sparkfun ELECTRONICS

SparkFun ESP32 Thing

Development Workshop

Agus Kurniawan

Copyright

SparkFun ESP32 Thing Development Workshop

Agus Kurniawan

1st Edition, 2017

Table of Contents

Preface

This book was written to help anyone want to get started with SparkFun ESP32 Thing board development. It describes the basic elements of SparkFun ESP32 Thing development using Espressif IoT Development Framework.

Agus Kurniawan

Berlin, April 2017

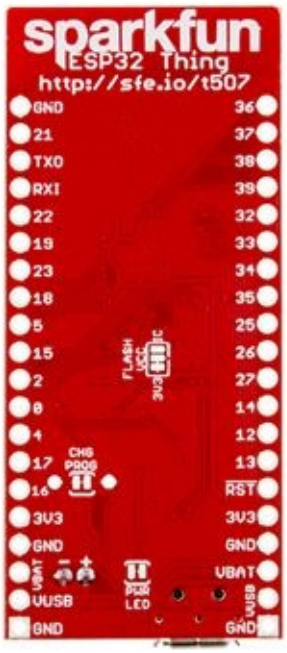
1. Preparing Development Environment

1.1 SparkFun ESP32 Thing

The SparkFun ESP32 Thing is a development board based on Espressif’s ESP32. This board has built-in WiFi and Bluetooth modules.

The following is the board features:

- Dual-core Tensilica LX6 microprocessor
- Up to 240MHz clock frequency
- 520kB internal SRAM
- Integrated 802.11 BGN WiFi transceiver
- Integrated dual-mode Bluetooth (classic and BLE)
- 2.2 to 3.6V operating range
- 2.5 µA deep sleep current
- 28 GPIO
- 10-electrode capacitive touch support
- Hardware accelerated encryption (AES, SHA2, ECC, RSA-4096)
- 4MB Flash memory
- Integrated LiPo Battery Charger



Officially you can buy this board on https://www.sparkfun.com/products/13907. You also can buy this product on your local electronic store.

1.2 Electronics Components

We need electronic components to build our testing, for instance, Resistor, LED, sensor devices and etc. I recommend you can buy electronic component kit. We can use electronics kit from Arduino to be developed on SparkFun ESP32 Thing. The following is a list of electronics kit which can be used in our case.

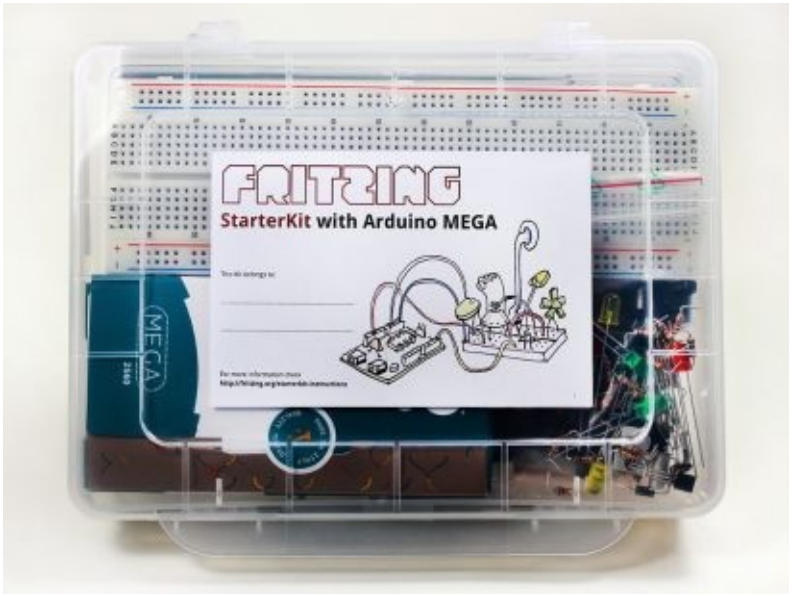
1.2.1 Arduino Starter Kit

Store website: http://arduino.cc/en/Main/ArduinoStarterKit

1.2.2 Fritzing

Store website: http://shop.fritzing.org/ .

You can buy Fritzing Starter Kit with Arduino UNO or Fritzing Starter Kit with Arduino Mega.



1.2.3 Cooking-Hacks: Arduino Starter Kit

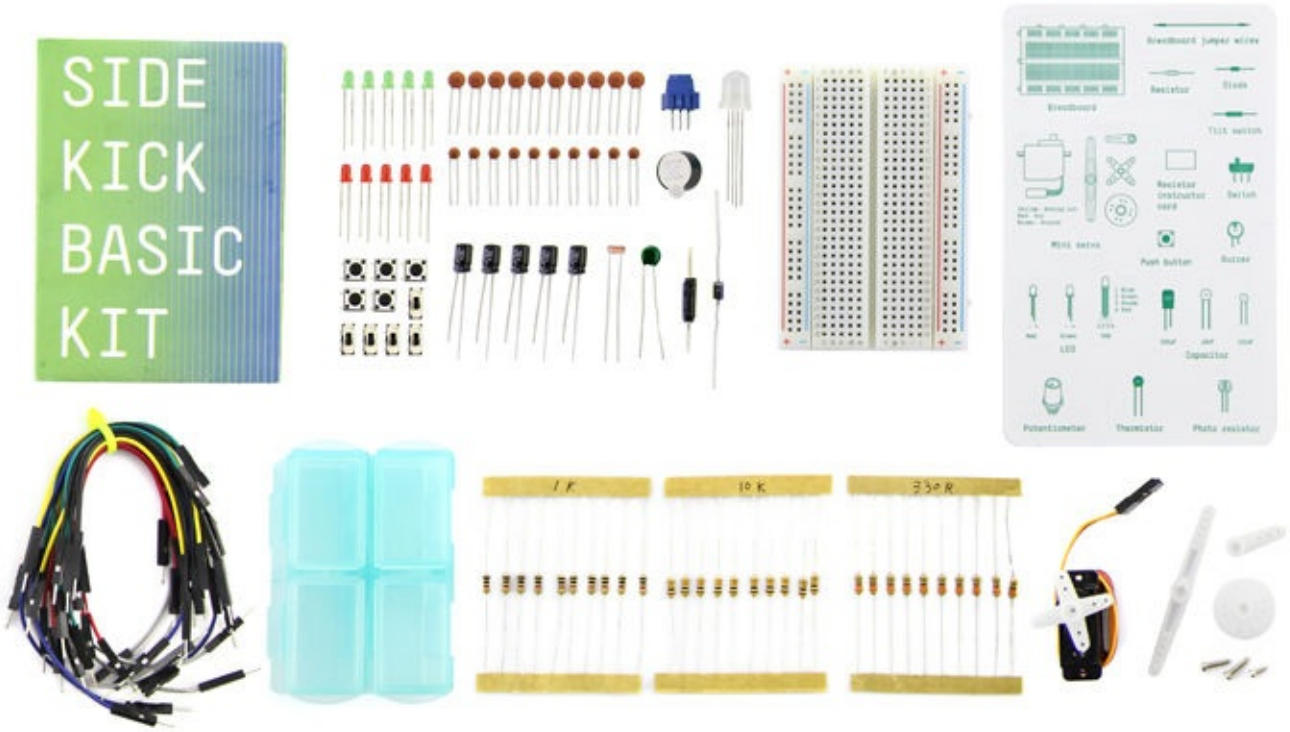
Store website: http://www.cooking-hacks.com/index.php/shop/arduino/starter-kits/arduino-starter-kit.html

1.2.4 Arduino Sidekick Basic kit v2

Store website: http://www.seeedstudio.com/depot/Sidekick-Basic-Kit-for-Arduino-V2-p-1858.html

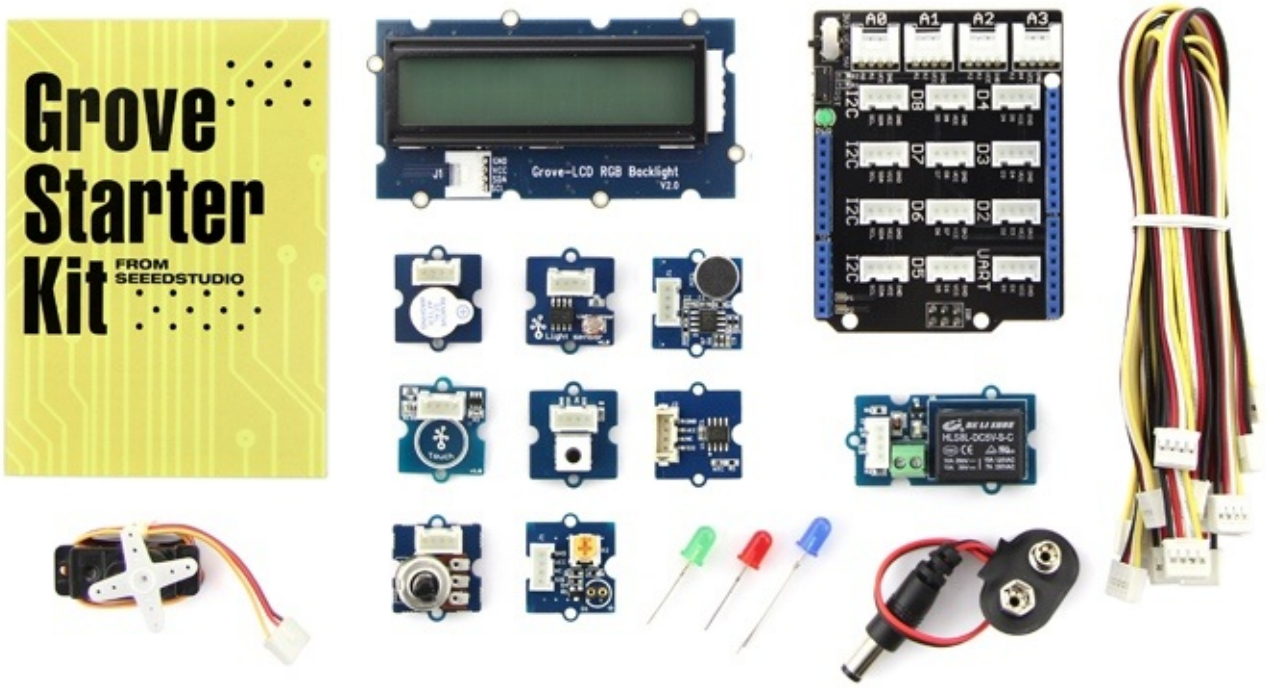
You also can find this kit on this online store.

http://www.exp-tech.de/seeed-studio-sidekick-basic-kit-for-arduino-v2



1.2.5 Grove - Starter Kit for Arduino

Another option, you can buy this kit on Seeedstudio, http://www.seeedstudio.com/depot/Grove-Starter-Kit-for-Arduino-p-1855.html .



1.2.6 DFRobot - Arduino Kit for Beginner v3

DFRobot provides Arduino kit too. You can buy it on the following website.

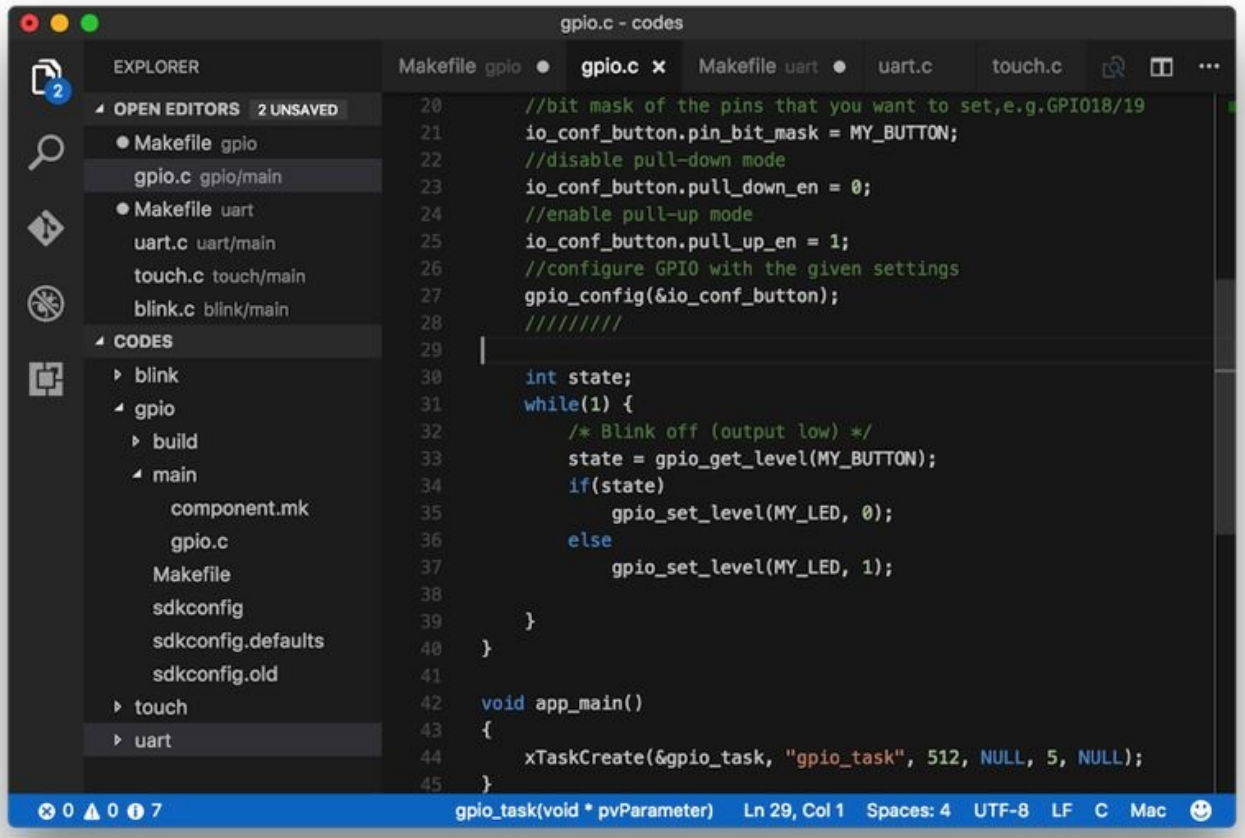
http://www.dfrobot.com/index.php?route=product/product&path=35_49&product_id=345

1.3 Development Tools

To develop app with SparkFun ESP32 Thing target, I use Espressif IoT Development Framework. You can follow the installation process in these links

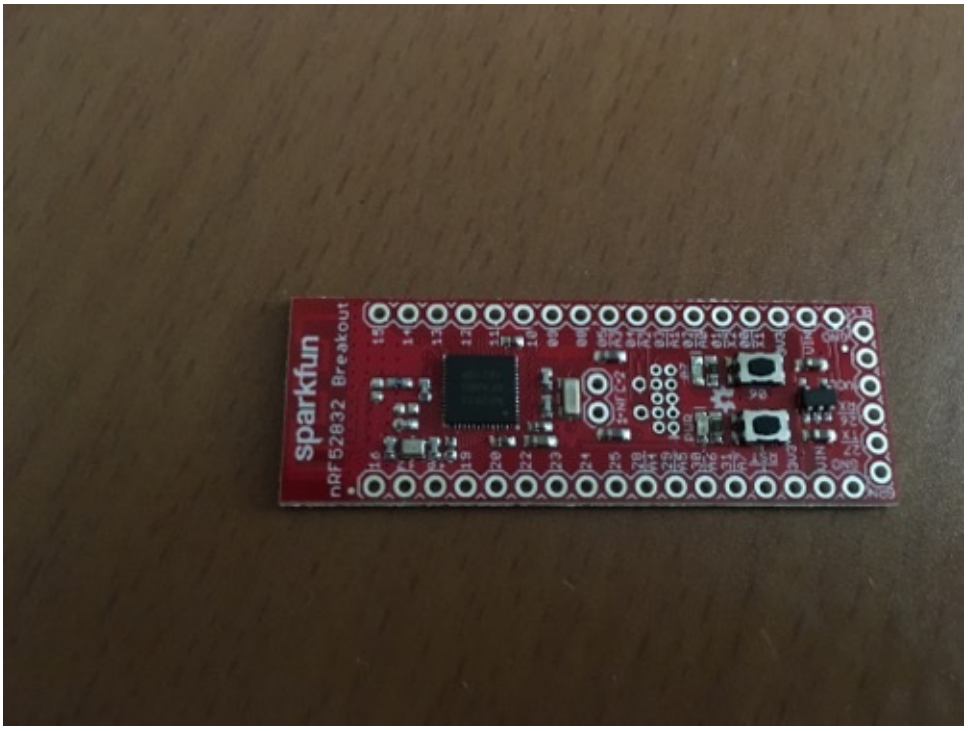
- Windows, http://esp-idf.readthedocs.io/en/latest/windows-setup.html
- Mac, http://esp-idf.readthedocs.io/en/latest/macos-setup.html
- Linux, http://esp-idf.readthedocs.io/en/latest/linux-setup.html

To write program, you can use any text editor. In this book, I use Visual Studio Code from Microsoft. It runs on any platform. You can download and install it on https://code.visualstudio.com.



1.4 Testing

For testing, I used SparkFun ESP32 Thing on Windows 10, macOS and Ubuntu.



I also used Arduino Sidekick Basic kit for electronic components.

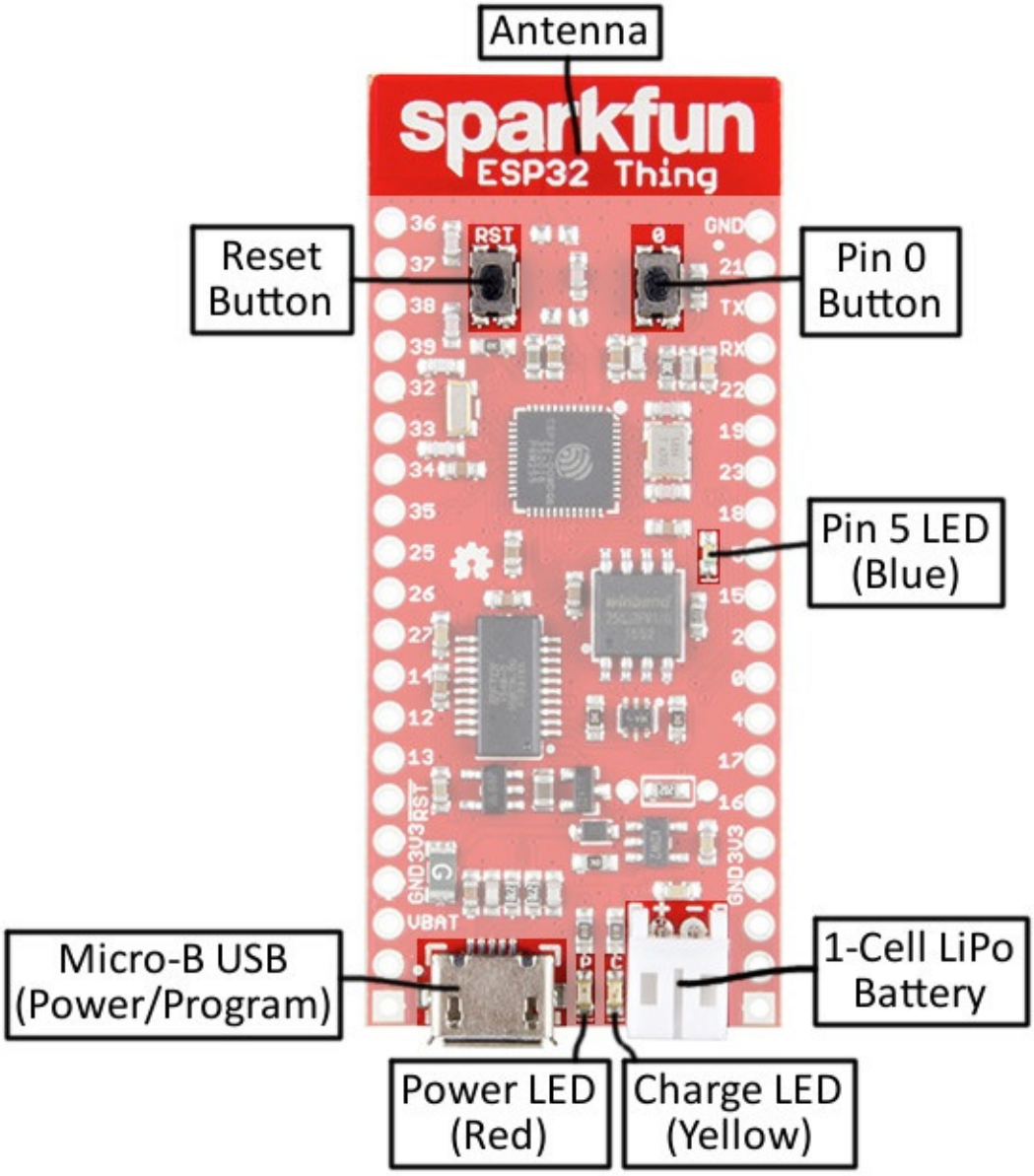
2. Setting Up SparkFun ESP32 Thing

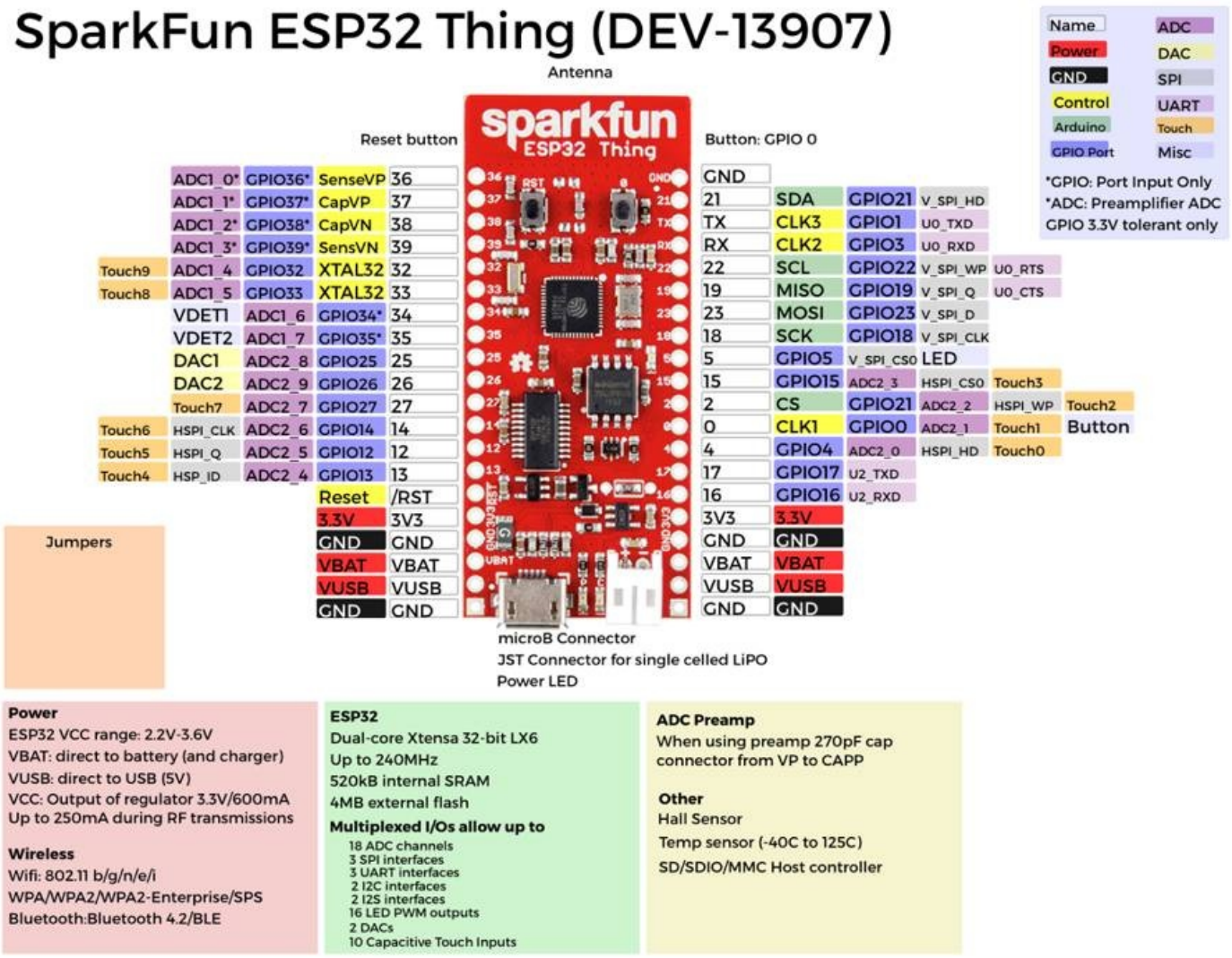
This chapter explains how to work on setting up SpakFun ESP32 Thing board.

2.1 Getting Started

In this chapter, we learn how to get started with SpakFun ESP32 Thing board. We try to build a simple app, Blink, on SpakFun ESP32 Thing board using Espressif IoT Development Framework.

The following is a scheme of SpakFun ESP32 Thing board.





(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

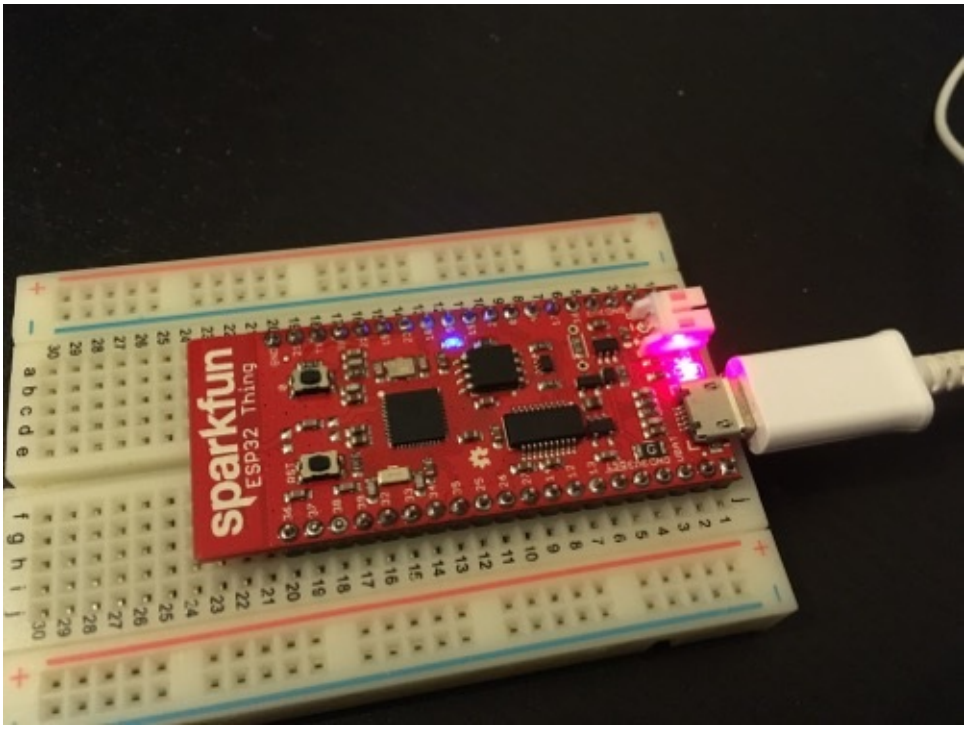
2.2 Installing Espressif IoT Development Framework

To develop app with SparkFun ESP32 Thing target, I use Espressif IoT Development Framework. Further information about this framework, you can read it on https://github.com/espressif/esp-idf. You can follow the installation process in these links

- Windows, http://esp-idf.readthedocs.io/en/latest/windows-setup.html
- Mac, http://esp-idf.readthedocs.io/en/latest/macos-setup.html
- Linux, http://esp-idf.readthedocs.io/en/latest/linux-setup.html

2.3 Connecting SparkFun ESP32 Thing board to Computer

After installed Espressif IoT Development Framework on your platform included the driver of SparkFun ESP32. Now we connect SparkFun ESP32 Thing board to computer via microUSB cable as follows:



On Terminal, you can check it. For Windows, you can check it on Device Manager. In Mac, you can type this command to check your board.

```
$ ls /dev/cu*
$ ls /dev/tty.*
```

The following is my Terminal output. It shows /dev/cu.usbserial-DN02MX9H and /dev/tty.usbserial-DN02MX9H.

If you see it, you’re ready to develop SparkFun ESP32 board.

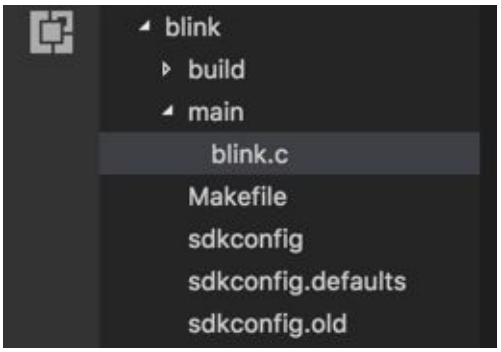
2.4 Hello SparkFun ESP32 Thing: Blinking LED

In this section, we build a blinking LED program using Blink program from Espressif IoT Development Framework. SparkFun ESP32 Thing board provides onboad LED which is connected on pin 5 (GPIO5).

Let’s start to write our Blink program as follows

- Create a folder, blink
- Inside blink folder, you create a new file, Makefile
- Then, create a folder, main
- Come inside main folder and create a new file, blink.c

You can check the project structure as below.



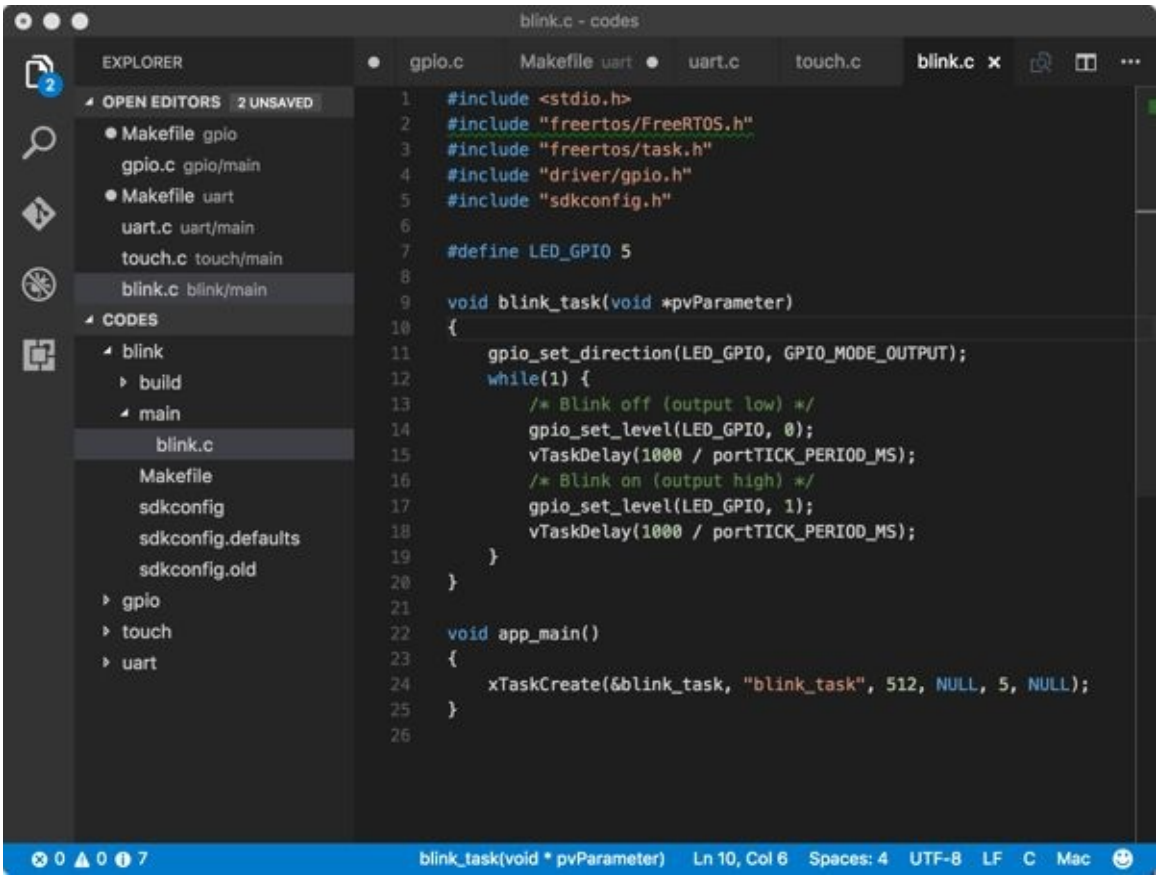
In Makefile file, you can type this script

```
PROJECT_NAME := blink

include $(IDF_PATH)/make/project.mk
```

The program in blink.c file can be written as follows.

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

#define LED_GPIO 5

void blink_task(void *pvParameter)
{
    gpio_set_direction(LED_GPIO, GPIO_MODE_OUTPUT);
    while(1) {
        /* Blink off (output low) */
        gpio_set_level(LED_GPIO, 0);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
```

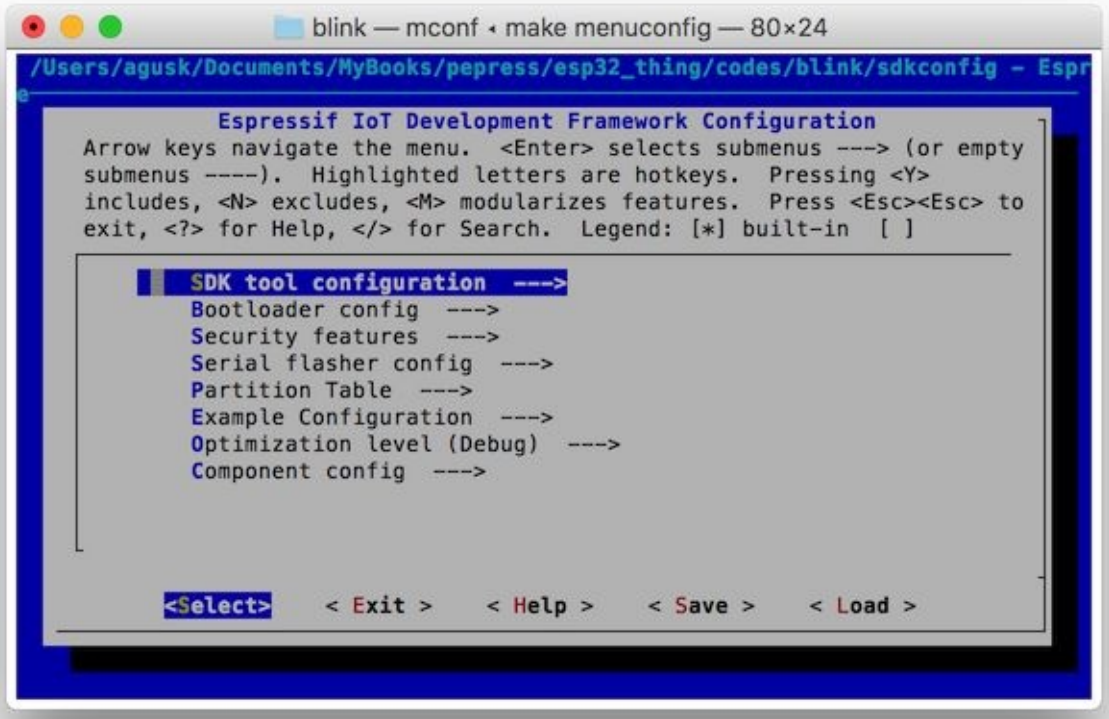
```
        /* Blink on (output high) */
        gpio_set_level(LED_GPIO, 1);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

void app_main()
{
    xTaskCreate(&blink_task, "blink_task", 512, NULL, 5, NULL);
}
```



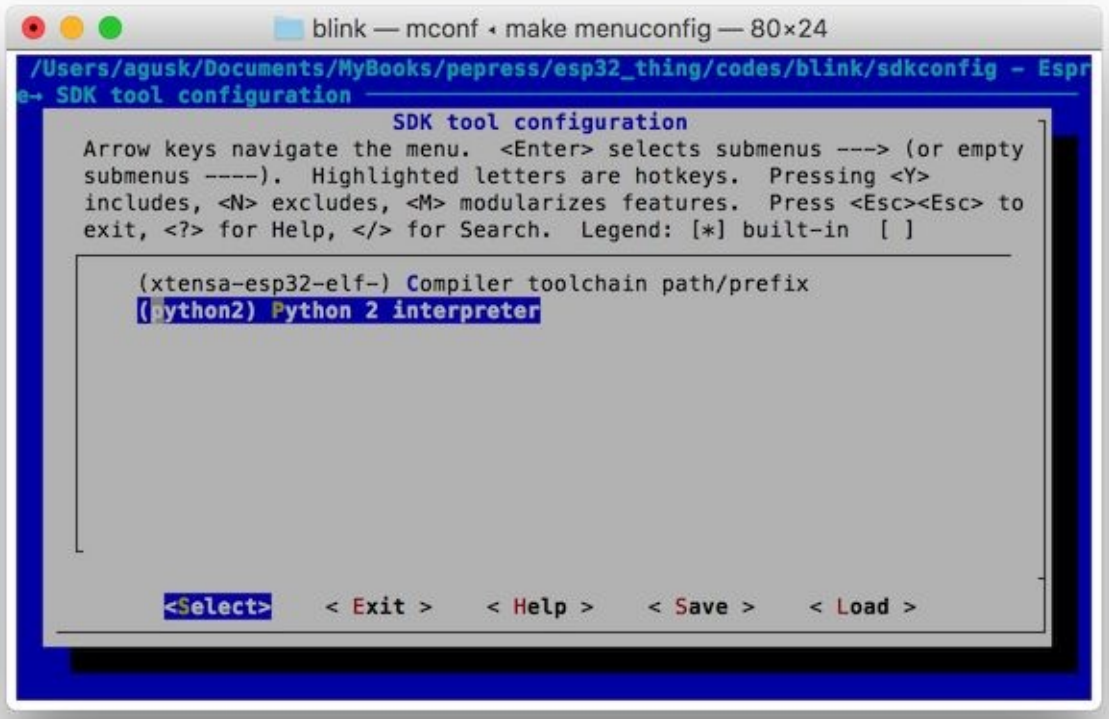
sdkconfig and sdkconfig.defaults files can generate using menuconfig.

```
$ make menuconfig
```

You can see the form as below.

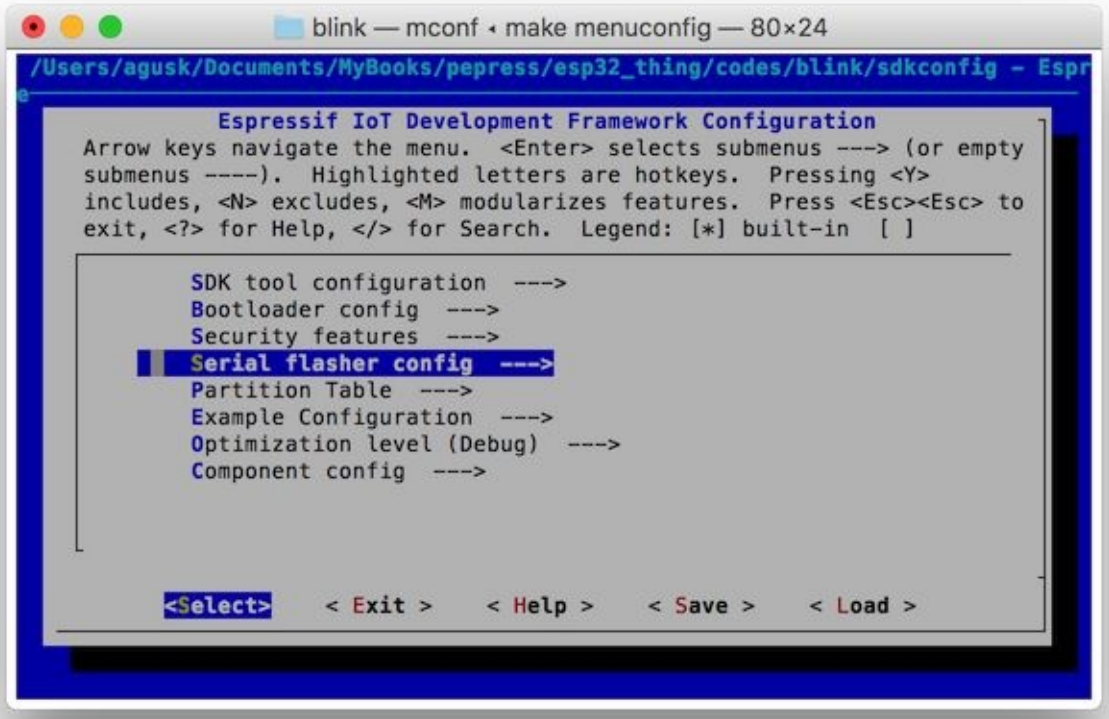


Select SDK tool configuration to configure the compiler included your Python.

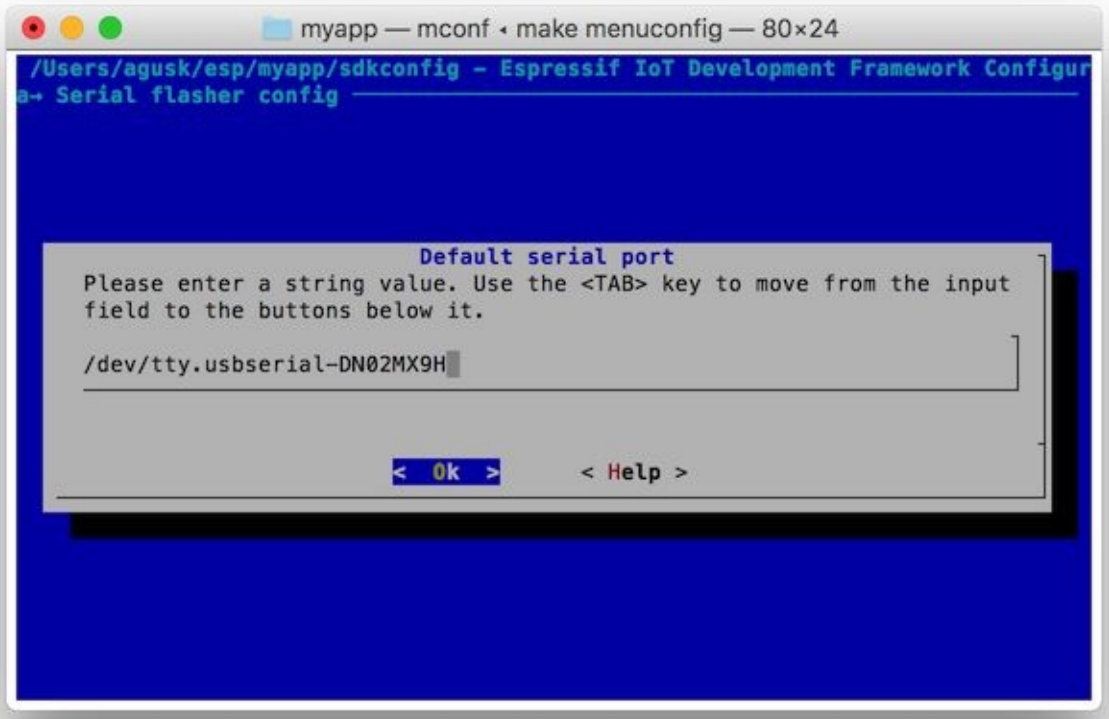


You also need to configure a serial port of SparkFun ESP32 board. Select Serial flasher

config.



Then, fill your serial port of the board.



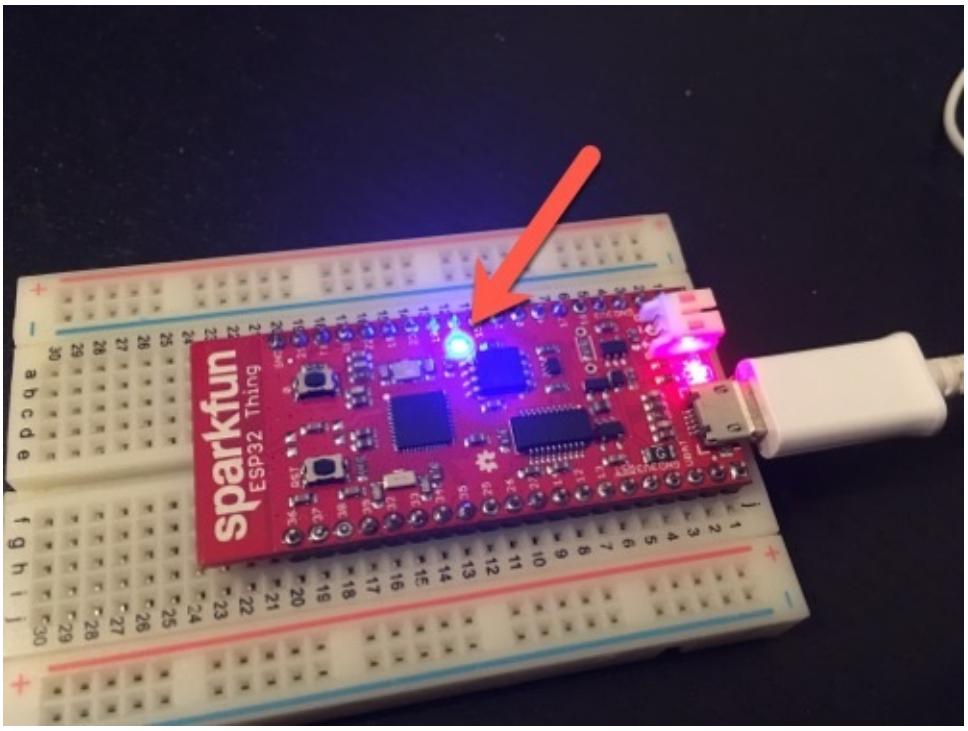
Save this config.

Now you can compile and upload the program into SparkFun ESP32 board. Type this command.

```
$ make flash
```

For the first compiling, it takes several minutes to complete.

If succeed, you can see the blinking LED.


sparkfun
ESP32 Thing

2.5 Updating and Erasing Program

If you have modified the program and want to upload to SparkFun ESP32 Thing board, you can update your program and then upload to SparkFun ESP32 board. Type this command.

```
$ make flash
```

You also can erase installed program from the board. You can type this command.

```
$ make erase_flash
```

2.6 Troubleshooting

Make sure you have done in section 2.2 for installation. You may need to update the framework library from Github, https://github.com/espressif/esp-idf. Using git, you can update it.

If your board does not recognized by computer, make sure you have installed the driver. Try to plug out from computer and then plug in again the board via micro USB. You may need to restart your computer in order to detect it.

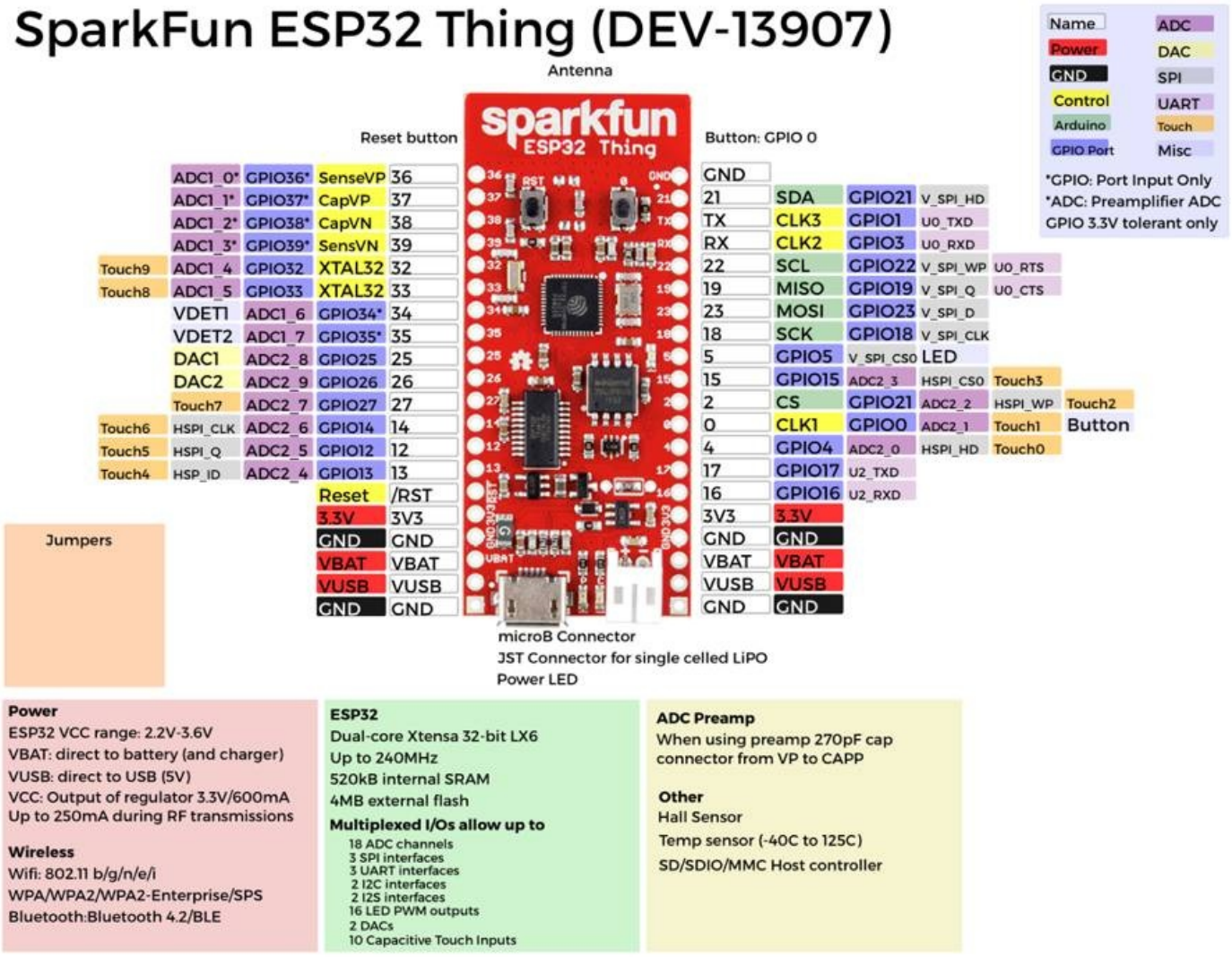
3. GPIO Programming

In this chapter I’m going to explain how to work with GPIO on SparkFun ESP32 Thing and write a program for demo.

3.1 Getting Started

In general, GPIO can be used to control digital I/O on SparkFun ESP32 Thing. To write data on SparkFun ESP32 Thing GPIO, we can use gpio_set_level() and use gpio_get_level() to read data from GPIO. Further information about GPIO API, you can read it on http://esp-idf.readthedocs.io/en/latest/api/peripherals/gpio.html.

You can see SparkFun ESP32 Thing GPIO pinout on the following Figure.



(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

In this chapter, we build a program to illustrate how SparkFun ESP32 Thing GPIO work. We need a LED and a pushbutton.

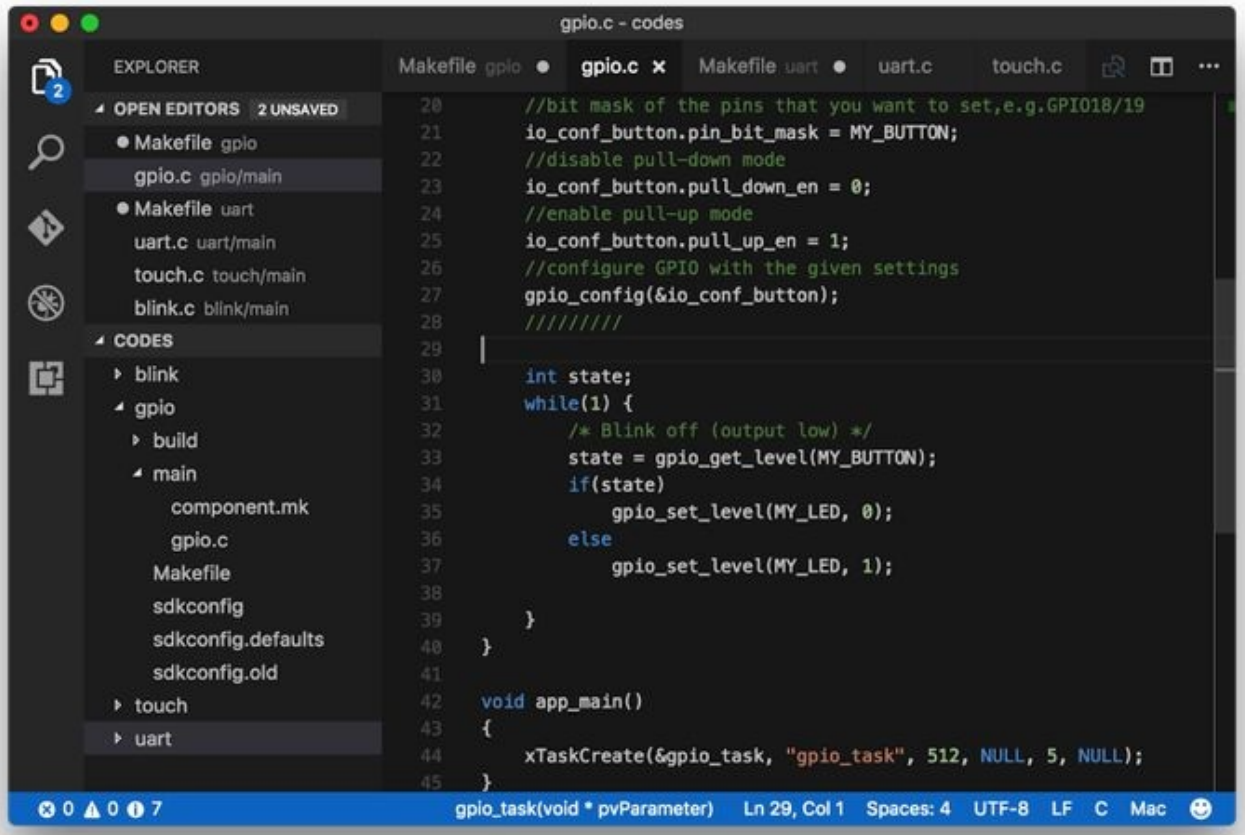
Let’s start!.

3.2 Wiring

We use built-in LED and pushbutton so there is no wiring in our demo. The built-in LED is connected to GPIO5 and built-in pushbutton is connected to GPIO0.

3.3 Writing a Program

You can start to create your project by following in section 2.4. The following is our project structure.



You need to add component.mk file with blank content. If not, you may get error while compiling.

In gpio.c file, we build a program to detect if the button is pressed. If pressed, LED will be lighting.

The following is completed code.

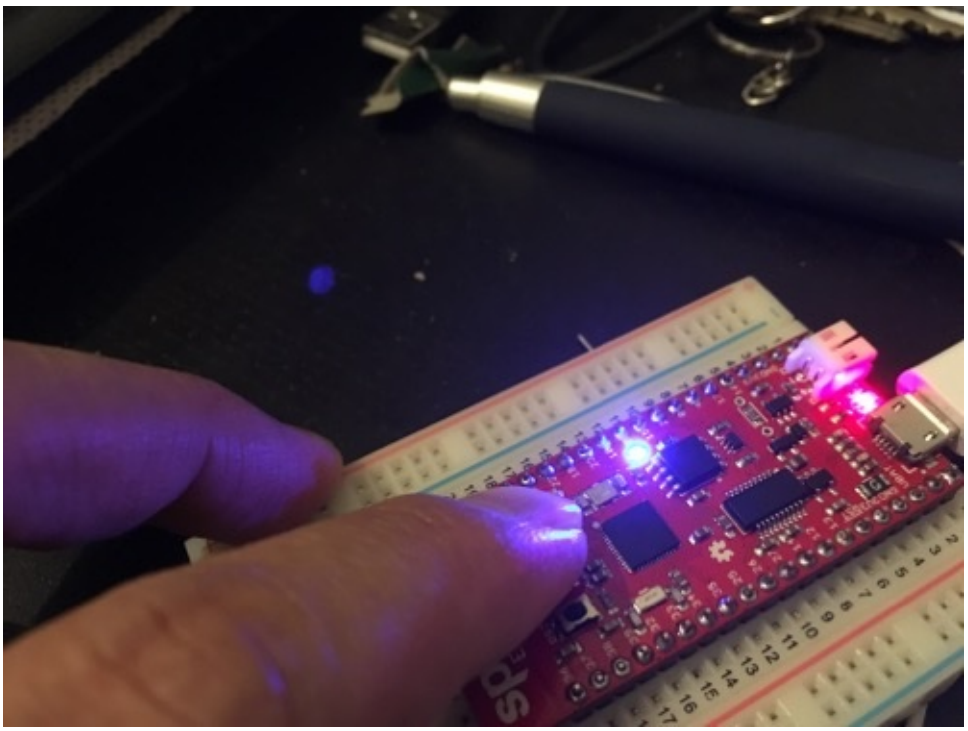
```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

#define MY_LED 5
#define MY_BUTTON 0

void gpio_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);
    gpio_set_direction(MY_BUTTON, GPIO_MODE_INPUT);
```

```
    gpio_set_level(MY_LED, 0);
    while(1) {
        if(gpio_get_level(MY_BUTTON))
            gpio_set_level(MY_LED, 0);
        else
            gpio_set_level(MY_LED, 1);
    }
}

void app_main()
{
    xTaskCreate(&gpio_task, "gpio_task", 512, NULL, 5, NULL);
}
```

Save all files.

3.4 Testing

Now you can compile and upload this program to SparkFun ESP32 Thing board. For testing, try to press pushbutton. You should see a lighting LED.



4. UART

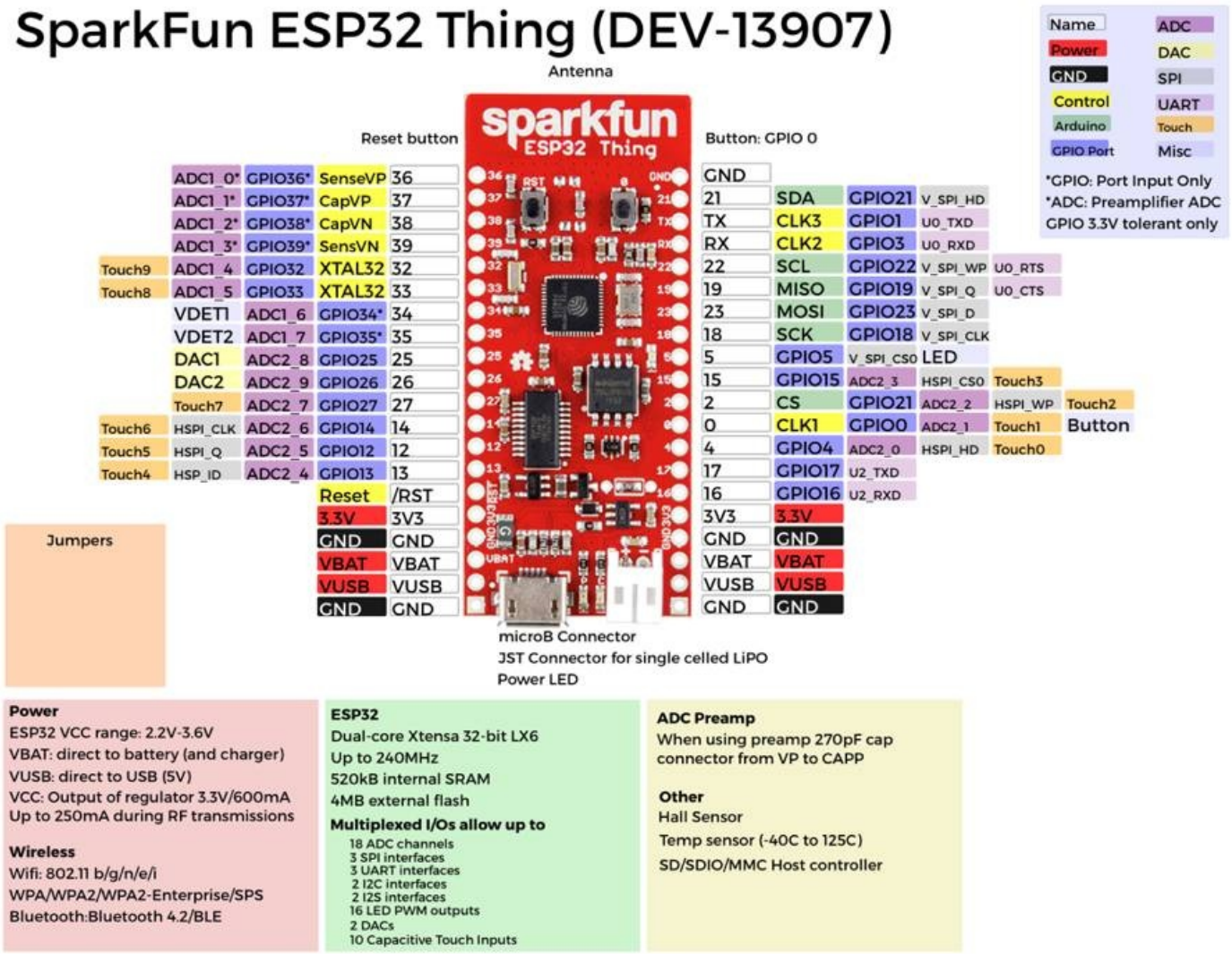
In this chapter I’m going to explain how to access UART on SparkFun ESP32 Thing board.

4.1 Getting Started

SparkFun ESP32 Thing provides UART which can be accessed via Espressif IoT Development framework. Further information about UART API, you can read it on http://esp-idf.readthedocs.io/en/latest/api/peripherals/uart.html.

In this chapter, we try to access SparkFun ESP32 Thing UART via serial adapter which is used to upload a program too.

Let’s start!.



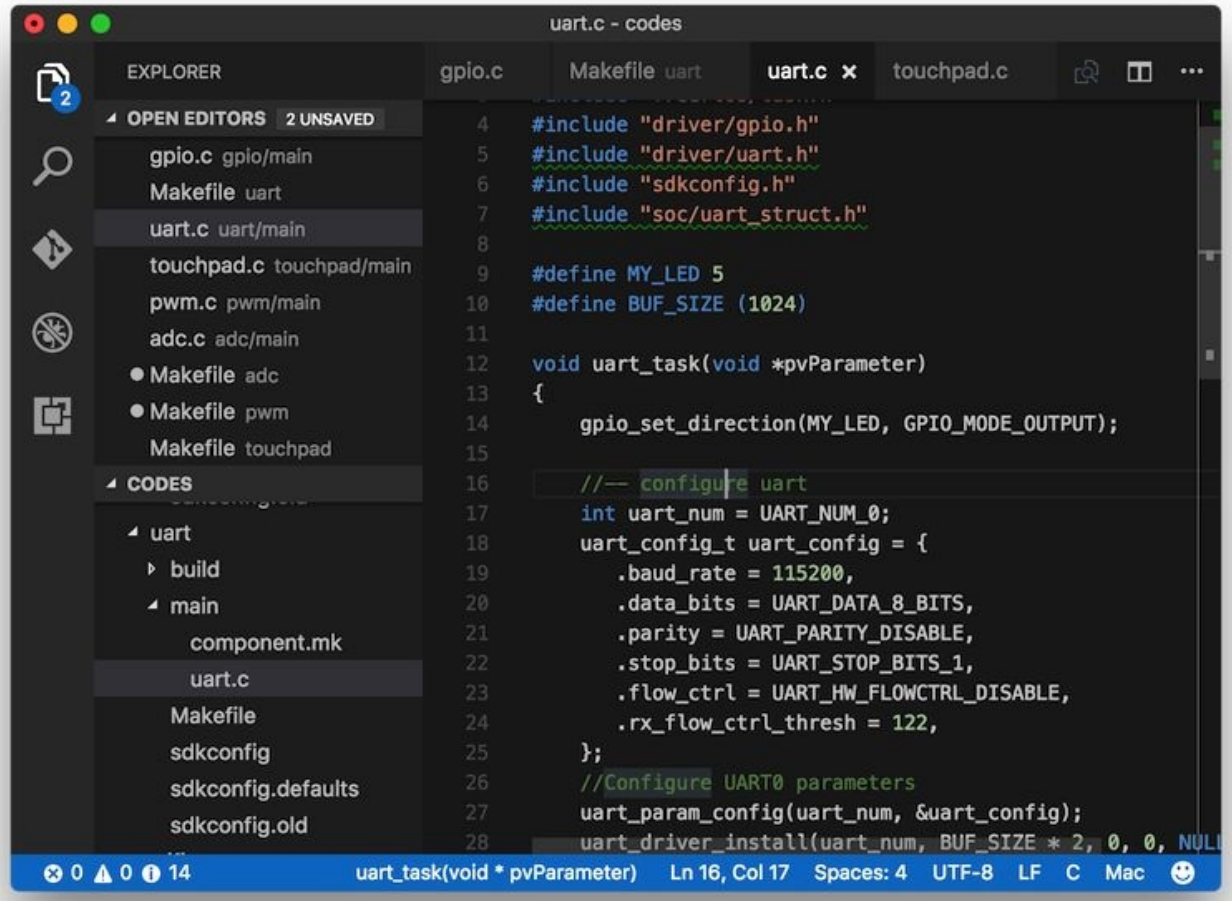
(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

4.2 Wiring

In this scenario, we don’t perform any wiring.

4.3 Writing a Program

Create our project structure in the following figure.



In uart.c file, we develop our program to send counter data to UART. The following is complete code.

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "sdkconfig.h"
#include "soc/uart_struct.h"

#define MY_LED 5
#define BUF_SIZE (1024)

void uart_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);

    //-- configure uart
    int uart_num = UART_NUM_0;
```

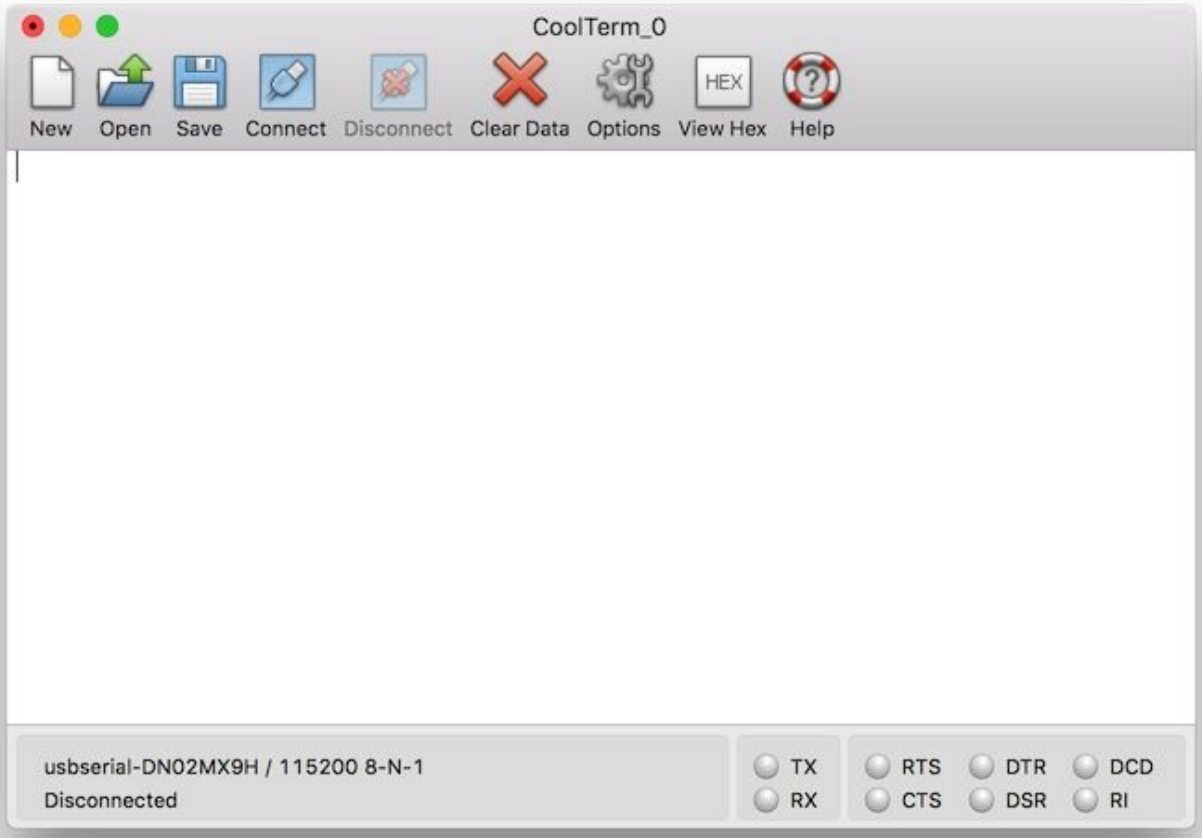
```
    uart_config_t uart_config = {
       .baud_rate = 115200,
       .data_bits = UART_DATA_8_BITS,
       .parity = UART_PARITY_DISABLE,
       .stop_bits = UART_STOP_BITS_1,
       .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
       .rx_flow_ctrl_thresh = 122,
    };
    //Configure UART0 parameters
    uart_param_config(uart_num, &uart_config);
    uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0);

    char data[15];
    int counter = 1;
    while(1) {
        sprintf(data,"Data %d\r\n", counter);

        //Write data back to UART
        gpio_set_level(MY_LED, 1);
        uart_write_bytes(uart_num, (const char*) data, sizeof(data));
        gpio_set_level(MY_LED, 0);
        vTaskDelay(3000 / portTICK_PERIOD_MS);

        counter++;
        if(counter>50)
            counter = 1;
    }
}

void app_main()
{
    xTaskCreate(&uart_task, "uart_task", 1024, NULL, 10, NULL);
}
```

Save this program.

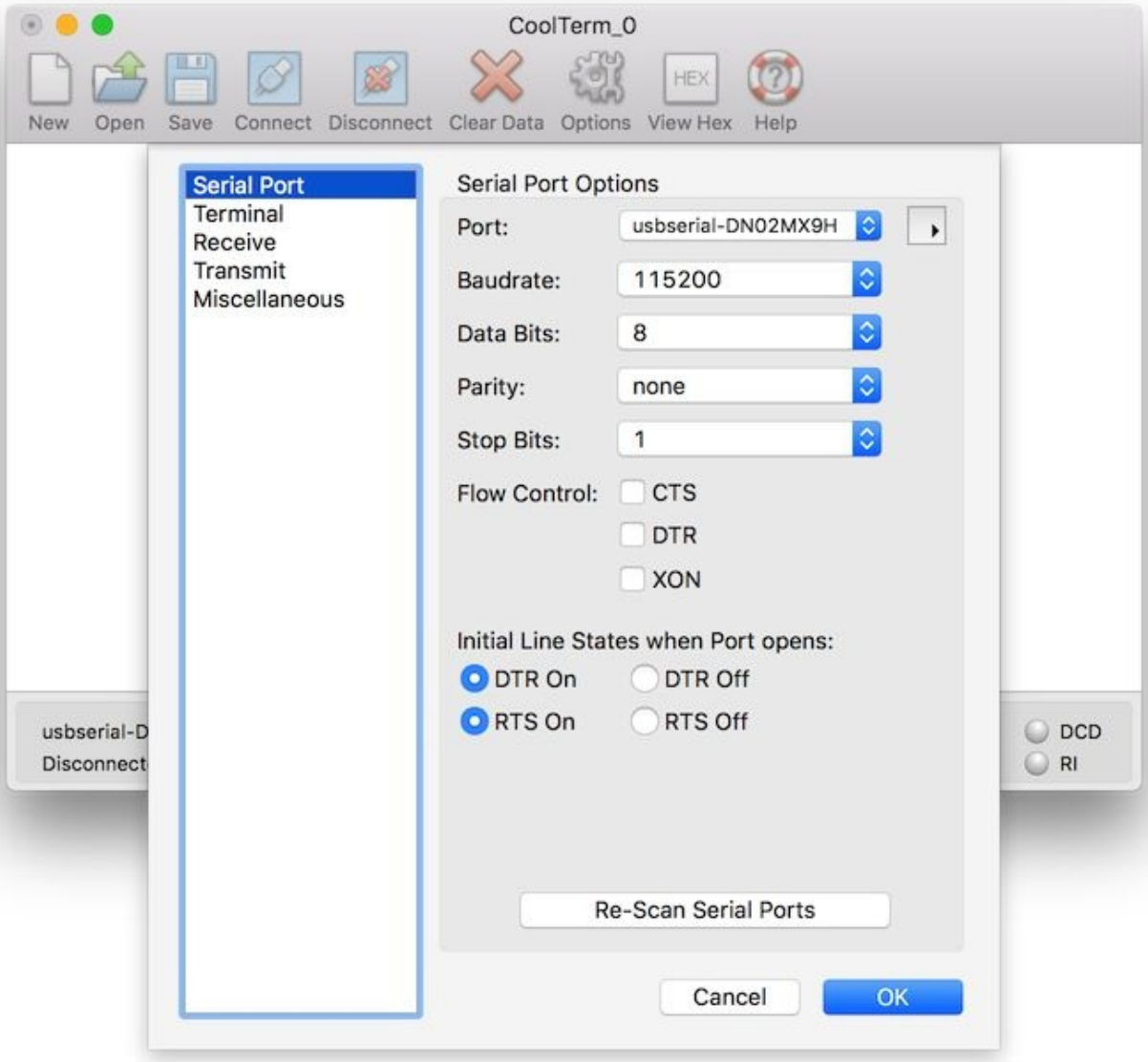
4.4 Testing

Now you can upload and run program.

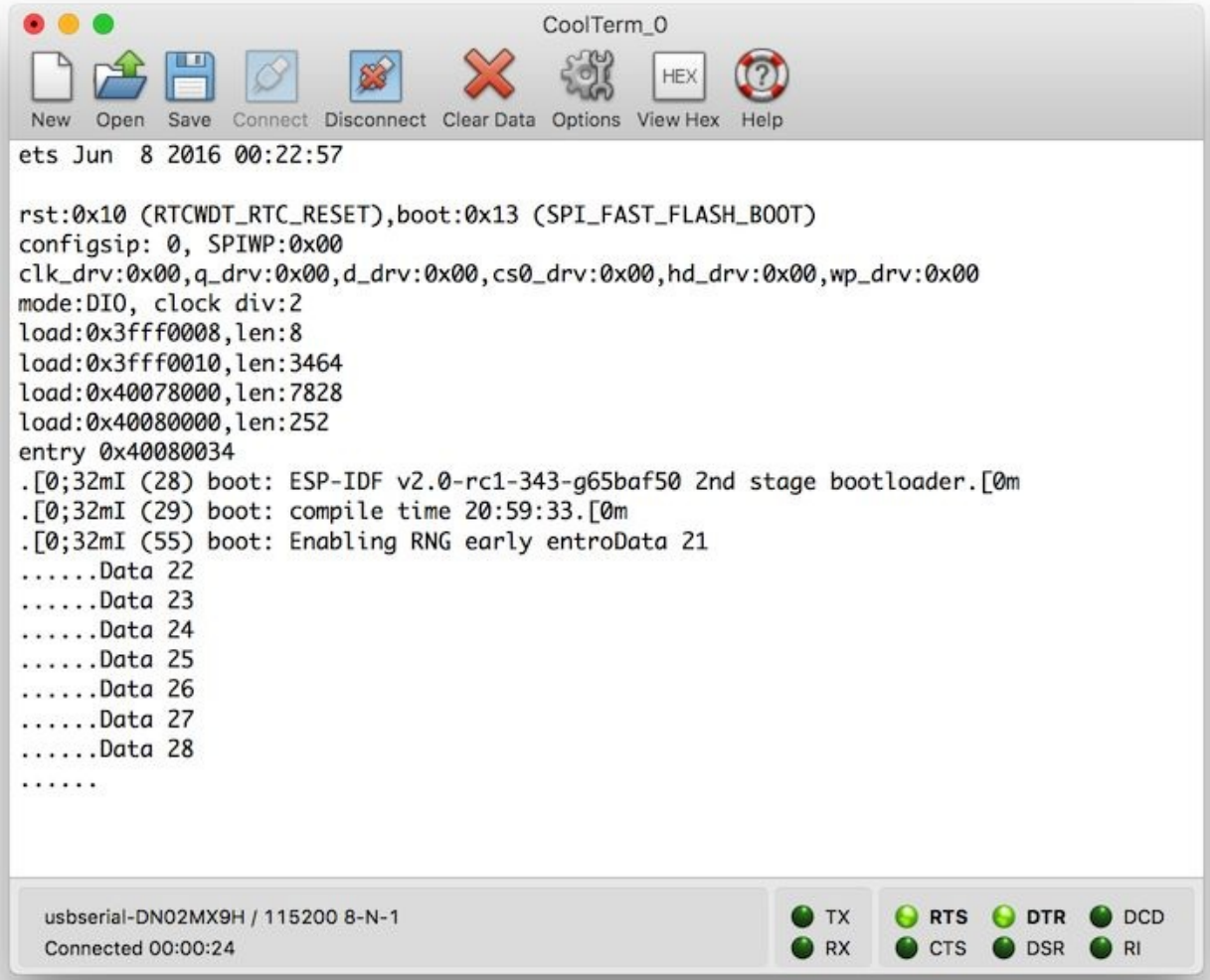
To see the UART output, you can use any serial tool such Arduino Serial Monitor tool from Arduino IDE. Set baud 115200 and Carriage return. You also can use CoolTerm, http://freeware.the-meiers.org.



You set a serial port of SparkFun ESP32 Thing board with baudrate 115200.



Then, try to connect. You should see output messages on this tool.

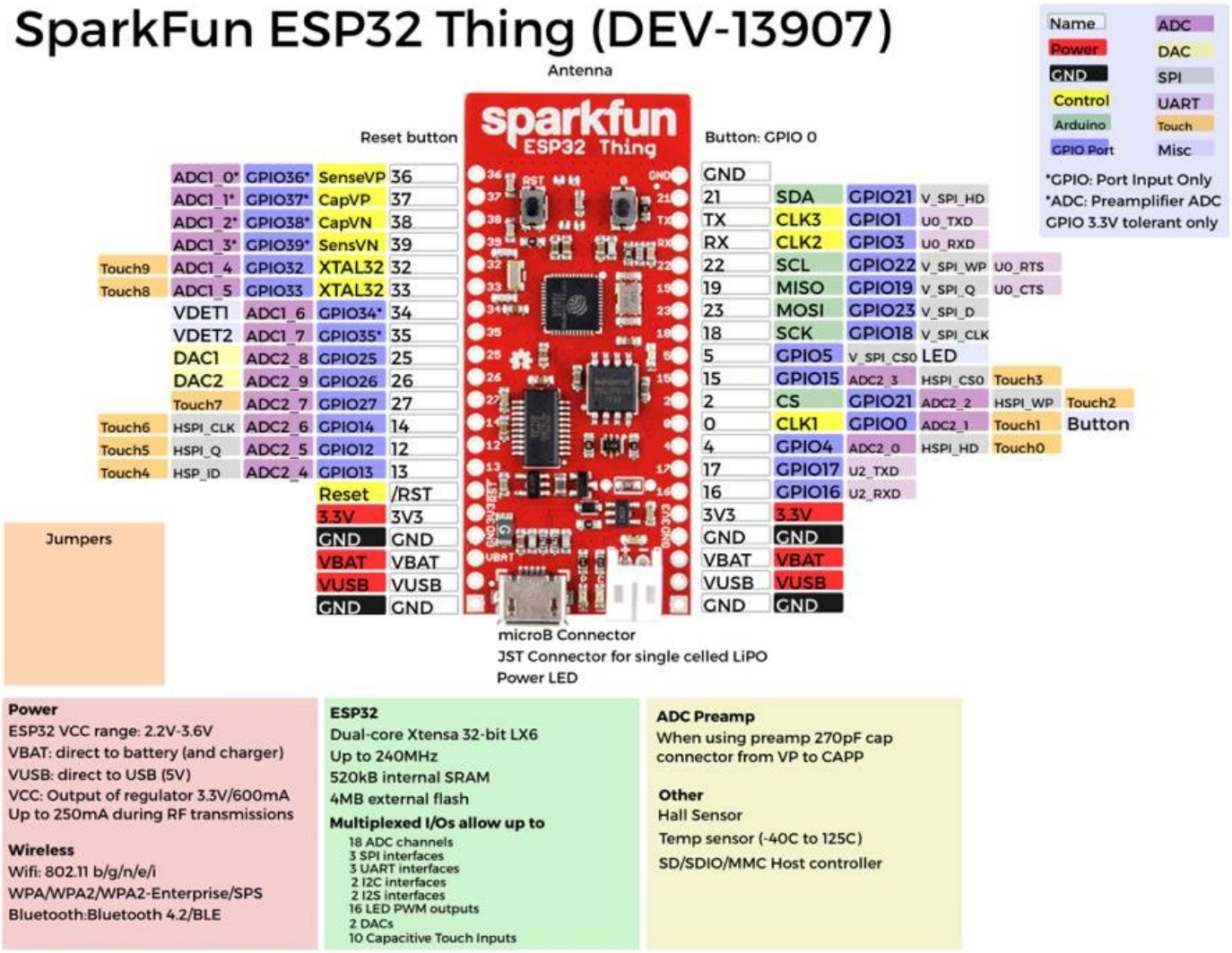

CoolTerm_0
New
Open
Save
Connect
Disconnect
Clear Data
Options
HEX
View Hex
Help
ets Jun 8 2016 00:22:57
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0x00
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0008,len:8
load:0x3fff0010,len:3464
load:0x40078000,len:7828
load:0x40080000,len:252
entry 0x40080034
.[0;32mI (28) boot: ESP-IDF v2.0-rc1-343-g65baf50 2nd stage bootloader.[0m
.[0;32mI (29) boot: compile time 20:59:33.[0m
.[0;32mI (55) boot: Enabling RNG early entroData 21
......Data 22
......Data 23
......Data 24
......Data 25
......Data 26
......Data 27
......Data 28
......
usbserial-DN02MX9H / 115200 8-N-1
Connected 00:00:24
TX
RX
RTS
CTS
DTR
DSR
DCD
RI

5. Touch Pad

In this chapter we explore how to work with touch pad on SparkFun ESP32 Thing.

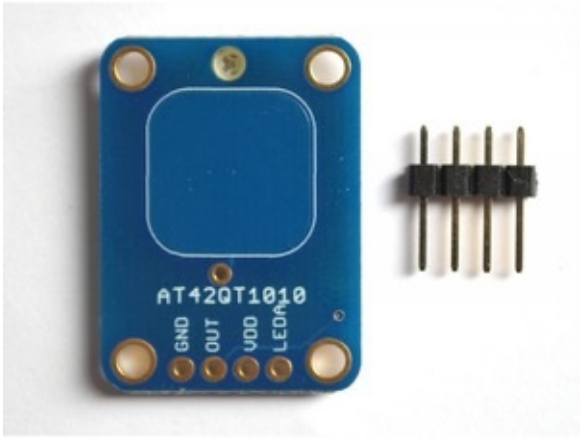
5.1 Getting Started

SparkFun ESP32 Thing provides touch sensor pins which we can use them. You can see touch pins on brown colors in the following layout.



(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

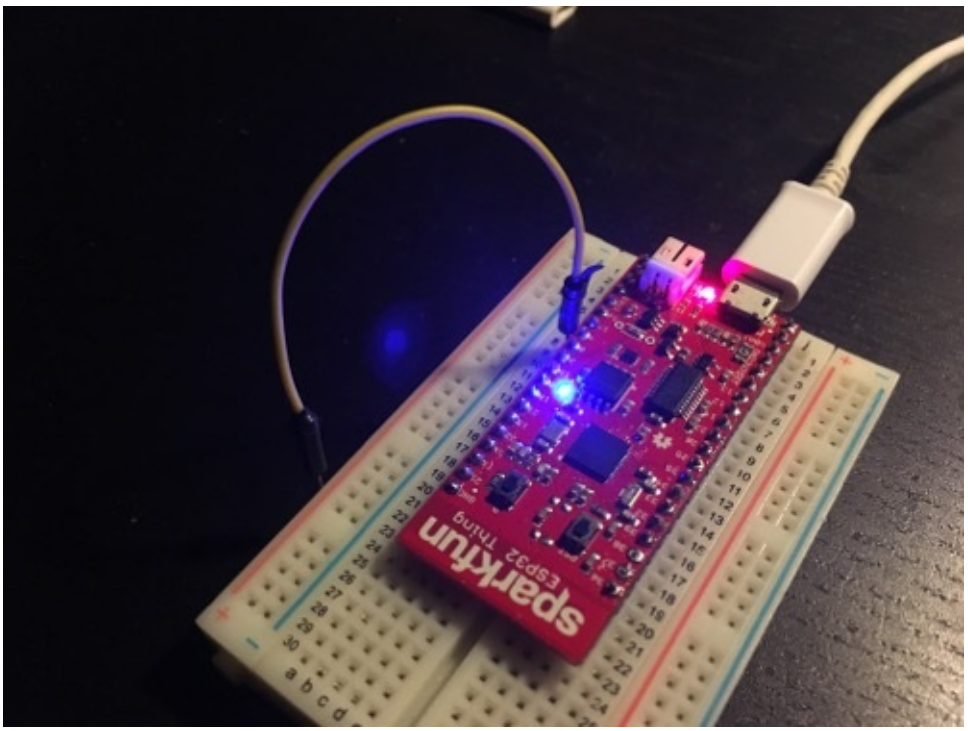
To work with touch sensor, you should not have sensor sensor device such as Standalone Momentary Capacitive Touch Sensor Breakout, https://www.adafruit.com/products/1374.



In this chapter, I use a jumper cable for testing.

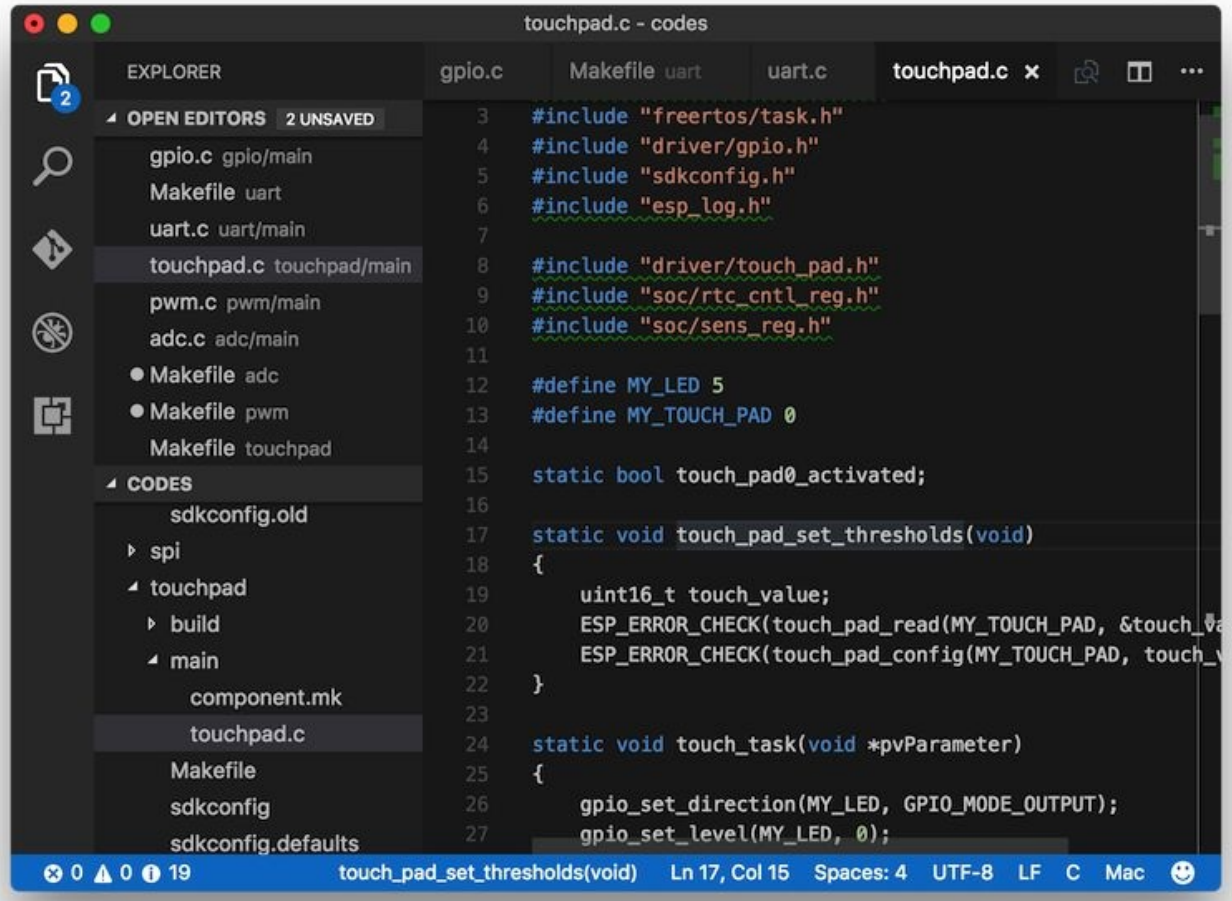
5.2 Wiring

For demo, I use Touch0 pin which is located on GPIO4 (pin 4) on SparkFun ESP32 Thing. Just connect a jumper cable on it.



5.3 Write Program

Firstly, we create a project structure as below.



Our program is written in touchpad.c file. We use touch_pad_read() to read touch sensor. For demo, we will turn on LED if Touch0 is touched. The following is complete code.

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"

#include "driver/touch_pad.h"
#include "soc/rtc_cntl_reg.h"
#include "soc/sens_reg.h"

#define MY_LED 5
#define MY_TOUCH_PAD 0

static bool touch_pad0_activated;
```

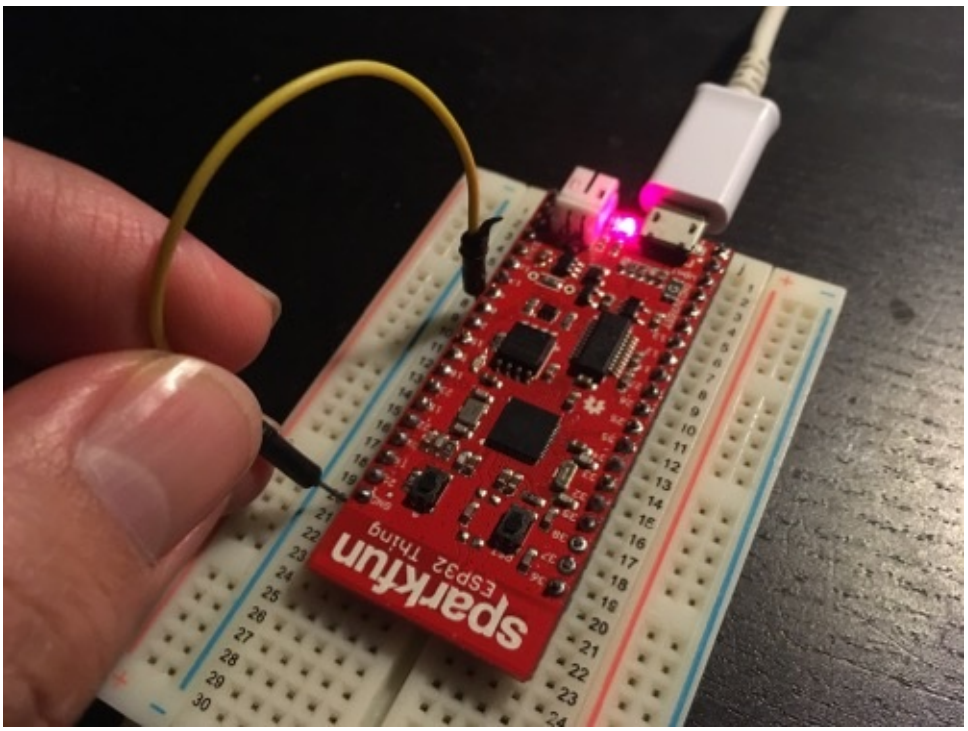
```
static void touch_pad_set_thresholds(void)
{
    uint16_t touch_value;
    ESP_ERROR_CHECK(touch_pad_read(MY_TOUCH_PAD, &touch_value));
    ESP_ERROR_CHECK(touch_pad_config(MY_TOUCH_PAD, touch_value/2));
}

static void touch_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);
    gpio_set_level(MY_LED, 0);

    static int led_counter;
    while(1) {

        if(touch_pad0_activated) {
            gpio_set_level(MY_LED, 1);
            vTaskDelay(300 / portTICK_PERIOD_MS);

            touch_pad0_activated = false;
            led_counter = 1;
        }


        if (led_counter++ % 500 == 0) {
            if(!touch_pad0_activated)
                gpio_set_level(MY_LED, 0);
        }

    }
}
static void touch_pad_rtc_intr(void * arg)
{
    uint32_t pad_intr = READ_PERI_REG(SENS_SAR_TOUCH_CTRL2_REG) & 0x3ff;
    uint32_t rtc_intr = READ_PERI_REG(RTC_CNTL_INT_ST_REG);
    //clear interrupt
    WRITE_PERI_REG(RTC_CNTL_INT_CLR_REG, rtc_intr);
    SET_PERI_REG_MASK(SENS_SAR_TOUCH_CTRL2_REG, SENS_TOUCH_MEAS_EN_CLR);

    if (rtc_intr & RTC_CNTL_TOUCH_INT_ST) {
        if ((pad_intr >> MY_TOUCH_PAD) & 0x01)
            touch_pad0_activated = true;
    }
}

void app_main()
{
    // // Initialize touch pad peripheral
    touch_pad_init();
    touch_pad_set_thresholds();
    touch_pad_isr_handler_register(touch_pad_rtc_intr, NULL, 0, NULL);
```

```
    xTaskCreate(&touch_task, "touch_task", 2048, NULL, 5, NULL);
}
```

Save all files.

5.4 Testing

Now you can compile and flash the program into SparkFun ESP32 Thing board. To simulate touch, you can touch the jumper cable. Then, LED will be lighting.

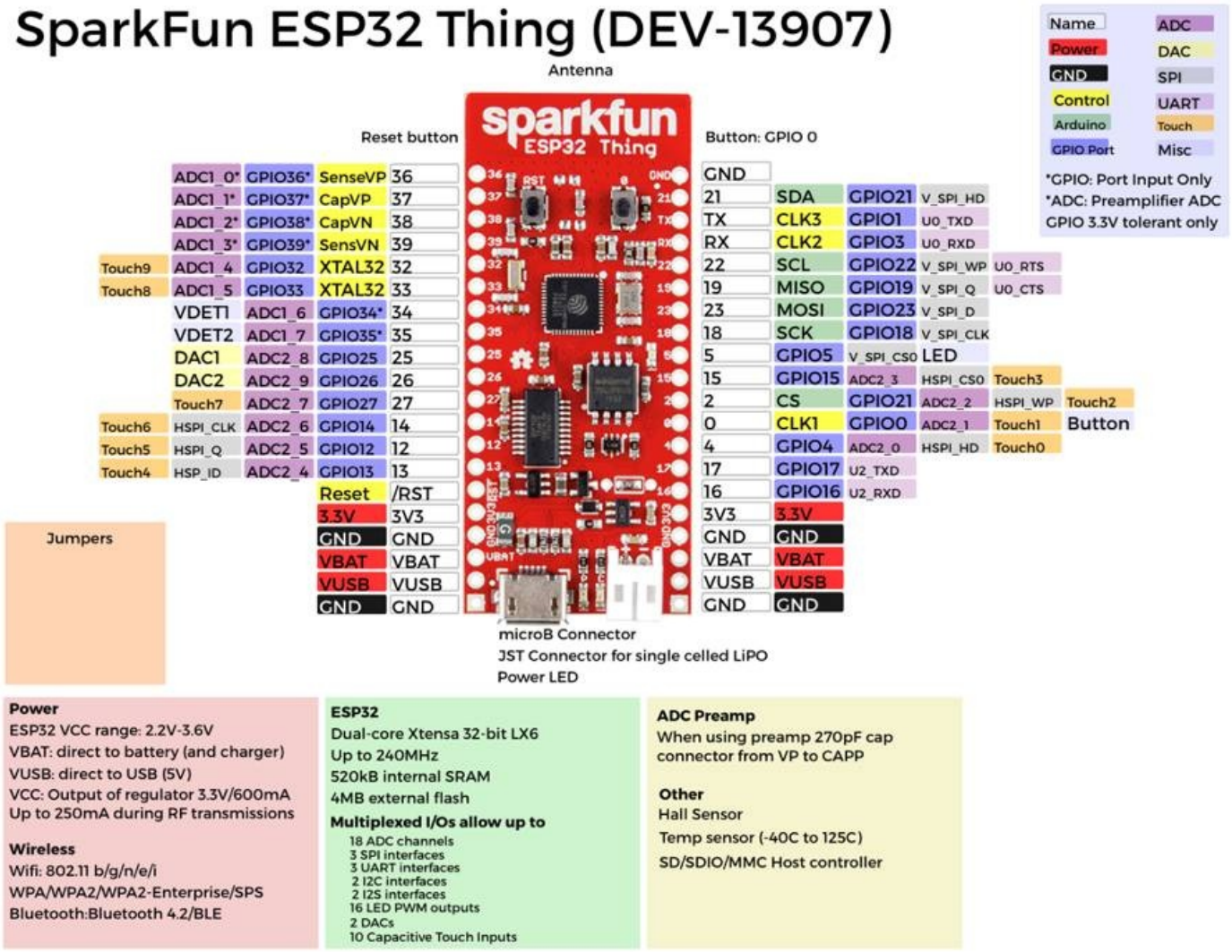


6. PWM and Analog Input

This chapter explains how to work with SparkFun ESP32 Thing Analog I/O.

6.1 Getting Started

SparkFun ESP32 Thing board provides Analog I/O which can be connected to sensor or actuator devices. See the following of SparkFun ESP32 Thing GPIO.



(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

In this chapter, we try to access SparkFun ESP32 Thing Analog I/O using Arduino software. There are two scenarios for our cases:

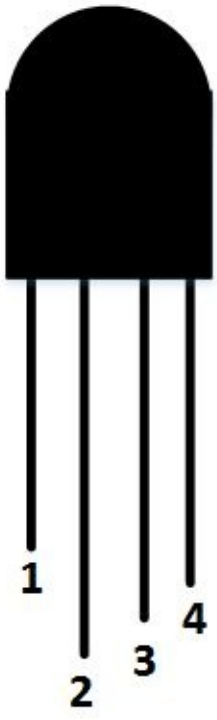
- Controlling RGB LED
- Reading Analog input using Potentiometer

Let’s start.

6.2 Demo Analog Output (PWM) : RGB LED

In this scenario we build a program to control RGB LED color using SparkFun ESP32 Thing Analog output (PWM). RGB LED has 4 pins that you can see it on Figure below.

To understand these pins, you can see the following Figure.



Note:

- Pin 1: Red
- Pin 2: Common pin

- Pin 3: Green
- Pin 4: Blue

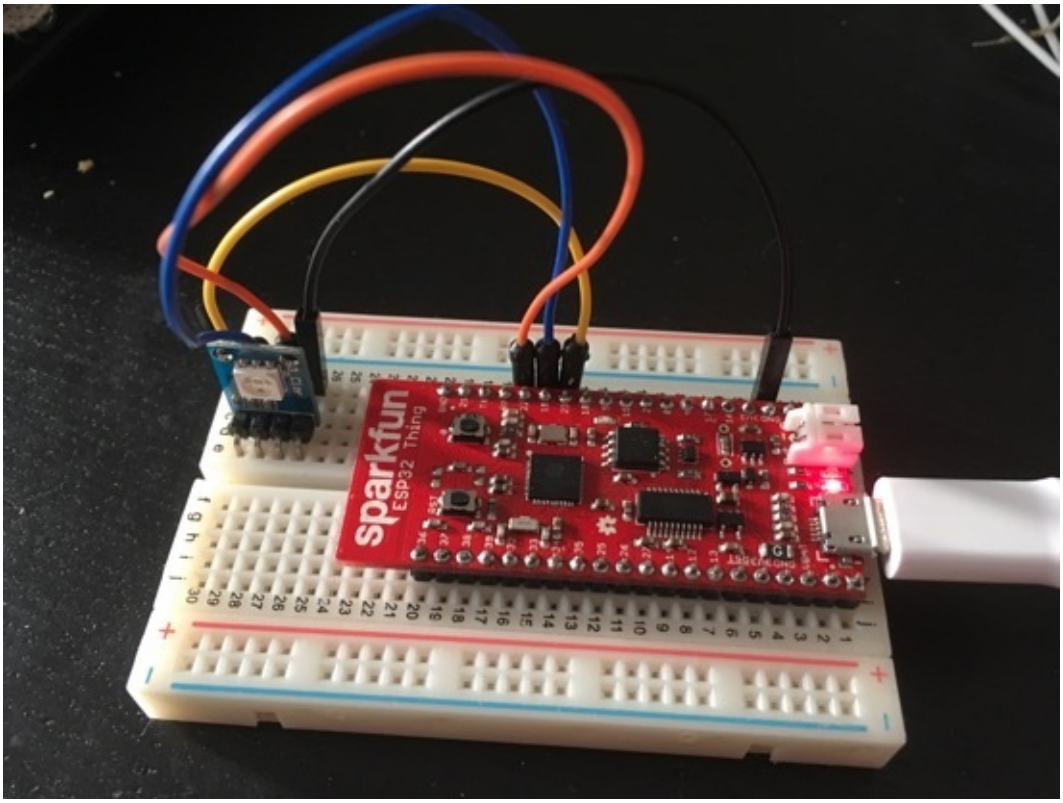
Now we can start to build a program and hardware implementation.

6.2.1 Wiring

For our testing, we configure the following PWM pins.

- RGB LED pin 1 (red) is connected to SparkFun ESP32 Thing pin GPIO22
- RGB LED pin 2 is connected to SparkFun ESP32 Thing 3V3 (VCC +3.3V)
- RGB LED pin 3 (green) is connected to SparkFun ESP32 Thing pin GPIO19
- RGB LED pin 4 (blue) is connected to SparkFun ESP32 Thing pin GPIO23

Here is a sample implementation with SparkFun ESP32 Thing and RGB Led.

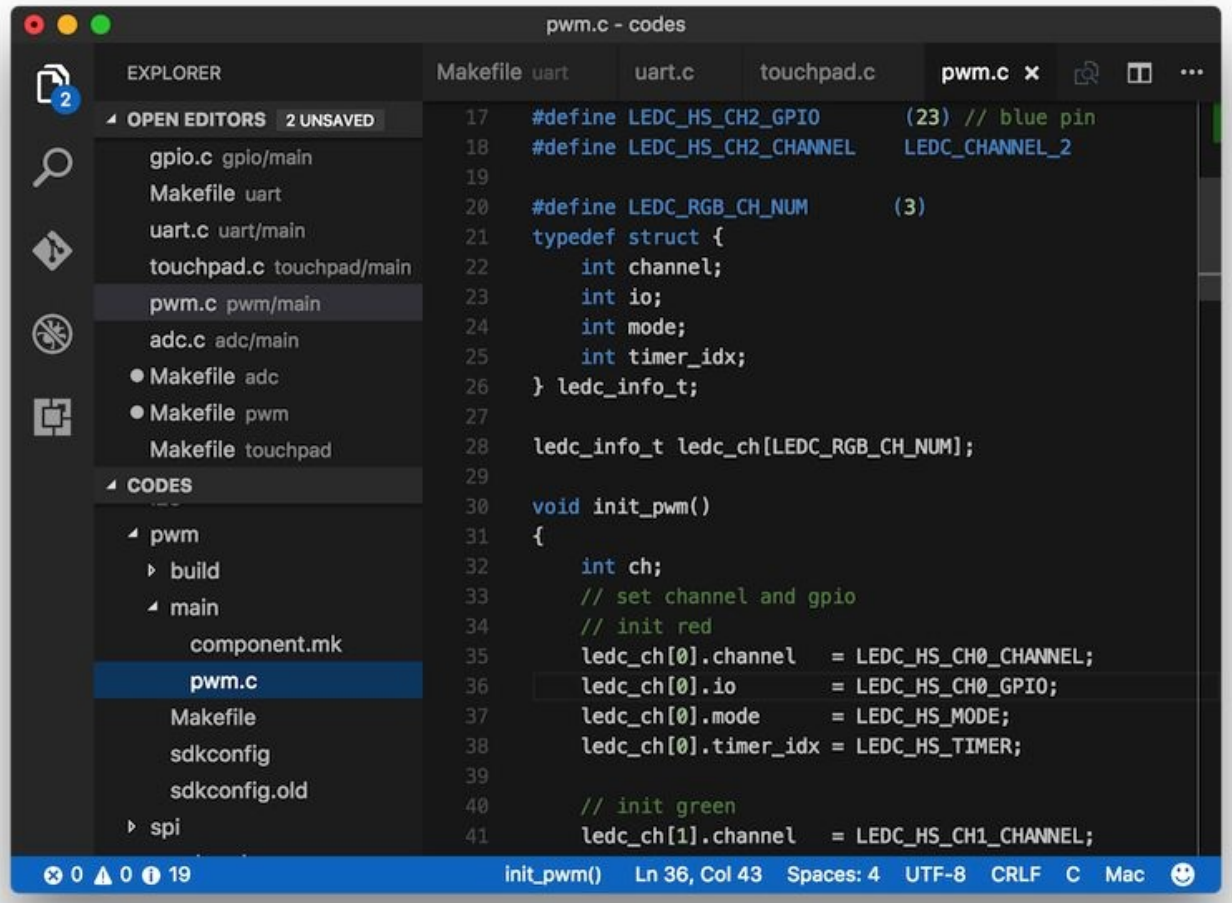


6.2.2 Writing Program

To display a certain color, we must combine colors from red, green, blue.
SparkFun ESP32 Thing provides API for PWM using ledc_set_duty() with input value

from 0 to 1023.

The following is our project structure.



Let”s start to build a program in pwm.c file. The following is complete code.

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/xtensa_api.h"
#include "freertos/queue.h"
#include "driver/ledc.h"
#include "esp_attr.h"
#include "esp_err.h"


#define LEDC_HS_TIMER          LEDC_TIMER_0
#define LEDC_HS_MODE           LEDC_HIGH_SPEED_MODE
#define LEDC_HS_CH0_GPIO       (22) // red pin
#define LEDC_HS_CH0_CHANNEL    LEDC_CHANNEL_0
#define LEDC_HS_CH1_GPIO       (19) // green pin
#define LEDC_HS_CH1_CHANNEL    LEDC_CHANNEL_1
#define LEDC_HS_CH2_GPIO       (23) // blue pin
#define LEDC_HS_CH2_CHANNEL    LEDC_CHANNEL_2
```

```
#define LEDC_RGB_CH_NUM       (3)
typedef struct {
    int channel;
    int io;
    int mode;
    int timer_idx;
} ledc_info_t;

ledc_info_t ledc_ch[LEDC_RGB_CH_NUM];

void init_pwm()
{
    int ch;
    // set channel and gpio
    // init red
    ledc_ch[0].channel   = LEDC_HS_CH0_CHANNEL;
    ledc_ch[0].io        = LEDC_HS_CH0_GPIO;
    ledc_ch[0].mode      = LEDC_HS_MODE;
    ledc_ch[0].timer_idx = LEDC_HS_TIMER;

    // init green
    ledc_ch[1].channel   = LEDC_HS_CH1_CHANNEL;
    ledc_ch[1].io        = LEDC_HS_CH1_GPIO;
    ledc_ch[1].mode      = LEDC_HS_MODE;
    ledc_ch[1].timer_idx = LEDC_HS_TIMER;

    // init blue
    ledc_ch[2].channel   = LEDC_HS_CH2_CHANNEL;
    ledc_ch[2].io        = LEDC_HS_CH2_GPIO;
    ledc_ch[2].mode      = LEDC_HS_MODE;
    ledc_ch[2].timer_idx = LEDC_HS_TIMER;

    ledc_timer_config_t ledc_timer = {
        .bit_num = LEDC_TIMER_10_BIT, //set timer counter 10 bit number
        .freq_hz = 5000,              //set frequency of pwm
        .speed_mode = LEDC_HS_MODE,   //timer mode,
        .timer_num = LEDC_HS_TIMER    //timer index
    };
    //configure timer0 for high speed channels
    ledc_timer_config(&ledc_timer);

    // initialize channel config
    for (ch = 0; ch < LEDC_RGB_CH_NUM; ch++) {
        ledc_channel_config_t ledc_channel = {
            //set LEDC channel 0
            .channel = ledc_ch[ch].channel,
            //set the duty for initialization.(duty range is 0 ~ ((2**bi
            .duty = 0,
            //GPIO number
            .gpio_num = ledc_ch[ch].io,
            //GPIO INTR TYPE, as an example, we enable fade_end interrup
            .intr_type = LEDC_INTR_FADE_END,
```

```
            //set LEDC mode, from ledc_mode_t
            .speed_mode = ledc_ch[ch].mode,
            //set LEDC timer source, if different channel use one timer,
            //the frequency and bit_num of these channels should be the
            .timer_sel = ledc_ch[ch].timer_idx,
        };
        //set the configuration
        ledc_channel_config(&ledc_channel);
    }

    //initialize fade service.
    ledc_fade_func_install(0);
}

void set_rgb_color(uint32_t red, uint32_t green, uint32_t blue)
{
    // value should 0...1024 for 10 bit number
    ledc_set_duty(ledc_ch[0].mode, ledc_ch[0].channel,red);
    ledc_update_duty(ledc_ch[0].mode, ledc_ch[0].channel);

    ledc_set_duty(ledc_ch[1].mode, ledc_ch[1].channel,green);
    ledc_update_duty(ledc_ch[1].mode, ledc_ch[1].channel);

    ledc_set_duty(ledc_ch[2].mode, ledc_ch[2].channel,blue);
    ledc_update_duty(ledc_ch[2].mode, ledc_ch[2].channel);
}

void app_main()
{
    init_pwm();

    while (1) {
        // 10 bit - duty=0..1023
        // red
        set_rgb_color(1023,0,0);
        vTaskDelay(2000 / portTICK_PERIOD_MS);

        // blue
        set_rgb_color(0,1023,0);
        vTaskDelay(2000 / portTICK_PERIOD_MS);

        // green
        set_rgb_color(0,0,1023);
        vTaskDelay(2000 / portTICK_PERIOD_MS);

        // yellow
        set_rgb_color(1023,1023,0);
        vTaskDelay(2000 / portTICK_PERIOD_MS);

        // purple
        set_rgb_color(323,0,323);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
```

```
        // aqua
        set_rgb_color(0,1023,1023);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
}
```

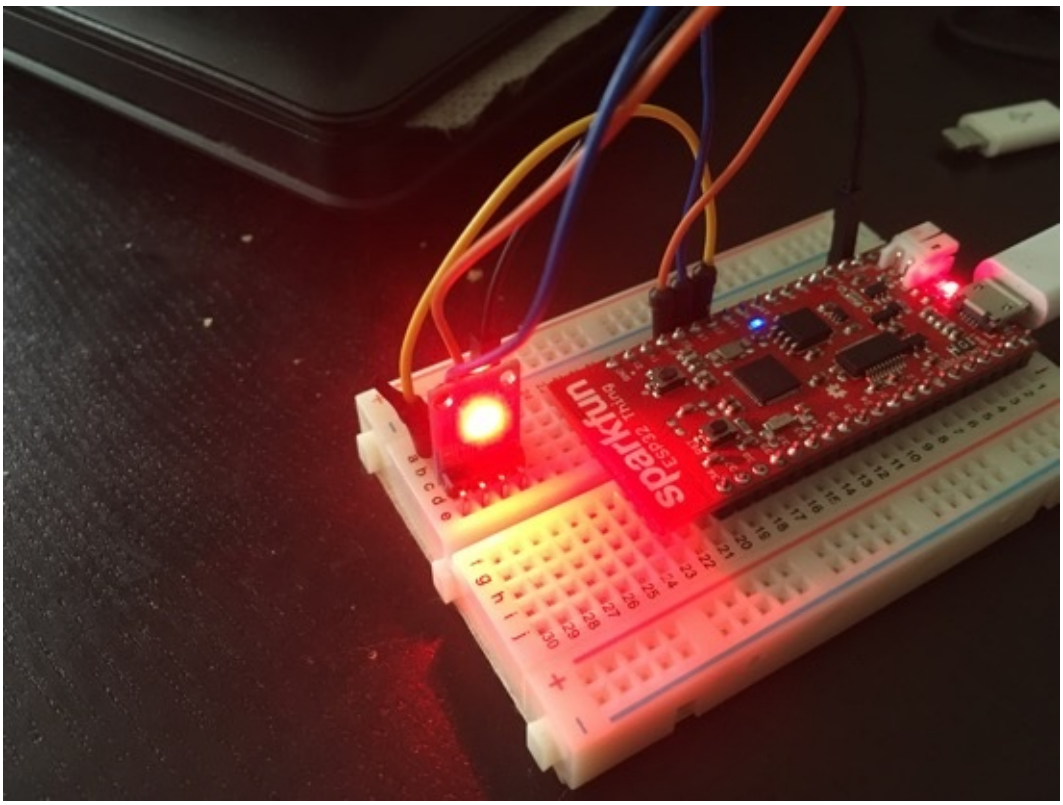
Save this program.

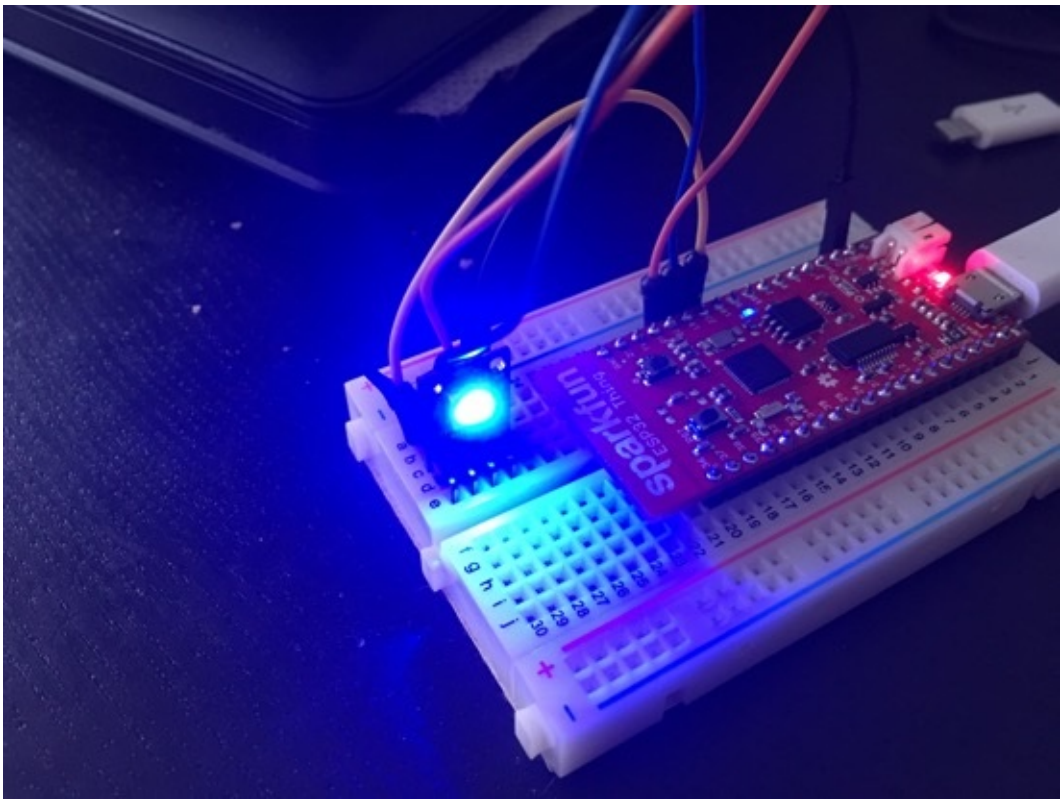
This program will generate six colors: red, green, blue, yellow, purple, and aqua.

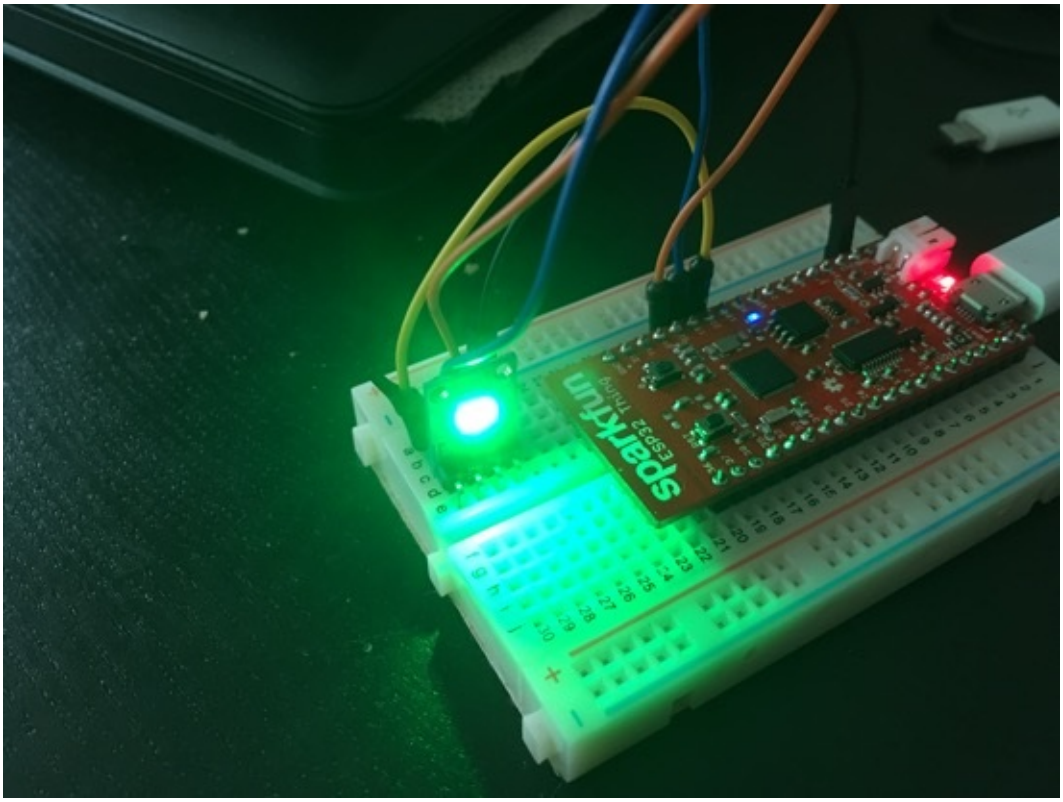
6.2.3 Testing

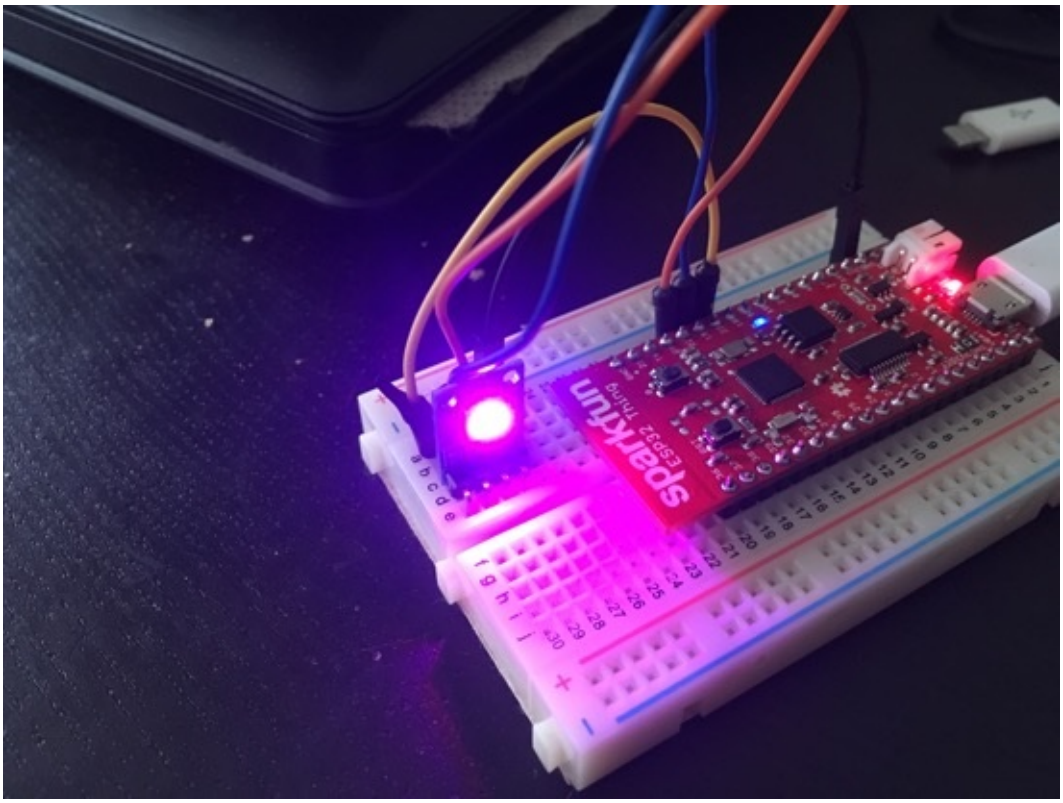
Upload and run the program. You should see several color on RGB LED.

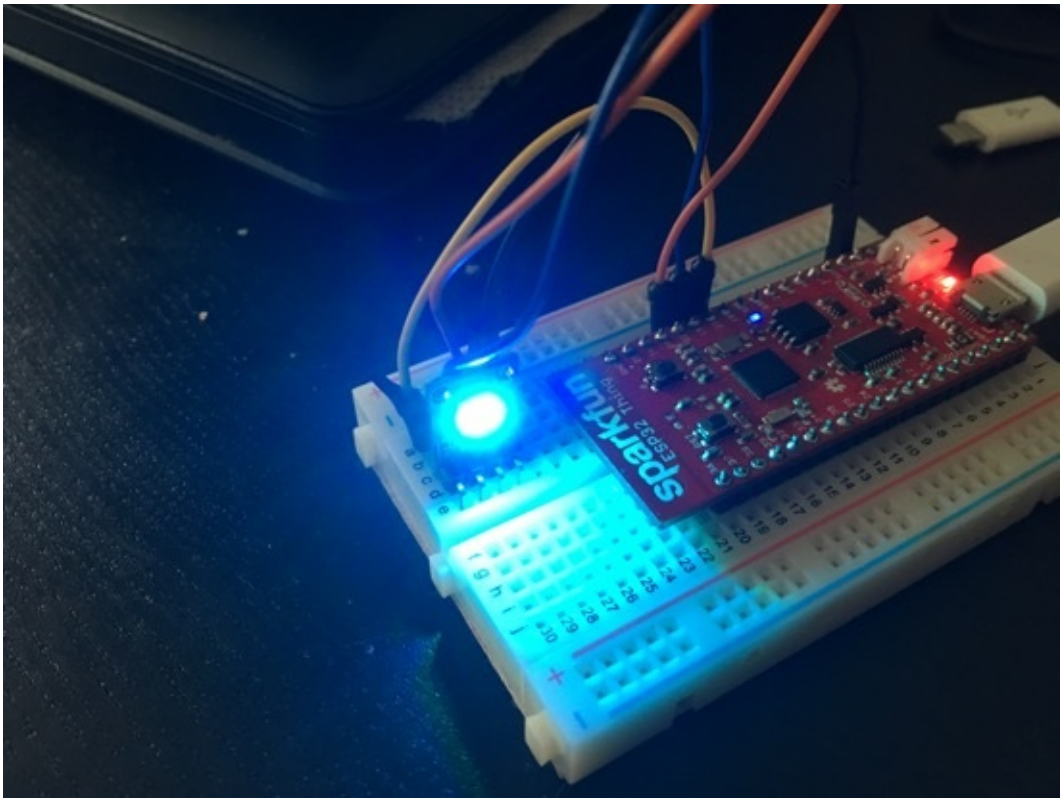
The following is a sample demo on RGB LED.




SparkFun
ESP32 Thing

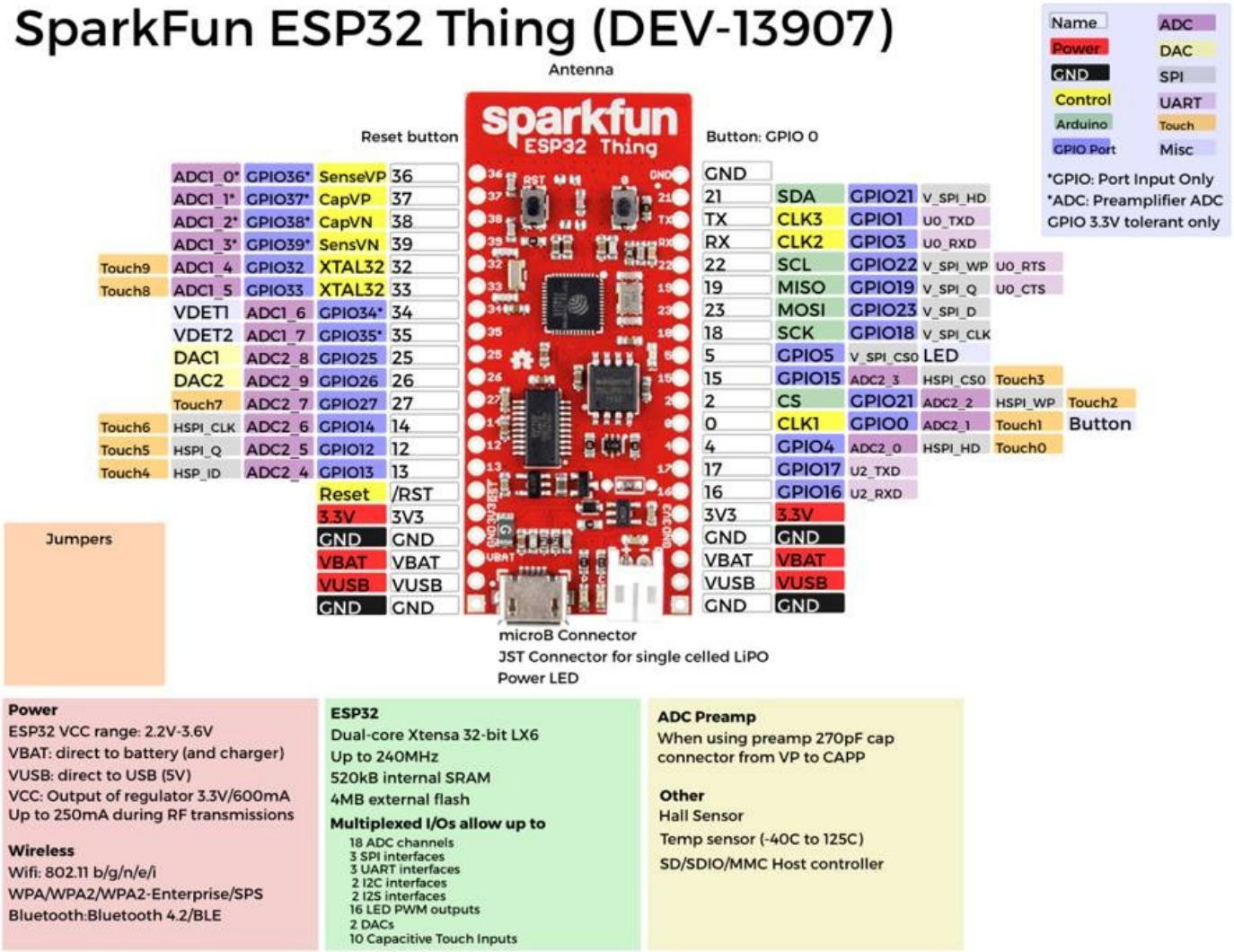

SparkFun
ESP32 Thing


SparkFun
ESP32 Thing


SparkFun
ESP32 Thing

6.3 Demo Analog Input: Working with Potentiometer

In this section, we learn how to read analog input on SparkFun ESP32 Thing board. For illustration, I use Potentiometer as analog input source. Our scenario is to read analog value from Potentiometer. Then, display it on Serial tool.



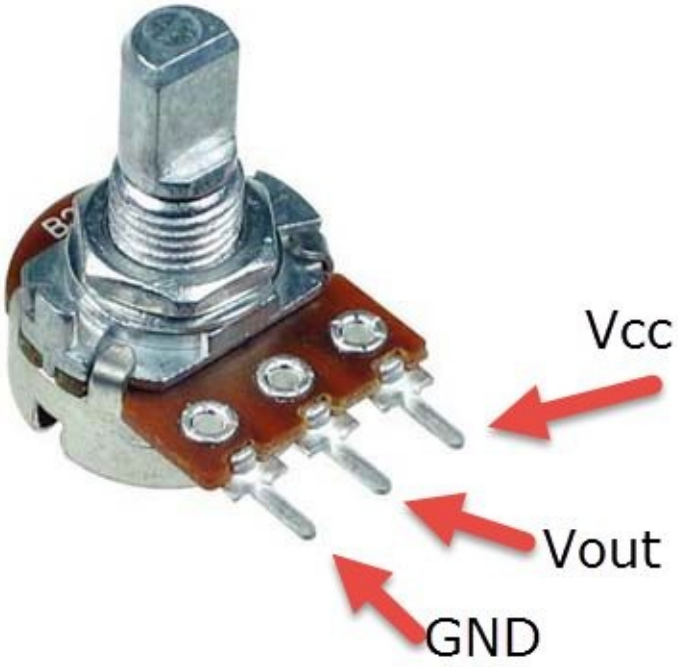
(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

SparkFun ESP32 Thing has several ADC pins that you can see it above. If you want to work with many analog input, you must expand it using ICs based ADC. In this section, we are working on SparkFun ESP32 Thing ADC on GPIO36.

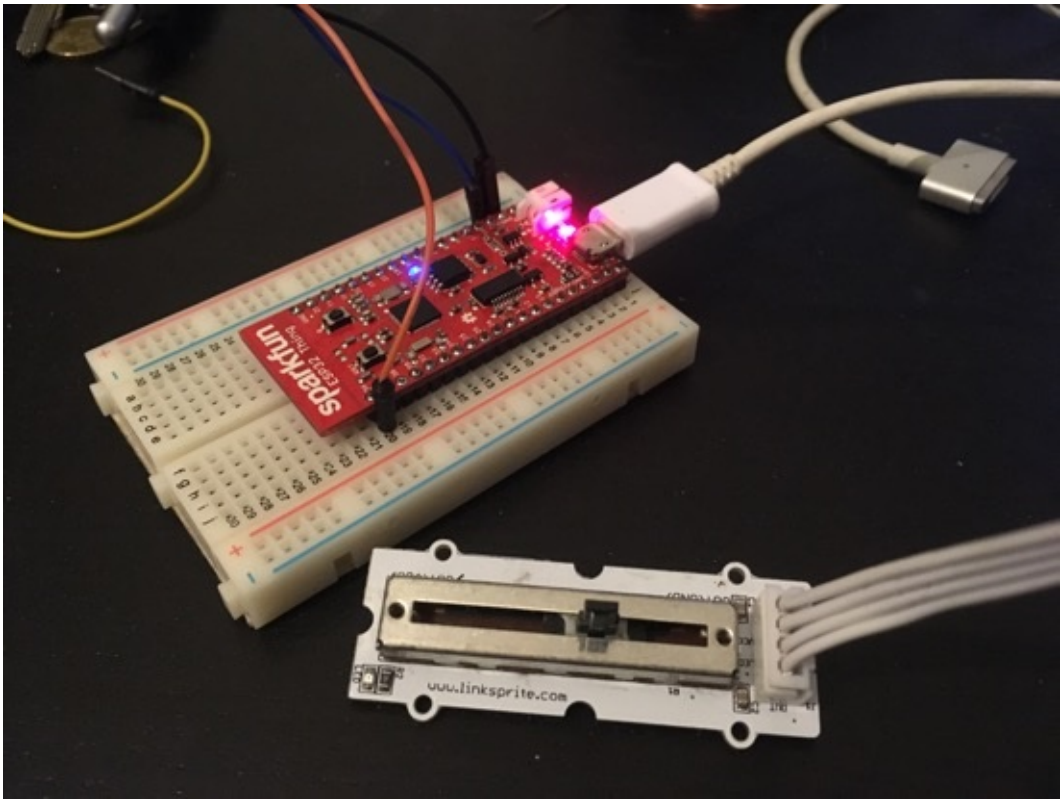
Let’s start!.

6.3.1 Wiring

To understand Potentiometer, you see its scheme in Figure below.

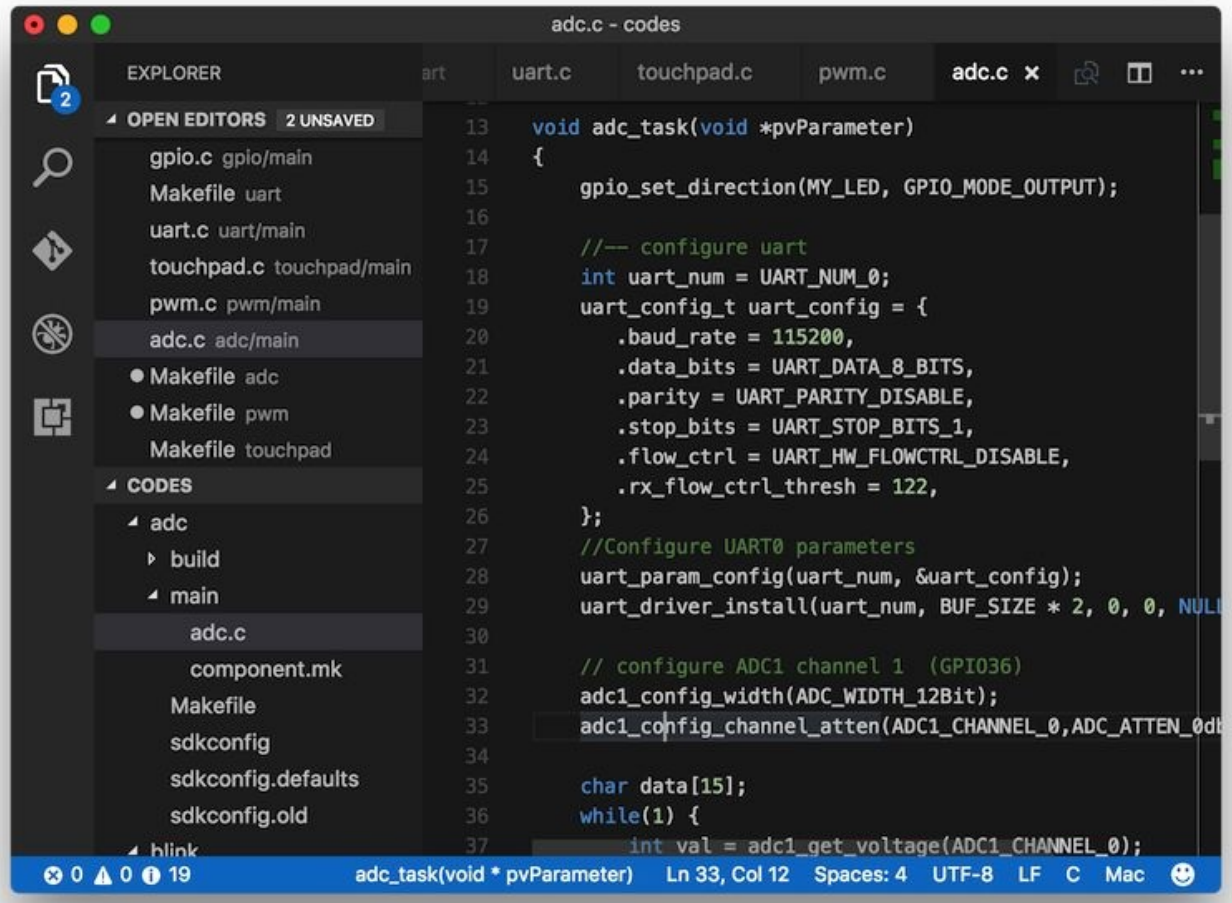


You can connect VCC to SparkFun ESP32 Thing board on 3V3 pin (VCC +3.3V). Vout to SparkFun ESP32 Thing board Analog input ADC1_0 (GPIO36). In addition, GND to SparkFun ESP32 Thing board GND. The following is hardware implementation. I use slide potentiometer.



6.3.2 Writing Program

You can start to create project structure as follows.



Firstly, modify adc.c file. Write the following codes.

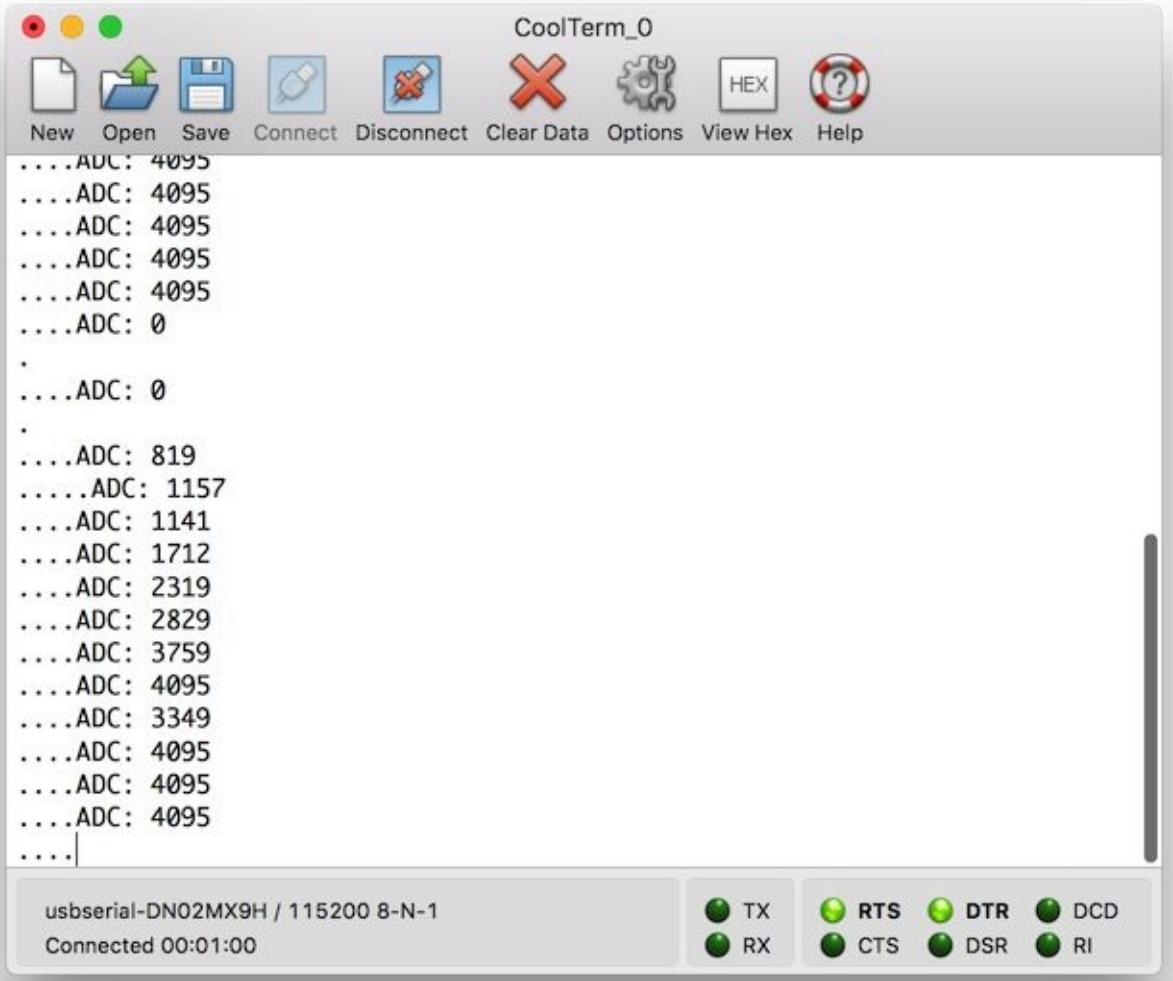
```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "sdkconfig.h"
#include "soc/uart_struct.h"
#include "driver/adc.h"

#define MY_LED 5
#define BUF_SIZE (1024)

void adc_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);

    //-- configure uart
    int uart_num = UART_NUM_0;
    uart_config_t uart_config = {
       .baud_rate = 115200,
       .data_bits = UART_DATA_8_BITS,
```

```
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .rx_flow_ctrl_thresh = 122,
    };
    //Configure UART0 parameters
    uart_param_config(uart_num, &uart_config);
    uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0);

    // configure ADC1 channel 1  (GPIO36)
    adc1_config_width(ADC_WIDTH_12Bit);
    adc1_config_channel_atten(ADC1_CHANNEL_0,ADC_ATTEN_0db);

    char data[15];
    while(1) {
        int val = adc1_get_voltage(ADC1_CHANNEL_0);
        sprintf(data,"ADC: %d\r\n", val);

        //Write data back to UART
        gpio_set_level(MY_LED, 1);
        uart_write_bytes(uart_num, (const char*) data, sizeof(data));
        gpio_set_level(MY_LED, 0);

        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
}

void app_main()
{
    xTaskCreate(&adc_task, "adc_task", 1024*3, NULL, 10, NULL);
}
```

Save all files.

6.3.3 Testing

Upload and run this program. If success, you can see analog value using Serial tool. In this case, I use CoolTerm. The following is program output:


CoolTerm_0
New
Open
Save
Connect
Disconnect
Clear Data
Options
HEX
View Hex
Help
....ADC: 4095
....ADC: 4095
....ADC: 4095
....ADC: 4095
....ADC: 4095
....ADC: 0
.
....ADC: 0
.
....ADC: 819
.....ADC: 1157
....ADC: 1141
....ADC: 1712
....ADC: 2319
....ADC: 2829
....ADC: 3759
....ADC: 4095
....ADC: 3349
....ADC: 4095
....ADC: 4095
....ADC: 4095
....
usbserial-DN02MX9H / 115200 8-N-1
Connected 00:01:00
TX
RX
RTS
CTS
DTR
DSR
DCD
RI

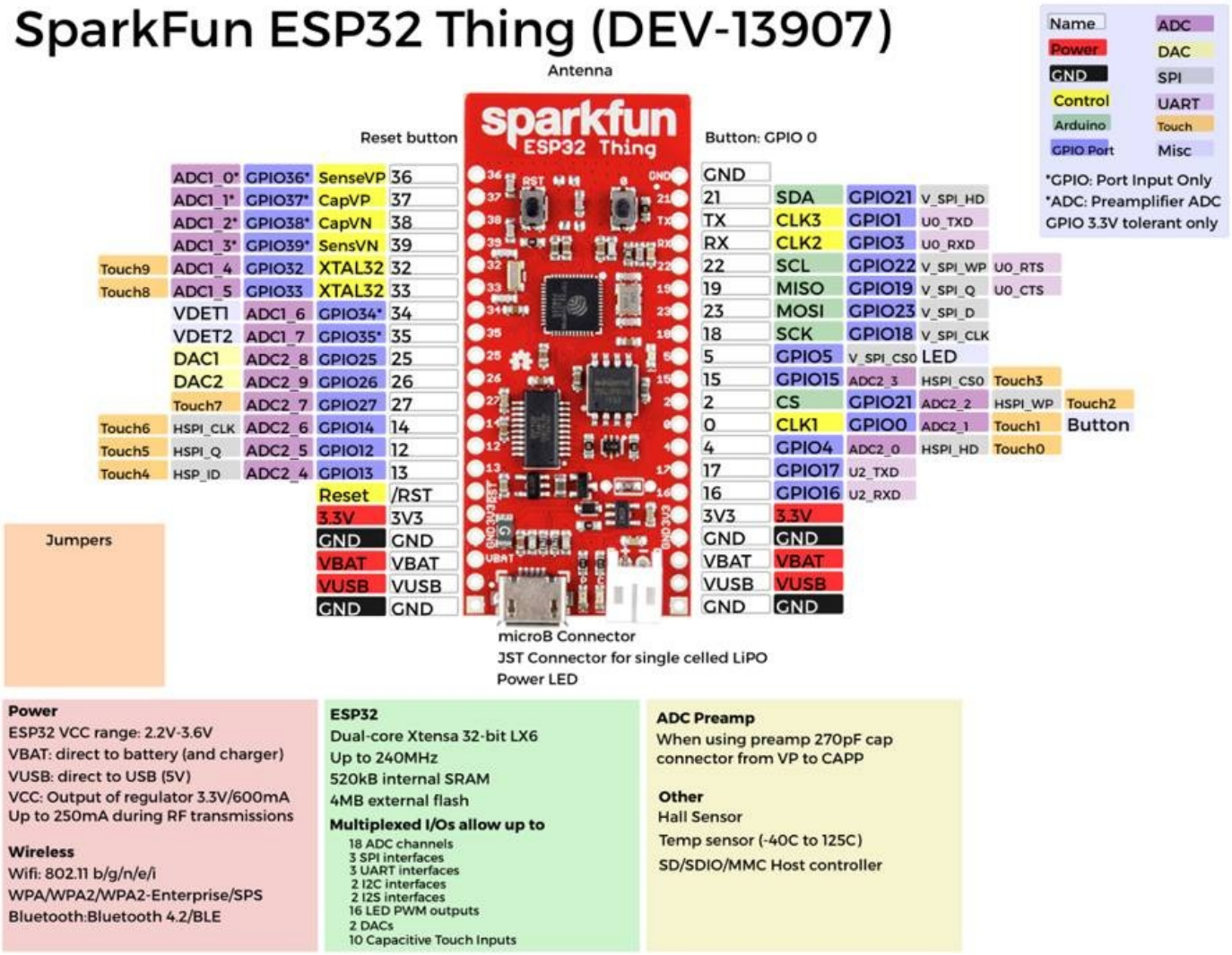
7. Working with I2C

In this chapter we learn how to work with I2C on SparkFun ESP32 Thing board using Arduino program.

7.1 Getting Started

The I2C (Inter-Integrated Circuit) bus was designed by Philips in the early ’80s to allow easy communication between components which reside on the same circuit board. TWI stands for Two Wire Interface and for most marts this bus is identical to I²C. The name TWI was introduced by Atmel and other companies to avoid conflicts with trademark issues related to I²C.

I2C bus consists of two wires, SDA (Serial Data Line) and SCL (Serial Clock Line). You can see I2C pins on SparkFun ESP32 Thing board, shown in Figure below.



(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

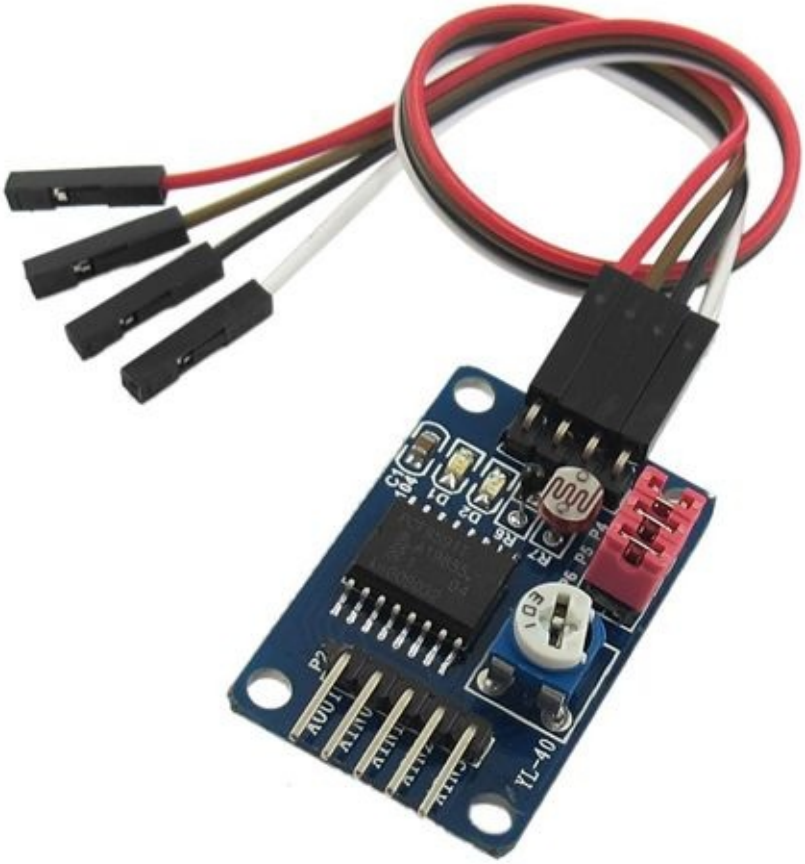
For testing, I used PCF8591 AD/DA Converter module with sensor and actuator devices. You can find it on the following online store:

- Amazon, http://www.amazon.com/PCF8591-Converter-Module-Digital-Conversion/dp/B00BXX4UWC/
- eBay, http://www.ebay.com
- Dealextreme, http://www.dx.com/p/pcf8591-ad-da-analog-to-digital-digital-to-

analog-converter-module-w-dupont-cable-deep-blue-336384

- Aliexpress, http://www.aliexpress.com/

In addition, you can find this device on your local electronics store/online store.



This module has mini form model too, for instance, you can find it on Amazon, http://www.amazon.com/WaveShare-PCF8591T-Converter-Evaluation-Development/dp/B00KM6X2OI/ .

This module use PCF8591 IC and you can read the datasheet on the following URLs.

- http://www.electrodragon.com/w/images/e/ed/PCF8591.pdf
- http://www.nxp.com/documents/data_sheet/PCF8591.pdf

In this chapter, we build a program to access sensor via I2C using Espressif IoT Development Framework on SparkFun ESP32 Thing board.

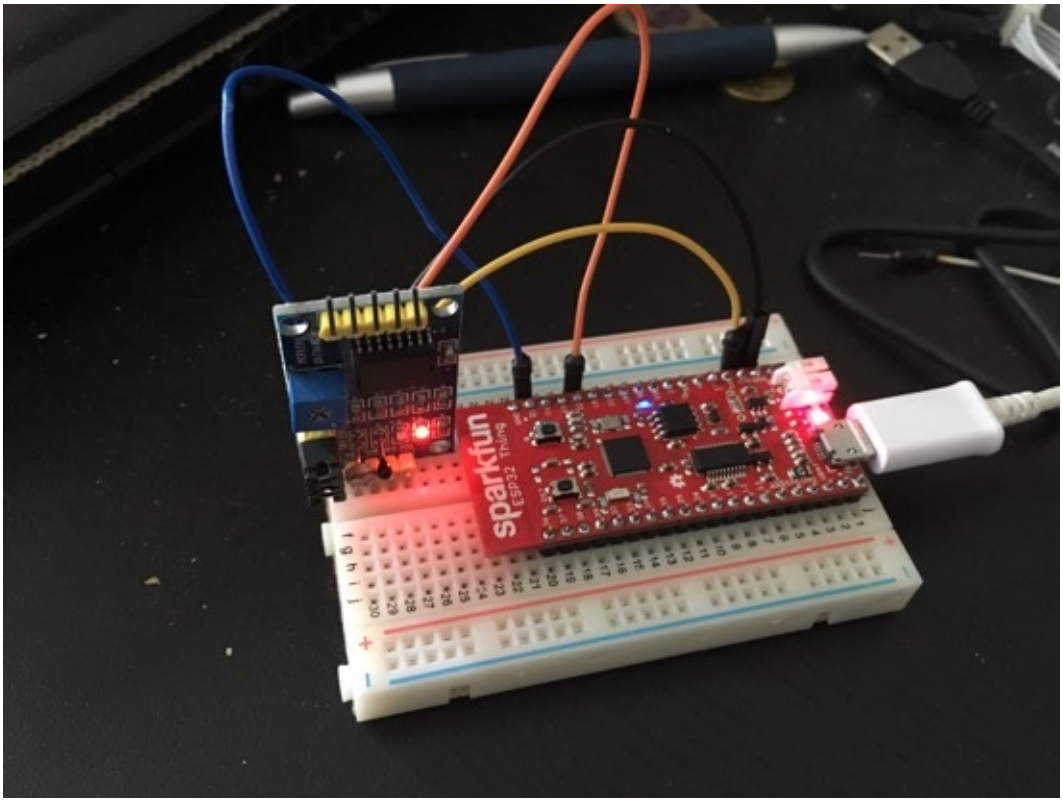
7.2 Writing Program

We use PCF8591 AD/DA Converter as I2C source. You can connect PCF8591 AD/DA Converter to SparkFun ESP32 Thing board directly.

The following is our wiring lab:

- PCF8591 AD/DA Converter SDA —> SparkFun ESP32 Thing SDA (GPIO21)
- PCF8591 AD/DA Converter SCL —> SparkFun ESP32 Thing CLK (GPIO22)
- PCF8591 AD/DA Converter VCC —> SparkFun ESP32 Thing VCC 3V3
- PCF8591 AD/DA Converter GND —> SparkFun ESP32 Thing GND

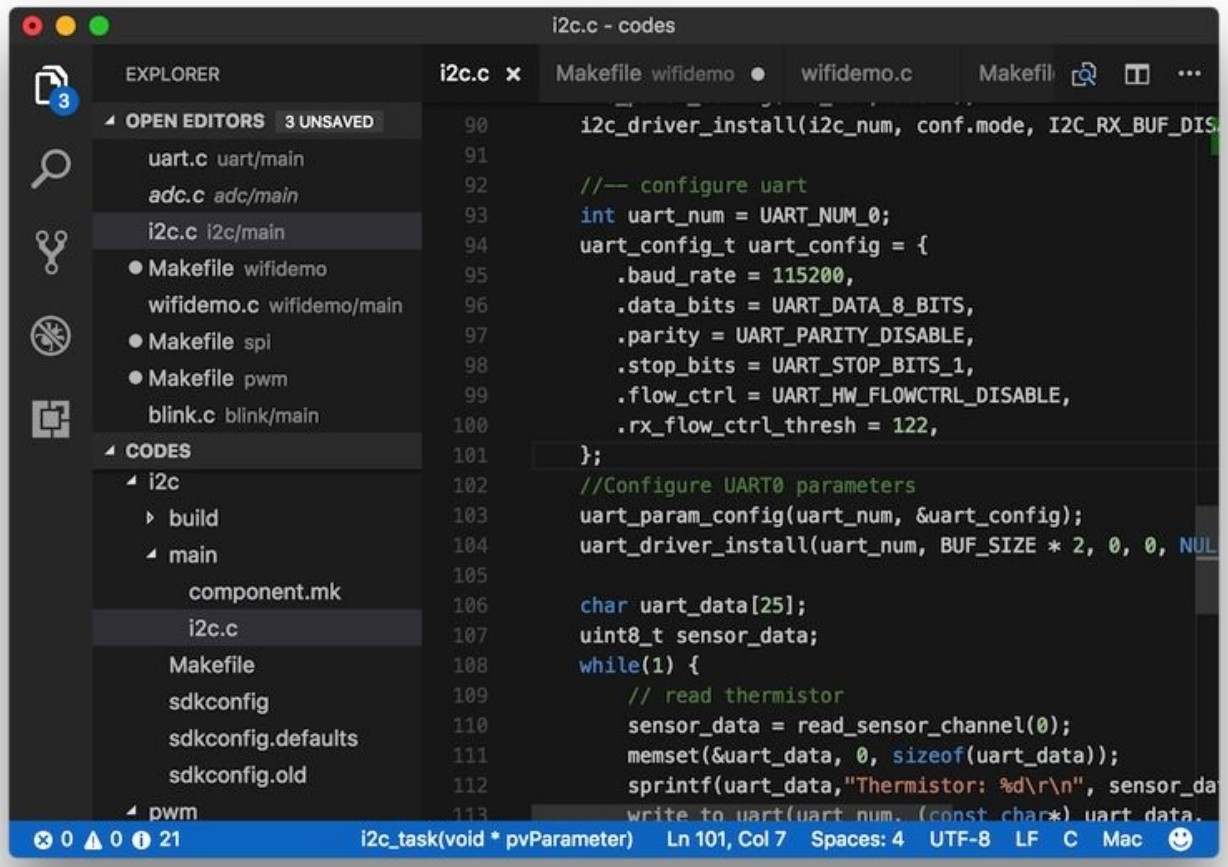
Hardware implementation can be shown in Figure below.



7.3 Writing Program

We use I2C on SparkFun ESP32 Thing board using Wire library like Arduino way. PCF8591 AD/DA Converter module has three sensor devices: Thermistor, Photo-voltaic cell and Potentiometer. This module runs on I2C bus with address 0x90. In this case, we read all sensor data.

The following is project structure.



In i2c.c file, you can write the following complete codes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "driver/i2c.h"
#include "sdkconfig.h"
#include "soc/uart_struct.h"
```

```c
#define MY_LED 5
#define BUF_SIZE (1024)
#define DATA_LENGTH 2

#define I2C_SCL_IO    22    /*!< gpio number for I2C master clock */
#define I2C_SDA_IO    21    /*!< gpio number for I2C master data  */
#define I2C_NUM I2C_NUM_0   /*!< I2C port number for master dev */
#define I2C_TX_BUF_LEN  (2*DATA_LENGTH) /*!<I2C slave tx buffer size */
#define I2C_RX_BUF_LEN  (2*DATA_LENGTH) /*!<I2C slave rx buffer size */
#define I2C_TX_BUF_DISABLE   0   /*!< I2C master do not need buffer */
#define I2C_RX_BUF_DISABLE   0   /*!< I2C master do not need buffer */
#define I2C_MASTER_FREQ_HZ    100000     /*!< I2C master clock frequency

#define WRITE_BIT  I2C_MASTER_WRITE /*!< I2C master write */
#define READ_BIT   I2C_MASTER_READ  /*!< I2C master read */
#define ACK_CHECK_EN   0x1     /*!< I2C master will check ack from slave
#define ACK_CHECK_DIS  0x0     /*!< I2C master will not check ack from s
#define ACK_VAL    0x0         /*!< I2C ack value */
#define NACK_VAL   0x1         /*!< I2C nack value */

#define PCF8591 (0x90 >> 1) // I2C bus address
#define PCF8591_ADC_CH0 0x00 // thermistor
#define PCF8591_ADC_CH1 0x01 // photo-voltaic cell
#define PCF8591_ADC_CH2 0x02
#define PCF8591_ADC_CH3 0x03 // potentiometer

void write_to_uart(int uart_num,const char * data,size_t size)
{
    gpio_set_level(MY_LED, 1);
    uart_write_bytes(uart_num,data, size);
    gpio_set_level(MY_LED, 0);
}

uint8_t read_sensor_channel(int channel){
    int i2c_num = I2C_NUM;
    uint8_t sensor_data_h, sensor_data_l;

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, PCF8591 << 1 | WRITE_BIT, ACK_CHECK_EN);

    if(channel==0)
        i2c_master_write_byte(cmd, PCF8591_ADC_CH0, ACK_CHECK_EN);
    if(channel==1)
        i2c_master_write_byte(cmd, PCF8591_ADC_CH1, ACK_CHECK_EN);
    if(channel==3)
        i2c_master_write_byte(cmd, PCF8591_ADC_CH3, ACK_CHECK_EN);
    i2c_master_stop(cmd);
    i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);

    cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);
```

```
    i2c_master_write_byte(cmd, PCF8591 << 1| READ_BIT, ACK_CHECK_EN);
    i2c_master_read_byte(cmd, &sensor_data_h, ACK_VAL);
    i2c_master_read_byte(cmd, &sensor_data_l, NACK_VAL);
    i2c_master_stop(cmd);
    i2c_master_cmd_begin(i2c_num, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);

    return sensor_data_l;
}

void i2c_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);

    // ini I2C master
    int i2c_num = I2C_NUM;
    i2c_config_t conf;
    conf.mode = I2C_MODE_MASTER;
    conf.sda_io_num = I2C_SDA_IO;
    conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
    conf.scl_io_num = I2C_SCL_IO;
    conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
    conf.master.clk_speed = I2C_MASTER_FREQ_HZ;
    i2c_param_config(i2c_num, &conf);
    i2c_driver_install(i2c_num, conf.mode, I2C_RX_BUF_DISABLE, I2C_TX_BU

    //-- configure uart
    int uart_num = UART_NUM_0;
    uart_config_t uart_config = {
       .baud_rate = 115200,
       .data_bits = UART_DATA_8_BITS,
       .parity = UART_PARITY_DISABLE,
       .stop_bits = UART_STOP_BITS_1,
       .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
       .rx_flow_ctrl_thresh = 122,
    };
    //Configure UART0 parameters
    uart_param_config(uart_num, &uart_config);
    uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0);

    char uart_data[25];
    uint8_t sensor_data;
    while(1) {
        // read thermistor
        sensor_data = read_sensor_channel(0);
        memset(&uart_data, 0, sizeof(uart_data));
        sprintf(uart_data,"Thermistor: %d\r\n", sensor_data);
        write_to_uart(uart_num, (const char*) uart_data, sizeof(uart_dat

        // read photo-voltaic
        sensor_data = read_sensor_channel(1);
        memset(&uart_data, 0, sizeof(uart_data));
        sprintf(uart_data,"Photo-voltaic: %d\r\n", sensor_data);
```

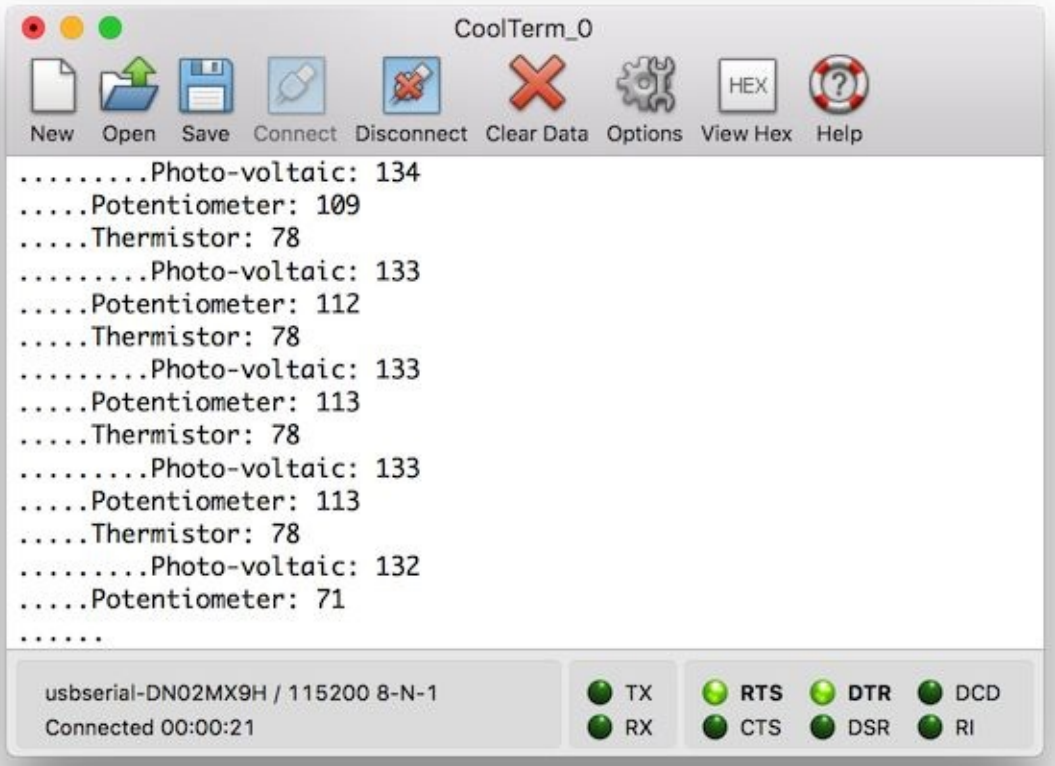
```
        write_to_uart(uart_num, (const char*) uart_data, sizeof(uart_dat

        // read potentiometer
        sensor_data = read_sensor_channel(3);
        memset(&uart_data, 0, sizeof(uart_data));
        sprintf(uart_data,"Potentiometer: %d\r\n", sensor_data);
        write_to_uart(uart_num, (const char*) uart_data, sizeof(uart_dat

        vTaskDelay(3000 / portTICK_PERIOD_MS);
    }
}

void app_main()
{
    xTaskCreate(&i2c_task, "i2c_task", 1024*4, NULL, 10, NULL);
}
```

Save all program.

7.4 Testing

Now you can upload and run the program to SparkFun ESP32 Thing board board.

If success, open Serial tool with baudrate 115200 and connect to SparkFun ESP32 Thing to see the program output. The following is a sample output.



8. Working With SPI

In this chapter I’m going to explain how to work with SPI on SparkFun ESP32 Thing board.

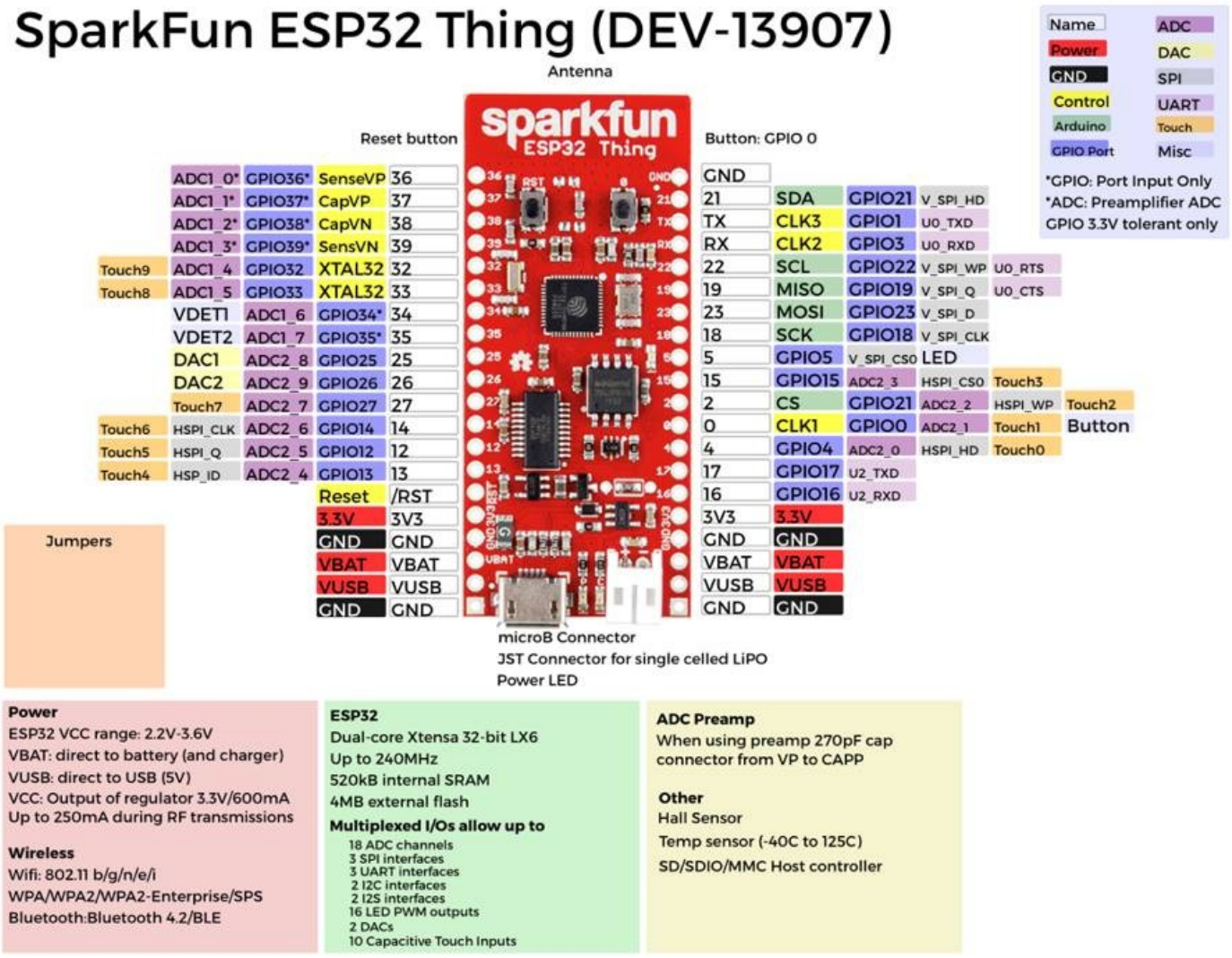
8.1 Getting Started

The Serial Peripheral Interface (SPI) is a communication bus that is used to interface one or more slave peripheral integrated circuits (ICs) to a single master SPI device; usually a microcontroller or microprocessor of some sort.

SPI in SparkFun ESP32 Thing board can be defined on the following pins:

- MOSI on pin 13
- MISO on pin 12
- SCLK on pin 14 (SCL)

You can see these SPI pins on SparkFun ESP32 Thing board, shown in Figure below.



(source: https://cdn.sparkfun.com/assets/learn_tutorials/5/0/7/esp32-thing-graphical-datasheet-v02.png)

We can use SPI on SparkFun ESP32 Thing board with SPI master and slave modes. We develop program based SPI using SPI library, http://esp-

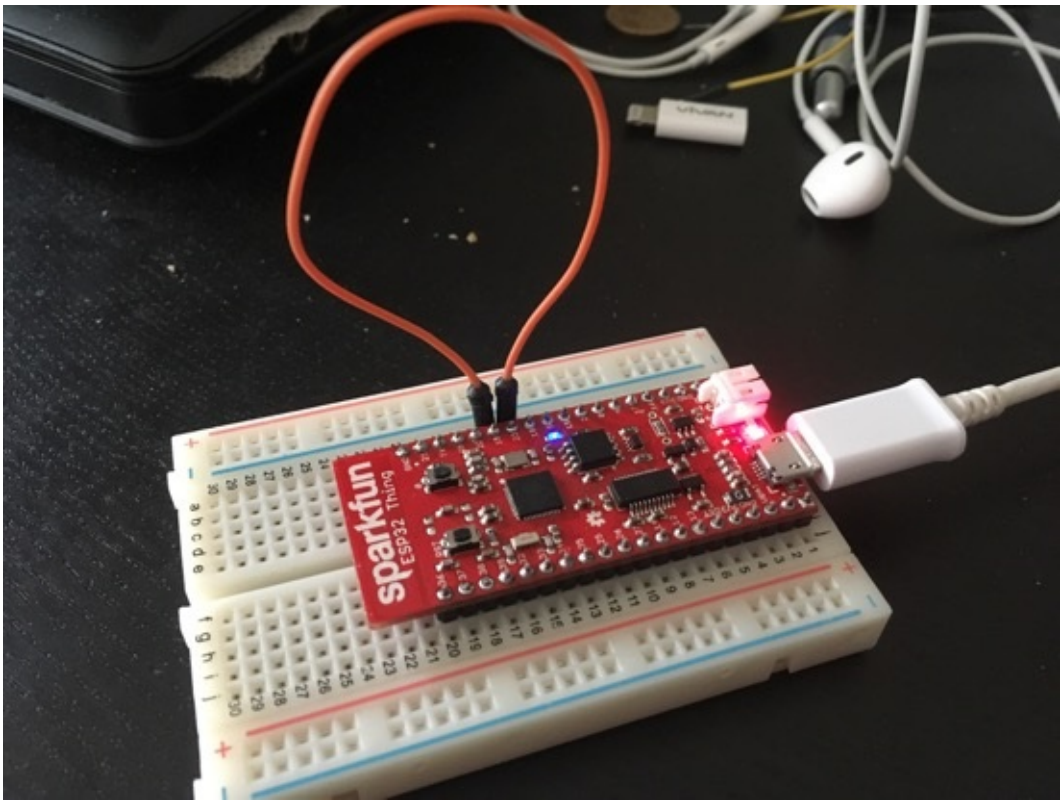
idf.readthedocs.io/en/latest/api/peripherals/spi_master.html.

In this chapter, we build a SPI Loopback app. Let’s start!.

8.2 Wiring

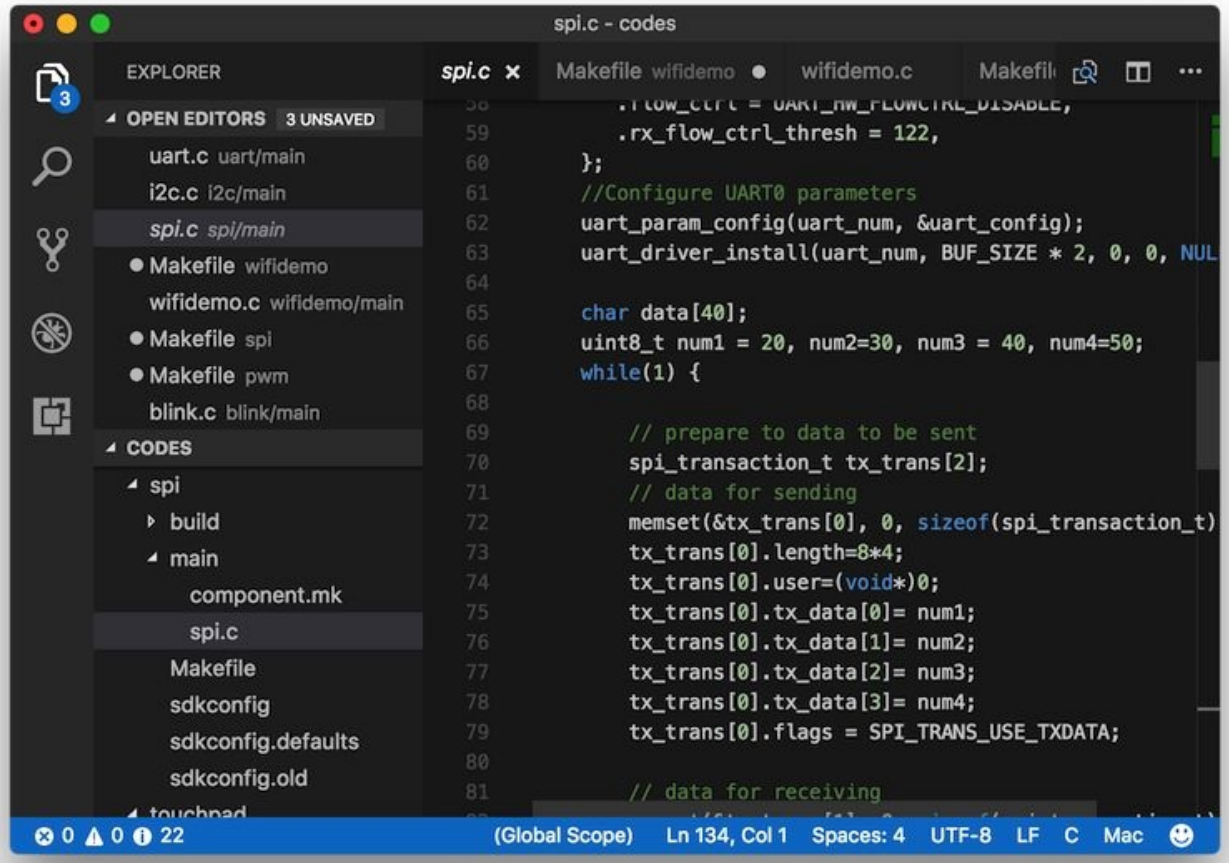
To develop SPI loopback, we can connect MOSI pin to MISO pin. This means you connect pin GPIO19 to pin GPIO23 using cable.

The following is a sample of wiring.



8.3 Writing a Program

Now you can create your project. The following is my project structure.



Firstly, we write a program for SparkFun ESP32 Thing on spi.c file. We send 4 bytes to SPI and the listen incoming message. The following is complete code.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "sdkconfig.h"
#include "esp_system.h"
#include "soc/uart_struct.h"
#include "driver/spi_master.h"

#define MY_LED 5
#define BUF_SIZE (1024)
```

```
#define PIN_NUM_MISO 19
#define PIN_NUM_MOSI 23
#define PIN_NUM_CLK  18
#define PIN_NUM_CS   21

//void spi_task(void)
void spi_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);

    // configure spi
    //esp_err_t ret;
    spi_device_handle_t spi;
    spi_bus_config_t buscfg={
        .miso_io_num=PIN_NUM_MISO,
        .mosi_io_num=PIN_NUM_MOSI,
        .sclk_io_num=PIN_NUM_CLK,
        .quadwp_io_num=-1,
        .quadhd_io_num=-1
    };
    spi_device_interface_config_t devcfg={
        .clock_speed_hz=10000000,               //Clock out at 10 MHz
        .mode=0,                                //SPI mode 0
        .spics_io_num=PIN_NUM_CS,               //CS pin
        .queue_size=7,                          //We want to be able to
 //       .pre_cb=ili_spi_pre_transfer_callback,  //Specify pre-transfer
    };
    //Initialize the SPI bus
    spi_bus_initialize(HSPI_HOST, &buscfg, 1);
    //assert(ret==ESP_OK);
    //Attach the SPI bus
    spi_bus_add_device(HSPI_HOST, &devcfg, &spi);
    //assert(ret==ESP_OK);

    //-- configure uart
    int uart_num = UART_NUM_0;
    uart_config_t uart_config = {
       .baud_rate = 115200,
       .data_bits = UART_DATA_8_BITS,
       .parity = UART_PARITY_DISABLE,
       .stop_bits = UART_STOP_BITS_1,
       .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
       .rx_flow_ctrl_thresh = 122,
    };
    //Configure UART0 parameters
    uart_param_config(uart_num, &uart_config);
    uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0);

    char data[40];
    uint8_t num1 = 20, num2=30, num3 = 40, num4=50;
    while(1) {

        // prepare to data to be sent
```

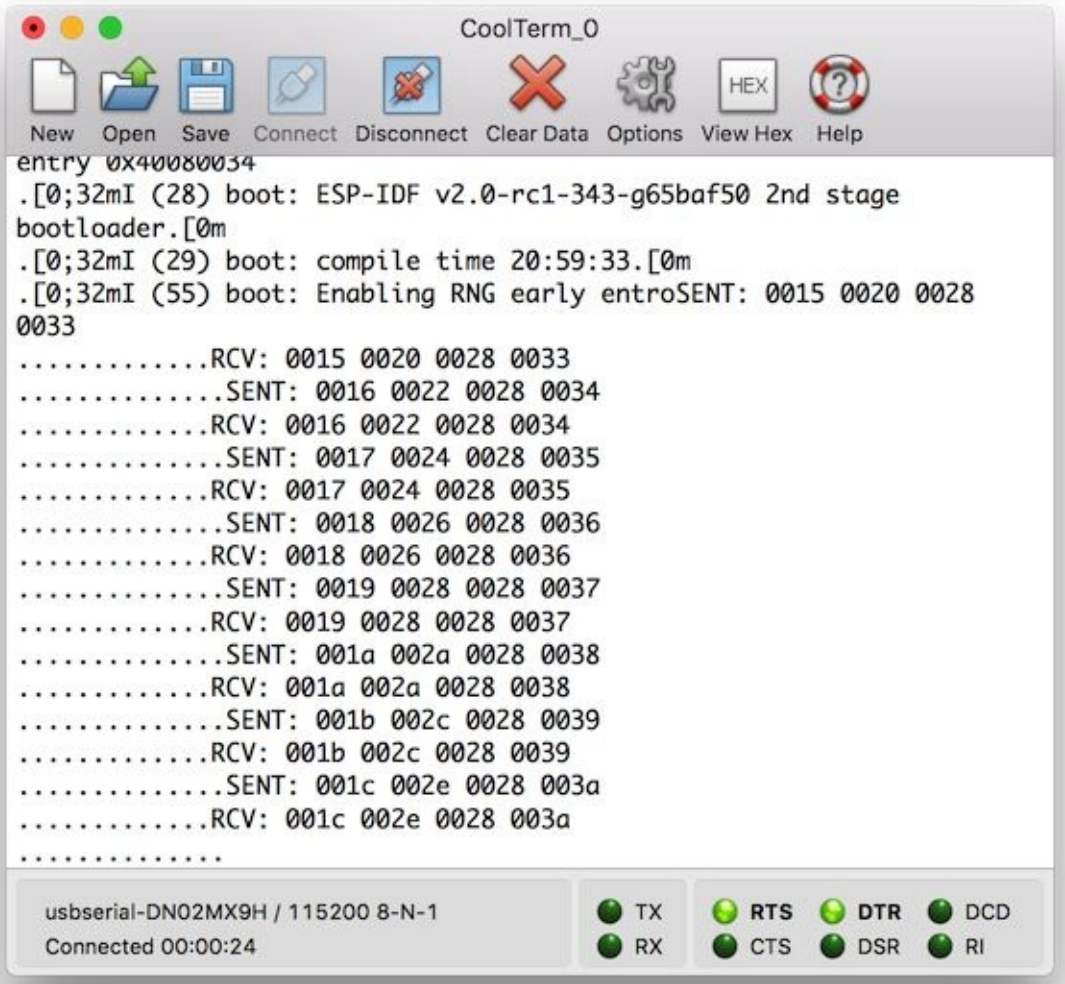
```
        spi_transaction_t tx_trans[2];
        // data for sending
        memset(&tx_trans[0], 0, sizeof(spi_transaction_t));
        tx_trans[0].length=8*4;
        tx_trans[0].user=(void*)0;
        tx_trans[0].tx_data[0]= num1;
        tx_trans[0].tx_data[1]= num2;
        tx_trans[0].tx_data[2]= num3;
        tx_trans[0].tx_data[3]= num4;
        tx_trans[0].flags = SPI_TRANS_USE_TXDATA;

        // data for receiving
        memset(&tx_trans[1], 0, sizeof(spi_transaction_t));
        tx_trans[1].length=8*4;
        tx_trans[1].user=(void*)0;
        tx_trans[0].rx_data[0]= 0;
        tx_trans[0].rx_data[1]= 0;
        tx_trans[0].rx_data[2]= 0;
        tx_trans[0].rx_data[3]= 0;
        tx_trans[1].flags = SPI_TRANS_USE_RXDATA;

        // send data
        spi_device_queue_trans(spi, &tx_trans[0], portMAX_DELAY);
        spi_device_queue_trans(spi, &tx_trans[1], portMAX_DELAY);
        //assert(ret==ESP_OK);

        // wait to receive data
        spi_transaction_t *rtrans;
        spi_device_get_trans_result(spi, &rtrans, portMAX_DELAY);
        spi_device_get_trans_result(spi, &rtrans, portMAX_DELAY);
        //assert(ret==ESP_OK);

        //Write data back to UART
        gpio_set_level(MY_LED, 1);
        memset(&data, 0, sizeof(data));
        sprintf(data,"SENT: %04x %04x %04x %04x\r\n", num1,num2,num3,num
        uart_write_bytes(uart_num, (const char*) data, sizeof(data));

        memset(&data, 0, sizeof(data));
        sprintf(data,"RCV: %04x %04x %04x %04x\r\n", rtrans->rx_data[0],
            rtrans->rx_data[1],rtrans->rx_data[2],rtrans->rx_data[3]);
        uart_write_bytes(uart_num, (const char*) data, sizeof(data));

        gpio_set_level(MY_LED, 0);
        vTaskDelay(3000 / portTICK_PERIOD_MS);

        num1++;num2++;
        num2++;num4++;
        if(num1>50)
            num1 = 20;
        if(num2>60)
            num2 = 30;
        if(num3>70)
```

```
            num1 = 40;
        if(num4>80)
            num1 = 50;
    }
}

void app_main()
{
    //spi_task();
    xTaskCreate(&spi_task, "spi_task", 1024*4, NULL, 10, NULL);
}
```

Save this code.

8.4 Testing

Now you can upload program to SparkFun ESP32 Thing board. If done, open Serial tool. You should see received data from SPI. The program output from CoolTerm tool.



9. Connecting to a Network

In this chapter I’m going to explain how to connect WiFi from SparkFun ESP32 Thing.

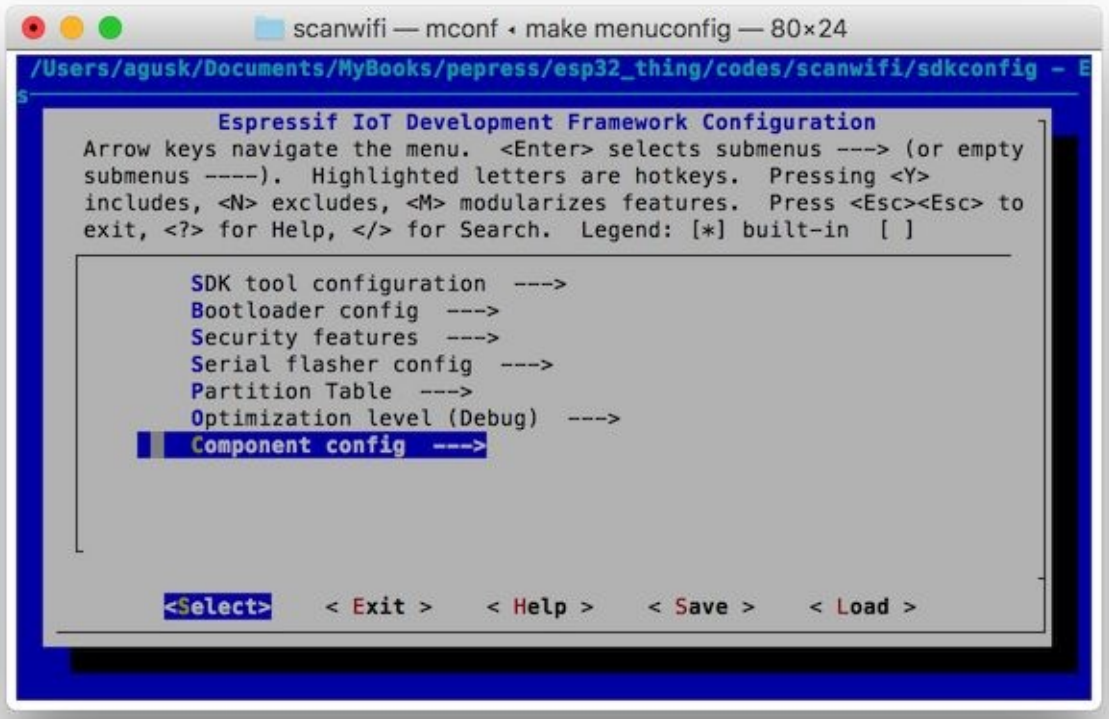
9.1 Getting Started

SparkFun ESP32 Thing board has built-in WiFi module so we can use it to connect existing WiFi or work as AP.

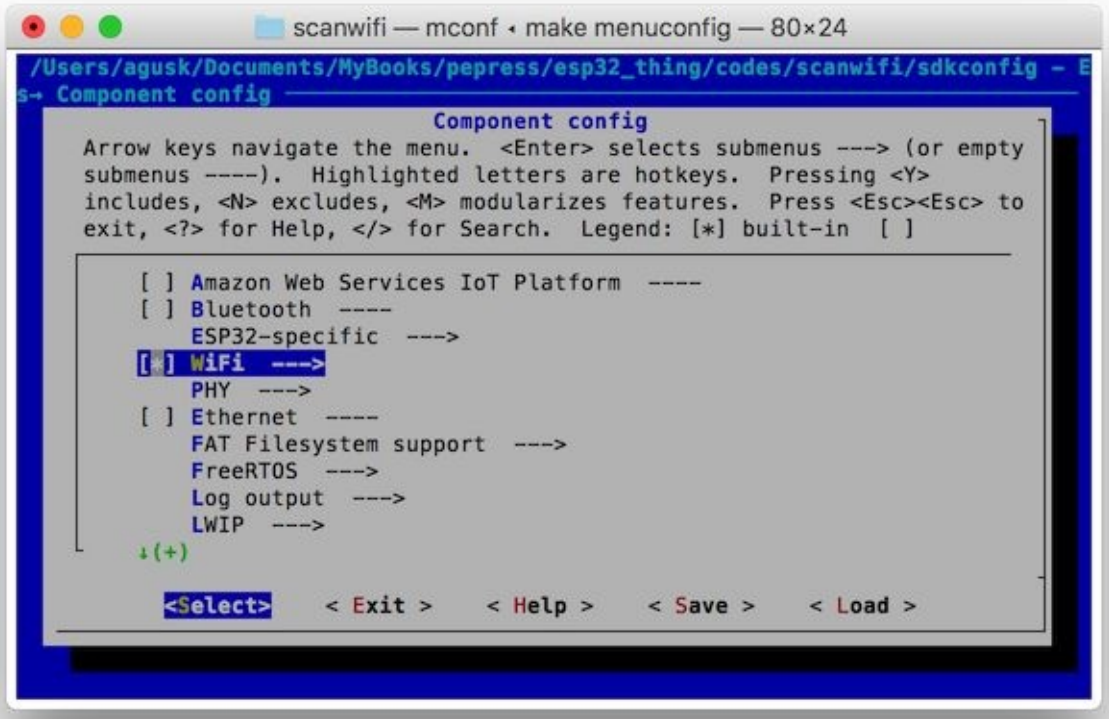
In this chapter, we learn how to connect existing WiFi and access a website.

9.2 Enabling WiFi Module

To work with WiFi module, we should enable it via menuconfig so our compiler will add our WiFi library. On menuconfig in your current project folder, select Component config.

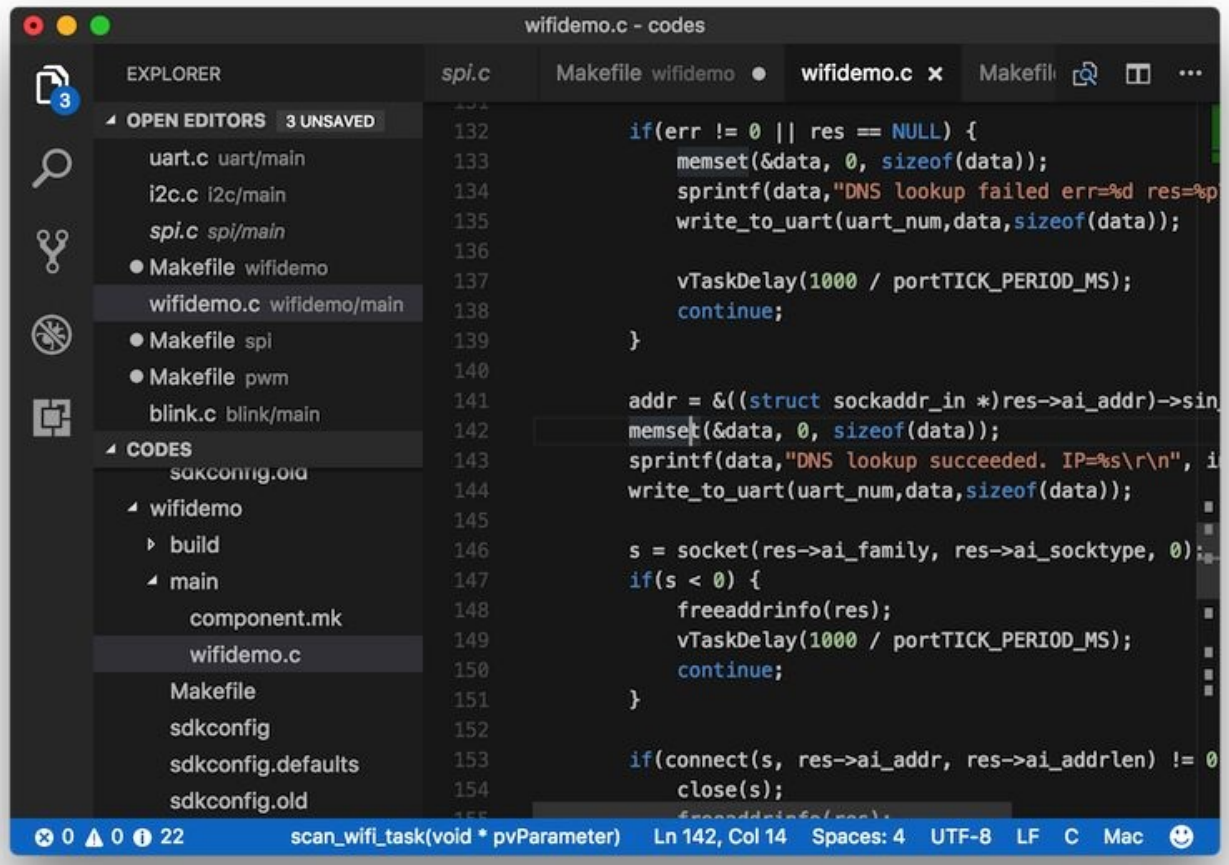


Then, select WiFi by pressing Space bar. Save all configurations.


scanwifi — mconf ◂ make menuconfig — 80×24
/Users/agusk/Documents/MyBooks/pepress/esp32_thing/codes/scanwifi/sdkconfig - E
s→ Component config
Component config
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
[ ] Amazon Web Services IoT Platform ----
[ ] Bluetooth ----
ESP32-specific --->
[*] WiFi --->
PHY --->
[ ] Ethernet ----
FAT Filesystem support --->
FreeRTOS --->
Log output --->
LWIP --->
↓(+)
<Select> < Exit > < Help > < Save > < Load >

9.3 Accessing Web Server

For testing, we build a simple app to connect existing WiFi and send request to a web server. The following is our project structure.



In wifidemo.c file, we write our program. You can write the following complete codes.

```
#include <string.h>
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "sdkconfig.h"
#include "soc/uart_struct.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "lwip/err.h"
#include "lwip/sockets.h"
#include "lwip/sys.h"
```

```
#include "lwip/netdb.h"
#include "lwip/dns.h"

#define MY_LED 5
#define BUF_SIZE (1024)

#define EXAMPLE_WIFI_SSID "WiFi-SSID"
#define EXAMPLE_WIFI_PASS "SSID-key"

/* FreeRTOS event group to signal when we are connected & ready to make
static EventGroupHandle_t wifi_event_group;

/* The event group allows multiple bits for each event,
   but we only care about one event - are we connected
   to the AP with an IP? */
const int CONNECTED_BIT = BIT0;

#define WEB_SERVER "aguskurniawan.net"
#define WEB_PORT 80
#define WEB_URL "http://www.aguskurniawan.net/"

static const char *REQUEST = "GET " WEB_URL " HTTP/1.1\n"
    "Host: "WEB_SERVER"\n"
    "User-Agent: esp-idf/1.0 esp32\n"
    "\n";


static esp_err_t event_handler(void *ctx, system_event_t *event)
{
    switch(event->event_id) {
    case SYSTEM_EVENT_STA_START:
        esp_wifi_connect();
        break;
    case SYSTEM_EVENT_STA_GOT_IP:
        xEventGroupSetBits(wifi_event_group, CONNECTED_BIT);
        break;
    case SYSTEM_EVENT_STA_DISCONNECTED:
        /* This is a workaround as ESP32 WiFi libs don't currently
           auto-reassociate. */
        esp_wifi_connect();
        xEventGroupClearBits(wifi_event_group, CONNECTED_BIT);
        break;
    default:
        break;
    }
    return ESP_OK;
}

static void initialise_wifi(void)
{
    tcpip_adapter_init();
    wifi_event_group = xEventGroupCreate();
    ESP_ERROR_CHECK( esp_event_loop_init(event_handler, NULL) );
```

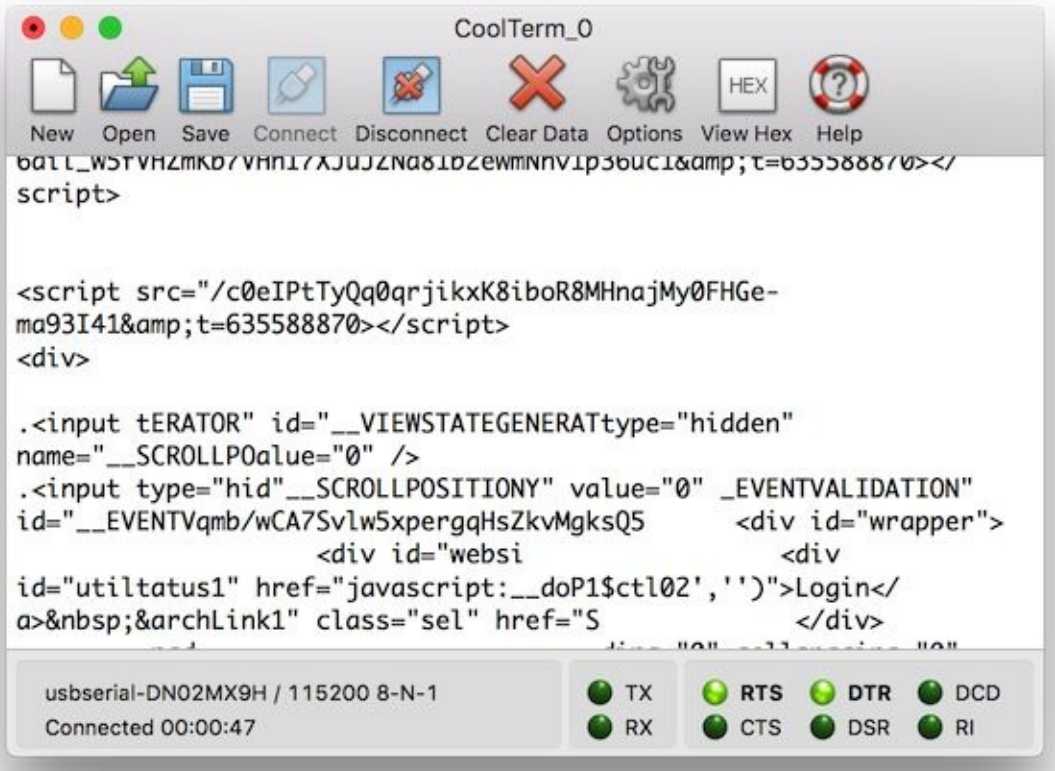
```
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK( esp_wifi_init(&cfg) );
    ESP_ERROR_CHECK( esp_wifi_set_storage(WIFI_STORAGE_RAM) );
    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_WIFI_SSID,
            .password = EXAMPLE_WIFI_PASS,
        },
    };
    ESP_ERROR_CHECK( esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK( esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config)
    ESP_ERROR_CHECK( esp_wifi_start() );
}

void write_to_uart(int uart_num,const char * data,size_t size)
{
    gpio_set_level(MY_LED, 1);
    uart_write_bytes(uart_num,data, size);
    gpio_set_level(MY_LED, 0);
}

void scan_wifi_task(void *pvParameter)
{
    gpio_set_direction(MY_LED, GPIO_MODE_OUTPUT);

    //-- configure uart
    int uart_num = UART_NUM_0;
    uart_config_t uart_config = {
       .baud_rate = 115200,
       .data_bits = UART_DATA_8_BITS,
       .parity = UART_PARITY_DISABLE,
       .stop_bits = UART_STOP_BITS_1,
       .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
       .rx_flow_ctrl_thresh = 122,
    };
    //Configure UART0 parameters
    uart_param_config(uart_num, &uart_config);
    uart_driver_install(uart_num, BUF_SIZE * 2, 0, 0, NULL, 0);

    const struct addrinfo hints = {
        .ai_family = AF_INET,
        .ai_socktype = SOCK_STREAM,
    };
    struct addrinfo *res;
    struct in_addr *addr;
    int s, r;
    char recv_buf[64];

    char data[30];

    while(1) {
        xEventGroupWaitBits(wifi_event_group, CONNECTED_BIT,
                            false, true, portMAX_DELAY);
```

```
        memset(&data, 0, sizeof(data));
        sprintf(data,"Connected to AP\r\n");
        write_to_uart(uart_num,data,sizeof(data));

        int err = getaddrinfo(WEB_SERVER, "80", &hints, &res);

        if(err != 0 || res == NULL) {
            memset(&data, 0, sizeof(data));
            sprintf(data,"DNS lookup failed err=%d res=%p\r\n", err, res
            write_to_uart(uart_num,data,sizeof(data));

            vTaskDelay(1000 / portTICK_PERIOD_MS);
            continue;
        }

        addr = &((struct sockaddr_in *)res->ai_addr)->sin_addr;
        memset(&data, 0, sizeof(data));
        sprintf(data,"DNS lookup succeeded. IP=%s\r\n", inet_ntoa(*addr)
        write_to_uart(uart_num,data,sizeof(data));

        s = socket(res->ai_family, res->ai_socktype, 0);
        if(s < 0) {
            freeaddrinfo(res);
            vTaskDelay(1000 / portTICK_PERIOD_MS);
            continue;
        }

        if(connect(s, res->ai_addr, res->ai_addrlen) != 0) {
            close(s);
            freeaddrinfo(res);
            vTaskDelay(4000 / portTICK_PERIOD_MS);
            continue;
        }
        memset(&data, 0, sizeof(data));
        sprintf(data,"..connected\r\n");
        write_to_uart(uart_num,data,sizeof(data));
        freeaddrinfo(res);

        if (write(s, REQUEST, strlen(REQUEST)) < 0) {
            close(s);
            vTaskDelay(4000 / portTICK_PERIOD_MS);
            continue;
        }

        /* Read HTTP response */
        memset(&data, 0, sizeof(data));
        sprintf(data,"HTTP Response:\r\n");
        write_to_uart(uart_num,data,sizeof(data));
        do {
            bzero(recv_buf, sizeof(recv_buf));
            r = read(s, recv_buf, sizeof(recv_buf)-1);
```

```
            memset(&data, 0, sizeof(data));
            sprintf(data,"%s",recv_buf);
            write_to_uart(uart_num,data,sizeof(data));
        } while(r > 0);

        memset(&data, 0, sizeof(data));
        sprintf(data,"..Done!!\r\n");
        write_to_uart(uart_num,data,sizeof(data));

        close(s);
        vTaskDelay(3000 / portTICK_PERIOD_MS);
    }
}

void app_main()
{
    nvs_flash_init();
    initialise_wifi();

    xTaskCreate(&scan_wifi_task, "scan_wifi_task", 1024*4, NULL, 10, NUL
}
```

Save all program. Please change your WiFi SSID and its key.

Compile and upload the program to SparkFun ESP32 Thing. Open serial tool to see the program output.



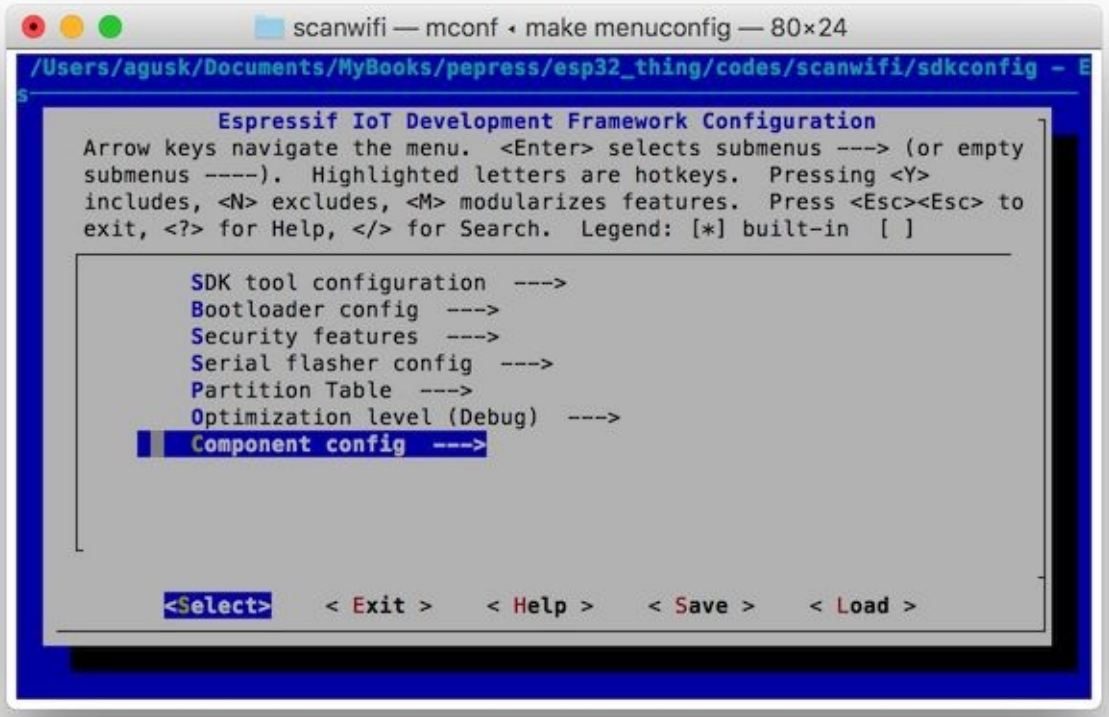
10. Bluetooth Programming

In this chapter I’m going to explain how to work with Bluetooth in SparkFun ESP32 Thing board.

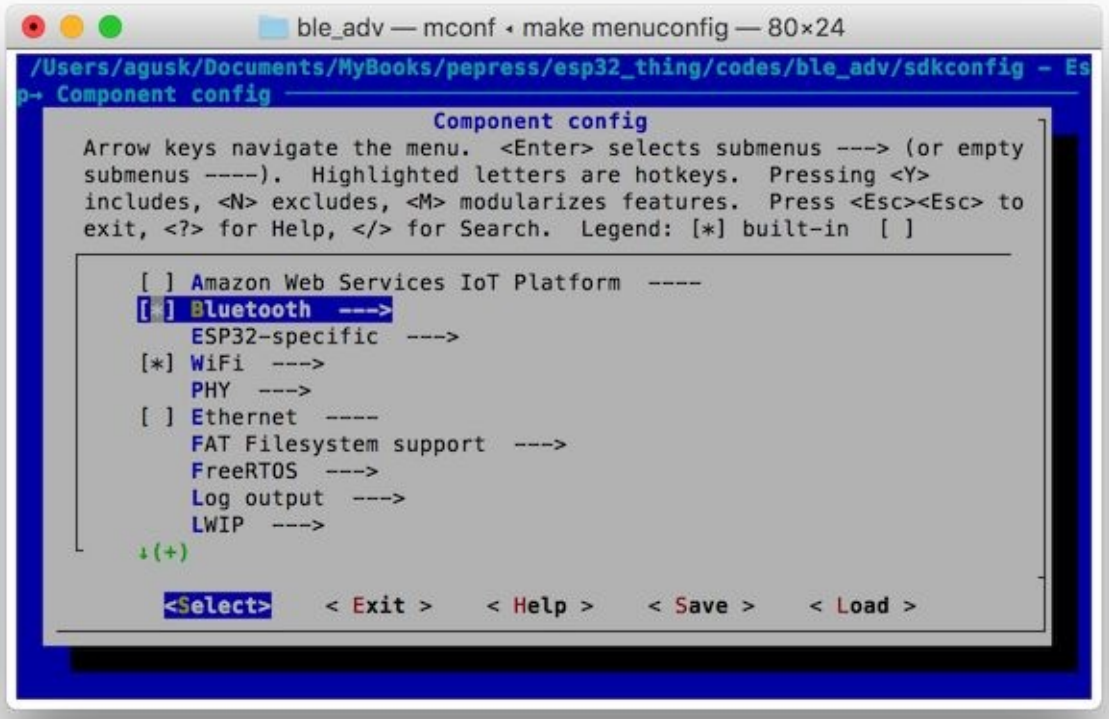
10.1 Working with Bluetooth

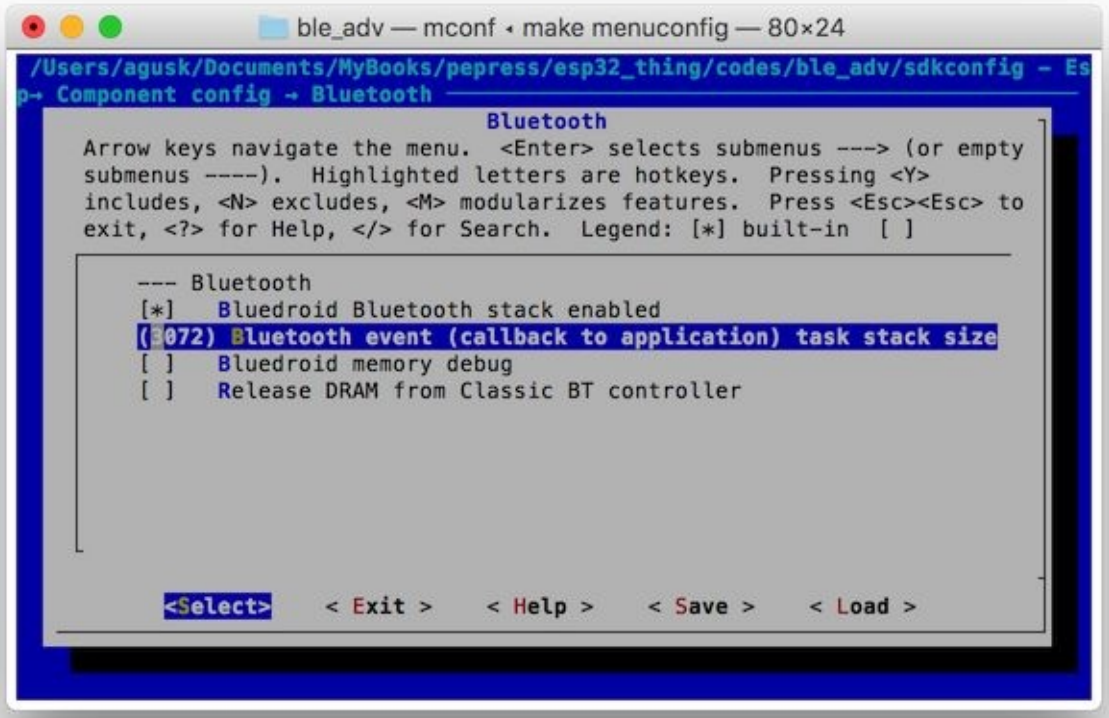
SparkFun ESP32 Thing has built-in BLE so we can use it in our program. You can read how to work with BLE in SparkFun ESP32 Thing board on this site, http://esp-idf.readthedocs.io/en/latest/api/bluetooth/index.html.

You should enable Bluetooth library via menuconfig. Select Component config.



Then, select Bluetooth. Save your configurations.





For testing, we can use the program sample from the framework. This is ble_adv program which transmit beacon. The following is the complete of ble_adv program.

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "bt.h"
#include <string.h>

#define HCI_H4_CMD_PREAMBLE_SIZE           (4)

/*  HCI Command opcode group field(OGF) */
#define HCI_GRP_HOST_CONT_BASEBAND_CMDS    (0x03 << 10)            /* 0x
#define HCI_GRP_BLE_CMDS                   (0x08 << 10)

#define HCI_RESET                          (0x0003 | HCI_GRP_HOST_CONT_B
#define HCI_BLE_WRITE_ADV_ENABLE           (0x000A | HCI_GRP_BLE_CMDS)
#define HCI_BLE_WRITE_ADV_PARAMS           (0x0006 | HCI_GRP_BLE_CMDS)
#define HCI_BLE_WRITE_ADV_DATA             (0x0008 | HCI_GRP_BLE_CMDS)

#define HCIC_PARAM_SIZE_WRITE_ADV_ENABLE        (1)
#define HCIC_PARAM_SIZE_BLE_WRITE_ADV_PARAMS    (15)
#define HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA      (31)

#define BD_ADDR_LEN     (6)                     /* Device address length
typedef uint8_t bd_addr_t[BD_ADDR_LEN];         /* Device address */

#define UINT16_TO_STREAM(p, u16) {*(p)++ = (uint8_t)(u16); *(p)++ = (uin
#define UINT8_TO_STREAM(p, u8)   {*(p)++ = (uint8_t)(u8);}
#define BDADDR_TO_STREAM(p, a)   {int ijk; for (ijk = 0; ijk < BD_ADDR_L
#define ARRAY_TO_STREAM(p, a, len) {int ijk; for (ijk = 0; ijk < len;

enum {
    H4_TYPE_COMMAND = 1,
    H4_TYPE_ACL     = 2,
    H4_TYPE_SCO     = 3,
    H4_TYPE_EVENT   = 4
};

static uint8_t hci_cmd_buf[128];

/*
 * @brief: BT controller callback function, used to notify the upper lay
 *         controller is ready to receive command
 */
static void controller_rcv_pkt_ready(void)
{
    printf("controller rcv pkt ready\n");
}

/*
 * @brief: BT controller callback function, to transfer data packet to u
 *         controller is ready to receive command
 */
static int host_rcv_pkt(uint8_t *data, uint16_t len)
{
```

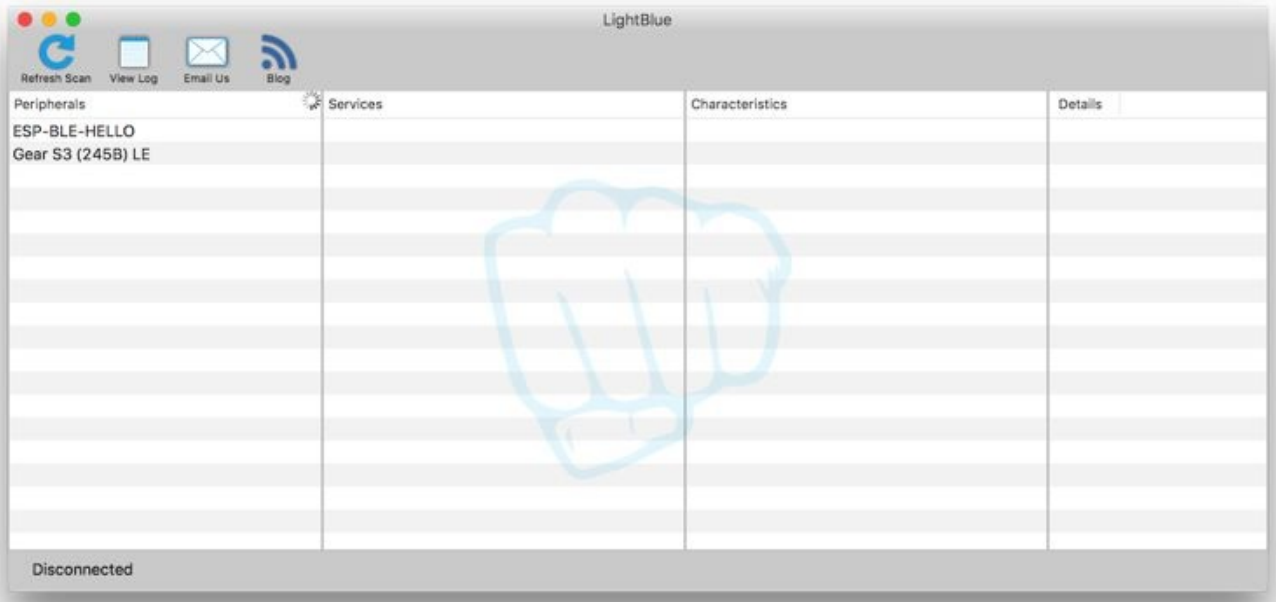
```
    printf("host rcv pkt: ");
    for (uint16_t i = 0; i < len; i++) {
        printf("%02x", data[i]);
    }
    printf("\n");
    return 0;
}

static esp_vhci_host_callback_t vhci_host_cb = {
    controller_rcv_pkt_ready,
    host_rcv_pkt
};

static uint16_t make_cmd_reset(uint8_t *buf)
{
    UINT8_TO_STREAM (buf, H4_TYPE_COMMAND);
    UINT16_TO_STREAM (buf, HCI_RESET);
    UINT8_TO_STREAM (buf, 0);
    return HCI_H4_CMD_PREAMBLE_SIZE;
}

static uint16_t make_cmd_ble_set_adv_enable (uint8_t *buf, uint8_t adv_e
{
    UINT8_TO_STREAM (buf, H4_TYPE_COMMAND);
    UINT16_TO_STREAM (buf, HCI_BLE_WRITE_ADV_ENABLE);
    UINT8_TO_STREAM  (buf, HCIC_PARAM_SIZE_WRITE_ADV_ENABLE);
    UINT8_TO_STREAM (buf, adv_enable);
    return HCI_H4_CMD_PREAMBLE_SIZE + HCIC_PARAM_SIZE_WRITE_ADV_ENABLE;
}

static uint16_t make_cmd_ble_set_adv_param (uint8_t *buf, uint16_t adv_i
        uint8_t adv_type, uint8_t addr_type_own,
        uint8_t addr_type_dir, bd_addr_t direct_bda,
        uint8_t channel_map, uint8_t adv_filter_policy)
{
    UINT8_TO_STREAM (buf, H4_TYPE_COMMAND);
    UINT16_TO_STREAM (buf, HCI_BLE_WRITE_ADV_PARAMS);
    UINT8_TO_STREAM  (buf, HCIC_PARAM_SIZE_BLE_WRITE_ADV_PARAMS );

    UINT16_TO_STREAM (buf, adv_int_min);
    UINT16_TO_STREAM (buf, adv_int_max);
    UINT8_TO_STREAM (buf, adv_type);
    UINT8_TO_STREAM (buf, addr_type_own);
    UINT8_TO_STREAM (buf, addr_type_dir);
    BDADDR_TO_STREAM (buf, direct_bda);
    UINT8_TO_STREAM (buf, channel_map);
    UINT8_TO_STREAM (buf, adv_filter_policy);
    return HCI_H4_CMD_PREAMBLE_SIZE + HCIC_PARAM_SIZE_BLE_WRITE_ADV_PARA
}


static uint16_t make_cmd_ble_set_adv_data(uint8_t *buf, uint8_t data_len
{
```

```
    UINT8_TO_STREAM (buf, H4_TYPE_COMMAND);
    UINT16_TO_STREAM (buf, HCI_BLE_WRITE_ADV_DATA);
    UINT8_TO_STREAM  (buf, HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA + 1);

    memset(buf, 0, HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA);

    if (p_data != NULL && data_len > 0) {
        if (data_len > HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA) {
            data_len = HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA;
        }

        UINT8_TO_STREAM (buf, data_len);

        ARRAY_TO_STREAM (buf, p_data, data_len);
    }
    return HCI_H4_CMD_PREAMBLE_SIZE + HCIC_PARAM_SIZE_BLE_WRITE_ADV_DATA
}

static void hci_cmd_send_reset(void)
{
    uint16_t sz = make_cmd_reset (hci_cmd_buf);
    esp_vhci_host_send_packet(hci_cmd_buf, sz);
}

static void hci_cmd_send_ble_adv_start(void)
{
    uint16_t sz = make_cmd_ble_set_adv_enable (hci_cmd_buf, 1);
    esp_vhci_host_send_packet(hci_cmd_buf, sz);
}

static void hci_cmd_send_ble_set_adv_param(void)
{
    uint16_t adv_intv_min = 256; // 160ms
    uint16_t adv_intv_max = 256; // 160ms
    uint8_t adv_type = 0; // connectable undirected advertising (ADV_IND
    uint8_t own_addr_type = 0; // Public Device Address
    uint8_t peer_addr_type = 0; // Public Device Address
    uint8_t peer_addr[6] = {0x80, 0x81, 0x82, 0x83, 0x84, 0x85};
    uint8_t adv_chn_map = 0x07; // 37, 38, 39
    uint8_t adv_filter_policy = 0; // Process All Conn and Scan

    uint16_t sz = make_cmd_ble_set_adv_param(hci_cmd_buf,
                  adv_intv_min,
                  adv_intv_max,
                  adv_type,
                  own_addr_type,
                  peer_addr_type,
                  peer_addr,
                  adv_chn_map,
                  adv_filter_policy);
    esp_vhci_host_send_packet(hci_cmd_buf, sz);
}
```

```c
static void hci_cmd_send_ble_set_adv_data(void)
{
    char *adv_name = "ESP-BLE-HELLO";
    uint8_t name_len = (uint8_t)strlen(adv_name);
    uint8_t adv_data[31] = {0x02, 0x01, 0x06, 0x0, 0x09};
    uint8_t adv_data_len;

    adv_data[3] = name_len + 1;
    for (int i = 0; i < name_len; i++) {
        adv_data[5 + i] = (uint8_t)adv_name[i];
    }
    adv_data_len = 5 + name_len;

    uint16_t sz = make_cmd_ble_set_adv_data(hci_cmd_buf, adv_data_len, (
    esp_vhci_host_send_packet(hci_cmd_buf, sz);
}

/*
 * @brief: send HCI commands to perform BLE advertising;
 */
void bleAdvtTask(void *pvParameters)
{
    int cmd_cnt = 0;
    bool send_avail = false;
    esp_vhci_host_register_callback(&vhci_host_cb);
    printf("BLE advt task start\n");
    while (1) {
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        send_avail = esp_vhci_host_check_send_available();
        if (send_avail) {
            switch (cmd_cnt) {
            case 0: hci_cmd_send_reset(); ++cmd_cnt; break;
            case 1: hci_cmd_send_ble_set_adv_param(); ++cmd_cnt; break;
            case 2: hci_cmd_send_ble_set_adv_data(); ++cmd_cnt; break;
            case 3: hci_cmd_send_ble_adv_start(); ++cmd_cnt; break;
            }
        }
        printf("BLE Advertise, flag_send_avail: %d, cmd_sent: %d\n", sen
    }
}

void app_main()
{
    esp_bt_controller_init();

    if (esp_bt_controller_enable(ESP_BT_MODE_BTDM) != ESP_OK) {
        return;
    }

    xTaskCreatePinnedToCore(&bleAdvtTask, "bleAdvtTask", 2048, NULL, 5,
}
```

Compile and upload the program to SparkFun ESP32 Thing board.

Using Bluetooth/BLE scanner app such as LightBlue in my macOS, we can see our advertising beacon from the board. It shows as ESP-BLE-HELLO.



Source Code

You can download source code on http://www.aguskurniawan.net/book/esp32_b2198.zip .

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net