сеть другой функции (например, $\cos x$ или $|x - \pi|$) и измерьте её качество на новых точках.
5. ★★★ Соберите данные трёх жестов минимум в трёх сеансах на класс, обучите сеть, составьте матрицу ошибок на отдельном сеансе и подберите порог `MIN_CONFIDENCE`.
6. ★★★★ Обучите ту же задачу в Keras, преобразуйте в int8 `.tflite` и запустите в TFLM; сравните размер, время и точность со своей реализацией.

Документация для продолжения TensorFlow Lite для микроконтроллеров, esp-tflite-micro, ESP-DL.

Урок 43. Итоговый проект: от идеи до защиты

Цель	довести собственную идею до работающего устройства и уметь рассказать о нём за пять минут
Опирается на	все предыдущие уроки; опирайтесь на те, что нужны вашей теме
Результат	готовое устройство, электрическая схема, описание проекта и выступление на защите
Понадобится	детали выбранной темы (приложение B), компьютер для документации, 4–6 недель

43.1 Сначала: проект — это история с началом и концом

В уроках вы делали отдельные опыты, каждый из которых отвечал на один вопрос. Проект — это когда несколько опытов складываются в устройство, у которого есть задача, пользователь и результат. В нём мало просто «что-то мигает»: нужно понять, *что* устройство решает, *как* оно это делает и *как проверить*, что получилось. Такой урок не про новый код, а про умение довести дело до конца.

43.2 Как выбрать тему

Хорошая тема объединяет три условия: она вам интересна, решаема за 4–6 недель и опирается на уроки, которые вы уже освоили. Ниже — идеи на выбор. Можно взять свою, но проверьте её тем же способом: по таблице подсчитайте, на какие уроки она опирается и какие новые детали нужны.

№	Идея	Что делает устройство	Уроки
1	Метеостанция	измеряет температуру и влажность, показывает на OLED и пишет журнал на SD	5, 6, 10, 15
2	Парковочный радар	измеряет расстояние до препятствия, показывает лентой и звуком	4, 15, 17
3	Умная теплица	включает полив по влажности и освещению, ведёт журнал	5, 9, 15, 16
4	Сигнализация	реагирует на движение или дверь и присылает сообщение по MQTT	3, 8, 12, 15
5	Пульт для презентаций	листает слайды кнопками и регулирует громкость	3, 5, 19
6	Лампа-настроение	лента меняет цвета эффектами, а режим выбирается с телефона	8, 11, 17
7	Сетевые часы	показывают московское время на OLED и ленте, яркость по освещённости	5, 6, 17, 18
8	Робот-тележка	едет и объезжает препятствия по дальномеру	4, 15, 16
9	Камера наблюдения за растением	снимает кадры раз в час и показывает поток по запросу (с согласия окружающих)	8, 10, 18, 20
10	Автономный датчик	спит от батарейки, просыпается и отправляет данные другой плате	9, 12, 15
11	Умный вентилятор	мотор включается по температуре, порог хранится в памяти	4, 11, 15, 16
12	Игра «Реакция»	лента зажигается случайно, нужно успеть нажать кнопку; рекорд сохраняется	3, 11, 17

43.3 Проекты на возможностях ESP32-S3

После расширенного модуля можно выбрать аудиорегистратор I²S (29), измеритель импульсов RMT/PCNT (30–31), сеть датчиков CAN/RS-485 (32), цветную панель (33), USB-пульт с консолью CDC (34), датчик с ULP-пробуждением (36) или распознавание жестов IMU (37). Для каждой темы задайте измеримый результат: долю потерянных блоков аудио, погрешность частоты, время восстановления связи, задержку кадра, расход энергии или матрицу ошибок модели. Испытания из урока 38 включите в защиту. Для локальной панели используйте ESPUI (39), для выдачи настроек сети — отдельный этап WiFiManager (40). Не требуется использовать всю периферию одновременно: правильно обоснованный выбор и подтверждённый результат важнее количества модулей.

43.4 Техническое задание на полстраницы

ТЗ — это договор с самим собой. Оно отвечает на вопрос «что считается готовым проектом». Напишите его до покупки деталей и первой строчки кода. Используйте такую форму:

1. **Название** и одно предложение: для кого и зачем это устройство.
2. **Функции**: 3–5 пунктов вида «устройство должно…».
3. **Входы и выходы**: какие датчики и кнопки измеряют мир, какие исполнители им отвечают.
4. **Условия и ограничения**: питание, размеры, безопасность (напряжение, ток, съёмка людей).
5. **Критерий успеха**: измеримая проверка, например «температура отличается от термометра не больше чем на 1 °C».
6. **Состав**: список деталей (используйте приложение В).

43.5 Блок-схема устройства

Блок-схема помогает увидеть проект целиком до сборки. Рисуйте слева входы, справа выходы, в центре ESP32-S3, а питание и связь — отдельно. Для каждой стрелки подпишите сигнал и вывод платы.

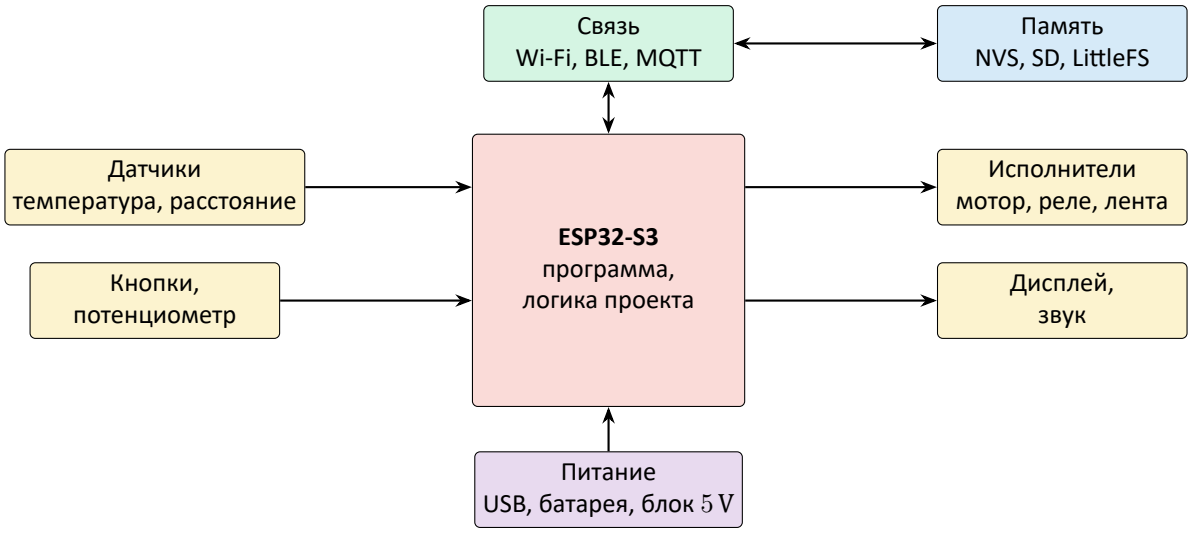


Рис. 43.1: Шаблон блок-схемы: входы слева, выходы справа, связь и память сверху, питание снизу. Подпишите на стрелках сигналы и номера GPIO вашего устройства

43.6 План по неделям

Рассчитывайте на пять-шесть недель. Самая частая причина провала — пустая последняя неделя, в которую уходит «всё остальное».

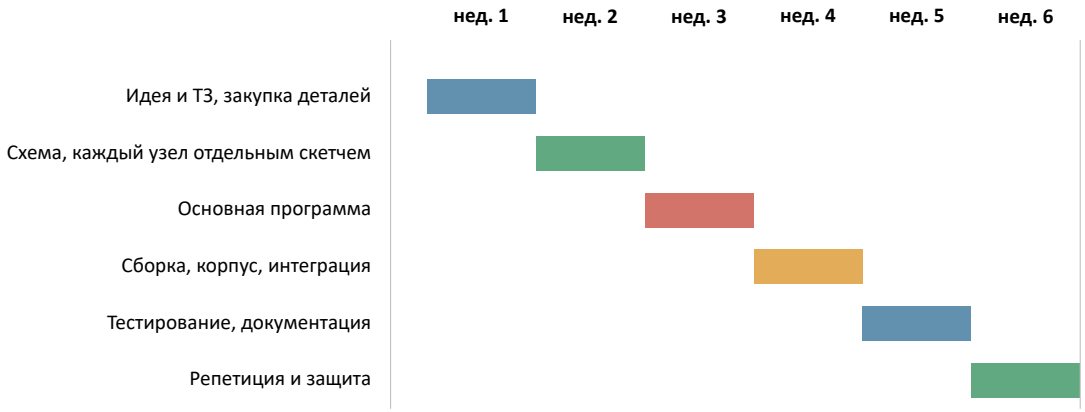


Рис. 43.2: План проекта на шесть недель. К пятой неделе устройство должно уже работать: остаётся время на проверку и документацию

43.7 Чек-лист готовности

Перед защитой пройдите по списку. Если хоть один пункт пустой, это тема для разговора с педагогом.

- ❑ Устройство включается от своего источника питания и работает без компьютера.
- ❑ После выключения и повторного включения оно возвращается к рабочему состоянию.
- ❑ Каждая функция из ТЗ проверена, результат записан.
- ❑ Электрическая схема нарисована и совпадает с реальной сборкой.
- ❑ Код разложен по функциям, в нём нет паролей, а комментарии объясняют, *почему*.
- ❑ Мощные нагрузки подключены через ключ или драйвер, питание продумано (урок 16).
- ❑ Подготовлены запасной вариант демонстрации (видео или фото) и копия проекта на втором носителе.
- ❑ Есть описание проекта на 1–2 страницы: цель, схема, инструкция, выводы.

43.8 Защита за пять минут

Пять минут — это мало, поэтому структура строгая. Говорите о главном и не читайте текст с листа.

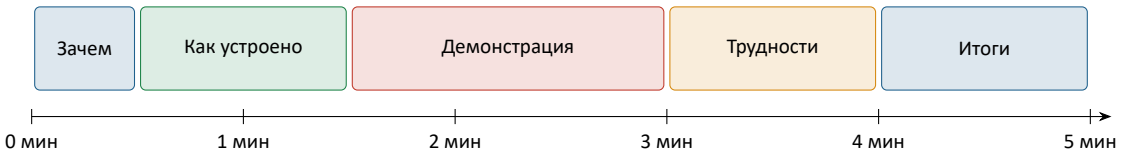


Рис. 43.3: Распределение времени на защите проекта

Время	Блок	Что сказать
0:00–0:30	Зачем	какую задачу решает устройство и для кого
0:30–1:30	Как устроено	блок-схема, главные узлы, какие уроки пригодились
1:30–3:00	Демонстрация	устройство работает вживую по сценарию из ТЗ
3:00–4:00	Трудности	одна ошибка, как вы её нашли и исправили
4:00–5:00	Итоги	что получилось, что улучшить, ответы на вопросы

43.9 Критерии оценки

Критерий	Баллы	Что оценивается
Работоспособность	0–30	устройство выполняет все функции из ТЗ, стабильно работает без вмешательства
Код	0–20	понятные имена, функции, комментарии; нет бесконечных ожиданий и паролей в тексте
Схема и сборка	0–15	схема читаема и совпадает с устройством; питание и земля продуманы
Документация	0–15	ТЗ, схема, инструкция, результаты проверок, список деталей
Выступление	0–20	укладывается в 5 минут, понятно рассказывает, отвечает на вопросы
Итого	**100**	85 и более — отлично; 65–84 — хорошо; 40–64 — удовлетворительно

Осторожно

Типичные ошибки проектов.

- **Слишком большая идея.** «Умный дом целиком» не получится за пять недель. Выберите одну комнату, одну функцию и доведите её.
- **Всё в последний день.** Сборка и отладка занимают вдвое больше времени, чем кажется. Не оставляйте подключение на финал.
- **Проект без схемы.** «Помню, что куда подключено» перестаёт работать через неделю. Фотографируйте и рисуйте схему сразу.
- **Питание «от чего придётся».** Мотор и лента от порта ноутбука приводят к перезагрузкам. Проверьте ток заранее.
- **Демонстрация без запасного плана.** Wi-Fi в зале может не работать. Подготовьте видео и снимки.
- **Пароли в коде.** Не показывайте и не публикуйте SSID и пароль; перед публикацией замените их заглушками.

Итоги урока

- Хороший проект начинается с ТЗ и измеримого критерия успеха, а не с покупки деталей.
- План, схема и чек-лист нужны, чтобы к защите устройство, документация и рассказ были готовы.
- Защита — это пять минут: зачем, как устроено, демонстрация, трудности, итоги.

Проверь себя

1. Какие три условия должна выполнять хорошая тема проекта?
2. Что такое критерий успеха и чем он отличается от списка функций?
3. Что должно быть на блок-схеме устройства и почему на ней отдельно рисуют питание?

Задания для самостоятельной работы

★ повторить ★★ изменить ★★★ создать своё ★★★★ вызов

1. ★ Выберите тему из таблицы или предложите свою.
2. ★ Запишите три требования к устройству: что оно должно делать, как это проверить, чем измерить.
3. ★★ Напишите ТЗ на полстраницы по форме урока.
4. ★★ Нарисуйте блок-схему и электрическую схему устройства с номерами GPIO.
5. ★★★ Составьте перечень деталей и план по неделям с контрольными точками.
6. ★★★★ Проведите репетицию защиты на 5 минут с таймером, запишите замечания слушателей и исправьте слабые места.

Приложение А. Справочник

Как пользоваться этой таблицей

Номер на проводе и номер в программе — это номер GPIO, а не порядковый номер контакта на гребёнке. Сначала найдите на pinout надпись GPIO, затем сверьте её с таблицей. Для первых опытов берите GPIO4, GPIO5, GPIO6, GPIO7, GPIO15–18: они подходят для LED, кнопки и большинства простых датчиков.

Осторожно

Все GPIO ESP32-S3 работают с логикой **3,3 В**. Контакт 5V — питание платы, а не сигнал для GPIO. Общий GND соединяют у всех устройств; светодиод всегда подключают через резистор; мотор, реле и ленту не подключают к GPIO напрямую.

Питание и основные контакты

Контакт	Что это	Как использовать новичку
3V3	выход стабилизатора 3,3 В	питание маломощного датчика, потенциометра, OLED
5V	питание от USB / вход платы	не соединять с GPIO; не считать логическим HIGH
GND	общая точка схемы	соединить с GND каждого датчика и отдельного источника питания
EN / RST	сброс чипа	кратко нажать, если программа зависла
BOOT (GPIO0)	режим загрузки	удерживать при сбросе, только если IDE не прошивает плату

Быстрый выбор GPIO для учебных опытов

Задача	Рекомендуемые GPIO	Примечание
LED, кнопка, цифровой датчик	4, 5, 6, 7, 15–18	начните с GPIO4 (LED) и GPIO5 (кнопка)
Потенциометр / аналоговый датчик	1–10 (ADC1)	ADC1 продолжает работать с Wi-Fi
UART-модуль	17 и 18	TX ESP соединяется с RX модуля, и наоборот
I^2C	8 (SDA), 9 (SCL)	можно переназначить в коде; нужны подтяжки на длинной линии
Нативный USB	19, 20	не использовать как обычные GPIO, если нужен USB

Перед подключением нового модуля

1. Найдите напряжение питания и логические уровни модуля: 3,3 В или 5 В.
2. Соедините GND модуля с GND ESP32-S3.
3. Для LED поставьте резистор; для нагрузки используйте драйвер/транзистор и отдельное питание.
4. Проверьте, что выбранный GPIO не попал в таблицу «опасных» ниже.
5. В первый запуск смотрите Serial Monitor: он быстрее показывает ошибку, чем гадание по проводам.

«Опасные» выводы ESP32-S3

Выводы	Особенность
GPIO0, 3, 45, 46	strapping: влияют на режим загрузки; GPIO0 — кнопка BOOT
GPIO19, 20	нативный USB (D−, D+)
GPIO26–32	SPI-Flash — не использовать
GPIO33–37	заняты при Octal Flash/PSRAM (35–37 — при Octal PSRAM)
GPIO39–42	JTAG (MTCK, MTDO, MTDI, MTMS) — свободны, если не нужна JTAG-отладка по этим выводам
GPIO43, 44	UART0 (TX, RX) — сообщения загрузчика
GPIO11–20	ADC2 используется также Wi-Fi; для курса выбираем ADC1
GPIO22–25	не существуют

Карта курса

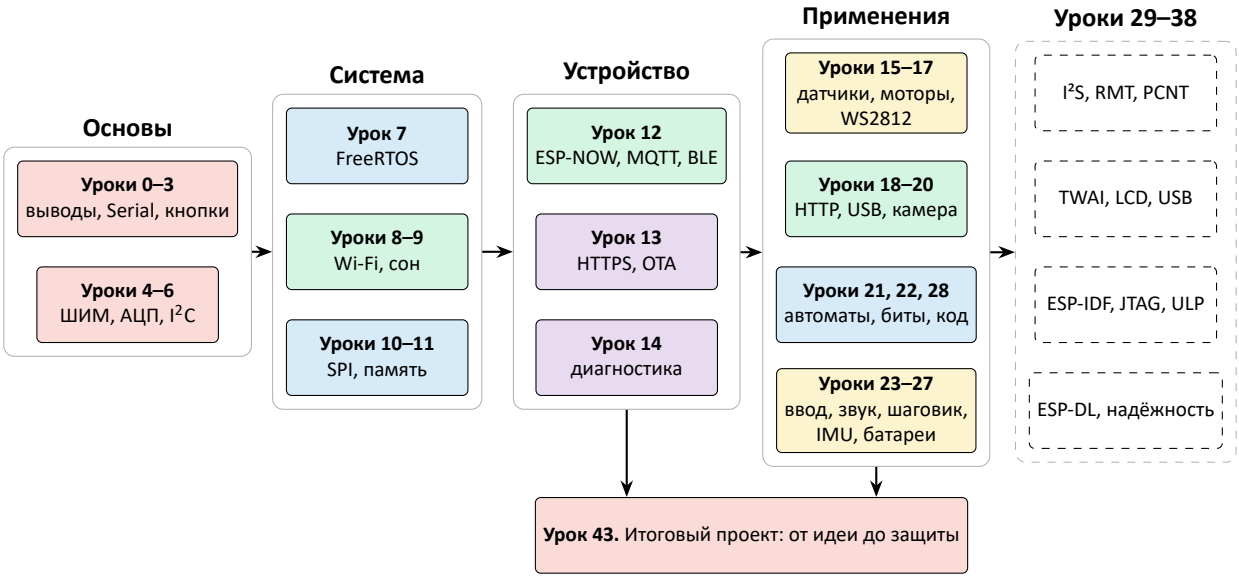


Рис. А.1: Карта курса: четыре ступени, итоговый проект и направления для дальнейшего роста. Расширенный модуль 29–38 изучает аппаратные возможности ESP32-S3 до итогового проекта 43.

Углубление: инструменты и TinyML

Урок 41 переносит проект из Arduino IDE в PlatformIO: версии библиотек, тесты на компьютере и отладка описаны файлом. Урок 42 продолжает урок 37: полный путь нейросети от данных до платы с квантованием и измерением качества, времени и памяти. Оба урока стоят перед итоговым проектом 43 и дают инструменты для его выполнения.

Маршрут веб-интерфейсов

Урок 8 объясняет собственные HTTP/HTML/CSS/JavaScript, 18 — JSON и работу с файлами, 39 — готовые виджеты ESPUI, 40 — выдачу настроек сети. Рабочая панель и портал настройки имеют разные задачи; итоговый проект 43 объединяет их через явные режимы.

Полезные источники

- ESP32-S3 Datasheet и Technical Reference Manual — `https://www.espressif.com/en/support/documents/technical-documents`
- Документация Arduino-ESP32 — `https://docs.espressif.com/projects/arduino-esp32/`

- Руководство ESP-IDF — `https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/`
- Описание платы DevKitC-1 — раздел *ESP32-S3-DevKitC-1 User Guide* на `https://docs.espressif.com`

Приложение В. Компоненты для курса

Список рассчитан на одно рабочее место. Детали из верхней части таблицы нужны почти в каждом уроке; нижняя часть — для отдельных опытов и заданий.

Компонент	Кол-во	Уроки
Плата ESP32-S3-DevKitC-1 (модуль N8R8 или N16R8)	1	все
USB-кабель *с передачей данных* (Type-C или micro-USB по плате)	1	все
Макетная плата на 830 точек и набор проводов «папа–папа»	1	все
Светодиоды 5 мм: красный, жёлтый, зелёный	по 2	1–4, 7, 8
Резисторы 220 Ω	10	1–4
Резисторы 10 kΩ, 33 kΩ, 100 kΩ	по 2	3, 5
Конденсаторы 100 nF	5	5, 10
Тактовые кнопки	3	3, 7, 9, 40
Потенциометр 10 kΩ	1	5–7, 39
OLED-дисплей SSD1306 128×64, интерфейс I²C	1	6, 7
Фоторезистор	1	5 (задание)
RGB-светодиод с общим катодом	1	4 (задание)
Пьезопищалка (пассивная)	1	4 (задание), 24
Сервопривод SG90 и отдельный источник 5 V	1	4
Датчик BME280 или AHT20 (I²C)	1	6 (задание)
Модуль microSD с логикой 3,3 В и карта памяти	1	10
Вторая плата ESP32-S3 (или любая ESP32)	1	2, 12
Мультиметр	1	5, 9, 14
Датчик температуры DS18B20 (корпус или герметичный щуп) и резистор 4,7 kΩ	1	15
Датчик температуры и влажности DHT22 (AM2302), модуль с резистором	1	15
Ультразвуковой дальномер HC-SR04 и резисторы 1 kΩ и 2 kΩ для делителя	1	15
N-MOSFET логического уровня (IRLZ44N, AO3400 или аналог)	2	16
Диоды 1N4007 или 1N4148, резисторы 100 Ω и 10 kΩ	по 2	16
Модуль реле на 1 канал (5 В, с оптопарой)	1	16
Драйвер мотора TB6612FNG или DRV8833 (модуль)	1	16
Коллекторный мотор постоянного тока 3 – 6 V (например, N20 или TT)	1	16
Лента, кольцо или модуль WS2812B на 8 пикселей, резистор 330 Ω, конденсатор 1000 µF	1	17
Блок питания 5 V на 2 A и более (для ленты и мотора)	1	16, 17
Плата ESP32-S3-CAM с камерой OV2640, шлейф и microSD-карта	1	20
Светодиоды для светофора: красный, жёлтый, зелёный (из основного набора)	по 1	21
Сдвиговый регистр 74HC595 и семисегментный индикатор с общим катодом	по 1	22
Модуль энкодера KY-040 и матричная клавиатура 4×4	по 1	23
Шаговый двигатель 28BYJ-48 с платой драйвера ULN2003	1	25
Модуль акселерометра и гироскопа MPU6050 (GY-521)	1	26, 42
Держатель на три батарейки AA или powerbank	1	27
Li-ion аккумулятор с платой защиты и модуль зарядки TP4056 (по решению педагога)	1	27
Цифровой микрофон INMP441	1	29

Логический анализатор с входами 3,3 В	1 на группу	30, 31
CAN-трансиверы с логикой 3,3 В и резисторы 120 Ом	2	32
RS-485-трансиверы с логикой 3,3 В	2	32
SPI-дисплей ST7789 240×240 с CS	1	33
Плата с RGB-экраном / USB Host и управляемым VBUS	по желанию	33, 34
Кнопка и внешний резистор 10 кОм	по 1	36

Совет

Проверяйте, что USB-кабель передаёт данные: многие кабели от зарядных устройств только питают плату, и тогда компьютер её «не видит».

Приложение C. Словарь терминов

Термин	Что означает	Урок
1-Wire	шина с одним проводом данных и подтяжкой; на ней может работать несколько датчиков с уникальными адресами (например, DS18B20)	15
BLE	Bluetooth Low Energy: экономичный радиоканал для связи с телефоном рядом с устройством	12
Backtrace	цепочка адресов функций, которые выполнялись в момент сбоя	14
Brownout	аварийный сброс из-за просадки напряжения питания	14
CS (Chip Select)	линия SPI, которой ведущий выбирает одно ведомое устройство	10
Deep-sleep	глубокий сон: работает только RTC-домен, пробуждение — это перезапуск программы	9
Duty cycle	коэффициент заполнения ШИМ: доля времени, когда сигнал в HIGH	4
DVP	параллельный интерфейс камеры: 8 линий данных и сигналы PCLK, VSYNC, HREF	20
ESP-NOW	протокол Espressif для коротких сообщений между платами без роутера	12
FreeRTOS	операционная система реального времени, на которой работает Arduino-ESP32	7
Git	система хранения истории изменений проекта в виде снимков (коммитов)	28
GPIO	вывод общего назначения: цифровой вход или выход микроконтроллера	0, 1
H-мост	схема из четырёх ключей, позволяющая менять полярность на моторе, то есть направление вращения	16
HID	класс USB-устройств ввода: клавиатура, мышь, мультимедийные клавиши	19
HSV	цветовая модель: оттенок, насыщенность, яркость; удобна для радуги	17
I^2C	двухпроводная шина с адресами устройств: SDA (данные) и SCL (такт)	6
IMU	инерциальный датчик: акселерометр и гироскоп в одном корпусе (например, MPU6050)	26
ISR	обработчик прерывания: короткая функция, которую процессор вызывает по событию	3
JPEG	формат сжатого изображения; камера OV2640 формирует его сама	20
JSON	текстовый формат данных: пары «имя: значение» в фигурных скобках	18
LEDC	периферия ESP32-S3 для формирования ШИМ-сигналов	4
Li-ion	литий-ионный аккумулятор: $3{,}0$ – $4{,}2$ V, требует платы защиты и специальной зарядки	27

Термин	Что означает	Урок
LittleFS	файловая система во Flash: файлы и папки без SD-карты	18
MJPEG	видеопоток из последовательности JPEG-кадров	20
MOSFET	полевой транзистор, управляемый напряжением на затворе; ключ для мощной нагрузки	16
MQTT	протокол обмена сообщениями через брокер по темам (topics)	12
NTP	протокол синхронизации времени с серверами в сети	18
NVS	хранилище «ключ–значение» во Flash для настроек	11
OTA	обновление прошивки по сети без USB-кабеля	13
PSRAM	внешняя оперативная память модуля для больших буферов	11
RMT	периферия ESP32-S3 для точных последовательностей импульсов (WS2812, ИК-пульты)	17
RTC-память	небольшая память, которая сохраняется в deep-sleep	9
SCCB	двухпроводная шина настройки камеры, совместимая с I^2C	20
SPI	быстрая синхронная шина: SCK, MOSI, MISO и CS	10
Strapping-выводы	выводы, уровень которых читается при сбросе и задаёт режим загрузки	3
TLS	протокол шифрования и проверки подлинности сервера (основа HTTPS)	13
UART	асинхронный последовательный порт: TX, RX и общая земля	2
Watchdog	сторожевой таймер: перезагружает чип, если программа зависла	14
WS2812	адресный RGB-светодиод с микросхемой внутри; управляется одним проводом	17
Адресный светодиод	светодиод с микросхемой, цвет которого задаётся данными по одному проводу	17
Акселерометр	датчик ускорения; по направлению силы тяжести показывает наклон	26
АЦП	аналого-цифровой преобразователь: превращает напряжение в число	5
Аттенюатор	делитель на входе АЦП, расширяющий диапазон измеряемых напряжений	5
Бит, байт	бит — один двоичный разряд (0 или 1); байт — восемь битов	22
Гироскоп	датчик скорости поворота вокруг осей	26
Делитель напряжения	два резистора, на среднем выводе которых получается часть входного напряжения	5, 15
Драйвер мотора	микросхема с H-мостом, которая управляет мотором по сигналам платы	16
Дребезг	многократное замыкание-размыкание механического контакта при нажатии	3
Задача	независимая функция с бесконечным циклом под управлением FreeRTOS	7
Класс	тип данных, объединяющий переменные и функции для работы с ними	28
Комплементарный фильтр	способ смешать показания гироскопа и акселерометра, убирая шум и дрейф	26
Конечный автомат	описание поведения устройства через состояния, события и переходы	21
Маска	число с единицами в нужных битах; используется в битовых операциях	22
Мьютекс	«ключ», который защищает общие данные от одновременного доступа задач	7
Оптопара	прибор, передающий сигнал светом и электрически разделяющий цепи	16
Очередь	потокобезопасный буфер для передачи данных между задачами	7
Подтяжка	резистор, задающий уровень на входе, пока к нему ничего не подключено	3
Прерывание	сигнал, по которому процессор откладывает основную программу и вызывает ISR	3

Термин	Что означает	Урок
Регистр	ячейка памяти, биты которой управляют аппаратурой чипа или датчика	22, 26
Реле	электромагнитный выключатель, механически замыкающий контакты	16
Сдвиговый регистр	микросхема, принимающая биты по одному проводу и выдающая их на несколько выходов (74HC595)	22
Тайм-аут	предел времени ожидания, после которого программа перестаёт ждать	15, 18
Таймер аппаратный	счётчик внутри чипа, вызывающий прерывание через точные интервалы	21
Шаговый двигатель	двигатель, поворачивающийся одинаковыми шагами по командам программы	25
ШИМ	широтно-импульсная модуляция: управление средним уровнем долей времени включения	4
Энкодер	ручка без упоров, сообщающая направление и число щелчков поворота	23
I²S	цифровой поток аудио с линиями BCLK, WS и данных	29
RMT	аппаратная передача и приём последовательностей уровней заданной длительности	30
PCNT	аппаратный счётчик фронтов	31
MCPWM	периферия управления связанными PWM-выходами	31
TWAI	контроллер Classical CAN; нужен внешний трансивер	32
DMA	обмен блоками между памятью и периферией без обслуживания каждого байта CPU	29, 33
PHY	физический приёмопередатчик интерфейса	34
JTAG	интерфейс отладки процессора	35
ULP	маломощный сопроцессор с отдельной программой	36
Инференс	применение обученной модели к новому входу	37
Provisioning	первичная выдача устройству настроек и учётных данных	38
WebSocket	постоянный двусторонний канал между браузером и сервером	39
Callback	функция, которую библиотека вызывает при событии	39
ESPUI	конструктор веб-панели из C++ с готовыми элементами	39
Captive portal	страница входа или настройки, на которую направляют подключённый браузер	40
WiFiManager	библиотека выдачи устройству настроек Wi-Fi через портал	40
PlatformIO	система сборки проектов с описанием в файле `platformio.ini`: плата, ядро, библиотеки и их версии	41
Окружение (env)	раздел `[env:имя]` файла `platformio.ini` с настройками одной сборки	41
Модульный тест	короткая программа, подающая в функцию или класс известные данные и проверяющая ответ	41
TinyML	машинное обучение на микроконтроллерах: модель обучают на компьютере, а на плате выполняют только расчёт	42
Нейрон	вычисление: сумма входов с весами плюс смещение, затем функция активации	42
ReLU	функция активации $\max(0, z)$: отрицательное число заменяется нулём	42
Softmax	превращение ответов слоя в доли, дающие в сумме единицу	42
Переобучение	модель запомнила примеры и плохо работает на новых данных	42
Квантование	хранение чисел модели в меньшем числе бит, например int8 вместо float32	42
Матрица ошибок	таблица «настоящий класс — ответ модели»	37, 42

Приложение D. Шпаргалка по C++ для Arduino

Здесь собраны конструкции языка, которые встречаются во всех уроках. Это не учебник C++, а справка: открывайте её, когда забыли, как записывается условие или цикл.

Типы данных

Тип	Диапазон на ESP32-S3	Когда использовать
`int`	от −2 147 483 648 до 2 147 483 647	обычные целые числа, счётчики
`uint8_t`	0…255	байт, яркость, значение цвета
`uint32_t`	0…4 294 967 295	время в `millis()`, большие неотрицательные числа
`float`	около 7 значащих цифр	дробные числа (температура, напряжение)
`bool`	`true` или `false`	флаги, состояния «включено/выключено»
`char`	один символ `'A'`	отдельные символы
`String`	строка произвольной длины	текст в Arduino (удобно, но использует память)

```
int count = 0;                  // целое
float temp = 23.5;              // дробное
bool ledOn = false;             // логическое
uint32_t lastTime = 0;          // для millis()
String name = "ESP32";          // строка
```

Константы

Значение, которое не должно меняться, объявляют с `const`. Компилятор не даст случайно изменить его, а имена принято писать ЗАГЛАВНЫМИ буквами.

```
const int LED_PIN = 4;          // номер вывода
constexpr float VREF = 3.3;     // известно ещё на этапе компиляции
#define BUTTON_PIN 5            // старый способ: просто подстановка текста
```

Операторы

Операторы	Назначение
`+ - * / %`	арифметика; `%` — остаток от деления
`== != < > <= >=`	сравнение (результат `true` или `false`)
`&& || !`	«и», «или», «не»
`= += -= ++ --`	присваивание, увеличение и уменьшение
`& | ^ << >>`	побитовые операции (маски, сдвиги)

```
int a = 7 % 3;                  // 1 (остаток)
bool ok = (a > 0) && (a < 5);   // true
count++;                        // то же, что count = count + 1
mask |= (1 << 3);               // включить бит номер 3
```

Условия: if / else и switch

```
if (temp > 30) {
  digitalWrite(FAN_PIN, HIGH);
} else if (temp < 25) {
  digitalWrite(FAN_PIN, LOW);
} else {
  // между 25 и 30 ничего не меняем
}
```

```
switch (mode) {                 // выбор по значению
  case 0: Serial.println("выкл"); break;
  case 1: Serial.println("авто"); break;
  default: Serial.println("?");
}
```

Циклы: for и while

```
for (int i = 0; i < 5; i++) {   // пять повторов: i = 0..4
  Serial.println(i);
}

uint32_t start = millis();
while (!ready && millis() - start < 5000) {  // ждём не дольше 5 с
  delay(10);
}
```

Цикл `while (true)` без выхода и без тайм-аута в учебных программах не пишите: плата «повиснет» (урок 14).

Функции

Функция даёт имя повторяющемуся действию. У неё есть тип результата, имя и список параметров. Тип `void` означает «ничего не возвращает».

```
int clampPercent(int x) {       // вернуть число, ограниченное 0..100
  if (x < 0) return 0;
  if (x > 100) return 100;
  return x;
}

void blinkOnce(int pin, int ms) {   // ничего не возвращает
  digitalWrite(pin, HIGH);
  delay(ms);
  digitalWrite(pin, LOW);
}

int level = clampPercent(130);  // level = 100
```

Массивы

```
int pins[3] = {4, 5, 6};        // три элемента: индексы 0, 1, 2
int n = sizeof(pins) / sizeof(pins[0]);  // число элементов = 3

for (int i = 0; i < n; i++) {
  pinMode(pins[i], OUTPUT);
}
// pins[3] -- ошибка: выход за границы массива!
```

Структуры

Структура объединяет несколько значений под одним именем: удобно для пакета данных (урок 12) или для описания датчика.

```
struct Reading {
  float temp;
  float humidity;
```

```
  uint32_t time;
};

Reading r = {22.5, 40.0, 0};
r.temp = 23.1;                  // обращение к полю через точку
Serial.println(r.temp);
```

Область видимости и static

Переменная, объявленная внутри функции, существует только в ней и при каждом вызове создаётся заново. Переменная вне функций (глобальная) видна всем. Модификатор `static` внутри функции сохраняет значение между вызовами, но оставляет имя видимым только в этой функции.

```
int total = 0;                  // глобальная: видна везде

void step() {
  int local = 0;                // создаётся заново при каждом вызове
  static int calls = 0;         // помнит значение между вызовами
  local++;
  calls++;
  total += 10;
}
```

Типичные ошибки

Ошибка	Как правильно
`if (x = 5)` — присваивание вместо сравнения	`if (x == 5)`: сравнение записывают двумя знаками
Деление целых: `int a = 1 / 2;` даёт 0	хотя бы одно число дробное: `1.0 / 2` или `(float)1 / 2`
Переполнение: `uint8_t b = 250; b += 10;` даёт 4	выбирайте тип с запасом (`int`, `uint32_t`)
Сравнение времени: `millis() > t + 1000` ломается через 49 суток	пишите вычитание: `millis() - t > 1000`
Выход за границы массива	индексы от 0 до размер-1; проверяйте индекс
Точка с запятой сразу после условия: `if (x > 0);`	блок после условия ставят без неё: `if (x > 0) { …}`
Сравнение строк через `==` с `char*`	для `String` можно `==`, для `char*` — `strcmp()`

```
int half = 7 / 2;               // 3, а не 3.5: целочисленное деление
float f = 7 / 2.0;              // 3.5
uint8_t b = 250;
b += 10;                        // переполнение: теперь b = 4
if (millis() - lastTime >= 1000) { lastTime = millis(); }  // верно
```

Приложение Е. Ответы на вопросы «Проверь себя»

Сначала ответьте сами и только потом сверяйтесь. Ответы краткие: полное объяснение есть в тексте урока.

Вводная глава «Сигналы»

1. Выключатель передаёт два состояния; ручка громкости задаёт плавно меняющуюся величину.
2. 1,5 В находится в неопределённой области: гарантированного цифрового уровня нет.
3. $2^5 = 32$ варианта.
4. $32 + 8 + 4 + 1 = 45$.

Вводная глава «Выводы платы»

1. Это короткое замыкание: ток ограничен почти только сопротивлением проводов и источника.
2. В опытах выбираем ADC1, GPIO1–GPIO10; ADC2 используется также радиомодулем.
3. Мотор требует большего тока, чем может дать GPIO; нужен драйвер или транзистор с защитой.

Урок 0. Знакомство с ESP32-S3

1. Чип — сама микросхема; модуль — чип вместе с Flash-памятью, кварцем и антенной. N16R8 означает 16 МБ Flash и 8 МБ PSRAM.
2. «UART» подключён через микросхему-преобразователь: это самый надёжный путь для прошивки и сообщений. «USB» идёт прямо к чипу и позволяет плате быть USB-устройством и отлаживаться через JTAG.
3. Войти в загрузчик вручную: зажать BOOT, нажать и отпустить RST, отпустить BOOT, запустить загрузку, после неё нажать RST.

Урок 1. Цифровой выход

1. Резистор ограничивает ток, иначе сгорит светодиод или вывод. $R = (U_{\text{вых}} - U_F)/I$, например $(3{,}3 - 2{,}0)/0{,}006 \approx 220$ Ом.
2. Во время `delay()` программа стоит на месте и не может читать кнопки, датчики и сеть.
3. Вычитание беззнаковых чисел даёт верную разницу даже после переполнения счётчика, а сравнение `now > last + INTERVAL` — нет.

Урок 2. Последовательный порт

1. У сторон нет общего тактового сигнала, длительность бита задана заранее выбранной скоростью. При разных скоростях биты читаются не в те моменты, и получается «мусор».
2. TX одного устройства соединяют с RX другого и наоборот. Общая земля нужна, потому что напряжение сигнала измеряется относительно неё.
3. Чтобы не ждать данных, которых ещё нет: программа читает только пришедшее и сразу продолжает работу.

Урок 3. Цифровой вход

1. Вход, никуда не подключённый, ловит помехи и читается случайно. Подтягивающий резистор задаёт уровень по умолчанию.
2. Из-за дребезга: контакты при замыкании несколько раз подпрыгивают, и программа видит серию нажатий.
3. Можно изменить переменную или установить флаг. Нельзя вызывать `delay()`, `Serial.print()` и другие медленные функции.

Урок 4. ШИМ

1. Это доля периода, когда сигнал в HIGH. Среднее напряжение равно этой доле, умноженной на $3{,}3$ V.
2. Чем выше частота, тем меньше доступное разрешение: $f_{\max} \approx 80\,\text{MHz}/2^{\text{бит}}$.

3. Глаз воспринимает яркость нелинейно; без коррекции яркость быстро растёт в начале шкалы и почти не меняется в конце.

Урок 5. АЦП

1. Она учитывает заводскую калибровку чипа и сразу возвращает милливольты.
2. ADC2 используется также Wi-Fi, и доступ зависит от режима драйвера; для курса выбираем ADC1 (**GPIO1**–**GPIO10**).
3. Через делитель из двух резисторов, который понижает напряжение до допустимого. Конденсатор сглаживает помехи на входе АЦП.

Урок 6. Шина I²C

1. Устройства могут только притягивать линию к земле; в HIGH её возвращают подтягивающие резисторы.
2. Адреса всех ответивших устройств. Если ответа нет — ошибка в питании, проводах, выводах или адресе.
3. Отправка буфера занимает десятки миллисекунд; постоянная перерисовка затормозит всю программу.

Урок 7. FreeRTOS

1. У задачи свой бесконечный цикл, стек и приоритет; планировщик сам переключает задачи. `loop()` — это одна из таких задач.
2. Очередь — чтобы передавать данные от одной задачи другой. Мьютекс — чтобы защитить общее состояние, которое читают и меняют несколько задач.
3. По перезагрузке с сообщением `Stack canary` и по малому остатку, который показывает `uxTaskGetStackHighWaterMark()`.

Урок 8. Wi-Fi и веб-сервер

1. Станция подключается к существующей сети роутера; точка доступа сама создаёт сеть, к которой подключаются другие.
2. Браузер отправляет запрос GET /toggle, плата выполняет обработчик и отвечает перенаправлением 303, браузер заново запрашивает главную страницу.
3. Чтобы программа не зависла навсегда, если сети нет или пароль неверен.
4. GET читает ресурс, POST отправляет данные или команду. HTTP не шифрует ни один метод; для шифрования нужен HTTPS (урок 13).
5. HTML задаёт структуру и содержание, CSS — оформление, JavaScript — поведение в браузере.
6. `border-radius: 50%;` при одинаковых ширине и высоте элемента.

Урок 9. Сон

1. После light-sleep программа продолжается со следующей строки; после deep-sleep чип перезапускается и `setup()` выполняется заново.
2. Переменные в RTC-памяти, объявленные с `RTC_DATA_ATTR`.
3. От длительности и тока активной фазы (особенно работы Wi-Fi): ток сна на несколько порядков меньше.

Урок 10. SPI

1. У шины нет адресов: ведущий выбирает устройство линией CS, и отвечать вправе только выбранное.
2. Длинные провода и макетная плата искажают фронты сигналов; на высокой частоте появляются ошибки.
3. До закрытия часть данных остаётся в буфере и пропадёт при отключении питания.

Урок 11. Память

1. Счётчик пробуждений — в RTC-памяти, пароль — в NVS, большой буфер — в PSRAM.

2. Ресурс ячеек Flash ограничен; частая запись быстро их изнашивает.
3. `psramFound()` должен вернуть `true`, а указатель после `ps_malloc()` не должен быть `nullptr`.

Урок 12. ESP-NOW, MQTT, BLE

1. Пульт без роутера — ESP-NOW; умный дом — MQTT; настройка с телефона — BLE.
2. Брокер хранит последнее такое сообщение и сразу отдаёт его каждому новому подписчику.
3. Сетевые функции медленные и могут блокироваться; в обработчике это приводит к срабатыванию watchdog и сбоям.

Урок 13. HTTPS и OTA

1. У сертификата есть срок действия; не зная текущей даты, устройство не может его проверить.
2. Соединение шифруется, но подлинность сервера не проверяется: устройство не отличит настоящий сервер от подмены.
3. Новая прошивка пишется в свободный слот, а рабочая остаётся нетронутой: при сбое обновления есть к чему вернуться.

Урок 14. Диагностика

1. Адреса функций меняются при каждой сборке; чужой ELF даст правдоподобные, но ложные строки.
2. О просадке питания. Искать нужно в источнике, кабеле и мощной нагрузке, а не в программе.
3. Регулярно отдавать процессор: `vTaskDelay()`, ожидание очереди с тайм-аутом, никаких долгих циклов без пауз.

Урок 15. Датчики

1. Питание, логические уровни, интерфейс, время и период опроса, предельные значения.
2. Одно измерение занимает до двух секунд; при более частых запросах датчик возвращает старые данные или ошибку.
3. ECHO выдаёт 5 V, а вход платы рассчитан на $3{,}3$ V. Расстояние в сантиметрах: время эха в микросекундах × 0,0343 / 2.

Урок 16. Силовая нагрузка

1. Мотору нужны сотни миллиампер и другое напряжение, а вывод выдерживает лишь десятки миллиампер при $3{,}3$ V.
2. Он гасит выброс напряжения при отключении катушки. Диод ставят параллельно катушке, катодом к плюсу питания.
3. Напряжение управляющего сигнала отсчитывается от земли; без общей земли драйвер «не понимает» уровни платы.

Урок 17. Адресные светодиоды

1. Длительностью HIGH: короткий импульс — ноль, длинный — единица; длина бита одинакова.
2. До $30 \cdot 60$ мА $= 1{,}8$ А. Нужен отдельный источник 5 V и ограничение яркости в программе.
3. Резистор гасит выбросы и отражения в линии данных, конденсатор сглаживает бросок тока при включении.

Урок 18. HTTP, JSON, время, файлы

1. Плата узнаёт IP-адрес сервера через DNS, устанавливает соединение, отправляет запрос и ждёт код состояния и тело ответа.
2. Ответ может прийти неполным или не в формате JSON; без проверки программа будет работать с мусором.
3. NVS — небольшие настройки «ключ–значение»; LittleFS — файлы во Flash платы; SD-карта — большие объёмы на съёмном носителе.

Урок 19. Нативный USB

1. «UART» даёт только последовательный порт через микросхему-мост; нативный USB идёт прямо к чипу, который может стать клавиатурой, мышью и другими устройствами.
2. Tools → USB Mode: USB-OTG (TinyUSB).
3. Отключить кабель от разъёма «USB», войти в загрузчик кнопками BOOT и RST и залить исправленную программу через «UART».

Урок 20. Камера

1. Восемь линий данных, тактовый сигнал PCLK, сигналы кадра VSYNC и строки HREF, опорная частота XCLK. По SCCB (это I^2C) плата записывает настройки сенсора.
2. Он возвращает буфер кадра драйверу. Без него свободные буферы закончатся, и следующий кадр получить не удастся.
3. PSRAM (кадр хранится в ней), схема разделов Huge APP (программа с камерой большая) и USB CDC On Boot (чтобы видеть сообщения).

Урок 21. Таймеры и конечные автоматы

1. Таймер срабатывает по прерыванию со строгим периодом, даже если основная программа занята; проверка `millis()` зависит от скорости `loop()`.
2. Состояние — что устройство делает сейчас (горит красный). Событие — что произошло (истекло время, нажата кнопка). Переход — смена состояния по событию (красный → красный с жёлтым).
3. Вместе с состоянием нужно обновить время входа и выходы; если это сделано в одном месте, ничего нельзя забыть.

Урок 22. Биты, байты и регистры

1. $32 + 8 + 4 + 1 = 45$; по четвёркам: 0010 = 2, 1101 = D, то есть `0x2D`.
2. `x &= ~(1 << 5);`
3. Пока биты вдвигаются, они «бегут» по выходам. Защёлка обновляет все восемь выходов одновременно, когда передача закончена.

Урок 23. Органы управления

1. В момент спада сигнала A читают уровень B: HIGH — одно направление, LOW — другое.
2. Кнопки стоят на пересечениях 4 строк и 4 столбцов; программа опрашивает строки по очереди.
3. «Спокойный» уровень зависит от длины провода, влажности и питания, поэтому порог нельзя задать заранее.

Урок 24. Звук

1. Пассивной нужен сигнал нужной частоты, активная пищит сама от постоянного напряжения. Если подать 3,3 V: пассивная щёлкнет, активная запищит.
2. Ровно в два раза.
3. Пока идёт `delay()`, программа не реагирует на кнопки и датчики.

Урок 25. Шаговый двигатель

1. Обычный мотор просто вращается и не знает положения; сервопривод держит заданный угол в пределах 180°; шаговый вращается сколько угодно, но шагами, которые можно считать.
2. $4096 \cdot 45/360 = 512$ полушагов.
3. Ротор не успевает за полем и пропускает шаги; программа продолжает считать и теряет настоящее положение вала.

Урок 26. Акселерометр и гироскоп

1. `(int16_t)((старший << 8) | младший)`.
2. Малая ошибка скорости при накоплении складывается, и угол постепенно уплывает (дрейф).
3. Доля акселерометра вырастет: дрейф исчезнет быстрее, но в угол попадут шум и вибрация.

Урок 27. Автономное питание

1. Перезаряд, глубокий разряд и короткое замыкание повреждают аккумулятор и могут привести к возгоранию; плата защиты отключает его в этих случаях.
2. Напряжение до $4{,}2\,\mathrm{V}$ выше предела АЦП. Через большие резисторы течёт ничтожный ток, а малые сами разряжали бы аккумулятор.
3. Около $1000/20 = 50$ часов, на практике на 20–30 % меньше.

Урок 28. Порядок в коде

1. Данные и функции собраны вместе и защищены от случайной порчи; для второй кнопки достаточно создать второй объект.
2. В `.h` — объявление класса (что он умеет), в `.cpp` — тела функций (как он это делает).
3. После каждого рабочего шага и перед рискованной правкой; в описании — что теперь умеет программа.

Урок 29. Цифровой звук: I²S и микрофон

1. I^2S передаёт аудио с тактами BCLK/WS; I^2C — адресованные команды по SDA/SCL.
2. Полезные данные и длина передаваемого слота различаются; важны знак и выравнивание.
3. Получены относительные единицы отсчётов. Для дБ SPL нужны чувствительность и акустическая калибровка.

Урок 30. Точные импульсы: периферия RMT

1. 250 мкс.
2. PCNT считает фронты, а не все длительности и порядок уровней.
3. До подтверждения окончания; раньше нельзя освобождать или изменять его.

Урок 31. Счётчики и движение: PCNT и MCPWM

1. 250 при точном окне 250 мс.
2. После обнуления на границе обычная разность показаний будет неверной.
3. Пауза между переключениями плеч моста, снижающая риск их одновременного открытия.

Урок 32. Проводная сеть: TWAI/CAN и RS-485

1. Нужен трансивер логических и дифференциальных уровней.
2. На двух физических концах, с учётом уже установленных резисторов.
3. Функция ставит кадр в очередь; факт приёма и обработки нужно подтвердить.

Урок 33. Цветной экран: SPI, RGB, память и DMA

1. 153600 байт; два полных буфера — 307200 байт.
2. SPI записывает команды и пиксели в контроллер, RGB-панель требует потока с синхронизацией.
3. У драйверов разные требования к доступу, выравниванию и буферам.

Урок 34. USB глубже: CDC, накопитель и роль Host

1. Да: это поток байтов, границы задаёт приложение.
2. Файловые системы и кэши получают независимые несогласованные изменения.
3. Нужны подходящее питание VBUS, соединение, ограничение тока и стек класса.

Урок 35. ESP-IDF и JTAG: увидеть выполнение

1. Типы операндов определяют деление; дробь теряется до присваивания float.
2. Он содержит адреса, символы и связь со строками именно этой сборки.
3. Задачи останавливаются, а тайм-ауты удалённых устройств и часть периферии продолжаются.

Урок 36. ULP: наблюдение, пока основные ядра спят

1. Нужны компилятор RISC-V, встраивание образа и настройка RTC-памяти.
2. Нет: момент нажатия имеет произвольную фазу относительно опроса.
3. Ток моста USB, стабилизатора, LED и модулей может доминировать.

Урок 37. Распознавание на устройстве: данные, модель и ESP-DL

1. Обучение настраивает параметры, инференс применяет их.
2. Отсчёты зависимы: получается утечка информации и завышенная оценка.
3. Нужны ещё входы, промежуточные тензоры, кадры, стек и сеть.

Урок 38. От макетки к устройству: запуск, восстановление и защита

1. Нужны конфигурация загрузчика и самопроверка/подтверждение новой прошивки.
2. Первый проверяет подлинность запускаемого кода, вторая защищает содержимое внешней Flash.
3. Переключение при повторе меняет результат; повтор установки того же значения идемпотентен.

Урок 39. ESPUI: веб-панель из C++

1. ID обозначает элемент интерфейса. GPIO обозначает электрический вывод; связь между ними задаёт программа.
2. Callback может работать в сетевой задаче: он проверяет ввод и передаёт команду, а изменение модели и долгая работа выполняются в назначенном месте.
3. Страница загружается по HTTP, команды идут по отдельному WebSocket. В ESPUI 2.2.4 его аутентификация зависит от отдельного флага, выключенного по умолчанию.

Урок 40. WiFiManager: настройка сети

1. Первый позволяет войти во временную сеть настройки; второй нужен устройству для подключения к выбранному роутеру.
2. В неблокирующем режиме процесс обслуживания продолжается после setup(), поэтому объект должен сохраняться.
3. Два сервера не могут независимо занять один адрес и порт. Нужен общий сервер либо раздельные режимы с освобождением ресурсов.

Урок 41. PlatformIO: профессиональная сборка проекта

1. Проект собирается одинаково на любом компьютере и через год: версии ядра и библиотек не зависят от того, что установлено в среде.
2. Время передают снаружи, поэтому тест может подать любую последовательность моментов и уровней, не дожидаясь настоящих миллисекунд и не имея платы.
3. В Git: `platformio.ini`, `src`, `include`, `lib`, `test`. Не в Git: `.pio` (создаётся заново), файлы с паролями.

Урок 42. TinyML: нейросеть внутри микроконтроллера

1. Выигрыш: вес занимает 1 байт вместо 4, расчёт быстрее. Риск: потеря точности, поэтому качество измеряют после квантования.
2. Сеть могла запомнить примеры (переобучение). Качество показывают только данные, которых не было при обучении, лучше целые отдельные сеансы.
3. Softmax всегда распределяет единицу между известными классами, даже для незнакомого входа. Порог уверенности и повтор решения защищают от ложных срабатываний.

Урок 43. Итоговый проект

1. Она интересна автору, решаема за 4–6 недель и опирается на уже освоенные уроки.
2. Это измеримая проверка, по которой видно, что проект готов. Список функций говорит, что устройство должно делать, а критерий — как убедиться, что оно это делает.
3. Входы, выходы, микроконтроллер, связь и память. Питание рисуют отдельно, потому что от него зависит работа всех блоков и именно в нём чаще всего ошибаются.

Приложение F. Материалы для педагога

Этот раздел помогает встроить пособие в программу объединения. Все цифры примерные: распределение часов зависит от возраста и подготовки группы, числа рабочих мест и наличия деталей.

Кому адресован курс

Курс рассчитан на обучающихся, которые впервые знакомятся с микроконтроллерами (ориентировочно 7–11 классы). Знание языка C++ не требуется: нужные конструкции вводятся по мере необходимости и собраны в приложении «Шпаргалка по C++». Желательны уверенная работа с компьютером и знание физики на уровне понятий «напряжение», «ток», «сопротивление».

Планируемые результаты

По окончании курса обучающийся:

- соблюдает правила безопасной работы с электронными компонентами и источниками питания;
- читает простые принципиальные схемы, собирает их на макетной плате и проверяет мультиметром;
- пишет программы для ESP32-S3: работает с цифровыми и аналоговыми выводами, ШИМ, таймерами и прерываниями, строит программу без блокирующих задержек;
- подключает датчики и исполнительные устройства по интерфейсам UART, I^2C, SPI, 1-Wire;
- использует Wi-Fi и Bluetooth LE для обмена данными и управления устройством;
- находит причину неисправности по сообщениям платы, измерениям и документации;
- самостоятельно проектирует, собирает, документирует и защищает собственное устройство.

Примерное тематическое планирование (72 часа)

Вариант рассчитан на год занятий по два академических часа в неделю.

Раздел	Главы пособия	Часы
Введение: безопасность, макетная плата, сигналы, выводы и программа	вводные главы	2
Основы: выводы, Serial, кнопки, ШИМ, АЦП, I^2C	уроки 0–6	14
Система: FreeRTOS, Wi-Fi, сон, SPI, память	уроки 7–11	10
Связь и надёжность: ESP-NOW, MQTT, BLE, HTTPS, OTA, диагностика	уроки 12–14	6
Датчики и исполнительные устройства	уроки 15–17, 23–26	14
Сеть, USB и камера	уроки 18–20	6
Приёмы программирования: таймеры и автоматы, биты, порядок в коде	уроки 21, 22, 28	6
Автономное питание	урок 27	2
Итоговый проект: разработка и защита	урок 43	12
Всего		**72**

Сокращённый вариант (36 часов): вводные главы (2 ч), уроки 0–9 (20 ч), уроки 15 и 16 (4 ч), мини-проект по упрощённой схеме урока 43 (10 ч).
Урок 8 расширен блоком HTML, CSS и JavaScript. В короткой программе достаточно запустить готовую панель; подробный разбор и задания можно перенести в проектные часы или отдельный модуль.

Уроки не обязательно проходить подряд. После урока 6 порядок можно менять: в строке «Опирается на» в начале каждого урока перечислено, что нужно знать заранее. Уроки 21, 22 и 28 полезно пройти раньше, если группа быстро осваивает материал.

Как построить занятие

Этап	Минуты	Содержание
Вход в тему	10	вопросы «Проверь себя» прошлого урока; раздел «Сначала» нового урока
Объяснение	20	теория с рисунками пособия; разбор готового примера по строкам
Сборка и запуск примера	25	каждый собирает схему и добивается работы примера; раздел «Если не работает»
Самостоятельный опыт	25	изменение примера под новую задачу; помощь педагога по запросу
Итог	10	что получилось, типичные ошибки, задание на дом по уровню (★–★★★★)

Формы контроля

- **Текущий контроль** — вопросы «Проверь себя» (ответы в приложении) и работающий самостоятельный опыт на каждом занятии.
- **Промежуточный контроль** — шесть заданий четырёх уровней сложности в конце уроков (два ★, два ★★, одно ★★★, одно ★★★★); достаточно одного выполненного задания на урок, уровень обучающийся выбирает сам, но рекомендуйте идти по порядку и не пропускать лёгкие.
- **Итоговая аттестация** — защита проекта по критериям урока 43.

Организационные советы

- Первое занятие начинайте с главы «Техника безопасности и рабочее место»; правила стоит повторять перед уроками 16, 25 и 27, где появляются моторы и аккумуляторы.
- Одно рабочее место — один комплект из приложения «Компоненты для курса». Работа в парах допустима, но каждый должен сам собрать схему хотя бы через раз.
- Имя и пароль учебной сети Wi-Fi выдавайте на занятии; не используйте школьную административную сеть. Для уроков 8, 12, 13 и 18 удобна отдельная точка доступа $2{,}4$ GHz.
- Перед уроками 15, 17 и 18 заранее установите библиотеки на всех компьютерах: скачивание во время занятия отнимает много времени.
- Если деталей не хватает, часть опытов уроков 0–6 и 21–23 можно выполнять в симуляторе Wokwi.
- Урок 20 (камера) требует другой платы; его можно провести как демонстрацию.

Расширение по ESP32-S3: ещё 40 часов

Базовое планирование на 72 часа выше сохраняется как отдельный маршрут: уроки 0–28 и проект 43. Новые уроки 29–38 образуют дополнительный модуль на 40 часов: по четыре часа на тему. Маршрут без веб-модуля 39–40 — 112 часов. Он требует отдельных рабочих мест для проводных шин, аудиомодулей, экранов и установленной ESP-IDF для уроков 35–36. На 36-часовом маршруте новые темы можно использовать как демонстрации, без обещания полного освоения.

Изучайте модуль 29–38 до веб-модуля 39–40 и проекта 43; для JTAG и ULP заранее проверьте ESP-IDF и доступ к нативному USB.

Веб-панели и настройка: ещё 8 часов

Урок 39 (ESPUI) — четыре часа: готовые элементы, callbacks, очередь, два браузера и телеметрия. Урок 40 (WiFiManager) — четыре часа: режимы настройки и работы, окно доступа и восстановление. Полный маршрут теперь 120 часов: базовые 72, расширение ESP32-S3 40, веб-модуль 8. При недостатке времени минимальную ESPUI-панель можно показать после урока 8; очередь требует урока 7. Зависимости заранее установите на все рабочие места. Для проверки нужны два браузера и отдельная учебная сеть 2,4 ГГц; аппаратные приводы в первом опыте замените LED.

Инструменты и TinyML: ещё 8 часов

Уроки 41 (PlatformIO) и 42 (TinyML) — по четыре часа; они стоят после веб-модуля и перед итоговым проектом 43 и дают инструменты для него. PlatformIO удобен всем, кто ведёт проект в команде или в Git. TinyML опирается на урок 26 (акселерометр) и урок 37. Для урока 41 заранее скачайте пакеты PlatformIO (около гигабайта) и проверьте, что на компьютерах есть компилятор C++ для тестов на компьютере. Для урока 42 нужен Python с NumPy; обучение сети занимает секунды, а основное время уходит на сбор данных. Оба урока можно пропустить без потери связности проекта. Полный маршрут: 128 часов.