Architektura mikrokontrolerów PIC16F8x

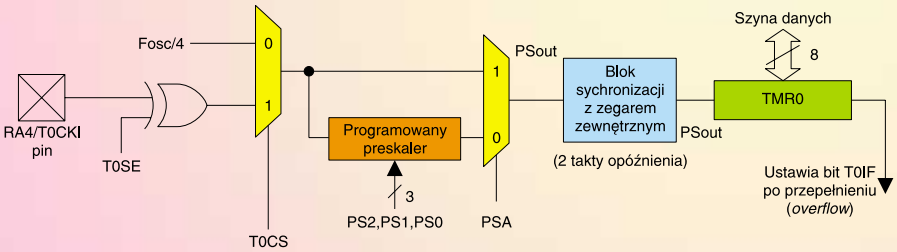
część 2

W drugiej części artykułu kontynuujemy opis funkcji rejestrów SFR w procesorach PIC16F8x, których dobra znajomość jest niezbędna do efektywnego wykorzystywania tych mikrokontrolerów.

Omówimy kolejny rejestr z obszaru SFR - TMR0 REGISTER (o adresie 01h). Jest to 8-bitowy licznik (liczący do przodu), którego zawartość można zapisywać (zakres 0...255) i odczytywać. Po wpisaniu wartości inkrementacja licznika jest zatrzymywana na czas trwania dwóch cykli instrukcyjnych, niezależnie od źródła zliczanych impulsów. Trzeba to uwzględnić przy obliczaniu wartości wpisywanej do licznika. Licznik TMR0 może pracować w dwóch trybach: timera i licznika. Wybór trybu jest realizowany przez odpowiednie ustawienie bitu T0CS w OPTION_REG (rys. 4).

Należy dodać, że pomiędzy źródło impulsów a licznik można włączyć programowany preskaler (wstępny dzielnik), poprzez wyzerowanie bitu PSA w OPTION_REG. Programowanie stopnia podziału preskalera następuje przez ustawienie bitów PS0...PS2 również w OPTION_REG. W trybie timera (T0CS=0) źródłem zliczanych impulsów jest wewnętrzny sygnał o częstotliwości oscylatora mikrokontrolera podzielonej przez cztery. W trybie licznika (T0CS=1) źródłem impulsów jest sygnał zewnętrzny podawany na wyprowadzenie RA4/T0CKI. Poprzez odpowiednie ustawienie bitu T0SE w OPTION_REG można wybrać zbocze impulsu, przy którym następuje inkrementacja licznika. Zliczanie zewnętrznych impulsów jest synchronizowane wewnętrznym sygnałem zegarowym mikroprocesora. Występuje tutaj opóźnienie między zboczem zewnętrznego impulsu, a odpowiadającym mu zwiększeniem stanu licznika (pomiar czasu pomiędzy dwoma zboczami sygnału zewnętrznego obarczony jest błędem *±4*tosc*). Synchronizacja przebiegu podawanego na RA4/T0CKI odbywa się poprzez dwukrotne próbkowanie wyjścia preskalera w każdym cyklu rozkazowym (fazy: Q2 i Q4). Z tego powodu poziom wysoki na wyjściu preskalera musi trwać co najmniej *2*tosc*, aby mógł być w ogóle wykryty przez układ próbkowania.

Przepełnienie licznika TMR0 może generować przerwanie. W momencie przepełnienia ustawiana jest flaga T0IF w rejestrze INTCON. Flaga ta musi być zerowana przez procedurę obsługi przerwania. Przerwanie zostanie zgłoszone, jeżeli bit maski T0IE w INTCON REGISTER będzie jedynką. Schemat blokowy TMR0 przedstawiony jest na **rys. 8**, a specyfikację rejestrów z nim związanych pokazano na **rys. 9**.



Ustawienie na jeden bitu PSA w OPTION_REG powoduje przypisanie preskalera do licznika watchdoga WDT (rys. 4). W takim przypadku przebieg podawany jest bezpośrednio na wejście licznika TMR0.

Rys. 8.

Przedstawmy przykład inicjalizacji licznika TMR0. Preskaler przypisany jest do TMR0 i dzieli wstępnie wewnętrzny przebieg *Fosc/4* przez 32.

```
 bcf STATUS,RP0 ;bank0
 clrf TMR0      ;zeruj TMR0
                ;i preskaler
 bsf STATUS,RP0 ;bank1
 bcf OPTION_REG,T0CS    ;przebieg
                ;Fosc/4
 bcf OPTION_REG,PSA     ;preskaler
                ;do TMR0
  bcf OPTION_REG,PS0    ;preskaler
                ;1/32
 bcf OPTION_REG,PS1
 bsf OPTION_REG,PS2

 bcf STATUS,RP0 ;bank0
 movlw stala_zliczania
         ;interesująca nas wartość
 movwf TMR0 ;wpis do licznika
```

Oczywiście można wpisać cały bajt do OPTION_REG. W przykładzie użyto celowo rozkazów *bcf* i *bsf*, aby pokazać możliwość ustawienia poszczególnych bitów.

Kolejne cztery rejestry obszaru SFR: EECON1, EECON2, EEDATA i EEADR (**rys. 10**) obsługują 64 bajty pamięci danych typu EEPROM (od adresu 00h do adresu 3fh), które można zapisywać i odczytywać podczas normalnej pracy mikrokontrolera.

Dostęp do tej pamięci jest możliwy tylko w sposób pośredni. W EEDATA zawarta jest dana, która jest zapisywana lub odczytywana. Adres tej danej wpisany jest do 8-bitowego rejestru EEADR. Ponieważ zaimplementowane są

Adres		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	(1)	(2)
01h	TMR0	Timer0 module's register								xxxx xxxx	uuuu uuuu
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 0000
81h	OPTION	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	xxx1 1111	xxx1 1111

Uwagi
x - wartość niewiadoma
u - wartość nie zmieniana
(1) wartości po włączeniu zasilania
(2) wartości po reset
Pola zaznaczone na szaro nie dotyczą TMR0

Rys. 9.

Adres		Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	(1)	(2)
08h	EEDATA	EEPROM data register								xxxx xxxx	uuuu uuuu
09h	EEADR	EEPROM address register								xxxx xxxx	uuuu uuuu
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	---0 q000
89h	EECON2	EEPROM control register 2								---- ----	---- ----

Uwagi:
x - wartość niewiadoma
u - wartość nie zmieniana
q - wartość nie zmieniana
(1) wartości po włączeniu zasilania
(2) wartości po zerowaniu

Rys. 10.

tylko 64 bajty, to dwa najstarsze bity EEADR muszą być wyzerowane.

Przed operacją zapisywania wywoływana jest procedura automatycznego kasowania zawartości komórki pamięci, do której będzie wpisywana nowa wartość. Czas zapisywania jest kontrolowany przez specjalny wewnętrzny timer i może się zmieniać w zależności od temperatury i napięcia zasilania. Ustawienie bitu zabezpieczenia przed odczytem powoduje, ze programator nie ma dostępu do pamięci EEPROM, ale mikrokontroler może realizować operacje zapisu i odczytu.

Aby odczytać dane z pamięci należy wpisać adres odczytywanej komórki do EEADR. Wpisanie jedynki do bitu RD rejestru EECON1 (**rys. 11**) inicjuje operację odczytu z pamięci. Bitu tego nie można wyzerować programowo. Jest zerowany automatycznie po zakończeniu operacji odczytu. Dane pojawiają się w EEDATA tak szybko, że można je czytać już w następnej instrukcji. Zawartość EEDATA nie zmienia się do momentu następnego odczytu lub zapisu do pamięci. Przykład procedury odczytu komórki o adresie 10h w pamięci EEPROM:

```
adree   equ 0x10    ;adres 10h
_rdeeprom movlw adree   ;adres w W
 movwf EEADR
 bsf STATUS,RP0     ;bank 1
 bsf EECON1,RD      ;inicjuj odczyt
 bcf STATUS,RP0     ;bank 0
 movf EEDATA,W ;dana z pamięci do W
```

Zapisywanie do pamięci EEPROM nie jest już tak proste. W pierwszej kolejności należy wpisać do EEADR adres, a do EEDATA zapisywany bajt. Następnie trzeba ustawić bit WREN rejestru EECON1 (wpisać jedynkę). Bit ten jest zerowany po włączeniu zasilania. Wprowadzenie dodatkowego bitu, który trzeba ustawiać przed operacją zapisu, pomaga uniknąć błędnych zapisów do pamięci po włączeniu zasilania, zanikach napięcia lub przy błędnym wykonywaniu programu. To jeszcze nie wszystko. Należy wykonać teraz specyficzną operację wpisania sekwencji wartości 55h i AAh do rejestru EECON2. Bez tej operacji zapis do pamięci nie rozpocznie się. Dopiero teraz wpisanie jedynki do bitu WR inicjuje operację zapisu do pamięci. Po jej zakończeniu zerowany jest bit WR i wpisywana jest jedynka do EEIF (rejestr EECON1). Jeżeli wpis do pamięci EEPROM zostanie nieoczekiwanie przerwany przez zerowanie wymuszone na wejściu MCLR lub zerowanie od watchdoga, to do bitu WRERR zostanie wpisana jedynka. W takim przypadku należy powtórzyć operację wpisu. Typowy czas zapisu wynosi ok. 10ms. Istnieją dwa sposoby sprawdzenia, że bajt jest już zapisany. Pierwszy z nich polega na odblokowaniu przerwania od kompletnego zapisu EEPROM (bit EEIE=1 w INTCON REGISTER). Wówczas po zakończeniu wpisu zgłaszane jest przerwanie. W procedurze obsługi należy wyzerować flagę EEIF. Drugi sposób polega na sprawdzaniu w pętli czy EEIF=1, lub czy WR=0 (bit EEIE=0). Poniżej podano przykład procedury wykorzystującej sprawdzanie w pętli bitu WR.

```
_wreeprom movlw adree
 movwf EEADR   ;zapisanie adresu
 movf W,dana
 movwf EEDATA  ;zapisanie danej

 bsf STATUS,RP0 ;bank 1
 bcf INTCON,GIE ;zablokowanie
                ;wszystkich przerwań
 bsf EECON1,WREN ;pozwolenie
                ;na wpis
 movlw 0x55     ;konieczna sekwencja
                ;inicjująca wpis
 movwf EECON2
 movlw 0xaa
 movwf EECON2
bsf EECON1,WR  ;start wpisu WR=1
bsf INTCON,GIE ;odblokowanie
               ;przerwań
_sprwrbtfsc EECON1,WR
               ;sprawdzanie bitu WR
goto _sprwr    ;WR=1
bcf EECON1,WREN   ;wpis kompletny
       ;i zablokowanie następnego
bcf STATUS,RP0 ;bank0
```

Producent zaleca, aby w trakcie wykonywania sekwencji zapisywania do EECON2 były zablokowane wszystkie przerwania. Dobrym zwyczajem jest też utrzymywanie bitu WREN w stanie zero, a wpisywanie jedynki tylko na czas wpisu do pamięci EEPROM. Zaleca się też, aby po każdym zapisie wykonać procedurę weryfikacji (odczytanie zapisanego bajtu i porównanie z zapisywanym). Wszystkie przedstawione tutaj mechanizmy mają na celu uchronienie przed wpisaniem przypadkowych danych do pamięci, w której przechowywane są zazwyczaj dość istotne dane.

Do omówienia pozostały jeszcze rejestry: PCLATCH (adresy 0ah i 8ah), PCL (adresy 02h i 82h), INDF (adresy 00h i 80h) oraz FSR (adresy 04h i 84h). Rejestr PCL zawiera 8 mniej znaczących bitów licznika rozkazów. Można go zapisywać i odczytywać. Rejestr PCLATCH zawiera 5 bardziej znaczących bitów licznika i nie można go bezpośrednio odczytywać ani zapisywać. Na rys. 12 przedstawiono sposób tworzenie licznika rozkazów PC z rejestrów PCL i PCLATCH w dwu różnych sytuacjach.

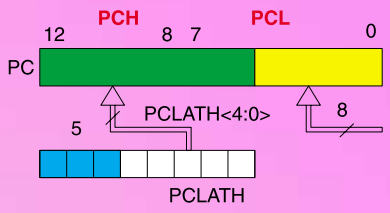
Możliwość wpisywania do rejestru PCL, i tym samym wykonywania sko-

bit 7:5 niezaimplementowane: czytane jako'0'

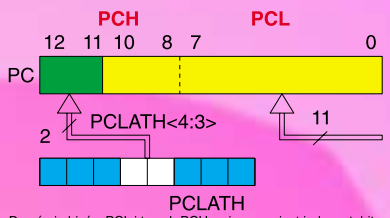
U	U	U	R/W-0	R/W-x	R/W-0	R/S-0	R/S-x
—	—	—	EEIF	WRERR	WREN	WR	RD
bit7							bit0

bit 4 EEIF
1 = operacja wpisu do EEPROM jest kompletna (musi być zerowana programowo)
0 = operacja wpisu nie jest jeszcze kompletna, lub jeszcze się nie rozpoczęła
bit 3 WRERR: bit błędu EEPROM
1 = operacja wpisu nieoczekiwanie zakończona
(MCLR reset lub WDT reset podczas operacji)
0 = operacja wpisu kompletna
bit 2 WREN: EEPROM bit zezwolenia na wpis
1 = zezwolenie na wpis
0 = zabronienie wpisu
bit 1 WR: EEPROM bit wpisu
1 = rozpoczyna cykl wpisu do pamięci. Można tylko wpisać jedynkę. Jest zerowany tylko sprzętowo
0 = cykl wpisu do EEPROM jest kompletny
bit 0 RD: EEPROM bit odczytu
1 = rozpoczyna cykl odczytu z pamięci . Można tylko wpisać jedynkę. Jest zerowany tylko sprzętowo
0 = cykl odczytu nie rozpoczęty

Rys. 11.



Do PCL wpisywany jest wynik działania instrukcji np. addwf PCL,f



Do ośmiu biyów PCL i trzech PCH wpisywany jest jedenastobitowy adres skoku z kodu rozkazu. Dwa pozostałe bity PCH są wpisywane z PCLATCH. Tak jest tworzony licznik rozkazów w wyniku działania rozkazów *goto* i *call*.

Rys. 12.

ków, wykorzystywana jest w procedurach odczytywania stałych umieszczonych w pamięci programu. Przeanalizujmy to na przykładzie:

```
z_stała  equ 0x0f
 movlw 0x02      ;do W przesunięcie
          ;względem adresu _rdconst
 call _rdconst
 movwf z_stała  ;zapamiętanie
      ;stałej w komórce pamięci ram
  .......
  .......
_rdconst addwf PCL,f     ;do PCL
           ;wynik operacji PCL+W
 retlw 0x35
 retlw 0x00
 retlw 0x12        ;powrót z proc.
          ;_rconst w W zawarte 0x12
 retlw 0x44
```

W wyniku działania tego fragmentu programu, do komórki *z_stała* pamięci RAM zostanie wpisana stała 12h. Gdyby w momencie wywołania *_rdconst* w rejestrze W była wartość np. 0, to do *z_stała* zostanie wpisana wartość 35h. Trzeba pamiętać, aby wartość W w momencie wywołania *_rdconst* nie zawierała większego przesunięcia niż ostatni rozkaz *retlw*. Kompilator asemblera MPASM zawiera dyrektywę *DT*, która pozwala uniknąć żmudnego wpisywania *retlw* dla większych tablic danych.

W naszym przykładzie procedurę *_rdconst* można zapisać inaczej:

```
_rdconst addwf PCL,f
 dt 0x35,0x00,0x12,0x44,
```

a kompilator sam już wpisze konieczne instrukcje *retlw*.

Rejestry INDF i FSR służą do pośredniego adresowania pamięci RAM. Każdą komórkę tej pamięci można zaadresować bezpośrednio podając jej adres w kodzie rozkazu, np. *movwf 0x20* - wpisz zawartość W do komórki o adresie 20h. Często jednak jest potrzebny bufor danych w pamięci RAM. Poszczególne elementy tego bufora są adresowane za pomocą wskaźnika, który jest zawarty w jakimś rejestrze. Do tego można wykorzystywać mechanizm adresowania pośredniego. Aby zaadresować komórkę pamięci należy do rejestru FSR wpisać jej adres. Jeżeli chcemy odczytać komórkę o adresie zawartym w FSR, to trzeba teraz odczytać zawartość INDF.

Jeżeli chcemy wpisać daną pod adres określony w FSR, to trzeba tą daną wpisać do INDF. Pokażemy to na przykładzie odczytywania danych z bufora:

```
   movlw 0x0c
   movwf FSR      ;do FSR adres
            ;początku bufora danych

movf INDF,w     ;do W zawartość
           ;komórki o adresie 0ch
.....
.....
incf FSR,f    ;FSR=FSR+1 następna
              ;pozycja w buforze
movf INDF,w  ;do W zawartość
              ;komórki o adresie 0dh
.....
```

Na tym kończymy opisywanie rejestrów z obszaru SFR.

Tomasz Jabłoński, AVT
tomasz.jablonski@ep.com.pl