|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | **Protokół projektu układu testującego lokalizację trilateracją w systemie Człowiek za burtą przy użyciu BLE** | | | | | | Data wystawienia: | |
|  |  | |
|  | Doc# | 3/CZB/002 |
|  | Nr wniosku NCBR: | | POIR.01.01.01-00-0196/19 | | | Nazwa projektu: | | Smart Yacht |
|  | Rozpoczęcie testów: | |  | | Zakończenie testów: | |  | |

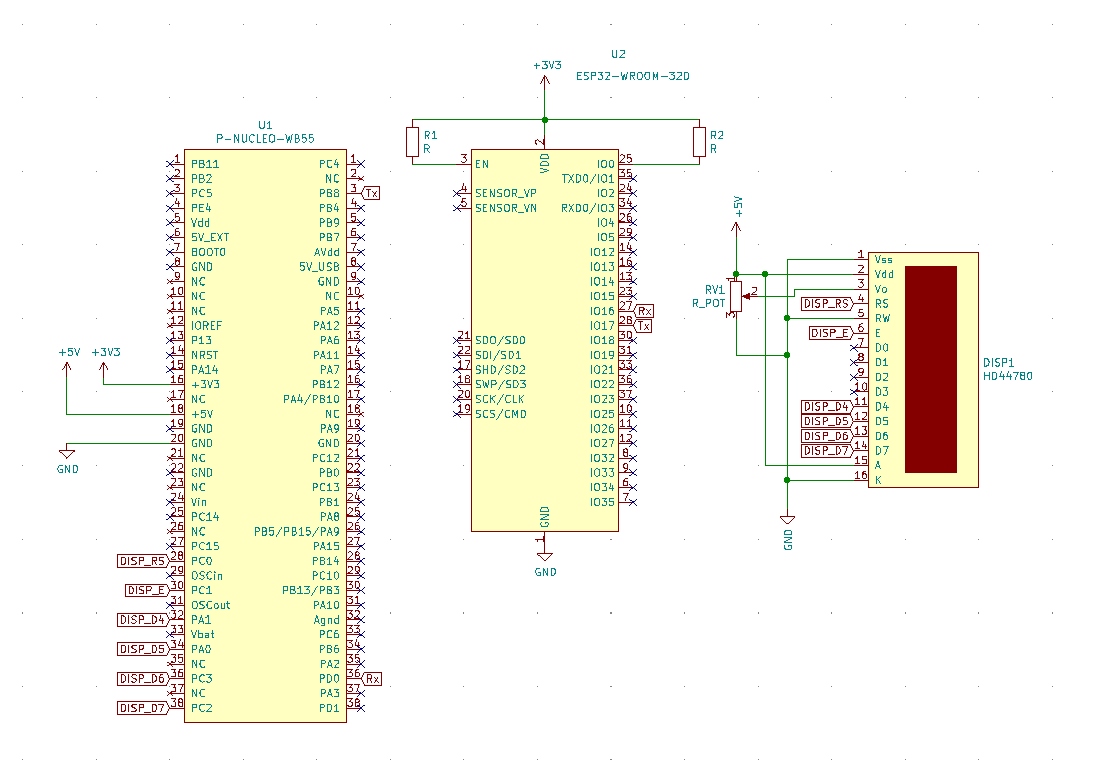
1. **Założenia**

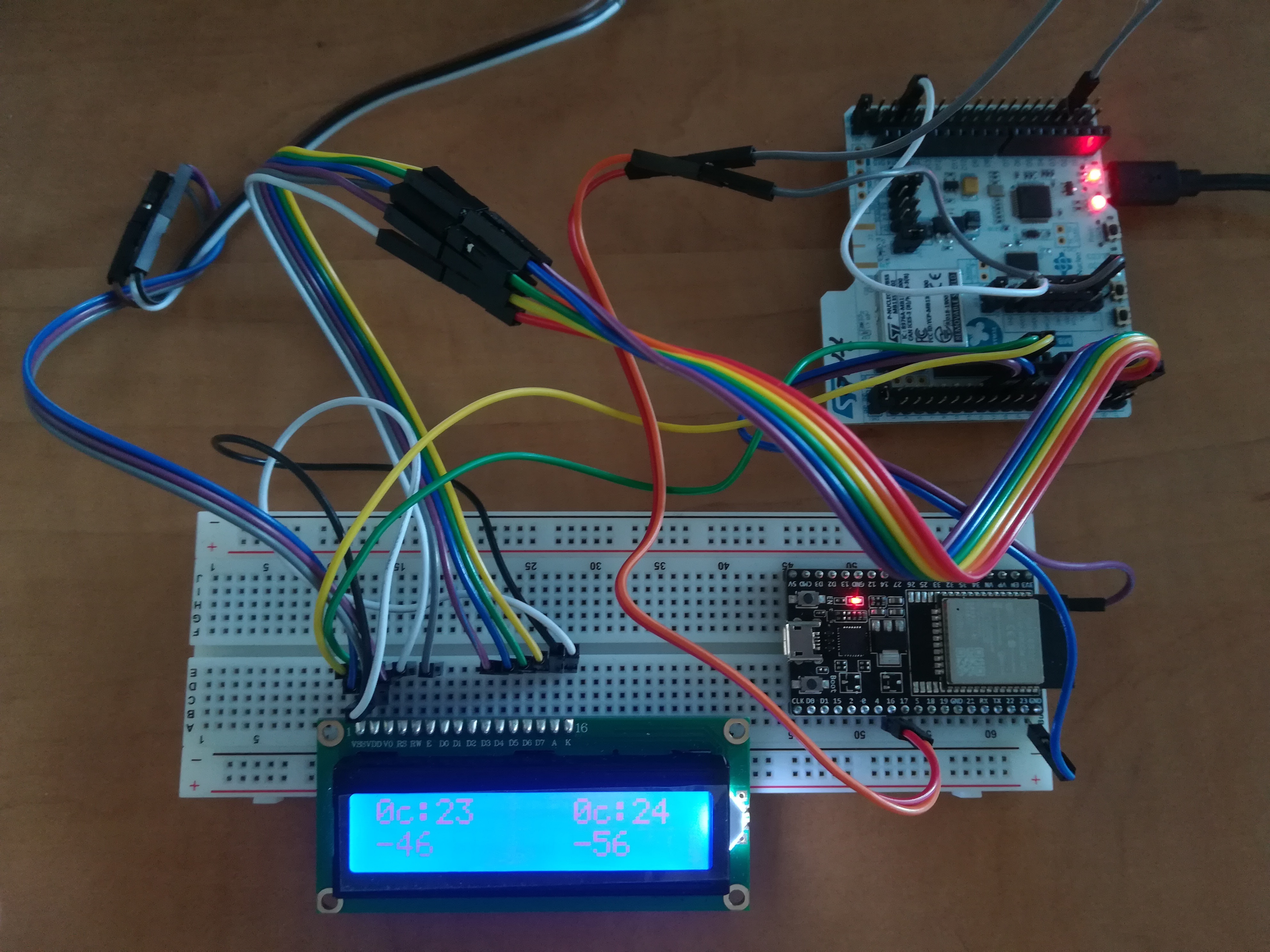
Celem jest zbudowanie układu testującego metodę trilateracji. Odbiornik musi umożliwiać odczyt mocy sygnału z dwóch nadajników iBeacon z dystansu <5m i pokazywać ją na wyświetlaczu, wraz z fragmentem adresu MAC. Powinien też być mobilny – wymagany jest odczyt mocy sygnału z różnych miejsc(symulacja wielu odbiorników).

Nadajnik iBeacon musi charakteryzować się małymi wymiarami, które umożliwią wbudowanie go w opaskę na nadgarstek na dalszych etapach projektu.

1. **Hardware**
   1. Skaner

Bazą dla układu testującego jest płytka P-NUCLEO-WB55, do której przy pomocy interfejsu UART został podłączony SoC ESP32-WROOM-32D który posłuży jako skaner Bluetooth Low Energy(BLE). Interfejs użytkownika stanowi wyświetlacz alfanumeryczny LCD. Całość została zasilona przez USB przy pomocy power banka- układ skanera będzie przenoszony pomiędzy punktami pomiarowymi.

Figura 1: Schemat testowego skanera

Figura 2: Testowy układ skanera

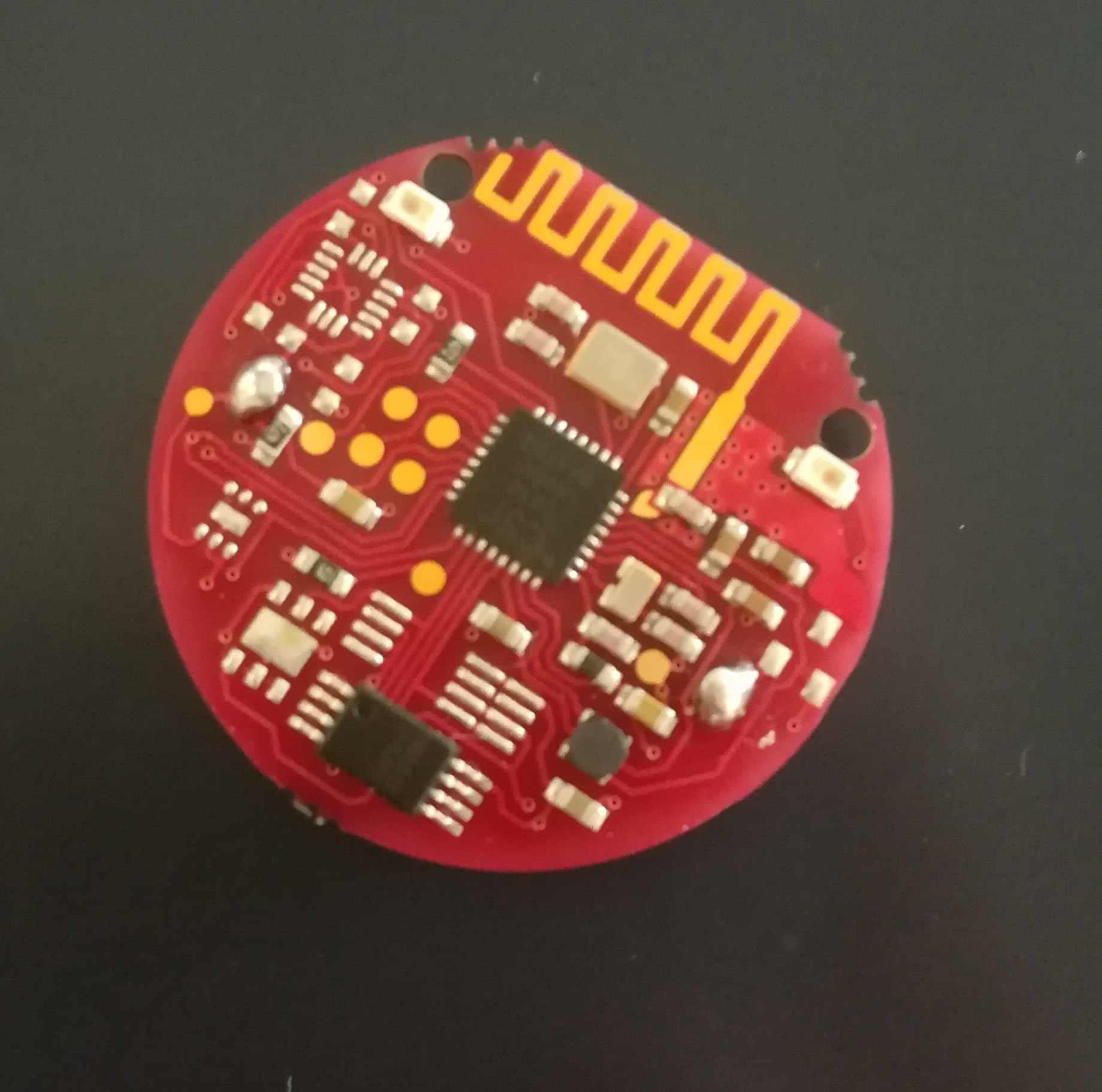
* 1. Transmiter

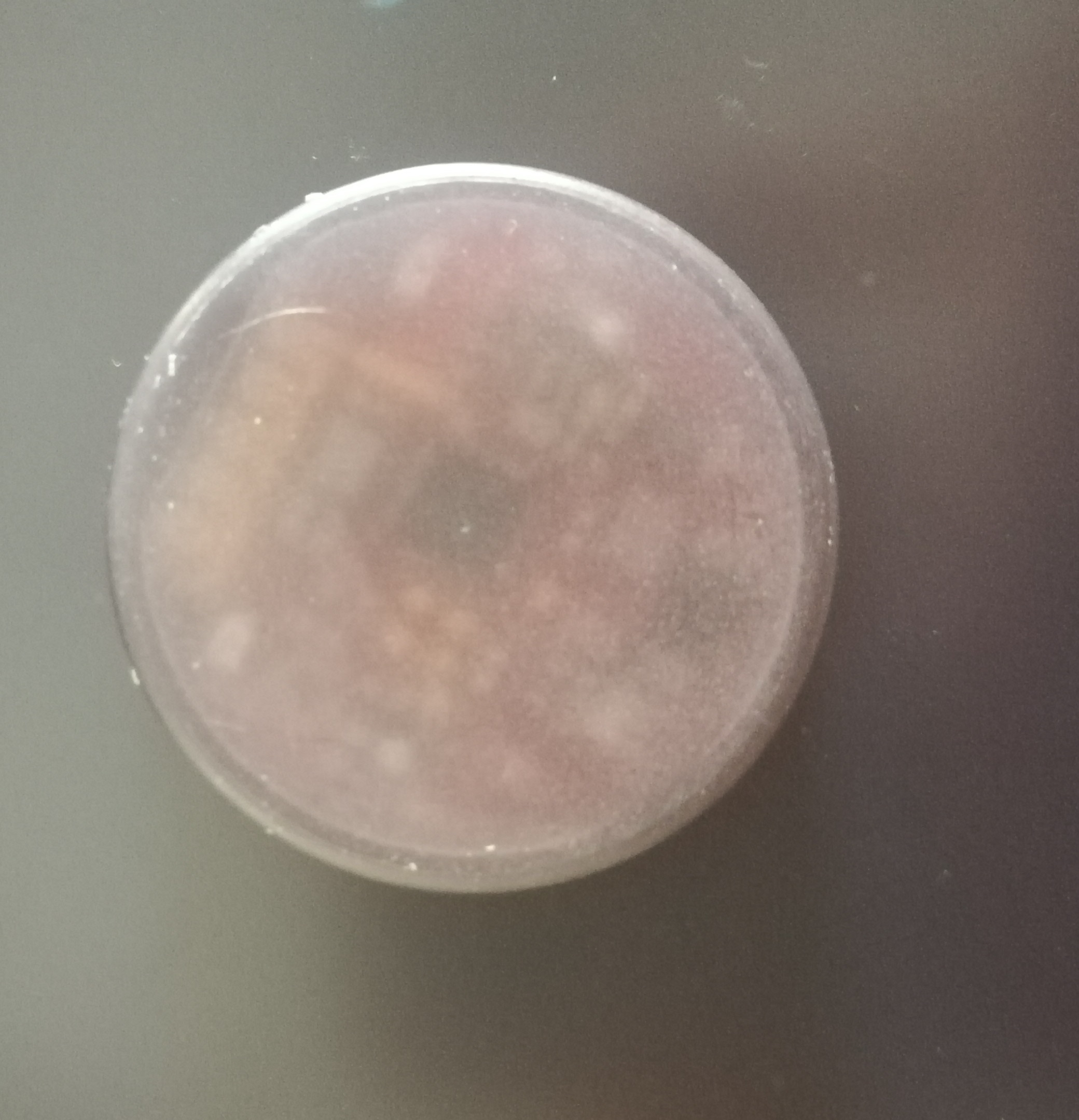
Jako urządzenie lokalizowane posłużył moduł iNode Beacon- zasilany bateryjnie konfigurowalny nadajnik Bluetooth Low Energy, zwany dalej „Tagiem”.

Służy on do przesyłania ramek rozgłoszeniowych, tzw. Advertisements zgodnych ze standardem BLE zawierających UUID oraz adres MAC taga.

Dla projektu istotna jest modyfikacja standardowego UUID służącego do lokalizowania obiektu- nie można dopuścić do tego, aby został on pomylony ze standardowym tagiem, który ktoś może mieć przy sobie. Oprócz tego iNode Beacon pozwala również na wybór pożądanej mocy nadawania spośród poniższych wartości:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5% | 10% | 12,5% | 25% | 33% | 50% | 80% | 100% |
| -18dBm | -14dBm | -10dBm | -6dBm | -2dBm | +2dBm | +5dBm | +8dBm |

Figura 3: Tag bez obudowy

Figura 4: Tag w obudowie

1. **Software**
   1. Moduł odbiorczy

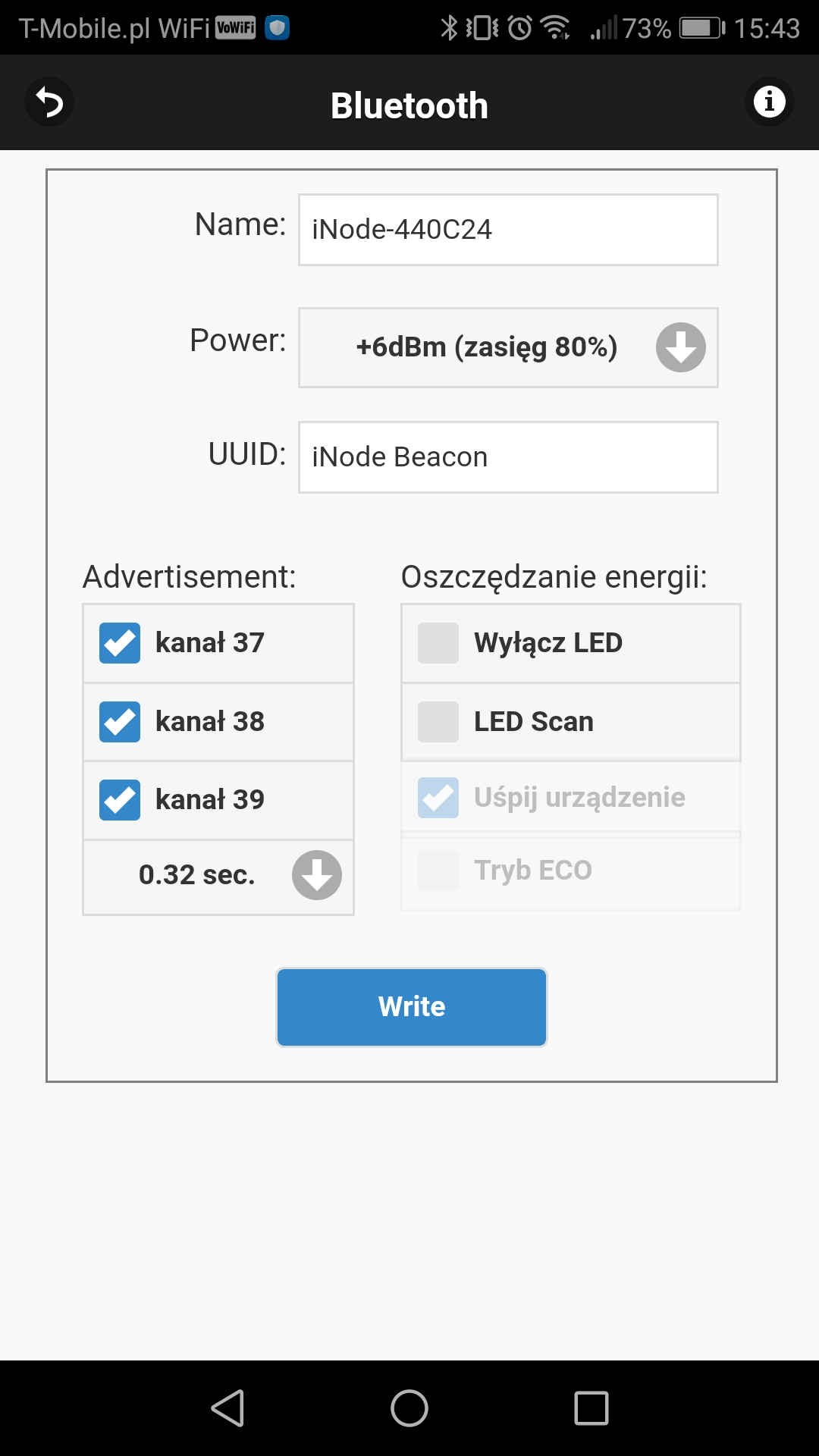
ESP32 został wyposażony w standardowy AT Firmware w wersji 2.1.0.0. Jest to oprogramowanie producenta(Espressif) pozwalające na komunikację z ESP32 przy pomocy interfejsu UART standardem Hayes AT, powszechnie używanym do komunikacji komputerów z modemami.

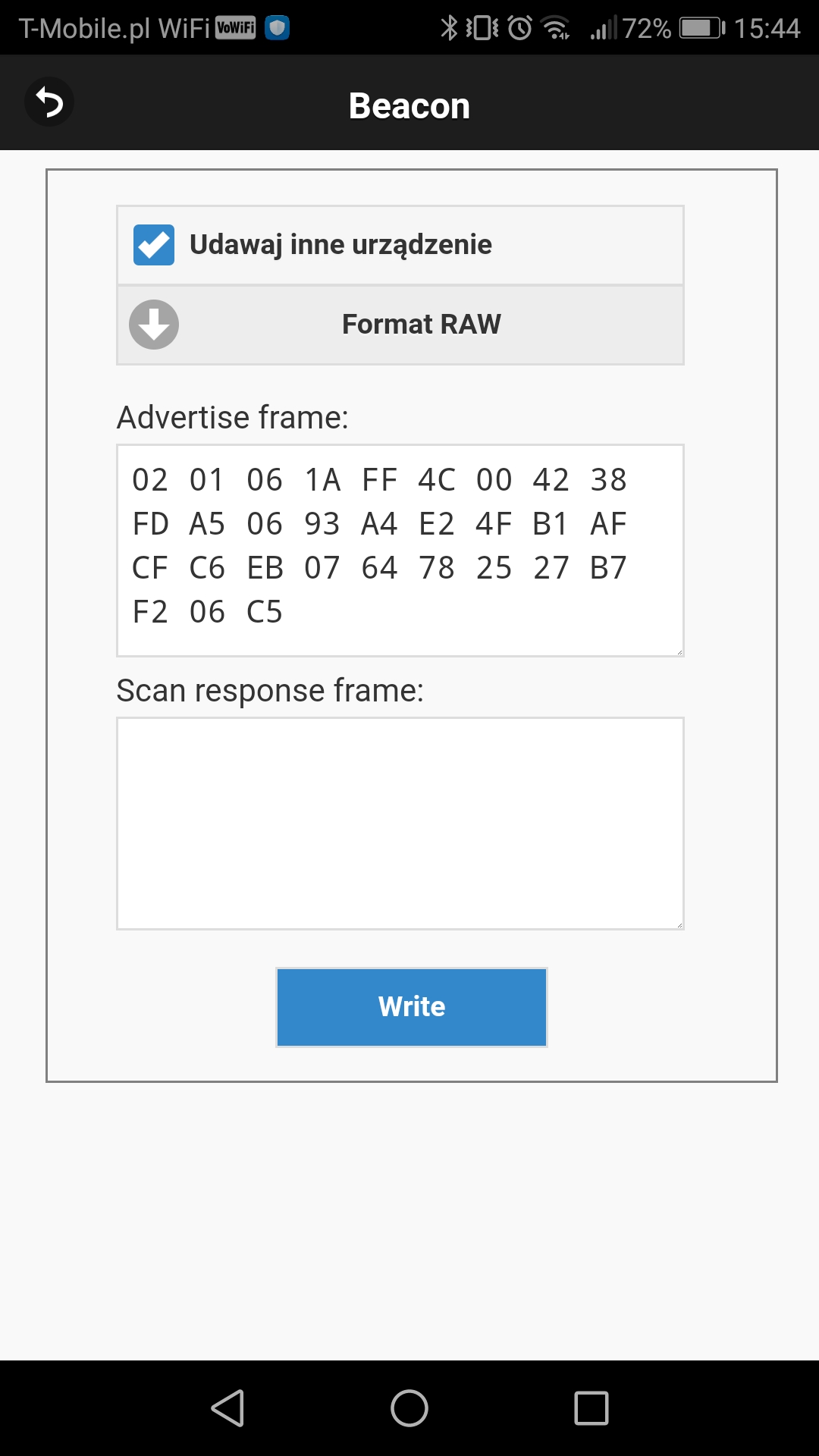
3.2 iBeacon

jest fabrycznie wyposażony w firmware rozsyłający ramki rozgłoszeniowe(tzw. Advertisements) zawierające prefix, UUID oraz adres MAC, w standardzie BLE(Bluetooth Low Energy). W trybie normalnej pracy nie jest możliwe nawiązanie połączenia z modułem. Po wyjęciu i włożeniu baterii iBeacon przechodzi w tryb programowania na 2 minuty. W tym czasie, łącząc się z nim przy pomocy komputera lub telefonu używając dedykowanej aplikacji iNode Setup można zmodyfikować moc nadawania, częstotliwość wysyłania ramek a także zmienić UUID. Do testów wybrana została moc nadawania 80%. Zmieniony został również prefix ramki iBeacon w celu wykluczenia pomyłek.

|  |  |
| --- | --- |
| Standardowa ramka iBeacon |  |
| Prefix: | 02 01 06 1A FF 4C 00 **02 15** |
| UUID: | FDA50693-A4E2-4FB1-AFCF-C6EB07647825 |
| ID sklepu: | 27 B7 |
| ID lokalizacji: | F2 06 |
| Moc nadawania: | C5 |

|  |  |
| --- | --- |
| Modyfikacja ramki |  |
| Prefix: | 02 01 06 1A FF 4C 00 **42 38** |
| UUID: | FDA50693-A4E2-4FB1-AFCF-C6EB07647825 |
| ID sklepu: | 27 B7 |
| ID lokalizacji: | F2 06 |
| Moc nadawania: | C5 |

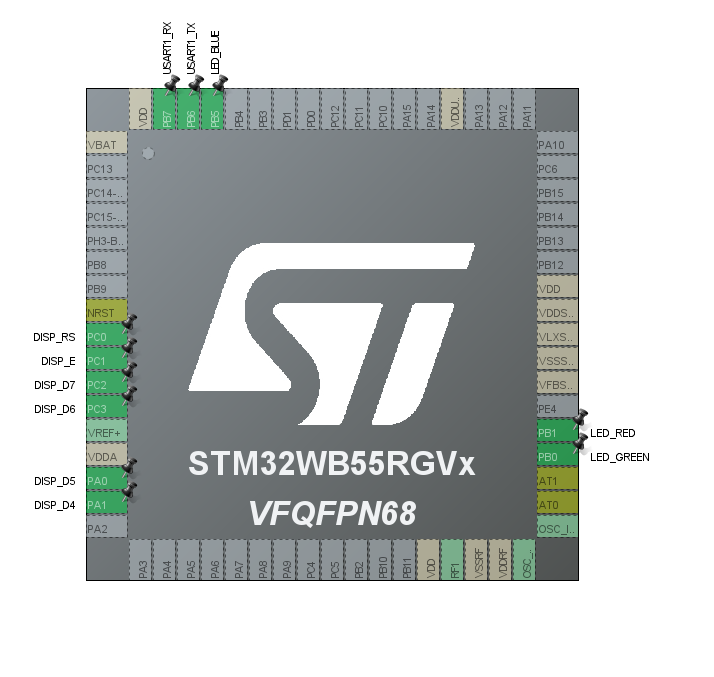
Figura 5: iNode Setup- moc nadawania

Figura 6: iNode Setup- modyfikacja ramki rozgłoszeniowej

3.3 Skaner

STM32 został wyposażony w program testowy mający za zadanie uruchomienie ESP32, skonfiguranie go jako klienta BLE oraz uruchomienie skanowania. Następnie STM odbiera od ESP32 ramki rozgłoszeniowe wszystkich urządzeń BLE w zasięgu, szuka pośród nich zmodyfikowanych ramek iBeacon a następnie wypisuje na wyświetlaczu fragment adresu MAC dwóch pierwszych znalezionych urządzeń wraz z ich aktualnym zasięgiem, co pozwala na późniejsze obliczenie dystansu.

Zostały oprogramowane również dwie diody. Zielona zapala się kiedy zostaną odebrane dane od ESP32, natomiast niebieska kiedy w odebranej ramce znajdzie się iBeacon.

Figura 7: Pinout STM32 w układzie skanera

**Kod programu, plik main.c:**

**#include** "main.h”

**#include** "../Display/an\_disp.h"

**#include** <string.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <stdbool.h>

**#include** "../Interrupts/interrupts.h"

**#include** "../AT/AT.h"

**IWDG\_HandleTypeDef** hiwdg;

**UART\_HandleTypeDef** huart1;

**DMA\_HandleTypeDef** hdma\_usart1\_rx;

**uint8\_t** receiveBuffer[BUFFER\_SIZE];

**char** receiveData[BUFFER\_SIZE];

**char** iBeacon[61];

**char** signal[3];

**char** MAC[17];

**int8\_t** signalInt;

**char** \*pointerToUUID;

**char** \*pointerToSignal;

**char** \*pointerToMAC;

**struct** **beacon** {

**char** MAC[18];

**char** signal[4];

bool busy;

**int8\_t** filteredSignal;

} beacon1, beacon2, beacon3, beaconBusyPattern;

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_DMA\_Init**(**void**);

**static** **void** **MX\_USART1\_UART\_Init**(**void**);

**static** **void** **MX\_IWDG\_Init**(**void**);

**static** **void** **MX\_NVIC\_Init**(**void**);

**int** **main**(**void**) {

**HAL\_Init**();

**SystemClock\_Config**();

**MX\_GPIO\_Init**();

**MX\_DMA\_Init**();

**MX\_USART1\_UART\_Init**();

**MX\_IWDG\_Init**();

**MX\_NVIC\_Init**();

**memset**(&beacon1, '0', **sizeof**(beacon1));

**memset**(&beacon2, '0', **sizeof**(beacon2));

**memset**(&beacon3, '0', **sizeof**(beacon3));

beacon1.filteredSignal = -30;

beacon2.filteredSignal = -30;

beacon3.filteredSignal = -30;

beacon1.busy = false;

beacon2.busy = false;

beacon3.busy = false;

**strcpy**(beaconBusyPattern.MAC, "00:00:00:00:00:00");

**strcpy**(beaconBusyPattern.signal, "-99");

beaconBusyPattern.busy = true;

**lcdInit**();

**lcdStr**("test");

**HAL\_UARTEx\_ReceiveToIdle\_DMA**(AT\_UART, receiveBuffer, BUFFER\_SIZE);

**HAL\_Delay**(200);

**ATrfPower**(78, 0, 0, 0);

**ATbleInit**(AT\_BLE\_CLIENT);

**ATbleScanEnable**(0);

**strcpy**(iBeacon,

"0201061aff4c004238fda50693a4e24fb1afcfc6eb0764782527b7f206c5");

flagReceived = 0;

**HAL\_IWDG\_Init**(&hiwdg);

**while** (1) {

**if** (flagReceived == 1) {

**HAL\_IWDG\_Refresh**(&hiwdg);

**memcpy**(&receiveData, &receiveBuffer, BUFFER\_SIZE);

//search for ibeacon UUID in received data.

pointerToUUID = **strstr**(receiveData, iBeacon);

pointerToSignal = pointerToUUID - 4;

pointerToMAC = pointerToUUID - 23;

**if** (pointerToUUID != 0) { //if beacon detected

**do** {

**HAL\_GPIO\_WritePin**(LED\_BLUE\_GPIO\_Port, LED\_BLUE\_Pin, 1);

//read signal

**memcpy**(signal, pointerToSignal, 3);

//read MAC

**memcpy**(MAC, pointerToMAC, 17);

**if** (signal[0] == '-' && MAC[2] == ':') { //check values

**if** (**memcmp**(MAC, beacon1.MAC, 17) == 0) {

**memcpy**(beacon1.signal, signal, 3);

beacon1.busy = true;

**if** (**atoi**(beacon1.signal) / 10

> beacon1.filteredSignal) {

beacon1.filteredSignal++;

} **else** **if** (**atoi**(beacon1.signal) / 10

< beacon1.filteredSignal) {

beacon1.filteredSignal--;

}

} **else** **if** (**memcmp**(MAC, beacon2.MAC, 17) == 0) {

**memcpy**(beacon2.signal, signal, 3);

beacon2.busy = true;

**if** (**atoi**(beacon2.signal) / 10

> beacon2.filteredSignal) {

beacon2.filteredSignal++;

} **else** **if** (**atoi**(beacon2.signal) / 10

< beacon2.filteredSignal) {

beacon2.filteredSignal--;

}

} **else** **if** (**memcmp**(MAC, beacon3.MAC, 17) == 0) {

**memcpy**(beacon3.signal, signal, 3);

beacon3.busy = true;

**if** (**atoi**(beacon3.signal) / 10

> beacon3.filteredSignal) {

beacon3.filteredSignal++;

} **else** **if** (**atoi**(beacon3.signal) / 10

< beacon3.filteredSignal) {

beacon3.filteredSignal--;

}

} **else** {

**if** (beacon1.busy == false) {

**memcpy**(beacon1.MAC, MAC, 17);

**memcpy**(beacon1.signal, signal, 3);

beacon1.busy = true;

} **else** **if** (beacon2.busy == false) {

**memcpy**(beacon2.MAC, MAC, 17);

**memcpy**(beacon2.signal, signal, 3);

beacon2.busy = true;

} **else** **if** (beacon3.busy == false) {

**memcpy**(beacon3.MAC, MAC, 17);

**memcpy**(beacon3.signal, signal, 3);

beacon3.busy = true;

}

}

// print MAC/signal on display

**lcdLocate**(0, 0);

**lcdChar**(beacon1.MAC[12]);

**lcdChar**(beacon1.MAC[13]);

**lcdChar**(beacon1.MAC[14]);

**lcdChar**(beacon1.MAC[15]);

**lcdChar**(beacon1.MAC[16]);

**lcdLocate**(0, 1);

**lcdInt**(beacon1.filteredSignal);

**lcdStr**(" ");

**lcdLocate**(10, 0);

**lcdChar**(beacon2.MAC[12]);

**lcdChar**(beacon2.MAC[13]);

**lcdChar**(beacon2.MAC[14]);

**lcdChar**(beacon2.MAC[15]);

**lcdChar**(beacon2.MAC[16]);

**lcdLocate**(10, 1);

**lcdInt**(beacon2.filteredSignal);

**lcdStr**(" ");

}

//clear needle

**memset**(pointerToUUID, '0', 59);

//search for next needle

pointerToUUID = **strstr**(receiveData, iBeacon);

pointerToSignal = pointerToUUID - 4;

pointerToMAC = pointerToUUID - 23;

**HAL\_GPIO\_WritePin**(LED\_BLUE\_GPIO\_Port, LED\_BLUE\_Pin, 0);

} **while** (pointerToUUID != 0);

}

flagReceived = 0;

**HAL\_GPIO\_WritePin**(LED\_GREEN\_GPIO\_Port, LED\_GREEN\_Pin, 0);

**void** **SystemClock\_Config**(**void**) {

**RCC\_OscInitTypeDef** RCC\_OscInitStruct = { 0 };

**RCC\_ClkInitTypeDef** RCC\_ClkInitStruct = { 0 };

**RCC\_PeriphCLKInitTypeDef** PeriphClkInitStruct = { 0 };

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI

| RCC\_OSCILLATORTYPE\_LSI1 | RCC\_OSCILLATORTYPE\_MSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.MSIState = RCC\_MSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.MSICalibrationValue = RCC\_MSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.MSIClockRange = RCC\_MSIRANGE\_6;

RCC\_OscInitStruct.LSIState = RCC\_LSI\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_MSI;

RCC\_OscInitStruct.PLL.PLLM = RCC\_PLLM\_DIV1;

RCC\_OscInitStruct.PLL.PLLN = 32;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLR = RCC\_PLLR\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = RCC\_PLLQ\_DIV2;

**if** (**HAL\_RCC\_OscConfig**(&RCC\_OscInitStruct) != *HAL\_OK*) {

**Error\_Handler**();

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK4 | RCC\_CLOCKTYPE\_HCLK2

| RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_PCLK1

| RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.AHBCLK2Divider = RCC\_SYSCLK\_DIV2;

RCC\_ClkInitStruct.AHBCLK4Divider = RCC\_SYSCLK\_DIV1;

**if** (**HAL\_RCC\_ClockConfig**(&RCC\_ClkInitStruct, FLASH\_LATENCY\_3) != *HAL\_OK*) {

**Error\_Handler**();

}

PeriphClkInitStruct.PeriphClockSelection = RCC\_PERIPHCLK\_SMPS

| RCC\_PERIPHCLK\_USART1;

PeriphClkInitStruct.Usart1ClockSelection = RCC\_USART1CLKSOURCE\_PCLK2;

PeriphClkInitStruct.SmpsClockSelection = RCC\_SMPSCLKSOURCE\_HSI;

PeriphClkInitStruct.SmpsDivSelection = RCC\_SMPSCLKDIV\_RANGE1;

**if** (**HAL\_RCCEx\_PeriphCLKConfig**(&PeriphClkInitStruct) != *HAL\_OK*) {

**Error\_Handler**();

}

**static** **void** **MX\_NVIC\_Init**(**void**) {

/\* USART1\_IRQn interrupt configuration \*/

**HAL\_NVIC\_SetPriority**(*USART1\_IRQn*, 0, 0);

**HAL\_NVIC\_EnableIRQ**(*USART1\_IRQn*);

/\* DMA1\_Channel1\_IRQn interrupt configuration \*/

**HAL\_NVIC\_SetPriority**(*DMA1\_Channel1\_IRQn*, 0, 0);

**HAL\_NVIC\_EnableIRQ**(*DMA1\_Channel1\_IRQn*);

}

**static** **void** **MX\_IWDG\_Init**(**void**) {

hiwdg.Instance = IWDG;

hiwdg.Init.Prescaler = IWDG\_PRESCALER\_32;

hiwdg.Init.Window = 4095;

hiwdg.Init.Reload = 4095;

**if** (**HAL\_IWDG\_Init**(&hiwdg) != *HAL\_OK*) {

**Error\_Handler**();

}

**static** **void** **MX\_USART1\_UART\_Init**(**void**) {

huart1.Instance = USART1;

huart1.Init.BaudRate = 115200;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_NONE;

huart1.Init.Mode = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

huart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

huart1.Init.ClockPrescaler = UART\_PRESCALER\_DIV1;

huart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

**if** (**HAL\_UART\_Init**(&huart1) != *HAL\_OK*) {

**Error\_Handler**();

}

**if** (**HAL\_UARTEx\_SetTxFifoThreshold**(&huart1, UART\_TXFIFO\_THRESHOLD\_1\_8)

!= *HAL\_OK*) {

**Error\_Handler**();

}

**if** (**HAL\_UARTEx\_SetRxFifoThreshold**(&huart1, UART\_RXFIFO\_THRESHOLD\_1\_8)

!= *HAL\_OK*) {

**Error\_Handler**();

}

**if** (**HAL\_UARTEx\_DisableFifoMode**(&huart1) != *HAL\_OK*) {

**Error\_Handler**();

}

**static** **void** **MX\_DMA\_Init**(**void**) {

\_\_HAL\_RCC\_DMAMUX1\_CLK\_ENABLE();

\_\_HAL\_RCC\_DMA1\_CLK\_ENABLE();

**static** **void** **MX\_GPIO\_Init**(**void**) {

**GPIO\_InitTypeDef** GPIO\_InitStruct = { 0 };

\_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

**HAL\_GPIO\_WritePin**(GPIOC,

DISP\_RS\_Pin | DISP\_E\_Pin | DISP\_D7\_Pin | DISP\_D6\_Pin, *GPIO\_PIN\_RESET*);

**HAL\_GPIO\_WritePin**(GPIOA, DISP\_D5\_Pin | DISP\_D4\_Pin, *GPIO\_PIN\_RESET*);

**HAL\_GPIO\_WritePin**(GPIOB, LED\_GREEN\_Pin | LED\_RED\_Pin | LED\_BLUE\_Pin,

*GPIO\_PIN\_RESET*);

GPIO\_InitStruct.Pin = DISP\_RS\_Pin | DISP\_E\_Pin | DISP\_D7\_Pin | DISP\_D6\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(GPIOC, &GPIO\_InitStruct);

GPIO\_InitStruct.Pin = DISP\_D5\_Pin | DISP\_D4\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(GPIOA, &GPIO\_InitStruct);

GPIO\_InitStruct.Pin = LED\_GREEN\_Pin | LED\_RED\_Pin | LED\_BLUE\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

**HAL\_GPIO\_Init**(GPIOB, &GPIO\_InitStruct);

**void** **Error\_Handler**(**void**) {

**\_\_disable\_irq**();

**while** (1) {

}

**#ifdef** USE\_FULL\_ASSERT

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

}

**#endif**

**Kod programu, plik AT.c(komunikacja z ESP32):**

**#include "../AT/AT.h"**

**void** **ATbleInit**(**uint8\_t** role) {

**char** command[14]; //for sprintf

**uint8\_t** commandUint[14]; //for HAL\_UART\_Transmit- casted

**sprintf**(command, "AT+BLEINIT=%d", role); //make AT command

**for** (**uint8\_t** i = 0; i <= 11; i++) {

commandUint[i] = command[i]; //cast command for HAL

}

commandUint[12] = '\r';

commandUint[13] = '\n'; //add line feed

// commandUint[3] = 'k'; //corrupted message- just for debug.

**HAL\_UART\_Transmit**(AT\_UART, commandUint, **sizeof**(commandUint), AT\_TIMEOUT);

**HAL\_Delay**(AT\_INTERVAL);

}

**void** **ATbleScanEnable**(**uint8\_t** interval) {

**char** command[16];

**uint8\_t** commandUint[16];

**sprintf**(command, "AT+BLESCAN=1,%d", interval);

**for** (**uint8\_t** i = 0; i <= 15; i++) {

commandUint[i] = command[i]; //cast command for HAL

}

commandUint[14] = '\r';

commandUint[15] = '\n'; //add line feed

**HAL\_UART\_Transmit**(AT\_UART, commandUint, **sizeof**(commandUint), AT\_TIMEOUT);

**HAL\_Delay**(AT\_INTERVAL);

}

**void** **ATbleAdvData**(**char** \*advData, **uint8\_t** size){

**char** command[18+size];

**uint8\_t** commandUint[18+size];

**char** string[size];

**memcpy**(string, advData, size);

**sprintf**(command, "AT+BLEADVDATA=\"%s\"", string);

**for**(**uint8\_t** i = 0; i<=size+18; i++){

commandUint[i] = command[i];

}

commandUint[size+15] = '"';

commandUint[size+16] = '\r';

commandUint[size+17] = '\n';

**HAL\_UART\_Transmit**(AT\_UART, commandUint, **sizeof**(commandUint), AT\_TIMEOUT);

**HAL\_Delay**(AT\_INTERVAL);

}

**void** **ATbleAdvStart**(**void**){

**uint8\_t** command[16] = "AT+BLEADVSTART\r\n";

**HAL\_UART\_Transmit**(AT\_UART, command, **sizeof**(command), AT\_TIMEOUT);

**HAL\_Delay**(AT\_INTERVAL);

}

**void** **ATrfPower**(**uint8\_t** wifi, **uint8\_t** bleAdv, **uint8\_t** bleScan, **uint8\_t** bleCon){

**union** **char2int**{

**char** character[21];

**uint8\_t** uint[21];

}txBuffer;

**sprintf**(txBuffer.character, "AT+RFPOWER=%d,%d,%d,%d\r\n", wifi, bleAdv, bleScan, bleCon);

**HAL\_UART\_Transmit**(AT\_UART, txBuffer.uint, **sizeof**(txBuffer.uint), AT\_TIMEOUT);

**HAL\_Delay**(AT\_INTERVAL);

**}**

**Kod programu, plik an\_disp.c(obsługa wyświetlacza):**

**#include "../Display/an\_disp.h"**

**void** **lcdSendHalf**(**uint8\_t** data) {

LCD\_E\_HIGH;

**HAL\_GPIO\_WritePin**(LCD\_D4\_PORT, LCD\_D4\_PIN, (data & 0x01));

**HAL\_GPIO\_WritePin**(LCD\_D5\_PORT, LCD\_D5\_PIN, (data & 0x02));

**HAL\_GPIO\_WritePin**(LCD\_D6\_PORT, LCD\_D6\_PIN, (data & 0x04));

**HAL\_GPIO\_WritePin**(LCD\_D7\_PORT, LCD\_D7\_PIN, (data & 0x08));

LCD\_E\_LOW;

}

**void** **lcdWriteByte**(**uint8\_t** data) {

**lcdSendHalf**(data >> 4);

**lcdSendHalf**(data);

**HAL\_Delay**(1);

}

**void** **lcdWriteCmd**(**uint8\_t** cmd) {

LCD\_RS\_LOW;

**lcdWriteByte**(cmd);

}

**void** **lcdChar**(**char** data) {

LCD\_RS\_HIGH;

**lcdWriteByte**(data);

}

**void** **lcdInit**(**void**) {

**HAL\_Delay**(15);

LCD\_E\_LOW;

LCD\_RS\_LOW;

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x03);

**HAL\_Delay**(5);

**lcdSendHalf**(0x02);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_FUNC | LCD\_4\_BIT | LCDC\_TWO\_LINE | LCDC\_FONT\_5x7);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_ONOFF | LCD\_DISP\_ON);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCD\_CLEAR);

**HAL\_Delay**(5);

**lcdWriteCmd**( LCDC\_ENTRY\_MODE | LCD\_EM\_SHIFT\_CURSOR | LCD\_EM\_RIGHT);

**HAL\_Delay**(5);

}

**void** **lcdLocate**(**uint8\_t** x, **uint8\_t** y) {

**switch** (y) {

**case** 0:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE1 + x));

**break**;

**case** 1:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE2 + x));

**break**;

**case** 2:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE3 + (x - 12)));

**break**;

**case** 3:

**lcdWriteCmd**( LCDC\_SET\_DDRAM | (LCD\_LINE4 + (x - 12)));

**break**;

}

}

**void** **lcdStr**(**char** \*text) {

**while** (\*text)

**lcdChar**(\*text++);

}

**void** **lcdInt**(**int** data){

**char** buffer[20];

**sprintf**(buffer, "%d", data);

**lcdStr**(buffer);

}

**Kod programu, plik interrupts.c(obsługa przerwania):**

**#include "interrupts.h"**

**void** **HAL\_UARTEx\_RxEventCallback**(**UART\_HandleTypeDef** \*huart, **uint16\_t** Size) {

**if** (huart->Instance == USART1) {

**HAL\_GPIO\_WritePin**(LED\_GREEN\_GPIO\_Port, LED\_GREEN\_Pin, 1);

flagReceived = 1;

**HAL\_UARTEx\_ReceiveToIdle\_DMA**(&huart1, receiveBuffer, BUFFER\_SIZE);

}

}

1. **Wnioski**

Układ testowy działa poprawnie i spełnia założenia- umożliwia identyfikację i odczyt mocy sygnału z dwóch tagów.

Wykonał: Bartosz Pracz