Sprawozdanie

Systemy wbudowane

Ćwiczenie 2:

Konfiguracja sygnałów zegarowych.
Porty wejścia/wyjścia ogólnego przeznaczenia

Wykonanie:

Busłowski Tomasz

Suchwałko Tomasz

(Grupa PS3)

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

Zakres Materiału

1. Sygnały zegarowe i ich konfiguracja.
2. Budowa portów wejścia/wyjścia ogólnego przeznaczenia w mikrokontrolerze STM32F103.
3. Programowanie portów i sterowanie kierunkiem portu.
4. Techniki programowania pętli opóźniających (instrukcje nop, goto, decfsz)

Zadania do wykonania

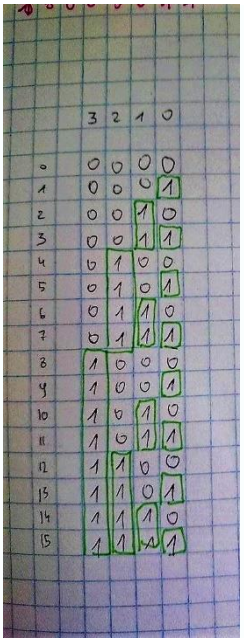
1. Napisz program, który miga 4 diodami w taki sposób, że każda następna miga 2 razy szybciej niż poprzednia.
2. Napisz program, który zapala i gasi diodę przyporządkowaną danemu przyciskowi (SW0 – LED0, SW1 - LED1, itd.)
3. Napisz program, który wykorzystuje joystick do sterowania przesuwającą się cyklicznie zapaloną diodą. (W – przesuw w lewo, E – w prawo, N – zwiększ szybkość przesuwania, S – zmniejsz szybkość przesuwania). Dodatkowo przyciskami SW0 i SW1 należy zwiększać i zmniejszać liczbę przesuwających się zapalonych diod.

Zadanie 1

Treść: Napisz program, który miga 4 diodami w taki sposób, że każda następna miga 2 razy szybciej niż poprzednia.

Realizacja:
Połączyliśmy kabelkami porty 4 diód(0-3) z odpowiednimi portami GPIOB(PB0-PB3). Aby zrealizować zadanie musieliśmy zastanowić się jak zakodować sekwencję zapalania diód. Musieliśmy dla każdej z nich oddzielnie odmierzać ustaloną ilość czasu po którym nastąpi zmiana stanu diody.

Rozpisaliśmy na kartce stany diód w interwałach(Rysunek 1). Jednostką był czas migania najszybszej diody. Pierwotnie zaimplementowaliśmy program w taki sposób, że zmienialiśmy stan diód wywołując napisaną przez nas funkcję (void change(int nr)) zmieniającą stan danej diody: diody nr0 => co interwał, diody nr1 => co 2 interwały, diody nr3 => co 4 interwały i diody nr 5 => co 8 interwałów.(Rysunek 2.)



Rysunek 1

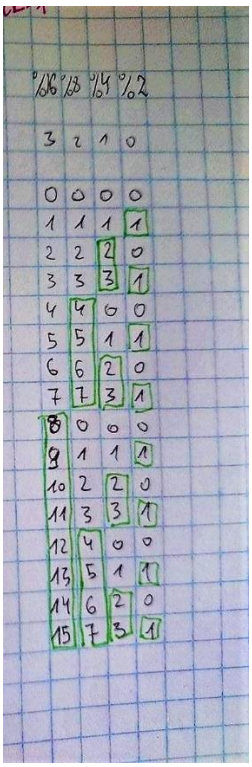
```
while (1)
{
  for(i = 0; i < 0x80000ul; i++);
    change(0);
  for(i = 0; i < 0x80000ul; i++);
    change(0); change(1);
  for(i = 0; i < 0x80000ul; i++);
    change(0);
  for(i = 0; i < 0x80000ul; i++);
    change(0);change(1);change(2);

  for(i = 0; i < 0x80000ul; i++);
    change(0);
  for(i = 0; i < 0x80000ul; i++);
    change(0); change(1);
  for(i = 0; i < 0x80000ul; i++);
    change(0);
  for(i = 0; i < 0x80000ul; i++);
    change(0);change(1);change(2);change(3);

};
```

Rysunek 2

Jednak nie byliśmy usatysfakcjonowani z takiego zapisu. Nurtowało nas poczucie, że należałoby to zaimplementować sprytniej. Zmieniliśmy podejście: stworzyliśmy tablicę 4 intów która symulowała tabelkę z Rysunku1. Po każdym interwale, wykorzystując dzielenie modulo 2,4,8,16 aktualizowaliśmy stan tablicy i przesyłaliśmy ją do funkcji setAllLigts(), która zapalała odpowiednie diody.



Rysunek 4

```
while (1)
{
  for(i = 0; i < 0x80000ul; i++);

  for(i=1; i<=4; i++)
  {
    x = (int)pow(2.0,(double)i);              // 2,  4,  8,  16
    tab[i-1] = ((counter%x)<(x/2)) ? 0 : 1;   // <1, <2, <4, <8
  }
  setAllLigts(tab);

  if((++counter) == 16) { counter = 0; }
};
```

Rysunek 5

```
void setAllLigts(int tab[4])
{
  GPIO_WriteBit(GPIOB, GPIO_Pin_0, tab[0]==1 ? Bit_SET : Bit_RESET);
  GPIO_WriteBit(GPIOB, GPIO_Pin_1, tab[1]==1 ? Bit_SET : Bit_RESET);
  GPIO_WriteBit(GPIOB, GPIO_Pin_2, tab[2]==1 ? Bit_SET : Bit_RESET);
  GPIO_WriteBit(GPIOB, GPIO_Pin_3, tab[3]==1 ? Bit_SET : Bit_RESET);
}
```

Rysunek 6

Zadanie 2

Treść: Napisz program, który zapala i gasi diodę przyporządkowaną danemu przyciskowi (SW0 – LED0, SW1 - LED1, itd.)

Realizacja:

```
while (1)
{
  port_data = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
  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) // czy to przycisk 1?
  {
    for(i = 0; i < p; i++);
    GPIO_WriteBit(GPIOB, GPIO_Pin_0, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_0)));
  }

  if (temp & 0x02) // czy to przycisk 2?
  {
    for(i = 0; i < p; i++);
    GPIO_WriteBit(GPIOB, GPIO_Pin_1, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_1)));
  }

  if (temp & 0x04) // czy to przycisk 3?
  {
    for(i = 0; i < p; i++);
    GPIO_WriteBit(GPIOB, GPIO_Pin_2, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_2)));
  }

  if (temp & 0x08) // czy to przycisk 4?
  {
    for(i = 0; i < p; i++);
    GPIO_WriteBit(GPIOB, GPIO_Pin_3, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_3)));
  }
};
```

Rysunek 7

Zadanie 3

Treść: Napisz program, który wykorzystuje joystick do sterowania przesuwającą się cyklicznie zapaloną diodą. (W – przesuw w lewo, E – w prawo, N – zwiększ szybkość przesuwania, S – zmniejsz szybkość przesuwania). Dodatkowo przyciskami SW0 i SW1 należy zwiększać i zmniejszać liczbę przesuwających się zapalonych diod.

Realizacja:

W naszej implementacji pętla while(Rysunek 8) z maina wygląda bardzo skromnie. Większość zadań odbywa się w wywoływanych z niej funkcji.

```
  while (1)
  {
      port_data = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
      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

      permission=true;
      setTable(tab, direction, dl, &lewyKoniec); // ustal tablice
      setAllLigts(tab); // zapal jedynki
      wait(&p, &port_data, &temp, &button_state, &direction, tab, &lewyKoniec, &dl, &permission);
};
```

Rysunek 8

Funkcja setTable(Rysunek 9) wstawia w tablicy tab[8] 1 w miejsca gdzie diody mają być zapalone i 0 gdzie mają być zgaszone. Zmienna direction zapamiętuje kierunek ruchu (0- w lewo, 1- w prawo). Zmienna lewyKoniec zapamiętuje indeks tablicy gdzie znjaduje się lewy koniec "pociągu diód". Znana nam funkcja z zadania 1 setAllLights(Rysunek 10) zapala diody korzystając z tablicy tab.

```
void setTable(int tab[8], int d, int dl, int* lewyKoniec)
{

  if(d == 0) // jesli w lewo
  {
    tab[((*lewyKoniec)-(dl-1)+8)%8] = 0; // zerujemy prawy koniec ; +8 bo w C nie liczy modulo z ujemnych
    (*lewyKoniec) = ((*lewyKoniec)+1)%8; // ustalamy nowy lewy koniec
    tab[(*lewyKoniec)] = 1;              // wstawiamy 1 z lewej strony
  }
  else // jesli w prawo
  {
    tab[(*lewyKoniec)] = 0;                // zerujemy lewy koniec
    (*lewyKoniec) = ((*lewyKoniec)-1+8)%8; // ustalamy nowy lewy koniec ; +8 bo w C nie liczy modulo z ujemnych
    tab[((*lewyKoniec)-(dl-1)+8)%8] = 1;   // wstawiamy 1 z lewej strony ; +8 bo w C nie liczy modulo z ujemnych
  }

}
```

Rysunek 9

```
void setAllLigts(int tab[8])
{
   GPIO_WriteBit(GPIOB, GPIO_Pin_0, tab[0]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_1, tab[1]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_2, tab[2]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_3, tab[3]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_4, tab[4]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_5, tab[5]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_6, tab[6]==1 ? Bit_SET : Bit_RESET);
   GPIO_WriteBit(GPIOB, GPIO_Pin_7, tab[7]==1 ? Bit_SET : Bit_RESET);
}
```

Rysunek 10

Następnie program kontynuuje swój przebieg w funkcji wait(Rysunek 11), która nasłuchuje czy został wciśnięty któryś z klawiszy.

```
void wait(unsigned long int* p, uint8_t* port_data, uint8_t* temp, uint8_t* button_state,
          int* direction, int tab[], int* lewyKoniec, int* dl, bool* permisison)
{
    int i, j;
    for(i=0; i<(*p); i++)
    {
        (*port_data) = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
        (*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) != 0)
        {
            if((*temp) & 0x20) { (*p)/=1.2; }
            else
            if((*temp) & 0x40) { (*p)*=1.2; }
            else
            if((((*direction)==1) && ((*temp) & 0x10))) { (*direction)=0; }
            else
            if((((*direction)==0) && ((*temp) & 0x80))) { (*direction)=1; }
            else
            if(((*temp) & 0x01) && (*permisison)==true) // SW0, zmniejszamy
            {
            else
            if(((*temp) & 0x02) && (*permisison)==true) // SW1, zwiekszamy
            {
        }
    }
}
```

Rysunek 11

Rozwinięcie warunków wciśnięcia przycisków zmniejszania/zwiększania “pociągu diód”:

```
            if(((*temp) & 0x01) && (*permisison)==true) // SW0, zmniejszamy
            {
                if((*dl)>1)
                {
                    if((*direction) == 0)
                    {
                        tab[((*lewyKoniec)-(*dl)+1+8)%8] = 0;
                        (*dl) -= 1;
                    }
                    else
                    {
                        tab[(*lewyKoniec)] = 0;
                        (*lewyKoniec) = ((*lewyKoniec)-1+8)%8; // +8 bo w C nie liczy modulo z ujemnych
                        (*dl) -= 1;
                    }
                    (*permisison)=false;
                }
            }
            else
            if(((*temp) & 0x02) && (*permisison)==true) // SW1, zwiekszamy
            {
                if((*dl)<7)
                {
                    if((*direction) == 0)
                    {
                        tab[((*lewyKoniec)-(*dl)+1+8)%8] = 1;
                        (*dl) += 1;
                    }
                    else
                    {
                        (*lewyKoniec) = ((*lewyKoniec)+1)%8; // +8 bo w C nie liczy modulo z ujemnych
                        tab[(*lewyKoniec)] = 1;
                        (*dl) += 1;
                    }
                    (*permisison)=false;
                }
```

Rysunek 12

Kody źródłowe plików main.c:

Zadanie 1

```
1.  #include "stm32f10x.h"
2.  #include <math.h>
3.
4.  void GPIO_Config(void);
5.  void RCC_Config(void);
6.
7.  void setAllLigts(int tab[4]);
8.
9.  int main(void)
10. {
11.   volatile unsigned long int i;
12.     int counter = 0;
13.     int x;
14.     int tab[4];
15.
16.   //konfiguracja systemu
17.   RCC_Config();
18.   GPIO_Config();
19.   /*Tu nalezy umiescic ewentualne dalsze funkcje konfigurujace system*/
20.   GPIO_ResetBits(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO
    _Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);
21.
22.
23.   while (1)
24.     {
25.         for(i = 0; i < 0x40000ul; i++);
26.
27.         for(i=1; i<=4; i++)
28.         {
29.             x = (int)pow(2.0,(double)i);              // 2,  4,  8,  16
30.             tab[i-1] = ((counter%x)<(x/2)) ? 0 : 1;   // <1, <2, <4, <8
31.         }
```

```
32.         setAllLigts(tab);
33.
34.         if((++counter) == 16) { counter = 0; }
35.   };
36.
37.
38.   return 0;
39. }
40.
41. void setAllLigts(int tab[4])
42. {
43.     GPIO_WriteBit(GPIOB, GPIO_Pin_0, tab[0]==1 ? Bit_SET : Bit_RESET);
44.     GPIO_WriteBit(GPIOB, GPIO_Pin_1, tab[1]==1 ? Bit_SET : Bit_RESET);
45.     GPIO_WriteBit(GPIOB, GPIO_Pin_2, tab[2]==1 ? Bit_SET : Bit_RESET);
46.     GPIO_WriteBit(GPIOB, GPIO_Pin_3, tab[3]==1 ? Bit_SET : Bit_RESET);
47. }
48.
49.
50. //konfigurowanie sygnalow taktujacych
51. void RCC_Config(void)
52. {
53.   ErrorStatus HSEStartUpStatus;                          //zmienna opisujaca rezultat
    uruchomienia HSE
54.
55.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
56.
57.   RCC_DeInit();                                          //Reset ustawien RCC
58.   RCC_HSEConfig(RCC_HSE_ON);                             //Wlaczenie HSE
59.   HSEStartUpStatus = RCC_WaitForHSEStartUp();            //Odczekaj az HSE bedzie gotowy
60.   if(HSEStartUpStatus == SUCCESS)
61.   {
62.     FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//
63.     FLASH_SetLatency(FLASH_Latency_2);                   //ustaw zwloke dla pamieci Flash;
    zaleznie od taktowania rdzenia
64.                                                          //0:<24MHz; 1:24~48MHz; 2:>48MHz
65.     RCC_HCLKConfig(RCC_SYSCLK_Div1);                     //ustaw HCLK=SYSCLK
66.     RCC_PCLK2Config(RCC_HCLK_Div1);                      //ustaw PCLK2=HCLK
67.     RCC_PCLK1Config(RCC_HCLK_Div2);                      //ustaw PCLK1=HCLK/2
68.     RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); //ustaw PLLCLK = HSE*9 czyli 8MHz *
    9 = 72 MHz
69.     RCC_PLLCmd(ENABLE);                                  //wlacz PLL
70.     while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);  //odczekaj na poprawne uruchomienie
    PLL
71.     RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);           //ustaw PLL jako zrodlo sygnalu
    zegarowego
72.     while(RCC_GetSYSCLKSource() != 0x08);                //odczekaj az PLL bedzie sygnalem
    zegarowym systemu
73.
74.   /*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w
    programie peryferiow*/
75.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);//wlacz taktowanie portu GPIO B
76.
77.   } else
78.     {
79.   }
80. }
81.
82.
83. void GPIO_Config(void)
84. {
85.   //konfigurowanie portow GPIO
86.   GPIO_InitTypeDef  GPIO_InitStructure;
87.
88.     GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
```

```
89.
90.   /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO potrzebnych w
    programie*/
91.   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;
92.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
93.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
94.   GPIO_Init(GPIOB, &GPIO_InitStructure);
95. }
```

Zadanie 2

```
1.  #include "stm32f10x.h"
2.
3.  void GPIO_Config(void);
4.  void RCC_Config(void);
5.
6.  int main(void)
7.  {
8.    volatile unsigned long int i;
9.      volatile unsigned long int p = 0x40000ul;
10.     uint8_t button_state=0xFF, temp=0, port_data ;
11.
12.   //konfiguracja systemu
13.   RCC_Config();
14.   GPIO_Config();
15.
16.   GPIO_ResetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 );
17.     GPIO_ResetBits(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GP
    IO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);
18.
19.     while (1)
20.     {
21.         port_data = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
22.         temp = port_data ^ button_state; // czy stan przycisków sie zmienil?
23.         temp &= button_state; // czy to byla zmiana z 1 na 0?
24.         button_state = port_data; // zapamietaj nowy stan
25.
26.         if (temp & 0x01) // czy to przycisk 1?
27.         {
28.             for(i = 0; i < p; i++);
29.             GPIO_WriteBit(GPIOB, GPIO_Pin_0, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GP
    IO_Pin_0)));
30.         }
31.
32.         if (temp & 0x02) // czy to przycisk 2?
33.         {
```

```
34.             for(i = 0; i < p; i++);
35.             GPIO_WriteBit(GPIOB, GPIO_Pin_1, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GP
    IO_Pin_1)));
36.         }
37.
38.         if (temp & 0x04) // czy to przycisk 3?
39.         {
40.             for(i = 0; i < p; i++);
41.             GPIO_WriteBit(GPIOB, GPIO_Pin_2, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GP
    IO_Pin_2)));
42.         }
43.
44.         if (temp & 0x08) // czy to przycisk 4?
45.         {
46.             for(i = 0; i < p; i++);
47.             GPIO_WriteBit(GPIOB, GPIO_Pin_3, (BitAction)(1 - GPIO_ReadOutputDataBit(GPIOB, GP
    IO_Pin_3)));
48.         }
49.   };
50.   return 0;
51. }
52.
53.
54. void RCC_Config(void)
55. //konfigurowanie sygnalow taktujacych
56. {
57.   ErrorStatus HSEStartUpStatus;                          //zmienna opisujaca rezultat
    uruchomienia HSE
58.
59.   RCC_DeInit();                                          //Reset ustawien RCC
60.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
61.   RCC_HSEConfig(RCC_HSE_ON);                             //Wlaczenie HSE
62.   HSEStartUpStatus = RCC_WaitForHSEStartUp();            //Odczekaj az HSE bedzie gotowy
63.   if(HSEStartUpStatus == SUCCESS)
64.   {
65.     FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//
66.     FLASH_SetLatency(FLASH_Latency_2);                   //ustaw zwloke dla pamieci Flash;
    zaleznie od taktowania rdzenia
67.                                                          //0:<24MHz; 1:24~48MHz; 2:>48MHz
68.     RCC_HCLKConfig(RCC_SYSCLK_Div1);                     //ustaw HCLK=SYSCLK
69.     RCC_PCLK2Config(RCC_HCLK_Div1);                      //ustaw PCLK2=HCLK
70.     RCC_PCLK1Config(RCC_HCLK_Div2);                      //ustaw PCLK1=HCLK/2
71.     RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); //ustaw PLLCLK = HSE*9 czyli 8MHz *
    9 = 72 MHz
72.     RCC_PLLCmd(ENABLE);                                  //wlacz PLL
73.     while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);  //odczekaj na poprawne uruchomienie
    PLL
74.     RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);           //ustaw PLL jako zrodlo sygnalu
    zegarowego
75.     while(RCC_GetSYSCLKSource() != 0x08);                //odczekaj az PLL bedzie sygnalem
    zegarowym systemu
76.
77.   /*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w
    programie peryferiow*/
78.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB, ENABLE);//wlacz
    taktowanie portu GPIO A
79.
80.   } else {
81.   }
82. }
83.
84.
85.
86. void GPIO_Config(void)
```

```
87. {
88.   //konfigurowanie portow GPIO
89.   GPIO_InitTypeDef  GPIO_InitStructure;
90. GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
91.   /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO potrzebnych w
    programie*/
92.   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;
93.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
94.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
95.   GPIO_Init(GPIOB, &GPIO_InitStructure);
96.
97.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
98.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
99.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;      //wejscie bez podciagania
100.          GPIO_Init(GPIOA, &GPIO_InitStructure);
101.
102.        }
```

Zadanie 3

```
1.  #include "stm32f10x.h"
2.
3.  typedef int bool;
4.  #define true 1
5.  #define false 0
6.
7.  void GPIO_Config(void);
8.  void RCC_Config(void);
9.
10. void wait(unsigned long int* p, uint8_t* port_data, uint8_t* temp, uint8_t* button_state, int
    * direction, int tab[], int* lewyKoniec, int* dl, bool* permisison);
11. void setAllLigts(int tab[8]);
12. void setTable(int tab[8], int d, int dl, int* lewyKoniec);
13.
14.
15. int main(void)
16. {
17.   volatile unsigned long int i;
18.     int direction=1;       // 0-w lewo, 1-w prawo
19.     volatile unsigned long int p = 0x40000ul;
20.     uint8_t button_state=0xFF, temp=0, port_data ;
21.     int lewyKoniec=3; // lewy poczatek pociagu
22.     int dl=4;   // dlugosc pociagu
23.     bool permission=true;
24.
25.     int tab [8];
26.     for(i=0; i<8; i++)
27.     {
28.         tab[i] = 0;
29.     }
30.     tab[3]=1;
31.     tab[2]=1;
32.     tab[1]=1;
33.     tab[0]=1;
34.
35.   //konfiguracja systemu
```

```
36.   RCC_Config();
37.   GPIO_Config();
38.   GPIO_ResetBits(GPIOA, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO
    _Pin_5 | GPIO_Pin_6 | GPIO_Pin_7 );
39.   GPIO_ResetBits(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO
    _Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);
40.
41.
42.     while (1)
43.     {
44.         port_data = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
45.         temp = port_data ^ button_state; // czy stan przycisków sie zmienil?
46.         temp &= button_state; // czy to byla zmiana z 1 na 0?
47.         button_state = port_data; // zapamietaj nowy stan
48.
49.         permission=true;
50.         setTable(tab, direction, dl, &lewyKoniec); // ustal tablice
51.         setAllLigts(tab); // zapal jedynki
52.         wait(&p, &port_data, &temp, &button_state, &direction, tab, &lewyKoniec, &dl, &permis
    sion);
53.   };
54.
55.
56.
57.   return 0;
58. }
59.
60.
61. void wait(unsigned long int* p, uint8_t* port_data, uint8_t* temp, uint8_t* button_state, int
    * direction, int tab[], int* lewyKoniec, int* dl, bool* permisison)
62. {
63.     int i, j;
64.     for(i=0; i<(*p); i++)
65.     {
66.         (*port_data) = GPIO_ReadInputData(GPIOA); //czytaj port GPIOA
67.         (*temp) = (*port_data) ^ (*button_state); // czy stan przycisków sie zmienil?
68.         (*temp) &= (*button_state); // czy to byla zmiana z 1 na 0?
69.         (*button_state) = (*port_data); // zapamietaj nowy stan
70.         if((*temp) != 0)
71.         {
72.             if((*temp) & 0x20) { (*p)/=1.2; }
73.             else
74.             if((*temp) & 0x40) { (*p)*=1.2; }
75.             else
76.             if((((*direction)==1) && ((*temp) & 0x10))) { (*direction)=0; }
77.             else
78.             if((((*direction)==0) && ((*temp) & 0x80))) { (*direction)=1; }
79.             else
80.             if(((*temp) & 0x01) && (*permisison)==true) // SW0, zmniejszamy
81.             {
82.                 //for(j = 0; j < (*p); j++);
83.                 if((*dl)>1)
84.                 {
85.                     if((*direction) == 0)
86.                     {
87.                         tab[((*lewyKoniec)-(*dl)+1+8)%8] = 0;
88.                         (*dl) -= 1;
89.                     }
90.                     else
91.                     {
92.                         tab[(*lewyKoniec)] = 0;
93.                         (*lewyKoniec) = ((*lewyKoniec)-1+8)%8; // +8 bo w C nie liczy modulo
    z ujemnych
94.                         (*dl) -= 1;
```

```
95.                     }
96.                     (*permisison)=false;
97.                 }
98.             }
99.             else
100.                     if(((*temp) & 0x02) && (*permisison)==true) // SW1, zwiekszamy
101.                     {
102.                         //+for(j = 0; j < (*p); j++);
103.                         if((*dl)<7)
104.                         {
105.                             if((*direction) == 0)
106.                             {
107.                                 tab[((*lewyKoniec)-(*dl)+1+8)%8] = 1;
108.                                 (*dl) += 1;
109.                             }
110.                             else
111.                             {
112.                                 (*lewyKoniec) = ((*lewyKoniec)+1)%8; // +8 bo w C nie liczy
    modulo z ujemnych
113.                                 tab[(*lewyKoniec)] = 1;
114.                                 (*dl) += 1;
115.                             }
116.
117.                             (*permisison)=false;
118.                         }
119.                     }
120.                 }
121.             }
122.         }
123.
124.
125.         void setAllLigts(int tab[8])
126.         {
127.             GPIO_WriteBit(GPIOB, GPIO_Pin_0, tab[0]==1 ? Bit_SET : Bit_RESET);
128.             GPIO_WriteBit(GPIOB, GPIO_Pin_1, tab[1]==1 ? Bit_SET : Bit_RESET);
129.             GPIO_WriteBit(GPIOB, GPIO_Pin_2, tab[2]==1 ? Bit_SET : Bit_RESET);
130.             GPIO_WriteBit(GPIOB, GPIO_Pin_3, tab[3]==1 ? Bit_SET : Bit_RESET);
131.             GPIO_WriteBit(GPIOB, GPIO_Pin_4, tab[4]==1 ? Bit_SET : Bit_RESET);
132.             GPIO_WriteBit(GPIOB, GPIO_Pin_5, tab[5]==1 ? Bit_SET : Bit_RESET);
133.             GPIO_WriteBit(GPIOB, GPIO_Pin_6, tab[6]==1 ? Bit_SET : Bit_RESET);
134.             GPIO_WriteBit(GPIOB, GPIO_Pin_7, tab[7]==1 ? Bit_SET : Bit_RESET);
135.         }
136.
137.
138.         void setTable(int tab[8], int d, int dl, int* lewyKoniec)
139.         {
140.
141.             if(d == 0) // jesli w lewo
142.             {
143.                 tab[((*lewyKoniec)-(dl-1)+8)%8] = 0; // zerujemy prawy koniec ; +8 bo w C
    nie liczy modulo z ujemnych
144.                 (*lewyKoniec) = ((*lewyKoniec)+1)%8; // ustalamy nowy lewy koniec
145.                 tab[(*lewyKoniec)] = 1;              // wstawiamy 1 z lewej strony
146.             }
147.             else // jesli w prawo
148.             {
149.                 tab[(*lewyKoniec)] = 0;                // zerujemy lewy koniec
150.                 (*lewyKoniec) = ((*lewyKoniec)-1+8)%8; // ustalamy nowy lewy koniec ; +8 bo
    w C nie liczy modulo z ujemnych
151.                 tab[((*lewyKoniec)-(dl-1)+8)%8] = 1;   // wstawiamy 1 z lewej strony ; +8 bo
    w C nie liczy modulo z ujemnych
152.             }
153.
154.         }
```

```
155.
156.
157.
158.
159.
160.
161.         //konfigurowanie sygnalow taktujacych
    ////////////////////////////////////////////////////////////////////////////
162.         void RCC_Config(void)
163.         {
164.           ErrorStatus HSEStartUpStatus;                          //zmienna opisujaca
    rezultat uruchomienia HSE
165.
166.           RCC_DeInit();                                          //Reset ustawien RCC
167.             RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_AFIO, ENABLE);
168.           RCC_HSEConfig(RCC_HSE_ON);                             //Wlaczenie HSE
169.           HSEStartUpStatus = RCC_WaitForHSEStartUp();            //Odczekaj az HSE bedzie
    gotowy
170.           if(HSEStartUpStatus == SUCCESS)
171.           {
172.             FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//
173.             FLASH_SetLatency(FLASH_Latency_2);                   //ustaw zwloke dla pamieci
    Flash; zaleznie od taktowania rdzenia
174.                                                                  //0:<24MHz; 1:24~48MHz;
    2:>48MHz
175.             RCC_HCLKConfig(RCC_SYSCLK_Div1);                     //ustaw HCLK=SYSCLK
176.             RCC_PCLK2Config(RCC_HCLK_Div1);                      //ustaw PCLK2=HCLK
177.             RCC_PCLK1Config(RCC_HCLK_Div2);                      //ustaw PCLK1=HCLK/2
178.             RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9); //ustaw PLLCLK = HSE*9
    czyli 8MHz * 9 = 72 MHz
179.             RCC_PLLCmd(ENABLE);                                  //wlacz PLL
180.             while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET);  //odczekaj na poprawne
    uruchomienie PLL
181.             RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);           //ustaw PLL jako zrodlo
    sygnalu zegarowego
182.             while(RCC_GetSYSCLKSource() != 0x08);                //odczekaj az PLL bedzie
    sygnalem zegarowym systemu
183.
184.           /*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w
    programie peryferiow*/
185.             RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB, ENABLE);//wl
    acz taktowanie portu GPIO A
186.
187.           } else {
188.           }
189.         }
190.
191.
192.
193.         void GPIO_Config(void)
194.         {
195.           //konfigurowanie portow GPIO
196.           GPIO_InitTypeDef  GPIO_InitStructure;
197.         GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
198.           /*Tu nalezy umiescic kod zwiazany z konfiguracja poszczegolnych portow GPIO
    potrzebnych w programie*/
199.           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;
200.           GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
201.           GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      //wyjscie push-pull
202.           GPIO_Init(GPIOB, &GPIO_InitStructure);
203.
204.           GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
205.           GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
206.           GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;      //wejscie bez
    podciagania
207.           GPIO_Init(GPIOA, &GPIO_InitStructure);
208.
209.         }
```

+