Procesory Nios II w układach FPGA (5)

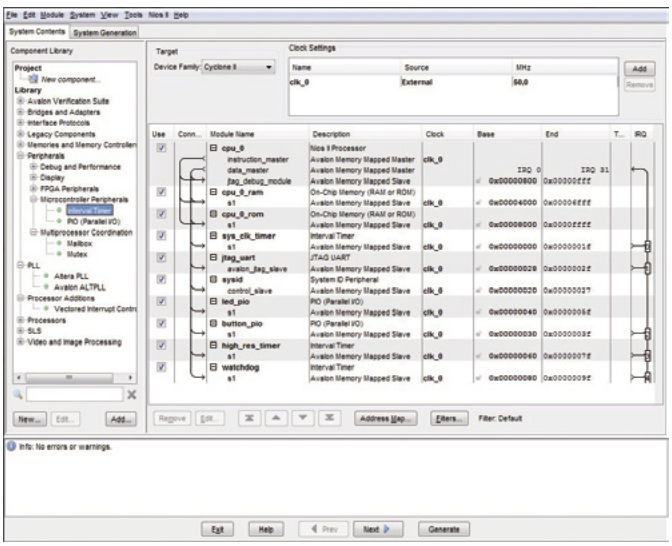
Obsługa timerów z biblioteki bloków IP

Jednym z powszechnych zadań systemów mikroprocesorowych są: cykliczne uruchamianie funkcji, odmierzanie czasu oraz pomiar odcinków czasu. W piątej i ostatniej części kursu o procesorach Nios II w układach FPGA przedstawiono sposób obsługi timerów z biblioteki bloków IP firmy Altera. Godny uwagi jest fakt, że w zależności od konfiguracji sprzętowej, bloki te mogą pełnić funkcje timera systemowego, prostego licznika czasu o dużej rozdzielczości a nawet watchdoga.

Firma Altera udostępnia wszechstronny blok IP Interval Timer. W bibliotece bloków IP programu SOPC Builder jest on umieszczony w zakładce *Peripherals -> Microcontroller Peripherals*. W zależności od wprowadzonych przez użytkownika ustawień konfiguracyjnych, może pełnić różne funkcje. Do głównych zadań bloku Interval Timer należą:

– System Clock – timer systemowy oraz cyklicznie uruchamiane alarmy,
– Timestamp Timer – licznik znaczników czasu,
– Watchdog.

Do systemu cyfrowego zaprojektowanego w poprzedniej części kursu należy dodać dwa bloki IP Interval Timer o nazwie *high_res_timer* (będzie to Timestamp Driver) oraz watchdog. Projekt tak zdefiniowanego systemu cyfrowego przedstawiono na **rysunku 46**.



Rys. 46. Projekt systemu cyfrowego z watchdogiem

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 10460, pass: 0646g3n0

System Clock i alarmy

Jeżeli w systemie cyfrowym z procesorem Nios II zostanie zdefiniowany timer System Clock, to programista uzyskuje dostęp do następujących funkcji z biblioteki HAL:

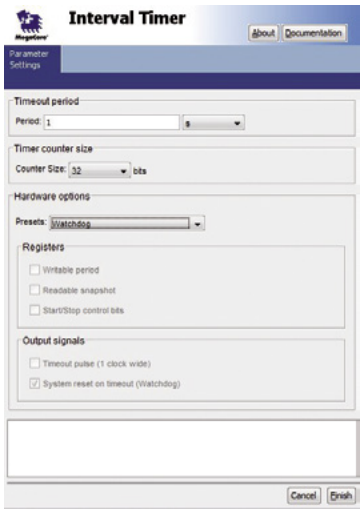
– `alt_ticks_per_second()` – funkcja zwraca częstotliwość timera systemowego w taktach na sekundę,
– `alt_nticks()` – zwraca liczbę taktów timera systemowego, która upłynęła od momentu uruchomienia systemu.

List. 7. Obsługa alarmów

```
#include "system.h" //opis systemu
#include "sys/alt_alarm.h" // alt_sysclk_init()
#include "alt_types.h" // typy altery
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"

// Alarm obsługujący diody led
alt_alarm led_alarm;
alt_u32 led;

alt_u32 led_alarm_func(void* kontekst){
  alt_u32 * p_led = 0;
  p_led = (alt_u32 *) kontekst;
  IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, ~(*p_led));
  (*p_led)++;
  return alt_ticks_per_second();
}

int main()
{
  alt_alarm_start(
    &led_alarm,
    alt_ticks_per_second(),
    led_alarm_func,
    (alt_u32 *) &led);

  /* KONIEC PROGRAMU */
  while (1);

  return 0;
}
```

Dostępne stają się również funkcje obsługi alarmów:
`alt_alarm_start()`
`alt_alarm_stop()`

Przykład użycia funkcji timera systemowego oraz alarmów przedstawiono na **listingu 7**. Alarmy są zdefiniowanymi przez użytkownika funkcjami, wywoływanymi w określonym odstępie czasu. Funkcja `alt_alarm_start` rejestruje nowy alarm oraz go uaktywnia. Jako parametry przyjmuje:



Rys. 47. Konfiguracja watchdoga

- `alt_alarm* alarm` – wskaźnik do struktury opisującej parametry alarmu,
- `alt_u32 nticks` – liczba taktów między kolejnymi uruchomieniami alarmu,
- `alt_u32 (*callback)(void* context)` – wskaźnik do funkcji użytkownika,
- `void* context` – wskaźnik do danych użytkownika przekazywanych do funkcji callback.

Funkcja wywoływana w alarmie powinna zwracać liczbę taktów, po których ma być ponownie uruchomiona lub zero, jeżeli ma być uruchomiona tylko raz. Aby wyłączyć wcześniej zdefiniowany alarm, należy wywołać funkcję `alt_alarm_stop`, podając wskaźnik do struktury alt_alarm jako parametr.

Watchdog

Watchdog jest jednym z elementów systemu mikroprocesorowego, który wpływa na zwiększenie niezawodności projektowanego urządzenia. Układ watchdog jest dostępny w praktycznie wszystkich mikrokontrolerach. Implementacja bloku IP firmy Altera umożliwia dodanie w łatwy sposób takiego układu do systemu cyfrowego z procesorem Nios II. Konfigurację bloku IP timera działającego jako watchdog pokazano na **rysunku 47**. Aby włączyć układ watchdoga, program procesora Nios II musi wpisać „1” do bitu `START` w rejestrze kontrolnym timera. Po włączeniu watchdoga nie może on zostać zatrzymany. Gdy zliczana przez watchdog wartość dojdzie do zera, zostanie ustawiony sygnał żądania zerowania na wyjściu `reset_request`. Aby temu zapobiec, procesor musi przed upływem czasu zdefiniowanego w programie *SOPC Builder* wpisać dowolną wartość do rejestru okresu `periodh` lub `periodl`. Wartość ta jest ignorowana przez układ, nie ma więc możliwości zmiany okresu watchdoga. Korzystając z tego bloku IP, należy więc rozważnie dobrać okres watchdoga na etapie projektowania systemu w programie *SOPC Builder*.

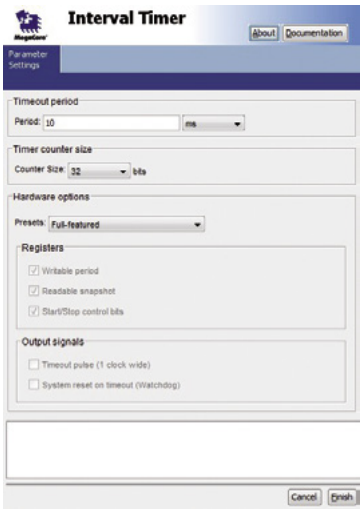
Przykładowy program obsługi układu watchdog został przedstawiony na **listingu 2**. W programie zostały zdefiniowane dwie funkcje:

- `watchdog_start()` – włącza układ watchdog,
- `watchdog_reset()` – restartuje timer watchdoga.

Obie korzystają z makr zdefiniowanych w plikach nagłówkowych `system.h` (adres bazowy bloku watchdog) oraz `altera_avalon_timer_regs`. W drugim pliku znajdują się makra dostępu do poszczególnych rejestrów bloku IP timera.

Timestamp Driver

Blok *IP timer* może być również użyty jako licznik znaczników czasu (*timestamp timer*). Układ tego typu może posłużyć do wyznaczania, jaki czas upłynął między dwoma zdarzeniami w programie. Aby timer dział w trybie licznika znaczników czasu, należy go tak skonfigurować, żeby procesor mógł zmieniać zawartość rejestru `period` i nie może być użyty jednocześnie jako timer systemowy. Najprościej jest dodać do systemu cyfrowego drugi blok timera i wybrać zdefiniowane przez producenta ustawienia *full-featured* (**rysunek 48**). Aby oprogramowanie warstwy HAL mogło korzystać z tak skonfigurowanego timera, należy w edytorze projektu BSP wybrać z listy odpowiedni timer w zakładce *Settings -> Common -> hal -> timestamp_timer*.



Rys. 48. Konfiguracja timera timestamp

W bibliotece HAL znajdują się funkcje obsługi timera skonfigurowanego jako *timestamp_timer*. Są to:

- `alt_timestamp_start()` – służy do włączenia lub wyzerowania uruchomionego timera,
- `alt_timestamp()` – zwraca chwilową zawartość licznika timera,
- `alt_timestamp_freq()` – zwraca częstotliwość z jaką pracuje licznik czasu.

Na **listingu 3** umieszczono przykładowy program wyznaczający czas wykonywania wybranych funkcji programu. Takie zastosowanie licznika umożliwia prostą ocenę wydajności systemu lub wybranych funkcji. Po uruchomieniu programu z **listingu 3** w konsoli Nios II EDS powinien pojawić się komunikat:

```
Start programu!
ooooooooooooooooooo
Wyniki pomiarów:
Start     : 0       : 0 us
```

List. 2. Obsługa układu watchdog

```
#include "system.h" //opis systemu
#include "sys/alt_stdio.h" // alt_putstr()
#include "alt_types.h" // typy altery
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include <unistd.h> // sleep
// uruchomienie układu watchdog
void watchdog_start(){
  IOWR_ALTERA_AVALON_TIMER_CONTROL(
    WATCHDOG_BASE,
    (1 << ALTERA_AVALON_TIMER_CONTROL_START_OFST)
  );
}
// zerowanie licznika watchdog
void watchdog_reset(){
  IOWR_ALTERA_AVALON_TIMER_PERIODH(
    WATCHDOG_BASE,
    0x77);
}
int main()
{
  alt_u32 led = 0;
  alt_putstr("\nStart programu!\n");
  watchdog_start();
  while(1){
    led ++;
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, ~led);
    alt_putstr("o");
    usleep(100000); // odczekaj 100 ms
    if(led < 100){
      watchdog_reset();
    }
  }
  /* KONIEC PROGRAMU */
  while (1);
  return 0;
}
```

List. 3. Przykładowy program oceny wydajności

```
#include "system.h" //opis systemu
#include "alt_types.h" // typy altery
#include "sys7alt_stdio.h" // alt_putchar(), alt_putstr()
#include "sys/alt_timestamp.h" // funkcje timera
timestamp
#include "altera_avalon_timer_regs.h"
// Testowa funkcja 1
void funkcja1(){
  volatile alt_u32 i;
  for(i = 0; i < 200; i++){
    i = i;
  }
}
// Testowa funkcja 1
void funkcja2(){
  alt_u32 i;
  for(i =0; i < 20; i++){
    alt_putchar('o');
  }
}
// Funkcja do wypisywania liczb całkowitych
void wypisz_liczbe(alt_u32 liczba){
  alt_u32 cyfry[32];
  alt_u32 i=0;
  // wyznacz kolejne cyfry liczby
  do{
    cyfry[i] = liczba%10;
    liczba /= 10;
    i++;
  } while(liczba > 0);
  // wypisz liczby do konsoli
  while(i > 0){
    alt_putchar(cyfry[i-1] + '0');
    i--;
  }
}
// Pomocnicza funkcja do wypisywania wyników pomiarów
void wypisz_wynik(char * opis, alt_u32 wynik){
  alt_putstr("\n");
  alt_putstr(opis);
  alt_putstr("\t: ");
  wypisz_liczbe(wynik);
  alt_putstr("\t: ");
  wypisz_liczbe(((alt_u64)wynik * 1000000) / alt_
timestamp_freq());
  alt_putstr(" us");
}
int main() {
  alt_u32 start, czas1, stop;
  alt_putstr("\nStart programu!\n");
  // Wykonanie pomiarów
  start = alt_timestamp_start();  // właczenie timera
  funkcja1();
  czas1 = alt_timestamp();    // pobranie próbki czasu
  funkcja2();
  stop = alt_timestamp();    // pobranie próbki czasu
  // Wypisanie wyników
  alt_putstr("\nWyniki pomiarow:");
  wypisz_wynik("Start", start);
  wypisz_wynik("Funkcja1", czas1 - start);
  wypisz_wynik("Funkcja2", stop - czas1);
  /* KONIEC PROGRAMU */
  while (1);
  return 0;
}
```


```
Funkcja1   : 11245    : 224 us
Funkcja2   : 5249     : 104 us
```

Podsumowanie

W kursie o procesorach Nios II przedstawiono opis niezwykle interesujących procesorów typu *softcore* firmy Altera. W kolejnych częściach kursu opisano sposób implementacji tych procesorów w układach FPGA oraz sposób tworzenia zintegrowanych systemów cyfrowych w programie Quartus II oraz SOPC Builder. W kursie znalazł się opis podstawowych zadań programistycznych, od konfiguracji projektów, poprzez tworzenie aplikacji, aż po debugowanie kodu programu w środowisku Nios II EDS. Zaprezentowane przykłady programów powinny stanowić dobry początek dla programistów chcących bliżej zapoznać się z procesorami Nios II.

Maciej Gołaszewski
golaszewski.maciej@gmail.com