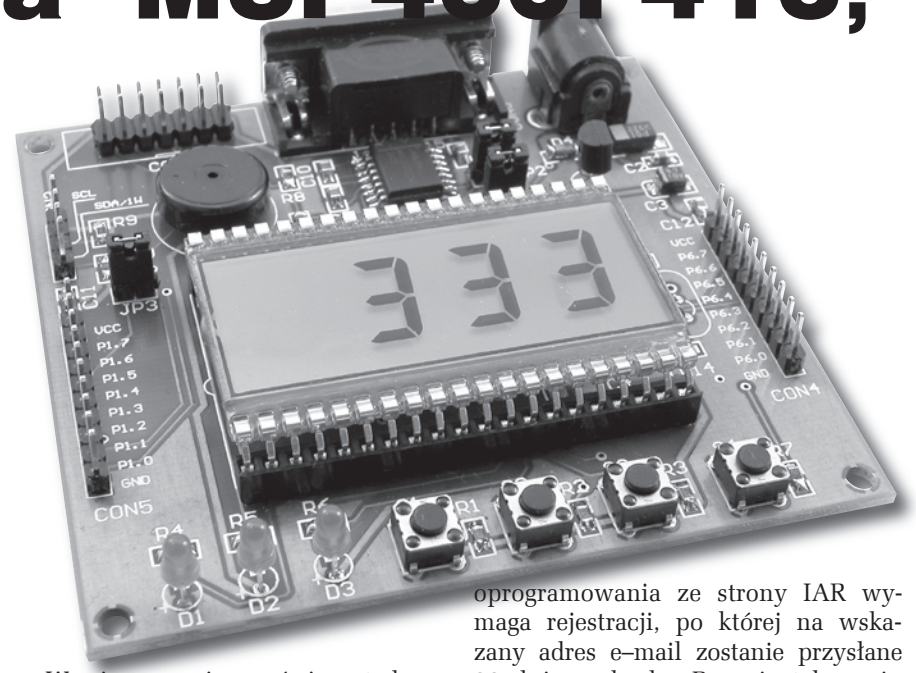
Zestaw startowy dla procesora MSP430F413, część 2

AVT–920

Z dostępnych na rynku mikrokontrolerów trudno jest jednoznacznie wybrać najlepszy. Każdy ma jakieś swoje zalety i wady. Nawet popularność takiego, czy innego układu nie przesądza o jego walorach użytkowych. Najlepiej sprawdzić je samodzielnie za pomocą odpowiedniego zestawu uruchomieniowego. Taki zestaw dla mikrokontrolerów firmy Texas Instruments prezentujemy poniżej.

Rekomendacje:

zestaw uruchomieniowy polecamy wszystkim mikroprocesorowcom planującym poznanie nowych mikrokontrolerów rodziny MSP430.



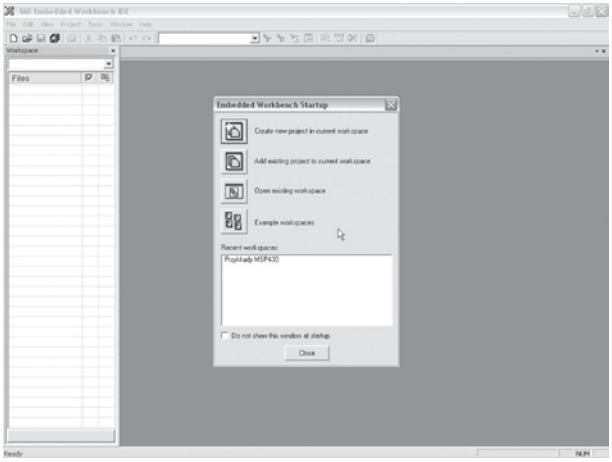
W pierwszej części artykułu przedstawiono opis wykonania płytki zestawu uruchomieniowego dla mikrokontrolerów MSP430F413. Czytelnicy zainteresowani tematem zapewne zdążyli już wykonać część sprzętową, pora zatem na omówienie oprogramowania.

Oprogramowanie

Do tworzenia oprogramowania na mikrokontrolery MSP430 zostanie wykorzystany pakiet firmy IAR. Zawiera on kompilator języka C, a także emulator umożliwiający zarówno symulację pracy procesora, jak również kontrolę jego pracy w układzie rzeczywistym za pośrednictwem interfejsu JTAG.

Specjalną, darmową wersję (z ograniczeniem generowanego kodu), która nosi nazwę Kickstart, można pobrać ze strony TI *http://www–s.ti.com/sc/techzip/slac050.zip*, natomiast ze strony IAR *http://www.iar.com/Products?name=EW430* można pobrać ograniczoną czasowo (30 dni) wersję testową, która nie posiada ograniczenia generowanego kodu. Pobranie oprogramowania ze strony IAR wymaga rejestracji, po której na wskazany adres e–mail zostanie przysłane 30–dniowe hasło. Po zainstalowaniu i uruchomieniu programu na ekranie pojawi się okno główne oraz okno pomocnicze (**rys. 6**), które umożliwia utworzenie nowego projektu w bieżącym obszarze roboczym, włączenie istniejącego projektu do obszaru roboczego, utworzenie nowego obszaru lub otwarcie przykładowych obszarów roboczych. Poniżej zostanie przedstawiony sposób tworzenia obszaru roboczego, co powinno ułatwić Czytelnikowi tworzenie własnych projektów.

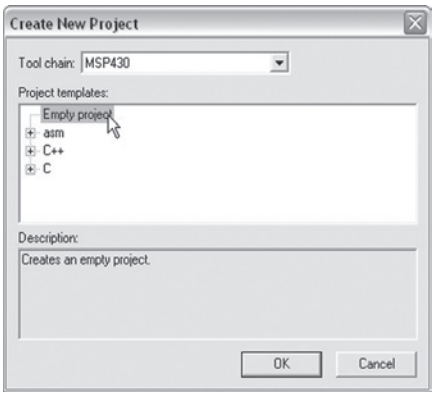
Wybieramy zatem pierwszą ikonę – „Create New project In current workspace", a w nowo otwartym oknie wybieramy opcję *Empty*



Rys. 6. Okno powitalne kompilatora

PODSTAWOWE PARAMETRY

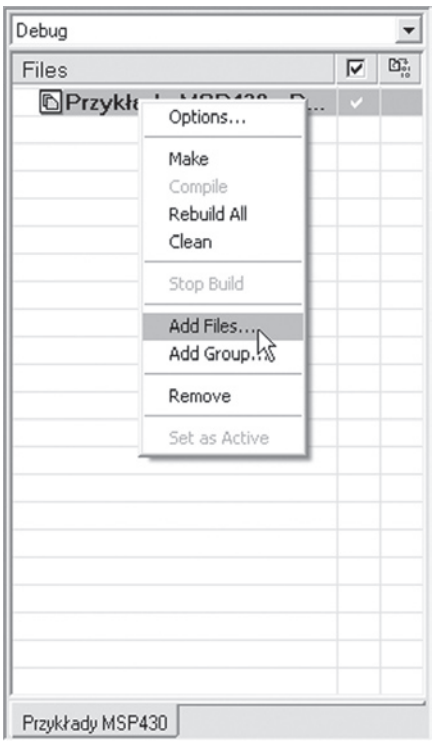
- Płytka o wymiarach 86 x 82 mm
- Zasilanie 5...12 V DC
- Peryferia dostępne dla użytkownika: wyświetlacz LCD 3 1/2 cyfry, interfejs RS232, trzy diody świecące, trzy przyciski, brzęczyk
- Złącza: porty P1 i P6, interfejs I²C i 1Wire, interfejs JTAG służący do programowania mikrokontrolera



Rys. 7. Okno tworzenia nowego projektu

Rys. 8. Wybór rodzaju tworzonego pliku

Project (**rys. 7**), następnie należy podać nazwę projektu oraz miejsce, w którym zostanie on zapisany na dysku. Dalej w menu „File" wybieramy *Save Workspace*, również wskazując miejsce, gdzie obszar roboczy ma zostać zapisany. W ten sposób został utworzony projekt, w którym można umieścić właściwe pliki zawierające kod źródłowy programu.



Rys. 9. Włączanie plików do projektu

List. 1. Program mrugania diodą D1

```
#include  <msp430x41x.h>      //rejestry MSP430F413
#define D1  0x04              //dioda D1

void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;   //zatrzymaj watchdog timer
  P5DIR |= D1;                //Port P5.2 jako wyjscie
  P5OUT|=D1;                  //dioda LED1 wlaczona
  for (;;)
  {
    unsigned int i;

    P5OUT ^= D1;              //Zmienia stan P5.2 na przeciwny
    i = 50000;                //pauza
    while (i-- != 0);
  }
}
```

List. 2. Program generowania sygnału 4096 Hz na wyjściu P1.5

```
//-------------------------------------------------------------//
#include <msp430x41x.h>
#define BUZ  0x20             //port P1.5
void main (void)
{ unsigned int i;
  WDTCTL = WDTPW + WDTHOLD;   // watchdog timer zatrzymany
  FLL_CTL0 = XCAP18PF;        // kondensatory wewnetrzne
  FLL_CTL1|=FLL_DIV_8;        // P1.5=ACLK/8 =4096Hz
  P1DIR |= 0x20;              // P1.5 jako wyjscie
  P1SEL |= 0x20;              // P1.5  wyjscie ACLK

  for (;;)
  {
   P1SEL ^= BUZ ;     // przelacza P1.5  pomiędzy ACLK i P1OUT
   i = 50000;                 // Pauza
   while (i-- >0);
  }
}
//-------------------------------------------------------------//
```

W celu utworzenia pliku źródłowego należy w menu „File" wybrać *New*, a w nowo otwartym oknie (**rys. 8**) *Source/Text*. W oknie edycyjnym zostanie otwarty pusty dokument, który należy zapisać na dysku (najlepiej w katalogu, w którym znajduje się plik projektu) nadając mu nazwę na przykład *dioda.c* (z rozszerzeniem .c). Plik ten będzie zawierał tworzony program i dlatego należy włączyć go do projektu poprzez kliknięcie prawym klawiszem myszki na nazwie utworzonego projektu („Przykłady MSP430"), w otwartym menu (**rys. 9**) należy wybrać *Add files* i wskazać ścieżkę dostępu do pliku *dioda.c*. Dalej należy kliknąć dwa razy na ten plik, aby został otwarty w oknie edycyjnym.

Jako przykład stworzymy program, który będzie naprzemiennie zapalał i gasił diodę D1. Program ten jest przedstawiony na **list. 1.** Wpisujemy go w oknie edycyjnym. Przed przejściem do kompilacji należy ustawić parametry projektu poprzez jego zaznaczenie (kliknięcie na jego nazwie), a następnie w menu „Project" wybranie *Options* (szczegóły poszczególnych kroków przedstawiono na **rys. 10**). W pierwszej kolejności należy ustalić typ procesora wybierając go z listy okienka numer 3 (MSP430F413), następnie w okienku 4 ustalamy rodzaj generowanych plików. W oknie 5 wybieramy, czy program będzie testowany poprzez symulator programowy (opcja *Simulator*,) czy przez sprzętowy *FET Debugger*. Dla zestawu startowego współpracującego z interfejsem JTAG należy wybrać opcję *FET Debugger*, co pozwoli na kontrolę procesora pracującego w układzie rzeczywistym. W takim przypadku dla ostatecznej konfiguracji należy jeszcze (w oknie 6) wybrać port, do którego jest podłączony interfejs JTAG, zatwierdzić wszystkie zmiany i powrócić do głównego okna kompilatora.

Po skonfigurowaniu projektu można przejść do kompilacji i testowania napisanego programu. W tym celu należy zaznaczyć w projekcie plik o nazwie *dioda.c*, a następnie w menu

List. 3. Przykład zastosowania modułu FLL

```
//-------------------------------------------------------------------------//
#include „msp430x41x.h"

void main (void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  SCFI0 |= FN_2;            // x2 DCO
  FLL_CTL0 |= XCAP18PF;     // kondensatory wewnetrzne
  SCFQCTL = 74;             // (74+1) x 32768 = 2.45Mhz
  P1DIR = 0x22;             // P1.1 i P1.5 jako wyjscia
  P1SEL = 0x22;             // P1.1 i P1.5 wyjscie MCLK i ACLK

  while (1);                //petla nieskonczona
}
//-------------------------------------------------------------------------//
```

List. 4. Przykład odczytu klawiatury i sterowania diodami LED

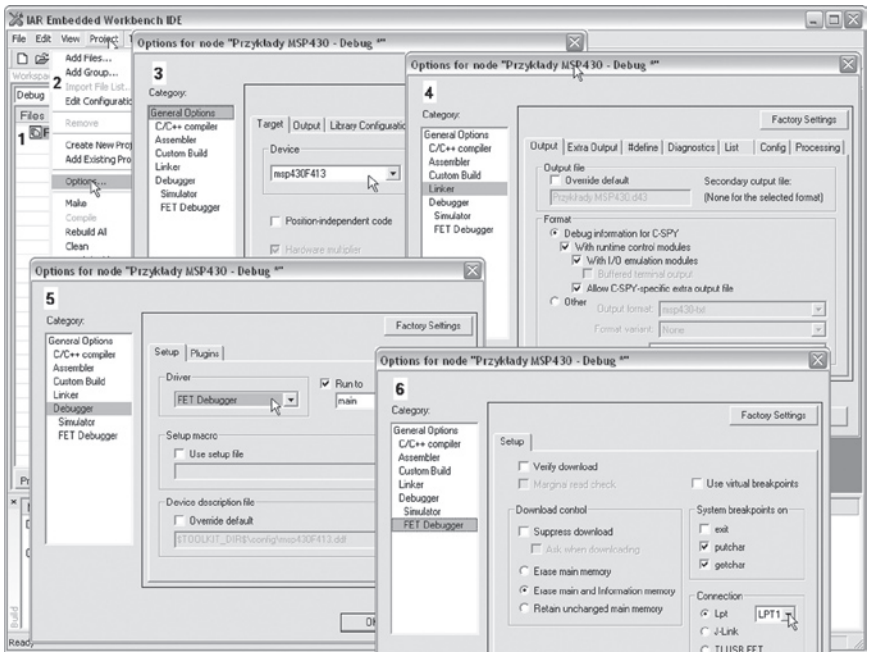
```
//*********************************************************************//
#include  <msp430x41x.h>

#define D1    0x04              //port D1
#define D2    0x08              //port D2
#define D3    0x10              //port D3
#define S1    0x20              //port S1
#define S2    0x40              //port S2
#define S3    0x80              //port S3
#define S1IN  (P5IN&S1)         //wejscie S1
#define S2IN  (P5IN&S2)         //wejscie S2
#define S3IN  (P5IN&S3)         //wejscie S3

void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;  //zatrzymaj watchdog timer
  P5DIR |= D1;                //Port P5.2 jako wyjscie
  P5OUT |= D1;                //dioda D1 wylaczona
  P5DIR |= D2;                //Port P5.3 jako wyjscie
  P5OUT |= D2;                //dioda D1 wylaczona
  P5DIR |= D3;                //Port P.4 jako wyjscie
  P5OUT |= D3;                //dioda D1 wylaczona
  P5DIR &=~S1;                //port P5.5 jako wejscie
  P5DIR &=~S2;                //port P5.6 jako wejscie
  P5DIR &=~S3;                //port P5.7 jako wejscie
  for (;;)
  {
    unsigned int i;
   if (S1IN==0)               //jesli nacisniety przycisk S1
       {for (i=0;i<5000;i++);  //czekaj 6ms
          if (S1IN==0)         //sprawdz ponownie czy nacisniety S1
             {P5OUT^=D1;       //jeli tak to zmien stan LED1
              while (S1IN==0); //czekaj na zwolnienie S1
              for (i=0;i<5000;i++);     //pauza 6ms
             }
       }

    if (S2IN==0)               //jesli nacisniety przycisk S2
       {for (i=0;i<5000;i++);  //czekaj 6ms
          if (S2IN==0)         //sprawdz ponownie czy nacisniety S2
             {P5OUT^=D2;       //jeli tak to zmien stan LED2
              while (S2IN==0); //czekaj na zwolnienie S2
              for (i=0;i<5000;i++);     //pauza 6ms
             }
       }
    if (S3IN==0)               //jesli nacisniety przycisk S2
       {for (i=0;i<5000;i++);  //czekaj 6ms
          if (S3IN==0)         //sprawdz ponownie czy nacisniety S2
             {P5OUT^=D3;       //jeli tak to zmien stan LED2
              while (S3IN==0); //czekaj na zwolnienie S2
              for (i=0;i<5000;i++);     //pauza 6ms
             }
       }
  }

}
//*********************************************************************//
```



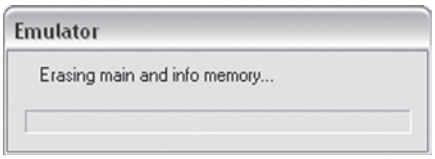
Rys. 10. Ustawianie parametrów projektu

„Project" wybrać *Compile*, dla skompilowania całego projektu w tym samym menu należy wybrać *Make*. Te same czynności można wykonać naciskając odpowiednie ikony na belce przycisków w głównym oknie kompilatora. Po kompilacji w oknie „Messages" zostanie pokazany przebieg kompilacji oraz ewentualne błędy. Po prawidłowym skompilowaniu można przejść do sprawdzenia działania programu programując procesor (wcześniej należy złącze interfejsu JTAG dołączyć do złącza CON2 zestawu startowego). Programowanie rozpoczyna się wybraniem opcji *Debug* w menu „Project" lub naciśnięciem przycisku na głównej belce. Podczas programowania będzie widoczne okno postępu (**rys. 11**), a po prawidłowym zaprogramowaniu pamięci procesora zostanie otwarte okno Debugera (**rys. 12**). W oknie tym będzie można uruchamiać, a także testować program. Widoczny jest tu między innymi kod pisanego programu w języku C, instrukcje w asemblerze oraz komunikaty Debugera. Możliwe jest także podglądanie rejestrów, symulacja terminala lub wyświetlacza LCD. Dodatkowe funkcje umożliwiające wykonanie powyższych zadań są dostępne w menu „View".

Polecenia umożliwiające uruchomienie programu są dostępne w menu „Debug" (również w postaci przycisków na belce narzędziowej – kursor myszki na rys. 12). Aby uruchomić program zapisany w pamięci procesora należy wybrać polecenie *Go*. Program będzie wykonywany z pełną prędkością, dioda D1 będzie zapalana i gaszona. Polecenie *Break* spowoduje zatrzymanie procesora, a *Reset* powrót do początku programu. Przydatnym poleceniem, szczególnie przy testowaniu poprawności działania programów jest tryb pracy krokowej *Step Over*. Dzięki niemu można śledzić kolejne kroki programu w „zwolnionym tempie". Jeśli testowany program pracuje prawidłowo, to poleceniem *Stop Debugging* można zakończyć nadzorowaną pracę procesora, okno Debugera zostanie zamknięte i nastąpi powrót do okna kompilatora. Jeśli przed wyjściem z trybu Debugera, procesor został uruchomiony z pełną prędkością, to po powrocie do okna kompilatora jego praca nie zostanie zatrzymana. Także złącze interfejsu JTAG może być przez cały czas podłączone, ponieważ jest on dołączany automatycznie do procesora na czas programowania i kontrolowania poprzez Debuger (poprzez układ zawarty w JTAG–u), a w trybie kompilacji jest odłączony umożliwiając normalną pracę procesora.

Przykładowe programy

Procesory rodziny MSP430 są w Polsce stosunkowo mało popularne, niewiele jest więc materiałów pomocniczych umożliwiających ich poznanie. Poniżej zostaną przedsta-



Rys. 11. Okno programowania procesora

wione proste procedury pozwalające na „ożywienie" zestawu startowego. Wszystkie przykłady należy wprowadzić analogicznie jak w przypadku list. 1, tworząc osobne projekty w osobnych obszarach roboczych (Workspace) lub dodając kolejne projekty do tego samego obszaru.

Jako pierwszy przykład zostanie omówiona procedura obsługi brzęczyka. Program realizujący tę funkcję jest przedstawiony na **list. 2**. Brzęczyk zostanie wysterowany przebiegiem o częstotliwości 4096 Hz poprzez podział częstotliwości generatora i skierowanie go na wyjście portu P1.5. Cała procedura sprowadza się do odpowiedniej konfiguracji rejestrów procesora (zworka JP3 musi zostać zwarta). W pętli *for* jest wykonywana dodatkowo cykliczna zmiana portu P1.5 pomiędzy trybem wyjścia sygnału ACLK, a rejestrem wyjściowym portu (P1OUT), przez co generowany dźwięk jest impulsowy.

W przykładzie drugim (**list. 3**) przedstawiono wykorzystanie modułu FLL+, dzięki któremu z przebiegu o częstotliwości 32,768 kHz jest wytwarzany przebieg o częstotliwości 2,45 MHz. Może on służyć do taktowania procesora. Częstotliwość ta jest uzyskiwana w wyniku mnożenia podstawowej częstotliwości przez 75 (75x32768). Dodatkowo, oba przebiegi zostały skierowane na zewnątrz układu: przebieg 2,45 MHz do portu P1.1, przebieg 32,768 kHz do portu P1.5. Dzięki temu mamy możliwość sprawdzenia częstotliwości za pomocą miernika oraz zorientowania się, czy przebiegi w ogóle są generowane.

Kolejny przykład (**list. 4**) pokazuje, jak obsłużyć klawiaturę oraz jak wysterować diody świecące. Ponieważ przyciski i diody są dołączone do tego samego portu, wyprowadzenia P5.2...P5.4 są skonfigurowane jako wyjścia, natomiast wyprowadzenia P5.5...P5.7 jako wejścia. Działanie programu polega na zmianie stanu diody po każdorazowym naciśnięciu odpowiadającego przycisku: D1 dla S1, D2 dla S2, D3 dla S3.

List. 5. Program obsługi wyświetlacza LCD

```
//*******************************************************************************//
#include „msp430x41x.h"
// deklaracja zmiennych //
unsigned int val=0;
unsigned int h;
unsigned int i;
unsigned int dig_pntr;
unsigned int k;
char *LCD = LCDMEM;

// definicja tablicy  //
char digit[40] = {
0x11, 0x11,  // "0"    LCD segment a+b & c+d = mlodsze dwa bajty
0x11, 0x00,  // "0"    LCD segment e+f & g+h = starsze dwa bajty
0x10, 0x01,  // "1"
0x00, 0x00,  // "1"
0x11, 0x10,  // "2"
0x01, 0x01,  // "2"
0x11, 0x11,  // "3"
0x00, 0x01,  // "3"
0x10, 0x01,  // "4"
0x10, 0x01,  // "4"
0x01, 0x11,  // "5"
0x10, 0x01,  // "5"
0x01, 0x11,  // "6"
0x11, 0x01,  // "6"
0x11, 0x01,  // "7"
0x00, 0x00,  // "7"
0x11, 0x11,  // "8"
0x11, 0x01,  // "8"
0x11, 0x11,  // "9"
0x10, 0x01,  // "9"
};

void clear_lcd ();                    //zerowanie LCD
void put_lcd (unsigned int value);  //wyswietla wartosc 0...1999

//poczatek programu
void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;           // stop watchdog timer
  FLL_CTL0 = XCAP18PF;                // wewnetrzne kondensatory
  // inicjalizacja sterownika LCD w trybie statycznym //
  LCDCTL = 0x65;                      //statyczny LCD, segmenty = 0 - 23
  BTCTL  = BTFRFQ1+BTFRFQ0;           // BTCTL ;set fLCD = ACLK / 256 BTFRFQ1
  P2DIR = 0xFF;                       // P2 wyjscie
  P3DIR = 0xFF;                       // P3 wyjscie
  P4DIR = 0xFF;                       // P4 wyjscie
  P5DIR = 0x04;                       // P5<1:0> wyjscie

  clear_lcd ();                       //kasuj LCD
  put_lcd (val);                      //wyswietl wartosc poczatkowa

 //------------glowna petla--------//
  while (1)
     {
      P5OUT ^= 0x04;                  //Zmienia stan P5.2 na przeciwny
      k = 30000;                      //pauza
      while (--k>0);
      if (++val>1999) val=0;
      put_lcd (val);                  //wyswietl wartosc na LCD
     }
}

// wylacza wszystkie segmenty LCD//
void clear_lcd ()
{char i;
char *LCD = LCDMEM;
  for (i=0; i<12; i++)
  {
    LCD[i] = 0x00;
  }
}

//wyswietla podana liczbe z zakresu 0...1999
void put_lcd (unsigned int value)
{
char *LCD = LCDMEM;
// wyswietla 3 mlodsze cyfry //
  for (h=0; h<3; h++)                 //petla wyslania 3 mlodszych cyfr do wyswie-
tlacza
  {
    dig_pntr = 4* (value%10);         //ustawienie adresu odczytu z tablicy digit
    for (i=0; i<4; i++)               //petla zapisu 4bajtow definiujacych cyfre
    {
      LCD[i] = digit[dig_pntr++];     //zapis do pamieci wyswietlacza cyfry
    }                                 //zdefiniowanej w tablicy digit
    value /= 10;                      //dzielenie aby wyswietlic kolejna cyfre licz-
by wejsciowej
    LCD += 4;                         //ustawienia adresu w pamieci LCD dla nastep-
nej cyfry
  }

  // jesli liczba > 999, to wyswietla 1 na czwartej cyfrze LCD//
  if (value == 1)
  {
    LCDM12 |= 0x10;                   //wyswietl 1
  }

 }
//*******************************************************************************//
```

Kolejnym przykładem są procedury umożliwiające obsługę wyświetlacza LCD. Ze względu na swoją złożoność zostaną omówione dokładniej niż dotychczasowe.

Na **list. 5** przedstawiono program, który umożliwia wyświetlanie na wyświetlaczu liczby z zakresu 0...1999. Na początku programu inicjowany jest procesor oraz sterownik wyświetlacza. Sterownik zostaje skonfigurowany do

List. 6. Procedury obsługi portu szeregowego

```
//***************************************************************************//
#define RXD   0x02           // RXD  P1.1
#define TXD   0x01           // TXD  P1.0
#define Bitime_5  0x47       // czas trwania 1/2 bitu    zegar 1048576Hz
#define Bitime  0x6C         // czas trwania bitu ~9620 baud
unsigned int RXTXData;
unsigned char BitCnt;
void TX_Byte (void);
void RX_Ready (void);

#include <msp430x41x.h>

void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
  FLL_CTL0 |= XCAP14PF;      // wewnetrzne kondensatory
  CCTL0 = OUT;               // TXD Idle as Mark
  TACTL = TASSEL_2 + MC_2;   // SMCLK, continuous mode
  P1SEL = TXD + RXD;         // P1.0/1 TA0 w trybie TXD/RXD
  P1DIR = TXD;               // wyjscie TXD  P1

// Główna petla
  for (;;)
  {
   RX_Ready ();               //przelaczenie w tryb odbioru danych
   _BIS_SR (CPUOFF + GIE);    // tryb czuwania i przerwanie aktywne
   TX_Byte ();                // wysyla odebrany bajt
  }
}


// wysyla bajt podany w buforze RXTXData
void TX_Byte (void)
{
  BitCnt = 0xA;              // wysyla 8 bitow +1 start+1stop=10
  CCR0 = TAR;                // Timer A w trybie licznika
  CCR0 += Bitime;            // czas trwania bitu
  RXTXData |= 0x100;         // dodanie bitu stopu do bufora RXTXData
  RXTXData = RXTXData << 1;  // dodanie bitu startu
  CCTL0 = OUTMOD0 + CCIE;    // TXD = mark = idle
  while ( CCTL0 & CCIE );    // czekaj na zakonczenie wysylania bajtu
}

// przygotowanie do odbioru danych
void RX_Ready (void)
{
  BitCnt = 0x8;              // liczba bitów do odbioru
  CCTL0 = SCS + CCIS0 + OUTMOD0 + CM1 + CAP + CCIE;   // Sync, Neg Edge, Capture
}


// Przerwanie od Timer A0
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  CCR0 += Bitime;            // czas bitu do CCR0
// Odbior RX
  if (CCTL0 & CCIS0)         // RX w rejestrze CCI0B?
  {
    if ( CCTL0 & CAP )       // jesli tryb Capture, to poczatek bitu startu
    {
    CCTL0 &= ~ CAP;          //przelacz CCTL0 z trybu capture na tryb compare
    CCR0 += Bitime_5;
    }
    else
    {
    RXTXData = RXTXData >> 1;
      if (CCTL0 & SCCI)      // wpisz bit do RXTDData
        RXTXData |= 0x80;
      BitCnt --;             // Jesli odebrane wszystkie bity
      if ( BitCnt == 0)
        {
         CCTL0 &= ~ CCIE;    // odebrane wszytkie bity ->wylacz przerwanie
         _BIC_SR_IRQ (CPUOFF); //tryb czuwania
        }
//---
    }
  }
// TX nadawanie
  else
  {
    if ( BitCnt == 0)
    CCTL0 &= ~ CCIE;         // Jesli wszystkie bity wyslane
    else                     //to wylacza przerwanie
    {
      CCTL0 |=  OUTMOD2;     // TX Space
      if (RXTXData & 0x01)
        CCTL0 &= ~ OUTMOD2; // Znacznik TX
      RXTXData = RXTXData >> 1;
      BitCnt --;
    }
  }
}//***************************************************************************//
```

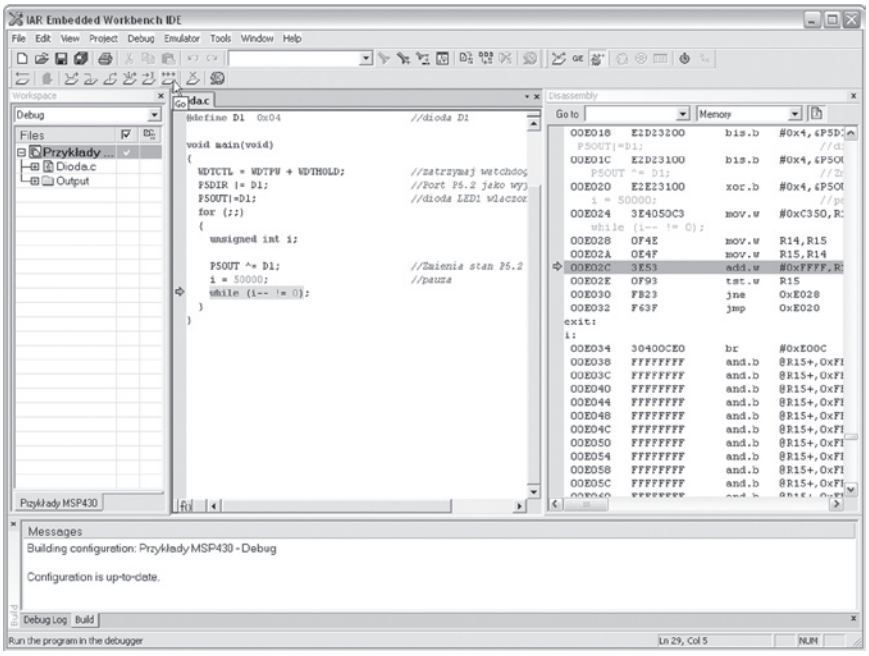
pracy statycznej, umożliwiając obsługę 24 segmentów wyświetlacza. Po skonfigurowaniu sterownika wyświetlanie danych na wyświetlaczu sprowadza się tylko do wykonania odpowiednich wpisów w pamięci danych wyświetlacza, a cała procedura odświeżania jest wykonywana automatycznie. Pamięć ta jest zorganizowana tak, aby możliwe było umieszczenie danych o segmentach także w trybie dynamicznym, dlatego w przedstawionym układzie nie są wykorzystywane wszystkie bity z każdej komórki. Wyświetlenie cyfry wiąże się przez to z koniecznością zdefiniowania poszczególnych segmentów. Została dlatego utworzona tablica stałych *digit*, w której są zawarte odpowiednie dane umożliwiające wyświetlenie poszczególnych cyfr na wyświetlaczu 7–segmentowym. Procedura *clear_lcd ()* służy do kasowania zawartości wyświetlacza poprzez zapisanie całej pamięci wyświetlacza zerami. Powoduje to wygaszenie wszystkich segmentów. Procedura *put_lcd (unsigned int value)* pozwala natomiast na wyświetlenie liczby w postaci dziesiętnej podanej jako argument funkcji. W głównej pętli programu wykonywane jest cykliczne zwiększanie wartości liczby *val* i wyświetlanie jej na wyświetlaczu. W ten sposób został utworzony licznik zliczający w górę od 0 do 1999. Dodatkowo w czasie zliczania mruga dioda D1.

Ostatnim przykładem będzie obsługa portu szeregowego. Na **list. 6** przedstawiono konfigurację procesora tak, aby mógł odbierać i wysyłać dane zgodnie ze specyfikacją standardu RS232. Prędkość transmisji jest ustalona na wartość 9600 bps, a działanie przykładowego programu polega na odebraniu pojedynczego bajtu danych z komputera i wysłaniu go z powrotem. Mikrokontroler jest taktowany przebiegiem zewnętrznym o częstotliwości 32,768 kHz, jednak jest to wartość zbyt niska do zrealizowania transmisji danych z założoną prędkością, dlatego poprzez moduł FLL+ jest ona podwyższona do wartości 1048567 Hz. Transmisja szeregowa jest realizowana przy użyciu licznika Timer_A3 ponieważ procesor nie posiada sprzętowego sterownika. Nadawanie i odbiór poszczególnych bitów jest realizowane w przerwaniu o nazwie Timer_A3, a w zależności od trybu (nadawania lub odbioru) kolejne bity są odbierane z portu P1.1 lub są wysyłane na linię P1.0. Buforem danych szeregowych w obu przypadkach jest zmienna *RXTXData*. W trybie odbioru w buforze tym znajdzie się odebrany bajt, a przy wysyłaniu danych wcześniej należy wpisać do tej zmiennej odpowiednią daną. Główna pętla programu jest ograniczona poleceniem *for* i składa się z wywołania funkcji *RX_Ready ()*, która przygotowuje procesor do trybu odbierania danych. Następnie procesor jest przełączany w tryb czuwania i włączane jest przerwanie. Jeśli na linii P1.1 pojawi się stan niski, to procesor zostanie przełączony w tryb aktywny, zo-

stanie odebrany bajt danych, a następnie poprzez wywołanie funkcji *TX_Byte ()* zostanie wysłany do komputera poprzez linię P1.0.

Do przesyłania danych pomiędzy procesorem, a komputerem można użyć dowolnego programu terminala (np. HyperTerminal), który należy skonfigurować do pracy z prędkością 9600 bps, jeden bit startu, jeden bit stopu, bez kontroli parzystości.

Do zestawu startowego zostaną dołączone pliki przedstawionych programów skonfigurowane i włączone do jednego obszaru roboczego, ułatwiając proces kompilacji i programowania procesora. W głównym katalogu będzie się znajdować plik *Przykłady MSP430.eww*. Jego uruchomienie jest najprostszym sposobem na wczytanie wszystkich przykładów do kompilatora, gdyż plik ten jest skojarzony z kompilatorem i zawiera informacje o projektach zawartych w danym obszarze roboczym. Oczywiście w ramach ćwiczeń programy te można wpisać ręcznie.



Rys. 12. Okno debugera

Ponadto na stronie producenta *http://www–s.ti.com/sc/techzip/slac017.zip* są udostępnione liczne przykłady programów w języku C, które pozwolą zapoznać się z właściwościami procesorów rodziny MSP430.

Krzysztof Pławsiuk, EP
krzysztof.plawsiuk@ep.com.pl