

Sprawozdanie

Systemy wbudowane



Ćwiczenie 4: Wyświetlacze 7-segmentowe LED.

Wykonanie:

Busłowski Tomasz

Suchwałko Tomasz

Skrouba Kamil

Zawadzka Magdalena

(Grupa PS3)

Prowadzący zajęcia: **dr inż. Adam Klimowicz**

SW, semestr VI, 04-03-2017, Wydział Informatyki, Politechnika Białostocka

Zakres Materiału

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

Zadania do 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.

Zadanie 1

Treść:

Napisz program wyświetlający co 1 sek. wszystkie cyfry w systemie szesnastkowym na pojedynczym wyświetlaczu.

Realizacja:

Wyprowadzenia złącza Con14 podłączyliśmy do wyprowadzeń portu GPIOB(Con1x) w następujący sposób: A->PB0, B->PB1...,G->PB6, .(kropka)->PB7, dodatkowo 0->GND. Zadanie polegało na cyklicznym podmienianiu wyświetlanej cyfry szesnastkowej na wyświetlaczu(0,1,2...E,F). Kluczowa była funkcja, która w zależności od otrzymanego parametru, ustalała wartość, jaką należy przesłać ustawiając konfigurację zapalonych i zgaszonych segmentów na wyświetlaczu. Oto i ona:

```
// konwersja na kod wyswietlacza 7-segmentowego
uint16_t intTo7seg(uint8_t cyfra)
{
    uint16_t result;
    switch (cyfra)
    {
        case 0: result=0xC0; break; // 11000000 0
        case 1: result=0xF9; break; // 11111001 1
        case 2: result=0xA4; break; // 10100100 2
        case 3: result=0xB0; break; // 10110000 3
        case 4: result=0x99; break; // 10011011 4
        case 5: result=0x92; break; // 10010010 5
        case 6: result=0x82; break; // 10000010 6
        case 7: result=0xF8; break; // 11111000 7
        case 8: result=0x80; break; // 10000000 8
        case 9: result=0x90; break; // 10010000 9
        case 10: result=0x88; break; // 10001000 a
        case 11: result=0x83; break; // 10000011 b
        case 12: result=0xC6; break; // 11000110 c
        case 13: result=0xA1; break; // 10100001 d
        case 14: result=0x86; break; // 10000110 e
        case 15: result=0x8E; break; // 10001110 f
        default: result=0xFF; break; // 11111111 - nic do wyswietlenia
    }

    return result;
}
```

Rysunek 1

Do ustalania opóźnienia wykorzystaliśmy napisaną przez nas funkcję wykorzystującą przerwania z ostatnich zajęć – Delay(). W mainie jedynie wywołujemy tę metodę z parametrem 1000 – inicjujemy w ten sposób cykliczne wywoływanie się przerwania co 1000ms.

```
void Delay (uint32_t ms) // nasza funkcja opóźnienia wykorzystująca timer SysTick
{
    if (SysTick_Config(SysTick_Frequency / 1000 * ms))
    {
        while(1);
    }
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
}

int main(void)
{
    volatile unsigned long int j;
    //konfiguracja systemu
    RCC_Config();
    GPIO_Config();

    Delay(1000);

    while (1)
    {
    };

    return 0;
}
```

Rysunek 2

Przerwanie wywołuje jedynie jedną funkcję – zdefiniowaną w mainie - displayNextDigit();

```
extern void displayNextDigit();

void SysTick_Handler(void)
{
    displayNextDigit();
}
```

Rysunek 3

Właściwa akcja programu rozgrywa się właśnie w tej funkcji:

```
void displayNextDigit()
{
    GPIO_Write(GPIOB, intTo7seg(counter++));
    if(counter==16) { counter=0; };
}
```

Rysunek 4

Podsumowując, co sekundę wywoływane jest przerwanie. W przerwaniu wywołujemy funkcję displayNextDigit(). Funkcja ta wyświetla na wyświetlaczu cyfry szesnastkowe od 0 do F.

Zadanie 2

Treść:

Wykorzystaj przerwanie od timera SysTick do sterowania wyświetlaczami w trybie multipleksowanym.

Realizacja:

Wyprowadzenia złącza Con14 podłączyliśmy do wyprowadzeń portu GPIOB w następujący sposób: segA -> PB0, segB -> PB1, ..., segG -> PB6, segDP -> PB7, dodatkowo: K0 -> PB8,..., K3 -> PB11.

Podobnie jak w poprzednim zadaniu, wykorzystaliśmy przerwania. W mainie wywołujemy funkcję Delay z parametrem (interruptFrequency) który określa, co ile ms ma się wywoływać przerwanie. W przerwaniu wywoływana jest jedynie zdefiniowana w mainie funkcja step() która jest sercem programu.

```
5. void GPIO_Config(void);
6. void RCC_Config(void);
7. uint16_t intro7seg(uint8_t);           // konwersja na kod wyświetlacza 7-segmentowego
8. void Delay (uint32_t ms);              // powoduje wywoływanie przerwania z częstotliwością określoną w parametrze
9. void step();                           // wykonuje się w każdym przerwaniu
10. void displayDigitOnNextDisplay(uint8_t x); // wyświetla cyfry na jednym wyświetlaczu, pozostałe gasi
11.
12.
13. uint8_t displayNumber = 0;             // numer wyświetlacza który ma być aktualnie zapalony 0..3
14. uint32_t interruptCounter = 0;         // zlicza przerwania 0..changeDigitTime/interruptFrequency
15. uint8_t digitNumber = 0;              // numer wyświetlanej cyfry 0..15
16. uint32_t changeDigitTime = 1000;      // co ile ms ma zmienić się wyświetlana cyfra
17. uint32_t interruptFrequency = 5;       // co ile ms ma się wywołać przerwanie
18.
19.
20.
21. int main(void)
22. {
23.     //konfiguracja systemu
24.     RCC_Config();
25.     GPIO_Config();
26.
27.     Delay(interruptFrequency);
28.
29.     while (1)
30.     {
31.     };
32.
33. }
```

Rysunek 5

```

147. extern void step();
148.
149. void SysTick_Handler(void)
150. {
151.     step();
152. }

```

Rysunek 6

Funkcja step() wykonuje się w każdym przerwaniu. Jej zadaniem jest wyświetlenie cyfry na jednym z wyświetlaczy (displayNumber 0..3) i wygaszenie pozostałych wyświetlaczy. Dodatkowo, jest zwiększany interruptCounter zliczający wywołane przerwania. Jeżeli jego wartość jest równa ilorazowi changeDigitTime (co ile ms ma się zmienić wyświetlana cyfra) i interruptFrequency, to zwiększamy digitNumber (reprezentację wyświetlanej cyfry). Stosując ten wzór zapewniliśmy to, że przy zmianie częstotliwości wywołania przerwania, wyświetlana cyfra będzie zmieniała się co sekundę.

```

39. void step()
40. {
41.     displayDigitOnNextDisplay(displayNumber++);
42.
43.     if(displayNumber == 4) { displayNumber=0; };
44.     if(interruptCounter++ == changeDigitTime/interruptFrequency)
45.     {
46.         interruptCounter = 0;
47.         if(digitNumber++ == 15) { digitNumber=0; };
48.     }
49. }

```

Rysunek 7

Funkcja `displayDigitOnNextDisplay` wyświetla cyfrę na jednym z wyświetlaczy i wygasza pozostałe wyświetlacze. Do konwersji wartości do kodu wyświetlacza zastosowaliśmy funkcję z poprzedniego zadania – `intTo7seg(uint_t cyfra)`.

```
53. void displayDigitOnNextDisplay(uint8_t x)
54. {
55.     uint16_t d1, d2, d3; // trzy wyświetlacze które należy zgasić
56.     uint16_t d0;         // wyświetlacz, który należy zapalić
57.
58.     GPIO_Write(GPIOB, intTo7seg(digitNumber)); // zapal segmenty wyświetlacza
59.
60.     switch(x)
61.     {
62.         case 0: d0 = GPIO_Pin_11;
63.                 d1 = GPIO_Pin_8;
64.                 d2 = GPIO_Pin_9;
65.                 d3 = GPIO_Pin_10;
66.                 break;
67.
68.         case 1: d0 = GPIO_Pin_10;
69.                 d1 = GPIO_Pin_8;
70.                 d2 = GPIO_Pin_9;
71.                 d3 = GPIO_Pin_11;
72.                 break;
73.
74.         case 2: d0 = GPIO_Pin_9;
75.                 d1 = GPIO_Pin_8;
76.                 d2 = GPIO_Pin_10;
77.                 d3 = GPIO_Pin_11;
78.                 break;
79.
80.         case 3: d0 = GPIO_Pin_8;
81.                 d1 = GPIO_Pin_9;
82.                 d2 = GPIO_Pin_10;
83.                 d3 = GPIO_Pin_11;
84.                 break;
85.     }
86.
87.     GPIO_ResetBits(GPIOB, d0); // włącz wyświetlacz DS3
88.     GPIO_SetBits(GPIOB, d1 | d2 | d3); // wyłącz pozostałe
89. }
```

Rysunek 8

Zadanie 3

Treść:

Napisz program odmierzający czas (stoper) działający z dokładnością do 0.1 sek. Zaimplementuj start, zatrzymanie i reset stopera przy pomocy przycisków.

Realizacja:

Wykonanie zadania w głównej mierze polegało na zmodyfikowaniu funkcji napisanych do realizacji poprzedniego zadania (`displayDigitOnNextDisplay(uint8_t x), intTo7seg(uint8_t), stoper()`) - wszystko nadal oczywiście opierało się na przerwaniach, dopisania funkcji realizującej logikę stopera na 4 wyświetlaczach 7-segmentowych i podłączeniu dwóch przycisków do obsługi stopera (1 - start/pause, 2 - stop).

Połączyliśmy porty czterech wyświetlaczy 7-segmentowych tak jak w poprzednim zadaniu, a dodatkowo porty dwóch pierwszych przycisków SW0 i SW1 do odpowiednich portów GPIOA.

Do realizacji stopera potrzebowaliśmy czterech zmiennych po jednej na każdy wyświetlacz:

c – części dziesiętne sekund (100 milisekund) – wyświetlacz DS0

s1 – jedności sekund – wyświetlacz DS1

s2 – dziesiątki sekund – wyświetlacz DS2

m – minuty – wyświetlacz DS3



Rysunek 9

Funkcja main wygląda tak samo, a do zmiennych globalnych dodaliśmy potrzebne dla nas zmienne do obsługi stopera i przycisków:

```

1  #include "stm32f10x.h"
2  #include "stdbool.h"
3  #define SysTick_Frequency 9000000
4
5  void GPIO_Config(void);
6  void RCC_Config(void);
7  uint16_t intTo7seg(uint8_t, bool dot); // konwersja na kod wyświetlacza 7-segmentowego
8  void Delay (uint32_t ms); // powoduje wywoływanie przerwania z częstotliwością określoną w parametrze
9  void step(void); // wykonuje się w każdym przerwaniu
10 void displayDigitOnNextDisplay(uint8_t x); // wyświetla cyfrę na jednym wyświetlaczu, pozostałe gasi
11 void Stoper(void); // logika stopera
12 uint8_t displayNumber = 0; // numer wyświetlacza który ma być aktualnie zapalony 0..3
13 uint32_t interruptCounter = 0; // zlicza przerwania 0..changeDigitTime/interruptTime
14 uint8_t digitNumber = 0; // numer wyświetlanej cyfry 0..15
15 uint32_t changeDigitTime = 99; // co ile ms ma zmienić się wyświetlana cyfra
16 uint32_t interruptfrequency = 1; // co ile ms ma się wywołać przerwanie
17
18 bool isStoper = false; //czy stoper jest aktywny
19 uint8_t m=0; //minuty
20 uint8_t s1=0; //dziesiątki sekund
21 uint8_t s2=0; //jedności sekund
22 uint8_t c=0; //milisekundy
23 uint8_t button_state=0xFF, temp=0, port_data ; //do obsługi przycisków
24
25 int main(void)
26 {
27     //konfiguracja systemu
28     RCC_Config();
29     GPIO_Config();
30
31     Delay(interruptfrequency);
32
33     while (1)
34     {
35     };
36 }

```

Rysunek 10

Zmodyfikowana funkcja step() dodatkowo obsługuje wcisnięcia dwóch przycisków i kontroluje logikę stopera:

```

40 void step()
41 {
42     // standardowy sposób na odczytanie wcisnięcia przycisków
43     port_data = GPIO_ReadInputData(GPIOA); // czytaj port GPIOA
44     temp = port_data ^ button_state; // czy stan przycisków się zmienił?
45     temp &= button_state; // czy to była zmiana z 1 na 0?
46     button_state = port_data; // zapamiętaj nowy stan
47     if (temp & 0x01){ // jeśli wcisnięty 1 przycisk, to pauza/kontynuacja zliczania stopera
48         isStoper = !isStoper;
49     }
50     if (temp & 0x02){ // jeśli wcisnięty 2 przycisk, to zatrzymaj stoper i wyzeruj wartości
51         isStoper = false;
52         c=0;
53         s1=0;
54         s2=0;
55         m=0;
56     }
57
58     displayDigitOnNextDisplay(displayNumber++);
59     if(displayNumber == 4) { displayNumber=0; };
60     if(interruptCounter++ == changeDigitTime)
61     {
62         interruptCounter = 0;
63         if(isStoper) // jeśli stoper jest aktywny to dodanie kolejnych 100 milisekund do odmierzonego czasu
64         {
65             Stoper();
66         }
67     }
68 }

```

Rysunek 11

Zmodyfikowana funkcja displayDigitOnNextDisplay dla konkretnego wyświetlacza wyświetla konkretne wartości stopera (minuty, sekundy albo milisekundy)

```
72 void displayDigitOnNextDisplay(uint8_t x)
73 {
74     uint16_t d1, d2, d3; // trzy wyświetlacze które należy zgasic
75     uint16_t d0;         // wyświetlacz, który należy zapalic
76
77     switch(x)
78     {
79         case 0: GPIO_Write(GPIOB, intTo7seg(m, true)); // zapal minuty z kropka
80                 d0 = GPIO_Pin_11;
81                 d1 = GPIO_Pin_8;
82                 d2 = GPIO_Pin_9;
83                 d3 = GPIO_Pin_10;
84                 break;
85         case 1: GPIO_Write(GPIOB, intTo7seg(s2, false)); // zapal 'dziesiątki' sekund bez kropki
86                 d0 = GPIO_Pin_10;
87                 d1 = GPIO_Pin_8;
88                 d2 = GPIO_Pin_9;
89                 d3 = GPIO_Pin_11;
90                 break;
91         case 2: GPIO_Write(GPIOB, intTo7seg(s1, true)); // zapal 'jedności' sekund z kropka
92                 d0 = GPIO_Pin_9;
93                 d1 = GPIO_Pin_8;
94                 d2 = GPIO_Pin_10;
95                 d3 = GPIO_Pin_11;
96                 break;
97         case 3: GPIO_Write(GPIOB, intTo7seg(c, false)); // zapal milisekundy bez kropki
98                 d0 = GPIO_Pin_8;
99                 d1 = GPIO_Pin_9;
100                d2 = GPIO_Pin_10;
101                d3 = GPIO_Pin_11;
102                break;
103     }
104     GPIO_ResetBits(GPIOB, d0); // włącz wyświetlacz DS3
105     GPIO_SetBits(GPIOB, d1 | d2 | d3); // wyłącz pozostałe
106 }
```

Rysunek 12

Zmodyfikowana funkcja intTo7seg(uint8_t cyfra, bool dot) dodatkowo jako drugi parametr przyjmuje wartość bool, który mówi czy ma wyświetlić cyfrę z kropką czy nie:

```
132 uint16_t intTo7seg(uint8_t cyfra, bool dot)
133 {
134     // zmodyfikowana funkcja konwertująca liczby z systemu dziesiętnego na system szesnastkowy
135     // z drugim parametrem typu bool:
136     // false - zwykła konwersja
137     // true  - konwersja z dopisaniem kropki do wyświetlenia
138     uint16_t result;
139     switch (cyfra) {
140         case 0: result=0xC0; break; // 11000000 0
141         case 1: result=0xF9; break; // 11111001 1
142         case 2: result=0xA4; break; // 10100100 2
143         case 3: result=0xB0; break; // 10110000 3
144         case 4: result=0x99; break; // 10011001 4
145         case 5: result=0x92; break; // 10010010 5
146         case 6: result=0x82; break; // 10000010 6
147         case 7: result=0xF8; break; // 11111000 7
148         case 8: result=0x80; break; // 10000000 8
149         case 9: result=0x90; break; // 10010000 9
150         case 10: result=0x88; break; // 10001000 a
151         case 11: result=0x83; break; // 10000011 b
152         case 12: result=0xC6; break; // 11000110 c
153         case 13: result=0xA1; break; // 10100001 d
154         case 14: result=0x86; break; // 10000110 e
155         case 15: result=0x8E; break; // 10001110 f
156         default: result=0xFF; break; // 11111111 - nic do wyświetlenia
157     }
158     if(dot) result -= 128; // jeśli ma wyświetlić z kropka to ustawiamy zero na najstarszym bicie
159
160     return result;
161 }
```

Rysunek 13

Nowo dopisana funkcja Stoper() realizująca logikę stopera:

```
108 void Stoper(void)
109 {
110     //prosta implementacja stopera - funkcja stoper jest wywoływana do 100 milisekund
111     if(++c > 9) // dodawanie co 100 milisekund od 0 do 9
112     {
113         c = 0;
114         if(++s1 > 9) // dodawanie jednosci sekund od 0 do 9
115         {
116             s1 = 0;
117             if(++s2 > 5) // dodawanie dziesiatek sekund od 0 do 5 (0 - 59)
118             {
119                 s2 = 0;
120                 if(++m > 9) // dodawanie minut od 0 do 9, jesli stoper zliczyl 9 minut
121                     // 59 sekund 900 milisekund (9:59:9) to zacznie liczyć od zera (0:00:0)
122                     {
123                         c=0;
124                         s1=0;
125                         s2=0;
126                         m=0;
127                     }
128             }
129         }
130     }
131 }
```

Rysunek 14

Kody źródłowe:

Zadanie 1

main.c

```
1. #include "stm32f10x.h"
2. void GPIO_Config(void);
3. void RCC_Config(void);
4. #define SysTick_Frequency 9000000 // 9MHz
5.
6.
7. uint16_t intTo7seg(uint8_t cyfra);
8. void displayNextDigit();
9.
10. uint8_t counter = 0;
11.
12.
13. void Delay (uint32_t ms) // nasza funkcja opóznienia wykorzystująca timer SysTick
14. {
15.     if (SysTick_Config(SysTick_Frequency / 1000 * ms))
16.     {
17.         while(1);
18.     }
19.     SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
20. }
21.
22.
23.
24. int main(void)
25. {
26.     volatile unsigned long int j;
```

SW, semestr VI, 04-03-2017, Wydział Informatyki, Politechnika Białostocka

```

27. //konfiguracja systemu
28. RCC_Config();
29. GPIO_Config();
30.
31. Delay(1000);
32.
33. while (1)
34. {
35. };
36.
37.
38. return 0;
39. }
40.
41. void displayNextDigit()
42. {
43.     GPIO_Write(GPIOB, intTo7seg(counter++));
44.     if(counter==16) { counter=0; };
45. }
46.
47. // konwersja na kod wyswietlacza 7-segmentowego
48. uint16_t intTo7seg(uint8_t cyfra)
49. {
50.     uint16_t result;
51.     switch (cyfra)
52.     {
53.         case 0: result=0xC0; break; // 11000000 0
54.         case 1: result=0xF9; break; // 11111001 1
55.         case 2: result=0xA4; break; // 10100100 2
56.         case 3: result=0xB0; break; // 10110000 3
57.         case 4: result=0x99; break; // 10011011 4
58.         case 5: result=0x92; break; // 10010010 5
59.         case 6: result=0x82; break; // 10000010 6
60.         case 7: result=0xF8; break; // 11111000 7
61.         case 8: result=0x80; break; // 10000000 8
62.         case 9: result=0x90; break; // 10010000 9
63.         case 10: result=0x88; break; // 10001000 a
64.         case 11: result=0x83; break; // 10000011 b
65.         case 12: result=0xC6; break; // 11000110 c
66.         case 13: result=0xA1; break; // 10100001 d
67.         case 14: result=0x86; break; // 10000110 e
68.         case 15: result=0x8E; break; // 10001110 f
69.         default: result=0xFF; break; // 11111111 - nic do wyswietlenia
70.     }
71.
72.     return result;
73. }
74.
75.
76.
77.
78. void RCC_Config(void)
79. //konfigurowanie sygnalow taktujacych
80. {
81.     ErrorStatus HSEStartUpStatus; //zmienna opisujaca rezultat
      uruchomienia HSE
82.
83.     RCC_DeInit(); //Reset ustawien RCC
84.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
85.     RCC_HSEConfig(RCC_HSE_ON); //Wlaczenie HSE
86.     HSEStartUpStatus = RCC_WaitForHSEStartUp(); //Odczekaj az HSE bedzie gotowy
87.     if(HSEStartUpStatus == SUCCESS)
88.     {
89.         FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//
90.         FLASH_SetLatency(FLASH_Latency_2); //ustaw zwloke dla pamieci Flash;
      zaleznie od taktowania rdzenia
91.         //0:<24MHz; 1:24~48MHz; 2:>48MHz

```

```

92.   RCC_HCLKConfig(RCC_SYSCLK_Div1);           //ustaw HCLK=SYSCLK
93.   RCC_PCLK2Config(RCC_HCLK_Div1);           //ustaw PCLK2=HCLK
94.   RCC_PCLK1Config(RCC_HCLK_Div2);           //ustaw PCLK1=HCLK/2
95.   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); //ustaw PLLCLK = HSE*9 czyli 8MHz *
    9 = 72 MHz
96.   RCC_PLLCmd(ENABLE);                       //włącz PLL
97.   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET); //odczekaj na poprawne uruchomienie
    PLL
98.   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); //ustaw PLL jako zrodlo sygnalu
    zegarowego
99.   while(RCC_GetSYSCLKSource() != 0x08);      //odczekaj az PLL bedzie sygnałem
    zegarowym systemu
100.
101.   /*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych
    w programie peryferiow*/
102.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB, ENABLE); //wl
    acz taktowanie portu GPIO A
103.
104.   }
105.   else {}
106.   }
107.
108.
109.
110.   void GPIO_Config(void)
111.   {
112.       //konfigurowanie portow GPIO
113.       GPIO_InitTypeDef GPIO_InitStructure;
114.       GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
115.
116.       /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO
    potrzebnych w programie*/
117.       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;
118.       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
119.       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;           //wyjscie push-pull
120.       GPIO_Init(GPIOB, &GPIO_InitStructure);
121.
122.       GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
123.       GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
124.       GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;      //wejście bez
    podciagania
125.       GPIO_Init(GPIOA, &GPIO_InitStructure);
126.   }

```

Zmiany w pliku stm32f10x_it.c

```

135.
136. extern void displayNextDigit();
137.
138.
139. void SysTick_Handler(void)
140. {
141.     displayNextDigit();
142. }

```

Zadanie 2

main.c

```
1. #include "stm32f10x.h"
2. #define SysTick_Frequency 9000000
3.
4.
5. void GPIO_Config(void);
6. void RCC_Config(void);
7. uint16_t intTo7seg(uint8_t);           // konwersja na kod wyświetlacza 7-segmentowego
8. void Delay (uint32_t ms);             // powoduje wywoływanie przerwania z
    częstotliwością określoną w parametrze
9. void step();                          // wykonuje się w każdym przerwaniu
10. void displayDigitOnNextDisplay(uint8_t x); // wyświetla cyfry na jednym wyświetlaczu,
    pozostałe gasi
11.
12.
13. uint8_t displayNumber = 0;            // numer wyświetlacza który ma być aktualnie zapalony
    0..3
14. uint32_t interruptCounter = 0;        // zlicza przerwy 0..changeDigitTime/interruptTime
15. uint8_t digitNumber = 0;             // numer wyświetlanej cyfry 0..15
16. uint32_t changeDigitTime = 1000;     // co ile ms ma zmienić się wyświetlana cyfra
17. uint32_t interruptfrequency = 5;     // co ile ms ma się wywołać przerwanie
18.
19.
20.
21. int main(void)
22. {
23.     //konfiguracja systemu
24.     RCC_Config();
25.     GPIO_Config();
26.
27.     Delay(interruptfrequency);
28.
29.     while (1)
30.     {
31.     };
32.
33. }
34.
35.
36.
37.
38.
39. void step()
40. {
41.     displayDigitOnNextDisplay(displayNumber++);
42.
43.     if(displayNumber == 4) { displayNumber=0; };
44.     if(interruptCounter++ == changeDigitTime/interruptfrequency)
45.     {
46.         interruptCounter = 0;
47.         if(digitNumber++ == 15) { digitNumber=0; };
48.     }
49. }
50.
51.
52.
53. void displayDigitOnNextDisplay(uint8_t x)
54. {
55.     uint16_t d1, d2, d3; // trzy wyświetlacze które należy zgasić
```



```

56.     uint16_t d0;           // wyswietlacz, ktory nalezy zapalic
57.
58.     GPIO_Write(GPIOB, intTo7seg(digitNumber)); // zapal segmenty wyswietlacza
59.
60.     switch(x)
61.     {
62.         case 0: d0 = GPIO_Pin_11;
63.                 d1 = GPIO_Pin_8;
64.                 d2 = GPIO_Pin_9;
65.                 d3 = GPIO_Pin_10;
66.                 break;
67.
68.         case 1: d0 = GPIO_Pin_10;
69.                 d1 = GPIO_Pin_8;
70.                 d2 = GPIO_Pin_9;
71.                 d3 = GPIO_Pin_11;
72.                 break;
73.
74.         case 2: d0 = GPIO_Pin_9;
75.                 d1 = GPIO_Pin_8;
76.                 d2 = GPIO_Pin_10;
77.                 d3 = GPIO_Pin_11;
78.                 break;
79.
80.         case 3: d0 = GPIO_Pin_8;
81.                 d1 = GPIO_Pin_9;
82.                 d2 = GPIO_Pin_10;
83.                 d3 = GPIO_Pin_11;
84.                 break;
85.     }
86.
87.     GPIO_ResetBits(GPIOB, d0); // włącz wyswietlacz DS3
88.     GPIO_SetBits(GPIOB, d1 | d2 | d3); // wyłącz pozostałe
89. }
90.
91.
92. // konwersja na kod wyswietlacza 7-segmentowego
93. uint16_t intTo7seg(uint8_t cyfra)
94. {
95.     uint16_t result;
96.     switch (cyfra)
97.     {
98.         case 0: result=0xC0; break; // 11000000 0
99.         case 1: result=0xF9; break; // 11111001 1
100.         case 2: result=0xA4; break; // 10100100 2
101.         case 3: result=0xB0; break; // 10110000 3
102.         case 4: result=0x99; break; // 10011011 4
103.         case 5: result=0x92; break; // 10010010 5
104.         case 6: result=0x82; break; // 10000010 6
105.         case 7: result=0xF8; break; // 11111000 7
106.         case 8: result=0x80; break; // 10000000 8
107.         case 9: result=0x90; break; // 10010000 9
108.         case 10: result=0x88; break; // 10001000 a
109.         case 11: result=0x83; break; // 10000011 b
110.         case 12: result=0xC6; break; // 11000110 c
111.         case 13: result=0xA1; break; // 10100001 d
112.         case 14: result=0x86; break; // 10000110 e
113.         case 15: result=0x8E; break; // 10001110 f
114.         default: result=0xFF; break; // 11111111 - nic do wyswietlenia
115.     }
116.
117.     return result;
118. }
119.
120.
121.
122. void Delay (uint32_t ms) // nasza funkcja opóźnienia wykorzystująca timer SysTick

```



```

123.     {
124.         if (SysTick_Config(SysTick_Frequency / 1000 * ms))
125.         {
126.             while(1);
127.         }
128.         SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
129.     }
130.
131.
132.
133.     void RCC_Config(void)
134.     //konfigurowanie sygnalow taktujacych
135.     {
136.         ErrorStatus HSEStartUpStatus; //zmienna opisujaca
            rezultat uruchomienia HSE
137.
138.         RCC_DeInit(); //Reset ustawien RCC
139.         RCC_HSEConfig(RCC_HSE_ON); //Wlaczenie HSE
140.         HSEStartUpStatus = RCC_WaitForHSEStartUp(); //Odczekaj az HSE bedzie
            gotowy
141.         if(HSEStartUpStatus == SUCCESS)
142.         {
143.             FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable); //
144.             FLASH_SetLatency(FLASH_Latency_2); //ustaw zwloke dla pamieci
            Flash; zaleznie od taktowania rdzenia
145.             //0:<24MHz; 1:24~48MHz;
            2:>48MHz
146.             RCC_HCLKConfig(RCC_SYSCLK_Div1); //ustaw HCLK=SYSCLK
147.             RCC_PCLK2Config(RCC_HCLK_Div1); //ustaw PCLK2=HCLK
148.             RCC_PCLK1Config(RCC_HCLK_Div2); //ustaw PCLK1=HCLK/2
149.             RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); //ustaw PLLCLK = HSE*9
            czyli 8MHz * 9 = 72 MHz
150.             RCC_PLLCmd(ENABLE); //włącz PLL
151.             while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET); //odczekaj na poprawne
            uruchomienie PLL
152.             RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); //ustaw PLL jako zrodlo
            sygnalu zegarowego
153.             while(RCC_GetSYSCLKSource() != 0x08); //odczekaj az PLL bedzie
            sygnałem zegarowym systemu
154.
155.             /*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w
            programie peryferiow*/
156.             RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Per
            iph_AFIO, ENABLE); //włącz taktowanie portu GPIO A
157.
158.         } else {
159.         }
160.     }
161.
162.
163.
164.     void GPIO_Config(void)
165.     {
166.         //konfigurowanie portow GPIO
167.         GPIO_InitTypeDef GPIO_InitStructure;
168.
169.         // disable JTAG
170.         GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
171.
172.         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;;
173.         GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
174.         GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //wyjscie push-pull
175.         GPIO_Init(GPIOB, &GPIO_InitStructure);
176.
177.         GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_1
            1 ;

```

```

178.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
179.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;           //wyjscie open drain
180.     GPIO_Init(GPIOB, &GPIO_InitStructure);
181.
182. }

```

Zmiany w pliku stm32f10x_it.c

```

146.
147. extern void step();
148.
149. void SysTick_Handler(void)
150. {
151.     step();
152. }

```

Zadanie 3

main.c

```

#include "stm32f10x.h"

#include "stdbool.h"

#define SysTick_Frequency 9000000

void GPIO_Config(void);
void RCC_Config(void);

uint16_t intTo7seg(uint8_t, bool dot);           // konwersja na kod
wyswietlacza 7-segmentowego

void Delay (uint32_t ms);                        // powoduje wywoływanie
przerwania z czestotliwoscia okreslona w parametrze

void step(void);                                // wykonuje sie w kazdym
przerwaniu

void displayDigitOnNextDisplay(uint8_t x); // wyswietla cyfry na jednym
wyswietlaczu, pozostale gasi

void Stoper(void);

// logika stopera

uint8_t displayNumber = 0;                      // numer wyswietlacza ktory ma byc
aktualnie zapalony 0..3

```

```

uint32_t interruptCounter = 0;           // zlicza przerwania
0..changeDigitTime/interruptTime

uint8_t  digitNumber = 0;               // numer wyswietlanej cyfry 0..15
uint32_t changeDigitTime = 99;         // co ile ms ma zmienic sie
wyswietlana cyfra

uint32_t interruptfrequency = 1;       // co ile ms ma sie wywolac
przerwanie

bool isStoper = false;                  //czy
stoper jest aktywny

uint8_t m=0; //minuty
uint8_t s1=0; //dziesiatki sekund
uint8_t s2=0; //jednosci sekund
uint8_t c=0; //milisekundy
uint8_t button_state=0xFF, temp=0, port_data ; //do obsługi przycisków

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

    Delay(interruptfrequency);

    while (1)
    {
        };
    }

void step()
{
    // standardowy sposób na odczytanie wciśnięcia przycisków

```

```

GPIOA          port_data = GPIO_ReadInputData(GPIOA); // czytaj port

                temp = port_data ^ button_state;          // czy
stan przycisków sie zmienil?

                temp &= button_state;
                // czy to byla zmiana z 1 na 0?

                button_state = port_data;
                // zapamietaj nowy stan

                if (temp & 0x01){                          // jesli wcisniety 1
przycisk, to pauza/kontynuacja zliczania stopera
                    isStoper = !isStoper;

                }

                if (temp & 0x02){                          // jesli wcisniety 2
przycisk, to zatrzymaj stoper i wyzeruj wartosci
                    isStoper = false;
                    c=0;
                    s1=0;
                    s2=0;
                    m=0;

                }

                displayDigitOnNextDisplay(displayNumber++);
                if(displayNumber == 4) { displayNumber=0; };
                if(interruptCounter++ == changeDigitTime)
                {

                    interruptCounter = 0;

                    if(isStoper)                          // jesli
stoper jest aktywny to dodanie kolejnych 100 milisekund do odmierzonego
czasu

                        {

                            Stoper();

                        }

                }

            }

```

```

void displayDigitOnNextDisplay(uint8_t x)
{
    uint16_t d1, d2, d3; // trzy wyświetlacze które należy
    zgasic

    uint16_t d0;         // wyświetlacz, który należy zapalic

    switch(x)
    {
        case 0:    GPIO_Write(GPIOB, intTo7seg(m, true)); //
        zapal minuty z kropka

                        d0 = GPIO_Pin_11;
                        d1 = GPIO_Pin_8;
                        d2 = GPIO_Pin_9;
                        d3 = GPIO_Pin_10;
                        break;

        case 1: GPIO_Write(GPIOB, intTo7seg(s2, false)); // zapal
        'dziesiątki' sekund bez kropki

                        d0 = GPIO_Pin_10;
                        d1 = GPIO_Pin_8;
                        d2 = GPIO_Pin_9;
                        d3 = GPIO_Pin_11;
                        break;

        case 2: GPIO_Write(GPIOB, intTo7seg(s1, true)); // zapal
        'jedności' sekund z kropka

                        d0 = GPIO_Pin_9;
                        d1 = GPIO_Pin_8;
                        d2 = GPIO_Pin_10;
                        d3 = GPIO_Pin_11;
                        break;

        case 3: GPIO_Write(GPIOB, intTo7seg(c, false)); // zapal
        milisekundy bez kropki

                        d0 = GPIO_Pin_8;

```

```

        d1 = GPIO_Pin_9;
        d2 = GPIO_Pin_10;
        d3 = GPIO_Pin_11;
        break;
    }
    GPIO_ResetBits(GPIOB, d0); // wlacz wyswietlacz DS3
    GPIO_SetBits(GPIOB, d1 | d2 | d3); // wylacz pozostale
}

void Stoper(void)
{
    //prosta implementacja stopera - funkcja stoper jest
    //wywoływana do 100 milisekund
    if(++c > 9) // dodawanie co 100 milisekund od 0 do 9
    {
        c = 0;
        if(++s1 > 9) // dodawanie jednosci sekund od 0 do 9
        {
            s1 = 0;
            if(++s2 > 5) // dodawanie dziesiatek sekund od 0 do
5 (0 - 59)
            {
                s2 = 0;
                if(++m > 9) // dodawanie minut od 0 do 9,
jesli stoper zliczyl 9 minut
                // 59
                // 900 milisekund (9:59:9) to zacznie liczyc od zera (0:00:0)
                {
                    c=0;
                    s1=0;
                    s2=0;
                    m=0;
                }
            }
        }
    }
}

```

```

    }
}

uint16_t intTo7seg(uint8_t cyfra, bool dot)
{
    // zmodyfikowana funkcja konwertujaca liczby z systemu
    dziesietnego na system szesnastkowy
    // z drugim parametrem typu bool:
    // false - zwykla konwersja
    // true  - konwersja z dopisaniem kropi do wyswietlenia
    uint16_t result;
    switch (cyfra){
        case 0: result=0xC0; break; // 11000000 0
        case 1: result=0xF9; break; // 11111001 1
        case 2: result=0xA4; break; // 10100100 2
        case 3: result=0xB0; break; // 10110000 3
        case 4: result=0x99; break; // 10011001 4
        case 5: result=0x92; break; // 10010010 5
        case 6: result=0x82; break; // 10000010 6
        case 7: result=0xF8; break; // 11111000 7
        case 8: result=0x80; break; // 10000000 8
        case 9: result=0x90; break; // 10010000 9
        case 10: result=0x88; break; // 10001000 a
        case 11: result=0x83; break; // 10000011 b
        case 12: result=0xC6; break; // 11000110 c
        case 13: result=0xA1; break; // 10100001 d
        case 14: result=0x86; break; // 10000110 e
        case 15: result=0x8E; break; // 10001110 f
        default: result=0xFF; break; // 11111111 -
    }
    nic do wyswietlenia
}

```

```

        if(dot) result -= 128;          // jesli ma wyswietlic z
kropka to ustawiamy zero na najstarszym bicie

        return result;
}

void Delay (uint32_t ms) // nasza funkcja opóznienia wykorzystujaca timer
SysTick
{
    if (SysTick_Config(SysTick_Frequency / 1000 * ms))
    {
        while(1);
    }
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
}

void RCC_Config(void)
{
    //konfigurowanie sygnalow taktujacych
    ErrorStatus HSEStartUpStatus;
    //zmienna opisujaca rezultat uruchomienia HSE
    //
    RCC_DeInit();
    //Reset ustawien RCC
    RCC_HSEConfig(RCC_HSE_ON);
    //Wlaczenie HSE
    HSEStartUpStatus = RCC_WaitForHSEStartUp();
    //Odczekaj az HSE bedzie gotowy
    if(HSEStartUpStatus == SUCCESS)
    {
        FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//

```



```

FLASH_SetLatency(FLASH_Latency_2);
//ustaw zwloke dla pamieci Flash; zaleznie od taktowania rdzenia

//0:<24MHz; 1:24~48MHz; 2:>48MHz
RCC_HCLKConfig(RCC_SYSCLK_Div1);
//ustaw HCLK=SYSCLK

RCC_PCLK2Config(RCC_HCLK_Div1);
//ustaw PCLK2=HCLK

RCC_PCLK1Config(RCC_HCLK_Div2);
//ustaw PCLK1=HCLK/2

RCC_PLLConfig(RCC_PLLSource_HSE_Div1,
RCC_PLLMul_9); //ustaw PLLCLK = HSE*9 czyli 8MHz * 9 = 72 MHz
RCC_PLLCmd(ENABLE);
//wlacz PLL

while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) ==
RESET); //odczekaj na poprawne uruchomienie PLL

RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
//ustaw PLL jako zrodlo sygnalu zegarowego

while(RCC_GetSYSCLKSource() != 0x08);
//odczekaj az PLL bedzie sygnałem zegarowym systemu

/*Tu nalezy umiescic kod zwiazany z
konfiguracja sygnalow zegarowych potrzebnych w programie peryferiow*/

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE); //wlacz taktowanie
portu GPIO A

    } else {
    }

}

```

```

void GPIO_Config(void)

```

```

{

    //konfigurowanie portow GPIO
    GPIO_InitTypeDef  GPIO_InitStructure;

    // disable JTAG

    GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

```

SW, semestr VI, 04-03-2017, Wydział Informatyki, Politechnika Białostocka

```

        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;
//wyjście 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;
//wyjście open drain

        GPIO_Init(GPIOB, &GPIO_InitStructure);

}

```

Plik stm32f10x_it.c jak w zadaniu 2.