**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**

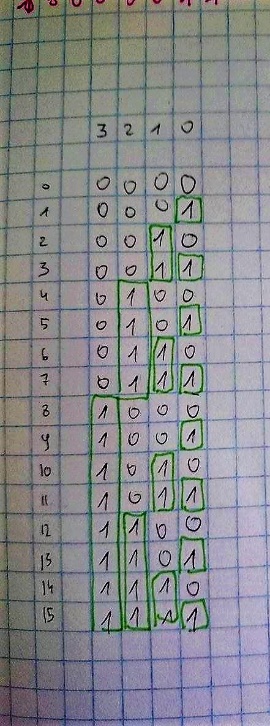
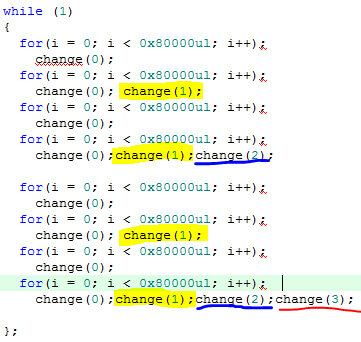
1. Napisz program, który miga 4 diodami w taki sposób, że każda następna miga 2 razy szybciej niż poprzednia.
2. Napisz program, który zapala i gasi diodę przyporządkowaną danemu przyciskowi   
   (SW0 – LED0, SW1 - LED1, itd.)
3. Napisz program, który wykorzystuje joystick do sterowania przesuwającą 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.

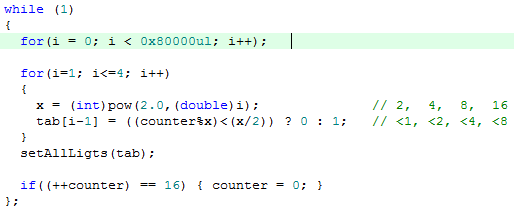
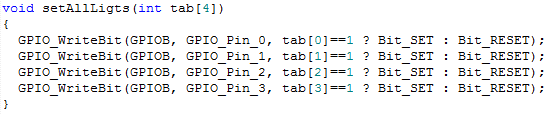
Rozpisaliśmy na kartce stany diód w interwałach(Rysunek 1). Jednostką był czas migania najszybszej diody. Pierwotnie zaimplementowaliśmy program w taki sposób, że zmienialiśmy stan diód wywołując napisaną przez nas funkcję (void change(int nr)) zmieniającą stan danej diody: diody nr0 => co interwał,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



Rysunek 2

Rysunek 1

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 6

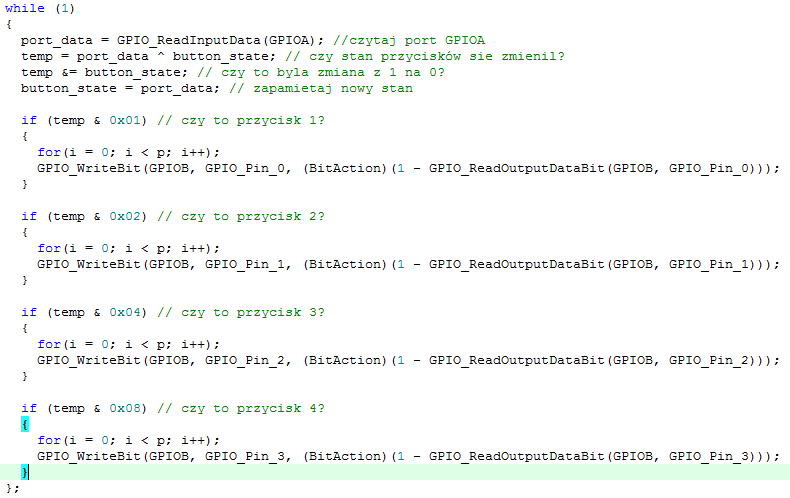
Rysunek 5

Rysunek 4

**Zadanie 2**

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

**Realizacja:**



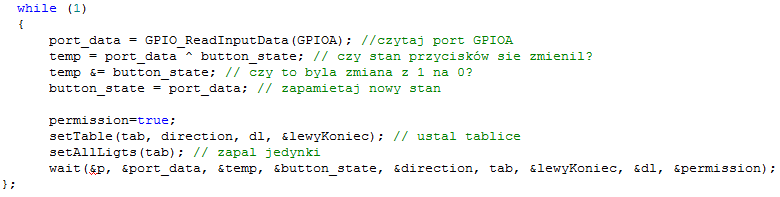
Rysunek 7

**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.

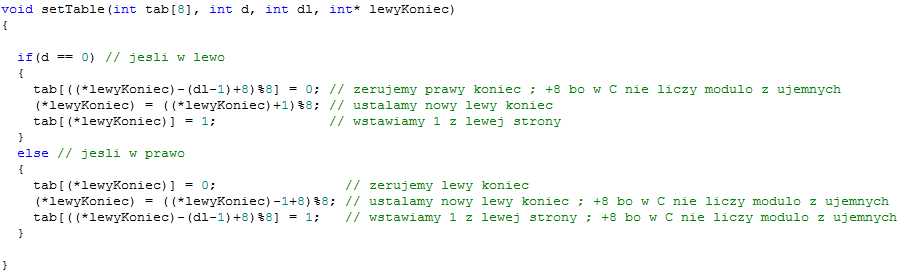
**Realizacja:**

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

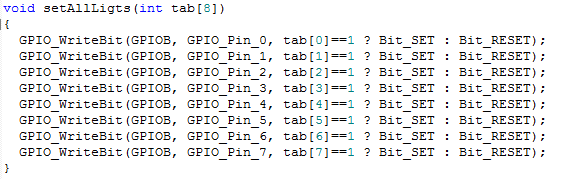


Rysunek 8

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.

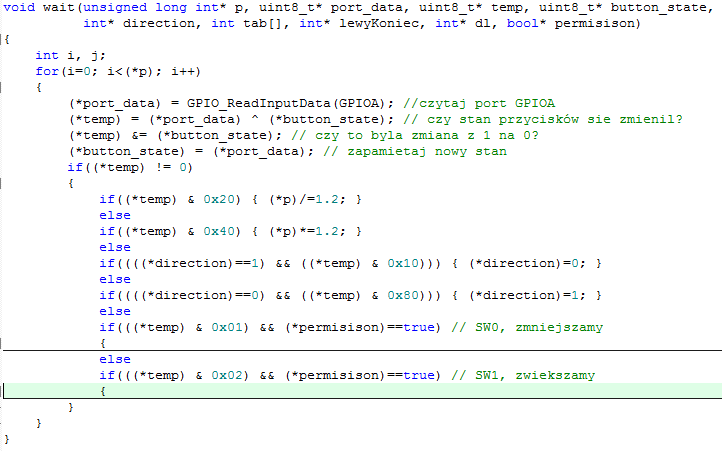


Rysunek 9



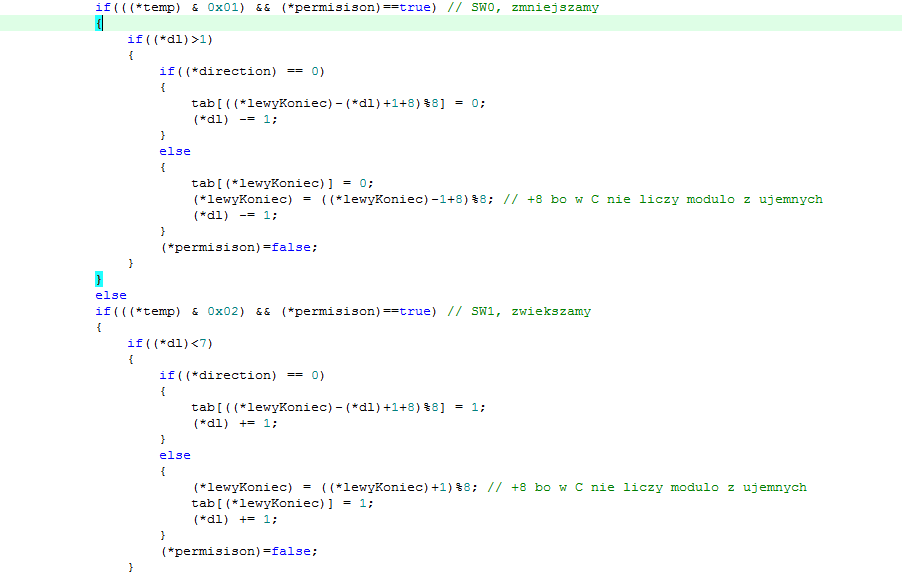
Rysunek 10

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



Rysunek 11

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



Rysunek 12

**Kody źródłowe plików main.c:**

**Zadanie 1**

1. #include "stm32f10x.h"
2. #include <math.h>
4. void GPIO\_Config(void);
5. void RCC\_Config(void);
7. void setAllLigts(int tab[4]);
9. int main(void)
10. {
11. volatile unsigned long int i;
12. int counter = 0;
13. int x;
14. int tab[4];
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);

23. while (1)
24. {
25. for(i = 0; i < 0x40000ul; i++);
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);
34. if((++counter) == 16) { counter = 0; }
35. };

38. return 0;
39. }
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. }

50. *//konfigurowanie sygnalow taktujacych*
51. void RCC\_Config(void)
52. {
53. ErrorStatus HSEStartUpStatus;                          *//zmienna opisujaca rezultat uruchomienia HSE*
55. RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB | RCC\_APB2Periph\_AFIO, ENABLE);
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*
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*
77. } else
78. {
79. }
80. }

83. void GPIO\_Config(void)
84. {
85. *//konfigurowanie portow GPIO*
86. GPIO\_InitTypeDef  GPIO\_InitStructure;
88. GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);
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"
3. void GPIO\_Config(void);
4. void RCC\_Config(void);
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 ;
12. *//konfiguracja systemu*
13. RCC\_Config();
14. GPIO\_Config();
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 | GPIO\_Pin\_5 | GPIO\_Pin\_6 | GPIO\_Pin\_7);
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*
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, GPIO\_Pin\_0)));
30. }
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, GPIO\_Pin\_1)));
36. }
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, GPIO\_Pin\_2)));
42. }
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, GPIO\_Pin\_3)));
48. }
49. };
50. return 0;
51. }

54. void RCC\_Config(void)
55. *//konfigurowanie sygnalow taktujacych*
56. {
57. ErrorStatus HSEStartUpStatus;                          *//zmienna opisujaca rezultat uruchomienia HSE*
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*
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*
80. } else {
81. }
82. }


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);
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);
102. }

**Zadanie 3**

1. #include "stm32f10x.h"
3. typedef int bool;
4. #define true 1
5. #define false 0
7. void GPIO\_Config(void);
8. void RCC\_Config(void);
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);

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**;
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;
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);

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*
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, &permission);
53. };


57. return 0;
58. }

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. }
117. (\*permisison)=**false**;
118. }
119. }
120. }
121. }
122. }

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. }

138. void setTable(int tab[8], int d, int dl, int\* lewyKoniec)
139. {
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. }
154. }





161. *//konfigurowanie sygnalow taktujacych ////////////////////////////////////////////////////////////////////////////*
162. void RCC\_Config(void)
163. {
164. ErrorStatus HSEStartUpStatus;                          *//zmienna opisujaca rezultat uruchomienia HSE*
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*
184. */\*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w programie peryferiow\*/*
185. RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_GPIOB, ENABLE);*//wlacz taktowanie portu GPIO A*
187. } else {
188. }
189. }


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);
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);
209. }

**+**