Politechnika Wrocławska

Laboratorium
Podstaw Techniki Mikroprocesorowej

Data: 30.05.2021	**Dzień:** Poniedziałek
Grupa: E06-89aj	**Godzina:** 12:30
TEMAT ĆWICZENIA: Projekt końcowy	
Nr. indeksu	**Nazwisko i Imię**
252872	Wieczorek Michał
252887	Mielcarz Remigiusz

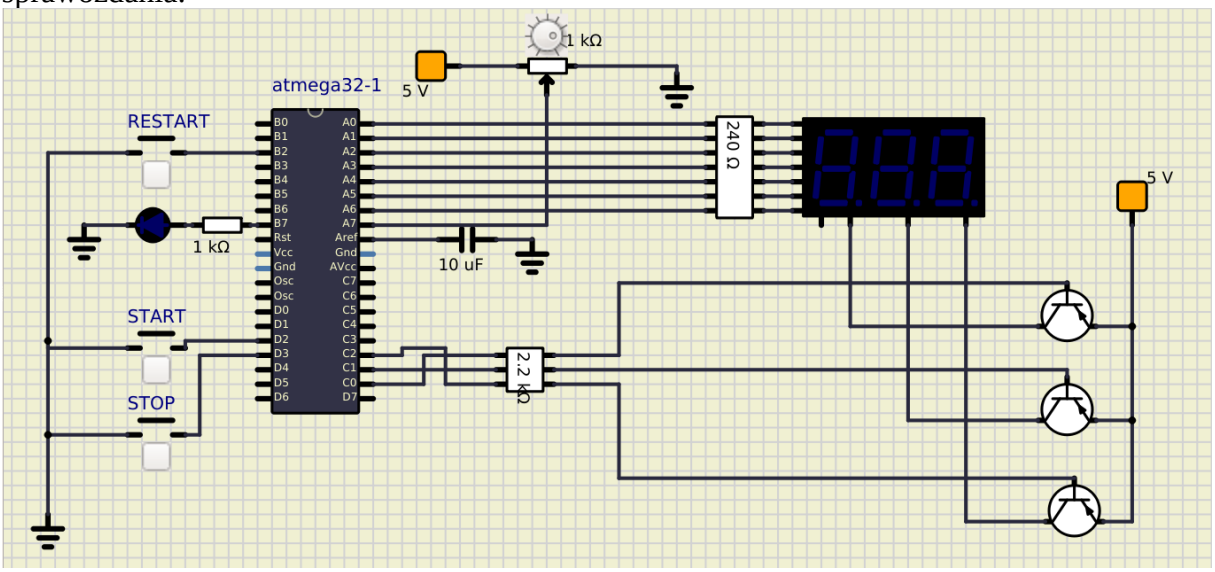
1. **Cel ćwiczenia**

Celem ćwiczenia jest napisanie programu, który będzie wykorzystywał wszystkie prędzej poznane elementy. Program zakłada:
- wyświetlenie ustawionej za pomocą potencjometru wartości na wyświetlaczu
- dodanie przycisku START i STOP wykorzystującego przerwania
- użycie diody, sygnalizującej wartość zero

Program ma działać tak, aby po uruchomieniu na wyświetlaczu wyświetlała się wartość od 0 do 100 w zależności od pozycji potencjometru. Następnie obracając potencjometrem możemy zmieniać wartość. Przycisk START odpowiada za rozpoczęcie odliczania od ustalonej wartości w dół a STOP za zatrzymanie odliczania. Dodatkowo dodaliśmy przycisk RESET, który powoduje powrót stanu wyświetlacza do tego sprzed rozpoczęcia odliczania. Dioda ma odpowiadać za sygnalizację końca odliczania, czyli równoważnie za zapalenie się, gdy na wyświetlaczu widnieje wartość 0.

2. **Przebieg ćwiczenia**

Ćwiczenie zaczęliśmy od stworzenia układu, zawierającego wszystkie elementy konieczne do wykonania się programu. Zasada działania układu została już opisana w poprzedniej części sprawozdania.



Rysunek 1 Schemat układu projektowego w SimulIDE

W dalszej części zostaną opisane poszczególne części kodu.

Tworzenie kodu zaczęliśmy od dodanie odpowiednich bibliotek oraz definicji: inicjalizacji LEDów, włączenia i wyłączenia LEDów, definicji portów i rejestru kierunkowego oraz definicji inicjalizacyjnych dla przycisków. Fragment kodu odpowiedzialny za tę część został przedstawiony poniżej.

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

// Definijcje inicjalizacyjne dla diody LED
#define INIT_LED  DDRB  |= (1 << PB7)
#define ON_LED    PORTB |= (1 << PB7)
#define OFF_LED   PORTB &= ~(1 << PB7)

// Definicje inicjalizacyjne dla przycisków
#define INIT_SWSTART   DDRD &= ~(1<<PD2) //START
#define INIT_SWSTOP    DDRD &= ~(1<<PD3) //STOP
#define INIT_SWRESTART DDRB &= ~(1<<PB2) //RESTART

#define SW_CONFIG_PULLUP_START   PORTD |= (1 << PD2)
#define SW_CONFIG_PULLUP_STOP    PORTD |= (1 << PD3)
#define SW_CONFIG_PULLUP_RESTART PORTB |= (1 << PB2)

// Definicje portu i rejestru kierunkowego dla segmentów wyświetlacza
#define SEGMENTY_PORT       PORTA
#define SEGMENTY_KIERUNEK   DDRA

// Definicje portu i rejestru kierunkowego dla anod wyświetlacza
#define ANODY_PORT          PORTC
#define ANODY_KIERUNEK      DDRC

// Definicje bitow dla poszczegolnych anod
#define ANODA_1             (1<<PC0)
#define ANODA_2             (1<<PC1)
#define ANODA_3             (1<<PC2)
#define MASKA_ANODY (ANODA_1 | ANODA_2 | ANODA_3)   //0000 0111

// Definicje bitow dla poszczegolnych segmentow
#define SEG_A               (1<<0)
#define SEG_B               (1<<1)
#define SEG_C               (1<<2)
#define SEG_D               (1<<3)
#define SEG_E               (1<<4)
#define SEG_F               (1<<5)
#define SEG_G               (1<<6)
```

Kolejne było zadeklarowanie zmiennych nieulotnych, które będą wykorzystywane w dalszej części programu.

```
// Zmienne
volatile uint8_t cyfra[4];
volatile int odliczanie      = 0;
volatile int wartosc         = 0;
volatile int pomiar          = 1; // Zezwól na początku programu na odczytywanie z przetwornika
volatile uint8_t licznik_ovf = 0;
```

Następnie została stworzona tablica 15 – sto elementowa **cyfry[15]**, wykorzystywana później w częściach związanych z wyświetlaczem multipleksowanym, oraz funkcja **init()** inicjalizująca odpowiednie piny dla poszczególnych segmentów, odpowiednie porty dla przycisków oraz ustawiająca TIMER0 i TIMER2.

```
const uint8_t cyfry[15] = {
        ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_F),          //0    ~0011 1111  czyli 1100 0000 podłączone anodami do masy
        ~(SEG_B|SEG_C),                                  //1    ~0000 0110  czyli 1111 1001
        ~(SEG_A|SEG_B|SEG_D|SEG_E|SEG_G),                //2    ~0101 1011  czyli 1010 0100
        ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_G),                //3
        ~(SEG_B|SEG_C|SEG_F|SEG_G),                      //4
        ~(SEG_A|SEG_C|SEG_D|SEG_F|SEG_G),                //5
        ~(SEG_A|SEG_C|SEG_D|SEG_E|SEG_F|SEG_G),          //6
        ~(SEG_A|SEG_B|SEG_C|SEG_F),                      //7
        ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_E|SEG_F|SEG_G),    //8
        ~(SEG_A|SEG_B|SEG_C|SEG_D|SEG_F|SEG_G),          //9
};

void init(void)
{
    SEGMENTY_KIERUNEK |= SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G; // (DDRA  = x111 1111) Ustawienie rejestru kierunkowego portu segmentow na wyjscie
    SEGMENTY_PORT |= SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;       // (PORTA = x111 1111) Wygaszenie wszystkich katod - stan wysoki

    ANODY_KIERUNEK |= ANODA_1 | ANODA_2 | ANODA_3; // (DDRC = xxxx x111) Ustawienie anod na wyjscia
    ANODY_PORT |= ANODA_1 | ANODA_2 | ANODA_3;     // (PORTC = xxxx x111) Wygaszenie wszystkich wyswietlaczy - stan wysoki

    INIT_SWSTART; // Inicjalizacja przcisków
    INIT_SWSTOP;
    INIT_SWRESTART;

    SW_CONFIG_PULLUP_START; // Pull UP do przycisków
    SW_CONFIG_PULLUP_STOP;
    SW_CONFIG_PULLUP_RESTART;

    // Ustawienia TIMER0
    TCCR0 |= (1<<WGM01); // Tryb CTC
    TCCR0 |= (1<<CS02) | (1<<CS00); // Preskaler = 1024
    OCR0 = 53; // Dodatkowy podział przez 53    (11059200 Hz / 1024 / 200Hz[oczekiwana wartość])-1 = 53
    TIMSK |= (1<<OCIE0); // zezwolenie na przerwanie CompareMatch

    // Ustawienia TIMER2
    TCCR2 |= (1 << CS21) | (1<<CS22);     // prescaler na 256
    TIMSK |= (1 << TOIE2);
    TCNT2 = 5;
}
```

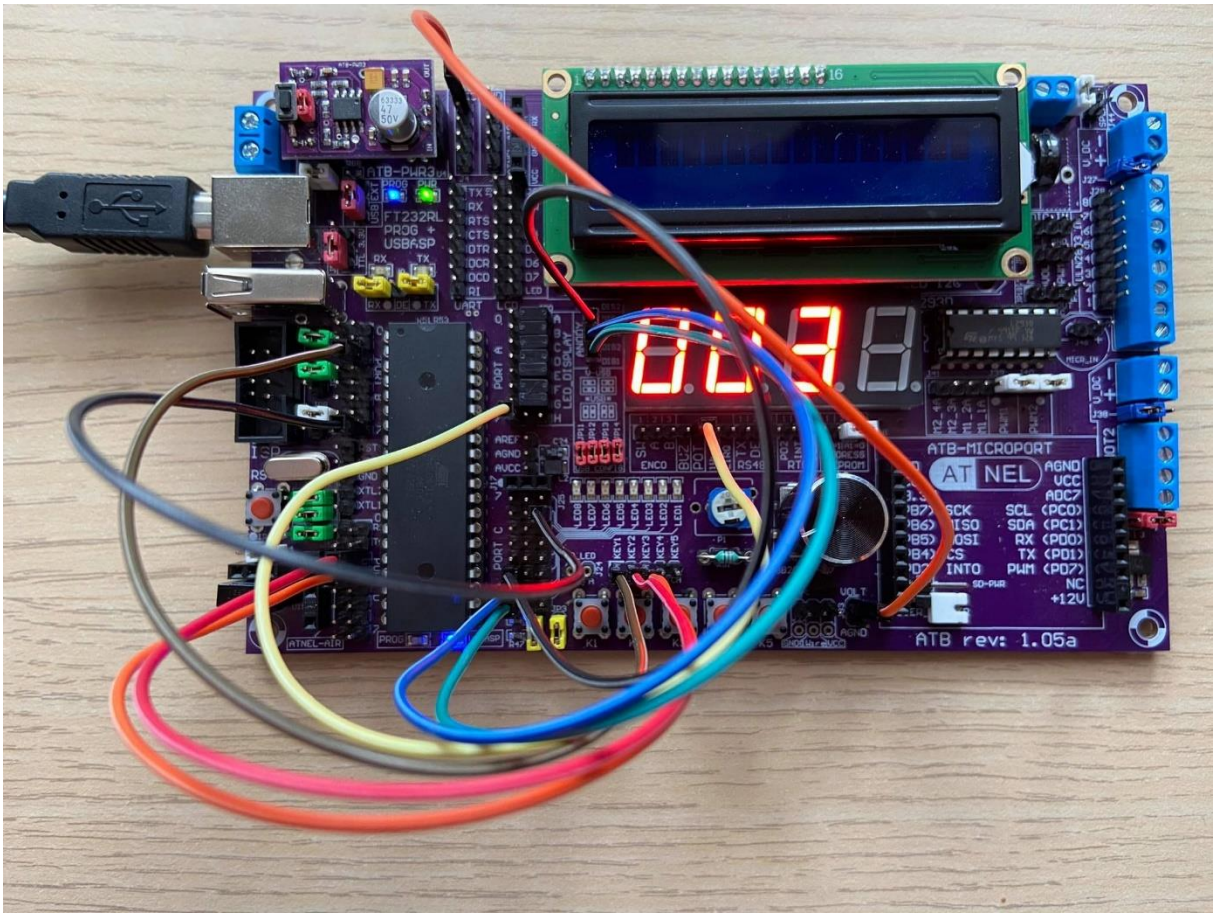
Kolejna część kodu dotyczyła przerwań. Pierwsze przerwanie powoduje rozpoczęcie odliczania i jednocześnie zatrzymanie pomiaru (START), drugie przerwanie odliczania (STOP), a trzecie ponowne zezwolenie na wykonywanie pomiaru oraz zakończenie odliczania (RESTART). Czwarte przerwanie dotyczy przepełnienia overflow za pomocą zmiennej nieulotnej licznik_ovf. Polega ono na odejmowaniu wartości do zera jeśli nastąpi przepełnienie.
Mając poprawnie zdefiniowane odpowiednie funkcje, stałe i zmienne mogliśmy przejść do funkcji main. Na samym początku oprócz inicjalizacji, wybraliśmy źródło odniesienia, uruchomiliśmy przetwornik oraz ustawiliśmy odpowiednie przerwania. Po definicji wykonaliśmy wstępne sprawdzenie działania wyświetlacza oraz diody.

```
int main(void)
{
    init();

    uint8_t z1, z2, z3;

    ADMUX |= ((1 <<  REFS0) | (1 << MUX0) | (1 << MUX1) | (1 << MUX2)); //wybor zrodla napiecia odniesienia (AVCC (+5V) s.214) i pinu pomiarowego ADC (ADC7, bo na pinie PA7 (s.215))

    ADCSRA |= ((1 << ADEN) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2)); //uruchomienie przetwornika (s.216) i ustawienie czestotliwosci jego pracy (preskaler 128 (s.217))

    sei(); // zezwolenie na globalne przerwania

    MCUCR |= (1 << ISC01);          // INT0 na zbocze opadajace
    MCUCR |= (1 << ISC11);          // INT1 na zbocze opadajace

    GICR |= (1 << INT0);        // przerwanie na START //PD2
    GICR |= (1 << INT1);        // przerwanie na STOP  //PD3
    GICR |= (1 << INT2);        // przerwanie na RESET //PB2

    // Sprawdzenie czy działa wyświetlanie liczb
    cyfra[0] = 0;
    cyfra[1] = 8;
    cyfra[2] = 8;
    _delay_ms(2500);
    cyfra[0] = 3;
    cyfra[1] = 5;
    cyfra[2] = 9;
    _delay_ms(2500);

    //Sprawdzenie działania diody LED
    ON_LED;
    _delay_ms(2000);
    OFF_LED;
```

Zasadnicza część programu wykonuje się wewnątrz głównej pętli. Tam uruchamiana jest najpierw konwersja a później załączenie lub wyłączenie diody sygnalizacyjnej, w zależności od warunku. Na koniec zostaje wyświetlona aktualna wartość na wyświetlaczu.

```
    while(1)
    {
        ADCSRA |= (1 << ADSC); // uruchamia konwersje (s.216)

        if(pomiar == 1) // Jeśli jest zezwolenie na pomiar -> wykonaj go
        {
        while(ADCSRA & (1 << ADSC)); // oczekiwanie na zakonczenie konwersji
        wartosc = ADC;              // odczyt pomiaru z przetwornika
        wartosc /= 10;
        }

        if(wartosc == 0 && pomiar == 0) // Jeśli wartość pomiaru dosięgła zera i jest brak zezwolenia na pomiar -> uruchom diodę LED
        ON_LED;
        else
        OFF_LED;

        z1 = wartosc/100;
        if(z1<1) cyfra[0] = 0; else cyfra[0]=1;
        z2 = (wartosc-(z1*100)) / 10;
        cyfra[1] = z2;
        z3 = wartosc % 10;
        cyfra[2] = z3;
        _delay_ms(50);

    }
    return 0;
}
```

3. Prezentacja działania programu na płytce uruchomieniowej ATB 1.05a



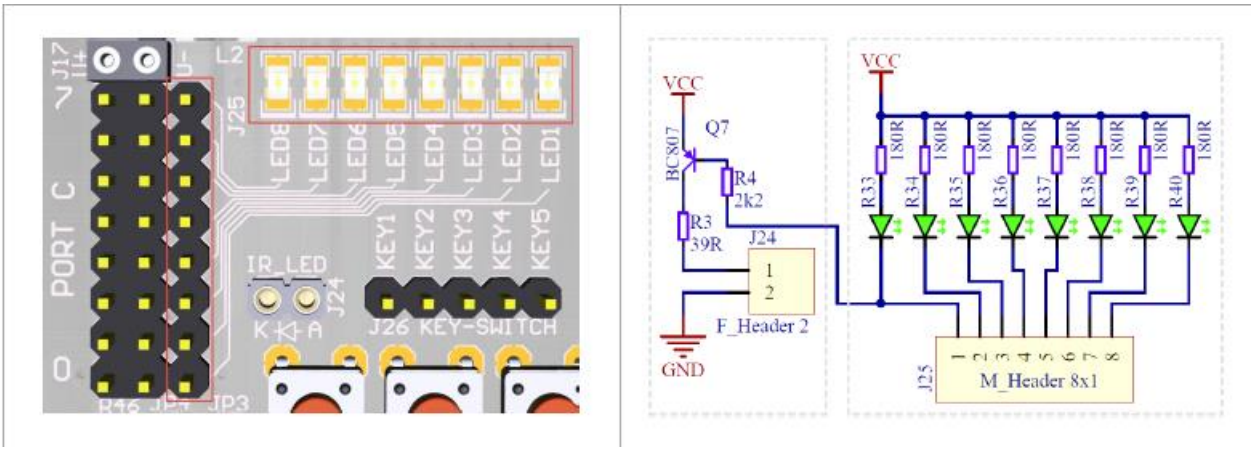
Rysunek 2 Schemat podłączenia elementów na płytce ATB 1.05A

Na płytce uruchomieniowej zostały zwarte zworkami segmenty A-G do portu A.
Następnie połączono przewodami wyświetlacz LED pinami DIG1-DIG3 do portu C.
Do portu D zostały podłączone przyciski K2-K3, a do portu B przycisk RESET K1.
Potencjometr został podłączony od pinu POT do PA7. W zestawie poprowadzono specjalną linię masy analogowej, która jest dobrze odseparowana od masy cyfrowej. Dlatego pomiaru napięć z przygotowanym dzielnikiem napięcia dokonujemy za pomocą wyprowadzeń złącza J33 opisanego na PCB jako VOLT. W skład dzielnika wchodzi wlutowany fabrycznie potencjometr montażowy o rezystancji 20 kΩ, a także przewlekany rezystor R31 usadowiony w podstawce precyzyjnej o wartości 10 kΩ. Za pomocą suwaka potencjometru zmieniamy stosunek podziału całego dzielnika zmieniając zakres mierzonych napięć, albo precyzyjnie dobieramy podział dzielnika do aktualnego napięcia dostępnego na wyprowadzeniu **AREF** mikrokontrolera.

Na potrzebę testu na płytce zmieniono odrobinę kod:

```
6  // Definijcje inicjalizacyjne dla diody LED
7  #define INIT_LED  DDRB  |= (1 << PB7)
8  #define OFF_LED   PORTB |= (1 << PB7)
9  #define ON_LED    PORTB &= ~(1 << PB7)
```

Ponieważ na płytce diody LED są podłączone anodami do VCC:



Rysunek 3 Schamat podłączeń diod LED

W SimulIDE zastosowaliśmy podłączenie katodą do GND.

4. Wnioski i podsumowanie

Wszystkie zadania projektowe zostały zrealizowane poprawnie. Do realizacji wykorzystaliśmy prędzej poznane elementy takie jak: załączanie i wyłączanie diody, sterowanie wyświetlaczem multipleksowanym, przerwania, wbudowany przetwornik ADC oraz timery.
Program został przetestowany także na płytce uruchomieniowej ATB 1.05a i napisany w programie Eclipse.
Prezentacje powyższych schematów zostały zawarte w pliku .rar.