Mikrokontrolery z rdzeniem ARM, część 14

System przerwań

Bieżący stan układu peryferyjnego mikrokontrolera można określić poprzez odczytanie odpowiedniego rejestru SFR (tą metodą posługiwaliśmy się w poprzednio opisywanych przez nas przykładach). Taka metoda jest dobra tylko wtedy, gdy nie zależy nam na szybkiej reakcji na zdarzenie. W przypadku, gdy wymagana jest szybka odpowiedź, mikrokontroler musiałby cały czas cyklicznie badać stan rejestru SFR w celu wykrycia zdarzenia i nie mógłby w tym czasie robić nic innego. Z tych właśnie względów wszystkie mikrokontrolery są wyposażone w mechanizm przerwań.

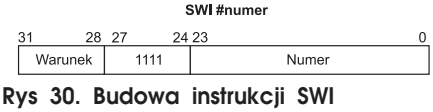
W momencie, gdy nastąpi jakieś zdarzenie (np. odebranie znaku poprzez port szeregowy), wówczas zgłaszane jest przerwanie informujące o tym mikrokontroler. W wyniku, tego przerywane jest wykonanie bieżącego programu i następuje skok do podprogramu obsługi zdarzenia. Po zakończeniu mikrokontroler wraca do wykonania programu w miejscu, w którym został on przerwany. W prostych mikrokontrolerach 8–bitowych na przykład 8051 działanie przerwań było bardzo proste, mianowicie wystąpienie jakiegoś zdarzenia powodowało skok pod konkretny adres w pamięci. System przerwań mikrokontrolerów LPC213x/214x jest zdecydowanie bardziej skomplikowany z uwagi na to, że sam rdzeń mikrokontrolera posiada tylko dwie linie wejść przerywających (IRQ oraz FIQ), dlatego konieczne stało się wprowadzenie kontrolera przerwań, podobnie jak ma to miejsce w komputerach klasy PC.

W bieżącym odcinku zapoznamy się z działaniem systemu przerwań w mikrokontrolerach LPC213x. Przypomnimy podstawowe wiadomości dotyczące obsługi przerwań poprzez rdzeń ARM7TDMI, zapoznamy się z budową kontrolera przerwań VIC (*Vectorized Interrupt Controller*) oraz sposobem obsługi przerwań zewnętrznych. Zapoznamy się także z mechanizmem generowania przerwań programowych z wykorzystaniem instrukcji SWI.

Przerwania programowe

W systemach mikroprocesorowych idea przerwań programowych polega na przerwaniu wykonania bieżącego programu w momencie napotkania specjalnej instrukcji i skok pod odpowiedni wektor przerwania, tak jakby było to przerwanie generowane sprzętowo. Czytelnikom może się wydać bezcelowe wprowadzanie specjalnej instrukcji przerywającej, ponieważ do złudzenia przypomina ona zwykłe wywołanie CALL, czyli skok do podprogramu. Działanie to polega na zapamiętaniu na stosie lub w odpowiednim rejestrze adresu bieżącej instrukcji, a następnie kontynuację wykonywania programu od adresu będącego argumentem instrukcji. Jest to zasadnicza wada tego mechanizmu, ponieważ musimy znać dokładny adres skoku. Na przykład, chcąc wywołać jakąś funkcję systemu operacyjnego musielibyśmy dokładnie wiedzieć, że znajduje się ona pod konkretnym adresem w pamięci. Wiadomo, że z czasem oprogramowanie takie jak system operacyjny ewoluuje, w efekcie czego nie da się zagwarantować, że konkretna procedura będzie znajdować się pod tym samym adresem, gdyż prowadziłoby to do dużego marnotrawstwa pamięci. Aby zapobiec bałaganowi w tej kwestii, w systemach mikroprocesorowych wprowadzono instrukcję przerwań programowych, której wywołanie z danym argumentem gwarantuje przekazanie sterowania programu zawsze pod ten sam adres w pamięci. W efekcie tego, niezależnie od wersji systemu operacyjnego oraz zmian w mapie pamięci, będziemy mogli zagwarantować jednolity interfejs wywołań systemowych. W mikroprocesorach x86 przerwanie programowe jest generowane instrukcją INT, której wywołanie powoduje skok pod adres, znajdujący się w tablicy wektorów przerwań. W przypadku mikrokontrolerów ARM7TDMI–S sprawa jest prostsza, ponieważ wywołanie przerwania programowego (instrukcja SWI) powoduje wygenerowanie wyjątku *software interrupt* i skok pod niezmienny adres 0x00000008. Dodatkowo w momencie zgłoszenia wyjątku zmieniany jest tryb ochrony z bieżącego na *supervisor*. W związku z tym, podczas normalnego wykonania programu mikroprocesor może pracować w bezpiecznym trybie użytkownika, a w momencie potrzeby wykonania jakiejś funkcji systemowej procedura przerwania systemowego ma dostęp do wszystkich zasobów. Wywołanie instrukcji przerwania programowego jest jedynym możliwym sposobem na przejście z trybu użytkownika do trybu uprzywilejowanego. Na **rys. 30** przedstawiono sposób interpretacji instrukcji SWI przez rdzeń ARM7.

Bity 31…28 – jak zwykle – zawierają kod warunkowy instrukcji, bity 27...24 zawierają właściwy kod instrukcji SWI (1111b), natomiast pozostałe bity (23…0) mogą być wykorzystane przez procedurę obsługi wyjątku do określenia podprogramu, jaki ma zostać wykonany w zależności od tego, jaką liczbę zawierają bity (23...0). Na przykład wpisanie instrukcji SWI #8 spowoduje umieszczenie w bitach (23.0) rozkazu wartości 8. Aby program obsługi wyjątku mógł określić numer przerwania, musi odczytać z pamięci programu instrukcję SWI, a następnie wyciągnąć z niej argument znajdujący się w bitach 23...0. W momencie wystąpienia wyjątku, do licznika rozkazów wpisywany jest wektor przerwania SWI (0x00000008) oraz do rejestru LR wpisywana jest zawartość licznika rozkazów, tak więc odejmując od licznika rozkazów liczbę 4 otrzymamy adres instrukcji SWI. W wyniku odczytania danych spod tego adresu otrzymamy kod instrukcji SWI, a w wyniku zamaskowania 8 najstarszych bitów otrzymamy numer przerwania SWI. Niektóre kompilatory wspierają bezpośrednio obsługę me-



Rys 30. Budowa instrukcji SWI

chanizmu przerwań programowych, natomiast w przypadku używanego przez nas kompilatora GCC cała obsługa spoczywa na programiście. Jeżeli chcemy przekazywać dodatkowe parametry do procedury obsługi danego wątku, możemy to zrobić za pośrednictwem wartości przekazywanych do dowolnych rejestrów, a następnie w procedurze obsługi przerwania programowego możemy odczytać ich zawartość. Aby poznać sposób realizacji przerwań programowych, napiszemy prosty program (plik *ep6a.zip* na CD–EP1/2007B), który za pomocą przerwania SWI będzie włączał oraz wyłączał diody LED znajdujące się na płytce ZL6ARM. Do działania programu musimy zmodyfikować zawartość pliku startowego *boot.s* Pierwsza modyfikacja polega na przydzieleniu kilkudziesięciu bajtów na obszar stosu dla trybu *supervisor*:

```
.equ   SVC_Stack_Size,
0x00000020
```

Kolejną zmianą, jakiej musimy dokonać, to ustawienie adresu funkcji obsługi wyjątku przerwania programowego:

```
SWI_Addr:      .word
SwiIntHandler
```

Program przestawiono na **list. 3**.

Procedura obsługi wyjątku ma taką samą nazwę jak ta, która jest przypisana w pliku *boot.s*. Została ona zadeklarowana jako *extern „C”* przez co kompilator C++ nie zmienia nazwy tej funkcji oraz z modyfikatorem *__attribute__ ((interrupt(„SWI”)))*, co informuje kompilator, że jest to funkcja obsługi wyjątku przerwania programowego. W przypadku, gdyby została ona zadeklarowana jako zwykła funkcja, mikroprocesor zwyczajnie by się zawiesił, ponieważ inna jest funkcja wyjścia kończąca obsługę wyjątku SWI, o czym była mowa we wcześniejszej części kursu. W procedurze obsługi wyjątku posłużono się bardzo ciekawą właściwością kompilatora, umożliwiającą przypisanie zmiennej lokalnej do określonego rejestru. W naszym przypadku zmiennej *link_ptr* przypisano rejestr LR (R14), tak więc wykonując instrukcję *switch (*(link_ptr–1) & 0x00FFFFFF)* możemy określić numer przerwania programowego, jakie spowodowało wyjątek. W naszym przypadku przerwanie SWI #1 włącza LED–y, których maska bitowa przekazana jest w rejestrze R0, natomiast przerwanie programowe SWI #2 wyłącza je. Użycie dodatkowego rejestru (R0), w którym przekazujemy maskę bitową diod, miało na celu pokazanie sposobu przekazywania dodatkowych parametrów do procedur przerwań programowych. Działanie funkcji *SwiLedOn*/*SwiLedOff* polega na przepisaniu do rejestru R0 maski bitowej diod oraz wywołania przerwania programowego SWI #1/SWI #2. Działanie programu głównego opiera się na cyklicznym wywoływaniu funkcji *SwiLedOn*/*SwiLedOff*, co powoduje błyskanie diod D7, D4, D1, D0 na płytce uruchomieniowej.

List. 3.

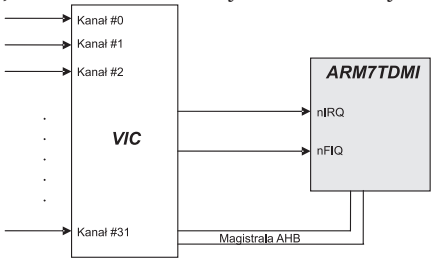
```
#include "lpc213x.h"

//Definicja LEDOW
#define LEDS (0xFF<<16)
#define LEDDIR IO1DIR
#define LEDSET IO1SET
#define LEDCLR IO1CLR

//Deklaracja funkcji przerwania SWI
extern "C" void SwiIntHandler(void) __attribute__ ((interrupt("SWI")));
//Funkcja przerwania SWI
void SwiIntHandler(void)
{
  //Rejestr R14-4 zawiera adres instrukcji SWI
  register unsigned int* link_ptr asm ("r14");
  //Rejestr R0 zawiera parametr przekazany do SWI
  register unsigned int param asm ("r0");
  param &= 0xff;
  param <<= 16;
  switch(*(link_ptr-1) & 0x00FFFFFF)
  {
  //Zalacz Ledy (SWI #1)
  case 0x01:
     LEDSET = param;
     break;
  //Wylacz Ledy (SWI #2)
  case 0x02:
      LEDCLR = param;
     break;
  }
}

//Funkcja wlaczajaca LEDY poprzez SWI
static inline void SwiLedOn(unsigned int LedSt)
{
  asm volatile
  (
     "mov r0,%[ledon]\n"
     "swi #1\n"
     ::[ledon]"r"(LedSt):"r0"
  );
}

//Funkcja wylaczajaca LEDY poprzez SWI
static inline void SwiLedOff(unsigned int LedSt)
{
  asm volatile
  (
     "mov r0,%[ledon]\n"
     "swi #2\n"
     ::[ledon]"r"(LedSt):"r0"
  );
}


/* Funkcja glowna main */
int main(void)
{
  //Ledy jako wyjscie
  LEDDIR |= LEDS;
  //Petla nieskonczona
  while(1)
  {
     //Wlacz D7,D4,D1,D0
     SwiLedOn(0x93);
     //Czekaj
     for(int i=0;i<2000000;i++);
     //Wylacz D7,D4,D1,D0
     SwiLedOff(0x93);
     //Czekaj
     for(int i=0;i<2000000;i++);
  }
  return 0;
}
```

Zapalanie i gaszenie diod LED za pośrednictwem przerwań programowych jest oczywiście lekką przesadą, ponieważ powinny być one wykorzystywane jako funkcję obsługi systemu operacyjnego, ale przykład ten ma pokazać sposób, w jaki można z nich korzystać we własnych aplikacjach. Jako ciekawe zastosowanie nasuwa mi się tutaj wykorzystanie przerwań SWI do konfiguracji systemu przerwań zewnętrznych mikrokontrolera LPC21xx. W pliku startowym *boot.s* należy ustawić procesor w tryb użyt-

Tab. 19. Podłączenie poszczególnych kanałów kontrolera VIC do układów peryferyjnych

Nr kanału	Urządzenie	Zgłaszane przerwania
#0	WDT	Przerwanie WATCHDOG
#1	-	Zarezerwowane dla przerwań programowych
#2	Rdzeń ARM	Embedded ICE (RX)
#3	Rdzeń ARM	Embedded ICE (TX)
#4	TIMER0	Match (0…3) Capture (0…3)
#5	TIMER1	Match (0…3) Capture (0…3)
#6	UART0	Status linii (RLS) Rejestr nadajnika pusty (THRE) Odebrane dane są dostępne (RDA) Przeterminowanie odebrania znaku (CTI)
#7	UART1	Status linii (RLS) Rejestr nadajnika pusty (THRE) Odebrane dane są dostępne (RDA) Przeterminowanie odebrania znaku (CTI) Przerwanie status modemu (MSI)
#8	PWM0	Match (0…6)
#9	I2C	Zmiana stanu (SI)
#10	SPI0	Przerwanie SPI (SPIF) Mode Fault (MODF)
#11	SPI1 (SSP)	FIFO nadajnika w połowie puste (TXRIS) FIFO odbiornika w połowie pełne (RXRIS) Przeterminowanie odbioru (RTRIS) Nadpisany bufor odbiornika (RORRIS)
#12	PLL	PLL Lock (PLOCK)
#13	RTC	Zwiększenie licznika (RTCCIF) Alarm (RTCALF)
#14	System	Przerwanie zewnętrzne 0 (EINT0)
#15	System	Przerwanie zewnętrzne 1 (EINT1)
#16	System	Przerwanie zewnętrzne 2 (EINT2)
#17	System	Przerwanie zewnętrzne 3 (EINT3)
#18	ADC0	Zakończona konwersja przetwornika
#19	I2C1	Zmiana stanu (SI)
#20	BOD	Wykryto zanik napięcia zasilającego
#21	ADC1	Zakończona konwersja przetwornika ADC1

kownika, wówczas tylko wywołanie SWI z odpowiednim parametrem będzie umożliwiało zmianę ustawień systemu przerwań, co w istotny sposób może podnieść bezpieczeństwo działania systemu. Wektoryzowany kontroler przerwań (VIC) można skonfigurować tak, aby odwołanie do rejestrów kontrolera było możliwe tylko w uprzywilejowanym trybie ochrony.



Rys. 31. Dołączenie kontrolera przerwań do rdzenia ARM7

Przerwania sprzętowe – kontroler przerwań VIC

Jak wiemy z poprzednich odcinków kursu, rdzeń ARM7TDMI–S posiada tylko dwa wejścia przerwań zewnętrznych FIQ oraz IRQ. Przerwanie FIQ jest przerwaniem szybkim o najwyższym priorytecie i najkrótszym czasie reakcji, powinno być one wykorzystywane do pisania czasowo krytycznych procedur obsługi przerwań. Natomiast przerwanie IRQ powinniśmy wykorzystywać do obsługi przerwań, które nie wymagają czasowo krytycznej reakcji. Przerwania obsługi są jednopoziomowe i w momencie wejścia do procedury obsługi nie może być zgłoszone kolejne przerwanie tej samej kategorii. Przerwanie szybkie FIQ może natomiast przerwać działanie procedury obsługi przerwania IRQ. Ponieważ dwie linie obsługi przerwań to zdecydowanie za mało mikrokontrolery LPC21xx zostały wyposażone w wektoryzowany kontroler przerwań VIC. Sposób połączenia kontrolera przerwań z rdzeniem ARM7 przedstawiono na **rys. 31**.

W **tab. 19** przedstawiono podłączenie poszczególnych kanałów kontrolera VIC do układów peryferyjnych.

Przerwania FIQ

Kontroler przerwań umożliwia skonfigurowanie każdej z 32 linii tak, aby sygnał aktywny na danej linii generował przerwanie szybkie FIQ. Można tego dokonać poprzez ustawienie bitu odpowiadającego numerowi kanału w rejestrze **`VICIntSelect (0xFFFFF00C)`**. Ustawienie odpowiedniego bitu spowoduje, że dane przerwanie będzie zgłaszane jako FIQ, natomiast jego wyzerowanie spowoduje, że wybrane przerwanie będzie zgłaszane jako IRQ. Na przykład, jeżeli chcemy, aby przerwanie od portu UART0 (kanał #6) było zgłaszane jako FIQ, musimy ustawić bit 6 w rejestrze *VICIntSelect*. Kontroler VIC umożliwia podłączenie więcej niż jednego kanału do linii FIQ. Wówczas jakiekolwiek przerwanie na jednej z tych linii spowoduje zgłoszenie przerwania FIQ. Określenie, który kanał zgłosił przerwanie jest możliwe poprzez zbadanie zawartości rejestru **`VICFIQStatus (0xFFFFF004)`**. Jeżeli dana linia przerwania została zakwalifikowana jako FIQ i przerwanie od tej linii zostało zgłoszone, wówczas ustawiany jest bit odpowiadający numerowi kanału przerwania. Jak wiadomo określenie kanału zgłaszającego przerwanie i podjęcie stosownej reakcji zajmuje pewien czas, dlatego generalnie nie powinniśmy przypisywać do linii FIQ więcej niż jednego przerwania. Jeżeli oczywiście chcemy, aby przerwanie to było wykorzystywane zgodnie z przeznaczeniem, czyli jako przerwanie szybkie. Aby przerwanie FIQ zostało zgłoszone, musimy w rejestrze **`VICIntEnable (0xFFFFF010)`** ustawić bit odpowiadający numerowi kanału przerwania. Zapis 1 na odpowiednim bicie danego rejestru spowoduje, że dane przerwanie będzie zgłaszane, natomiast wpisanie 0 nie przynosi żadnego efektu. Do kasowania zezwolenia na przerwanie danego kanału służy rejestr **`VICIntEnClear (0xFFFFF014)`**. Wpisanie do niego jedynki na odpowiednim bicie spowoduje wyłączenie danej linii przerwania, natomiast wpisanie zera nie przynosi żadnego efektu. Ponadto musimy pamiętać, że aby przerwanie FIQ zostało w ogóle zgłoszone, musi być ono odblokowane w jednostce centralnej. W momencie wystąpienia przerwania FIQ, mikrokontroler skacze pod adres *0x0000001C*, pod który musimy wpisać skok do odpowiedniej procedury obsługi przerwania FIQ. Przed zakończeniem obsługi przerwania, którego epilog jest wypełniany przez kompilator C/C++ musimy pamiętać, aby wyzerować flagę zgłoszenia przerwania w zgłaszającym urządzeniu peryferyjnym. Gdy o tym zapomnimy, wówczas dane przerwanie będzie zgłaszane bez przerwy. Generalnie w mikrokontrolerach LPC21xx kasowanie flag przerwań odbywa się poprzez wpisanie jedynki w rejestrze przerwań wybranego urządzenia peryferyjnego, a nie jak w innych mikrokontrolerach gdzie wpisanie 0 kasowało wybraną flagę zgłoszenia przerwania.

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