



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 Salaouyou

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 zboczach sygnału na wyprowadzeniu RB0/INT = 1 – przerwanie zgłaszane przy narastającym zboczach 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 zboczach na wyprowadzeniu RA4/T0CKI = 1 – zwiększenie wartości TMR0 ma następować przy opadającym zboczach 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 zmienione instrukcje: **bcf** ↔ **bsf** w liniach: 9, 10, 11)

```
;; 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   bcf OPTION_REG, TMR0CS ; cz stotliwo taktowania TMR0:
4   ; zegar wewn trzny / 4
5   bcf OPTION_REG, TMR0SE ; zwi kszenie warto ci TMR0 przy
6   ; narastaj cym zboczu na RA4/TOCKI
7   bcf OPTION_REG, PSA ; preskaler przyporz dkowany do
8   TMR0
9   bcf OPTION_REG, PS2 ; ustawienie preskalera
10  bsf OPTION_REG, PS1 ; ...
11  bsf OPTION_REG, PS0 ; ... na 1/16
12  bcf STATUS, RP0 ; wybór banku 0
13  return
```

4.2. Procedura opóźniająca

```
;; Wykonanie opó nienia TMR0
;; Parametr: W - pocz tkowa warto rejestru TMR0
;; Timer i preskaler musz by zainicjalizowane wcze niej
1 delay_tmr0:
2   movwf TMR0 ; TMR0 = W (parametr procedury)
3   nop
4   btfs INTCON, TMR0IF ; czekamy na przepe nienie licznika
5   goto $-1 ; czekaj dopóki INTCON<TMR0IF>
6   ; jest wyzerowany
7   bcf INTCON, TMR0IF ; wyczyszczenie flagi przepe nienia
8   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 opó nienia TMR0 z u yciem preskalera 1/16	
1	test_delay_16:	
2	movlw .255	; testowane opó nienia od tej warto ci
3		; TMR0 do zera
4	movwf TMR0VAL	
5	call init_presc_16	; 12 cykli - inicjalizacja preskalera
6	loop_test_16:	
7	movf TMR0VAL, 0	; W = kolejna warto TMR0 do testów
8	call delay_tmr0	
9	decfsz TMR0VAL, 1	
10	goto loop_test_16	
11		
12	clrw	; test dla TMR0 = 0
13	call delay_tmr0	
14	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_{presc}	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 odmierzania czasów dłuższych niż 65547 cykli (μs) został utworzony dodatkowy program, który te wartości wyznacza.

5.1. Szkielety procedur opóźniających

Procedury odmierzające konkretną wartość opóźnienia wyglądają w podobny sposób:

5.1.1. Procedura delay1_Xcykli (mniejsze wartości opóźnień)

1	delay1_Xcykli:	; 2 cykle call delay1_
2	movlw c_tmr0	; 1 cykl (pocz tkowa warto
3		; rejestru TMR0)
4	call init_presc__ c_presc	; 12 cykli (inicjalizacja
5		; preskalera)
6	call delay_tmr0	; zale y od c_tmr0 i c_presc
7	< dodatkowe opóźnienie >	; opcjonalnie
8	return	; 2 cykle

5.1.2. Procedura delay2_Xcykli (większe wartości opóźnień)

```
1 delay2_Xcykli:                ; 2 cykle call delay2_
2     movlw    c_iter           ; 1 cykl ( liczba wywo a
3                               ; delay_tmr0 )
4     movwf    TMRCNT           ; 1 cykl
5     call     init_presc__c_presc ; 12 cykli ( inicjalizacja
6                               ; preskalera )
7 loop_delay2_X:                ; p tla wykonuj ca si TMRCNT razy
8     movlw    c_tmr0           ; 1 cykl
9     call     delay_tmr0       ; zale y od c_tmr0 i c_presc
10    decfsz   TMRCNT, f        ; 1 / 2 cykle
11    goto     loop_delay2_X    ; 2 / 0 cykli
12    movlw    c_ddel           ; opcjonalnie - odmierzenie
13                               ; dodatkowego opoznienia TMR0
14    call     delay_tmr0       ; opcjonalnie
15    < dodatkowe opóźnienie > ; opcjonalnie
16    return                    ; 2 cykle
17
```

5.1.3. 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:

```
$ ./tmr_delay 80 10 255      # wymagane opóźnienie 80 cykli = 80 µs,  
                             # dopuszczalne dodatkowe opóźnienie: 10 cykli,  
                             # maksymalnie 255 iteracji w delay2_  
### Obliczanie dla procedury delay1_:  
    exp_cycle: 80  
    exp_cycle_min: 70  
TMR0: 235, PRES:  2, CYCLE:      71, DELTA: 9  
TMR0: 234, PRES:  2, CYCLE:      71, DELTA: 9  
TMR0: 245, PRES:  4, CYCLE:      71, DELTA: 9  
TMR0: 233, PRES:  2, CYCLE:      74, DELTA: 6  
TMR0: 232, PRES:  2, CYCLE:      77, DELTA: 3  
TMR0: 231, PRES:  2, CYCLE:      77, DELTA: 3  
TMR0: 244, PRES:  4, CYCLE:      77, DELTA: 3  
TMR0: 250, PRES:  8, CYCLE:      77, DELTA: 3  
TMR0: 253, PRES: 16, CYCLE:      77, DELTA: 3  
TMR0: 230, PRES:  2, CYCLE:      80, DELTA: 0  
TMR0: 243, PRES:  4, CYCLE:      80, DELTA: 0
```

Procedura opóźniająca o 80 µs może wyglądać następująco (przedostatni wiersz powyższego wyniku wywołania)

1	delay1_80us:	; 2 cykle call delay1_
2	movlw .230	; 1 cykl (pocz tkowa warto
3		; rejestru TMR0)
4	call init_presc_2	; 12 cykli (inicjalizacja
5		; preskalera)
6	call delay_tmr0	; zale y od c_tmr0 i c_presc
7	return	; 2 cykle

Przykład 2:

```
$ ./tmr_delay 1000000 10 255  # wymagane opóźnienie 1000000 cykli = 1 s,  
                             # dopuszczalne dodatkowe opóźnienie: 10 cykli,  
                             # maksymalnie 255 iteracji w delay2_  
### Obliczanie dla procedury delay1_:  
    exp_cycle: 1000000  
    exp_cycle_min: 999990  
### Nie znaleziono żadnych rozwiązań dla podanych parametrów i procedury delay1_  
  
### Obliczanie dla procedury delay2_:  
    exp_cycle: 1000000  
    exp_cycle_min: 999990  
    exp_delta: 10  
    max_iter: 255  
  
TMR0:  3, PRES: 16, C_ITER:  246,      CYCLE:      999515, DELTA:      485  
Add: movlw .227 ; call delay_tmr0 on end  NEW_CYCLE:      999990, NEW_DELTA: 10
```

TMR0: 120, PRES: 32, C_ITER: 228, Add: movlw .115 ; call delay_tmr0 on end	CYCLE: 995465, DELTA: 4535 NEW_CYCLE: 999990, NEW_DELTA: 10
TMR0: 114, PRES: 32, C_ITER: 219, Add: movlw .201 ; call delay_tmr0 on end	CYCLE: 998219, DELTA: 1781 NEW_CYCLE: 999990, NEW_DELTA: 10
TMR0: 86, PRES: 32, C_ITER: 183, Add: movlw .203 ; call delay_tmr0 on end	CYCLE: 998282, DELTA: 1718 NEW_CYCLE: 999990, NEW_DELTA: 10
...	
TMR0: 5, PRES: 32, C_ITER: 124, Add: movlw .189 ; call delay_tmr0 on end	CYCLE: 997845, DELTA: 2155 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 186, PRES: 64, C_ITER: 220, Add: movlw . 83 ; call delay_tmr0 on end	CYCLE: 988917, DELTA: 11083 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 114, PRES: 64, C_ITER: 109, Add: movlw .135 ; call delay_tmr0 on end	CYCLE: 992244, DELTA: 7756 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 107, PRES: 64, C_ITER: 103, Add: movlw . 1 ; call delay_tmr0 on end	CYCLE: 983667, DELTA: 16333 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 92, PRES: 64, C_ITER: 94, Add: movlw . 68 ; call delay_tmr0 on end	CYCLE: 987957, DELTA: 12043 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 174, PRES: 128, C_ITER: 94, Add: movlw .162 ; call delay_tmr0 on end	CYCLE: 987957, DELTA: 12043 NEW_CYCLE: 1000000, NEW_DELTA: 0
TMR0: 215, PRES: 256, C_ITER: 94, Add: movlw .209 ; call delay_tmr0 on end	CYCLE: 987957, DELTA: 12043 NEW_CYCLE: 1000000, NEW_DELTA: 0

Procedura opóźniająca o 1 sekundę może wyglądać następująco (trzeci wynik od końca)

1	delay2_1s:	; 2 cykle call delay2_
2	movlw .94	; 1 cykl (liczba wywo a
3		; delay_tmr0)
4	movwf TMRCNT	; 1 cykl
5	call init_presc_64	; 12 cykli (inicjalizacja
6		; preskalera)
7	loop_delay2_1s:	; p tla wykonuj ca si TMRCNT razy
8	movlw .92	; 1 cykl
9	call delay_tmr0	; zale y od c_tmr0 i c_presc
10	decfsz TMRCNT, f	; 1 / 2 cykle
11	goto loop_delay2_1s	; 2 / 0 cykli
12	movlw .68	; odmierzenie
13		; dodatkowego opoznienia TMR0
14	call delay_tmr0	;
15	return	; 2 cykle

6. Rozwiązanie zadań

6.1. Zrealizować opóźnienie 50 μ s

```
$ ./tmr_delay 50 10 255      # wymagane opóźnienie 50 cykli = 50  $\mu$ s,  
                             # dopuszczalne dodatkowe opóźnienie: 10 cykli,  
                             # maksymalnie 255 iteracji w delay2_  
### Obliczanie dla procedury delay1_  
exp_cycle: 50  
exp_cycle_min: 40  
TMR0: 250, PRES: 2, CYCLE: 41, DELTA: 9  
TMR0: 249, PRES: 2, CYCLE: 41, DELTA: 9  
...  
TMR0: 251, PRES: 4, CYCLE: 47, DELTA: 3  
TMR0: 245, PRES: 2, CYCLE: 50, DELTA: 0
```

Procedura opóźniająca:

1	delay_50us:	; 2 cykle call delay1_
2	movlw .245	; 1 cykl (pocz tkowa warto
3		; rejestru TMR0)
4	call init_presc_2	; 12 cykli (inicjalizacja
5		; preskalera)
6	call delay_tmr0	; zale y od c_tmr0 i c_presc
7	return	; 2 cykle

Wyniki symulacji:

Przed:

Stopwatch: 0, Total Simulated: 18
Instruction Cycles: 0
Time (uSecs): 0.000000
Processor Frequency (MHz): 4.000000

Code window (lines 17-32):

```
17 ;; Zmienne  
18 TMRcnt equ 0x20  
19 TMR0VAL equ 0x21  
20  
21 ;; bity rejestru OPTION_REG  
22 TMR0CS equ 5  
23 TMR0SE equ 4  
24  
25 main:  
26 call main_init  
27 clrf INTCON  
28 ;call test_delay_256  
29 call delay_50us  
30 call delay_51ms  
31 call delay_561ms  
32 call delay_2200ms
```

Po:

Stopwatch: 50, Total Simulated: 68
Instruction Cycles: 50
Time (uSecs): 50.000000
Processor Frequency (MHz): 4.000000

Code window (lines 20-32):

```
20  
21 ;; bity rejestru OPTION_REG  
22 TMR0CS equ 5  
23 TMR0SE equ 4  
24  
25 main:  
26 call main_init  
27 clrf INTCON  
28 ;call test_delay_256  
29 call delay_50us  
30 call delay_51ms  
31 call delay_561ms  
32 call delay_2200ms
```

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_  
...  
### Nie znaleziono żadnych rozwiązań dla podanych parametrów i procedury delay1_  
...  
TMR0: 14, PRES: 4, C_ITER: 51, CYCLE: 50099, DELTA: 901  
Add: movlw . 34 ; call delay_tmr0 on end NEW_CYCLE: 51000, NEW_DELTA: 0  
...
```

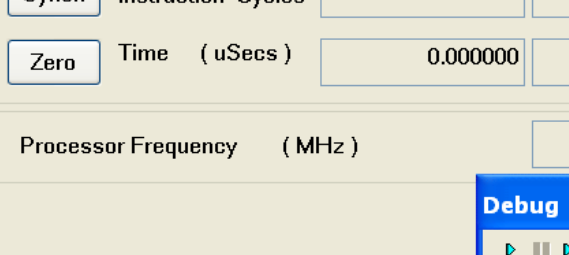
Procedura opóźniająca:

```

1 delay_51ms: ; 2 cykle call delay2_
2     movlw     .51 ; 1 cykl ( liczba wywo a
3                 ; delay_tmr0 )
4     movwf     TMRcnt ; 1 cykl
5     call      init_presc_4 ; 12 cykli ( inicjalizacja
6                 ; preskalera )
7 loop_51ms: ; p tla wykonuj ca si TMRcnt razy
8     movlw     .14 ; 1 cykl
9     call      delay_tmr0 ; zale y od c_tmr0 i c_presc
10    decfsz    TMRcnt, f ; 1 / 2 cykle
11    goto      loop_51ms ; 2 / 0 cykli
12    movlw     .34 ; odmierzenie
13              ; dodatkowego opoznienia TMR0
14    call      delay_tmr0 ;
15    return    ; 2 cykle

```

Wyniki symulacji:



Stopwatch **Total Simulated**

Synch Instruction Cycles 0 68

Zero Time (uSecs) 0.000000 68.000000

Processor Frequency (MHz) 4.000000

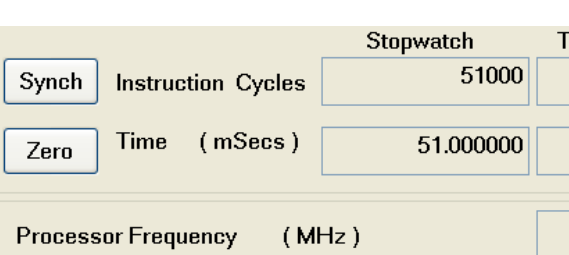
Debug [X]

▶ || ▶▶ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽

```

21 ;; bity rejestrui OPTION_REG
22 TMR0CS equ 5
23 TMR0SE equ 4
24
25 main:
26     call    main_init
27     clrf    INTCON
28     ;call   test_delay_256
29     call    delay_50us
30     call    delay_51ms
31     call    delay_561ms
32     call    delay_2200ms
33     goto    $

```



Stopwatch **Total Simulated**

Synch Instruction Cycles 51000 51068

Zero Time (mSecs) 51.000000 51.068000

Processor Frequency (MHz) 4.000000

Debug [X]

▶ || ▶▶ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽

```

21 ;; bity rejestrui OPTION_REG
22 TMR0CS equ 5
23 TMR0SE equ 4
24
25 main:
26     call    main_init
27     clrf    INTCON
28     ;call   test_delay_256
29     call    delay_50us
30     call    delay_51ms
31     call    delay_561ms
32     call    delay_2200ms
33     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_
...
### Nie znaleziono żadnych rozwiązań dla podanych parametrów i procedury delay1_
...
TMR0:  53, PRES:  64, C_ITER:   42,          CYCLE:          546269, DELTA:          14731
Add: movlw . 26 ; call delay_tmr0 on end      NEW_CYCLE:       561000, NEW_DELTA:  0
...
```

Procedura opóźniająca:

1	delay_561ms:		; 2 cykle call delay2_
2	movlw .42		; 1 cykl (liczba wywo a
3			; delay_tmr0)
4	movwf TMRCNT		; 1 cykl
5	call init_presc_64		; 12 cykli (inicjalizacja
6			; preskalera)
7	loop_561ms:		; p tla wykonuj ca si TMRCNT razy
8	movlw .53		; 1 cykl
9	call delay_tmr0		; zale y od c_tmr0 i c_presc
10	decfsz TMRCNT, f		; 1 / 2 cykle
11	goto loop_561ms		; 2 / 0 cykli
12	movlw .26		; odmierzenie
13			; dodatkowego opoznienia TMR0
14	call delay_tmr0		;
15	return		; 2 cykle

Wyniki symulacji:

		Stopwatch	Total Simulated
Synch	Instruction Cycles	0	51068
Zero	Time (uSecs)	0.000000	51068.000000
Processor Frequency (MHz)		4.000000	

Debug

```

21 ;; bity rejestru OPTION_REG
22 TMR0CS equ 5
23 TMR0SE equ 4
24
25 main:
26 call main_init
27 clrf INTCON
28 ;call test_delay_256
29 call delay_50us
30 call delay_51ms
31 call delay_561ms
32 call delay_2200ms
33 goto $
34

```

		Stopwatch	Total Simulated
Synch	Instruction Cycles	561000	612068
Zero	Time (mSecs)	561.000000	612.068000
Processor Frequency (MHz)		4.000000	

Debug

```

21 ;; bity rejestru OPTION_REG
22 TMR0CS equ 5
23 TMR0SE equ 4
24
25 main:
26 call main_init
27 clrf INTCON
28 ;call test_delay_256
29 call delay_50us
30 call delay_51ms
31 call delay_561ms
32 call delay_2200ms
33 goto $
34

```

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_
...
### Nie znaleziono żadnych rozwiązań dla podanych parametrów i procedury delay1_
...
TMR0:  69, PRES: 256, C_ITER:   45,          CYCLE:      2154932, DELTA:      45068
Add: movlw . 80 ; call delay_tmr0 on end      NEW_CYCLE:    2199999, NEW_DELTA:  1
...
```

Procedura opóźniająca:

1	delay_2200ms:		; 2 cykle call delay2_
2	movlw	.45	; 1 cykl (liczba wywo a
3			; delay_tmr0)
4	movwf	TMRCNT	; 1 cykl
5	call	init_presc_256	; 12 cykli (inicjalizacja
6			; preskalera)
7	loop_2200ms:		; p tla wykonuj ca si TMRCNT razy
8	movlw	.69	; 1 cykl
9	call	delay_tmr0	; zale y od c_tmr0 i c_presc
10	decfsz	TMRCNT, f	; 1 / 2 cykle
11	goto	loop_2200ms	; 2 / 0 cykli
12	movlw	.80	; odmierzenie
13			; dodatkowego opoznienia TMR0
14	call	delay_tmr0	;
15	nop		; 1 cykl dodatkowego opó nienia
	return		; 2 cykle

Wyniki symulacji:

The top screenshot shows the initial state of the simulation:

- Stopwatch: 0
- Total Simulated: 612068
- Time (uSecs): 0.000000
- Processor Frequency (MHz): 4.000000

The bottom screenshot shows the state after 2.2 seconds:

- Stopwatch: 2200000
- Total Simulated: 2812068
- Time (Secs): 2.200000
- Processor Frequency (MHz): 4.000000

The code on the right shows the assembly for the delay routine:

```

22 TMR0CS equ 5
23 TMR0SE equ 4
24
25 main:
26 call main_init
27 clrf INTCON
28 ;call test_delay_256
29 call delay_50us
30 call delay_51ms
31 call delay_561ms
32 call delay_2200ms
33 goto $
34
35 main_init:

```