Mikrokontrolery Xmega (2)

Przerwania

Mikrokontrolery AVR firmy Atmel zdobyły w Polsce ogromną popularność. Dotychczas producent oferował dwie rodziny: ATtiny oraz ATmega, które różniły się możliwościami i ceną, choć sposób ich programowania był identyczny. Wprowadzając najnowszą rodzinę, Xmega, firma dokonała istotnych zmian w budowie mikrokontrolera a tym samym – w sposobie pisania programów.

W procesorach ATmega kontroler przerwań był bardzo prosty w budowie. Można go było jedynie włączyć lub wyłączyć. W mikrokontrolerach Xmega układ kontroli przerwań został znacznie rozbudowany i jest traktowany jako pełnoprawny układ peryferyjny o nazwie PMIC (*Programmable Multilevel Interrupt Controller*). Największą zaletą takiego rozwiązania jest, że użytkownik może decydować o priorytecie przerwania – do dyspozycji oddano mu trzy: niski, średni i wysoki. W praktyce oznacza to, że procedura obsługi przerwania o priorytecie niskim może być przerwana przez przerwanie o priorytecie średnim lub wysokim. Przerwanie o priorytecie wysokim nie może być przerwane wcale (wyjątek: wykrycie nieprawidłowego sygnału taktującego). Dzięki temu możemy zdecydować, które zadania są dla nas najważniejsze, by procesor mógł na nie reagować jak najszybciej, a mniej ważne zadania zostawił do dokończenia na później. Jest też dostępny *Scheduler Round-Robin*, by mieć kontrolę nad kolejnością wykonywanych przerwań o tym samym priorytecie w razie natłoku zgłoszeń. Jest to konieczne, gdyż w procesorach Xmega mamy wiele różnych źródeł przerwań do dyspozycji. Na przykład, w ATxmega128A3U jest ponad 100 wektorów przerwań!

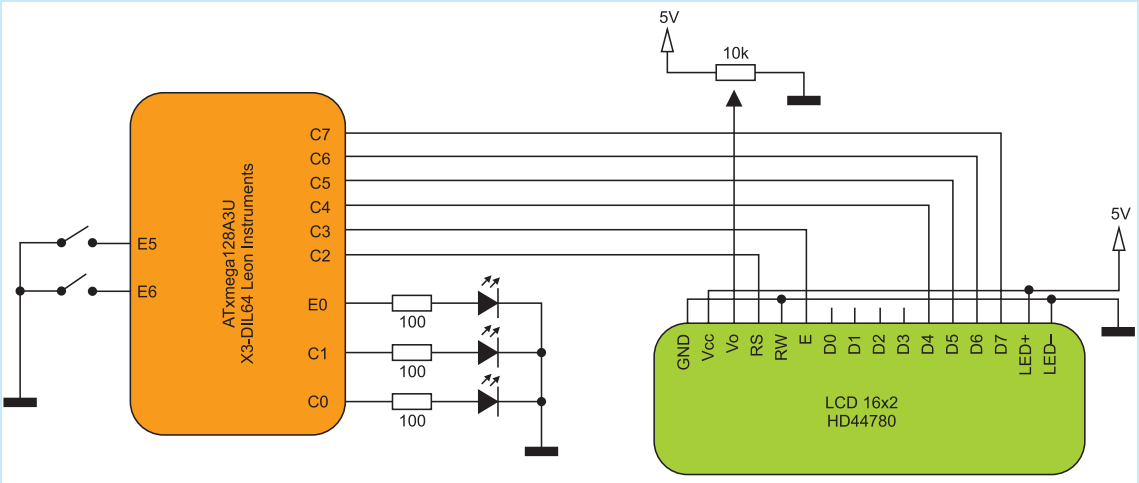
Przerwania dzielą się na maskowalne i niemaskowane. Niemaskowalne jest zaledwie jedno – wykrycie nieprawidłowego działania generatora sygnału zegarowego. Przerwania maskowalne mogą generować wszystkie układy peryferyjne i podobnie jak w ATmega, musimy je odblokować, aby móc je wykorzystać. Kolejnym podobieństwem jest konieczność użycia makra *sei()*, aby uruchomić system przerwań.

W tej części kursu wykonamy nieskomplikowany program demonstrujący działanie przerwań o różnych priorytetach. Poznamy również jak skonfigurować przerwania INT od portów, omawianych w poprzedniej części kursu w EP 2013/11. W pętli głównej CPU będzie zajmowało się wyłącznie sterowaniem diody LED dołączonej do wyprowadzenia B0. Przerwania będą wywoływane przyciskami dołączonymi do wejść portów E5 i E6, a procedury tych przerwań będą powodowały migotanie diodami, odpowiednio, dołączonymi do wyjść portów C1 i C0.

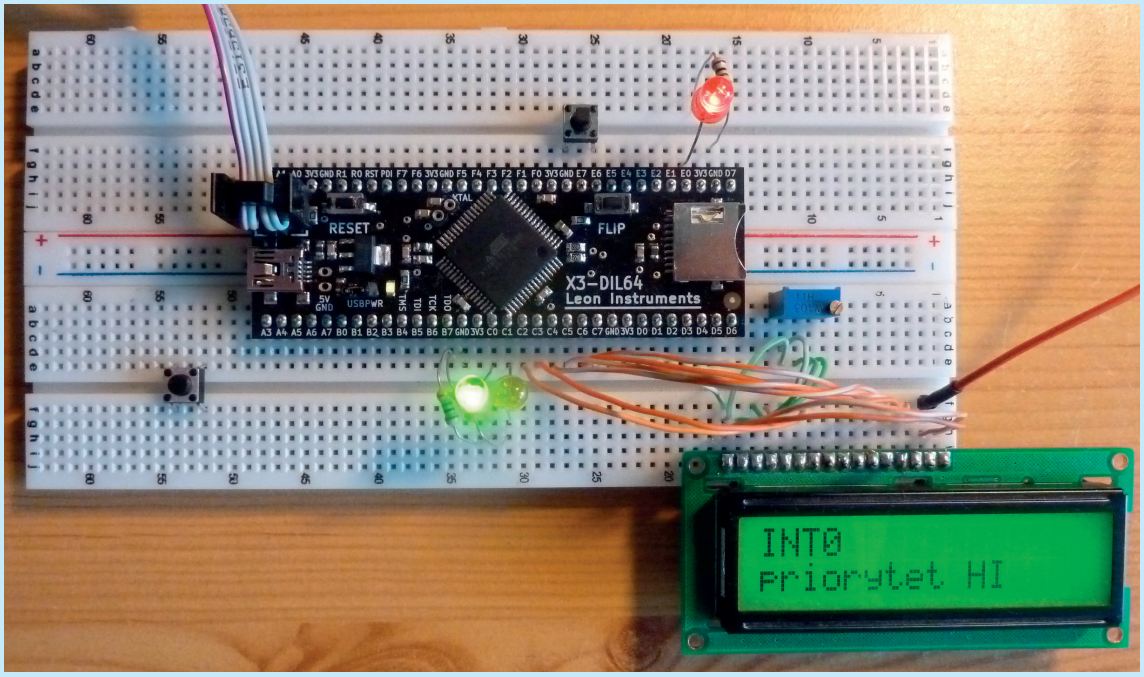
Oprócz sterowania świeceniem diod wykorzystamy również popularny wyświetlacz tekstowy 16×2 ze sterownikiem HD44780. Do sterowania nim użyjemy gotowej biblioteki, którą zamieszczamy w materiałach dodatkowych do artykułu, na płycie CD oraz serwerze FTP. Biblioteka jest autorstwa Radosława Kwietna (*http://radzio.dxp.pl*), a ja jedynie zmodyfikowałem ją do zastosowania z mikrokontrolerami Xmega.

Schemat układu wykorzystywanego w tej części kursu przedstawiono na **rysunku 1**, a zdjęcie układu zbudowanego na płytce stykowej przedstawia **fotografia 2**.

Program zaczynamy, jak zwykle, od konfigurowania portów (**listing 1**). W tym fragmencie programu jedyną nowością jest *PORT_ISC_FALLING_gc* wpisane do rejestrów *PORTxCTRL*. W ten sposób decydujemy, jakie konkretnie zdarzenie ma wywoływać przerwanie. *FALLING*



Rysunek 1. Schemat układu demonstrującego działanie przerwań



Fotografia 2. Zdjęcie układu zmontowanego na płytce stykowej

Listing 1. Konfigurowanie portów mikrokontrolera
```
PORTE.DIRSET = PIN0_bm;  // pin E0 jako wyjście
PORTC.DIRSET = PIN0_bm | PIN1_bm;  // pin C0 i C1 jako wyjście
PORTE.DIRCLR = PIN5_bm | PIN6_bm;  // pin E5 i E6 jako wejście
PORTE.PIN5CTRL = PORT_OPC_PULLUP_gc|  // pull-up na E5
PORT_ISC_FALLING_gc;  // przerwanie ma wywoływać zbocze opadające
PORTE.PIN6CTRL = PORT_OPC_PULLUP_gc|  // pull-up na E6
PORT_ISC_FALLING_gc;  // przerwanie ma wywoływać zbocze opadające
```

Listing 2. Konfigurowanie wejść przerwań INT0 i INT1
```
PORTE.INT0MASK = PIN5_bm;  // pin E5 ma generować przerwania INT0
PORTE.INT1MASK = PIN6_bm;  // pin E6 ma generować przerwania INT1
PORTE.INTCTRL = PORT_INT0LVL_HI_gc|  // poziom HI dla przerwania INT0 portu E
PORT_INT1LVL_LO_gc;  // poziom LO dla przerwanie INT1 portu E
```

Listing 3. Konfigurowanie kontrolera przerwań
```
PMIC.CTRL = PMIC_HILVLEN_bm | // włączenie przerwań o priorytecie HI
  PMIC_LOLVLEN_bm;  // włączenie przerwań o priorytecie LO
sei();
```

Listing 4. Pętla głowna programu
```
while(1)
{
  LcdClear();  // czyszczenie wyświetlacza
  Lcd(„main”);  // wyświetlenie napisu
  PORTB.OUTTGL = PIN0_bm;  // mruganie diodą na B0
  _delay_ms(500);  // czekanie 500ms
}
```

Listing 5. Procedura obsługi przerwania INT0
```
ISR(PORTE_INT0_vect)
{   procedura przerwania INT0 portu E
  for(uint8_t i=0; i<20; i++)
  {  // 10-krotne mrugnięcie diodą na C0
    LcdClear();  // czyszczenie wyświetlacza
    Lcd(„INT0”);  // wyświetlenie napisu
    Lcd2;  // przejście do drugiej linii
    Lcd(„priorytet HI”);
    PORTC.OUTTGL =        PIN0_bm;          // mruganie diodą na C0
    _delay_ms(100);
  }
}
```

Listing 6. Procedura obsługi przerwania INT1
```
ISR(PORTE_INT1_vect)
{  // procedura przerwania INT1 portu E
  for(uint8_t i=0; i<20; i++)
  {  // 10-krotne mrugnięcie diodą na C1
    LcdClear();  // czyszczenie wyświetlacza
    Lcd(„INT1”);  // wyświetlenie napisu
    Lcd2;  // przejście do drugiej linii
    Lcd(„priorytet LO”);
    PORTC.OUTTGL  = PIN1_bm;  // mruganie diodą na C1
    _delay_ms(100);
  }
}
```

oznacza zbocze opadające, a możliwe są jeszcze opcje: *RISING* (zbocze rosnące), *BOTHEDGES* (zbocze rosnące lub opadające) oraz *LEVEL*, czyli poziom logiczny – w tym wypadku logiczne zero.

Każdy port w mikrokontrolerze Xmega może generować przerwanie INT0 i INT1, jednak to my sami możemy wybrać, który wejście ma to przerwanie wywoływać. Mało tego – kilka wejść może wywoływać tę samą procedurę przerwania! Nic nie stoi też na przeszkodzie, by pojedyncze wejście wywoływało oba przerwania, INT0 i INT1, chociaż takie rozwiązanie raczej nie ma sensu praktycznego.

Na przykładzie fragmentu programu zamieszczonego na **listingu 2** można zobaczyć, w jaki sposób przypisać wejścia do poszczególnych przerwań. Następnie, w rejestrze INTCTRL musimy ustalić priorytety przerwań INT0 i INT1 – dostępne opcje to LO, MED, HI albo można przerwanie wyłączyć wpisując *PORT_INTxLVL_OFF_gc*.

Przejdźmy teraz do skonfigurowania kontrolera przerwać PMIC. Musimy w jego rejestrze CTRL odblo-

Listing 7. Przykładowy program opisywany w artykule

```
#define  F_CPU    2000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include „hd44780.h”

int main(void)
{
// konfigurowanie portów
  PORTE.DIRSET = PIN0_bm;                  // pin E0 jako wyjście
  PORTC.DIRSET = PIN0_bm | PIN1_bm;        // pin C0 i C1 jako wyjście
  PORTE.DIRCLR = PIN5_bm | PIN6_bm;        // pin E5 jako wejście
  PORTE.PIN5CTRL = PORT_OPC_PULLUP_gc|     // pull-up na E5
    PORT_ISC_FALLING_gc;  // przerwanie ma wywoływać zbocze opadające
    PORTE.PIN6CTRL = PORT_OPC_PULLUP_gc|   // pull-up na E6
    PORT_ISC_FALLING_gc;  // przerwanie ma wywoływać zbocze opadające
// konfigurowanie przerwań od wejść portów
  PORTE.INT0MASK = PIN5_bm;  // pin E5 ma generować przerwania INT0
  PORTE.INT1MASK = PIN6_bm;  // pin E6 ma generować przerwania INT1
  PORTE.INTCTRL = PORT_INT0LVL_HI_gc|  // poziom HI dla przerwania INT0 portu E
    PORT_INT1LVL_LO_gc;  // poziom LO dla przerwanie INT1 portu E
// włączenie przerwań
  PMIC.CTRL = PMIC_HILVLEN_bm |  // włączenie przerwań o priorytecie HI
    PMIC_LOLVLEN_bm;  // włączenie przerwań o priorytecie LO
  sei();  // globalne włączenie przerwań
  LcdInit();  // inicjalizacja wyświetlacza
// pętla główna
  while(1)
  {
    LcdClear();  // czyszczenie wyświetlacza
    Lcd(„main”);  // wyświetlenie napisu
    PORTE.OUTTGL = PIN0_bm;  // mruganie diodą na B0
    _delay_ms(500);  // czekanie 500ms
  }
}

ISR(PORTE_INT0_vect)
{  // procedura przerwania INT0 portu E
  for(uint8_t i=0; i<20; i++)
  {  // 10-krotne mrugnięcie diodą na C0
    LcdClear();  // czyszczenie wyświetlacza
    Lcd(„INT0”);  // wyświetlenie napisu
    Lcd2;  // przejście do drugiej linii
    Lcd(„priorytet HI”);
    PORTC.OUTTGL = PIN0_bm;  // mruganie diodą na C0
    _delay_ms(100);
  }
}

ISR(PORTE_INT1_vect)
{  // procedura przerwania INT1 portu E
  for(uint8_t i=0; i<20; i++)
  {  // 10-krotne mrugnięcie diodą na C1
    LcdClear();  // czyszczenie wyświetlacza
    Lcd(„INT1”);  // wyświetlenie napisu
    Lcd2;  // przejście do drugiej linii
    Lcd(„priorytet LO”);
    PORTC.OUTTGL = PIN1_bm;  // mruganie diodą na C1
    _delay_ms(100);
  }
}
```

kować przerwania o priorytecie HI oraz LO. Na koniec, należy wpisać instrukcję *sei()*, dobrze znaną z ATmega i ATtiny, aby procesor mógł obsługiwać przerwania (**listing 3**).

Pętlę główną programu testowego pokazano na **listingu 4**. Jest ona nieskomplikowana i w zasadzie nie wymaga żadnego opisu. Na początku jest czyszczony wyświetlacz i jest wyświetlany komunikat, a później jest negowany poziom na wyprowadzaniu portu B0 i odmierzane opóźnienie 0,5 s.

Przykładową procedurę obsługi przerwania INT0 zamieszczono na **listingu 5**, natomiast INT1 na **listingu 6**. Każdy port może generować przerwanie INT0 i INT1 – stąd nazwa przerwania PORTE_INT0_vect. Procedura zawiera pętlę, dzięki której dioda na pinie C0 mrugnie dziesięciokrotnie. Obie procedury obsługi przerwań sa bardzo podobne.

Zobaczmy, jak oprogramowanie działa w praktyce, wykorzystując do tego celu płytkę rozwojową X3-DIL64. Po wgraniu programu mruga dioda dołączona do B0. Naciśnięcie przycisków wywołuje odpowiadające im procedury przerwań. Co się stanie, jeśli wciśniemy przycisk dołączony do wejścia E6, wywołujący przerwanie o priorytecie niskim, a chwilę potem do wejścia E5, który wywoła przerwanie o priorytecie wyższym? Dioda dołączona do wyjścia C1 przestanie mrugać, a zacznie połączona z C0. Kiedy dioda dołączona do wyjścia C0 przestanie migotać, procesor wróci do procedury sterującej diodą dołączoną do C1.

Należy wyraźnie zaznaczyć, że procedury obsługi przerwań powinny być wykonywane jak najszybciej i nie powinno być w nich żadnych funkcji opóźniających. W tym artykule została zastosowana funkcja *_delay_ms()*, aby móc zobaczyć działanie przerwań i ich priorytetów. W „normalnym” programie stosowanie opóźnień w przerwaniach jest wysoce niewskazane. Kompletny program zamieszczono na **listingu 7**.

Dominik Leon Bieczyński
http://leon-instruments.blogspot.com