Mikrokontrolery z rdzeniem ARM, część 18

Interfejsy szeregowe: UART (dokończenie) i I²C

W każdym systemie mikroprocesorowym zachodzi potrzeba wymiany informacji z otoczeniem – na przykład z innymi komputerami, czy urządzeniami peryferyjnymi podłączanymi do systemu. Najbardziej naturalnym sposobem przesyłania danych są magistrale równoległe, w których dane przesyłane są jednocześnie w porcjach odpowiadających długości słowa maszynowego mikroprocesora. Jednak taki sposób przesyłania danych jest kłopotliwy ze względu na dużą liczbę połączeń, gdzie już nawet w przypadku prostego 8–bitowego systemu mikroprocesorowego musimy podłączyć 8 linii danych, 16 linii adresowych i kilkanaście linii sterujących. W przypadku połączenia równoległego z innymi urządzeniami zewnętrznymi na przykład dwoma systemami mikroprocesorowymi znajdującymi się na przeciwnych końcach pomieszczenia wiązałoby się z koniecznością użycia drogich wielożyłowych przewodów.

Program z list. 8, chociaż jest w pełni funkcjonalny. ma pewną wadę: mianowicie port szeregowy mikrokontrolera obsługiwany jest w programie głównym poprzez sprawdzanie rejestru LSR. W przypadku, gdy współpracuje on z terminalem nie jest to szczególnie uciążliwe, jednak gdy potrzebna jest komunikacja z wykorzystaniem jakiegoś protokołu (pomimo istnienia 15 znakowego bufora FIFO) istnieje duże prawdopodobieństwo utraty danych, gdy mikroprocesor będzie zajęty jakąś pracochłonną „protokołową" czynnością obliczeniową. Z tego powodu porty szeregowe w mikrokontrolerach LPC mają możliwość zgłaszania przerwań. Port szeregowy możemy zaprogramować tak, aby wysyłanie i odbieranie danych odbywało się za pomocą procedury przerwania natomiast samo wysyłanie danych w programie głównym sprowadzać się będzie do zapisania lub odczytania bufora. W układzie 16550 do dyspozycji mamy dodatkowy rejestr **IIR** (*Interrupt Information Register*), który pozwala określić przyczynę wystąpienia przerwania:

FIFO_EN		–	–	IDENT			INT
7	6	5	4	3	2	1	0

Rys. 44. Rejestr U1IIR (0xE0010008) U0IIR (0xE000C008)

INT – flaga zgłoszenia przerwania, aktywna w stanie 0

IDENT – określa przyczynę wystąpienia przerwania:

011 – zmiana statusu linii
010 – odebrano nowe dane
110 – timeout. Przez okres 3,5–4,5 znaku nie odebrano żadnych danych
001 – bufor nadajnika jest wolny
000 – przerwanie od linii modemowych (Tylko LPC 2134/2136/2138/2144/2146/2148)

FIFO_EN – załączenie kolejki FIFO są to odpowiedniki bitu EN z rejestru FCR.

Odczytując zawartość rejestru **IIR** w procedurze obsługi przerwania możemy określić powód wystąpienia przerwania. Zmiana statusu linii (011), mówi nam że zawartość rejestru LSR zmieniła się czego przyczyną mogło być wystąpienie jakiegoś błędu. Zdarzenie Timeout (110) informuje nas, że zakończono odbieranie danych, ale bufor FIFO odbiornika nie zapełnił się. Zdarzenie przerwania od linii modemowych występuje tylko w przypadku drugiego portu szeregowego w mikrokontrolerach 21x4/21x6/21x8. Aby przerwania w ogóle zostały wygenerowane należy je wcześniej włączyć w rejestrze **IER** (*Interrupt Enable Register*), który pozwala nam na aktywację pożądanego rodzaju przerwania:

CTS_EN	–	–	–	MSR_EN	RLS_EN	THRE_EN	RBR_EN
7	6	5	4	3	2	1	0

Rys. 45. Rejestr U1IER (0xE0010004) U0IER (0xE000C004)

RBR_EN – Flaga aktywacji przerwania od odebranych znaków oraz od przeterminowania znaku (timeout).

THRE_EN – Flaga aktywacji przerwania informującego o pustym buforze nadajnika.

RLS_EN – Flaga aktywacji przerwania informującego o zmianie rejestru statusu (RLS).

MSR_EN – Flaga aktywacji przerwania informującego o zmianie rejestru statusu linii modemu (MSR).

CTS_EN – Flaga aktywacji przerwania od linii CTS (tylko drugi UART układów 21x4/6/8).

Oprócz aktywacji generowania przerwania w samym układzie portu szeregowego, musimy odpowiednio skonfigurować wektoryzowany kontroler przerwań (VIC). Na **list. 9** przedstawiono procedury obsługi portu szeregowego wysyłające i odbierające informację z terminala ale tym razem z wykorzystaniem systemu przerwań.

Podobnie jak poprzednio funkcja *Uart0Init()* inicjalizuje port szeregowy z prędkością przekazaną jako argument. Procedura ta jest prawie identyczna jak poprzednio, różni się jedynie ustawieniem rejestru U0IER tak aby zgłaszane było przerwanie od nadanych oraz odebranych znaków, oraz inicjalizacją kontrolera przerwań VIC. Przerwanie od portu szeregowego zostało w VIC ustawione jako wektoryzowane. W momencie zgłoszenia przerwania od portu szeregowego mikroprocesor rozpoczyna wykonywanie funkcji *Uart0Int*, która stanowi procedurę jego obsługi. Najpierw określana jest przyczyna wystąpienia przerwania poprzez odczytanie rejestru IIR. W przypadku gdy okaże się, że jest to przerwanie od przeterminowania lub odbioru znaku, wówczas

wszystkie odebrane znaki są przepisywane w pętli *while()* z kolejki FIFO portu do bufora. Natomiast, gdy wystąpi przerwanie od pustego bufora nadajnika, wówczas dane z bufora przesyłane są do kolejki FIFO nadajnika Na zakończenie procedury obsługi zerujemy rejestr *VICVectAddr* informując kontroler przerwań o zakończeniu procedury obsługi. Wysłanie znaku w programie głównym wykonujemy za pomocą funkcji *Uart0PutChar()*, która albo umieszcza znak do nadania w buforze, jeżeli nadajnik portu szeregowego jest uruchomiony, lub rozpoczyna nadawanie wpisując pierwszy znak do rejestru U0THR. Przed operacją na buforze oraz jego wskaźnikach musimy pamiętać o wyłączeniu przerwania od portu szeregowego. W przeciwnym przypadku w momencie zmiany jakiejś zmiennej związanej z buforem mogło by wejść przerwanie, a ponieważ pozostałe zmienne związane z obsługą bufora nie były by jeszcze zaktualizowane groziło by to błędnym działaniem programu. Funkcja *Uart0GetChar()* służy do odebrania znaków z portu szeregowego, w której najpierw jest sprawdzana ilość odebranych znaków i jeżeli jest ona większa od zera wówczas wyłączane są przerwania pobierany jest znak z bufora a następnie przerwania włączane są ponownie. W pliku *ep8b.zip* znajduje się cały kod programu wykorzystujący powyższe funkcje. Po uruchomieniu wysyła on napis powitalny, a następnie na czeka na komendy z terminala. Zaimplementowano tutaj jedną komendę SET=n (gdzie n=0…255), której wywołanie powoduje zapalenie diod D0…D7 zgodnie z reprezentacją bitową wpisanej liczby.

Mikrokontrolery LPC214x oprócz wspomnianego tutaj dodatkowego rejestru dzielnika, posiadają dodatkowy układ sprzętowy pozwalający na automatyczne wykrywanie prędkości transmisji, jednak z uwagi na ograniczone łamy tego artykułu oraz to, że jest to rozszerzeniem standardu 16550 nie występującym w mikrokontrolerach LPC213x zostanie on w tym miejscu omówiony.

Interfejs I²C

Kolejnym bardzo popularnym interfejsem szeregowym jest I²C. Interfejs RS232, najczęściej wykorzystywany był do podłączenia systemu mikroprocesorowego z odległymi komputerami lub innymi systema-

List. 9. Procedury obsługi portu szeregowego wysyłające i odbierające dane z wykorzystaniem przerwań

```
#include "lpc213x.h"
#include "uart.h"
#include "armint.h"

//Bufor odbiornika
static unsigned char rxbuf[TXBUF_SIZE +10];
//Bufor nadajnika
static unsigned char txbuf[RXBUF_SIZE+10];
//Liczniki nadajnika i odbiornika
volatile static unsigned short txcnt, rxpos, txpos, rxcnt;
//Znacznik zajetosci nadajnika
volatile static unsigned char busy;

//Definicje ustawien pinow RXD i TXD
//Ustawienia kontrolera VIC
#define TXD0_P00_SEL (1<<0)
#define RXD0_P01_SEL (1<<2)
//Zapelnienie bufora fifo odbiornika
#define U0IIR_RDA_INT  0x04
//Przez 3,5 znaku nie odebrano danych
#define U0IIR_CTI_INT 0x0C
//Bufor fifo nadajnika pusty
#define U0IIR_THRE_INT 0x02
//8 bitow danych
#define U0LCR_8Bit_Data 3
//1 bit stopu
#define U0LCR_1Bit_Stop 0
//Brak bitu parzystosci
#define U0LCR_No_Parity 0
//Bufor FIFO na 14 znakow
#define U0FCR_14Char_Fifo (3<<6)
#define U0_VIC (1<<6)
#define U0_VIC_BIT 6
#define VIC_IRQSLOT_EN (1<<5)
//Definicja naglowku przerwania
static void Uart0Int(void) __attribute__ ((interrupt("IRQ")));


//Funkcja przerwania
void Uart0Int(void)
{
  unsigned char irqstat = U0IIR & 0x0F;
  unsigned char tmp;
  if(irqstat==U0IIR_RDA_INT || irqstat==U0IIR_CTI_INT)
  {
     //odczytuj dane z fifo dopuki sa
     while(U0LSR & U0LSR_RDR)
     {
        if(rxcnt < RXBUF_SIZE)
        {
           rxbuf [(rxpos + rxcnt++) % RXBUF_SIZE] = U0RBR;
        }
        else
        {
           tmp = U0RBR;
           break;
        }
     }
  }
  if(irqstat==U0IIR_THRE_INT)
  {
     //Mozna nadawac nowy znak
     if(txcnt)
     {
        busy = 1;
        txcnt--;
           U0THR = txbuf[txpos++];
           if(txpos >= TXBUF_SIZE) txpos = 0;
     }
     else
     {
        tmp = U0IIR;
        busy = 0;
     }
  }
  //Informacja dla kontrolera przerwan
  VICVectAddr = 0;
}

/* Inicjalizacja Uart0 */
void Uart0Init(unsigned short BaudRate)
{
  //Funkcje wyjsciowe UART
  PINSEL0 |= TXD0_P00_SEL | RXD0_P01_SEL;
  //DLAB = 1
  U0LCR = U0LCR_Divisor_Latch_Access_Bit;
  //Ustaw predkosci transmisji
  U0DLL = (unsigned char)BaudRate;
  U0DLM = (unsigned char)(BaudRate>>8);
  //Ustawienie 8,n,1
  U0LCR = U0LCR_8Bit_Data | U0LCR_1Bit_Stop | U0LCR_No_Parity;
  //Wlacz FIFO na 14 znakow
  U0FCR = U0FCR_14Char_Fifo | U0FCR_FIFO_Enable;
  //Funkcja przerwania wektora 1
  VICVectAddr1 = (int)Uart0Int;
  //Zalaczenie slotu 1
```

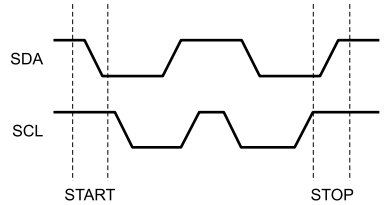
List. 9. c.d.

```
  VICVectCntl1 = U0_VIC_BIT | VIC_IRQSLOT_EN;
  //Kasuj ewentualne znaczniki odbioru nadania znaku
  U0LSR = U0IIR = 0;
  //Wlaczenie przerwan od RX TX
  U0IER = U0IER_RBR_Interrupt_Enable | U0IER_THRE_Interrupt_Enable;
  //Zalaczenie przerwania od UART0
  VICIntEnable = U0_VIC;
  //Odblokuj przerwania
  enable_irq();
}


//Nadawanie znaku
void Uart0PutChar(char c)
{
   //Gdy wszystkie bajty, czekaj na zwolnienie
   while(txcnt >= TXBUF_SIZE);
   //Wylacz na moment przerwanie od RS
   cpu_t irqs = disable_irq();
   U0IER = 0;
   restore_irq(irqs);
   if(busy)
   {
     txbuf[(txpos + txcnt++) % TXBUF_SIZE] = c;
   }
   else
   {
     U0THR = c;
     busy = 1;
   }
   //Wlaczenie przerwan od RX TX
   irqs = disable_irq();
   U0IER = U0IER_RBR_Interrupt_Enable | U0IER_THRE_Interrupt_Enable;
   restore_irq(irqs);
}

//Odebranie znaku
char Uart0GetChar(void)
{
  int c;
  while (!rxcnt);    //Czekaj az bedzie znak
  //Wylacz na moment przerwanie od RS
  cpu_t irqs = disable_irq();
  U0IER = 0;
  restore_irq(irqs);
  rxcnt--;
     c = rxbuf[rxpos++];
     if (rxpos >= RXBUF_SIZE) rxpos = 0;
  //Wlaczenie przerwan od RX TX
  irqs = disable_irq();
  U0IER = U0IER_RBR_Interrupt_Enable | U0IER_THRE_Interrupt_Enable;
  restore_irq(irqs);
  return c;
}
```

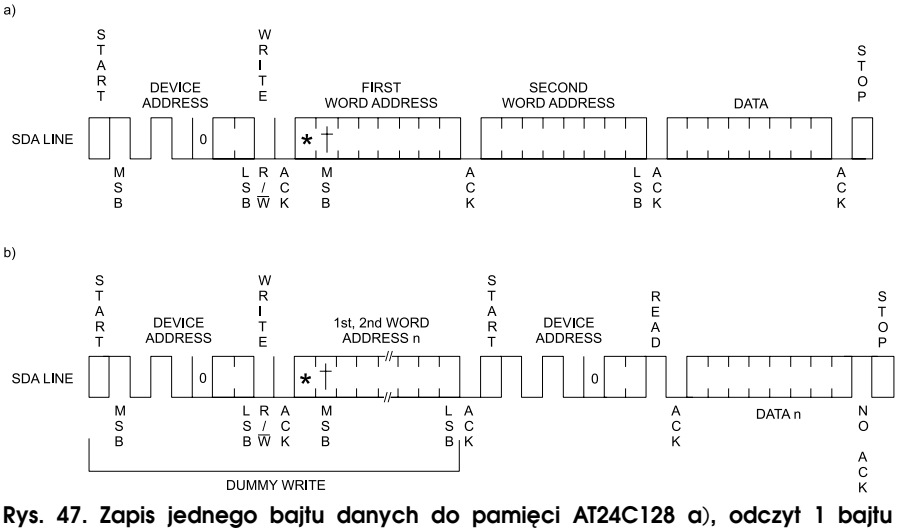
mi mikroprocesorowymi, natomiast interfejs I²C wykorzystujemy do podłączenia dodatkowych układów peryferyjnych do mikrokontrolera w ramach tego samego systemu. Są to różnego rodzaju małe pamięci szeregowe, przetworniki A/C i C/A, klawiatury wyświetlacze itp. Komunikacja standardzie I²C odbywa się szeregowo synchronicznie z zastosowaniem dwóch linii SDA oraz SCL. Są to linie dwukierunkowe podłączone do „dodatniej" szyny zasilającej za pośrednictwem rezystorów podciągających o wartości kilkunastu kΩ. Jeżeli szyna jest zwolniona (brak transmisji) obie linie znajdują się w stanie wysokim. Podczas transmisji danych impulsy taktujące na szynie SCL wysyłane są w ilości jeden impuls na każdy bit przesyłanych danych. Podczas wysokiego stanu na linii SCL stan linii SDA musi być stabilny – stan tej linii może ulegać zmianie tylko podczas niskiego stanu na linii SCL. Sygnał START służy do identyfikacji początku transmisji – po pojawieniu się tego stanu szyna uważana jest za zajętą aż do momentu wystąpienia sygnału STOP, czyli po stwierdzeniu pojawienia się narastającego zbocza na linii SDA przy wysokim stanie linii SCL (**rys. 46**).



Rys. 46. Definicja bitów start i stop

Każdy bajt danych przesyłanych po szynie I²C musi składać się z 8 bitów, przy czym dane nadawane są począwszy od bitu najbardziej znaczącego. Liczba bajtów danych przesyłanych w ramach jeden transmisji nie jest ograniczona. Każda operacja przesłania bajtu danych powinna się zakończyć bitem potwierdzenia. Impuls taktujący związany z bitem potwierdzenia jest generowany przez urządzenie nadrzędne. Do magistrali I²C może być podłączonych wiele układów nadrzędnych i podrzędnych, jednak najczęściej spotykaną sytuacją będzie mikrokontroler, który pełni rolę nadrzędną oraz od jednego do kilkunastu urządzeń podrzędnych. Każde urządzenie na magistrali I²C ma swój 7 bitowy adres (najmłodszy 8 bit adresu określa kierunek przepływu danych) Nie wdając się w dalsze szczegóły na **rys. 47** przedstawiono przebiegi podczas odczytu i zapisu pamięci AT24C128, którą będziemy wykorzystywać w dalszej części kursu.

Gdy do magistrali będzie podłączone tylko jedno urządzenie nadrzędne będące mikrokontrolerem, cały protokół I²C ulega znacznemu uproszczeniu i jest łatwy do zrealizowania na drodze programowej. Większość prostych mikrokontro-



Rys. 47. Zapis jednego bajtu danych do pamięci AT24C128 a), odczyt 1 bajtu danych z pamięci AT24C128 b)

Tab. 5. Przypisania linii I2C do portów I/O mikrokontrolerów LPC213x

Sygnał	Linia (I2C0)	Linia (I2C1)	Opis
SDA	P0.3	P0.14	Linia danych magistrali I²C
SCL	P0.2	P0.11	Linia zegarowa magistrali I²C

lerów 8 bitowych nie ma w ogóle wbudowanych sprzętowych kontrolerów I²C. Mikrokontrolery LPC213x/214x posiadają wbudowane dwa sprzętowe kontrolery magistrali I²C, które mogą pracować w trybie nadrzędnym *(master)* lub podrzędnym *(slave)* Posiadają także wbudowany układ pozwalający sterować prędkością transmisji, oraz układ arbitrażu magistrali pozwalający na pracę wielu układów nadrzędnych na jednej magistrali. W **tab. 5** przedstawiono linie magistral I²C mikrokontrolerów LPC213x.

Protokół I²C posługuje się transmisją synchroniczną, ale z uwagi na duże możliwości konfiguracji częstotliwości taktowania układów peryferyjnych (PCLK), oraz na różne maksymalne prędkości transmisji magistrali I²C (Tryb standardowy – 100 kHz lub tryb szybki – 400 kHz) wprowadzono specjalne rejestry SCLL SCLH, które pozwalają na swobodne ustalenie częstotliwości taktowania magistrali (**rys. 48** i **rys. 49**).

SCLL															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Rys. 48. Rejestr I2C0SCLL (E001C014) oraz I2C1SCLL (E005C014)

SCLH															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Rys. 49. Rejestr I2C0SCLH (E001C010) oraz I2C1SCLH (E005C010)

Częstotliwość taktowania magistrali I²C możemy wyznaczyć według następującego wzoru:

$$F_{I2c} = \frac{P_{clk}}{SCLH + SCLL}$$

Obsługa sprzętowego interfejsu I²C oparta jest na zdarzeniach, kontroler śledzi magistralę I²C i w momencie zmiany stanu magistrali w rejestrze STAT wystawia kod zdarzenia jednocześnie zgłaszając przerwanie. Program obsługi możemy napisać albo jako procedurę obsługi przerwania (zalecane), albo możemy w pętli śledzić jego zawartość.

STAT							
7	6	5	4	3	2	1	0

Rys. 50. Rejestr I2C0STAT (0xE001C004) I2C1STAT (0xE005C004)

W **tab. 6** zebrano poszczególne zdarzenia/stany na magistrali I²C dla kontrolera pracującego w trybie nadrzędnym.

W momencie zgłoszenia przerwania procedura obsługi powinna sprawdzić jakie zdarzenie miało miejsce i podjąć odpowiednie czynności. Na przykład jeżeli adres do zapisu został nadany, wówczas wywołana zostaje procedura obsługi przerwania, która powinna nadać dane, czego można dokonać poprzez wydanie odpowiedniego polecenia kontrolerowi I²C. Do wydawania poleceń kontrolerowi służą rejestry CONCLR i CONSET wpisanie jedynki do rejestru CONSET powoduje ustawienie wybranego bitu, natomiast wpisanie jedynki do rejestru CONCLR powoduje skasowanie odpowiedniego bitu.

–	I2EN	STA	STO	SI	AA	–	–
7	6	5	4	3	2	1	0

Rys. 51. Rejestr I2C0NSET (0xE001C000) I2C1CONSET (0xE005C000)

–	I2ENC	STAC	STOC	SIC	AAC	–	–
7	6	5	4	3	2	1	0

Rys. 52. Rejestr I2C0NSET (0xE001C018) I2C1CONSET (0xE005C018)

I2EN – ustawienie tego bitu powoduje włączenie interfejsu I²C, w przeciwnym wypadku jest on wyłączony i zmiany na liniach SDA i SCL nie są śledzone.

STA – ustawienie tego bitu powoduje przejście kontrolera w tryb master i nadanie bitu STARTU, lub wysłanie bitu ponownego startu. Bit ten musi być skasowany po-

Tab. 6. Zdarzenia/stany na magistrali I²C dla kontrolera pracującego w trybie master

Kod	Zdarzenie
0x08	Bit START został nadany
0x10	Powtórzony bit startu został nadany
0x18	Adres urządzenia I²C (kierunek zapis) został nadany z potwierdzeniem (ACK)
0x20	Adres urządzenia I²C (kierunek zapis) został wysłany brak potwierdzenia (ACK)
0x28	Dane zostały wysłane z potwierdzeniem (ACK)
0x30	Dane zostały wysłane brak potwierdzenia (ACK)
0x38	Utrata magistrali (inne urządzenie master przejęło magistralę)
0x40	Adres urządzenia I²C (kierunek odczyt) został nadany z potwierdzeniem (ACK)
0x48	Adres urządzenia I²C (kierunek odczyt) został wysłany brak potwierdzenia (ACK)
0x50	Dane zostały odebrane i wysłano potwierdzenie (ACK)
0x58	Dane zostały odebrane i nie wysłano potwierdzenia (ACK)
0xF8	Stan jałowy na magistrali nic się nie dzieje

przez wpisanie jedynki do rejestru CONCLR zaraz po jego nadaniu.

STO – ustawienie tego bitu powoduje nadanie bitu stopu na magistrali I²C, jest on zerowany automatycznie gdy zostanie nadany.

SI – Bit ten jest ustawiany w momencie zmiany stanu na magistrali, czyli jest to flaga zgłoszenia przerwania od kontrolera I²C. Aby kontroler podjął wykonywanie kolejnej akcji bit ten należy wyzerować.

AA – ustawienie tego bitu powoduje, wysłanie bitu potwierdzenia ACK, natomiast jego wyzerowanie powoduje brak nadawania bitu potwierdzenia.

Do rejestru **DAT** przesyłane są dane w trybie odczytu, oraz w trybie zapisu należy umieścić tam dane do wysłania:

DAT							
7	6	5	4	3	2	1	0

Rys. 53. Rejestr I2C0DAT (0xE001C008) I2C1DAT (0xE005C008)

Lucjan Bryndza, EP
lucjan.bryndza@ep.com.pl

UWAGA! pliki do kursu zostaną zamieszczone na CD-EP6/2007B