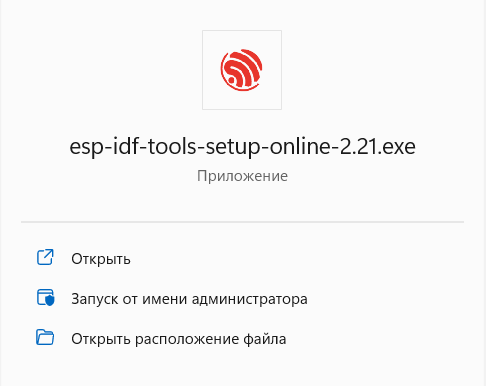
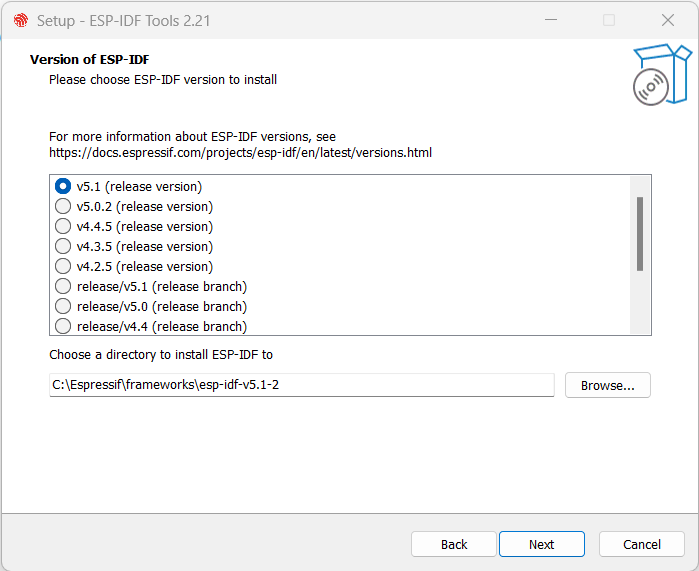
Инструкция для работы с МК ESP32 DevKitC V4 Wroom 32U

**Задача 1. Научиться работать с ESP32, установить среду разработки, научиться запускать дебаг (при наличии)**

1. Для того, чтобы скачать инсталлер для работы с ESP-IDF, перейдите по [ссылке](https://dl.espressif.com/dl/esp-idf/?idf=4.4) и загрузите Universal Online Installer.
2. В меню «Пуск» введите ESP и нажмите открыть

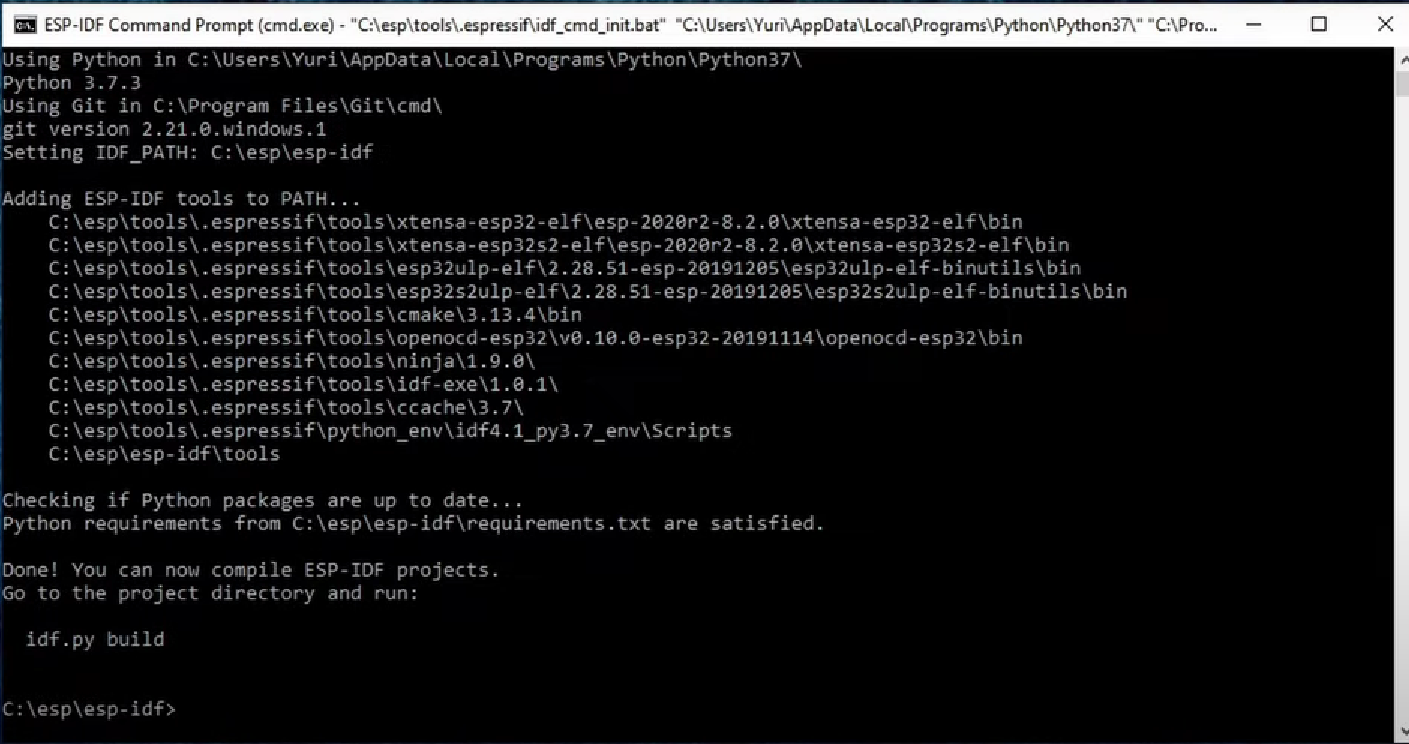


1. Откройте скачанный файл и нажимайте «Далее (Next)» пока не увидите данное окно.



Выберите версию v5.1 (release version), укажите путь для вашего проекта и дождитесь загрузки.

1. После установки откройте ESP-IDF Command Prompt, либо просто нажмите Close.
2. Дождитесь установки всех компонентов.



1. Подключите МК к ПК с помощью кабеля USB-A to Micro-USB.
2. Соберите схему со светодиодом и резистором, подключив его к пину 21.
3. Зайдите в путь проекта командой cd PATH/TO/PROJECT, далее в папку Espressif/frameworks/esp-idf-v5.1/examples/get-started/blink/main и откройте файл blink\_example\_main.c и в коде измените 22 строку, где написано #define BLINK\_GPIO и поставьте значение 21.
4. Введите в консоли команду cd.. и введите команду idf.py build.
5. После завершения загрузки введите idf.py -p (PORT) flash , где PORT – COM порт для подключения микроконтроллера. Когда в консоли появится надпись Connection… необходимо зажать кнопку BOOT на плате.
6. Для отображения информации о МК введите idf.py -p (PORT) monitor.
7. Как итог: светодиод моргает с периодичностью 1 сек.

Дебаг возможен только сторонними аппаратными средствами.

Удобнее будет воспользоваться VS Code в качестве текстового редактора, а также чтобы создавать новые проекты со всеми необходимыми для работы файлами.

Переходим во вкладку Extensions и скачиваем Espressif IDF. После установки нажимаем в верхнем меню View – Command Pallete – New Project. После создания проекта, написания кода – нажимаем на шестеренку снизу ESP-IDF SDK Configuration Editor (menuconfig).

**Задача 2. Научиться пользоваться портами GPIO (например кнопкой и светодиодом). (Архив 2\_1\_blink.rar и Архив 2\_2\_blink.rar)**

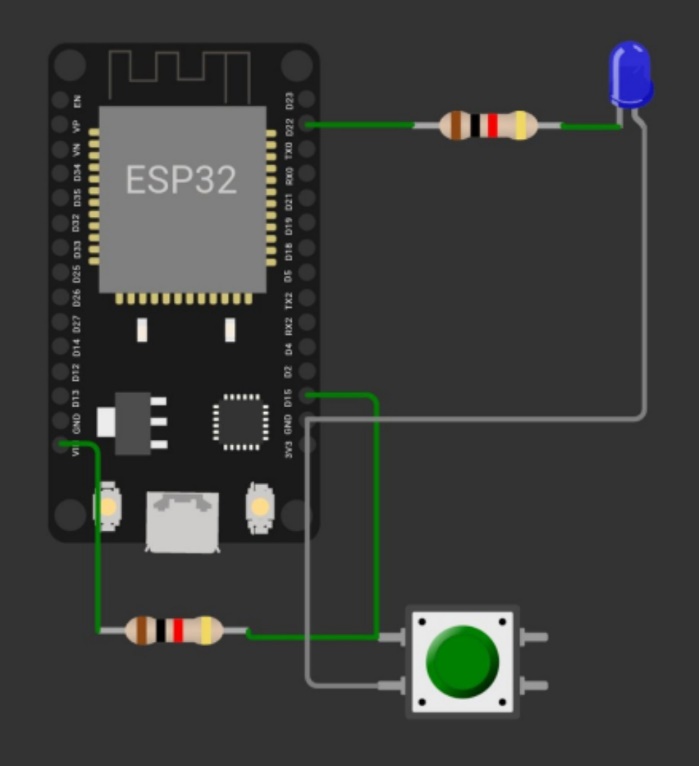
Светодиод отвечает за питание на плате, не настраиваемый.

На плате имеются две кнопки: EN и BOOT. EN отвечает за перезагрузку платы. BOOT необходима для прошивки платы.

В качестве примера мы собрали схему, состоящую из подтягивающего резистора на 3 кОм, резистор для светодиода 3 кОм и сам светодиод, тактовая кнопка на два выхода.

По схеме мы подключили к пину 22 резистор, последовательно подключили светодиод, вторая нога которого уходит на землю. Также к 16 пину подключили кнопку через подтягивающий резистор.

*Схема*:



*Код для МК*: (**Архив 2\_1\_blink.rar)**

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "esp\_log.h"

#include "led\_strip.h"

#include "sdkconfig.h"

static const char \*TAG = "second task";

#define BLINK\_GPIO 22 // пин для светодиода

#define BUTTON\_GPIO 15 // пин для кнопки

void app\_main(void)

{

esp\_rom\_gpio\_pad\_select\_gpio(BLINK\_GPIO); // указываем пин

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT); // указываем вход-выход

esp\_rom\_gpio\_pad\_select\_gpio(BUTTON\_GPIO);

gpio\_set\_direction(BUTTON\_GPIO, GPIO\_MODE\_INPUT);

gpio\_pullup\_en(BUTTON\_GPIO); // проверка на нажатие

gpio\_pulldown\_dis(BUTTON\_GPIO); // проверка на отжатие

while (1) {

gpio\_set\_level(BLINK\_GPIO, 0); // выключаем светодиод

vTaskDelay(250 / portTICK\_PERIOD\_MS); // задержка 250 мс

gpio\_set\_level(BLINK\_GPIO, 1); // включаем светодиод

vTaskDelay(250 / portTICK\_PERIOD\_MS);

if (gpio\_get\_level(BUTTON\_GPIO) != 0) vTaskDelay(250 / portTICK\_PERIOD\_MS); // если кнопка нажата, то задержка 250 мс, иначе 3с

else vTaskDelay(3000 / portTICK\_PERIOD\_MS);

}

}

В конечном итоге светодиод будет моргать с задержкой 0.25 секунд, а при нажатии кнопки задержка будет составлять 3 секунды, после чего задержка вернется в начальное значение.

Код ниже меняет задержку по нажатию на кнопку. Если кнопка не нажата, то задержка 0.25 с, если кнопка была нажата, то теперь задержка 3 секунды. **(Архив 2\_2\_blink.rar)**

void app\_main(void) {

esp\_rom\_gpio\_pad\_select\_gpio(BLINK\_GPIO); // указываем пин

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT); // указываем вход-выход

esp\_rom\_gpio\_pad\_select\_gpio(BUTTON\_GPIO);

gpio\_set\_direction(BUTTON\_GPIO, GPIO\_MODE\_INPUT);

gpio\_pullup\_en(BUTTON\_GPIO);

gpio\_pulldown\_dis(BUTTON\_GPIO);

int diplodok = 0;

while (1) {

if (gpio\_get\_level(BUTTON\_GPIO) == 0){

diplodok++;

printf("%d\n", diplodok);

}

if ((diplodok % 2) == 0)

{

gpio\_set\_level(BLINK\_GPIO, 0);

vTaskDelay(250 / portTICK\_PERIOD\_MS);

gpio\_set\_level(BLINK\_GPIO, 1);

vTaskDelay(250 / portTICK\_PERIOD\_MS);

}

else

{

gpio\_set\_level(BLINK\_GPIO, 0);

vTaskDelay(3000 / portTICK\_PERIOD\_MS);

gpio\_set\_level(BLINK\_GPIO, 1);

vTaskDelay(3000 / portTICK\_PERIOD\_MS);

}

}

}

**Задача 3. Научиться использовать прерывания. (Архив 3\_interrupt.rar)**

Код ниже каждые 3 секунды срабатывает прерывание и вызывается обработчик прерываний – функция timer\_isr\_callback. **(Архив 3\_interrupt.rar)**

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "esp\_log.h"

#include "led\_strip.h"

#include "sdkconfig.h"

#include "driver/timer.h"

static const char \*TAG = "third task";

#define BLINK\_GPIO 22

static void IRAM\_ATTR timer\_isr\_callback(void \*args) { // функция, срабатывающая по прерыванию

ESP\_DRAM\_LOGW("timer0", "Hardware timer alarm!"); // запись в логи о срабатывании прерывания

gpio\_set\_level(BLINK\_GPIO, 1);

}

void app\_main(void)

{

esp\_rom\_gpio\_pad\_select\_gpio(BLINK\_GPIO);

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT);

timer\_config\_t config = { // конфиг для таймера

.divider = 80, // делитель

.counter\_dir = TIMER\_COUNT\_UP,

.counter\_en = TIMER\_PAUSE,

.alarm\_en = TIMER\_ALARM\_EN,

.auto\_reload = TIMER\_AUTORELOAD\_EN,

};

timer\_init(TIMER\_GROUP\_0, TIMER\_0, &config); // инициализация

timer\_set\_counter\_value(TIMER\_GROUP\_0, TIMER\_0, 0); // начало отсчета таймера с 0

timer\_set\_alarm\_value(TIMER\_GROUP\_0, TIMER\_0, 3\*1000\*1000); // конец таймера 3 секунды

timer\_isr\_callback\_add(TIMER\_GROUP\_0, TIMER\_0, timer\_isr\_callback, NULL, 0); // привязываем обработчик для прерывания

timer\_enable\_intr(TIMER\_GROUP\_0, TIMER\_0);

timer\_start(TIMER\_GROUP\_0, TIMER\_0);

ESP\_LOGI("main", "Hardware timer stated");

while(1){

gpio\_set\_level(BLINK\_GPIO, 0);

vTaskDelay(500 / portTICK\_PERIOD\_MS);

ESP\_LOGI("main", "vTaskDelay(500) timeout");

}

}

**Задача 4. Научиться подключаться по Wi-Fi к смартфону, реализовать управление платой через смартфон (например, при нажатии кнопки на смартфоне – загорается светодиод, подключенный к плате, и наоборот).** (**архив 4\_1\_wifi.rar и архив 4\_2\_wifi.rar)**

Код ниже инициализирует Wi-Fi модуль на плате и переключает его в режим точки доступа. Название точки доступа – «ESP32», пароль «password».

Функция wifi-event-handler – обработчик нашей точки доступа, при подключении пользователя к ней будет выдаваться сообщение в логи о том, что подключен пользователь с определенным MAC-адресом, ему назначен идентификатор, а также будет загораться светодиод, подключенный через резистор к 22 пину.

Функция wifi\_init\_softap инициализирует точку доступа, устанавливает обработчик (в функции esp\_event\_handler\_instance\_register()). В структуре wifi\_config задается название, длина названия, пароль, максимальное количество подключений.

**(архив 4\_1\_wifi.rar)**

#include <string.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_mac.h"

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "lwip/err.h"

#include "lwip/sys.h"

#include <stdio.h>

#include "driver/gpio.h"

#include "led\_strip.h"

#include "sdkconfig.h"

#define EXAMPLE\_ESP\_WIFI\_SSID "ESP32" // имя сети

#define EXAMPLE\_ESP\_WIFI\_PASS "password" // пароль

#define EXAMPLE\_ESP\_WIFI\_CHANNEL CONFIG\_ESP\_WIFI\_CHANNEL

#define EXAMPLE\_MAX\_STA\_CONN CONFIG\_ESP\_MAX\_STA\_CONN

static const char \*TAG = "wifi softAP";

#define BLINK\_GPIO 22

static void wifi\_event\_handler(void\* arg, esp\_event\_base\_t event\_base,

int32\_t event\_id, void\* event\_data)

{

esp\_rom\_gpio\_pad\_select\_gpio(BLINK\_GPIO); // выбираем пин светодиода

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT); // указываем выход

if (event\_id == WIFI\_EVENT\_AP\_STACONNECTED) { // при подключении

wifi\_event\_ap\_staconnected\_t\* event = (wifi\_event\_ap\_staconnected\_t\*) event\_data;

ESP\_LOGI(TAG, "station "MACSTR" join, AID=%d",

MAC2STR(event->mac), event->aid); // данные для логов

gpio\_set\_level(BLINK\_GPIO, 1); // загорается светодиод

} else if (event\_id == WIFI\_EVENT\_AP\_STADISCONNECTED) { // при отключении

wifi\_event\_ap\_stadisconnected\_t\* event = (wifi\_event\_ap\_stadisconnected\_t\*) event\_data;

ESP\_LOGI(TAG, "station "MACSTR" leave, AID=%d",

MAC2STR(event->mac), event->aid);

gpio\_set\_level(BLINK\_GPIO, 0); // выключается светодиод

}

}

void wifi\_init\_softap(void) // инициализация

{

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

esp\_netif\_create\_default\_wifi\_ap();

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT();

ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));

// прописываем обработчик для подключений

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(WIFI\_EVENT,

ESP\_EVENT\_ANY\_ID,

&wifi\_event\_handler,

NULL,

NULL));

wifi\_config\_t wifi\_config = {

.ap = {

.ssid = EXAMPLE\_ESP\_WIFI\_SSID,

.ssid\_len = strlen(EXAMPLE\_ESP\_WIFI\_SSID),

.channel = EXAMPLE\_ESP\_WIFI\_CHANNEL,

.password = EXAMPLE\_ESP\_WIFI\_PASS,

.max\_connection = EXAMPLE\_MAX\_STA\_CONN,

#ifdef CONFIG\_ESP\_WIFI\_SOFTAP\_SAE\_SUPPORT

.authmode = WIFI\_AUTH\_WPA3\_PSK,

.sae\_pwe\_h2e = WPA3\_SAE\_PWE\_BOTH,

#else /\* CONFIG\_ESP\_WIFI\_SOFTAP\_SAE\_SUPPORT \*/

.authmode = WIFI\_AUTH\_WPA2\_PSK,

#endif

.pmf\_cfg = {

.required = true,

},

},

};

if (strlen(EXAMPLE\_ESP\_WIFI\_PASS) == 0) {

wifi\_config.ap.authmode = WIFI\_AUTH\_OPEN;

}

ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_AP));

ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(WIFI\_IF\_AP, &wifi\_config));

ESP\_ERROR\_CHECK(esp\_wifi\_start());

ESP\_LOGI(TAG, "wifi\_init\_softap finished. SSID:%s password:%s channel:%d",

EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS, EXAMPLE\_ESP\_WIFI\_CHANNEL);

}

void app\_main(void)

{

esp\_err\_t ret = nvs\_flash\_init();

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK(ret);

ESP\_LOGI(TAG, "ESP\_WIFI\_MODE\_AP");

wifi\_init\_softap(); // инициализация и обработчик

}

Код ниже позволяет управлять светодиодом со смартфона и наоборот. (**архив 4\_2\_wifi.rar)**

Создание проекта: для его работы необходимо воспользоваться для удобства расширением ESP-IDF Explorer в VS Code. Создаем проект с названием WiFi (например), в поле директории проекта выбираем путь до Espressif/frameworks/esp-idf-v5.1/examples/get-started, выбираем ESP32 module. Потом в поле расширения выбираем ESP-IDF и нажимаем Create project using template sample\_project. Создадим в проекте в main папке файл с названием «Kconfig.projbuild» и туда вставляем код:

menu "Example Configuration"

config ESP\_WIFI\_SSID

string "WiFi SSID"

default "myssid"

help

SSID (network name) for the example to connect to.

config ESP\_WIFI\_PASSWORD

string "WiFi Password"

default "mypassword"

help

WiFi password (WPA or WPA2) for the example to use.

config ESP\_MAXIMUM\_RETRY

int "Maximum retry"

default 5

help

Set the Maximum retry to avoid station reconnecting to the AP unlimited when the AP is really inexistent.

endmenu

В данном файле можно ввести необходимый SSID и пароль, к которой будет подключаться микроконтроллер.

После этого в файле main.c вставляем код и сохраняем проект

#include <stdio.h>

#include <stdlib.h>

#include <string.h> //Requires by memset

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_system.h"

#include "esp\_spi\_flash.h"

#include <esp\_http\_server.h>

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "freertos/event\_groups.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "driver/gpio.h"

#include <lwip/sockets.h>

#include <lwip/sys.h>

#include <lwip/api.h>

#include <lwip/netdb.h>

#include <led\_strip.h>

#include "sdkconfig.h"

#define LED\_PIN 22 // пин светодиода

// страница для отображения при включении

char on\_resp[] = "<!DOCTYPE html><html><head><style type=\"text/css\">html { font-family: Arial; display: inline-block; margin: 0px auto; text-align: center;}h1{ color: #070812; padding: 2vh;}.button { display: inline-block; background-color: #b30000; //red color border: none; border-radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}.button2 { background-color: #364cf4; //blue color}.content { padding: 50px;}.card-grid { max-width: 800px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));}.card { background-color: white; box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);}.card-title { font-size: 1.2rem; font-weight: bold; color: #034078}</style> <title>ESP32 WEB SERVER</title> <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"> <link rel=\"icon\" href=\"data:,\"> <link rel=\"stylesheet\" href=\"https://use.fontawesome.com/releases/v5.7.2/css/all.css\" integrity=\"sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr\" crossorigin=\"anonymous\"> <link rel=\"stylesheet\" type=\"text/css\" ></head><body> <h2>ESP32 WEB SERVER</h2> <div class=\"content\"> <div class=\"card-grid\"> <div class=\"card\"> <p><i class=\"fas fa-lightbulb fa-2x\" style=\"color:#c81919;\"></i> <strong>GPIO22</strong></p> <p>GPIO state: <strong> ON</strong></p> <p> <a href=\"/led2on\"><button class=\"button\">ON</button></a> <a href=\"/led2off\"><button class=\"button button2\">OFF</button></a> </p> </div> </div> </div></body></html>";

//страница для отображения при выключении

char off\_resp[] = "<!DOCTYPE html><html><head><style type=\"text/css\">html { font-family: Arial; display: inline-block; margin: 0px auto; text-align: center;}h1{ color: #070812; padding: 2vh;}.button { display: inline-block; background-color: #b30000; //red color border: none; border-radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}.button2 { background-color: #364cf4; //blue color}.content { padding: 50px;}.card-grid { max-width: 800px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));}.card { background-color: white; box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);}.card-title { font-size: 1.2rem; font-weight: bold; color: #034078}</style> <title>ESP32 WEB SERVER</title> <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\"> <link rel=\"icon\" href=\"data:,\"> <link rel=\"stylesheet\" href=\"https://use.fontawesome.com/releases/v5.7.2/css/all.css\" integrity=\"sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr\" crossorigin=\"anonymous\"> <link rel=\"stylesheet\" type=\"text/css\"></head><body> <h2>ESP32 WEB SERVER</h2> <div class=\"content\"> <div class=\"card-grid\"> <div class=\"card\"> <p><i class=\"fas fa-lightbulb fa-2x\" style=\"color:#c81919;\"></i> <strong>GPIO22</strong></p> <p>GPIO state: <strong> OFF</strong></p> <p> <a href=\"/led2on\"><button class=\"button\">ON</button></a> <a href=\"/led2off\"><button class=\"button button2\">OFF</button></a> </p> </div> </div> </div></body></html>";

static const char \*TAG = "espressif"; // TAG for debug

int led\_state = 0;

#define EXAMPLE\_ESP\_WIFI\_SSID CONFIG\_ESP\_WIFI\_SSID

#define EXAMPLE\_ESP\_WIFI\_PASS CONFIG\_ESP\_WIFI\_PASSWORD

#define EXAMPLE\_ESP\_MAXIMUM\_RETRY CONFIG\_ESP\_MAXIMUM\_RETRY

/\* FreeRTOS event group to signal when we are connected\*/

static EventGroupHandle\_t s\_wifi\_event\_group;

/\* The event group allows multiple bits for each event, but we only care about two events:

\* - we are connected to the AP with an IP

\* - we failed to connect after the maximum amount of retries \*/

#define WIFI\_CONNECTED\_BIT BIT0

#define WIFI\_FAIL\_BIT BIT1

static int s\_retry\_num = 0;

static void event\_handler(void \*arg, esp\_event\_base\_t event\_base,

int32\_t event\_id, void \*event\_data)

{

if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_START)

{

esp\_wifi\_connect();

}

else if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_DISCONNECTED)

{

if (s\_retry\_num < EXAMPLE\_ESP\_MAXIMUM\_RETRY)

{

esp\_wifi\_connect();

s\_retry\_num++;

ESP\_LOGI(TAG, "retry to connect to the AP");

}

else

{

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_FAIL\_BIT);

}

ESP\_LOGI(TAG, "connect to the AP fail");

}

else if (event\_base == IP\_EVENT && event\_id == IP\_EVENT\_STA\_GOT\_IP)

{

ip\_event\_got\_ip\_t \*event = (ip\_event\_got\_ip\_t \*)event\_data;

ESP\_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip\_info.ip));

s\_retry\_num = 0;

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_CONNECTED\_BIT);

}

}

void connect\_wifi(void)

{

s\_wifi\_event\_group = xEventGroupCreate();

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

esp\_netif\_create\_default\_wifi\_sta();

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT();

ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));

esp\_event\_handler\_instance\_t instance\_any\_id;

esp\_event\_handler\_instance\_t instance\_got\_ip;

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(WIFI\_EVENT,

ESP\_EVENT\_ANY\_ID,

&event\_handler,

NULL,

&instance\_any\_id));

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(IP\_EVENT,

IP\_EVENT\_STA\_GOT\_IP,

&event\_handler,

NULL,

&instance\_got\_ip));

wifi\_config\_t wifi\_config = {

.sta = {

.ssid = EXAMPLE\_ESP\_WIFI\_SSID,

.password = EXAMPLE\_ESP\_WIFI\_PASS,

/\* Setting a password implies station will connect to all security modes including WEP/WPA.

\* However these modes are deprecated and not advisable to be used. Incase your Access point

\* doesn't support WPA2, these mode can be enabled by commenting below line \*/

.threshold.authmode = WIFI\_AUTH\_WPA2\_PSK,

},

};

ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_STA));

ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(WIFI\_IF\_STA, &wifi\_config));

ESP\_ERROR\_CHECK(esp\_wifi\_start());

ESP\_LOGI(TAG, "wifi\_init\_sta finished.");

/\* Waiting until either the connection is established (WIFI\_CONNECTED\_BIT) or connection failed for the maximum

\* number of re-tries (WIFI\_FAIL\_BIT). The bits are set by event\_handler() (see above) \*/

EventBits\_t bits = xEventGroupWaitBits(s\_wifi\_event\_group,

WIFI\_CONNECTED\_BIT | WIFI\_FAIL\_BIT,

pdFALSE,

pdFALSE,

portMAX\_DELAY);

/\* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually

\* happened. \*/

if (bits & WIFI\_CONNECTED\_BIT)

{

ESP\_LOGI(TAG, "connected to ap SSID:%s password:%s",

EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

}

else if (bits & WIFI\_FAIL\_BIT)

{

ESP\_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",

EXAMPLE\_ESP\_WIFI\_SSID, EXAMPLE\_ESP\_WIFI\_PASS);

}

else

{

ESP\_LOGE(TAG, "UNEXPECTED EVENT");

}

vEventGroupDelete(s\_wifi\_event\_group);

}

esp\_err\_t send\_web\_page(httpd\_req\_t \*req)

{

int response;

if (led\_state == 0){

response = httpd\_resp\_send(req, off\_resp, HTTPD\_RESP\_USE\_STRLEN); // если был вызван off - то переход на страницу off\_resp

}

else { // иначе на on\_resp

response = httpd\_resp\_send(req, on\_resp, HTTPD\_RESP\_USE\_STRLEN);

}

return response;

}

esp\_err\_t get\_req\_handler(httpd\_req\_t \*req) // переход на нужную страницу в зависимости от места вызова

{

return send\_web\_page(req);

}

esp\_err\_t led\_on\_handler(httpd\_req\_t \*req) // при вызове - включение светодиода

{

gpio\_set\_level(LED\_PIN, 1);

led\_state = 1;

return send\_web\_page(req);

}

esp\_err\_t led\_off\_handler(httpd\_req\_t \*req) // при вызове - выключение светодиода

{

gpio\_set\_level(LED\_PIN, 0);

led\_state = 0;

return send\_web\_page(req);

}

httpd\_uri\_t uri\_get = { // обработчики и страницы для перехода

.uri = "/",

.method = HTTP\_GET,

.handler = get\_req\_handler,

.user\_ctx = NULL};

httpd\_uri\_t uri\_on = {

.uri = "/led2on",

.method = HTTP\_GET,

.handler = led\_on\_handler,

.user\_ctx = NULL};

httpd\_uri\_t uri\_off = {

.uri = "/led2off",

.method = HTTP\_GET,

.handler = led\_off\_handler,

.user\_ctx = NULL};

httpd\_handle\_t setup\_server(void) // подключение обработчиков

{

httpd\_config\_t config = HTTPD\_DEFAULT\_CONFIG();

httpd\_handle\_t server = NULL;

if (httpd\_start(&server, &config) == ESP\_OK)

{

httpd\_register\_uri\_handler(server, &uri\_get);

httpd\_register\_uri\_handler(server, &uri\_on);

httpd\_register\_uri\_handler(server, &uri\_off);

}

return server;

}

void app\_main()

{

// Initialize NVS

esp\_err\_t ret = nvs\_flash\_init();

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND)

{

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK(ret);

ESP\_LOGI(TAG, "ESP\_WIFI\_MODE\_STA");

connect\_wifi();

// GPIO initialization

esp\_rom\_gpio\_pad\_select\_gpio(LED\_PIN);

gpio\_set\_direction(LED\_PIN, GPIO\_MODE\_OUTPUT);

led\_state = 0;

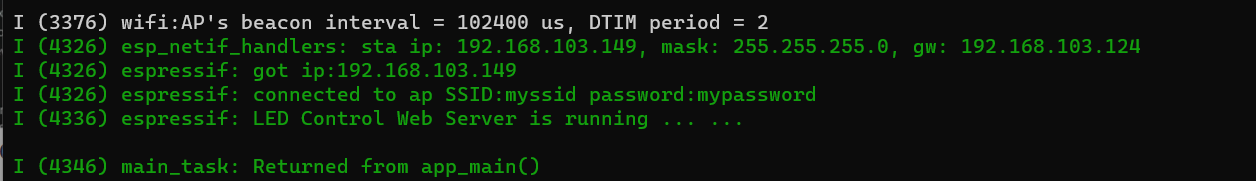
ESP\_LOGI(TAG, "LED Control Web Server is running ... ...\n");

setup\_server();

}

*Принцип работы:*

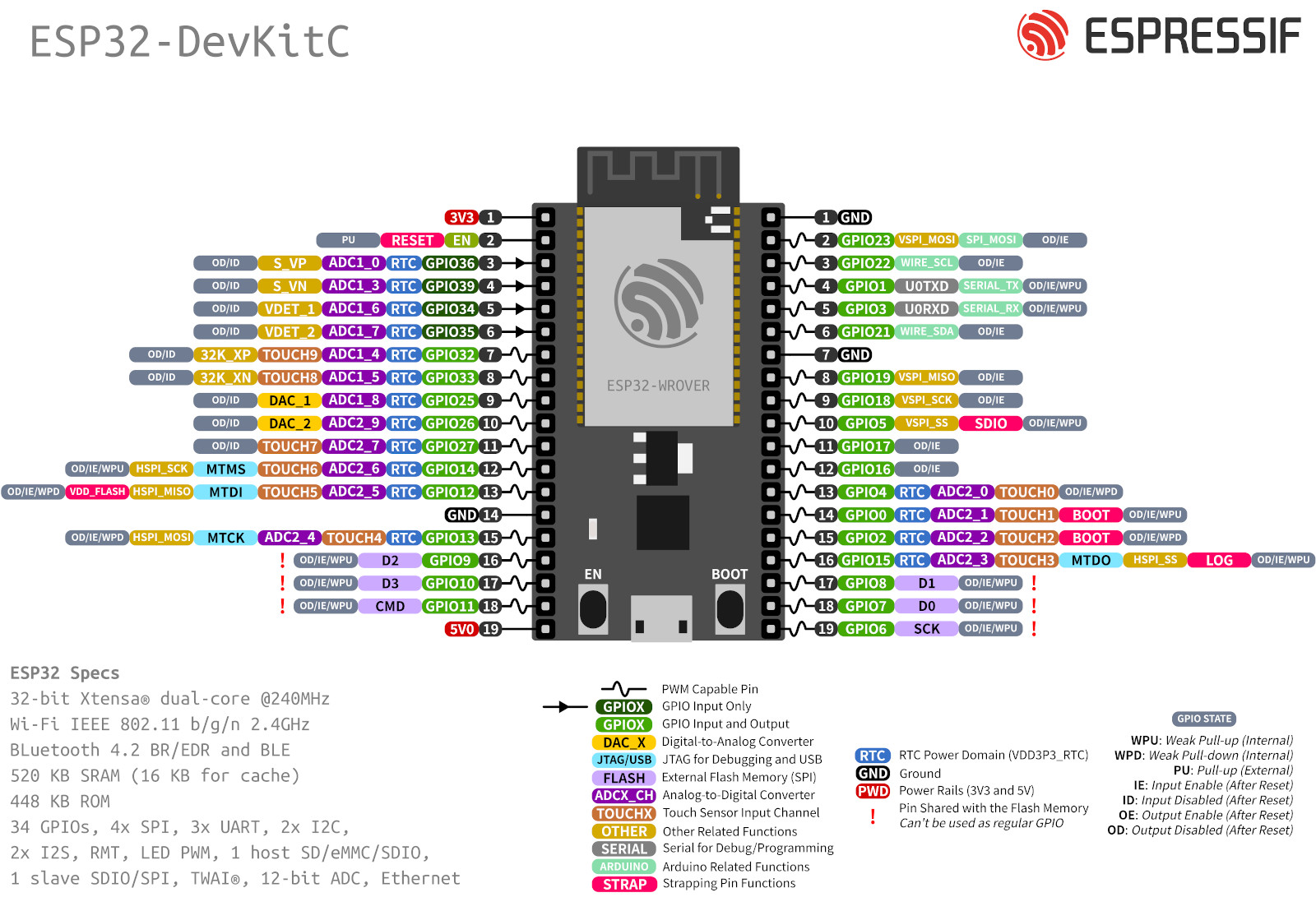
После того, как мы забилдили проект (idf.py build), зашили код (idf.py -p COMx flash) и запустили его (командой idf.py -p COMx monitor) микроконтроллер пытается подключаться к сети myssid с паролем mypassword (данные настройки можно сменить в файле как описано выше), в логах monitor’а будет выдан IP-адрес (на рисунке ниже это строка got ip: 192.168.103.149).



Теперь необходимо с устройства, которое является точкой доступа, перейти по адресу 192.168.103.149 (в данном примере), откроется сайт и при нажатии на кнопку ON будет загораться светодиод на 22 пине.

**Задача 5. Запустить АЦП на плате. (архив 5\_adc.rar)**

Микроконтроллер ESP32 DevKit V4 Wroom 32U обладает двумя 12-разрядными АЦП, всего 18 аналоговых каналов.



В качестве примера мы используем канал 1 и пин 4 (GPIO 32).

Код ниже выводит в idf.py -p COMx monitor сырое значение АЦП и обработанное в мВ.

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/adc.h"

#include "esp\_adc\_cal.h"

static esp\_adc\_cal\_characteristics\_t adc1\_chars;

void app\_main(void)

{

esp\_adc\_cal\_characterize(ADC\_UNIT\_1, ADC\_ATTEN\_DB\_11, ADC\_WIDTH\_BIT\_DEFAULT, 0, &adc1\_chars);

adc1\_config\_width(ADC\_WIDTH\_BIT\_DEFAULT);

adc1\_config\_channel\_atten(ADC1\_CHANNEL\_4, ADC\_ATTEN\_DB\_11);

while (1)

{

int adc\_value = adc1\_get\_raw(ADC1\_CHANNEL\_4);

printf("ADC row value: %d\t", adc\_value);

uint32\_t mV = esp\_adc\_cal\_raw\_to\_voltage(adc\_value, &adc1\_chars);

printf("mV: %ld\n", mV);

vTaskDelay(500/ portTICK\_PERIOD\_MS);

}

}

Функции:

adc1\_config\_width – для настройки разрядности АЦП. Мы выбираем DEFAULT (12 разрядов).

adc1\_config\_channel\_atten – задается диапазон входного напряжения, в качестве параметров выбираем переменную с каналом-пином и параметр затухания из таблицы ниже:

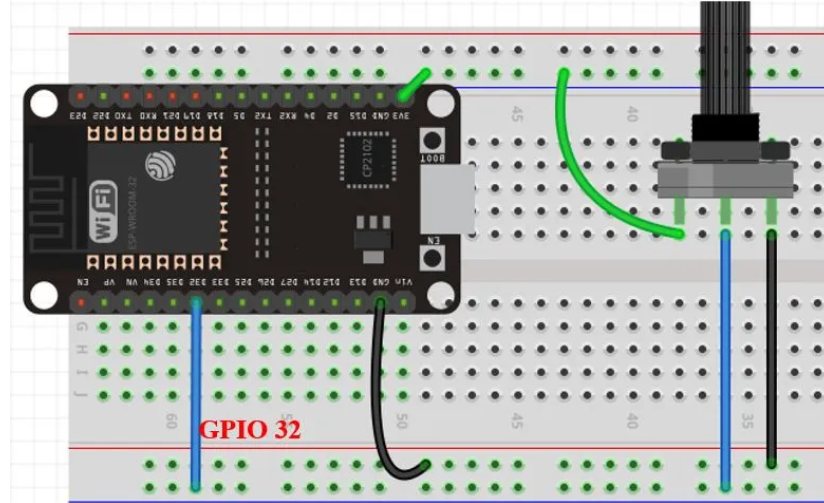
| **Затухание** | **Параметр затухания** | **Диапазон входного напряжения** |
| --- | --- | --- |
| 0dB | ADC\_ATTEN\_DB\_0 | ~100-950 mV |
| 2.5dB | ADC\_ATTEN\_DB\_2\_5 | ~100-1250mV |
| 6dB | ADC\_ATTEN\_DB\_6 | ~150-1750mV |
| 11dB | ADC\_ATTEN\_DB\_11 | ~150-2450mV |

esp\_adc\_cal\_characterize – калибровка АЦП. Последний параметр – откалиброванное значение АЦП.

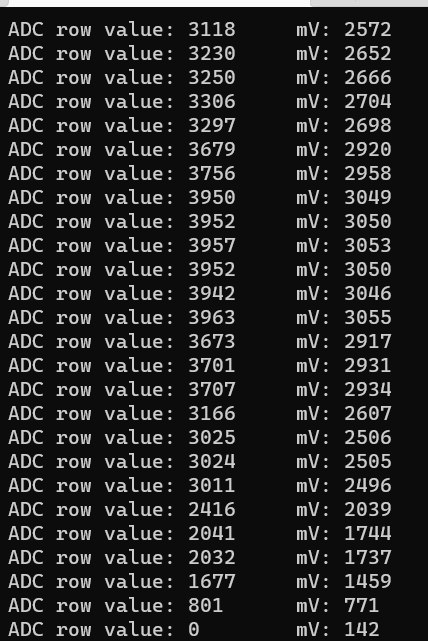
adc1\_get\_raw – получаем сырое значение из канала.

esp\_adc\_cal\_raw\_to\_voltage – переводим сырое значение в мВ.

Схема, необходимая для работы:



В мониторе получим такие строки, значение которых будет меняться в зависимости от потенциометра:



**Задача 6. Научиться выводить показания АЦП на смартфон. (Архив 6\_adc.rar)**

Для вывода показаний АЦП мы используем код для веб-сервера из задачи 4, немного его модифицировав. А также возьмем код и схему из задачи 5.

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/adc.h"

#include "esp\_adc\_cal.h"

#include <string.h> //Requires by memset

#include "esp\_system.h"

#include "esp\_spi\_flash.h"

#include <esp\_http\_server.h>

#include "esp\_wifi.h"

#include "esp\_event.h"

#include "freertos/event\_groups.h"

#include "esp\_log.h"

#include "nvs\_flash.h"

#include "esp\_netif.h"

#include "driver/gpio.h"

#include <lwip/sockets.h>

#include <lwip/sys.h>

#include <lwip/api.h>

#include <lwip/netdb.h>

#define ESP\_WIFI\_SSID "myssid"

#define ESP\_WIFI\_PASSWORD "mypassword"

#define ESP\_MAXIMUM\_RETRY 5

const int DS\_PIN = 4;

static const char \*TAG = "esp32 webserver";

static esp\_adc\_cal\_characteristics\_t adc1\_chars;

static EventGroupHandle\_t s\_wifi\_event\_group;

#define WIFI\_CONNECTED\_BIT BIT0

#define WIFI\_FAIL\_BIT BIT1

static int s\_retry\_num = 0;

int wifi\_connect\_status = 0;

int adc\_value;

int mV;

void checkParam(){

adc\_value = adc1\_get\_raw(ADC1\_CHANNEL\_4);

printf("ADC row value: %d\t", adc\_value);

mV = esp\_adc\_cal\_raw\_to\_voltage(adc\_value, &adc1\_chars);

printf("mV: %d\n", mV);

}

char html\_page[] = "<!DOCTYPE HTML><html>\n"

"<head>\n"

" <title>ESP-IDF ADC Web Server</title>\n"

" <meta http-equiv=\"refresh\" content=\"0.5\">\n"

" <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">\n"

" <link rel=\"stylesheet\" href=\"https://use.fontawesome.com/releases/v5.7.2/css/all.css\" integrity=\"sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr\" crossorigin=\"anonymous\">\n"

" <link rel=\"icon\" href=\"data:,\">\n"

" <style>\n"

" html {font-family: Arial; display: inline-block; text-align: center;}\n"

" p { font-size: 1.2rem;}\n"

" body { margin: 0;}\n"

" .topnav { overflow: hidden; background-color: #7c0e61; color: white; font-size: 1.7rem; }\n"

" .content { padding: 20px; }\n"

" .card { background-color: white; box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5); }\n"

" .cards { max-width: 700px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); }\n"

" .reading { font-size: 2.8rem; }\n"

" .card.temperatureC { color: #0e7c7b; }\n"

" .card.temperatureF { color: #7c200e; }\n"

" </style>\n"

"</head>\n"

"<body>\n"

" <div class=\"topnav\">\n"

" <h3>ESP-IDF ADC WEB SERVER</h3>\n"

" </div>\n"

" <div class=\"content\">\n"

" <div class=\"cards\">\n"

" <div class=\"card temperatureC\">\n"

" <h4><i class=\"fas fa-upload\"></i> row data</h4><p><span class=\"reading\">%d</span></p>\n"

" </div>\n"

" <div class=\"card temperatureF\">\n"

" <h4><i class=\"fas fa-bolt\"></i> mV</h4><p><span class=\"reading\">%d</span></p>\n"

" </div>\n"

" </div>\n"

" </div>\n"

"</body>\n"

"</html>";

static void event\_handler(void \*arg, esp\_event\_base\_t event\_base,

int32\_t event\_id, void \*event\_data)

{

if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_START)

{

esp\_wifi\_connect();

}

else if (event\_base == WIFI\_EVENT && event\_id == WIFI\_EVENT\_STA\_DISCONNECTED)

{

if (s\_retry\_num < ESP\_MAXIMUM\_RETRY)

{

esp\_wifi\_connect();

s\_retry\_num++;

ESP\_LOGI(TAG, "retry to connect to the AP");

}

else

{

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_FAIL\_BIT);

}

wifi\_connect\_status = 0;

ESP\_LOGI(TAG, "connect to the AP fail");

}

else if (event\_base == IP\_EVENT && event\_id == IP\_EVENT\_STA\_GOT\_IP)

{

ip\_event\_got\_ip\_t \*event = (ip\_event\_got\_ip\_t \*)event\_data;

ESP\_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip\_info.ip));

s\_retry\_num = 0;

xEventGroupSetBits(s\_wifi\_event\_group, WIFI\_CONNECTED\_BIT);

wifi\_connect\_status = 1;

}

}

void connect\_wifi(void)

{

s\_wifi\_event\_group = xEventGroupCreate();

ESP\_ERROR\_CHECK(esp\_netif\_init());

ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());

esp\_netif\_create\_default\_wifi\_sta();

wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT();

ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));

esp\_event\_handler\_instance\_t instance\_any\_id;

esp\_event\_handler\_instance\_t instance\_got\_ip;

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(WIFI\_EVENT,

ESP\_EVENT\_ANY\_ID,

&event\_handler,

NULL,

&instance\_any\_id));

ESP\_ERROR\_CHECK(esp\_event\_handler\_instance\_register(IP\_EVENT,

IP\_EVENT\_STA\_GOT\_IP,

&event\_handler,

NULL,

&instance\_got\_ip));

wifi\_config\_t wifi\_config = {

.sta = {

.ssid = ESP\_WIFI\_SSID,

.password = ESP\_WIFI\_PASSWORD,

/\* Setting a password implies station will connect to all security modes including WEP/WPA.

\* However these modes are deprecated and not advisable to be used. Incase your Access point

\* doesn't support WPA2, these mode can be enabled by commenting below line \*/

.threshold.authmode = WIFI\_AUTH\_WPA2\_PSK,

},

};

ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_STA));

ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(WIFI\_IF\_STA, &wifi\_config));

ESP\_ERROR\_CHECK(esp\_wifi\_start());

ESP\_LOGI(TAG, "wifi\_init\_sta finished.");

/\* Waiting until either the connection is established (WIFI\_CONNECTED\_BIT) or connection failed for the maximum

\* number of re-tries (WIFI\_FAIL\_BIT). The bits are set by event\_handler() (see above) \*/

EventBits\_t bits = xEventGroupWaitBits(s\_wifi\_event\_group,

WIFI\_CONNECTED\_BIT | WIFI\_FAIL\_BIT,

pdFALSE,

pdFALSE,

portMAX\_DELAY);

/\* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually

\* happened. \*/

if (bits & WIFI\_CONNECTED\_BIT)

{

ESP\_LOGI(TAG, "connected to ap SSID:%s password:%s",

ESP\_WIFI\_SSID, ESP\_WIFI\_PASSWORD);

}

else if (bits & WIFI\_FAIL\_BIT)

{

ESP\_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",

ESP\_WIFI\_SSID, ESP\_WIFI\_PASSWORD);

}

else

{

ESP\_LOGE(TAG, "UNEXPECTED EVENT");

}

vEventGroupDelete(s\_wifi\_event\_group);

}

esp\_err\_t send\_web\_page(httpd\_req\_t \*req)

{

int response;

checkParam();

char response\_data[sizeof(html\_page) + 50];

memset(response\_data, 0, sizeof(response\_data));

sprintf(response\_data, html\_page, adc\_value, mV);

response = httpd\_resp\_send(req, response\_data, HTTPD\_RESP\_USE\_STRLEN);

vTaskDelay(500/ portTICK\_PERIOD\_MS);

return response;

}

esp\_err\_t get\_req\_handler(httpd\_req\_t \*req)

{

return send\_web\_page(req);

}

httpd\_uri\_t uri\_get = {

.uri = "/",

.method = HTTP\_GET,

.handler = get\_req\_handler,

.user\_ctx = NULL};

httpd\_handle\_t setup\_server(void)

{

httpd\_config\_t config = HTTPD\_DEFAULT\_CONFIG();

httpd\_handle\_t server = NULL;

if (httpd\_start(&server, &config) == ESP\_OK)

{

httpd\_register\_uri\_handler(server, &uri\_get);

}

return server;

}

void app\_main()

{

// Initialize NVS

esp\_err\_t ret = nvs\_flash\_init();

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND)

{

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK(ret);

esp\_adc\_cal\_characterize(ADC\_UNIT\_1, ADC\_ATTEN\_DB\_11, ADC\_WIDTH\_BIT\_DEFAULT, 0, &adc1\_chars);

adc1\_config\_width(ADC\_WIDTH\_BIT\_DEFAULT);

adc1\_config\_channel\_atten(ADC1\_CHANNEL\_4, ADC\_ATTEN\_DB\_11);

ESP\_LOGI(TAG, "ESP\_WIFI\_MODE\_STA");

connect\_wifi();

if (wifi\_connect\_status)

{

setup\_server();

ESP\_LOGI(TAG, "Web Server is up and running\n");

}

else

ESP\_LOGI(TAG, "Failed to connected with Wi-Fi, check your network Credentials\n");

}

Как итог мы собираем проект (команда idf.py build), зашиваем (idf.py -p COMx flash) и запускаем монитор (idf.py -p COMx monitor). В логах монитора будет выдан устройству ip-адрес, на который мы заходим и видим данные с АЦП в реальном времени.

**Задача 7. Научиться использовать Bluetooth на ESP32. (архив 7\_bluetooth.rar)**

Код ниже реализует управление портами GPIO через Bluetooth. Так, создается точка Bluetooth с именем ESP\_SPP\_ACCEPTOR, с помощью телефона подключаемся к ней и заходим в приложение Bluetooth Terminal, там подключаемся к нашему устройству и посылаем команды «on red led», «on green led», «off red led», «off green led». Для работы собираем схему из двух светодиодов и резисторов для них на 21 и 22 портах.

include <stdint.h>

#include <string.h>

#include <stdbool.h>

#include <stdio.h>

#include <inttypes.h>

#include "nvs.h"

#include "nvs\_flash.h"

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "esp\_bt.h"

#include "esp\_bt\_main.h"

#include "esp\_gap\_bt\_api.h"

#include "esp\_bt\_device.h"

#include "esp\_spp\_api.h"

#include "spp\_task.h"

#include "time.h"

#include "sys/time.h"

#include "esp\_vfs.h"

#include "sys/unistd.h"

#include "driver/gpio.h"

#include "led\_strip.h"

#define RED\_PIN 22

#define GREEN\_PIN 21

#define SPP\_TAG "SPP\_ACCEPTOR\_DEMO"

#define SPP\_SERVER\_NAME "SPP\_SERVER"

#define EXAMPLE\_DEVICE\_NAME "ESP\_SPP\_ACCEPTOR"

static const esp\_spp\_sec\_t sec\_mask = ESP\_SPP\_SEC\_AUTHENTICATE;

static const esp\_spp\_role\_t role\_slave = ESP\_SPP\_ROLE\_SLAVE;

#define SPP\_DATA\_LEN 100

static char \*bda2str(uint8\_t \* bda, char \*str, size\_t size)

{

if (bda == NULL || str == NULL || size < 18) {

return NULL;

}

uint8\_t \*p = bda;

sprintf(str, "%02x:%02x:%02x:%02x:%02x:%02x",

p[0], p[1], p[2], p[3], p[4], p[5]);

return str;

}

// функция spp\_read\_handle выполняет чтение буфера из bluetooth

// если в буфере находится строка on red - то включается светодиод и так далее

// в 91-94 сравниваются строки с буфером и на основе условий выполняются действия

static void spp\_read\_handle(void \* param)

{

int size = 0;

int fd = (int)param;

do {

char\* spp\_data = malloc(sizeof(char) \* SPP\_DATA\_LEN);

/\* The frequency of calling this function also limits the speed at which the peer device can send data. \*/

size = read(fd, spp\_data, SPP\_DATA\_LEN);

if (size < 0) {

break;

}

else if (size == 0) {

/\* There is no data, retry after 500 ms \*/

vTaskDelay(500 / portTICK\_PERIOD\_MS);

}

else {

ESP\_LOGI(SPP\_TAG, "fd = %d data\_len = %d", fd, size);

//esp\_log\_buffer\_hex(SPP\_TAG, spp\_data, size);

char on\_red = NULL;

char on\_green = NULL;

char off\_red = NULL;

char off\_green = NULL;

off\_red = strstr(spp\_data, "off red");

on\_red = strstr(spp\_data, "on red");

on\_green = strstr(spp\_data, "on green");

off\_green = strstr(spp\_data, "off green");

if (off\_red != NULL) {

ESP\_LOGI(SPP\_TAG, "Off red led");

gpio\_set\_level(RED\_PIN, 0);

}

if (off\_green != NULL) {

ESP\_LOGI(SPP\_TAG, "Off green led");

gpio\_set\_level(GREEN\_PIN, 0);

}

if (on\_red != NULL) {

ESP\_LOGI(SPP\_TAG, "On red led");

gpio\_set\_level(RED\_PIN, 1);

}

if (on\_green != NULL) {

ESP\_LOGI(SPP\_TAG, "On green led");

gpio\_set\_level(GREEN\_PIN, 1);

}

//To avoid task watchdog

vTaskDelay(10 / portTICK\_PERIOD\_MS);

}

free(spp\_data);

}

while (1);

spp\_wr\_task\_shut\_down();

}

static void esp\_spp\_cb(uint16\_t e, void \*p)

{

esp\_spp\_cb\_event\_t event = e;

esp\_spp\_cb\_param\_t \*param = p;

char bda\_str[18] = {0};

switch (event) {

case ESP\_SPP\_INIT\_EVT:

if (param->init.status == ESP\_SPP\_SUCCESS) {

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_INIT\_EVT");

/\* Enable SPP VFS mode \*/

esp\_spp\_vfs\_register();

} else {

ESP\_LOGE(SPP\_TAG, "ESP\_SPP\_INIT\_EVT status:%d", param->init.status);

}

break;

case ESP\_SPP\_DISCOVERY\_COMP\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_DISCOVERY\_COMP\_EVT");

break;

case ESP\_SPP\_OPEN\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_OPEN\_EVT");

break;

case ESP\_SPP\_CLOSE\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_CLOSE\_EVT status:%d handle:%"PRIu32" close\_by\_remote:%d", param->close.status,

param->close.handle, param->close.async);

break;

case ESP\_SPP\_START\_EVT:

if (param->start.status == ESP\_SPP\_SUCCESS) {

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_START\_EVT handle:%"PRIu32" sec\_id:%d scn:%d", param->start.handle, param->start.sec\_id,

param->start.scn);

esp\_bt\_dev\_set\_device\_name(EXAMPLE\_DEVICE\_NAME);

esp\_bt\_gap\_set\_scan\_mode(ESP\_BT\_CONNECTABLE, ESP\_BT\_GENERAL\_DISCOVERABLE);

} else {

ESP\_LOGE(SPP\_TAG, "ESP\_SPP\_START\_EVT status:%d", param->start.status);

}

break;

case ESP\_SPP\_CL\_INIT\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_CL\_INIT\_EVT");

break;

case ESP\_SPP\_SRV\_OPEN\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_SRV\_OPEN\_EVT status:%d handle:%"PRIu32", rem\_bda:[%s]", param->srv\_open.status,

param->srv\_open.handle, bda2str(param->srv\_open.rem\_bda, bda\_str, sizeof(bda\_str)));

if (param->srv\_open.status == ESP\_SPP\_SUCCESS) {

spp\_wr\_task\_start\_up(spp\_read\_handle, param->srv\_open.fd);

}

break;

case ESP\_SPP\_VFS\_REGISTER\_EVT:

if (param->vfs\_register.status == ESP\_SPP\_SUCCESS) {

ESP\_LOGI(SPP\_TAG, "ESP\_SPP\_VFS\_REGISTER\_EVT");

esp\_spp\_start\_srv(sec\_mask, role\_slave, 0, SPP\_SERVER\_NAME);

} else {

ESP\_LOGE(SPP\_TAG, "ESP\_SPP\_VFS\_REGISTER\_EVT status:%d", param->vfs\_register.status);

}

break;

default:

break;

}

}

static void esp\_spp\_stack\_cb(esp\_spp\_cb\_event\_t event, esp\_spp\_cb\_param\_t \*param)

{

/\* To avoid stucking Bluetooth stack, we dispatch the SPP callback event to the other lower priority task \*/

spp\_task\_work\_dispatch(esp\_spp\_cb, event, param, sizeof(esp\_spp\_cb\_param\_t), NULL);

}

void esp\_bt\_gap\_cb(esp\_bt\_gap\_cb\_event\_t event, esp\_bt\_gap\_cb\_param\_t \*param)

{

switch (event) {

case ESP\_BT\_GAP\_AUTH\_CMPL\_EVT:{

if (param->auth\_cmpl.stat == ESP\_BT\_STATUS\_SUCCESS) {

ESP\_LOGI(SPP\_TAG, "authentication success: %s", param->auth\_cmpl.device\_name);

esp\_log\_buffer\_hex(SPP\_TAG, param->auth\_cmpl.bda, ESP\_BD\_ADDR\_LEN);

} else {

ESP\_LOGE(SPP\_TAG, "authentication failed, status:%d", param->auth\_cmpl.stat);

}

break;

}

case ESP\_BT\_GAP\_PIN\_REQ\_EVT:{

ESP\_LOGI(SPP\_TAG, "ESP\_BT\_GAP\_PIN\_REQ\_EVT min\_16\_digit:%d", param->pin\_req.min\_16\_digit);

if (param->pin\_req.min\_16\_digit) {

ESP\_LOGI(SPP\_TAG, "Input pin code: 0000 0000 0000 0000");

esp\_bt\_pin\_code\_t pin\_code = {0};

esp\_bt\_gap\_pin\_reply(param->pin\_req.bda, true, 16, pin\_code);

} else {

ESP\_LOGI(SPP\_TAG, "Input pin code: 1234");

esp\_bt\_pin\_code\_t pin\_code;

pin\_code[0] = '1';

pin\_code[1] = '2';

pin\_code[2] = '3';

pin\_code[3] = '4';

esp\_bt\_gap\_pin\_reply(param->pin\_req.bda, true, 4, pin\_code);

}

break;

}

#if (CONFIG\_BT\_SSP\_ENABLED == true)

case ESP\_BT\_GAP\_CFM\_REQ\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_BT\_GAP\_CFM\_REQ\_EVT Please compare the numeric value: %"PRIu32, param->cfm\_req.num\_val);

esp\_bt\_gap\_ssp\_confirm\_reply(param->cfm\_req.bda, true);

break;

case ESP\_BT\_GAP\_KEY\_NOTIF\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_BT\_GAP\_KEY\_NOTIF\_EVT passkey:%"PRIu32, param->key\_notif.passkey);

break;

case ESP\_BT\_GAP\_KEY\_REQ\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_BT\_GAP\_KEY\_REQ\_EVT Please enter passkey!");

break;

#endif

case ESP\_BT\_GAP\_MODE\_CHG\_EVT:

ESP\_LOGI(SPP\_TAG, "ESP\_BT\_GAP\_MODE\_CHG\_EVT mode:%d", param->mode\_chg.mode);

break;

default: {

ESP\_LOGI(SPP\_TAG, "event: %d", event);

break;

}

}

return;

}

void app\_main(void)

{

gpio\_reset\_pin(RED\_PIN);

gpio\_reset\_pin(GREEN\_PIN);

gpio\_set\_direction(RED\_PIN, GPIO\_MODE\_OUTPUT);

gpio\_set\_direction(GREEN\_PIN, GPIO\_MODE\_OUTPUT);

char bda\_str[18] = {0};

esp\_err\_t ret = nvs\_flash\_init();

if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

ESP\_ERROR\_CHECK(nvs\_flash\_erase());

ret = nvs\_flash\_init();

}

ESP\_ERROR\_CHECK( ret );

ESP\_ERROR\_CHECK(esp\_bt\_controller\_mem\_release(ESP\_BT\_MODE\_BLE));

esp\_bt\_controller\_config\_t bt\_cfg = BT\_CONTROLLER\_INIT\_CONFIG\_DEFAULT();

if (esp\_bt\_controller\_init(&bt\_cfg) != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s initialize controller failed", \_\_func\_\_);

return;

}

if (esp\_bt\_controller\_enable(ESP\_BT\_MODE\_CLASSIC\_BT) != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s enable controller failed", \_\_func\_\_);

return;

}

if (esp\_bluedroid\_init() != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s initialize bluedroid failed", \_\_func\_\_);

return;

}

if (esp\_bluedroid\_enable() != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s enable bluedroid failed", \_\_func\_\_);

return;

}

if (esp\_bt\_gap\_register\_callback(esp\_bt\_gap\_cb) != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s gap register failed: %s\n", \_\_func\_\_, esp\_err\_to\_name(ret));

return;

}

if (esp\_spp\_register\_callback(esp\_spp\_stack\_cb) != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s spp register failed", \_\_func\_\_);

return;

}

spp\_task\_task\_start\_up();

esp\_spp\_cfg\_t bt\_spp\_cfg = BT\_SPP\_DEFAULT\_CONFIG();

if (esp\_spp\_enhanced\_init(&bt\_spp\_cfg) != ESP\_OK) {

ESP\_LOGE(SPP\_TAG, "%s spp init failed", \_\_func\_\_);

return;

}

#if (CONFIG\_BT\_SSP\_ENABLED == true)

/\* Set default parameters for Secure Simple Pairing \*/

esp\_bt\_sp\_param\_t param\_type = ESP\_BT\_SP\_IOCAP\_MODE;

esp\_bt\_io\_cap\_t iocap = ESP\_BT\_IO\_CAP\_IO;

esp\_bt\_gap\_set\_security\_param(param\_type, &iocap, sizeof(uint8\_t));

#endif

/\*

\* Set default parameters for Legacy Pairing

\* Use variable pin, input pin code when pairing

\*/

esp\_bt\_pin\_type\_t pin\_type = ESP\_BT\_PIN\_TYPE\_VARIABLE;

esp\_bt\_pin\_code\_t pin\_code;

esp\_bt\_gap\_set\_pin(pin\_type, 0, pin\_code);

ESP\_LOGI(SPP\_TAG, "Own address:[%s]", bda2str((uint8\_t \*)esp\_bt\_dev\_get\_address(), bda\_str, sizeof(bda\_str)));

}