MSP430: mikrokontrolery, które (prawie) nie pobierają prądu, część 3

Środowisko programistyczne

KONKURS str. 8

Kontynuujemy prezentację narzędzi programistycznych, które wykorzystamy do przygotowania pierwszych projektów dla mikrokontrolerów z rodziny MSP430.

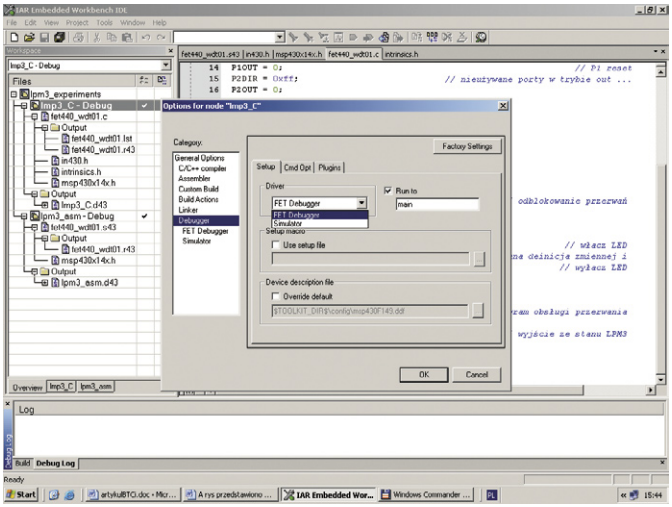
Przyjrzyjmy się dokładniej czynnościom związanym z uruchomieniem projektu. Na stronie *msp430.ep.com.pl* umieszczono kompletny projekt wykorzystany w artykule. Po zainstalowaniu oprogramowania IAR można uruchomić projekt w systemie Windows dwa razy klikając na zbiorze `lpm3_experiments.eww`. W wyniku pojawi się okno jak na **rys. 6**.

W oknie *Files* można zobaczyć drzewo projektu. Stan jak na rysunku pojawi się gdy rozwiniemy wszystkie punkty drzewa projektu oznaczone jako +. W zasadzie jedynym zadaniem czekającym użytkownika jest decyzja, czy chce załadować program do mikrokontrolera czy też uruchomić go przy użyciu symulatora. Wygląd ekranu podczas rzeczywistego uruchomienia programu przedstawiono już na **rys. 7**. Zobaczmy teraz jak przeprowadzić uruchomienie programu z użyciem symulatora.

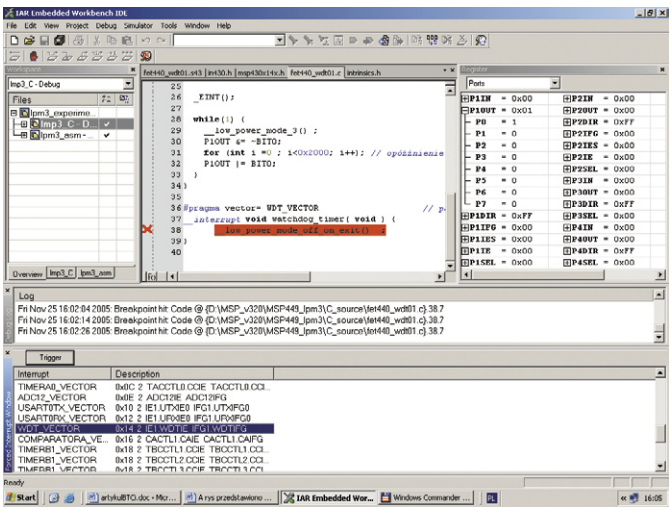
Jeśli po załadowaniu oprogramowania ustawimy pułapkę (*Breakpoint*) we wnętrzu podprogramu obsługi przerwania i uruchomimy program (np. poleceniem *Go*) to w wyniku wykonania instrukcji LPM3 program zatrzyma się czekając na przerwanie od układu WDT. Przerwanie to można zasymulować naciskając widoczny na ilustracji przycisk *Trigger*. Aby wywołać okno wyboru i wywoływania przerwań należy użyć opcji *Forced Interrupts* w menu *Simulator*. Każde przerwanie można zasymulować asynchronicznie, przez przyciśnięcie przycisku *Trigger* lub napisać makrodefinicję i zaprogramować moment pojawienia się przerwania łącznie z zaprogramowaniem losowego pojawienia się przerwania (menu *Simulator* / *Interrupt Setup*).

Inne możliwości

O tym, że można pisać programy C++ dla mikrokontrolera MSP430 już wspomniano. Wystarczy tyko zmienić bibliotekę *runtime*. Zamiast CLIB należy użyć DLIB. W związku z programowaniem obiektowym oraz programowaniem w języku C można uaktywnić opcje o nazwie *MISRA C rules*. Opcja ta nie jest dostępna w ograniczonej wersji kompilatora. W ogólnym zarysie opcja ta generuje ostrzeżenia i błędy o przekroczeniu zasad programowania w języku C. Przykładowo nie powinno się wykonywać logicznych operacji na liczbach *integer* ze zna-



Rys. 6. Wygląd okien środowiska *KickStart* pobranego ze strony TI (w chwili decyzji o sposobie uruchamiania przykładu; poprzez JTAG na płytce czy w symulatorze)



Rys. 7. Wygląd okien środowiska po załadowaniu przykładowego projektu

Mikrokontrolery MSP430 w skrócie (na przykładzie MSP430F4xx):

- Napięcie zasilania od 1,8 V do 3,6 V
- Ultra niski pobór mocy;
  – W trybie aktywnym 280 μA dla 1 MHz, 2,2 V
  – W trybie „Standby" 1,1 μA
  – W trybie „Off" (podtrzymanie pamięci RAM): 0,1 μA
- Pięć trybów uśpienia
- Wybudzanie z trybów uśpienia w 6 μs
- 16–botowa architektura RISC, cykl rozkazowy 125 ns
- 12–bitowy przetwornik A/C z wewnętrznym napięciem odniesienia, układem Sample&Hold i autoskanowaniem
- 16–bitowy timer_B z trzema+ lub siedmioma++ rejestrami przechwytu/porównania
- Komparator w strukturze mikrokntrolera
- Interfejs komunikacji szeregowej (USART)
  Programowo wybierany tryb asynchroniczny UART lub synchroniczny SPI:
  – Dwa USART–y (USART0 i USART1) – w układach MSP430x44x
  – Jeden USART (USART0) – w układach MSP430x43x
- Nadzór wolnych zmian napięcia zasilania (Brownout Detector)
- Układ nadzoru napięcia zasilania z programowanym wykrywanym poziomem
- Programowanie szeregowe w układzie, zbędne jest zewnętrzne napięcie programujące, programowany bit bezpieczeństwa zawartości pamięci programu
- Wbudowany sterownik LCD maks. 160 segmentów
- Rodzina zawiera
  – MSP430F435:
    16 kB + 256 B Pamięć FLASH
    512 B RAM
  – MSP430F436:
    24 kB + 256 B Pamięć FLASH
    1 kB RAM
  – MSP430F437:
    32 kB + 256 B Pamięć FLASH
    1 kB RAM
  – MSP430F447:
    32 kB + 256 B Pamięć FLASH
    1 kB RAM
  – MSP430F448:
    48 kB + 256 B Pamięć FLASH
    1 kB RAM
  – MSP430F449:
    60 kB + 256 B Pamięć FLASH
    1 kB RAM

+ mikrokontrolery 'F435, 'F436, i 'F437
++ mikrokontrolery 'F447, 'F448 i 'F449

Opis
Rodzina mikrokontrolerów o ultraniskim poborze mocy MSP430 firmy Texas Instruments składa się z wielu układów charakteryzujących się zróżnicowanymi układami peryferyjnymi. Mikrokontrolery wyposażono w kilka trybów pracy energooszczędnej, ich architekturę zoptymalizowano pod kątem wydłużenia czasu życia baterii zasilającej w urządzeniach przenośnych. Układy zawierają szybką 16–bitową jednostkę CPU o architekturze RISC, 16–bitowe rejestry, generatory stałych warunkujące maksymalną efektywność kodu programu. Sterowany cyfrowo oscylator (DCO) umożliwia wybudzanie mikrokontrolera z energooszczędnych stanów uśpienia w czasie krótszym niż 6 μs.
Mikrokontrolery grup MSP430x43x i MSP430x44x dysponują dwoma wbudowanymi 16–bitowymi timerami, szybkim, 12–bitowym przetwornikiem A/C, jednym lub dwoma synchronicznymi lub asynchronicznymi interfejsami szeregowymi (USART), konfigurowalnymi liniami we/wy i sterownikiem pola ekspozycyjnego LCD do 160 segmentów.
Typowe zastosowania obejmują systemy sensorów mogących odbierać sygnały analogowe, przetwarzać je na wartości cyfrowe, przetwarzać cyfrowo zapisane dane i odsyłać do nadrzędnych systemów lub po przetworzeniu eksponować na polu LCD. Timery czynią te mikrokontrolerów idealnymi do przemysłowych zastosowań sterujących, jak zliczanie drgań, sterowanie sinikami, w miernikach energii, miernikach przenośnych, itp. Sprzętowa jednostka mnożąca rozszerza wydajność i pozwala na szerokie rozwiązania współpracy z kompatybilnymi sprzętowo i programowo układami przetwarzania.

kiem. Nie powinno być w programie kodu, którego nie można wykonać. Nie należy używać liczb oktalnych zaczynających się od wartości 0.itd. Nie przestrzeganie tych zasad prowadzi do kłopotliwych błędów. Ustawiona opcja MISRA spowoduje wywołanie odpowiedniego ostrzeżenia w trakcie kompilacji.

Jacek Majewski
Mariusz Kaczor
Krzysztof Kardach

W mikrokontrolerach MSP430 trzeba nauczyć się operowania bitami. Ta umiejętność jest potrzebna we wszystkich współczesnych mikrokontrolerach, ponieważ nie mają one operacji manipulowania bitami, jakia istnieje w mikrokontrolerach typu '51.
Spróbujmy pokazać jak wykonuje się operacje ustawiania i kasowania bitów w mikrokontrolerach MSP. Załóżmy że chcemy manipulowć bitami 1 i 5 portu P1OUT. Jeśli wiemy dokładnie jakie bity mamy ustawić a jakie wyzerować to możemy zapisać to jako:

```
P1OUT = 0x12
```

Zapis ten spowoduje ustawienie bitu1 i bitu 5 a pozotałe bity zostaną wyzerowane. Jeśli zamierzamy ustawić tylko bity 1 i 5 a pozostałe pozostawić bez zmiany to można to zrealizować w sposób następujący P1OUT |= 0x12. W przypadku gdy chcemy wyzerować bity 1 i 5 a pozostałe pozostawić bez zmian to można to zapisać w postaci

```
P1OUT &= ~0x12
```

Jeśli dodamy definicje bitów w postaci

```
#define BIT0 (0x0001)
#define BIT1 (0x0002)
#define BIT2 (0x0004) ...
```

To zapisy przedstawione powyżej staną się bardziej czytelne. Definicje bitów znajdują się w zbiorze msp430x14x.h
Zapis ustawienia bitów może teraz wyglądać

```
P1OUT |= BIT1 + BIT5
```

lub

```
P1OUT |= BIT1 | BIT5
```

Podobnie aby wyzerować bity trzeba napisać

```
P1OUT &= ~ (BIT1 + BIT5)
```

lub

```
P1OUT |= ~ (BIT1 | BIT5)
```

Powyżej przedstawione zapisy wymagają uważnego przeanalizowania bo będą często używane w programach dla MSP430. Aby wyczerpać zagadnienie ustawiania i zerowania bitów warto wspomnieć, że gdyby wprowadzić nieco inna definicje bitów

```
#define bit0 0
#define bit1 1
#define bit2 2...
```

to operacje ustawiania bitów można zapisać jako

```
P1OUT |= ( (1<<bit1) + (1<<bit5) )
```

Podobnie operacja zerowania przybierze wtedy postać

```
P1OUT &= ~ ( (1<<bit1) + (1<<bit5) )
```

Trzeba podkreślić że mimo iż te notacje wydają się strasznie skomplikowane to nie zajmują po skompilowaniu więcej kodu niż te które przedstawiono poprzednia. Dzieje się tak dlatego, że rolę tłumaczenia na postać wynikową wykonuje preprocesor kompilatora a nie kompilator.
W mikrokontrolerach AVR w kompilatorze GCC zastosowano jeszcze inne rozwiązanie. Zdefiniowano funkcje biblioteczne *cbi* i *sbi* odpowiadające za zerowanie (*clear*) lub ustawianie wskazanych bitów (*set*). Dla tych którzy preferują taki styl programowania pozostaje napisanie własnych funkcji dla mikrokontrolerów MSP430.