INSTYTUT INFORMATYKI

Wydział Informatyki, Elektroniki i Telekomunikacji AGH

Laboratorium Techniki Mikroprocesorowej

Asynchroniczna transmisja szeregowa realizowana przez interfejs USART/RS232

Opracował dr inż. W. Zaborowski. Kraków 2020-2023. Wersja 20230110.0.

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się ze sposobem szeregowego asynchronicznego przesyłania danych za pośrednictwem interfejsu zgodnego ze standardem RS232C, oraz obsługą tego typu transmisji przez mikrokontroler jednoukładowy z rodziny ATmega i komputer klasy PC.

W trakcie ćwiczenia należy zaobserwować i zinterpretować przebiegi sygnałów elektrycznych na warstwie fizycznej łącza RS232 a także wykazać się umiejętnością oprogramowania układów we/wy mikrokontrolera Atmega 2560 i komputera PC odpowiedzialnych z realizację transmisji w tym standardzie.

Wymagane wiadomości

- Podstawy obsługi środowiska Arduino,
- Obsługa komunikacji szeregowej przez USART z poziomu bibliotek Arduino,
- Budowa i sposób programowania interfejsu USART w mikrokontrolerach rodziny ATmega.
- Budowa i sposób programowania interfejsu RS232C w komputerach PC.

Wykorzystywany sprzęt

- Arduino MEGA 2560
- Urządzenie laboratoryjne NI ELVIS II z płytą prototypową
- Komputer klasy PC z zainstalowanym środowiskiem Arduino oraz oprogramowaniem NI Elvis MX

Literatura

1. Arduino – Open–source electronic prototyping platform. Dostępny w Internecie: https://www.arduino.cc/
2. https://www.arduino.cc/en/Reference/Serial
3. Atmel Corp.: Atmel *ATmega 640/V-1280/V-1281/V-2560/V-2561/V, 8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash datasheet.*. Rev.2549Q-02/2014. Dostępny w Internecie: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf
4. Pena E., Legaspi M.G.: UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter. Dostępny w Internecie: https://www.analog.com/media/en/analog-dialogue/volume-54/number-4/uart-a-hardware-communication-protocol.pdf
5. Metzger P.: *Anatomia PC*, Wyd. Helion 1996
6. Mielczarek W.: *Szeregowe interfejsy cyfrowe.* Wyd. Helion 1993

2 Informacje wprowadzające

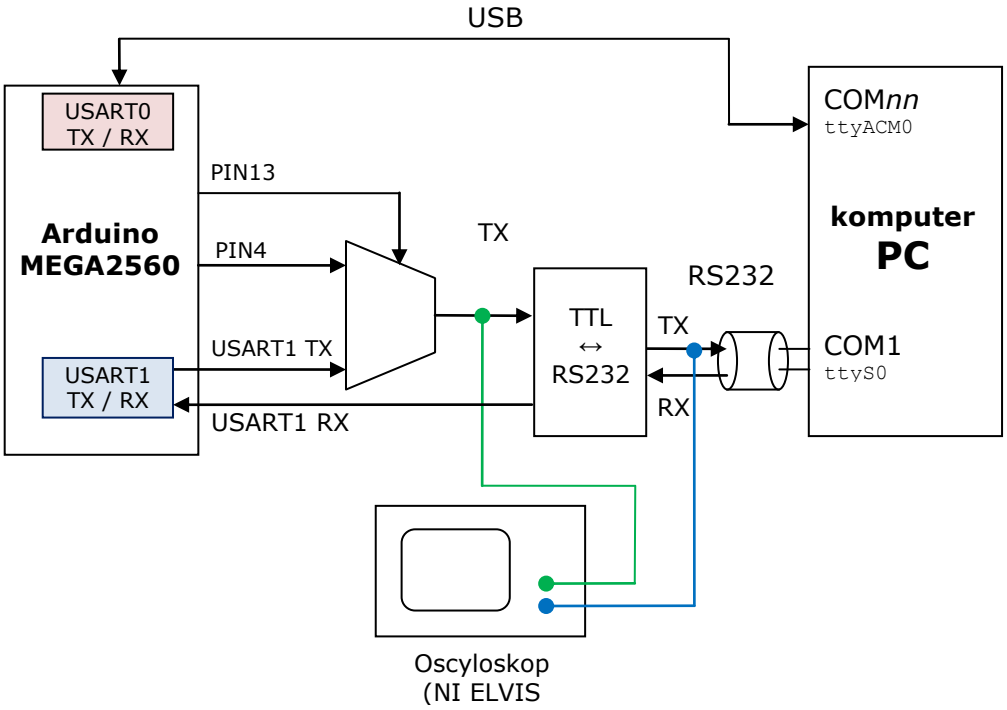
Ćwiczenie może być realizowane zarówno w formie stacjonarnej jak i zdalnej. W przypadku zdalnej realizacji ćwiczenia nie ma technicznych możliwości samodzielnego przeprowadzenia obserwacji opisanych w punkcie 3.2.

2.1 Opis stanowiska laboratoryjnego

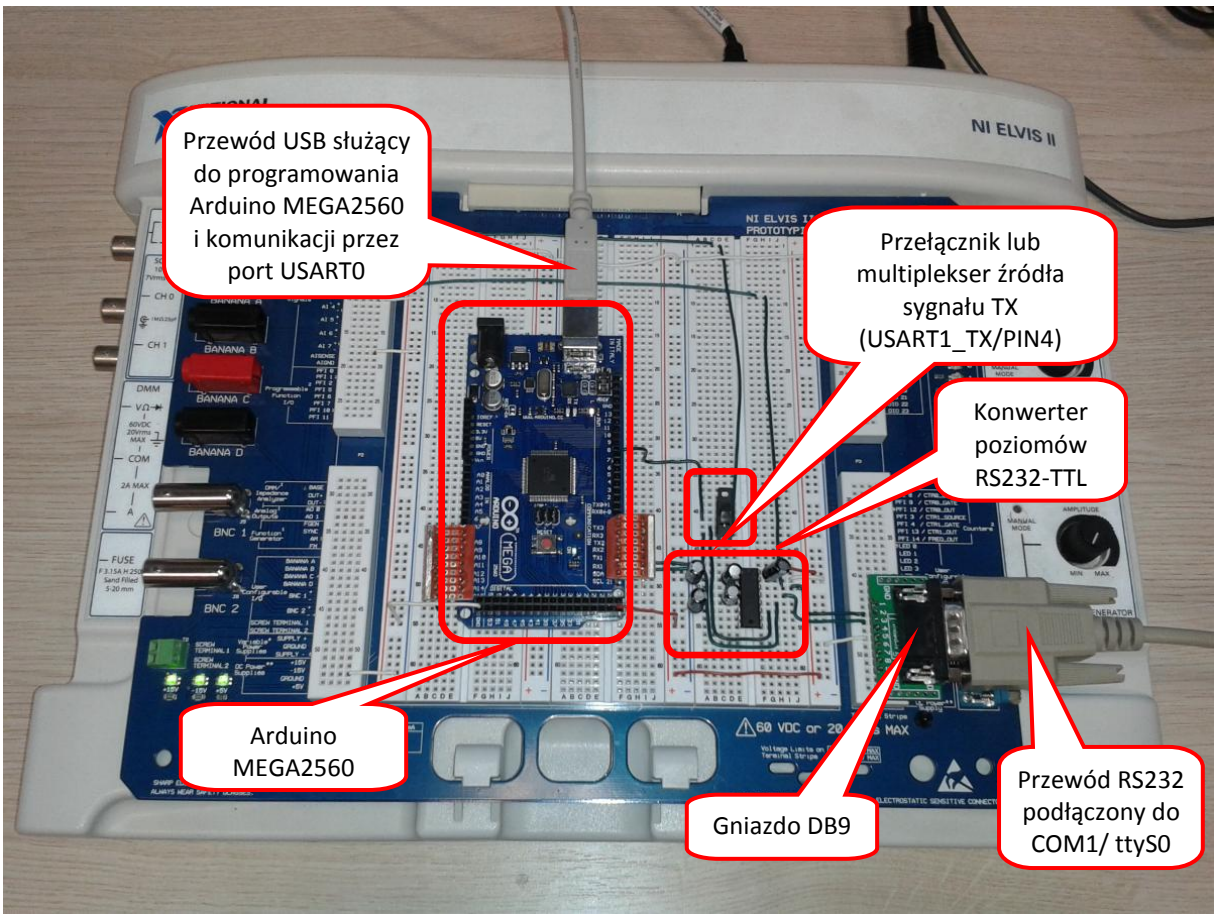
Do realizacji ćwiczenia wykorzystuje się stanowisko laboratoryjne składające się z :

- zestawu uruchomieniowego z rodziny Arduino[1], wyposażonego w mikrokontroler ATmega 2560 (Arduino MEGA2560),
- urządzenia pomiarowego NI ELVIS II oraz komputera PC z zainstalowanym niezbędnym oprogramowaniem narzędziowym,
- zmontowanego na płycie prototypowej urządzenia NI ELVIS II układu konwertera poziomów napięć opartego o układ MAX232 (lub jego odpowiednik),

Schemat stanowiska przedstawia Rys. 1 a jego wygląd fotografia zamieszczona na Rys. 2.



Rys. 1 Schemat budowy stanowiska laboratoryjnego



Rys. 2 Stanowisko laboratoryjne

Poszczególne elementy stanowiska zostały ze sobą połączone w następujący sposób:

- Płytka Arduino MEGA za pomocą przewodu USB do portu USB komputera PC ,
- Urządzenie NI ELVIS II za pomocą przewodu USB do komputera PC,
- Umieszczone na płycie prototypowej urządzenia NI ELVIS gniazdo DB9 połączone za pomocą przewodu RS232 do portu COM1 komputera PC,
- Przy pomocy przewodów zrealizowano następujące połączenia:
    - Linia TX1(PIN18) Arduino MEGA podłączona do multipleksera źródła sygnału TX,
    - Wyprowadzenie GPIO PIN 4 podłączone do multipleksera źródła sygnału TX,
    - Wyjście multipleksera źródła sygnału TX do wyprowadzenia AI1+ i wejścia T1IN (nóżka 11) układu konwertera poziomów logicznych MAX232 ,
    - Linia sterująca multipleksera źródła sygnału TX dołączona jest do wyprowadzenia PIN13 Arduino.
    - Linia RX1 (PIN19) Arduino MEGA dołączona do wyjścia R1OUT (nóżka 12) układu konwertera poziomów napięć MAX232,
    - Sygnał z pinu2 złącza DB9 (RS232 TxD) do wejścia R1IN (nóżka 13) układu konwertera poziomów napięć MAX232 i wyprowadzenia AI4+,
    - Sygnał z pinu 3 złącza DB9 (RS232 RxD) do wejścia T1OUT (nóżka 14) układu konwertera poziomów napięć MAX232,
    - Aplikację układu MAX232,
    - Niezbędne połączenia GND i zasilania.

Uwaga! Multiplekser źródła sygnału TX służy do wyboru źródła transmitowanych z mikrokontrolera danych. Domyślnym i wykorzystywanym przez większą część ćwiczenia źródłem jest wyjście TX interfejsu USART1 mikrokontrolera. Multiplekser jest sterowany sygnałem z wyprowadzenia PIN13 Arduino i skonfigurowany został w taki sposób, aby domyślnym źródłem sygnału TX było wyjście TX interfejsu USART1. Oznacza to, że o ile nie podana została w sposób jawny informacja, że należy zmienić stan wyprowadzenia PIN13, to nie należy w żaden sposób próbować sterować tym wyprowadzeniem lub skonfigurować je jako wyjście i ustawić na nim stan wysoki.
Przystępując do realizacji zadania 3.5 należy wysterować PIN13 stanem niskim. Wówczas źródłem sygnału TX stanie się wyprowadzenie PIN4 Arduino.

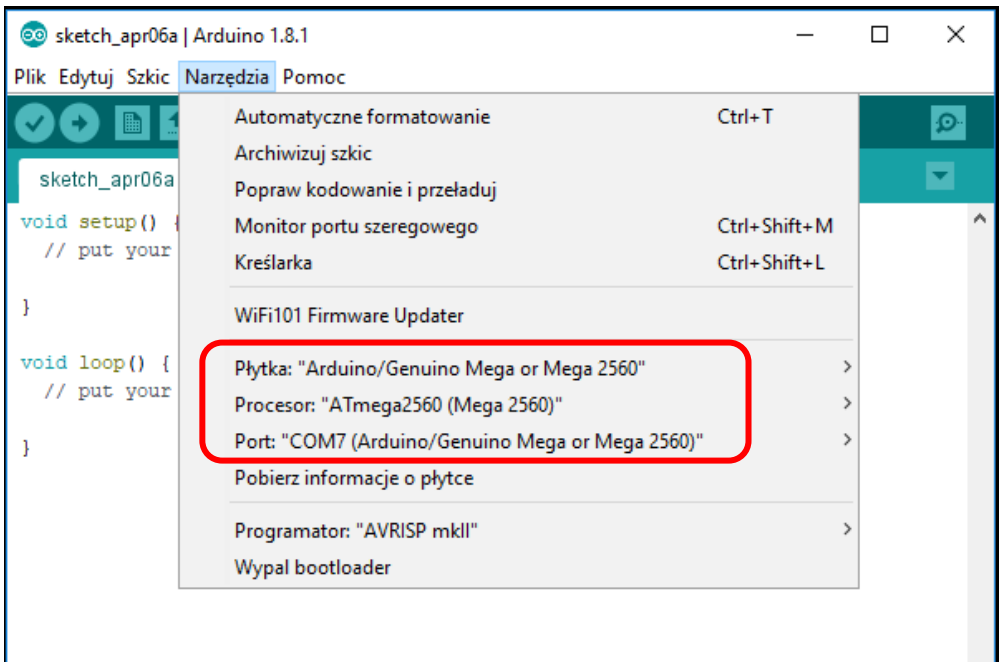
3 Wykonanie ćwiczenia

3.1 Zadanie wprowadzające

3.1.1 Wstępna konfiguracja środowiska Arduino IDE

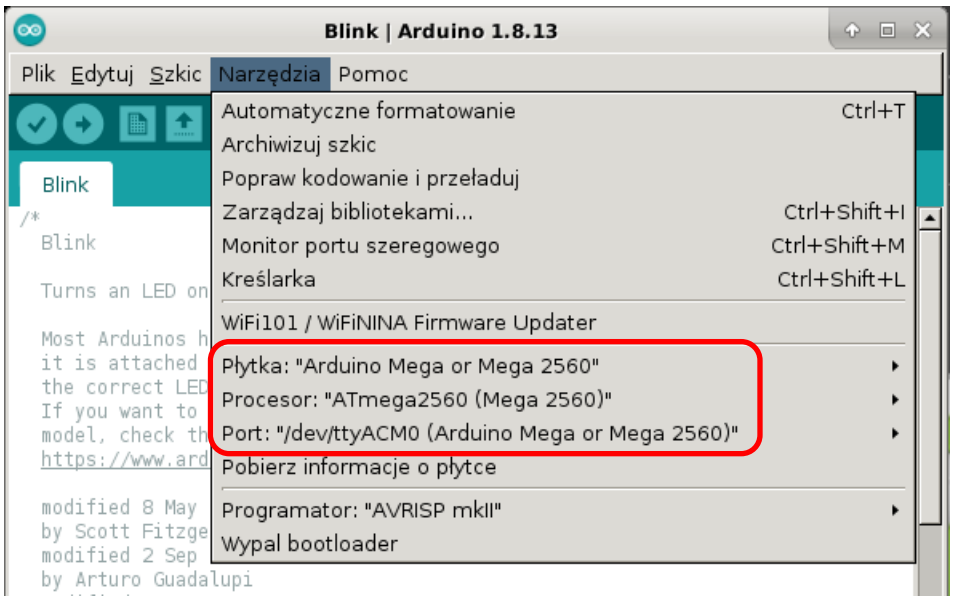
Ćwiczenie jest realizowane z wykorzystaniem platformy Arduino, wraz z którą dostarczane jest Arduino IDE. Środowisko Arduino IDE umożliwia tworzenie aplikacji dla wielu różnych platform sprzętowych, w tym urządzeń opartych o platformę Arduino. Niewłaściwa konfiguracja środowiska uniemożliwi uruchomienie jakiegokolwiek programu na mikrokontrolerze, a zatem przed podjęciem prób uruchamiania jakichkolwiek aplikacji, należy zweryfikować poprawność konfiguracji IDE i w razie potrzeby dokonać odpowiednich zmian.

W ćwiczeniu wykorzystywana jest płytka Arduino MEGA2560 z mikrokontrolerem ATmega2560[3]. Zatem pierwszym krokiem jest sprawdzenie czy środowisko jest skonfigurowane do pracy z właściwą platformą. W tym celu należy rozwinąć menu Narzędzia i sprawdzić pozycje: „Płytka: ”, „Procesor:”, „Port:”. Przykładowa, prawidłowa konfiguracja widoczna jest na Rys. 3 (Uwaga! Numer portu COM może być inny. Zdecydowanie **nie powinien** być to port COM1). W razie potrzeby należy wprowadzić odpowiednie zmiany w konfiguracji poczynając od pozycji „Płytka:”.



Rys. 3 Konfiguracja Arduino IDE

W przypadku gdy środowisko Arduino IDE uruchomione zostało na komputerze pracującym pod kontrolą systemu operacyjnego z rodziny Linux, zmianie ulegają nazwy portów komunikacyjnych na charakterystyczne dla tych systemów. W tej sytuacji Arduino IDE komunikuje się z płytką Arduino MEGA2560 najczęściej przez port /dev/ttyACM0. Przykładową konfigurację Arduino IDE pracującego pod kontrolą systemu Linux pokazano na Rys. 4.



Rys. 4 Konfiguracja Arduino IDE w systemie Linux

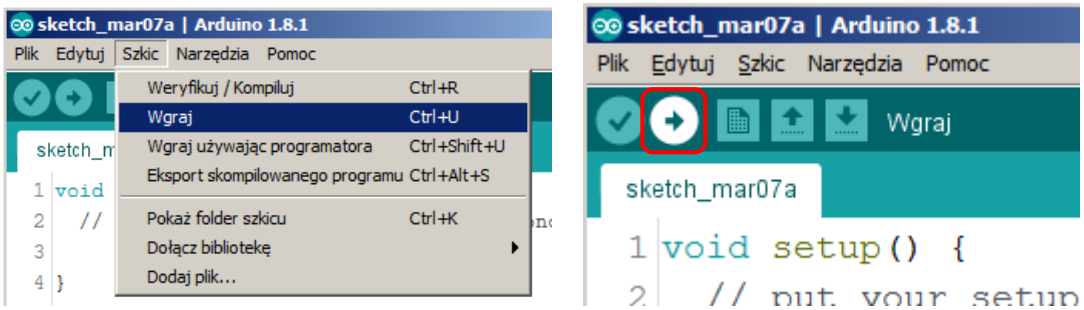
Zestawy uruchomieniowe należące do rodziny Arduino posiadają co najmniej jeden port komunikacyjny USART. Wykorzystywany jest on przez środowisko Arduino IDE do programowania pamięci FLASH mikrokontrolera (wykorzystuje w tym celu bootloader). Rozwiązanie takie znacząco ułatwia programowanie wspomnianej juz pamięci FLASH mikrokontrolera, jednak ogranicza i komplikuje wykorzystanie portu USART mikrokontrolera w aplikacjach użytkownika. Poza programowaniem pamięci FLASH, połączenie to w łatwy sposób można wykorzystać do komunikacji pomiędzy komputerem i aplikacją mikrokontrolera. Środowisko Arduino IDE dostarcza w tym celu własny emulator terminala, uruchamiany z menu **Narzędzia→Monitor portu szeregowego**.

3.1.2 Sprawdzenie poprawności działania stanowiska

Wysokopoziomowa obsługa komunikacji szeregowej poprzez port (lub porty) USART mikrokontrolera realizowana jest za pośrednictwem klasy Serial [2].

Przed przystąpieniem do realizacji ćwiczenia zalecane jest zapoznanie się z dokumentacja klasy Serial. Warto również przeanalizować jeden z dostarczanych wraz ze środowiskiem przykładowych szkiców np.: ASCIITable (dostępne z menu Arduino IDE: **Plik→Przykłady→04.Communication→ASCIITable**) wykorzystujący tą klasę.
Uwaga! – wybrany szkic otworzy się w nowym oknie.

Przy okazji uruchamiania wymienionego wcześniej przykładu, można przeprowadzić weryfikację poprawności konfiguracji środowiska Arduino IDE. Po załadowaniu wymienionego wcześniej szkicu **ASCIITable**, należy go skompilować i zaprogramować nim mikrokontroler. W tym celu z menu **Szkic** wybieramy polecenie **Wgraj** lub wciskamy odpowiedzialny za tą samą akcje, widoczny pod paskiem menu przycisk z ikoną strzałki w prawo (obie te czynności są widoczne na Rys. 5).

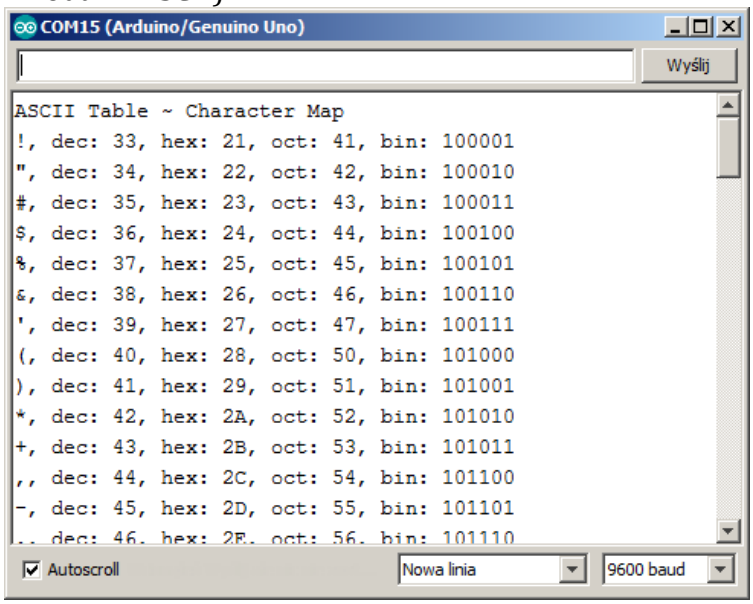


Rys. 5 Kompilacja i ładowanie programu do Arduino

W tym momencie na pasku poniżej okna z kodem źródłowym pojawi się kolejno informacja o kompilowaniu szkicu a następnie o jego ładowaniu do pamięci mikrokontrolera. Na zakończenie poprawnego wgrywania skompilowanego kodu wyświetlany jest komunikat „Ładowanie zakończone."

Jeśli pojawi się inny komunikat, w szczególności komunikat o błędzie, należy sprawdzić poprawność podłączenia płytki Arduino do komputera oraz konfiguracji Arduino IDE.

Jeśli natomiast ładowanie zakończy się sukcesem, można uruchomić monitor portu szeregowego („**Narzędzia→Monitor portu szeregowego**" lub Ctrl+Shift+M) i obserwować, pokazany na Rys. 6, rezultat wykonania przykładowego programu (w postaci listy znaków wraz z ich kodami ASCII).



Rys. 6 Wykonanie programu ASCIITable

3.1.3 Testowanie komunikacji przez USART1 i port COM1/ttyS0

Ostatnim etapem testowania stanowiska jest sprawdzenie poprawności komunikacji z portem COM1/ttyS0 komputera PC. W Arduino MEGA połączenie z portem COM1/ ttyS0 obsługiwane jest za pośrednictwem portu USART1, którego sygnały dostępne są na wyprowadzeniach PIN18(TX_1) i PIN19(RX_1).

Działający poprawnie szkic ASCIITable należy poddać modyfikacji polegającej na dodaniu inicjalizacji transmisji przez port USART1 i powieleniu wysyłania znaków przez ten port. Zmodyfikowany szkic należy skompilować i uruchomić na Arduino.

Na komputerze PC należy uruchomić dowolny program emulujący terminal (np. ExtraPUTTY lub TeraTerm) i połączyć się z portem COM1/ttyS0. Należy również ustawić prawidłowe parametry transmisji (standardowo jest to prędkość 9600bps, 8 bitów danych, 1 bit stopu, brak kontroli przepływu). Przy prawidłowej konfiguracji wszystkich elementów stanowiska (szczególną uwagę należy zwrócić na położenie przełącznika źródła sygnału TX), w oknie właśnie uruchomionego terminala powinna wyświetlić się identyczna treść do tej, wyświetlającej się w monitorze portu szeregowego środowiska Arduino.

3.2 Obserwacja sygnałów na liniach portu USART mikrokontrolera i liniach transmisyjnych RS232

W trakcie tej części ćwiczenia należy przeprowadzić obserwacje i pomiary parametrów sygnałów na wyjściach portu USART1 mikrokontrolera oraz na liniach transmisyjnych interfejsu RS232.

UWAGA! W przypadku zdalnej realizacji ćwiczenia, nie ma możliwości technicznych wykonania tego zadania samodzielnie. Może zostać ono zrealizowane w formie pokazu.

Mikrokontroler ATmega2560 wyposażony jest w 4 identyczne interfejsy USART (numerowane kolejno od 0 do 3). Jak wspomniano wcześniej, port USART0 jest wykorzystywany przez bootloader i środowisko Arduino IDE m.in. do programowania pamięci FLASH mikrokontrolera. Sama transmisja danych pomiędzy mikrokontrolerem a komputerem przez ten port została „dodatkowo schowana" w połączeniu USB.

Aby zaobserwować „fizyczny" sposób przesyłania informacji pomiędzy mikrokontrolerem a komputerem przez interfejs zgodny ze standardem RS232, wygodnie będzie wykorzystać jeden z dodatkowych portów USART mikrokontrolera oraz zrealizować dodatkowe połączenie pomiędzy mikrokontrolerem a komputerem PC.

Sygnały elektryczne na wyjściach portów USART mikrokontrolera nie są kompatybilne z sygnałami interfejsu RS232. Konieczne jest zastosowanie odpowiedniego konwertera. Rolę takiego konwertera może pełnić układ MAX232 lub jego odpowiednik. Moduł konwertera został podłączony pomiędzy linie RX i TX portu USART1 mikrokontrolera ATmega2560 a złącze DB9 znajdujące się na płycie prototypowej. Dodatkowo zrealizowane zostały połączenia umożliwiające obserwację za pomocą urządzenia NI ELVIS II sygnałów bezpośrednio na wyjściach portu USART1 mikrokontrolera oraz w przewodzie transmisyjnym RS232.

Jak już wspomniano wcześniej, w komputerze PC wykorzystany zostanie sprzętowy port COM1/ttyS0, który służy do komunikacji zgodnej ze standardem RS232. Odpowiednie sygnały elektryczne będą przesyłane pomiędzy stanowiskiem laboratoryjnym a komputerem PC przez typowy, zakończony wtykami DB-9, przewód transmisyjny RS232 typu **DTE-DCE**. Komputer będzie pełnił rolę urządzenia DTE a Arduino rolę urządzenia DCE. Do obserwacji sygnałów elektrycznych wykorzystane zostanie urządzenie laboratoryjne NI ELVIS II, które wraz z odpowiednim oprogramowaniem będzie pełniło funkcje cyfrowego oscyloskopu.

3.2.1 Oprogramowanie mikrokontrolera ATmega2560

Dla ułatwienia obserwacji przebiegów na oscyloskopie, należy przygotować program (szkic) dla Arduino, który będzie realizował następujące czynności:

1. Zainicjalizuje port USART 1 do transmisji z jedną ze standardowych prędkości (2400 b/s, 4800 b/s, 9600 b/s, 19200 b/s)
2. W nieskończonej pętli będzie realizował następujące czynności:
    a) Wysłanie wybranego znaku przez port USART1
    b) Odczekanie 2-3 ms
    c) Ponowne wysłanie tego samego znaku przez port USART1
    d) Wysłanie znaku o kodzie o 1 ASCII o jeden większym lub o 1 mniejszym niż wysłany w punktach a)) i c))
    e) Odczekanie 10ms

Taki sposób wysyłania znaków umożliwi w miarę łatwe, prawidłowe zidentyfikowanie na zarejestrowanych przy pomocy oscyloskopu przebiegach momentu zakończenia ramki (SDU) zawierającej przesyłane przez RS232 dane.

Format ramki zawierającej przesyłane dane należy ustalić na 8N1(jest to format domyślnie wykorzystywany przez klasę Serial).

Nie zaleca się obserwacji przebiegów przy prędkościach większych niż 19200 ponieważ mogą wystąpić problemy z poprawnym zarejestrowaniem sygnałów przy pomocy oscyloskopu.

3.2.2 Obserwacje i pomiary

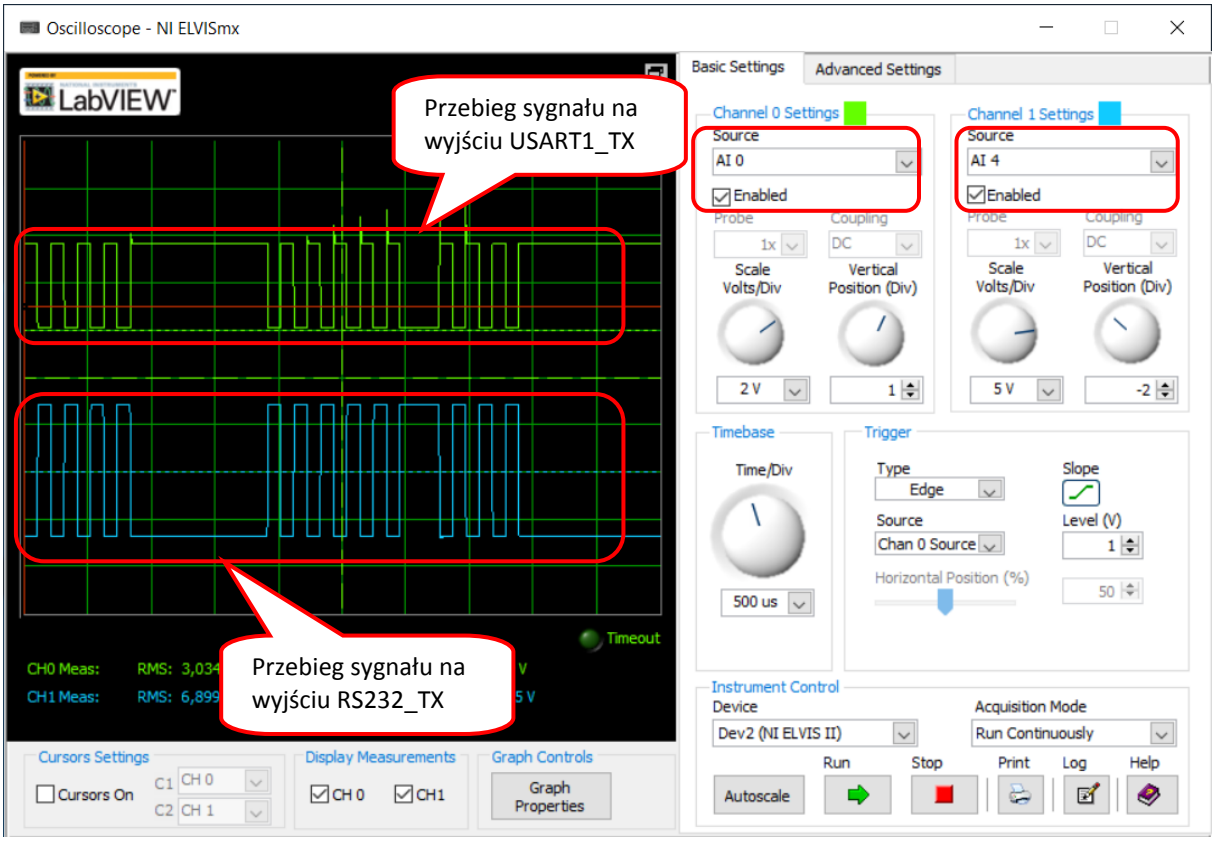
UWAGA! – W przypadku zdalnego wykonywania zadania nie ma możliwości technicznych własnoręcznego przeprowadzenia omówionych w dalszej części pomiarów.

Przy pomocy oscyloskopu urządzenia NI ELVIS II należy zarejestrować proces przesłania ramki (SDU) zawierającej wybrany znak. Należy w tym celu uruchomić oprogramowanie NI ELVISmx Oscilloscope (odpowiedni skrót umieszczony jest na pulpicie) i odpowiednio je skonfigurować. Rejestrowane sygnały zostały podłączone do urządzenia NI ELVIS II w następujący sposób:

- sygnał z wyjścia USART1TX do wejścia AI0
- sygnał z linii RS232_TX dostępnej na wyprowadzeniu nr 2 złącza DSUB do wejścia AI4.

Oscyloskop należy skonfigurować w taki sposób aby na kanale 0 widoczny był przebieg rejestrowany na wyjściu USART1TX a na kanale 1 przebieg sygnału na linii RS232_TX.

Prawidłowo skonfigurowany oscyloskop wraz z przykładowymi, poprawnymi przebiegami sygnałów zarejestrowanymi na wyjściach USART1TX i RS232_TX jest widoczny na Rys. 7.



Rys. 7 Konfiguracja oscyloskopu.

Przystępując do obserwacji i wiedząc jak teoretycznie powinna wyglądać ramka SDU na wyjściu USART1TX, warto się zastanowić nad wybraniem znaku, o takim kodzie ASCII, który pozwoli łatwo rozpoznać poszczególne bity (np. znaki o kodach ASCII w których druga cyfra szesnastkowa ma wartość 5).

Wykorzystując kursory należy dokonać pomiaru wartości napięć na wyjściu USART1TX oraz na linii TX RS232 odpowiadających przesyłaniu bitu o wartości 0 i o wartości 1. Należy również zmierzyć czas trwania całej ramki i czas przesyłania pojedynczego bitu. Sposób przeprowadzenia pomiarów jest objaśniony w dalszej części tego rozdziału.

W sprawozdaniu można umieścić, wypełnioną odczytanymi z zarejestrowanych przebiegów wartościami, tabelę:

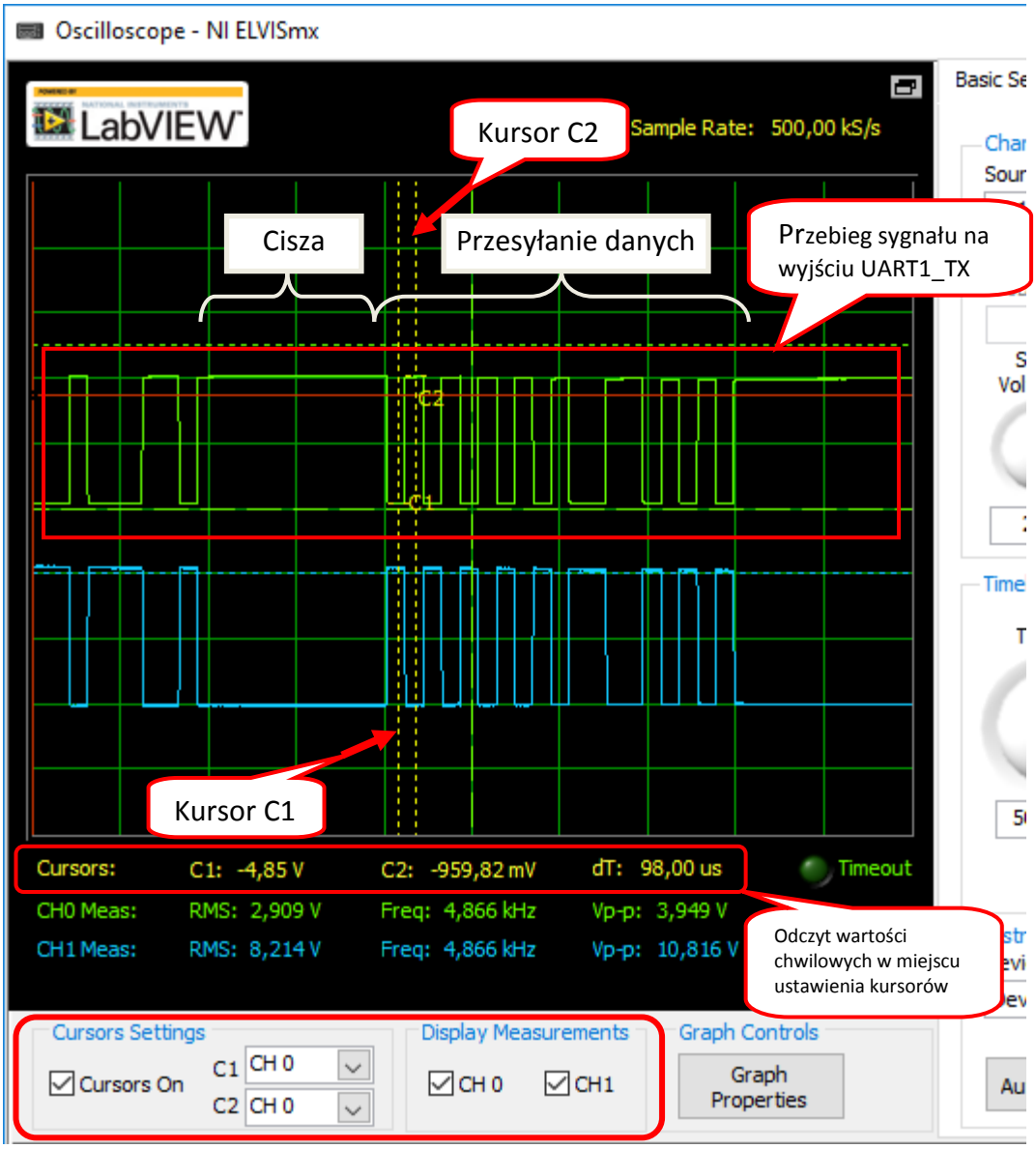
	Wyjście USART_TX	Wyjście RS232_TX
Napięcie odpowiadające bitowi o wartości **1**		
Napięcie odpowiadające bitowi o wartości **0**		
Prędkość transmisji, format ramki		
Czas przesłania całej ramki (SDU)		
Czas trwania pojedynczego bitu o wart. **0**		
Czas trwania pojedynczego bitu o wart. **1**		

Należy również dokonać interpretacji danych zarejestrowanych przy pomocy oscyloskopu, zaznaczając na nich fragmenty przebiegów odpowiadające ramce SDU, bitowi startu, bitowi stopu, wartości poszczególnych bitów oraz określając jaki znak ASCII był przesyłany podczas obserwacji.

Obserwacje i pomiary należy udokumentować. Właściwy sposób dokumentowania jest objaśniony w dalszej części tego rozdziału.

Pomiary parametrów sygnału na wyjściu USART1_TX mikrokontrolera

Konfiguracja oscyloskopu umożliwiająca pomiar napięć na wyjściu USART1_TX mikrokontrolera została opisana wcześniej. Na Rys. 8 pokazano szczegółowe informacje w jaki sposób korzystać z kursorów oscyloskopu w celu poprawnego odczytania parametrów rejestrowanych przebiegów.

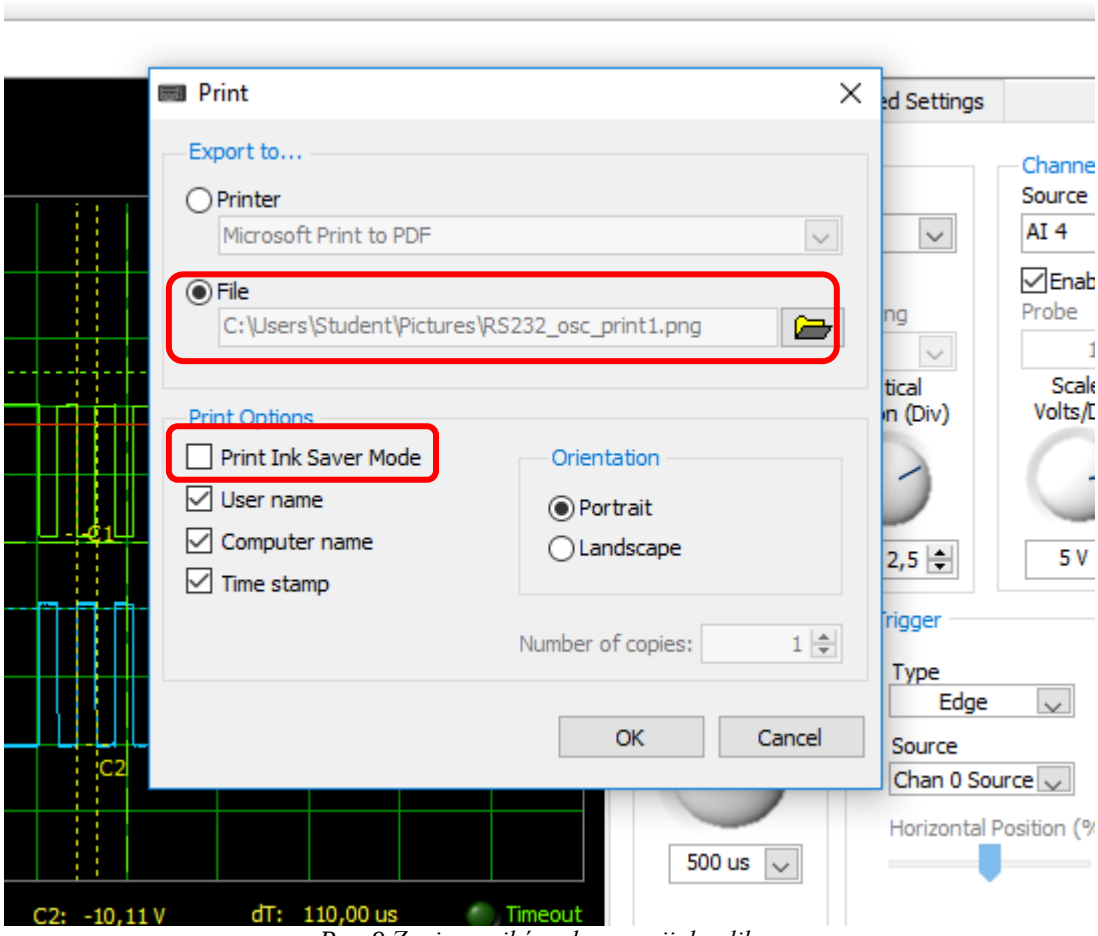


Rys. 8 Pomiary parametrów sygnałów na wyjściu UART1_TX

Aby móc korzystać z kursorów należy zaznaczyć pole **Cursors On**. Oscyloskop udostępnia dwa kursory oznaczane **C1** i **C2**. Dla sprawnego przeprowadzenia pomiaru należy zadbać aby oba kursory pokazywały wartości rejestrowane w kanale 0 (**CH 0**) oscyloskopu (ten kanał rejestruje napięcie na wyjściu USART1_TX) i dla obu wejść były zaznaczone pola **Display Measurements** odpowiedzialne za wyświetlanie danych rejestrowanych przez kursory.

Następnie za pomocą myszy należy ustawić kursor C1 w miejscu które odpowiada stanowi niskiemu na wyjściu USART1_TX a kursor C2 ustawić w miejscu odpowiadającym stanowi wysokiemu na tym samym wyjściu (najlepiej ustawić kursory w połowie czasu fragmentu przebiegu odpowiadającego przesyłaniu poszczególnych bitów, tak jak to pokazano na Rys. 8). W tym momencie można odczytać wartości napięć odpowiadające stanowi wysokiemu i niskiemu.

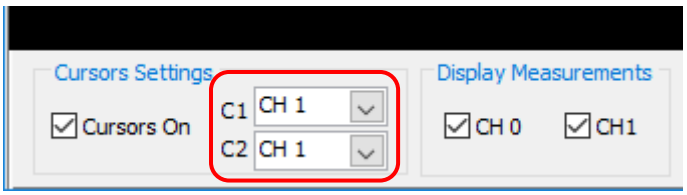
Obserwacje należy udokumentować korzystając z funkcji **Print** oscyloskopu (przycisk staje się aktywny po zatrzymaniu rejestracji danych przez oscyloskop). Po wciśnięciu przycisku **Print** pojawia się, widoczne na Rys. 9 okno, w którym wybiera się jakie dane mają zostać utrwalone oraz wskazuje plik w którym zostaną zapisane odpowiednie dane. Przed zapisaniem danych, należy koniecznie **odznaczyć** opcję **Print Ink Saver Mode**.



Rys. 9 Zapis wyników obserwacji do pliku.

Pomiary parametrów sygnału na wyjściu RS232_TX

Aby uzyskać wartości napięć odpowiadających odpowiednio przesyłaniu bitu o wartości 0 oraz 1 na wyjściu RS232_TX wystarczy przełączyć kursory ustawione podczas pomiaru napięć na wyjściu USART1_TX, w tryb w którym będą pokazywały chwilowe wartości rejestrowane w kanale 1 (CH 1) oscyloskopu (tak jak to jest pokazane na Rys. 10). Nie należy zmieniać położenia kursorów.



Rys. 10 Ustawienie trybu pracy kursorów dla pomiaru napięć sygnału RS232_TX

Odczytane wartości napięć należy zanotować do sprawozdania.

Pomiar czasu przesyłania jednego bitu

Pomiar można przeprowadzić dla sygnału na wyjściu USART1_TX lub RS232_TX. Jeśli znak, którego przesyłanie jest obserwowane został dobrze dobrany, wystarczy ustawić kursor C1 w taki sposób, aby znalazł się na początku przesyłanego bitu a kursor C2 na jego końcu.

Czas trwania transmisji bitu widoczny jest wówczas w polu odczytowym kursorów jako wartość wyświetlana przy **dT.**

Pomiar czasu transmisji całej ramki danych (SDU)

Pomiar czasu transmisji całej ramki jest bardzo podobny do pomiaru czasu transmisji pojedynczego bitu, wymaga jednak prawidłowego zidentyfikowania momentu w którym ramka się zaczyna i w którym się kończy.

Początek ramki jest łatwy do zidentyfikowania, ponieważ przed nim w medium transmisyjnym trwa cisza, reprezentowana przez stan wysoki. Transmisja rozpoczyna się zawsze bitem startu, którego wartość wynosi 0. Opisany w punkcie 3.2.1 program testowy ułatwia to zadanie. W prawidłowo zidentyfikowanym miejscu rozpoczęcia ramki należy ustawić kursor C1.

Trudniej jest prawidłowo zidentyfikować koniec ramki, ponieważ bit stopu ma wartość 1 a po nim zwykle następuje cisza w medium transmisyjnym, która również odpowiada wartości 1. Pomocne może się okazać narysowanie na kartce papieru teoretycznego przebiegu sygnału powstałego jako rezultat wykonania programu testowego. Ułatwieniem może się również okazać zdobyta wcześniej wiedza związana czasem trwania transmisji pojedynczego bitu. W prawidłowo zidentyfikowanym miejscu przebiegu należy ustawić kursor C2. Zanotować do sprawozdania wynik pomiaru, oraz udokumentować go za pomocą funkcji **Print**.

W ramach opracowania zebranych danych należy na zarejestrowanych przebiegach zidentyfikować i opisać znaczenie poszczególnych bitów oraz na ich podstawie określić jaki znak był transmitowany.

3.3 Sterownik USART cz.1

W ramach tej części ćwiczenia, należy przygotować niskopoziomowy sterownik poru USART1 mikrokontrolera ATmega2560 oraz przykładową aplikacje dla Arduino prezentującą jego działanie. Sterownik nie może być „opakowaniem" korzystającym z wysokopoziomowych bibliotek dla Arduino (np. z metod dostarczanych przez klasę Serial).

Realizacja tego zadania wymaga znajomości na poziomie podstawowym budowy i zasady działania interfejsu USART mikrokontrolera ATmega2560. W ramach przygotowania do realizacji tego zadania, należy zapoznać się z rozdziałem 22 (USART -

Universal Synchronous Asynchronous Receiver Transceiver) dokumentacji mikrokontrolera [3].

Od strony komputera PC do komunikacji wykorzystywany będzie sprzętowy port COM1/ttyS0. Na komputerze należy uruchomić (o ile nie zostało to zrobione wcześniej) dowolny program terminalowy (np. ExtraPutty, Putty lub TerraTerm) i połączyć się z Arduino za pośrednictwem odpowiednio skonfigurowanego portu COM1/ttyS0 (należy zwrócić uwagę na prędkość transmisji oraz format przesyłanych danych). W przypadku niepoprawnej konfiguracji portu COM1 ttyS0, w oknie terminala mogą pojawiać się przypadkowe, niepoprawne „dziwne" znaki.

Przygotowanie sterownika wymagać będzie odwołania się do odpowiednich rejestrów interfejsu USART1 oraz znajdujących się w nich bitów. Aby móc posługiwać się symbolicznymi nazwami rejestrów urządzeń we/wy (np. odwołanie do rejestru prędkości transmisji realizować poprzez symboliczną nazwę *UBRRn*) i znajdujących się w nich bitów (np. bit 4 rejestru *UCSR0B* nosi nazwę symboliczną *RXEN0* i steruje załączaniem i wyłączaniem części odbiorczej portu USART0), należy dołączyć do projektu plik nagłówkowy **"avr/io.h"** który zawiera odpowiednie definicje.

Przygotowany sterownik powinien dostarczać minimum trzech funkcji:

void Init () – inicjalizująca niezbędne elementy mikrokontrolera i konfigurująca parametry transmisji: szybkość i format przesyłanych danych.

char GetChar () – odbierająca znak z portu UART,

void PutChar(char znak) – wysyłająca znak przez port UART.

Najbardziej złożonym zadaniem w tej części ćwiczenia jest przygotowanie funkcji inicjalizującej.

Rejestry interfejsu USART powinny zostać skonfigurowane w taki sposób aby mikrokontroler wysyłał odbierał dane z prędkością 2400b/s w formacie **8,N,1** (8 bitów danych, brak bitu parzystości, 1bit stopu).
Do wyznaczenia wartości jaka powinna zostać wpisana do rejestru USART odpowiedzialnego za kontrolę prędkości transmisji konieczna jest znajomość częstotliwości zegara taktującego mikrokontroler. Arduino MEGA 2560 taktowane jest zegarem o częstotliwości 16MHz. Informacje niezbędne do tego znajdują się w rozdziale 22 dokumentacji mikrokontrolera[3] .

Funkcja *GetChar ()* powinna oczekiwać na nadejście znaku (testując odpowiedni znacznik w rejestrze kontrolno-statusowym UCSR*n*A interfejsu USART1), a następnie kopiować odebrany znak z bufora portu szeregowego.

Funkcja *PutChar (...)* powinna sprawdzać, czy rejestr danych do wysłania może przyjąć nowy znak (odpowiada za to znacznik UDRE*n*), umieszczać przeznaczony do wysłania znak w rejestrze danej do wysłania, oraz poczekać, do momentu, aż znak zostanie wysłany.

W celu przetestowania poprawności działania napisanych sterowników, należy stworzyć oprogramowanie demonstrujące poprawność ich działania. Powinna to być aplikacja odbierająca pojedynczy znak przesłany do portu USART1 mikrokontrolera (jest to port, przez który mikrokontroler jest połączony z portem COM1/ttyS0

komputera PC), i odsyłająca go powrotem do nadawcy powiększony o 1. Prowadzący może udzielić własnych wskazówek odnośnie realizacji tego zadania.

Do testowania i prezentacji poprawności działania aplikacji mikrokontrolera należy po stronie komputera PC skorzystać z dowolnego programu terminalowego (Putty, TerraTerm ,ExtraPutty), pamiętając o odpowiedniej konfiguracji połączenia.

Przeprowadzić eksperyment polegający na tym, że z funkcji wysyłającej znak zostanie usunięty warunek sprawdzania czy można wysyłać kolejny znak, a następnie wysłać kolejno dwa różne znaki. Na oknie terminala uruchomionego na komputerze PC zaobserwować co zostanie odebrane.

Wyniki prac należy przedstawić prowadzącemu do oceny a odpowiednie wnioski zmieścić w sprawozdaniu.

3.4 Sterownik USART cz.2

Jak wcześniej wspomniano, interfejsy USART w mikrokontrolerach rodziny ATmega może nadawać/odbierać znaki z różnymi prędkościami.
W ramach tej części ćwiczenia należy zaproponować i zaimplementować modyfikację funkcji inicjalizującej USART1 w taki sposób, by programista mógł przy wywołaniu określać, z jaką prędkością będzie się USART komunikował.
Powinny być dostępne wszystkie możliwe do uzyskania, typowe prędkości, od 2400 do 115200 b/s. Sposób przekazania informacji o prędkości do funkcji inicjalizującej jest dowolny.

Poprawnie działające rozwiązanie należy przedstawić prowadzącemu lub udokumentować w inny sposób.

3.5 Programowa implementacja nadajnika USART

Celem tej części ćwiczenia jest stworzenie programowej implementacji nadajnika USART mogącego wysyłać dane z Arduino za pośrednictwem dowolnie wybranego wyprowadzenia GPIO mikrokontrolera ATmega2560. **Konfiguracja stanowiska laboratoryjnego wymusza wykorzystanie w tym celu tylko i wyłącznie wyprowadzenia PIN4.**

Rozpoczynając realizację tego zadania należy pamiętać o przełączeniu multipleksera źródła sygnału TX (należy w tym celu skonfigurować wyprowadzenie PIN13 jako wyjście i ustawić na nim stan niski).

Realizując to zadanie można korzystać wyłącznie z następujących funkcji:

`pinMode(PIN,DIR);` - ustalenie kierunku pracy wyprowadzenia **`PIN`** mikrokontrolera (**`INPUT : INPUT_PULLUP : OUTPUT`**),

`digitalWrite(PIN,STATE);` - ustawianie stanu wyprowadzenia **`PIN`** na wartość **`STATE = (LOW : HIGH)`**

`delayMicroseconds(long TIME);`- funkcja realizująca opóźnienie podawane w mikrosekundach.

Format przesyłanych danych można ustalić na podstawie informacji znajdujących się m.in. w rozdziale 22 .4 dokumentacji mikrokontrolera [3].

Czas wysyłania poszczególnych bitów należy obliczyć lub ustalić na podstawie wyników pomiarów zrealizowanych w ramach zadania 3.2.

Do demonstracji poprawnego działania stworzonej funkcji można wykorzystać szkic przygotowany w ramach realizacji zadania 3.3.

Po uzyskaniu poprawnej transmisji (w oknie emulatora terminala na komputerze PC poprawnie wyświetlać się będą znaki lub komunikaty wysyłane z Arduino) należy:

a) Jeśli ćwiczenie jest realizowane w trybie stacjonarnym, zarejestrować przy pomocy oscyloskopu przykładową wysłaną ramkę SDU, określając czas trwania transmisji całej ramki oraz czas transmisji pojedynczego bitu. Zarejestrowany przebieg umieścić w sprawozdaniu oraz udzielić odpowiedzi na następujące pytanie: *Czy czas transmisji pojedynczego bitu jest identyczny z czasem opóźnienia jaki był ustawiony w funkcji wysyłającej?*

b) przeprowadzić eksperyment polegający na sprawdzeniu w jakim zakresie można zmieniać czas nadawania pojedynczego bitu, aby odbiornik w komputerze PC był w stanie poprawnie odebrać wysyłane z Arduino dane. Wynik eksperymentu zamieścić w sprawozdaniu.

3.6 Zadanie dodatkowe - obserwacje różnych formatów ramek

Powtórzyć obserwacje realizowane w ramach zadania 3.2 przesyłając te same znaki ale wymuszając różne formaty ramki SDU np. 8E1, 8O1, 8N2 (lub inne, podane przez prowadzącego).

Konfigurację formatu ramki podaje się jako drugi parametr wywołania przeciążonej metody, inicjalizującej połączenie szeregowe (**`serial.begin(speed,config)`** – szczegółowe informacje na ten temat znajdują się w dokumentacji metody **`begin()`** klasy **Serial**[2] Arduino).

Zebrane w ten sposób dane należy udokumentować i opracować analogicznie jak dane pochodzące z obserwacji realizowanych w ramach zadania 3.2.