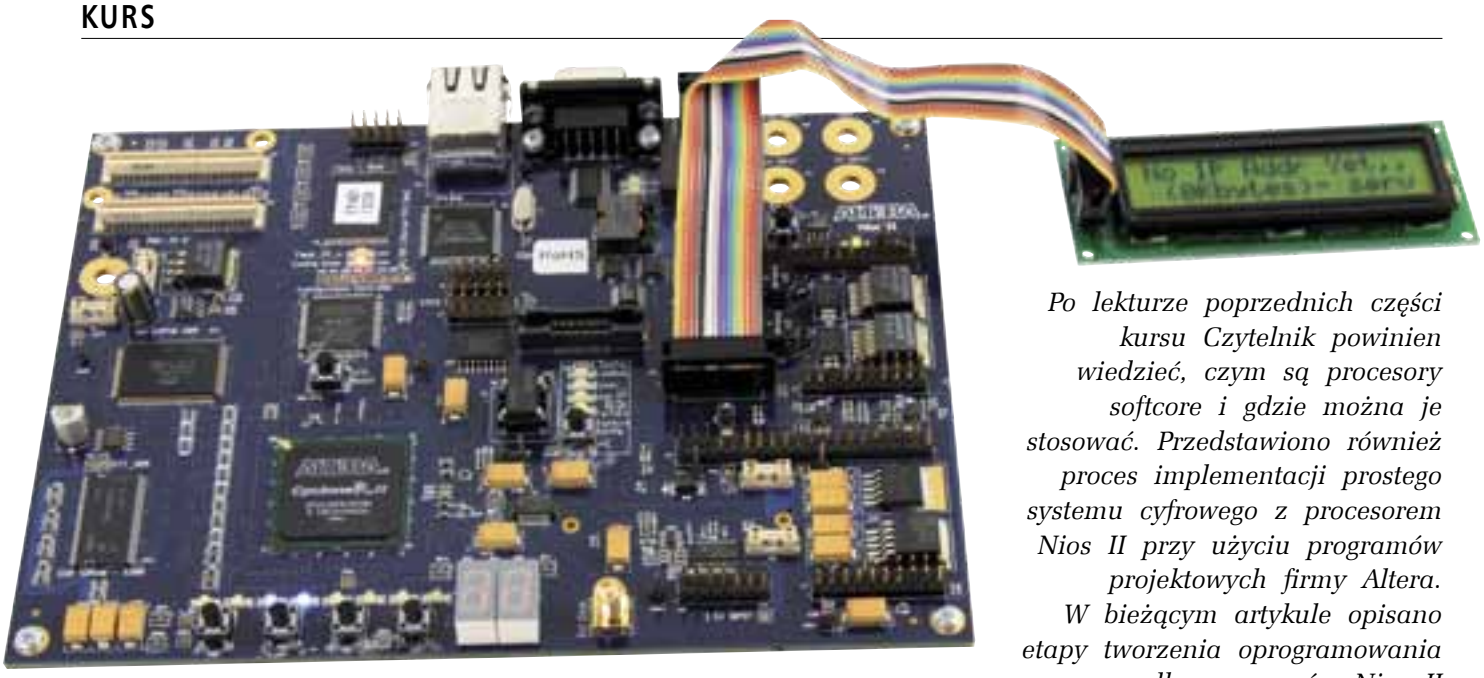


Po lekturze poprzednich części kursu Czytelnik powinien wiedzieć, czym są procesory softcore i gdzie można je stosować. Przedstawiono również proces implementacji prostego systemu cyfrowego z procesorem Nios II przy użyciu programów projektowych firmy Altera. W bieżącym artykule opisano etapy tworzenia oprogramowania dla procesorów Nios II w środowisku programistycznym Nios II SBT na przykładzie obsługi portów GPIO.

Procesory Nios II w układach FPGA (3)

Obsługa portów GPIO

Część 3 kursu jest wprowadzeniem do omówienia programowej obsługi podstawowych bloków peryferyjnych mikroprocesora Nios II dostępnych w programie SOPC Builder. W artykule omówiono obsługę portów GPIO. W tym celu zostanie zmodyfikowany projekt cyfrowego systemu utworzonego w drugiej części kursu.

Przydatną funkcją środowiska projektowego Quartus II jest kopiowanie projektów. Umożliwia to tworzenie podobnych programów bez żmudnego przepisywania części kodu.

Przed dodaniem do systemu portów GPIO należy utworzyć kopię projektu `WitajSwiecie` o nazwie „GPIO”. W tym celu należy go otworzyć, a następnie z menu *Project* wybrać *Copy Project*, wpisać nazwę i wskazać miejsce na dysku, gdzie zostaną zapisane pliki (na przykład folder *porty_gpio* – **rysunek 17**).

Rysunek 17. Kopiowanie projektu w programie Quartus II

Porty I/O procesora Nios II

Do tak utworzonego projektu dodane zostaną porty GPIO. Firma Altera oferuje blok IP do obsługi portów GPIO o nazwie PIO (*Parallel IO*). W bibliotece bloków IP programu do tworzenia systemów cyfrowych SOPC builder umieszczono go w zakładce *Peripherals* > *Microcontroller Peripherals* (**rysunek 18**). Jest to uniwersalny blok IP, który w zależności od konfiguracji, może pracować jako: port wyjściowy (*output*), wejściowy (*input*) lub dwukierunkowy (*bidirectional*).

Rysunek 18. Umiejscowienie bloku IP PIO w bibliotece programu SOPC Builder

Na płytce zestawu ewaluacyjnego są dostępne cztery przyciski oraz osiem diod LED. Do projektu należy

Dodatkowe materiały na CD i FTP:
ftp://ep.com.pl, user: 16195, pass: 4k17u606

dodać dwukrotnie blok IP z ustawieniami konfiguracyjnymi pokazanymi na **rysunku 19** i **rysunku 20**, odpowiednio, do obsługi diod LED i przycisków. Jak można zauważyć, blok PIO umożliwia tworzenie

Rysunek 19. Ustawienia konfiguracyjne bloków PIO dla portu wyjściowego do sterowania diod LED

portów o maksymalnej szerokości wynoszącej 32 bity. Można zauważyć, że projektant systemu cyfrowego w układzie FPGA może dodać wiele takich portów i jest ograniczony tylko liczbą dostępnych wyprowadzeń układu programowalnego oraz przestrzenią adresową procesora Nios II. Porty mogą mieć nazwę zgodną z pełnioną funkcją, co ułatwia pracę programiście.

Na **rysunku 21** przedstawiono projekt opisywanego systemu, w którym utworzono dwa porty: `led_pio` do sterowania diodami LED i `button_pio` do odczytywania stanu przycisków. Po dodaniu portów GPIO należy wygenerować system.

Po zamknięciu programu SOPC Builder program Quartus II wyświetli zapytanie o aktualizację symbolu systemu. Po odświeżeniu symbolu graficznego należy dodać wyprowadzenia do końcówek układu FPGA oraz przypisać odpowiednie końcówki do linii sygnałowych. Poprawiony plik edytora schematów programu Quartus II zamieszczono na **rysunku 22**. W dymkach widocznych przy symbolach wyprowadzeń IO program Quartus wyświetla przypisane końcówki układu FPGA. Są one widoczne dopiero po kompilacji projektu.

Listing 1. Plik system.h z opisem systemu cyfrowego

```
#ifndef __SYSTEM_H_
#define __SYSTEM_H_

/* Include definitions from linker script generator */
#include "linker.h"


/*
 * CPU configuration
 *
 */

#define ALT_CPU_ARCHITECTURE "altera_nios2"
#define ALT_CPU_BIG_ENDIAN 0
#define ALT_CPU_BREAK_ADDR 0x820
#define ALT_CPU_CPU_FREQ 50000000u
#define ALT_CPU_CPU_ID_SIZE 1
#define ALT_CPU_CPU_ID_VALUE 0x0
#define ALT_CPU_CPU_IMPLEMENTATION "tiny"
#define ALT_CPU_DATA_ADDR_WIDTH 16
#define ALT_CPU_DCACHE_LINE_SIZE 0
#define ALT_CPU_DCACHE_LINE_SIZE_LOG2 0
#define ALT_CPU_DCACHE_SIZE 0
#define ALT_CPU_EXCEPTION_ADDR 0x8020
#define ALT_CPU_FLUSHDA_SUPPORTED
#define ALT_CPU_FREQ 50000000
#define ALT_CPU_HARDWARE_DIVIDE_PRESENT 0
#define ALT_CPU_HARDWARE_MULTIPLY_PRESENT 0
#define ALT_CPU_HARDWARE_MULX_PRESENT 0
#define ALT_CPU_HAS_DEBUG_CORE 1
#define ALT_CPU_HAS_DEBUG_STUB
#define ALT_CPU_HAS_JMPI_INSTRUCTION
#define ALT_CPU_ICACHE_LINE_SIZE 0
#define ALT_CPU_ICACHE_LINE_SIZE_LOG2 0
#define ALT_CPU_ICACHE_SIZE 0
#define ALT_CPU_INST_ADDR_WIDTH 16
#define ALT_CPU_NAME "cpu_0"
#define ALT_CPU_RESET_ADDR 0x8000
...
/*
 * Define for each module class mastered by the CPU
 *
 */

#define __ALTERA_AVALON_JTAG_UART
#define __ALTERA_AVALON_ONCHIP_MEMORY2
#define __ALTERA_AVALON_PIO
#define __ALTERA_AVALON_SYSID
#define __ALTERA_AVALON_TIMER
#define __ALTERA_NIOS2
...
/*
 * led_pio configuration
 *
 */

#define ALT_MODULE_CLASS_led_pio altera_avalon_pio
#define LED_PIO_BASE 0x40
#define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 1
#define LED_PIO_CAPTURE 0
#define LED_PIO_DATA_WIDTH 8
#define LED_PIO_DO_TEST_BENCH_WIRING 0
#define LED_PIO_DRIVEN_SIM_VALUE 0x0
#define LED_PIO_EDGE_TYPE "NONE"
#define LED_PIO_FREQ 50000000u
#define LED_PIO_HAS_IN 0
#define LED_PIO_HAS_OUT 1
#define LED_PIO_HAS_TRI 0
#define LED_PIO_IRQ -1
#define LED_PIO_IRQ_INTERRUPT_CONTROLLER_ID -1
#define LED_PIO_IRQ_TYPE "NONE"
#define LED_PIO_NAME "/dev/led_pio"
#define LED_PIO_RESET_VALUE 0x0
#define LED_PIO_SPAN 32
#define LED_PIO_TYPE "altera_avalon_pio"
```

Środowisko projektowe Nios II EDS

Do tworzenia programów dla procesora Nios II służy środowisko programistyczne Nios II EDS (*Embedded Developemnt Suite*). Umożliwia ono zarządzanie projektami oprogramowania dla różnych systemów cyfrowych, kompilację kodu programu oraz debugowanie. Środowisku Nios II EDS umożliwia korzystanie z dwóch różnych procesów projektowych przy tworzeniu programów:

– Nios II Software Build Tools (SBT),

– Nios II IDE.

Nios II IDE był używany we wcześniejszych wersjach środowiska projektowego. Nios II SBT jest aktualnie wspieranym dla procesorów Nios II i domyślnie uruchamiany przy otwieraniu środowiska projektowego. Oba procesy projektowe są zbudowane na bazie środowiska Eclipse, korzystają z tych samych programów narzędziowych przy tworzeniu kodu wynikowego, a programy i biblioteki utworzone przy użyciu jednego procesu mogą być używane przez drugi. Jednak różnią się one sposobem tworzenia pliku makefile z poleceniami dla kompilatora, przez co nie mogą być między sobą wymieniane. Dodatkowo, systemy cyfrowe utworzone w programie SOPC Builder w wersji 7.0 lub niższej nie mogą być używane w procesie projektowym Nios II SBT.

W programie Nios II SBT można tworzyć trzy rodzaje projektów: aplikacje, BSP oraz biblioteki użytkownika. W projekcie

Rysunek 20. Ustawienia konfiguracyjne bloków PIO dla portu wejściowego do odczytywania stanu przycisków

aplikacji jest przechowywany kod źródłowy programu dla procesora Nios II. Projekt BSP jest biblioteką programową z opisem warstwy sprzętowej systemu cyfrowego, dla którego jest tworzona aplikacja. Zawiera on sterowniki programowe i niezbędne pliki nagłówkowe dla jednego procesora Nios II w danym systemie cyfrowym. Jeżeli w systemie cyfrowym są dwa lub więcej procesory Nios II, to dla każdego z nich należy utworzyć odrębny projekt BSP opcjonalnie zawierający dodatkowe biblioteki programowe oferowane przez partnerów firmy Altera, takie jak na przykład system operacyjny µC/OS-II (bez licencji). Projekt bibliotek programowych służy do gromadzenia często używanych w różnych projektach funkcji, na przykład biblioteki funkcji graficznych.

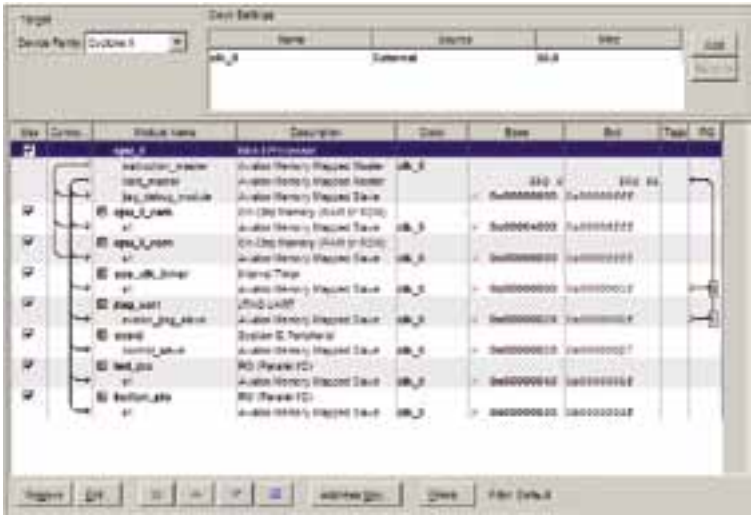
Projekt BSP

Jak wspomniano, projekt BSP zawiera definicję środowiska uruchomieniowego procesora Nios II oraz bibliotekę z funkcjami obsługi peryferiów dostępnych w systemie cyfrowym, dla którego jest on tworzony. Projekt BSP izoluje więc projekt aplikacji od definicji zasobów sprzętowych, takich jak: mapa pamięci procesora, peryferia czy konfiguracja procesora. W programie Nios II SBT można utworzyć dwa rodzaje projektów BSP: korzystających z warstwy abstrakcji sprzętu Altera HAL lub pisanych dla systemu operacyjnego µC/OS-II. System operacyjny µC/OS-II oraz programy pisane dla niego korzystają z biblioteki programowej Altera HAL (porównaj **rysunek 23 a** i **b**).

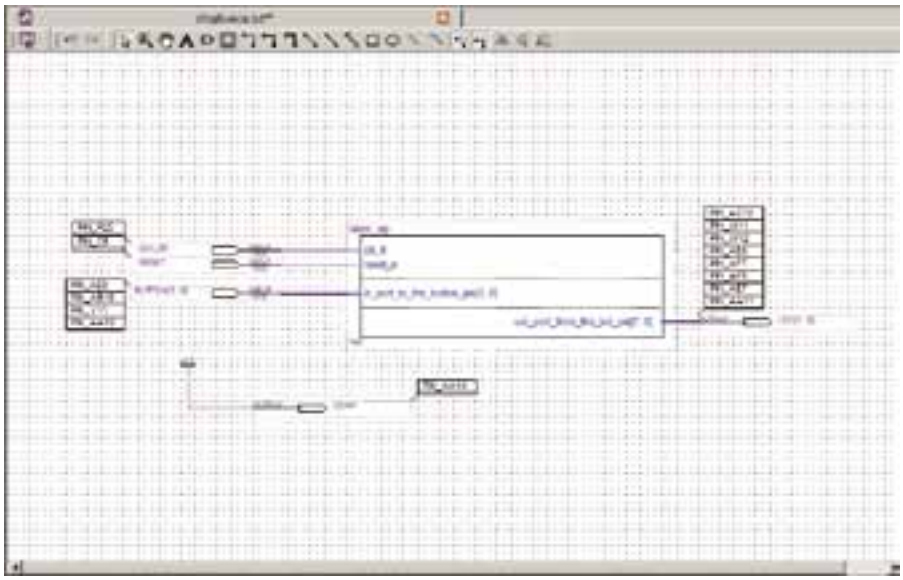
Aby utworzyć nowy projekt BSP należy w programie Nios II SBT z menu *File* wybrać opcję *New –> Nios II Board Support Package* (**rysunek 24**). Następnie wprowadzić nazwę projektu, wskazać plik **.sopc* z opisem systemu cyfrowego oraz wybrać typ projektu. W dalszej części kursu będą tworzone projekty bez systemu operacyjnego, dlatego należy wybrać opcję *Altera HAL*. Na **rysunku 25** przedstawiono widok okna kreatora tworzenia projektu BSP.

Po rozwinięciu drzewa utworzonego projektu BSP (**rysunek 26**) można zauważyć, że program Nios II SBT utworzył m.in. pliki nagłówkowe, definicje linkera dla procesora Nios II oraz dołączył biblioteki ze sterownikami układów peryferyjnych (folder *drivers*) i bibliotekę *Altera HAL* (folder HAL). Wśród utworzonych plików znajduje się `summary.html`, który można otworzyć w przeglądarce WWW. Zawiera on podstawowe informacje o procesorze i systemie cyfrowym (**rysunek 27**).

Jednym z najważniejszych plików opisujących peryferia i konfigurację sprzętową procesora Nios II jest plik nagłówkowy

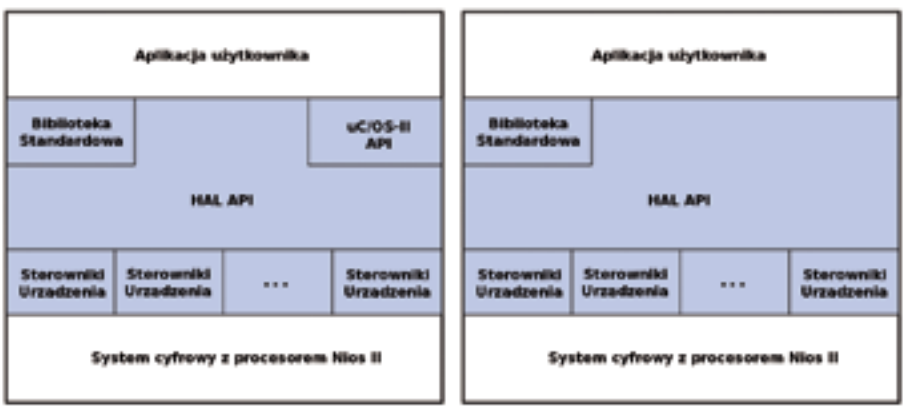


Rysunek 21. Projekt systemu cyfrowego z portami GPIO



Rysunek 22. Plik edytora schematowego z projektowanym systemem cyfrowym

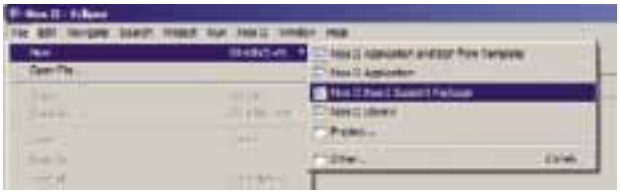
`system.h` znajdujący się w głównym katalogu projektu BSP. Zawiera on makrodefinicje używane przez funkcje biblioteki HAL. Na **listingu 1** zamieszczono fragment tego pliku, z którego można odczytać m.in.: częstotliwość sygnału zegarowego (`ALT_CPU_FREQ – 50 MHz`) i wersję procesora (`ALT_CPU_CPU_IMPLEMENTATION – Nios II/e`). W pliku tym są również zdefiniowane dostępne peryferia mikroprocesora. Każde z urządzeń peryferyjnych ma podane ustawienia konfiguracyjne w pliku `system.h`. Na listingu 1 przedstawiono ustawienie konfiguracyjne tylko dla portu GPIO, do którego dołączono diody LED zestawu ewaluacyjnego. Nazwy definicji zaczynają się od nazwy modułu zdefiniowanej w programie SOPC Builder. Definicje te



Rysunek 23. Porównanie programów dla procesora Nios II: a) tylko z biblioteką Altera HAL, b) z systemem operacyjnym µC/OS-II

Listing 2. Plik sterowników programowych bloku PIO

```
#ifndef __ALTERA_AVALON_PIO_REGS_H__
#define __ALTERA_AVALON_PIO_REGS_H__

#include <io.h>

#define IOADDR_ALTERA_AVALON_PIO_DATA(base)            __IO_CALC_ADDRESS_
NATIVE(base, 0)
#define IORD_ALTERA_AVALON_PIO_DATA(base)              IORD(base, 0)
#define IOWR_ALTERA_AVALON_PIO_DATA(base, data)        IOWR(base, 0, data)

#define IOADDR_ALTERA_AVALON_PIO_DIRECTION(base)       __IO_CALC_ADDRESS_
NATIVE(base, 1)
#define IORD_ALTERA_AVALON_PIO_DIRECTION(base)         IORD(base, 1)
#define IOWR_ALTERA_AVALON_PIO_DIRECTION(base, data)   IOWR(base, 1, data)

#define IOADDR_ALTERA_AVALON_PIO_IRQ_MASK(base)        __IO_CALC_ADDRESS_
NATIVE(base, 2)
#define IORD_ALTERA_AVALON_PIO_IRQ_MASK(base)          IORD(base, 2)
#define IOWR_ALTERA_AVALON_PIO_IRQ_MASK(base, data)    IOWR(base, 2, data)

#define IOADDR_ALTERA_AVALON_PIO_EDGE_CAP(base)        __IO_CALC_ADDRESS_
NATIVE(base, 3)
#define IORD_ALTERA_AVALON_PIO_EDGE_CAP(base)          IORD(base, 3)
#define IOWR_ALTERA_AVALON_PIO_EDGE_CAP(base, data)    IOWR(base, 3, data)

#define IOADDR_ALTERA_AVALON_PIO_SET_BIT(base)        __IO_CALC_ADDRESS_
NATIVE(base, 4)
#define IORD_ALTERA_AVALON_PIO_SET_BITS(base)          IORD(base, 4)
#define IOWR_ALTERA_AVALON_PIO_SET_BITS(base, data)    IOWR(base, 4, data)

#define IOADDR_ALTERA_AVALON_PIO_CLEAR_BITS(base)       __IO_CALC_ADDRESS_
NATIVE(base, 5)
#define IORD_ALTERA_AVALON_PIO_CLEAR_BITS(base)         IORD(base, 5)
#define IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(base, data)   IOWR(base, 5, data)

/* Defintions for direction-register operation with bi-directional PIOs */
#define ALTERA_AVALON_PIO_DIRECTION_INPUT  0
#define ALTERA_AVALON_PIO_DIRECTION_OUTPUT 1

#endif /* __ALTERA_AVALON_PIO_REGS_H__ */
```



Rysunek 24. Tworzenie nowego projektu BSP

są używane przy korzystaniu ze sterowników programowych.

Nie należy samodzielnie modyfikować plików projektu wygenerowanych przez program Nios II SBT, gdyż błędne wpisy mogą spowodować złe działanie biblioteki programowej Altera HAL lub uniemożliwić uruchomienie programu. Do modyfikacji ustawień projektu służy opcja *Nios II –> BSP editor* z menu kontekstowego projektu (wywoływane prawym przyciskiem myszy). *BSP editor* umożliwia m.in. zmianę sposobu kompilacji, włączenie nieużywanych funkcji, zmianę stopnia optymalizacji kompilatora, czy wybór standardowego urządzenia wyjściowego, o ile w systemie dostępne są peryferia tego typu (np. JTAG_UART, wyświetlacz znakowy LCD czy UART). Dla prostych aplikacji, które nie korzystają na przykład z funkcji standardowego wyjścia czy dostępu do pamięci Flash, można w zakładce *Common* bezpiecznie zaznaczyć opcję użycia zredukowanych sterowników urządzeń (*enable_reduced_device_drivers*) i zoptymalizowanej biblioteki języka C (*enable_small_C_library*). W celu dalszego zmniejszenia objętości kodu wynikowego programu można odznaczyć wsparcie dla języka C++ (*enable_c_plus_plus*) w zakładce *Advanced*. Po wprowadzeniu zmian należy nacisnąć przycisk *Generate* i wygenerować pliki z ustawieniami. Omówione powyżej ustawienia pokazano na **rysunku 28**.

Rysunek 25. Kreator nowego projektu BSP dla procesora Nios II

Listing 3. Szablon programu głównego dla procesora Nios II z definicjami podstawowych bibliotek

```
#include "system.h"
#include "alt_types.h"

int main() {
   return 0;
}
```

Rysunek 26. Kreator projektu BSP

Dla każdego układu peryferyjnego dołączane są pliki nagłówkowe z definicjami stałych oraz z funkcjami obsługi. Przykładowy plik nagłówkowy sterowników portu GPIO przedstawiono na **listingu 2**. Są to makrodefinicje dostępu do rejestrów por-

Rysunek 27. Zawartość wygenerowanego pliku summary.html

tu GPIO. Korzystają one z podstawowych funkcji do odczytu (IORD) i zapisu danych (IOWR) pod adresem względnym w stosunku do adresu bazowego (parametr base) w przestrzeni adresowej mikroprocesora Nios II. Kolejne makra służą do odczytywania lub zapisywania wartości rejestrów konfiguracyjnych bloku PIO:

- PIO_DATA – rejestr danych portu GPIO. Przy zapisie ustawia wartości wyjściowe portu, przy odczycie zwraca poziomy logiczne na wejściach portu.
- PIO_DIRECTION – ustawienia kierunku poszczególnych bitów portu. Wartość 1 oznacza wyjście a wartość 0 wejście.
- PIO_IRQ_MASK – rejestr maskowania generowania przerwania dla wejść,
- PIO_EDGE_CAP – rejestr wykrywania wystąpienia zbocza narastającego lub opadającego na końcówkach. Wartość 1 oznacza wystąpienie zbocza opadającego lub narastającego w zależności od ustawień zdefiniowanych na etapie konfiguracji bloku PIO w programie SOPC Builder.
- PIO_SET_BITS i PIO_CLEAR_BITS służą do ustawiania lub zerowania indywidualnych bitów w rejestrze wyjściowym.

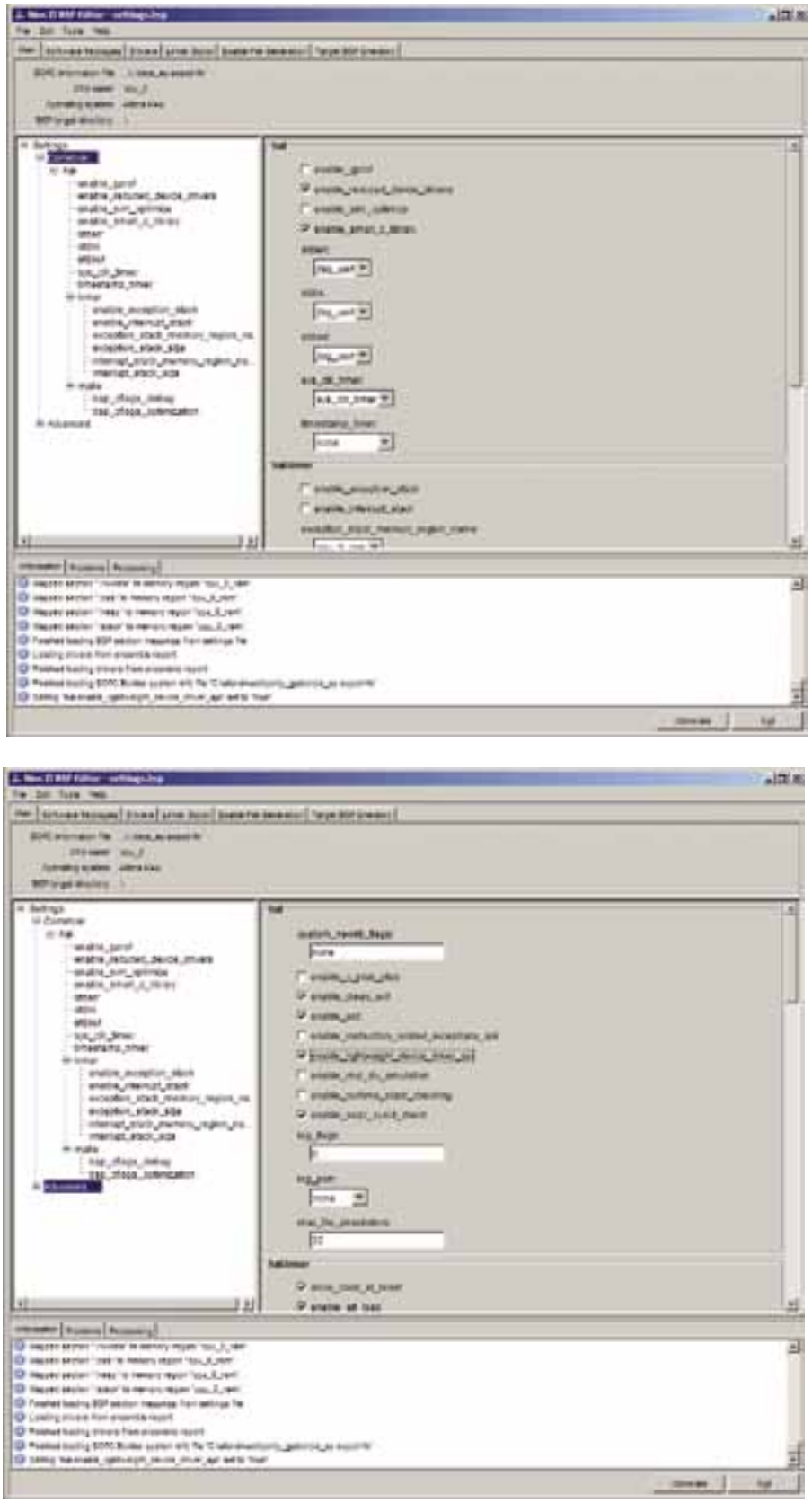
Poszczególne rejestry są dostępne wyłącznie po zaznaczeniu odpowiednich opcji w kreatorze bloku PIO w programie SOPC Buidler. Na przykład dla portu `led_pio` rejestry kierunkowy, przerwań będą niedostępne, gdyż został on skonfigurowany jako port wyjściowy.

Przykład użycia makro definicji do zapisu heksadecymalnej wartości 0x0F do portu `led_pio` wygląda następująco:

```
IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x0F);
```

LED_PIO_BASE jest adresem bazowym bloku IP w systemie cyfrowym. Wywołanie tego makra powoduje zapisanie w rejestrze zerowym wartości 0x0F.

Projekt BSP można skompilować niezależnie od projektu aplikacji. W razie potrzeby, kompilacja skojarzonego projektu BSP jest uruchamiana przed kompilacją projektu aplikacji.



Rysunek 28. Ustawienia konfiguracyjne projektu BSP

Projekty Aplikacji

W celu utworzenia projektu aplikacji dla procesora Nios II należy z menu *File* wybrać opcję *New –> Nios II Application*. Na **rysunku 29** pokazano kreator dodawania nowego projektu aplikacji. W kreatorze należy podać jego nazwę oraz wskazać wcześniej utworzony projekt BSP.

Nowoutworzony projekt nie zawiera żadnych plików. Pliki źródłowe są do niego dodawane z menu *File* poleceniem *New –> Source File*. Do projektu aplikacji należy dodać plik `main.c`, w którym zostanie utworzona funkcja główną programu. Na **listingu 3** przedstawiono przykładową zawartość pliku `main.c`, który może posłużyć jako szablon dla nowych aplikacji. Oprócz głównej funkcji programu w pliku znajdują się definicje dołączenia plików nagłówkowych `system.h` (z projektu BSP) oraz `alt_types.h` z biblioteki Altera HAL. Plik `alt_types` definiuje typy danych używane przez bibliotekę HAL. Najczęściej stosowanymi będą dodatnie, stałoprzecinkowe zmienne 32-bitowe (*alt_32u*) lub ze znakiem (alt_32). Zmienne o innej liczbie bitów (8, 16 lub 64) zostały zdefiniowane analogicznie, np.: *alt_8u*.

Rysunek 29. Kreator nowego projektu aplikacji dla procesora Nios II

Ciekawą funkcjonalnością środowiska Eclipse i usprawnieniem procesu pisania programu jest podpowiadanie jego kodu. Funkcję tę uruchamia się skrótem klawiaturowym CTRL+spacja (**rysunek 30**). Po jej uruchomieniu środowisko Nios II SBT tworzy listę zmiennych, definicji preprocesora i funkcji, których nazwa rozpoczyna się od wprowadzonych znaków. Funkcjonalność ta jest szczególnie przydatna przy wprowadzaniu bardzo długich i trudnych do zapamiętania definicji, na przykład z pliku `system.h`. Podpowiadanie kodu programu wyświetla wyłącznie te nazwy, które są widoczne w danym pliku, na przykład poprzez dodanie odpowiedniego pliku nagłówkowego. Funkcja ta może nie działać prawidłowo po dodaniu pierwszego pliku do nowo utworzonego projektu. Aby program odpowiednio odczytał zależności bibliotek programowych, należy uruchomić kompilację projektu.

Na **listingu 4** przedstawiono prosty program, który w pętli *while()* inkrementuje zmienną *licznik*. Zanegowana wartość zmiennej jest wystawiana na wyprowadzenia portu *led_pio*. Do pliku *main.c* dołączono dwa pliki nagłówkowe: *altera_avalon_pio_regs.h* z podstawowymi definicjami rejestrów bloku PIO oraz *unistd.h*.

Biblioteka *unistd.h* zawiera funkcje API systemów operacyjnych budowanych w standardzie POSIX (m.in. Linux, Mac OS X). W implementacji biblioteki *unistd.h* dla procesorów Nios II dostępne są wszystkie funkcje z API POSIX, jednak niektóre z nich (np. *link()*, *unlink()*) zawsze zwracają kod błędu. W programie z listingu 4 użyto funkcji *usleep()*, która blokuje wykonywanie programu na liczbę milisekund podaną jako argument wywołania.

Na **listingu 5** przedstawiono zmodyfikowany program z listingu 4, w którym dodano obsługę przycisków. W programie przy każdym wywołaniu pętli *while()* jest odczytywany stan przycisków. Należy zwrócić uwagę, że w wywołaniu funkcji odczytu jest podawany adres bazowy bloku PIO `button_pio`, który zdefiniowano w pliku nagłówkowym `system.h`. Wciśnięcie przycisku powoduje zaświecenie się odpowiadającej mu diody dołączonej do wyprowadzenia led7...4.

Listing 4. Przykład obsługi portu GPIO

```
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>

int main() {
    alt_u32 licznik = 0;
    while(1){
        licznik++;
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, ~licznik);
        if(licznik >= 255){
            licznik = 0;
        }
        usleep(100000);
    }
    return 0;
}
```

Rysunek 30. Funkcja podpowiadania kodu w środowisku Eclipse

Listing 5. Przykład obsługi dwóch portów GPIO

```
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>

int main() {
    alt_u32 licznik = 0;
    alt_u32 przyciski = 0;
    alt_u32 led = 0;
    while(1){
        licznik++;
        przyciski = IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE);
        if(licznik >= 15){
            licznik = 0;
        }
        led = (0xF0 & (przyciski<<4) | (0x0F & licznik));
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, ~led);
        usleep(100000);
    }
    return 0;
}
```

Podsumowanie

W artykule przedstawiono proces tworzenia aplikacji dla „miękkich" procesorów Nios II. Ze względu na stosunkowo prostą i szybką możliwość zmiany parametrów cyfrowego systemu wewnątrz układów FPGA, oprogramowanie dla tych procesorów jest tworzone dwuetapowo. Rozdzielenie projektu sterowników programowych (BSP) od projektu aplikacji umożliwia na przykład testowanie aplikacji na różnych konfiguracjach sprzętowych, lub tworzenie kilku aplikacji, korzystających z jednego projektu BSP. W artykule omówiono podstawowe ustawienia konfiguracyjne projektów BSP oraz przedstawiono na przykładzie obsługi portu IO korzystanie z podstawowych funkcji biblioteki Altera HAL.

Maciej Gołaszewski, EP
maciej.golaszewski@ep.com.pl