Examples

Link to GitHub repository

- Hello World
- RGB LED
- Switch
- ADC
- RS232
- RS485
- I2C Scanner
- Ethernet
- WiFi
- More Examples

Hello World

ESP32 Open IoT and IIoT Gateways (P01 & P02)

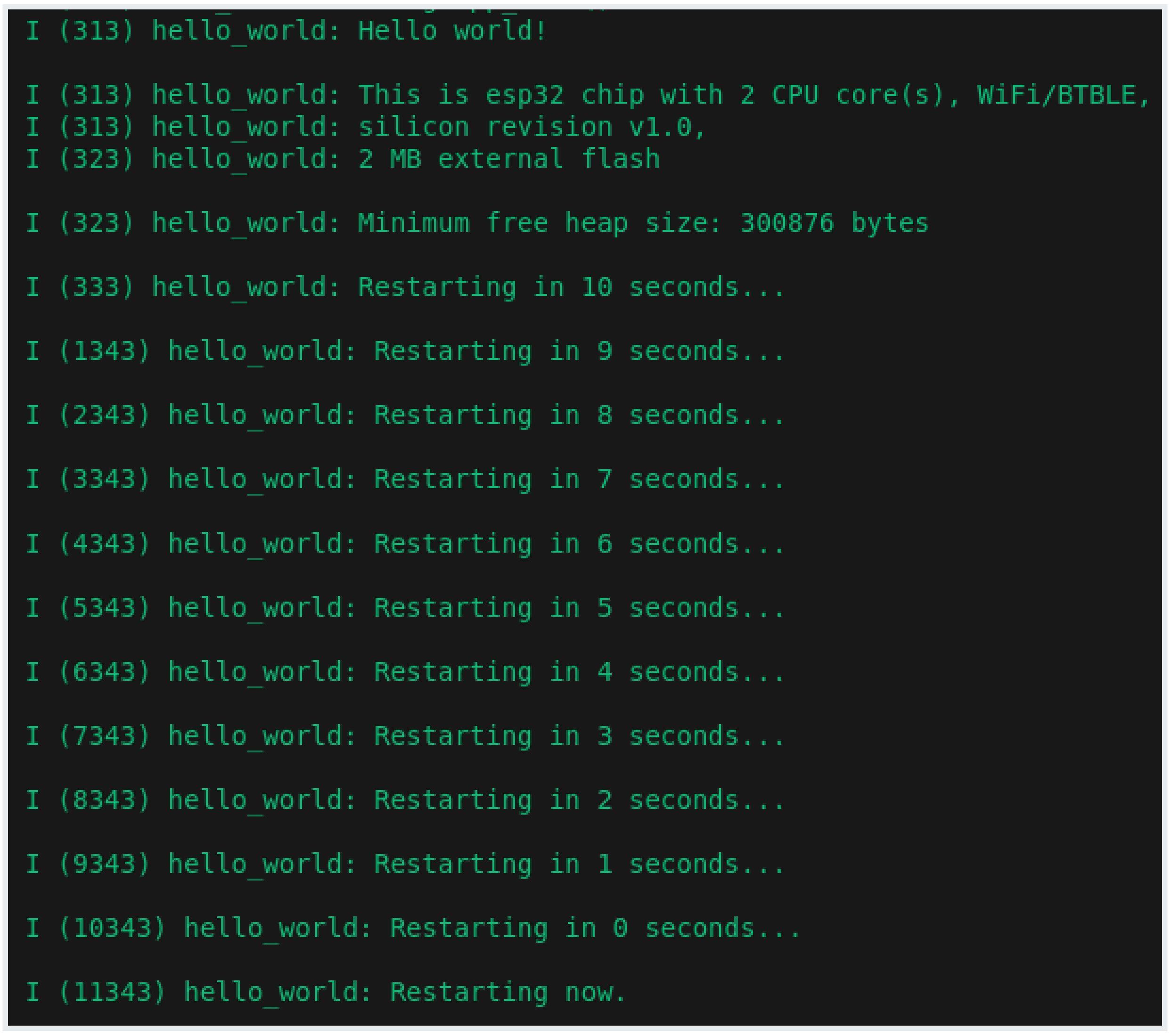
Link to repositories:

- C
- MicroPython

Description

This example use UART0 in the MCU which is reserved for the console. Simple string and device info is printed on the console and then the MCU restarts after 10 seconds.

Console output



```
I (313) hello_world: Hello world!

I (313) hello_world: This is esp32 chip with 2 CPU core(s), WiFi/BTBLE,
I (313) hello_world: silicon revision v1.0,
I (323) hello_world: 2 MB external flash

I (323) hello_world: Minimum free heap size: 300876 bytes

I (333) hello_world: Restarting in 10 seconds...

I (1343) hello_world: Restarting in 9 seconds...

I (2343) hello_world: Restarting in 8 seconds...

I (3343) hello_world: Restarting in 7 seconds...

I (4343) hello_world: Restarting in 6 seconds...

I (5343) hello_world: Restarting in 5 seconds...

I (6343) hello_world: Restarting in 4 seconds...

I (7343) hello_world: Restarting in 3 seconds...

I (8343) hello_world: Restarting in 2 seconds...

I (9343) hello_world: Restarting in 1 seconds...

I (10343) hello_world: Restarting in 0 seconds...

I (11343) hello_world: Restarting now.
```

RGB LED

ESP32 Open IoT and IIoT Gateways (P01 & P02)

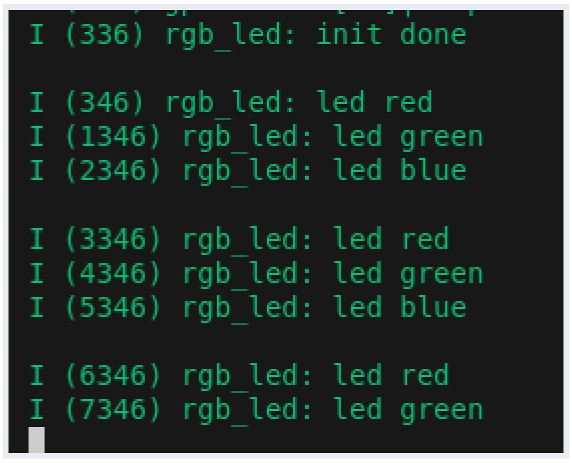
Link to repositories:

- C
- MicroPython

Description

This example shows how to use the programmable RGB LED via the 74HC595BQ shift register. After flashing the device, the RGB LED should change its color between red, green and blue. It can be used to check if the device was connected and flashed properly.

Console output



```
I (336) rgb_led: init done

I (346) rgb_led: led red
I (1346) rgb_led: led green
I (2346) rgb_led: led blue

I (3346) rgb_led: led red
I (4346) rgb_led: led green
I (5346) rgb_led: led blue

I (6346) rgb_led: led red
I (7346) rgb_led: led green
```

Switch

ESP32 Open IoT and IIoT Gateways (P01 & P02)

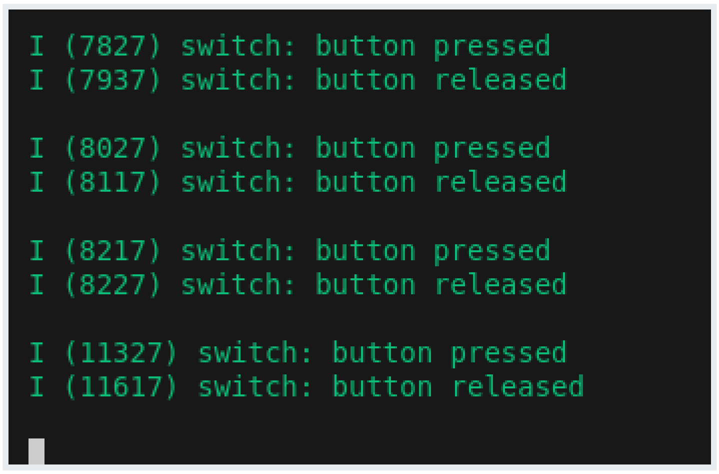
Link to repositories:

- C
- MicroPython

Description

This example shows how to use the button located on the board. There is a simple interrupt handler implemented with a queue used to send a parameter from the handler to a main loop. The board should react on every button press or release with a communicate on the console log.

Console output



```
I (7827) switch: button pressed
I (7937) switch: button released

I (8027) switch: button pressed
I (8117) switch: button released

I (8217) switch: button pressed
I (8227) switch: button released

I (11327) switch: button pressed
I (11617) switch: button released
```

ADC

ESP32 Open IoT and IIoT Gateways (P01 & P02)

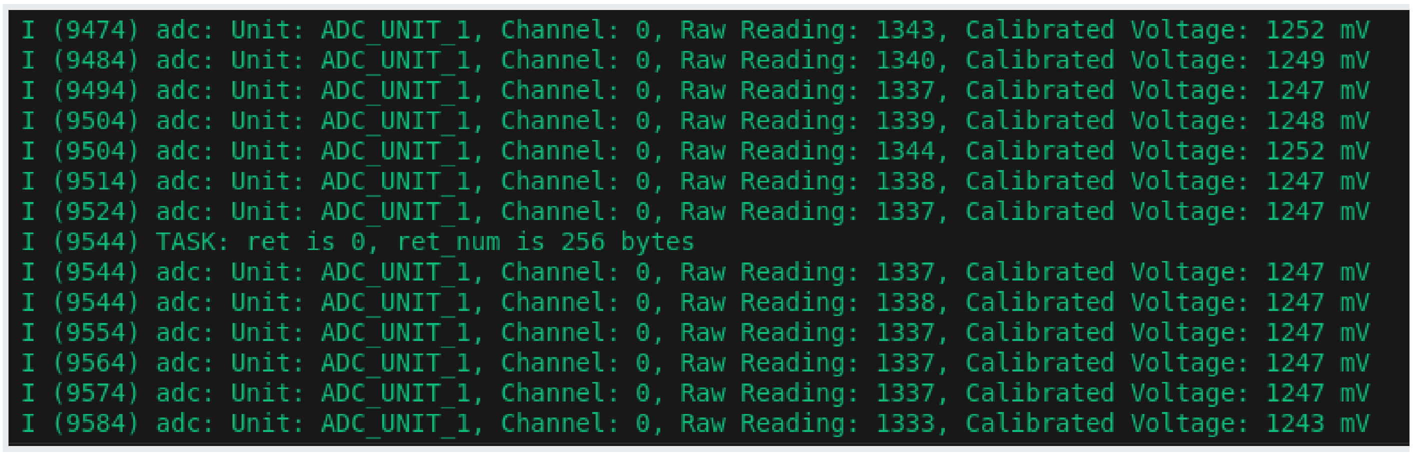
Link to repositories:

- C
- MicroPython

Description

This example shows how to use the ADC (analog-digital converter) located on the board. In the beginning of the program there is an ADC calibration performed. Thanks to that a raw reading can be converted to a corresponding voltage level.

Console output



```
I (9474) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1343, Calibrated Voltage: 1252 mV
I (9484) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1340, Calibrated Voltage: 1249 mV
I (9494) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9504) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1339, Calibrated Voltage: 1248 mV
I (9504) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1344, Calibrated Voltage: 1252 mV
I (9514) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1338, Calibrated Voltage: 1247 mV
I (9524) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9544) TASK: ret is 0, ret_num is 256 bytes
I (9544) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9544) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1338, Calibrated Voltage: 1247 mV
I (9554) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9564) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9574) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1337, Calibrated Voltage: 1247 mV
I (9584) adc: Unit: ADC_UNIT_1, Channel: 0, Raw Reading: 1333, Calibrated Voltage: 1243 mV
```

RS232

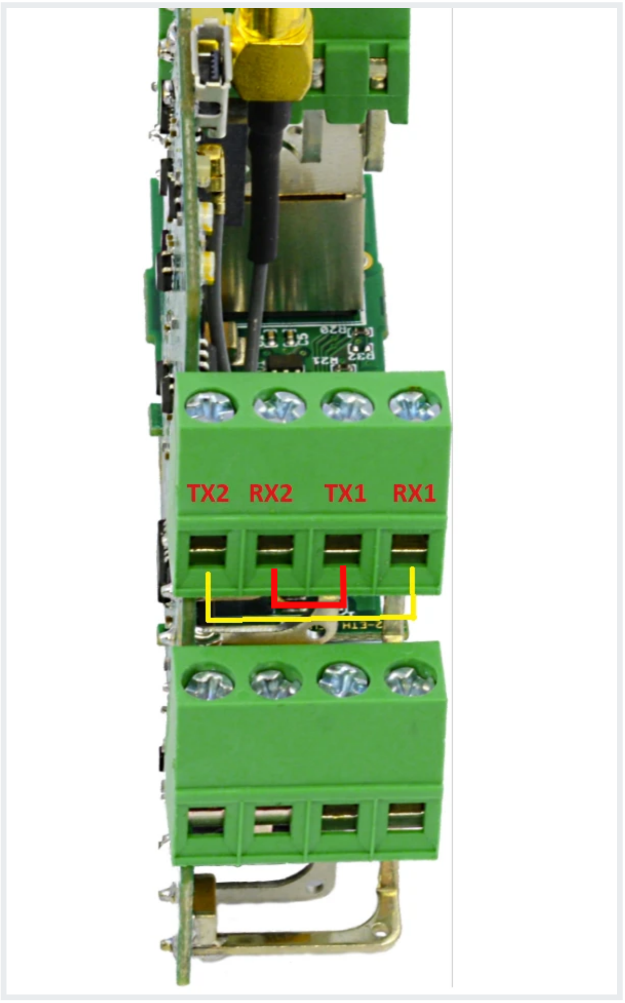
ESP32 Open IoT and IIoT Gateways (P01 & P02)

Link to repositories:

- C
- MicroPython

Connection

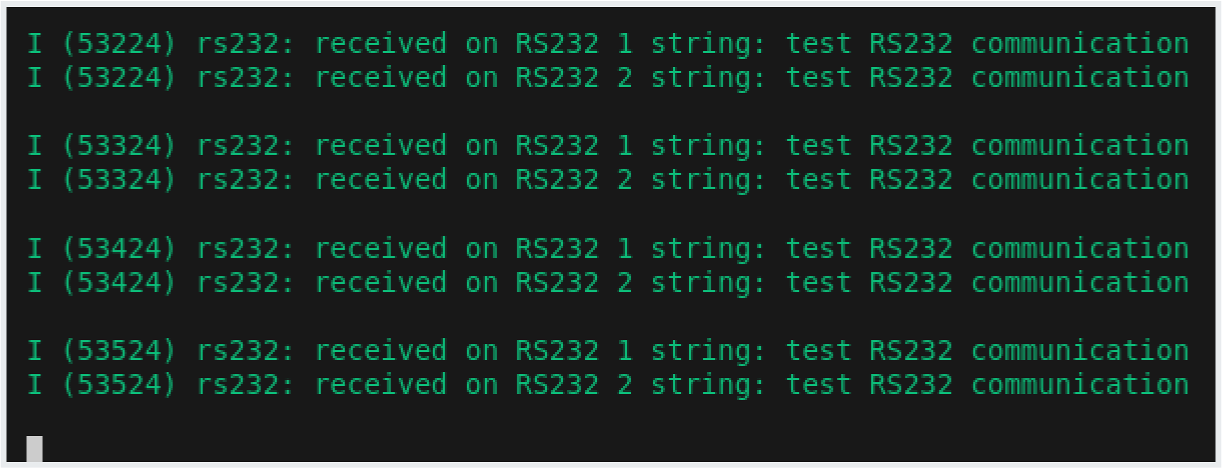
Connect RS232_1 port to the RS232_2 port in the loopback configuration.



Description

This example shows how to handle the RS232 communication via UART. After flashing both ports should send and read data from each other.

Console output



```
I (53224) rs232: received on RS232 1 string: test RS232 communication
I (53224) rs232: received on RS232 2 string: test RS232 communication

I (53324) rs232: received on RS232 1 string: test RS232 communication
I (53324) rs232: received on RS232 2 string: test RS232 communication

I (53424) rs232: received on RS232 1 string: test RS232 communication
I (53424) rs232: received on RS232 2 string: test RS232 communication

I (53524) rs232: received on RS232 1 string: test RS232 communication
I (53524) rs232: received on RS232 2 string: test RS232 communication
```

RS485

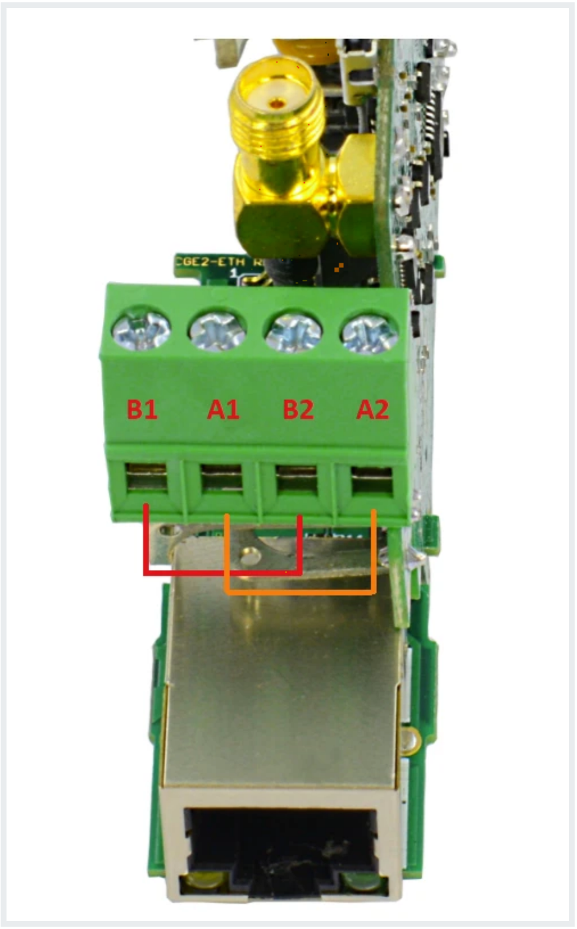
ESP32 Open IoT and IIoT Gateways (P01 & P02)

Link to repositories:

- C
- MicroPython

Connection

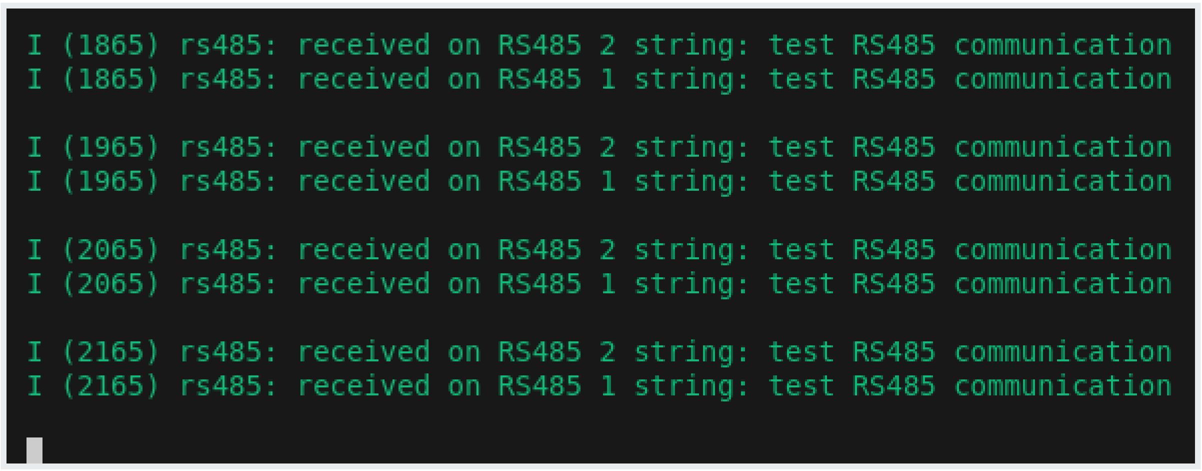
Connect RS485_1 port to the RS485_2 port in the loopback configuration.



Description

This example shows how to handle the RS485 communication via UART and UART-RS485 MAX481 converter. There is also a function implemented for each line termination via the 74HC595BQ shift register. After flashing both ports should send and read data from each other.

Console output



```
I (1865) rs485: received on RS485 2 string: test RS485 communication
I (1865) rs485: received on RS485 1 string: test RS485 communication

I (1965) rs485: received on RS485 2 string: test RS485 communication
I (1965) rs485: received on RS485 1 string: test RS485 communication

I (2065) rs485: received on RS485 2 string: test RS485 communication
I (2065) rs485: received on RS485 1 string: test RS485 communication

I (2165) rs485: received on RS485 2 string: test RS485 communication
I (2165) rs485: received on RS485 1 string: test RS485 communication
```

I2C Scanner

ESP32 Open IoT and IIoT Gateways (P01 & P02)

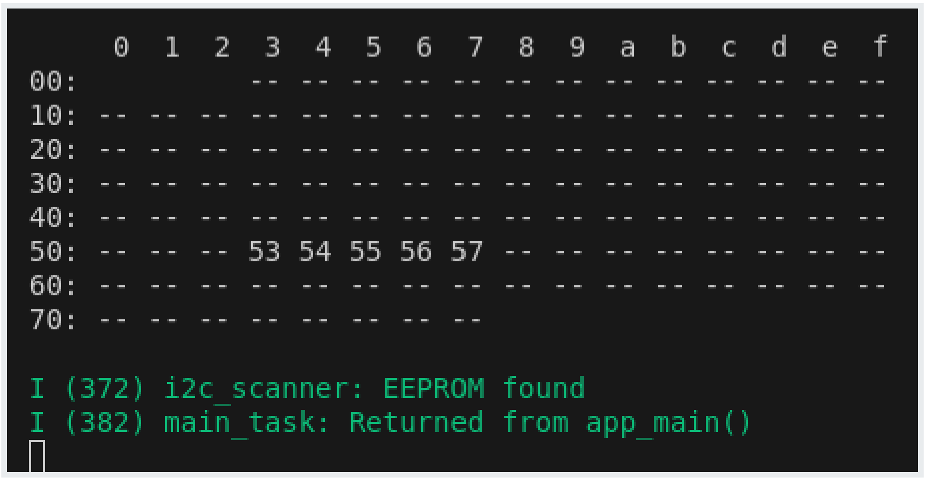
Link to repositories:

- C
- MicroPython

Description

This example scans through the I2C adresses from 0x03 to 0x78 and lists all available devices. It can be used to detect new I2C devices connected to the board.

Console output



```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 54 55 56 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --

I (372) i2c_scanner: EEPROM found
I (382) main_task: Returned from app_main()
```

Ethernet

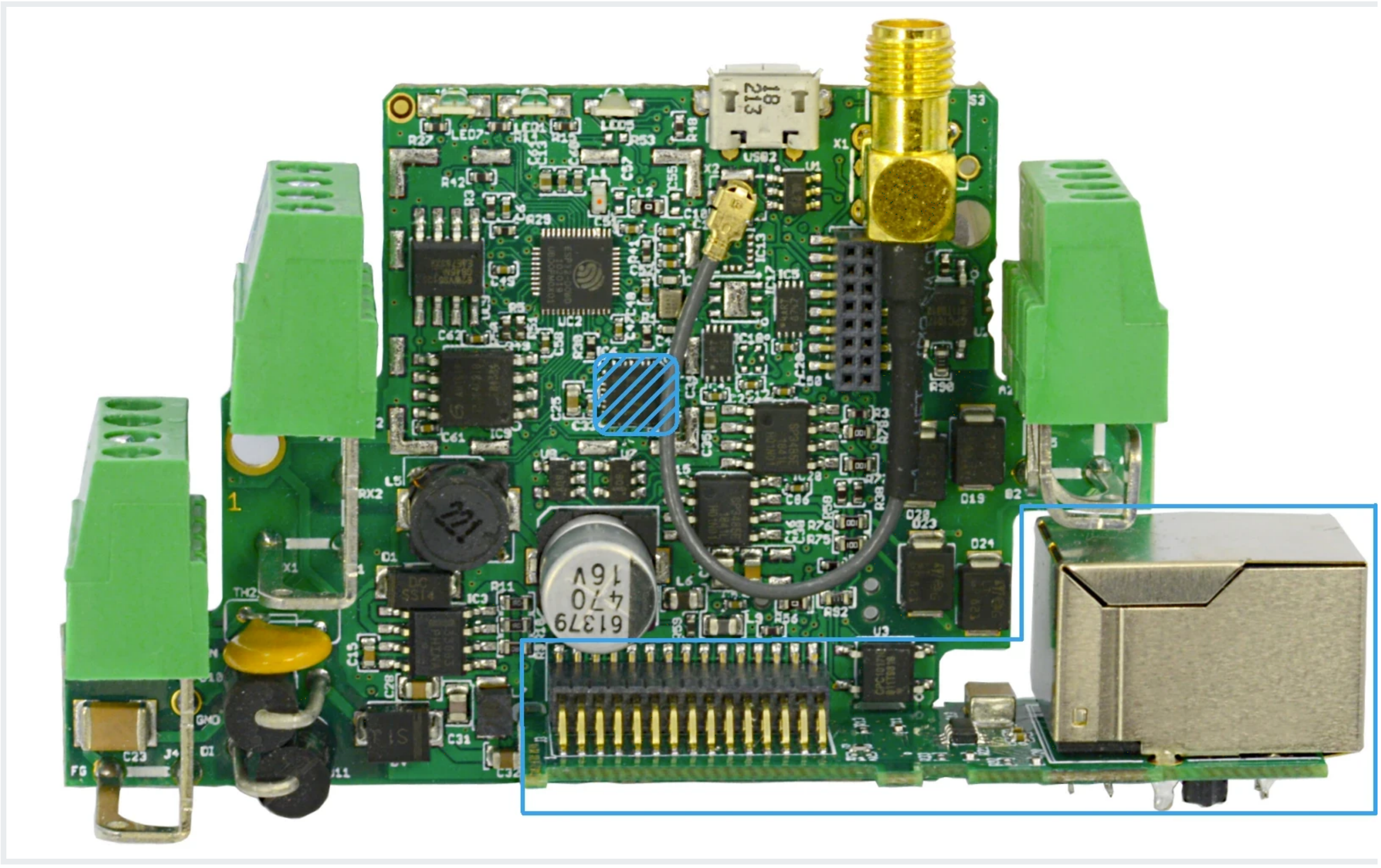
ESP32 Open IoT and IIoT Gateways (P01 & P02)

Link to repositories:

- C
- MicroPython

Connection

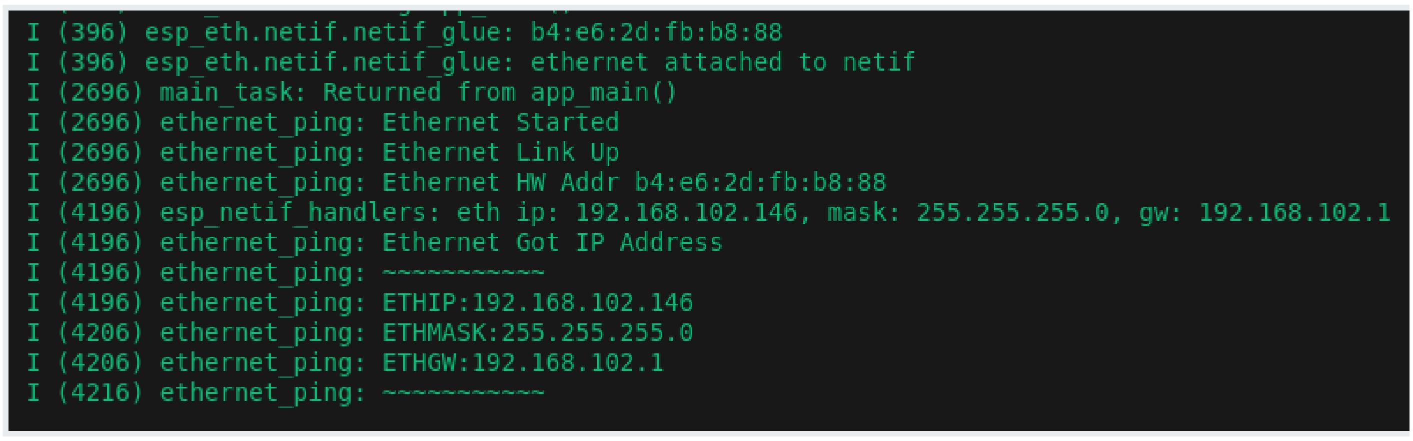
ETHERNET adapter should be connected to WLAN with a RJ45 cable.



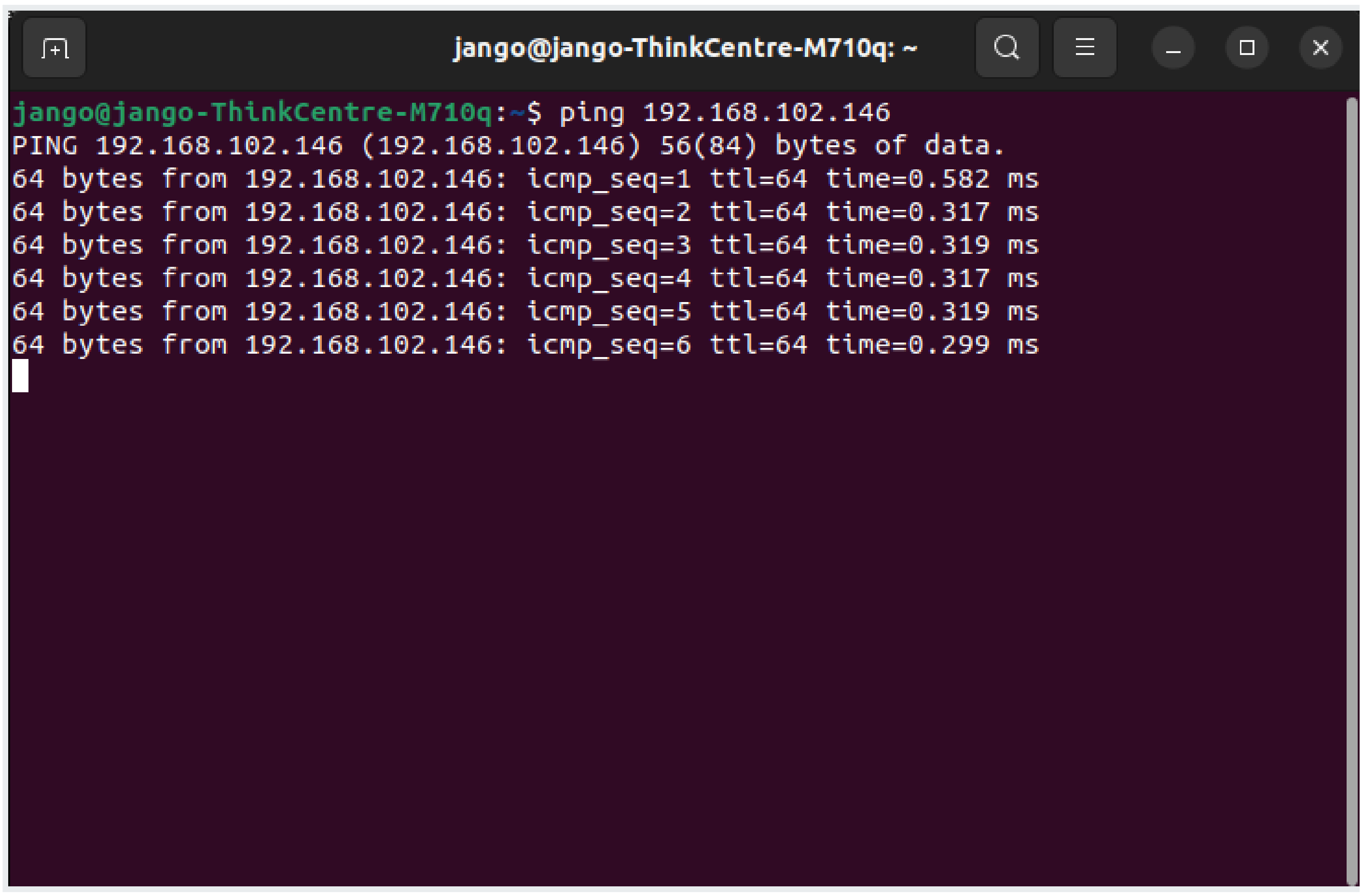
Description

This example demonstrates basic usage of ETHERNET driver. After flashing the CGE2 will try to obtain an IP address in a network it will be connected to. Once it will be done it will be possible to ping the device in this network. This example is a good base for developing own network solutions.

Console output



```
I (396) esp_eth.netif.netif_glue: b4:e6:2d:fb:b8:88
I (396) esp_eth.netif.netif_glue: ethernet attached to netif
I (2696) main_task: Returned from app_main()
I (2696) ethernet_ping: Ethernet Started
I (2696) ethernet_ping: Ethernet Link Up
I (2696) ethernet_ping: Ethernet HW Addr b4:e6:2d:fb:b8:88
I (4196) esp_netif_handlers: eth ip: 192.168.102.146, mask: 255.255.255.0, gw: 192.168.102.1
I (4196) ethernet_ping: Ethernet Got IP Address
I (4196) ethernet_ping: ~~~~~~~~~~~
I (4196) ethernet_ping: ETHIP:192.168.102.146
I (4206) ethernet_ping: ETHMASK:255.255.255.0
I (4206) ethernet_ping: ETHGW:192.168.102.1
I (4216) ethernet_ping: ~~~~~~~~~~~
```



```
jango@jango-ThinkCentre-M710q:~$ ping 192.168.102.146
PING 192.168.102.146 (192.168.102.146) 56(84) bytes of data.
64 bytes from 192.168.102.146: icmp_seq=1 ttl=64 time=0.582 ms
64 bytes from 192.168.102.146: icmp_seq=2 ttl=64 time=0.317 ms
64 bytes from 192.168.102.146: icmp_seq=3 ttl=64 time=0.319 ms
64 bytes from 192.168.102.146: icmp_seq=4 ttl=64 time=0.317 ms
64 bytes from 192.168.102.146: icmp_seq=5 ttl=64 time=0.319 ms
64 bytes from 192.168.102.146: icmp_seq=6 ttl=64 time=0.299 ms
```

WiFi

ESP32 Open IoT and IIoT Gateways (P01 & P02)

Link to repositories:

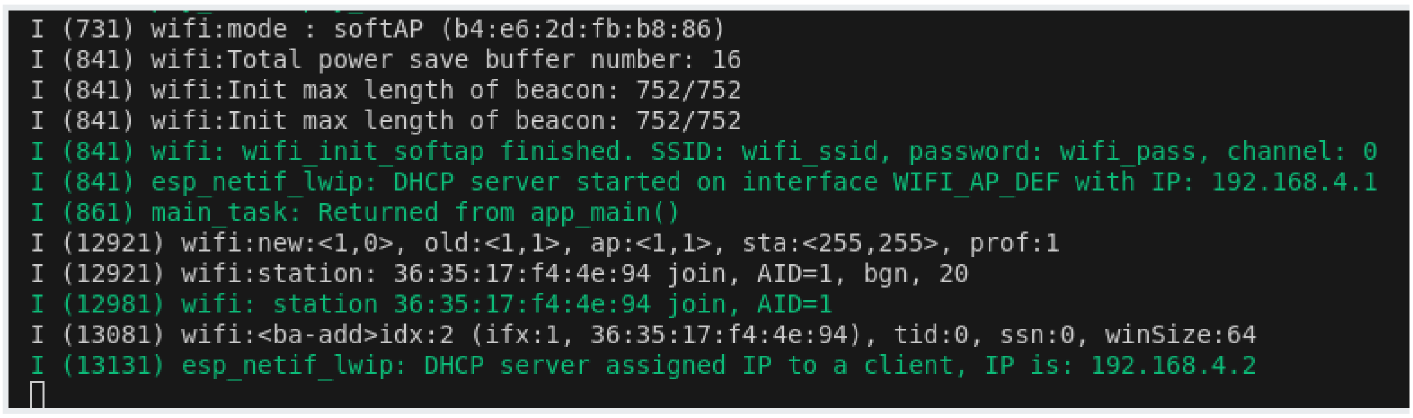
- C
- MicroPython

Description

This example shows how to create a Wi-Fi station or access point with CGE2. Access point is a device that hosts WI-FI network. Station is a device that can be connected to an existing Wi-Fi network. User can add their own network names, passwords and security features. Keep in mind that external antenna will extend the range of the device.

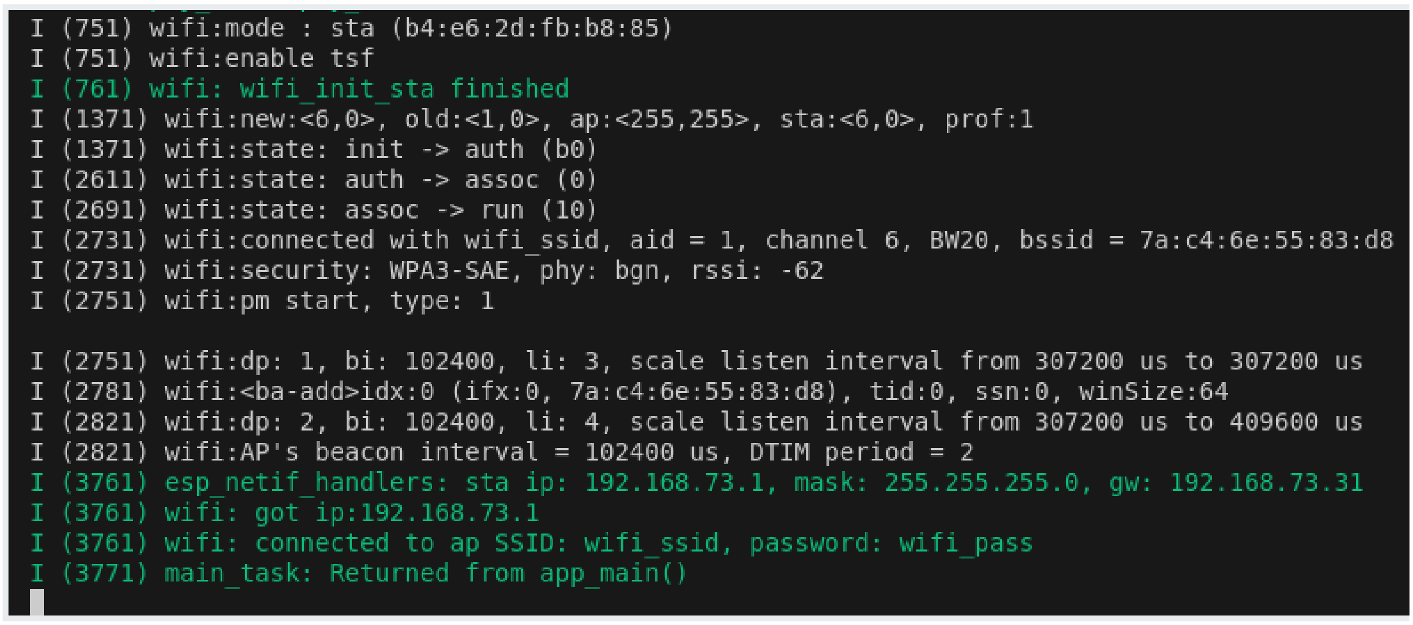
Console output

Access point



```
I (731) wifi:mode : softAP (b4:e6:2d:fb:b8:86)
I (841) wifi:Total power save buffer number: 16
I (841) wifi:Init max length of beacon: 752/752
I (841) wifi:Init max length of beacon: 752/752
I (841) wifi: wifi_init_softap finished. SSID: wifi_ssid, password: wifi_pass, channel: 0
I (841) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with IP: 192.168.4.1
I (861) main_task: Returned from app_main()
I (12921) wifi:new:<1,0>, old:<1,1>, ap:<1,1>, sta:<255,255>, prof:1
I (12921) wifi:station: 36:35:17:f4:4e:94 join, AID=1, bgn, 20
I (12981) wifi: station 36:35:17:f4:4e:94 join, AID=1
I (13081) wifi:<ba-add>idx:2 (ifx:1, 36:35:17:f4:4e:94), tid:0, ssn:0, winSize:64
I (13131) esp_netif_lwip: DHCP server assigned IP to a client, IP is: 192.168.4.2
```

Station



```
I (751) wifi:mode : sta (b4:e6:2d:fb:b8:85)
I (751) wifi:enable tsf
I (761) wifi: wifi_init_sta finished
I (1371) wifi:new:<6,0>, old:<1,0>, ap:<255,255>, sta:<6,0>, prof:1
I (1371) wifi:state: init -> auth (b0)
I (2611) wifi:state: auth -> assoc (0)
I (2691) wifi:state: assoc -> run (10)
I (2731) wifi:connected with wifi_ssid, aid = 1, channel 6, BW20, bssid = 7a:c4:6e:55:83:d8
I (2731) wifi:security: WPA3-SAE, phy: bgn, rssi: -62
I (2751) wifi:pm start, type: 1

I (2751) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (2781) wifi:<ba-add>idx:0 (ifx:0, 7a:c4:6e:55:83:d8), tid:0, ssn:0, winSize:64
I (2821) wifi:dp: 2, bi: 102400, li: 4, scale listen interval from 307200 us to 409600 us
I (2821) wifi:AP's beacon interval = 102400 us, DTIM period = 2
I (3761) esp_netif_handlers: sta ip: 192.168.73.1, mask: 255.255.255.0, gw: 192.168.73.31
I (3761) wifi: got ip:192.168.73.1
I (3761) wifi: connected to ap SSID: wifi_ssid, password: wifi_pass
I (3771) main_task: Returned from app_main()
```

More Examples

ESP32 Open IoT and IIoT Gateways (P01 & P02)

More examples from Espressif can be found here.

Also the ESP-IDF documentation can found here.

MicroPython examples and documentation can be found here.