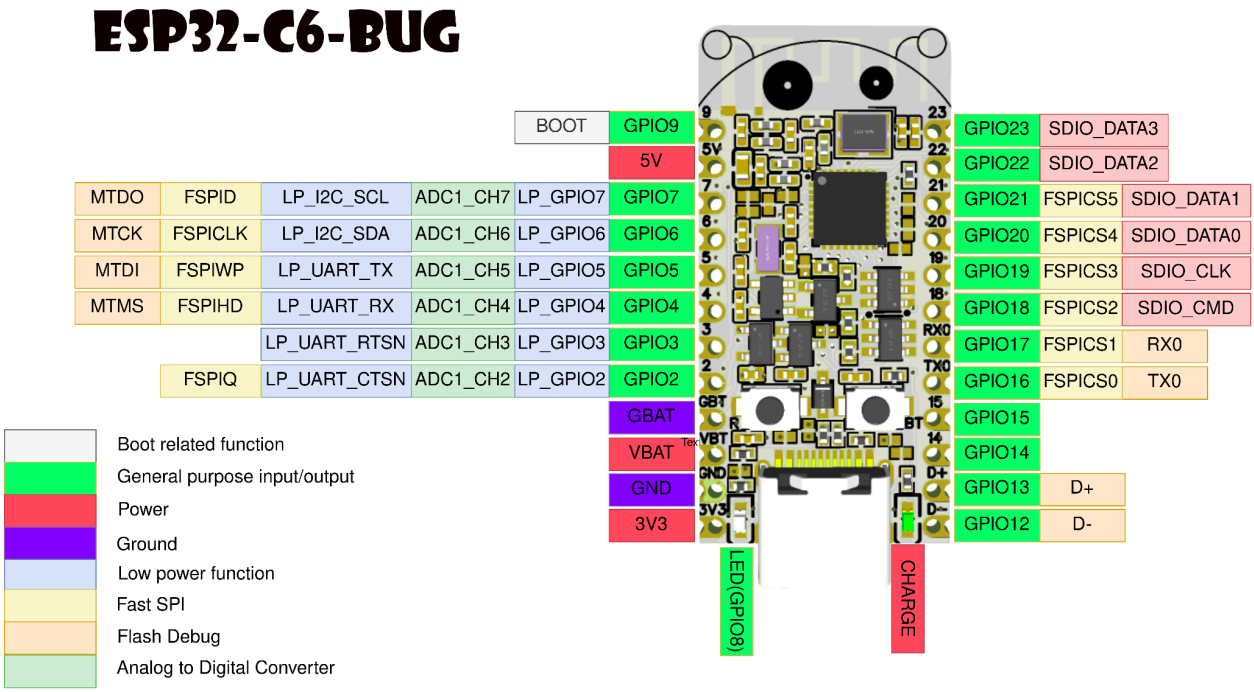
Table of contents

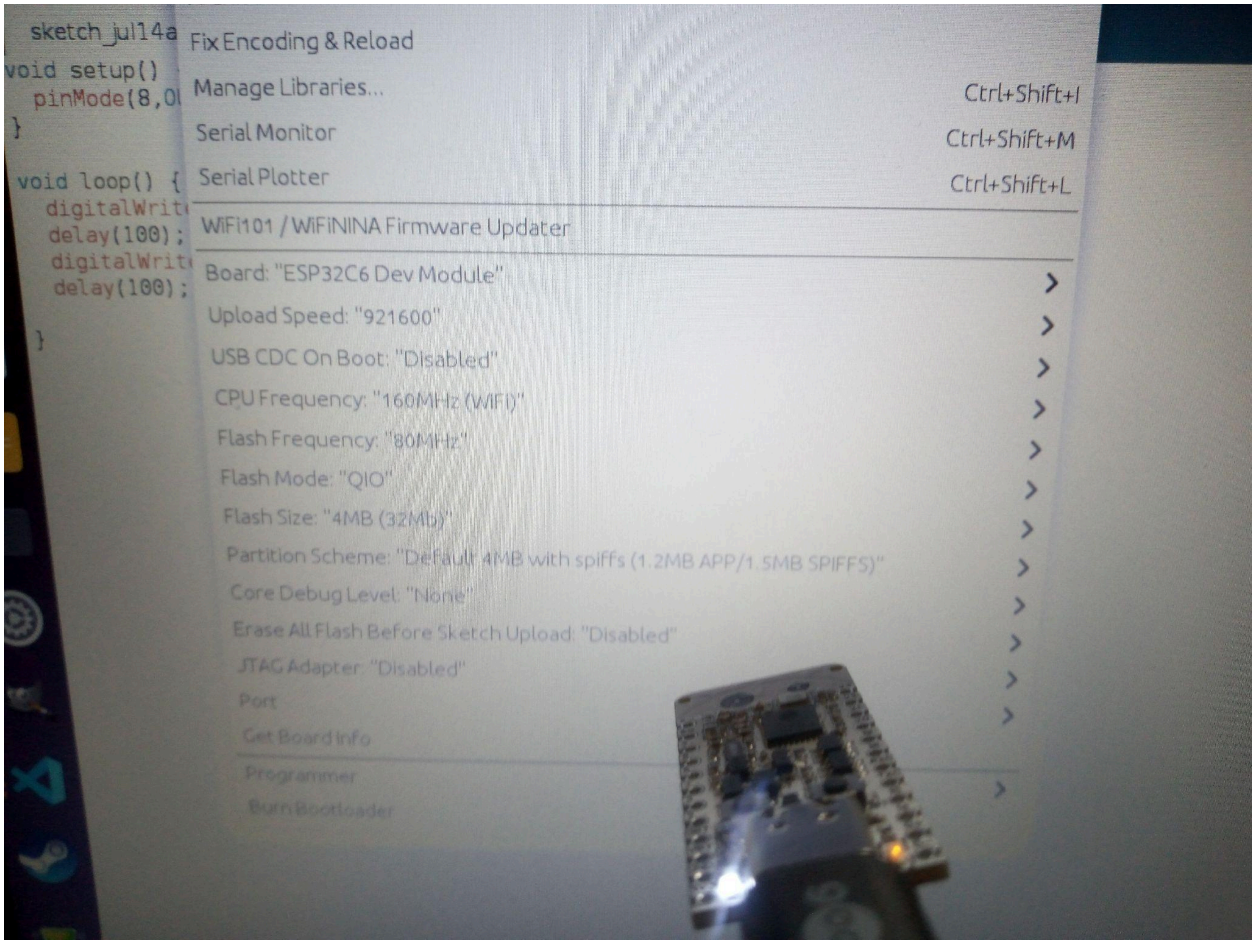
What is it?

The ESP32-C6-Bug is a tiny development board based on the latest C6 MCU of the ESP32 family with reduced power consumption and battery support. Supported wireless protocols include Wi-Fi 6, BLE 5, Thread, Matter and Zigbee.

Pinout and supported frameworks



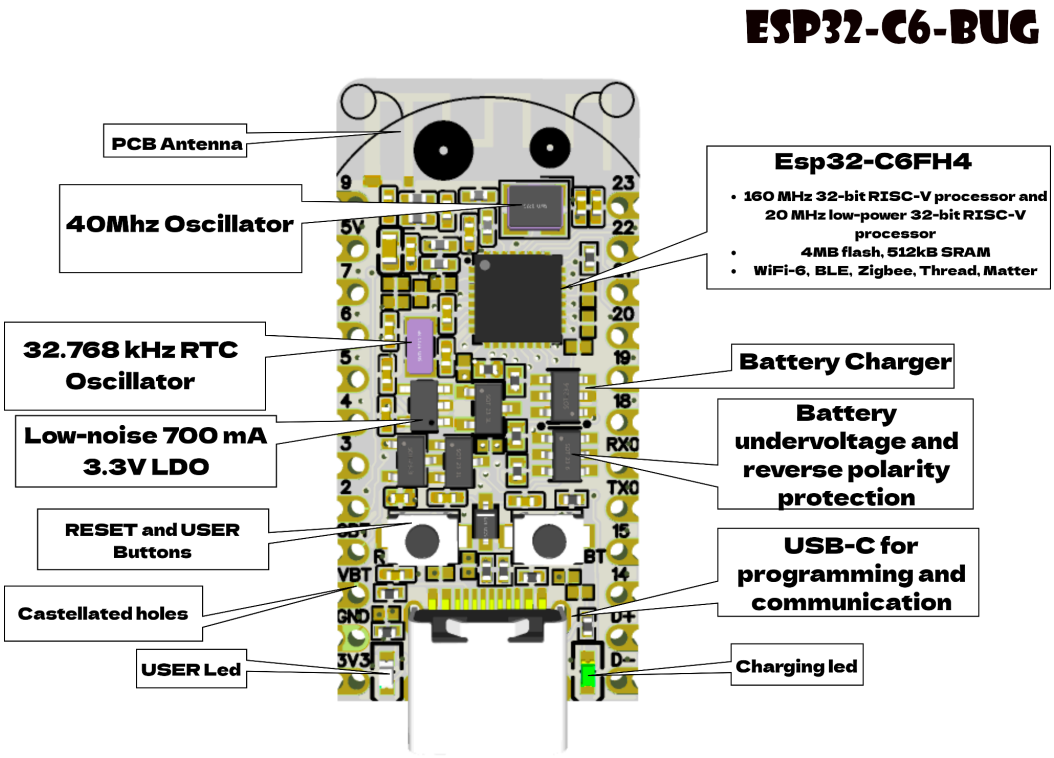
Picture 1: Pinout and pin descriptions



Programming Esp32-C6 in Arduino

Following this tutorial you can install custom Arduino Core that supports Esp32-C6:
https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/installing.html
The required core: https://github.com/espressif/arduino-esp32/tree/idf-release/v5.1

Features



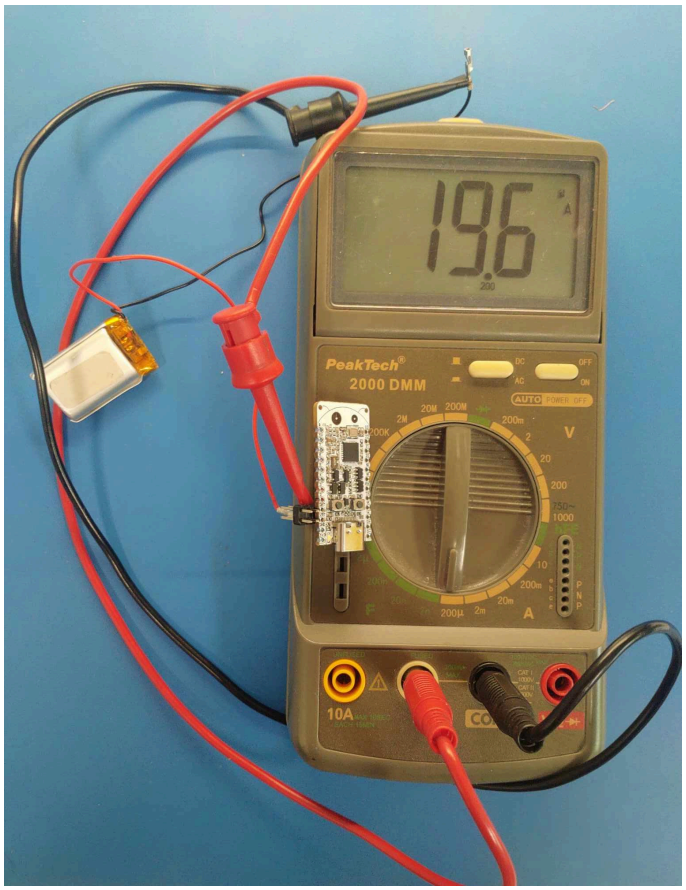
Picture 2: Features Diagram

Chip features

- 32-bit RISC-V 160MHz processor
- 32-bit RISC-V 20MHz low-power processor
- **512 MB SRAM, 4MB Flash**
- **WiFi-6, BLE 5 + IEEE 802.15.4 radio (Zigbee, Thread, Matter…)**
- 2.4 GHz Wi-Fi 6 (802.11ax) radio also supports the 802.11b/g/n for backward compatibility.
- SPI, UART, I2C, I2S, RMT, TWAI, PWM, SDIO, Motor Control PWM, 12-bit ADC and a temperature sensor.

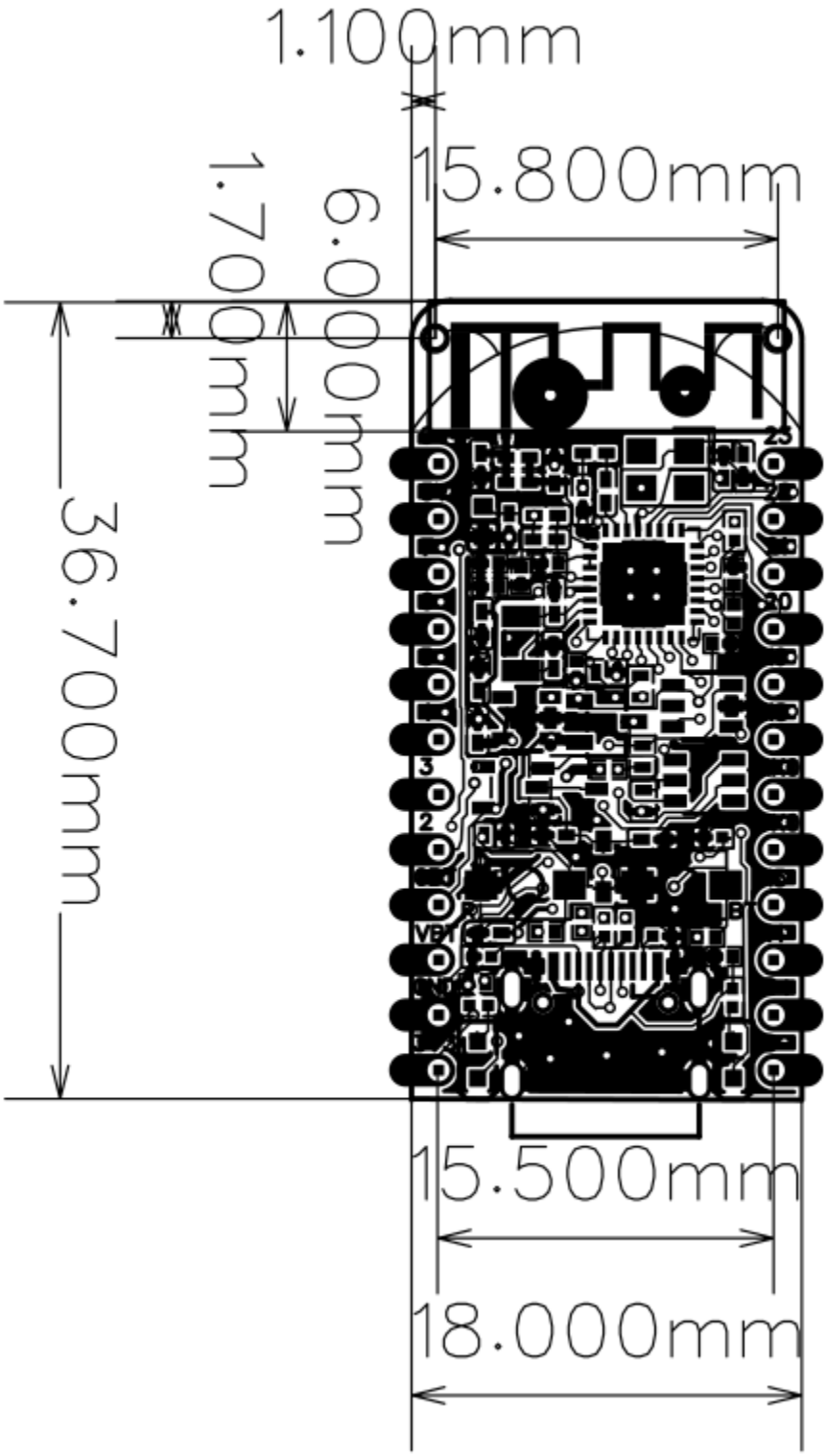
Board features

- Battery undercharge and reverse polarity protection.
- Battery charge circuit.
- On board battery level measurement circuit.
- External 32.768 kHz RTC oscillator and 40 MHz oscillator.
- Reset and User-controlled buttons.
- Charging indication and User-controlled leds.
- **20 uA** deep sleep power consumption(with Timer wake up).
- 700 mA low-noise LDO.
- Castellated holes.
- Pin names on both sides.
- USB-C for programming and communication.
- Fits on the breadboard.
- 19 GPIOs exposed.
- Two M1 mounting holes.



20uA sleep mode wake up

Dimensions

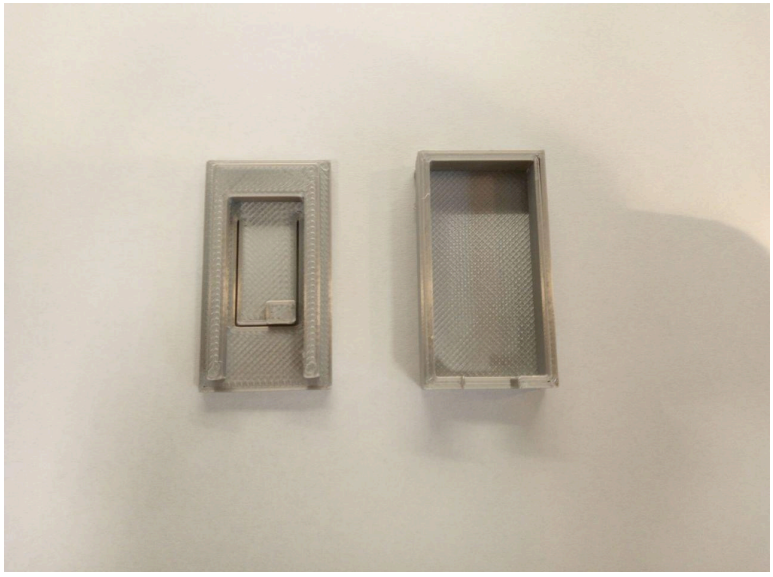


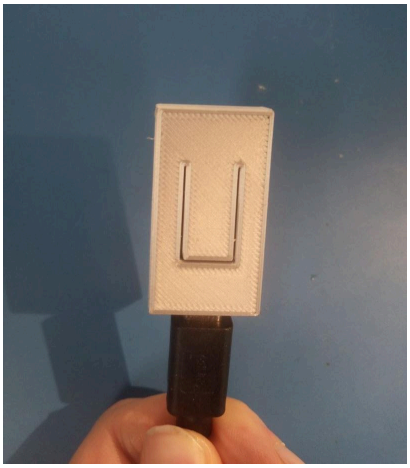
Existing solutions comparison

	Esp32-C6-Bug	ESP32-C6-DevKitC-1	ESP32-C6-DevKitM-1
Manufacturer	Prokyber	Espressif	Espressif
Chip	Esp32-C6-FH4	ESP32-C6	ESP32-C6
Flash size	4MB	8MB	4MB
Battery charging	Yes	No	No
Battery protection	Yes	No	No
Battery measurement	Yes	No	No
32.768 kHz RTC crystal	Yes	No	No
Size	18x36.7mm	25.4x53.63mm	25.4x58.15mm
Mounting holes	M1x2	No	No
Castellated holes	Yes	No	No
GPIO	19+1(LED only)+2(32kHz crystal)	23	22
Price on Mouser(USD)	25	10	10

Enclosure

A simple 3d printable enclosure was made and is available on thingiverse:



Link:https://www.thingiverse.com/thing:6084609

Support Links

- Schematics, dimensions, photos.
- Programming Esp32-c6 chip with ESP-IDF: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c6/get-started/index.html
- ESP-IDF Basic examples are provided by Espressif and can be found here https://github.com/espressif/esp-idf/tree/release/v5.1/examples/wifi
  Tested examples are:
  get_stated/blink, get_stated/hello_world, wifi/scan, wifi/getting_started/softAP, wifi/getting_started/station, **zigbee/light_sample, system/deep_sleep,** peripherals/adc/continious_read.
- Instructables example projects.
- 3d printable enclosure - https://www.thingiverse.com/thing:6084609

Possible applications

Since the chip is very new, complex projects will require a significant amount of work, however here is some inspiration for you :

- Multiprotocol USB dongle: USB-Zigbee/Thread/Matter dongle (like this https://www.home-assistant.io/skyconnect/)
- WiFi-Openthread router https://github.com/espressif/esp-idf/tree/master/examples/openthread
- WiFi-Zigbee gateway https://github.com/espressif/esp-idf/tree/master/examples/zigbee/esp_zigbee_gateway
- Ethernet-Zigbee/Thread/Matter bridge
- Light switch and Bulb control via Zigbee: https://github.com/espressif/esp-idf/tree/master/examples/zigbee/light_sample

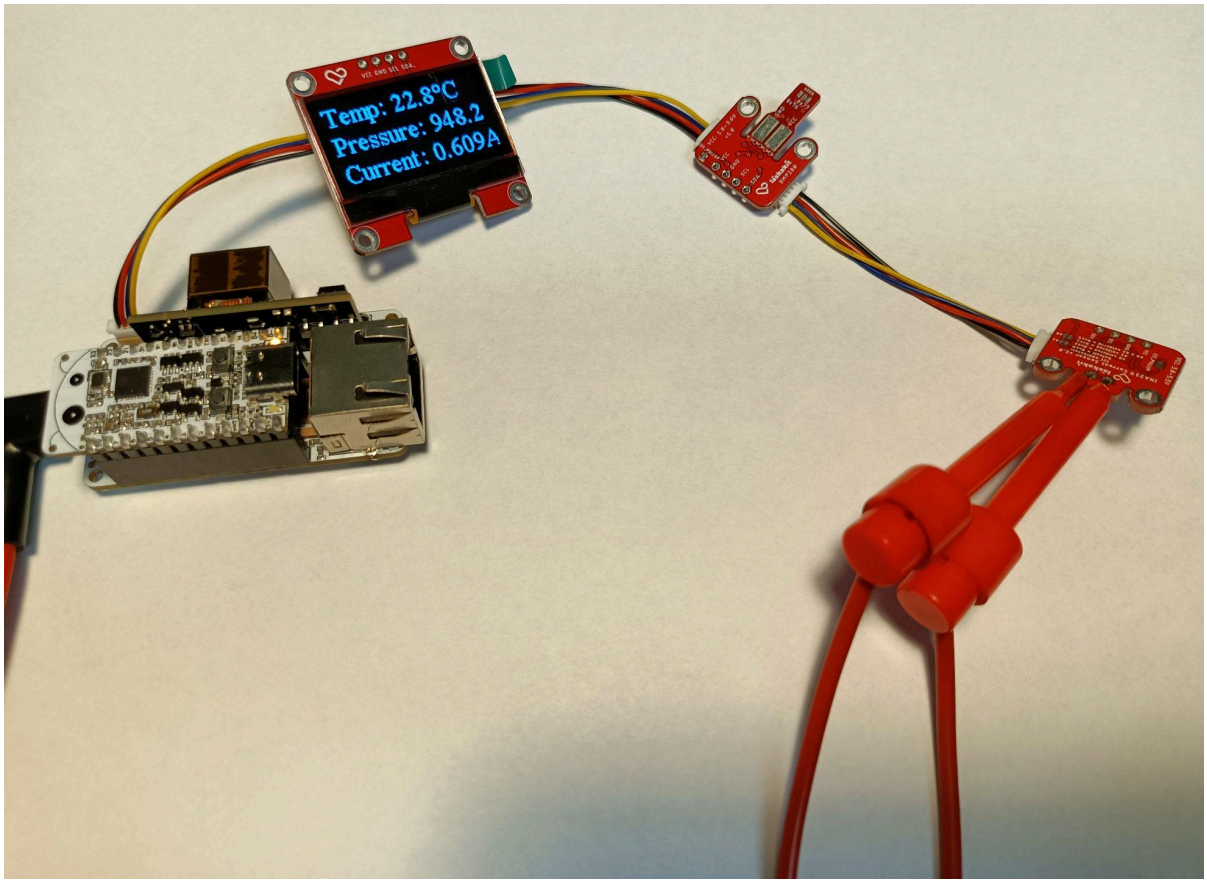
Demos

ESPHome demos

Demo 1: Connecting multiple I2C sensors to Homeassistant

Intro

Programming HomeAssistants sensors and devices is done via .yaml scripts. As of Februaray 2024 the Esp32-C6 is not fully supported, however basic functionality like Wifi and peripheral access is supported. Sensors that use I2C can be connected with ease using Stemma QT connector of Esp32-Eth-Bug



What do I need to make that?

- Esp32-C6-BUG
- Esp32-Eth-BUG
- USB Type-C for flashing

- Some I2C sensors supported by ESPHome with Stemma QT connector. I used:
    - SH1106 128x64 based display.
    - INA219 current sensor.
    - Bmp280 temperature and pressure sensor.

Result

The device is connected to HomeAssistant, sensor data are shown on the display and can be accessed via Home Assistant.

How do I do that?

- Go to Add-ons and open ESP Home add on, you can also click this link to open it https://my.home-assistant.io/redirect/config_flow_start?domain=esphome
- Install ESP Home if you don't have it yet
- Open ESP Home add-on
- Click NEW DEVICE at bottom right
- Don't install the firmware on the device yet, just click skip or continue
- Enter name of the device and WIFI detail if it asks you
- Click edit on the device you just created, delete the generated configuration and replace it by this:

```
esphome:
 name: esp32c6bug
 friendly_name: esp32c6bug

esp32:
 board: esp32-c6-devkitc-1
 variant: ESP32C6
 framework:
   platform_version:
https://github.com/stintel/platform-espressif32#esp32-c6-test
   type: esp-idf
   version: 5.1.0
# Enable logging
logger:

# Enable Home Assistant API
api:
#  encryption:
#    key: "generate api key here https://esphome.io/components/api"

ota:
```

```
#  password: "enter some password containing only small letters and
numbers"

wifi:
 ssid: !secret wifi_ssid
 password: !secret wifi_password

i2c:
 sda: 21
 scl: 20
 scan: true

display:
 - platform: ssd1306_i2c
   model: "SH1106 128x64"
  # reset_pin: D0
   address: 0x3C
   lambda: |-
     it.printf(0, 0, id(tnr1), "Temp: %.1f°C ", id(temp).state );
     it.printf(0, 22, id(tnr1), "Pressure: %.1f ", id(pressure).state );
     it.printf(0, 44, id(tnr1), "Current: %.3fA ", id(current).state );

font:
 - file: "fonts/times-new-roman.ttf"
   id: tnr1
   size: 20
 - file: "fonts/times-new-roman.ttf"
   id: tnr2
   size: 35

sensor:
 - platform: ina219
   address: 0x40
   shunt_resistance: 0.1 ohm
   current:
     name: "INA219 Current"
     id: current
   power:
     name: "INA219 Power"
   bus_voltage:
```

```
    name: "INA219 Bus Voltage"
  shunt_voltage:
    name: "INA219 Shunt Voltage"
  max_voltage: 32.0V
  max_current: 3.2A
  update_interval: 2s

- platform: bmp280
  temperature:
    name: "Outside Temperature"
    oversampling: 16x
    id: temp
  pressure:
    name: "Outside Pressure"
    id: pressure
  address: 0x77
  update_interval: 60s
```

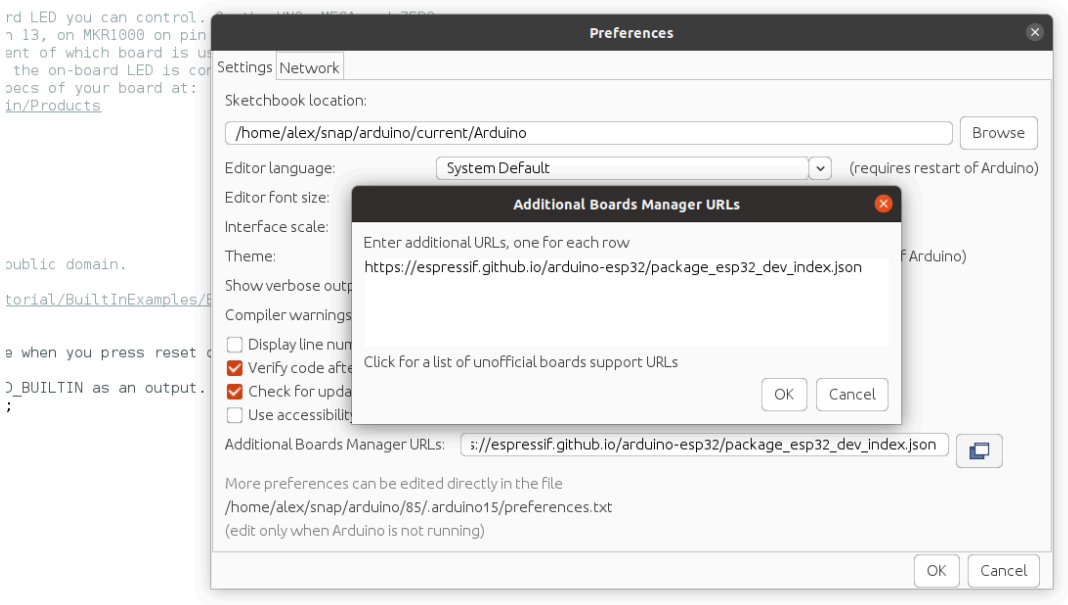
- If you want uncomment OTA password and API encryption and enter ota and api passwords
- Flash the device, click upload and
    - If you are using https click plug into this computer, connect the device, select serial port and wait for the code to upload
    - If you are not using https click manual download, then go to https://web.esphome.io/ click connect, connect the device, select serial port, then click install and select the file you just downloaded
- After installing, the device should automatically pop up in homeassistant, go to settings, Devices & Services. The device should be at the top, click the configure button, if you enabled api encryption it will ask for the key.

Arduino Framework demos

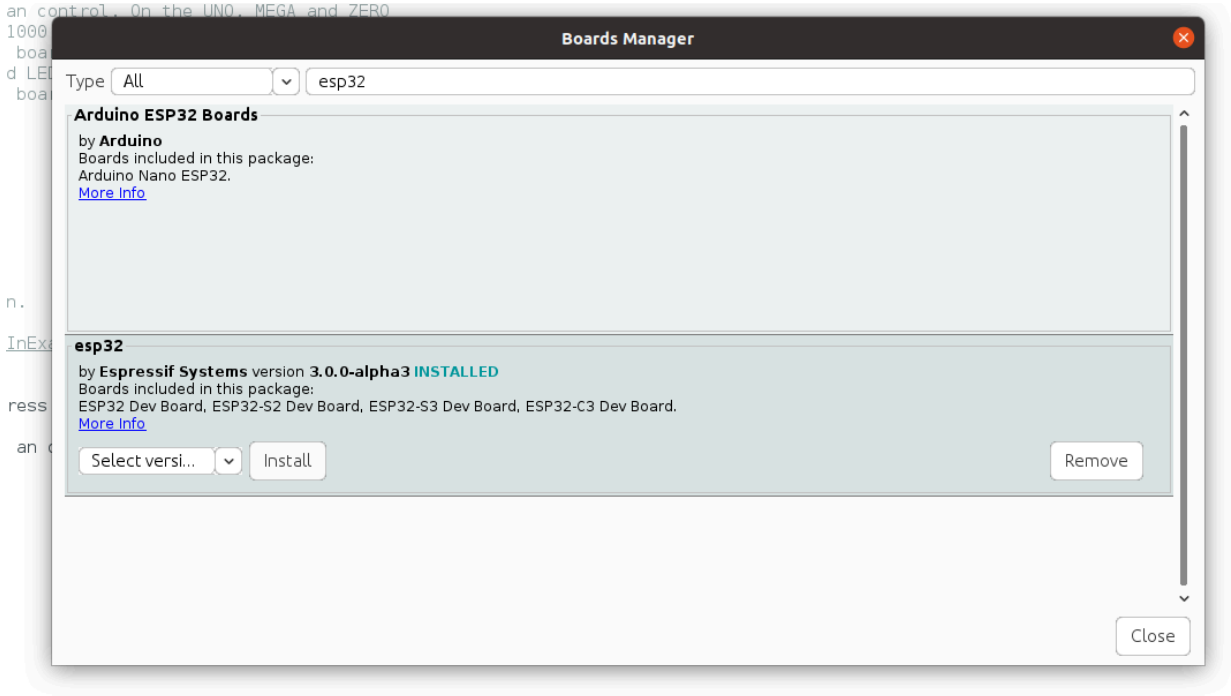
Esp32-C6 is now officially supported in the latest 3.0.0 Arduino core. To install the newest core you should insert index.json link(
Stable -
https://espressif.github.io/arduino-esp32/package_esp32_index.json
or
Development (Supports Esp32-C6 as of 5.2.2023) -
https://espressif.github.io/arduino-esp32/package_esp32_dev_index.json

)

from https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html
Into the preferences tab of Arduino IDE.



After this step the 3.0.0 Arduino core will be available for installation via boards manager.



When you installed the newest core, you should see the Esp32C6 Dev Module in Tools/Board/Esp32 Arduino tab.

Demo 2: Ethernet Example

Intro

The Esp32-Bug-Eth shield allows easy connection to Ethernet, this example shows how and straightforward it is. After the board is programmed, you can disconnect the shield from PC and power it via PoE!

What do I need to make that?

- Esp32-C6-BUG
- Esp32-Eth-BUG
- USB Type-C for flashing
- Ethernet cable

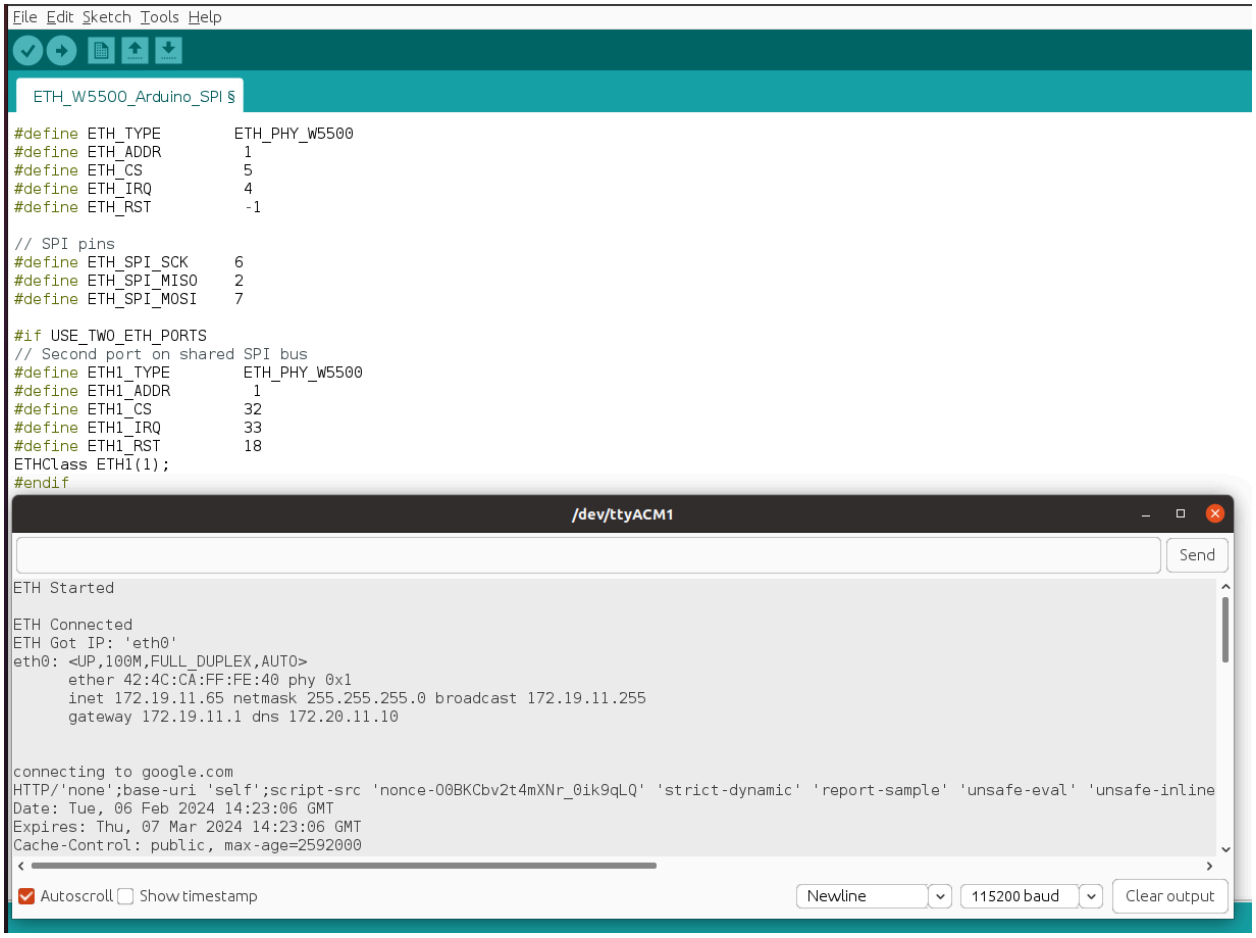
Result

The Esp32-C6-Bug is connected to the Internet via Ethernet cable. From here you can easily start development of other Ethernet applications.

How do I do that?

1. Open your Arduino IDE
2. Open File/Examples/Ethernet/ETH_W5500_Arduino_SPI example

3. Change the pin definitions according to the provided screenshot



4. Set Tools/USB CDC enabled on Boot to True
5. Ensure Tools/board is set to Esp32C6 Dev Module
6. Insert Esp32-C6-Bug into Esp32-Eth-Bug
7. Connect Esp32-Eth-Bug to your PC
8. Set Tools/Port to the port of your Esp32-C6-Bug
9. Flash the code
10. Connect the Ethernet cable
11. Open port monitor
12. Observe the output(your board should connect get IP and connect to the Internet)

Demo 3: Stemma QT I2C sensors chain example

Intro

The Esp32-Bug-Eth also has a stemma QT connector, which makes it compatible with many Adafruit peripheral sensors. Daisy chaining is a very easy and comfortable way of connecting many of them together.
Depending on your sensors, there are different approaches to program them. However there is one general thing you can do for all the possible sensor chains and it's I2C Scanning.
The I2C scanning process will 'ping' all the connected sensors and return their respective addresses.

What do I need to make that?

- Esp32-C6-BUG
- Esp32-Eth-BUG
- USB Type-C for flashing
- Some sensors with Stemma QT connector

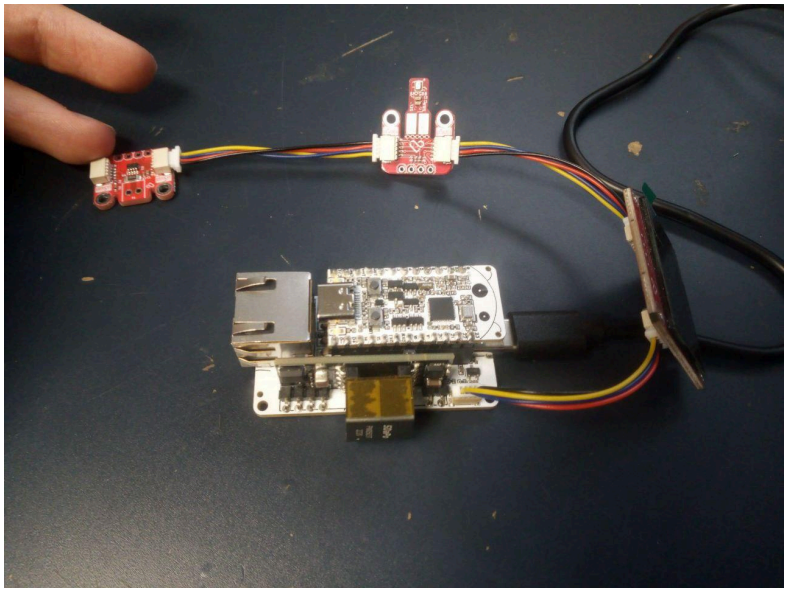
Result

Multiple sensors are connected to Esp32-Bug-Eth using only one connector.
The I2C scan returns the addresses of all of them:

```
#include "Wire.h"

void setup() {
  Serial.begin(115200);
  Wire.begin(21,20);
}

void loop() {
  byte error, address;
  int nDevices = 0;
  delay(5000);

  Serial.println("Scanning for I2C devices ...");
  for(address = 0x01; address < 0x7f; address++){
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0){
      Serial.printf("I2C device found at address 0x
      nDevices++;
    } else if(error != 2){
      Serial.printf("Error %d at address 0x%02X\n",
    }
  }
  if (nDevices == 0){
    Serial.println("No I2C devices found");
  }
}
```

```
I2C device found at address 0x3C
I2C device found at address 0x40
I2C device found at address 0x77
Scanning for I2C devices ...
I2C device found at address 0x3C
I2C device found at address 0x40
I2C device found at address 0x77
Scanning for I2C devices ...
I2C device found at address 0x3C
I2C device found at address 0x40
I2C device found at address 0x77
Scanning for I2C devices ...
I2C device found at address 0x3C
I2C device found at address 0x40
I2C device found at address 0x77

Autoscroll  Show timestamp
```

Three sensors connected into the i2c chain



How do I do that?

Well the approach is quite straightforward.

1. Insert Esp32-C6-Bug into Esp32-Eth-Bug
2. Connect all the sensors to Esp32-Eth-Bug
3. Connect Esp32-Eth-Bug to your PC
4. Open Wire/WireScan example
5. Add Wire.begin(21,20); after Serial.begin(115200);
6. Set Tools/USB CDC enabled on Boot to True
7. Ensure Tools/board is set to Esp32C6 Dev Module
8. Set Tools/Port to the port of your Esp32-C6
9. Flash the example
10. Observe the output in Serial Monitor, the number of found addresses should be the same as the number of connected devices(assuming all the devices have different addresses)

CircuitPython Demos

To use Esp32-C6-Bug Circuitpython demos are based around the Circuitpython port for Esp32-C6 which can be found here
https://github.com/adafruit/circuitpython/tree/main/ports/espressif

Building and flashing CircuitPython

To work with CircuitPython it’s necessary to build and flash it. The whole process is quite straightforward and is also described in the CircuitPython repo.

- Open your command line and clone the CircuitPython repo (be sure to user recursive as CircuitPython has some external dependencies):
  git clone --recursive https://github.com/adafruit/circuitpython.git
- Run cd ports/espressif from circuitpython/ to enter the espressif port directory
- Run ./esp-idf/install.sh to prepare the esp-idf framework (After this initial installation, you must add the ESP-IDF tools to your path.)
- Run 'source ./esp-idf/export.sh'
- Then run 'make BOARD=espressif_esp32c6_devkitm_1_n4 PORT=/dev/ttyACM0 flash' to flash the CircuitPython(Replace the port id)
- After the board is flashed you can access REPL via Serial terminal

When the flashing process finishes the REPL becomes available via serial port.

Esp-IDF demos

ESP-IDF demos are based on examples from official ESP-IDF repository(https://github.com/espressif/esp-idf/tree/release/v5.1/examples)
First you need to install ESP-IDF following this tutorial:(https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/)
Here is a quick video demonstrating how it is done:[video is in preparation]

Demo 4: BLE advertising

Intro

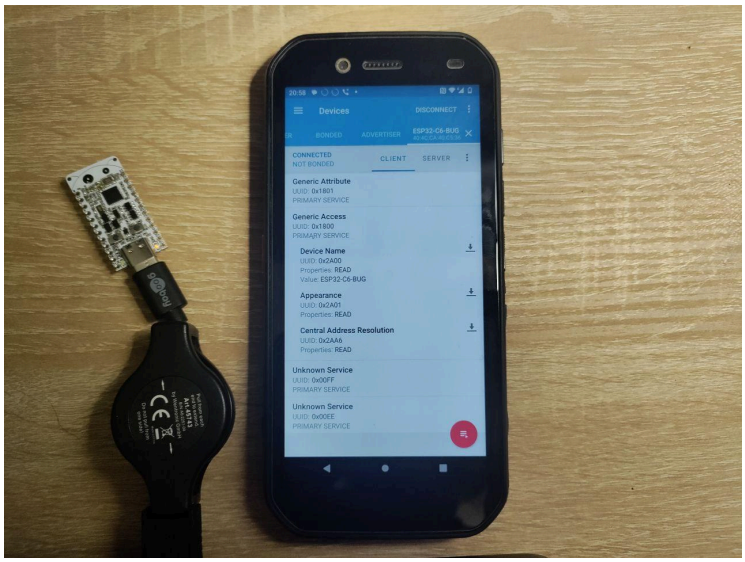
This example demonstrates how to connect Esp32-C6-Bug to your phone via BLE. It is largely based on https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble. After the successful connection it is possible to read data from Esp32-C6-Bug. This example uses ESP-IDF as a framework for programming, so be sure to install it before starting.

What do I need to make that?

- Esp32-C6-BUG
- USB Type-C for flashing

Result

Esp32-C6-BUG is advertising data, you can read it via your phone



How do I do that?

After ESP-IDF is configured you can run the following commands:
1)alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
Use your path to esp-idf/export.sh
2)get_idf
3)cp -r $IDF_PATH/examples/bluetooth/bluedroid/ble/gatt_server .
To copy example for ble
4)update line 37 and line 53 in gatt_server/main/gatts_demo.c to set device name:

```
#include "esp_gatts_api.h"
#include "esp_bt_defs.h"
#include "esp_bt_main.h"
#include "esp_gatt_common_api.h"

#include "sdkconfig.h"

#define GATTS_TAG "ESP32-C6-BUG-DEMO"

///Declare the static function
static void gatts_profile_a_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param);
static void gatts_profile_b_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if, esp_ble_gatts_cb_param_t *param);

#define GATTS_SERVICE_UUID_TEST_A   0x00FF
#define GATTS_CHAR_UUID_TEST_A      0xFF01
#define GATTS_DESCR_UUID_TEST_A     0x3333
#define GATTS_NUM_HANDLE_TEST_A     4

#define GATTS_SERVICE_UUID_TEST_B   0x00EE
#define GATTS_CHAR_UUID_TEST_B      0xEE01
#define GATTS_DESCR_UUID_TEST_B     0x2222
#define GATTS_NUM_HANDLE_TEST_B     4

#define TEST_DEVICE_NAME            "ESP32-C6-BUG"
#define TEST_MANUFACTURER_DATA_LEN  17

#define GATTS_DEMO_CHAR_VAL_LEN_MAX 0x40

#define PREPARE_BUF_MAX_SIZE 1024

static uint8_t char1_str[] = {0x11,0x22,0x33};
static esp_gatt_char_prop_t a_property = 0;
static esp_gatt_char_prop_t b_property = 0;

static esp_attr_value_t gatts_demo_char1_val =
{
    .attr_max_len = GATTS_DEMO_CHAR_VAL_LEN_MAX,
    .attr_len     = sizeof(char1_str),
    .attr_value   = char1_str,
};
```

5)idf.py flash
6)Open program for BLE monitoring(for example NRFConnect) and connect to your newly flashed board
7)The whole process is demonstrated in this short video[video is ready]

Demo 5: Connecting two Esp32-C6-Bugs with Zigbee

Intro

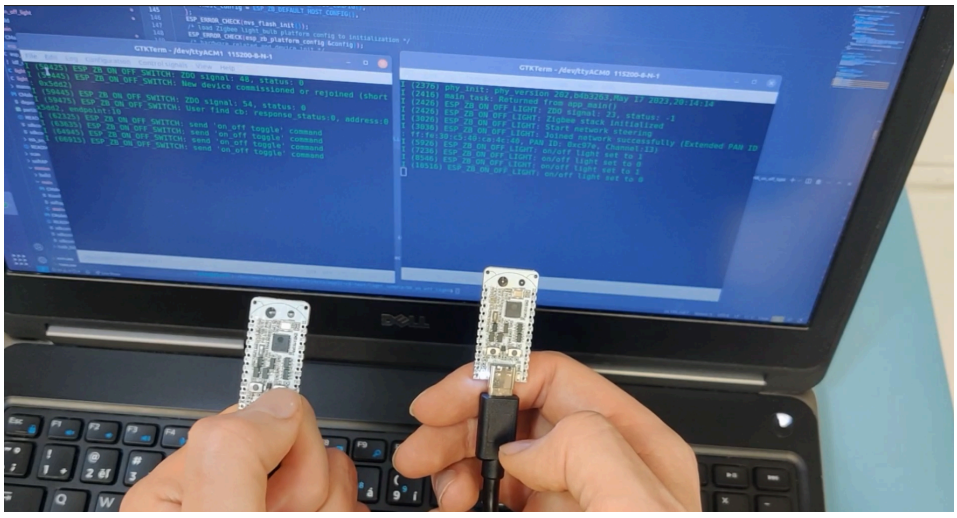
Zigbee is a low-power standard targeted at battery-powered devices in wireless control and monitoring applications[wikipedia]. Esp32-C6-Bug supports both Zigbee coordinator and end-device roles.
This example demonstrates how to connect two Esp32-C6-Bugs via Zigbee. One of them is End-Device, the second one is Zigbee Coordinator. It is based on https://github.com/espressif/esp-idf/tree/master/examples/zigbee/light_sample .

What do I need to make that?

- 2xEsp32-C6-BUG
- 2xUSB Type-C for flashing

Result

Esp32-C6-BUG Coordinator acts as a switch, Esp32-C6-BUG End device acts as light-bulb. After you press the button on Esp32-C6-BUG Coordinator the light changes state on the Esp32-C6-BUG End device.



How do I do that?

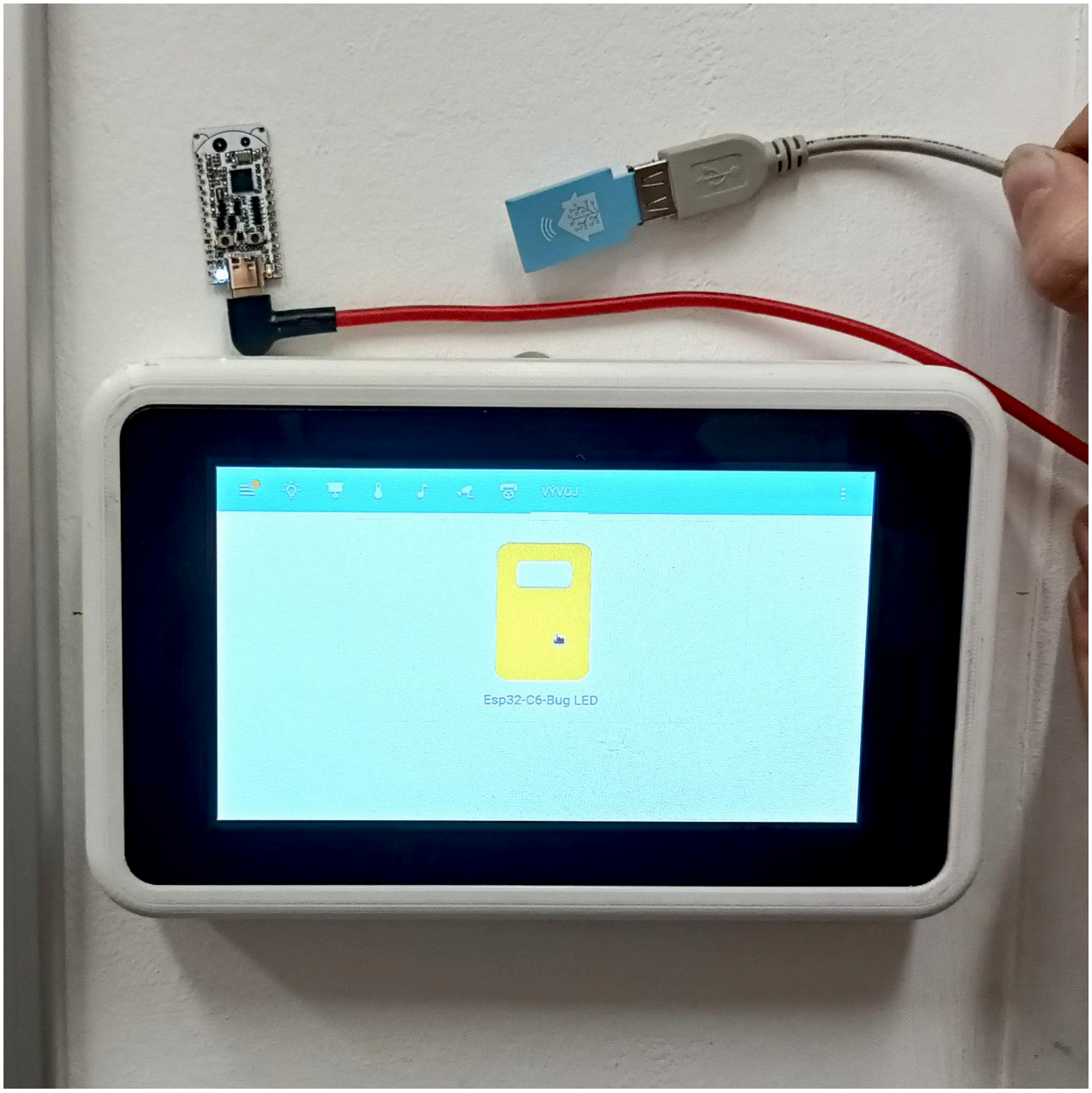
After ESP-IDF is configured you can run the following commands:
1)alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
Use your path to esp-idf/export.sh
2)get_idf
3)cp -r $IDF_PATH/examples/zigbee/light_sample .

4)cd light_sample/HA_on_off_switch
5)idf.py set-target esp32c6
6)Connect the first Esp32-C6-Bug to your PC
7)idf.py flash
After this command the code will be uploaded to Esp32-C6-Bug coordinator
8)Disconnect the first Esp32-C6-Bug to your PC, connect the second Esp32-C6-Bug to your PC
9) cd ..
10)cd ..
11)idf.py set-target esp32c6
12)idf.py flash(After this command the code will be uploaded to Esp32-C6-Bug End device)
13)repeat steps 4-12 for for light_sample/HA_on_off_light directory for the second Esp32-C6-Bug.
14)Connect both boards to usb and open the serial monitor,
15) Reset the ‘switch’-board then the ‘bulb’-board
16)observe the pairing process(The switch board should be enabled first, so it can for the network)
17)when you press the user button on the ‘switch’-board, the ‘led’ button should output that the LED stage is changed

Demo 6: Connecting to Home Assistant via Zigbee

Intro

Esp32-C6-Bug integration into Home Assistant can be done using multiple approaches, one of them is via Zigbee using Home Assistant SkyConnect. Of course the functionality is limited, however you can make a simple Zigbee controlled LED(or bulb) by slightly modifying the previous example.



What do I need to make that?

- Esp32-C6-BUG
- Home Assistant SkyConnect
- USB Type-C for flashing

Result

The Esp32-C6-Bug is connected to Home Assistant via Zigbee. When the button is pressed in Home Assistant the Esp32-C6-Bug Led changes state.

How do I do that?

- The first step is to modify examples/zigbee/light_sample/HA_on_off_light from previous demo, so the LED actually changes it's state when the button is pressed(Of course you can later add some relay or MOSFET to this pin to control something else rather than a simple LED)
- Open light_sample/HA_on_off_light/esp_zb_light.c file
- Let's add some blinking functionality. To do that, it's necessary to initialize the led pin using:

```
#define BLINK_GPIO 8
static void configure_led(void)
{
   gpio_reset_pin(BLINK_GPIO);
   gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
}
```

- Then toggle pin based on the current switch position using(The `s_led_state` must be changed based on switch position):

```
uint8_t s_led_state = 0;

static void blink_led(void)
{
   /* Set the GPIO level according to the state (LOW or HIGH)*/
   gpio_set_level(BLINK_GPIO, s_led_state);
}
```

- Go to esp_zb_light.c and add the blink_led and configure_led functions(don't forget to add s_led_state variable and BLINK_GPIO define) to the top of the code, before attr_cb function
- First add configure_led() call to your main:

```
void app_main(void)
{
   esp_zb_platform_config_t config = {
       .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
       .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
   };
   ESP_ERROR_CHECK(nvs_flash_init());
   /* load Zigbee light_bulb platform config to initialization */
   ESP_ERROR_CHECK(esp_zb_platform_config(&config));
   /* hardware related and device init */
   light_driver_init(LIGHT_DEFAULT_OFF);
   configure_led();
   xTaskCreate(esp_zb_task, "Zigbee_main", 4096, NULL, 5, NULL);
```

```
}
```

- Then add blink_led() and s_led_state = !value to attr_cb() function:

```
void attr_cb(uint8_t status, uint8_t endpoint, uint16_t cluster_id,
uint16_t attr_id, void *new_value)
{
    if (cluster_id == ESP_ZB_ZCL_CLUSTER_ID_ON_OFF) {
        uint8_t value = *(uint8_t *)new_value;
        if (attr_id == ESP_ZB_ZCL_ATTR_ON_OFF_ON_OFF_ID) {
            /* implemented light on/off control */
            ESP_LOGI(TAG, "on/off light set to %hd", value);
            s_led_state = !value;
            blink_led();
            light_driver_set_power((bool)value);
        }
    } else {
        /* Implement some actions if needed when other cluster
changed */
        ESP_LOGI(TAG, "cluster:0x%x, attribute:0x%x changed ",
cluster_id, attr_id);
    }
}
```

- At this point the code should be ready.
- Flash the Esp32-C6-Bug with the modified code(the flashing sequence is the same as in previous example(enter the directory, get_idf, set-target, flash…)). After the code is flashed the device will enter pairing mode.
- Integrate the SkyConnect into your Home Assistant using this guide: https://skyconnect.home-assistant.io/new-zigbee/
- From now on the device should be available in Home Assistant as a simple bulb.

Demo 7: Connecting two Esp32-C6-Bugs with Thread

Intro

Thread is a low-power and low-latency wireless mesh networking protocol. OpenThread is its open source version from google. The OpenThread CLI exposes configuration and management APIs via a command line interface.
This example demonstrates OpenThread CLI functionality by allowing the user to run OpenThread cli commands on Esp32-C6-BUG directly. The example is based on https://github.com/espressif/esp-idf/tree/master/examples/openthread/ot_cli

What do I need to make that?

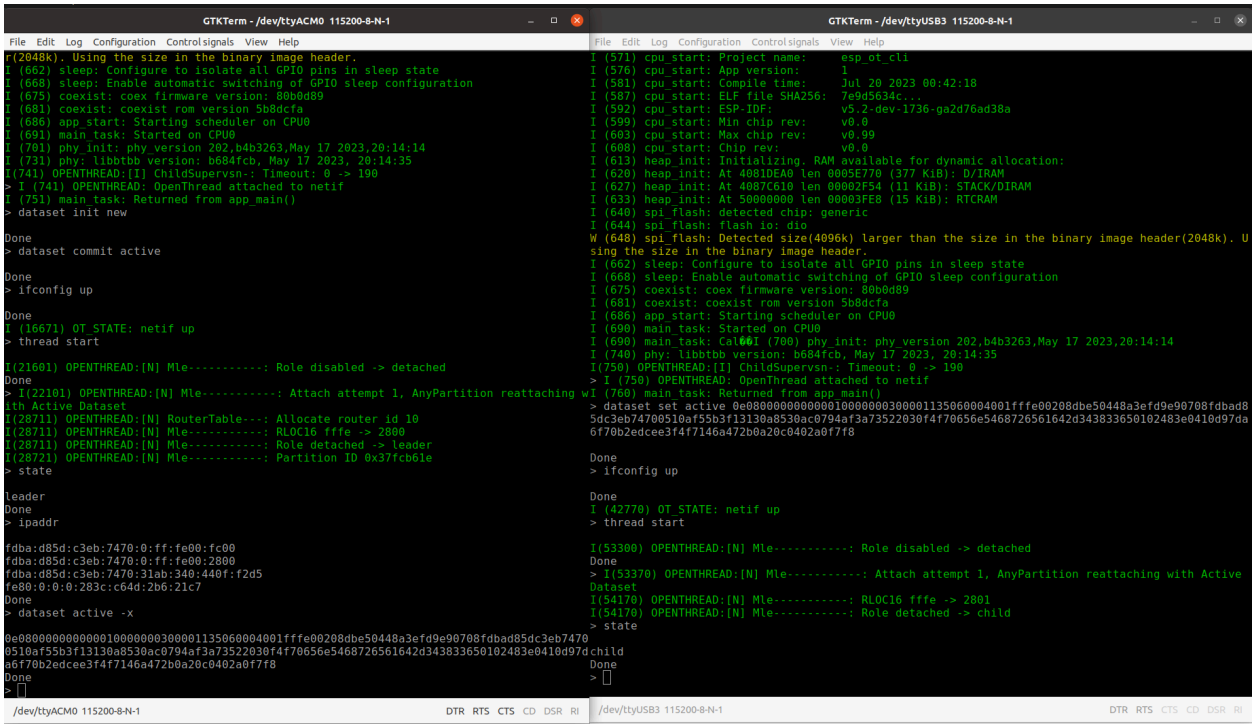
- 2xEsp32-C6-BUG
- 2xUSB Type-C for flashing
- 2xUSB Uart converters for sending commands(1 is enough if you can switch it between chips)

Result

You will have two Esp32-C6-Bugs connected to the same network, one as network leader, second as network child. You can also freely run other commands from OpenThread CLI Api to explore OpenThread yourself.
Some OpenThread commands:

```
>  help
I(7058) OPENTHREAD:[INFO]-CLI-----: execute command: help
bbr
bufferinfo
ccathreshold
channel
child
childip
childmax
childsupervision
childtimeout
coap
contextreusedelay
counters
dataset
delaytimermin
diag
discover
dns
domainname
eidcache
eui64
extaddr
extpanid
factoryreset
...
```

Serial Outputs from two interconnected Esp32-C6-Bugs



How do I do that?

After ESP-IDF is configured you can run the following commands:
1)alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
Use your path to esp-idf/export.sh
2)get_idf
3)cp -r $IDF_PATH/examples/openthread/ot_cli .
4)cd ot_cli
5)idf.py set-target esp32c6

6)connect the first board
7)idf.py flash
8)disconnect the first board, connect the second board
9)idf.py flash
10)go to you favorite Serial port terminal(I used gtkterm), Connect USB-Serial converter to the second board(SERIAL_CONVERTER_TX->BUG_RX,SERIAL_CONVERTER_ RX->BUG_TX)
11)send the commands from the picture above:

1. > dataset init new
2. > dataset commit active
3. > ifconfig up
4. > thread start
5. # After some seconds
6. > state

The output should be :

‘leader
Done’
Then get active dataset using:
> dataset active -x
Save the output
12)Connect the second boad to power and to USB-Serial converter
13)Send the following commands from the picture above:

1. >dataset set active [insert saved dataset from step 11]
2. >ifconfig up
3. >thread start
4. # After some time
5. >state

The output should be:
‘child
Done’

You can also run other commands from the OpenThread API and check their functionality:
https://github.com/openthread/openthread/blob/main/src/cli/README.md

Demo 8: Esp32-C6 OpenThread Router: WiFi+OpenThread

This example demonstrates how to enable both WiFi and Thread on one Esp32-C6 MCU and use it as Wifi to Thread bridge. The example is largely based on https://github.com/espressif/esp-idf/tree/release/v5.1/examples/openthread/ot_br

What do I need to make that?

- 2xEsp32-C6-BUG
- 2xUSB Type-C for flashing
- 2xUSB Uart converters for sending commands(1 is enough if you can switch it between chips)

Result

The first Esp32-C6-Bug will act as a bidirectional bridge and forward traffic Wifi<->Thread.
The second Esp32-C6-Bug will act as an end device connected to the first Esp32-C6-Bug via Thread.
You will be able to ping the second(Thread-connected) Esp32-C6-Bug from your local Wifi.

How do I do that?

After ESP-IDF is configured you can run the following commands:
1)alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
Use your path to esp-idf/export.sh

2)get_idf
3)cp -r $IDF_PATH/examples/openthread/ot_br .
4)cd ot_br
5)idf.py set-target esp32c6

6)connect the first board(bridge)

7)idf.py flash

8)go to you favorite serial port terminal(I used gtkterm), Connect USB-Serial converter to the second board(SERIAL_CONVERTER_TX->BUG_RX,SERIAL_CONVERTER_ RX->BUG_TX)

9)write ‘wifi connect -s SSID -p password’ command to the serial port terminal to connect to WiFi

10)to check the state run ‘wifi state’

11)go to Demo 3 section and create Thread network

12)run the following commands on your PC:

sudo sysctl -w net/ipv6/conf/wlan0/accept_ra=2

sudo sysctl -w net/ipv6/conf/wlan0/accept_ra_rt_info_max_plen=128

Please replace wlan0 with the real name of your Wi-Fi network interface.

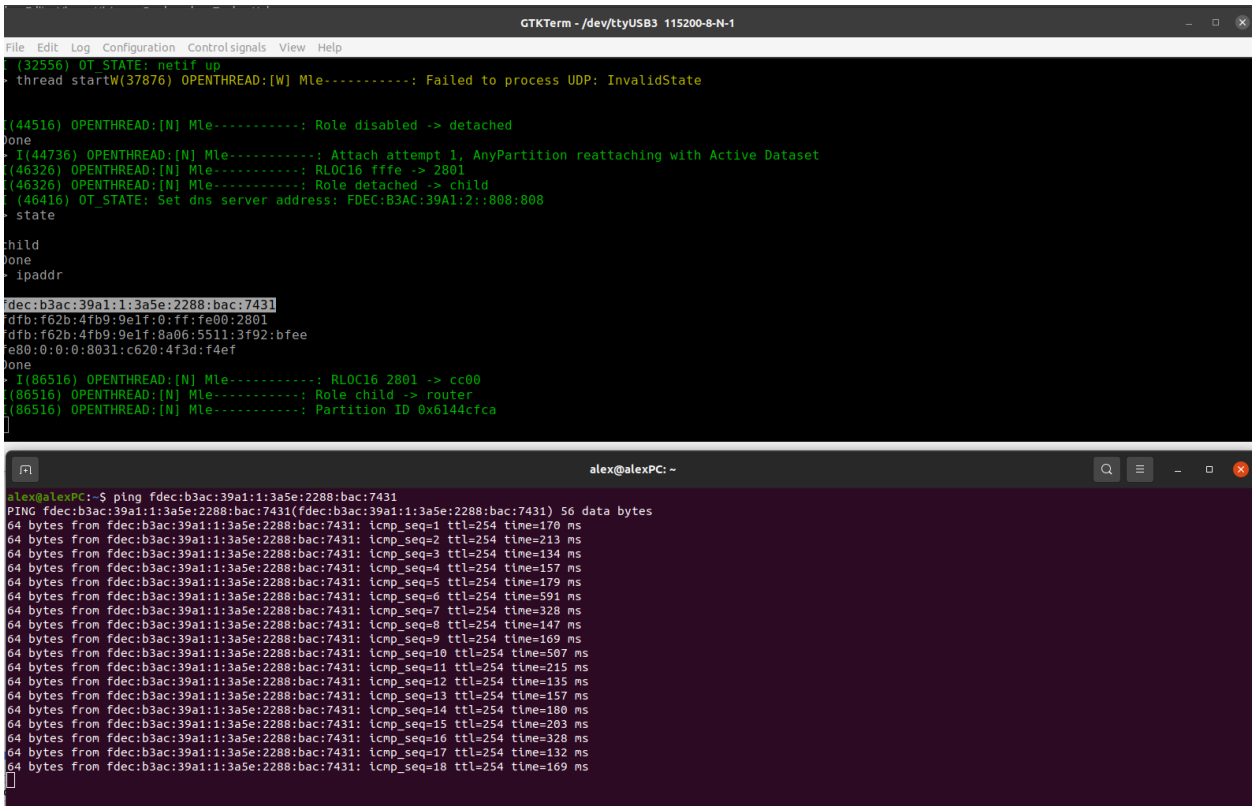
At this point the Bridge should be ready

13)Connect the second Esp32-C6-Bug, flash it with Demo 3 firmware and connect it to the Thread network using guidelines from Demo 3.

14) After the Second board is connected check the Thread ip using ipaddr command and ping it from your pc using

ping ‘IP YOU GOT FROM ipaddr’

The end result is shown in the picture:



Demo 9: Esp32-C6 Zigbee sleepy device

This example demonstrates how to create a zigbee sleepy end device. Most of the time the end device sleeps. After the command is issued from zigbee coordinator node(another Esp32-C6-Big), the device wakes up and handles the command. Zigbee sleepy end device consumption is around 60uA, while it sleeps. The code for this demo can be found here: https://github.com/espressif/esp-zigbee-sdk/tree/main/examples/esp_zigbee_sleep/sleepy_end_device

What do I need to make that?

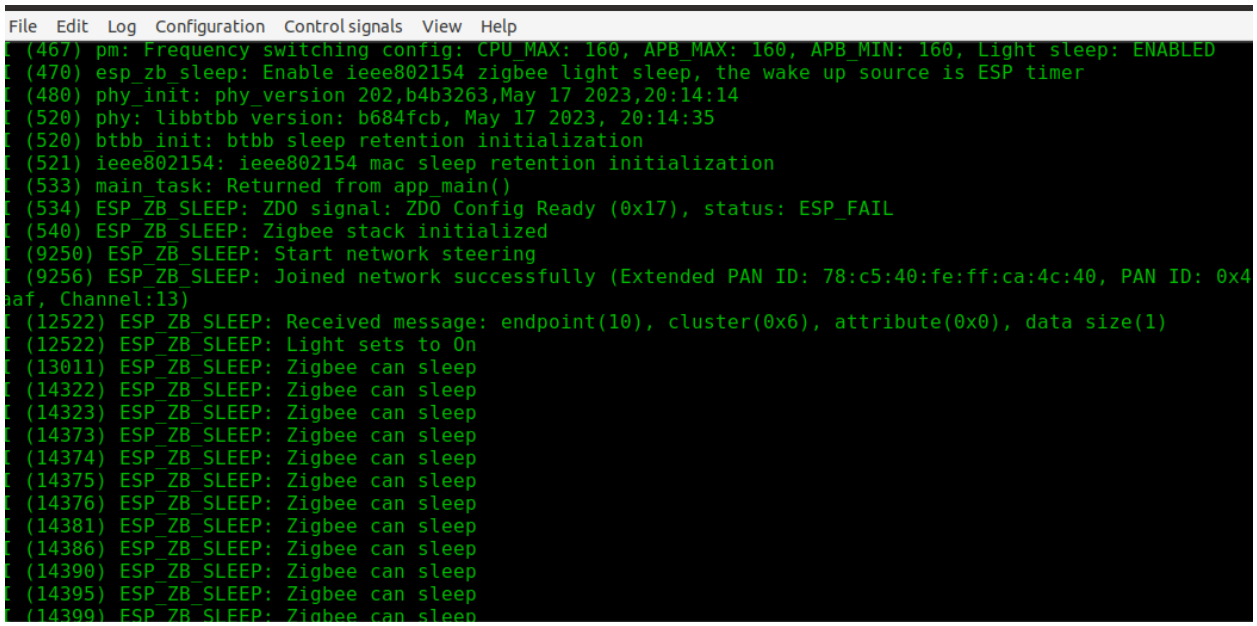
- 2xEsp32-C6-BUG
- 2xUSB Type-C for flashing
- 1x li-ion battery for the sleepy end node

Result

The Esp32-C6-Bug sleepy end device connects to the Zigbee network created by Esp32-C6-Bug acting as coordinator. After you press the button on coordinator device and the sleepy device wakes up, the data are sent to the sleepy end device. The sleepy device receives the data and handles the command.

How do I do that?

1) Configure Zigbee coordinator using demo 2

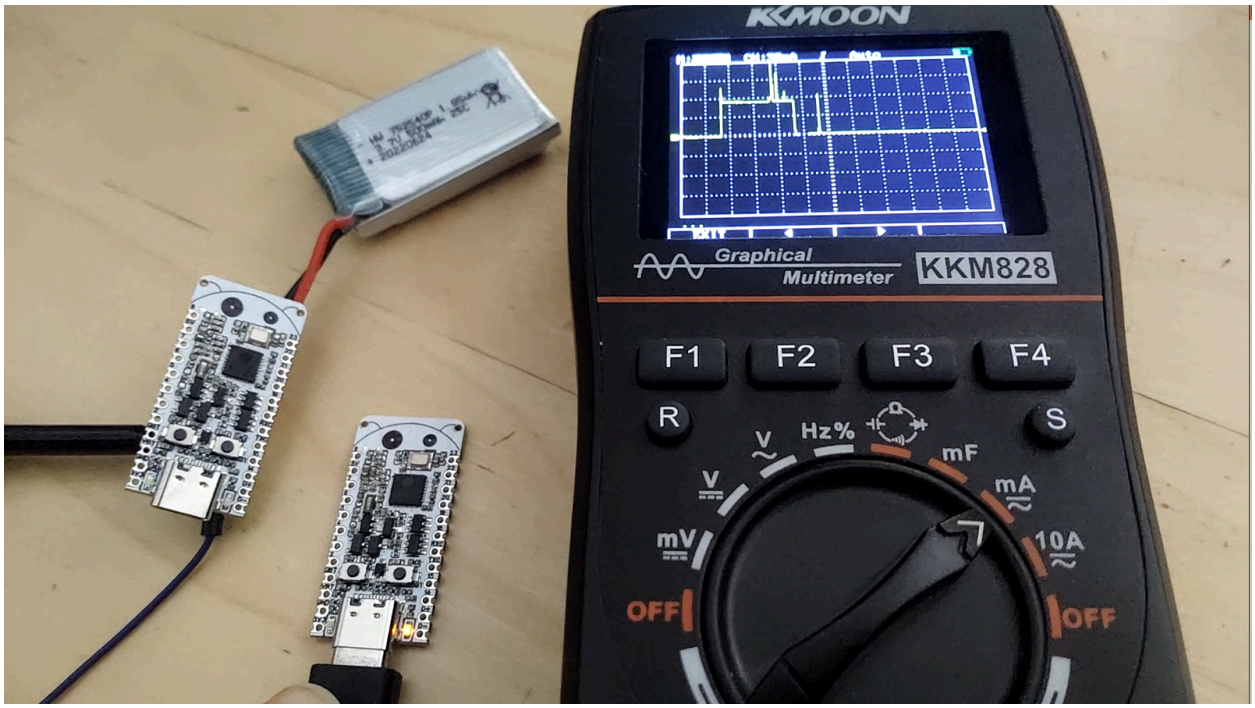
2) Download the Zigbee SDK:
   git clone
   https://github.com/espressif/esp-zigbee-sdk/tree/main/examples/esp_zigbee_sleep/sleepy_end_device
3) Go to examples/esp_zigbee_sleep/sleepy_end_device
4) Run:
   alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
   (Use your path to esp-idf/export.sh)
   get_idf
   idf.py set-target esp32c6
   Idf.py flash
5) After the Zigbee coordinator created the network(Check it's serial output) reset the sleepy device, it should connect to it.
6) 

7) When you push the button on your Zigbee coordinator you should see output like this on sleepy device

8)

```
File  Edit  Log  Configuration  Control signals  View  Help
I (15840) ESP_ZB_SLEEP: Zigbee can sleep
I (15845) ESP_ZB_SLEEP: Zigbee can sleep
I (15849) ESP_ZB_SLEEP: Zigbee can sleep
I (15854) ESP_ZB_SLEEP: Zigbee can sleep
I (15859) ESP_ZB_SLEEP: Zigbee can sleep
I (15863) ESP_ZB_SLEEP: Zigbee can sleep
I (15868) ESP_ZB_SLEEP: Zigbee can sleep
I (15872) ESP_ZB_SLEEP: Zigbee can sleep
I (15877) ESP_ZB_SLEEP: Zigbee can sleep
I (15882) ESP_ZB_SLEEP: Zigbee can sleep
I (15886) ESP_ZB_SLEEP: Zigbee can sleep
I (15891) ESP_ZB_SLEEP: Zigbee can sleep
I (15895) ESP_ZB_SLEEP: Zigbee can sleep
I (15900) ESP_ZB_SLEEP: Zigbee can sleep
I (15905) ESP_ZB_SLEEP: Zigbee can sleep
I (15909) ESP_ZB_SLEEP: Zigbee can sleep
I (15914) ESP_ZB_SLEEP: Zigbee can sleep
I (17932) ESP_ZB_SLEEP: Received message: endpoint(10), cluster(0x6), attr
ibute(0x0), data size(1)
I (17932) ESP_ZB_SLEEP: Light sets to Off
I (18376) ESP_ZB_SLEEP: Zigbee can sleep
I (23227) ESP_ZB_SLEEP: Received message: endpoint(10), cluster(0x6), attr
ibute(0x0), data size(1)
I (23277) ESP_ZB_SLEEP: Light sets to On
I (23726) ESP_ZB_SLEEP: Zigbee can sleep
```

After the device wakes up you should see the current spike from 60uA to 60mA



Manufacturing Plan

The manufacturing will be handled by JLCPCB,

Fulfillment & Logistics

After the manufacturing is complete, the boards will be placed to ESD bugs together with pin headers and sent to Crowdsupply. After this, the boards will be distributed worldwide.

Risks & Challenges

- Arduino support - unknown release date