Podstawy Techniki Mikroprocesorowej 1 (C/C++)

Martyna Żukowska i Krystian Pacyniak

May 2021

1 Wstep Teoretyczny

Mikrokontrolery Atmega32 są mikrokontrolerami 8-bitowymi, co oznacza że wszystkie rejestry mają pojemność po 8 bitów. Powoduje to, że mogą one wykonywać jedynie proste operacje.

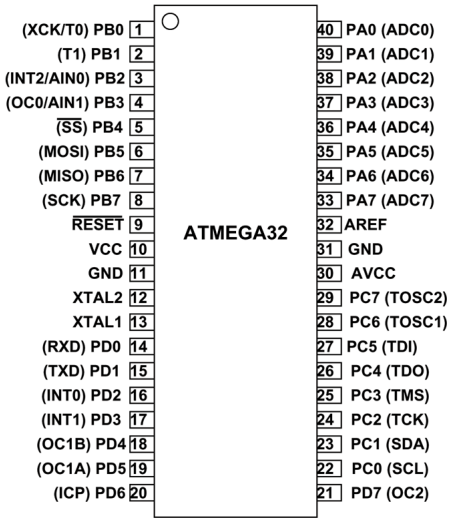
1.1 Operacje bitowe

Operacje bitowe działa na jednym bądź więcej ciągu bitów lub liczbie w zapisie dwójkowym. Między innymi wyróżniamy operacje

- NOT - zaprzeczenie $\sim$
- AND - iloczyn &
- OR - suma |
- Przesuniecie w prawo >>
- Przesuniecie w lewo <<

Przesunięcia bitowe to operacja na liczbach w systemie dwójkowym polegającym na przesunięciu wszystkich pozycji binarnych w lewo/prawo

1.2 Sterowania portami



Rysunek 1. Mikroprocesor na którym działamy

Sterowanie:

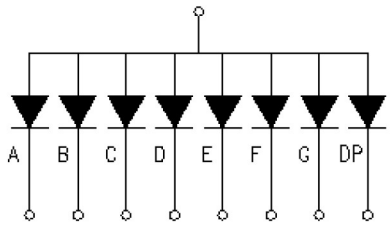
- PortA (wejścia PA0-PA7) polecenie DDRA/PORTA/PINA
- PortB (wejścia PB0-PB7) polecenie DDRB/PORTB/PINB
- PortC (wejścia PC0-PC7) polecenie DDRC/PORTC/PINC
- PortD (wejścia PD0-PD7) polecenie DDRD/PORTD/PIND

Poszczególne polecenia oznaczają

1. Rejestr DDRx - rejestr kierunkowy
   - 0 - ustawienie wprowadzenia jako wejście
   - 1 - ustawienie wprowadzenia jako wyjście
2. Rejestr PORTx - rejestr wyjściowy gdy DDRx==0
   - 0 - stan wysokiej ipedancji - nic się nie dzieje
   - 1 - pull-UP
3. Rejestr PORTx - rejestr wyjściowy gdy DDRx==1
   - 0 - ustawienie wprowadzenia jako stan niski
   - 1 - ustawienie wprowadzenia jako stan wysoki
4. Rejestr PINx - rejestr wejściowy tylko do odczytu
   - 0 - stan niski na wprowadzeniu
   - 1 - stan wysoki na wprowadzeniu

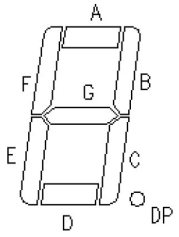
1.3 Wyświetlacz 7 segmentowy

Korzystamy z pojedynczego 7 segmentowego wyświetlacza, którego wewnętrzny schemat wygląda tak:



Rysunek 2.1 Schemat Elektroniczny wyświetlacza 7-seg.

Jak nazwa wskazuje jest to wyświetlacz posiadający 7 diod (A,B,C,D,E,F,G) i dodatkową 8 diodę kropkę. Pozwala to na wyświetlenie wszystkich cyfr i liter alfabetu, za pomocą tylko tych 7 diod, włączając i wyłączając odpowiednie z nich.



Rysunek 2.2 Schemat Graficzny wyświetlacza 7-seg.

2 Cel ćwiczenia

Celem ćwiczenia jest stworzenie countera który za pomocą dwóch przycisków zwiększa i zmniejsza liczbę wyświetlana na wyświetlaczu 7 segmentowym

3 Inicjalizacja portów - przesunięcia bitowe

Po załączeniu odpowiednich bibliotek, zaczynamy kod od zdefiniowania poleceń INIT_LED (ustawienie odpowiedniego portu jako gotowego do pracy), LED_ON (ustawienie stanu wysokiego na odpowiednim wyprowadzeniu), LED_OFF (ustawienie stanu niskiego na odpowiednim wyprowadzeniu). Należy tak zrobić dla wszystkich 8 diod LED.

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

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

Rysunek 3. Przykład kodu dla diody LED 1 - na wyświetlaczu segmentowym dioda LED A podpiętego do PA0

Inicjujemy przełącznik do zwiększania (SW1) i zmniejszania countera (SW2) w podobny sposób, jednak bez definicji ustawienia wyprowadzenia na stan niski, bo przycisk gdy go nie przyciskami sam się ustawia ma stan niski.

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

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

Rysunek 4. Kod dla obu przycisków podpiętych (sterowania) do PD7 i PC3

4 Inicjalizacja portów - kod heksagonalny

Porty można również zainicjalizować poprzez natychmiastowe przypisanie kodu w systemie szesnastkowym. Gotowe wartości w Hex-Value są dostępny tutaj Tabela segmentów.

```
#define ZERO 0x3F
#define ONE 0x06
#define TWO 0x5B
#define THREE 0x4F
#define FOUR 0x66
#define FIVE 0x6D
#define SIX 0x7D
#define SEVEN 0x07
#define EIGHT 0x7F
#define NINE 0x6F
```

Rysunek 5. Kod Hex. ustawiając gotowe liczby na segementarzu

```
void init_7_seg()
{
  DDRA = 0x7F;
  PORTA = 0X00;
}
```

Rysunek 6. Inicjalizacja wyświetlacza 7 segmentowego Wejścia i wyjścia (Portów). Odpowiednik Rysunku 4.

Stosując tę procedurę możemy uniknąć pisania x-dziesiątek linijek kodów i inicjalizacji. Jest to dosyć intuicyjne ponieważ, wynika to z faktu że na wyświetlaczu mamy 8 elementów (w tym 1 kropkę). Jest to analogia do 8-bitów czyli 1 bajta pamięci. Możemy dzięki temu wykorzystać system dwójkowy oraz jego podstawowe operacje. Przesunięcie w lewo lub w prawo to też odpowiednio mnożenie lub dzielenie wartości wejściowej przez 2 do potęgi ilości przesunięć np. operacja binarna.

00000010 « 2 to 00001000 czyli w decymalnym $2 * 2^{(2)}$ to 8.

Badając konkretne bity możemy przejść z systemu binarnego na heksagonalny.
Do inicjalizacji pokazanej na Rysunkach 5 oraz 6 wykorzystałem inicjalizację klasyczną,to znaczy że zapis bitowy reprezentuje sekwencję 'gfedcba' - początek od prawej strony zapisu binarnego.
Jeżeli ma się świecić 2 bez elementu f oraz c jak na rysunku 2.2 to mamy zapis 1011011_2,
gdzie 1 odpowiada zapalaniu się diody, a 0 jej wyłączeniu. Zatem $1011011_2 = 0x5B_{16}$.

I inne symbolicznie:

$8_{wywietlacz} = 1111111_{Bin} = 0x7F_{Hex}$ oraz

$0_{wywietlacz} = 0111111_{Bin} = 0x3F_{Hex}$.

Wyjątek $nic_{wywietlacz} = 0000000_{Bin} = 0x00_{Hex}$

Oczywiście w kodzie binarnym użyliśmy 7 bitów a nie 8. 8 bit stojący za g w sekwencji hebrajskiej to kropka (DP), która w naszym zadaniu nie bierze udziału praktycznego
Strona użyta do przeliczania na CITO systemów własnoręcznie: Calculla

5 Funkcja Switch

```
void display_number(int no)
{
  switch(no)
  {
    case 0:
        PORTA = ZERO;
        break;
    case 1:
        PORTA = ONE;
        break;
    case 2:
        PORTA = TWO;
        break;
    case 3:
        PORTA = THREE;
        break;
    case 4:
        PORTA = FOUR;
        break;
    case 5:
        PORTA = FIVE;
        break;
    case 6:
        PORTA = SIX;
        break;
    case 7:
        PORTA = SEVEN;
        break;
    case 8:
        PORTA = EIGHT;
        break;
    case 9:
        PORTA = NINE;
        break;
    }
}
```

Rysunek 7. Kod funkcji Display number realizujący procedurę wyświetlenia konkretnej cyfr w skróconym zapisie

Na podstawie zadanej wartości będziemy w funkcji głównej będziemy sterować odpowiednimi portami zapalając odpowiednie diody.

6 Realizacja funkcji main

```
INIT_SWD;
PULLUP_SWD;
INIT_SWC;
PULLUP_SWC;
init_7_seg();
// initcjalizacja bramek i portów

int new_number = 0;
```

Rysunek 8.1 Etap I. Inicjalizacja odpowiednich wejść do podjęcia akcji w programie.

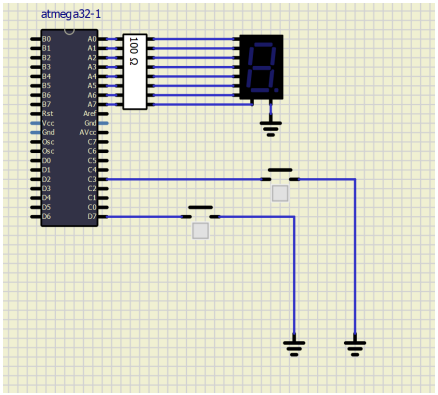
```
while(1)
{
    display_number(new_number);

    if(!(PIND & (1<<PD7)))
    {
        while(!(PIND & (1<<PD7)));
        new_number++;
        if(new_number == 10)
        {
            new_number = 9;
        }
    }
    // inkrementacja segmentarza
    if(!(PINC & (1<<PC3)))
    {
        while(!(PINC & (1<<PC3)));
        new_number--;
        if(new_number == -1)
        {
            new_number= 0;
        }
    }
    // dekrementacja segmentarza
}
```

Rysunek 8.2 Etap II. Główny Algorytm który będzie sterował działaniem wyświetlacza.

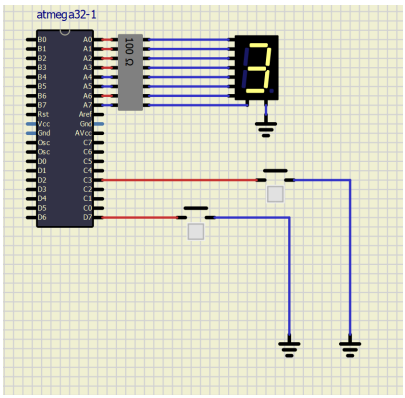
Funkcje odpowiedzialną za działanie programu można było wykonać na kilka sposobów. Postanowiliśmy wybrać taką która najlepiej odzwierciedla algorytm działania programu. Powyższy kod działa tak: Program Zaczyna działać i stale wyświetlać wartość 0. Następnie jeśli Zostanie wciśnięty odpowiedni guzik w programie SIMULIDE, czyli wykona się działanie na konkretnym porcie oraz dopiero ,gdy guzik zostanie puszczony (pętla wewnętrzna while), to wtedy nastąpi odpowiednio inkrementacja lub dekrementacja zmiennej odpowiedzialnej za sterowanie funkcją włączającej konkretną cyfrę na wyświetlaczu. Program wyposażony jest również w bezpieczniki uniemożliwiające po kliknięciu zmianę wartości zmiennej poza zakresem wartości od 0 do 9.

7 Działanie programu - schemat

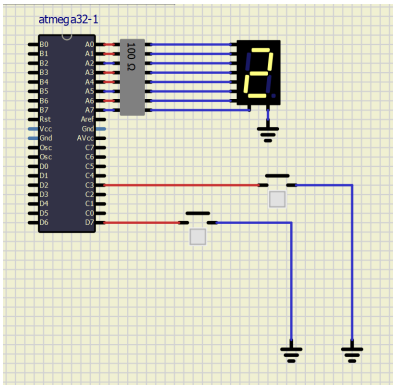
Aby wyświetlacz działał podłączyć go należy do odpowiednich pinów przez rezystor (w tym wypadku 100Ω). Do tego wykorzystany została drabinka rezystancyjna (8 elementowa).



Rysunek 9.1 Schemat układu przed podłączeniem



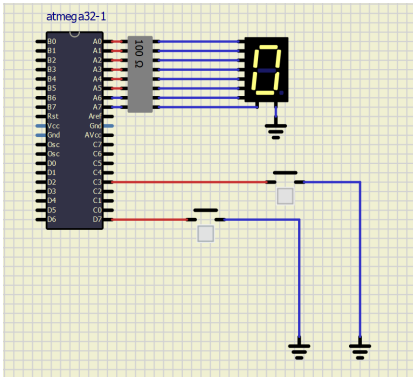
Rysunek 9.2 Schemat układu po trzykrotnym wciśnięciu przycisku zwiększającego wartość



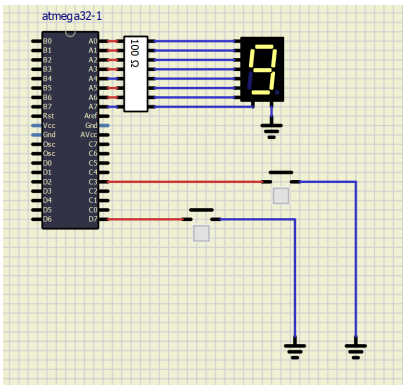
Rysunek 9.3 Schemat układu po jednokrotnym wciśnięciu przycisku zmniejszającego wartość

Dodatkowo została tu sprawdzona ewentualność przetrzymania przycisku i również w tym przypadku counter działa poprawnie (zmniejsza/zwiększa się wartość jednokrotnie).

Został też przeprowadzony testy możliwości wyjścia poza zakres countera, czy wartości cały czas rosną, czy zostają na odpowiedniej wartości 0/9.



Rysunek 9.4 Schemat układu po przekroczeniu zakresu dolnego



Rysunek 9.5 Schemat układu po przekroczeniu zakresu górnego

8 Podsumowanie

- Wykorzystując analogię binarnego zapisu pamięci w 7 tudzież 8 bitach i ilości segmentów badanego urządzenia można użyć gotowego kodu Heksagonalnego do uruchamiania konkretnych segmentów wyświetlacza cyfrowego.
- W programie według założeń pominięta została biblioteka $< util.delay.h >$. Zamiast tego wykorzystano funkcje która pozwala na zatrzymanie programu w wyniku kliknięcia - akcji.
- Wartości cyfr reagują na każdą akcję nie przekraczając zakresu wartości 0-9 co świadczy o jak najbardziej poprawnym wykonaniu zadania. Zadanie jest wykonalne przy oporze 100 jak i 200Ω