LABORATORIUM – ELEKTRONIKA

Multipleksowane sterowanie wyświetlaczy LED/LCD

1. Wstęp

W różnego rodzaju układach bardzo często zachodzi potrzeba sterowania wyświetlaczami LED/LCD. Układy takie w ogólności możemy podzielić na dwa rodzaje. Pierwszy wykorzystuje sterowniki wbudowane w układ wyświetlacza, drugi opiera się o bezpośrednie sterowanie poszczególnymi segmentami wyświetlacza. W tym drugim przypadku, dopóki potrzebna liczba wyprowadzeń nie jest duża i układ sterujący (np. mikrokontroler) dysponuje odpowiednią liczbą pinów wyjściowych, nie ma problemu i takim wyświetlaczem można sterować statycznie, tzn. przykładając napięcie pomiędzy elektrodę wspólną i segment, który ma zostać aktywowany. Technika ta pozwala uzyskać najlepszy kontrast wyświetlacza, niestety wymaganie by każdy segment miał osobne wyprowadzenie jest bardzo ograniczające, np. sterowanie stosunkowo niewielką matrycą 16x10 pikseli wymaga 160 wyprowadzeń (po jednym dla każdego segmentu). Z tego też powodu rzadko stosuje się takie sterowanie, znacznie częściej matryce/wyświetlacze się multipleksuje. Idea takiego podejścia polega na tym, że wyprowadzenia wyświetlacza są dzielone (np. kilka diod LED na jednej linii), a inny układ steruje zasilaniem poszczególnych grup. Odpowiednio szybkie zapalanie i gaszenie poszczególnych grup, w połączeniu z bezwładnością ludzkiego wzroku sprawia, że widz nie zauważa migania wyświetlacza, kosztem jedynie niewielkiego zmniejszenia kontrastu.

Celem ćwiczenia jest zatem zaprogramowanie multipleksowanego sterowania wyświetlaczy optoelektronicznych, na przykładzie sterowania mikrokontrolerowego 4 wyświetlaczy siedmiosegmentowych.

2. Wykonanie ćwiczenia.

2.1. Uruchomić środowisko programistyczne *AVR Studio*. Automatycznie powinno pojawić się okno tworzenia/otwierania projektu. Utworzyć więc nowy projekt, z następującymi opcjami:

- Okno pierwsze:
  - AVR GCC -> nadać nazwę swojemu projektowi, wskazać jego lokalizację (najlepiej Pulpit) zaznaczyć opcje *Create initial file* i *Create folder*.
- Okno drugie:
  - po lewej AVR Simulator, po prawej wybrać używany mikrokontroler (**Atmega32 lub 16 – sprawdzić na obudowie**) -> Finish.

2.2. Po utworzeniu projektu należy jeszcze ustawić częstotliwość taktowania według używanego kwarcu uzupełniając ją tutaj:

✓ Project -> Configuration Options -> Frequency.
(**16 MHz** – trzeba ją wpisać z zerami).

2.3. Za szkielet programu niech posłuży kod pokazany poniżej, tego treść należy wpisać do okna programu:

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

#define WYSW PORTA //gdzie wyświetlacz
#define STER PORTB   //gdzie sterowanie wyświetlaczem

int main(void) //program główny
        {
        // TUTAJ ZNAJDZIE SIĘ KONFIGURACJA
        while(1) //pętla wykonywana w nieskończoność (czyli do resetu lub wyłączenia zasilania)
                {
                        // TUTAJ ZNAJDZIE SIĘ WŁAŚCIWY PROGRAM
                }
        }
```

2.3.1. Jak widać w kodzie kolejno:

- ✓ załączane biblioteki używane w projekcie (#include…),
- ✓ określa się do jakiego portu podłączamy wyświetlacz, a który wykorzystamy do sterowania nim (#define…),

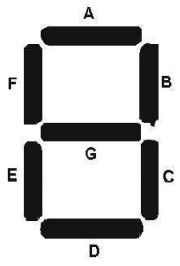
a następnie mamy dostęp do programu głównego (int main(void)), który zawiera miejsce (// TUTAJ KONFIGURACJA) na początkową konfigurację układu (czyli np. określenie trybu działania portów mikrokontrolera) i pętlę while, której zawartość (// TUTAJ PROGRAM) będzie się wykonywała aż do utraty zasilania/resetu.

2.4. Na początku sprawimy, że na jednym z wyświetlaczy zapali się określona liczba. W tym celu należy na porcie A wystawić stan jej odpowiadający, a na porcie B określić który wyświetlacz ma zadziałać. Ale zanim można do tego przystąpić, należy określić tryb działania obu tych portów. Nie chcemy się z wyświetlaczem komunikować, a jedynie mu rozkazywać, to po prostu należy oba porty mikrokontrolera ustawić w tryb wyjściowy. W tym celu we fragmencie kodu określającym początkową konfigurację (// TUTAJ KONFIGURACJA) należy wstawić kod:

```
DDRA = 0xFF;
DDRB = 0xFF;
```

Załóżmy, że piny mikrokontrolera są podłączone do wyświetlaczy w sposób następujący:

pin		segment
7		DP (kropka)
6		G
5		F
4		E
3		D
2		C
1		B
0		A



Zatem teraz jeżeli na port B, instrukcją STER=0xFE wystawimy stan 11111110, czyli szesnastkowo FE*, to zdecydujemy że zapali się wyświetlacz pierwszy (licząc od prawej).

Jeżeli z kolei na port A, instrukcją WYSW=0xF8 wystawimy stan 11111000, czyli szesnastkowo F8, to na pierwszym wyświetlaczu zapali się nam liczba 7 (odpowiadająca zaświeceniu segmentów C, B i A – C, B i A są niejako umieszczone „od końca" w sekwencji 11111000).

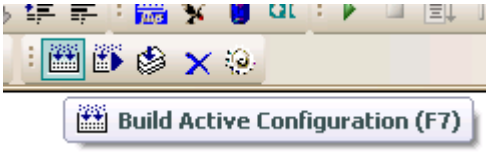
* Dlaczego 1111 1110 = FE? Bo 1111 to dziesiętnie 15, a szesnastkowo F, natomiast 1110 to dziesiętnie 14 czyli szesnastkowo E.

Na tej samej zasadzie przykładowo 01111100 = 7C, ponieważ 0111 = 8*0+4*1+ 2*1 +1*1 = 7, natomiast 1100 = 8*1+4*1+ 2*0 +1*0 = 12 = C.

A zatem, we fragmencie kodu zawierającym właściwy program (// TUTAJ PROGRAM) należy wstawić:

```
STER=0xFE;
WYSW=0xF8;
```

2.5. Sprawdźmy ten program. W tym celu należy go skompilować: Można to wykonać przyciskiem:



Jeżeli program skompilował się bez błędu należy go zgrać go mikrokontrolera. W tym celu należy:

- ✓ uruchomić program PonyProg,
- ✓ kolejno (rysunek poniżej):
  - ustawić rodzinę i typ używanego mikrokontrolera (1) - **typ sprawdzić na obudowie - jak wcześniej**,
  - otworzyć plik programu (2), czyli plik *.hex utworzony w folderze projektu w trakcie wcześniejszej kompilacji,
  - wgrać go do mikrokontrolera (3).

2.6. Jeżeli wszystko zadziałało, możemy przystąpić do właściwej treści ćwiczenia czyli sterowania multipleksowanego. Idea wyświetlania multipleksowego opisana we wstępie do ćwiczenia, w odniesieniu sterowania 4 wyświetlaczami siedmiosegmentowymi sprowadza się w zasadzie do podania na wyjścia portu zapętlonej następującej sekwencji:

1. zasil segment 1,
2. podaj liczbę do wyświetlenia na segmencie pierwszym,
3. zasil segment 2,
4. podaj liczbę…

itd.

Tyle, że sekwencja powinna być wykonywana z wprowadzeniem pewnych opóźnień. Opóźnienie powinno być dobrane na następującej zasadzie: na tyle szybko, by oko nie widziało migania/przełączania wyświetlaczy, ale na tyle wolno, by zdążyły się one przełączać (czyli to co wyświetla się na sąsiednich segmentach nie „nakładało się" na siebie).

Opóźnienia można wprowadzić wywołaniem funkcji (po każdej parze danych wyżej instrukcji, czyli po 2, po 4, itd.):

_delay_ms(n);

Gdzie n= czas opóźnienia (w milisekundach).

Do wykonania:

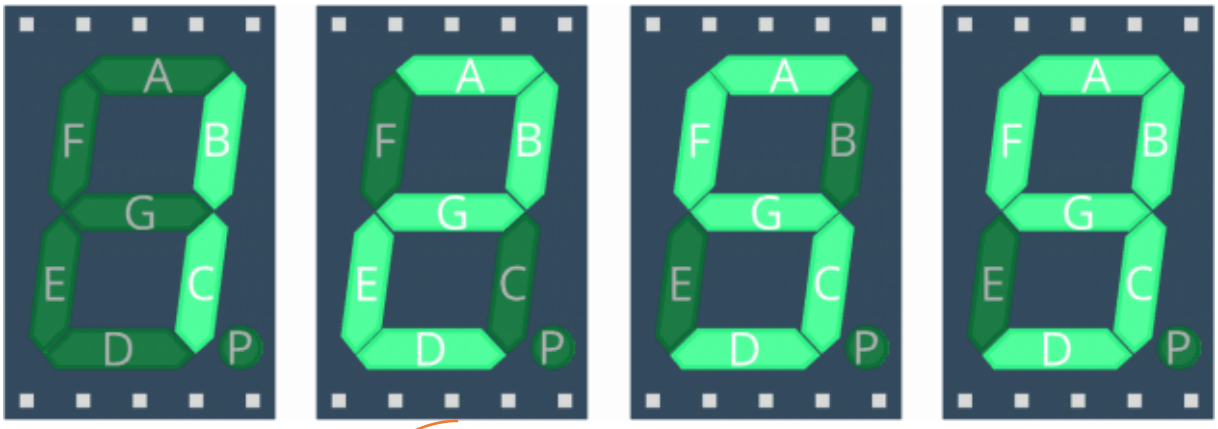
2.7. Wyświetlić na segmentach wyświetlacza aktualną datę w formacie **DD.MM.**

2.8. Zmodyfikować program tak, by wyświetlać na zmianę datę i numer laboratorium, czyli **A219** (po sekundzie każde) – najłatwiej to zrobić korzystając z pętli for, której konstrukcja jest następująca:

2.9.

```
int i;
for(i=0; i<33; i++)
 {
  tu instrukcja(e) do wykonania w pętli (wykona(ją) się 33 razy)
 }
```

EWENTUALNA POMOC:



WYŚWIETLENIE DWÓJKI NA DRUGIEJ POZYCJI (CZYLI TRZECIEJ OD PRAWEJ):

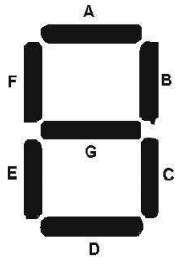
STER=0xFB; // 1111 1**0**11 bo 8*1+4*1+ 2*1 +1*1 =15 = F, a 8*1+4***0**+ 2*1 +1*1 = 11 = B

(zero trzecie od końca, bo właśnie na trzecim od końca segmencie ma się zapalić dwójka)

WYSW=0xA4; //1010 0100 bo 8*1+4*0+ 2*1 +1*0 =10 = A, a 8*0+4*1+ 2*0 +1*0 = 4 = 4

(zera na miejscach segmentów, które się mają świecić, więc, by wyświetlić dwójkę, będą to segmenty G, E, D, B i A):

pin		segment
7		DP (kropka) = 1
6		G = 0
5		F = 1
4		E = 0
3		D = 0
2		C = 1
1		B = 0
0		A = 0



3. Literatura

[1] Hadam P.: *Projektowanie systemów mikroprocesorowych*, Wydawnictwo btc, Warszawa 2004

[2] *Elektronika Praktyczna Plus - Displays*, Wydawnictwo AVT, Warszawa 2007

Opracowanie ćwiczenia: Seweryn Lipiński