SPRAWOZDANIE

SYSTEMY WBUDOWANE

Układy peryferyjne mikrokontrolera AVR.
Timery.

IMIĘ I NAZWISKO: Jacek Wójcik

NUMER ĆWICZENIA: 6

Grupa laboratoryjna: 10

Data wykonania ćwiczenia: 17.11.2021

Spis treści

1. Zadanie 1 - kod

1.1 main.c

```c
#include <avr/io.h>
#include "lcd.h"

void show_number(int num)       // funkcja wyświetlająca numer
{
    LCD_CLEAR();                // czyszczę wyświetlacz
    char buffer[5];             // tworzę bufor dla funkcji itoa
    itoa(num, buffer, 10);      // przetwarzam kod klawisza na ciąg znaków
    buffer[4]= '\0';            // dodaję pusty znak na koniec żeby utworzyć poprawny ciąg znaków w C
    LCD_WRITE(buffer);          // wyświetlam znak
}

int main(void)
{
    PORTB = 0x01;               // podciągam PINB 0 pod zasilanie

    LCD_INIT();                 // Inicjalizuje wyświetlacz
    show_number(0);

    TCCR0 |= 0x07;              // Włączam timer ustawiając zródło zegara jako zbocze narastające
    //TCCR0 |= 0x06;                // Włączam timer ustawiając zródło zegara jako zbocze opadające

    while (1)
    {
        _delay_ms(200);                 // Czekam 200ms
        if(TIFR & 0x01)                 // Jeżeli wykryję przepełnienie to wyłączam timer
        {
            TCCR0 &= 0xFC;              // Wyłączam timer
        }
        else
        {
            show_number(TCNT0);         // Jeżeli nie ma przepełnienia wyświetlam stan licznika
        }
    }
}

```

1.2 lcd.h

```h
#ifndef LCD_H_
#define LCD_H_

#define F_CPU 1000000               // Ustawiam częstotliwość na zgodną z zestawem w laboratorium

#include <avr/io.h>                 // Dołączam biblioteki do obsługi portów i opóźnień
#include <util/delay.h>

void LCD_CLEAR();                   // Funkcja czyszcząca cały wyświetlacz

void LCD_INIT();                    // Funkcja inicjalizująca wyświetlacz

void LCD_WRITE(char *str);          // Funkcja wypisująca dane na wyświetlacz

void LCD_XY(int x, int y);          // Funkcja ustawiająca kursor na danej pozycji na wyświetlaczu

void LCD_CLEAR_XY(int x, int y);    // Funkcja czyszcząca wyświetlacz od danej pozycji

#endif

```

1.3 lcd.c

```c
#include "lcd.h"

void sendHalfByte(char data)
{
    PORTA |= 0x02;                          // Wysyłam flagę ENABLE
    PORTA = (PORTA & 0x0F) | (data & 0xF0);     // Wysyłam pół bajta
    PORTA &= 0xFD;                          // Usuwam flagę ENABLE
}

void sendByte(char data)
{
    sendHalfByte(data);                     // Wysyłam górną połowę bajtu
    _delay_ms(2);                           // Opóźnienie w celu wykrycia stanu "0" na pinie ENABLE
    sendHalfByte(data << 4);                // Wysyłam dolną połowę bajtu
}

void sendCommand(char data)
{
    sendByte(data);                         // Wysyłam komendę
    _delay_ms(5);                           // Czekam na przetworzenie komendy
}

void sendChar(char data)
{
    sendByte(data);                         // Wysyłam znak
    _delay_ms(2);                           // Czekam na wypisanie znaku
}

void LCD_CLEAR()
{
    PORTA &= 0xFE;                          // Przełączam LCD w tryb wprowadzania komend
    _delay_ms(2);                           // Opóźnienie w celu wykrycia stanu "0" na pinie RS
    sendCommand(0x01);                      // Czyszczę wyświetlacz
    PORTA |= 0x01;                          // Przełączam LCD w tryb wprowadzania danych
}

```

```
36
37  void LCD_INIT()
38  {
39      _delay_ms(15);                          // Czekam aż wyświetlacz LCD zostanie zainicjalizowany
40
41      DDRA |= 0xF3;                           // Ustawiam część linii A
42
43      // Inicjalizacja standardowymi bajtami
44      sendHalfByte(0x30);
45      _delay_ms(5);                           // Czekam zgodnie z dokumentacją
46      sendHalfByte(0x30);
47      _delay_ms(1);
48      sendCommand(0x32);
49
50      // Inicjalizacja ustawień wyświetlacza
51      sendCommand(0x28);                      // Ustawiam tryb 2 linii i znaków 5x8
52      sendCommand(0x08);                      // Wyłączam wyświetlacz zgodnie z dokumentacją
53      sendCommand(0x01);                      // Czyszczę wyświetlacz
54      sendCommand(0x06);                      // Ustawiam kierunek wyprowadzania tekstu i sposób wyprowadzania na wyświetlacz
55      sendCommand(0x0C);                      // Włączam wyświetlacz i wyłączam kursor
56      PORTA |= 0x01;                          // Przełączam LCD w tryb wprowadzania danych
57  }
58
59  void LCD_WRITE(char *str)
60  {
61      while(*str)                             // Dopóki nie dojdę to null terminatora wypisuję kolejne znaki
62      {
63          sendChar(*str++);                   // Przesuwam się do kolejnego znaku po wypisaniu go na wyświetlaczu
64      }
65  }
66
67  void LCD_XY(int x, int y)
68  {
69      PORTA &= 0xFE;                          // Przełączam LCD w tryb wprowadzania komend
70      _delay_ms(2);                           // Opóźnienie w celu wykrycia stanu "0" na pinie RS
71      sendCommand(0x80 | x | y << 6);         // Ustawiam kursor na odpowowiedniej pozycji
72      PORTA |= 0x01;                          // Przełączam LCD w tryb wprowadzania danych
73  }
74
```

```
74
75   void LCD_CLEAR_XY(int x, int y)
76   {
77       LCD_XY(x,y);                                // Ustawiam kursor na pozycji od której chcę czyścić
78       if(y == 0)                                  // Sprawdzam czy to pierwsza linia
79       {
80           while(x++ < 16)                         // Dopóki nie dojdę do końca linii
81           {
82               LCD_WRITE(" ");                     // Wypisuję pusty znak na wyjście wyświetlacza
83           }
84           x = 0;                                  // Ustawiam pierwszą składową adresu na początek linii
85           y++;                                    // Ustawiam drugą składową adresu na drugą linię
86           LCD_XY(x,y);                            // Ustawiam kursor na podany adres -> początek drugiej linii
87       }
88       while(x++ < 16)                             // Dopóki nie dojdę do końca linii
89       {
90           LCD_WRITE(" ");                         // Wypisuję pusty znak na wyjście wyświetlacza
91       }
92   }
93
```

2. Zadanie 1 - opis

Zadanie 1 polegało na napisaniu programu zliczającego kolejne naciśnięcia przycisku podłączonego do linii T0 za pomocą timera 0 działającego w trybie NORMAL i używającego źródła sygnału z linii T0.

Wyświetlanie zliczonych naciśnięć należało wykonać na wyświetlaczu lub linijce LED, ja wybrałem wyświetlacz obsługiwany moją własną biblioteką.

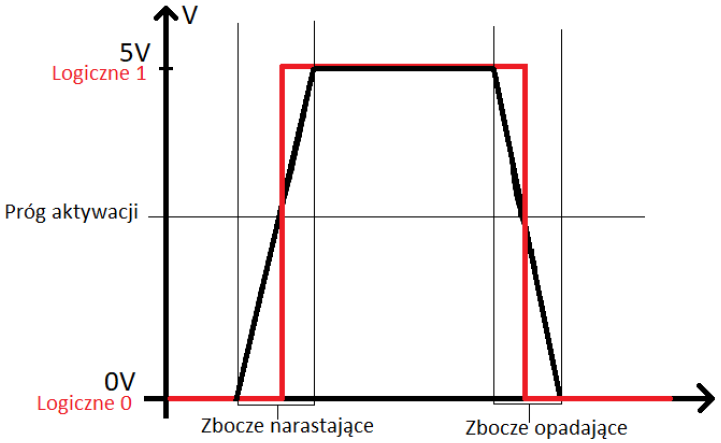
Dodatkowo zadanie posiadało dwa warianty realizacji - korzystanie z sygnału na zboczu narastającym lub opadającym. Wybrałem korzystanie z sygnału na zboczu narastającym, i zakomentowałem instrukcję wyboru zbocza opadającego.

Na początek podciągnąłem linię T0 z do zasilania. Następnie dołączyłem moją bibliotekę do obsługi wyświetlacza LCD. Jej implementacja jest identyczna jak na poprzednich laboratoriach, dlatego nie będę jej tutaj opisywał.

Następnie zainicjalizowałem timer 0. Zgodnie z wymaganiami ustawiłem tryb działania timera na NORMAL i źródło sygnału jako zbocze narastające na linii T0.

Tryb NORMAL oznacza, że timer zinkrementuje rejestr TCNT0 co każdy sygnał z źródła do momentu przepełnienia (dla TCNT0 to wartość 255). Wtedy jest ustawiana flaga TOV0, rejestr TCNT0 jest zerowany i liczenie następuje od nowa.

Źródło sygnału jako zbocze narastające oznacza, że sygnał zostanie wygenerowany w momencie, gdy stan sygnału zmieni się z wartości 0 na wartość 1. Analogicznie zbocze opadające to zmiana sygnału z 1 na 0. Jako że te wartości reprezentowane są jako poziom napięcia w układzie, należy pamiętać że zmiana napięcia nie jest natychmiastowa, ale stopniowa, co jest zaprezentowane na moim szkicu poniżej.



Po inicjalizacji timera przeszedłem do pętli głównej. Tam, zgodnie z poleceniem co 200 ms odmierzane funkcją **_delay_ms(200)** sprawdzałem czy została ustawiona flaga przepełnienia. Jeżeli tak, to zatrzymywałem timer poprzez usunięcie źródła sygnału. Jeżeli nie doszło do przepełnienia, wyświetlać stan licznika przekazując wartość rejestru TCNT0 do mojej funkcji **show_number()**, która zmieniała tę wartość na ciąg znaków i wyświetlała go na wyświetlaczu za pomocą funkcji **LCD_WRITE()** z mojej biblioteki.

3. Zadanie 2 - kod

```c
#define F_CPU 1000000               // Ustawiam częstotliwość na zgodną z zestawem w laboratorium

#include <avr/io.h>                 // Dołączam biblioteki do obsługi portów i opóźnień
#include <util/delay.h>

int main(void)
{
    DDRB = 0x01;                    // Ustawiam PORTB jako wyjście
    PORTB = 0x01;               // Włączam LED

    OCR0 = 146;             // ustawiam timer na odliczenie 150ms

    TCCR0 |= 0x0D;              // Włączam timer z preskalerem 1/1024 w trybie CTC

    while (1)
    {
        if(TIFR & 1 << OCF0)            // Sprawdzam czy timer odliczył wymaganą ilość czasu
        {
            TIFR |= 1 << OCF0;          // Zeruję flagę porównania
            if(PORTB)                   // Jeżeli LED jest zapalony
            {
                PORTB = 0x00;               // Wyłączam LED
                OCR0 = 49;              // ustawiam timer na odliczenie 50ms
            }
            else
            {
                PORTB = 0x01;               // Włączam LED
                OCR0 = 146;             // ustawiam timer na odliczenie 150ms
            }
        }
    }
}

```

4. Zadanie 2 - opis

Zadanie 2 polegało na wygenerowaniu kwadratowego kształtu fali o okresie 200ms i wypełnieniu 75% przy użyciu timera0 w trybie CTC i z preskalerem 1/1024.
Realizacja zadania polegała na 3 elementach:

1. Obliczenie ile czasu w ms będzie trwał stan 1 i stan 0

Znając okres fali (200ms) i wypełnienie (75%), stan wysoki obliczę z wzoru:
`Stan 1 = <okres fali> * <wypełnienie> = 200ms * 75% = 150ms`
Stan 0 będzie zajmował pozostałą część czasu, czyli:
`Stan 0 = 200ms - 150ms = 50ms`

2. Obliczenie wartości rejestru OCR0

Do przeliczenia czasu w ms na wartość OCR0 posłużę się następującym wzorem:
`OCR0 = <czas> * <częstotliwość mikrokontrolera> * <preskaler>`
Dla stanu 1 będzie to:
`OCR0 = 150ms * 1000000Hz * 1/1024 = 146,48 ≈ 146`
Dla stanu 0 będzie to:
`OCR0 = 50ms * 1000000Hz * 1/1024 = 48,82 ≈ 49`
Jako że OCR0 przyjmuje jedynie wartości całkowite, musiałem zaokrąglić wyniki do 146 dla stanu 1 (co daje mi 149,5s) i 49 dla stanu 0 (co daje mi 50,1s). W ten sposób uzyskałem przebieg następujących właściwościach:
Okres: `149,5s + 50,1s = 199,6s`
Wypełnienie: `149,5s / 199,6s * 100% = 74,9%`

3. Zaprogramowanie generatora fali

Na początek muszę podpiąć LED-a do pinu 0 PORTB, żebym mógł zobaczyć przebieg w postaci zapalającego i gasnącego LED-a.
Potem inicjalizuję timer0 - najpierw ustawiam wartość OCR0 na 146, co odpowiada pierwszej części przebiegu (149,5ms). Następnie włączam timer poprzez ustawienie preskalera na 1/1024 oraz trybu timera na tryb CTC.

W nieskończonej pętli while sprawdzam czy doszło do ustawienia flagi porównania timer0. Jeżeli tak, to czyszczę tę flagę, a następnie w zależności od tego, która to faza (czy LED jest włączony czy nie) ustawiam wartość OCR0 odpowiednią dla następnej fazy (czyli jeżeli jesteśmy w fazie 1 - LED jest włączony, to wyłączamy LED-a i ustawiamy OCR0 na czas fazy 2, czyli na wartość 49).
Warto zauważyć, że czyszczenie flagi porównania odbywa się poprzez zapisanie do niej logicznej wartości 1, a nie 0, tak jakbym czyści się rejestry np. PORTB czy DDRB.

5. Wnioski

Podczas tych laboratoriów nauczyłem się następujących rzeczy

1. W trybie normal rejestr TCNT0 jest zerowany w momencie przepełnienia, w trybie CTC w momencie gdy wartość TCNT0 jest równa wartości OCR0
2. Dowiedziałem się że flagi timera czyści się poprzez ustawienie ich na logiczne 1, tak samo jak się je ustawia. Przesłanie logicznego 0 nic nie zmienia niezależnie od stanu flagi, co jest bardzo mylące.
3. Nauczyłem się jak obliczyć czasy poszczególnych faz fali prostokątnej znając jej okres i poziom wypełnienia, a także nauczyłem się jak przeliczać te czasy na wartości, które należy wpisać do rejestrów, żeby wygenerować taką falę za pomocą mikrokontrolera.
4. Nauczyłem się jak wykorzystać klawisz z klawiatury jako samodzielny przycisk - wiersz nr.1 podłączyłem do linii T0 mikrokontrolera a kolumnę nr.1 podłączyłem do zera, przez co po podciągnięciu linii T0 (czyli PINB0) do zerowego potencjału uzyskałem zwyczajny przycisk.
5. Odkryłem ciekawą anomalię: gdy w czasie wykonywania zadania 1 omyłkowo zapomniałem wyłączyć wyświetlanie stanu licznika (czyli rejestru TCNT0) po zatrzymaniu timera, licznik nie zatrzymał się pomimo tego, że timer 1 nie działał. Moja pętla główna z błędem wyglądała następująco:

```
23      while (1)
24      {
25          _delay_ms(200);                 // Czekam 200ms
26          show_number(TCNT0);         // BŁĄD, jednak daje interesujące wyjście
27          if(TIFR & 0x01)                 // Jeżeli wykryję przepełnienie to wyłączam timer
28          {
29              TCCR0 &= 0xFC;              // Wyłączam timer
30          }
31      }
32  }
33
```

Popełniłem błąd zakładając, że po przepełnieniu licznik po prostu powinien się zatrzymać na ostatniej wartości, tj. 255. Jednak zamiast tego licznik za każdym razem zwracał inną wartość z zakresu 0-255. Przypominało to sytuację, jak gdyby ktoś inkrementował licznik z ogromną częstotliwością (znacznie większą niż ta z którą odczytywałem jego wartość) i być może dlatego odczytywanie jego stanu zwracało w zasadzie losowe wartości.

Co ciekawe, identyczne zachowanie udało mi się odwzorować w symulatorze HAPSIM, jednak podczas symulacji krok po kroku odkryłem, że timer0 w momencie, gdy flaga przepełnienia została ustawiona, a timer nie został wyłączony, ustawia swój preskaler na 1/256. To może wyjaśniać, dlaczego w momencie, gdy zastosowałem błędne rozwiązanie, na wyświetlaczu pojawiały się w zasadzie losowe cyfry, tzn. preskaler powodował inkrementację licznika z tak dużą częstotliwością (3905 inkrementacji na sekundę), że odczytywanie jego stanu co 200ms dawało za każdym razem inną liczbę.