Własny komponent Qsys, czyli jak wykonać własny IP Core

Narzędzie Altera Qsys umożliwia budowanie złożonych systemów poprzez łączenie komponentów takich jak procesory, kontrolery pamięci, porty wejścia/wyjścia, itp. Umożliwia także tworzenie własnych komponentów z wykorzystaniem języka Verilog oraz VHDL. W tym artykule opiszę, jak utworzyć i wykorzystać własny komponent do obsługi wyświetlacza LED, na przykładzie zestawu MAXimator oraz płytki MAXimator expander.

Wszystkie komponenty wybierane i podłączane w narzędziu Qsys mają zestandaryzowane interfejsy nazwane Avalon. W ich skład wchodzą następujące typy:

- Avalon Clock Interface – interfejs wejściowy lub wyjściowy sygnału zegarowego.
- Avalon Reset Interface – interfejs sygnału reset.
- Avalon Interrupt Interface – interfejs przerwań, wykorzystywany do informowania procesora o zdarzeniu, które powinien obsłużyć.
- Avalon Memory-Mapped Interface (Avalon MM) – interfejs we/wy mapowany w przestrzeni adresowej.
- Avalon Streaming Interface (Avalon ST) – interfejs obsługujący pakietowy, jednokierunkowy przepływ danych.
- Avalon Conduit Interface – interfejs służący do wyprowadzania sygnałów na zewnątrz systemu NIOS.

W projekcie użyjemy interfejsu Avalon MM oraz projektu z **EP 12/2016 Pierwsze kroki z FPGA (6) MAXimator jak Arduino**. Pierwszy krok to dostosowanie kodu VHDL do wymagań interfejsu Avalon MM (**rysunek 1**). Trzeba dodać kilka sygnałów: **resetn, write, chipselect, byteenable**, jak na **rysunku 2**.

Cały system zbudowany na MAXimatorze jest zerowany poziomem niskim. Jeśli **resetn** przyjmie wartość logiczną 0, wtedy następuje wyświetlenie zer na wyświetlaczu oraz zaświecenie kropek. W układzie wyświetlanie stanu sygnału *reset* nie za bardzo ma sens i jest zrobione tylko po to, aby można było zaobserwować ten sygnał.

Poziom wysoki sygnału ***write*** wraz z sygnałem ***chipselect*** aktywuje odczyt magistrali Avalon i dokonuje zapisu do wewnętrznego rejestru wyświetlacza liczby do wyświetlenia. W tym przykładzie szerokość magistrali jest 16-bitowa. Sygnał ***byteenable*** służy do wybierania, który bajt ma być odczytany z magistrali. Na przykład, jeśli ***byteenable*** przyjmie wartość binarną 0b11, to zostaną odczytane obydwa bajty. Jeśli przyjmie 0b01, to będzie odczytany tylko młodszy bajt. Jeśli przyjmie 0b10, to przesłany będzie starszy bajt. Dla magistrali 32-bitowej ***byteenable*** ma wtedy 4 bity, przy 64-bitowej ma 8 bitów, itd.

```
entity x7seg is
    port ( x : in  STD_LOGIC_VECTOR (15 downto 0);
           clk : in  STD_LOGIC;
           seg : out  STD_LOGIC_VECTOR (6 downto 0);
           dig : out  STD_LOGIC_VECTOR (3 downto 0));
end x7seg;
```

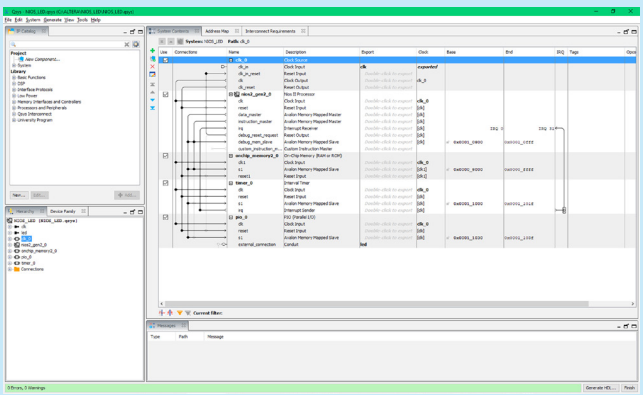
Rysunek 1. Dostosowanie kodu VHDL do wymagań interfejsu Avalon MM

```
entity x7seg is
    port (
          x          : in STD_LOGIC_VECTOR (15 downto 0);
          clk        : in STD_LOGIC;
          resetn     : in STD_LOGIC;
          write      : in STD_LOGIC;
          chipselect : in STD_LOGIC;
          byteenable : in STD_LOGIC_VECTOR(1 DOWNTO 0);
          seg        : out  STD_LOGIC_VECTOR (7 downto 0);
          dig        : out  STD_LOGIC_VECTOR (3 downto 0));
end x7seg;
```

Rysunek 2. Dodanie sygnałów resetn, write, chipselect, byteenable

Qsys

Przygotowałem obsługę wyświetlacza w VHDL-u – cały program jest dostępny w materiałach dodatkowych do projektu. Zawartość należy rozpakować w dowolnym miejscu, najlepiej w katalogu projektu NIOS_LED. Następnie uruchamiamy Quartusa, wybieramy projekt NIOS_LED, rozbudujemy go o obsługę wyświetlacza LED. Uruchamiamy Qsys, otwieramy NIOS_LED.qsys. W okienku **IP Catalog** dwukrotnie klikamy na ***New Component...*** (**rysunek 3**). W polu **Name** i **Display name** wpisujemy dowolną nazwę komponentu, ja wybrałem **hex7led**. Przechodzimy do zakładki **Files**, w sekcji **Synthesis Files** klikamy **Add File...** Przechodzimy do katalogu, w którym mamy rozpakowany plik 7seg_AVALON.7zip i dodajemy wszystkie pliki. Na pliku **x7seg.vhd** klikamy w kolumnie

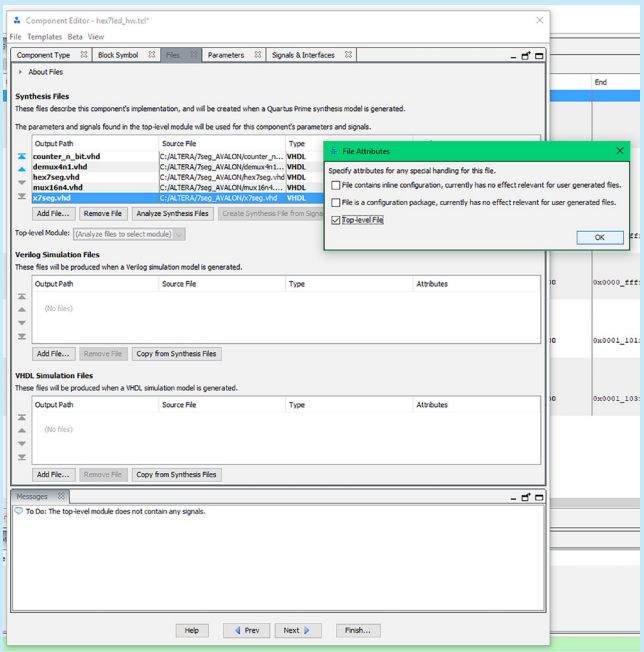


Rysunek 3. Okno IP Catalog – New Component...

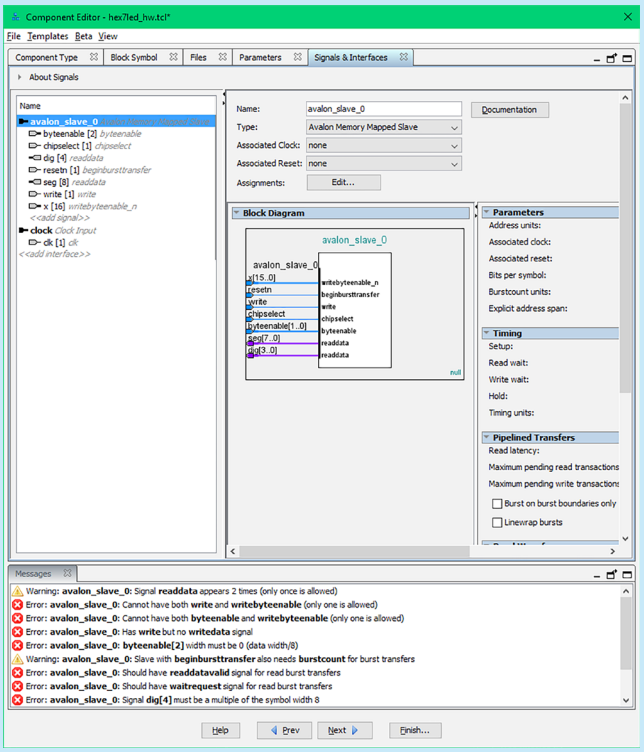
Attributes i wybieramy **Top-level File** (**rysunek 4**). Klikamy **Analyze Synthesis Files** oraz w sekcjach **Verilog Simulation Files** i **VHDL Simulation Files** na przycisk **Copy from Synthesis Files**. Przechodzimy do zakładki **Signal & Interfaces** (**rysunek 5**).

Trzeba to teraz uporządkować. Należy obserwować okienko *Messages*, jest ono aktualizowane na bieżąco i podpowiada, które sygnały trzeba poprawić. Sygnały ***dig, resetn, seg*** nie należą do magistrali Avalon, trzeba je z tej sekcji usunąć. ***Byteenable, chipselect, write*** są prawidłowo zidentyfikowane, sygnał **x** trzeba poprawić na **writedata**. Klikamy na **<<add interface>>** i wybieramy **Reset Input**, możemy zmienić nazwę reset i klikamy **<<add signal>>** w sekcji *reset* i wybieramy **reset**. W polu **Name** wpisujemy nazwę taką samą, jaką mamy w projekcie w VHDL-u, czyli **resetn**. Wybieramy też **Signal Type: reset_n**.

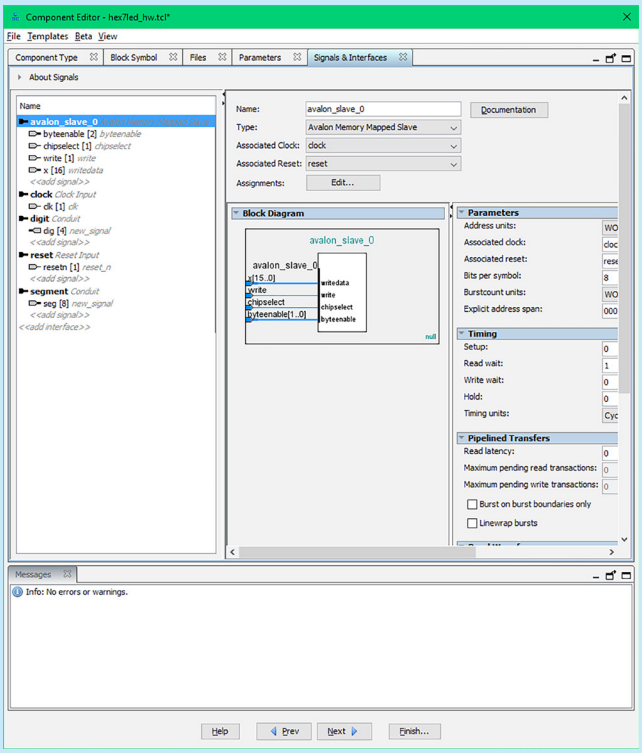
W tej chwili mamy prawidłowo skonfigurowany interfejs Avalon, trzeba jeszcze dodać interfejs wyświetlacza. Klikamy <<add interface>> i wybieramy conduct. Wpisujemy nazwę np. segment, klikamy <<add signal>>, wybieramy gwiazdkę. Tu wprowadzamy nazwę i szerokość bitów takie, jakie są w projekcie w VHDL-u (**Name: seg, Width: 8**) oraz zmieniamy **Direction** na **output**. Tak samo robimy dla wyboru cyfry (**Name: dig, Width: 4, Direction: output**). Wybieramy sekcję **avalon_slave_0** i w polach **Associated Clock** wybieramy **clock**, a w **Associated Reset** wybieramy **reset**. Jeśli wszystko zrobiliśmy prawidłowo na dole w **Messages** nie powinno być żadnych błędów i ostrzeżeń (**rysunek 6**). Klikamy **Finish** i mamy swój własny komponent w Qsysie. Wybieramy w okienku **IP Catalog** nasz nowy komponent i wciskamy **Add...** a następnie **Finish.** Podłączamy ***clock, reset*** oraz ***avalon_slave_0*** tylko do ***data_master***. Następnie wybieramy z menu *System* → *Assign base Address*, w celu przydzielenia adresu dla naszego wyświetlacza. Klikamy na dole po prawej **Generate HDL...**, w tym momencie mamy wygenerowany nowy cały system wraz z nowym komponentem do obsługi wyświetlacza. Po skończonej operacji wciskamy **Finish** i przechodzimy do Quartusa (**rysunek 7**).



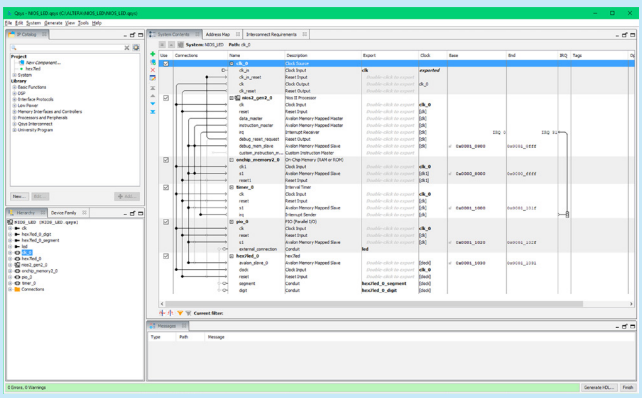
Rysunek 4. Top-level File



Rysunek 5. Zakładka Signal & Interfaces



Rysunek 6. Komunikat o poprawnym przebiegu syntezy



Rysunek 7. Okno robocze programu Quartus

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **92822**, pass: **37euo8qf**

Uruchamiamy kompilację (Ctrl+L), wykona się jeszcze prawidłowo, ale nie zadziała po tej operacji. W Pin Plannerze pojawią się nowe wyprowadzenia, które trzeba podłączyć do wyświetlacza na expanderze. Uruchamiamy Pin Planner i podpinamy tak jak na **rysunku 8**.

Eclipse

Po zakończeniu pracy z Pin Plannerem czas na poprawienie programu. Wybieramy z menu *Tools* ➝ *Nios II Software Build Tools for Eclipse*. W Eclipse, w Project Explorer, klikamy prawym przyciskiem myszy na NIOS_LED_bsp i z menu wybieramy *NIOS II* ➝ *Generate BSP*. W funkcji **irr()** dopisujemy linię *IOWR_ALTERA_AVALON_PIO_DATA(HEX7LED_0_BASE, counter);* (**rysunek 9**).

Przebudowujemy cały projekt, z menu wybieramy *Project* ➝ *Build All* (Ctrl+B). To jeszcze nie wszystko – mamy kod, który można uruchomić na procesorze NIOS poprzez interfejs JTAG. Trzeba wygenerować wsad do FPGA, klikamy prawym myszy w Project Explorerze na NIOS_LED i wybieramy *Make Targets* ➝ *Build...* (Shift+F9) i klikamy dwa razy na **mem_init_generate**. Zamykamy Eclipse i w Quartusie uruchamiamy kompilacje, (Ctrl+L) i wgrywamy wygenerowany wsad do MAXimatora. Na wyświetlaczu powinno pojawić się zliczanie w górę w systemie szesnastkowym.

Podsumowanie

Artykuł ma być instrukcją, jak krok po kroku używać własnych IP Core w systemach opartych na procesorze NIOS II, skupia w sobie najważniejsze kroki i etapy w procesie adaptacji własnego kodu do współpracy z magistralą Avalon. Środowisko Quartus jest bardzo złożone, przy projektowaniu pomyłka lub pominięcie jakiegoś etapu powoduje błędne działanie projektu. Starałem się zachować odpowiednią kolejność wykonywanych czynności przy projektowaniu, tak aby zminimalizować możliwość pomyłek.

Mam nadzieję, że zachęciłem programistów do sięgnięcia po układy FPGA, a także osoby znające Verilog lub VHDL do tworzenia własnych bibliotek IP Core w Quartusie. W razie problemów, pytań lub propozycji tematów związanych z ww. układami zapraszam na forum ***http://microgeek.eu***.

Mariusz Księżak

Node Name	Direction	Location
altera_reserved_tck	Input	PIN_H3
altera_reserved_tdi	Input	PIN_G1
altera_reserved_tdo	Output	PIN_H1
altera_reserved_tms	Input	PIN_H2
clk_clk	Input	PIN_L3
hex7led_0_digit_new_signal[3]	Output	PIN_D16
hex7led_0_digit_new_signal[2]	Output	PIN_D15
hex7led_0_digit_new_signal[1]	Output	PIN_E16
hex7led_0_digit_new_signal[0]	Output	PIN_E15
hex7led_0_segment_new_signal[7]	Output	PIN_F16
hex7led_0_segment_new_signal[6]	Output	PIN_G16
hex7led_0_segment_new_signal[5]	Output	PIN_G15
hex7led_0_segment_new_signal[4]	Output	PIN_H16
hex7led_0_segment_new_signal[3]	Output	PIN_H15
hex7led_0_segment_new_signal[2]	Output	PIN_J16
hex7led_0_segment_new_signal[1]	Output	PIN_J15
hex7led_0_segment_new_signal[0]	Output	PIN_L16
led_export[3]	Output	PIN_R16
led_export[2]	Output	PIN_P16
led_export[1]	Output	PIN_N16
led_export[0]	Output	PIN_M16
~ALTERA_CONFIG_SEL~	Input	PIN_F8
~ALTERA_nCONFIG~	Input	PIN_E8
~ALTERA_nSTATUS~	Input	PIN_F7
~ALTERA_CONF_DONE~	Input	PIN_E7
<<new node>>		

Rysunek 8. Połączenia w Pin Plannerze

```
static void irr(){
    counter++;
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, counter );
    IOWR_ALTERA_AVALON_PIO_DATA(HEX7LED_0_BASE, counter );
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
}
```

Rysunek 9. Modyfikacja funkcji irr()