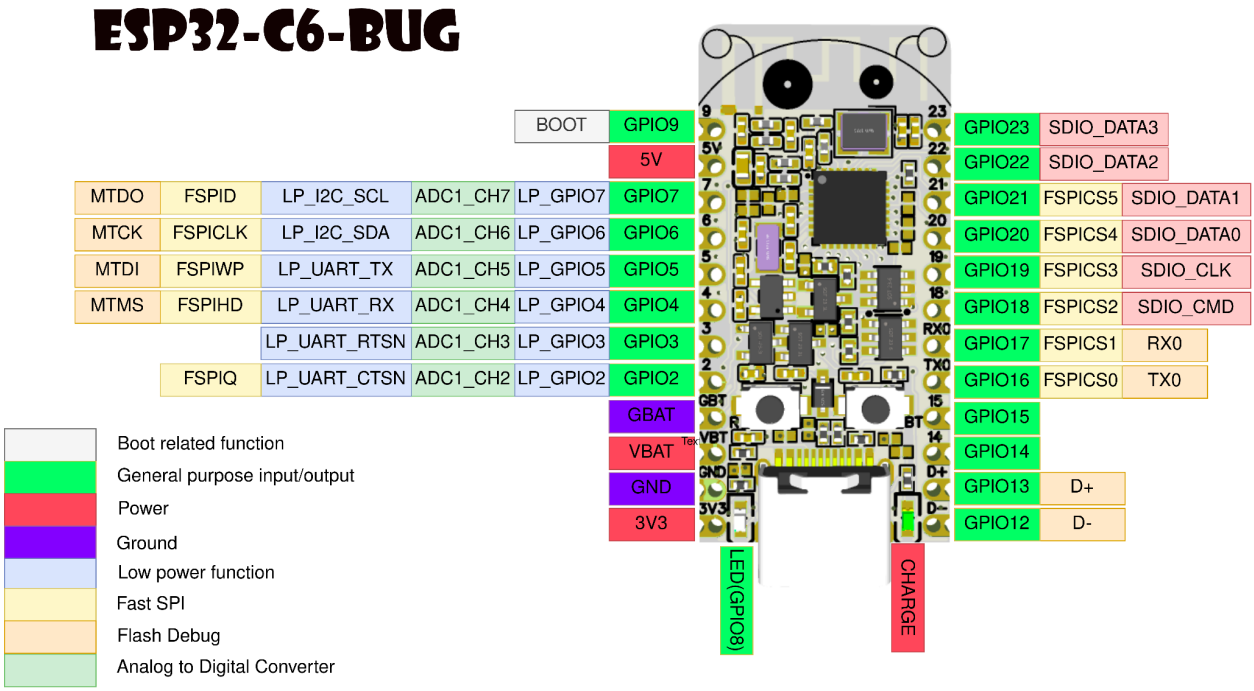
What is it?

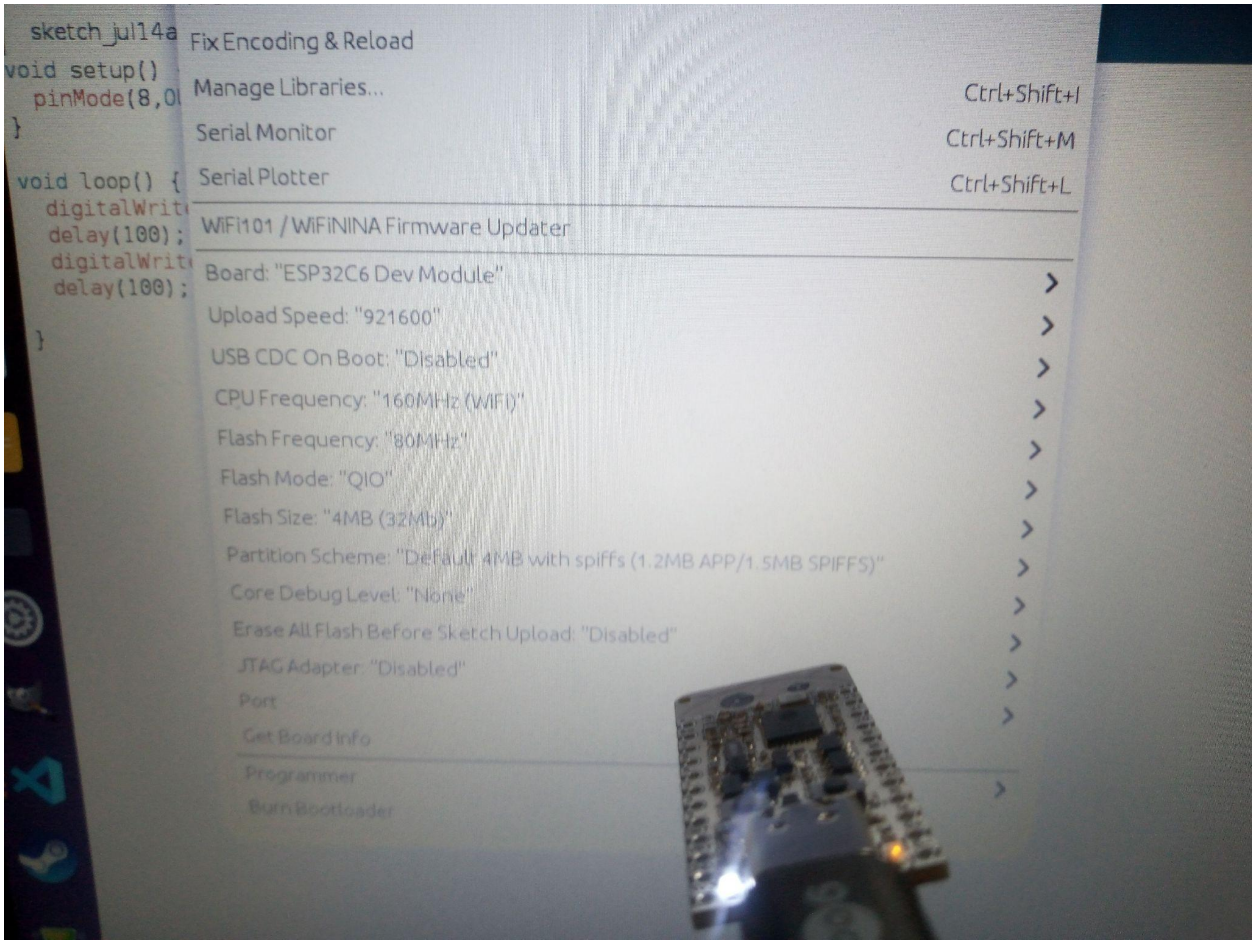
The ESP32-C6-Bug is a tiny development board based on the latest C6 MCU of the ESP32 family with reduced power consumption and battery support. Supported wireless protocols include Wi-Fi 6, BLE 5, Thread, Matter and Zigbee.

Pinout and supported frameworks



Picture 1: Pinout and pin descriptions

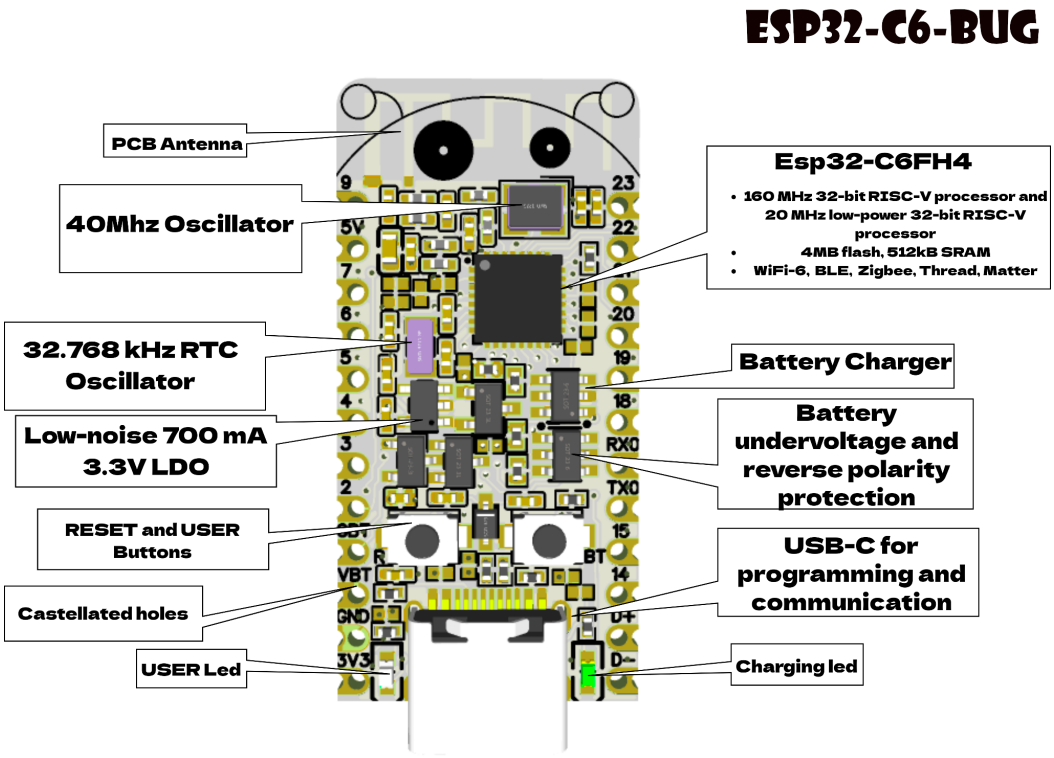
Supported Platforms: Arduino IDE, ESP-IDF



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



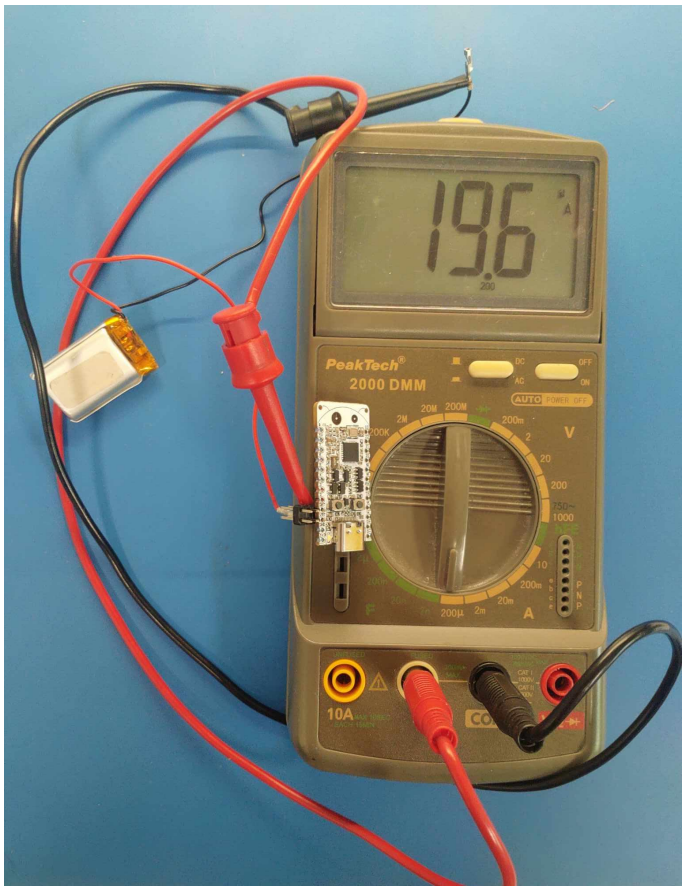
Picture 2: Features Diagram

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.

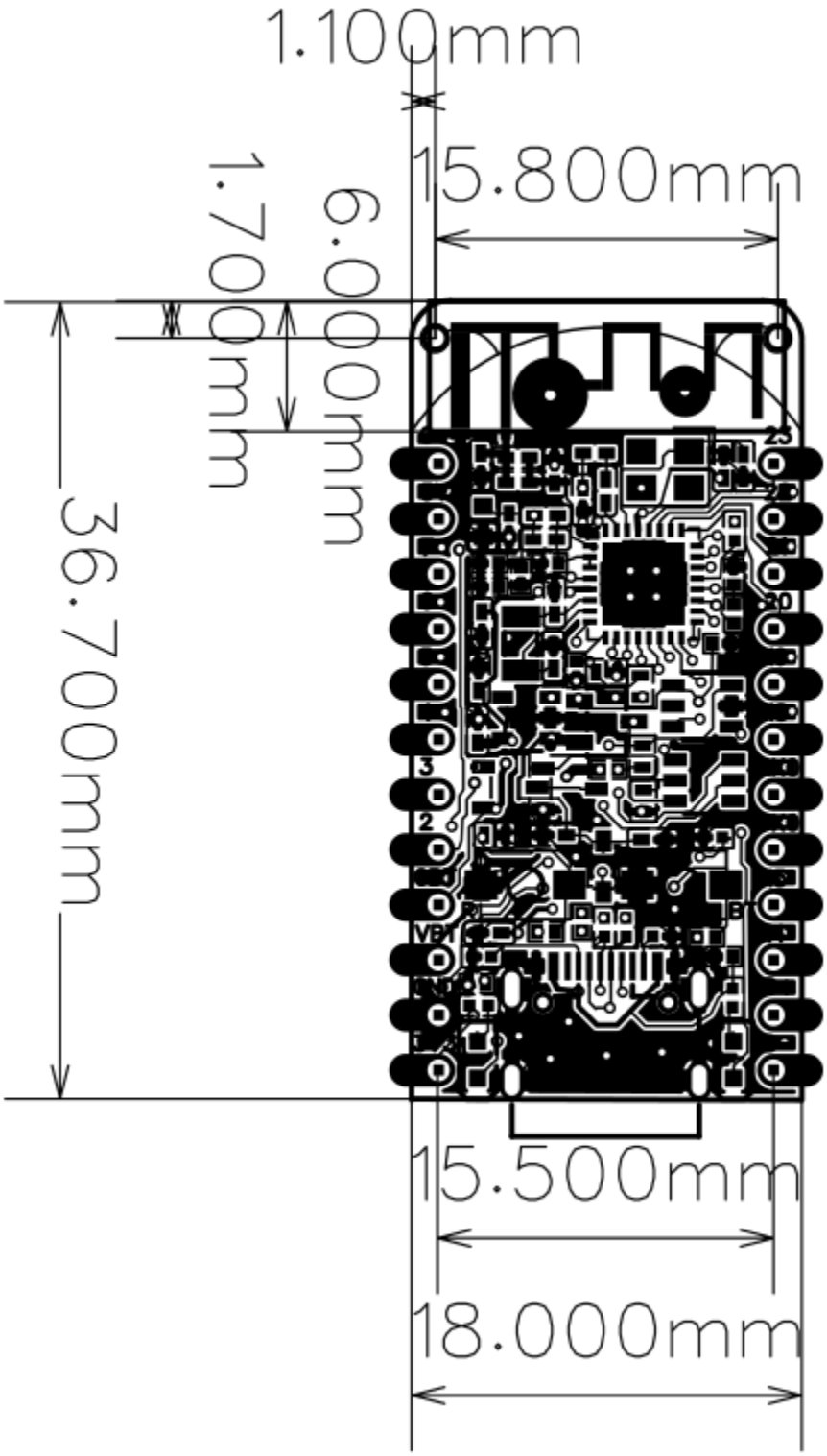
Board features

- 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

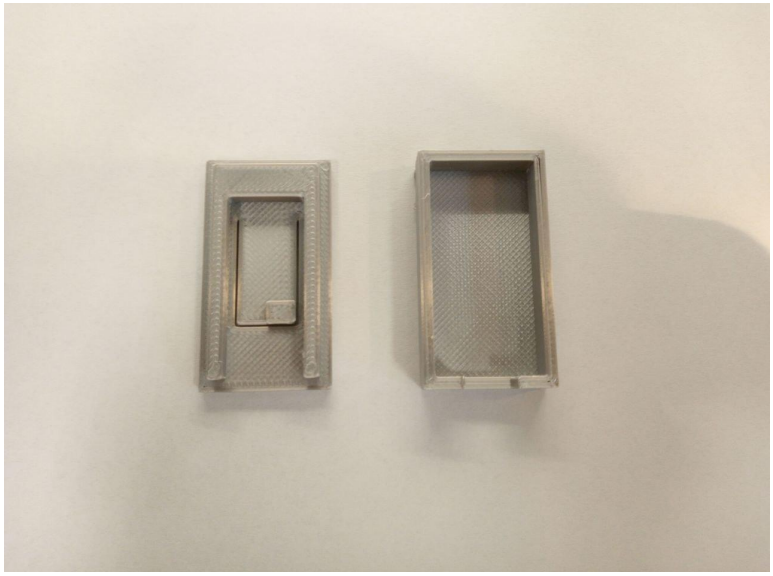


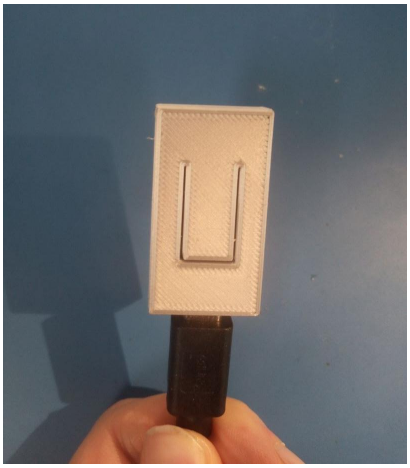
Existing solutions comparison

	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

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 1: BLE advertising

Intro

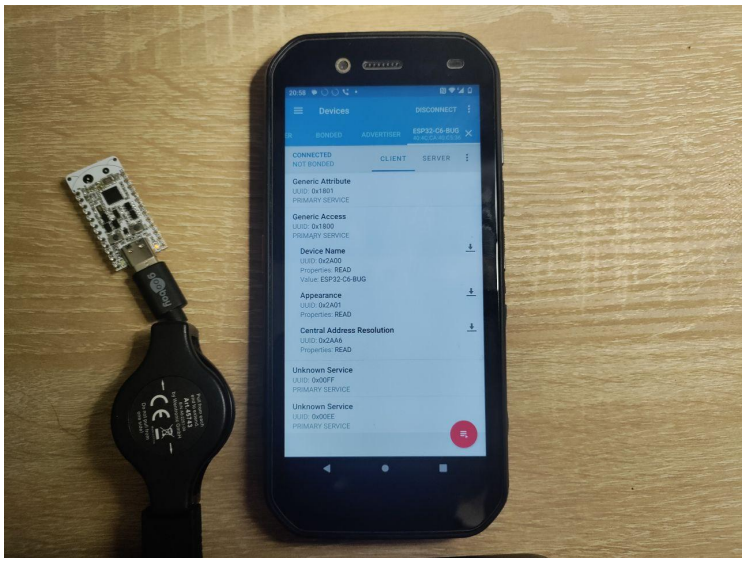
This example demonstrates how to connect Esp32-C6-Bug to your phone via BLE. It is largely based on https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble. After the successful connection it is possible to read data from Esp32-C6-Bug. This example uses ESP-IDF as a framework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

You will need:

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

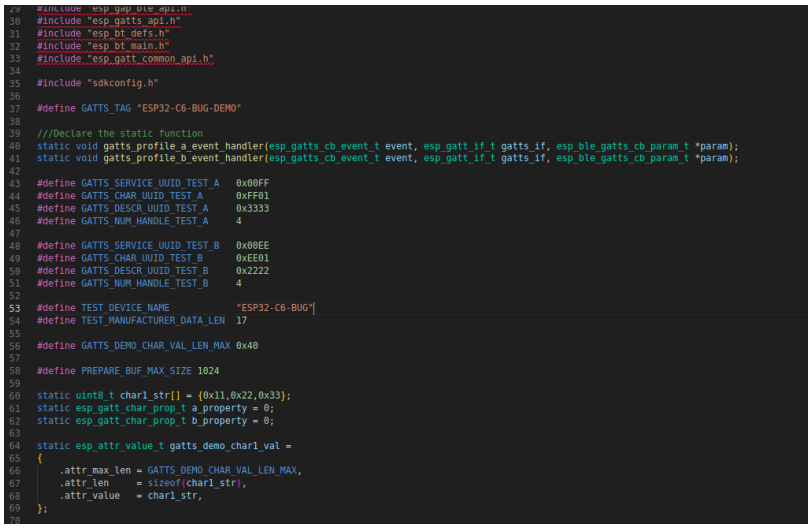
Result

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_gap_ble_api.h"
#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 2: 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 .

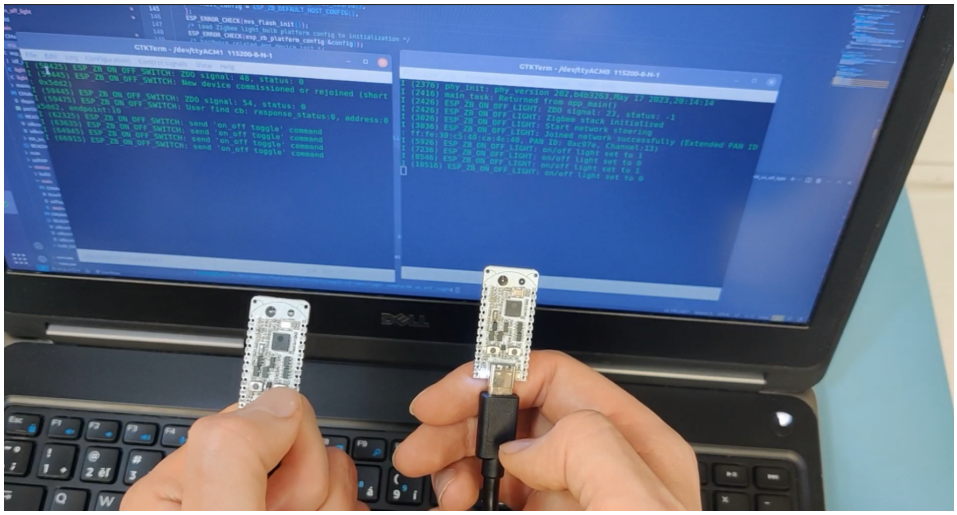
You will need:

- 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

Demo 3: 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

You will need:

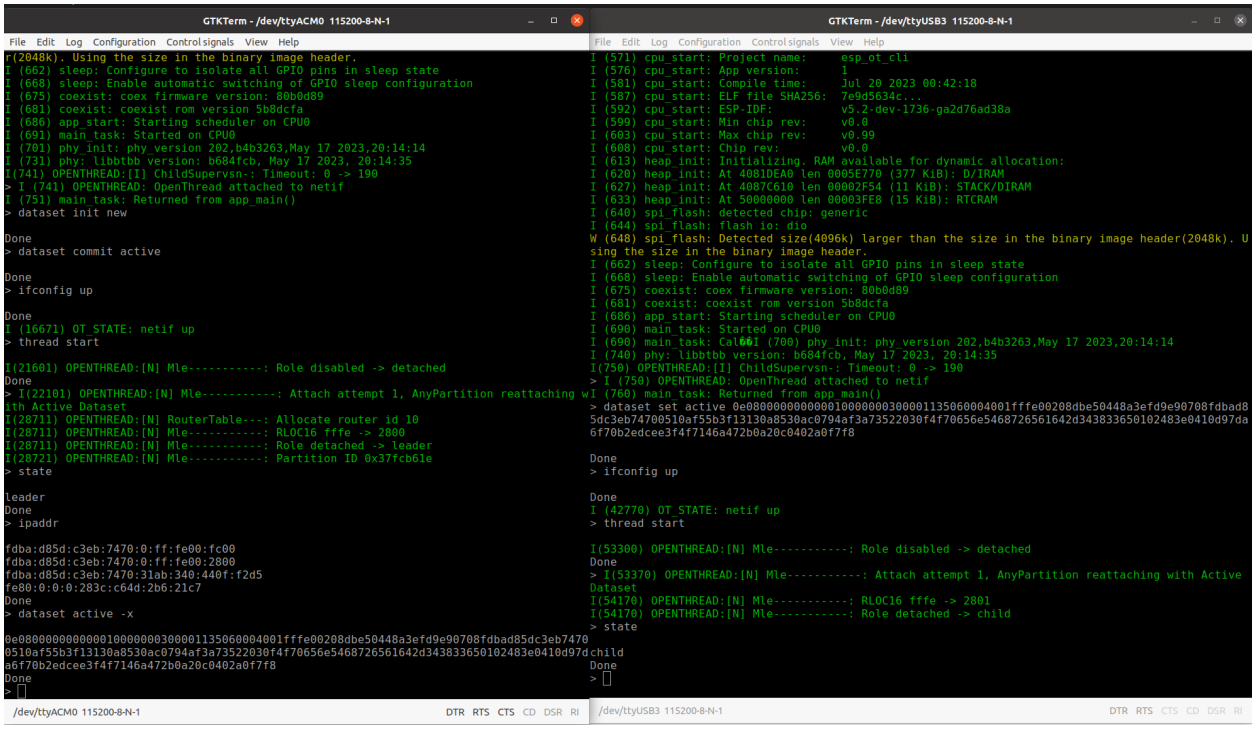
- 2xEsp32-C6-BUG
- 2xUSB Type-C for flashing
- 2xUSB Uart converters for sending commands(1 is enough if you can switch it between chips)

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 4: 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

You will need:

- 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.

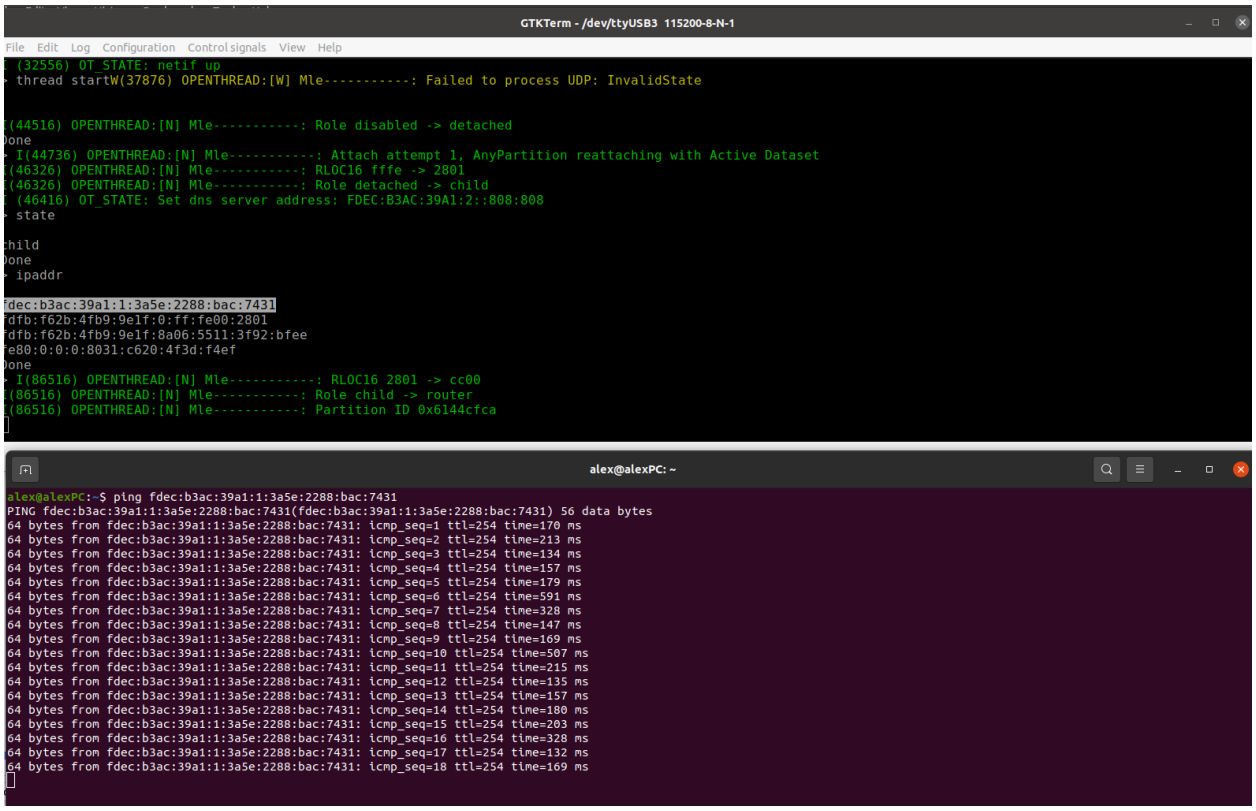
At this point the Bridge should be ready

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 5: 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

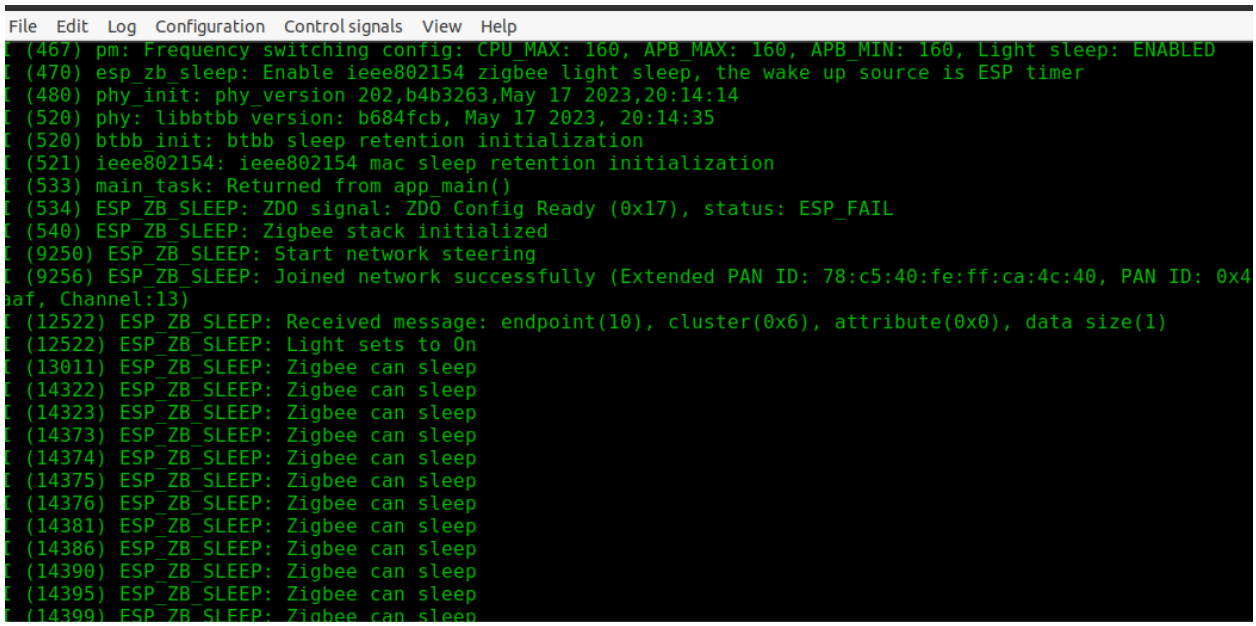
You will need:

- 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?

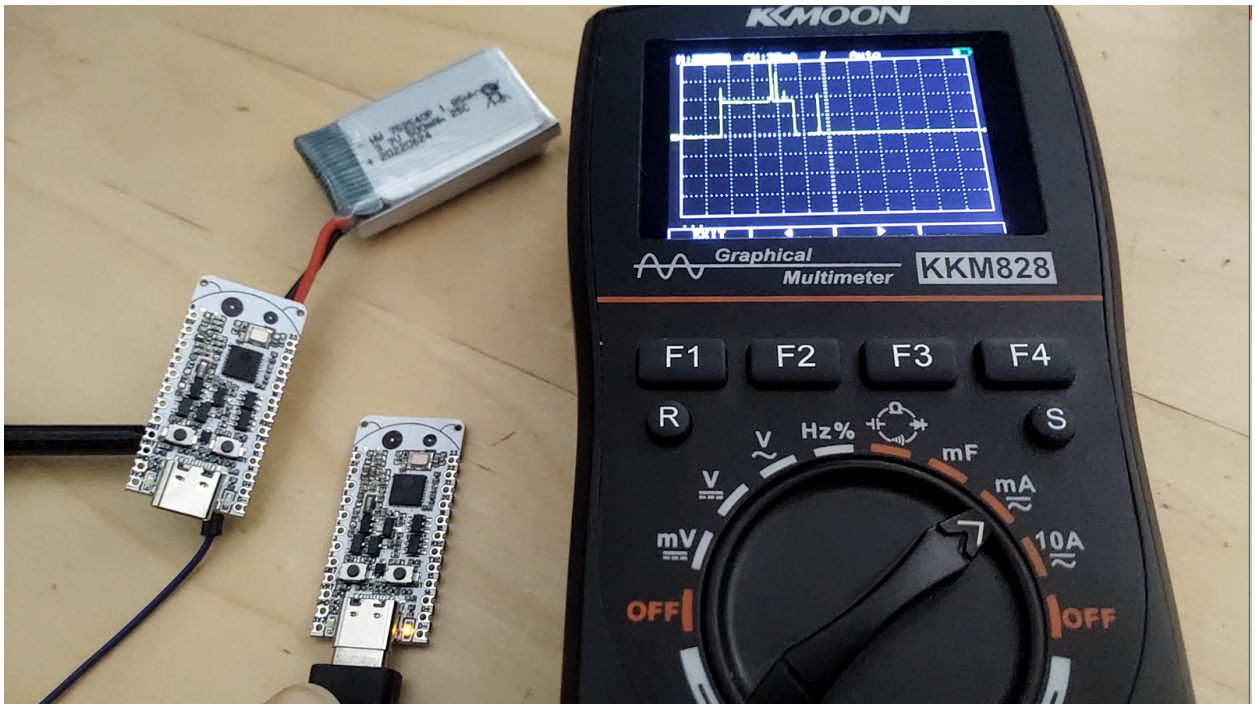
1) Configure Zigbee coordinator using demo 2
2) Download the Zigbee SDK:
   git clone
   https://github.com/espressif/esp-zigbee-sdk/tree/main/examples/esp_zigbee_sleep/sleepy_end_device
3) Go to examples/esp_zigbee_sleep/sleepy_end_device
4) Run:
   alias get_idf='. /home/alex/Documents/programs/espidf/esp-idf/export.sh'
   (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