Temat: Magistrala CAN - transmitter i sniffer

Studia magisterskie rok akademicki: 2023
Elektronika i Telekomunikacja

AGH

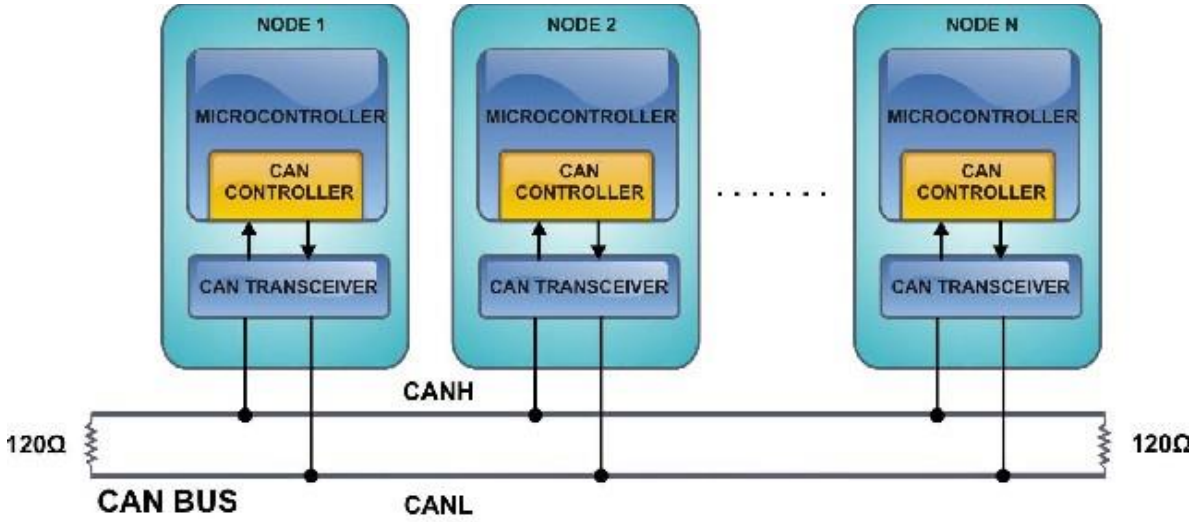
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Tomasz Bednorz
Cezary Szczepański

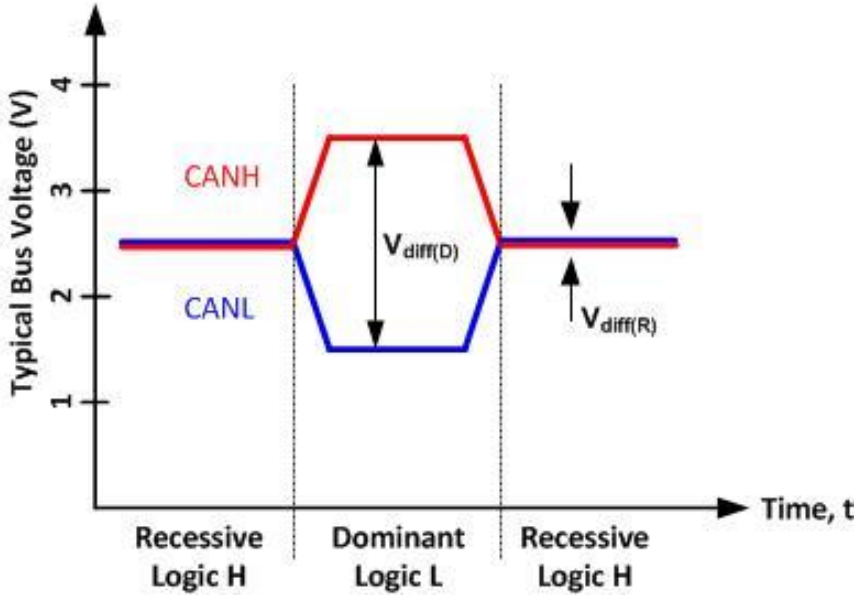
1. Teoria

a) Magistrala CAN

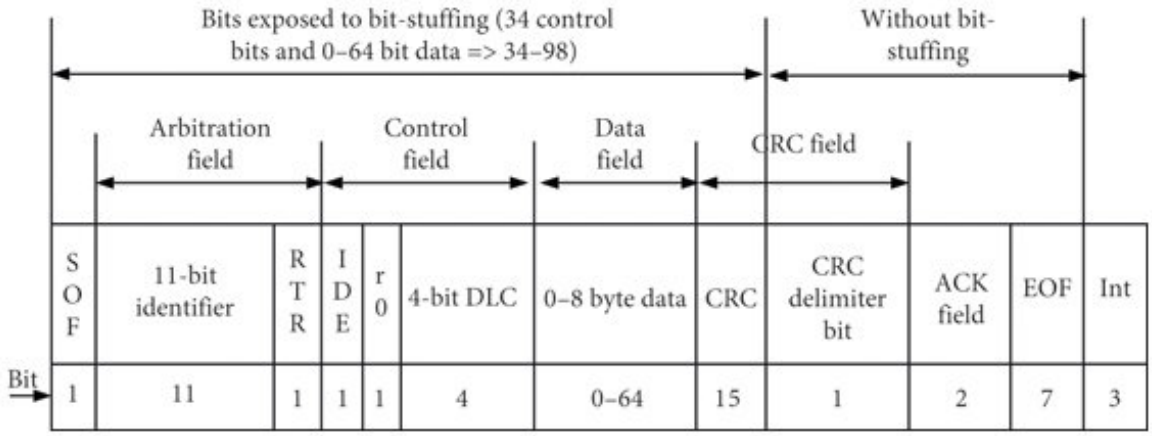
Magistrala CAN nie ma wyodrębnionej jednostki nadrzędnej dlatego należy do grupy magistral typu multi-master. Komunikacja ma charakter rozgłoszeniowy ponieważ komunikaty nadawane na magistralę odbierane są przez wszystkie urządzenia.



Node podpięty do magistrali składa się z urządzenia zawierającego CAN controller oraz najczęściej podpinany jest zewnętrzny CAN transceiver umożliwiający transmisję sygnałów różnicowych, dzięki czemu można uzyskać dużą odporność na zakłócenia, prędkość transmisji danych oraz niezawodność.

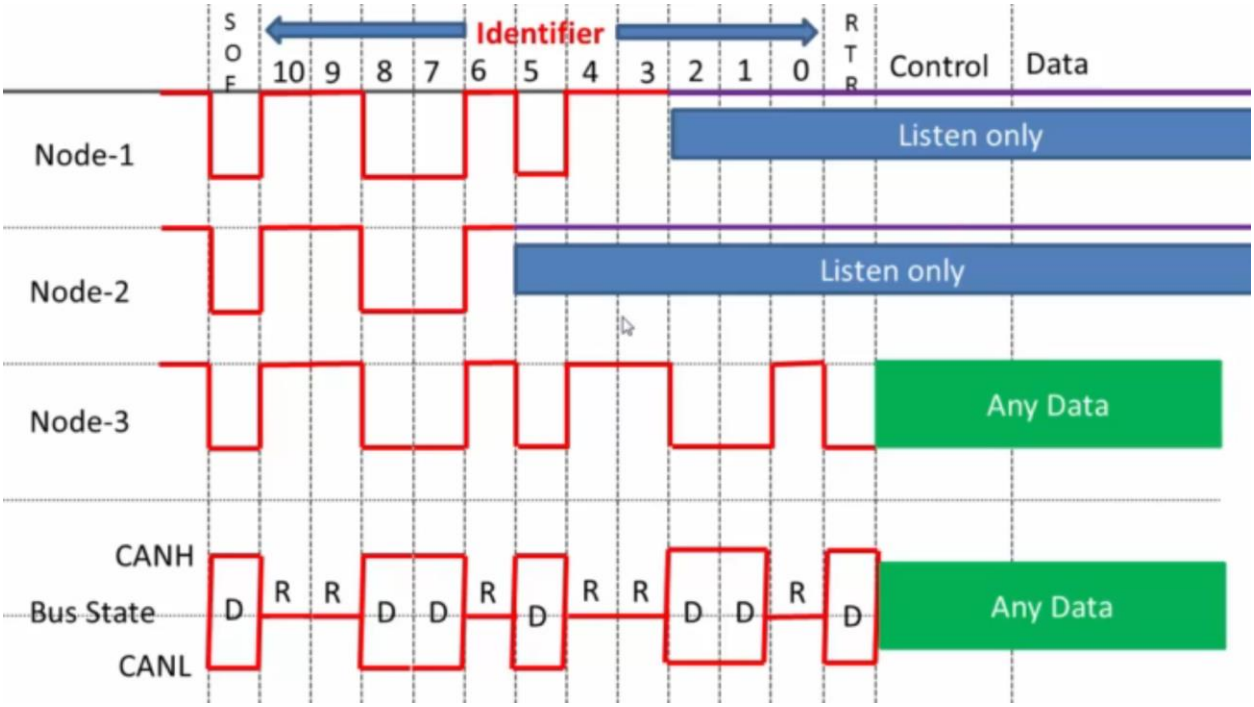


b) Format ramki danych 2.0A

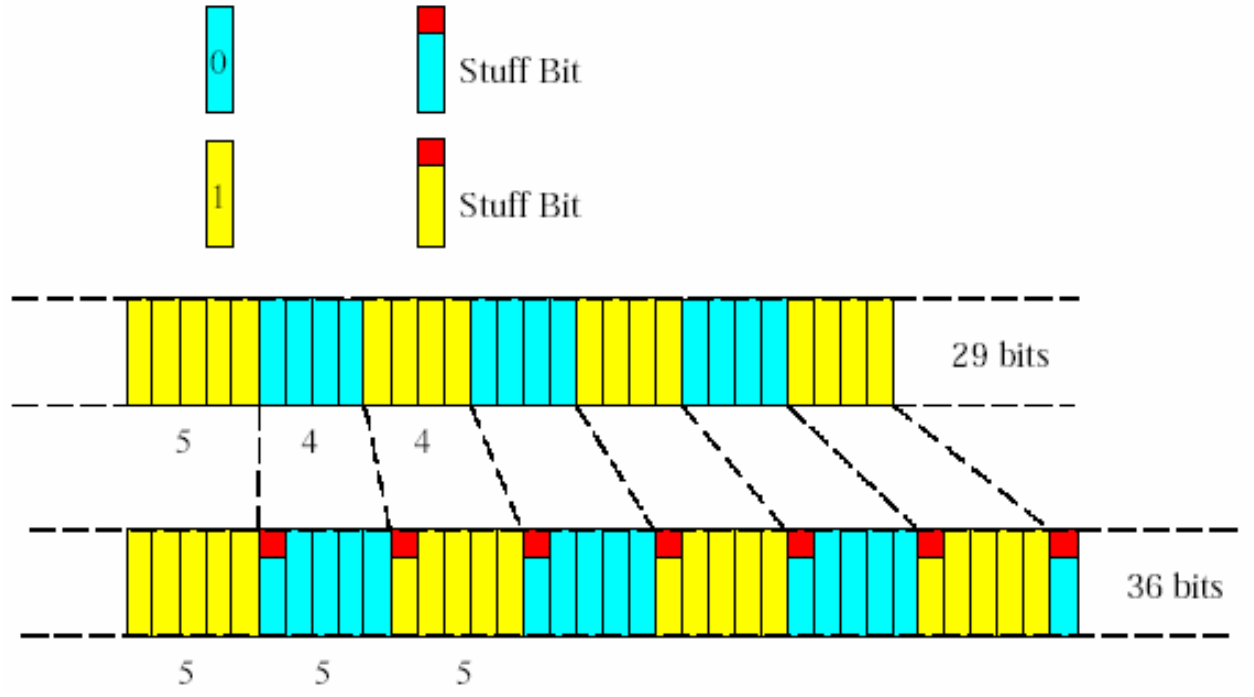


- Start of Frame (SOF): Początek ramki. Jest to sygnał informujący o rozpoczęciu przesyłania danych.
- Identifier Field (ID): To pole zawiera identyfikator, który jednoznacznie identyfikuje ramkę i określa jej priorytet. W magistrali CAN 2.0A identyfikator ma długość 11 bitów.
- Remote Transmission Request (RTR): Określa, czy ramka jest żądaniem transmisji od innego urządzenia (jeśli RTR = 1) czy samymi danymi (jeśli RTR = 0).
- IDE (Identifier extension): bit informujący o rozszerzonym 29 bitowym identyfikatorze.
- (DLC - Data Length Code): ilość transmitowanych bajtów (0-8).
- Data Field (Data): dane.
- CRC Field (Cyclic Redundancy Check): Suma kontrolna, która umożliwia odbiorcy sprawdzenie, czy dane zostały przesłane bez błędów.
- Acknowledge Field (ACK): Potwierdzenie odebrania ramki wystawiane przez pozostałe urządzenia na magistrali.
- End of Frame (EOF): Sygnał kończący ramkę.

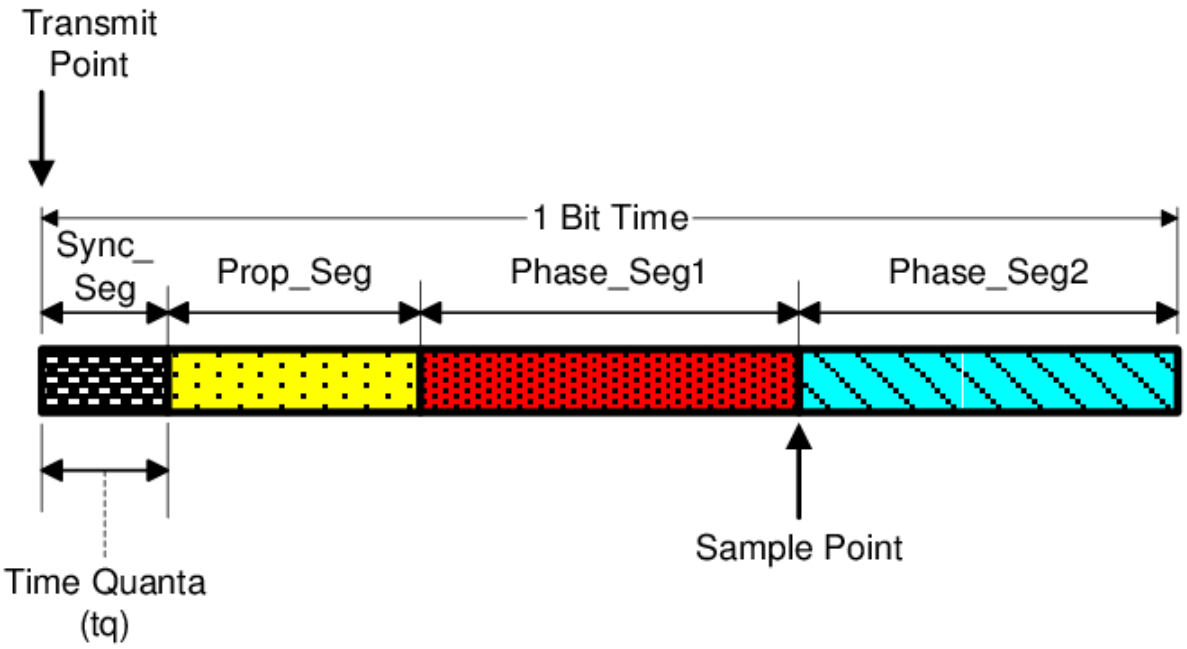
c) Arbitraż



d) Bit stuffing

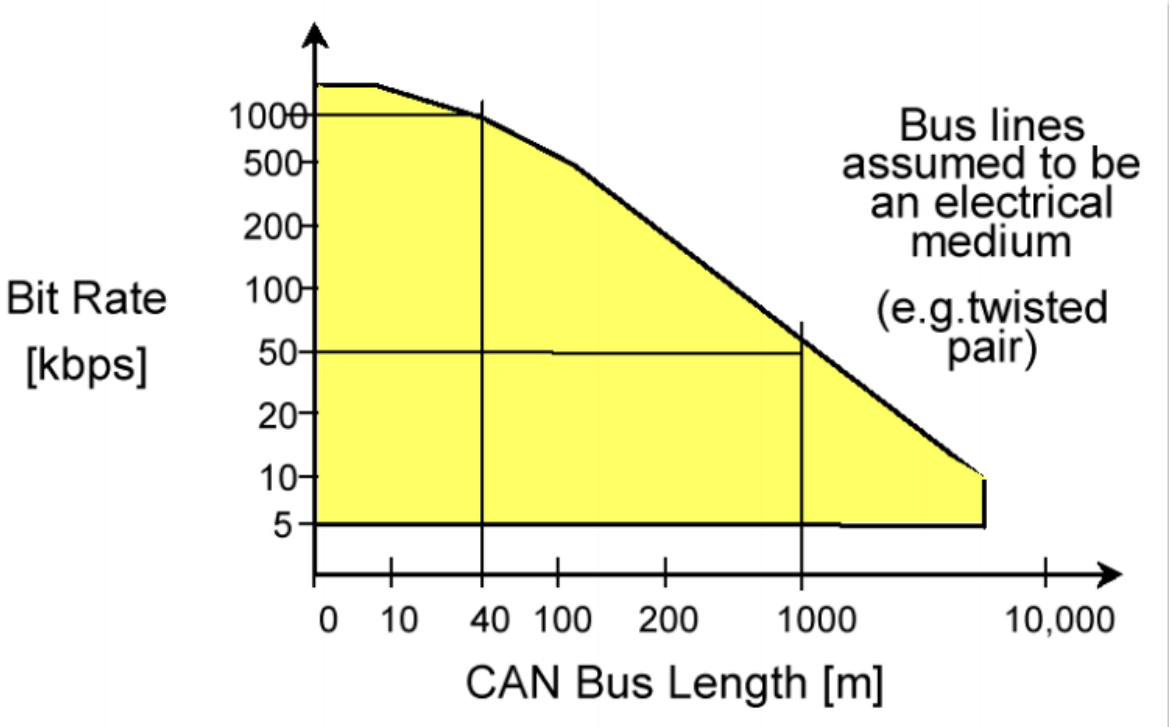


e) Taktowanie bitu



CAN bit time calculation: http://www.bittiming.can-wiki.info/

f) Prędkość transmisji danych



g) bxCAN w STM32F7xx

Reference manual (s.1531):
https://www.st.com/resource/en/reference_manual/rm0410-stm32f76xxx-and-stm32f77xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

40.2 bxCAN main features

- Supports CAN protocol version 2.0 A, B Active
- Bit rates up to 1 Mbit/s
- Supports the Time Triggered Communication option

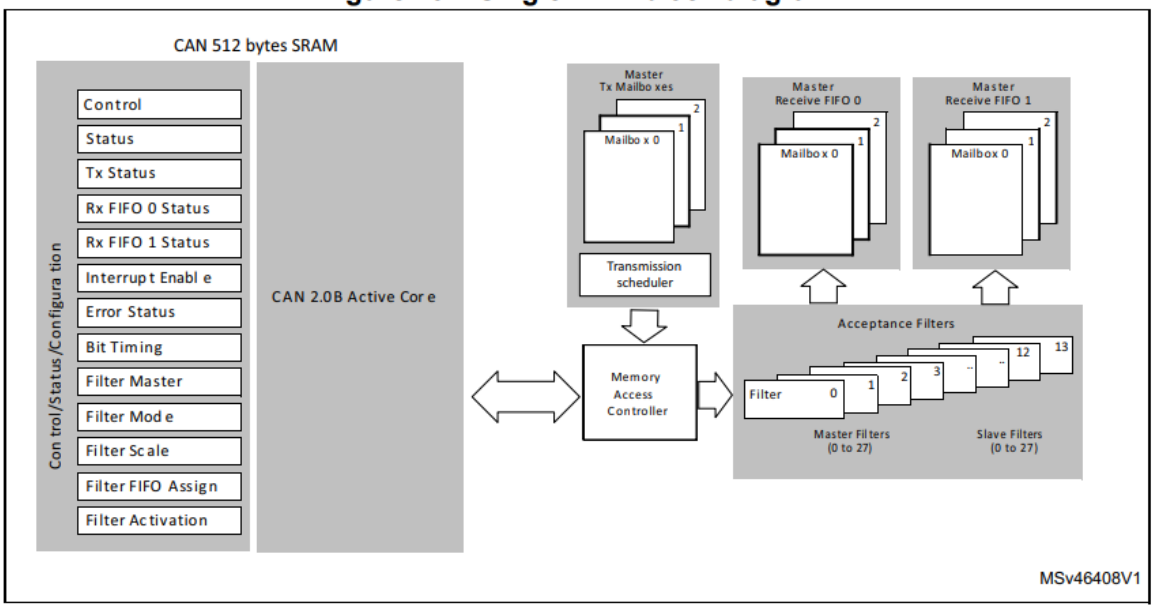
Transmission
- Three transmit mailboxes
- Configurable transmit priority
- Time Stamp on SOF transmission

Reception
- Two receive FIFOs with three stages
- Scalable filter banks:
  - 28 filter banks shared between CAN1 and CAN2 for dual CAN
  - 14 filter banks for single CAN
- Identifier list feature
- Configurable FIFO overrun
- Time Stamp on SOF reception

Time-triggered communication option
- Disable automatic retransmission mode
- 16-bit free running timer
- Time Stamp sent in last two data bytes

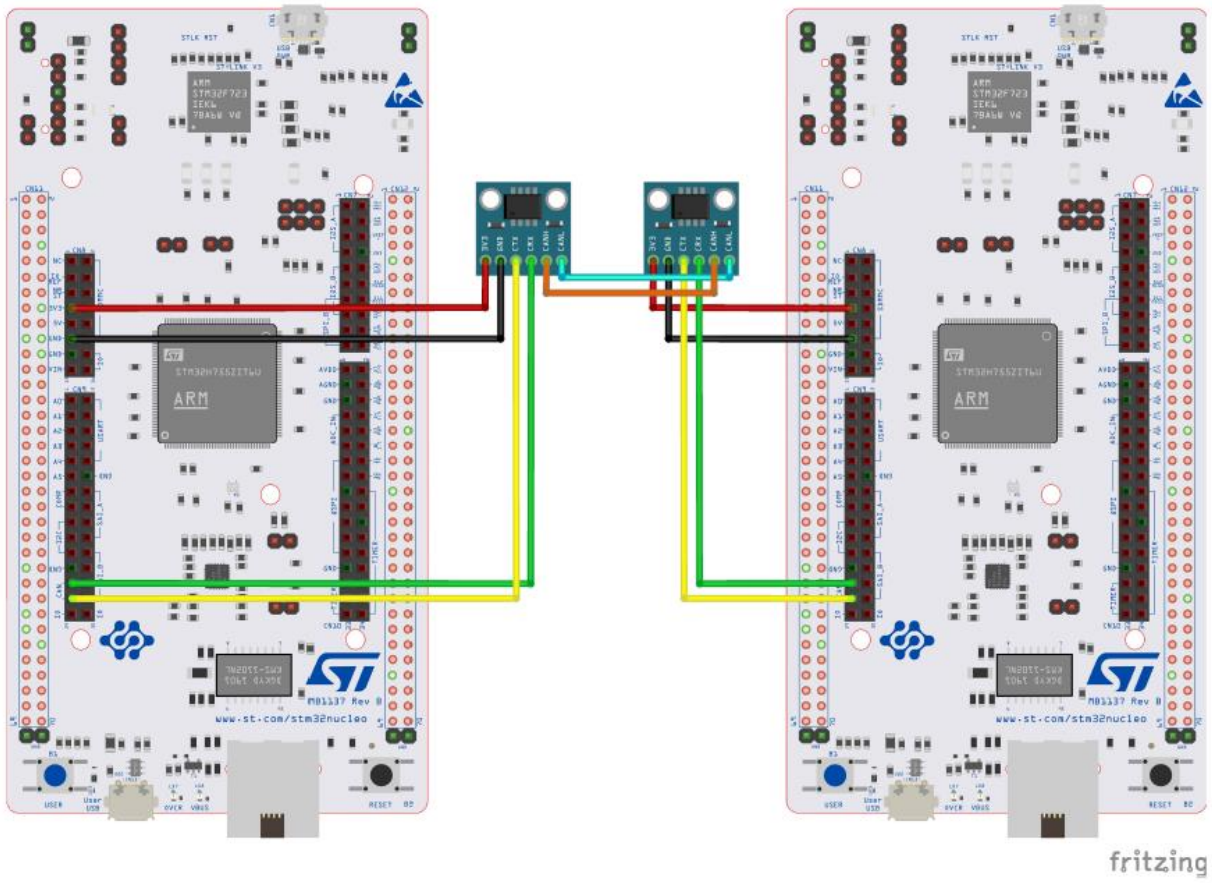
Management
- Maskable interrupts
- Software-efficient mailbox mapping at a unique address space

Figure 494. Single-CAN block diagram



2. Schemat projektu

Realizacja projektu opiera się na dwóch układach NUCLEO od STM oraz zewnętrznych transceiverach. Jedna z płytek odpowiedzialna jest za transmisję danych, a druga za ich nasłuchiwanie na magistrali.



Funkcje transmittera:

- CLI umożliwiający transmisję danych (port szeregowy),
- transmisja danych.

Funkcje sniffera:

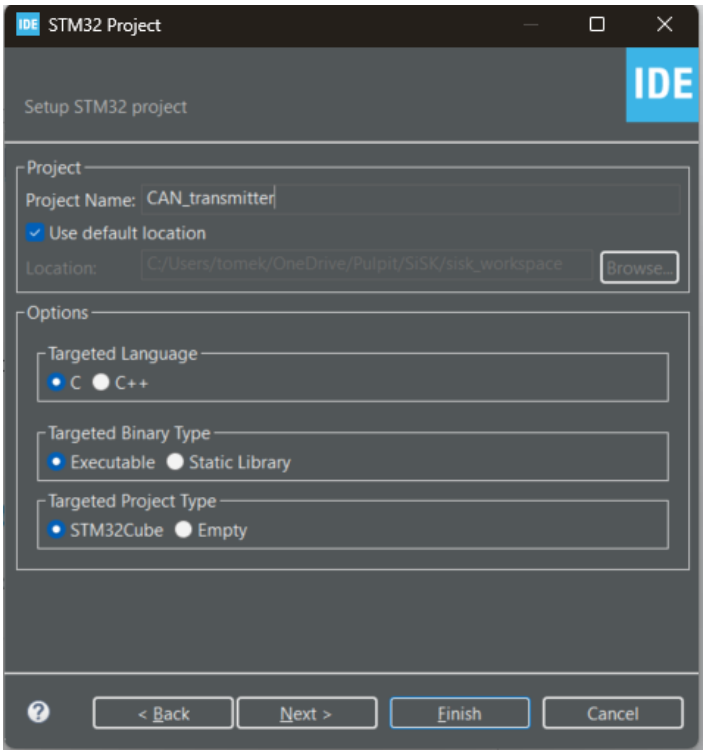
- odbiór danych,
- wyświetlanie odebranych danych za pomocą portu szeregowego.

3. Implementacja CAN transmittera

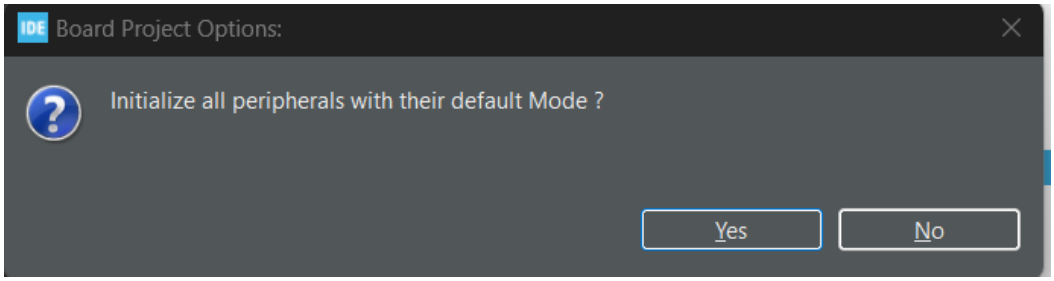
a) Konfiguracja projektu:

Tworzymy nowy projekt. W zakładce “Board Selector” szukamy płytki NUCLEO-F767ZI.

Nadajemy projektowi nazwę, a następnie klikamy “Finish”.

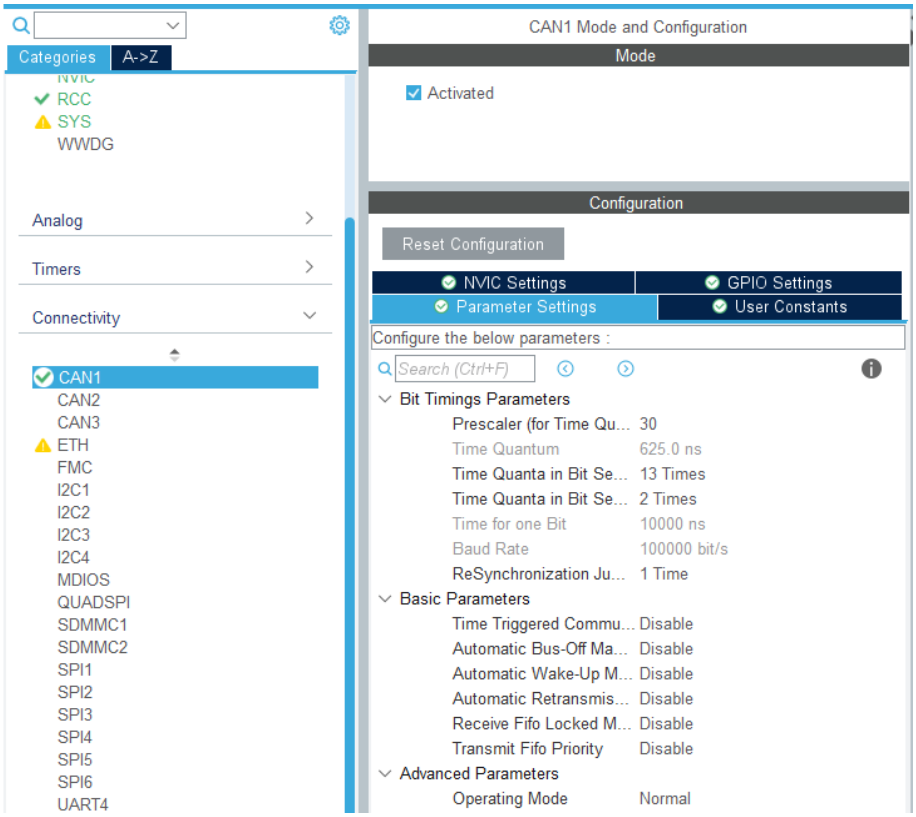


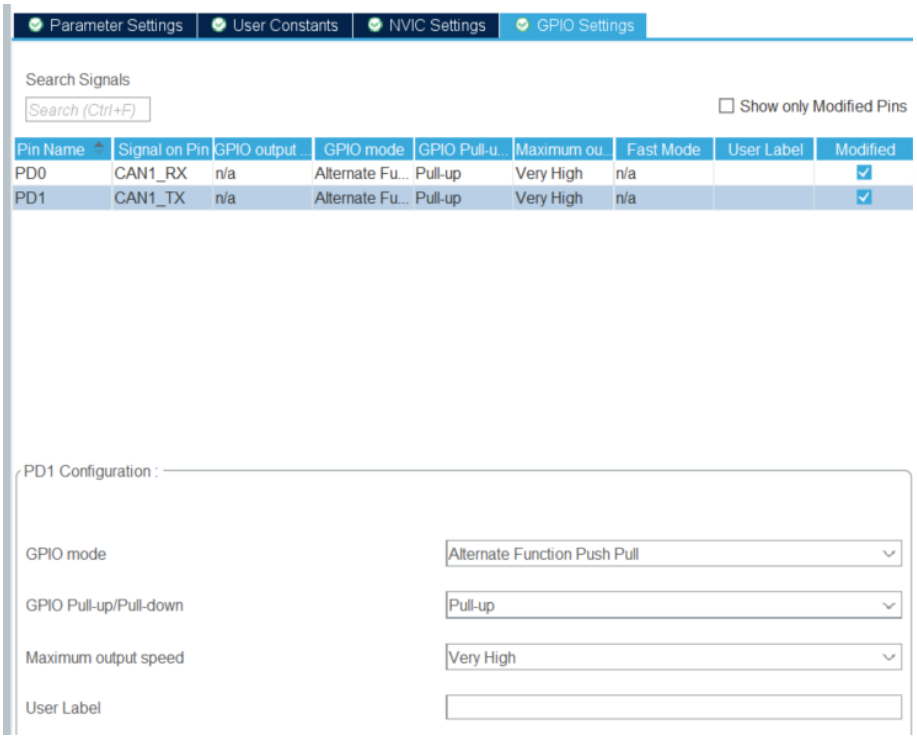
W okienku pytającym o inicjalizację peryferiów w trybie domyślnym klikamy “Yes”.



b) Konfiguracja CAN1

W zakładce “Connectivity” wybieramy “CAN1” oraz konfigurujemy zgodnie z poniższymi rysunkami.

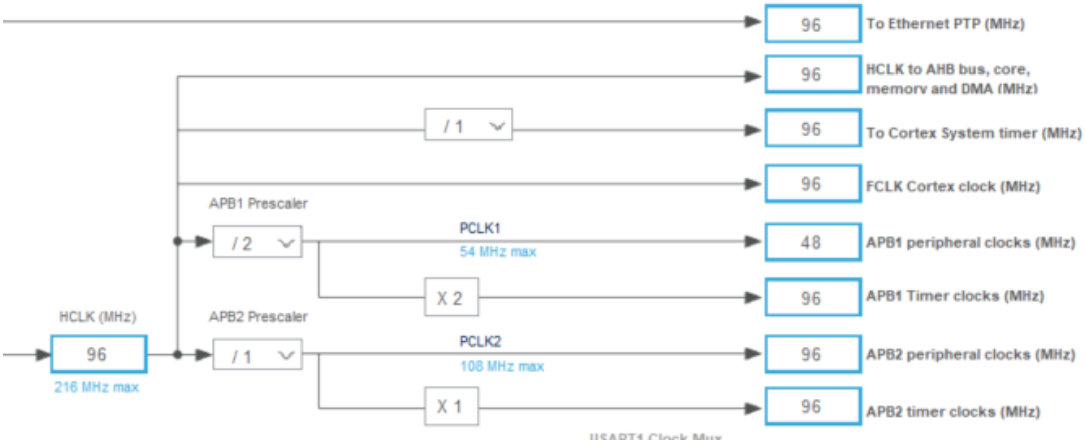




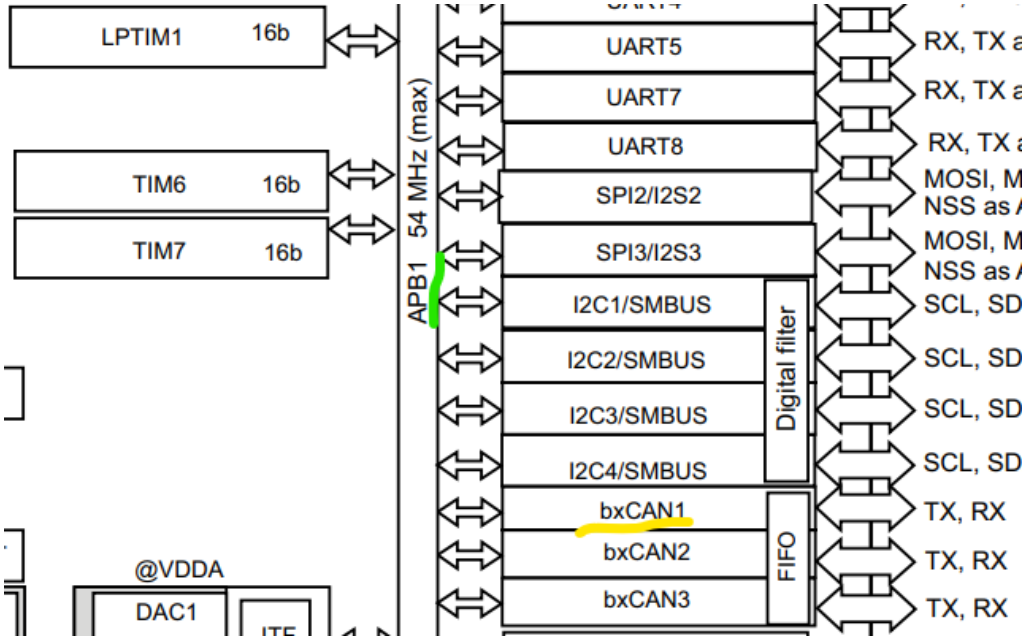
Baud Rate został skonfigurowany na 100000 bit/s. Zostało to osiągnięte poprzez podzielenie częstotliwości 48Mhz na magistrali APB1 przy użyciu Prescalera o wartości 30 oraz przez 16TQ (time quanta) zgodnie z konfiguracją: 16TQ = 1TQ synch seg + 13TQ segment 1 + 2TQ segment 2.

48Mhz / 30 (Prescaler) / 16 (TQ) = 100000 bit/s

Domyślna konfiguracja zegara:

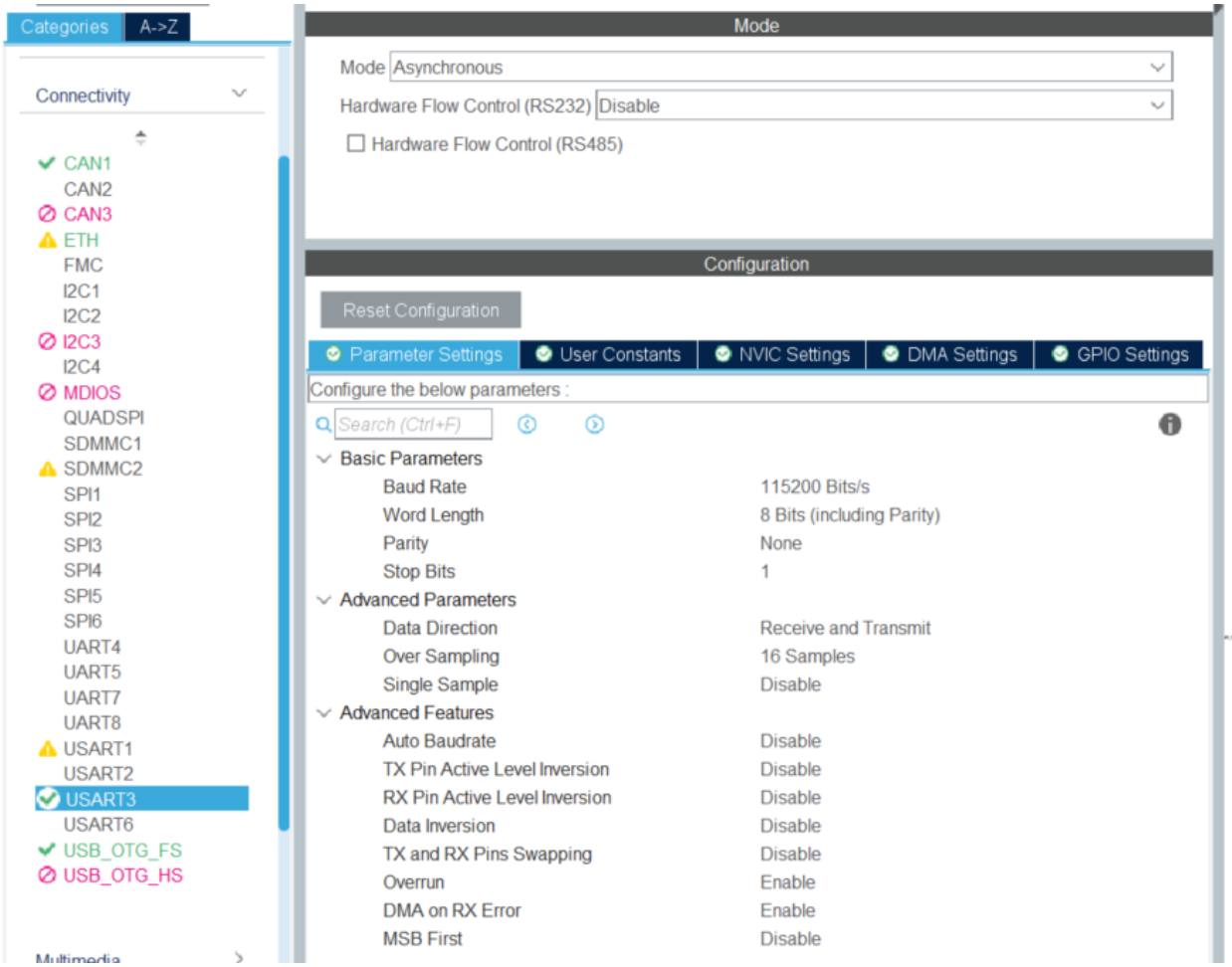


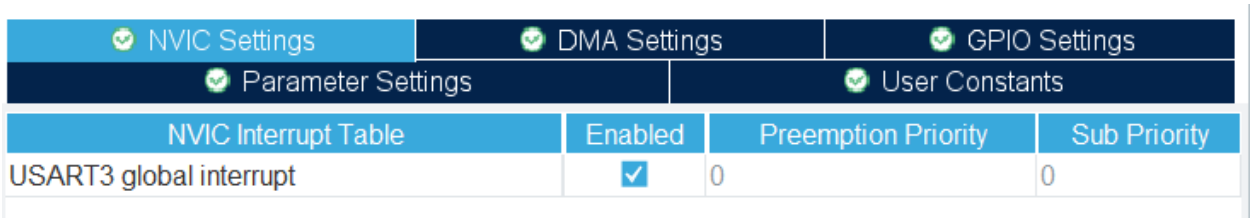
Datasheet (s.20): https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html



c) Konfiguracja USART3

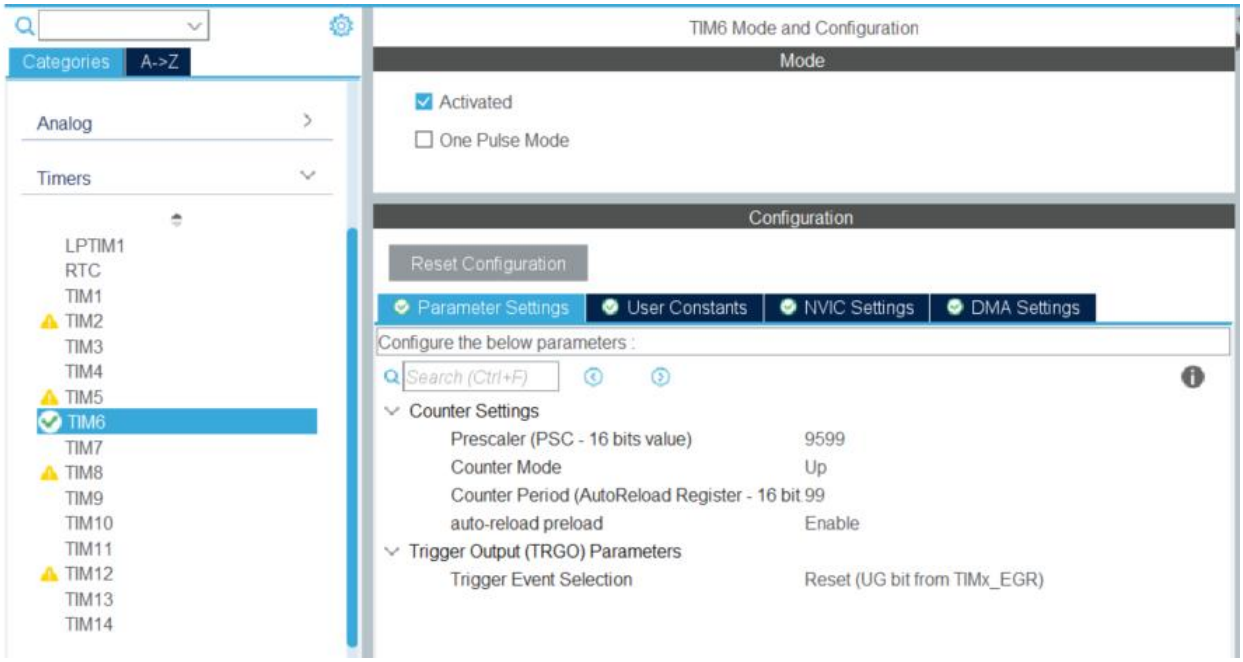
USART3 na płytce NUCLEO jest połączony z ST-LINKiem, a następnie wyprowadzony na przewód USB.
W zakładce “Connectivity” wybieramy “USART3” oraz konfigurujemy zgodnie z poniższymi rysunkami.

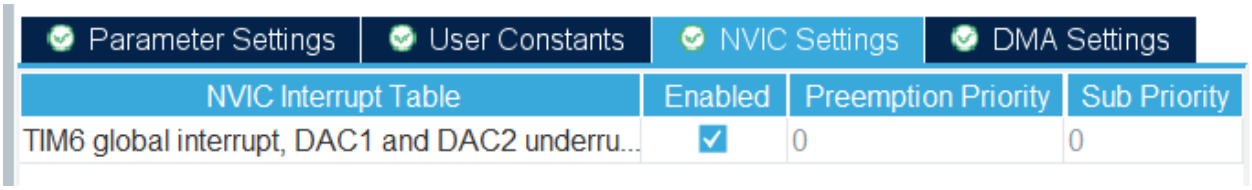




d) Konfiguracja TIM6

Zadaniem timera jest możliwość cyklicznego wysyłania ramek na magistralę CAN. W zakładce “Timers” wybieramy “TIM6” oraz konfigurujemy zgodnie z poniższymi rysunkami.

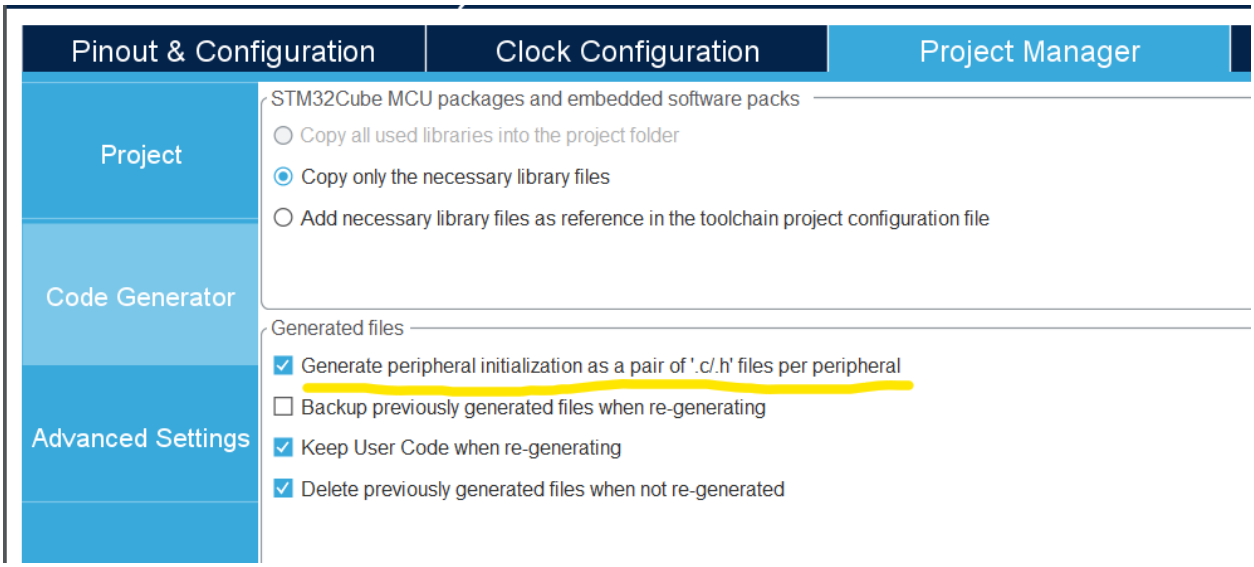




Co 10ms będzie generowane przerwanie, w trakcie którego transmitowane będą ramki zaplanowane przez użytkownika przy pomocy CLI.

e) Generowanie kodu:

W zakładce “Project Manager” warto zaznaczyć poniższą opcję, aby konfiguracja każdego z peryferiów została wygenerowana do osobnych plików.

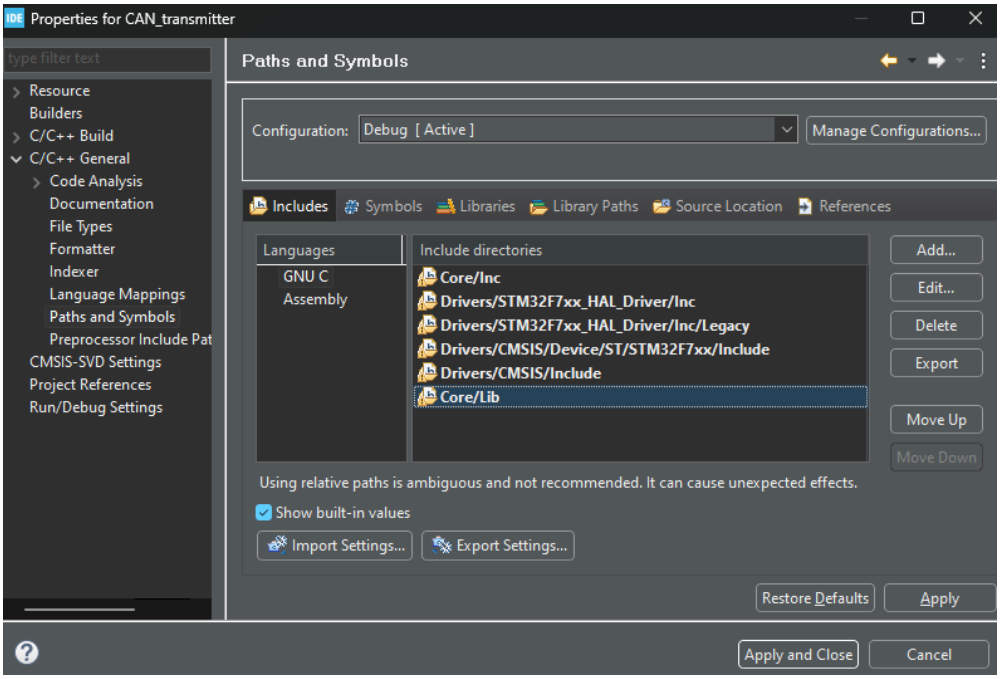


Po kliknięciu “Ctrl+s” należy wygenerować kod.

f) Dodanie biblioteki

Bibliotekę należy pobrać z githuba: **LINK**
Potrzebny folder do umieszczenia w projekcie znajduje się w folderze:
Libs/CAN_Transmitter

Aby zaimportować bibliotekę do projektu należy kliknąć prawym prawym przyciskiem myszy na projekt, a następnie: “Properties / C/C++ General / Path and Symbols” oraz dodać pobrany folder. Poniżej zaprezentowano zaimportowanie biblioteki po umieszczeniu jej w folderze “Core”.



g) Modyfikacja kodu

Poniżej przedstawione są fragmenty kodu, które należy umieścić w pliku “main.c”.

Importowanie bibliotek:

```
/* USER CODE BEGIN Includes */

#include "app_can.h"
#include "app_uart.h"

/* USER CODE END Includes */
```

Inicjalizacja modułów oraz timera w trybie przerwań:

```
  /* USER CODE BEGIN 2 */

  AppCan_Init();
  AppUart_Init();

  /* Enable TIM6 - task scheduler */
  HAL_TIM_Base_Start_IT(&htim6);

  /* USER CODE END 2 */
```

Implementacja callbacków od przerwań:

- cykliczne wywoływanie zadania z modułu AppCan,
- procesowanie danych przychodzących poprzez port szeregowy.

```
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM6)
  {
    AppCan_Task();
  }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if(huart->Instance == USART3)
  {
    AppUart_ProcessInput();
  }
}

/* USER CODE END 4 *
```

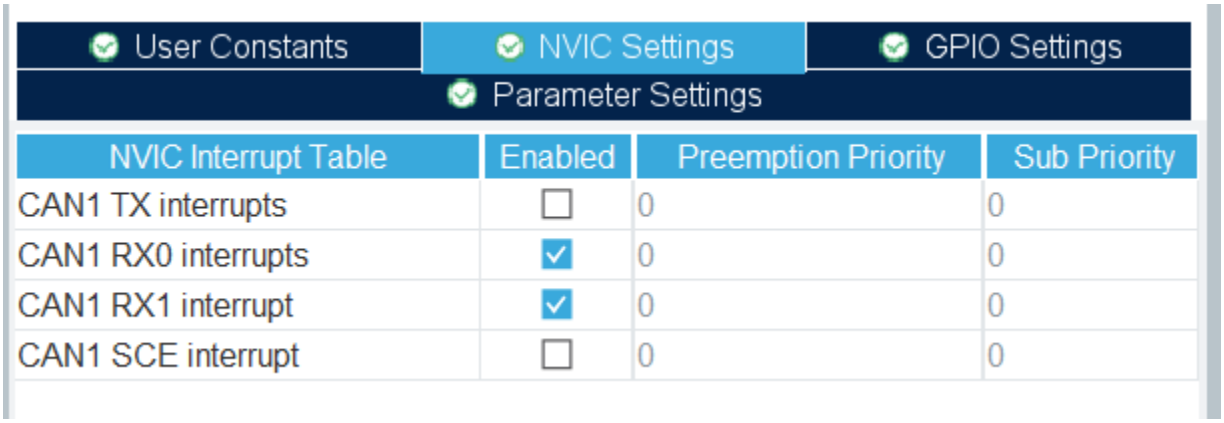
4. Implementacja CAN sniffera

a) Konfiguracja projektu:

Analogicznie do CAN sniffera.

b) Konfiguracja CAN1

Analogicznie do CAN sniffera.
Należy dodatkowo aktywować przerwania od odbioru danych.



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
CAN1 TX interrupts	☐	0	0
CAN1 RX0 interrupts	☑	0	0
CAN1 RX1 interrupt	☑	0	0
CAN1 SCE interrupt	☐	0	0

c) Konfiguracja USART3

Analogicznie do CAN sniffera.

d) Generowanie kodu:

Analogicznie do CAN sniffera.

e) Dodanie biblioteki

Analogicznie do CAN sniffera.
Potrzebny folder do umieszczenia w projekcie znajduje się w folderze:
Libs/CAN_Sniffer

f) Modyfikacja kodu

Poniżej przedstawione są fragmenty kodu, które należy umieścić w pliku “main.c”.

Importowanie biblioteki:

```
/* USER CODE BEGIN Includes */
#include "sniffer.h"
/* USER CODE END Includes */
```

Inicjalizacja modułu Sniffer:

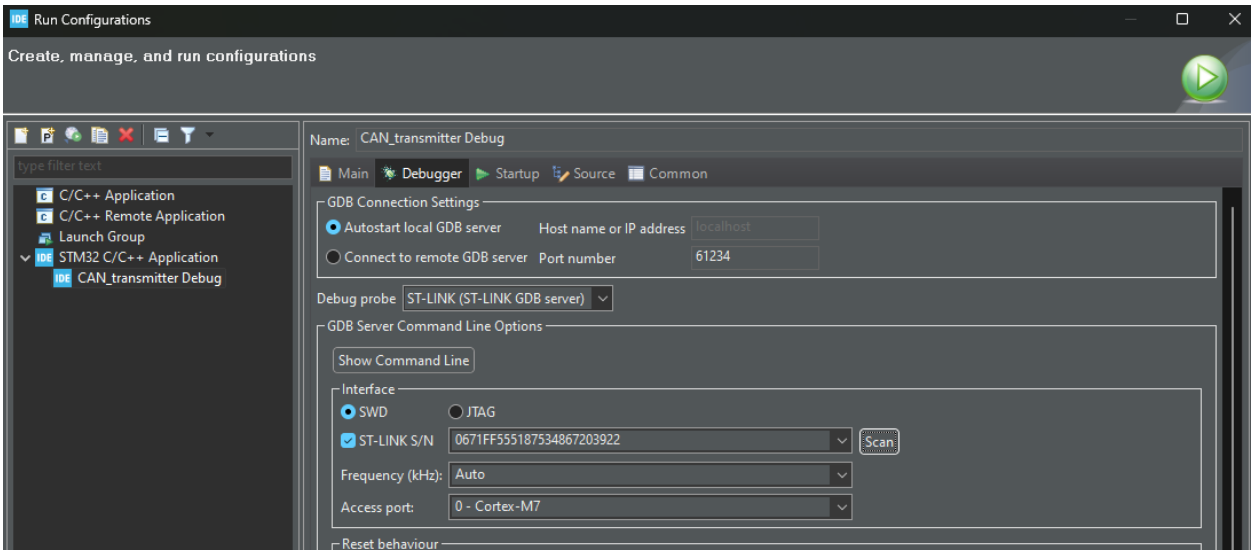
```
  /* USER CODE BEGIN 2 */
  Sniffer_Init();
  /* USER CODE END 2 */
```

Implementacja callbacków od przerwań (odbiór danych z magistrali CAN, a następnie ich transmisja poprzez port szeregowy):

```
/* USER CODE BEGIN 4 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
  CAN_RxHeaderTypeDef rx_header;
  uint8_t rx_msg[8];

  HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &rx_header, rx_msg);

  Sniffer_Transmit(rx_msg, rx_header.DLC, rx_header.StdId);
}

void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
  CAN_RxHeaderTypeDef rx_header;
  uint8_t rx_msg[8];

  HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO1, &rx_header, rx_msg);

  Sniffer_Transmit(rx_msg, rx_header.DLC, rx_header.StdId);
}
/* USER CODE END 4 */
```

5. Flashowanie

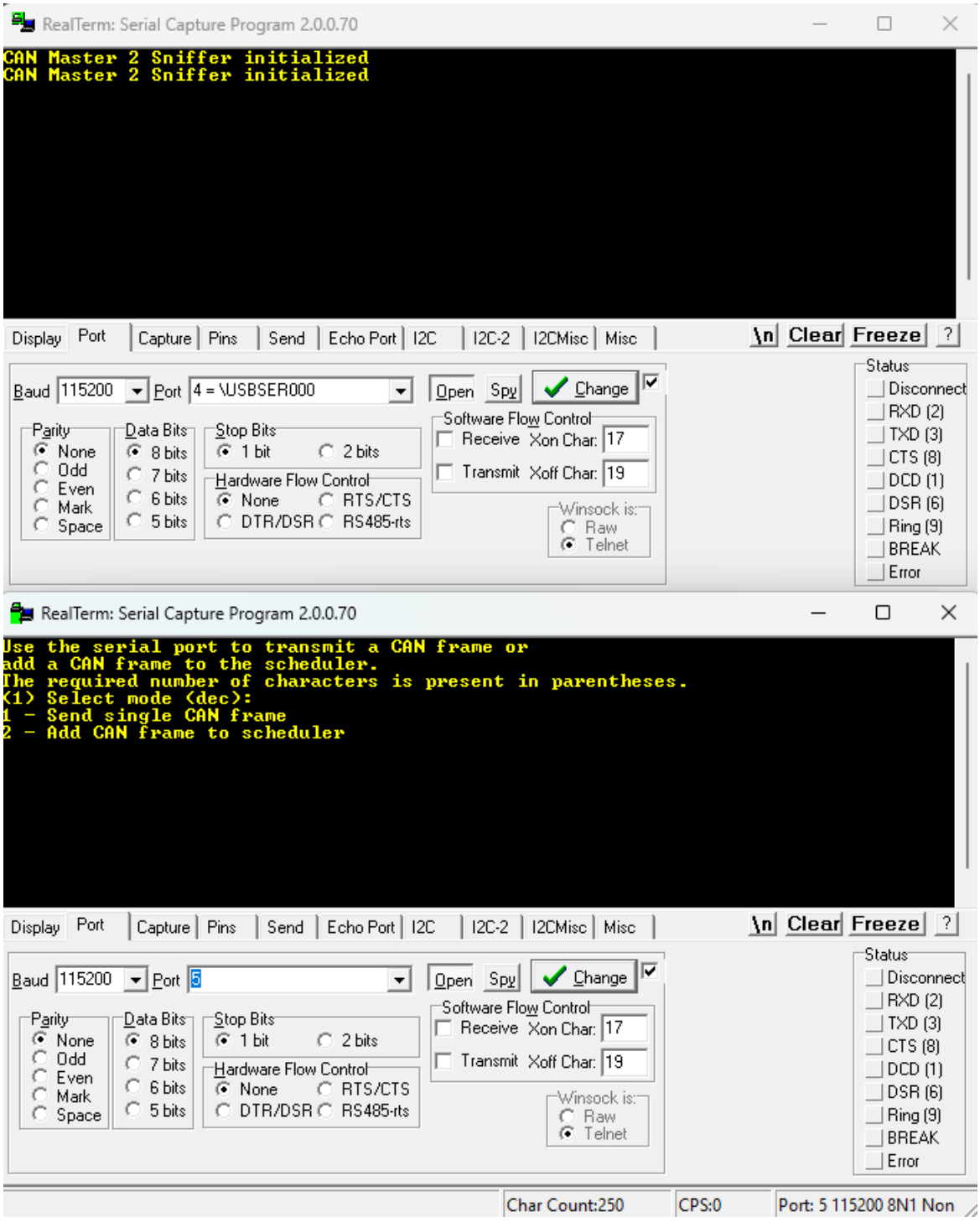
Budujemy projekt. Następnie klikamy przycisk **Run**. W konfiguracji w zakładce “Debugger” zaznaczamy “ST-LINK S/N” i klikamy “Scan”. Następnie wybieramy ID jednej z płytek i klikamy “Run”



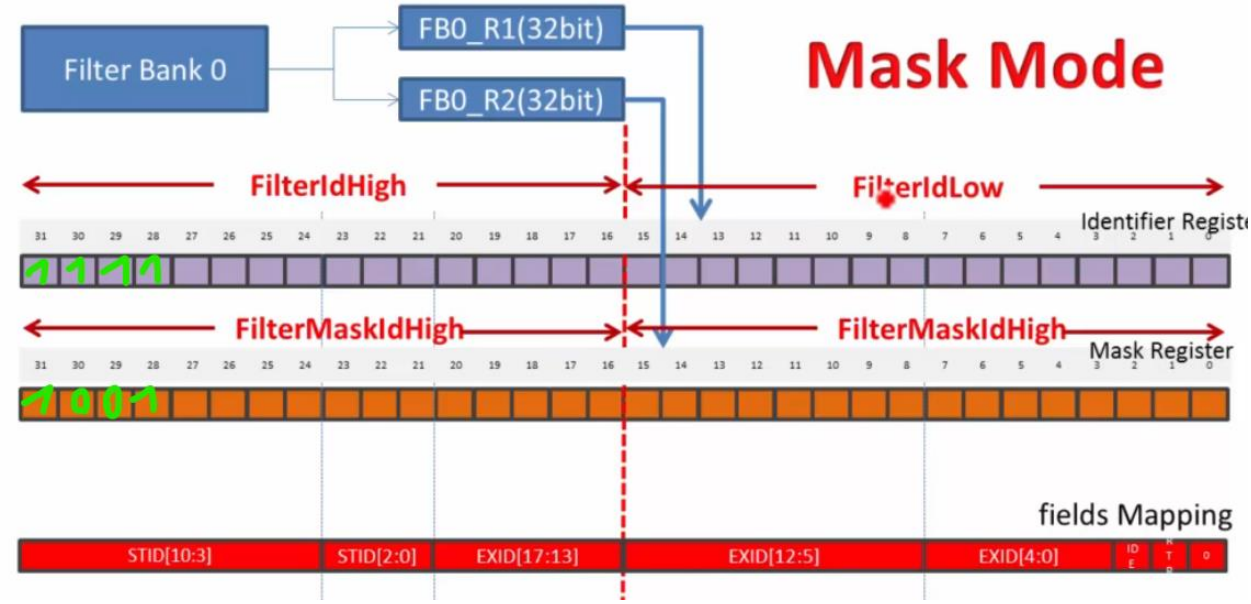
Proces ten powtarzamy dla CAN transmittera oraz CAN sniffera.

6. Komunikacja poprzez port szeregowy

Przy pomocy odpowiedniego oprogramowania np. RealTerm należy otworzyć port szeregowy dla obu płytek z baudrate 115200.
Można kliknąć przycisk reset na płytkach, aby sprawdzić poprawność działania. Powinny zostać wyświetlone komunikaty jak poniżej. Następnie należy skomunikować się z CAN transmitterem w celu wysłania żądania transmisji danych na magistrali CAN. Ilość bajtów do transmisji w każdym cyklu znajduje się w nawiasach. Np. “**<1>** *Select node (dec)*:”


RealTerm: Serial Capture Program 2.0.0.70
CAN Master 2 Sniffer initialized
CAN Master 2 Sniffer initialized
Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc
\n Clear Freeze ?
Baud 115200 Port 4 = \USBSER000
Open Spy Change
Parity None Odd Even Mark Space
Data Bits 8 bits 7 bits 6 bits 5 bits
Stop Bits 1 bit 2 bits
Hardware Flow Control None RTS/CTS DTR/DSR RS485-rts
Software Flow Control Receive Xon Char: 17 Transmit Xoff Char: 19
Winsock is: Raw Telnet
Status Disconnect RXD (2) TXD (3) CTS (8) DCD (1) DSR (6) Ring (9) BREAK Error
RealTerm: Serial Capture Program 2.0.0.70
Use the serial port to transmit a CAN frame or
add a CAN frame to the scheduler.
The required number of characters is present in parentheses.
(1) Select mode (dec):
1 - Send single CAN frame
2 - Add CAN frame to scheduler
Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc
\n Clear Freeze ?
Baud 115200 Port 5
Open Spy Change
Parity None Odd Even Mark Space
Data Bits 8 bits 7 bits 6 bits 5 bits
Stop Bits 1 bit 2 bits
Hardware Flow Control None RTS/CTS DTR/DSR RS485-rts
Software Flow Control Receive Xon Char: 17 Transmit Xoff Char: 19
Winsock is: Raw Telnet
Status Disconnect RXD (2) TXD (3) CTS (8) DCD (1) DSR (6) Ring (9) BREAK Error
Char Count:250
CPS:0
Port: 5 115200 8N1 Non

7. Filtracja ramek



Do włączenia funkcji filtrowania odbieranych ramek należy dokonać modyfikacji w bibliotece sniffera.

sniffer.c : l.40

```
    CAN_FilterTypeDef canfil;
    canfil.FilterBank = 0;
    canfil.FilterMode = CAN_FILTERMODE_IDMASK;
    canfil.FilterFIFOAssignment = CAN_RX_FIFO0;
    canfil.FilterIdHigh = 0xF000;
    canfil.FilterIdLow = 0x0000;
    canfil.FilterMaskIdHigh = 0x9000;
    canfil.FilterMaskIdLow = 0x0000;
    canfil.FilterScale = CAN_FILTERSCALE_32BIT;
    canfil.FilterActivation = ENABLE;
```

Ustawienie filtrowania pozwala modyfikować, które ID Sniffer będzie czytał. W naszym przypadku wymagamy by pierwsze 4 bity w ID wynosiły 4'b1001. Dla powyższego przypadku ramka z ID 0x480 zostanie odebrana przez CAN sniffera, a 0x580 nie.
Domyślnie filtracja jest włączona, ale bez przepuszczane są wszystkie ramki.