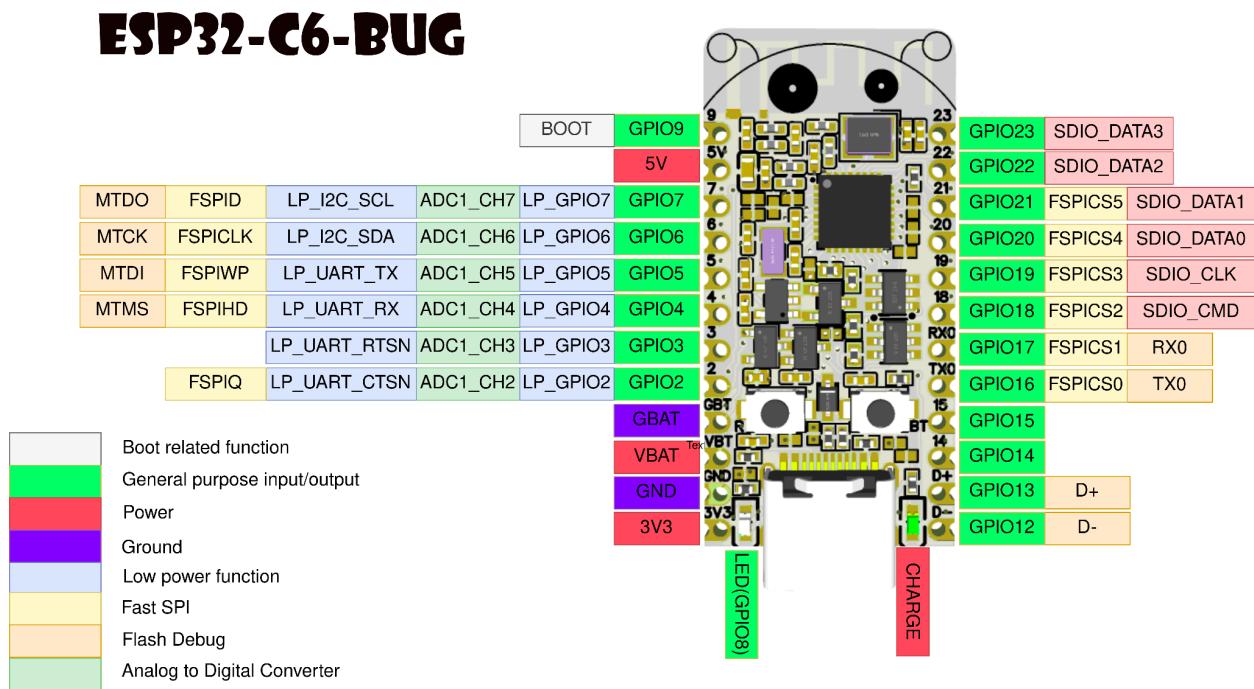


What is it?

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.

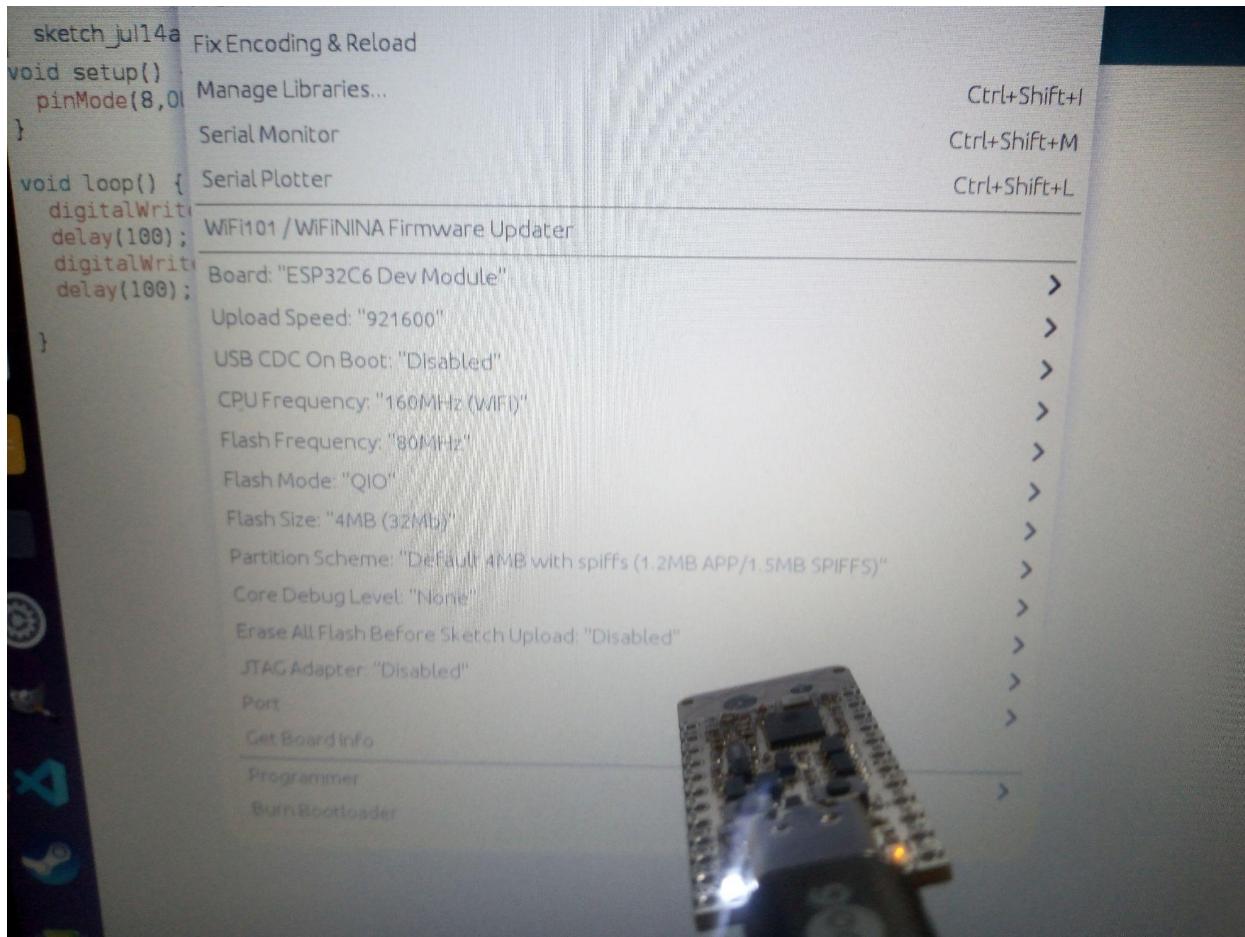
Pinout and supported frameworks



Picture 1: Pinout and pin descriptions

Supported Platforms: Arduino IDE, ESP-IDF





Programming Esp32-C6 in Arduino

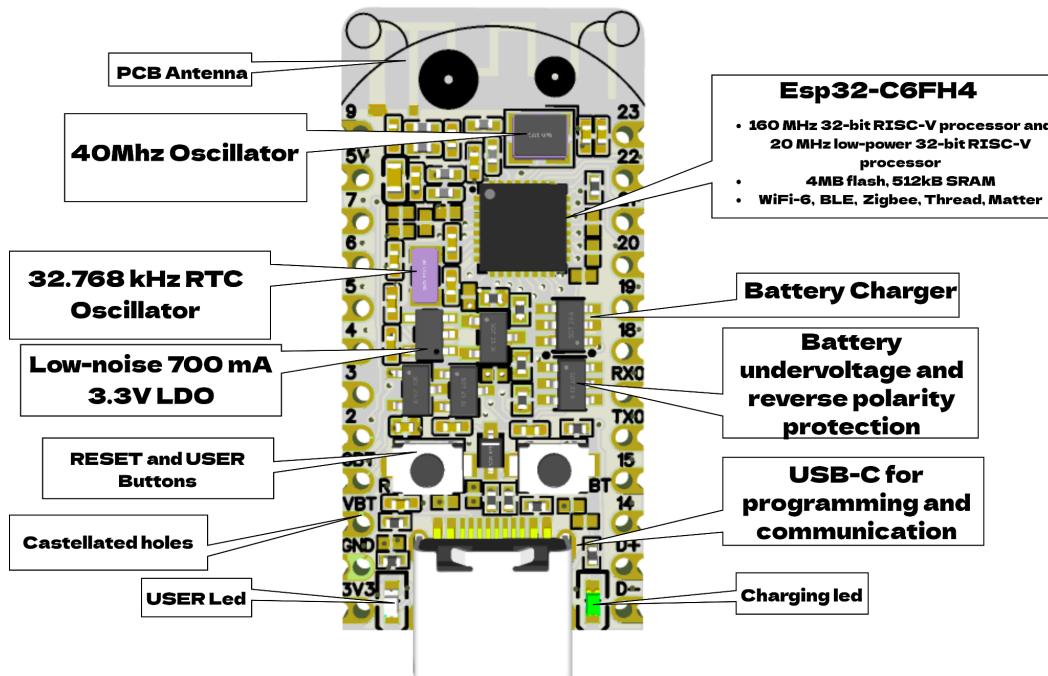
Following this tutorial you can install custom Arduino Core that supports Esp32-C6:

<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

ESP32-C6-BUG



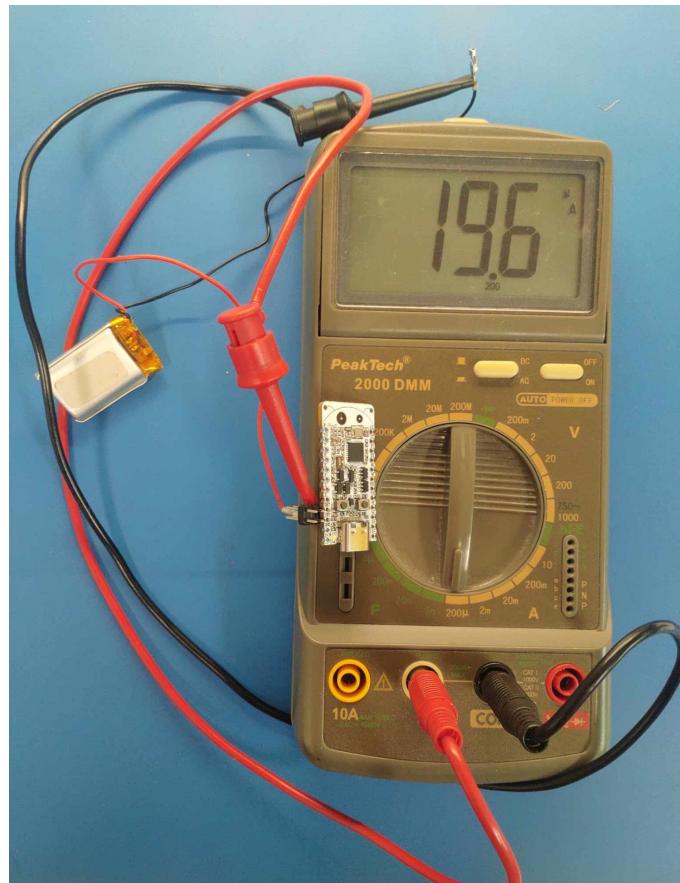
Picture 2: Features Diagram

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.

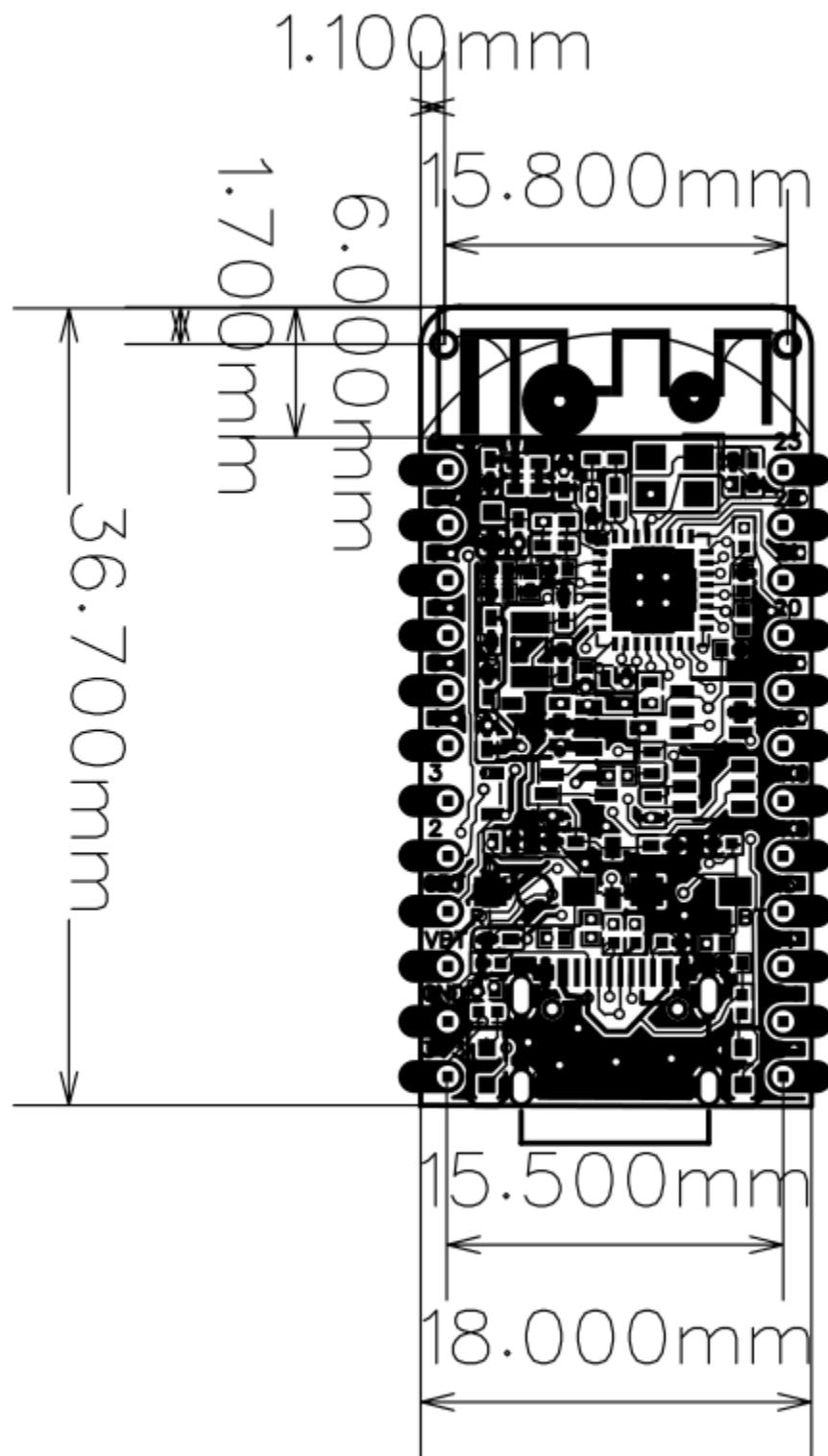
Board features

- 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



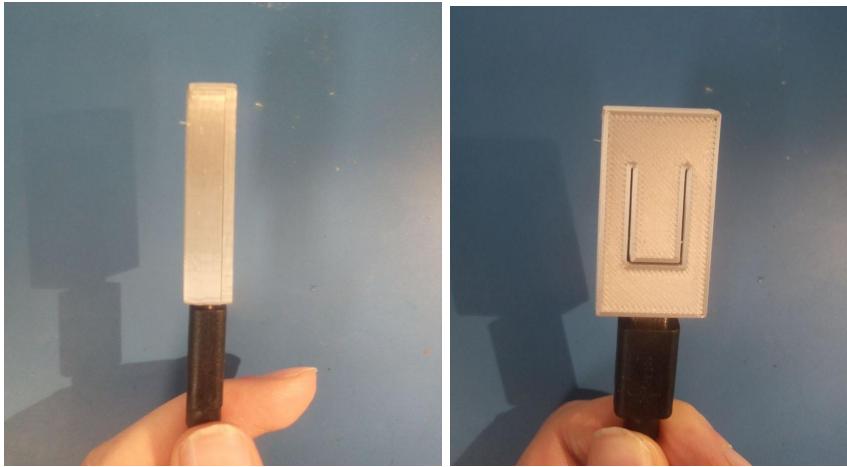
Existing solutions comparison

	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_gat_eway
- 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

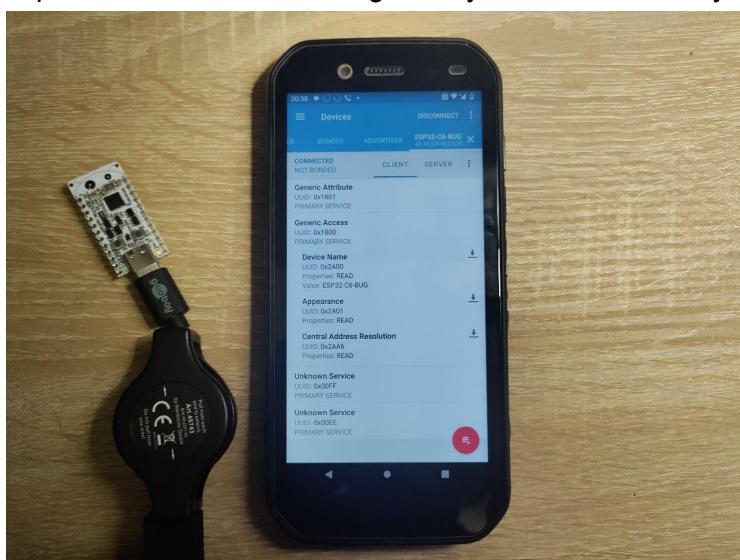
This example demonstrates how to connect Esp32-C6-Bug to your phone via BLE. It is largely based on <https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble>. After the successful connection it is possible to read data from Esp32-C6-Bug. This example uses ESP-IDF as a framework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

```
29 #include "esp_gatt_ue_api.h"
30 #include "esp_gatts_api.h"
31 #include "esp_bt_defs.h"
32 #include "esp_bt_main.h"
33 #include "esp_gatt_common_api.h"
34
35 #include "sdkconfig.h"
36
37 #define GATT_TAG "ESP32-C6-BUG-DEMO"
38
39 //Declare the static function
40 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);
41 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);
42
43 #define GATT_SERVICE_UUID_TEST_A 0x00FF
44 #define GATT_CHAR_UUID_TEST_A 0xFF01
45 #define GATT_DESCR_UUID_TEST_A 0x3333
46 #define GATT_NUM_HANDLE_TEST_A 4
47
48 #define GATT_SERVICE_UUID_TEST_B 0x00EE
49 #define GATT_CHAR_UUID_TEST_B 0xEE01
50 #define GATT_DESCR_UUID_TEST_B 0x2222
51 #define GATT_NUM_HANDLE_TEST_B 4
52
53 #define TEST_DEVICE_NAME "ESP32-C6-BUG"
54 #define TEST_MANUFACTURER_DATA_LEN 17
55
56 #define GATT_DEMO_CHAR_VAL_MAX 0x40
57
58 #define PREPARE_BUF_MAX_SIZE 1024
59
60 static uint8_t char1_str[] = {0x11,0x22,0x33};
61 static esp_gatt_char_prop_t a_property = 0;
62 static esp_gatt_char_prop_t b_property = 0;
63
64 static esp_attr_value_t gatts_demo_char1_val =
65 {
66     .attr_max_len = GATT_DEMO_CHAR_VAL_LEN_MAX,
67     .attr_len = sizeof(char1_str),
68     .attr_value = char1_str,
69 };
70
```

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/espidf/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
ccatthreshold
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 nv 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 board 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/espidf/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:

GTKTerm - /dev/ttyUSB3 115200-8-N-1

```

File Edit Log Configuration Control signals View Help
: (32556) OT STATE: netif up
-> thread startW(37876) OPENTHREAD:[W] Mle-----: Failed to process UDP: InvalidState

:(44516) OPENTHREAD:[N] Mle-----: Role disabled -> detached
Done
-> I(44736) OPENTHREAD:[N] Mle-----: Attach attempt 1, AnyPartition reattaching with Active Dataset
:(46326) OPENTHREAD:[N] Mle-----: RLOC16 fffe -> 2801
:(46326) OPENTHREAD:[N] Mle-----: Role detached -> child
-> (46416) OT_STATE: Set dns server address: FDEC:B3AC:39A1:2::808:808
-> state

child
Done
-> ipaddr
fdec:b3ac:39a1:1:3a5e:2288:bac:7431
dfb:f62b:4fb9:9e1f:0:ff:fe00:2801
dfb:f62b:4fb9:9e1f:8a06:5511:3f92:bfee
e80:0:0:0:8031:c620:4f3d:f4ef
Done
-> I(86516) OPENTHREAD:[N] Mle-----: RLOC16 2801 -> cc00
:(86516) OPENTHREAD:[N] Mle-----: Role child -> router
:(86516) OPENTHREAD:[N] Mle-----: Partition ID 0x6144cfca
[]

alex@alexPC: ~
alex@alexPC:~$ ping fdec:b3ac:39a1:1:3a5e:2288:bac:7431
PING fdec:b3ac:39a1:1:3a5e:2288:bac:7431(fdec:b3ac:39a1:1:3a5e:2288:bac:7431) 56 data bytes
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=1 ttl=254 time=170 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=2 ttl=254 time=213 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=3 ttl=254 time=134 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=4 ttl=254 time=157 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=5 ttl=254 time=179 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=6 ttl=254 time=591 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=7 ttl=254 time=323 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=8 ttl=254 time=147 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=9 ttl=254 time=169 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=10 ttl=254 time=507 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=11 ttl=254 time=215 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=12 ttl=254 time=135 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=13 ttl=254 time=157 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=14 ttl=254 time=180 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=15 ttl=254 time=203 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=16 ttl=254 time=328 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=17 ttl=254 time=132 ms
64 bytes from fdec:b3ac:39a1:1:3a5e:2288:bac:7431: icmp_seq=18 ttl=254 time=169 ms
[]
```