

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 zapalenie 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 (najprawdopodobniej Atmega32) -> Finish.

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

- ✓ Project -> Configuration Options -> Frequency.
(najprawdopodobniej 8, 12 lub 16 MHz – trzeba ją wpisać z zerami).

2.3. Za szkielet programu niech posłuży szablon dostarczony przez prowadzącego (najprawdopodobniej na pulpicie), jego zawartość wkleić do okna swojego projektu:

```
#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 KONFIGURACJA

    while(1) //pętla wykonywana w nieskończoność (czyli do resetu lub wyłączenia zasilania)
    {
        // TUTAJ PROGRAM
    }
}
```

2.3.1. Jak widać szablon kolejno:

- ✓ załącza biblioteki używane w projekcie (#include...),
- ✓ określa 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.

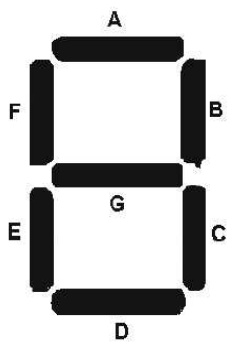
2.4. Na początku sprawimy, że na jednym z wyświetlaczy zapali się określona liczba. W tym celu należy na porcie A wystawić stan jej odpowiadający, a na porcie B określić który wyświetlacz ma zadziałać. Ale zanim można do tego przystąpić, należy określić tryb działania obu tych portów. Ponieważ nie chcemy 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 (pod "//TUTAJ KONFIGURACJA") należy wstawić kod:

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

2.5. Poraysterować wyświetlacze, tzn. zdecydować **który** wyświetlacz zadziała i **co** się na nim wyświetli.

Założmy, 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 jeżeli na port B wystawimy stan 11111110, czyli szesnastkowo FE*, to zdecydujemy że zapali się wyświetlacz pierwszy od prawej (decyduje o tym 0 na końcu tego bajtu).

Jeżeli **jednocześnie** na port A wystawimy stan 11111000, czyli szesnastkowo F8, to na wyświetlaczu wybranym przed chwilą zapali się nam liczba 7 (odpowiadająca zaświeceniu segmentów A, B i C, czyli tym bitom, dla których wystawione zostały zera).

* Dlaczego 1111 1110 = FE? Bo 1111 to binarnie 15, a szesnastkowo F, natomiast 1110 to binarnie 14 i szesnastkowo E. Na tej samej zasadzie 01111100 = 7C, ponieważ 0111 = 7, natomiast 1100 = 12 = C.

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

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

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



Jeżeli program skompilował się bez błędów należy go zgrać na mikrokontroler. W tym celu należy:

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



2.7. Jeżeli wszystko zadziało, 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 (fragment pogrubiony został już zrealizowany w punkcie 2.5, więc do Was należy rozbudowa programu):

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:

```
_delay_ms(n);
```

Gdzie n= czas opóźnienia (w milisekundach - można go dobrać eksperymentalnie, sprawdzając efekty różnych opóźnień, da się go też dobrać analitycznie).

Do wykonania:

2.8. Wyświetlić na segmentach wyświetlacza aktualną datę w formacie DDMM.

2.9. Zmodyfikować program tak, by wyświetlać na zmianę datę i numer laboratorium, czyli A219 (po sekundzie każde) - w realizacji tego podpunktu pomoc może skorzystać z pomocy na następnej stronie, najsensowniejsza wydaje się instrukcja **for**, ale to Wasz wybór.

Ewentualna pomoc:

```
int i;
for(i=0; i<50; i++)
{
    tu instrukcja(e) do wykonania w pętli (wykona(ją) się 50 razy)
}
```

```
int i=0;
do
{
    tu instrukcja(e) do wykonania w pętli (wykona(ją) się 51 razy)
    i++;
}
while(i <= 50);
```

```
int i;
if(i>5)
{
    to, co ma się stać jeśli i jest większe od 5
}
else
{
    to, co ma się stać jeśli nie
}
```

```
int i;
switch(i)
{
    case 1:
        to, co ma się dziać, gdy i=1
        break;
    case 2:
        to, co ma się dziać, gdy i=2
        break;
    //(case'ów może być oczywiście więcej...)
    default:
        to, co ma się dziać, jeśli żaden z wcześniejszych warunków nie
        będzie spełniony
}
```

3. Literatura

- [1] Hadam P.: *Projektowanie systemów mikroprocesorowych*, Wydawnictwo btc, Warszawa 2004
- [2] *Elektronika Praktyczna Plus - Displays*, Wydawnictwo AVT, Warszawa 2007