

# Politechnika Białostocka Wydział Informatyki

# **Systemy Wbudowane**

Sprawozdanie nr 7 Timer0

Konrad Kotelczuk Karol Kamil Kowalski Dawid Kozak Dariusz Mikołajczuk

Pracownia specjalistyczna nr 7 Prowadzący: prof. dr hab. inż. Valery Salauyou

25 kwietnia 2013

# 1. Rejestry konfiguracyjne TMR0

# 1.1. Rejestr INTCON

7	6	5	4	3	2	1	0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
R/W-0	R/W - 0	R/W - x					

Legenda: R – bit może być odczytywany, W – zapisywany, – n – wartość po włączeniu zasilania (POR)

Nr.	Nazwa	Znaczenie	Przyjmowane wartości	uwagi
7	GIE	Zezwolenie na wszystkie niezamaskowane aktualnie przerwania	= 0 – przerwania wyłączone = 1 – przerwania włączone	_
6	EEIE	Maska przerwania generowanego po zakończonym zapisie bajtu w pamięci EEPROM	= 0 – przerwanie zablokowane = 1 – przerwanie odblokowane	_
5	TOIE	Maska przerwania generowanego w momencie przepełnienia licznika TMR0	= 0 – przerwanie zablokowane = 1 – przerwanie odblokowane	_
4	INTE	Maska przerwania zewnętrznego z linii INT	= 0 – przerwanie zablokowane = 1 – przerwanie odblokowane	_
3	RBIE	Maska przerwania zewnętrznego zgłaszanego przy zmianach sygnału na liniach RB4RB7 portu PORTB	= 0 – przerwanie zablokowane = 1 – przerwanie odblokowane	_
2	TOIF	Znacznik zgłoszenia przerwania po przepełnieniu licznika TMR0	= 0 – licznik TMR0 nie przepełnił się = 1 – nastąpiło przepełnienie licznika TMR0, czyli zmiana stanu licznika z FFh na 00h	Znacznik musi być zerowany programowo
1	INTF	Znacznik zgłoszenia przerwania zewnętrznego na linii RB0/INT	= 0 – przerwanie INT nie wystąpiło = 1 – wystąpiło przerwanie zewnętrzne INT, czyli zmiana stanu na wejściu RB0	Znacznik musi być zerowany programowo
0	RBIF	Znacznik zgłoszenia przerwania od zmiany stanu na liniach RB4RB7 portu PORTB	= 0 – brak zmiany stanu na liniach RB4RB7 = 1 – nastąpiła zmiana stanu na liniach RB4RB7 portu PORTB	W procedurze obsługi przerwania należy odczytać stan linii RB4RB7 aby umożliwić wykrywanie kolejncyh zmian. Znacznik RBIF musi być zerowany programowo.

# 1.2. Rejestr OPTION\_REG

7	6	5	4	3	2	1	0
!RBPU	INTEDG	T0CS	TOSE	PSA	PS2	PS1	PS0
R/W-1	R/W - 1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W - 1

Legenda: R – bit może być odczytywany, W – zapisywany, – n – wartość po włączeniu zasilania (POR)

Leger	idu. IX OII	t moze być odczytywany,	*	Lapisy	wany	, ii waitose po w	Tączemu zusmama (1 OTC)
Nr.	Nazwa	Znaczenie	Przyjmowane wartości			vartości	uwagi
7	!RBPU	Bit dołączający /odłączający wbudowane rezystory podciągające do napięcia zasilającego linii portu PORTB	= 0 – rezystory podciągające dołączone = 1 – rezystory podciągające odłączone				Rezystory podciągające domyślnie są odłączone.  Ponadto zostają one odłączane gdy linia portu zostanie skonfigurowana jako wyjście.
6	INTEDG	Bit wyboru zbocza, przy którym przerwanie zewnętrzne INT ma zostać zgłoszone	= 0 – przerwanie zgłaszane przy opadającym zboczu sygnału na wyprowadzeniu RB0/INT = 1 – przerwanie zgłaszane przy narastającym zboczu sygnału na wyprowadzeniu RB0/INT			ezu sygnału na RB0/INT zgłaszane przy oczu sygnału na	
5	TOCS	Bit wyboru źródła zliczanych impulsów dla timera TMR0	= 0 – TMR0 ma zliczać impulsy sygnału o częstotliwości sygnału zegarowego podzielonego przez 4 = 1 – TMR0 ma zliczać impulsy sygnału z wyprowadzenia RA4/T0CKI				_
4	TOSE	Bit wyboru zbocza, przy którym następuje zwiększenie wartości timera TMR0	= 0 – zwiększenie wartości TMR0 ma następować przy narastającym zboczu na wyprowadzeniu RA4/T0CKI = 1 – zwiększenie wartości TMR0 ma następować przy opadającym zboczu na wyprowadzeniu RA4/T0CKI			orzy narastającym wadzeniu e wartości TMR0 orzy opadającym	
3	PSA	Bit przyporządkowania preskalera do timera TMR0 lub licznika WDT	= 0 – preskaler przyporządkowany do timera TMR0 = 1 – preskaler przyporządkowany do licznika WDT			orzyporządkowany	_
20	PS2PS0	Bity wyboru wartości	PS2	PS1	PS0	Podział dla TMR0	Podział dla WDT
		współczynnika podziału preskalera	0	0	0	1:2	1:1
			0	0	1	1:4	1:2
			0	1	0	1:8	1:4
			0	1	1	1:16	1:8
			1	0	0	1:32	1:16
			1	0	1	1:64	1:32
			1	1	0	1:128	1:64
			1	1	1	1:256	1:128

# 2. Obsługa TMR0

Obsługa TMR0 może być opisana następująco:

- 1. Inicjalizacja:
  - i. wyczyszczenie bitów T0CS, T0SE, PSA rejestru OPTION REG (bank 1)
  - ii. ustawienie odpowiedniej wartości preskalera poprzez modyfikację bitów PS2, PS1, PS0 rejestru OPTION REG
  - iii. umieszczenie odpowiedniej wartości w rejestrze TMR0 (bank 0)
- 2. Oczekiwanie w pętli dopóki bit T0IF rejestru INTCON (bank 0) nie zostanie ustawiony
- 3. Wyczyszczenie bitu T0IF rejestru INTCON

# 3. Obliczania odmierzanych opóźnień

Po zainicjalizowaniu timera i ustawieniu odpowiedniej wartości preskalera, czas odmierzany przez TMR0 może być obliczony przy pomocy następującego wzoru:

$$T = \frac{4 \cdot P_{resc}}{f_{osc}} (256 - TMR0)$$

$$T - \text{czas odmierzany przez timer [cykle] (dla ZL4PIC: 1 cykl = 1 \mu s)}$$

$$f_{osc} - \text{częstotliwość zegara [MHz] (4 MHz dla ZL4PIC)}$$

$$P_{resc} - \text{wartość preskalera: } \{2, 4, 8, 16, 32, 64, 128, 256\}$$

$$TMR0 - \text{początkowa wartość rejestru TMR0: } \{0...255\}$$

#### 4. Testowanie TMR0

### 4.1. Inicjalizacja preskalera

Zostało utworzone **8 procedur** inicjalizujących TMR0 i preskaler. Przykładowa procedura inicjalizująca preskaler na 1/16 (pozostałe procedury mają trzy ostatnie bity w stałej w linii 5)

```
;; Inicjalizacja preskalera TMRO na 1/16
   ;; Czas wykonania procedury: 12 cykli
   init_presc_16:
1
2
     bsf
             STATUS, RPO
                                ; wybór banku 1 – OPTION_REG
3
             OPTION_REG, 0
     movf
     andlw b'10000000'
4
                                ; wyczyść wszystkie bity poza (7)
5
     iorlw b'01000011'
                                ; INTEDG=1, TOCS=0, TOSE=0, PSA=0, PS= 1/16
6
     movwf OPTION_REG
7
                                ; wybór banku O
     bof
             STATUS, RPO
8
     return
```

## 4.2. Procedura opóźniająca

```
;; Wykonanie opóźnienia TMRO
  ;; Parametr: W – początkowa wartość rejestru TMRO
  ;; Timer i preskaler muszą być zainicjalizowane wcześniej
 |delay_tmr0:
1
2
    movwf TMRO
                            ; TMRO = W (parametr procedury)
3
    nop
4
    btfss INTCON, TMROIF
                          ; czekaj na przepełnienie licznika
5
                            ; czekaj dopóki INTCON<TMROIF> jest wyzerowana
       goto $-1
6
            INTCON, TMROIF
                          ; wyczyszczenie flagi przepełnienia
    bcf
7
    return
```

#### 4.3. Procedura testowa

Do przetestowania TMR0 zostało utworzone **8 procedur testowych** (dla każdej z ośmiu wartości preskalera). Przykładowa procedura dla preskalera 16 wygląda następująco (pozostałe procedury odwołują się do innych procedur inicjalizujących w linii 5 oraz zawierają inne nazwy etykiet):

```
;; Procedura testowa opóźnienia TMRO z użyciem preskalera 1/16
   test_delay_16:
1
2
                             ; testowane opóźnienia od tej wartości TMRO do zera
     mov1w
               .255
3
     movwf
               TMROVAL
4
             init_presc_16  ; 10 cykli - inicjalizacja preskalera
     call
5
   loop_test_16:
                              ; W = kolejna wartość TMRO do testów
6
     movf
             TMROVAL, 0
7
             delay_tmr0
     call
8
     decfsz TMROVAL, 1
9
     goto
             loop_test_16
     clru.
                              ; test dla TMRO = 0
10
11
     call
             delay_tmr0
12
     return
```

#### 4.4. Testowanie

Każda z ośmiu procedur testowych została uruchomiona wielokrotnie w symulatorze MPLAB IDE, aby potwierdzić i zmodyfikować teoretyczny wzór na wartość opóźnienia.

Po uruchomieniu co najmniej kilkunastu testów dla każdej z procedur i różnych wartości początkowych rejestru TMR0, **opóźnienia realizowane przez procedurę delay\_tmr0** przedstawiają się następująco:

## 4.4.1. Wartości poprawki dla opóźnienia procedury delay tmr0

Wartość preskalera	Wartość poprawki [cykle] (zapis w C)				
2	10 + ( (TMR0 % 3 == 0) ? 0 : (3 - TMR0 % 3) )				
4	10 + ( (TMR0 + 1) % 3 )				
8	10 + ( (TMR0 % 3 == 0) ? 0 : (3 - TMR0 % 3) )				
16	10 + ( (TMR0 % 3 == 2) ? 0 : (1 + TMR0 % 3) )				
32	10 + ( (TMR0 % 3 == 0) ? 0 : (3 - TMR0 % 3) )				
64	10 + ( (TMR0 % 3 == 2) ? 0 : (1 + TMR0 % 3) )				
128	10 + ( (TMR0 % 3 == 0) ? 0 : (3 - TMR0 % 3) )				
256	10 + ( (TMR0 % 3 == 2) ? 0 : (1 + TMR0 % 3) )				

## 4.4.2. Zakresy czasów odmierzanych przez procedurę delay\_tmr0

- czas minimalny dla TMR0 = 255
- czas maksymalny dla TMR0 = 0

Wartość preskalera $P_{resc}$	czas minimalny [cykle]	czas maksymalny [cykle]
2	12	522 ( dla TMR0={0,1} )
4	15	1035
8	18	2058
16	27	4107
32	42	8202
64	75	16395
128	138	32778
256	267	65547

# 5. Odmierzanie dowolnego czasu

Ze względu na dość skomplikowany wzór na opóźnienie oraz niełatwy sposób odmierzenia czasów dłuższych niż 65547 cykli (μs) został utworzony dodatkowy program, który te wartości wyznacza.

### 5.1. Szkielet procedury delay1\_Xcykli (mniejsze wartości opóźnień)

```
delay1_Xcykli:
                                         ; 2 cykle call delay1_
2
               init_presc__<mark>c_presc</mark>
                                         ; 10 cykli: inicjalizacja preskalera
       call
3
       mov1w
               c_tmr0
                                         ; 1 cykl: początkowa wartość rej. TMRO
4
               delay_tmr0
                                         ; zależy od c_tmr0 i c_presc
       call
5
       < dodatkowe opóźnienie >
                                         ; opcjonalnie
6
                                         ; 2 cykle
       return
```

#### 5.2. Szkielet procedury delay2\_Xcykli (większe wartości opóźnienia)

```
1
   delay2_Xcykli:
                                      ; 2 cykle call delay2_
2
                                      ; 1 cykl: liczba wywołań delay_tmr0
       movlw
               c_iter
3
       movwf
               TMRCNT
                                      ; 1 cykl
4
                                      ; 10 cykli: inicjalizacja preskalera
       call
              init_presc__<mark>c_presc</mark>
5
   loop_delay2_X:
                                      ; pętla wykonująca się TMRCNT razy
6
       movlw c_tmr0
                                      ; 1 cykl
7
                                     ; zależy od c_tmr0 i c_presc
       call
               delay_tmr0
8
       decfsz TMRCNT, f
                                     ; 1 / 2 cykle
               loop_delay2_X
9
                                    ; 2 / 0 cykli
       goto
10
       mov1w
               c_dde1
                                    ; opcjonalnie – odmierzenie
                                     ; dodatkowego opóźnienia TMRO
11
       call
               delay_tmr0
12
        < dodatkowe opóźnienie >
                                    ; opcjonalnie
13
                                      ; 2 cykle
```

# 5.3. Program obliczający parametry procedur delay1\_Xcykli i delay2\_Xcykli

Do wyznaczenia wartości parametrów:

```
c_tmr0, c_presc, c_iter, c_ddel, oraz < dodatkowe opóznienie >
```

Wykorzystany został program napisany w C, program przyjmuje parametry:

- <exp cycle> oczekiwana liczba cykli opóźnienia procedury delay1 lub delay2
- <exp delta> dopuszczalna liczba cykli jakie należy umieścić w polu

```
< dodatkowe opóznienie >
```

• <max iter> – maksymalna dopuszczalna liczba iteracji c iter w procedurze delay2

#### Użycie:

```
$ ./tmr_delay <exp_cycle> <exp_delta> <max_iter>
$ ./tmr_delay --help
```

Program oraz wszystkie obliczenia dostępne są pod adresem: <a href="https://github.com/radomik/TimerCycleCount">https://github.com/radomik/TimerCycleCount</a>

#### Przykłady użycia:

#### Przykład 1: 30µs

```
$ ./tmr delay 30 10 255
                             # wymagane opóźnienie 30 cykli = 30 μs,
                                dopuszczalne dodatkowe opóźnienie: 10 cykli,
                              # maksymalnie 255 iteracji w delay2
### Obliczanie dla procedury delay1_:
  exp cycle: 30
  exp_cycle_min: 20
TMR0: 255, PRES:
                 2, CYCLE:
                                   27, DELTA: 3
TMR0: 254, PRES:
                  2, CYCLE:
                                   30, DELTA: 0
TMR0: 255, PRES:
                 4, CYCLE:
                                   30. DELTA: 0
```

Procedura opóźniająca o 30 μs może wyglądać następująco (przedostatni wiersz powyższego wyniku wywołania)

```
delay1_30us:
                                     ; 2 cykle call delay1_
                                     ; 10 cykli: inicjalizacja preskalera
2
       call
               init_presc_2
3
                .254
                                     ; 1 cykl: początkowa wartości rejestru TMRO
       movlw
4
       call
               delay_tmr0
5
                                     ; 2 cykle
       return
```

#### Przykład 2: 1 s

Procedura opóźniająca o 1 sekundę może wyglądać następująco:

```
1
                                          ; 2 cykle call delay2
   delay2_1s:
2
                                          ; 1 cykl: liczba wywołań delay_tmr0
                 .30
        mov1w
3
                                          ; 1 cykl
        movwf
                 TMRCNT
4
        call
                 init_presc_<mark>128</mark>
                                         ; 10 cykli: inicjalizacja preskalera
5
   loop_delay2_1s:
                                         ; petla wykonująca się TMRCNT razy
                                          ; 1 cykl
6
        mov1w
                 .0
7
        call
                 delay_tmr0
8
        decfsz
                 TMRCNT, f
                                         ; 1 / 2 cykle
9
                 loop_delay2_1s
                                         ; 2 / 0 cykli
        goto
                                         ; odmierzenie
10
        mov1w
                 .127
                 delay_tmr0
                                         ; dodatkowego opóźnienia TMRO
11
        call
12
        return
                                         ; 2 cykle
```

# 6. Rozwiązanie zadań

#### 6.1. Zrealizować opóźnienie 50 µs

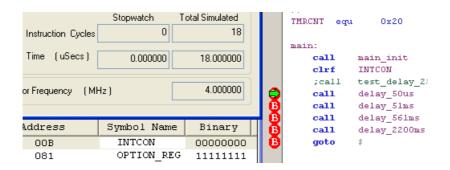
```
$ ./tmr delay 50 3 255
                               #
                                  wymagane opóźnienie 50 cykli = 50 μs,
                               #
                                  dopuszczalne dodatkowe opóźnienie: 3 cykli,
                               #
                                  maksymalnie 255 iteracji w delay2_
### Obliczanie dla procedury delay1_:
  exp_cycle: 50
  exp cycle min: 47
TMR0: 245, PRES:
                   2, CYCLE:
                                     48, DELTA: 2
                   2, C_ITER:
TMR0: 252, PRES:
                                                CYCLE:
                                                                   37, DELTA:
                                                                                  13
Add: movlw .255 ; call delay tmr0 on end
                                                NEW_CYCLE:
                                                                   50, NEW DELTA: 0
TMR0: 255, PRES:
                   4, C_ITER:
                                                CYCLE:
                                                                   34, DELTA:
                                                                                  16
                                  1,
Add: movlw .255 ; call delay_tmr0 on end
                                                NEW_CYCLE:
                                                                   50, NEW_DELTA: 0
```

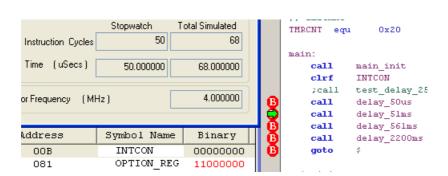
#### Procedura opóźniająca:

```
delay_50us:
                                   ; 2 cykle call delay1_
                                   ; 10 cykli (inicjalizacja preskalera)
2
       call
                init_presc_2
3
                .245
                                     1 cykl: początkowa wartości rejestru TMRO
       mov1w
4
       call
                delay_tmr0
5
                                   ; dodatkowe 2 cykle opóźnienia
       goto
               $+1
6
       return
                                   ; 2 cykle
```

#### Wyniki symulacji:

#### Przed:





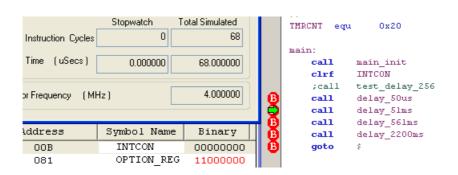
## 6.2. Zrealizować opóźnienie 51 ms

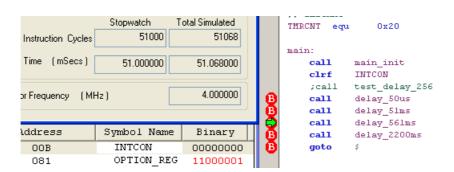
#### Procedura opóźniająca:

```
1
   delay_51ms:
                                    ; 2 cykle call delay2_
2
        mov1w
                 .50
                                     1 cykl: liczba wywołań delay_tmr0
3
        movwf
                 TMRCNT
                                      1 cykl
4
                                     10 cykli: inicjalizacja preskalera
        call
                 init_presc_4
5
   loop_51ms:
                                      pętla wykonująca się TMRCNT razy
6
        movlw
                 .10
                                      1 cykl
7
        call
                 delay_tmr0
8
                TMRCNT, f
                                     1 / 2 cykle
        decfsz
9
        goto
                 loop_51ms
                                    ; 2 / 0 cykli
10
        movlw
                 .13
                                      odmierzenie dodatkowego
                                     opóźnienia TMRO
        call
                 delay_tmr0
11
                                    ; 2 cykle
12
        return
```

#### Wyniki symulacji:

Przed:





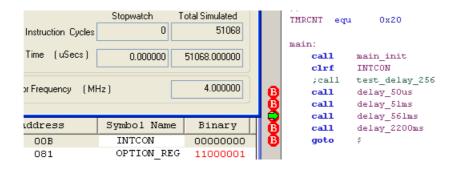
#### 6.3. Zrealizować opóźnienie 561 ms

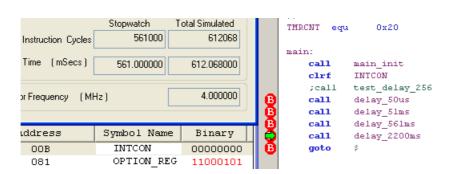
#### Procedura opóźniająca:

```
delay_561ms:
1
                                     ; 2 cykle call delay2_
2
        movlw
                 .35
                                       1 cykl: liczba wywołań delay_tmr0
3
        movwf
                 TMRCNT
                                       1 cykl
4
        call
                 init_presc_<mark>64</mark>
                                     ; 10 cykli: inicjalizacja preskalera
5
   loop_561ms:
                                       pętla wykonująca się TMRCNT razy
6
        movlw
                 .12
                                       1 cykl
7
        call
                 delay_tmr0
8
        decfsz
                 TMRCNT, f
                                       1 / 2 cykle
9
                 loop_561ms
                                     ; 2 / 0 cykli
        goto
10
                  .39
                                       odmierzenie dodatkowego
        movlw
11
                                     ; opóźnienia TMRO
        call
                 delay_tmr0
12
        return
                                     ; 2 cykle
```

#### Wyniki symulacji:

Przed:





#### 6.4. Zrealizować opóźnienie 2,20 s

#### Procedura opóźniająca:

```
delay_2200ms:
1
                                        ; 2 cykle call delay2_
2
        movlw
                  .45
                                          1 cykl: liczba wywołań delay_tmr0
3
        movwf
                 TMRCNT
                                          1 cykl
4
        call
                                        ; 10 cykli: inicjalizacja preskalera
                 init_presc_<mark>256</mark>
5
   loop_2200ms:
                                          pętla wykonująca się TMRCNT razy
6
        movlw
                 .69
                                          1 cykl
7
        call
                 delay_tmr0
8
        decfsz
                 TMRCNT, f
                                        ; 1 / 2 cykle
9
                 100p_2200ms
                                        ; 2 / 0 cykli
        goto
10
                 .80
                                        ; odmierzenie dodatkowego
        movlw
                                        ; opóźnienia TMRO
11
        call
                 delay_tmr0
12
        goto
                 $+1
                                          2 cykle dodatkowego opóźnienia
                                        ; 1 cykl dodatkowego opóźnienia
13
        nop
14
        return
                                          2 cykle
```

#### Wyniki symulacji:

Przed:

