Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Informatyki Elektroniki i Telekomunikacji

Instrukcja do ćwiczeń laboratoryjnych

Magistrala CAN – zastosowanie i implementacja

z przedmiotu Standardy i Systemy Komunikacyjne

Autorzy: Dominik Tendera, Jakub Stelmach, Jakub Kwiecień

Opiekun: dr Inż. Jacek Stępień

Grudzień 2025, Kraków

Spis treści

Wprowadzenie i cel ćwiczenia

Magistrala CAN (Controller Area Network) jest powszechnie stosowanym, niezawodnym protokołem komunikacyjnym wykorzystywanym w motoryzacji i systemach wbudowanych. Zapewnia odporność na zakłócenia, wysoką integralność danych oraz możliwość komunikacji wielu urządzeń na jednej linii transmisyjnej. Celem ćwiczenia jest praktyczne zapoznanie się z działaniem magistrali CAN poprzez konfigurację interfejsu w mikrokontrolerze STM32, dobór parametrów czasowych transmisji oraz realizację podstawowej komunikacji w ramach sieci CAN. Podczas laboratorium student nauczy się konfigurować interfejs CAN, wysyłać i odbierać ramki, a także analizować sygnały przy użyciu analizatora cyfrowego, co pozwoli zrozumieć strukturę ramek, filtrowanie, arbitraż oraz mechanizmy wykrywania błędów. Dzięki temu zdobywa umiejętności niezbędne do implementacji komunikacji CAN w systemach embedded.

1. Teoria

CAN - szeregowa magistrala komunikacyjna i protokół warstwy fizycznej oraz łącza danych modelu ISO/OSI opracowany przez firmę Bosch w latach 80. XX wieku. Znalazł zastosowanie głównie w:

- Motoryzacji,
- Automatyce przemysłowej,
- Awionice,
- Kolejnictwie,
- Innych branżach, gdzie konieczna jest bardzo wysoka niezawodność.

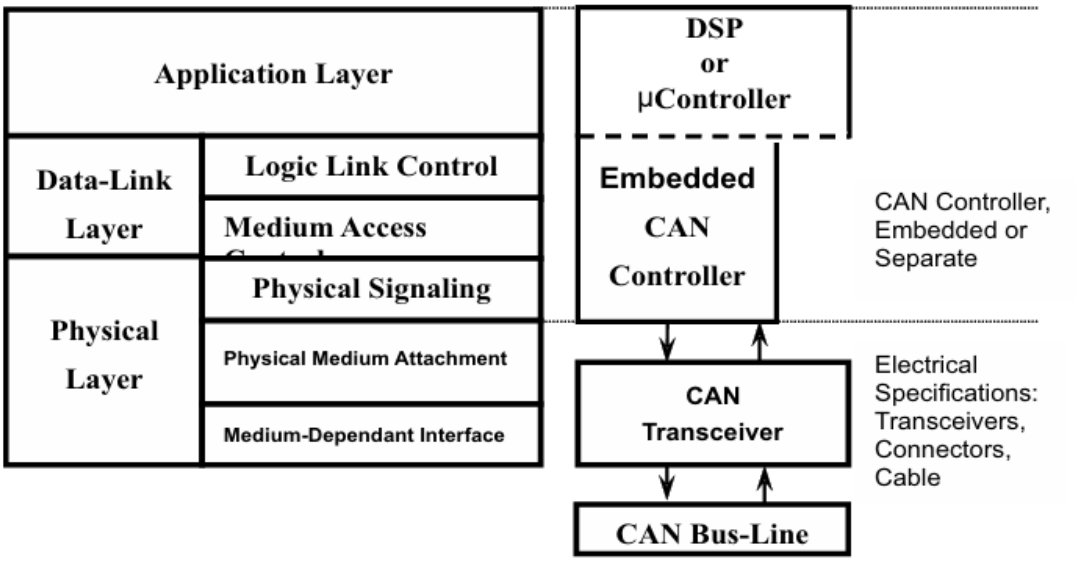


Figure 1. The Layered ISO 11898 Standard Architecture

Sieć CAN zazwyczaj składa się z:

- Warstwy fizycznej: przewodów magistrali, transceiverów CAN, które zapewniają transmisję różnicową; specyfikacja transceivera (np. Waveshare SN65HVD230) przedstawiona została w standardzie ISO11898,
- Warstwy łącza danych: kontrolera magistrali CAN, osobnego modułu (np. TCANXXXX od Texas Instruments) lub wbudowanego (mikrokontrolery STM32 z wbudowanym kontrolerem CAN).

Protokół CAN charakteryzuje się szeregiem zalet, które zadecydowały, że stał się tak popularny m.in. w motoryzacji i przemyśle. Do jego najważniejszych cech należą:

- Tylko 2 przewody jako medium transmisyjne, zazwyczaj skrętka miedziana,
- Transmisja różnicowa,
- Logika ze stanem dominującym (stan ‘0’) i stanem recesywnym (‘1’),

- System typu multi-master, transmisja rozgłoszeniowa,
- Arbitraż bezkolizyjny (CSMA/CR),
- Bardzo wysoka niezawodność sytemu, wiele mechanizmów wykrywania błędów,
- Bazowa szybkość transmisji do 1 Mb/s.

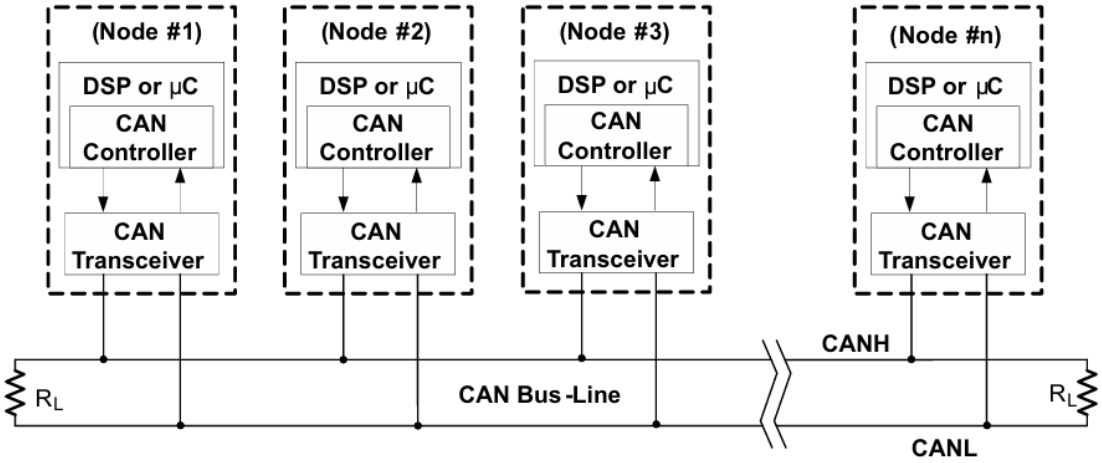
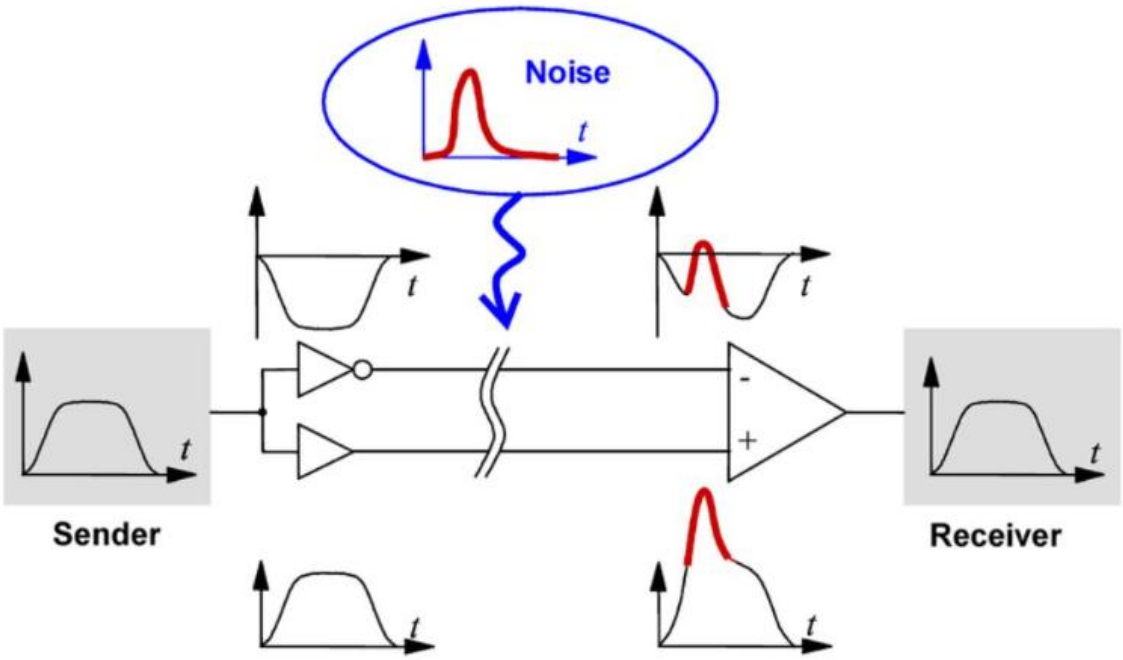


Figure 6. Details of a CAN Bus

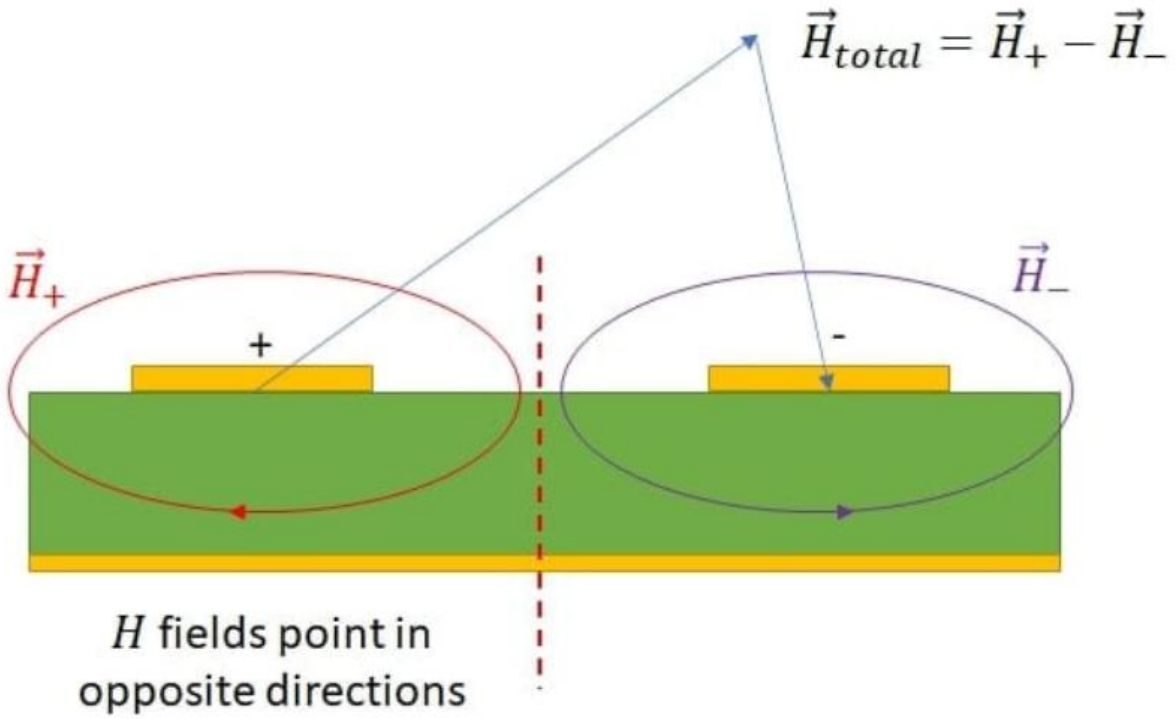
1.1. 1.1. Transmisja różnicowa

Warstwa fizyczna magistrali CAN opiera się na transmisji różnicowej. Transmisja różnicowa ma wiele zalet w stosunku do single-ended:

- Oporność na zaburzenia wspólne. Linie sygnałowe są wystawione na zaburzenia. W skrętce przewody sygnałowe umieszczone są w bliskiej odległości, więc zaburzenia sprzęgają się do obu linii w przybliżeniu tak samo, co jest ignorowane przez odbiornik różnicowy.



- Znacznie niższa emisja zakłóceń. Sygnał transmitowany w sposób różnicowy w parze skręconych przewodów emituje zaburzenia głównie w momencie przełączania. Ponieważ sygnały na obu liniach są w przeciwfazi, pola magnetyczne indukowane przez prądy znoszą się, więc emitowane zaburzenia są znacznie niższe niż w transmisji single-ended. Jest to szczególnie istotne m.in. w motoryzacji, co było kolejnym powodem takiej popularności CAN.



1.2. 1.2. Ramka CAN

Początkowo stancard ISO11898 specyfikował stadardową ramkę CAN, później ramka CAN został rozszerzona. Główną różnicą jest identyfikator ramki, który w standardowej ramce ma 11 bitów, co daje 2048 możliwych identyfikatorów, natomiast w ramce rozszerzonej identyfikator ma 29 bitów, co daje 537 milionów możliwości. W protokole CAN występuje stan dominujący (‘0’) i stan recesywny (‘1’), który można nadpisać stanem dominującym. Dzięki temu możliwe jest wykorzystanie wiele sprytnych mechanizmów kontrali błędów.

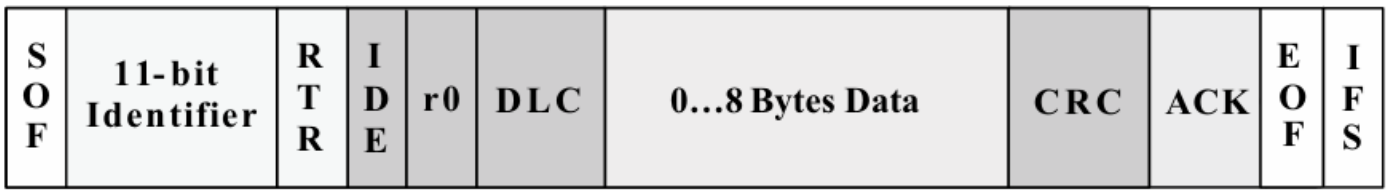


Figure 2. Standard CAN: 11-Bit Identifier

Standardowa ramka CAN ma następującą budowę:

- Podczas “ciszy” w kanale transmisyjnym nadawany jest bit recesywny (‘1’),
- SOF (*Start Of Frame*) - bit dominujący (‘0’) sygnalizujący początek ramki,
- 11-bitowy identyfikator – identyfikator węzła magistrali, opisane w punkcie 1.3.,
- RTR (*Remote Transmission Request*) - bit recesywny, jeśli węzeł nadaje dane; bit dominujący, jeśli węzeł transmitujący żąda transmisji danych od innego węzła, wtedy identyfikator jest identyfikatorem węzła, od którego następuje żądanie,
- IDE (*Identifier Extension*) - bit dominujący oznacza, że ramka jest standardową ramką CAN; bit recesywny oznacza ramkę rozszerzoną,
- r0 (*Reserved*) - bit zarezerwowany dla przyszłych zastosowań CAN,
- DLC (*Data Length Code*) - 4-bitowa długość pola danych w bajtach,
- Data – od 0 do 8 bajtów danych,
- CRC (*Cyclic Redundancy Check*) - 15-bitowa suma kontrolna (oraz 1-bitowy delimiter), służy sprawdzeniu poprawności przesłanych danych, zwiększa niezawodność protokołu,
- ACK (*Acknowledge*) - 1 bit potwierdzenia (oraz 1-bitowy delimiter); w przypadku poprawnie wysłanej ramki danych, inne węzły nadpisują bitem dominującym domyślny bit recesywny wysłany przez nadający węzeł,
- EOF (*End Of Frame*) - 7 bitów recesywnych, wyłącza nadziewanie bitami,
- IF (*Interframe Space*) - 7-bitowa przerwa między ramkami na potencjalne obliczenia węzłów.

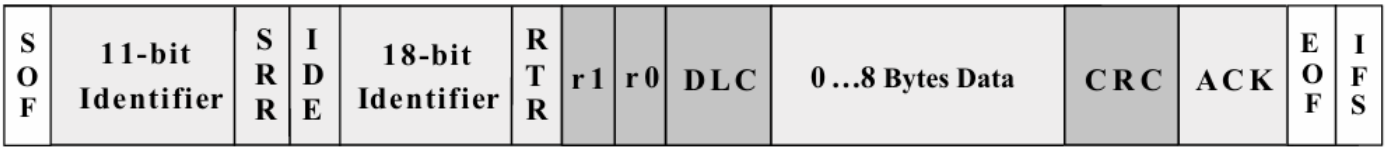


Figure 3. Extended CAN: 29-Bit Identifier

Ramka rozszerzona zawiera dodatkowo następujące pola:

- SRR (*Substitue Remote Request*) - zmiana pozycji RTR na standardową, za identyfikatorem,
- IDE (*Identifier Extension*) – bit recesywny oznacza ramkę rozszerzoną,
- R1 (*Reserved*) - bit zarezerwowany dla przyszłych zastosowań CAN.

1.3. 1.3. Kolizje (CSMA/CR)

W transmisji CAN wszystkie węzły mogą nadawać jednocześnie. Ponieważ CAN jest protokołem rozgłoszeniowym typu multi-master, na magistrali mogą występować kolizje. Problem ten rozwiązano stosując schemat nadawania, którym są stany recesywny i dominujący. Jeśli wiele węzłów zaczyna nadawać jednocześnie, w polu identyfikatora bity recesywne (‘1’) są nadpisywane dominującymi (‘0’), dzięki czemu urządzenia o niższym identyfikatorze otrzymują dostęp do medium. Ponieważ transceiver CAN cały czas bada stan magistrali, widzi, że wysłany identyfikator nie zgadza się z identyfikatorem na magistrali – przestaje nadawać (CSMA/CR). Jest to kolejna zaleta CAN – priorytetyzacja identyfikatorów, dzięki czemu węzły krytycze mają zapewniony dostęp do medium. Jest to szczególnie istotne w motoryzacji, gdzie np. systemy związane z bezpieczeństwem muszą zawsze zadziałać, kiedy to konieczne.

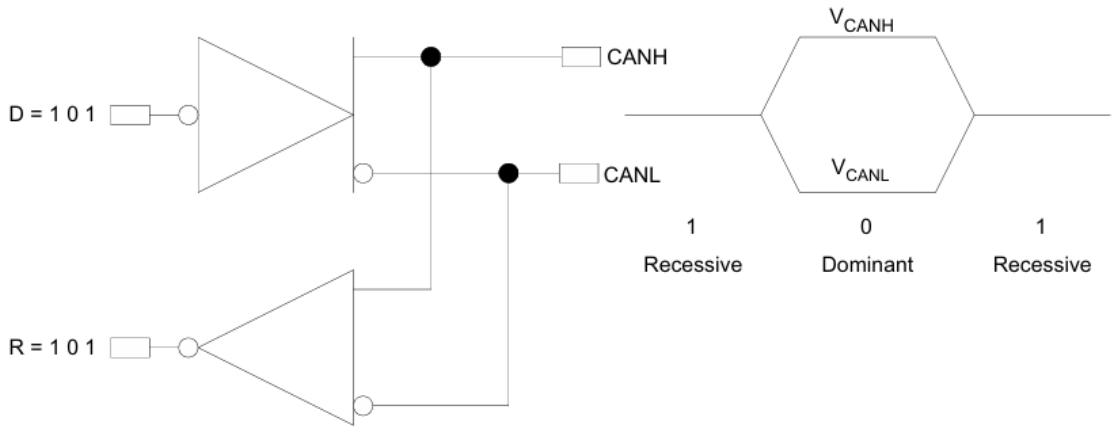


Figure 4. The Inverted Logic of a CAN Bus

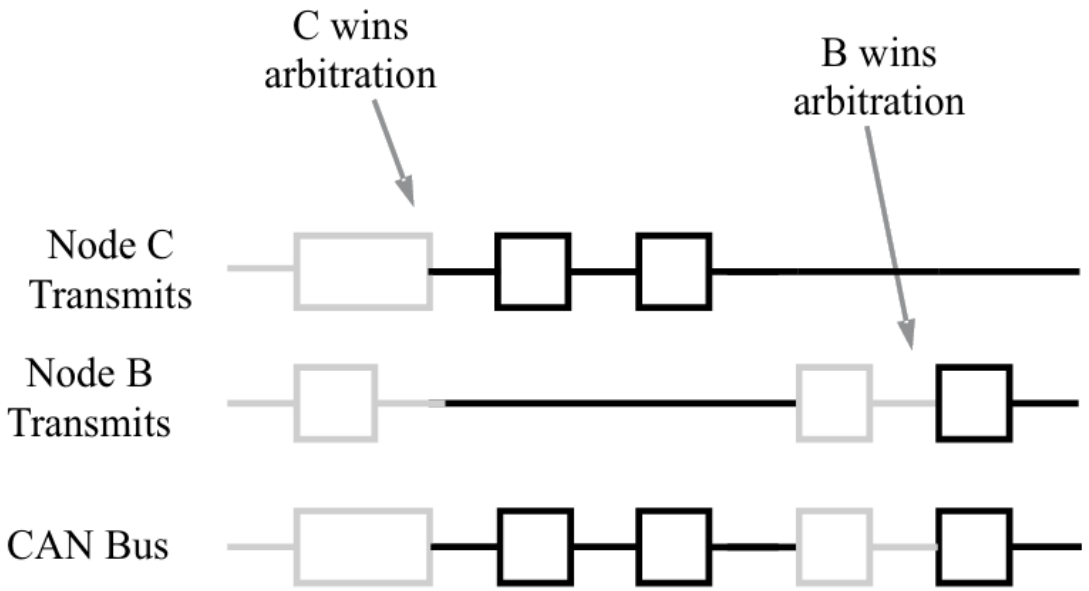


Figure 5. Arbitration on a CAN Bus

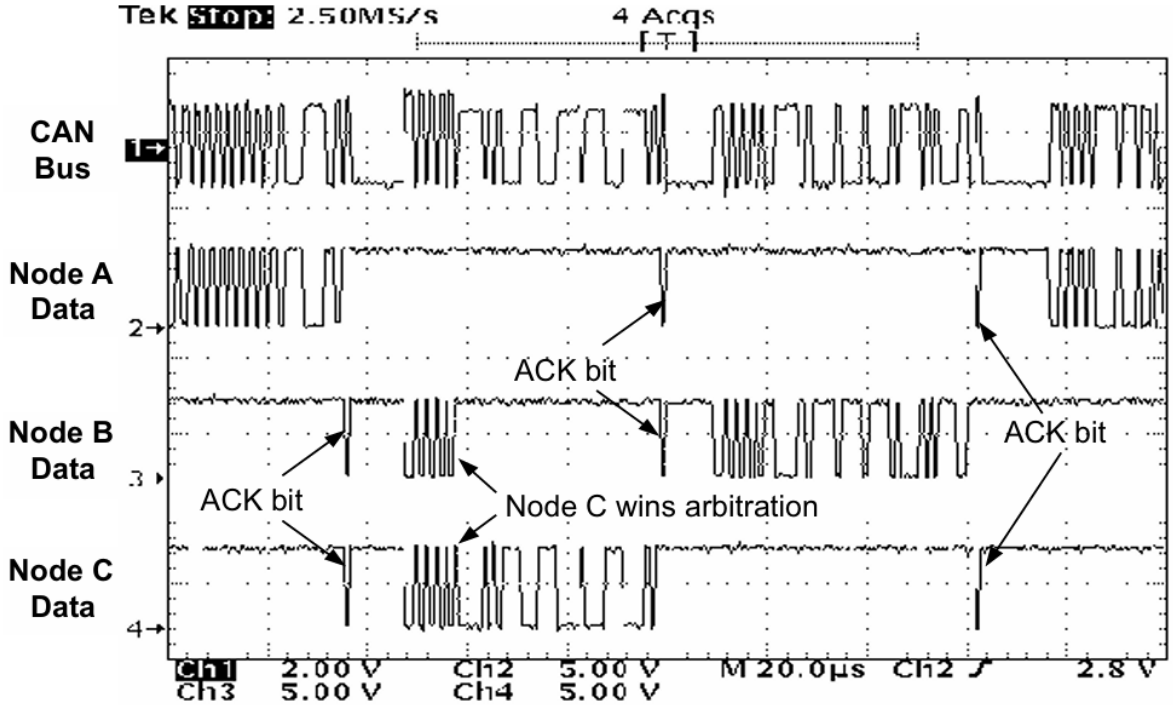


Figure 8. CAN Bus Traffic

1.4. 1.4. Mechanizmy wykrywania błędów

Protokół CAN jest niezwykle niezawodny, ponieważ zawiera wiele sprytnych mechanizmów wykrywania błędów transmisji:

- ACK – nie nadpisanie bitu ACK oznacza błędy trasnmisji,
- CRC – niepoprawna suma kontrolna oznacza błędy transmisji,

- SOF, EOF, ACK delimiter, CRC delimiter - domyślnie bity recesywne, nadpisanie ich oznacza błędy transmisji,
- Autoweryfikacja przez transceiwer - różnica wartości nadanej i realnie występującej na magistrali oznacza błędy transmisji; wyjątkiem są pola identyfikatora, gdzie następuje priorytetyzacja i arbitraż oraz pole ACK,
- Nadziewanie bitami – po kolejnych 5 bitach o tej samej wartości musi nastąpić bit o wartości przeciwnej; pomaga to w synchronizacji zegarów i pomaga to wyróżnić ramkę błędu oraz przerwę między ramkami (więcej niż 5 bitów o tej samej wartości).

1.5. 1.5. Ramka błędu

Ramka błędu składa się z 6 bitów dominujących. Jest wyraźnie rozróżnialna od innych ramek, ponieważ inne ramki są nadziewane bitami i nigdy nie występuje sytuacja, w której jest więcej niż 5 btów o tej samej polaryzacji. Wyjątkiem jest przerwa w nadawaniu, gdzie na magistrali są bity recesywne. Schemat wysłania ramki błędu jest następujący:

- Na magistrali wykryty zostaje błąd transmisji,
- Węzeł, który wykrył błąd zaczyna nadawać ramkę błędu - 6 bitów dominujących,
- Kolejne węzły wykrywają błąd - nadają ramkę błędu i same zaczynają nadać ramkę błędu,
- Sumarycznie zostaje wysłanych 12 bitów dominujących: ramka łędu oraz odpowiedź,
- Następuje delimiter (8 bitów recesywnych) oraz IFS (*Interframe Space*),
- Nadajnik retransmituje ramkę , ale nie jest pewne, że inny węzeł o wyższym priorytecie nie zacznie nadawać w tym samym momencie,
- Nadajnik zawiera licznik błędów, po przekroczeniu pewnej liczny błędów odłącza się od magistrali.

1.6. 1.6. Filtracja wiadomości

CAN jest protokołem rozgłoszeniowym, więc na każdym węźle ciąży obowiązek filtrowania nadchodzących ramek i decydowania, czy otrzymane dane są przeznaczone danego węzła. Rozwiązując ten problem sprzętowo można znacznie odciążyć procesor każdego układu na magistrali. Wykorzystuje się do tego filtry akceptacyjne. Dzięki temu do bufora odbiorczego przypisywane są jedynie te ramki, które dotyczą danego węzła.

2. Ćwiczenie

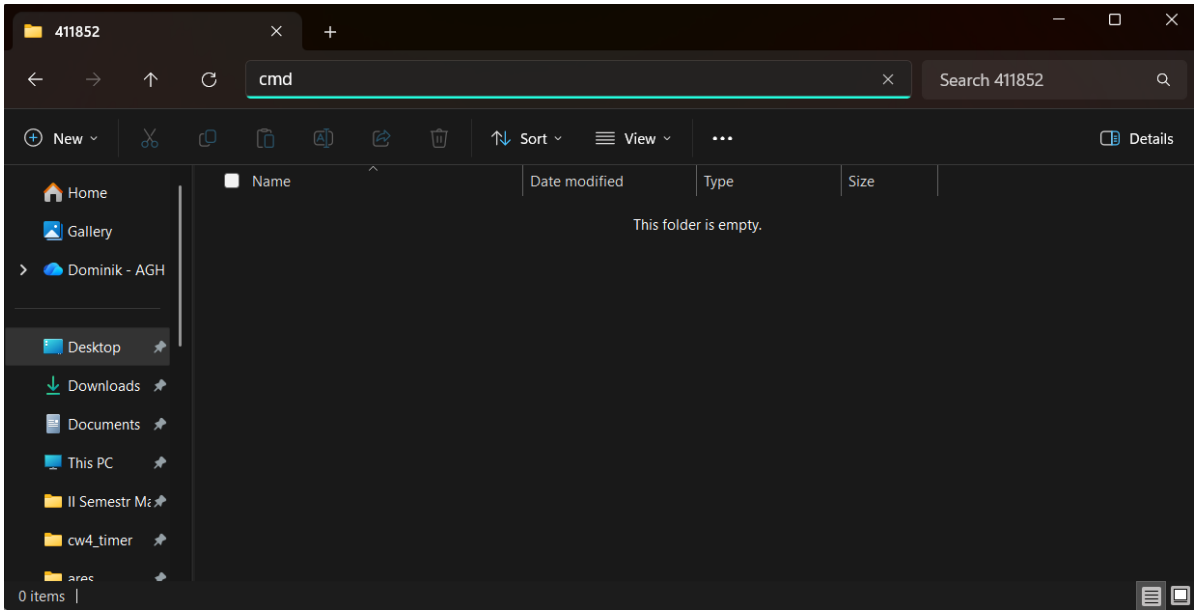
Zadanie należy wykonać na platformie NUCLEO-F446ZE z oprogramowaniu dostępnym w repozytorium https://github.com/Dominik-Tendera/CAN_laboratory.git

2.1. Pobranie projektu z repozytorium

Poniżej znajduje się podstawowy opis obsługi githuba.

Proszę utworzyć własny folder na pulpicie w nazwie wpisując swój numer indeksu

Będąc w folderze proszę wpisać **cmd** w pasku adresu w eksploratorze plików i kliknąć **Enter.**



Po uruchomieniu comand line wpisujemy kolejno :

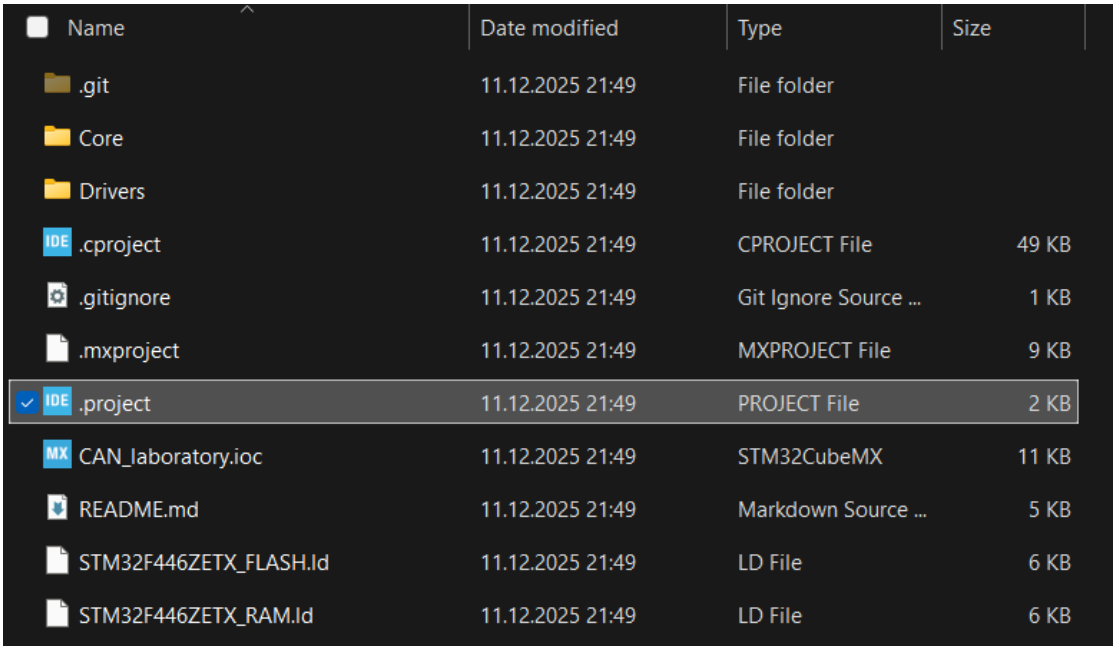
```
C:\Users\musie\Desktop\411852>git init
Initialized empty Git repository in C:/Users/musie/Desktop/411852/.git/

C:\Users\musie\Desktop\411852>git clone https://github.com/Dominik-Tendera/CAN_laboratory.git
Cloning into 'CAN_laboratory'...
remote: Enumerating objects: 129, done.
remote: Counting objects: 100% (129/129), done.
remote: Compressing objects: 100% (101/101), done.
remote: Total 129 (delta 27), reused 127 (delta 25), pack-reused 0 (from 0)
Receiving objects: 100% (129/129), 701.00 KiB | 4.58 MiB/s, done.
Resolving deltas: 100% (27/27), done.

C:\Users\musie\Desktop\411852>
```

Operacja powinna zakończyć się sukcesem

Następnie przechodzimy do folderu i otwieramy projekt w STM32CubeIDE, klikając dwukrotnie na **.project**

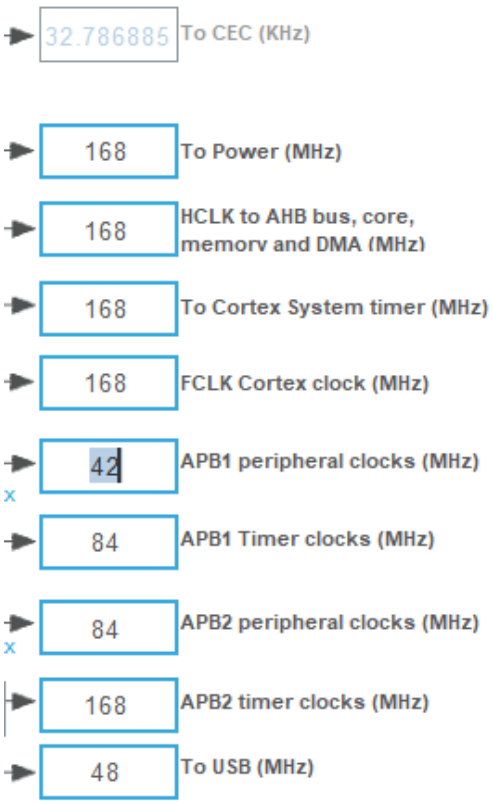


Name	Date modified	Type	Size
.git	11.12.2025 21:49	File folder	
Core	11.12.2025 21:49	File folder	
Drivers	11.12.2025 21:49	File folder	
.cproject	11.12.2025 21:49	CPROJECT File	49 KB
.gitignore	11.12.2025 21:49	Git Ignore Source ...	1 KB
.mxproject	11.12.2025 21:49	MXPROJECT File	9 KB
.project	11.12.2025 21:49	PROJECT File	2 KB
CAN_laboratory.ioc	11.12.2025 21:49	STM32CubeMX	11 KB
README.md	11.12.2025 21:49	Markdown Source ...	5 KB
STM32F446ZETX_FLASH.ld	11.12.2025 21:49	LD File	6 KB
STM32F446ZETX_RAM.ld	11.12.2025 21:49	LD File	6 KB

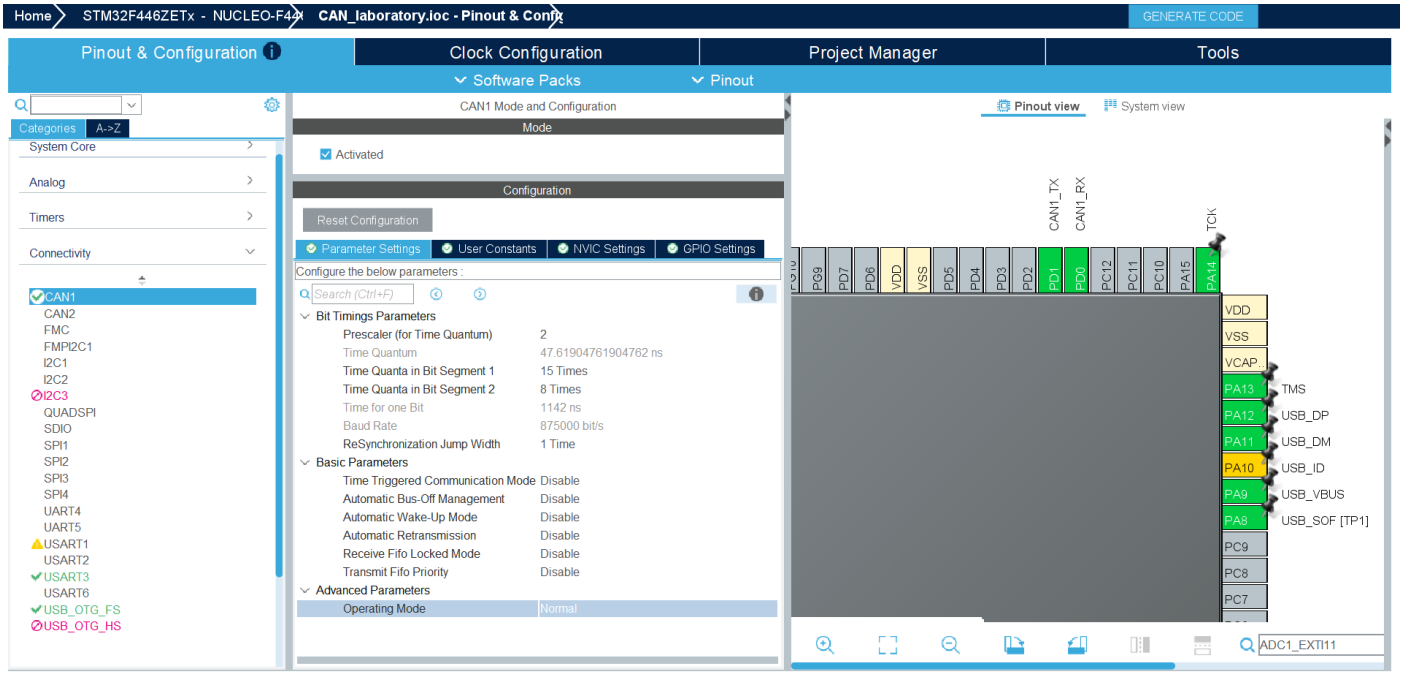
2.2. Ustawienie pliku konfiguracyjnego i magistrali CAN

Wchodzimy do pliku konfiguracyjnego **.ioc**, teraz należy ustawić

Na tej platformie CAN wykorzystuje zegar peryferyjny APB1, proszę ustawić wartość APB1 w zakładce clock configuration na **42MHz.**



Następnie należy aktywować CAN, poniżej znajduje się opis najważniejszych parametrów do ustawienia:



- **CAN Mode**: Normal (do pracy na magistrali), w razie problemów można wykorzystać tryb Loopback i spróbować odebrać własną wiadomość. Pinouty (PD0 = CAN1_RX, PD1 = CAN1_TX).
- **Prescaler**: dzieli zegar APB1 na takt CAN (większy prescaler → mniejsza prędkość).

Wizualne przedstawienie jak z czego składa się Bit CAN**: |SYNC|------BS1--------|BS2|**
Bit jest dzielony na pomniejsze TQ(time quants) dlatego zwiększając liczbę BS1 i BS2 zwiększamy liczbę kwantów które przypadają na jeden bit dlatego też prędkość magistrali się zmienia. Chcemy próbkować sygnał po tym gdy się ustabilizuje dlatego najlepiej jest to robić bliżej końca, ustawiając wyższe BS1.

- **SYNC_SEG** (Synchronization Segment): Stały segment o długości 1 TQ. Odpowiada za zsynchronizowanie kontrolera CAN ze zboczem sygnału na magistrali. Umożliwia wyrównanie początku bitu do rzeczywistego momentu przejścia sygnału.
- **BS1 (Time Segment 1):** Segment zawierający liczbę TQ od końca SYNC_SEG do punktu próbkowania. To BS1 określa, kiedy dokładnie nastąpi odczyt bitu. Im większa wartość BS1, tym później w czasie trwania bitu wykonywane jest próbkowanie, co zwiększa odporność na zakłócenia i opóźnienia zboczy. Punkt próbkowania zawsze znajduje się na końcu BS1.
- **BS2 (Time Segment 2):** Segment obejmujący liczbę TQ po punkcie próbkowania, aż do końca bitu. BS2 nie zmienia położenia punktu próbkowania — odpowiada za czas stabilizacji sygnału po jego odczycie. Właściwa długość BS2 zapewnia poprawne przetwarzanie kolejnego bitu i minimalizuje wrażliwość na jitter.
- **SJW (Synchronization Jump Width):** Maksymalna korekta, o którą kontroler CAN może zmodyfikować długość BS1, aby skompensować drobne różnice częstotliwości zegarów między

węzłami sieci. SJW umożliwia przesunięcie momentu próbkowania o maksymalnie SJW TQ. Typowe wartości to 1–2 TQ, co pozwala na stabilną synchronizację bez nadmiernych zmian w strukturze bitu.

- **DLC**: długość danych w ramce (ustawiana przy wysyłce; klasyczny CAN max 8).

$$f_{\text{CAN}} = \frac{f_{\text{APB}}}{\text{Prescaler} \cdot (1 + \text{BS1} + \text{BS2})}$$

gdzie:

- f_{CAN} – końcowa prędkość bitowa (bitrate),
- f_{APB} – częstotliwość zegara APB dla kontrolera CAN (np. APB1 = 42 MHz w STM32F4),
- **Prescaler** – dzielnik częstotliwości,
- **BS1** – Bit Segment 1 (Time Segment 1),
- **BS2** – Bit Segment 2 (Time Segment 2),
- **1** – Sync Segment (zawsze 1 time quantum).

Biorąc pod uwagę wszystkie powyższe wiadomości proszę ustawić szybkość transmisji magistrali na **494 117 bit/s** co odpowiada mniej więcej 500 kbps, czyli popularnej wartości używanej na rynku.

Następnie uruchomić odpowiednie przerwanie w konfiguracji CAN NVIC Settings, przy się do odbierania ramek w dalszej części:

Parameter Settings | User Constants | NVIC Settings | GPIO Settings

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

Przygotowane oprogramowanie ustawia filtry CAN tak aby przyjmować każde ramki pojawiające się na magistrali, natomiast warto odnotować, że można ustawić filtracje ID na poziomie hardwearowym.

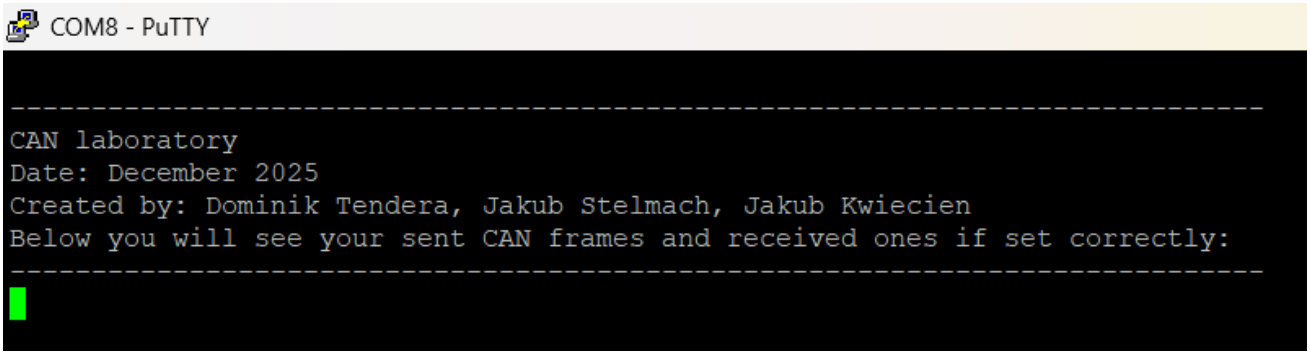
```
static void CAN_ConfigFilters_AcceptAll(void)
{
  CAN_FilterTypeDef filter = {0};
  filter.FilterBank = 0;                       // Use bank 0 for CAN1
  filter.FilterMode = CAN_FILTERMODE_IDMASK; // Mask mode
  filter.FilterScale = CAN_FILTERSCALE_32BIT;
  filter.FilterIdHigh = 0x0000;                // Accept all IDs
  filter.FilterIdLow = 0x0000;
  filter.FilterMaskIdHigh = 0x0000;
  filter.FilterMaskIdLow = 0x0000;
  filter.FilterFIFOAssignment = CAN_FILTER_FIFO0;
  filter.FilterActivation = ENABLE;
  filter.SlaveStartFilterBank = 14;            // Not used by CAN1 on F4

  if (HAL_CAN_ConfigFilter(&hcan1, &filter) != HAL_OK)
  {
    UART_Print("Filter config failed\r\n");
    Error_Handler();
  }
}
```

2.3. Schemat podłączenia

2.4. UART

Proszę znaleźć na jaką prędkość został ustawiony UART w projekcie oraz pod jakim portem szeregowym jest podpięta płytka, a następnie otworzyć połączenie szeregowe za pomocą **PuTTY,** po prawidłowym uruchomieniu w konsoli powinna być widoczny poniższa wiadomość:



2.5. Wysyłanie ramki CAN

Następnie za pomocą funkcji `CAN_SendText(const char *text)` wysyłać cyklicznie swoją własną wymyśloną wiadomość zawierającą maksymalnie 8 znaków.
Proszę znaleźć miejsce w kodzie gdzie jest ustawiony ID wysyłanej ramki i wpisać tam numer swojej grupy
Jeżeli uda się wysłać odpowiednią ramkę, zostanie ona wyświetlona w konsoli.

2.6. Odbiór ramki CAN

Proszę odebrać ramkę CAN zespołu obok ustawiając do tego filtrację i odczyt tylko i wyłącznie ich numer, do tego celu przydatnym będzie przeanalizowanie funkcji z przerwania Rx:
`HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)`
x: Po odpowiednim odebraniu ramki jej zawartość powinna być widoczna w konsoli, można skonsultować z grupą obok czy wiadomość się zgadza.

2.7. Obserwacja magistrali

Należy zacząć odczytywać cały ruch na magistrali, w konsoli powinny się pojawić wszystkie ramki. Proszę odczytać i zapisać ID oraz wiadomość ramki z nieznanego źródła

2.8. Wysyłka większej wiadomości

Proszę powrócić do odbierania tylko wiadomości grupy sąsiedniej, oraz wysłać własną wiadomość która ma więcej niż 8 znaków. Następnie zaobserwować zachowanie na magistrali i wyjaśnić zachowanie.

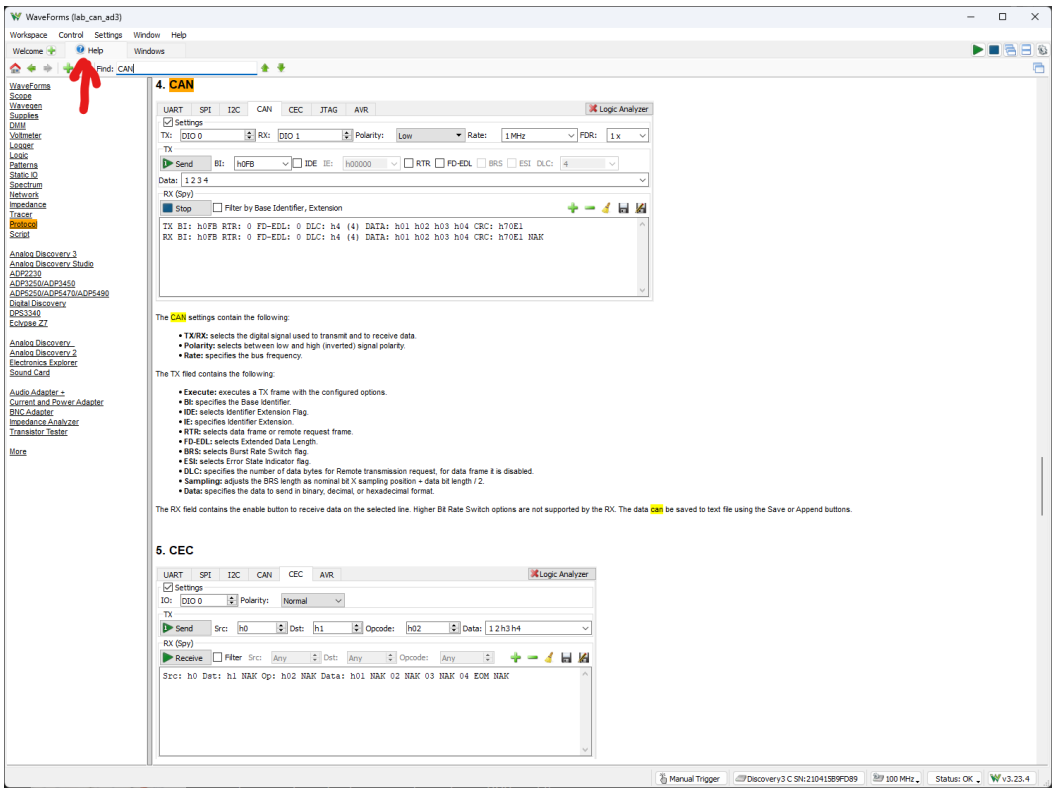
3. Wizualizacja magistrali za pomocą WaveForms

3.1. 3.1. Kofiguracja oprogramowania DIGILENT WaveForms

Analog Discovery to przenośne, wielofunkcyjne narzędzie pomiarowe firmy Digilent, łączące w sobie oscyloskop, generator funkcyjny, analizator logiczny i wiele innych funkcji. Umożliwia wykonywanie zaawansowanych pomiarów i testów elektronicznych przy użyciu komputera, co czyni go świetnym narzędziem edukacyjnym i prototypowym.
Strona producenta (wersja 3):
https://digilent.com/shop/analog-discovery-3/

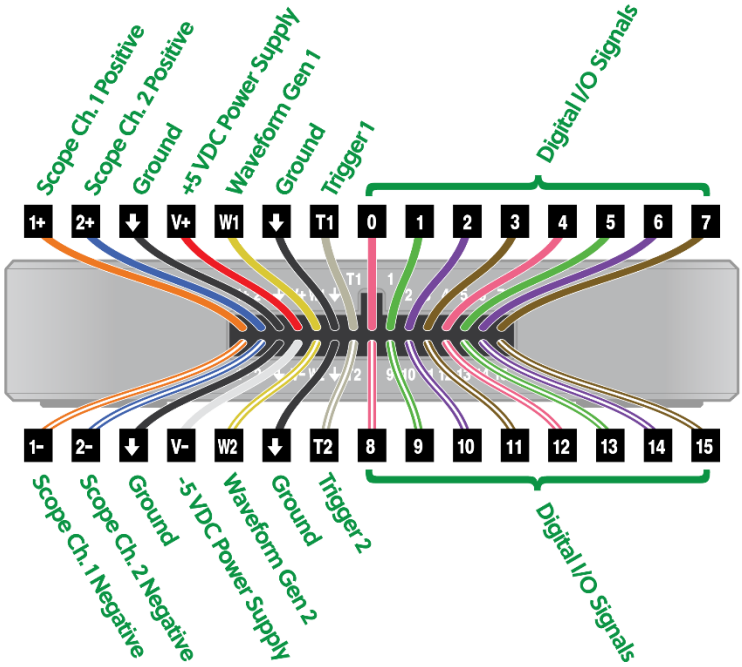
WaveForms to oprogramowanie firmy Digilent służące do obsługi urządzeń z serii Analog Discovery i Digital Discovery. Umożliwia korzystanie z wbudowanych narzędzi, takich jak oscyloskop, generator sygnałów, analizator logiczny czy zasilacz, zapewniając wygodny interfejs do pomiarów i analizy sygnałów.
https://digilent.com/reference/software/waveforms/waveforms-3/start

Producent udostępnia opis funkcji oraz pomoc w formie zakładki Help:



Schemat połączeń:

Opis wyprowadzeń Analog Discovery (uniwersalny dla wersji 2 oraz 3)



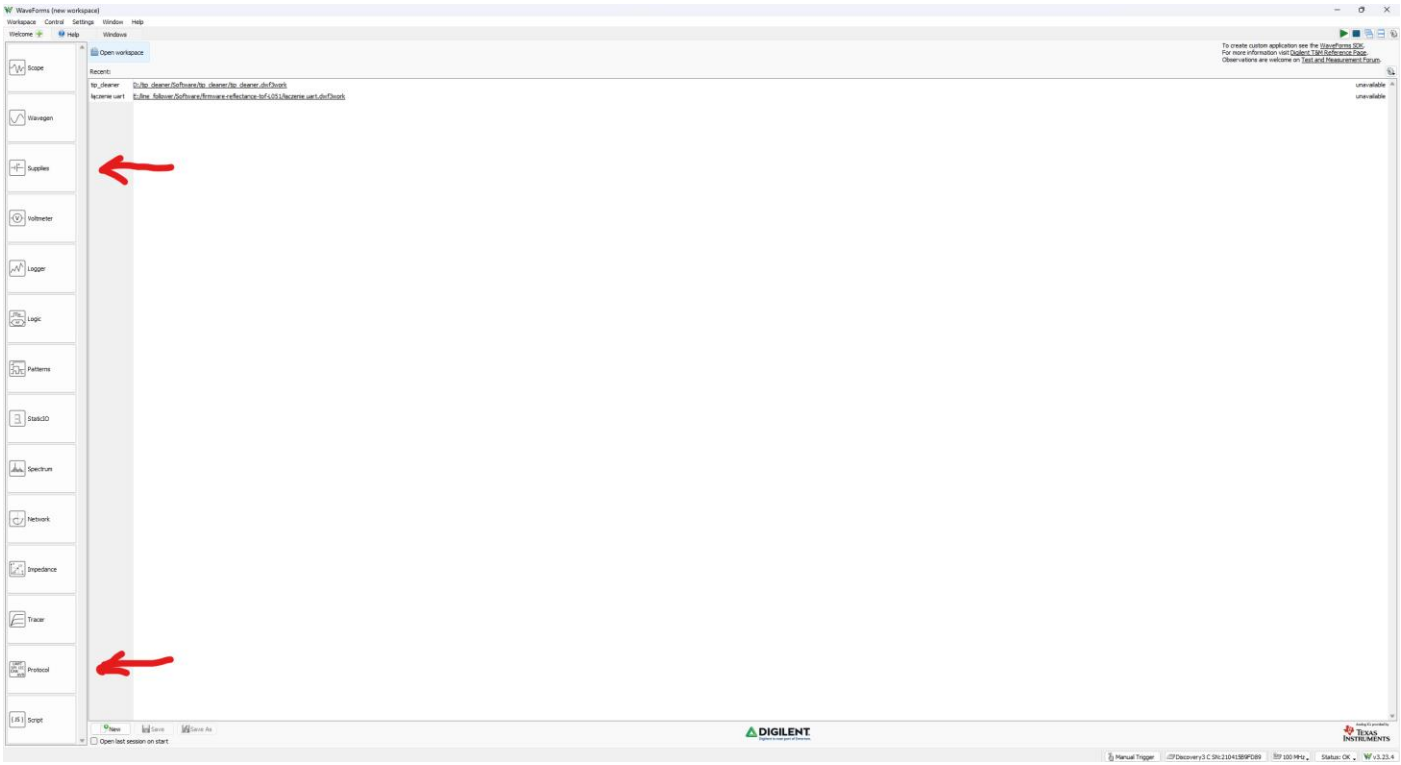
W zadaniu skorzystamy z pinów:

- Ground (górny lub dolny)
- 0 (DIO0)
- 1 (DIO1)8
- V+ (Wyjście zasilacza)

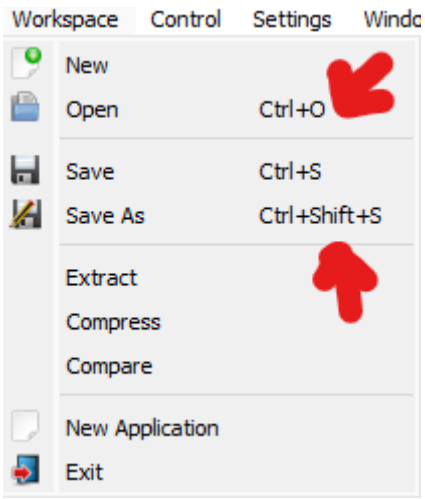
Proces konfiguracji:

1. Wstępna konfiguracja środowiska

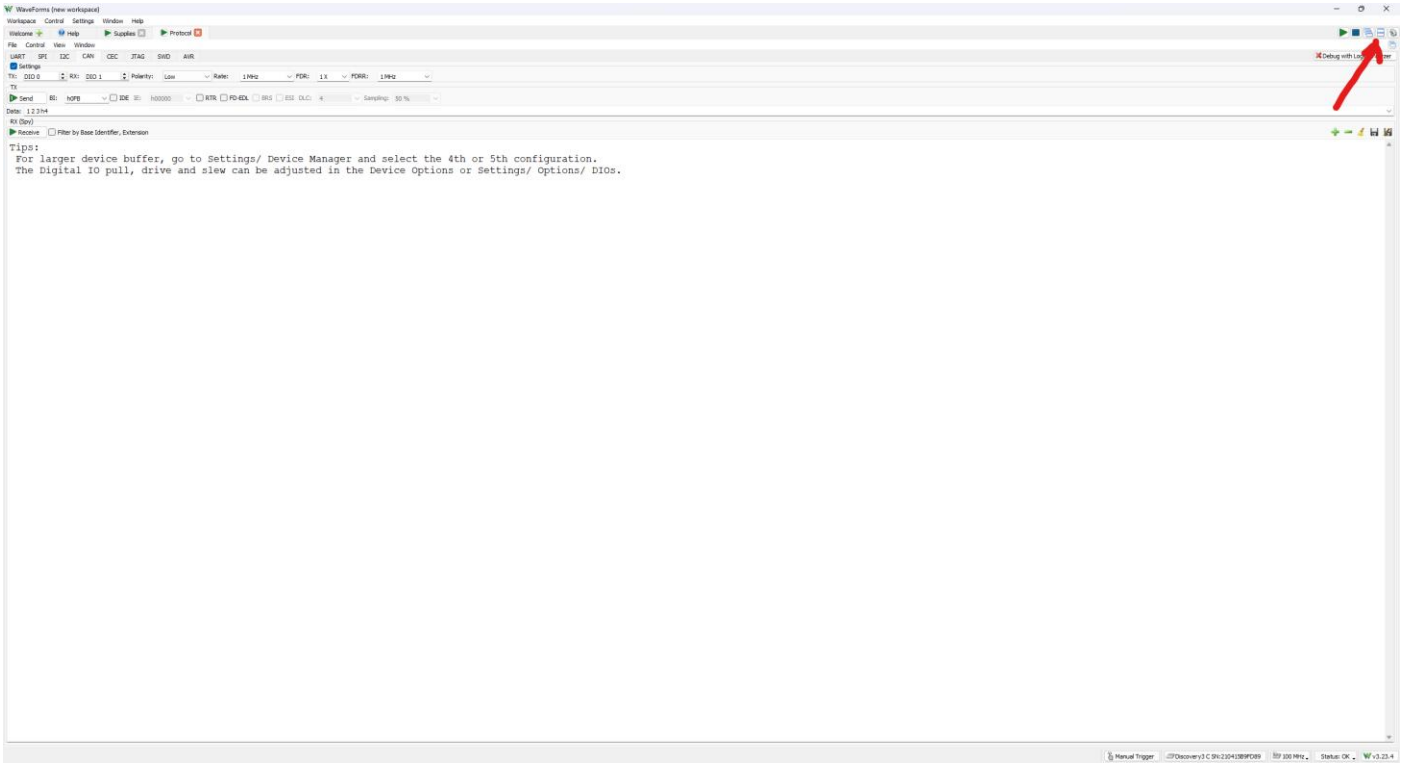
Po uruchomieniu programu wybieramy zakładki „Protocol” oraz „Supplies”. Potrzebujemy zasilić Transceiver za pomocą wbudowanego zasilacza (5V) oraz przechwycić komunikację CAN. Obie te funkcjonalności udostępnia nam urządzenie. (Kliknięcie „Welcome +” powoduje powrót do widoku głównego).

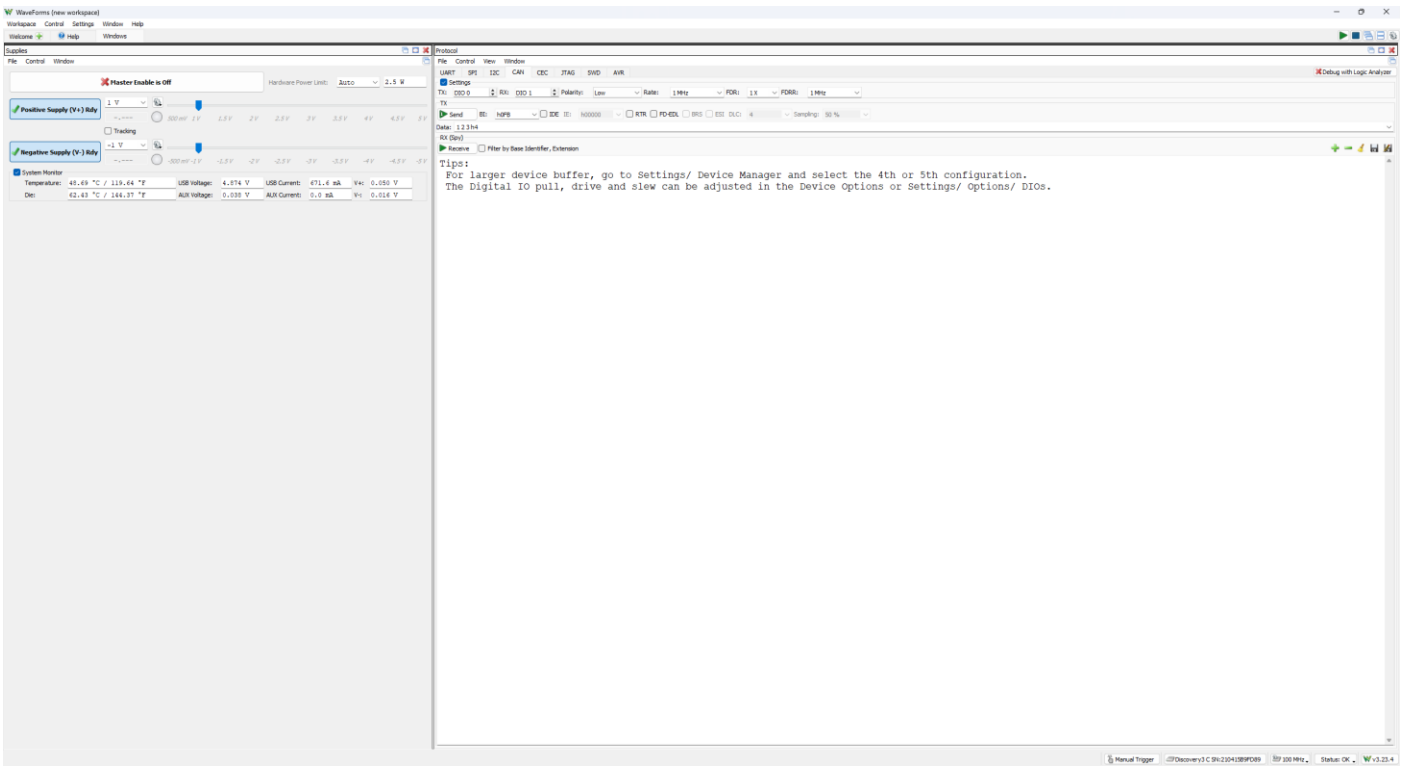


Jeżeli chcemy zapisać lub wczytać zapisany „workspace”, klikamy przycisk „workspace=>open”, a w celu zapisania – „workspace=> save as”. Workspace jest to szablon, przydatny w momencie kiedy chcemy bez ponownego konfigurowania środowiska powtórzyć pomiary/testy.



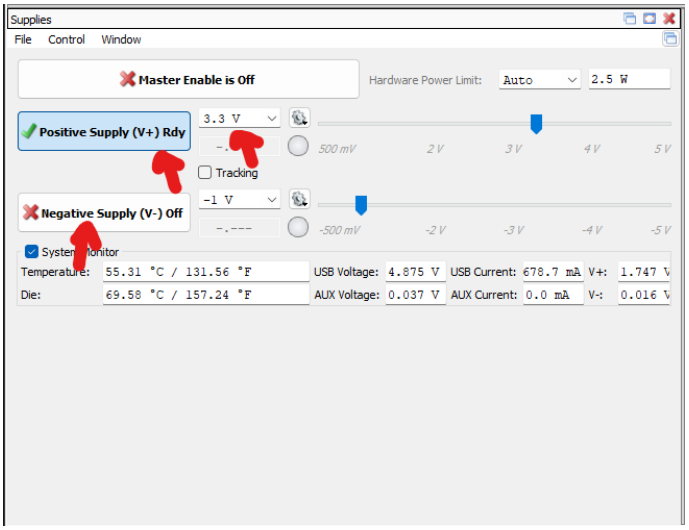
W celu ułatwienia obsługi możemy wyświetlić oba narzędzia w widoku podzielonego ekranu



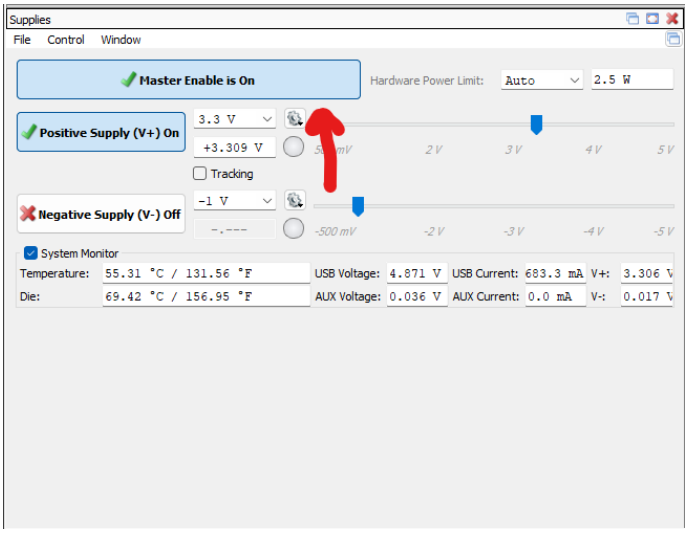


2. Konfiguracja Zasilacza

Na ekranie „Supplies” ustawiamy napięcie na kanale pierwszym na poziomie 3.3V (wpisując wartość w pole), ustawiamy „Negative Supply (-V) off” oraz „Negative Supply (+V) On”. W zakładce System Monitor możemy obserwować parametry zasilania. Analog Discovery pozwala na zasilanie go bezpośrednio ze złącza USB C lub z gniazda DC Jack (Aux). W naszym przypadku wystarczy zasilanie bezpośrednio ze złącza USB C.



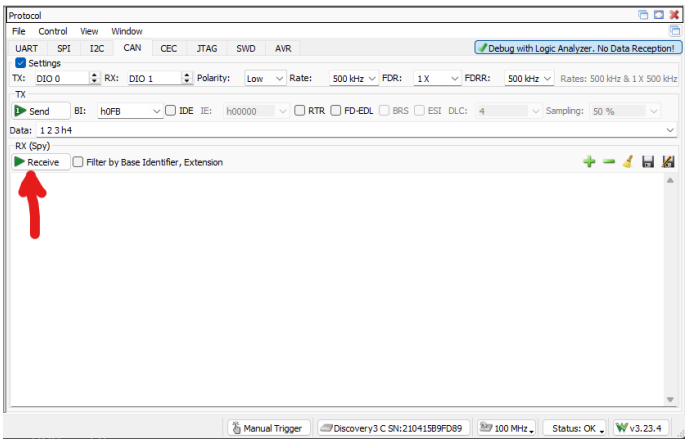
W celu włączenia zasilacza klikamy przycisk „Master Enable”



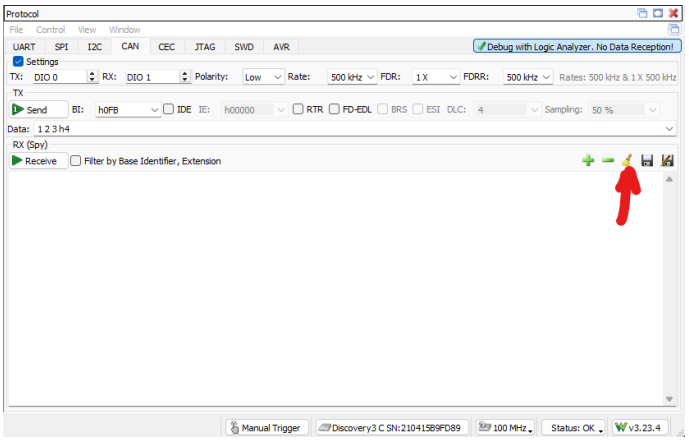
3. Konfiguracja widoku analizatora protokołów

W widoku „Protocol” wybieramy zakładkę „CAN” oraz ustawiamy parametry:

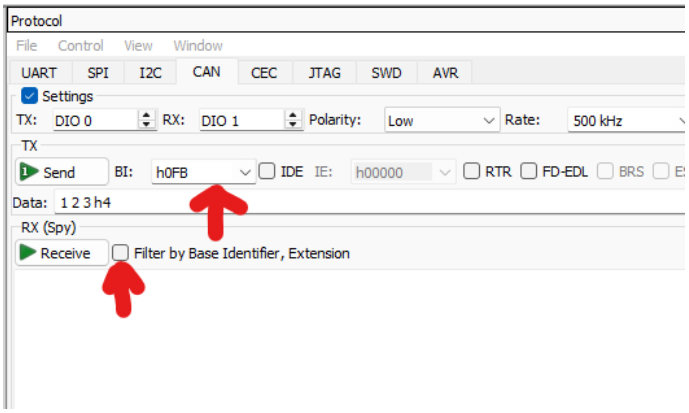
a. „Tx” DIO0 (pin cyfrowy na złączu Analog Discovery)
b. „Rx” DIO1 (pin cyfrowy na złączu Analog Discovery)
c. „Rate” 500kHz (zgodna z oprogramowaniem na STM32)
d. „Polarity” Low
e. „FDR” 1x
f. „FDDR” 500kHz
g. Na końcu klikamy przycisk „Rx (Spy)” – w poniższym ćwiczeniu będziemy tylko podglądać komunikacje w magistrali, więc reszta ustawień może pozostać w domyślnym ich stanie.



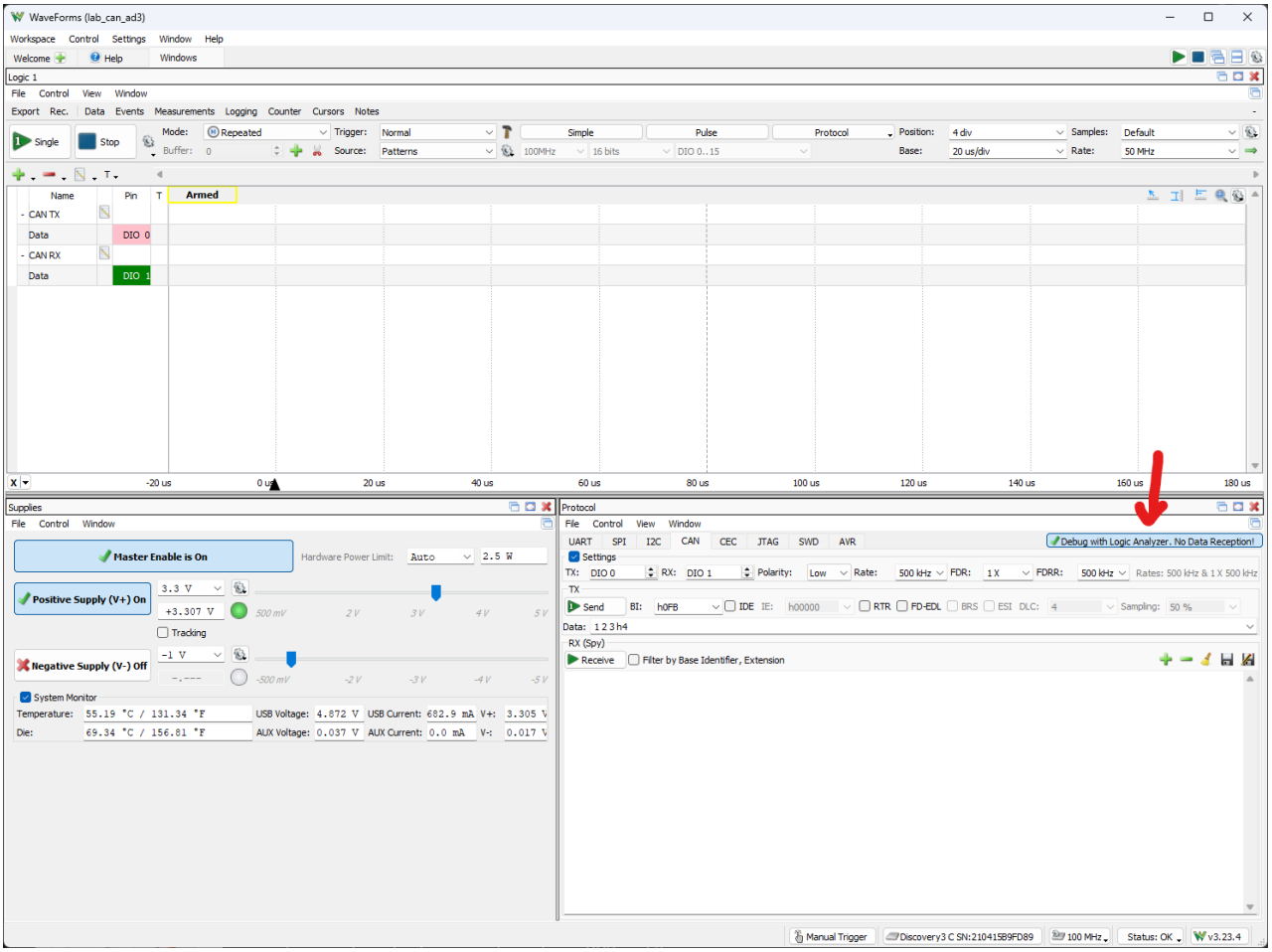
Do wyczyszczenia konsoli służy funkcja „clear”



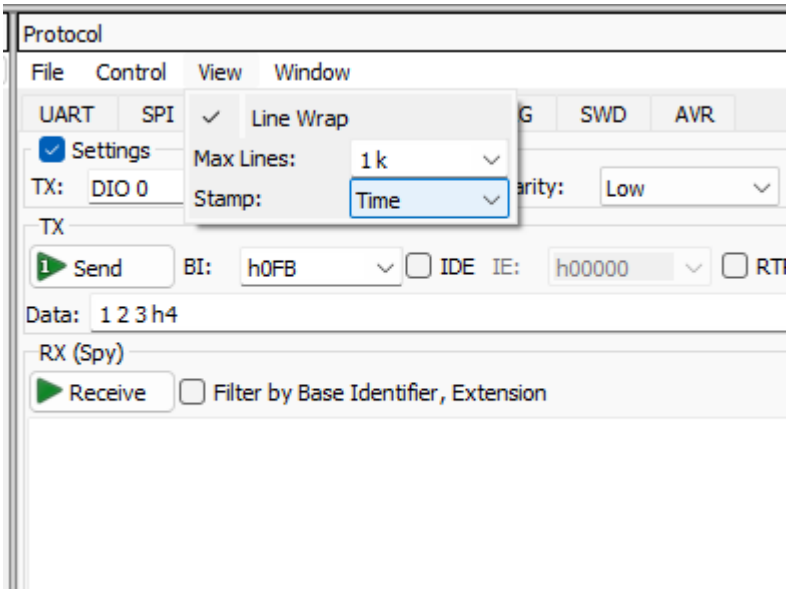
Możemy również filtrować odebrane ramki pod kątem wybranego identyfikatora



Po kliknięciu w przycisk „Debug with Logic Analyzer”, możemy podglądnąć stan linii Rx oraz Tx w formie przebiegów bitowych



W zakładce „View” możemy ustawić dodawanie znacznika czasu – przydatne jest wybranie opcji „Time”



Przydatne linki

[1] Projekt z instrukcją: https://github.com/Dominik-Tendera/CAN_laboratory.git

[2] Pinout nucleo: https://os.mbed.com/platforms/ST-Nucleo-F446ZE/

[3] stm32446 https://www.st.com/resource/en/datasheet/stm32f446mc.pdf