Podstawy Techniki Mikroprocesorowej 1
Sprawozdanie 2

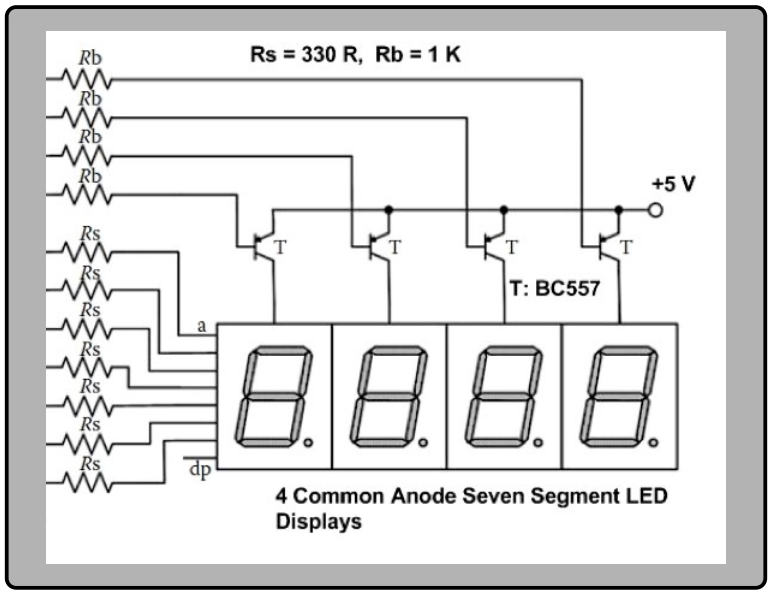
Martyna Żukowska i Krystian Pacyniak

2021

1 Wstep Teoretyczny

1.1 Multipleksowanie

W celu obsługi wyświetlacza 7 segmentowego składającego się z 4 kolumn, jedną z możliwych metod do jest multipleksowanie. Multipleksowanie polega na szybkim przełączaniu kolejnych segmentów i wyświetlaniu odpowiedniej zadanej wartości. Gdy wykona się to odpowiednio szybko ludzkie oko nie zauważy zapalania i gaśnięcia poszczególnych elementów, tylko zobaczy na wyświetlaczu stałą wartość. Algorytm multipleksowania polega na włączeniu kolumny X oraz wyświetlenie na niej cyfry dla wybranej kolumny, następnie wyłączenie kolumny X i powtórzenie działania dla innych kolumn. Podłączyć mikroprocesor z wyświetlaczem najlepiej przez tranzystory. Kolektory tranzystorów łączymy bezpośrednio z katodami wyświetlaczy. Bazy tranzystorów łączymy przez rezystor z mikroprocesorem. Emitery obu tranzystorów łączymy z masą.



Rysunek 1. Przykładowy schemat działania czterokolumnowego 7 segmentowego wyświetlacza

1.2 Przerwania

Przerwania to mechanizm, który pozwala mikrokontrolerowi na przerwanie bieżąco wykonywanych zadań, bez niepotrzebnego delaya. Wykonywane jest to przy pomocy innego impulsu. Źródłem przerwań

mogą być wewnętrzne lub zewnętrzne układy. Wewnętrzne układy to na przykład przepełnienie licznika (po odliczeniu ustalonej ilości impulsów), przetwornik jako informator o zakończeniu przetwarzania lub zdarzenie informujące o zakończeniu transmisji. Zewnętrzne układy to sygnały podawne na wejściu mikrokontrolera na przykład wciśnięcie przycisku, czy liczenie impulsów przez enkodery.

Dużą rolę odgrywają tutaj Bloki Funkcyjne które działają prawie jak funkcja, ale żeby dokładniej zrozumieć ich znaczenie trzeba się cofnąć do poziomu Assemblera. Tutaj dojdzie też dokładne pojęcie Rejestru. Natomiast Blok Funkcyjny na przykład potrafi niektóre zmienne nastawiać podczas działania przerwań.

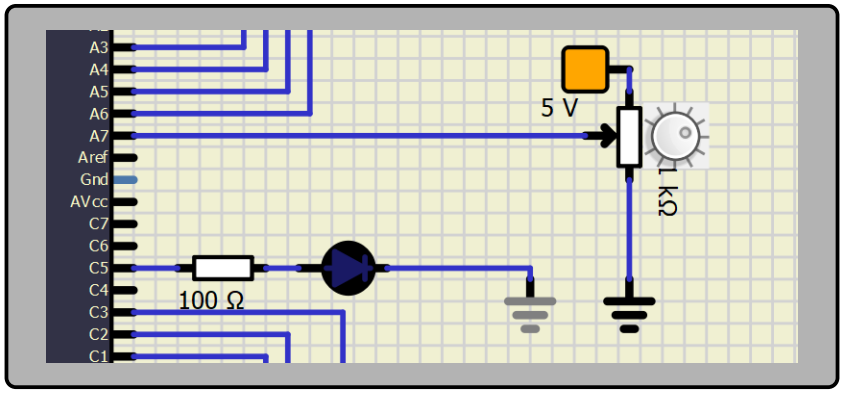
1.3 Przetwornik A/D

Przetwornik analogowy cyfrowy służy do zamiany sygnału analogowego na cyfrowy. Do poprawnej pracy przetwornika należy skonfigurować odpowiednie rejestry.

2 Wyświetlacz 4 kolumnowy 7 segmentowy

2.1 Inicjalizacja

W na samym początku należało zbudować układ z jedną dioda która zapalałaby się przy przekroczeniu zadanego zakresu na potencjometrze.



Rysunek 2. Schemat pierwszej części programu

Do tego na początek zainicjowano LED_ON i LED_OFF analogicznie, jak na wcześniejszych zajęciach oraz w odpowiedni sposób potencjometr, jak widać za poniższym kodzie

```
ADMUX |= (1<<REFS0); //ustawienie napiecia ref na napiecie zasilania
ADMUX |= ((1<<MUX0)| (1<<MUX1) | (1<<MUX2)); //wybor kanału, w tym wypadku kanał 7 (00111)

ADCSRA |= (1<<ADEN); // uruchomienie przetwornika ADC
ADCSRA |= ((1<<ADPS0) | (1<<ADPS1)); //Ustawienie preskalera na 8 (011) - ustaw czestotliwość na 8
```

Rysunek 3. Kod z inicjalizacją i komentarzem działania

Następnie wykonana została funkcja pomiar()

```
int pomiar(void)
{
    ADCSRA |= (1<<ADSC); //start konwersji
    while(ADCSRA & (1<<ADSC)); //WHILE NA ZERACH KONCZY ZADANIE
    return ADC;
}
```

Rysunek 4. Funkcja pomiar()

Która zwracała mierzoną wartość. Następnie napisano kod w int main() na bazie while(1), co powodowało, że działania są wykonywane cały czas po odpaleniu programu, a nie jednorazowo. Ma on za zadania przypisywać do zmiennej zm wartość pomiaru na potencjometrze. W momencie, gdy jest ona mniejsze niż zadana wartość równa 512 dioda ma być zgaszona, w innym wypadu ma być zapalona.

```
while(1)
{
    int zm=pomiar();
    if(zm<512)
    {
        LED_ON;
    }
    else
    {
        LED_OFF;
    }
```

Rysunek 5. Kod docelowy

W kolejnych krokach należało wyświetlać otrzymana wartość na 4 kolumnowym wyświetlaczu 7 segmentowym. Do tego skorzystano z funkcji ze wcześniejszych zajęć switch(cyfra), która miała zapalać odpowiednio diody na wybranych wyświetlaczu 7 segmentowym, zgodnie z wybraną cyfrą. Podobnie jak wcześniej tak samo zainicjowano na początek programu LED_ON i LED_OFF, każdej z diod oraz dodatkowo zainicjowano INIT_1, INIT_2, INIT_3 i INIT_4, które były odpowiedzialne za uruchomienie odpowiedniej kolumny w 4 kolumnowym 7 segmentowym wyświetlaczu.

```
#define INIT_1   DDRC |= (1<<PC3)  //output - wyjście bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_1       PORTC |= (1<<PC3)
#define OFF_1   PORTC &= ~(1<<PC3)

#define INIT_2   DDRC |= (1<<PC2)  //output - wyjście bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_2       PORTC |= (1<<PC2)
#define OFF_2   PORTC &= ~(1<<PC2)

#define INIT_3   DDRC |= (1<<PC1)  //output - wyjście bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_3       PORTC |= (1<<PC1)
#define OFF_3   PORTC &= ~(1<<PC1)

#define INIT_4   DDRC |= (1<<PC0)  //output - wyjście bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_4       PORTC |= (1<<PC0)
#define OFF_4   PORTC &= ~(1<<PC0)
```

Rysunek 6. Kod inicjowania INIT_1, INIT_2, INIT_3 i INIT_4

Oczywiście można ten kod napisać jeszcze prościej tworząc funkcję Init7Seg() i inicjując cały blok PIN-ów jak w poprzednim sprawozdaniu i również ten program będzie działać bo wszystkie kolumny wyświetlacza wykonują tę samą czynność i działają jednocześnie ale można to również było napisać

rozkładając na czynniki pierwsze jak powyżej bo np jakbyś chcieli aby 4 kolumna inaczej działa wtedy taki kod elastycznie można przekształcić.

Następnie należało rozdzielić wartość zmiennej zm na jedności, dziesiątki, setki i tysiące tak, aby multipleksowanie działało. Do tego zastosowano wykorzystywanie reszty przy dzieleniu. W następnych krokach przy pomocy delaya wyświetlano odpowiednią część w odpowiedniej kolumnie wyświetlacza przy pomocy funkcji wyświetl(cyfrę) i odpowiedniej wartości jedności, dziesiątek, setek lub tysięcy.

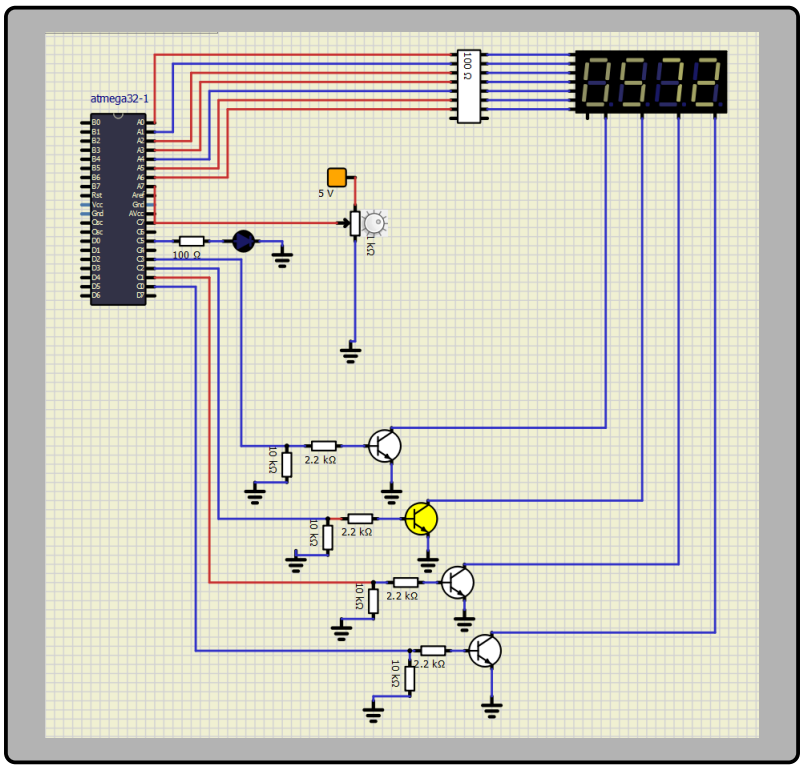
```
    int a=zm/1000;

    int b=(zm-a*1000)/100;
    int c=(zm-a*1000-b*100)/10;
    int d=zm-a*1000-b*100-c*10;

    ON_1;
    wyswietlacz(a);
    _delay_ms(10);
    OFF_1;
    ON_2;
    wyswietlacz(b);
    _delay_ms(10);
    OFF_2;
    ON_3;
    wyswietlacz(c);
    _delay_ms(10);
    OFF_3;
    ON_4;
    wyswietlacz(d);
    _delay_ms(10);
    OFF_4;
}

return 0;
```

Rysunek 7. Kod wyświetlania odpowiedniej wartości w odpowiedniej kolumnie, w odpowiednim czasie

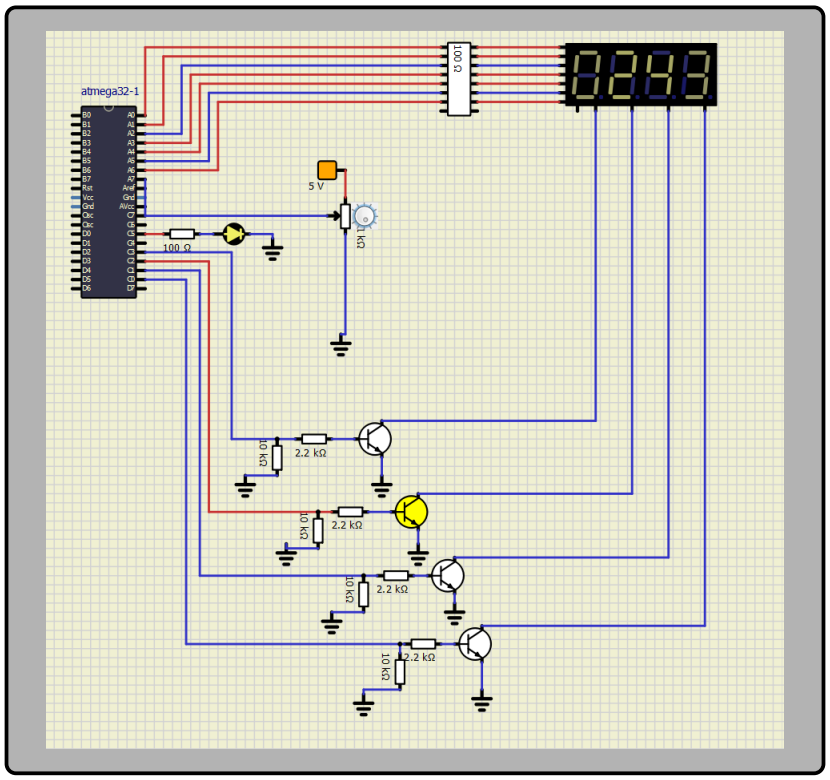
2.2 Symulacja

Poniżej jest zaprezentowany poprawnie działający program.

Jego wizualizacja znajduje się w folderze naszym jako gif. nie dało się go załączyć bo od 2021 Adobe Acrobat nie wspiera multimediów tego rodzaju dla plików pdf



Rysunek 8. Działanie programu poza zakresem 512

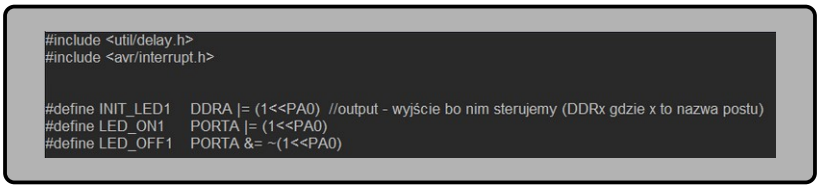


Rysunek 9. Działanie programu w zakresie

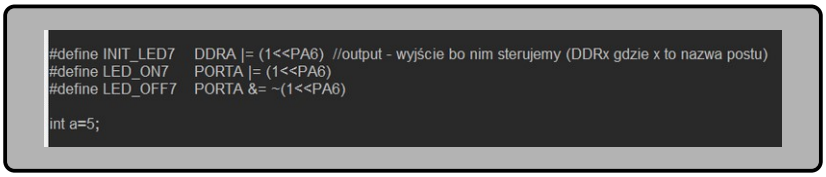
3 Counter na przerwaniach

3.1 Inicjalizacja

Zadanie polegało na przerobieniu projektu countera z wcześniejszych zajęć na taki, który działał by na przerwaniach. Na samym początku zostały zainicjowane diody i przyciski analogicznie jak we wcześniejszych projektach, plus int a, zmienna od cyfry która będzie zmieniana.

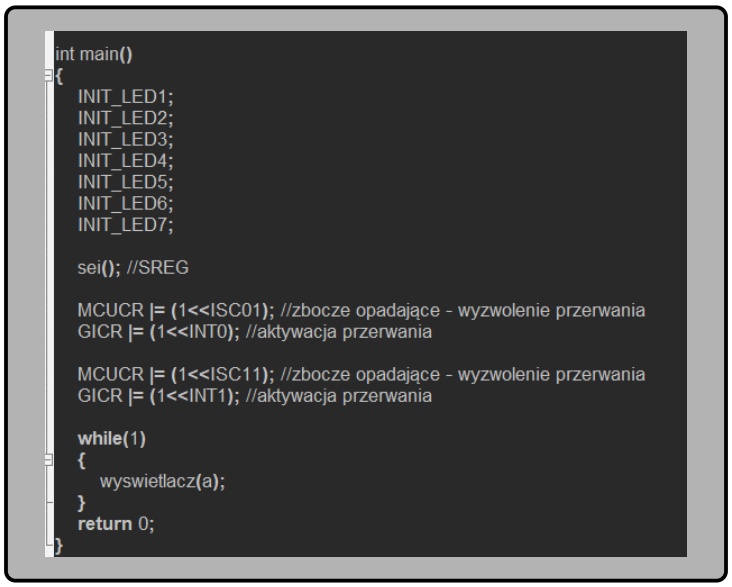


Rysunek 10. Inicjalizacja 1 elementu i bibliotek przerwań i opóźnień



Rysunek 11. Inicjalizacja ostatniego elementu sterującego diodą i naszej zmiennej a

Dalej korzystamy z funkcji switch wyświetl(cyfra) ze wcześniejszych zadań. W głównej części kodu po deklaracji każdej diody wybieramy odpowiednie piny i deklarujemy wyzwolenia przerwania oraz jej aktywacje dla rejestru INT0, jak i dla INIT1.



Rysunek 12. Kod int main()

Tym razem w pętli while(1) jest wywoływana jedynie funkcja, która będzie wyświetlała odpowiednią wartość zmiennej a na wyświetlaczu 7 segmentowym. Zmiana zmiennej a jest zależna od wcześniej zadeklarowanych operacji w blokach funkcyjnych INIT0 i INIT1

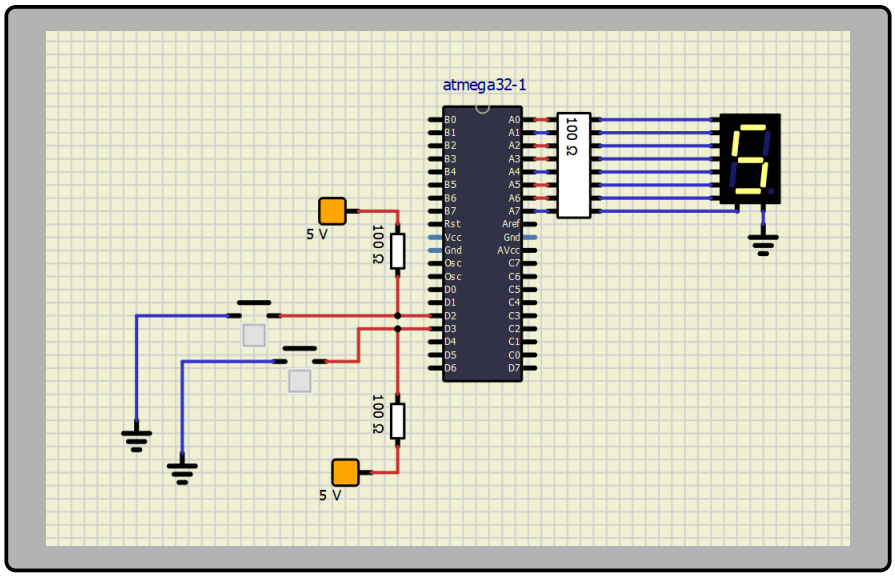
```
ISR(INT0_vect)

{
   if(a==0)
   {
      a=a;
   }
   else
   {
      a--;
   }
}

ISR(INT1_vect)

{
   if(a==9)
   {
      a=a;
   }
   else
   {
      a++;
   }
}
```

Rysunek 13. Inicjalizacja Operacji w blokach funkcyjnych dla zmiennej przy działaniu przerwań

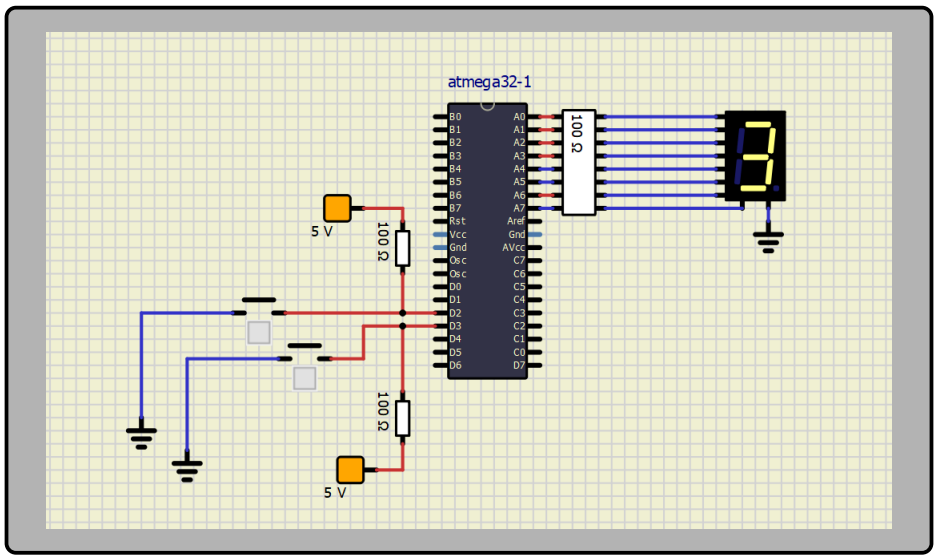
3.2 Symulacja

Poniżej jest zaprezentowany poprawnie działający program.

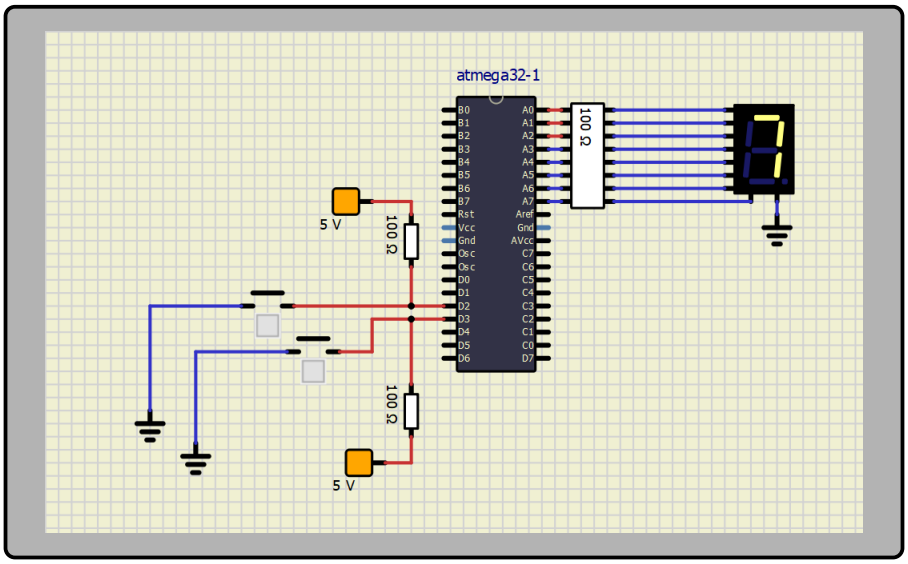
Jego wizualizacja znajduje się w folderze naszym jako gif.



Rysunek 14. Działanie programu na starcie



Rysunek 15. Działanie programu po wciśnięciu 2 razy counter w dół



Rysunek 16. Działanie programu po wciśnięciu 4 razy counter w góre

4 Timer

4.1 Inicjalizacja

Zadaniem doprowadzić aby nasz program z lab1 tudzież lab2 działał samoistnie na bazie przerwań. Czyli nasza dioda miała migać z częstotliwością raz na 1 sekundę. I akcja miała się wykonywać w kółko bez żadnej interwencji w przyciski wejścia czy akcje.

```
#define F_CPU 8000000L // predkosc micropc

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

#define INIT_LED_PC0 DDRC |= (1<<PC0)
#define LED_ON  PORTC |= (1<<PC0)
#define LED_OFF PORTC &= ~(1<<PC0)
```

Rysunek 17. Bilboteki i PRE Dyrektywy

Jak widać zaczyna od dyrektyw PRE Procesora. Warto zadeklarować sobie dyrektywę z prędkością procesora. Oraz dodać odpowiednie bibliotek dla Timera i przerwań. Nadto klasycznie inicjalizujemy PINy sterujące diodą + (wł/wył)

```
int counter= 125;

ISR(TIMER0_OVF_vect)
{

    TCNT0 = 5;
    if(!counter--)
    {
        if(PINC & (1<<PC0))
            LED_OFF;
        else
            LED_ON;
        counter= 125;
    }
}
```

Rysunek 18. Blok funkcyjny odpowiedzialny za poprawną akcję diody

Istotą przy działaniu tych operacji było prawidłowe dobranie wartości countera, który odczytaliśmy z otrzymanego pliku Xlsx. Parametry zostały dobrane do Mikroprocesora taktującego na 8MHz. Istotą działania jest tutaj zapewnienie aby counter był nastawiany z powrotem na 125 inaczej operacja zapalania diody by nie działała.

F_CPU	F_CPU	preskalar	F_PCU/prescalar [Hz]	T [s]	T [ms]	T [us]	licznik [s]	licznik [ms]	licznik [us]	licznik
MHZ							250			[Hz]
8	8000000	1	8000000	0,000000125	0,000125	0,125	0,00003125	0,03125	31,25	32000
	8000000	8	1000000	0,000001	0,001	1	0,00025	0,25	250	4000
	8000000	32	250000	0,000004	0,004	4	0,001	1	1000	1000
	8000000	64	125000	0,000008	0,008	8	0,002	2	2000	500
	8000000	128	62500	0,000016	0,016	16	0,004	4	4000	250
	8000000	256	31250	0,000032	0,032	32	0,008	8	8000	125
	8000000	1024	7812,5	0,000128	0,128	128	0,032	32	32000	31,25
fzadane[kHz]	10000			0,0001	0,1	100				

Rysunek 19. Dobór Parametru

```c
int main(void)
{
    INIT_LED_PC0;
    sei();

    TCCR0 |= (1<<CS02);     //preskaler na 256
    TCNT0 = 5;              //wartość początkowa licznika na 5 (liczymy o 5 do 255)
    TIMSK |= (1<<TOIE0);    //włączenie przerwań

    while(1);
}
```

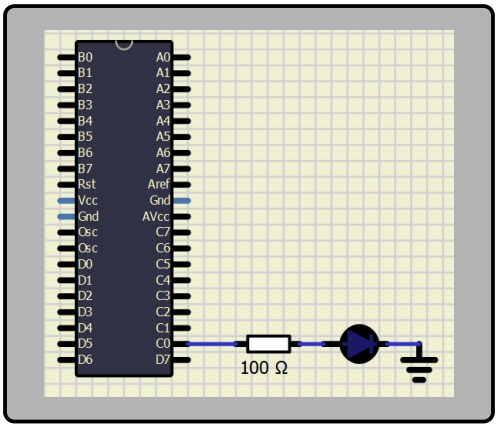
Rysunek 20. Kod działającej w nieskończoność funkcji main

Istotą jest tu nastawienie odpowiednich wartości dla działania przerwań i ich odpalenie. Ich wartość również były użyte do wyznaczenia parametru countera i i uwalniania impulsów diody w czasie 8ms.

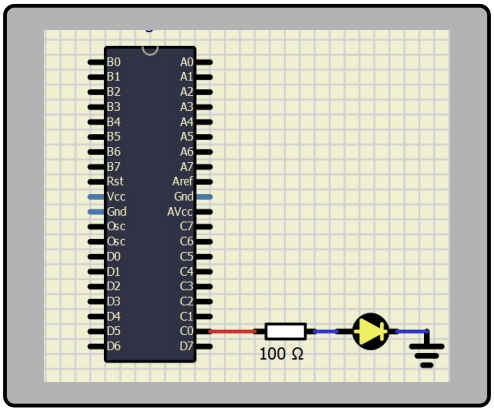
4.2 Symulacja

Poniżej jest zaprezentowany poprawnie działający program.

Jego wizualizacja znajduje się w folderze naszym jako gif.

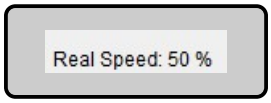


Rysunek 21. Schemat działania po uruchomieniu



Rysunek 22. Moment migania diody

Tutaj cenna uwaga widać to tylko na wizualizacji w gifie dołączonej do naszego folderu. Dioda miga co 2 sec. Wynika to z faktu iż symulacja opóźnia działa. Let's Prove it.



Rysunek 23. Real Speed

5 Bibliografia

- Laboratorium + Excel_dobieranie_parametrów
- Interrupt_Library
- Atmega32_Datasheet

6 Podsumowanie

- Widać przerwania mają bardzo wiele zastosowań i w ogóle Technika Mikroprocesorów. Tutaj istotne jest odpowiednie dobranie elementów elektronicznych oraz właściwe zaprogramowanie mikroprocesora które byśmy oprogramowali za pomocą programatora. A tak to można sobie przetestować program zdalnie. Choć nie zawsze to działa dla pewnych wyjątków. Jak te prezentowane podczas ostatniego laboratorium
- Jednym z zastosowań przerwań do migania diody i rozszerzonej palety rejestrów oraz bloków funkcyjnych zapewne można dostrzec przy działaniu Semaforów kolejowych które nadto że wykonują pewne procedury to reagują jeszcze na pewne warianty typu timery i czujniki ruchu 2 semafory dalej w przypadku ruchu pojazdów kolejowych. Choć zapewne Za programowanie takiego systemu odpowiadają program pisane w języku LAD tudzież LOGO! to jednak można by było to wykorzystać jako aspekt działania diody w ujęciu elektronicznym. Można tu bardzo pofantazjować.
- Przerwania w timerach bardzo ułatwiają prace i pisanie kodów bo program "nie stoi w miejscu" i czeka aż się skończy czas odliczania (np. delay), tylko w między czasie może wykonywać kilka innych czynności nie przejmując się timerem.

7 Całe kody

7.1 Zad 4

```c
#include <avr/io.h>
#include <util/delay.h>
#define INIT_LED    DDRC |= (1<<PC5)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON      PORTC |= (1<<PC5)
#define LED_OFF     PORTC &= ~(1<<PC5)

#define INIT_LED1   DDRA |= (1<<PA0)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON1         PORTA |= (1<<PA0)
#define LED_OFF1    PORTA &= ~(1<<PA0)

#define INIT_SW1    DDRC &= ~(1<<PC3) // nie ma rezystora wiec pull-up
#define PULLUP_SW1  PORTC |= (1<<PC3)

#define INIT_LED2   DDRA |= (1<<PA1)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON2         PORTA |= (1<<PA1)
#define LED_OFF2    PORTA &= ~(1<<PA1)

#define INIT_SW2    DDRD &= ~(1<<PD7) // nie ma rezystora wiec pull-up
#define PULLUP_SW2  PORTD |= (1<<PD7)

#define INIT_LED3   DDRA |= (1<<PA2)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON3         PORTA |= (1<<PA2)
#define LED_OFF3    PORTA &= ~(1<<PA2)

#define INIT_LED4   DDRA |= (1<<PA3)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON4         PORTA |= (1<<PA3)
#define LED_OFF4    PORTA &= ~(1<<PA3)

#define INIT_LED5   DDRA |= (1<<PA4)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON5         PORTA |= (1<<PA4)
#define LED_OFF5    PORTA &= ~(1<<PA4)

#define INIT_LED6   DDRA |= (1<<PA5)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON6         PORTA |= (1<<PA5)
#define LED_OFF6    PORTA &= ~(1<<PA5)

#define INIT_LED7   DDRA |= (1<<PA6)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON7         PORTA |= (1<<PA6)
#define LED_OFF7    PORTA &= ~(1<<PA6)

#define INIT_1  DDRC |= (1<<PC3)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_1        PORTC |= (1<<PC3)
#define OFF_1   PORTC &= ~(1<<PC3)

#define INIT_2  DDRC |= (1<<PC2)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_2        PORTC |= (1<<PC2)
#define OFF_2   PORTC &= ~(1<<PC2)

#define INIT_3  DDRC |= (1<<PC1)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_3        PORTC |= (1<<PC1)
#define OFF_3   PORTC &= ~(1<<PC1)

#define INIT_4  DDRC |= (1<<PC0)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define ON_4        PORTC |= (1<<PC0)
#define OFF_4   PORTC &= ~(1<<PC0)

```

```
57  void wyswietlacz(int cyfra) {
58    //Instrukcja switch ustawia odpowiednie stany na wyjsciach
59    //w zaleznosci od podanej cyfry
60    switch (cyfra) {
61      case 0:
62              LED_ON1;
63              LED_ON2;
64              LED_ON3;
65              LED_ON4;
66              LED_ON5;
67              LED_ON6;
68              LED_OFF7;
69      break;
70      case 1:
71              LED_OFF1;
72              LED_ON2;
73              LED_ON3;
74              LED_OFF4;
75              LED_OFF5;
76              LED_OFF6;
77              LED_OFF7;
78      break;
79      case 2:
80              LED_ON1;
81              LED_ON2;
82              LED_OFF3;
83              LED_ON4;
84              LED_ON5;
85              LED_OFF6;
86              LED_ON7;
87      break;
88      case 3:
89              LED_ON1;
90              LED_ON2;
91              LED_ON3;
92              LED_ON4;
93              LED_OFF5;
94              LED_OFF6;
95              LED_ON7;
96      break;
97      case 4:
98              LED_OFF1;
99              LED_ON2;
100             LED_ON3;
101             LED_OFF4;
102             LED_OFF5;
103             LED_ON6;
104             LED_ON7;
105
106     break;
107     case 5:
108             LED_ON1;
109             LED_OFF2;
110             LED_ON3;
111             LED_ON4;
112             LED_OFF5;
```

```
110             LED_ON3;
111             LED_ON4;
112             LED_OFF5;
113             LED_ON6;
114             LED_ON7;
115
116     break;
117     case 6:
118             LED_ON1;
119             LED_OFF2;
120             LED_ON3;
121             LED_ON4;
122             LED_ON5;
123             LED_ON6;
124             LED_ON7;
125     break;
126     case 7:
127             LED_ON1;
128             LED_ON2;
129             LED_ON3;
130             LED_OFF4;
131             LED_OFF5;
132             LED_OFF6;
133             LED_OFF7;
134     break;
135     case 8:
136             LED_ON1;
137             LED_ON2;
138             LED_ON3;
139             LED_ON4;
140             LED_ON5;
141             LED_ON6;
142             LED_ON7;
143     break;
144     case 9:
145             LED_ON1;
146             LED_ON2;
147             LED_ON3;
148             LED_ON4;
149             LED_OFF5;
150             LED_ON6;
151             LED_ON7;
152     break;
153   }
154 }
155
156 int pomiar(void)
157 {
158     ADCSRA |= (1<<ADSC); //start konwersji
159     while(ADCSRA & (1<<ADSC)); //WHILE NA ZERACH KONCZY ZADANIE
160     return ADC;
161 }
162
163 int main()
164 {
165     INIT_LED;
```

```c
int main()
{
    INIT_LED;
    INIT_LED1;
    INIT_LED2;
    INIT_LED3;
    INIT_LED4;
    INIT_LED5;
    INIT_LED6;
    INIT_LED7;
    INIT_1;
    INIT_2;
    INIT_3;
    INIT_4;
    ADMUX |= (1<<REFS0); //ustawienie napiecia ref na napiecie zasilania
    ADMUX |= ((1<<MUX0)| (1<<MUX1) | (1<<MUX2)); //wybor kana□u, w tym wypadku kana□ 7 (00111)

    ADCSRA |= (1<<ADEN); // uruchomienie przetwornika ADC
    ADCSRA |= ((1<<ADPS0) | (1<<ADPS1)); //Ustawienie preskalera na 8 (011) - ustaw czestotliwo□□ na 8

    while(1)
    {
        int zm=pomiar();
        if(zm<512)
        {
            LED_ON;
        }
        else
        {
            LED_OFF;
        }

        int a=zm/1000;

        int b=(zm-a*1000)/100;
        int c=(zm-a*1000-b*100)/10;
        int d=zm-a*1000-b*100-c*10;

        ON_1;
        wyswietlacz(a);
        _delay_ms(10);
        OFF_1;
        ON_2;
        wyswietlacz(b);
        _delay_ms(10);
        OFF_2;
        ON_3;
        wyswietlacz(c);
        _delay_ms(10);
        OFF_3;
        ON_4;
        wyswietlacz(d);
        _delay_ms(10);
        OFF_4;
    }
    return 0;
}
```

7.2 Zad 5

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


#define INIT_LED1   DDRA |= (1<<PA0)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON1         PORTA |= (1<<PA0)
#define LED_OFF1    PORTA &= ~(1<<PA0)

#define INIT_SW1    DDRC &= ~(1<<PC3) // nie ma rezystora wiec pull-up
#define PULLUP_SW1  PORTC |= (1<<PC3)

#define INIT_LED2   DDRA |= (1<<PA1)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON2         PORTA |= (1<<PA1)
#define LED_OFF2    PORTA &= ~(1<<PA1)

#define INIT_SW2    DDRD &= ~(1<<PD7) // nie ma rezystora wiec pull-up
#define PULLUP_SW2  PORTD |= (1<<PD7)

#define INIT_LED3   DDRA |= (1<<PA2)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON3         PORTA |= (1<<PA2)
#define LED_OFF3    PORTA &= ~(1<<PA2)

#define INIT_LED4   DDRA |= (1<<PA3)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON4         PORTA |= (1<<PA3)
#define LED_OFF4    PORTA &= ~(1<<PA3)

#define INIT_LED5   DDRA |= (1<<PA4)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON5         PORTA |= (1<<PA4)
#define LED_OFF5    PORTA &= ~(1<<PA4)

#define INIT_LED6   DDRA |= (1<<PA5)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON6         PORTA |= (1<<PA5)
#define LED_OFF6    PORTA &= ~(1<<PA5)

#define INIT_LED7   DDRA |= (1<<PA6)  //output - wyj◆cie bo nim sterujemy (DDRx gdzie x to nazwa postu)
#define LED_ON7         PORTA |= (1<<PA6)
#define LED_OFF7    PORTA &= ~(1<<PA6)

int a=5;

void wyswietlacz(int cyfra) {
  //Instrukcja switch ustawia odpowiednie stany na wyjsciach
  //w zaleznosci od podanej cyfry
  switch (cyfra) {
    case 0:
            LED_ON1;
            LED_ON2;
            LED_ON3;
            LED_ON4;
            LED_ON5;
            LED_ON6;
            LED_OFF7;
    break;

```

```
    case 1:
            LED_OFF1;
            LED_ON2;
            LED_ON3;
            LED_OFF4;
            LED_OFF5;
            LED_OFF6;
            LED_OFF7;
    break;
    case 2:
            LED_ON1;
            LED_ON2;
            LED_OFF3;
            LED_ON4;
            LED_ON5;
            LED_OFF6;
            LED_ON7;
    break;
    case 3:
            LED_ON1;
            LED_ON2;
            LED_ON3;
            LED_ON4;
            LED_OFF5;
            LED_OFF6;
            LED_ON7;
    break;
    case 4:
            LED_OFF1;
            LED_ON2;
            LED_ON3;
            LED_OFF4;
            LED_OFF5;
            LED_ON6;
            LED_ON7;

    break;
    case 5:
            LED_ON1;
            LED_OFF2;
            LED_ON3;
            LED_ON4;
            LED_OFF5;
            LED_ON6;
            LED_ON7;

    break;
    case 6:
            LED_ON1;
            LED_OFF2;
            LED_ON3;
            LED_ON4;
            LED_ON5;
            LED_ON6;
            LED_ON7;
    break;
```

```
    case 7:
            LED_ON1;
            LED_ON2;
            LED_ON3;
            LED_OFF4;
            LED_OFF5;
            LED_OFF6;
            LED_OFF7;
    break;
    case 8:
            LED_ON1;
            LED_ON2;
            LED_ON3;
            LED_ON4;
            LED_ON5;
            LED_ON6;
            LED_ON7;
    break;
    case 9:
            LED_ON1;
            LED_ON2;
            LED_ON3;
            LED_ON4;
            LED_OFF5;
            LED_ON6;
            LED_ON7;
    break;
  }
}

ISR(INT0_vect)
{
    if(a==0)
    {
        a=a;
    }
    else
    {
        a--;
    }
}

ISR(INT1_vect)
{
    if(a==9)
    {
        a=a;
    }
    else
    {
        a++;
    }
}

int main()
{
```

```
int main()
{
    INIT_LED1;
    INIT_LED2;
    INIT_LED3;
    INIT_LED4;
    INIT_LED5;
    INIT_LED6;
    INIT_LED7;

    sei(); //SREG
    MCUCR |= (1<<ISC01); //zbocze opadaj?ce - wyzwolenie przerwania
    GICR |= (1<<INT0); //aktywacja przerwania

    MCUCR |= (1<<ISC11); //zbocze opadaj?ce - wyzwolenie przerwania
    GICR |= (1<<INT1); //aktywacja przerwania

    while(1)
    {
        wyswietlacz(a);
    }
    return 0;
}
```

7.3 Zad 6

```
#define F_CPU 8000000L // predkosc micropc

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

#define INIT_LED_PC0 DDRC |= (1<<PC0)
#define LED_ON  PORTC |= (1<<PC0)
#define LED_OFF PORTC &= ~(1<<PC0)

int counter= 125;

ISR(TIMER0_OVF_vect)
{

    TCNT0 = 5;
    if(!counter--)
    {
        if(PINC & (1<<PC0))
            LED_OFF;
        else
            LED_ON;
        counter= 125;
    }
}


int main(void)
{
    INIT_LED_PC0;
    sei();

    TCCR0 |= (1<<CS02);     //preskaler na 256
    TCNT0 = 5;              //wartość początkowa licznika na 5 (liczymy o 5 do 255)
    TIMSK |= (1<<TOIE0);    //włączenie przerwań

    while(1);
}
```