**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**

Zakres Materiału

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

Zadania do wykonania

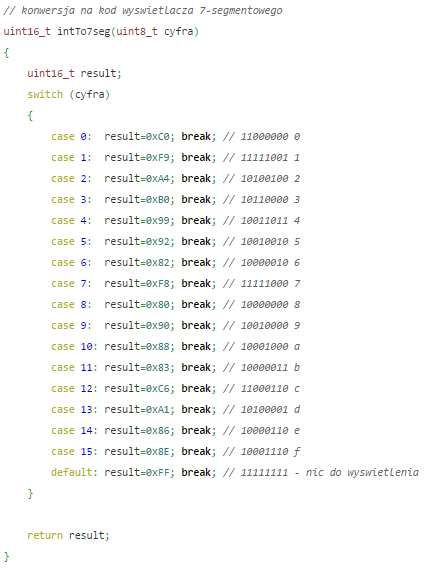
1. Napisz program wyświetlający co 1 sek. wszystkie cyfry w systemie szesnastkowym na pojedynczym wyświetlaczu.
2. Wykorzystaj przerwanie od timera SysTick do sterowania wyświetlaczami w trybie multipleksowanym.
3. Napisz program odmierzający czas (stoper) działający z dokładnością do 0.1 sek. Zaimplementuj 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:



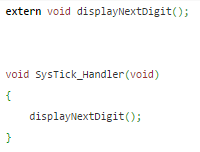
Rysunek 1

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.



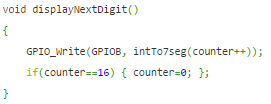
Rysunek 2

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



Rysunek 3

Właściwa akcja programu rozgrywa się właśnie w tej funkcji:



Rysunek 4

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

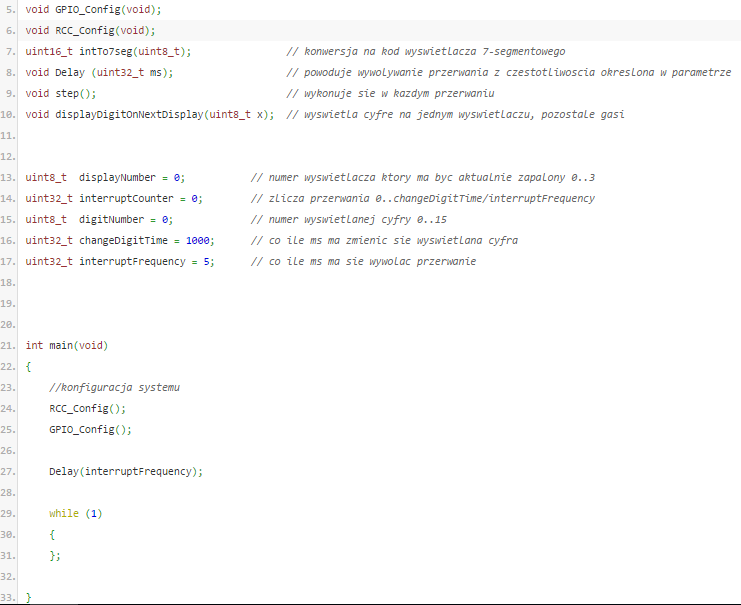
Zadanie 2

**Treść:**

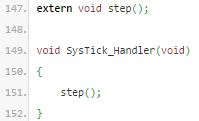
Wykorzystaj przerwanie od timera SysTick do sterowania wyświetlaczami w trybie multipleksowanym.

**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.

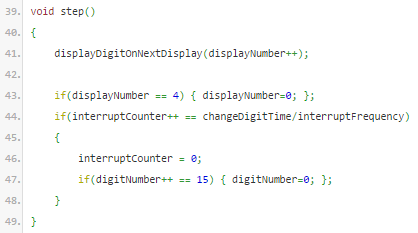


Rysunek 5



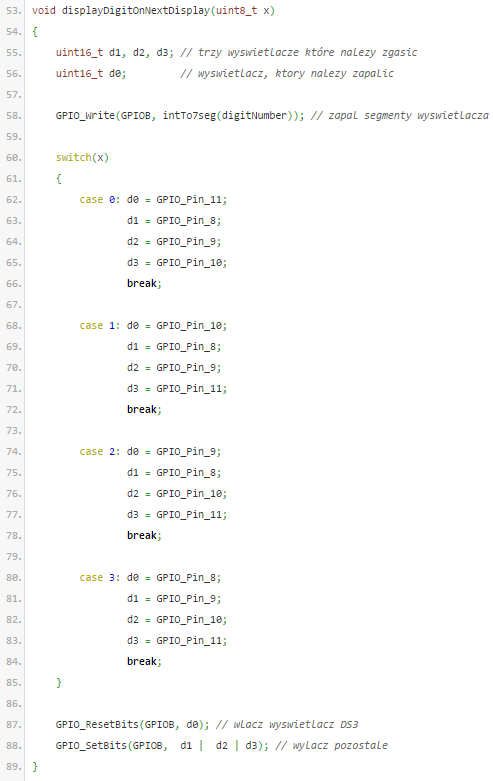
Rysunek 6

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 sięzmienić 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.



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).



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:**

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

7. uint16\_t intTo7seg(uint8\_t cyfra);
8. void displayNextDigit();
10. uint8\_t counter = 0;

13. void Delay (uint32\_t ms) *// nasza funkcja opóznienia wykorzystujaca timer SysTick*
14. {
15. if (SysTick\_Config(SysTick\_Frequency / 1000 \* ms))
16. {
17. while(1);
18. }
19. SysTick\_CLKSourceConfig(SysTick\_CLKSource\_HCLK\_Div8);
20. }


24. int main(void)
25. {
26. volatile unsigned long int j;
27. *//konfiguracja systemu*
28. RCC\_Config();
29. GPIO\_Config();
31. Delay(1000);
33. while (1)
34. {
35. };

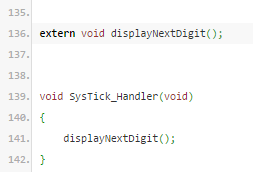
38. return 0;
39. }
41. void displayNextDigit()
42. {
43. GPIO\_Write(GPIOB, intTo7seg(counter++));
44. if(counter==16) { counter=0; };
45. }
47. *// konwersja na kod wyswietlacza 7-segmentowego*
48. uint16\_t intTo7seg(uint8\_t cyfra)
49. {
50. uint16\_t result;
51. switch (cyfra)
52. {
53. case 0:  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. }
72. return result;
73. }



78. void RCC\_Config(void)
79. *//konfigurowanie sygnalow taktujacych*
80. {
81. ErrorStatus HSEStartUpStatus;                          *//zmienna opisujaca rezultat uruchomienia HSE*
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);                                  *//wlacz 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 sygnalem zegarowym systemu*
101. */\*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w programie peryferiow\*/*
102. RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_GPIOB, ENABLE);*//wlacz taktowanie portu GPIO A*
104. }
105. else {}
106. }


110. void GPIO\_Config(void)
111. {
112. *//konfigurowanie portow GPIO*
113. GPIO\_InitTypeDef  GPIO\_InitStructure;
114. GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);
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);
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;      *//wejscie bez podciagania*
125. GPIO\_Init(GPIOA, &GPIO\_InitStructure);
126. }

Zmiany w pliku stm32f10x\_it.c



Zadanie 2

main.c

1. #include "stm32f10x.h"
2. #define SysTick\_Frequency 9000000

5. void GPIO\_Config(void);
6. void RCC\_Config(void);
7. uint16\_t intTo7seg(uint8\_t);                *// konwersja na kod wyswietlacza 7-segmentowego*
8. void Delay (uint32\_t ms);                   *// powoduje wywolywanie przerwania z czestotliwoscia okreslona w parametrze*
9. void step();                                *// wykonuje sie w kazdym przerwaniu*
10. void displayDigitOnNextDisplay(uint8\_t x);  *// wyswietla cyfre na jednym wyswietlaczu, pozostale gasi*

13. uint8\_t  displayNumber = 0;           *// numer wyswietlacza ktory ma byc aktualnie zapalony 0..3*
14. uint32\_t interruptCounter = 0;        *// zlicza przerwania 0..changeDigitTime/interruptTime*
15. uint8\_t  digitNumber = 0;             *// numer wyswietlanej cyfry 0..15*
16. uint32\_t changeDigitTime = 1000;      *// co ile ms ma zmienic sie wyswietlana cyfra*
17. uint32\_t interruptfrequency = 5;      *// co ile ms ma sie wywolac przerwanie*


21. int main(void)
22. {
23. *//konfiguracja systemu*
24. RCC\_Config();
25. GPIO\_Config();
27. Delay(interruptfrequency);
29. while (1)
30. {
31. };
33. }




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


53. void displayDigitOnNextDisplay(uint8\_t x)
54. {
55. uint16\_t d1, d2, d3; *// trzy wyswietlacze które nalezy zgasic*
56. uint16\_t d0;         *// wyswietlacz, ktory nalezy zapalic*
58. GPIO\_Write(GPIOB, intTo7seg(digitNumber)); *// zapal segmenty wyswietlacza*
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**;
68. case 1: d0 = GPIO\_Pin\_10;
69. d1 = GPIO\_Pin\_8;
70. d2 = GPIO\_Pin\_9;
71. d3 = GPIO\_Pin\_11;
72. **break**;
74. case 2: d0 = GPIO\_Pin\_9;
75. d1 = GPIO\_Pin\_8;
76. d2 = GPIO\_Pin\_10;
77. d3 = GPIO\_Pin\_11;
78. **break**;
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. }
87. GPIO\_ResetBits(GPIOB, d0); *// wlacz wyswietlacz DS3*
88. GPIO\_SetBits(GPIOB,  d1 |  d2 | d3); *// wylacz pozostale*
89. }

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. }
117. return result;
118. }


122. void Delay (uint32\_t ms) *// nasza funkcja opóznienia wykorzystujaca timer SysTick*
123. {
124. if (SysTick\_Config(SysTick\_Frequency / 1000 \* ms))
125. {
126. while(1);
127. }
128. SysTick\_CLKSourceConfig(SysTick\_CLKSource\_HCLK\_Div8);
129. }


133. void RCC\_Config(void)
134. *//konfigurowanie sygnalow taktujacych*
135. {
136. ErrorStatus HSEStartUpStatus;                          *//zmienna opisujaca rezultat uruchomienia HSE*
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);                                  *//wlacz 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 sygnalem zegarowym systemu*
155. */\*Tu nalezy umiescic kod zwiazany z konfiguracja sygnalow zegarowych potrzebnych w programie peryferiow\*/*
156. RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_GPIOB | RCC\_APB2Periph\_AFIO, ENABLE);*//wlacz taktowanie portu GPIO A*
158. } else {
159. }
160. }


164. void GPIO\_Config(void)
165. {
166. *//konfigurowanie portow GPIO*
167. GPIO\_InitTypeDef  GPIO\_InitStructure;
169. *// disable JTAG*
170. GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable, ENABLE);
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);
177. GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_8 | GPIO\_Pin\_9 | GPIO\_Pin\_10 | GPIO\_Pin\_11 ;
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);
182. }

Zmiany w pliku stm32f10x\_it.c

