Ćwiczenie 4: Wyświetlacze siedmiosegmentowe LED

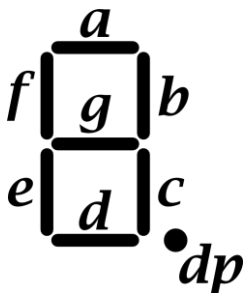
Zakres materiału

1. Budowa wyświetlacza siedmiosegmentowego.
2. Typy wyświetlaczy siedmiosegmentowych.
3. Wyświetlanie multipleksowane.
4. Sposoby podłączenia różnych wyświetlaczy LED do mikrokontrolera.

Wprowadzenie do ćwiczeń

1. Wyświetlacz siedmiosegmentowy LED

Wyświetlacz siedmiosegmentowy zbudowany jest z siedmiu diod LED ustawionych w kształcie cyfry „8”. Wyświetlenie żądanego znaku wymaga zapalenia odpowiednich segmentów. Budowa wyświetlacza została przedstawiona na rysunku poniżej.



PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
dp	*g*	*f*	*e*	*d*	*c*	*b*	*a*

Segmenty wyświetlacza w przykładach podłączone są do portu GPIOB tak, jak pokazano na rysunku. Kod, który należy wysłać do portu, aby wyświetlić żądany znak, określony jest za pomocą kombinacji jedynek (segment zgaszony) i zer (segment zapalony). Na przykład do wyświetlenia cyfry 2 należy zgasić segmenty *a*, *b*, *d*, *e* i *g*, co daje kod B'10100100' (0xA4). Jest to oczywiście słuszne dla wyświetlaczy ze wspólną anodą, które zastosowano na płytce. Dla wyświetlaczy ze wspólną katodą należałoby zanegować kod cyfry.

Tablice konwersji służą do zamiany wartości, będącej indeksem tablicy, na kod. Są zazwyczaj one statyczne (ich zawartość nie zmienia się w trakcie wykonywania programu). Można je zrealizować przy pomocy tablicy lub specjalnej funkcji konwertującej (Przykład 1).

Aby wyświetlić znak na pojedynczym wyświetlaczu należy na linie segmentów podać odpowiedni kod znaku oraz aktywować wyświetlacz na którym znak ma być wyświetlony przez podanie stanu niskiego na linię sterującą anodą wyświetlacza (w przypadku wyświetlacza ze wspólną katodą – stanu wysokiego).

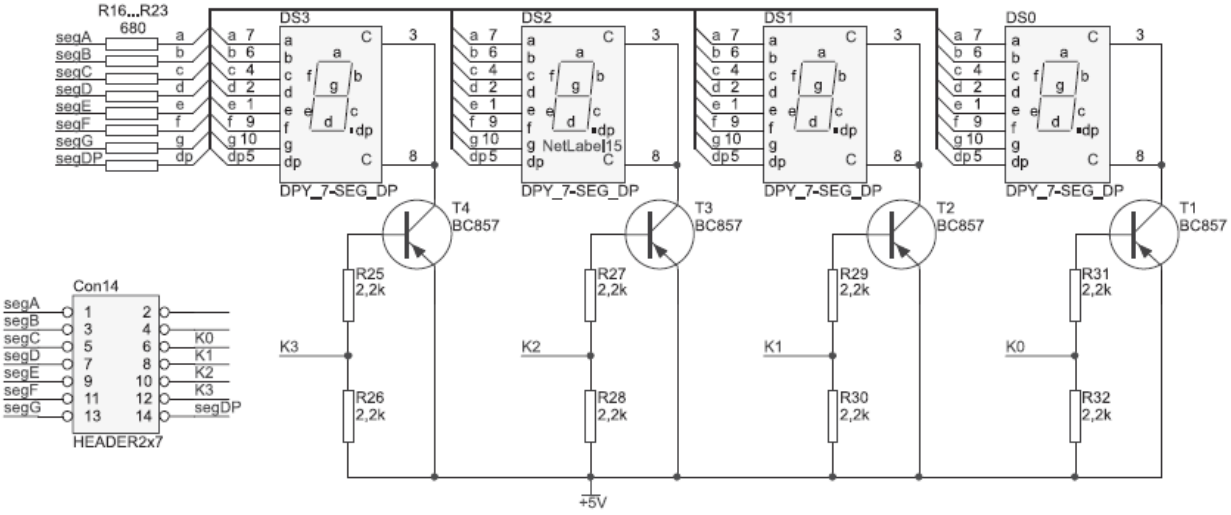
2. Wyświetlanie multipleksowane

Wyświetlanie multipleksowane (inaczej sekwencyjne) jest to taki sposób wyświetlania, w którym w dowolnej chwili aktywny jest zawsze tylko jeden wyświetlacz. Wyświetlacze włączane są po kolei na krótki czas. Ponieważ oko ludzkie wykazuje bezwładność i nie potrafi rozdzielić obrazów zmieniających się z

częstotliwością większą, niż 25 Hz, powstaje wrażenie obrazu statycznego, w którym aktywne są wszystkie wyświetlacze.

Zaletą tego sposobu wyświetlania jest znaczące zmniejszenie ilości sygnałów niezbędnych do wysterowania wyświetlaczy, wadą – migotanie, szczególnie widoczne podczas poruszania głową.

W układzie wyświetlania multipleksowanego sygnały sterujące poszczególnymi segmentami wyświetlaczy są połączone ze sobą, dzięki czemu do wysterowania segmentów wszystkich wyświetlaczy potrzeba 7 (lub 8, jeżeli używana jest kropka dziesiętna) sygnałów, po jednym dla każdego segmentu. Dodatkowo potrzebne są sygnały, które odpowiadają za włączanie poszczególnych wyświetlaczy, po jednym na każdy wyświetlacz. Na płytce zestawu uruchomieniowego ZL30ARM potrzeba 12 sygnałów do wysterowania czterocyfrowego wyświetlacza (8 sygnałów segmentów i 4 sygnały włączenia wyświetlaczy). Sposób podłączenia wyświetlaczy na płytce pokazano na rysunku poniżej:



Jak widać, zastosowano tu wyświetlacze ze wspólną anodą, więc zapalenie danego segmentu odbywa się przez podanie zera na linię segmentu, natomiast aktywacja danego wyświetlacza wymaga podania zera na linie sterujące bazami tranzystorów PNP. Ze względu na różnice napięć zasilających mikrokontroler (3,3V) i emitery tranzystorów (5V) do dezaktywacji tranzystora bardziej właściwym wydaje się być utrzymanie wyjścia sterującego z mikrokontrolera w stanie wysokiej impedancji, niż podanie logicznej jedynki. Dlatego wyprowadzenia mikrokontrolera służące do sterowania tranzystorami powinny być skonfigurowane w trybie OD (z otwartym drenem).

Wyświetlanie multipleksowane należy realizować w następujący sposób (przykład 2):

– na GPIOB (piny 0 – 7), który podłączony jest do segmentów wyświetlaczy, należy podać kod aktywujący odpowiednie segmenty;

– wyzerować w porcie GPIOB (piny 8 – 11) odpowiedni bit aktywujący wyświetlacz (PB8, PB9, PB10 lub PB11) pozostałe bity ustawiając na 1;

– odczekać pewien czas i powtórzyć procedurę od początku dla kolejnych wyświetlaczy.

Czas, który należy odczekać wynosi:

$$t = \frac{1}{4 f_R}$$

gdzie f_R jest to częstotliwość odświeżania obrazu (większa niż 25Hz).

W przypadku stosowania wyświetlaczy ze wspólną katodą, do sterowania wygodniej jest użyć tranzystorów NPN i włączać poszczególne wyświetlacze (poprzez tranzystory) wysokim stanem logicznym.

Pytania kontrolne

1. Jakie symbole można wyświetlić za pomocą wyświetlacza siedmiosegmentowego?
2. W jaki sposób są realizowane i do czego służą tablice konwersji?
3. Co to jest wyświetlanie multipleksowane?
4. Ile sygnałów koniecznych jest do wyświetlenia danych na 10 wyświetlaczach siedmiosegmentowych bez użycia wyświetlania multipleksowanego?
5. Ile sygnałów koniecznych jest do wyświetlenia danych na 10 wyświetlaczach siedmiosegmentowych z użyciem wyświetlania multipleksowanego?
6. Jakie są wady i zalety wyświetlania multipleksowanego?
7. W jaki sposób można zrealizować wyświetlanie multipleksowane mając do dyspozycji wyświetlacze LED ze wspólną anodą i wspólną katodą?

Przykłady:

1. Wyświetlenie cyfry na pojedynczym wyświetlaczu

Program wyświetla na wyświetlaczu DS0 cyfrę 5. Wyprowadzenia złącza Con14 podłączono do wyprowadzeń portu GPIOB (Con1x) w następujący sposób: A → PB0, B → PB1,…, G → PB6, . (kropka) → PB7, dodatkowo 0 → GND. Sposób konfiguracji GPIO i RCC nie wymaga komentarza. Kody źródłowe programu – w pliku **cw4p1.c**:

```
#include "stm32f10x.h"

void GPIO_Config(void);
void RCC_Config(void);
uint16_t int27seg(uint8_t);

int main(void)
{
 //konfiguracja systemu
 RCC_Config();
 GPIO_Config();

  GPIO_Write(GPIOB, int27seg(5));

  while (1);

  return 0;
}

// konwersja na kod wyświetlacza 7-segmentowego
uint16_t int27seg(uint8_t cyfra)
{
  uint16_t result;
  switch (cyfra){
        case 0: result=0xC0; break; // 11000000
        case 1: result=0xF9; break; // 11111001
        case 2: result=0xA4; break; // 10100100
        case 3: result=0xB0; break; // 10110000
        case 4: result=0x99; break; // 10011001
        case 5: result=0x92; break; // 10010010
        case 6: result=0x82; break; // 10000010
        case 7: result=0xF8; break; // 11111000
        case 8: result=0x80; break; // 10000000
        case 9: result=0x90; break; // 10010000
        default: result=0xFF; break; // 11111111 - nic do wyswietlenia
        }
  return result;

}
```

2. Wyświetlanie multipleksowane

Program wyświetlający na 4 wyświetlaczach liczbę „1234”. Wyprowadzenia złącza Con14 podłączono do wyprowadzeń portu GPIOB w następujący sposób: segA → PB0, segB → PB1,…, segG → PB6, segDP → PB7, dodatkowo: K0 → PB8,..., K3 →PB11. Sposób konfiguracji RCC nie wymaga komentarza, kod funkcji int27seg() jest taki sam jak w przykładzie 1 (kod źródłowy znajduje się w pliku: **cw4p2.c**).

```
#include "stm32f10x.h"

void GPIO_Config(void);
void RCC_Config(void);
uint16_t int27seg(uint8_t);
void delay5ms(void);

int main(void)
{
  //konfiguracja systemu
  RCC_Config();
  GPIO_Config();
  /*Tu nalezy umiescic ewentualne dalsze funkcje konfigurujace system*/

       while (1) {
              GPIO_Write(GPIOB, int27seg(1)); // zapal segmenty
              GPIO_ResetBits(GPIOB, GPIO_Pin_11); // wlacz wyswietlacz DS3
              GPIO_SetBits(GPIOB, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10); // wylacz pozostale
              delay5ms(); // opóznienie ok. 5ms

              GPIO_Write(GPIOB, int27seg(2)); // zapal segmenty
              GPIO_ResetBits(GPIOB, GPIO_Pin_10); // wlacz wyswietlacz DS2
              GPIO_SetBits(GPIOB, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_11); // wylacz pozostale
              delay5ms(); // opóznienie ok. 5ms

              GPIO_Write(GPIOB, int27seg(3)); // zapal segmenty
              GPIO_ResetBits(GPIOB, GPIO_Pin_9); // wlacz wyswietlacz DS1
              GPIO_SetBits(GPIOB, GPIO_Pin_8 | GPIO_Pin_10 | GPIO_Pin_11); // wylacz pozostale
              delay5ms(); // opóznienie ok. 5ms

              GPIO_Write(GPIOB, int27seg(4)); // zapal segmenty
              GPIO_ResetBits(GPIOB, GPIO_Pin_8); // wlacz wyswietlacz DS0
              GPIO_SetBits(GPIOB, GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11); // wylacz pozostale
              delay5ms(); // opóznienie ok. 5ms
       };
  return 0;
}

void GPIO_Config(void)
{
       //konfigurowanie portow GPIO
       GPIO_InitTypeDef  GPIO_InitStructure;

       // disable JTAG
       GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

       GPIO_InitStructure.GPIO_Pin  =  GPIO_Pin_0  |  GPIO_Pin_1  |  GPIO_Pin_2  |  GPIO_Pin_3  |
GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;;
       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
       GPIO_Init(GPIOB, &GPIO_InitStructure);

       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 ;
       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;      //wyjscie open drain
       GPIO_Init(GPIOB, &GPIO_InitStructure);

}

// opoznienie ok. 5ms
void delay5ms(void)
{
       volatile unsigned long int i;
       for (i=0; i<32767; i++) ;
}
```

Zadania do samodzielnego wykonania:

1. Napisz program wyświetlający co 1 sek. wszystkie cyfry w systemie szesnastkowym na pojedynczym wyświetlaczu.
2. Wykorzystaj przerwanie od timera SysTick do sterowania wyświetlaczami w trybie multipleksowanym.
3. Napisz program odmierzający czas (stoper) działający z dokładnością do 0.1 sek. Zaimplementuj start, zatrzymanie i reset stopera przy pomocy przycisków.
4. Napisz program, który będzie odmierzał czas od wartości ustalonej przez prowadzącego do 0. Na 20 sekund przed końcem cyfry mają zacząć migać, a 10 sekund przed końcem częstotliwość migania powinna się zwiększyć dwukrotnie.