



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 Salauiyou

25 kwietnia 2013

1. Rejestry konfiguracyjne TMR0

1.1. Rejestr INTCON

7	6	5	4	3	2	1	0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
R/W – 0	R/W – 0	R/W – 0	R/W – 0	R/W – 0	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	T0IE	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 RB4...RB7 portu PORTB	= 0 – przerwanie zablokowane = 1 – przerwanie odblokowane	–
2	T0IF	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 RB4...RB7 portu PORTB	= 0 – brak zmiany stanu na liniach RB4...RB7 = 1 – nastąpiła zmiana stanu na liniach RB4...RB7 portu PORTB	W procedurze obsługi przerwania należy odczytać stan linii RB4...RB7 aby umożliwić wykrywanie kolejnych zmian. Znacznik RBIF musi być zerowany programowo.

1.2. Rejestr OPTION_REG

7	6	5	4	3	2	1	0
!RBPU	INTEDG	T0CS	T0SE	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)

Nr.	Nazwa	Znaczenie	Przyjmowane wartości	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	–																																													
5	T0CS	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	T0SE	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	–																																													
3	PSA	Bit przyporządkowania preskalera do timera TMR0 lub licznika WDT	= 0 – preskaler przyporządkowany do timera TMR0 = 1 – preskaler przyporządkowany do licznika WDT	–																																													
2...0	PS2...PS0	Bity wyboru wartości współczynnika podziału preskalera	<table> <tr> <th>PS2</th><th>PS1</th><th>PS0</th><th>Podział dla TMR0</th><th>Podział dla WDT</th></tr> <tr> <td>0</td><td>0</td><td>0</td><td>1:2</td><td>1:1</td></tr> <tr> <td>0</td><td>0</td><td>1</td><td>1:4</td><td>1:2</td></tr> <tr> <td>0</td><td>1</td><td>0</td><td>1:8</td><td>1:4</td></tr> <tr> <td>0</td><td>1</td><td>1</td><td>1:16</td><td>1:8</td></tr> <tr> <td>1</td><td>0</td><td>0</td><td>1:32</td><td>1:16</td></tr> <tr> <td>1</td><td>0</td><td>1</td><td>1:64</td><td>1:32</td></tr> <tr> <td>1</td><td>1</td><td>0</td><td>1:128</td><td>1:64</td></tr> <tr> <td>1</td><td>1</td><td>1</td><td>1:256</td><td>1:128</td></tr> </table>	PS2	PS1	PS0	Podział dla TMR0	Podział dla WDT	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	
PS2	PS1	PS0	Podział dla TMR0	Podział dla WDT																																													
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 - czas odmierzany przez timer [cykle] (dla ZL4PIC: 1 cykl = 1 μs)

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

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

$TMR0$ - 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 TMR0 na 1/16
;; Czas wykonania procedury: 12 cykli
1  init_presc_16:
2      bsf      STATUS, RP0          ; wybór banku 1 - OPTION_REG
3      movf     OPTION_REG, 0
4      andlw    b'10000000'          ; 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      bcf      STATUS, RP0          ; wybór banku 0
8      return
```

4.2. Procedura opóźniająca

```
;; Wykonanie opóźnienia TMR0
;; Parametr: W - początkowa wartość rejestru TMR0
;; Timer i preskaler muszą być zainicjalizowane wcześniej
1  delay_tmr0:
2      movwf    TMR0                ; TMR0 = W (parametr procedury)
3      nop
4      btfss    INTCON, TMR0IF      ; czekaj na przepełnienie licznika
5      goto     $-1                 ; czekaj dopóki INTCON<TMR0IF> jest wyzerowana
6      bcf      INTCON, TMR0IF      ; wyczyszczenie flagi przepełnienia
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 TMR0 z użyciem preskalera 1/16
1  test_delay_16:
2      movlw    .255                ; testowane opóźnienia od tej wartości TMR0 do zera
3      movwf    TMR0VAL
4      call     init_presc_16      ; 10 cykli - inicjalizacja preskalera
5  loop_test_16:
6      movf     TMR0VAL, 0           ; W = kolejna wartość TMR0 do testów
7      call     delay_tmr0
8      decfsz   TMR0VAL, 1
9      goto     loop_test_16
10     clrw                     ; test dla TMR0 = 0
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 odmierzanym 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 (μ s) został utworzony dodatkowy program, który te wartości wyznacza.

5.1. Szkielet procedury `delay1_Xcykli` (mniejsze wartości opóźnień)

```
1 delay1_Xcykli:                ; 2 cykle call delay1_
2     call    init_presc__c_presc ; 10 cykli: inicjalizacja preskalera
3     movlw   c_tmr0              ; 1 cykl: początkowa wartość rej. TMR0
4     call    delay_tmr0           ; zależy od c_tmr0 i c_presc
5     < dodatkowe opóźnienie >    ; opcjonalnie
6     return                      ; 2 cykle
```

5.2. Szkielet procedury `delay2_Xcykli` (większe wartości opóźnień)

```
1 delay2_Xcykli:                ; 2 cykle call delay2_
2     movlw   c_iter              ; 1 cykl: liczba wywołań delay_tmr0
3     movwf   TMRCNT              ; 1 cykl
4     call    init_presc__c_presc ; 10 cykli: inicjalizacja preskalera
5 loop_delay2_X:                ; pętla wykonująca się TMRCNT razy
6     movlw   c_tmr0              ; 1 cykl
7     call    delay_tmr0           ; zależy od c_tmr0 i c_presc
8     decfsz  TMRCNT, f           ; 1 / 2 cykle
9     goto    loop_delay2_X       ; 2 / 0 cykli
10    movlw   c_ddel              ; opcjonalnie - odmierzenie
11    call    delay_tmr0           ; dodatkowego opóźnienia TMR0
12    < dodatkowe opóźnienie >    ; opcjonalnie
13    return                      ; 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 opóźnienie >`

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 opóźnienie >`
- `<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:

<https://github.com/radomik/TimerCycleCount>

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)

1	delay1_30us:	; 2 cykle call delay1_
2	call init_presc_2	; 10 cykli: inicjalizacja preskalera
3	movlw .254	; 1 cykl: początkowa wartości rejestru TMR0
4	call delay_tmr0	;
5	return	; 2 cykle

Przykład 2: 1 s

```
$ ./tmr_delay 1000000 3 255      # wymagane opóźnienie 1000000 cykli = 1 s,  
                                  # dopuszczalne dodatkowe opóźnienie: 3 cykli,  
                                  # maksymalnie 255 iteracji w delay2_  
...  
### Nie znaleziono żadnych rozwiązań dla podanych parametrów i procedury delay1_  
  
### Obliczanie dla procedury delay2_:  
...  
TMR0:  0, PRES: 128, C_ITER:  30,          CYCLE:  983475, DELTA:  16525  
Add: movlw .127 ; call delay_tmr0 on end    NEW_CYCLE:  1000000, NEW_DELTA: 0  
...
```

Procedura opóźniająca o 1 sekundę może wyglądać następująco:

1	delay2_1s:	; 2 cykle call delay2_
2	movlw .30	; 1 cykl: liczba wywołań delay_tmr0
3	movwf TMRCNT	; 1 cykl
4	call init_presc_128	; 10 cykli: inicjalizacja preskalera
5	loop_delay2_1s:	; pętla wykonująca się TMRCNT razy
6	movlw .0	; 1 cykl
7	call delay_tmr0	;
8	decfsz TMRCNT, f	; 1 / 2 cykle
9	goto loop_delay2_1s	; 2 / 0 cykli
10	movlw .127	; odmierzenie
11	call delay_tmr0	; dodatkowego opóźnienia TMR0
12	return	; 2 cykle

6. Rozwiązanie zadań

6.1. Zrealizować opóźnienie 50 μ s

```
$ ./tmr_delay 50 3 255      # wymagane opóźnienie 50 cykli = 50  $\mu$ s,
                             # 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
...
TMR0: 252, PRES:  2, C_ITER:    1,          CYCLE:      37, DELTA:    13
Add: movlw .255 ; call delay_tmr0 on end    NEW_CYCLE:    50, NEW_DELTA: 0

TMR0: 255, PRES:  4, C_ITER:    1,          CYCLE:      34, DELTA:    16
Add: movlw .255 ; call delay_tmr0 on end    NEW_CYCLE:    50, NEW_DELTA: 0
...
```

Procedura opóźniająca:

1	delay_50us:	; 2 cykle call delay1_
2	call init_presc_2	; 10 cykli (inicjalizacja preskalera)
3	movlw .245	; 1 cykl: początkowa wartości rejestru TMR0
4	call delay_tmr0	;
5	goto \$+1	; dodatkowe 2 cykle opóźnienia
6	return	; 2 cykle

Wyniki symulacji:

Przed:

Stopwatch		Total Simulated
Instruction Cycles	0	18
Time (uSecs)	0.000000	18.000000
or Frequency (MHz)	4.000000	

Address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11111111

```
...
TMRCNT equ    0x20

main:
    call    main_init
    clrf    INTCON
    ;call   test_delay_25
    call    delay_50us
    call    delay_51ms
    call    delay_561ms
    call    delay_2200ms
    goto    $
```

Po:

Stopwatch		Total Simulated
Instruction Cycles	50	68
Time (uSecs)	50.000000	68.000000
or Frequency (MHz)	4.000000	

Address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11000000

```
...
TMRCNT equ    0x20

main:
    call    main_init
    clrf    INTCON
    ;call   test_delay_25
    call    delay_50us
    call    delay_51ms
    call    delay_561ms
    call    delay_2200ms
    goto    $
```

6.2. Zrealizować opóźnienie 51 ms

```
$ ./tmr_delay 51000 0 255      # wymagane opóźnienie 51000 cykli = 51 ms,
                                # dopuszczalne dodatkowe opóźnienie: 0 cykli,
                                # maksymalnie 255 iteracji w delay2_
...
TMR0:  10, PRES:   4, C_ITER:   50,          CYCLE:          50015, DELTA:          985
Add: movlw . 13 ; call delay_tmr0 on end      NEW_CYCLE:      51000, NEW_DELTA:  0
...
```

Procedura opóźniająca:

1	delay_51ms:		; 2 cykle call delay2_
2	movlw	.50	; 1 cykl: liczba wywołań delay_tmr0
3	movwf	TMRCNT	; 1 cykl
4	call	init_presc_4	; 10 cykli: inicjalizacja preskalera
5	loop_51ms:		; pętla wykonująca się TMRCNT razy
6	movlw	.10	; 1 cykl
7	call	delay_tmr0	
8	decfsz	TMRCNT, f	; 1 / 2 cykle
9	goto	loop_51ms	; 2 / 0 cykli
10	movlw	.13	; odmierzenie dodatkowego
11	call	delay_tmr0	; opóźnienia TMR0
12	return		; 2 cykle

Wyniki symulacji:

Przed:

Stopwatch

Total Simulated

Instruction Cycles

0

68

Time (uSecs)

0.000000

68.000000

or Frequency (MHz)

4.000000

Address

Symbol Name

Binary

00B

INTCON

00000000

081

OPTION_REG

11000000

```

TMR0 equ 0x20

main:
    call    main_init
    clrf    INTCON
    ;call    test_delay_256
    call    delay_50us
    call    delay_51ms
    call    delay_561ms
    call    delay_2200ms
    goto    $

```

Po:

Stopwatch

Total Simulated

Instruction Cycles

51000

51068

Time (mSecs)

51.000000

51.068000

or Frequency (MHz)

4.000000

Address

Symbol Name

Binary

00B

INTCON

00000000

081

OPTION_REG

11000001

```

TMR0 equ 0x20

main:
    call    main_init
    clrf    INTCON
    ;call    test_delay_256
    call    delay_50us
    call    delay_51ms
    call    delay_561ms
    call    delay_2200ms
    goto    $

```

6.3. Zrealizować opóźnienie 561 ms

```
$ ./tmr_delay 561000 0 255      # wymagane opóźnienie 561000 cykli = 561 ms,
                                # dopuszczalne dodatkowe opóźnienie: 0 cykli,
                                # maksymalnie 255 iteracji w delay2_
...
TMR0: 12, PRES: 64, C_ITER: 35,      CYCLE: 547100, DELTA: 13900
Add: movlw .39 ; call delay_tmr0 on end NEW_CYCLE: 561000, NEW_DELTA: 0
...
```

Procedura opóźniająca:

1	delay_561ms:	; 2 cykle call delay2_
2	movlw .35	; 1 cykl: liczba wywołań delay_tmr0
3	movwf TMRCNT	; 1 cykl
4	call init_presc_64	; 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	goto loop_561ms	; 2 / 0 cykli
10	movlw .39	; odmierzenie dodatkowego
11	call delay_tmr0	; opóźnienia TMR0
12	return	; 2 cykle

Wyniki symulacji:

Przed:

Stopwatch

Total Simulated

Instruction Cycles

0

51068

Time (uSecs)

0.000000

51068.000000

or Frequency (MHz)

4.000000

address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11000001

B

B

B

B

B

```

...
TMRCNT equ 0x20

main:
    call main_init
    clrf INTCON
    ;call test_delay_256
    call delay_50us
    call delay_51ms
    call delay_561ms
    call delay_2200ms
    goto $

```

Po:

Stopwatch

Total Simulated

Instruction Cycles

561000

612068

Time (mSecs)

561.000000

612.068000

or Frequency (MHz)

4.000000

address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11000101

B

B

B

B

B

```

...
TMRCNT equ 0x20

main:
    call main_init
    clrf INTCON
    ;call test_delay_256
    call delay_50us
    call delay_51ms
    call delay_561ms
    call delay_2200ms
    goto $

```

6.4. Zrealizować opóźnienie 2,20 s

```
$ ./tmr_delay 2200000 3 255    # wymagane opóźnienie 2200000 cykli = 2,20 s,
                                # dopuszczalne dodatkowe opóźnienie: 3 cykle,
                                # maksymalnie 255 iteracji w delay2_
...
TMR0:  69, PRES: 256, C_ITER:   45,          CYCLE:          2154930, DELTA:          45070
Add: movlw . 80 ; call delay_tmr0 on end      NEW_CYCLE:       2199997, NEW_DELTA:  3
...
```

Procedura opóźniająca:

1	delay_2200ms:		; 2 cykle call delay2_
2	movlw .45		; 1 cykl: liczba wywołań delay_tmr0
3	movwf TMRcnt		; 1 cykl
4	call init_presc_256		; 10 cykli: inicjalizacja preskalera
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	goto loop_2200ms		; 2 / 0 cykli
10	movlw .80		; odmierzenie dodatkowego
11	call delay_tmr0		; opóźnienia TMR0
12	goto \$+1		; 2 cykle dodatkowego opóźnienia
13	nop		; 1 cykl dodatkowego opóźnienia
14	return		; 2 cykle

Wyniki symulacji:

Przed:

Stopwatch		Total Simulated
Instruction Cycles	0	612068
Time (uSecs)	0.000000	612068.000000
or Frequency (MHz)	4.000000	

Address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11000101


```

TMRcnt equ 0x20

main:
    call main_init
    clrf INTCON
    ;call test_delay_256
    call delay_50us
    call delay_51ms
    call delay_561ms
    call delay_2200ms
    goto $

```

Po:

Stopwatch		Total Simulated
Instruction Cycles	2200000	2812068
Time (Secs)	2.200000	2.812068
or Frequency (MHz)	4.000000	

Address	Symbol Name	Binary
00B	INTCON	00000000
081	OPTION_REG	11000111


```

TMRcnt equ 0x20

main:
    call main_init
    clrf INTCON
    ;call test_delay_256
    call delay_50us
    call delay_51ms
    call delay_561ms
    call delay_2200ms
    goto $

```