Mikrokontrolery z rdzeniem ARM, część 22

Przetwarzanie C/A, biblioteka standardowa

Przetworniki C/A są układami peryferyjnymi stosunkowo rzadko spotykanymi w typowych mikrokontrolerach. Producent mikrokontrolerów LPC, począwszy od wersji LPC21x2, wyposażyli je w jeden kanał konwersji C/A o rozdzielczości 10 bitów z buforowanym wyjściem napięciowym.

Przetwornik wbudowany w prezentowane mikrokontrolery charakteryzuje się czasem przetwarzania rzędu 1 µs. Wyjściem przetwornika jest sygnał AOUT (P0.25), który może przyjąć wartości napięć z zakresu 0...Vref. Przetwornik ten może być wykorzystany do różnorakich celów na przykład do generowania sygnału audio, a sterowanie nim jest naprawdę bardzo proste, ponieważ sam przetwornik posiada tylko jeden rejestr. Zamiana wartości cyfrowej na analogową sprowadza się do wpisania odpowiednich wartości do tego rejestru. Rejestrem tym jest **DACR** (*0xE006C000*), którego wykaz bitów przedstawiono na **rys. 72**.

Funkcje jego poszczególnych bitów są następujące:

VALUE – Zapisanie odpowiedniej wartości na tych bitach powoduje pojawienie się napięcia na wyjściu AOUT, będącego odzwierciedleniem wartości tych bitów. Wartość napięcia pojawiającego się na wyjściu AOUT możemy wyznaczyć według wzoru: U_{out}=(VALUE/1023)*Vref

–	…	BIAS	VALUE										…	…
31	…	16	15	14	13	12	11	10	9	8	7	6	…	0

Rys. 72. Rejestr DACR

List. 14. Program do odgrywania ciągu próbek z pliku WAV

```
#include „lpc213x.h”
#include „armint.h”
#include „wav.h”

#define P025_DAC_SEL (2<<18)
#define TIMER0_VIC (1<<4)
#define TIMER0_VIC_BIT 4
#define VIC_IRQSLOT_EN (1<<5)

static int AdcPos = 0;

//Przerwanie od Timera
void IrqTimerHandler(void) __attribute__ ((interrupt(„IRQ”)));

void IrqTimerHandler(void)
{
   //Zapis danych do DAC
   DACR = ((unsigned int)wav_file[AdcPos++])<<8;
   if(AdcPos>= wav_length) T0TCR = 0;
   //Kasuj zrodlo przerwania
   T0IR = T0IR_MR0;
   //Informacja dla VIC - koniec procedury przerwania
   VICVectAddr = 0;
}

/* Funkcja glowna main */
int main(void)
{
   PINSEL1 |= P025_DAC_SEL;
   //Preskaler wylaczony
   T0PR = 0;
   //Gdy warunek spelniony zeruj Timer i zglaszaj przerwanie
   T0MCR |= T0MCR_Interrupt_on_MR0 | T0MCR_Reset_on_MR0;
   //Przerwanie z częstotliwością 11khz
   T0MR0 = 2720;
   //Zeruj licznik i preskaler
   T0TCR = T0TCR_Counter_Reset;
   //Zalacz licznik T0
   T0TCR = T0TCR_Counter_Enable;
   //Wektor 0
   VICVectAddr0 = (unsigned int)IrqTimerHandler;
   VICVectCntl0 = TIMER0_VIC_BIT | VIC_IRQSLOT_EN;
   //Zalaczenie przerwania
   VICIntEnable = TIMER0_VIC;
   //Zalacz IRQ
   enable_irq();
   return 0;
}
```

BIAS – Bit ten pozwala na ustalenie prędkości pracy przetwornika C/A, a co się z tym wiąże umożliwia określenie prądu pobieranego przez przetwornik. W przypadku, gdy bit ten jest wyzerowany prąd pobierany przez przetwornik wynosi około 700 µA, a czas przetwarzania przetwornika wynosi 1 µs. W przypadku, gdy bit ten jest ustawiony, wówczas przetwornik charakteryzuje się zmniejszonym poborem prądu do 350 µA, ale równocześnie ulega zwiększeniu do 2,5 µs czas przetwarzania przetwornika.

Używanie tego przetwornika A/C jest naprawdę bardzo proste i sprowadza się tylko do wpisania wartości reprezentującej napięcie do rejestru **DACR**. Jedyną czynnością, o której musimy pamiętać to, ustawienie linii P0.25 portu za pomocą rejestru **PINSEL1**, tak, aby pełniła ona rolę wyjścia przetwornika C/A. Na zakończenie kursu napiszemy program (dostępny na CD–EP9/2007B pod nazwą *ep9c.zip*), który wykorzystując przetwornik C/A, będzie odtwarzał plik dźwiękowy zawarty w wewnętrznej pamięci Flash, na głośniczku wbudowanym w zestaw ZL6ARM. Plik dźwiękowy został zapisany bezpośrednio w pliku *wav.c*, w postaci próbek dźwiękowych i został przygotowany na podstawie zwykłego pliku w formacie *wav* z wykorzystaniem narzędzi *sox* oraz *awk*, które zawarte są w pakiecie Cygwin. Plik

Tab. 8. Skrócony opis funkcji w bibliotece stdio

Nazwa funkcji	Opis
_ssize_t _read_r(struct _reent *r,int file, void *ptr,size_t len)	funkcja odczytująca dane z pliku lub konsoli lub terminala
_ssize_t _write_r (struct _reent *r,int file,const void *ptr,size_t len)	funkcja zapisująca dane do pliku lub konsoli lub terminala
int _close_r(struct _reent *r,int file)	funkcja zamykająca plik
_off_t _lseek_r(struct _reent *r,int file,_off_t ptr,int dir)	funkcja określająca przesunięcie w pliku
int _fstat_r(struct _reent *r,int file,struct stat *st)	funkcja zwracająca status pliku
int isatty(int file)	funkcja zwracająca czy otwarty plik jest terminalem

dźwiękowy opiera się na zmiennej tablicowej *wav_file[]* zawierającej próbki sygnału w formacie 8–bitowym oraz *wav_length* określającej rozmiar próbek. Zmienne te zostały zadeklarowane ze słowem kluczowym *const*, przez co kompilator automatycznie traktując je jako dane stałe umieszcza je w obszarze pamięci Flash. Odtwarzanie pliku dźwiękowego sprowadza się do wysyłania poszczególnych próbek sygnału analogowego do przetwornika C/A z częstotliwością 11 kHz. Fragment programu do odgrywania ciągu próbek za pomocą przetwornika C/A przedstawiono na **list. 14**.

Odtwarzanie pliku dźwiękowego rozpoczyna się od razu po wyzerowaniu mikrokontrolera, a po jego zakończeniu jest zatrzymywane, dlatego aby ponownie odsłuchać zawartość pliku należy nacisnąć przycisk zerowania mikrokontrolera. Po wyzerowaniu rozpoczyna się wykonywanie funkcji głównej (*main*), w której na początku ustawiana jest linia P0.25 tak, aby pełniła rolę wyjścia przetwornika C/A. Przesyłanie „pustych" próbek do przetwornika odbywa się w przerwaniu od układu czasowo–licznikowego T0, z wykorzystaniem układu porównującego MR0, dlatego następną czynnością jest skonfigurowanie układu porównującego w taki sposób, aby zgłaszał przerwanie z częstotliwością 11 kHz. Następnie konfigurowany jest kontroler przerwań VIC, tak, aby przerwanie od T0 powodowało generowanie przerwania wektoryzowanego, a na koniec włączana jest globalna flaga zezwoleń na przerwanie. Całą pracę polegającą na przesyłaniu poszczególnych próbek sygnału z odpowiednią częstotliwością do przetwornika C/A wykonuje funkcja obsługi przerwania *IrqTimerHandler()*. W procedurze obsługi przerwania próbki przesuwane są na odpowiednią pozycję oraz przesyłane do rejestru przetwornika C/A. Dzieje się tak do momentu dopóki wszystkie próbki nie zostaną przesłane, natomiast po zakończeniu przesyłania ostatniej próbki zatrzymywany jest układ czasowo–licznikowy i cały proces odtwarzania pliku dźwiękowego kończy się. Ponowne rozpoczęcie odtwarzania pliku dźwiękowego rozpocznie się po wciśnięciu przycisku zerującego mikrokontroler. Przedstawiony tutaj przykład miał pokazać jedynie możliwość odtwarzania prostych plików dźwiękowych za pomocą przetwornika C/A, wykorzystano tutaj bezpośrednie przechowywanie parametrów próbek w pamięci Flash. W rzeczywistym programie odtwarzającym warto pomyśleć, o jakimś prostym algorytmie umożliwiającym skompresowanie próbek dźwiękowych, co pozwoli na zaoszczędzeniu dodatkowego miejsca w pamięci. Można również pliki dźwiękowe przechowywać w jakiejś dużej zewnętrznej pamięci takiej jak karta MMC, czy pamięć DATA Flash.

Biblioteka standardowa (stdio)

W prezentowanych przykładach posługiwaliśmy się bezpośrednio funkcjami biblioteki standardowej *<stdio.h>*, a dokładniej funkcją *printf* w celu wyświetlania komunikatów tekstowych bezpośrednio na terminalu szeregowym. W przypadku standardowych komputerów PC zadanie tej funkcji jest oczywiste i polega na wyświetleniu ciągu znaków bezpośrednio na monitorze komputera. W tym celu funkcja *printf* wywołuje odpowiednie funkcję systemu operacyjnego wyświetlające poszczególne znaki. W przypadku mikrokontrolerów nie mamy zdefiniowanego monitora ekranowego, jednak do tego celu można wykorzystać port szeregowy mikrokontrolera. Pisząc programy dla małych mikrokontrolerów pracujemy bez obecności systemu operacyjnego zapewniającego jednolity interfejs programowy, a każdy mikrokontroler posiada z reguły inny układ portu szeregowego, nie

List. 15. Funkcja umozliwiająca zapis danych do terminala

```
_ssize_t _write_r (struct _reent *r,int file,const void *ptr,size_t len)
{
   int i;
   const unsigned char *p;
   p = (const unsigned char*) ptr;

   for (i = 0; i < len; i++)
   {
      if (*p == ,\n' ) Uart0PutChar(,\r');
      Uart0PutChar(*p++);
   }
   return len;
}
```

List. 16. Funkcja umożliwiająca odczyt znaków z terminala i przekazywanie ich do wyższych funkcji biblioteki *stdio*

```
_ssize_t _read_r(struct _reent *r,int file, void *ptr,size_t len)
{
   char c;
   int  i;
   unsigned char *p;
   p = (unsigned char*)ptr;
   for (i = 0; i < len; i++)
   {
      c = Uart0GetChar();
      if (c == 0x0D)
      {
         *p='\0';
         break;
      }
      *p++ = c;
      Uart0PutChar(c);
   }
   return len - i;
}
```

List. 17. Funkcja *_close_r* służąca do zamykania plików
```
int _close_r(struct _reent *r,int file)
{
  return 0;
}
```

List. 18. Funkcja *_lseek_r* umożliwiająca zmian ę pozycji w pliku
```
_off_t _lseek_r(struct _reent *r,int file,_off_t ptr,int dir)
{
  return (_off_t)0;  /*  Always indicate we are at file beginning.  */
}
```

List. 19. Funkcja *_fstat_r* służąca do zwracania informacji o otwartym pliku
```
int _fstat_r(struct _reent *r,int file,struct stat *st)
{
  /*  Always set as character device.                */
  st->st_mode = S_IFCHR;
  return 0;
}
```

List. 20. Funkcja wykrywająca terminal
```
int isatty(int file)
{
  return 1;
}
```

da się więc bezpośrednio przygotować uniwersalnej biblioteki *stdio*, która pasowałaby do każdego mikrokontrolera. Aby umożliwić działanie tej biblioteki musimy wraz z programem przygotować pewien zestaw funkcji umożliwiających wysyłanie i odbieranie znaków, które są zależne od typu mikrokontrolera. Dla zapewnienia minimalnej funkcjonalności musimy zapewnić interfejs do następujących funkcji niskopoziomowych:

Funkcje te służą do wykonywania operacji na plikach, gdzie uchwyty do plików o wartości 0, 1, 2 są w systemach operacyjnych szczególnymi plikami, czyli standardowym wejściem i wyjściem. Ponieważ w naszym przypadku nie pracujemy pod kontrolą systemu operacyjnego i nie wykorzystujemy plików, funkcje te są bardzo proste i sprowadzają się jedynie do wysyłania i odbierania znaków z portu szeregowego. Do zapisywania danych do terminala wykorzystywana jest funkcja *_write_r,* którą przedstawiono na **list. 15**.

Działanie tej funkcji sprowadza się do wysłania wszystkich znaków przekazanych do funkcji jako argument prosto do terminala za pomocą funkcji *Uart0PutChar()*. Kolejną funkcją umożliwiającą współpracę z terminalem jest funkcja *_read_r* (**list. 16**), która służy do odczytywania znaków z terminala i przekazywania do wyższych funkcji biblioteki.

Działanie tej funkcji sprowadza się do pobierania poszczególnych znaków z terminala do momentu napotkania znaku końca linii, a następnie zwraca ona liczbę odczytanych bajtów. To są właśnie dwie główne funkcje, które odpowiadają za współpracę z terminalem. Oprócz tego musimy zdefiniować kilka funkcji pomocniczych, które nie będą nic robić oprócz zwracania odpowiednich parametrów. Funkcja *_close_r* (**list. 17**) służy do zamykania plików, a ponieważ my pracujemy tylko z terminalem i nie można go zamknąć, dlatego funkcja zwraca wartość 0. Kolejną funkcją jest funkcja *_lseek_r* umożliwiająca zmianę pozycji w pliku, ponieważ terminal nie posiada żadnej pozycji, więc funkcja ta zwraca zawsze wartość zerową bez wykonywania żadnych czynności.

Funkcja *_fstat_r* (**list. 19**) służy do zwracania informacji o otwartym pliku, w naszym przypadku zawsze zwracamy informację, że jest to urządzenie znakowe, natomiast funkcja *isatty* (**list. 20**) powinna zwracać wartość *prawda* w przypadku, gdy urządzenie jest terminalem, w naszym przypadku również powinna zwracać ona wartość prawdziwą.

To już są wszystkie funkcje niezbędne do tego, aby biblioteka standardowa umożliwiała wyświetlanie i odbieranie znaków do terminala za pomocą standardowych mechanizmów znanych z komputerów PC. W przypadku, gdybyśmy chcieli obsługiwać zapisywanie i odczytywanie plików za pomocą funkcji biblioteki standardowej, na przykład na karcie pamięci MMC, wówczas wspomniane wcześniej funkcję należy znacząco rozbudować, o dodatkowe mechanizmy. Konieczność zadeklarowania dodatkowych niskopoziomowych funkcji zapewnia bibliotece uniwersalność i niezależność od platformy systemowej i sprzętowej.

Zakończenie

Podczas kursu zapoznaliśmy się z możliwościami analogowymi mikrokontrolerów LPC213x/214x, które w sposób znaczący nie wyróżniają się niczym szczególnym na tle innych podobnych układów i należą do „klasyki" w tej klasie. Jednak rozdzielczość i dokładność przetworników A/C wbudowanych w mikrokontroler jest wystarczająca dla większości popularnych aplikacji, i tylko w przypadku bardziej zaawansowanych aplikacji pomiarowych użytkownik będzie zmuszony do zastosowania innych mikrokontrolerów (na przykład ADCU7000 również z rdzeniem ARM), lub użycia zewnętrznych przetworników. Wbudowany w mikrokontroler przetwornik C/A, umożliwia przetwarzanie wielkości cyfrowych na analogowe, co możemy wykorzystać na przykład do odtwarzania plików dźwiękowych.

To jest już ostatni odcinek tego kursu. Mam nadzieję, że przedstawione zagadnienia, informacje i przykłady pozwoliły Czytelnikom zapoznać się z możliwościami mikrokontrolerów LPC21xx. Cykl ten miał także pokazać, że mikrokontrolery z rdzeniem ARM nie są takie straszne, a posługiwanie się nimi wcale nie musi być dużo trudniejsze od programowania 8–bitowych mikrokontrolerów na przykład AVR–ów.

Niestety ograniczone łamy niniejszego kursu nie pozwoliły na przedstawienie wszystkich zagadnień, ale na podstawie przedstawionych materiałów użytkownik we własnym zakresie będzie mógł rozwinąć zagadnienia stosownie do swoich wymagań. W razie jakiś pytań wątpliwości oraz uwag na temat niniejszego kursu, czekam na kontakt.

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