Design and implementation of a miniature oscilloscope module with a usb interface.

Spis treści

Wykaz skrótów ... 5
1. Wstęp ... 7
1.1. Wprowadzenie ... 7
1.2. Zestaw ćwiczeniowy ... 8
1.3. Cel pracy ... 11
1.4. Założenia projektowe ... 12
1.5. Zakres pracy ... 13
2. Zastosowany elementy sprzętowe i programowe ... 15
2.1. Standardy komunikacyjne ... 15
2.1.1. SPI ... 15
2.1.2. I2C ... 15
2.1.3. USB ... 15
2.2. Moduły mikrokontrolera ... 17
2.2.1. Kontroler DMA ... 17
2.2.2. System zdarzeń ... 17
2.3. Oprogramowanie ... 18
2.3.1. Biblioteka ASF ... 18
2.3.2. Język programowania Python ... 18
2.3.3. Pakiety języka Python ... 18
3. Dobór parametrów oraz elementów systemu ... 23
3.1. Częstotliwość próbkowania ... 23
3.2. Długość rekordu pamięci ... 23
3.3. Mikrokontroler ... 24
3.4. Zabezpieczenia napięciowe kanałów pomiarowych ... 25
3.5. Stabilizator ... 27
3.6. Metoda transmisji danych pomiarowych ... 28
3.6.1. Założenia ... 28
3.6.2. Wyniki ... 28
3.6.3. Implementacja ... 30
3.7. Metoda wyświetlania wykresów ... 31
3.7.1. Założenia ... 31
3.7.2. Implementacja ... 31
3.7.3. Wyniki ... 34
4. Architektura systemu ... 35
4.1. Elementy ... 35
4.2. Funkcje podzespołów ... 36
4.2.1. Aplikacja użytkownika ... 36
4.2.2. Moduł oscyloskopowy ... 36
4.3. Współpraca aplikacja użytkownika – moduł oscyloskopowy ... 37

5. Moduł oscyloskopowy 39
5.1. Schemat akwizycji 39
5.2. Implementacja programu mikrokontrolera 40
5.3. Część sprzętowa 44
5.3.1. Schemat 44
5.3.2. Obwód drukowany 45
6. Aplikacja użytkownika 47
6.1. Opis funkcjonalności 47
6.2. Interfejs użytkownika 47
6.3. Implementacja 48
6.3.1. Program główny aplikacji użytkownika 48
6.3.2. Moduł Gui 49
7. Testy 53
7.1. Analiza sygnału testowego 53
7.2. Analiza sygnałów interfejsu SPI oraz I2C 54
8. Podsumowanie 59
8.1. Ograniczenia 60
8.2. Możliwości rozwoju 61
8.3. Wnioski 61

Spis listingów 63
Spis rysunków 64
Spis tabel 65
Bibliografia 66
Załączniki 67
Załącznik 1. Konfiguracja częstotliwości pracy magistrali SPI i I2C 67
Załącznik 2. Implementacja testu interfejsu USB 68

Wykaz skrótów

ADC	Analog to Digital Converter
ASF	Advanced Software Framework
CS	Chip Select
DAC	Digital to Analog Converter
DMA	Direct Memory Access
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPIO	General-Purpose Input Output
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
I2CSCLH	SCL Duty Cycle Register High Half Word
I2CSCLL	SCL Duty Cycle Register Low Half Word
IDE	Integrated Design Environment
IO	Input Output
LPT	Line Print Terminal
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
PC	Personal Computer
PCB	Printed Circuit Board
PCLK	Peripheral Clock
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
SCK	Serial Clock
SDA	Serial Data
SPCCR	SPI Clock Counter Register.
SPI	Serial Peripheral Interface
TVS	Transient Voltage Suppressor
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

1. Wstęp

W dzisiejszych czasach inżynierowie stoją przed coraz większymi wyzwaniami związanymi z pomiarami podczas projektowania i debugowania najnowocześniejszych systemów wbudowanych. Złożoność konstrukcji rośnie wraz z liczbą coraz bardziej zaawansowanych układów elektronicznych. Pomimo niezwykle wręcz szybkiego rozwoju technologicznego, od ponad stulecia oscyloskop pozostaje jednym z najważniejszych i niezastąpionych urządzeń do analizy sygnałów elektrycznych. Obecnie trudno wyobrazić sobie laboratorium elektronika bez tego przyrządu pomiarowego.

1.1. Wprowadzenie

Oscyloskop jest jednym z najbardziej wszechstronnych przyrządów pomiarowych. Pozwala użytkownikowi na wizualizację mierzonego sygnału poprzez prezentowanie jego napięcia w danym okresie czasu. Początkowo oscyloskopy budowane były w oparciu o lampę oscyloskopową (oscyloskop analogowy). Obecnie dzięki rozwojowi elektroniki oscyloskopy analogowe zostały wyparte przez cyfrowe wersje.

Oscyloskopy cyfrowe, umożliwiają badanie sygnału dzięki zastosowaniu przetwornika ADC. Mierzony sygnał jest próbkowany z zadaną częstotliwością i zapisany w pamięci oscyloskopu. Po zebraniu liczby próbek możliwej do zapełnienia ekranu przebieg jest prezentowany na wyświetlaczu.

Karty oscyloskopowe dołączane do komputera PC, stanowią alternatywę dla oscyloskopowych cyfrowych. Wyposażone są one w przetworniki ADC oraz ewentualne wzmacniacze wejściowe i układy wyzwalania. Zwykle pozwalają na podłączenie do komputera poprzez złącze USB. Producent wraz z tego typu modułem dostarcza oprogramowanie umożliwiające prezentowanie przebiegów na monitorze komputera oraz pozwala na modyfikację parametrów akwizycji.

Obecnie oscyloskopy cyfrowe wciąż są najczęściej spotykanym rozwiązaniem na rynku. Charakteryzują się wyższymi parametrami dynamicznymi w porównaniu do kart oscyloskopowych. Można jednak zauważyć wzrost popularności przystawek , która wynika z ich niskiej ceny. Związana jest ona z brakiem wbudowanego w układ wyświetlacza, którego rolę spełnia ekran komputera PC. W tabeli 1.1 zestawione ze sobą wybrane stacjonarne oscyloskopy cyfrowe oraz karty oscyloskopowe.

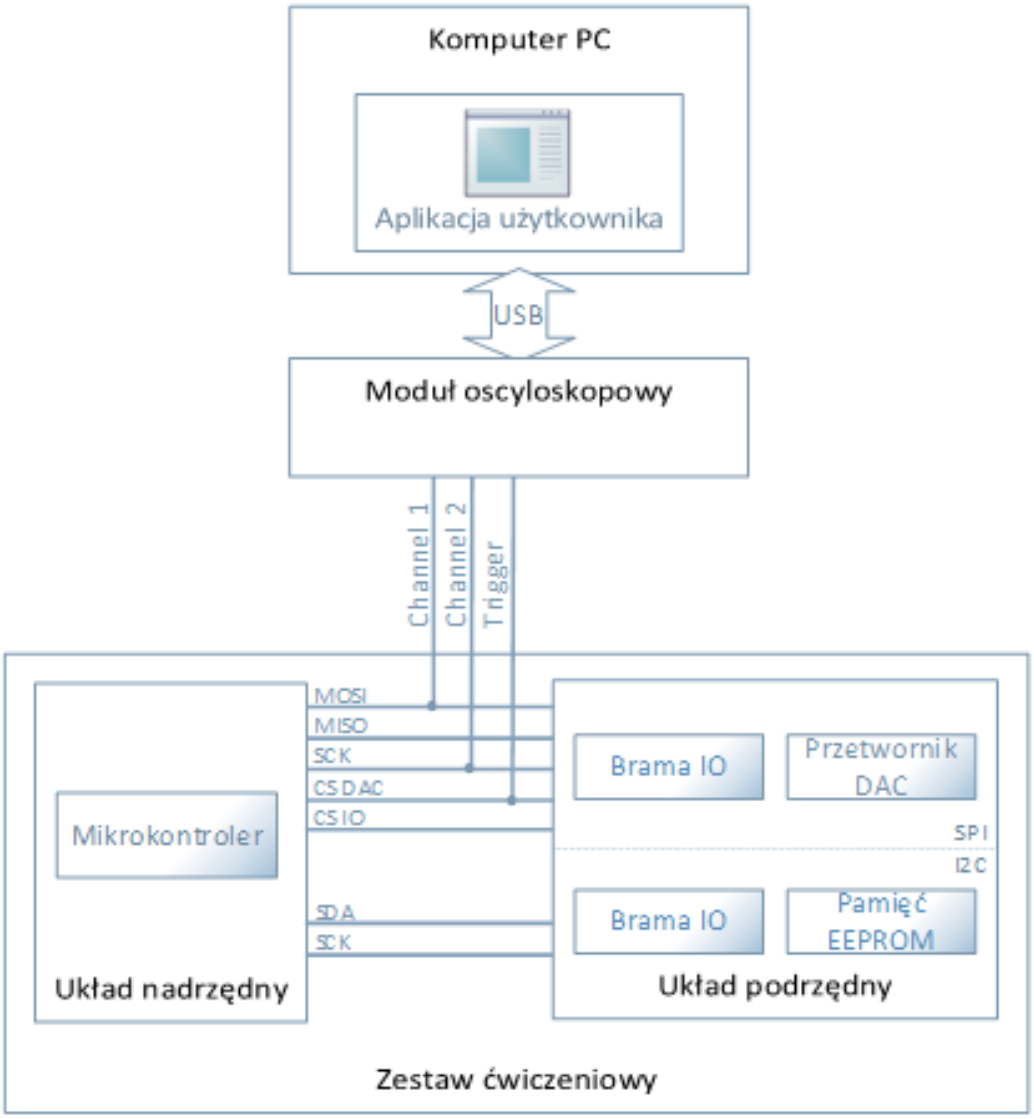
Tab. 1.1. Najważniejsze parametry wybranych oscyloskopów.

Rodzaj oscyloskopu	cyfrowy		karta	
Nazwa modelu	Rigol DS1022	Tektronix TDS3012B	Instrustar ISDS205A	Bit Scope 100
Liczba kanałów	2	2	2	2
Pasmo analogowe [MHz]	25	100	20	100
Częstotliwość próbkowania [MSa/s /kanał]	200	1250	48	40
Rekord pamięci [kB/kanał]	512	10	1000	64
Rozdzielczość [bit]	8	9	8	8
Cena brutto [USD]	400	2000	51	550

Jednym z wielu zastosowań oscyloskopów jest badanie protokołów komunikacyjnych. Na przedmiocie [redacted] prowadzone są m.in. ćwiczenia z interfejsów SPI oraz I2C. Oscyloskop jest niezbędnym przyrządem umożliwiającym praktyczne poznanie wspomnianych interfejsów oraz sprawdzenie poprawności wykonanych zadań.

1.2. Zestaw ćwiczeniowy

W trakcie zajęć, na których badane są protokoły komunikacyjne SPI oraz I2C, wykorzystywane są dwa zestawy uruchomieniowe. Pierwszy z nich pełni rolę układu nadrzędnego. Wyposażony jest on w mikrokontroler. Drugi z zestawów zawiera moduły peryferyjne, które pełnią rolę układów podrzędnych. Dla magistrali SPI układy peryferyjne to przetwornik DAC (MCP4921) oraz brama IO (MCP23S09T), natomiast dla magistrali I2C jest to brama IO (PCF8574) oraz pamięć EEPROM (MC24LC64).



Rys. 1.1. Schemat ideowy podłączenia modułu oscyloskopowego do zestawu ćwiczeniowego, na przykładzie interfejsu SPI.

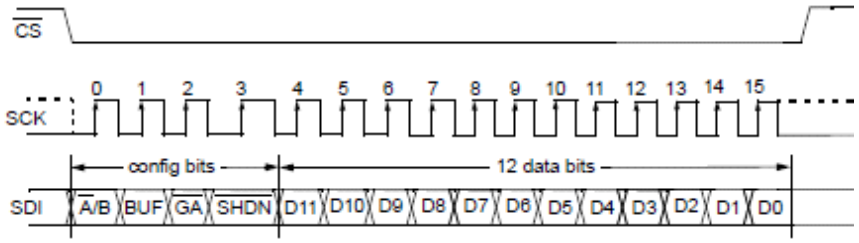
Wyznaczenie częstotliwości akwizycji oraz długości rekordu pamięci wymagać będzie znajomości minimalnej częstotliwości pracy zegara synchronizującego transmisje. Wynosi ona ok.59kHz przy domyślnym taktowaniu zegara mikrokontrolera, zarówno dla magistrali SPI jak i dla magistrali I2C.

W tabeli 1.2 przedstawiono transakcje występujące czasie komunikacji pomiędzy układem nadrzędny i podrzędnym zestawu ćwiczeniowego. Pokazano w niej została długość każdej z transakcji, którą określono na podstawie not katalogowych modułów biorących udział w komunikacji. Wynika z niej, że najdłuższa transakcja zachodzi dla interfejsu SPI i jest to ustawienie lub odczyt stanu na wyjściu portu bramy IO. Informacja ta posłuży do określenia właściwej długości rekordu danych.

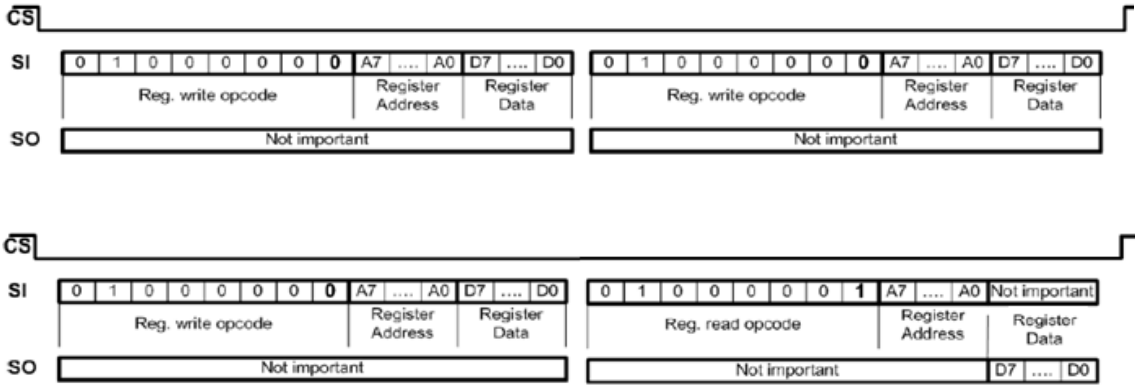
Tab. 1.2. Długości transakcji dla poszczególnych układów peryferyjnych.

Interfejs	Układ peryferyjny	Oznaczenie	Rodzaj transakcji	Liczba bitów na transakcje
SPI	przetwornik DAC	MCP4921	Ustawienie wartości napięcia wyjściowego układu	16
	brama IO	MCP23S09T	Ustawienie lub odczyt stanu na wyjściu portu	48
I2C	brama IO	PCF8574	Ustawienie lub odczyt stanu na wyjściu portu	18
	pamięć EEPROM	MC24LC64	Zapis lub odczyt bajta danych z pamięci	18

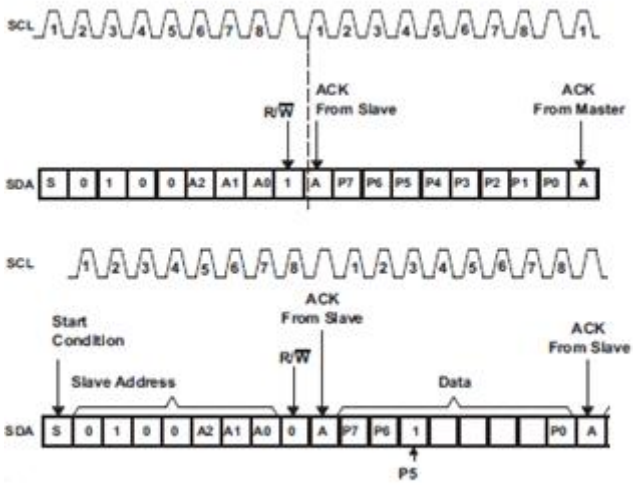
Na rysunkach 1.2, 1.3, 1.4 i 1.5 umieszczono ramki transakcji realizowanych przez układy peryferyjne znajdujące się w układzie podrzędnym zestawu ćwiczeniowego.



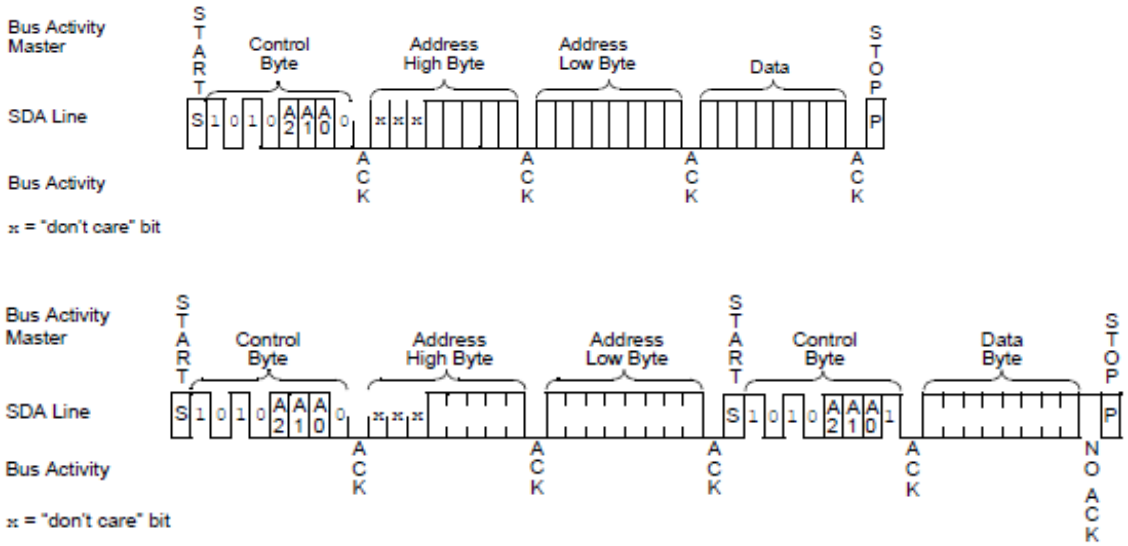
Rys. 1.2. Ustawienie wartości napięcia wyjściowego układu przetwornika DAC [18] - ramka transakcji(SPI).



Rys. 1.3. Ustawienie lub odczyt stanu na wyjściu portu bramy IO[17] - ramka transakcji(SPI).



Rys.1.4. Ustawienie lub odczyt stanu na wyjściu portu bramy IO [19] - ramka transakcji(I2C).



Rys. 1.5. Zapis lub odczyt bajta danych z pamięci EEPROM [16]- ramka transakcji(I2C).

1.3. Cel pracy

Celem niniejszej pracy jest projekt i realizacja miniaturowego modułu oscyloskopowego, który umożliwi monitorowanie linii sygnałowych interfejsu SPI oraz I2C, w zestawie ćwiczeniowym wykorzystywanym na zajęciach z przedmiotu [redacted]

1.4. Założenia projektowe

Znając główny cel określono założenia projektowe.

Parametry akwizycji:

- rozdzielczość czasowa przynajmniej 15 próbek na jeden cykl zegara (SPI/I2C),
- rozdzielczość napięciowa 8bit,
- zakres napięć wejściowych 0 - 3.3V,
- możliwość rejestracji transakcji o maksymalnej długości (rozdział 3.2),
- dwa kanały,
- wejście sygnału wyzwalającego,
- tryb wyzwalania normalny[1].

Parametry obwodu drukowanego:

- minimalne rozmiary,
- dwuwarstwowy,
- elementy na jednej stronie płytki,
- możliwość montażu ręcznego.

Parametry elektryczne:

- zasilanie ze złącza USB,
- zabezpieczenie przeciwprzepięciowe.

Aplikacja użytkownika:

- wyświetla na bieżąco zarejestrowane przebiegi,
- umożliwia przybliżanie i oddalanie wybranych fragmentów wykresu,
- pozwala na zatrzymanie/wznowienie akwizycji.

Inne:

- minimalny koszt budowy,
- transmisja danych do komputera PC z wykorzystaniem interfejsu USB,
- wyjście sygnału testowego,
- diody statusowe.

[1] Normalny tryb wyzwalania – to tryb pracy oscyloskopu, w którym wyświetlenie obserwowanego przebiegu następuje w momencie pojawienia się zdarzenia wyzwalającego. Kolejne wystąpienia zdarzenia wyzwalającego powodują odświeżenie prezentowanego przebiegu.

1.5. Zakres pracy

Osiągnięcie założonego celu projektowego oraz realizacja założeń wiązała się z koniecznością rozwiązania poniższych problemów:

- analiza wymagań projektowych,
- dobór modułu mikrokontrolera,
- zaprojektowania obwodu drukowanego oscyloskopu,
- montaż i uruchomienie oscyloskopu,
- opracowanie oprogramowania sterującego mikrokontrolerem,
- opracowanie oprogramowania aplikacji użytkownika,
- przeprowadzenie testów oscyloskopu.

2. Zastosowany elementy sprzętowe i programowe

2.1. Standardy komunikacyjne

2.1.1. SPI

Magistrala SPI jest szeregowym synchronicznym interfejsem typu full-duplex, pracującym w konfiguracji master-slave. Prędkość transmisji uwarunkowana jest możliwościami sprzętu. Magistrala SPI składa się z czterech linii sygnałowych:

- MOSI - linia danych przesyłanych z układu nadrzędnego do podrzędnego,
- MISO - linia danych przesyłanych z układu podrzędnego do nadrzędnego,
- SCK - sygnał zegarowy, generowany przez układ nadrzędny,
- CS - sygnał informujący układ podrzędny o rozpoczęciu transmisji przez układ nadrzędny.

2.1.2. I2C

Magistrala I2C jest szeregowym synchronicznym interfejsem typu half-duplex, pracującym w konfiguracji master-slave. Maksymalna prędkość wynosi w trybie standardowym 100kb/s, w trybie szybkim 400kb/s, a w trybie high speed 3.4 Mb/s. Magistrala I2C składa się z dwóch linii sygnałowych:

- SDA - linia danych,
- SCK - sygnał zegarowy, generowany przez układ nadrzędny.

2.1.3. USB

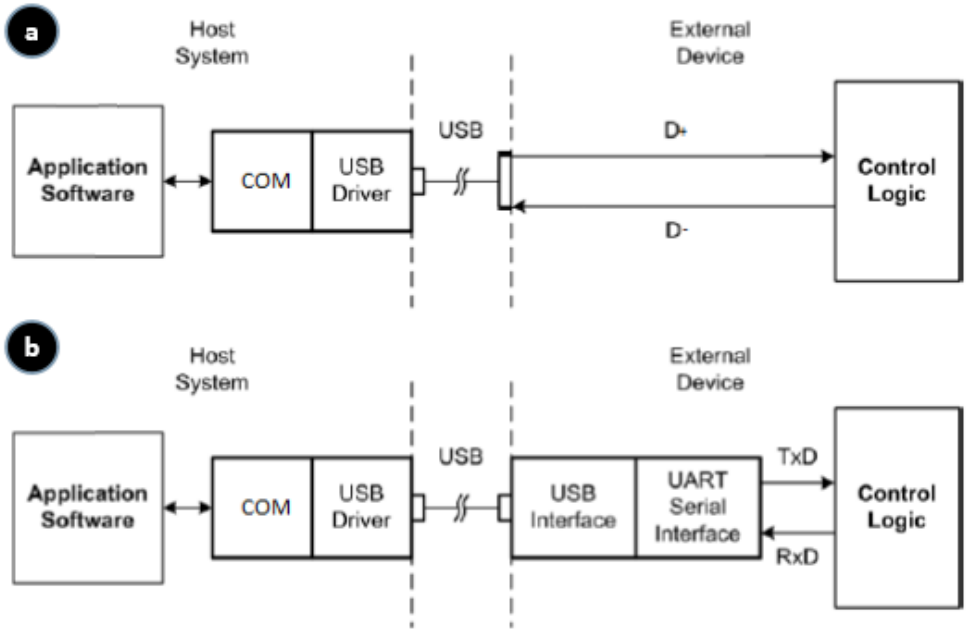
Interfejs USB jest szeregowym synchronicznym interfejsem typu half-duplex, pracującym w konfiguracji master-slave. Obecnie standard USB udostępnia cztery wersje różniące się prędkościami transmisji danych.

Tab. 2.1. Wersje interfejsu USB.

Wersja	Nazwa	Prędkość transmisji (Mbit/s)
1.0	Low speed	1,5
1.1	Full speed	12
2.0	High speed	480
3.0	Super speed	4000

Funkcje medium komunikacyjnego pełni 4-żyłowy kabel. Dane przesyłane są różnicowo liniami D+ oraz D-, natomiast dwa pozostałe sygnały to linie zasilania +5V oraz GND.

Standard USB definiuje różne klasy urządzeń korzystających z interfejsów USB, których użycie pozwala na ominięcie pisania własnych sterowników. Najprostszym sposobem użycia interfejsu USB jest wykorzystanie klasy CDC. Urządzenia korzystające z klasy CDC są widoczne w systemie operacyjnym jako standardowe porty szeregowe (wirtualne porty COM).



Rys. 2.1. Schemat implementacji interfejsu USB przy wykorzystaniu klasy CDC a) sprzętowa interfejs USB lub programowa emulacja b)interfejs UART wraz z konwerterem USB-UART.

Implementując interfejs USB do komunikacji pomiędzy mikrokontrolerem, a komputerem PC do wyboru są trzy możliwe warianty:

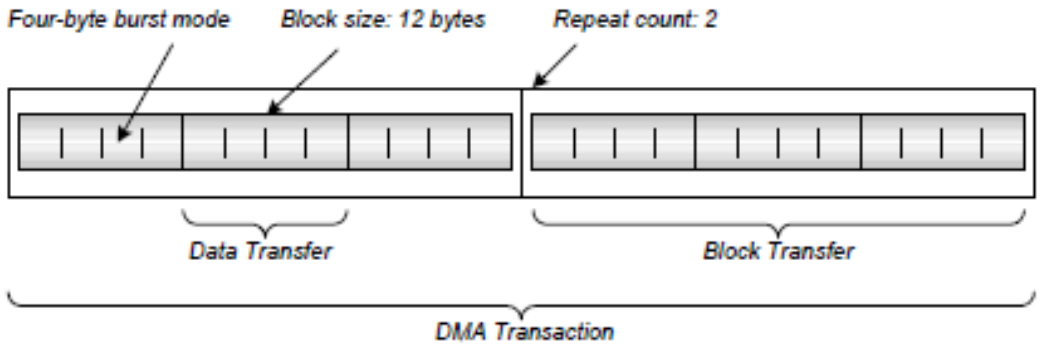
1. Sprzętowy interfejs USB (Rys. 2.1.a):
   - w pełni zgodny ze standardem USB,
   - maksymalna prędkość do12 Mb/s,
   - wymaga modelu mikrokontrolera z tym modułem.
2. Programowa emulacja USB (Rys. 2.1.a):
   - niecałkowicie zgodna ze standardem USB,
   - obciąża procesor ,
   - tanie rozwiązanie.
3. Interfejs UART oraz konwerter USB-UART (Rys 2.1.b):
   - dodatkowy koszt konwertera,
   - konwerter zajmuje dodatkowe miejsce na PCB,
   - maksymalna prędkość równa ok. 3Mb/s.

2.2. Moduły mikrokontrolera

2.2.1. Kontroler DMA

Kontroler DMA to układ elektroniczny, pozwalający na transfer danych pomiędzy dwoma układami peryferyjnymi mikrokontrolera (np. ADC i pamięć RAM) z pominięciem procesora.

Kompletna operacja zrealizowana przez kontroler DMA nosi nazwę transakcji. Każdą transakcję można podzielić na bloki. Natomiast bloki dzielą się na mniejsze niepodzielne fragmenty, które muszą zostać przesłane w sposób nieprzerwany (tzw. burst transfer).



Rys. 2.2. Schemat prezentujący relacje pomiędzy transakcją, blokiem i transferem burst.

2.2.2. System zdarzeń

System zdarzeń to układ elektroniczny umożliwiający połączenie dwóch lub większej ilości układów peryferyjnych, z których jeden jest generatorem, a pozostałe odbiorcą określonego zdarzenia. Dzięki tej funkcjonalności, system zdarzeń pozwala na pominięcie procesora w czasie komunikacji pomiędzy dwoma podsystemami. Większość układów posiada predefiniowaną listę zdarzeń, które mogą być generatorem sygnału dla innych układów.

2.3. Oprogramowanie

2.3.1. Biblioteka ASF

Biblioteki ASF stanowią wysoko-poziomowy interfejs do części sprzętowej mikrokontrolerów dostarczanych przez firmę Atmel. Wykorzystanie tychże bibliotek, przyspiesza proces konfiguracji warstwy sprzętowej procesora, pozwalając tym samym korzystać z układów peryferyjnych w bardzo prosty i przejrzysty sposób. Dzięki temu można dużo szybciej przejść do rozwijania właściwej części realizowanego projektu. Same biblioteki ASF mają bardzo obszernie przygotowaną dokumentację, z dużą ilością przykładów. Pakiet ten jest również zintegrowany z IDE Atmel Studio, które zostało wykorzystane do tworzenia oprogramowania mikrokontrolera.

2.3.2. Język programowania Python

Python jest językiem programowania wysokiego poziomu ogólnego przeznaczenia. Oferuje bogaty zestaw bibliotek standardowych oraz różnorodne struktury danych. Składnia języka zapewnia przejrzystość, zwięzłość i czytelność kodu źródłowego.

Python oprócz silnego trzonu z rozbudowanym zestawem funkcji wbudowanych, pozwala nam rozszerzać język o moduły zewnętrzne, które oferują nowe funkcjonalności. Odzwierciedlają one filozofię systemu Unix: „*Do one thing, do it well.*”[15]. Moduły zewnętrzne są często organizowane w pakiety. Pakiet jest to uporządkowany zbiór modułów, które mają ten sam cel.

2.3.3. Pakiety języka Python

2.3.3.1. IPython

IPython jest rozszerzonym interpreterem języka Python. Został uruchomiony w 2001 roku przez Fernando Peraza, który swój projekt określa jako: *"Tools for the entire life cycle of research computing"*[22].

IPython pozwala na efektywne używanie Pythona do interaktywnych obliczeń naukowych i przetwarzania danych. Z tego powodu zyskał dużą popularność wśród społeczności naukowych i inżynieryjnych oraz zaczął obsługiwać więcej języków programowania (Julia, R). Moduł ten umożliwia w czytelny sposób prezentowanie kodu źródłowego, edytowanie, debugowanie, indeksowanie oraz udostępnia szereg predefiniowanych funkcji nazywanych „magic functions”.

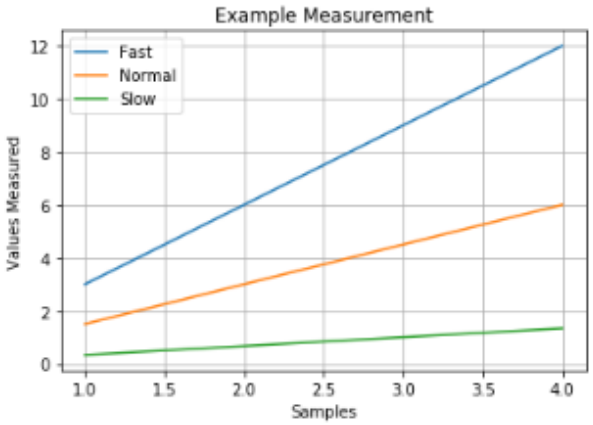
2.3.3.2. Matplotlib

Matplotlib to wysoce konfigurowalny, elastyczny i łatwy w użyciu pakiet Pythona służący do wizualizacji 2D. Pakiet ten udostępnia użytkownikowi szeroką listę możliwych do wygenerowania wykresów np. wykresy 2D i 3D, wykresy kołowe, histogramy i wiele innych. Matplotlib obsługuje interaktywny i nieinteraktywny tryb pracy, a także pozwala na korzystanie z wielu narzędzi umożliwiających budowanie GUI.

W napisaniu aplikacji użytkownika, użyto modułu pyplot z pakietu Matplotlib, który udostępnia zbiór funkcji do wizualizacji danych na wzór API oprogramowania Matlab. Poniżej przykład użyciu tego modułu w interaktywnej konsoli IPyhon.

Listing 2.1. Przykład wykorzystania metod biblioteki Matpltlib.

In [1]:
```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  %matplotlib inline
4
5  x = np.arange(1, 5)
6  plt.plot(x, x*3.0, label='Fast')
7  plt.plot(x, x*1.5, label='Normal')
8  plt.plot(x, x/3.0, label='Slow')
9
10 plt.grid(True)
11 plt.title('Example Measurement')
12 plt.xlabel('Samples')
13 plt.ylabel('Values Measured')
14 plt.legend(loc='upper left')
15 plt.show()
```



Jak widać na listingu 2.1, moduł pyplot jest prosty i intuicyjny w użyciu. Architektura pakietu Matplotlib obraca się wokół operacji, które umożliwiają użytkownikowi tworzenie, renderowania i aktualizowania obiektów opisanych w klasie *Figure*. Strukturę pakietu można podzielić na trzy warstwy:

- backend layer, umożliwia wyświetlenie obiektu klasy *Figure* w oknie użytkownika, renderowanie, obsługę zdarzeń pochodzących z wejścia klawiatury lub myszy
- artist layer, udostępnia zestaw obiektów możliwych do umieszczenia w przestrzeni obiektu *Figure* np.: wykres 2D, histogram, tekst, legenda
- scripting layer, udostępnia API, czyli interfejs programisty, np. pyplot

2.3.3.3. NumPy

NumPy jest biblioteką dostarczającą bogaty zestaw struktur danych i funkcji matematycznych, która ułatwia prace z dużymi zbiorami danych. W przeciwieństwie do domyślnych struktur oferowanych przez język Python, typy danych zapewniane przez NumPy obsługują operacje na wektorach, które charakteryzują się wydajnością i skracają czas wykonywania instrukcji. Listing 2.2. prezentuje wynik porównania czasu wykonywania się instrukcji sumowania miliona elementów o losowych wartościach, wykorzystując funkcję ze standardowej biblioteki oraz funkcje z biblioteki NumPy. Funkcja z pakietu NumPy wykonuję się w czasie ponad sto trzydzieści razy krótszym.

Listing 2.2. Przykład prezentujący zalety biblioteki NumPy w stosunku do funkcji standardowych.

```
In [1]: 1 import numpy as np
        2
        3 array = np.random.rand(1000000)
        4
        5 %timeit sum(array)
        6 %timeit np.sum(array)

267 ms ± 1.59 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
1.95 ms ± 14.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops e
ach)
```

Biblioteka NumPy została wykorzystana w implementacji modułu wykorzystanego w trakcie testów interfejsu USB, w tabeli 2.2. znajdują się użyte metody tego modułu.

Tab. 2.2. Wykorzystane metody biblioteki NumPy.

Metoda	Opis
subtract ( )	Odejmuje elementy drugiej tablicy od pierwszej tablicy
multiply ( )	Mnoży każdy element tablicy przez stałą
equal ( )	Porównanie odpowiadających wartości dwóch tablic
insert ( )	Wstawienie elementu do tablicy we wskazane miejsce
around ( )	Zaokrąglenie wartości do wskazanej części
min ( ), max ( )	Minimum i maximum
argmin ( ),arg max ( )	Indeks minimalnej i maksymalnej wartości tablicy
std ( )	Odchylenie standardowe
mean ( )	Średnia arytmetyczna
median ( )	Mediana

2.3.3.4. PySerial

PySerial to moduł dostarczający niskopoziomowe funkcje dostępu do portu szeregowego komputera PC. Biblioteka pySerial udostępnia wiele parametrów ułatwiających konfigurację i obsługę portu szeregowe m.in. pozwala na skonfigurowanie timeoutów oraz oferuje podstawową obsługę wyjątków.

Tab. 2.3. Wykorzystane metody biblioteki PySerial.

Metoda	Opis
Serial ( )	Tworzy obiekt klasy i otwiera komunikacje z portem
close ( )	Zamknięcie komunikacji z portem
read ( )	Odczytanie określonej liczby bajtów z bufora wejścia
write ( )	Zapisanie określone liczby bajtów do bufora wyjścia
flushInput ( )	Wyczyszczenie bufora wejściowego portu

2.3.3.5. time

Moduł time pozwala korzystać z funkcji, które udostępniają czas systemowy. Pomimo tego, że moduł ten zawiera się w pakiecie bibliotek standardowych to jej niektóre funkcje nie są dostępne na wszystkich platformach. Większość funkcji zdefiniowanych w tym module wywołuje funkcje z biblioteki języka C o tej samej nazwie.

Tab. 2.4. Wykorzystane metody biblioteki time.

Metoda	Opis
clock ( )	Zwraca obecny dzień, minutę, sekundę i mikrosekundę

3. Dobór parametrów oraz elementów systemu

3.1. Częstotliwość próbkowania

Według twierdzenia Kostielnikowa–Shannona, aby zrekonstruować sygnał, częstotliwość próbkowania powinna być dwukrotnie większa od najwyższej częstotliwości składowej rejestrowanego przebiegu. W rzeczywistości wymagana jest większa liczba próbek, aby można było uzyskać użyteczną informację na temat badanego przebiegu.

Na podstawie założonej liczby próbek na cykl zegara monitorowanego interfejsu(15) oraz minimalnej częstotliwości wspomnianego zegara (59kHz) obliczono minimalną wymagana częstotliwość próbkowania modułu oscyloskopowego. Wynosi ona 885kSa/s.

3.2. Długość rekordu pamięci

W celu oszacowania wymaganej długości rekordu pamięci, należy określić czas akwizycji, który umożliwi rejestrację najdłuższej transakcji. Częstotliwość sygnału zegarowego synchronizującego transmisje danych jest taki sam dla magistrali SPI oraz I2C. Z tego względu najdłuższa transakcja będzie miała miejsce w przypadku transmisji największej liczby bitów.

Z tabeli 1.2 wynika, że najdłuższa transakcją składa się z 48 bitów. Transmisja jednego bitu zajmuje ok. 17us (sygnał zegarowy 59kHz). Z tego względu długość transmisji najdłuższej ramki danych wyniesie 816us. Jako, że najdłuższa transakcja trwa 816us przyjmujemy, że czas pojedynczej akwizycji ma wynosić 1000us.

Następnie biorąc pod uwagę, że przyjęliśmy prędkość próbkowania na poziomie 885kSa/s (czyli co ok. 1,13us), to aby zarejestrować sygnał o czasie trwania 1000us, bufor pamięci musi wynosić ok. 885 próbek. Mając na względzie, że chcemy jednocześnie obserwować dwie linie sygnałowe to musimy go podwoić, czyli wyniesie 1770 próbek.

Zważywszy na zdefiniowane założenie, określające rozdzielczość jednej próbki (8 bit), to rzeczywista długość rekordu pamięci oscyloskopu powinna wynosić 1770B.

3.3. Mikrokontroler

Mikrokontrolery XMEGA zyskały w ostatnim czasie dużą popularność. Niewątpliwymi zaletami tych modułów jest niska cena oraz możliwość ich łatwego pozyskania w jednostkowych ilościach. Dużą zaletą jest również darmowe i proste środowisko programistyczne Atmel Studio, a także bogata dokumentacja z licznymi przykładami.

Mikrokontrolery XMEGA podzielone są na rodziny, które różnią się między sobą występującymi układami peryferyjnymi oraz ich parametrami. Układy wchodzące w skład jednej rodziny odróżnia ilość dostępnej pamięci oraz rodzaj obudowy. Rozkład sygnałów na wyprowadzeniach obudowy procesora oraz wykorzystane bloki funkcjonalne, są takie same w obrębie jednej rodziny. W celu dokonania odpowiedniego wyboru przeprowadzono analizę dostępnych mikrokontrolerów. Wyboru mikrokontrolera dokonano biorąc pod uwagę parametry zebrane w tabeli 3.1.

Tab. 3.1. Główne rodziny mikrokontrolerów XMEGA.

Rodzina	ADC maks. próbkowanie [ksps]	SRAM (KB)	Wersja USB	Obudowa	Liczba pinów
A1U	2000	4/8	2.0	TQFP, BGA, VBGA	100
A3U	2000	4/8/16	2.0	TQFP, QFN	64
A4U	2000	2/4/8	2.0	QFP, QFN, VFBGA	44
B1	200	4/8	2.0	TQFP, VFBGA	100
B3	300	4/8	2.0	TQFP, QFN, DRQFN	64
C3	300	2/4/8/16	1.1	TQFP, VQFN	64
C4	300	2/4	1.1	TQFP, QFN, VFBGA	44
D3	200	8/16/32	brak	TQFP, QFN	64
D4	200	1/2/4/8	brak	TQFP, VFBGA	44
E5	300	1/2/4	brak	QFP, QFN, UQFN	32

Dobór odpowiedniego mikrokontrolera rozpoczęto od analizy przetwornika ADC każdej z rodzin. Podsystem ten ma decydujące znaczenie w zrealizowaniu założeń projektowych. Z tabeli 3.1 wynika, że jedynie mikrokontrolery XMEGA z rodzin "A" są wstanie spełnić dobraną częstotliwość próbkowania (rozdział 3.1), czyli uzyskanie 885kSa/s z dwóch kanałów. Wyklucza to możliwość wykorzystania innych rodzin XMEGA w projekcie modułu oscyloskopowego. Maksymalne próbkowanie ADC w tej rodzinie mikrokontrolerów wynosi 2000 kSa/s, co przekłada się na 1000 kSa/s na kanał. Ponieważ zdecydowano, że będzie to nowa docelowa prędkość próbkowania projektowanego modułu oscyloskopowego. Zmiana dobranej częstotliwości próbkowania spowoduje także zmianę wymaganej pamięci RAM.

W kolejnym kroku skupiono się na rozmiarze pamięci RAM, która ma bezpośrednie przełożenie na możliwą liczbę zebranych próbek podczas pojedynczego

pomiaru. Po ponownym przeliczeniu, wymagana długość rekordu pamięci powinna wynosić 2kB.Z tego względu to, zakładamy, że 4kB pamięci podręcznej będzie wystarczające. Każda z rodzin typu "A" dysponuje założonym rozmiarem RAMu.

Czynnikiem, który nie może zostać pominięty, jest wyposażenie mikrokontrolera w sprzętowy interfejs USB. Umożliwi to bezpośrednie podłączenie pinów mikrokontrolera do odpowiednich styków gniazda USB. Jest to najlepsze rozwiązanie możliwe do zrealizowania z wykorzystaniem mikrokontrolera (rozdział 2.1.3.). Jak wynika z tabeli 3.1. każda z rodzin "A" jest wyposażona w tą funkcjonalność.

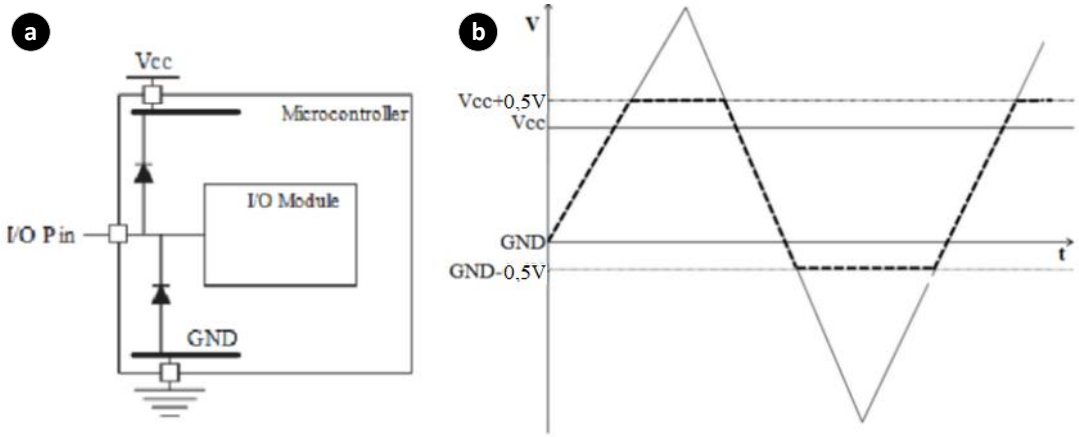
Istotny wpływ na dobór odpowiedniego układu ma również typ obudowy mikrokontrolera oraz jej rozmiar. Ze względu na wymagany montaż ręczny, moduł musi być dostępny w obudowie TQFP lub QFP. Każdy z modułów XMEGA z rodziny "A" spełnia ten warunek. Jednak biorąc pod uwagą liczbę pinów, najmniejsze rozmiary posiada układ z rodziny XMEGA A4U.

W wyniku powyższej analizy zdecydowano się na mikrokontroler XMEGA z rodziny A4U, a konkretnie moduł ATxmega32A4U-AU, gdyż jest on wyposażony w pamięć RAM o rozmiarze 4KB.

3.4. Zabezpieczenia napięciowe kanałów pomiarowych

Każdy pin IO, układu ATxmega32A4U-AU, chroniony jest za pomocą diod wbudowanych w strukturę mikrokontrolera (Rys 3.1.a). Zadaniem diod jest ochrona układu scalonego przed trwałym uszkodzeniem w wyniku pojawienia się na jednym z pinów, napięcia z poza maksymalnego dozwolonego zakresu.

W przypadku dobranego mikrokontrolera, maksymalna dozwolona wartość napięcia wynosi ±0,5V [21] zakresu pracy (0-3.3V). Na Rys 3.1.b zaprezentowano wpływ diod na napięcie panujące na pinio IO. Linia ciągła obrazuje napięcie na rezystorze znajdującym się przed diodami, natomiast linia przerywana pokazuje napięcia na pinie mikrokontrolera.



Rys. 3.1. Diody zabezpieczające porty IO mikrokontrolera: a)budowa układu[3], b)wpływ diod zabezpieczających procesor na napięcie panujące na pinie IO.

Sytuacjami przed którymi mają chronić wbudowane w mikrokontroler diody, są wyładowania elektrostatyczne (ESD). Zdarzenia te, to krótkie impulsy prądowe pojawiające się w przestrzeni pomiędzy obiektami o odpowiednio dużej różnicy potencjałów elektrostatycznych**.** W praktyce większość sytuacji dotyczy wyładowań z naelektryzowanego ciała człowieka przy kontakcie z innym obiektem.

W celu ochrony mikrokontrolera przed ryzykiem wystąpienia ciągłego napięcia z poza zakresu pracy, postawiono wyposażyć oscyloskop w dodatkowy układ zabezpieczający. Sytuacja taka może mieć miejsce w przypadku próby podłączenia portów oscyloskopu do układu pracującego w wyższym standardzie napięciowym.

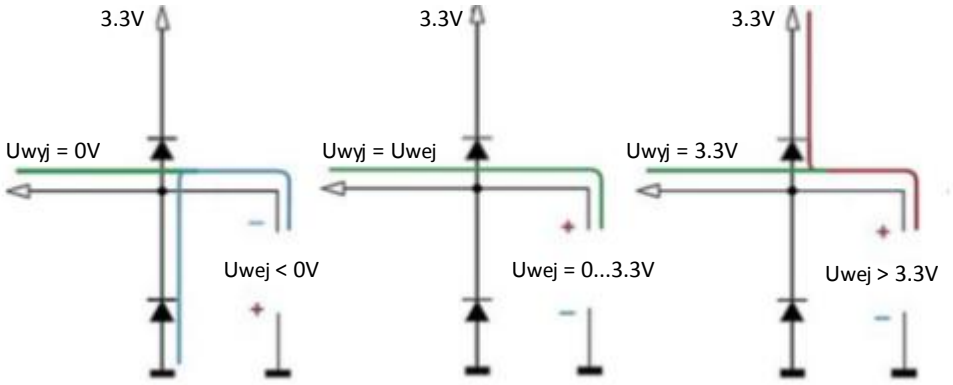
W celu realizacji tego zadania, do każdego z portu wejściowego i wyjściowego projektowanego modułu postanowiono dołączyć diody transil (TVS). Wybrano moduł SRDA 3.3-4 TBT pozwalający na podłączenie równoległe do 4 linii.

Tab. 3.2. Charakterystyka elektryczna układu SRDA 3.3-4 TBT [20].

Parameter	Symbol	Conditions	Minimum	Typical	Maximum	Units
Reverse Stand-Off Voltage	V_{RWM}				3.3	V
Punch-Through Voltage	V_{PT}	I_{PT} = 2µA	3.5			V
Snap-Back Voltage	V_{SB}	I_{SB} = 50mA	2.8			V
Reverse Leakage Current	I_R	V_{RWM} = 3.3V, T=25°C			1	µA
Clamping Voltage	V_C	I_{PP} = 1A, t_p = 8/20µs			5.3	V
Clamping Voltage	V_C	I_{PP} = 10A, t_p = 8/20µs			10	V
Clamping Voltage	V_C	I_{PP} = 25A, t_p = 8/20µs			15	V
Junction Capacitance	C_j	Between I/O pins and Ground V_R = 0V, f = 1MHz		8	15	pF
		Between I/O pins V_R = 0V, f = 1MHz		4		pF

Tab 3.2 prezentuje parametry elektryczne dobranego modułu. Jak widać napięcie przebicia dla diod z układu SRDA 3.3-4 TBT wynosi 3,5V. Z drugiej strony diody wbudowane w mikrokontroler zaczynają przewodzić przy napięciu około 3,8V. Wynika z tego, że dobrany układ zabezpieczający powinien odprowadzić nadmiar ładunków elektrycznych, zanim pojawią się one na wejściu mikrokontrolera. Dodatkowo z Tab3.2. wynika, że układu SRDA 3.3-4 TBT wprowadza niewielka ilość dodatkowej pojemności, a więc nie powinien mieć negatywnego wpływu na obserwowane sygnały.

Zasadę działania układu zabezpieczającego przedstawiono na rysunku 3.2.



Rys. 3.2. Trzy warianty funkcjonowania układu zabezpieczającego wejścia mikrokontroler. Ścieżka zielona to napięcie bezpieczne.

3.5. Stabilizator

Biorąc pod uwagę fakt, że moduł oscyloskopowy będzie zasilany ze złącza USB, czyli napięciem 5V, a mikrokontrolery XMEGA pracują w zakresie napięć 0-3.3V, wymagany jest dodatkowy układ obniżający napięcie zasilania.

Z tego powodu zdecydowano się na układ LM1117 w wersji 3.3V. Jest to regulator liniowy typu low-dropout. W celu poprawnej pracy układu i stabilności napięcia wyjściowego, wymagana jest obecność dwóch dodatkowych kondensatorów.

3.6. Metoda transmisji danych pomiarowych

W trakcie projektowania modułu oscyloskopowego sprawdzono dwie metody transmisji danych pomiarowych, które pozwoliły na dobór odpowiedniego interfejsu komunikacyjnego, tj. :

- wykorzystanie wbudowanego interfejsu USB,
- wykorzystanie interfejsu UART oraz modułu konwertera USB-UART.

Na przeprowadzenie testu przed wykonaniem obwodu drukowanego, pozwolił zestaw ćwiczeniowy wykorzystywany na przedmiocie „Podstawy projektowania systemów wbudowanych”. Zestaw ten wyposażony jest mikrokontroler ATxmega32A4U-AU i pozwala na komunikację z komputerem PC za pomocą interfejsu USB oraz UART przy wykorzystaniu konwertera USB-UART (FT231).

3.6.1. Założenia

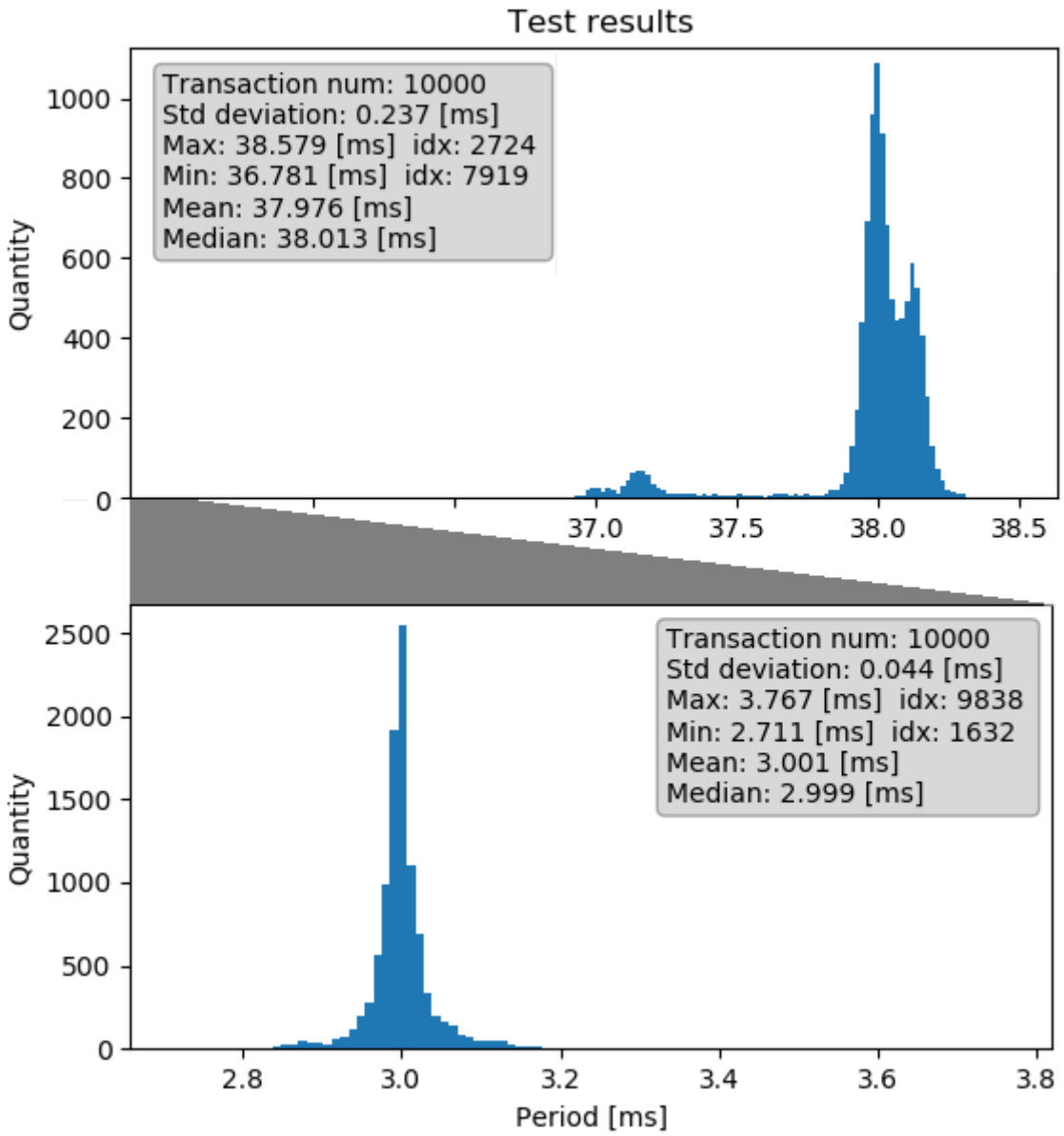
Test przeprowadzono w oparciu o następujące założenia:

- komunikacja z maksymalną możliwą szybkością,
- transakcje inicjuje program testowy poprzez wysłanie znaku startowego,
- wiadomość zwrotna w postaci bufora o rozmiarze 2kB,
- bufor danych wypełniony wartościami licznika modulo 256 ,
- weryfikacji odebranych danych, na podstawie poprawności sumy wszystkich elementów bufora,
- liczba wymaganych transakcji do zakończenia testu 10000 ,
- rejestracja parametrów transmisji i prezentacja wyników.

W przypadku protokołu USB, interfejs pracuje z maksymalną możliwą prędkością (patrz rozdział 2.1.3.). Natomiast dla interfejsu UART prędkość transmisji przyjmujemy 921600 bit\s. Pozostałe parametry transmisji są takie same dla obu protokołów.

3.6.2. Wyniki

W rezultacie przeprowadzonych testów otrzymano wykresy (Rys. 3.3) prezentujące parametry transmisji dla interfejsu USB oraz UART.



Rys. 3.3. Wyniki testu: górny wykres interfejs UART, dolny wykres interfejs USB.

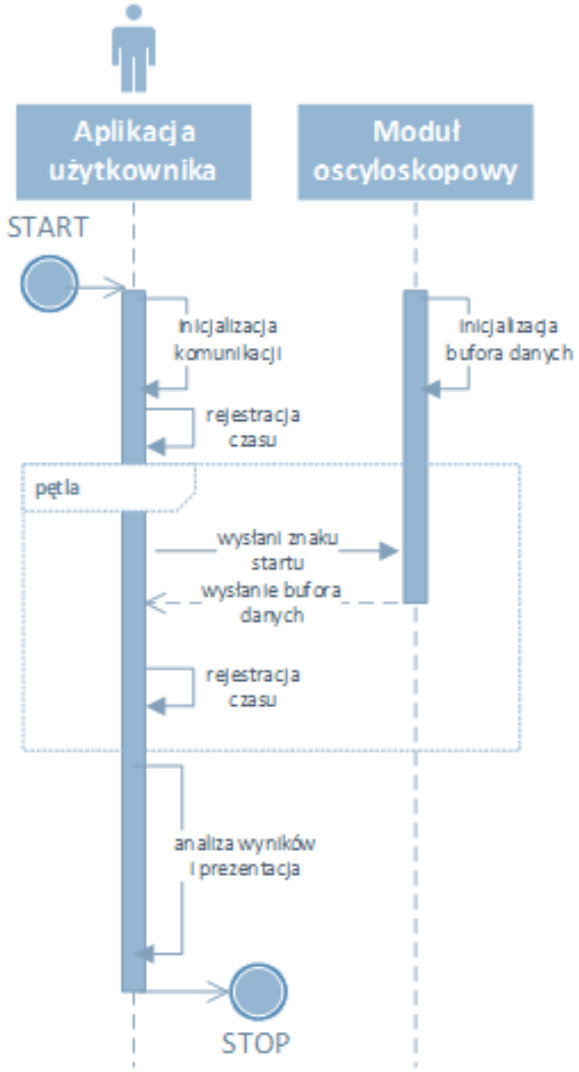
Otrzymane wyniki pozwoliły na wyciągnięcie następujących wniosków:

- wszystkie transakcję zarówno w przypadku interfejsu UART jak i USB dostarczyły poprawnych danych,
- biorąc pod uwagę średni czas transakcji, komunikacja z wykorzystaniem USB (3,001 ms) zaszła ponad 12 razy szybciej niż w przypadku UARTa (37,976 ms),
- mając na uwadze odchylenie standardowe jak również otrzymane wykresy, można stwierdzić, że komunikacja z wykorzystaniem USB przebiega w sposób stabilniejszy.

3.6.3. Implementacja

W celu przeprowadzenia testu, został napisany skrypt testowy. Skrypt został napisany w języku Python i pozwala na komunikację komputera PC z mikrokontrolerem. Zaimplementowany został także moduł test, umożliwiający weryfikację poprawności komunikacji oraz prezentację parametrów transmisji.

Na poniższym rysunku (Rys.3.4.) przedstawiono diagram sekwencji obrazujący przebieg testu interfejsu USB .



Rys. 3.4. Diagram sekwencji testu interfejsu USB.

3.7. Metoda wyświetlania wykresów

Celem testu był dobór odpowiedniego sposobu wyświetlania wykresu oraz wybór warstwy zaplecza do pakietu Matplotlib, która pozwoliłaby na maksymalną prędkość odświeżania prezentowanego wykresu.

3.7.1. Założenia

Test przeprowadzono w oparciu o następujące założenia:

- maksymalna prędkość odświeżania,
- wykres prezentowany w oknie o docelowej wielkości 12 x 6cali,
- prezentacja przebiegu składającego się ze 100 losowych próbek,
- czas przeprowadzanego testu 100 sekund,
- otrzymany wynik w średniej liczbie klatek na sekundę .

3.7.2. Implementacja

W podrozdziale tym zaprezentowano trzy sposoby wyświetlania wykresu przy użyciu domyślnej warstwy zaplecza (tkinter) oraz jedną metodę, która korzysta z warstwy zaplecza Qt. Przedstawione sposoby wyświetlania:

1. *Clear and plot* – wykorzystuje metody *clear()* oraz *plot(),* kolejno do usunięcia prezentowanego okna wykresu, a następnie zbudowania go na nowo.
2. *Edit line and update* – wykorzystuje *set_ydata()* oraz *draw()* do zaktualizowania prezentowanych danych i odświeżenia wykresu .
3. *Edit line and selective update* – wykorzystuje metody bezpośrednio z warstwy zaplecza umożliwiając zapamiętanie ostatnio zbudowanego okna, a następnie odświeżenie jedyni zmienionego elementu, czyli linii wykresu.
4. *Edit line and selective update (Qt Frame)* – ten sam sposób budowania wykresu jak w punkcie 3, ale przy użyciu warstwy *Qt.*

Listing 3.1. Implementacja testu pakietu Matplotlib, część 1.

Init package

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import time
```

```
In [2]: %matplotlib tk
```

1. Clear and plot

```
In [3]: fig, ax = plt.subplots(figsize=(12,6))

        num_plots  = 0
        time_start = time.time()
        while time.time()-time_start < 100:
            ax.clear()
            ax.plot(np.random.randn(100))
            plt.pause(1e-10)
            num_plots += 1

        print('Number of frames per second: ' + str(num_plots/100))
```

2. Edit line and update

```
In [4]: fig, ax = plt.subplots(figsize=(12,6))
        line, = ax.plot(np.random.randn(100))

        num_plots  = 0
        time_start = time.time()
        while time.time()-time_start < 100:
            line.set_ydata(np.random.randn(100))
            plt.draw()
            plt.pause(1e-10)
            num_plots += 1

        print('Number of frames per second: ' + str(num_plots/100))
```

Listing 3.2. Implementacja testu pakietu Matplotlib, część 2.

3. Edit line and selective update

```
In [5]: fig, ax = plt.subplots(figsize=(12,6))
        line, = ax.plot(np.random.randn(100))
        fig.canvas.draw()

        num_plots  = 0
        time_start = time.time()
        while time.time()-time_start < 100:
            line.set_ydata(np.random.randn(100))
            fig.canvas.restore_region(
                fig.canvas.copy_from_bbox(ax.bbox))
            ax.draw_artist(ax.patch)
            ax.draw_artist(line)
            fig.canvas.draw_idle()
            fig.canvas.flush_events()
            plt.pause(1e-10)
            num_plots += 1
        print('Number of frames per second: ' + str(num_plots/100))
```

4. Edit line and selective update (Qt Frame)

```
In [1]: import matplotlib
        matplotlib.use('Qt5Agg')
```

```
In [2]: import matplotlib.pyplot as plt
        import numpy as np
        import time
```

```
In [3]: fig, ax = plt.subplots(figsize=(12,6))
        line, = ax.plot(np.random.randn(100))
        fig.canvas.draw()
        plt.pause(1e-10)

        num_plots  = 0
        time_start = time.time()
        while time.time()-time_start < 100:
            line.set_ydata(np.random.randn(100))
            fig.canvas.restore_region(
                fig.canvas.copy_from_bbox(ax.bbox))
            ax.draw_artist(ax.patch)
            ax.draw_artist(line)
            fig.canvas.update()
            fig.canvas.flush_events()
            plt.pause(1e-10)
            num_plots += 1
        print('Number of frames per second: ' + str(num_plots/100))
```

3.7.3. Wyniki

Tabela 3.3 prezentuje otrzymane wyniki testów. Wynika z niej, że najlepszym rozwiązaniem jest zastosowanie metody polegającej na odświeżaniu jedynie zmienionej linii przy zastosowaniu warstwy zaplecza *Qt* .

Tab. 3.3. Porównanie testowanych metod umożliwiających wyświetlanie animacji.

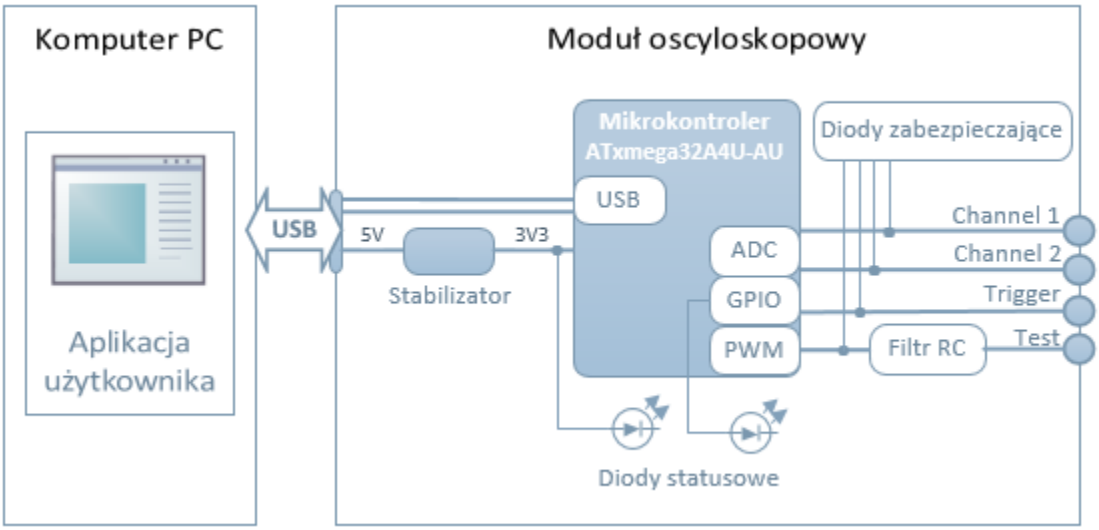
Metoda	Liczba klatek na sekundę
1.Clear and plot	8
2.Edit line and update	14
3.Edit line and selective update	15
4.Edit line and selective update (Qt Frame)	67

4. Architektura systemu

4.1. Elementy

W skład projektowanego systemu wchodzą dwa główne elementy: aplikacja użytkownika oraz moduł oscyloskopowy. Pomiędzy elementami systemu wymiana informacji zapewniona jest dzięki wykorzystaniu interfejsu USB. Aplikację użytkownika stanowić ma skrypt uruchamiany na komputerze PC. Moduł oscyloskopowy stanowić ma obwód drukowany, który ma być wyposażony w następujące elementy elektroniczne:

- Mikrokontroler,
- Terminale,
- Stabilizator,
- Diody zabezpieczające,
- Filtr RC,
- Diody statusowe.



Rys. 4.1. Schemat architektury kompletnego systemu.

4.2. Funkcje podzespołów

4.2.1. Aplikacja użytkownika

Umożliwia komunikację z modułem oscyloskopowym poprzez interfejs USB oraz pozwala na prezentowanie obserwowanych sygnałów na ekranie komputera.

4.2.2. Moduł oscyloskopowy

4.2.2.1. Mikrokontroler

Mikrokontroler odpowiada za:

- zbieranie próbek z obserwowanych sygnałów,
- transfer zebranych danych do komputera PC,
- obserwacje sygnału wyzwalacza,
- generowanie sygnału testowego,
- zarządzaniem stanem diody statusowej.

4.2.2.2. Terminale

Terminale stanowią interfejs elektromechaniczny modułu oscyloskopowego. Ich funkcje są następujące:

- Channel 1 - wejścia pierwszego kanału,
- Channel 2 - wejścia pierwszego kanału,
- Trigger - wejścia sygnału wyzwalającego,
- Test - wyjścia sygnału testowego.

4.2.2.3. Stabilizator

Kolejnym elementem modułu oscyloskopowego jest stabilizator. Zadaniem tego układu jest konwersja napięcia dostarczonego do modułu poprzez styk USB (5V) do napięcia pracy mikrokontrolera (3.3V).

4.2.2.4. Diody zabezpieczające

Każdy z portów jest równolegle podłączony do układu zabezpieczającego. Rozwiązanie to ma za zadanie uchronić mikrokontroler przed zniszczeniem w wyniku wystąpienia przepięcia lub pojawienia się ciągłego napięcia z poza zakresu pracy mikrokontrolera.

4.2.2.5. Filtr RC

Filtr RC umieszczony jest szeregowo w linii sygnału testowego generowanego przez mikrokontroler. Na wejście filtra trafia sygnał PWM o wypełnieniu 50%. Zadaniem układu jest nadanie sygnałowi testowemu charakteru analogowego poprzez wprowadzenie stałej czasowej.

4.2.2.6. Diody statusowe

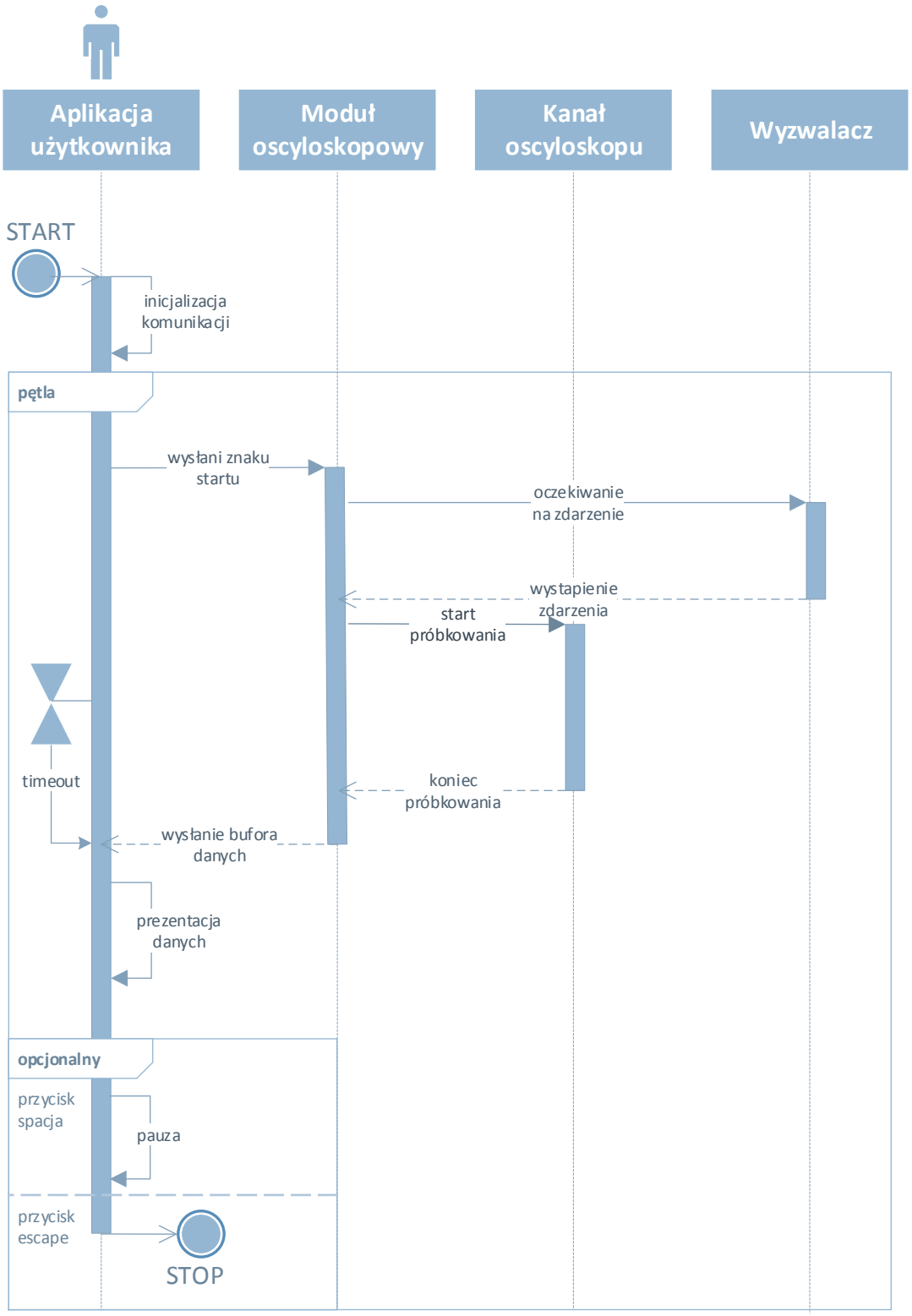
W układzie zostały umieszczone dwie diody statusowe. Pierwsza z nich informuje użytkownika o dostępie do zasilania Druga z nich pulsuje w trakcie odsyłania danych do komputera PC.

4.3. Współpraca aplikacja użytkownika – moduł oscyloskopowy

Projektowany system działa w klasycznej konfiguracji master-slave. Rolę układu nadrzędnego pełni aplikacja użytkownika, natomiast podrzędnego moduł oscyloskopowy. Schemat komunikacji prezentuje Rys. 4.2.

Praca aplikacji użytkownika rozpoczyna się od inicjalizacji komunikacji z modułem oscyloskopowym. Następnie program pracujący na komputerze PC wysyła do oscyloskopu znak startu pomiaru i oczekuje na odpowiedź. Po otrzymaniu zwrotnej wiadomości z danymi, następuje prezentacja wykresu. W przypadku braku pojawienia się wiadomości zwrotnej, po określonym czasie pojawia się timeout. W tym wypadku ponownie prezentowane są ostatnio odebrane dane lub wartości domyślne, gdy wcześniej nie udało się zarejestrować żadnego pomiaru. Opcjonalnie użytkownik może zatrzymać prezentacje danych za pomocą przycisku spacji lub zakończyć działanie programu poprzez wciśniecie przycisku „*escape”*.

Moduł oscyloskopowy odpowiada na zapytania pochodzące z aplikacji użytkownika. Po otrzymaniu znaku startu akwizycji moduł oczekuje na wystąpienie sygnału wyzwalającego. Po jego pojawieniu się oscyloskop rozpoczyna pomiar i gdy zostanie zapełniony bufor z danymi natychmiast zostaje odesłany do komputera. Następnie moduł znowu przechodzi w stan oczekiwania na znak startu pochodzący z aplikacji użytkownika.



Rys. 4.2. Diagram sekwencji działania systemu.

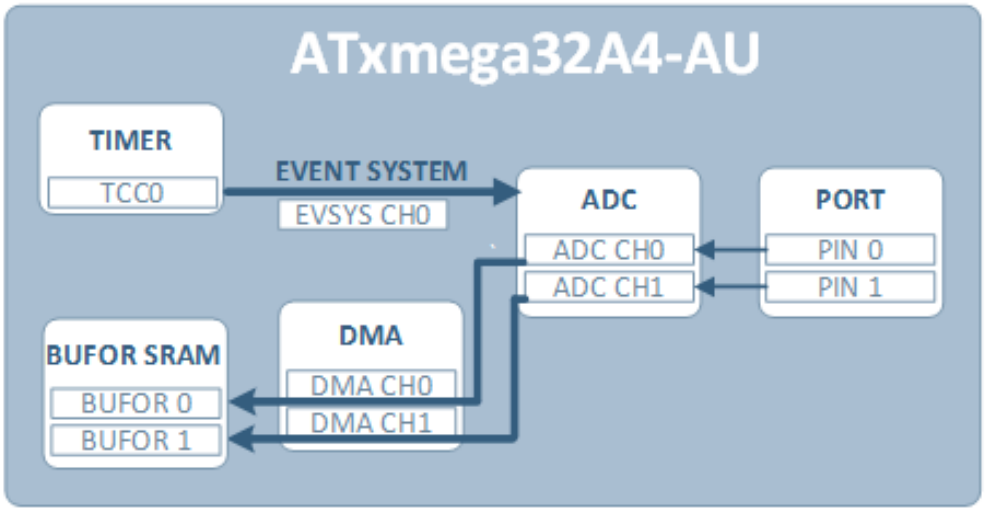
5. Moduł oscyloskopowy

Budowa modułu oscyloskopowego wymagała napisania programu sterującego mikrokontrolerem, który będzie w pełni wykorzystywał możliwości przetwornika ADC. Po realizacji tego zadania zaprojektowany i wykonany został obwód drukowany oscyloskopu.

5.1. Schemat akwizycji

W celu osiągnięcie dobranej częstotliwości próbkowania (1000 kSa/s na kanał) przyjęto przedstawiony na rysunku 5.1 schemat akwizycji.

Sygnał wyzwalający próbkowanie generowany jest co 1us przez timer, który dzięki wykorzystaniu systemu zdarzeń wyzwoli konwersję kolejno na dwóch kanałach przetwornika ADC. Do przesyłania wyników konwersji wykorzystano dwa kanały kontrolera DMA. Dane transportowane są bezpośrednio po zakończeniu konwersji na danym kanale przetwornika ADC przez odpowiadający mu kanał DMA. Wyniki umieszczane są w pamięci RAM w buforze przypadającym na dany kanał DMA.



Rys. 5.1. Schemat zasady działania układu próbkującego.

5.2. Implementacja programu mikrokontrolera

W poniższym rozdziale zaprezentowany został kod programu sterującego mikrokontrolerem. Do jego napisania wykorzystano biblioteki ASF stanowiące wysokopoziomowy interfejs dla wykorzystanych układów peryferyjnych mikrokontrolera.

Listing 5.1. prezentuje pierwszą część programu mikrokontrolera. W pierwszej kolejności zaimportowane zostały wymagane pliki nagłówkowe (1-2). Następnie zdefiniowane zostały makrodefinicje preprocesora(4-22), które zostaną wykorzystane w zainicjalizowaniu układów peryferyjnych mikrokontrolera. Kolejne linie programu (24-28) to prototypu funkcji inicjalizujących wykorzystane moduły sprzętowe. Na końcu listingu (36) znajdują się deklaracje bufora przeznaczonego na próbki monitorowanych sygnałów.

Listing 5.1. Program mikrokontrolera, część 1.

```
1    #include <asf.h>
2    #include "conf_usb.h"
3
4    #define RAM_BUFFER_SIZE_2000   2000
5    #define RAM_BUFFER_SIZE_1000   1000
6
7    #define ADC_GAIN               1
8    #define ADC_NR_TRIGGER_CHANNEL 2
9    #define ADC_EVENT_CHANNEL      3
10   #define ADC_CLOCK_RATE         2000000
11
12   #define DMA_CHANNEL_0          0
13   #define DMA_CHANNEL_1          1
14
15   #define TIMER_RESOLUTION       1000000
16   #define TIMER_PERIOD           1
17
18   #define PWM_FREQUENCY          12500
19   #define PWM_DUTY_CYCLE         50
20
21   #define IO_TRIGGER             IOPORT_CREATE_PIN(PORTC, 1)
22   #define IO_CTRL_LED            IOPORT_CREATE_PIN(PORTC, 4)
23
24   static void adc_init(void);
25   static void dma_init(void);
26   static void evsys_init(void);
27   static void tc_init(void);
28   static void pwm_enable(void);
29
30   struct buffer
31   {
32          uint8_t buffer_channel_0[RAM_BUFFER_SIZE_1000];
33          uint8_t buffer_channel_1[RAM_BUFFER_SIZE_1000];
34   };
35
36   struct buffer buffer_samples;
```

Listing 5.2 przedstawia funkcję główną main programu mikrokontrolera. Wykonywanie jej rozpoczyna się od zainicjalizowania (40-50) i uruchomienia (52-55) poszczególnych układów peryferyjnych mikrokontrolera. Następnie wykonywana jest nieskończoną pętla, w której realizowane są właściwe funkcje modułu oscyloskopowego. Mikrokontroler oczekuje na znak startu akwizycji (59). Po jego otrzymaniu czeka na wystąpienie zdarzenia wyzwalającego (60-61), czyli pojawienia się opadającego zbocza. Wystąpienie zdarzenia wyzwalającego uruchamia dwa kanały DMA (63-64), które transportują wyniki konwersji z rejestrów przetwornika ADC do bufora znajdującego się w pamięci RAM. Po jego zapełnieniu bufor odsyłany jest do komputera PC (68-69), a dioda statusowa zmienia swój stan na przeciwny (70).

Listing 5.2. Program mikrokontrolera, część 2.

```
38  int main(void)
39  {
40          sysclk_init();
41          pmic_init();
42          adc_init();
43          dma_init();
44          evsys_init();
45          tc_init();
46          ioport_init();
47
48          ioport_set_pin_dir(IO_TRIGGER, IOPORT_DIR_INPUT);
49          ioport_set_pin_mode(IO_TRIGGER, IOPORT_MODE_PULLUP);
50          ioport_set_pin_dir(IO_CTRL_LED, IOPORT_DIR_OUTPUT);
51
52          cpu_irq_enable();
53          adc_enable(&ADCA);
54          pwm_enable();
55          udc_start();
56
57          while (1)
58          {
59                  while(!udi_cdc_getc());
60                  while (0 == ioport_get_pin_level(IO_TRIGGER));
61                  while (1 == ioport_get_pin_level(IO_TRIGGER));
62
63                  dma_channel_enable(DMA_CHANNEL_0);
64                  dma_channel_enable(DMA_CHANNEL_1);
65
66                  if (udi_cdc_is_tx_ready())
67                  {
68                        while(dma_channel_is_busy(DMA_CHANNEL_1));
69                        udi_cdc_write_buf(&buffer_samples,RAM_BUFFER_SIZE_2000);
70                        ioport_toggle_pin_level(IO_CTRL_LED);
71                  }
72          }
73  }
```

Listing 5.3 prezentuje definicje funkcji inicjalizujących następujące układy peryferyjne mikrokontrolera (wykorzystano makrodefinicje z listingu 5.1):

- System zdarzeń (74-78), udostępniono sygnał zegarowy dla modułu, a następnie podłączono zdarzenie przepełnienia timera 1 do kanału 3 systemu zdarzeń.
- Timer (79-85), ustawiono rozdzielczość timera 1 na poziomie 1MHz oraz okres pojawienia się przepełnienia timera równy 1.
- PWM (86-92), ustawiono częstotliwość generowania sygnału PWM równą 12,5kHz i współczynnik wypełnienia równy 50%.
- Przetwornik ADC (93-111,) skonfigurowano dwa kanały przetwornika. Ustawiono tryb pracy ADC bez znaku, rozdzielczość 8 bit, napięcie referencyjne wewnętrzne mikrokontrolera (3,3V) oraz wzmocnienie równe 1. Przetwornik oczekuje na wystąpienie zdarzenia na kanale 3 systemu zdarzeń, które generowane jest przez timer 1. Wystąpienie zdarzenia wyzwala konwersją kolejno na 2 skonfigurowanych kanałach przetwornika.

Listing 5.3. Program mikrokontrolera, część 3.

```
74  static void evsys_init(void)
75  {
76          sysclk_enable_module(SYSCLK_PORT_GEN, SYSCLK_EVSYS);
77          EVSYS.CH3MUX = EVSYS_CHMUX_TCC1_OVF_gc;
78  }
79  static void tc_init(void)
80  {
81          tc_enable(&TCC1);
82          tc_set_wgm(&TCC1, TC_WG_NORMAL);
83          tc_write_period(&TCC1, TIMER_PERIOD);
84          tc_set_resolution(&TCC1, TIMER_RESOLUTION);
85  }
86  static void pwm_enable(void)
87  {
88          struct pwm_config pwm_cfg0;
89
90          pwm_init(&pwm_cfg0, PWM_TCC0, PWM_CH_A, PWM_FREQUENCY);
91          pwm_start(&pwm_cfg0, PWM_DUTY_CYCLE);
92  }
93  static void adc_init(void)
94  {
95          struct adc_config adc_conf;
96          struct adc_channel_config adcch_conf;
97
98          adc_read_configuration(&ADCA, &adc_conf);
99          adcch_read_configuration(&ADCA, ADC_CH0, &adcch_conf);
100         adc_set_conversion_parameters(&adc_conf,ADC_SIGN_OFF,ADC_RES_8,
101                                       ADC_REF_VCC);
102         adc_set_conversion_trigger(&adc_conf, ADC_TRIG_EVENT_SWEEP,
103                                    ADC_NR_TRIGGER_CHANNEL, ADC_EVENT_CHANNEL);
104         adc_set_clock_rate(&adc_conf, ADC_CLOCK_RATE);
105         adc_write_configuration(&ADCA, &adc_conf);
106
107         adcch_set_input(&adcch_conf, ADCCH_POS_PIN1, ADCCH_NEG_NONE, ADC_GAIN);
108         adcch_write_configuration(&ADCA, ADC_CH0, &adcch_conf);
109         adcch_set_input(&adcch_conf, ADCCH_POS_PIN5, ADCCH_NEG_NONE, ADC_GAIN);
110         adcch_write_configuration(&ADCA, ADC_CH1, &adcch_conf);
111 }
```

Na listingu 5.4 została umieszczona definicja funkcji, której zadaniem jest inicjalizacja kontrolera DMA. Funkcja ustawia dwa kanały DMA, po jednym dla każdego kanału ADC, które będą pracowały w przysyłaniu 1bajta. Adres źródłowy wskazuje na rejestr wynikowy odpowiedniego kanału przetwornika ADC, natomiast docelowy ustawiony jest na adres bufora w pamięci RAM. Transfer pomiędzy wymienionymi adresami wyzwalany jest po zakończeniu konwersji na odpowiadającym kanale przetwornika ADC.

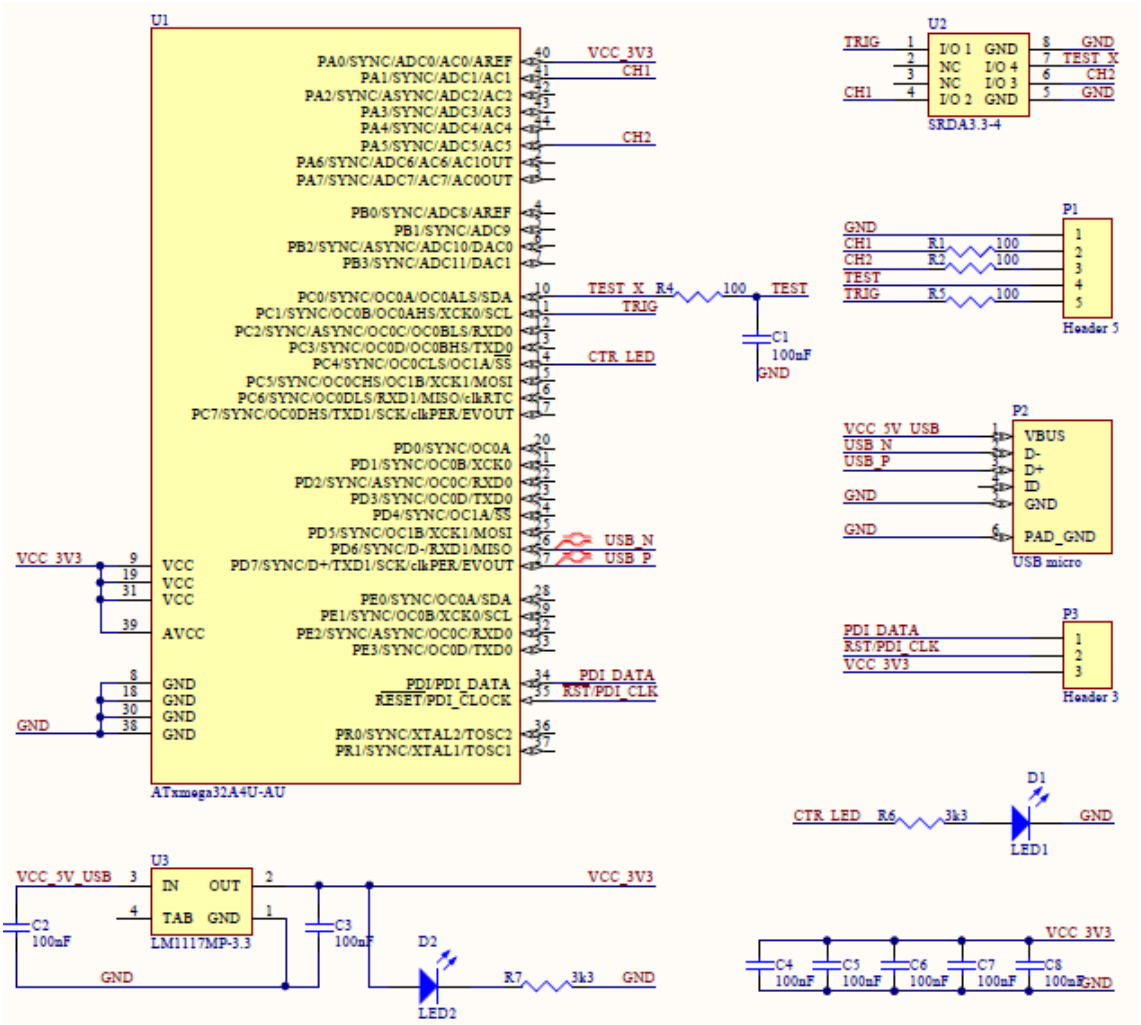
Listing 5.4. Program mikrokontrolera, część 4.

```
112  static void dma_init(void)
113  {
114         struct dma_channel_config dmach_conf0;
115         struct dma_channel_config dmach_conf1;
116         memset(&dmach_conf0, 0, sizeof(dmach_conf0));
117        memset(&dmach_conf1, 0, sizeof(dmach_conf1));
118
119         dma_channel_set_burst_length(&dmach_conf0, DMA_CH_BURSTLEN_1BYTE_gc);
120         dma_channel_set_transfer_count(&dmach_conf0, RAM_BUFFER_SIZE_1000);
121         dma_channel_set_src_reload_mode(&dmach_conf0, DMA_CH_SRCRELOAD_BURST_gc);
123         dma_channel_set_dest_reload_mode(&dmach_conf0,
124                                          DMA_CH_DESTRELOAD_TRANSACTION_gc);
125         dma_channel_set_src_dir_mode(&dmach_conf0, DMA_CH_SRCDIR_INC_gc);
126         dma_channel_set_dest_dir_mode(&dmach_conf0, DMA_CH_DESTDIR_INC_gc);
127         dma_channel_set_source_address(&dmach_conf0,(uintptr_t)&ADCA.CH0RES);
128         dma_channel_set_destination_address(&dmach_conf0,(uint16_t)(uintptr_t)
129                                             buffer_samples.buffer_channel_0);
130         dma_channel_set_trigger_source(&dmach_conf0,DMA_CH_TRIGSRC_ADCA_CH0_gc );
131         dma_channel_set_single_shot(&dmach_conf0);
132
133         dma_channel_set_burst_length(&dmach_conf1, DMA_CH_BURSTLEN_1BYTE_gc);
134         dma_channel_set_transfer_count(&dmach_conf1, RAM_BUFFER_SIZE_1000);
135         dma_channel_set_src_reload_mode(&dmach_conf1, DMA_CH_SRCRELOAD_BURST_gc);
136         dma_channel_set_dest_reload_mode(&dmach_conf1,
137                                          DMA_CH_DESTRELOAD_TRANSACTION_gc);
138        dma_channel_set_src_dir_mode(&dmach_conf1, DMA_CH_SRCDIR_INC_gc);
139         dma_channel_set_dest_dir_mode(&dmach_conf1, DMA_CH_DESTDIR_INC_gc);
140         dma_channel_set_source_address(&dmach_conf1,(uintptr_t)&ADCA.CH1RES);
141         dma_channel_set_destination_address(&dmach_conf1,(uint16_t)(uintptr_t)
142                                             buffer_samples.buffer_channel_1);
143         dma_channel_set_trigger_source(&dmach_conf1,DMA_CH_TRIGSRC_ADCA_CH1_gc );
144         dma_channel_set_single_shot(&dmach_conf1);
145
146         dma_enable();
147         dma_channel_write_config(DMA_CHANNEL_0, &dmach_conf0);
148         dma_channel_write_config(DMA_CHANNEL_1, &dmach_conf1);
149  }
```

5.3. Część sprzętowa

5.3.1. Schemat

Rys 5.2. przedstawia schemat elektryczny zaprojektowanego obwodu drukowanego. Na schemacie pokazano wszystkie wymagane połączenia pomiędzy elementami modułu oscyloskopowego. Dodatkowo oprócz układów wymienionych w rozdziale 4, w obwodzie drukowanym umieszczono złącze P3. Ma ono posłużyć do wgrania do pamięci mikrokontrolera programu sterującego.



Rys. 5.2. Schemat elektryczny projektowanego modułu.

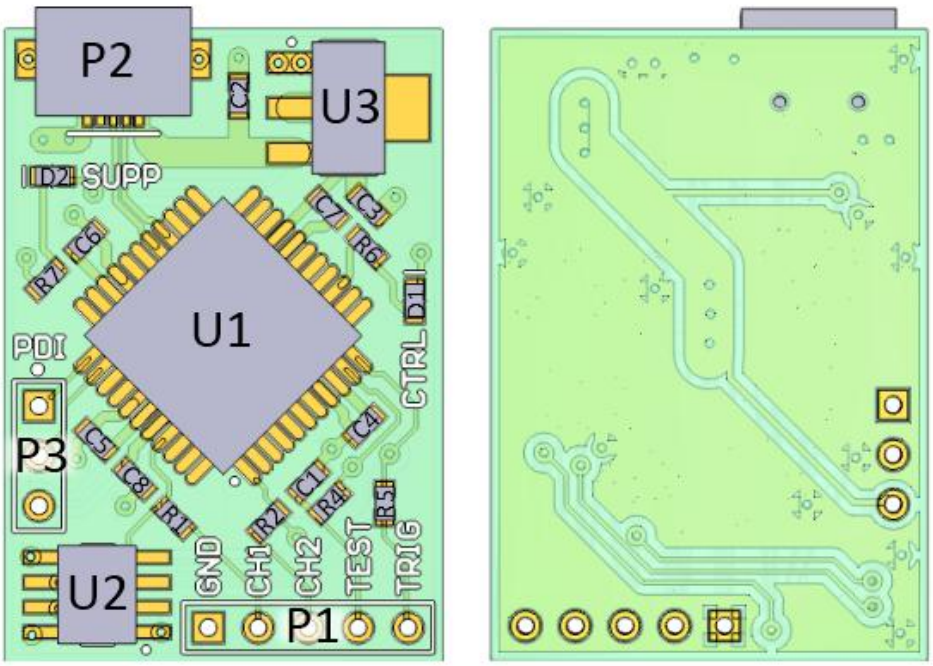
5.3.2. Obwód drukowany

Płytka PCB została zaprojektowana zgodnie z określonymi w rozdziale 1.4 założeniami. Parametry obwodu drukowanego:

- wymiary: 22 x 32 mm,
- obwód dwuwarstwowy,
- min szerokość ścieżek: 8 mil,
- min odstęp między ścieżkami: 6 mil,
- min otwór: 0,45 mm,
- min obwódka: 11 mil,
- opisy: warstwa TOP.

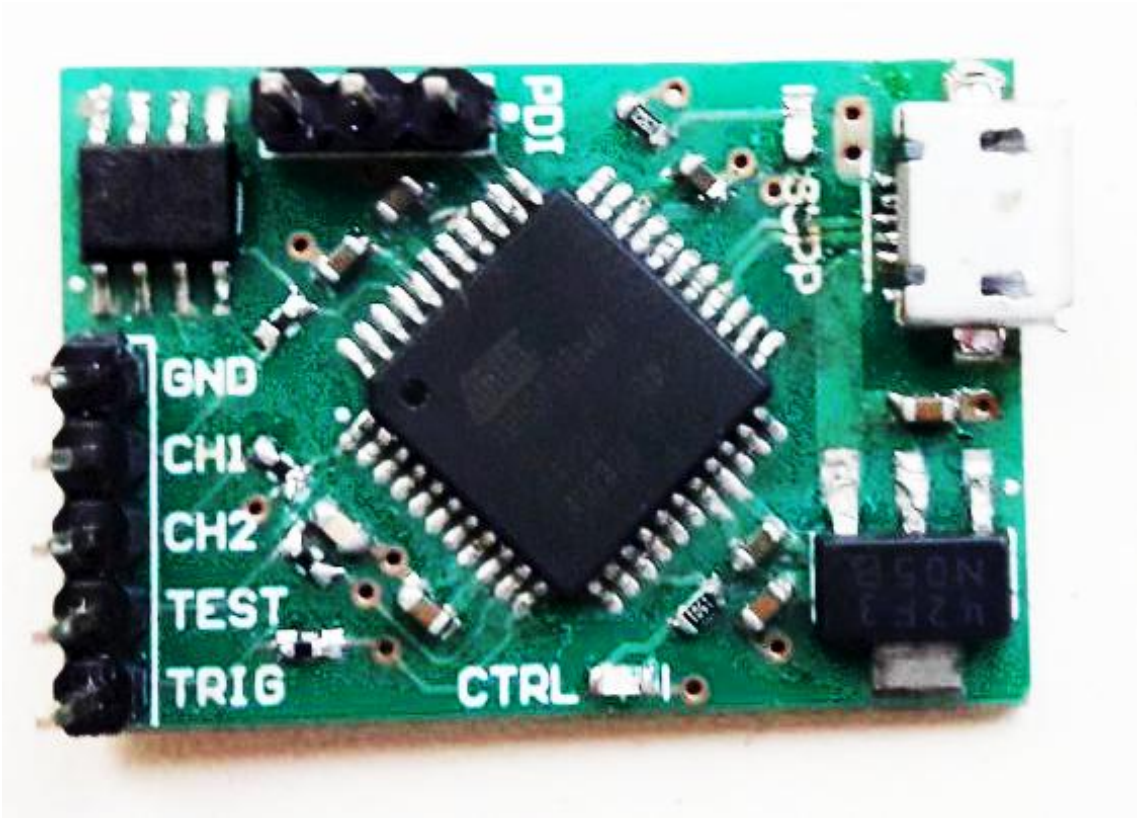
Podzespoły elektroniczne zostały rozmieszczone na płytce w sposób uporządkowany, pozwalając na prowadzenie możliwie krótkich ścieżek. Centralną część zajmuje element wymagający największej liczby połączeń z pozostałymi układami, czyli mikrokontroler. Wokół niego możliwie blisko ulokowane zostały kondensatory blokujące. Następnie na krawędziach obwodu rozmieszczono pozostałe układy elektroniczne oraz złącza. Wolne przestrzenie zostały zajęte przez diody sygnalizujące oraz rezystory leżące w torze kanałów wejściowych. Rozmieszczając poszczególne elementy wzięto również pod uwagę wymagane miejsce na opisy.

Na poniższym rysunku zaprezentowano rozmieszczenie elementów na zaprojektowanym obwodzie drukowanym.



Rys. 5.3. Rozmieszczenie elementów na płytce drukowanej, widok z góry i dołu.

Na poniższym rysunku znajduję się wykonany moduł oscyloskopowy.



Rys. 5.4. Rzeczywisty model wykonanego modułu oscyloskopowego.

6. Aplikacja użytkownika

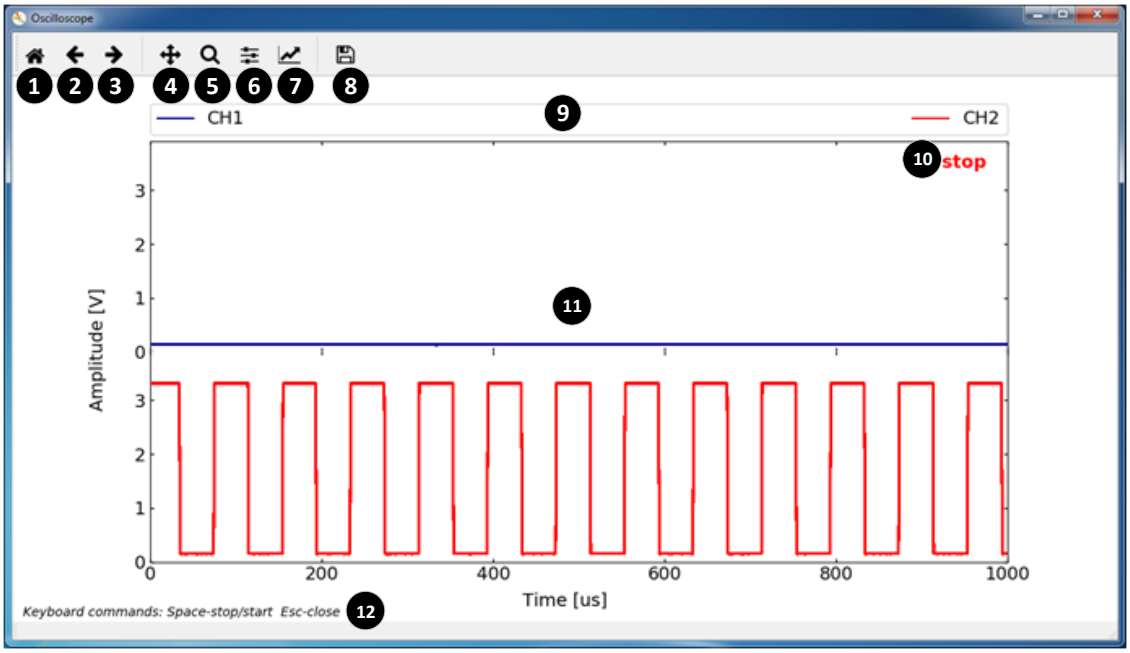
6.1. Opis funkcjonalności

Aplikacja użytkownika pozwala na:

- odbieranie i prezentowanie danych w postaci wykresu w czasie rzeczywistym,
- zatrzymanie oraz ponowny start akwizycji danych,
- przybliżanie i oddalanie prezentowanego wykresu,
- zmienianie rozmiaru okna prezentowanego wykresu,
- zapis otrzymanego wykresu do pliku graficznego.

6.2. Interfejs użytkownika

Interfejs użytkownika został przedstawiony na rysunku 6.1. Zbudowany jest on z dwóch głównych części, paska narzędzi oraz wykresu, gdzie prezentowane są zarejestrowane przebiegi.



Rys. 6.1. Aplikacja użytkownika, interfejs graficzny.

Opis poszczególnych elementów interfejsu:

1. Powrót do domyślnych ustawień wykresu.
2. Powrót do poprzedniego widoku.
3. Powrót do poprzedniego widoku po użyciu przycisku cofnij.
4. Powiększanie i przesuwanie. Przy użyciu prawego i lewego przycisku myszy umożliwia przesuwanie i powiększanie wykresu wzdłuż jednej z wybranych osi.
5. Powiększenia. Przy użyciu prawego przycisku myszy umożliwia zaznaczenie prostokątnego obszaru, który ma ulec powiększeniu.
6. Konfiguracja rozmiaru wykresu.
7. Konfiguracja parametrów wykresu.
8. Funkcja zapisu.
9. Legenda wykresu.
10. Status akwizycji.
11. Wykres.
12. Stopka informacyjna.

6.3. Implementacja

6.3.1. Program główny aplikacji użytkownika

Program główny pełni integralną funkcję aplikacji użytkownika. Wykorzystuje zaimplementowany moduł Gui do prezentacji wyników akwizycji oraz pakiet PySerial do komunikacji z modułem oscyloskopowym. Na listingu 6.1. oraz 6.2. została przedstawiono jego implementacja, którą wykonano w oparciu o diagram sekwencji działa systemu (Rys 4.2).

Listing 6.1. Program główny aplikacji użytkownika, inicjalizacja.

```
1    from Gui import *
2    import serial
3
4    CHANNEL_NR=2
5    SAMPLES_NR=1000
6
7    serial_port =serial.Serial(port='COM3',write_timeout=0.1,inter_byte_timeout=0.1)
8    serial_port.flushInput()
9    RTS= bytearray(b'\xff')
10
11   gui=GUI(CHANNEL_NR,SAMPLES_NR)
12   paused = False
13
```

Rozpoczynając inicjalizację programu (listing 6.1.), w pierwszej kolejności zostają zaimportowane niezbędne pakiety (1-2). Następnie zostaje zdefiniowana liczba obserwowanych kanałów oraz liczba próbek przydająca na poszczególny kanał (4-5).

W kolejnym kroku utworzony zostaje obiekt klasy *serial* (7), który pozwoli wykorzystać wirtualny port COM3. Dodatkowo określone zostają limity czasowe na dane wychodzące i przychodzące. W celu uniknięcia prezentacji przypadkowych danych zostaje wyczyszczony bufor wejściowy portu (3). Zdefiniowany zostaje również znak zapytania o zbierana przez oscyloskop dane (9).

Część inicjalizacyjną programu i kończy utworzenie obiektu klasy *Gui* (11). Zdefiniowana zostaje również zmienna pomocnicza (12), która pozwoli określenie obecnego stanu aplikacji.

Listing 6.2 prezentuje część właściwą programu głównego. Zawarta jest ona w niekończącej się pętli *while* , w której wykonują się sekwencyjnie trzy zadania:

- Sprawdzenie stanu i obsługa ostatnio wciśniętego przycisku (16-21).
- Zapytanie i próba odebrania danych z oscyloskopu (23-29).
- Prezentacja danych (31).
- Pauza umożliwiająca obsługę zdarzeń przez moduł pyplot (32).

Listing 6.2. Program główny aplikacji użytkownika, pętla główna.

```
14  while True:
15
16      key  = gui.get_key()
17      if key == ' ':
18          paused = not paused
19          gui.toggle_pause()
20      elif key  == 'escape':
21          break
22
23      if not paused:
24          try:
25              serial_port.write(RTS)
26              for channel in range(CHANNEL_NR):
27                  gui.set_y(channel, serial_port.read(SAMPLES_NR))
28          except:
29              pass
30
31      gui.plot()
32      gui.pause()
```

6.3.2. Moduł Gui

Moduł Gui jest niezależnym komponentem wchodzącym w skład aplikacji użytkownika. Jego zadaniem jest prezentacja badanych przebiegów na ekranie komputera. Zastosowanie takiego modułowego podejścia, pozwala na modyfikacje dowolnego z elementów, bez konieczności wprowadzania zmian w pozostałych

Moduł Gui udostępnia klasę pozwalającą na utworzenie i zarządzanie wykresem. Listing 6.3. prezentuje inicjalizację klasy GUI.

Funkcjonalności jakie udostępnia zaimplementowana klasa to :

- Prezentacja statusu akwizycji (aktywna/zatrzymana) (*toggle_pause*, 58-65).
- Ustawienie wartości dla określonego kanału (*set_y*, 67-69).
- Prezentacja odebranych danych na wykresie (*plot*,71-77).
- Zatrzymanie programu na 10ms, umożliwienie obsługi zdarzeń modułowi pyplot (*pause*, 79-80).
- Sprawdzenie ostatnio wciśniętego przycisku (*get_key*,82-86).

Na początku został zaimportowany pakiet matplotlib (1). Następnie dokonano wyboru warstwy zaplecza (2) oraz warstwy skryptowej (3) importowanego pakietu.

W kolejnych liniach (8-50) znajduję się konstruktor klasy GUI, który na podstawie informacji o liczbie kanałów oraz wielkości bufora próbek, dokonuje konfiguracji okna pomiarowego. Całą procedurę kończy wyświetlenie zdefiniowanego okna aplikacji użytkownika (52-53).

Listing 6.3. Moduł gui, część 1.

```
1    import matplotlib
2    matplotlib.use('Qt5Agg')
3    import matplotlib.pyplot as plt
4
5    COLORS_LIST = ('darkblue', 'red', 'green', 'gold')
6    CHANNEL_LIST =('CH1', 'CH2', 'CH3', 'CH4')
7
8    class GUI:
9        def __init__(self, channel_nr, samples_nr):
10
11           self.fig, self.ax = plt.subplots(2,1,sharex = True)
12           self.fig.subplots_adjust(hspace=0)
13           self.fig.canvas.mpl_connect('key_press_event', self.__set_key)
14           self.fig.canvas.set_window_title('Oscilloscope')
15           self.fig.set_size_inches(12, 6)
16
17           self.__channel_nr = channel_nr
18           self.__sample_nr = samples_nr
19           self.__line = list()
20           self.__key = None
21           self.__pause = False
22
23           self.__state = self.ax[0].text(0.95, 0.9, 'run',
24                          horizontalalignment='center', verticalalignment='center',
25                          transform=self.ax[0].transAxes, color='green',
26                          fontsize=14, fontweight='bold')
27
28           self.fig.text(0.5, 0.03, 'Time [us]', ha='center', fontsize=14)
29           self.fig.text(0.07, 0.5, 'Amplitude [V]', va='center',
30                         rotation='vertical', fontsize=14)
31           self.fig.text(0.01, 0.01, 'Keyboard commands: Space-stop/start  Esc-
32                         close', ha='left', fontsize= 10, style='italic')
33
```

```
34          for channel in range(self.__channel_nr):
35              canal, = self.ax[channel].plot(list(0 for i in range(samples_nr)),
36                       color = COLORS_LIST[channel], label =CHANNEL_LIST[channel])
37              self.ax[channel].axis([0, samples_nr, 0, 3.9])
38              self.ax[channel].tick_params(which='both', direction = 'in',
39                                           length=3, width=1, labelsize = 14,
40                                           bottom=True, left=True, top=True,
41                                           right=True,grid_alpha=0.3)
42              self.__line.insert(channel, canal)
43              if channel > 0:
44                  self.ax[channel].spines['top'].set_visible(False)
45              if channel < channel_nr - 1:
46                  self.ax[channel].spines['bottom'].set_visible(False)
47
48          self.ax[0].legend(self.__line, CHANNEL_LIST[:self.__channel_nr] ,loc=2,
49                            ncol=2, mode="expand",borderaxespad=0.,
50                            fontsize=14,bbox_to_anchor=(0., 1.08, 1., .102))
51
52          self.fig.canvas.draw()
53          plt.pause(1)
```

Na listing 6.4 przedstawione zostały metody jakie udostępnia klasa GUI. Wyjątkiem jest metoda prywatna (55) pozwalająca na zapisanie ostatnio wciśniętego przycisku na klawiaturze komputera PC.

Listing 6.4. Moduł gui, część 2.

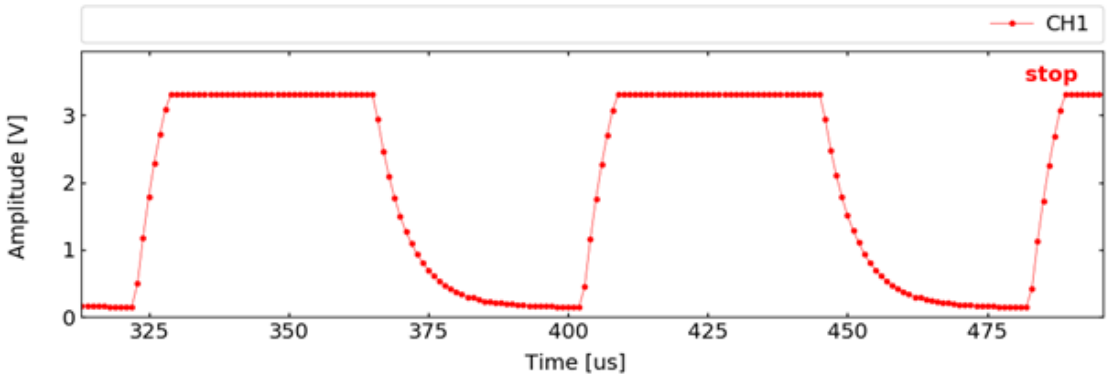
```
55      def __set_key(self, event):
56          self.__key = event.key
57
58      def toggle_pause(self):
59          self.__pause = not self.__pause
60          if self.__pause:
61              self.__state.set_text('stop')
62              self.__state.set_color('red')
63          else:
64              self.__state.set_text('run')
65              self.__state.set_color('green')
66
67      def set_y(self, channel, data):
68          self.__line[channel].set_ydata(
69                               [float(y/77.0) for y in data[:self.__sample_nr]])
70
71      def plot(self):
72          try:
73             for channel in range(self.__channel_nr):
74                 self.ax[channel].draw_artist(self.__line[channel])
75             self.fig.canvas.update()
76          except:
77              pass
78
79      def pause(self):
80          plt.pause(1e-2)
81
82      def get_key(self):
83          key = self.__key
84          if  key:
85              self.__key = None
86          return key
```

7. Testy

Celem wykonanych testów było sprawdzenie poprawności działania zbudowanego modułu oscyloskopowego i współpracującego z nim aplikacji użytkownika. Testowi poddano sygnał testowy. Sprawdzono również zaprojektowany system w docelowym środowisku pracy, czyli wykorzystano go do obserwacji transakcji zachodzących w zestawie ćwiczeniowym (rozdział 1.2).

7.1. Analiza sygnału testowego

Poniższy rysunek przedstawia fragment oscylogramu otrzymanego z sygnału testowego. Moduł oscyloskopowy próbkuje badany sygnał co 1us (1000kS/s), natomiast okres sygnału testowego wynosi 80us (12,kHz). Wynika z tego że powinniśmy otrzymać około 80 próbek na okres przebiegu testowego. Rysunek 7.1. potwierdza tą tezę, co wskazuje na poprawność wykonanego testu.

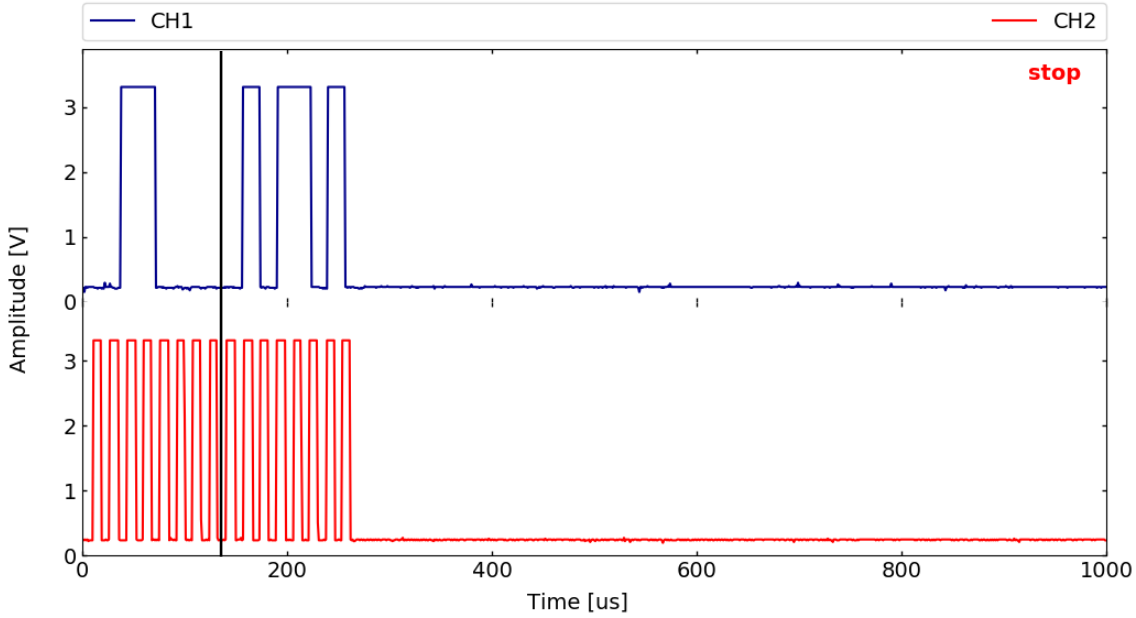


Rys. 7.1. Oscylogram sygnału testowego.

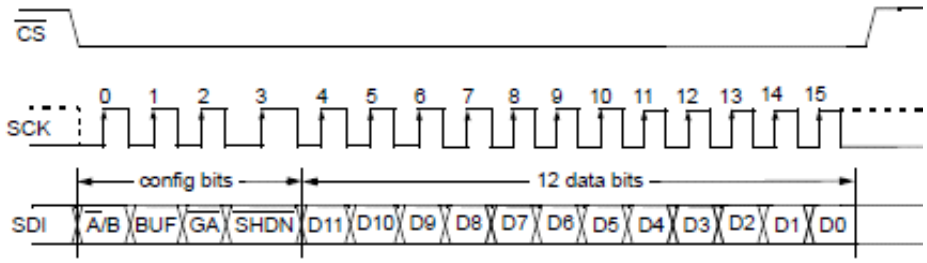
7.2. Analiza sygnałów interfejsu SPI oraz I2C

Test został wykonany w oparciu o wybrane transakcje z tabeli 1.2. Otrzymane oscylogramy monitorowanych transakcji, zestawiono z oczekiwanymi wynikami. Umieszczone na oscylogramach pionowe czarne linie, oddzielają przesyłane bajty danych.

Transakcja ustawienia wartości napięcia wyjściowego przetwornika DAC składa się z dwóch bajtów. Pierwszy bajt od lewej, zawiera cztery bity konfiguracyjne przetwornik (najstarsze). Przypisano im następujące wartości A/B - 0, BUF - 0, GA - 1, SCHDN - 1. Pozostałe cztery bity w tym słowie ustawione zostały na 0. Drugi przesyłany bajt zawiera bity danych i ustawiono mu wartość 0x5A (bin: 01011010).



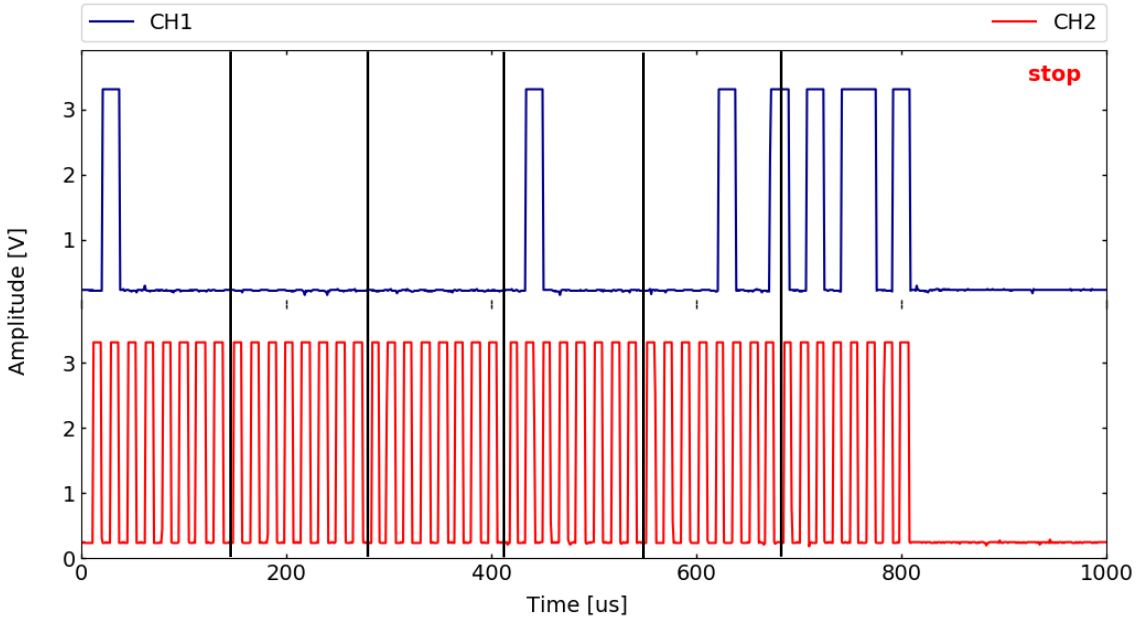
Rys. 7.2. Oscylogram dla transakcji ustawienie wartości napięcia wyjściowego przetwornika DAC.



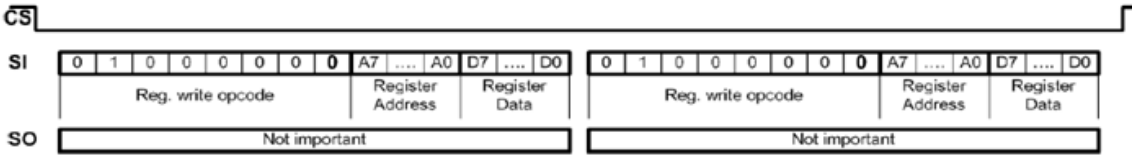
Rys. 7.3. Ustawienie wartości napięcia wyjściowego układu przetwornika DAC [18] - ramka transakcji(SPI).

Transakcja ustawienia stanu na wyjściu bramy IO składa się z sześciu bajtów. Układ nadrzędny wysyła jedynie dane do układu podrzędnego. Transmitowanym bajtom (linia MOSI) przypisano następujące wartości (od lewej):

- 0x40 (*Reg. write opcod*e) ,
- 0x00 (*Register Address*) ,
- 0x00 (*Register Data*) ,
- 0x40 (*Reg. write opcode*) ,
- 0x09 (*Register Address*) ,
- 0x5A (*Register Data*) .



Rys. 7.4. Oscylogram dla transakcji ustawienia stanu na wyjściu bramy IO.



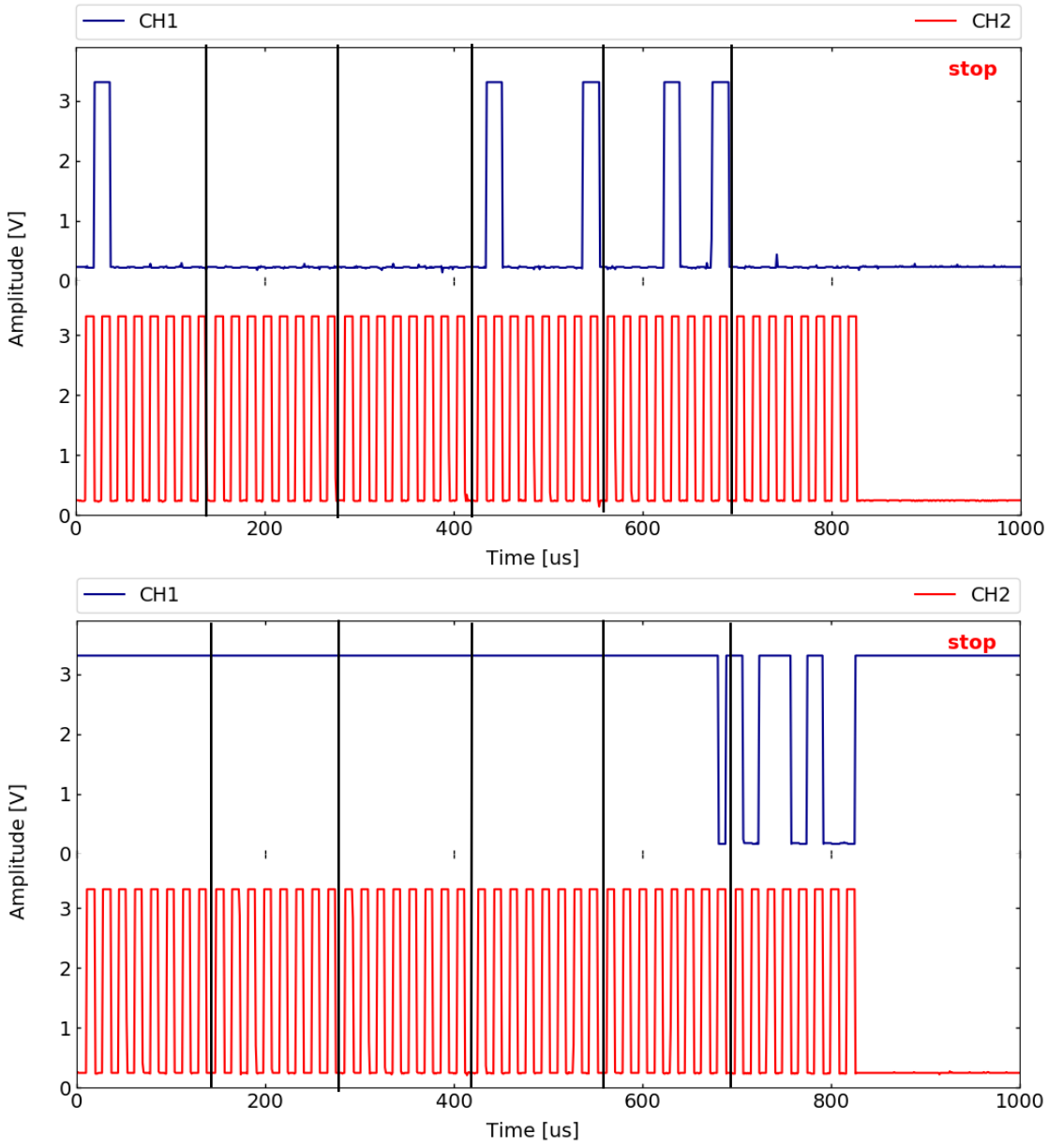
Rys. 7.5. Ustawienie stanu na wyjściu portu bramy IO[17] - ramka transakcji(SPI).

Transakcja odczytu stanu na wyjściu bramy IO składa się również z sześciu bajtów. Układ nadrzędny wysyła dane do układu podrzędnego poczym otrzymuje w odpowiedzi bajt danych określający stan bramy IO. Transmitowanym bajtom (linia MOSI) przypisano następujące wartości (od lewej):

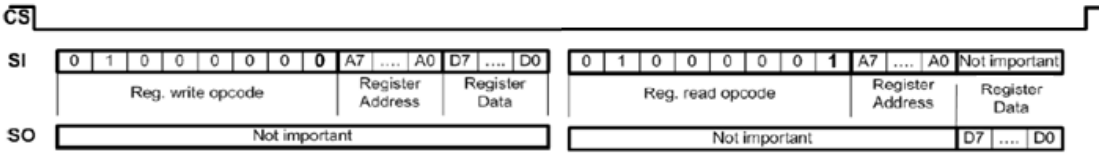
- 0x40 (*Reg. write opcod*e) ,
- 0x00 (*Register Address*) ,
- 0x00 (*Register Data*) ,
- 0x41 (*Reg. write opcode*) ,
- 0x09 (*Register Address*) ,

- 0x00 (*Register Data*) .

Otrzymane dane przez układ nadrzędny (linia MISO) to kolejno pięć nieznaczących bajtów oraz jeden bajt z informacją o stanie bramy IO. W tym przypadku stan bramy IO ustawiony był na 0x5A i taka wartość została odebrana.

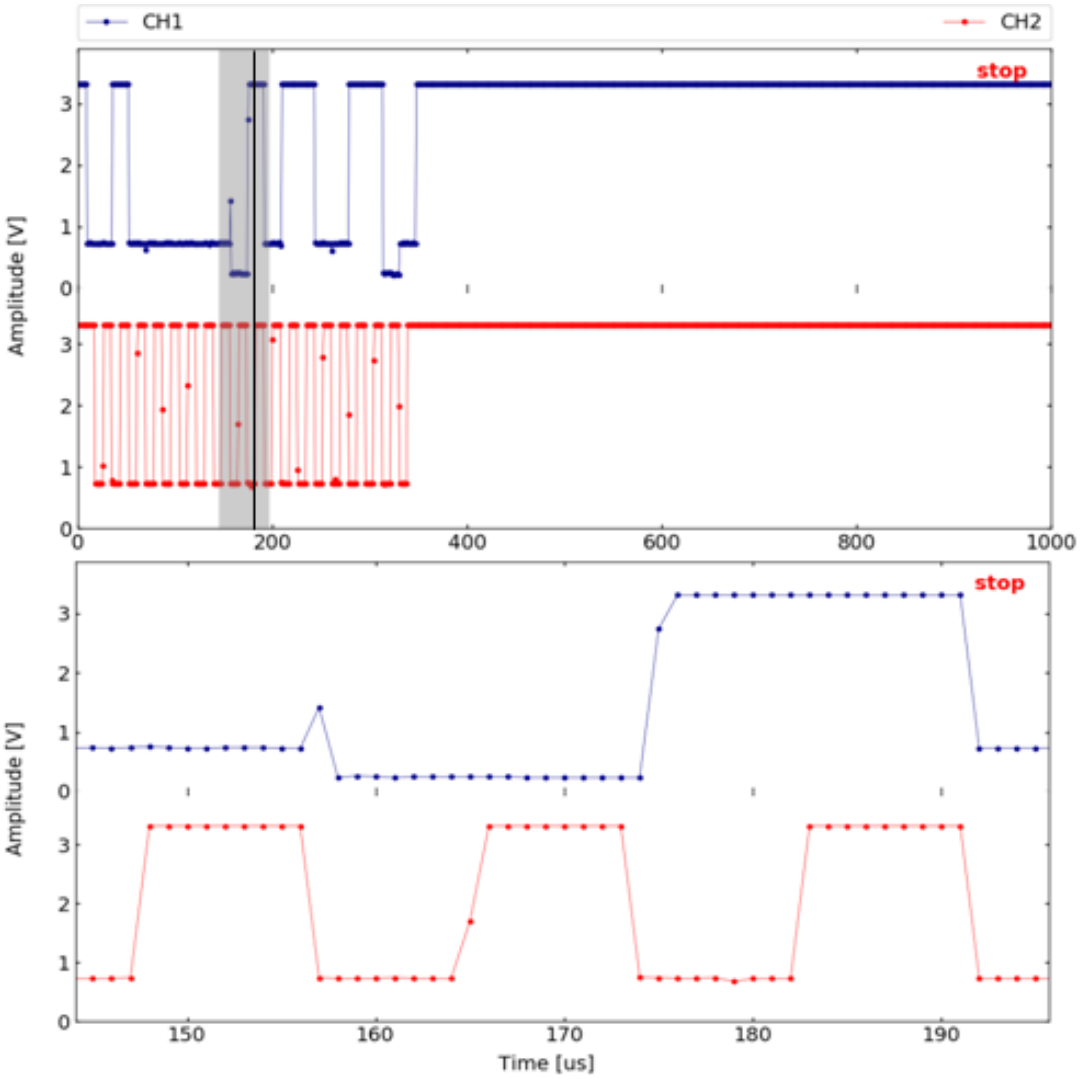


Rys. 7.6. Oscylogramy dla transakcji odczytu stanu na wyjściu bramy IO, górny wykres linia MOSI, dolny wykres linia MISO.

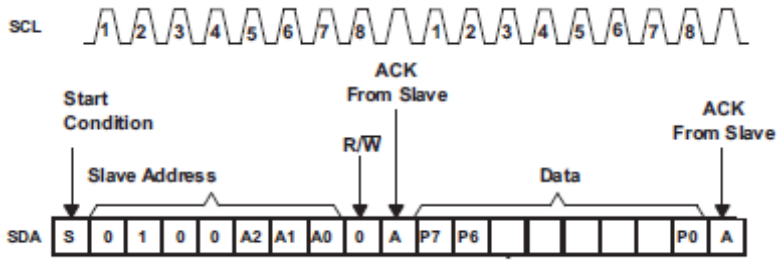


Rys. 7.7. Odczyt stanu na wyjściu portu bramy IO [17] - ramka transakcji(SPI).

Transakcja ustawienia stanu na wyjściu bramy IO składa się z dwóch bajtów. Siedem najstarszych bitów pierwszego przesyłanego bajtu określa adres układu podrzędnego, w tym przypadku to 0x20. Natomiast najmłodszy bit pierwszego słowa określa rodzaj transakcji, w tym wypadku zapis 0x00. Drugi bajt to dane przesyłane do zapisu w tym wypadku to 0xB3. Po każdym przesłanym bajcie układ podrzędny potwierdzanie otrzymanie danych wysyłając ACK, linia SDA zostaje ściągnięta do 0.



Rys. 7.8. Oscylogram dla transakcji ustawienia stanu na wyjściu bramy IO.



Rys. 7.9. Ustawienie stanu na wyjściu portu bramy IO [19] - ramka transakcji(I2C).

8. Podsumowanie

Celem powyższej pracy było zaprojektowanie i wykonanie miniaturowego modułu oscyloskopowego wykorzystującego interfejs USB. Projektowany moduł ma umożliwić monitorowanie linii sygnałowych interfejsu SPI oraz I2C, w trakcie realizacji ćwiczeń z interfejsów szeregowych.

Podczas realizacji pracy zaprojektowany został obwód drukowany modułu oscyloskopowego i napisany został program sterujący dla wykorzystanego mikrokontrolera. Zaimplementowano również aplikację użytkownika uruchamianą na komputerze PC.

Prace nad projektem rozpoczęto od określenia założeń projektowych. Następnie przystąpiono do doboru odpowiednich elementów sprzętowych oraz programowych pozwalających na spełnienie postawionych wymagań. W wyniku prac rozwojowych jak i testowych, powstało kompletne urządzenie. Ostatnim przeprowadzonym krokiem były testy funkcjonalne całego systemu, które potwierdziły jego przydatność.

W poniższej tabeli przedstawiono kosztorys budowy modułu oscyloskopowego. Uwzględnione zostały wyłącznie komponenty oraz czynności mające największy wpływ na koszt budowy zaprojektowanego modułu.

Tab. 8.1. Kosztorys budowy modułu oscyloskopowego[2].

Rodzaj wydatku	Cena [zł]
Mikrokontroler, ATxmega32A4U-AU	10.18
Gniazdo USB B micro, KEYSTONE 940	4.36
Diody zabezpieczające, SEMTECH SRDA3.3-4.TBT	3.84
Stabilizator, LM1117MP-3.3	3.15
Wykonanie płytki PCB	6.64
SUMA	**28.17**

[2] Kosztorys wykonano na podstawie cen oferowanych przez dostawcę komponentów elektronicznych TME oraz wykonawcy płytek PCB Satland. Ceny z dnia 25.08.2018 przy zakupie lub wykonaniu 20 sztuk.

W tabeli 8.2 przedstawiono parametry zaprojektowanego urządzenia w porównaniu z innymi popularnymi rozwiązaniami. Wynika z niej, że wykonany moduł oscyloskopowy charakteryzuję się znacznie niższymi parametrami od pozostałych przykładów. Pomimo tego są one wystarczające, aby spełniał postawiony przed nim cel. Dodatkowo jego koszt jest znacznie niższy w porównaniu z przedstawionymi rozwiązaniami.

Tab. 8.2. Najważniejsze parametry wybranych oscyloskopów uwzględniająca zaprojektowany moduł.

Rodzaj oscyloskopu	cyfrowy		karta		
Nazwa modelu	Rigol DS1022	Tektronix TDS3012B	Bit Scope 100	Instrustar ISDS205A	Projekt
Liczba kanałów	2	2	2	2	2
Pasmo analogowe [MHz]	25	100	100	20	0.200
Częstotliwość próbkowania [MSa/s /kanał]	200	1250	40	48	1
Rekord pamięci [kB/kanał]	512	10	64	1000	1
Rozdzielczość [bit]	8	9	8	8	8
Cena brutto [USD]	400	2000	550	51	7.5

8.1. Ograniczenia

Pomimo faktu, że zaprojektowany system spełnia podstawowe wymagania projektowe, to jednak posiada kilka słabych stron. Słabą stroną modułu oscyloskopowego jest brak informacji o stanie badanego przebiegu przed pojawieniem się zdarzenia wyzwalającego. Jednak największe zastrzeżenia można mieć co do aplikacji użytkownika uruchamianej na komputerze PC. Do poprawnego działania wymaga ona:

- zainstalowania odpowiednich pakietów języka Python,
- uprzedniego podpięcia modułu,
- podania numeru portu COM, do którego jest podpięty moduł,
- ciągłego podpięcia modułu.

8.2. Możliwości rozwoju

Istnieje wiele kierunków dalszego rozwoju zaprezentowanego systemu, które mogą wyeliminować jego słabe strony oraz zwiększyć funkcjonalność.

Możliwościami rozwoju aplikacji użytkownika jest:

- automatyczne rozpoznawanie numeru portu COM, do którego podłączony jest moduł oscyloskopowy, na podstawie identyfikatorów VID i PID,
- rozdzielenie sekwencyjnie wykonywanych zadań, na poszczególne wątki. Pozwoli to zwiększyć szybkość odpowiedzi na komendy zadawane przez użytkownika oraz skróci czas odświeżania oscyloskopu,
- udostępnienie aplikacji w postaci pliku wykonywalnego .exe.

Odnosząc się do modułu oscyloskopowego, obwód drukowany nie wymaga kolejnych modyfikacji. Natomiast krokiem umożliwiającymi jego rozwój jest rozbudowa programu sterującego mikrokontrolerem:

- rejestracja badanego sygnału przed pojawieniem się zdarzenia wyzwalającego,
- dodatkowe tryby wyzwalania (np. zbocze opadające)
- dodatkowe rodzaje pracy układu akwizycji sygnału (np. singel)
- możliwość konfiguracji parametrów akwizycji (np. szybkości próbkowania)

8.3. Wnioski

Podczas realizacji projektu przedstawionego w powyższej pracy magisterskiej, zostały wysnute następujące wnioski:

- wykorzystanie języka Python ułatwia proces przetwarzania i analizy danych,
- wykorzystanie bibliotek ASF pozwala na opracowanie programu sterującego mikrokontrolerem XMEGA w krótkim czasie, bez głębszej znajomości rejestrów mikrokontrolera,
- mikrokontrolery z rodziny XMEGA udostępniają dużą ilość układów peryferyjnych o dobrych parametrach w odniesieniu do ceny,
- zastosowanie w projekcie sprzętowego interfejsu USB pozwoliło na obniżenie kosztów elementów elektronicznych, zmniejszenie obwodu drukowanego oraz zwiększenie szybkości odświeżenia oscyloskopu poprzez skrócenie czasu transmisji danych do komputera PC.

Spis listingów

2.1. Przykład wykorzystania metod biblioteki Matpltlib 19
2.2. Przykład prezentujący zalety biblioteki NumPy w stosunku do funkcji standardowych 20
3.1. Implementacja testu pakietu Matplotlib, część 1. 32
3.2. Implementacja testu pakietu Matplotlib, część 2. 33
5.1. Program mikrokontrolera, część 1. 40
5.2. Program mikrokontrolera, część 2. 41
5.3. Program mikrokontrolera, część 3. 42
5.4. Program mikrokontrolera, część 4. 43
6.1. Program główny aplikacji użytkownika, inicjalizacja. 48
6.2. Program główny aplikacji użytkownika, pętla główna. 49
6.3. Moduł gui, część 1. 50
6.4. Moduł gui, część 2. 51

Spis rysunków

1.1. Schemat ideowy podłączenia modułu oscyloskopowego do zestawu ćwiczeniowego 9
1.2. Ustawienie wartości napięcia wyjściowego układu przetwornika DAC - ramka transakcji 10
1.3. Ustawienie lub odczyt stanu na wyjściu portu bramy IO - ramka transakcji.....10
1.4. Ustawienie lub odczyt stanu na wyjściu portu bramy IO - ramka transakcji.....11
1.5. Zapis lub odczyt bajta danych z pamięci EEPROM- ramka transakcji 11
2.1. Schemat implementacji interfejsu USB przy wykorzystaniu klasy CDC 16
2.2. Schemat prezentujący relacje pomiędzy transakcją, blokiem i transferem burst.....17
3.1. Diody zabezpieczające porty IO mikrokontrolera 25
3.2. Trzy warianty funkcjonowania układu zabezpieczającego wejścia mikrokontroler27
3.4. Wyniki testu: górny wykres interfejs UART, dolny wykres interfejs USB.....29
3.3. Diagram sekwencji testu interfejsu USB 30
4.1. Schemat architektury kompletnego systemu.....35
4.2. Diagram sekwencji działania systemu 38
5.1. Schemat zasady działania układu próbkującego 39
5.2. Schemat elektryczny projektowanego modułu. 45
5.3. Rozmieszczenie elementów na płytce drukowanej, widok z góry i dołu.....45
5.4. Rzeczywisty model wykonanego modułu oscyloskopowego 46
6.1. Aplikacja użytkownika, interfejs graficzny 47
7.1. Oscylogram sygnału testowego.....53
7.2. Oscylogram dla transakcji ustawienie wartości napięcia wyjściowego.....54
7.3. Ustawienie wartości napięcia wyjściowego układu przetwornika DAC - ramka transakcji 54
7.4. Oscylogram dla transakcji ustawienia stanu na wyjściu bramy IO.....55
7.5. Ustawienie stanu na wyjściu portu bramy IO - ramka transakcji 55
7.6. Oscylogramy dla transakcji odczytu stanu na wyjściu bramy IO 56
7.7. Odczyt stanu na wyjściu portu bramy IO - ramka transakcji.....56
7.8. Oscylogram dla transakcji ustawienia stanu na wyjściu bramy IO.....57
7.9. Ustawienie stanu na wyjściu portu bramy IO - ramka transakcji 57

Spis tabel

1.1. Najważniejsze parametry wybranych oscyloskopów. 8
1.2. Długości transakcji dla poszczególnych układów peryferyjnych 10
2.1. Wersje interfejsu USB 15
2.2. Wykorzystane metody biblioteki NumPy 21
2.3. Wykorzystane metody biblioteki PySerial 21
2.4. Wykorzystane metody biblioteki time 21
3.1. Główne rodziny mikrokontrolerów XMEGA 24
3.2. Charakterystyka elektryczna układu SRDA 3.3-4 TBT 26
8.1. Kosztorys budowy modułu oscyloskopowego 59
8.2. Najważniejsze parametry wybranych oscyloskopów uwzględniająca zaprojektowany moduł 60

Bibliografia

[1] Atmel AVR4030: Atmel Software Framework - Reference Manual

[2] Atmel Product Selection Guide Brochure

[3] AVR040: EMC Design Considerations

[4] AVR1001: Getting Started With the XMEGA Event System

[5] AVR1005: Getting started with XMEGA

[6] AVR1300: Using the AVR XMEGA ADC

[7] AVR1304: Using the XMEGA DMA Controller

[8] AVR1306: Using the XMEGA Timer/Counter

[9] Francuz T. AVR Praktyczne projekty Gliwice: Wydawnictwo Helion, 2013

[10] Francuz T. AVR Układy peryferyjne Gliwice: Wydawnictwo Helion, 2014

[11] Francuz T. Język C dla mikrokontrolerów AVR Wydanie II Gliwice: Wydawnictwo Helion, 2015

[12] Hughes J. M. Real World Instrumentation with Python Sebastopol: O`Reilly, 2010

[13] McKinney W. Python for Data Analysis Sebastopol: O`Reilly, 2012

[14] Root B. V. Interactive Applications Using Matplotlib Birmingham: O`Reilly, 2015

[15] Sandro T. Matplotlib for Python Developers Birmingham: Packt Publishing 2009

[16] Specyfikacja techniczna modułu 24AA64/24LC64/24FC64

[17] Specyfikacja techniczna modułu MCP23009/MCP23S09

[18] Specyfikacja techniczna modułu MCP4921/4922

[19] Specyfikacja techniczna modułu PCF8574

[20] Specyfikacja techniczna modułu SRDA3.3-4

[21] Specyfikacja techniczna układu XMEGA AU

[22] VanderPlas J. Python Data Science Handbook Sebastopol: O`Reilly, 2014

Załączniki

Załącznik 1. Konfiguracja częstotliwości pracy magistrali SPI i I2C

Konfiguracji dokonujemy poprzez wpisanie właściwych wartości do rejestrów konfiguracyjnych kontrolera SPI i I2C układu nadrzędnego. Zakładamy, że do kontrolerów dostarczany jest sygnał taktujący o domyślnej częstotliwości, czyli 15MHz. Częstotliwość pracy magistrali SPI, określono na podstawie formuły:

$$\text{bitrate SPI} = \frac{\text{PCLK}}{\text{SPCCR}}$$

zakładając:

$$\text{PCLK} = 15\text{MHz oraz SPCCR} = 254$$

stąd:

$$\text{bitrate SPI} = \text{ok. } 59\text{kHz}$$

gdzie:

bitrate SPI – *odpowiada częstotliwości sygnału SCK*
PCLK – *częstotliwość taktowania kontrolera SPI*
SPCCR – *rejestr konfiguracyjny kontrolera SPI*

Natomiast częstotliwość pracy magistrali I2C , określono na podstawie formuły:

$$\text{bitrate I2C} = \frac{\text{PCLK}}{\text{I2CSCLH} + \text{I2CSCLL}}$$

zakładając:

$$\text{PCLK} = 15\text{MHz oraz I2CSCLH}=127 \text{ i I2CSCLL}=127$$

stąd:

$$\text{bitrate I2C} = \text{ok. } 59\text{kHz}$$

gdzie:

bitrate I2C – *odpowiada częstotliwości sygnału I2C*
PCLK – *częstotliwość taktowania kontrolera I2C*
I2CSCLH, I2CSCLL – *rejestry konfiguracyjne kontrolera I2C*

Załącznik 2. Implementacja testu interfejsu USB

Skrypt testowy został napisany w języku Python i pozwala na komunikację komputera PC z mikrokontrolerem. Zaimplementowany został także moduł test, umożliwiający weryfikację poprawności komunikacji oraz prezentację parametrów transmisji.

Listing Skrypt testowy.

```
1    import serial
2    import time
3    import test
4
5    serial_port = serial.Serial(port='COM3', timeout = 1)
6    serial_port.flushInput()
7
8    data_length = 2000
9    data        = []
10   data_sums   = []
11   time_stamps = []
12
13   transaction_trigger = bytearray(b'\xff')
14   transaction_number  = 10000
15
16   time_start = time.clock()
17   for i in range(transaction_number):
18       serial_port.write(transaction_trigger)
19       data = serial_port.read(data_length)
20       data_sums.append(sum(data))
21       time_stamps.append(time.clock() - time_start)
22
23   if test.check_corretness_mod256(data_sums, data_length):
24       test.generate_report(time_start, time_stamps)
25   else:
26       print("test failed")
```

Listing Moduł test.

```
import numpy as np
import matplotlib.pyplot as plt

def generate_report(time_start, time_stamps):
    time_stamps_minus_offset = np.subtract(time_stamps, time_start)
    time_stamps_aux = np.insert(time_stamps_minus_offset[0:-1], 0, 0)
    transactions_time = np.subtract(time_stamps_minus_offset, time_stamps_aux)
    transactions_time = np.multiply(transactions_time, 1000)

    test_parameters = {
        'trans_number': transactions_time.size,
        'std': np.std(transactions_time),
        'max': np.max(transactions_time),
        'arg_max': np.argmax(transactions_time),
        'min': np.min(transactions_time),
        'arg_min': np.argmin(transactions_time),
        'mean': np.mean(transactions_time),
        'median': np.median(transactions_time)}

    for key, value in test_parameters.items():
        test_parameters[key] = np.around(value, decimals=3)

    fig, ax = plt.subplots(1)
    textstr = "Transaction num: {trans_number} \nStd deviation: {std} [ms] \n" \
              "Max: {max} [ms]  idx: {arg_max} \nMin: {min} [ms]  idx: "       \
              "{arg_min}\nMean: {mean} [ms]\nMedian: {median} [ms]"            \
              .format(**test_parameters)

    ax.hist(transactions_time, 100)
    box_properties = dict(boxstyle='round', facecolor='grey', alpha=0.3)
    ax.text(0.58, 0.95, textstr, transform=ax.transAxes, fontsize=10,
    horizontalalignment='left', verticalalignment='top', bbox=box_properties)

    ax.set_title('Test results')
    ax.set_ylabel('Quantity')
    ax.set_xlabel('Period [ms]')

    plt.show()

def check_corretness_mod256(data, sample_num):
    data_pattern_sum = 0
    for i in range(sample_num):
        data_pattern_sum += i % 256

    data_result = np.equal(data, data_pattern_sum)
    data_correctness = all(data_result)
    return data_correctness
```