AKADEMIA GÓRNICZO-HUTNICZA

Instrukcja laboratoryjna

Komunikacja bezprzewodowa z wykorzystaniem NRF24L01+ oraz STM32L476RG

z przedmiotu

Standardy i Systemy Komunikacyjne

Systemy Wbudowane, rok I studiów magisterskich

Jakub Górnisiewicz
Marcin Maj

14.12.2023

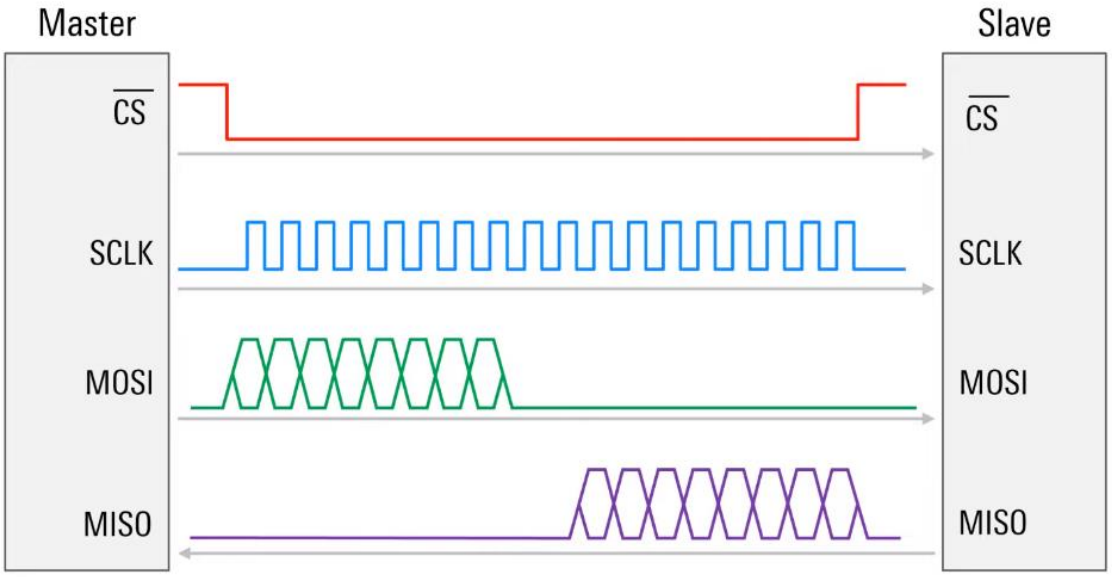
Spis treści

1. Szeregowy interfejs komunikacyjny – SPI

Jest to szeregowy interfejs komunikacyjny, posiada 4 porty: CS, SCLK, MOSI, MISO. Nie posiada adresowania. Jest „luźno" ustandaryzowany, to znaczy, że: nie ma określonych poziomów napięć, formatu ramki, ilości transmitowanych bitów, nie ma określonej kolejności bitów (Little endian/Big endian).

Najważniejsze cechy interfejsu:

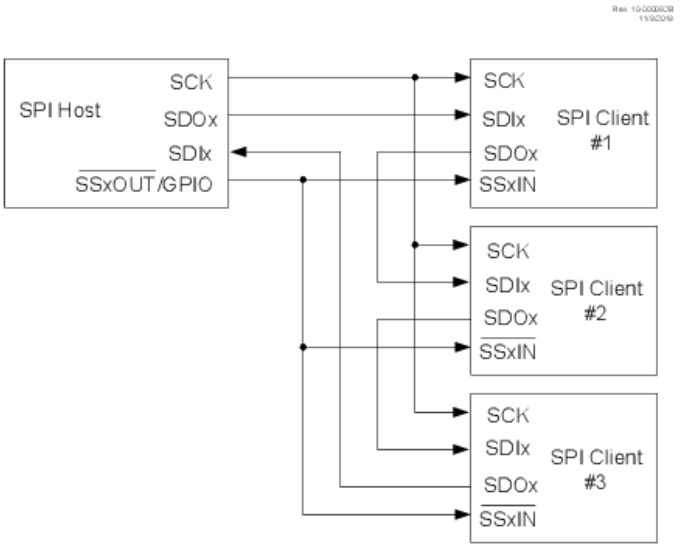
- Wspiera full-duplex
- Architektura typu master <-> slave, jeden master, co najmniej jeden slave
- Sygnały:
    - CS – Chip Select (występuje też pod nazwą SS – slave select), sygnał ten służy do wyboru urządzenia komunikującego się z masterem
    - SCLK – sygnał zegarowy, używany do synchronizacji
    - MOSI – Master Out Slave In, sygnał danych przesyłanych od mastera do slave
    - MISO – Master In Slave Out, sygnał danych przesyłanych od slave do mastera
- Tryby pracy zależne od polaryzacji zegara i fazy zegara:
    - CPOL – Zegar może być aktywny w stanie niskim lub wysokim
    - CPHA – Faza zegara to parametr który ustala próbkowanie danych na zboczu narastającym lub opadającym.



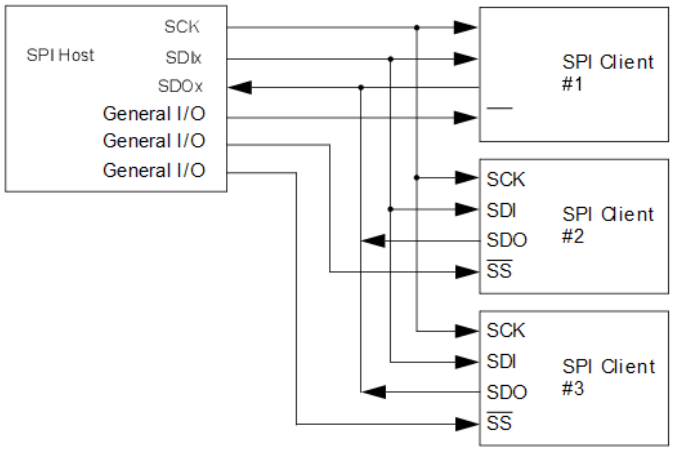
Rysunek 1. Transimsja danych z użyciem interfejsu SPI

Wybór slave'a odbywa się za pomocą ustalenia sygnału CS danego slave'a na stan niski. Stan ten utrzymuje się dopóki trwa komunikacja. Sygnał SCLK nie jest przesyłany kiedy CS jest w stanie wysokim.

Konfiguracje z wieloma slave’ami:



Rysunek 2. SPI w konfiguracji daisy chain

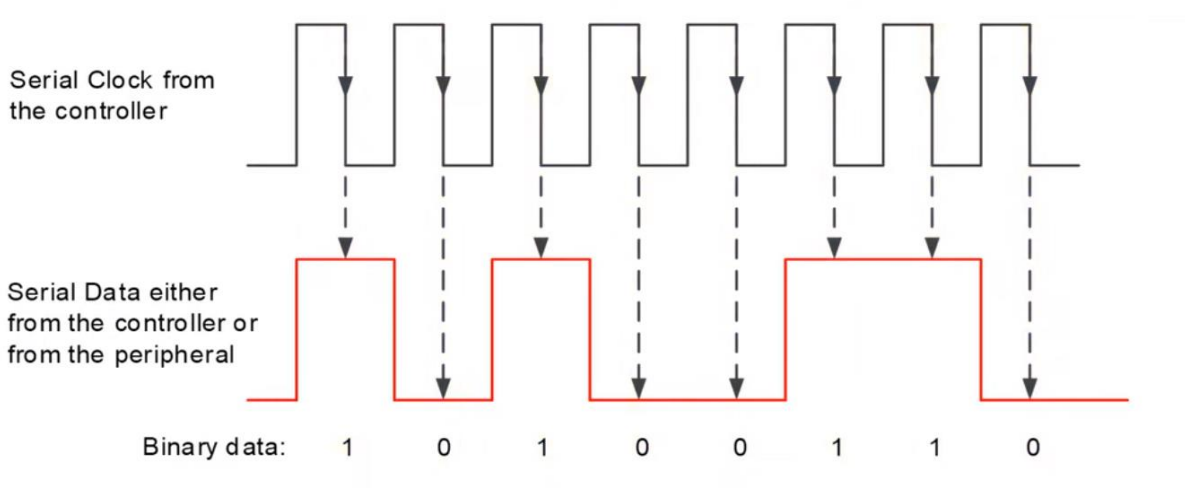


Rysunek 3. SPI w konfiguracji klasycznej

Klasyczna konfiguracja pracuje szybciej ale potrzebna jest większa ilość portów kontroli w masterze (hoście).

Z racji 2 parametrów CPOL i CPHA mamy 4 konfiguracje SPI, które w inny sposób próbkują dane.

Przykład próbkowania na zboczu opadającym:



Rysunek 4. Przykład działania dla jednej konfiguracji

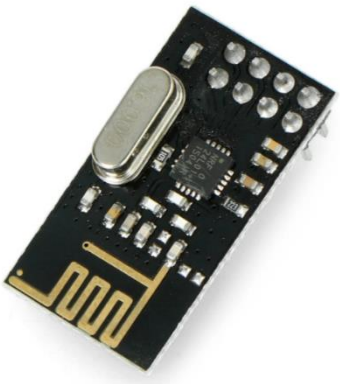
Więcej informacji na temat SPI znajduje się pod linkiem: Introduction to SPI Interface | Analog Devices, Understanding SPI - YouTube

2. Moduł radiowy nRF24L01+

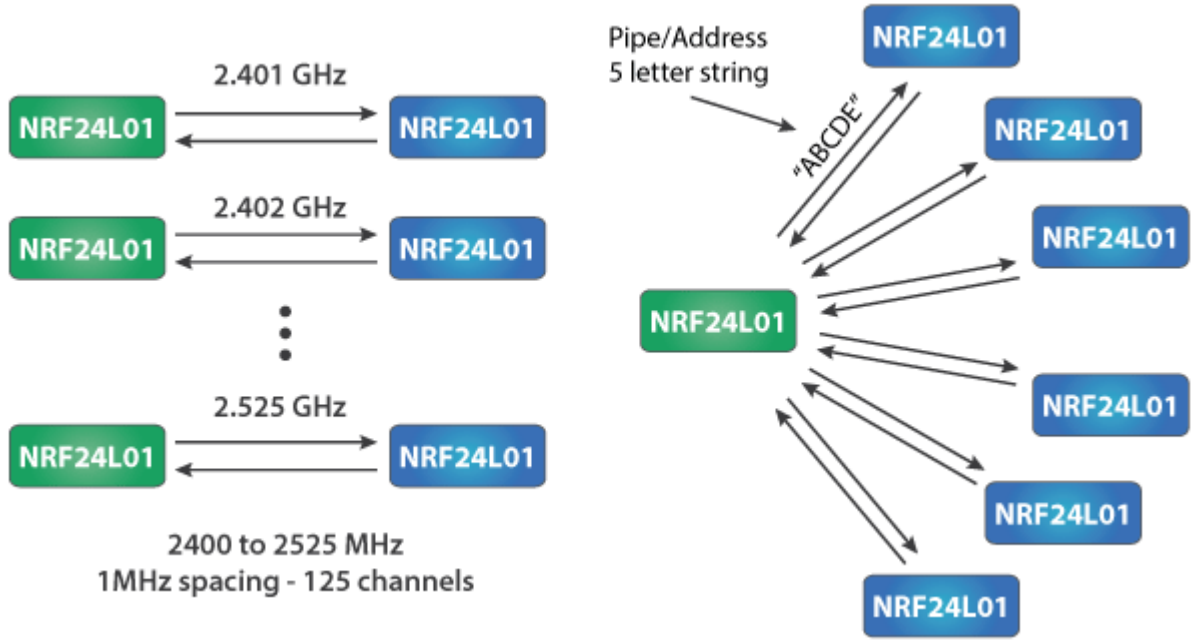
Jest to urządzenie firmy Nordic Semiconductor. Posiada odbiornik o regulowanym wzmocnieniu, nadajnik o przestrajanej mocy wyjściowej, syntezer częstotliwości, filtry RF, modulator i demodulator GFSK a także sprzęt potrzebny do obsługi protokołu WiFi. Wyposażony jest również w interfejs SPI.

Parametry techniczne:

- pasmo 2.4 GHz
- zasilanie 1.9 V – 3.6 V
- pobór prądu 11mA (przy 0 dBm mocy wejściowej)
- wbudowana antena
- wbudowana sprzętowa kolejka FIFO
- zasięg max 100m
- prędkości transmisji: 250 kbps, 1 Mbps, 2 Mbps
- przy 250kbps zasięg wynosi do 100 m
- modulacja GFSK
- wymiary 30 x 16 mm
- montaż THT, raster 2,54 mm



Rysunek 5. nRF24L01+



Rysunek 6. Konfiguracje lokalnej sieci nRF

Moduły nRF oferują do 125 różnych kanałów w zakresie 2.4 GHz do 2.525 GHz, jeden kanał pozwala na zaadresowanie do 6 urządzeń.

Technologia ShockBurst™:

W chipie zastosowano technologię ShockBurst™, dzięki której możemy przesyłać dane z wysokim datarate bez potrzeby wykorzystanie szybkiego mikrokontrolera dla procesowania danych i odzyskiwania zegara. Chip wyposażony w tą technologię posiada FIFO oraz zaimplementowane protokoły RF do szybkiego przesyłu danych. ShockBurst również redukuje pobór prądu

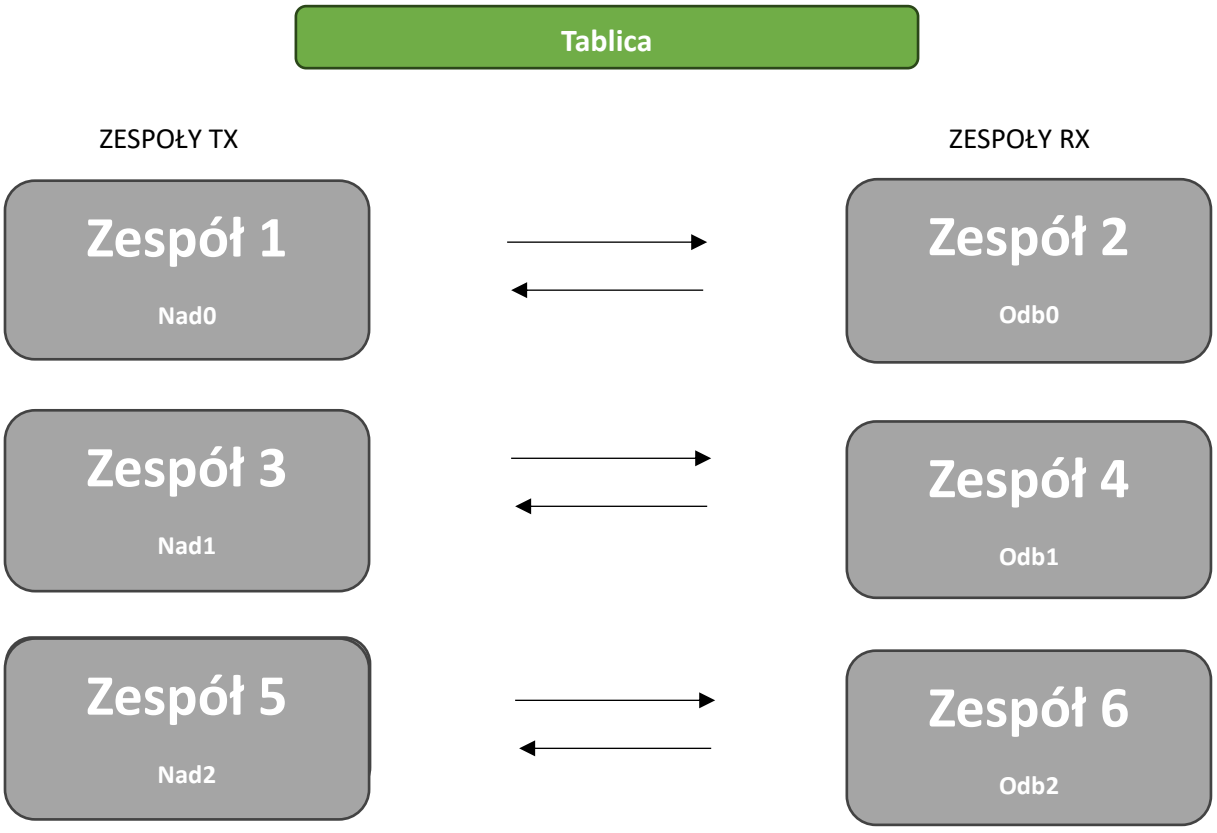
3. Cel ćwiczenia

Celem instrukcji laboratoryjnej jest zapoznanie się z protokołem komunikacyjnym SPI *(ang. Serial Peripheral Interface)* oraz modułem radiowym NRF24L01. Efektem finalnym będzie możliwość bezprzewodowego komunikowania się między parami studentów, co zweryfikujemy na ekranach komputerów.

Do realizacji projektu konieczne będzie (dla jednego zespołu):

- płytka rozwojowa STM32FE411RE, moduł radiowy NRF24L01
- komputer osobisty
- środowisko STM32CubeIDE w najnowszej wersji, które można pobrać ze strony producenta
- konto na portalu firmy STMicroelectronics: My ST Registration - STMicroelectronics
- klienta usług np. PuTTY.

Plan ułożenia zespołów w sali C3/301



Każdy zespół musi mieć swoją parę, w przypadku nieparzystej ilości zespołów należy stworzyć np. pojedyncze lub 3-osobowe zespoły.

W dalszych etapach w kodzie należeć będzie zmieniać „NadX” oraz „Odb X” na konkretne wartości np. „Nad2” zgodne z powyższym rysunkiem.

W dalszej części instrukcji wykorzystywanie zostanie przykład pary zero.

```
nRF24_SetRXAddress(0, "NadX");
nRF24_SetTXAddress("OdbX");
```

4. Przebieg ćwiczenia

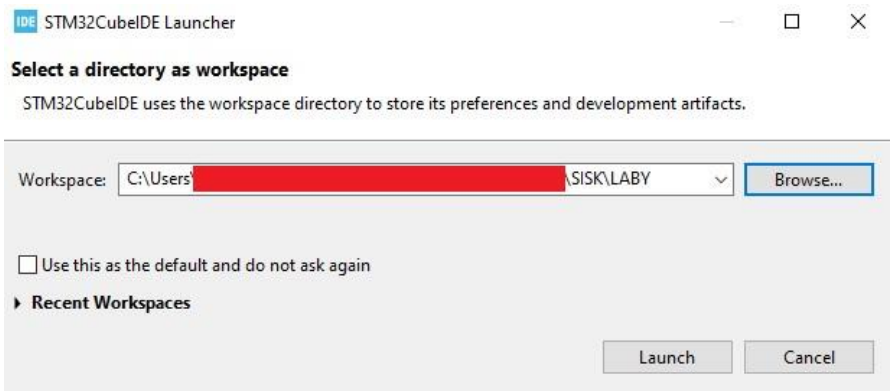
WERSJA PODSTAWOWA

1. Założenie nowego projektu w STM32CubeIDE, zalogowanie się na konto STM
2. Konfiguracja peryferiów (GPIO, UART, SPI)
3. Import biblioteki nRF24
4. Ustawienie typu transmisji RX/TX i adresu
5. Podłączenie przewodów
6. Wgranie programu na płytki
7. Uruchomienie terminalu portu szeregowego np. PuTTY
8. Weryfikacja programów

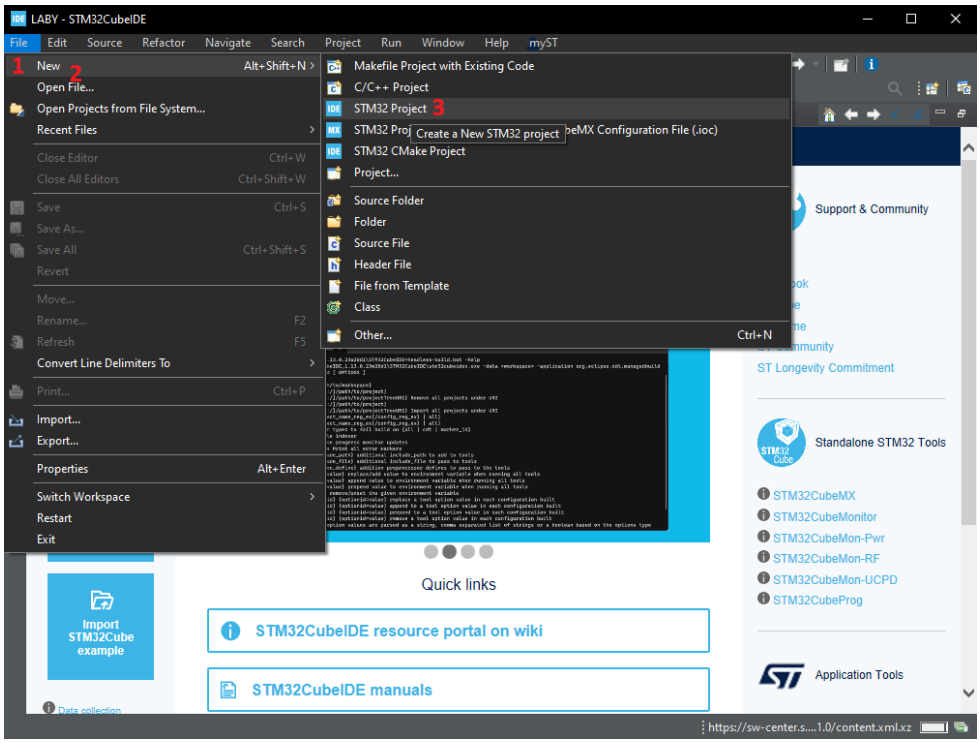
5. Software

A. Utworzenie nowego projektu

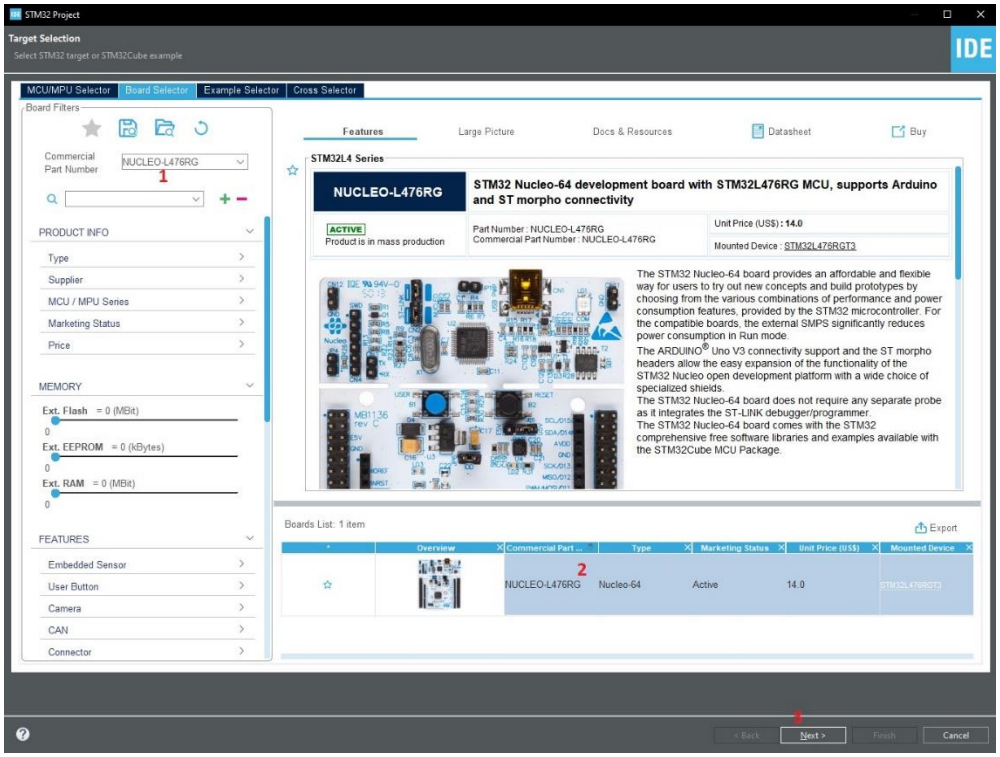
Po uruchomieniu CUBE IDE pojawia się okno wyboru przestrzeni roboczej, wybierz odpowiedni katalog np. SISK\PROJEKT a następnie wciśnij Launch.



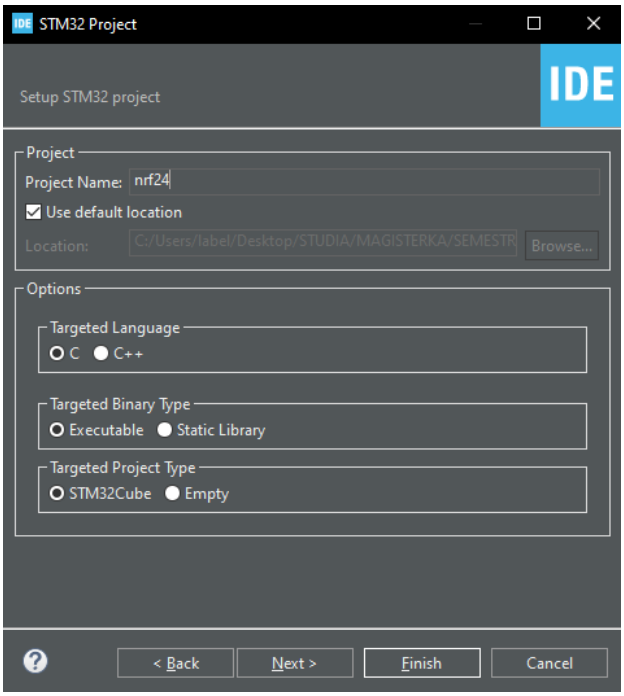
Z menu programu wybierz FILE -> New -> STM32 Project. Jeśli jest to pierwsze uruchomienie programu po instalacji automatycznie zostaną pobrane dodatkowe dane.



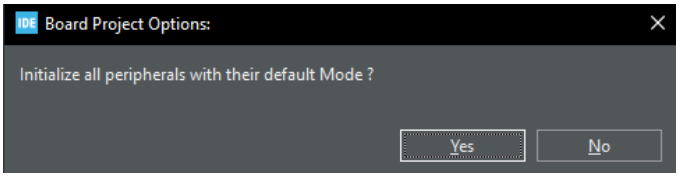
Następnym krokiem jest wybranie zakładki „Board Selector”, wyszukanie wykorzystywanej płytki NUCLEO-L476RG, wybranie jej poprzez wciśniecie na nią i przejście dalej.

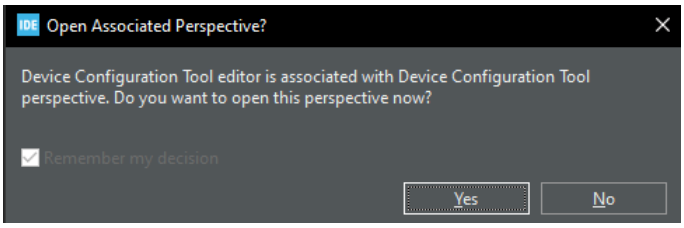


Następnym etapem jest nazwanie projektu. Nazwa może być dowolna np. nrf24. Opcje projektu należy pozostawić „defaultowe” a w razie potrzeby zmienić na takie, które są przedstawione na zrzucie ekranu. Następnie zakończyć tworzenie projektu „Finish”.

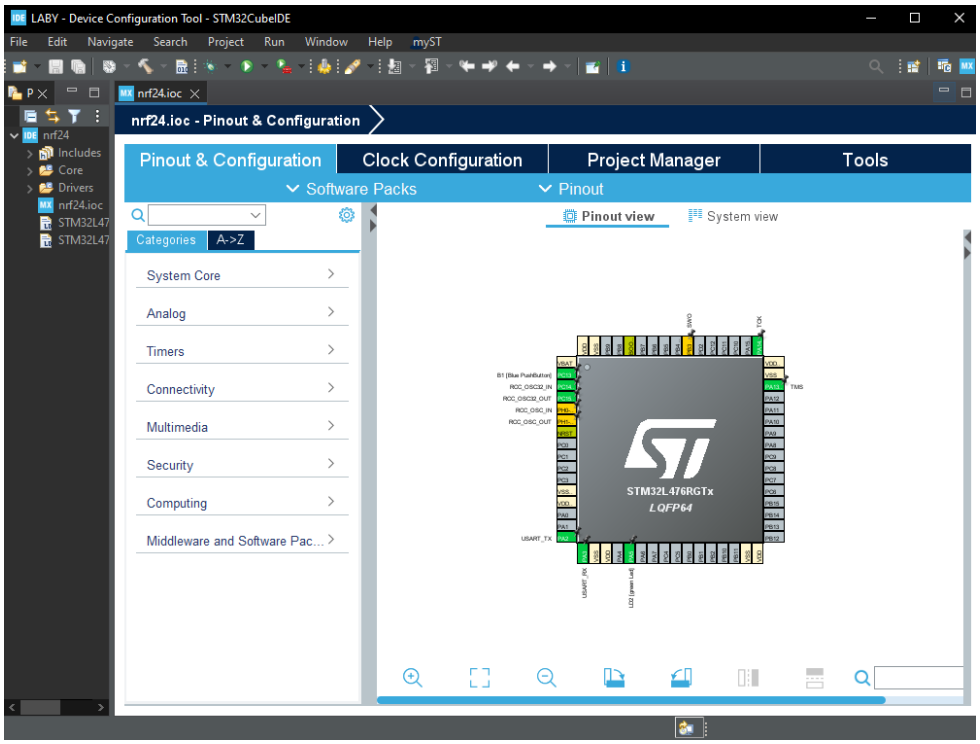


Następnie pojawią się dwa okna jeden po drugim. Pierwsze z nich służy do potwierdzenia zainicjalizowania podstawowych peryferiów płytki a drugie do konfiguracji widoku edytor. Dwukrotnie wciskamy „Yes”.



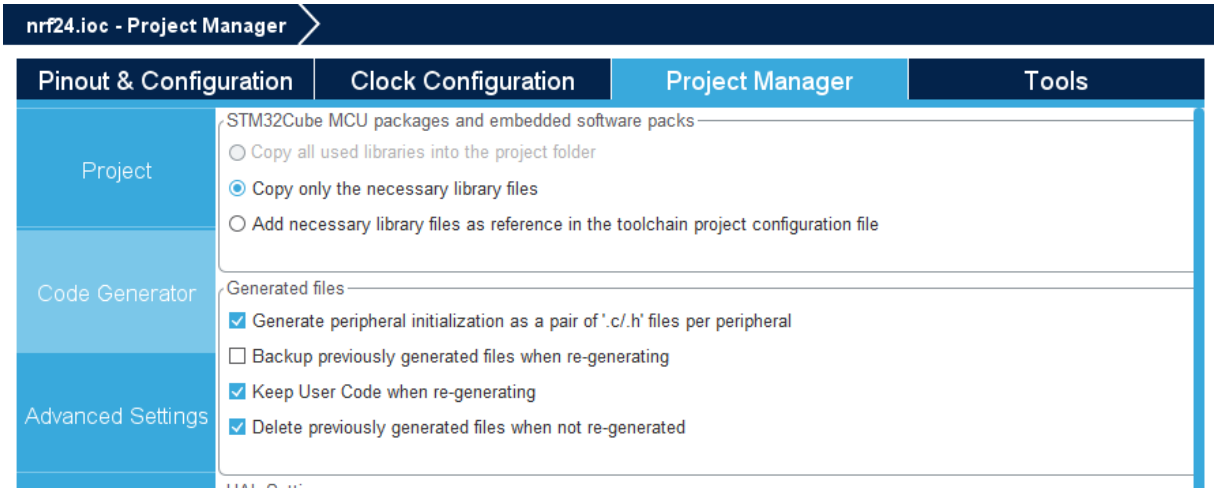


Po wykonaniu powyższych kroków pojawi nam się widok środowiska STM32CubeIDE z widokiem na .ioc, w którym możemy skonfigurować nasz układ.



Zanim jednak zaczniemy coś ustawiać na naszej płytce warto „ulepszyć” nasze środowisko pracy.

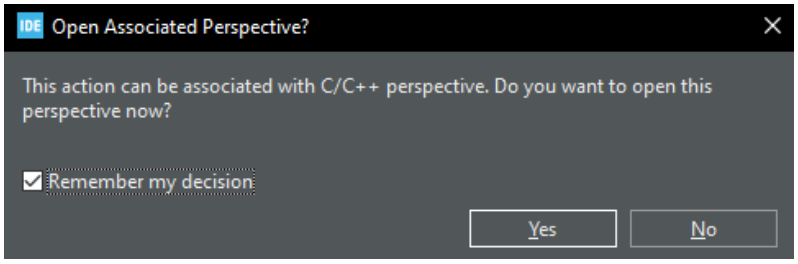
Należy przejść do zakładki project manager, następnie do code generator i w niej zaznaczyć opcję *„Generate peripheral initization as pair of ‘.c/.h’ files per peripheral”.* Dzięki temu po wygenerowaniu kodu nasze pliki nagłówkowe i źródłowe będą znajdować się w oddzielnych folderach.



Po zaznaczeniu używamy kombinacji ctrl+s, aby zapisać ustawienia. Program zapyta nas czy chcemy wygenerować kod, klikamy „Yes”*.

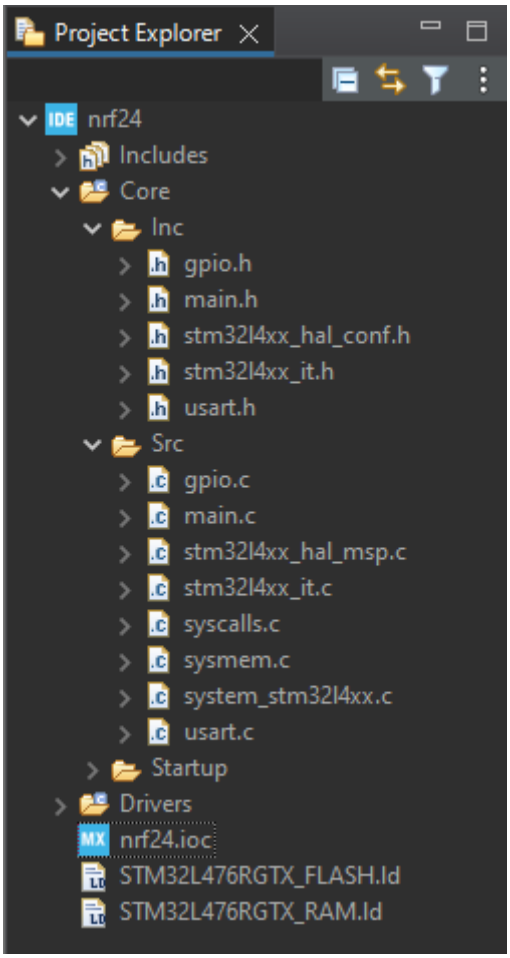
*Można dodatkowo zaznaczyć opcję „Remeber my decision”, wtedy nie będzie pojawiać nam się zapytanie co generację kodu. Niestety jeśli napiszemy kod w złym miejscu utracimy go. Bez zaznaczenia tej opcji mamy dodatkową chwilę „na przemyślenie swojego działania”.

Następnie pojawi nam się okno pytające o perspektywę. Zaznaczamy opcję i klikamy „Yes”.



W przypadku braku automatycznego wygenerowania kodu: Project -> Generate Code

Otrzymana hierarchia plików powinna wyglądać tak jak na poniższym screenshocie:



B. Konfiguracja peryferiów

Kolejnym krokiem jest przejście do pliku .ioc projektu oraz zakładki Pinout Configuration. Na początku będziemy konfigurować SPI do komunikacji z nRF24. Wybieramy SPI2 gdyż port PA5, który wykorzystuje SPI1 jest skonfigurowany jako dioda LED.

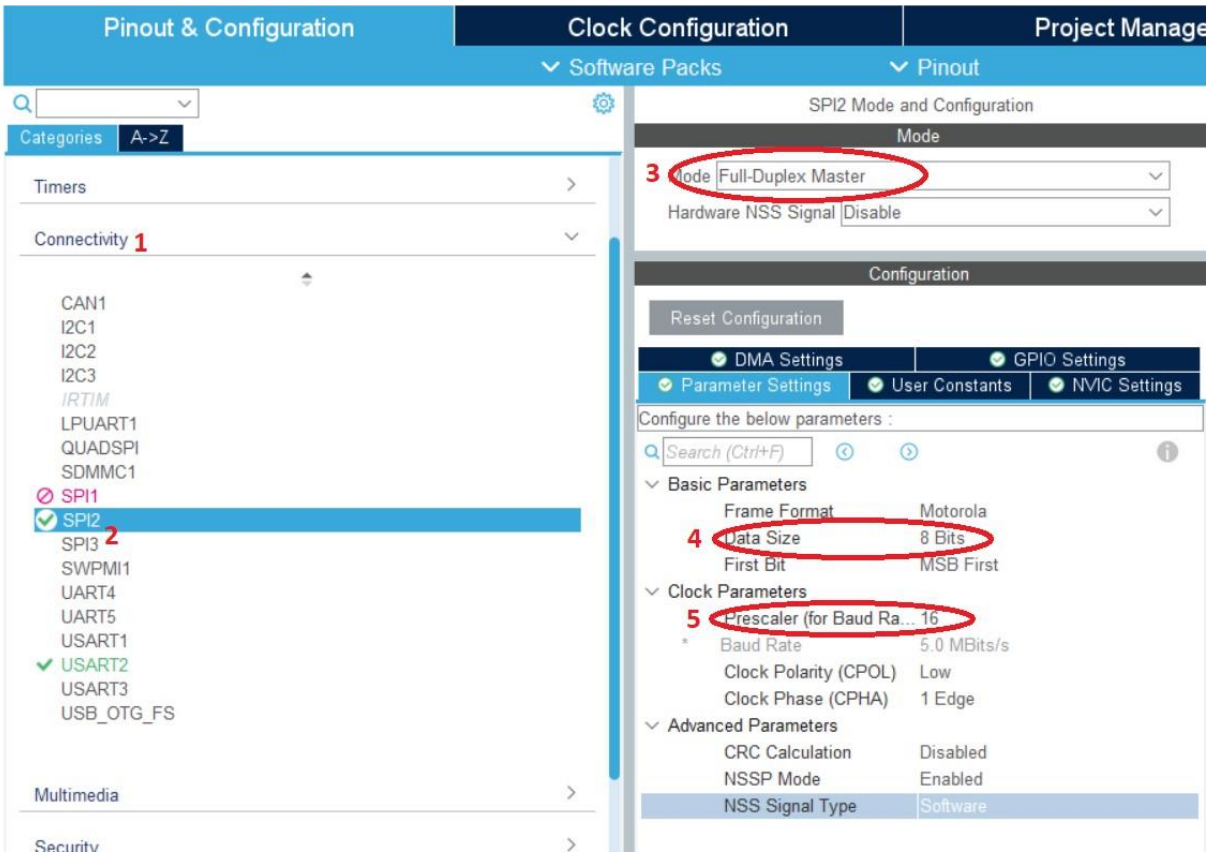
Connctivity -> SPI2 -> Mode -> Full Duplex Master

NIE NALEŻY ZMIENIAĆ KONFIGURACJI PA5, PONIEWAŻ DIODA BĘDZIE WYKORZYSTYWANA W DALSZEJ CZĘSCI PROJEKTU.

Parametry SPI2 do zmiany

Data Size: 8 Bits

Prescaler: 16



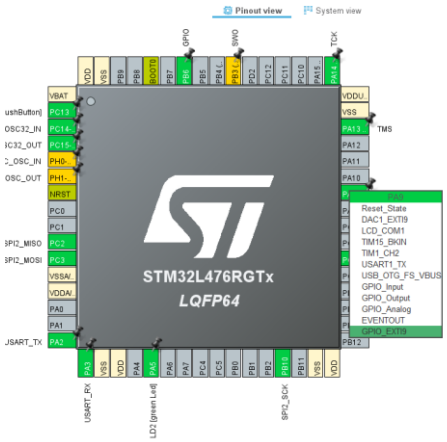
W menu System Core -> GPIO należy skonfigurować piny w następujący sposób, dodając odpowiednie etykiety oraz zweryfikować zgodność pinów SPI2.

Aby dokonać konfiguracji pinów należy przejść do „Pinout View”, odszukać odpowiedniego pinu i wybrać jego funkcjonalność oraz dodać etykiety:

PA9 - GPIO_EXTI9 – „NRF24_IRQ”

PB6 – GPIO_OUTPUT – „NRF24_CE”

PC7 – GPIO_OUTPUT – „NRF24_CSN”



GPIO Mode and Configuration

Configuration

Group By Peripherals

GPIO | Single Mapped Signals | RCC | SPI | SYS | USART | NVIC

Search Signals

Search (Ctrl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PA5	n/a	Low	Output Pus...	No pull-up a...	Low	n/a	LD2 [green ...	☑
PA9	n/a	n/a	External Int...	No pull-up a...	n/a	n/a	NRF24_IRQ	☑
PB6	n/a	Low	Output Pus...	No pull-up a...	Low	Disable	NRF24_CE	☑
PC7	n/a	Low	Output Pus...	No pull-up a...	Low	n/a	NRF24_CSN	☑
PC13	n/a	n/a	External Int...	No pull-up a...	n/a	n/a	B1 [Blue Pu...	☑

GPIO Mode and Configuration

Configuration

Group By Peripherals

GPIO | Single Mapped Signals | RCC | SPI | SYS | USART | NVIC

Search Signals

Search (Ctrl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PB10	SPI2_SCK	n/a	Alternate F...	No pull-up a...	Very High	n/a		☐
PC2	SPI2_MISO	n/a	Alternate F...	No pull-up a...	Very High	n/a		☐
PC3	SPI2_MOSI	n/a	Alternate F...	No pull-up a...	Very High	n/a		☐

Po ustawieniu pinoutu należy przegenerować kod wciskając symbol młotka  .

C. Import biblioteki

W projekcie wykorzystamy gotową bibliotekę do modułu nRF24 stworzoną przez Mateusza Salamona. Należy pobrać pliki dla nadajnika lub odbiornika, w zależności od tego którą część realizuje twoja grupa.

DLA ZESPOŁÓW TX:

Biblioteka dla **TX** jest dostępna pod tym ADRESEM.
Należy przekopiować podkreślone pliki:

Z folderu Core/Inc/nRF24/

nRF24.h

nRF24_Defs.h

Z folderu Core/Src/nRF24/

nRF24.c

oraz wkleić je w analogiczne miejsca poprzez eksplorator katalogów Windows.

DLA ZESPOŁÓW RX:

Biblioteka dla **RX** jest dostępna pod tym ADRESEM.

Należy przekopiować podrkeślone pliki:

Z folderu Core/Inc/nRF24/

nRF24.h

nRF24_Defs.h

Z folderu Core/Src/nRF24/

nRF24.c

oraz wkleić je w analogiczne miejsca poprzez eksplorator katalogów Windows.

UWAGA DLA OBU ZESPOŁÓW:

Plik nrf24.c zawiera inną hierarchię plików załączanych plików nagłówkowych.

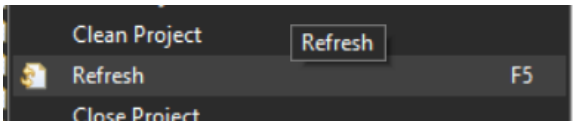
Należy zamienić z:

```
#include "nRF24/nRF24.h"
#include "nRF24/nRF24_Defs.h"
```

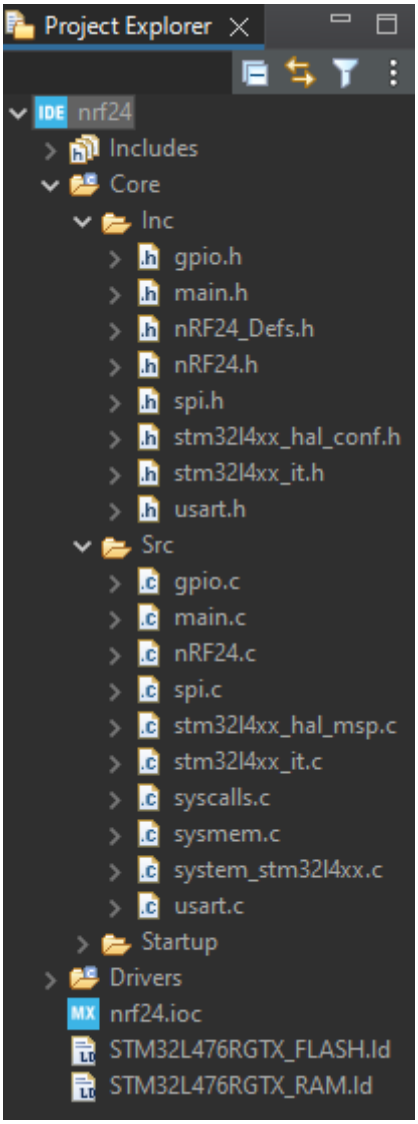
Na:

```
#include "nRF24.h"
#include "nRF24_Defs.h"
```

Po przekopiowaniu plików należy odświeżyć projekt w CubeIDE poprzez wciśniecie PPM->refresh lub wciśnięcie przycisku F5.



Struktura plików powinna wyglądać następująco (dla obu zespołów):



D. Kod dla TX

Plik main.c

Zaimportowanie plików nagłówkowych oraz deklaracja odpowiednich zmiennych:

```
#include "nRF24_Defs.h"
#include "nRF24.h"
volatile uint8_t nrf24_rx_flag, nrf24_tx_flag, nrf24_mr_flag;
uint8_t Nrf24_Message[NRF24_PAYLOAD_SIZE];
uint8_t Message[32];
uint8_t MessageLength;
```

Umożliwienie pisania na UART:

```
//UART///
int __io_putchar(int ch){
  if (ch == '\n') {
    __io_putchar('\r');
  }
  HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
  return 1;
}
```

Inicjalizacja nRF24 (w funkcji main) oraz nadanie adresów:

```
  nRF24_Init(&hspi2);
  nRF24_SetRXAddress(0, "NadX");
  nRF24_SetTXAddress("OdbX");
  nRF24_TX_Mode();
  uint8_t i;
```

W pętli głównej programu:

```
        for(i=0; i<5; i++)
        {
              MessageLength = sprintf(Message, "%d", i );
              nRF24_WriteTXPayload(Message);
              HAL_Delay(1);
              nRF24_WaitTX();
              HAL_Delay(1000);
        }
```

UWAGA: nie jest to kompletny i działający kod, twoim zadaniem jest uzupełnienie go o odpowiednie adresy.

E. Kod dla RX

Plik main.c

Zaimportowanie plików nagłówkowych oraz deklaracja odpowiednich zmiennych:

```
#include "stdio.h"
#include "nRF24_Defs.h"
#include "nRF24.h"
volatile uint8_t nrf24_rx_flag, nrf24_tx_flag, nrf24_mr_flag;
uint8_t Nrf24_Message[NRF24_PAYLOAD_SIZE];
uint8_t Message[32];
uint8_t MessageLength;
```

Funkcja helper do uart:

```
int __io_putchar(int ch){
  if (ch == '\n') {
    __io_putchar('\r');
  }
  HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
  return 1;
}
```

Inicjalizacja nRF24 (w funkcji main) oraz nadanie adresów:

```
  nRF24_Init(&hspi2);
  nRF24_SetRXAddress(0, "OdbX");
  nRF24_SetTXAddress("NadX");
  nRF24_RX_Mode();
```

do głównej pętli while należy wkleić warunek:

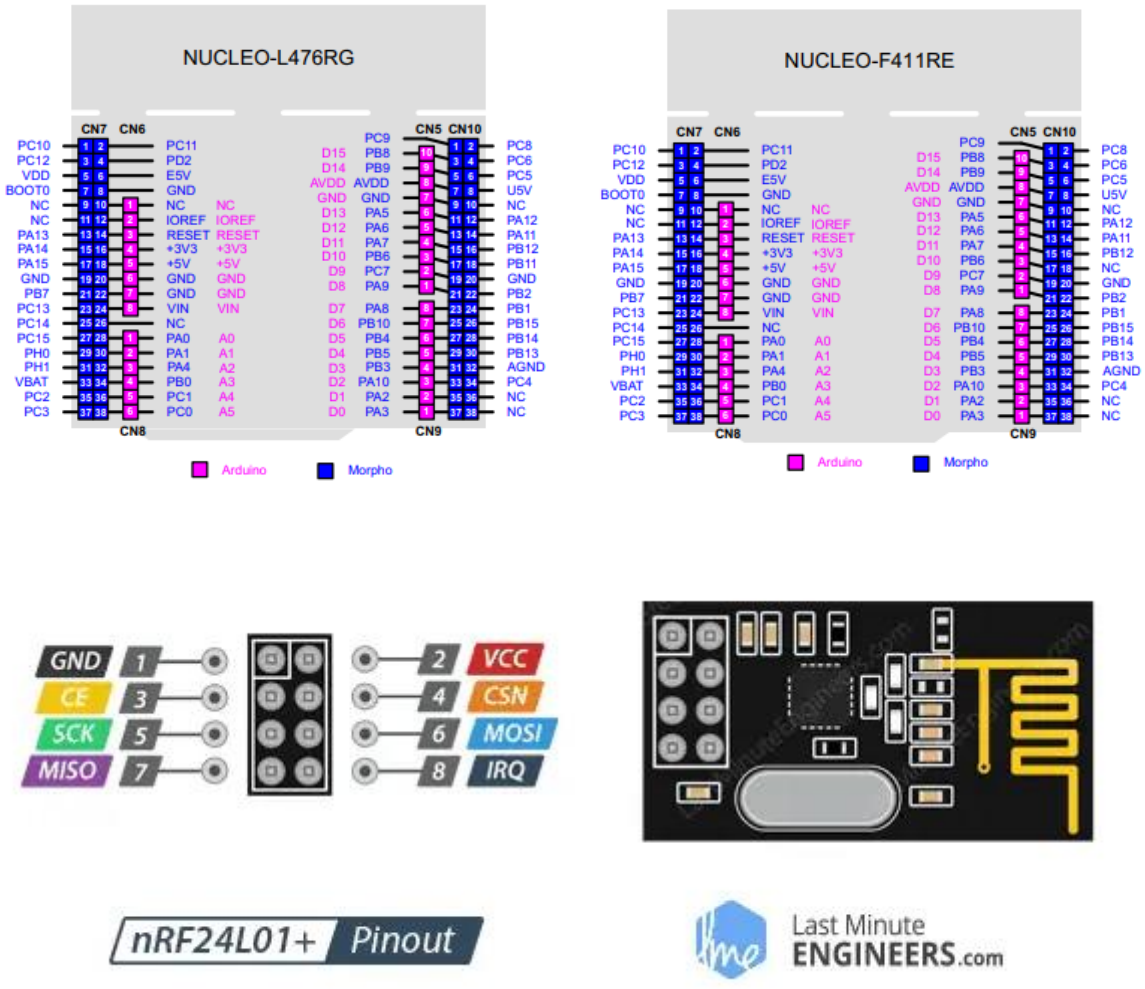
```
while (1)
  {
        if(nRF24_RXAvailible())
        {
              nRF24_ReadRXPaylaod(Nrf24_Message);
              MessageLength = sprintf(Message, "%c\n\r", Nrf24_Message[0]);
              HAL_UART_Transmit(&huart2, Message, MessageLength, 1000);

        }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
```

UWAGA: nie jest to kompletny i działający kod, twoim zadaniem jest uzupełnienie go o odpowiednie adresy.

6. Hardware

Pod ADRESEM znajduje się nota katalogowa, w której na stronie 28 i późniejszej widnieją schematy wyprowadzeń połączeń z różnych płytek NUCLEO. Należy odszukać wykorzystywany w projekcie model oraz podłączyć go zgodnie z zadeklarowanymi portami z nRF24.

Dla przypomnienia napięcie zasilające nRF24 wynosi 3.3V.



Rysunek 7. Schemat wyprowadzeń nRF24

Zródło: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/

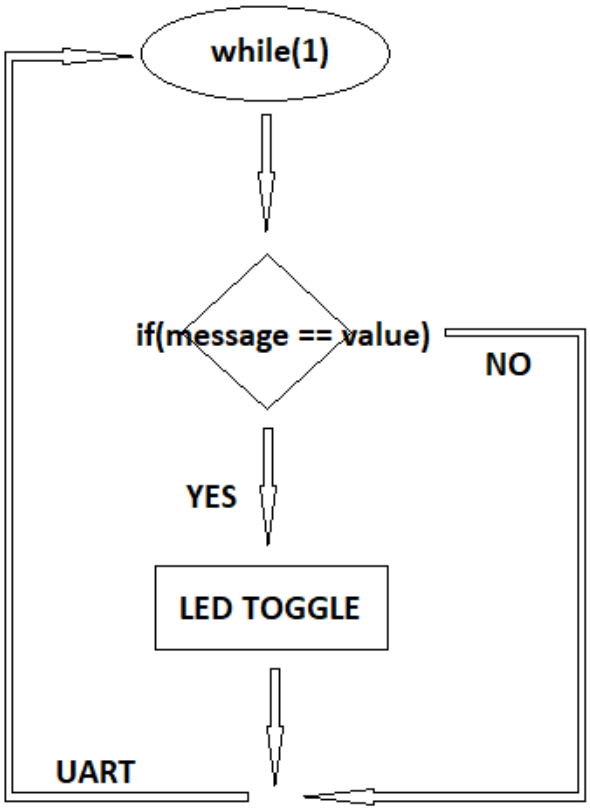
7. Zadania do wykonania

WERSJA ROZSZERZONA v1 (3.0 😊)

Dopisanie kodu w obu płytkach, tak aby była możliwość wysłania informacji (TX), dzięki której następuje przełączanie stanu diody LD2 (PA5) *ang.toggle* w płytce drugiego zespołu (RX).

W tym celu należy uzgodnić z drugim zespołem w parze jaką wartość wysyłamy oraz oczekujemy; sugerujemy aby wybrać liczbę z zakresu 6-9, 0x39 (ASCII) = 9(10).
Z racji nierównomiernej ilości kodu między RX i TX należy współpracować wewnątrz pary zespołu.

<u>Podpowiedź RX:</u>
W while dopisać if'a, który sprawdzałby wartość odebraną oraz ewentualnie zanegował stan diody LED.

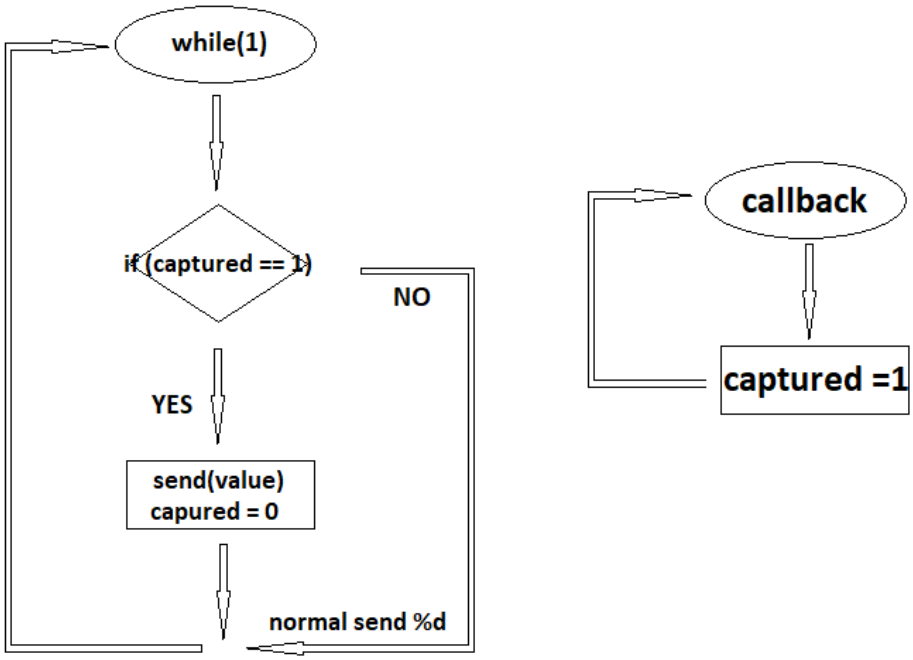



```
if(nRF24_RXAvailible())
{
	nRF24_ReadRXPaylaod(Nrf24_Message);
	if(Message[0]== # TODO){	//sprawdzenie ustalonej wartosci
		# TODO			//zmiana stanu diody LED
	}
	MessageLength = sprintf(Message, "%c\n\r", Nrf24_Message[0]);
	HAL_UART_Transmit(&huart2, Message, MessageLength, 1000);
}
```

Podpowiedź TX:

Włączyć możliwość przerwań GPIO. Wygenerować przerwania od przycisku B1, zmieniłoby wartość nowo utworzonej zmiennej odpowiedzialnej za maszynę stanów w while.

W while dopisać if'a (pojedyncza maszyna stanów), odpowiedzialnego za wywołanie funkcji wysyłającej.



Do wysłania informacji można wykorzystać przygotowaną funkcje:

```c
uint8_t message[32];
void send_light(uint8_t data)
{
    sprintf(message, "%d", data);
    nRF24_WriteTXPayload(message);
    HAL_Delay(1);
    nRF24_WaitTX();
}
```

```c
#define BUTTON_DEBOUNCE_TIME 250

uint32_t previous_time = 0;
uint32_t current_time = 0;
uint32_t button_caputered = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    current_time = HAL_GetTick();                    //time reading
    if((GPIO_Pin == B1_Pin) && (current_time-previous_time)>BUTTON_DEBOUNCE_TIME)
    {
       button_caputered = #TODO;
    }
    previous_time = current_time;                    //previous time assignment
}
```

```c
uint8_t message[32];
void send_light(uint8_t data)
{
    sprintf(message, "%d", data);
    nRF24_WriteTXPayload(message);
    HAL_Delay(1);
    nRF24_WaitTX();
}
```

```c
        for(i=0; i<5; i++)
        {
              //check captured of button
              if(button_caputered == #TODO){
                  //temporary value to send
                  uint8_t tmp = #TODO;
                  send_light(tmp);
                  button_caputered = #TODO;
              }
              MessageLength = sprintf(Message, "%d", i );
              nRF24_WriteTXPayload(Message);
              HAL_Delay(1);
              nRF24_WaitTX();
              HAL_Delay(1000);
        }
```

WERSJA ROZSZERZONA v2

Komunikacja 2 do 1; 3 do 1; 4 do 1 (tryb pracy broadcast).

WERSJA ROZSZERZONA v3

Każdy z każdym (tryb pracy multicast).

PRZYKŁADOWY KOD DLA WERSJI ROZSZERZONEJ v1

TX:

```
#define BUTTON_DEBOUNCE_TIME 250
```

```
uint32_t previous_time = 0;
uint32_t current_time = 0;
uint32_t button_caputered = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    current_time = HAL_GetTick();                                   //time
reading
    if((GPIO_Pin == B1_Pin) && (current_time -
previous_time)>BUTTON_DEBOUNCE_TIME)
    {
      button_caputered = 1;
    }
    previous_time = current_time;                                   //previous
time assignment
}
```

```
uint8_t message[32];
void send_light(uint8_t data)
{
    sprintf(message, "%d", data);
    nRF24_WriteTXPayload(message);
    HAL_Delay(1);
    nRF24_WaitTX();
}
```

```
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
        for(i=0; i<5; i++)
        {
              //check captured of button
              if(button_caputered == 1){
                  //temporary value to send
                  uint8_t tmp = 9;
                  send_light(tmp);
                  button_caputered = 0;
              }
              MessageLength = sprintf(Message, "%d", i );
              nRF24_WriteTXPayload(Message);
              HAL_Delay(1);
              nRF24_WaitTX();
              HAL_Delay(1000);
        }

  }
```

RX:

```
while (1)
{
	if(nRF24_RXAvailible())
	{
		nRF24_ReadRXPaylaod(Nrf24_Message);
		if(Message[0]==0x39){
			//HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
			HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
		}
		MessageLength = sprintf(Message, "%c\n\r", Nrf24_Message[0]);
		HAL_UART_Transmit(&huart2, Message, MessageLength, 1000);
	}
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```