MSP430: mikrokontrolery, które (prawie) nie pobierają prądu, część 2

Środowisko programistyczne

Oprogramowanie umożliwiające wygenerowanie kodu wynikowego o objętości do 4 kB można ściągnąć ze strony *www.iar.com* po uprzednim zarejestrowaniu się. Wbrew pozorom 4 kB kodu wynikowego to naprawdę bardzo dużo. Jeśli w programie nie ma dużych tablic i nie używa się wyrafinowanych i pamięciożernych konstrukcji językowych, to program wynikowy w większości przypadków zmieści się w takiej objętości. Warto pamiętać, że ograniczenie objętości dotyczy jedynie kodu stworzonego przez programistę. Używając bibliotek wbudowanych (np. funkcji *printf* czy funkcji matematycznych) można wygenerować kod o większej objętości.

Na **rys. 5** pokazano wygląd ekranu środowiska programowania firmy IAR (Workbench IDE).

Na zrzucie ekranowym są widoczne punkty zatrzymania (*breakpoints*). Środowisko umożliwia oglądanie rejestrów procesora po każdym zatrzymaniu. Można ponadto zdefiniować dowolne wyrażenie i oglądać jego zmiany w trakcie działania programu (patrz okno *watch*).

Na **list. 1** pokazano przykład prostego programu w języku C. Jego odpowiednik napisany w asemblerze pokazano na **list. 2**. Obydwa przykłady działają tak samo: po ustawieniu rejestrów procesora w pętli głównej wprowadza się procesor w tryb uśpienia *LPM3* i jednocześnie uaktywnia się układ *watchdog*, w tym przypadku w roli zwykłego licznika. W efekcie z częstotliwością

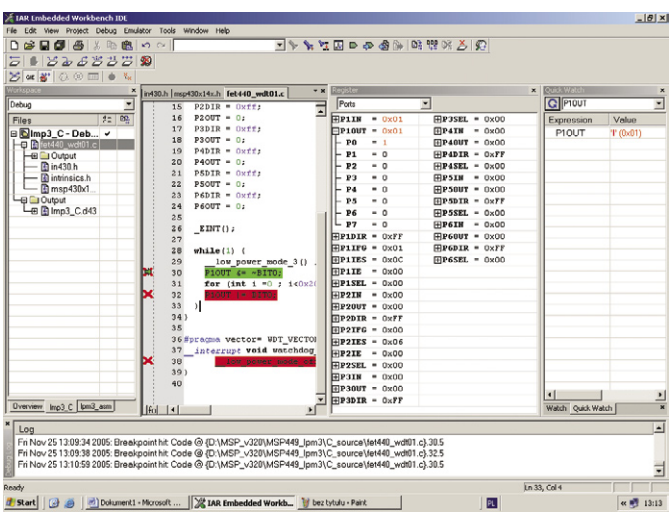
Istnieje wiele kompilatorów umożliwiających programowanie procesorów MSP w językach asembler i C. Firma TI w notach aplikacyjnych i dostarczanych zestawach ewaluacyjnych używa oprogramowania firmy IAR, ma ono bowiem niebagatelna zaletę: w wersji do 4 kB jest dostępne bezpłatnie.

Uwaga! W EPooL11/2007 opublikujemy nową wersję płyty z notami katalogowymi i narzędziami dla mikrokontrolerów z rodziny MSP430.

1 Hz jest uruchamiany podprogram obsługi przerwania, w którym mikrokontroler wybudza się ze stanu uśpienia. Procesor powraca do pętli głównej, gdzie zapala diodę LED i odmierza czas *delay* (ok. 0,2 s) korzystając z pętli oczekiwania, a następnie wyłącza diodę LED i usypia wchodząc w tryb *LPM3*. W trybie uśpienia wyłączony jest zegar taktujący rdzeń procesora. Działa jedynie oscylator ACLK o częstotliwości 32,768 kHz. Oscylator ten używany jest do taktowania układu *watchdog*. Obudzony procesor używa wewnętrznego układu *DCO* generującego przebiegi o częstotliwości ok. 800 kHz (procesory serii F1xx) lub 1 MHz (procesory serii F4xx). Sygnał z DCO – zegar systemowy MCLK – jest użyty do taktowania rdzenia procesora.

Sposób działania programu umożliwia pomiar poboru prądu. W trakcie działania powinno być to ok. 300 µA, a w trakcie uśpienia ok. 1 µA. Prąd w stanie uśpienia jest rzeczywiście mały i trzeba dysponować odpowiednim miernikiem by jego określenie było możliwe w warunkach domowego laboratorium.

Przyjrzyjmy się programowi w języku C. W wierszu #6 włącza się zbiór nagłówkowy *Header File*. Ten zbiór zawiera definicje wszystkich rejestrów, pól bitowych i przerwań procesora. Warto z niego korzystać zamiast samemu liczyć bity i adresy w pamięci procesora. Aby zobaczyć co jest w takim zbiorze wystarczy dwa razy klikać na nazwę zbioru w drzewie projektu w oknie *workspace*. W wierszu #11 znajduje się instrukcja `IE1 |= WDTIE`. Poleca ona ustawić bit *WDTE* w rejestrze *IE1* bez zmiany pozostałych bitów. Gdyby trzeba było wyzerować ten sam bit, bez naruszania innych bitów, to zapis ten wyglądać powinien `IE1 &= ~WDTIE`.



Rys. 5. Okno środowiska uruchomieniowego Workbench IDE w wersji dla mikrokontrolerów MSP430

List. 1. Prosty program demonstracyjny napisany w języku C

```
01 //-------------------------------------------------------------------------
02 // Pogram ilustruje działanie trybu oszędzania emergii LPM3.
03 // Zapalenie diody LED P1.0 i jej świecenie ma miejsce w trybie normalnej pracy
04 // Zgaszenie diody sygnalizuje tryb LPM3 (tryb uśpienia)
05 //-------------------------------------------------------------------------
06 #include <msp430x14x.h>                          // definicje rejestrów procesora
07
08 void main(void) {
09
10  WDTCTL = WDT_ADLY_1000 ;                        // czas działania WDT = 1s i
11  IE1 |= WDTIE;                                   // odblokowanie WDT
12
13  P1DIR = 0xff;                                   // P1 output
14  P1OUT = 0;                                      // P1 reset
15  P2DIR = 0xff;                                   // nieużywane porty w trybie out ...
16  P2OUT = 0;
17  P3DIR = 0xff;
18  P3OUT = 0;
19  P4DIR = 0xff;
20  P4OUT = 0;
21  P5DIR = 0xff;
22  P5OUT = 0;
23  P6DIR = 0xff;
24  P6OUT = 0;
25
26  _EINT();                                        // odblokowanie przerwań
27
28  while(1) {
29    __low_power_mode_3() ;
30   P1OUT &= ~BIT0;                                // włącz LED
31   for (int i =0 ; i<0x2000; i++);                // opóźnienie: lokalna deinicja i
32   P1OUT |= BIT0;                                 // wyłącz LED
33  }
34 }
35
36 #pragma vector= WDT_VECTOR                       // podprogram obsługi przerwania
37 __interrupt void watchdog_timer( void ) {
38   __low_power_mode_off_on_exit() ;               // wyjście ze stanu LPM3
39 }
```

Takie konstrukcje językowe występują w wierszach #30 i # 32 gdy zapala się i gasi diodę LED. Zadaniem instrukcji w wierszach #11...#24 jest ustawienie wszystkich portów P1...P6 jako porty wyjściowe. Dlaczego tak? Porty wejściowe, mając bardzo dużą oporność wewnętrzną mogą być podatne na zakłócenia (przypadkowe przełączenia). Ustawiając je w taki sposób istotnie zmniejszamy ich wrażliwość na zakłócenia oraz zyskujemy kontrolę nad podłączonymi do portów układami (np. diodami LED).

List. 2. Odpowiednik programu z list. 1 napisany w asemblerze

```
01 ;-------------------------------------------------------------------------
02 ; Pogram ilustruje działanie trybu oszczędzania energii LPM3.
03 ; Zapalenie diody LED P1.0 i jej świecenie ma miejsce w trybie działanie
04 ; Zgaszenie diody sygnalizuje tryb LPM3 (tryb uśpienia)
05 ;-------------------------------------------------------------------------
06 #include  „msp430x14x.h”
07
08        ORG   01100h                           ; start programu
09 RESET    mov.w  #0A00h,SP                     ; ustawienie stosu
10 SetupWDT  mov.w  #WDT_ADLY_1000,&WDTCTL        ; WDT ~1000ms: czas układu watchdog
11        bis.b  #WDTIE,&IE1                     ; odblokowanie przerwania WDT
12 SetupPx   mov.b  #0FFh,&P6DIR                  ; P6 output: wyjście
13        clr.b  &P6OUT                          ;
14        mov.b  #0FFh,&P5DIR                    ; P5 output: wyjście
15        clr.b  &P5OUT                          ;
16        mov.b  #0FFh,&P4DIR                    ; P4 output: wyjście
17        clr.b  &P4OUT                          ;
18        mov.b  #0FFh,&P3DIR                    ; P3 output: wyjście
19        clr.b  &P3OUT                          ;
20        mov.b  #0FFh,&P2DIR                    ; P2 output: wyjście
21        clr.b  &P2OUT                          ;
22        mov.b  #0FFh,&P1DIR                    ; P1 output: wyjście
23        clr.b  &P1OUT
24
25 loop    bis.w  #LPM3+GIE,SR                   ; tryb LPM3, odblokowanie przerwań
26        nop                                    ; konieczne dla debugera
27        bic.b  #001h,&P1OUT                    ; ustaw bit P1.0: włacz diodę LED
28        push  #2000h                           ; czas opóznienia na stos
29 wait     dec.w  0(SP)                         ; zmniejsz czas opóźnienia
30        jnz   wait                             ; czas opóźnienia upłynął ?
31        incd.w SP                              ; usuń ze stosu
32        bis.b  #001h,&P1OUT                    ; skasuj bit P1.0: wyłacz diodę LED
33        jmp   loop
34
35 ;-------------------------------------------------------------------------
36 WDT_ISR                                        ; budzenie ze stanu LPM3
37        bic.w  #LPM3,0(SP)                     ;
38        reti                                   ;
39                                               ;
40 ;-------------------------------------------------------------------------
41                                               ; wektory przerań MSP430x4xx
42        ORG   0FFFEh                           ; MSP430 RESET wektor
43        DW   RESET                             ;
44        ORG   0FFF4h                           ; WDT wektor
45        DW   WDT_ISR                           ;
46        END
```

W wierszu #26 wywołuje się makrodefinicję odblokowania przerwań. Funkcja **__low_power_mode_3()** pochodzi z biblioteki środowiska a jej definicja znajduje się w zbiorze ***intrinsic.h***. Zbiór ten jest automatycznie włączany gdy zostanie użyty zbiór definicji rejestrów procesora **include <msp430x14x.h>**.

W wierszu # 31 znajduje się instrukcja **for (int i =0; i<0x2000; i++); ...** Jest to operacja opóźnienia z definicją lokalną zmiennej int i. To jest instrukcja języka C++. I rzeczywiście. Kompilator IAR C jest kompilatorem obiektowym języka C++. Są oczywiście ograniczenia wynikające z możliwości procesora MSP430. To ograniczenie to obiekty wirtualne bez których można się bardzo dobrze obejść. Kompilator IAR jest obiektowy, ale nie ma obowiązku używania go w pełnej, najbardziej wyrafinowanej obiektowej postaci. Język C jest z pewnymi drobnymi niuansami podzbiorem języka C++. Daje to szansę rozwoju programistom, którzy chcą się nauczyć języka C++.

Zapis **#pragma vector= WDT_VECTOR** w wierszu # 36 jest nowością w ostatniej wersji kompilatora IAR i sygnalizuje zdefiniowanie funkcji obsługi przerwania związanej z określonym położeniem wektora w pamięci. Programy napisane dla wcześniejszych wersji kompilatora IAR muszą być konwertowane do takiej właśnie postaci.

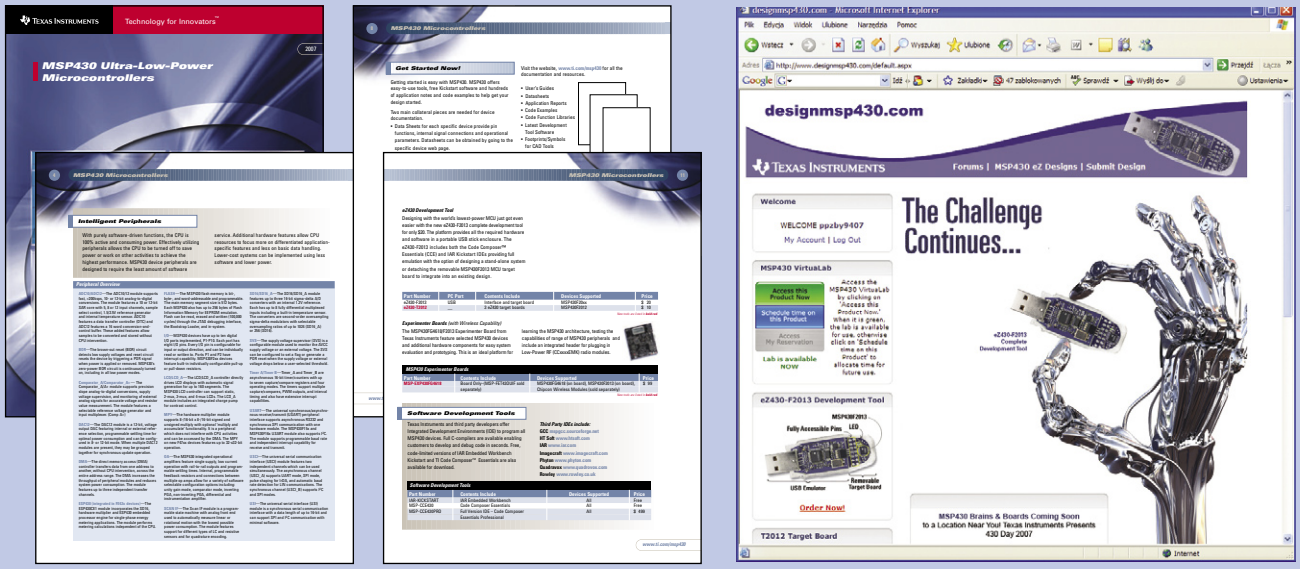
Informacje katalogowe oraz noty aplikacyjne dotyczące mikrokontrolerów z rodziny MSP430 są dostępne pod adresem:

http://www.ti.com/msp430

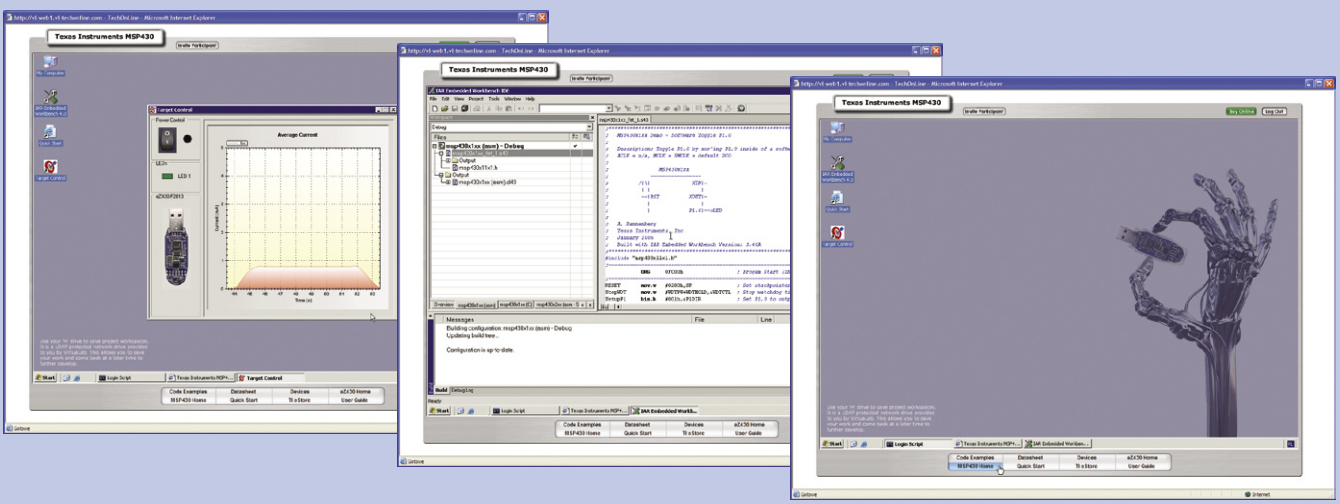
W EP11/2007 opublikujemy płytę CD–ROM z programami narzędziowymi, przykładami i dokumentacjami związanymi z mikrokontrolerami MSP430.

Czytelnicy zainteresowani poznaniem mikrokontrolerów MSP430 mogą obejrzeć projekty konkursowe, przygotowane przez inżynierów z całego świata na konkurs *Design MSP430 Challenge*, który odbył się w zeszłym roku. Opisy projektów oraz oprogramowanie przygotowane do nich są dostępne pod adresem:

http://www.designmsp430.com/



Czytelnicy, którzy chcą podjąć beznakładowo samodzielne próby z mikrokontrolerami MSP430 mogą skorzystać z internetowego laboratorium udostępnionego przez firmę Texas Instruments elektronikom. Dzięki niemu mamy dostęp do środowiska programistycznego firmy IAR oraz prostego symulatora zestawu uruchomieniowego eZ430.



Funkcja **`__low_power_mode_off_on_exit()`** jest zdefiniowana w tym samym zbiorze nagłówkowym ***intrinsic.h*** i służy do budzenia i przejścia procesora z trybu LPM3 do trybu aktywnego.

Asembler procesora MSP430 jest przejrzysty i łatwy do przyswojenia. Zapis **`bis.w`** oznacza ustawienie bitu (*bit set*) w słowie (*Word*). Podobnie **`clr.b`** oznacza zerowanie wybranych bitów w bajcie. Zapis **`#0A00h`** oznacza stała zapisaną heksadecymalnie. Konstrukcja **`&IE1`** oznacza 16–bitowy absolutny adres rejestru IE1.

Zinterpretujmy przykładowo instrukcję **`bic.w #LPM3,0(SP)`**. Oznacza ona wyzerowanie bitu LPM3 na stosie wskazywanym przez SP z przesunięciem 0 (czyli bezpośrednio na szczycie stosu). W procesorze MSP430 podczas wywoływania podprogramu obsługi przerwania zapisuje się rejestr statusowy i adres powrotu na stosie.

Bity sterujące usypianiem i budzeniem procesora znajdują się w rejestrze statusowym. Aby obudzić procesor i wyjść ze stanu LPM3 należy wyzerować bity maski LPM3. Zwróćmy jeszcze uwagę na instrukcję **`#include „msp430x14x.h”`** w wierszu #6. Poleca ona włączyć zbiór definicji rejestrów procesora MSP. Jest to ten sam zbiór i ten sam zapis jak w języku C.

Jacek Majewski
Mariusz Kaczor
Krzysztof Kardach