



WYDZIAŁ ELEKTRONIKI,  
TELEKOMUNIKACJI  
I INFORMATYKI

Imię i nazwisko studenta: Cezary Wieczorkowski

Nr albumu: 188584

Poziom kształcenia: studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Elektronika i telekomunikacja

Profil: Komputerowe systemy elektroniczne

## PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Projekt i budowa laboratoryjnego stanowiska sterującego szyną danych

Tytuł pracy w języku angielskim: Design and construction of a laboratory data bus control station

Opiekun pracy: dr inż. Kamil Stawiarski



## **Streszczenie**

Celem projektu było zaprojektowanie oraz uruchomienie zestawu laboratoryjnego umożliwiającego realizację ćwiczeń z sterowania szyną danych w laboratorium techniki cyfrowej. W ramach projektu dokonano analizy istniejącego stanowiska pod kątem możliwości rozbudowy oraz poprawy funkcjonalności. Następnie zaprojektowano oraz zbudowano prototyp nowego stanowiska uwzględniając wnioski z analizy poprzedniego.

**Słowa kluczowe:** Technika cyfrowa, stanowisko laboratoryjne, emulacja, szyna danych, ALU

**Dziedzina nauki i techniki, zgodne z wymogami OECD:** nauki inżynierijno-techniczne: automatyka, elektronika, elektrotechnika i technologie kosmiczne

## **Abstract**

The goal of this project was to design and construct a laboratory set that enables the performance of exercises on data bus control in a digital electronics laboratory. As part of the project, an analysis of the existing station was carried out in terms of the possibility of expansion and improvement of functionality. Then, a prototype of the new station was designed and built, taking into account the conclusions from the analysis of the previous one.

**Keywords:** Digital electronics, laboratory set, emulation, data bus, ALU

# Spis treści

<b>Wykaz ważniejszych oznaczeń i skrótów</b>	<b>7</b>
<b>1 Wstęp i cel pracy</b>	<b>9</b>
<b>2 Analiza istniejącego stanowiska</b>	<b>10</b>
2.1 Zasada działania stanowiska . . . . .	10
2.1.1 Opis stanowiska . . . . .	10
2.1.2 Elementy składowe stanowiska . . . . .	11
2.1.3 Interfejs użytkownika . . . . .	13
2.1.4 Tryby pracy stanowiska . . . . .	14
2.2 Krytyczna ocena stanowiska . . . . .	15
2.2.1 Założenia funkcjonalne . . . . .	15
2.2.2 Interfejs użytkownika . . . . .	15
2.2.3 Platforma sprzętowa . . . . .	15
<b>3 Koncepcja nowego stanowiska</b>	<b>17</b>
3.1 Założenia projektowe . . . . .	17
3.2 Projekt sprzętowy . . . . .	17
3.2.1 Wybór mikrokontrolera . . . . .	17
3.2.2 Elementy interfejsu użytkownika . . . . .	17
3.2.3 Dodatkowe peryferia . . . . .	18
3.2.4 Schemat blokowy sprzętowej części zestawu . . . . .	18
3.3 Architektura oprogramowania . . . . .	19
3.3.1 Struktura programu . . . . .	19
3.3.2 Emulacja elementów układu szyny danych . . . . .	19
3.3.3 Obsługa peryferiów sprzętowych . . . . .	20
3.3.4 Obsługa interfejsu użytkownika . . . . .	20
<b>4 Implementacja stanowiska</b>	<b>22</b>
4.1 Część sprzętowa . . . . .	22
4.1.1 Enkoder oraz wyświetlacz . . . . .	22
4.1.2 Diody LED oraz przyciski . . . . .	23
4.2 Konstrukcja prototypu . . . . .	24
4.3 Część programowa . . . . .	25
4.3.1 Obsługa diod LED oraz przełączników . . . . .	25
4.3.2 Symulacja jednostki arytmetyczno-logicznej . . . . .	26
4.3.3 Symulacja pozostałych elementów zestawu . . . . .	27
4.3.4 Obsługa interfejsu użytkownika . . . . .	30
4.3.5 Główna pętla programu . . . . .	32
<b>5 Testy stanowiska</b>	<b>33</b>
5.1 Program testujący operacje arytmetyczne . . . . .	33
5.2 Program testujący instrukcje skoku . . . . .	34
<b>6 Podsumowanie</b>	<b>35</b>

Bibliografia	36
Spis rysunków	37
Spis tabel	38

## **Wykaz ważniejszych oznaczeń i skrótów**

ALU	(ang. arithmetic logic unit) jednostka arytmetyczno-logiczna
RAM	(ang. random access memory) pamięć o dostępie swobodnym
LED	(ang. light emitting diode) dioda elektroluminescencyjna
LCD	(ang. liquid crystal display) wyświetlacz ciekłokrystaliczny
GPIO	(ang. general-purpose input/output) uniwersalne wejście/wyjście
I/O	(ang. input/output) wejście/wyjście
I2C	(ang. inter integrated circuit) szeregowy synchroniczny interfejs komunikacyjny
SDK	(ang. software development kit) zestaw narzędzi do rozwoju oprogramowania
PCI	(ang. peripheral component interconnect) interfejs komunikacyjny



# 1 Wstęp i cel pracy

Elektronika cyfrowa obecna jest w prawie każdym nowoczesnym urządzeniu. Jest to jedna z ważniejszych dziedzin elektroniki która umożliwia rozwój nowoczesnych technologii które leżą u podstaw współczesnego społeczeństwa. W procesie projektowania układów cyfrowych zawsze zachodzi potrzeba wymiany danych pomiędzy różnymi podzespołami układu. Jednym z sposobów realizacji przepływu danych jest zastosowanie szyny danych. Jest to zespół linii (przewodów) służących do przesyłania danych pomiędzy nadajnikami oraz odbiornikami do niej podłączonymi. Do szyny danych można podłączyć wiele nadajników oraz odbiorników danych jednak tylko jeden nadajnik może wysyłać dane w danym momencie, gdyż połączenie do wspólnej linii kilku nadajników może spowodować ich uszkodzenie. Zapobiega się temu stosując na wyjściach nadajników bufory trójstanowe. Układy kontrolne sterują stanem tych buforów tak aby tylko wyjście jednego nadajnika było aktywne w danym momencie, podczas gdy inne pozostają w stanie wysokiej impedancji.

Celem pracy jest budowa stanowiska laboratoryjnego do realizacji ćwiczeń z sterowaniem szyną danych w laboratorium techniki cyfrowej. To ćwiczenie ma za zadanie zapoznać studentów z sposobami realizacji komunikacji między różnymi układami cyfrowymi przy pomocy wspólnej magistrali danych. Rozwiązania tego typu są powszechnie stosowane w nowoczesnych układach cyfrowych takich jak mikrokontrolery oraz mikroprocesory umożliwiając wymianę danych pomiędzy różnymi podzespołami danego układu. W oparciu o szynę danych działa również wiele interfejsów znajdujących się w komputerach osobistych takich jak PCI czy też magistrala pamięci systemowej. Zrozumienie mechanizmów działania systemu opartego o magistralę danych pomoże studentom w zrozumieniu zasad działania bardziej skomplikowanych układów cyfrowych.

## 2 Analiza istniejącego stanowiska

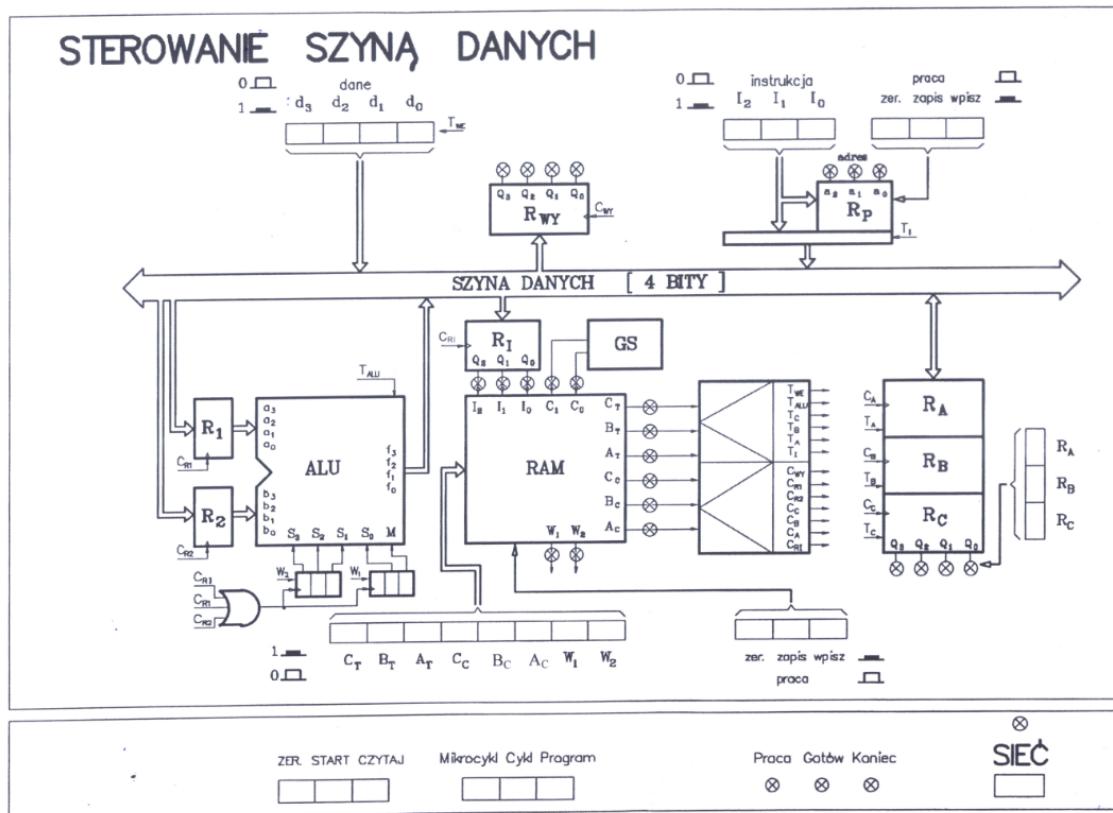
Poniższy rozdział został napisany na podstawie instrukcji obsługi zestawu [1].

## 2.1 Zasada działania stanowiska

### 2.1.1 Opis stanowiska

Zestaw laboratoryjny modeluje system komunikacji oparty na 4 bitowej szynie danych. W skład zestawu wchodzą urządzenia nadawcze i odbiorcze podłączone do szyny danych, pamięć RAM, logika sterująca procesem wymiany danych oraz elementy interfejsu użytkownika. Na rysunku 2.1 można zobaczyć płytę czołową zestawu która przedstawia jego schemat blokowy oraz interfejs użytkownika. Z pomocą interfejsu użytkownika można wprowadzić do pamięci RAM instrukcje które zostaną przez zestaw wykonane. Instrukcje te mogą realizować operacje arytmetyczne lub logiczne, zapisywać dane do rejestrów lub pobierać dane od użytkownika. Pozwala to na realizację prostych programów których efektem może być na przykład obliczanie kolejnych elementów ciągufibonacci czy wykonanie zadanej operacji matematycznej na liczbach podanych przez użytkownika.

Stanowisko laboratoryjne w pierwotnej wersji było zrealizowane w oparciu o układy logiczne z serii 74 połączone w sposób pokazany na płycie czołowej zestawu. Obecna wersja zestawu została zrealizowana w ramach pracy magisterskiej [2] i jest ona oparta o mikrokontroler Atmega 32 oraz Atmega 8 emulujące część fizycznych elementów zestawu. Mikrokontrolery te są wspomagane przez szereg układów takich jak bufory oraz przerzutniki. W dalszej części tego rozdziału zostało opisane działanie pierwotnej wersji zestawu.



Rysunek 2.1: Płyta czołowa zestawu

### 2.1.2 Elementy składowe stanowiska

#### Pamięć RAM

Pamięć ram w zestawie jest zorganizowana jako 32 słowa po 8 bitów.

$C_T$	$B_T$	$A_T$	$C_C$	$B_C$	$A_C$	$W_1$	$W_2$
-------	-------	-------	-------	-------	-------	-------	-------

Tabela 1: Symbole przypisane poszczególnym bitom słowa w pamięci RAM

Zawartość słów w pamięci RAM kontroluje pracę zestawu:

- bity  $C_T, B_T, A_T$  - wybór nadajnika na szynie
- bity  $C_C, B_C, A_C$  - wybór odbiornika na szynie
- bity  $W_1, W_2$  - wybór operacji ALU

Pamięć RAM zawiera instrukcje pracy zestawu. W celu wykonania kolejnych instrukcji należy zwiększyć o 1 obecny adres pamięci. Wykonanie jednej instrukcji z pamięci RAM nazywamy mikrocyklem.

Wejścia adresowe pamięci są połączone z rejestrem Rp (trzy najstarsze bity) oraz z generatorem Gs (dwa najmłodsze bity). Trzy najstarsze bity wyjścia danych pamięci połączone są z wejściami adresowymi dekodera 1 natomiast kolejne trzy z wejściami adresowymi dekodera 2. Dwa najmłodsze bity wyjścia danych pamięci połączone są wejściami informacyjnymi rejestrów przesuwnych instrukcji ALU.

#### Generator GS

Generator GS jest licznikiem modulo 4. Jego wyjścia połączone są z dwoma najmłodszymi wejściami adresowymi pamięci RAM. Podczas pracy generator wygeneruje kolejno wartości dwóch najstarszych bitów adresu od 00 do 11. W zależności od trybu pracy zestawu generowane są kolejne adresy pamięci do zapisu lub odczytu i wykonani jako instrukcje w mikrocyklu.

#### Dekoder sygnałów sterujących

Dekodery sygnałów sterujących to układy konwertujące liczby binarne na kod 1 z n. Dekoder 1 wytwarza sygnały sterujące nadajnikami które podawane są na wejścia sterujące przyłączaniem poszczególnych nadajników do szyny danych.

$C_T$	$B_T$	$A_T$	$T_I$	$T_A$	$T_B$	$T_C$	$T_{ALU}$	$T_{WE}$
0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	0	1	0	0	0
0	1	1	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	1

Tabela 2: Tablica prawdy dla dekodera 1

Dekoder 2 działa analogicznie do dekodera 1 ale wytwarza sygnały sterujące odbiornikami. Sygnały generowane przez dekoder 2 podawane są na wejścia zapisu poszczególnych odbiorników.

$C_C$	$B_C$	$A_C$	$C_{RI}$	$C_A$	$C_B$	$C_C$	$C_{R1}$	$C_{R2}$	$C_{WY}$
0	0	0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0
1	0	1	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	0	1

Tabela 3: Tablica prawdy dla dekodera 2

### Jednostka arytmetyczno logiczna ALU

Układ ten realizuje operacje arytmetyczne i logiczne na dwóch liczbach 4 bitowych zapisanych w rejestrach  $R_1$  oraz  $R_2$ . Rodzaj wykonywanej operacji określa pięciobitowe słowo  $M, S_3, S_2, S_1, S_0$  które jest zapisywane w dwóch 3 bitowych rejestrach przesuwnych. Na rysunku 2.1 pokazano sposób połączenia rejestrów przesuwnych w układzie. Na ich wejścia danych podane są sygnały  $W_1, W_2$  z pamięci RAM. Natomiast na wejścia zegarowe podana jest suma logiczna sygnałów  $C_{RI}, C_{R1}, C_{R2}$ . Z tych zależności wynika, że podczas projektowania instrukcji z użyciem ALU w celu ustalenia odpowiedniego kodu operacji należy przewidzieć wygenerowanie sygnałów  $C_{RI}, C_{R1}, C_{R2}$ . Tabela 4 przedstawia sposób kodowania operacji do wykonania przez ALU.

$I_2$	$I_1$	$I_0$	$C_1$	$C_0$	$C_T$	$B_T$	$A_T$	$C_C$	$B_C$	$A_C$	$W_1$	$W_2$
$X$	$X$	$X$	0	0							$M$	$S_3$
			0	1							$S_2$	$S_1$
			1	0							$X$	$S_0$
			1	1							$X$	$X$

Tabela 4: Zasada kodowania operacji do wykonania przez ALU

Jednostka arytmetyczno logiczna ma możliwość zrealizowania 16 operacji arytmetycznych oraz logicznych:

### Rejestry pomocnicze

Rejestry są układami przechowującymi dane. W zestawie występują następujące rejesty:

- $R_A, R_B, R_C$  - są to rejesty przeznaczone do wykorzystania przez użytkownika, mogą one zarówno odbierać jak i nadawać dane na szynę
- $R_1, R_2$  - przechowują argumenty operacji ALU, mogą tylko odbierać dane z szyny
- $R_{wy}$  - służy do prezentacji wyników operacji wykonywanych przez zestaw laboratoryjny, może tylko odbierać dane z szyny

Lp.	$S_3$	$S_2$	$S_1$	$S_0$	Operacje arytmetyczne ( $M = 0$ )	Operacje logiczne ( $M = 1$ )
1	0	0	0	0	$Y = 0$	$Y = \overline{R_1}$
2	0	0	0	1	$Y = R_1 \div R_2$	$Y = \overline{R_1} \wedge \overline{R_2}$
3	0	0	1	0	$Y = R_1 \times R_2 - 1$	$Y = \overline{R_1} \vee R_2$
4	0	0	1	1	$Y = R_1 + R_2$	$Y = 1$
5	0	1	0	0	$Y = R_1^2 - R_2^2$	$Y = R_1 \oplus R_2$
6	0	1	0	1	$Y = R_1 \times 2$	$Y = \overline{R_1} \iff \overline{R_2}$
7	0	1	1	0	$Y = \frac{R_1 + R_2}{2}$	$Y = R_1 \implies R_2$
8	0	1	1	1	$Y = R_1 \times R_2$	$Y = R_1 \wedge R_2$
9	1	0	0	0	$Y = R_1 \% R_2$	$Y = \overline{R_1} \implies \overline{R_2}$
10	1	0	0	1	$Y = R_1 \div 2$	$Y = \overline{R_1} \vee \overline{R_2}$
11	1	0	1	0	$Y = (R_1 \times R_2) - (R_1 + R_2)$	$Y = R_1 \vee R_2$
12	1	0	1	1	$Y = R_1 - R_2$	$Y = \overline{R_1} \oplus \overline{R_2}$
13	1	1	0	0	$Y = (R_1 - R_2)^2$	$Y = R_2$
14	1	1	0	1	$Y = R_1$	$Y = \overline{R_1} \vee \overline{R_2}$
15	1	1	1	0	$Y = R_1^2$	$Y = R_1 \iff R_2$
16	1	1	1	1	$Y = R_1 - 1$	$Y = \overline{R_1} \oplus \overline{R_2}$

Tabela 5: Lista operacji realizowanych przez ALU

- $R_I$  - przechowuje trzy najstarsze bity adresu pamięci, może tylko odbierać dane z szyny
- $R_p$  - to zespół ośmiu rejestrów przeznaczonych do przechowywania adresów pamięci RAM pod którymi znajdują się instrukcje do wykonania przez zestaw. Rejestr ten jest wykorzystywany w trybie pracy Program

### 2.1.3 Interfejs użytkownika

Na rysunku 2.1 poza schematem blokowym zestawu widzimy przyciski oraz diody LED służące do sterowania zestawem i monitorowania jego pracy.

#### Kontrola pracy zestawu

Do sterowania pracą zestawu służą następujące przyciski:

- ZER. - przycisk zerowania generatora GS
- START - przycisk rozpoczętyjący pracę zestawu
- CZYTAJ - przycisk inicjujący odczyt z klawiatury dane i kontynuację pracy zestawu
- Mikrocykl, Cykl, Program - przełączniki wyboru trybu pracy zestawu
- klawiatura dane - klawiatura do wprowadzania danych wejściowych na szynę danych

#### Zapis i odczyt danych z pamięci RAM

Do obsługi zapisu pamięci RAM służą przyciski:

- zer. - zerowanie adresu przy zapisie i odczycie pamięci RAM

- zapis/praca - wybór pomiędzy zapisem danych do pamięci RAM a pracą zestawu
- wpisz - zapis danych z klawiatury do RAM po jego wcisnięciu adres zwiększa się o 1
- klawiatura  $C_T - W_2$  - przyciski do ustawiania wartości bitowej słowa wpisywanego do pamięci RAM

### **Zapis i odczyt danych z $R_P$**

Zapis oraz odczyt danych z rejestru odbywa się podobnie jak w przypadku pamięci RAM z użyciem odpowiednich przycisków:

- klawiatura dane - klawiatura do wprowadzania danych wejściowych na szynę danych
- klawiatura instrukcja - klawiatura do wprowadzania wartości do rejestru  $R_P$
- zer. - zerowanie adresu przy zapisie i odczycie pamięci  $R_P$
- zapis/praca - wybór pomiędzy zapisem danych do pamięci  $R_P$  a pracą zestawu
- wpisz - zapis danych z klawiatury instrukcja do  $R_P$  po jego wcisnięciu adres zwiększa się o 1

### **Monitorowanie pracy zestawu**

Na płycie czołowej zestawu znajdują się również diody LED obrazujące wartości

- Adresu pamięci RAM
- Wartości słowa w pamięci RAM
- Wartości słowa w rejestrze  $R_WY$
- Wartości słowa w jednym z rejestrów  $R_A, R_B, R_C$  wybieranych przyciskami
- Adres rejestru  $R_P$

#### **2.1.4 Tryby pracy stanowiska**

Stanowisko posiada trzy tryby pracy:

- Mikrocykl - zestaw po wcisnięciu przycisku START wykona tylko jedną instrukcję z pamięci RAM, zostanie wygenerowana tylko jedna wartość w generatorze GS. Po jej wykonaniu zestaw zatrzyma się i będzie oczekiwał na ponowne wcisnięcie przycisku START.
- Cykl - zestaw po wcisnięciu przycisku START wykona cztery kolejne instrukcje z pamięci RAM. Następnie cztery kolejne zmiany wyjść w generatorze GS. Po wykonaniu czterech instrukcji zestaw zatrzyma się i będzie oczekiwał na ponowne wcisnięcie przycisku START.
- Program - po wcisnięciu przycisku START zestaw poda na wyjście rejestru  $R_P$  kolejno osiem instrukcji które zostały do niego uprzednio zapisane. Układ wygeneruje kolejne 32 stany generatora GS, zmiana wartości w rejestrze  $R_P$  nastąpi co czwartą wartość generatora GS.

Niezależnie od trybu pracy zestawu automatycznie generowane są tylko wartości generatora GS. W celu wykonania kolejnych instrukcji należy manualnie załadować wartość trzech najstarszych bitów adresu pamięci kolejnej instrukcji do rejestru  $R_I$ . W tym celu w pierwszej instrukcji każdego bloku czterech instrukcji w pamięci RAM należy umieścić wartość 000000XX. Taka wartość instrukcji odpowiada ustawnieniu rejestru  $R_I$  jako odbiornika a rejestru  $R_P$  jako nadajnika. W przypadku pracy w trybie Mikrocykl oraz Cykl wartość trzech najstarszych bitów adresu pamięci kolejnej instrukcji należy za każdym razem manualnie wprowadzić na klawiaturze instrukcji. Natomiast w trybie Program kolejne osiem wartości rejestru  $R_P$  zostanie automatycznie podanych na jego wyjście.

## 2.2 Krytyczna ocena stanowiska

### 2.2.1 Założenia funkcjonalne

Zestaw w obecnej formie pozwala na realizację prostych programów obrazujących działanie szyny danych w praktyce. Urządzenia podłączone do szyny współpracują ze sobą w sposób przemyślany pozwalają na realizowanie prostych operacji na jednostce ALU, zapis danych do rejestrów, pobieranie oraz prezentowanie danych użytkownikowi. Tryby pracy zestawu pozwalają na łatwe zaobserwowanie procesów zachodzących w zestawie podczas wykonywania programu. Tryby cykl i mikrocykl ułatwiają debugowanie programów przechodząc przez każdy cykl lub mikrocykl programu.

### 2.2.2 Interfejs użytkownika

Interfejs użytkownika zestawu opiera się na przyciskach jako urządzeniach wejściowych oraz diodach LED jako urządzeniach wyjściowych. Wizualizacja stanu poszczególnych rejestrów oraz bitów za pomocą diod LED jest czytelna i intuicyjna. W połączeniu ze schematem blokowym na płycie pozwala to w łatwy sposób zrozumieć jakie działania zestaw wykonuje w danym momencie. Możliwość podglądu adresu pamięci RAM oraz słowa pod nim zapisanego ułatwia znajdywanie błędów w programach uruchamianych na zestawie.

Wprowadzanie danych do zestawu (wartości pamięci RAM, rejestru  $R_P$  oraz danych na szynę) odbywa się za pomocą klawiatury. Jest to kilka przycisków gdzie każdy przycisk odpowiada jednemu bitowi wprowadzanego słowa. Wciśnięty przycisk odpowiada wartości 1 a wyciśnięty wartości 0. Wprowadzanie wartości danych binarnych w ten sposób jest intuicyjne i szybkie.

Pozostałe przyciski zestawu służą do kontroli jego pracy oraz kontroli procesów zapisu i odczytu wartości z pamięci RAM oraz rejestru  $R_P$ . Jest to sposób sprawdzony i stosunkowo intuicyjny. Głównym ograniczeniem tego podejścia jest trudność w dokonaniu zmian w interfejsie gdyż jego elementy są stałymi elementami płyty czołowej zestawu. Kolejnym ograniczeniem tego podejścia jest ograniczona możliwość sposobów zapisu oraz odczytu pamięci RAM oraz rejestru  $R_P$ . Zapis oraz odczyt musi się odbywać w sposób sekwencyjny co znaczco wydłuża proces w przypadku gdy zachodzi potrzeba zmiany lub odczytu wartości znajdujących się w dalszych regionach pamięci lub rejestru.

### 2.2.3 Platforma sprzętowa

W obecnej formie zestaw jest oparty o dwa mikrokontrolery Atmega 32 oraz Atmega 8. Układy te są odpowiedzialne za emulacje części zestawu. Pierwszy układ pełni rolę pamięci RAM, jednostki ALU,

rejestru  $R_I$ ,  $R_P$  oraz generatora GS. Podejście emulacyjne jest po części wymuszone ograniczoną dostępnością w handlu dedykowanych układów scalonych takich jak jednostki ALU ze względu na ich znikomą popularność nowych projektach. Drugi układ pełni rolę dekodera sygnałów sterujących. Pozostałe rejesty znajdujące się w zestawie zrealizowane zostały za pomocą układów LS74125 - cztery przerzutniki typu D. Komponenty zestawu wymieniają między sobą danę przy pomocy fizycznej szyny danych, wymusza to zastosowanie układów buforów trójstanowych na wyjściach nadajników.

Obecne rozwiązanie sprzętowe w powstało z myślą o zachowaniu kompatybilności z płytą czołową z pierwszej części zestawu [2]. Dużym problemem tego zestawu jest brak elastyczności w rozbudowie oraz naprawie błędów. Duża część elementów zestawu została zrealizowana na fizycznych układach scalonych co znaczaco utrudnia modyfikacje projektu. Rozbitie komponentów emulowanych pomiędzy dwa mikrokontrolery również nie jest optymalnym rozwiązaniem gdyż komplikuje to proces rozwoju oprogramowania oraz naprawy potencjalnych błędów. Dodatkową wadą obecnej platformy sprzętowej jest trudność modyfikacji interfejsu użytkownika spowodowana ciasną integracją elementów płyty czołowej ze sposobem działania zestawu.

### **3 Koncepcja nowego stanowiska**

#### **3.1 Założenia projektowe**

Po przeanalizowaniu poprzedniego zestawu laboratoryjnego wypracowano następujące założenia dla projektu nowego zestawu:

- Zestaw musi zachować bloki funkcjonalne oraz zasadę działania poprzedniego zestawu.
- Zestaw ma zachować tryby pracy: cykl, mikrocykl oraz program.
- Zestaw musi emulować komponenty rzeczywistego układu szyny danych.
- Zestaw musi umożliwiać łatwą naprawę błędów w jego funkcjonowaniu oraz rozbudowę o nowe funkcjonalności.
- Zestaw musi zachować przejrzysty sposób wizualizacji stanu poszczególnych komponentów.
- Zestaw musi posiadać uniwersalny interfejs użytkownika pozwalający na łatwą obsługę zestawu oraz późniejsze jego modyfikacje.

#### **3.2 Projekt sprzętowy**

##### **3.2.1 Wybór mikrokontrolera**

Ze względu na wybrane podejście symulacji programowej działania układu szyny danych, mikrokontroler jest centralnym elementem budowanego układu. W celu zapewnienia możliwości dalszej rozbudowy oraz wprowadzania poprawek do układu musi być to nowoczesna i łatwo dostępna platforma. Istniejąca wersja zestawu jest oparta o mikrokontroler Atmega 8 oraz Atmega 32. Są to urządzenia pracujące z częstotliwością 16 MHz oraz posiadające odpowiednio 1 Kb oraz 2 Kb pamięci RAM [3] [4]. Większość mikrokontrolerów dostępnych obecnie na rynku oferuje znaczco większe możliwości.

Jako mikrokontroler do budowy zestawu wybrano układ RP2040 firmy Rassbery Pi [5] obecny na płytce rozwojowej Rassbery Pi Pico. Jest to układ oparty o architekturę ARM Cortex M0+, posiada 264 Kb pamięci RAM oraz pracuje z częstotliwością 133 MHz. Układ posiada duży zapas zasobów sprzętowych co pozwoli na szerokie możliwości rozbudowy zestawu w przyszłości. Dodatkowo dostępność układu na płytce rozwojowej uproszcza proces budowy sprzętowej części zestawu gdyż płytka ta zawiera wszystkie elementy niezbędne do pracy mikrokontrolera.

##### **3.2.2 Elementy interfejsu użytkownika**

Interfejs użytkownika zestawu powinien umożliwiać łatwą obsługę zestawu oraz umożliwiać łatwe wprowadzenie zmian w jego strukturze. Takie wymagania najlepiej spełni interfejs oparty o wyświetlacz wyświetlający menu z opcjami do wyboru oraz interfejs do nawigacji pomiędzy nimi. Jako wyświetlacz zdecydowano się zastosować wyświetlacz znakowy LCD 20x4. Jest to wyświetlacz oparty o popularny układ sterujący HD44780 [6]. Dzięki prostocie obsługi oraz długiej obecności na rynku tego typu wyświetlacze są wspierane przez bardzo szeroką gamę oprogramowania.

W celu nawigacji po menu zdecydowano się zastosować enkoder kwadraturowy. Jest to element generujący impulsy na podstawie ruchu obrotowego. Pozwala on na stwierdzenie kierunku oraz prędkości obrotu. Tego typu urządzenie stosowane są w układach sterowania pracą silników lub jako elementy wejściowe interfejsu użytkownika. Enkodery stosowane jako element nawigacyjny

często posiadają wbudowany przycisk [7] dzięki czemu obracając enkoderem można wybierać opcje menu a naciskając go można potwierdzać wybór.

Dodatkowo w celu zapewnienia wizualizacji stanu elementów podczas pracy zestawu zdecydowano się ja w oryginalne zastosować diody LED do wizualizacji stanu adresów oraz słowa pamięci RAM, stanu rejestru  $R_{WY}$  oraz wybranego rejestru ( $R_A$ ,  $R_B$ ,  $R_C$ ).

Jako elementy do wprowadzania danych binarnych zdecydowano się zachować przełączniki jak w oryginalnym zestawie. Dla uproszczenia obsługi zestawu dane do pamięci RAM oraz rejestru  $R_P$  będą wprowadzane za pomocą jednego zestawu ośmiu przełączników natomiast dane użytkownika na szynę będą wprowadzana za pomocą osobnych czterech przełączników.

### 3.2.3 Dodatkowe peryferia

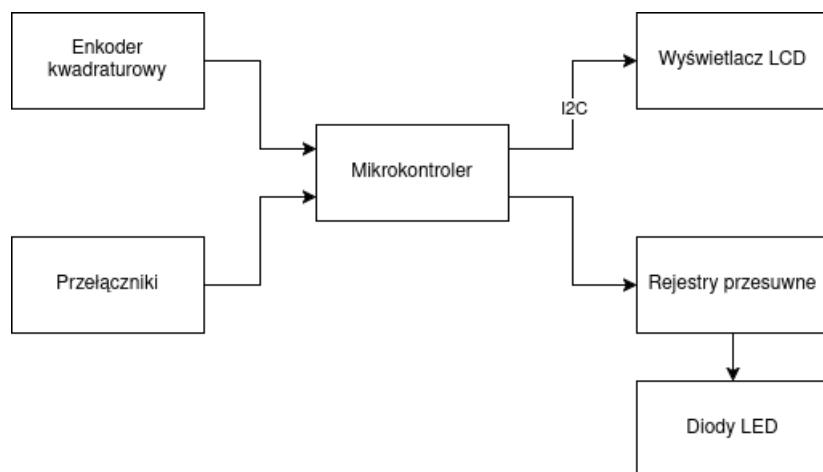
Sumując liczbę GPIO mikrokontrolera potrzebną do obsługi wybranych peryferiów otrzymujemy 43. Mikrokontroler RP2040 posiada 30 wyprowadzeń GPIO [5] co oznacza, że nie jest możliwe bezpośrednie podłączenie wszystkich peryferiów. W celu obsługi wszystkich peryferiów konieczne jest zastosowanie dodatkowych układów scalonych rozszerzających ilość dostępnych GPIO.

Jednym z prostszych tego typu układów jest rejestr przesuwny. Jest to układ który posiada wejście zegarowe oraz wejście danych. Po każdym zboczu zegara dane z wejścia danych są przesuwane do kolejnej komórki rejestru. Rejestry przesuwne posiadają wyjście danych z ostatniej komórki rejestru. Dzięki temu można połączyć kilka rejestrów kaskadowo. Jednym z takich układów jest 74HC595 [8]. Jest to 8 bitowy rejestr przesuwny z wyjściem równoległy. Rejestry te zastosowano do sterowania diodami LED. Użyto trzech układów ze względu na konieczność wysterowania 22 diod.

W celu redukcji ilości wyprowadzeń GPIO potrzebnych do obsługi wyświetlacza LCD zdecydowano się zastosować ekspander I/O PCF8574 [9]. Jest to układ który pozwala na sterowanie 8 bitowym portem wejścia/wyjścia za pomocą interfejsu I2C. Dzięki temu obsługa wyświetlacza zajmuje tylko dwa wyprowadzenia GPIO potrzebne do obsługi interfejsu I2C.

### 3.2.4 Schemat blokowy sprzętowej części zestawu

Na rysunku 3.1 przedstawiono schemat blokowy sprzętowej części zestawu.



Rysunek 3.1: Schemat blokowy sprzętowej części zestawu

### 3.3 Architektura oprogramowania

#### 3.3.1 Struktura programu

Wybrany mikrokontroler RP2040 posiada dostarczane przez producenta SDK dla języka C/C++. W skład zestawu wchodzi szereg bibliotek pozwalających na obsługę peryferiów mikrokontrolera. Są to biblioteki niskopoziomowe o niewielkim stopniu abstrakcji. Znaczącym ograniczeniem SDK jest niewielka ilość bibliotek do obsługi urządzeń dołączanych do mikrokontrolera. W związku z tym do napisania oprogramowania zestawu zdecydowano się wykorzystać framework Arduino [10]. Jest to popularne środowisko programistyczne wspierające wiele mikrokontrolerów oparte o język C++. Framework ten dostarcza wysokopoziomowe biblioteki do obsługi peryferiów mikrokontrolera. Ze względu na popularność Arduino framework dostępny jest wiele bibliotek z nim kompatybilnych napisanych przez społeczność.

Oprogramowanie zestawu podzielono na kilka modułów odpowiadających za poszczególne funkcjonalności zestawu:



Rysunek 3.2: Schemat blokowy oprogramowania zestawu

#### 3.3.2 Emulacja elementów układu szyny danych

Wybrane podejście symulacji programowej układu szyny danych wymaga zaimplementowania emulacji poszczególnych elementów układu:

- Pamięci RAM
- Jednostki ALU wraz z komponentami pomocniczymi
- Mechanizmy sterujące pracą układu
- Rejestru  $R_{WY}$
- Rejestru  $R_P$
- Rejestru  $R_A$
- Rejestru  $R_B$
- Rejestru  $R_C$
- Rejestru  $R_I$

Rejestry oraz pamięć RAM są strukturami przechowującymi dane. Ich implementacje programowo można rozwiązać stosując zmienne które podobnie jak rejesty przechowują dane. Pamięć RAM oraz  $R_P$  przechowują więcej niż jedną wartość dlatego ich reprezentacje w programie muszą być tablicami zmiennych o odpowiedniej długości.

Symulacja jednostki arytmetyczno-logicznej wymaga implementacji operacji przez nią wykonywanych. Operacje te można zaimplementować z użyciem operatorów logicznych oraz arytmetycznych obecnych w języku C++. Kolejną częścią jednostki ALU jest system dekodowania słowa sterującego wyborem operacji oraz sposób pobierania wartości słowa sterującego z kolejnych instrukcji programu. Implementacja tego elementu wymaga zaimplementowania funkcji dekodującej.

Ostatnią częścią konieczną do zaimplementowania jest system sterujący pracą układu. Kontrola pracy w pojedynczym mikrocyklu polega na zdekodowaniu pojedynczej instrukcji oraz wykonania na jej podstawie odpowiedniej operacji. Sterowanie pracą układu w trybach program oraz cykl można oprzeć na maszynie stanów która na podstawie aktualnego stanu układu oraz dokonuje odpowiednich przejść pomiędzy stanami.

### 3.3.3 Obsługa peryferiów sprzętowych

Oprogramowanie układu musi obsługiwać następujące peryferia:

- Wyświetlacz LCD
- Enkoder kwadraturowy
- Przełączniki
- Układy 74HC595 - diod LED

Przełączniki są podłączone bezpośrednio do końcówek GPIO mikrokontrolera więc ich status można odczytać za pomocą funkcji dostarczonych przez framework Arduino. Diody LED są sterowane za pomocą układów 74HC595. Sterowanie tymi układami wymaga generacji sygnału zegarowego oraz wystawiania kolejnych bitów danych na wejście danych układu zgodnie z taktem zegara. Zdecydowano się użyć biblioteki implementującej ten protokół w środowisku Arduino framework [11].

Obsługa wyświetlacza LCD odbywa się poprzez ekspander I/O PCF8574. Komunikacja z ekspanderem odbywa się za pomocą interfejsu I2C. Do obsługi wyświetlacza wykorzystano bibliotekę kompatybilną z frameworkm Arduino - LiquidCrystal PCF8574 [12]. Enkoder kwadraturowy jest obsługiwany za pomocą biblioteki wybranej do implementacji systemu menu opisanej w dalszej części dokumentu.

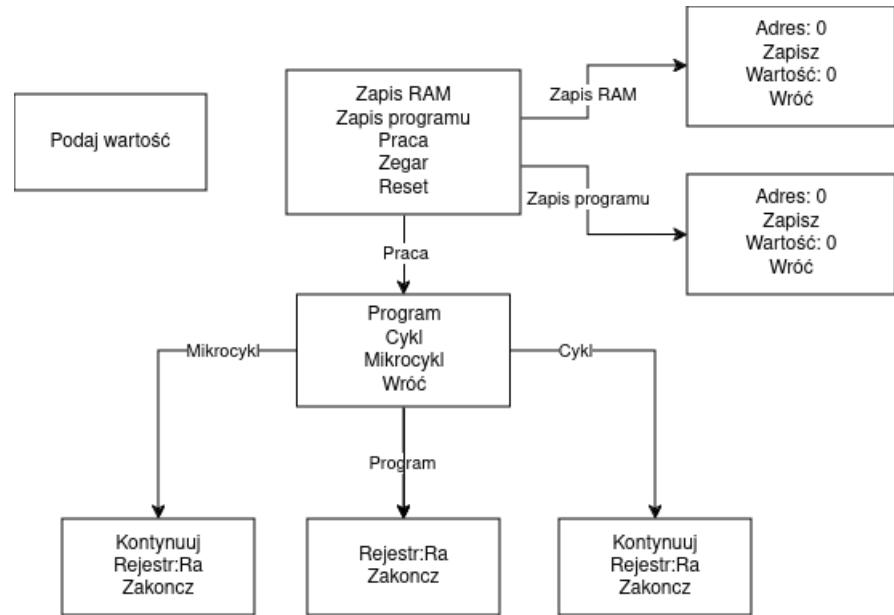
### 3.3.4 Obsługa interfejsu użytkownika

Interfejs użytkownika powinien udostępniać użytkownikowi następujące funkcje:

- Wybór trybu pracy zestawu
- Zapis oraz odczyt RAM oraz  $R_P$
- Wprowadzanie danych na szynę
- Wizualizacja stanu elementów układu
- Wybór aktualnie wyświetlonego rejestru

Wizualizacja stanów elementu układów jest realizowana za pomocą diod LED, pozostałe funkcjonalności należy zaimplementować w oparciu o wyświetlacz LCD oraz enkoder. Zdecydowano się

zastosować interfejs użytkownika w formie menu. Menu jest to element interfejsu użytkownika składający się z szeregu opcji pomiędzy którymi użytkownik może nawigować. Opcje te po wybraniu mogą prowadzić do kolejnych poziomów menu lub wykonywać konkretne akcje. W tym przypadku do menu będzie wyświetlane na wyświetlaczu LCD a nawigacja będzie odbywać się poprzez obrót oraz naciśnięcie enkodera. Na rysunku 3.3 przedstawiono strukturę menu dla projektowanego zestawu.



Rysunek 3.3: Diagram menu interfejsu użytkownika

Pierwszym menu widocznym dla użytkownika jest ogólny ekran nawigacyjny prowadzący do pozostałych opcji. Z jego poziomu można przejść do zapisu/odczytu pamięci RAM oraz  $R_P$ , wyboru trybu pracy zestawu oraz zresetować zestaw. Z poziomu ekranu wyboru trybu pracy można rozpoczęć pracę zestawu w jednym z trzech dostępnych trybów. Podczas pracy w dowolnym trybie użytkownik ma możliwość natychmiastowego zakończenia pracy zestawu oraz wyboru aktualnie wyświetlonego rejestru. W trybach cykl oraz mikrocykl znajduje się dodatkowa opcjami “Kontynuuj”, po wciśnięciu której zestaw wykonuje kolejny cykl/mikrocykl. Ostatni dostepny ekran jest wywoływany w przypadku gdy program działający na zestawie w danej instrukcji czyta dane z klawiatury “Dane”. Program jest zatrzymywany a użytkownik jest proszony o wprowadzenie danych. Po zatwierdzeniu przez użytkownika zestaw kontynuuje pracę.

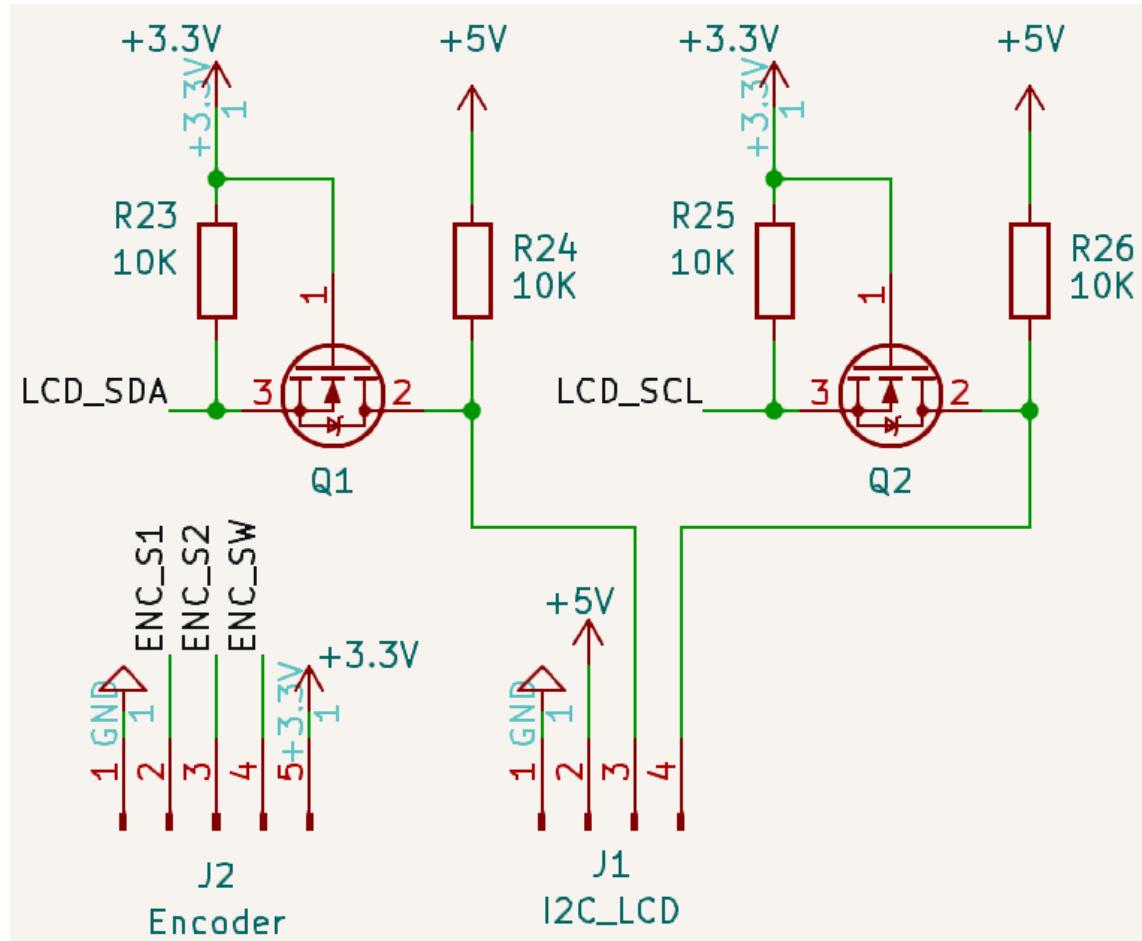
Do programowej implementacji systemu menu zdecydowano się wykorzystać bibliotekę Arduino menu [13]. Jest to biblioteka pozwalająca na implementację hierarchicznego systemu menu w prosty sposób. Posiada ona szereg wbudowanych peryferyjnych pozwalających na łatwą implementację opcji menu pełniących funkcje takie jak: wywołanie przypisanej funkcji, zmiana wartości zmiennej lub przejście do kolejnego poziomu menu. Dużą zaletą tej biblioteki są wbudowane sterowniki dla wielu urządzeń wejścia oraz wyjścia. Biblioteka bez natywnie wspiera obsługę enkodera kwadraturowego oraz wyświetlacza LCD za pomocą biblioteki LiquidCrystal PCF8574 [12].

## 4 Implementacja stanowiska

### 4.1 Część sprzętowa

#### 4.1.1 Enkoder oraz wyświetlacz

Sposób podłączenia wyświetlacza LCD oraz enkodera został przedstawiony na rysunku 4.1.



Rysunek 4.1: Schemat połączeń enkodera oraz wyświetlacza

Obsługa modułu enkodera wymaga podłączenia zasilania, sygnałów A i B - wyjść impulsów enkodera oraz sygnału SW - wyjścia przycisku. Sygnały A, B oraz SW zostały podłączone bezpośrednio do wyprowadzeń GPIO mikrokontrolera. Obsługa wyświetlacza wymaga podłączenia zasilania oraz sygnałów SCL i SDA dla interfejsu I2C używanego do komunikacji z ekspanderem I/O PCF8574 obsługującym wyświetlacz. Pierwotnie wyświetlacz oraz ekspander zasilono tak z napięcia 3,3 V. Podczas wstępnego uruchomienia zestawu zauważono, że obraz na wyświetlaczu jest niewyraźny. Było to spowodowane zbyt niskim napięciem zasilania dla poprawnego funkcjonowania podświetlenia wyświetlacza. Po zwiększeniu napięcia zasilania do 5 V problem ustąpił. Zmiana wartości napięcia zasilania wymagała zastosowania konwerterów poziomów logicznych na liniach SCL i SDA, ponieważ mikrokontroler RP2040 pracuje z napięciem 3,3 V. Zastosowany konwerter poziomów logicznych jest oparty o tranzystor MOSFET typu N. Ważną właściwością tego typu konwertera jest możliwość konwersji poziomów w obu kierunkach która jest w tym wypadku konieczna gdyż linia SDA jest linią dwukierunkową.

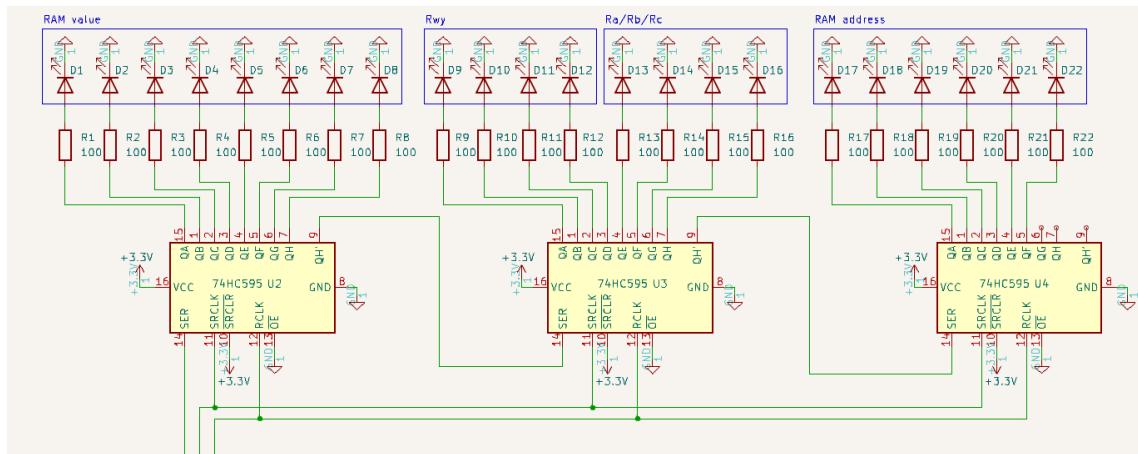
#### 4.1.2 Diody LED oraz przyciski

Diody LED są sterowane przez trzy rejestyry przesuwne 74HC595. Zgodnie z notą aplikacyjną [8] w celu sterowania układem należy wykonać następujące połączenia:

- VCC - zasilanie 3,3 V
- GND - masa
- SER - linia danych, w pierwszym układzie podłączona do wyprowadzenia GPIO mikrokontrolera natomiast w kolejnych układach podłączona do wyjścia danych poprzedniego układu
- $\overline{SRCLR}$  - linia resetu podłączona na stałe do napięcia zasilania
- SRCLK - linia zegara podłączone do GPIO mikrokontrolera
- RCLK - linia zegara rejestrów wyjściowych podłączone do GPIO mikrokontrolera
- $\overline{OE}$  - wyłączenie wyjścia podłączone na stałe do masy

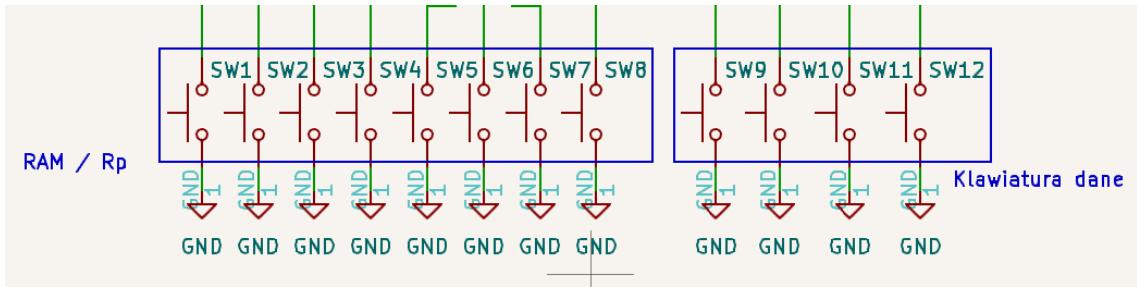
Na wyjściu układu w stanie wysokim pojawi się napięcie 3,3 volta, zastosowane diody LED charakteryzuje napięcie przewodzenia około 2 V. Charakterystyka zależności prądu od napięcia diody rośnie bardzo szybko po przekroczeniu napięcia przewodzenia. W celu ograniczenia prądu przepływającego przez diodę stosujemy szeregowy rezystor. Wartość tego rezystora obliczamy z następującego wzoru:

$$R = \frac{V_{CC} - V_{LED}}{I_{LED}} = \frac{3,3V - 2V}{10mA} = 100\Omega \quad (4.1)$$



Rysunek 4.2: Schemat połączeń rejestrów wraz z diodami LED

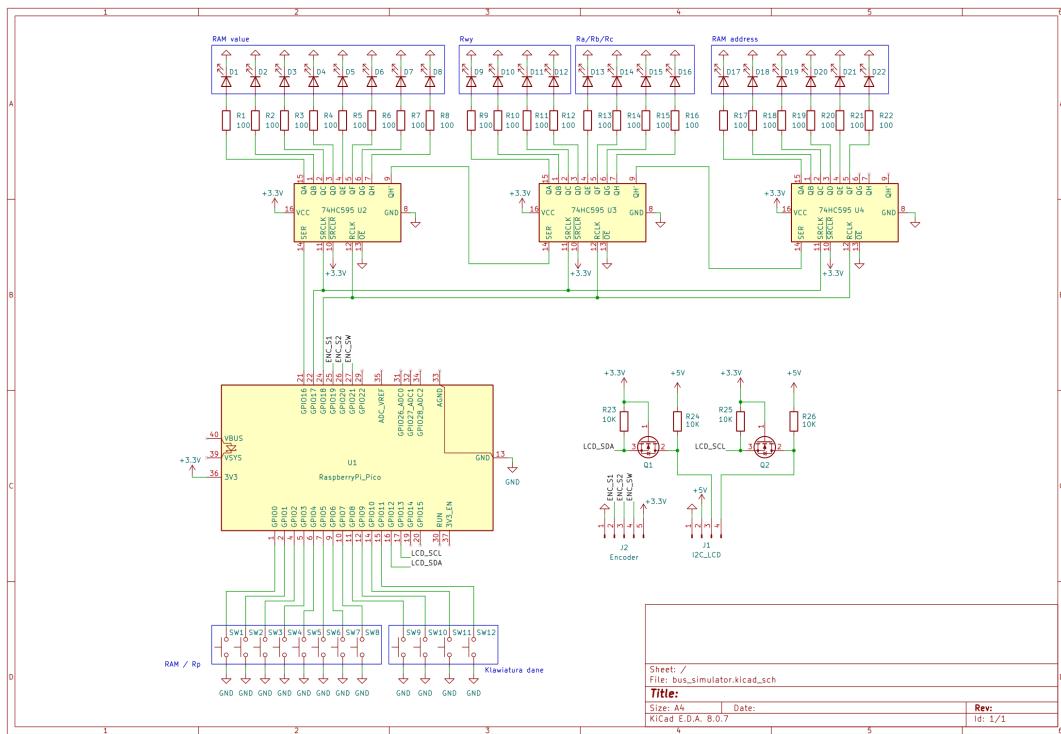
Przełączniki zostały podłączone bezpośrednio do wyprowadzeń GPIO mikrokontrolera. Włączenie przycisku powoduje zwarcie linii GPIO z masą co zostanie odczytane jako stan niski na danym wyprowadzeniu. Stan wysoki w przypadku rozwarcia przycisku jest wymuszany przez wbudowane w port wejściowy mikrokontrolera rezystory podciągające do napięcia zasilania.



Rysunek 4.3: Schemat połączeń przycisków

## 4.2 Konstrukcja prototypu

Poniżej przedstawiono pełen schemat elektryczny zestawu. Na podstawie schematu został wykonany prototyp.

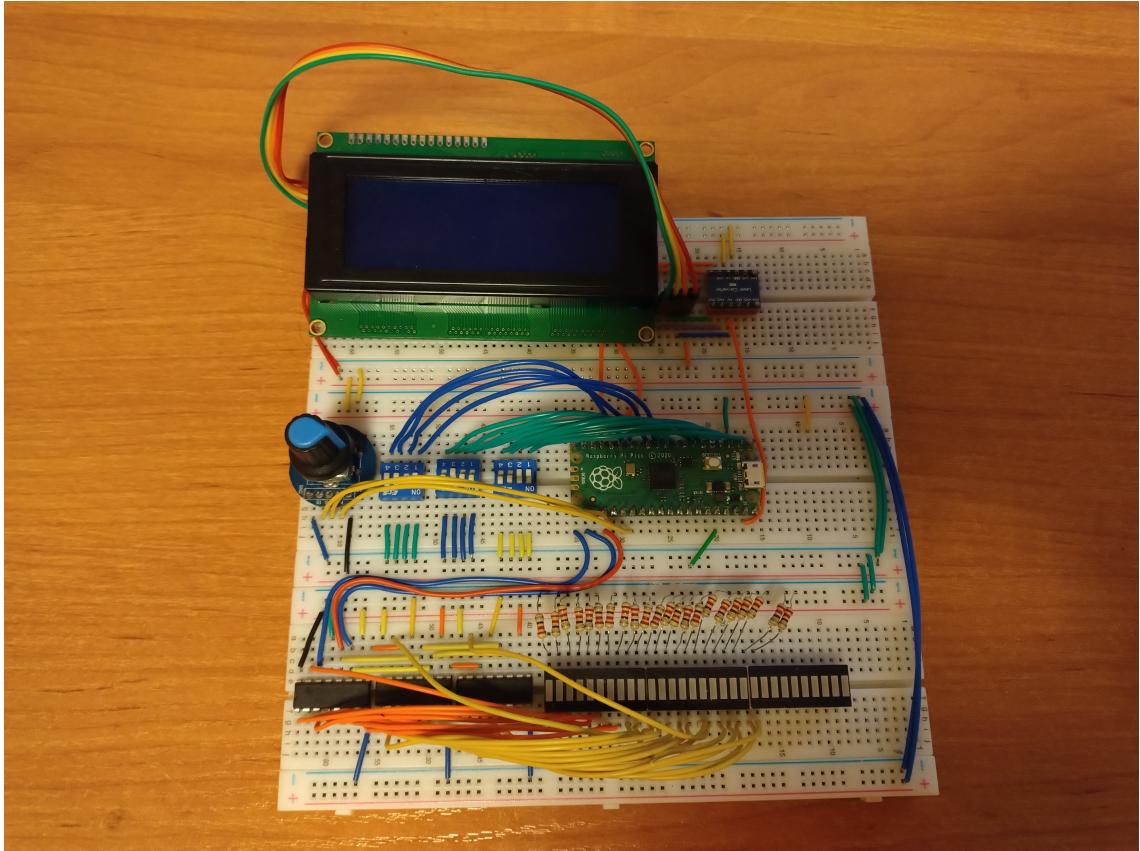


Rysunek 4.4: Schemat elektryczny zestawu

Prototyp został skonstruowany na płytce stykowej. Jest to płytka wykonana z plastiku z otworami w których umieszczane są nóżki elementów lub przewody. W środku plastikowej konstrukcji znajdują się blaszki które łączą się z elementami wprowadzonymi do otworów. W użytej płytce stykowej cztery sąsiednie otwory przypadają na jedną blaszkę. Takie rozwiązanie pozwala na szybkie modyfikacje konstruowanego oraz ułatwia testowanie budowanego układu elektrycznego. Jedną z wad układów na płytach stykowych są duże pojemności pomiędzy połączeniami wynikające z konstrukcji płytki. W przypadku budowanego zestawu nie stanowi to problemu gdyż nie występują w nim sygnały o wysokich częstotliwościach.

W konstrukcji wykorzystano trzy połączone ze sobą płytki stykowe. Na pierwszej umieszczono

mikrokontroler, przełączniki oraz enkoder. Na drugiej umieszczono rejestrzy przesuwne wraz z diodami LED. Na trzeciej umieszczono wyświetlacz LCD wraz z konwerterem poziomów logicznych.



Rysunek 4.5: Zestaw wykonany na płytce stykowej

### 4.3 Część programowa

#### 4.3.1 Obsługa diod LED oraz przełączników

Do obsługi rejestrów przesuwnych oraz przełączników została napisana klasa `UserIO`. Klasa ta posiada metody ustawiania stanu diod LED oraz odczytu stanu przełączników. Konstruktor klasy inicjalizuje wyrowadzenia GPIO do których połączone są przyciski jako wejścia oraz inicjalizuje obiekt `shift_register` klasy `ShiftRegister` który obsługuje rejestrzy przesuwne.

Do odczytu przycisków służy metoda `read_all_buttons` która zwraca stan wszystkich przycisków w postaci 16 bitowej liczby.

```
1  uint16_t userIO::read_all_buttons(){
2      uint16_t button_data = 0;
3      for (int i = 0; i < 12; i++){
4          button_data |= (digitalRead(buttons[i]) << i);
5      }
6      return button_data;
7 }
```

Wartość jest odczytywana w pętli wykonywanej dla każdego przycisku. W każdej iteracji odczytywany jest stan danego przycisku który jest następnie przesuwany o ilość bitów odpowiadającą numerowi przycisku otrzymana wartość poddawana bitowej sumie logicznej z wartością zmiennej zwracanej przez funkcję. W ten sposób otrzymujemy 16 bitową liczbę w której każdy bit odpowiada

stanowi jednego przycisku. Funkcje `read_user_input_buttons` oraz `read_data_input_buttons` odczytują odpowiednio dane z klawiatury wprowadzania danych na szynę oraz z klawiatury wprowadzania danych do RAM/ $R_P$ . Obydwie funkcje używają funkcji `read_all_buttons` do odczytu stanu przycisków a następnie wydobywają z odczytanej wartości odpowiednie bajty danych.

Obsługa diod LED odbywa się za pomocą metody `render_led`.

```

1   void userIO::render_led(const bus_cpu_status status){
2       uint8_t led_values[REG_NO] = {0, 0, 0};
3       led_values[0] = status.RAM_value;
4       led_values[1] = status.Rwy;
5       switch (displayed_register){
6           case Ra:
7               led_values[1] |= (status.Ra << 4);
8               break;
9           case Rb:
10              led_values[1] |= (status.Rb << 4);
11              break;
12           case Rc:
13               led_values[1] |= (status.Rc << 4);
14               break;
15           default:
16               led_values[1] |= (status.Ra << 4);
17               break;
18       }
19       led_values[2] = status.RAM_address;
20       shift_register.setAll(led_values);
21   }

```

Funkcja ta przyjmuje jako argument strukturę `bus_cpu_status` która zawiera wartości rejestrów, słowa pamięci RAM oraz adresu pamięci RAM. Przekazane wartości są przypisywane do tablicy w odpowiedniej kolejności tak aby wyświetliły się na odpowiednich diodach LED. W tej funkcji realizowane jest również przełączanie wyświetlanych rejestrów. W zależności od wartości zmiennej `displayed_register` na diodach LED wyświetlana jest wartość odpowiedniego rejestru. Na koniec wartości z tablicy są przekazywane do metody `setAll` obiektu `shift_register` która ustawia wartości w rejestrach przesuwnych.

#### 4.3.2 Symulacja jednostki arytmetyczno-logicznej

Symulacja jednostki arytmetyczno-logicznej została zrealizowana w klasie `ALU`. Klasa ta posiada metody realizujące zadaną operację oraz ustawiania kodu operacji. Metoda `set_opcode` ustawia kod operacji. W celu poprawnego ustawienia kodu powinna ona zostać wywołana trzy razy.

```

1   void alu::set_opcode(const uint8_t _opcode){
2       if(opcode_set_count == 2){
3           opcode = opcode << 1;
4           opcode |= _opcode;
5           opcode_set_count = 0;
6       }
7       else{
8           opcode = opcode << 2;
9           opcode |= _opcode;
10      }
11      opcode_set_count++;
12  }

```

Podczas pierwszych dwóch wywołań metoda kod operacji jest przesuwany o dwa bity w lewo, a następnie do dwóch najmłodszych bitów zostają zapisane dwa najmłodsze bity kodu przekazanego jako argument funkcji. W trzecim wywołaniu funkcji kod operacji jest przesuwany o jeden bit w lewo, a następnie do najmłodszego bitu zostaje przepisana wartość najmłodszego bitu kodu przekazanego do argumentu funkcji. W ten sposób uzyskujemy pełen pięciobitowy kod operacji.

Metoda `calculate` oblicza wynik operacji zadanej kodem operacji. Jako argumenty funkcja przyjmuje dwie liczby na których ma zostać wykonana operacja. Funkcja na początku rozpatruje najstarszy bit kodu operacji. W zależności od jego wartości wykonana zostanie operacja logiczna lub arytmetyczna. Następnie funkcja wykonuje odpowiednią operację zależnie od pozostałych czterech bitów kodu operacji. Ze względu na 4 bitową architekturę zestawu oraz brak implementacji bitu przeniesienia w oryginalnym zestawie funkcja zwraca tylko cztery najmłodsze bity wyniku operacji.

#### 4.3.3 Symulacja pozostałych elementów zestawu

Symulacja pozostałych elementów zestawu została zrealizowana w klasie CPU. Klasa ta implementuje funkcje symulujące działanie zestawu oraz szereg funkcji pomocniczych służących do sterowania pracą zestawu, ustawiania jego parametrów czy odczytu jego stanu. W klasie tej jako zmienne prywatne zdefiniowane są wszystkie rejestrów wchodzących w skład zestawu.

```

1   class bus_cpu{
2     private:
3       uint8_t Ra;
4       uint8_t Rb;
5       uint8_t Rc;
6       uint8_t Rwy;
7       uint8_t R1;
8       uint8_t R2;
9       uint8_t Ri;
10      uint8_t Ri_tmp;
11      uint8_t Gs;
12
13      uint8_t Rp_address;
14
15      std::array<uint8_t, RAM_SIZE> RAM;
16      std::array<uint8_t, PROGRAM_SIZE> Rp;
17
18      alu ALU;

```

Pomimo, że rejestrów są czterobitowe zostały zdefiniowane jako ośmiobitowe liczby całkowite bez znaku. Jest to spowodowane tym, że w języku C++ nie ma typów danych o mniejszej niż 8 liczbie bitów. Zdefiniowano także pamięć RAM oraz rejestr  $R_P$  jako tablice o odpowiednich rozmiarach. Klasa posiada również obiekt klasy `ALU` symulującą jednostkę arytmetyczno-logiczną.

Podstawową funkcją klasy jest funkcja `process_microcycle` która realizuje wykonuje jeden mikrocykl. W pierwszej kolejności na podstawie wartości rejestrów  $R_I$  oraz wartości generatora stanów  $G_S$  obliczany jest adres w pamięci RAM z którego odczytana zostanie instrukcja. Następnie z odczytanej instrukcji wydobywane są bity operacji dla ALU, kod transmitema oraz odbiornika.

```

1   ucycle_status bus_cpu::process_microcycle(){
2     uint8_t address = calculate_address();
3     uint8_t instruction = RAM[address];
4     uint8_t alu_opcode = instruction & 0b00000011;
5     uint8_t rx_ctrl = (instruction & 0b00011100) >> 2;
6     uint8_t tx_ctrl = (instruction & 0b11100000) >> 5;

```

```

7     uint8_t* transmiter;
8     uint8_t* receiver;
9     uint8_t alu_result = 0;

```

Symulacja transmisji danych na szynie odbywa się na zasadzie wskaźników. W zależności od wartości kodu transmitera oraz odbiornika do wskaźników **transmiter** oraz **receiver** przypisywane są odpowiednie adresy odpowiednich zmiennych reprezentujących odbiorniki oraz nadajniki. W przypadku gdy odbiornikiem jest  $R_I$ ,  $R_1$  lub  $R_2$  dodatkowo ustawiany jest kod operacji dla ALU. Na koniec wskaźniki wartość pod adresem transmitera jest przypisywana pod adres odbiornika.

W przypadku gdy nadajnikiem jest klawiatura wprowadzania danych na szynę funkcja zwraca kod **NEED\_INPUT** który jest interpretowany przez maszynę stanów sterującą wykonaniem programu. Kiedy dane zostaną pobrane przez użytkownika ustawiona zostaje flaga **user\_input\_ready** która powoduje poprawne wykonanie transmisji przy kolejnym wywołaniu funkcji przetwarzania mikrocyklu.

```

1     case 5:
2         if(!user_input_ready){
3             return NEED_INPUT;
4         }
5         transmiter = &user_input;
6         user_input_ready = false;
7         break;

```

Kiedy funkcja zakończy przetwarzanie mikrocyklu wartość generatora GS jest inkrementowana. Jeżeli wartość generatora przekroczy wartość 3 zwiększyła się jest adres rejestru  $R_P$  a generator jest resetowany. Wartość zmiennej  $R.I$  jest ustawiana na wartość zmiennej  $R.I_{tmp}$ . Wartość rejestru  $R_I$  musi pozostać stała przez okres trwania cyklu (4 mikrocykle) ponieważ jest ona używana do obliczenia adresu pamięci RAM z którego odczytywana jest instrukcja. Po przepeleniu się generatora GS można zmienić wartość w rejestrze  $R_I$ .

```

1     Gs++;
2     if(Gs > 3){
3         Rp_address++;
4         Gs = 0;
5         ALU.clear_opcode();
6         Ri = Ri_tmp;
7     }
8     return SUCCESS;
9 }

```

Funkcja **schedule\_execution** jest funkcją sterującą pracą zestawu. Funkcja ta jest wywoływana w głównej pętli programu. Kontrola pracy zestawu odbywa się za pomocą maszyny stanów:

- **EXECUTION** - zestaw wykonuje mikrocykl
- **CYCLE\_DONE** - zakończyło się wykonywanie cyklu lub mikrocyklu
- **USER\_INPUT** - zestaw oczekuje na wprowadzenie danych przez użytkownika
- **DONE** - zestaw zakończył pracę
- **READY** - zestaw jest gotowy do rozpoczęcia pracy
- **EXCEPTION** - wystąpił błąd

Kiedy zestaw jest w stanie EXECUTION funkcja wywołuje funkcję `process_microcycle` która wykonuje mikrocykl. W przypadku pracy w trybie Program funkcja implementuje dodatkowy mechanizm opóźniający wykonywanie kolejnych mikrocykli. Jest to konieczne ze względu na dużą prędkość mikrokontrolera która powoduje że zmiany stanów na diodach LED wizualizujących pracę zestawu są zbyt szybkie.

```

1   if (internal_state == EXECUTION){
2       uint64_t time_difference = millis() - last_successful_ucycle;
3       if(time_difference > execution_speed || type == CYCLE || type ==
4           MICRO_CYCLE){
5           ucycle_return = process_microcycle();
6       }
7   }
```

Następnie przetwarzana jest wartość zwracana przez funkcję `process_microcycle`. W przypadku gdy funkcja zwróciła kod SUCCESS dla trybów pracy Cykl oraz Mikrocykl sprawdzane są warunki zakończenia cyklu pracy zestawu. Następnie jest sprawdzany warunek zakończenia pracy zestawu niezależnie od trybu pracy. W przypadku zestaw wykonał wszystkie instrukcje znajdujące się w rejestrze  $R_P$  zestaw przechodzi w stan DONE.

```

1         switch (ucycle_return){
2             case SUCCESS:
3                 last_successful_ucycle = millis();
4                 switch (type){
5                     case CYCLE:
6                         if(Gs == 0){
7                             internal_state = CYCLE_DONE;
8                         }
9                         break;
10                    case MICRO_CYCLE:
11                        internal_state = CYCLE_DONE;
12                        break;
13                    case PROGRAM:
14                        break;
15                    default:
16                        internal_state = EXCEPTION;
17                        break;
18                }
19                if (Rp_address > 15){
20                    internal_state = DONE;
21                }
22                break;
23            case NEED_INPUT:
24                internal_state = USER_INPUT;
25                break;
26            case ERROR:
27                internal_state = EXCEPTION;
28                break;
29            default:
30                internal_state = EXCEPTION;
31                break;
32        }
33    }
34    return internal_state;
35 }
```

W przypadku gdy funkcja zwróciła kod NEED\_INPUT zestaw przechodzi w stan USER\_INPUT.

W tym stanie zestaw oczekuje na wprowadzenie wartości przez użytkownika. Po wprowadzeniu wartości zestaw przechodzi w stan EXECUTION i kontynuuje pracę. W przypadku wystąpienia błędu zestaw przechodzi w stan EXCEPTION. Na koniec funkcja zwraca stan Wewnętrznej maszyny stanów.

#### 4.3.4 Obsługa interfejsu użytkownika

W celu obsługi interfejsu użytkownika przy pomocy biblioteki Arduino menu w pierwszej kolejności należy zdefiniować sterowniki urządzeń wejściowych oraz wyjściowych. W naszym przypadku urządzeniami wejściowymi jest obrót enkoderem oraz wbudowany w enkoder przycisk, urządzeniem wyjściowym jest wyświetlacz LCD. Biblioteka Arduino menu dostarcza sterowniki enkodera oraz przycisku w formie klas.

```
1     encoderIn<ENC_S2,ENC_S1> encoder;//simple quad encoder driver
2     encoderInStream<ENC_S2,ENC_S1> encStream(encoder,ENC_SENSITIVITY);
3
4     //a keyboard with only one key as the encoder button
5     keyMap encBtn_map []={{-ENC_SW,defaultNavCodes[enterCmd].ch}};
6     keyIn<1> encButton(encBtn_map);//1 is the number of keys
7
8     //input from the encoder + encoder button + serial
9     menuIn* inputsList []={&encStream,&encButton};
10    chainStream<2> in(inputsList); //2 is the number of inputs
```

Urządzenia wejściowe obsługiwane są na zasadzie strumieni. W powyższym kodzie zdefiniowano strumień wejściowy **chainStream** który jest połączeniem strumieni enkodera oraz przycisku.

Obsługa strumienia wyjściowego dla wyświetlacza LCD wymaga zadeklarowania obiektu klasy **LiquidCrystal\_PCF8574** oraz przy pomocy makra **MENU\_OUTPUTS** dostarczanego przez bibliotekę Arduino menu przekazania obiektu do listy wyjść dla menu.

```
1     LiquidCrystal_PCF8574 lcd(0x27);
2     MENU_OUTPUTS(out, 3, LCD_OUT(lcd,{0,0,20,4}), NONE);
```

Następnym krokiem jest deklaracja głównego okna menu. Jest to pierwsze okno które jest wyświetlane po uruchomieniu programu. Tworzenie okna menu odbywa się przy pomocy makra **MENU**. Makro to przyjmuje szereg argumentów które decydują o tytule, wyglądzie, oraz reakcjach okna na zdarzenia. Jako argumenty przekazywane są również opcje które będą dostępne w oknie. Okno jest ustawiane jako główne przy pomocy makra **NAVROOT**.

```
1     MENU(busMainMenu, "Szyna danych",doNothing,noEvent,wrapStyle
2     ,SUBMENU(start_execution)
3     ,SUBMENU(ram_edit)
4     ,SUBMENU(program_edit)
5     ,FIELD(execution_clock,"Zegar","Hz",1,20,1,0,set_exec_speed_commit,anyEvent,
6     wrapStyle)
7     ,OP("Reset",reset_system_commit,enterEvent)
8 );
8     NAVROOT(nav, busMainMenu, 3, in, out);
```

Biblioteka Arduino menu posiada szereg wbudowanych typów opcji które można wykorzystać w oknach menu. W projekcie wykorzystano następujące opcje:

- **SUBMENU** - opcja która przenosi do innego okna
- **FIELD** - pozwala na edycję wartości oraz wyświetlanie wartości zmiennej
- **OP** - pole tekstowe które może reagować na zdarzenia takie jak wcisnięcie przycisku

- EXIT - powrót do poprzedniego okna

Obsługa opcji FIELD wymaga zdefiniowania zmiennej w której będzie przechowana zmieniana wartość oraz granic jakie wartość może przyjąć. W przykładzie opcji o nazwie "Zegar" znajdującej się na głównym ekranie zmienna `execution_clock` przechowuje wartość zegara, minimalna wartość to 1, maksymalna 20, a krok zmiany to 1. Dodatkowo zdefiniowano funkcje `set_exec_speed_commit` która jest wywoływana podczas zmian wartości zmiennej. Funkcja ta na bieżąco przekazuje wartość zmiennej do obiektu klasy odpowiedzialnej za symulację pracy zestawu.

```

1     uint8_t execution_clock = 10;
2
3     result set_exec_speed_commit(eventMask e, navNode& nav, prompt &item) {
4         cpu.set_execution_speed(1000 / execution_clock);
5         return proceed;
6     }

```

Obsługa opcji OP polega na zdefiniowaniu funkcji która zostaje wywołana po wcisnięciu przycisku. W przypadku opcji "Reset" znajdującej się na głównym ekranie wywoływana jest funkcja `reset_system_commit` która resetuje stan zestawu.

```

1     result reset_system_commit(eventMask e, navNode& nav, prompt &item) {
2         cpu.clear();
3         return proceed;
4     }

```

Zapis wartości do pamięcią RAM oraz  $R_P$  odbywa się na podobnej zasadzie. Zadeklarowano zmienne `ram_address_edit` oraz `ram_value` przechowujące obecną wartość adresu pamięci RAM oraz wartość w obecnej komórce pamięci RAM. Obydwie te zmienne są przypisywane do odpowiednich opcji FIELD w oknie edycji pamięci RAM.

```

1     uint8_t ram_address_edit = 0;
2     uint8_t ram_value = 0;
3
4     MENU(ram_edit, "Zapis RAM", doNothing, noEvent, noStyle
5           , FIELD(ram_address_edit, "Adres:", "", 0, 63, 1, 0, display_ram, anyEvent, wrapStyle)
6           , OP("Zapisz", edit_ram_commit, enterEvent)
7           , FIELD(ram_value, "Wartosc:", "", 0, 255, 0, 0, doNothing, noEvent, noStyle)
8           , EXIT("<Wroc")
9     );

```

Po wybraniu odpowiedniego adresu w opcji "Adres" należy wcisnąć opcję "Zapisz" która wywoła funkcję `edit_ram_commit`. Funkcja ta zapisuje wartość z klawiatury wprowadzania danych do pamięci RAM na adresie `ram_address_edit`. Oraz inkrementuje wartość w zmiennej przechowującej adres.

```

1     result edit_ram_commit(eventMask e, navNode& nav, prompt &item) {
2         cpu.set_RAM(ram_address_edit, io.read_data_input_buttons());
3         ram_address_edit++;
4         ram_value = cpu.get_RAM(ram_address_edit);
5         return proceed;
6     }

```

W celu zapewnienia interaktywnego podglądu wartości pamięci RAM pod aktualnie wybranym adresem do opcji "Adres" przypisano funkcję `display_ram` wywoływaną przy każdym zdarzeniu. Powoduje to wywołanie funkcji przy każdej zmianie adresu w polu edycji. Funkcja ta pobiera wartość z pamięci RAM pod adresem `ram_address_edit` i przypisuje ją do zmiennej `ram_value` co powoduje aktualizację wartości wyświetlanej w polu "Wartość".

```

1     result display_ram(eventMask e, navNode& nav, prompt &item) {
2         ram_value = cpu.get_RAM(ram_address_edit);
3         return proceed;
4     }

```

Ekran pobierania danych od użytkownika został zrealizowany w funkcji `user_input`. Funkcja ta jest wywoływana w momencie gdy zestaw oczekuje na dane od użytkownika. Funkcja wyświetla komunikat proszący o podanie wartości. Po wcisnięciu przycisku pobiera wartość z klawiatury.

```

1     result user_input(menuOut& o, idleEvent e) {
2         switch(e) {
3             case idleStart:
4                 o.print("Wprowadz wartosc");
5                 break;
6             case idleEnd:
7                 cpu.take_user_input(io.read_user_input_buttons());
8                 break;
9         }
10        return proceed;
11    }

```

#### 4.3.5 Główna pętla programu

Program rozpoczyna się od wywołania funkcji `setup`. W tej funkcji przeprowadzana jest inicjalizacja wyświetlacza LCD, enkodera oraz obiektu nawigacyjnego biblioteki Arduino Menu. Następnie wywoływana jest funkcja `loop` w której znajduje się główna logika programu. Funkcja ta jest wywoływana w nieskończonej pętli.

```

1     void loop() {
2         bus_cpu_state cpu_state;
3         nav.poll();
4         cpu_state = cpu.schedule_execution();
5
6         if(cpu_state == bus_cpu_state::USER_INPUT) {
7             nav.idleOn(user_input);
8         }
9         else if(cpu_state == bus_cpu_state::EXCEPTION) {
10            Serial.println("Zestaw napotkal blad");
11        }
12
13        io.set_displayed_register(displayed_register);
14        cpu.get_register_values(cpu_status);
15        io.render_led(cpu_status);
16        delay(50);
17    }

```

W głównej pętli programu jest wywoływana funkcja `poll` obiektu nawigacyjnego. Funkcja ta odpowiada za przetwarzanie zdarzeń związanych z interfejsem użytkownika. Następnie wywoływana jest funkcja `schedule_execution` która zarządza pracą symulatora szyny danych. W zależności od zwróconego stanu zestawu wywoływanie są odpowiednie funkcje. W przypadku gdy zestaw oczekuje na dane od użytkownika wywoływana jest funkcja `idleOn` która wywoła okno menu odpowiedzialne za wprowadzanie danych. W przypadku wystąpienia błędu zostanie wydrukowana informacja na porcie szeregowym. Kolejne trzy funkcje odpowiadają za aktualizacje stanu diod LED na podstawie aktualnych wartości rejestrów zestawu. Na końcu pętli programu znajduje się opóźnienie 50 milisekund które zapewnia stabilne działanie programu.

## 5 Testy stanowiska

Testy stanowiska przeprowadzono z wykorzystaniem programów testowych podobnych do programów uruchamianych na zestawie w ramach ćwiczeń laboratoryjnych. Programy testowe zostały napisane w taki sposób, aby sprawdzić poprawność jak największej ilości elementów zestawu.

### 5.1 Program testujący operacje arytmetyczne

Program testuje poprawność wykonywania operacji na jednostce arytmetyczno logicznej. Program został celowo napisany tak aby zapisywał wartości do wszystkich dostępnych odbiorników w celu przetestowania ich poprawnej implementacji. Program wykonuje następujące operacje:

- Pobranie danych z klawiatury wejściowej do  $R_A$ .
- Skopiowanie do  $R_A$  do do  $R_B$ .
- Skopiowanie do  $R_A$  do do  $R_B$ .
- Pomnożenie  $R_A$  przez 2 i zapisanie wyniku w  $R_A$ .
- Dodanie  $R_A$  do  $R_B$  i wystawienie wyniku do  $R_{wy}$ .

W pierwszym cyklu program pobiera dane z klawiatury wejściowej i zapisuje je do rejestru  $R_A$ . Następnie kopiuje je do rejestru  $R_B$  oraz  $R_C$ . W kolejnym cyklu zapisuje zawartość  $R_A$  do  $R_1$  oraz  $R_2$ , następnie zapisuje wartość na wyjściu jednostki ALU do rejestru  $R_{wy}$ . Zapis do rejestru  $R_2$  jest w tym wypadku wykonywany tylko w celu poprawnego ustalenia kodu operacji w jednostce ALU, gdyż wykonywana operacja mnoży zawartość rejestru  $R_1$  przez 2. W ostatnim cyklu program dodaje do siebie zawartości rejestrów  $R_A$  i  $R_B$  i zapisuje wynik w  $R_{wy}$ .

Nadajnik	Odbiornik	ALU	Adres	$R_P$
1 0 1	0 0 1	0 0	000000	0001
0 0 1	0 1 0	0 0	000001	
0 0 1	0 1 1	0 0	000010	
0 0 0	0 0 0	0 0	000011	

0 0 0	0 0 0	0 0	000100	0010
0 0 1	1 0 0	1 0	000101	
0 0 1	1 0 1	0 1	000110	
1 0 0	0 0 1	0 0	000111	

0 0 0	0 0 0	0 0	001000	0011
0 0 1	1 0 0	0 1	001001	
0 1 0	1 0 1	0 1	001010	
1 0 0	1 1 0	0 0	001011	

Tabela 6: Program testujący operacje arytmetyczne

## 5.2 Program testujący instrukcje skoku

Program testuje poprawność wykonywania skoku pomiędzy instrukcjami. Program wykonuje następujące operacje:

- Pobranie danych z klawiatury wejściowej do  $R_A$ .
- Wykonanie skoku w zależności od wartości  $R_A$ .
- Jeśli  $R_A = 1$  zanegowanie  $R_A$  i zapisanie wyniku w  $R_{wy}$ .
- Jeśli  $R_A = 2$  podzielenie  $R_A$  przez 2 i zapisanie wyniku w  $R_{wy}$ .

W celu wykonania skoku program w ostatniej mikroinstrukcji pierwszego cyklu wykonuje operacje zapisu zawartości rejestru  $R_A$  do rejestru  $R_I$ . Spowoduje to zinterpretowanie wartości w  $R_A$  jako czterech najstarszych bitów adresu, czyli adresu kolejnych czterech instrukcji. W tym wypadku zakładamy jeśli wartość w rejestrze  $R_A$  jest równa 1 program w kolejnym cyklu skoczy do instrukcji o adresie 000100. Jeśli wartość w rejestrze  $R_A$  jest równa 2 program w kolejnym cyklu skoczy do instrukcji o adresie 001000. Pod tymi rozpoczynają się cykle mikroinstrukcji odpowiednio dla zanegowania wartości w  $R_A$  i podzielenia wartości w  $R_A$  przez 2.

Nadajnik	Odbiornik	ALU	Adres	$R_P$
1 0 1	0 0 1	0 0	000000	0011
0 0 1	0 0 1	0 0	000001	
0 0 1	0 0 1	0 0	000010	
0 0 1	0 0 0	0 0	000011	

0 0 0	0 0 0	1 0	000100	0011
0 0 1	1 0 0	0 0	000101	
0 0 1	1 0 1	0 0	000110	
1 0 0	1 1 0	0 0	000111	

0 0 0	0 0 0	0 1	00100	0011
0 0 1	1 0 0	0 0	001001	
0 0 1	1 0 1	0 1	001010	
1 0 0	1 1 0	0 0	001011	

Tabela 7: Program testujący instrukcje skoku

W rozpatrywanym programie wartość w rejestrze  $R_P$  jest równa 0011. Dzięki temu po wykonaniu instrukcji warunkowej program powróci do wykonywania dalszych instrukcji. Gdyby wartości w rejestrze  $R_P$  rozpoczynały się od 1 i zwiększały o 1 jak w większości przypadków program przy wprowadzeniu wartości 1 wykonał by obie instrukcje warunkowe gdyż druga instrukcja nie została pominięta.

## **6 Podsumowanie**

Celem pracy była budowa stanowiska laboratoryjnego do realizacji ćwiczeń z sterowaniem szyną danych w laboratorium techniki cyfrowej.

W ramach pracy dokonano krytycznej analizy istniejącego stanowiska laboratoryjnego. Przeanalizowano stanowisko pod kątem jego funkcjonalności, architektury sprzętowej oraz interfejsu użytkownika.

Na podstawie wyników analizy sporządzono listę wymagań dla nowego stanowiska. Opracowano architekturę sprzętową oraz programową nowego stanowiska. Dokonano wyboru elementów sprzętowych oraz oprogramowania niezbędnego do realizacji nowego stanowiska.

Zaprojektowano oraz zbudowano prototyp części sprzętowej nowego stanowiska. Napisano oprogramowanie implementujące funkcjonalności nowego stanowiska.

Przeprowadzono testy prototypu stanowiska w celu weryfikacji poprawności działania oraz spełnienia wymagań postawionych przed nowym stanowiskiem.

Dalsze prace nad projektem powinny skupić się na przeniesieniu konstrukcji prototypu na płytę drukowaną. Należy również zaprojektować i wykonać obudowę stanowiska. Alternatywnie można rozważyć adaptacje obudowy istniejącego stanowiska do nowego projektu. Do oprogramowania stanowiska można dodać funkcję zapisu pamięci na podstawie pliku tekstowego przygotowanego przez użytkownika. Z pewnością usprawniło by to proces wykonywania ćwiczeń na stanowisku.

## Bibliografia

- [1] Mariusz Szostek. *Sterowanie szyną danych - instrukcja obsługi.*
- [2] Mariusz Szostek. "Sterowanie szyną danych". Prac. mag. Politechnika Gdańska, 2012.
- [3] *Atmega 8*. Rev.2486AA–AVR–02/2013. Atmel Corporation. Lut. 2013. URL: [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8\\_L\\_datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf).
- [4] *Atmega 32*. 2503Q–AVR–02/11. Atmel Corporation. Lut. 2011. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf>.
- [5] *RP2040*. Ver. eec2b0c-clean. Raspberry Pi Ltd. Paź. 2024. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf>.
- [6] *HD44780 LCD Starter Guide*. URL: [https://www-leland.stanford.edu/class/ee281/handouts/lcd\\_tutorial.pdf](https://www-leland.stanford.edu/class/ee281/handouts/lcd_tutorial.pdf) (term. wiz. 08.01.2025).
- [7] *Rotary Encoder*. URL: <https://electropeak.com/learn/rotary-encoder-how-it-works-how-to-use-with-arduino/> (term. wiz. 08.01.2025).
- [8] *SNx4HC595*. Texas Instruments. Paź. 2021. URL: <https://www.ti.com/lit/ds/symlink/sn74hc595.pdf>.
- [9] *PCF8574*. Texas Instruments. Wrz. 2024. URL: <https://www.ti.com/lit/ds/symlink/pcf8574.pdf>.
- [10] *Arduino framework for RP2040*. URL: <https://github.com/earlephilhower/arduino-pico> (term. wiz. 08.01.2025).
- [11] *Biblioteka Arduino 74HC595*. URL: <https://github.com/Simsso/ShiftRegister74HC595> (term. wiz. 08.01.2025).
- [12] *Biblioteka LCD PCF8574*. URL: [https://github.com/mathertel/LiquidCrystal\\_PCF8574](https://github.com/mathertel/LiquidCrystal_PCF8574) (term. wiz. 08.01.2025).
- [13] *Biblioteka Arduino menu*. URL: <https://github.com/neu-rah/ArduinoMenu> (term. wiz. 08.01.2025).

## **Spis rysunków**

2.1	Płyta czołowa zestawu . . . . .	10
3.1	Schemat blokowy sprzętowej części zestawu . . . . .	18
3.2	Schemat blokowy oprogramowania zestawu . . . . .	19
3.3	Diagram menu interfejsu użytkownika . . . . .	21
4.1	Schemat połączeń enkodera oraz wyświetlacza . . . . .	22
4.2	Schemat połączeń rejestrów wraz z diodami LED . . . . .	23
4.3	Schemat połączeń przycisków . . . . .	24
4.4	Schemat elektryczny zestawu . . . . .	24
4.5	Zestaw wykonany na płytce stykowej . . . . .	25

## **Spis tabel**

1	Symboly przypisane poszczególnym bitom słowa w pamięci RAM . . . . .	11
2	Tablica prawdy dla dekodera 1 . . . . .	11
3	Tablica prawdy dla dekodera 2 . . . . .	12
4	Zasada kodowania operacji do wykonania przez ALU . . . . .	12
5	Lista operacji realizowanych przez ALU . . . . .	13
6	Program testujący operacje arytmetyczne . . . . .	33
7	Program testujący instrukcje skoku . . . . .	34