

Configuração do Esp32

1. Instalação do Toolchain
2. Obtendo do ESP-IDF do GitHub
3. Instalação do ESP-IDF
4. baixe os drives
5. Instalação e configurações do Eclipse (opcional)

Instalação do ESP-IDF e do Toolchain

Certifique que possua o Git instalado em sua máquina se não acesse o <https://git-scm.com/downloads> e baixe para a sua versão de sistema operacional. com o git instalado acesse o site <https://github.com/espressif/esp-idf> e baixe o repositório em uma diretório criado para organizar os códigos no meu caso esse diretório se chama **esp**, utilize o parâmetro

```
cd ~ / esp
git clone --recursivo https://github.com/espressif/esp-idf.git
```

O ESP-IDF será baixado para `~/esp/esp-idf` em sua máquina.

o passo seguinte é baixar o conjunto de ferramentas de configuração (Standard Setup of Toolchain for Linux) o xtensa-esp32-elf acesse <https://github.com/espressif/esp-idf/blob/master/docs/en/get-started/linux-setup.rst> baixe a ferramenta para a sua distribuição linux no meu caso a distribuição que estou utilizando é manjaro linux um derivado Arch linux, utilizei o comando para baixar as dependências necessárias

Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python2-pyserial
python2-cryptography python2-future
```

logo em seguida:

ESP32 toolchain for Linux 64 bit no link

<https://github.com/espressif/esp-idf/blob/master/docs/en/get-started/linux-setup.rst#id9>

com os downloads concluídos coloque os arquivos no mesmo diretório do que está o esp-idf

```
~/esp/xtensa-esp32-elf/ |
```

para que ferramentas se comuniquem é necessário criar variáveis de ambiente os PATH, uma vez criado no seu `~/.profile` essa opção é interessante porque não é necessário criar novamente as variáveis de ambiente depois de reiniciar o computador, pelo fato de estar salvo no seu perfil.

```
export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

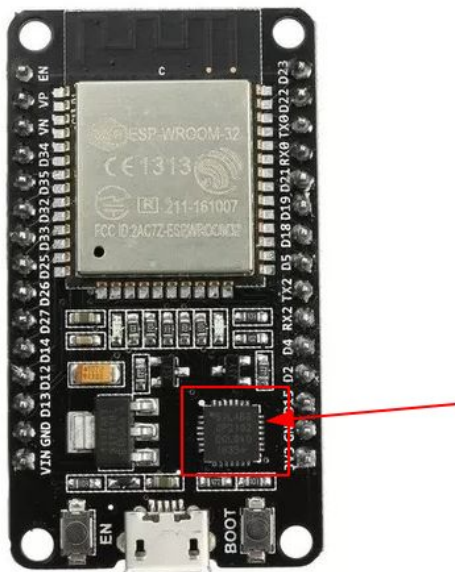
no caso as outras distribuições Linux CentOS 7, Ubuntu e Debian o último passo é a colocação do path, no caso do Arch Linux e Manjaro tem um passo a mais a instalação do xtensa-esp32-elf-gdb pois o Arch não utiliza a versão 5 do ncurses e sim a 6.1-1 baixado nos links

<https://aur.archlinux.org/packages/ncurses5-compatible-libs/> ou
<https://aur.archlinux.org/packages/lib32-ncurses5-compatible-libs/>

com as ferramentas instaladas no seu computador utilize o comando abaixo para baixar e criar um arquivo requirements.txt e ao mesmo tempo um path para o mesmo

```
python -m pip install --user -r $IDF_PATH/requirements.txt
```

por último instalar os drivers, localize qual a versão da placa que você possui, a placa que temos a disposição é a ESP32-DEVKITTV1 essa informação pode ser obtida no momento da compra, localização dos componentes de comunicação serial (o modelo do chip USB no caso a chip foi fabricado pela silabs)



ou informação impressa no verso da placa

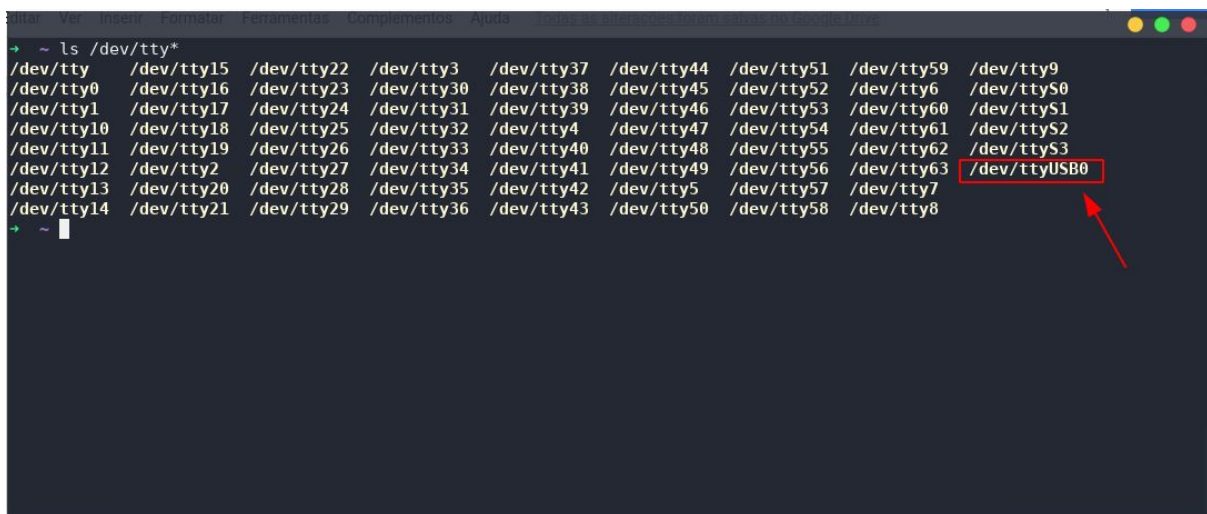


acesse o site e baixe o drive

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>

conecte o esp32 em uma das portas USB de seu computador e utilize o comando abaixo para identificar em qual porta está conectado o esp

```
ls /dev/tty*
```

A screenshot of a terminal window showing the output of the command 'ls /dev/tty*'. The output lists various serial ports. The entry '/dev/ttyUSB0' is highlighted with a red box, and a red arrow points to it from the right side of the terminal window.

```
+ ~ ls /dev/tty*
/dev/tty /dev/tty15 /dev/tty22 /dev/tty3 /dev/tty37 /dev/tty44 /dev/tty51 /dev/tty59 /dev/tty9
/dev/tty0 /dev/tty16 /dev/tty23 /dev/tty30 /dev/tty38 /dev/tty45 /dev/tty52 /dev/tty6 /dev/ttyS0
/dev/tty1 /dev/tty17 /dev/tty24 /dev/tty31 /dev/tty39 /dev/tty46 /dev/tty53 /dev/tty60 /dev/ttyS1
/dev/tty10 /dev/tty18 /dev/tty25 /dev/tty32 /dev/tty4 /dev/tty47 /dev/tty54 /dev/tty61 /dev/ttyS2
/dev/tty11 /dev/tty19 /dev/tty26 /dev/tty33 /dev/tty40 /dev/tty48 /dev/tty55 /dev/tty62 /dev/ttyS3
/dev/tty12 /dev/tty2 /dev/tty27 /dev/tty34 /dev/tty41 /dev/tty49 /dev/tty56 /dev/tty63 /dev/ttyUSB0
/dev/tty13 /dev/tty20 /dev/tty28 /dev/tty35 /dev/tty42 /dev/tty5 /dev/tty57 /dev/tty7
/dev/tty14 /dev/tty21 /dev/tty29 /dev/tty36 /dev/tty43 /dev/tty50 /dev/tty58 /dev/tty8
```

será importante saber em qual porta está conectado o esp para enviar o código para a porta correta.

Começar o projeto

Va na pasta de exemplos

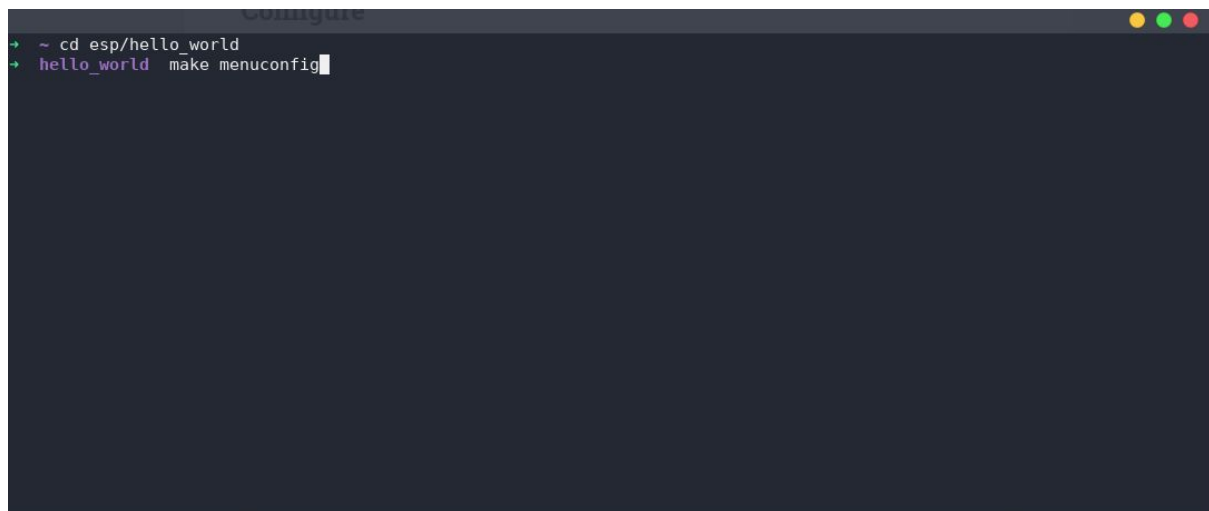
```
/home/wagner/esp/esp-idf/examples/get-started/hello_world/
```

copie o código para um diretório de testes

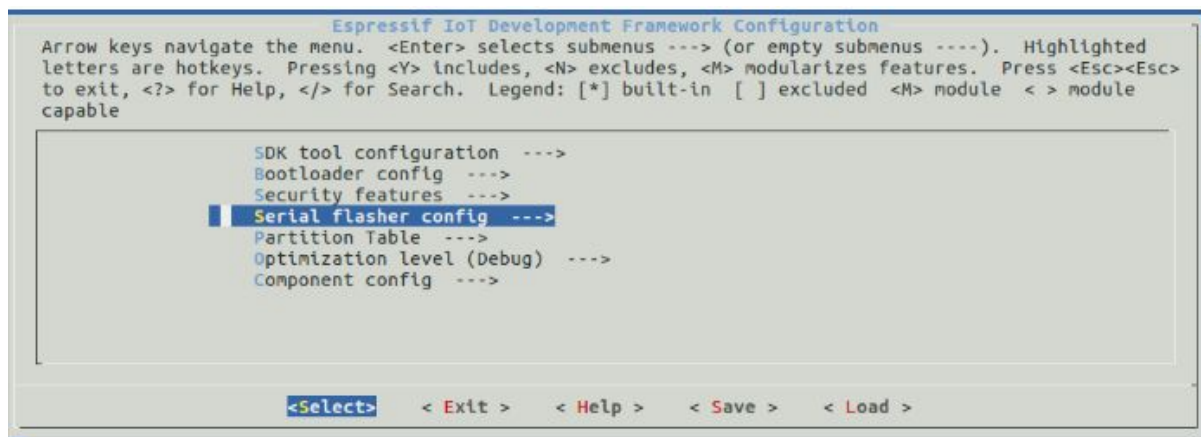
```
/home/wagner/esp/hello_world/
```

entre no diretório através do terminal e digite o comando para configurar os parâmetros que será enviado para o esp no processo de compilação

```
cd ~/esp/hello_world  
make menuconfig
```



com esse comando o terminal mudará para a janela de configuração do esp



selecione então o caminho

```
Serial flasher config > Default serial port
```

adicione a porta que o esp está conectado `/dev/ttyUSB0` salve as configurações, neste momento podemos escolher também a velocidade de transmissão de comunicação entre o esp e o computador o padrão é 115200 baud_rate contudo utilizamos a menor taxa de transmissão 9600 baud rate terminado de selecionar as configurações desejadas selecione save e depois exit.

De volta ao terminal utilize o comando

```
make flash
```

```
→ hello_world make flash
```

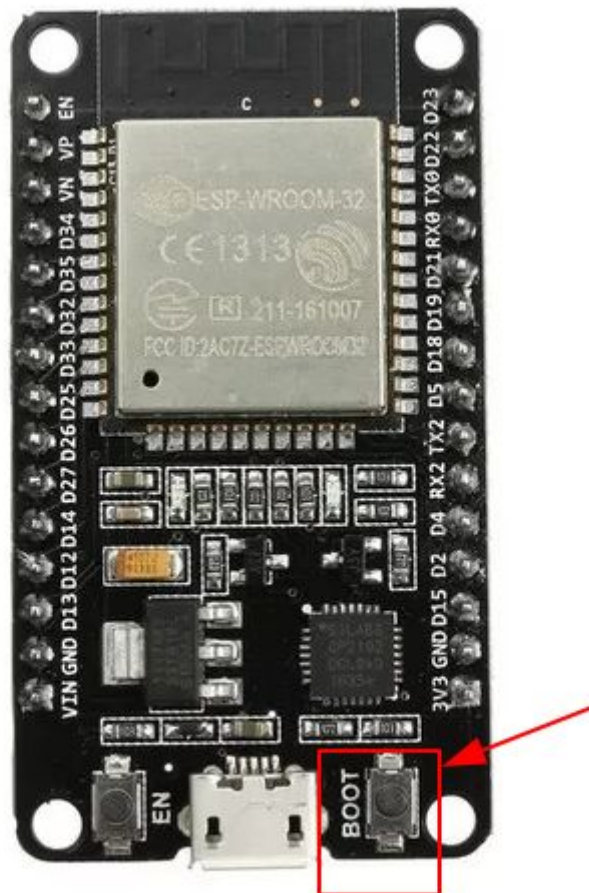
o comando irá compilar o seu código helloworld da pasta para o esp32

```
esptool.py v2.0-beta2
Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)...
esptool.py v2.0-beta2
Connecting.....
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Attaching SPI flash...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 11616 bytes to 6695...
Wrote 11616 bytes (6695 compressed) at 0x00001000 in 0.1 seconds (effective 920.5 kbit/s)...
Hash of data verified.
Compressed 408096 bytes to 171625...
Wrote 408096 bytes (171625 compressed) at 0x00010000 in 3.9 seconds (effective 847.3 kbit/s)..
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 8297.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

Pressionar o botão BOOT na placa

um fato muito importante na primeira tentativa de enviar o código o obtivemos um erro de conexão o qual não permitiu enviar o código pesquisando na internet descobrimos que no processo de conexão é necessário pressionar um botão na placa o BOOT para poder permitir o envio de códigos.



para poder verificar se está funcionando o comando `make monitor` no terminal e a saída será

```
$ make monitor
MONITOR
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
ets Jun  8 2016 00:22:57
...
```

```
...
Hello world!
Restarting in 10 seconds...
I (211) cpu_start: Starting scheduler on APP CPU.
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
```

para liberar o terminal pressione `Ctrl +]`.

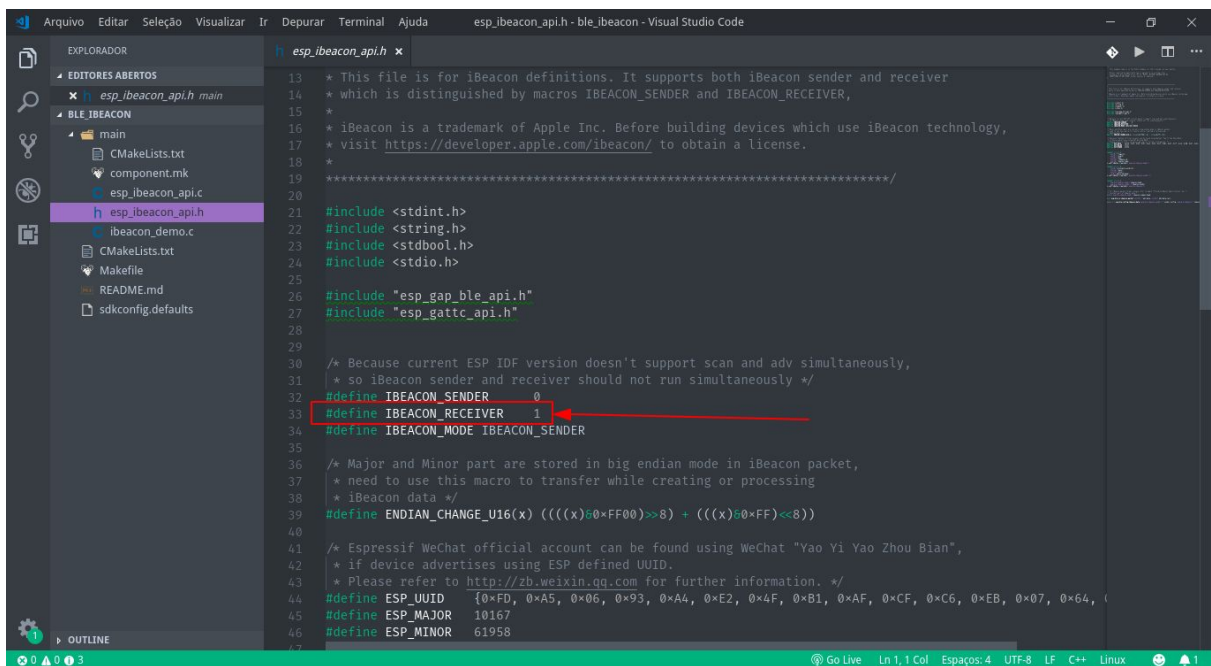
esse processo foi utilizado para colocar o código do ibeacon no esp32 entre no diretório de exemplos

```
/home/wagner/esp/esp-idf/examples/bluetooth/ble_ibeacon/
```

copie para uma área de teste

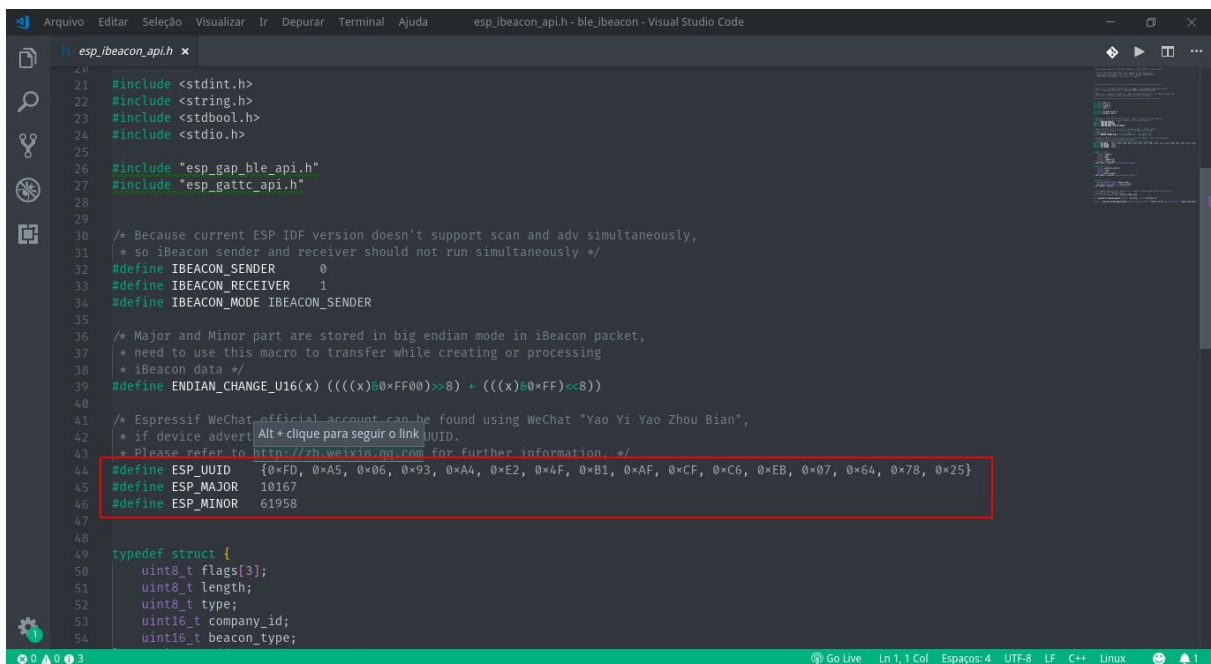
```
/home/wagner/esp/esp-idf/ble_ibeacon/
```

abra o arquivo `esp_ibeacon_api.h` e configure se o beacon será um receptor ou um transmissor

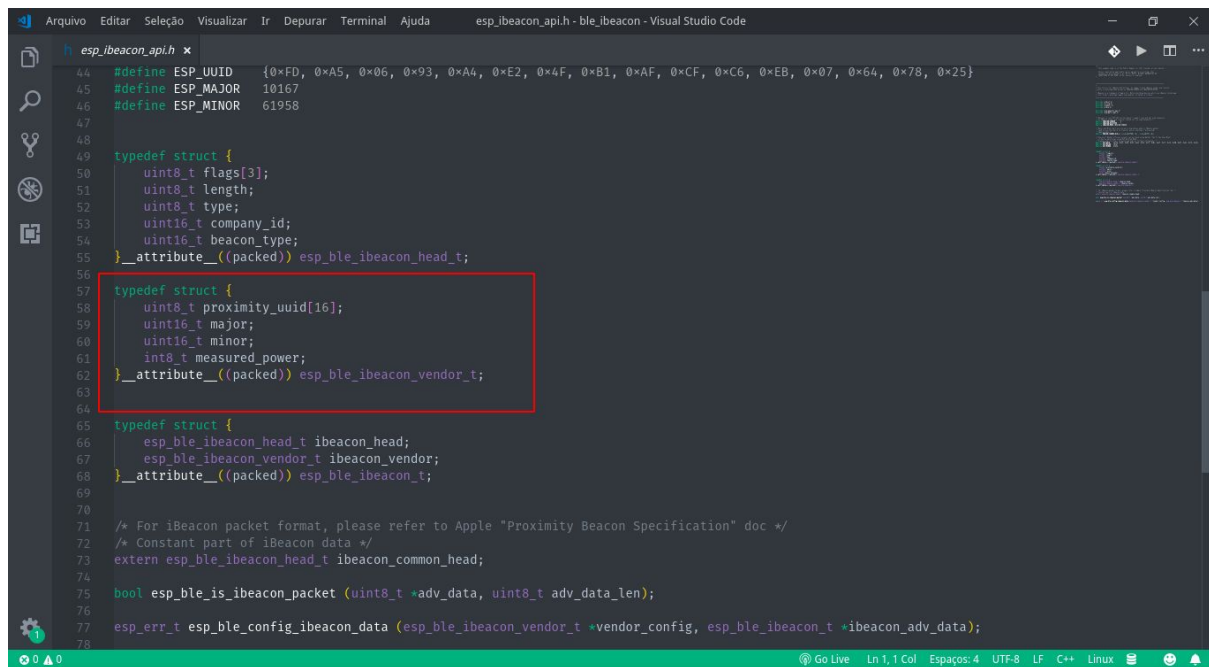


```
13 * This file is for iBeacon definitions. It supports both iBeacon sender and receiver
14 * which is distinguished by macros IBEACON_SENDER and IBEACON_RECEIVER,
15 *
16 * iBeacon is a trademark of Apple Inc. Before building devices which use iBeacon technology,
17 * visit https://developer.apple.com/ibeacon/ to obtain a license.
18 *
19 *****/
20
21 #include <stdint.h>
22 #include <string.h>
23 #include <stdbool.h>
24 #include <stdio.h>
25
26 #include "esp_gap_ble_api.h"
27 #include "esp_gattc_api.h"
28
29 /* Because current ESP IDF version doesn't support scan and adv simultaneously,
30 * so iBeacon sender and receiver should not run simultaneously */
31 #define IBEACON_SENDER 0
32 #define IBEACON_RECEIVER 1
33 #define IBEACON_MODE IBEACON_SENDER
34
35 /* Major and Minor part are stored in big endian mode in iBeacon packet,
36 * need to use this macro to transfer while creating or processing
37 * iBeacon data */
38 #define ENDIAN_CHANGE_U16(x) (((x)>>8)&0xFF) + (((x)<<8)&0xFF)
39
40 /* Espressif WeChat official account can be found using WeChat "Yao Yi Yao Zhou Bian",
41 * if device advertises using ESP defined UUID.
42 * Please refer to http://zb.weixin.qq.com for further information. */
43 #define ESP_UUID {0xFD, 0xA5, 0x06, 0x93, 0xA4, 0xE2, 0x4F, 0xB1, 0xAF, 0xCF, 0xC6, 0xEB, 0x07, 0x64, 0x78, 0x25}
44 #define ESP_MAJOR 10167
45 #define ESP_MINOR 61958
```

configure também o UUID, MAJOR, MINOR, PROXIMITY



```
21 #include <stdint.h>
22 #include <string.h>
23 #include <stdbool.h>
24 #include <stdio.h>
25
26 #include "esp_gap_ble_api.h"
27 #include "esp_gattc_api.h"
28
29 /* Because current ESP IDF version doesn't support scan and adv simultaneously,
30 * so iBeacon sender and receiver should not run simultaneously */
31 #define IBEACON_SENDER 0
32 #define IBEACON_RECEIVER 1
33 #define IBEACON_MODE IBEACON_SENDER
34
35 /* Major and Minor part are stored in big endian mode in iBeacon packet,
36 * need to use this macro to transfer while creating or processing
37 * iBeacon data */
38 #define ENDIAN_CHANGE_U16(x) (((x)>>8)&0xFF) + (((x)<<8)&0xFF)
39
40 /* Espressif WeChat official account can be found using WeChat "Yao Yi Yao Zhou Bian",
41 * if device advertises using ESP defined UUID.
42 * Please refer to http://zb.weixin.qq.com for further information. */
43 #define ESP_UUID {0xFD, 0xA5, 0x06, 0x93, 0xA4, 0xE2, 0x4F, 0xB1, 0xAF, 0xCF, 0xC6, 0xEB, 0x07, 0x64, 0x78, 0x25}
44 #define ESP_MAJOR 10167
45 #define ESP_MINOR 61958
46
47 typedef struct {
48     uint8_t flags[3];
49     uint8_t length;
50     uint8_t type;
51     uint16_t company_id;
52     uint16_t beacon_type;
53 }
```



```
44 #define ESP_UUID {0xFD, 0xA5, 0x06, 0x93, 0xA4, 0xE2, 0x4F, 0xB1, 0xAF, 0xCF, 0xC6, 0xEB, 0x07, 0x64, 0x78, 0x25}
45 #define ESP_MAJOR 10167
46 #define ESP_MINOR 61958
47
48
49 typedef struct {
50     uint8_t flags[3];
51     uint8_t length;
52     uint8_t type;
53     uint16_t company_id;
54     uint16_t beacon_type;
55 }__attribute__((packed)) esp_ble_ibeacon_head_t;
56
57 typedef struct {
58     uint8_t proximity_uuid[16];
59     uint16_t major;
60     uint16_t minor;
61     int8_t measured_power;
62 }__attribute__((packed)) esp_ble_ibeacon_vendor_t;
63
64
65 typedef struct {
66     esp_ble_ibeacon_head_t ibeacon_head;
67     esp_ble_ibeacon_vendor_t ibeacon_vendor;
68 }__attribute__((packed)) esp_ble_ibeacon_t;
69
70
71 /* For iBeacon packet format, please refer to Apple "Proximity Beacon Specification" doc */
72 /* Constant part of iBeacon data */
73 extern esp_ble_ibeacon_head_t ibeacon_common_head;
74
75 bool esp_ble_is_ibeacon_packet (uint8_t *adv_data, uint8_t adv_data_len);
76
77 esp_err_t esp_ble_config_ibeacon_data (esp_ble_ibeacon_vendor_t *vendor_config, esp_ble_ibeacon_t *ibeacon_adv_data);
78
```

com esses parâmetros configurado de acordo com sua aplicação podemos executar os **make flash** no momento da conexão aperta o botão **BOOT** na placa e por ultimo conferir se com o **make monitor**