Politechnika Opolska

L A B O R A T O R I U M

PRZEDMIOT: ***Technika mikroprocesorowa***

KIERUNEK STUDIÓW: ***Automatyka i Robotyka*** ROK STUDIÓW: ***III***

SPECJALNOŚĆ: **-**

SEMESTR: **VI** ROK AKADEMICKI: ***2019/2020***

Nr ćwiczenia: **3**

Temat ćwiczenia:

Obsługa timerów - MSP430.

Ćwiczenie wykonali:

Nazwisko:	*Imię:*	*Nazwisko:*	*Imię:*
1. Dziembowski	**Mateusz**	**2. --**	**--**
3. --	**--**	**4.**	

Uwagi:	*Data:*	*Ocena za sprawozdanie:*

1. Założenia

Wykorzystując zestaw MSP-EXP430G2 produkcji Texas Instruments, napisać program realizujący sygnalizator optyczny reagujący na kolejne naciśnięcia przycisku -z **wykorzystaniem timerów.** Stan programu (numer kolejnego naciśnięcia przycisku S2) odzwierciedlany będzie zachowaniem pokładowych diod świetlnych (LED1, LED2) w konfiguracji podanej w tabeli ***1.1*** poniżej ***:***

Tabela 1-1 *Tabela stanów sygnalizatora optycznego*

stan programu	LED1 czerwona	LED2 zielona
uruchomienie	OFF	OFF
pierwsze naciśnięcie przycisku S2	OFF	ON
drugie naciśnięcie przycisku S2	ON	ON
trzecie naciśnięcie przycisku S2	OFF	OFF
czwarte naciśnięcie przycisku S2	OFF	PULS
piąte naciśnięcie przycisku S2	PULS	OFF
szóste naciśnięcie przycisku S2	PULS	PULS
	synchronicznie	
siódme naciśnięcie przycisku S2	PULS	PULS
	naprzemiennie	
ósme naciśnięcie przycisku S2	OFF	OFF
	kasacja zliczania	

Jest to kolejne z cyklu, opracowanie sygnalizatora świetlnego, lecz w tym wykonaniu znikną pętle **for -** a pojawią się bloki odliczające typu **timer.** Opracowanie jest kontynuacją prac „*Obsługa portów -MSP430*" oraz „*Obsługa przerwań sprzętowych -MSP430*" autorstwa własnego. Stany sygnalizatora będą identyczne – zmieni się jedynie sposób odliczania czasu.

2. Opis kodu programu

Znaczenie fragmentu kodu zapisanego w liniach 20-32 zostało opisane w pracach poprzednich. Pojawił się nowy fragment dotyczący deklaracji bloków czasowych TA0 i TA1. W liniach 34 oraz 36 ustawiono przerwania wewnętrzne CPU dla obu jednostek. W liniach 35 oraz 37 ustawiono liczniki timerów na zliczające „w górę" . Opisywany fragment znajduje się poniżej :

```
20 int main(void)
21 {
22     WDTCTL = WDTPW + WDTHOLD;
23     P1DIR |= BIT0;                    // ustawienie  P1.0 (czerwona-LED) jako wyjście
24     P1DIR |= BIT6;                    // ustawienie  P1.6 (zielona-LED) jako wyjście
25     P1OUT &= ~BIT6;                   // ustawienie  P1.6 na off
26     P1OUT &= ~BIT0;                   // ustawienie  P1.0 na off
27     P1DIR &= ~BIT3;                   // ustawienie  P1.3 (przycisk S2) jako wejście
28     P1REN |=  BIT3;                   // wlaczenie rezystora pull-up na P1.3 (przycisk S2)
29 /* ustawienia dla przerwania na przycisku */
30     P1OUT |= BIT3;                    // gdy przycisk jest w górze
31     P1IES &= BIT3;                    // wybrane zbocze narastajace do wyzwolenia przerwania
32     P1IE |= BIT3;                     // włączenie przerwania na przycisku
33 /* ustawienia timerów */
34     TA0CCTL0 |= CCIE;                 // ustawienie przerwania timer_0
35     TA0CTL |= TASSEL_2  + TACLR ;     // liczenie w górę
36     TA1CCTL0 |= CCIE;                 // ustawienie przerwania timer_0
37     TA1CTL |= TASSEL_2  + TACLR ;     //liczenie w górę
38     TA0CCR0 = 1;                      // rejestr wartości dla timer_0
39     TA1CCR0 = 1;                      // rejestr wartości dla timer_1
40
41    _bis_SR_register(GIE);           // wlaczenie przerwan CPU
```

Podobnie jak w poprzednich pracach ,program wykonywany jest w pętli ***while*** zadeklarowanej w linii 43. Warunki stanu sygnalizatora , których świecenie pojedynczego elementu nie jest przerywane zostały określone jak poprzednio. Natomiast stany w których występują pulsacje wywołują przerwanie wewnętrzne w linii 54. Dla naciśnięć od 4 do 7 uruchamiany jest poleceniem **TA0CTL |= MC_1** pierwszy z użytych timerów. Składnia poniżej:

```
43 while(1)
44
45     { /*-------warunki i akcje poza timerami-------------------------------------*/
46
47           if (b==1)                    // pierwsze nacisniecie - dioda 2 on
48           { P1OUT |=BIT6;P1OUT &= ~BIT0;}
49           if (b==2)                    // drugie nacisniecie - dioda 1 on
50          { P1OUT |=BIT0; P1OUT &= ~BIT6;}
51           if (b==3)                   // trzecie nacisniecie - obie diody off i przypisanie b=0;
52           { P1OUT &= ~BIT6;P1OUT &= ~BIT0;}
53           if (b>=4%b<=7)                      // od 4 do 7  naciśnięcia
54           {TA0CTL |= MC_1; }         // wlaczenie timera 0
55
56
57       }
58 }
```

Inkrementacja zmiennej pomocniczej **b** jest identyczna jak w pracy „*Obsługa przerwań sprzętowych -MSP430*” . Wykonywane jest przerwanie w wykonywaniu programu głównego i przejście do linii 59 skryptu gdzie zmienna b zostanie zwiększona o 1 – ewentualnie zostanie wyzerowana po osiągnięciu wartości **b=8.** Następnie zostanie wyczyszczona flaga przerwania (linia 68) i po zwłoce z linii 69 program wróci do linii w której był przed wywołaniem przerwania przyciskiem S2. Dodatkowo pojawiły się wyłączenia timerów TA0 i TA1 w liniach 63-64. Składnia przerwania poniżej:

```
59 #pragma vector=PORT1_VECTOR
60
61 __interrupt void Port_1(void)
62 {
63     TA0CTL &= ~MC_3;                 // wyłączenie timera_0
64     TA1CTL &= ~MC_3;                 // wyłączenie timera_1
65     b=b+1;                           // inkrementacja zmiennej "b"
66     if (b==8)                        // wylaczenie
67     {   P1OUT &= ~BIT6; P1OUT &= ~BIT0; b=0;}
68     P1IFG &= ~BIT3;                    // czyszczenie pamięci przerwania
69     __delay_cycles(10000);             // trochę opóźnienia
70 }
```

Po natrafieniu na polecenie **TA0CTL |= MC_1** uruchamiany jest pierwszy z timerów TA0.W jego wnętrzu inkrementowana jest zmienna **czasomierz_1** i po odliczeniu nastawy **t_on** zostanie ona wyzerowana , timer wykona polecenie wyłączenia samego siebie i włączy następny -TA1 (linie 81-86). W trakcie tego przerwania podejmowane są różne stany diód sygnalizacyjnych w zależności od naciśnięcia przycisku S2 – linie 77-80. Opisana część poniżej:

```
71 /*-------TIMER_0-----------------------------------------*/
72 #pragma vector = TIMER0_A0_VECTOR
73 __interrupt void CCR0_ISR(void)
74 {
75     czasomierz_1++;
76      /*-------warunki i akcje w pierwszym timerze---------------------*/
77      if (b==4) {P1OUT |=BIT6;   P1OUT &= ~BIT0;}  // wszystkie wariancje
78      if (b==5) {P1OUT &= ~BIT6; P1OUT |=BIT0;  }
79      if (b==6) {P1OUT |=BIT6;P1OUT |=BIT0;}
80      if (b==7){P1OUT |=BIT6;P1OUT &= ~BIT0;}
81       if (czasomierz_1==t_on)
82           {
83           czasomierz_1=0;
84             TA0CTL &= ~MC_3;    // wyłączenie wlasne timera_0
85             TA1CTL |= MC_1;     // włączenie timera_1
86             __delay_cycles(4);
87           }
88 }
```

Timer TA0 w linii 85 włącza timer TA1 który ma bliźniaczą konstrukcję . Po jego wykonaniu program wraca do linii 53-54 w pętli głównej i ponownie do timera TA0. Taki stan się utrzyma aż zliczanie naciśnięcia przycisku S2 osiągnie wartość b=8. Składnia timera TA1 poniżej:

```
89 /*-------TIMER_1----------------------------------------*/
90 #pragma vector = TIMER1_A0_VECTOR        // Przerwanie po odliczeniu timer0
91 __interrupt void CCR1_ISR(void)
92 {
93     czasomierz_2++;
94      /*-------warunki i akcje w drugim timerze----------------------*/
95      if (b==4){P1OUT &= ~BIT6;P1OUT &= ~BIT0;}         // wszystkie wariancje
96      if (b==5) {P1OUT &= ~BIT6; P1OUT &= ~BIT0;   }
97      if (b==6) {P1OUT &= ~BIT6;P1OUT &= ~BIT0;}
98      if (b==7) {P1OUT &= ~BIT6;P1OUT |=BIT0;}
99       if (czasomierz_2==t_off)
100          {
101          czasomierz_2=0;
102            TA1CTL &= ~MC_3;// wyłączenie wlasne timera_1
103            __delay_cycles(4);
104          }
105 }
```

3. Pełen kod programu

```
/* PROGRAM SYGNALIZATORA OPTYCZNEGO 06062020*/
/* wykorzystanie timerów w MSP430*/
/* AiR_ns 2019/2020  */
/* Mateusz Dziembowski */

#include <msp430.h>
#include <intrinsics.h>
#include <stdio.h>

unsigned int b=0;                   // zmienna inkrementowana w przerwaniu
unsigned int i;                       // zmienna pomocnicza
unsigned int j;                       // zmienna pomocnicza
unsigned int p=0;
unsigned int t_on=30000;              // czas stanu wysokiego w pulsach
unsigned int t_off=30000;             // czas stanu niskiego w pulsach
int czasomierz_1=0;
int czasomierz_2=0;


int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= BIT0;                  // ustawienie  P1.0 (czerwona-LED) jako wyjście
    P1DIR |= BIT6;                  // ustawienie  P1.6 (zielona-LED) jako wyjście
    P1OUT &= ~BIT6;                 // ustawienie  P1.6 na off
    P1OUT &= ~BIT0;                 // ustawienie  P1.0 na off
    P1DIR &= ~BIT3;                 // ustawienie  P1.3 (przycisk S2) jako wejście
    P1REN |=  BIT3;                 // wlaczenie rezystora pull-up na P1.3 (przycisk S2)
/* ustawienia dla przerwania na przycisku */
    P1OUT |= BIT3;                  // gdy przycisk jest w górze
    P1IES &= BIT3;                  // wybrane zbocze narastajace do wyzwolenia przerwania
    P1IE |= BIT3;                   // włączenie przerwania na przycisku
/* ustawienia timerów */
    TA0CCTL0 |= CCIE;               // ustawienie przerwania timer_0
    TA0CTL |= TASSEL_2  + TACLR ;   // liczenie w góre
    TA1CCTL0 |= CCIE;               // ustawienie przerwania timer_0
    TA1CTL |= TASSEL_2  + TACLR ;   //liczenie w góre
    TA0CCR0 = 1;                    // rejestr wartości dla timer_0
    TA1CCR0 = 1;                    // rejestr wartości dla timer_1

   _bis_SR_register(GIE);          // wlaczenie przerwan CPU

while(1)

    { /*------warunki i akcje poza timerami---------------------------------------*/

          if (b==1)                   // pierwsze nacisniecie - dioda 2 on
          { P1OUT |=BIT6;P1OUT &= ~BIT0;}
          if (b==2)                   // drugie nacisniecie - dioda 1 on
         { P1OUT |=BIT0; P1OUT &= ~BIT6;}
          if (b==3)                   // trzecie nacisniecie - obie diody off i przypisanie b=0;
          { P1OUT &= ~BIT6;P1OUT &= ~BIT0;}
          if (b>=4%b<=7)                    // od 4 do 7  naciśnięcia
          {TA0CTL |= MC_1; }        // wlaczenie timera 0


      }
}
#pragma vector=PORT1_VECTOR

__interrupt void Port_1(void)
{
    TA0CTL &= ~MC_3;                // wyłączenie timera_0
    TA1CTL &= ~MC_3;                // wyłączenie timera_1
    b=b+1;                          // inkrementacja zmiennej "b"
    if (b==8)                       // wylaczenie
    {   P1OUT &= ~BIT6; P1OUT &= ~BIT0; b=0;}
    P1IFG &= ~BIT3;                 // czyszczenie pamieci przerwania
    __delay_cycles(10000);          // trochę opóźnienia
}
```

```
/*-------TIMER_0--------------------------------------*/
#pragma vector = TIMER0_A0_VECTOR
__interrupt void CCR0_ISR(void)
{
    czasomierz_1++;
     /*-------warunki i akcje w pierwszym timerze---------------------*/
     if (b==4) {P1OUT |=BIT6;   P1OUT &= ~BIT0;}  // wszystkie wariancje
     if (b==5) {P1OUT &= ~BIT6; P1OUT |=BIT0;  }
     if (b==6) {P1OUT |=BIT6;P1OUT |=BIT0;}
     if (b==7){P1OUT |=BIT6;P1OUT &= ~BIT0;}
      if (czasomierz_1==t_on)
          {
          czasomierz_1=0;
            TA0CTL &= ~MC_3;   // wyłączenie wlasne timera_0
            TA1CTL |= MC_1;    // włączenie timera_1
            __delay_cycles(4);
          }
 }
/*-------TIMER_1--------------------------------------*/
#pragma vector = TIMER1_A0_VECTOR       // Przerwanie po odliczeniu timer0
__interrupt void CCR1_ISR(void)
{
    czasomierz_2++;
     /*-------warunki i akcje w drugim timerze---------------------*/
     if (b==4){P1OUT &= ~BIT6;P1OUT &= ~BIT0;}        // wszystkie wariancje
     if (b==5) {P1OUT &= ~BIT6; P1OUT &= ~BIT0;  }
     if (b==6) {P1OUT &= ~BIT6;P1OUT &= ~BIT0;}
     if (b==7) {P1OUT &= ~BIT6;P1OUT |=BIT0;}
      if (czasomierz_2==t_off)
          {
          czasomierz_2=0;
            TA1CTL &= ~MC_3;// wyłączenie wlasne timera_1
            __delay_cycles(4);
          }
 }
```