Projektowanie własnego modułu na przykładzie modułów OPB_MEM oraz OPB_EPP

Autor: Zespół Rekonfigurowalnych Systemów Obliczeniowych www.fpga.agh.edu.pl
Ostatnia aktualizacja: 20.05.2006

Wstęp

Zadaniem tego dokumentu jest pokazanie jak zaprojektować własny moduł dołączony do magistrali OPB na przykładzie pamięć BRAM oraz jak posługiwać się modułem opb_epp służącym do komunikacji pomiędzy komputerem PC a płytą XSV (lub XSB przy zmienionych parametrach) poprzez port równoległy. Ponadto zadaniem tego projektu będzie pokazanie prostej metody symulacji swojego własnego projektu.

Układy Virtex posiadają wbudowaną pamięć blokową BlockRAM (w skrócie oznaczaną BRAM) o pojemności 4kb i kontrolowanej szerokości magistrali danych 1, 2, 4, 8 lub 16 bitów. W naszym projekcie użyty zostanie 1 BRAM o szerokości magistrali danych 8 bitów. Pojemność pamięci będzie zawsze równa 4kb czyli 512B (magistrala adresowa 9 bitowa).

Uwaga: Dokładniejszy opis wykonywanych w programie EDK czynności znajduje się w pliku (Embeded System Tools Guide) **est_guide.pdf** głównie w rozdziale **Microprocessor Peripherial Discription** i Peripherial Analyze Order.

Wszystkie moduły wymagane w tutorialu są dostępne na stronie OPB: http://galaxy.uci.agh.edu.pl/~jamro/opb.
Należą do nich następujące moduły: apsi, opb_epp, opb_mem, dane do tego tutorialu. Moduły te należy skopiować (rozpakować) do swojego katalogu roboczego EDK o nazwie np. tutorial w podkatologu pcores, czyli C:/My_Designs/tutorial/pcores. W przypadku korzystania z innych modułów np. MicroBlaze'a, analizatorastanów logicznych, pamięci zewnętrznej SRAM zaleca się również przeczytanie innych tutoriali.

Moduł Pamięci opb_mem

Projekt VHDL

Opis jak należy wykonać własny moduł w języku VHDL znajduje się w osobnym tutorialu: **tut_opb_vhdl,** z którym należy się zapoznać później podczas wykonywania swojego własnego modułu!

Każdy projekt jest sparametryzowany dlatego w opisie VHDL powinny znaleźć się następujące parametry:
C_OPB_AWIDTH – określa szerokość magistrali adresowej.
C_OPB_DWIDTH – określa szerokość magistrali danych
C_BASEADDR – określa adres bazowy urządzenia (slave).
C_HIGHADDR – określa adres końcowy urządzenia (slave).

C_REAL_DWIDTH (lub c_dwidth) - domyślnie szerokość magistrali danych magistrali OPB (C_OPB_DWIDTH) jest równa 32 bity, dlatego w przypadku gdy użytkownik potrzebuje tylko np. 8 bitów wprowadzono dodatkowo ten parametr. Dzięki temu można definiować prawdziwą szerokość magistrali danych, pozostałe młodsze bity (o indeksie od c_dwidth do 31) są na stałe podłączone do masy. Umożliwia to łatwe komunikowanie się z urządzeniami, które są podłączone do magistrali 32-bitowej jednak w rzeczywistości używają tylko 8 najstarszych bitów (0÷c_dwidth-1).

Projekt EDK

W ramach projektu wykorzystywane będą tylko dwa moduły (dostępne na stronie OPB):

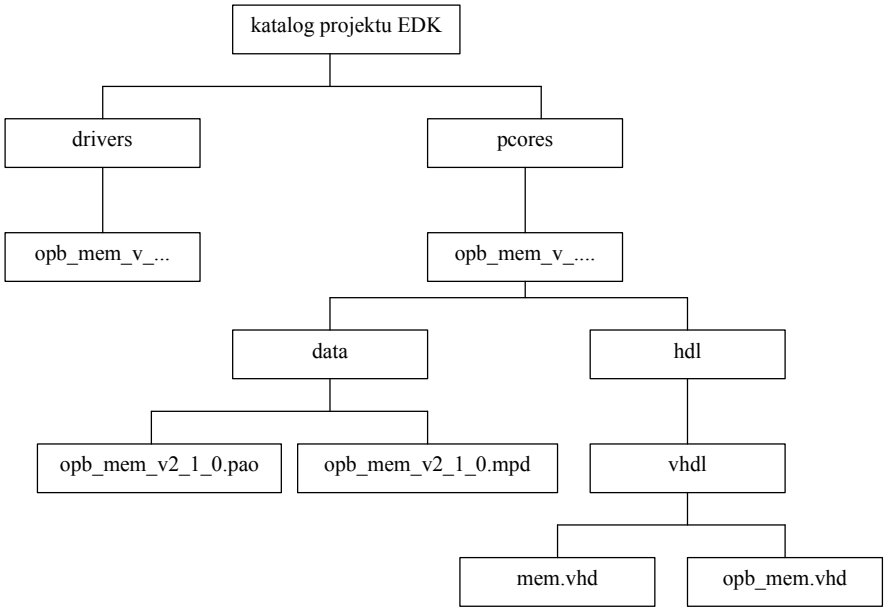
- opb_epp – moduł nadrzędny (master) sterowany przez port równoległy komputera.
- opb_mem – moduł pamięci bram przedstawiony powyżej.

Dodatkowo należy wykorzystać program apsi.exe wraz z opisem języka skryptu (moduł apsi).

Aby moduł opb_mem był widoczny w programie EDK oprócz plików *.vhd należy dostarczyć dwa dodatkowe pliki z rozszerzeniem:

1) MPD – (Microprocessor Peripheral Definition) opisujący funkcje modułu oraz użyte wyprowadzenia.
2) PAO - Peripheral Analyze Order – mówiący o kolejności kompilacji poszczególnych plików vhdl.

Aby pliki te były widoczne muszą one mieć ściśle określoną nazwę: nazwa_v_2_1_0.mpd oraz nazwa_v_2_1_0.pao. Gdzie: nazwa – nazwa projektu, v_2_1_0 dodawana końcówka (niezależna od wersji projektu czyli dla wersji projektu v_1.00 końcówka v_2_1_0 również musi być dodana). Pliki te muszą być umieszczone w ściśle określonych katalogach podanych poniżej:



Format pliku *.mpd został opisany w est_guide.pdf w rozdziale Microprocessor Peripherial Description. W ramach swoich projektów zalecane jest używanie pliku opb_mem_v_2_1_0.mpd jako pliku wzorcowego.

Plik *.mpd opisuje następujące elementy:

Typ urządzenia

przykład: rozpoczęcie definiowania modułu opb_mem, urządzenie peryferyjne.

BEGIN opb_mem
OPTION IPTYPE=PERIPHERAL ,
OPTION IMP_NETLIST= TRUE

Typ używanych magistral
przykład – urządzenie podrzędne (slave) na magistrali OPB
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE

Parametry układu (po słowie kluczowym generic).
przykład – określa domyślny adres bazowy (który trzeba zmienić), typ parametru, minimalny rozmiar oraz do jakiej magistrali ma się on tyczyć (urządzenie może być podłączone do więcej niż jednej magistrali).
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector, MIN_SIZE=0x2000, BUS=SOPB

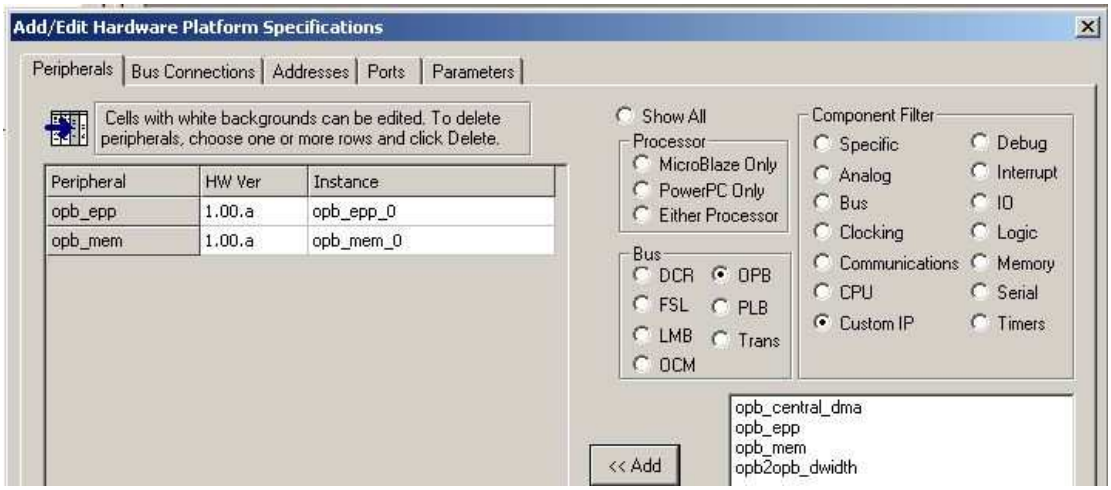
Wyprowadzenia układu (po słowie kluczowym port)
Przykład 1: Sygnał opb_clk, wejście, typ sygnału – sygnał zegara, podłączony do magistrali SOPB
PORT OPB_Clk = "", DIR=IN, SIGIS=CLK, BUS=SOPB
Przykład 2: Magistrala adresowa, wejściowa o rozmiarze c_opb_awidth podłączona do magistrali SOPB
PORT OPB_ABus = OPB_ABus, DIR=IN, VEC=[0:C_OPB_AWIDTH-1], BUS=SOPB
Warto w tym miejscu podkreślić znaczenie przypisania OPB_clk= "" (cydzysłowie) oznacza że sygnał ten będzie widoczny w programie EDK menu Project/Add Edit Cores/Ports. Podanie nazwy (brak cudzysłowia) powoduje, że sygnał jest automatycznie łączony i użytkownik nie ma wpływu na sposób połączenia sygnału.
Bardzo ważne jest również nazewnictwo sygnałów magistrali OPB, najlepiej jest używać nazewnictwa określonego w standardzie magistrali OPB i podanego w przykładzie.

Plik *.pao (Peripheral Analyze Order)
Zadaniem tego pliku jest określenie kompilacji poszczególnych plików *.vhdl. Najpierw powinny być kompilowane pliki podrzędne a następnie pliki nadrzędne. W naszym przypadku najpierw należy skompilować plik mem.vhd, opb_slave a następnie plik opb_mem.vhd dlatego zawartość pliku *.pao powinna być następująca:
lib opb_mem_v1_00_a mem
lib opb_mem_v1_00_a opb_slave1
lib opb_mem_v1_00_a opb_mem

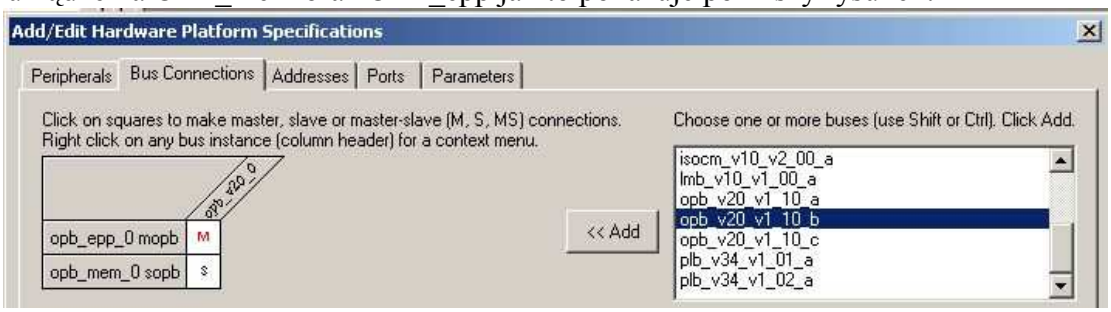
Projekt w EDK
Następnym etapem projektu jest wykorzystanie modułu opb_mem oraz modułu opb_epp, który jako urządzenie master będzie zapisywać i odczytywać zawartość pamięci opb_mem.

Po uruchomieniu programu EDK i wybraniu nowego projektu (menu File/New Project/Platform Studio), należy wybrać odpowiedni katalog roboczy EDK. W naszym przypadku będzie to katalog C:/My_Designs/tutorial (ścieżka nie może zawierać spacji!). Należy również wybrać odpowiedni układ FPGA. Dla płyty XSV będzie to układ: Virtex 300 PQ240 –4. Uwaga: po wybraniu wszystkich elementów wystąpi komunikat że plik *.mhs nie został zdefiniowany. Komunikat ten należy zignorować.

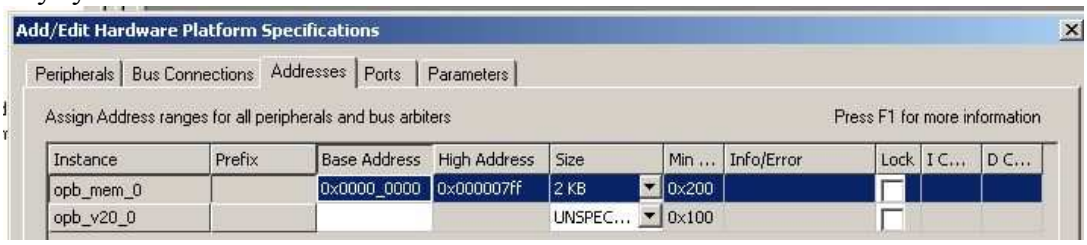
Następnie należy dodać elementy opb_epp oraz opb_mem jak to pokazuje poniższy rysunek (menu: Project/Add Eddit Cores), aby szybko znaleźć dany element dobrze jest ustawić odpowiedni filtr:



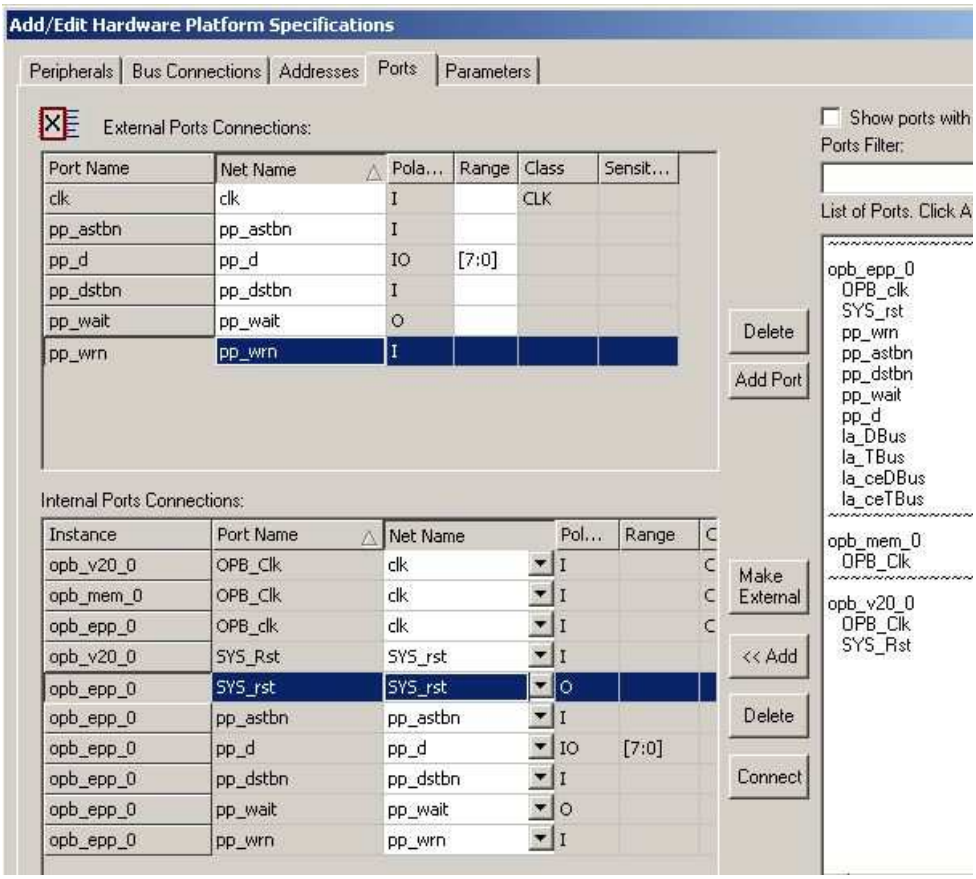
Następnie należy dodać magistralę OPB (może to być wersja: opb_v2_00_b) i podłączyć do niej urządzenia OPB_mem oraz OPB_epp jak to pokazuje poniższy rysunek:



Następnie należy ustawić odpowiednie lokacje adresowe dla urządzeń slave, pokazuje to poniższy rysunek:



Następnie należy dodać porty wejścia/wyjścia. Ilustruje to poniższy rysunek. W tym celu należy zaznaczyć dany sygnał w prawym oknie (można zaznaczyć wiele sygnałów na raz używając przycisku Ctrl lub Shift) i przycisnąć przycisk *Add* (nie mylić z *Add Port*). Następnie należy podłączyć wszystkie sygnały zegarowe OPB_Clk w jeden sygnał *clk* (zaznaczyć z klawiszem Ctrl, przycisnąć przycisk Connect oraz wpisać nazwę clk). Ponadto należy zmienić nazwy sygnałów w dolnym oknie poprzez usunięcie odpowiednich przedrostków. Następnie należy zaznaczyć odpowiednie sygnały zewnętrzne w dolnym oknie (wszystkie sygnały oprócz sygnału *SYS_Rst*) i wyprowadzić je na zewnątrz – przycisk *Make External*. Dodatkowo w górnym oknie należy zaznaczyć że port *pp_d* jest magistralą (Range [7:0]).



W parametrach (zakładka Parameters) w naszym przypadku nic nie ustawiamy. Należy zwrócić uwagę jednak na parametry *opb_epp*, dla którego oprócz ustawień analizatora stanów logicznych rozpoczynających się od c_la_* (analizator stanów logicznych jest omówiony w osobnym dokumencie) występuje parametr *c_real_dwidth*, który określa rzeczywistą szerokość magistrali danych OPB. W przypadku *c_real_dwidth* mniejszej od *c_opb_dwidth* najmłodsze bity magistrali danych są zawsze ustawione na 0. W naszym przykładzie parametr *c_real_dwidth* będzie miał niezmienioną szerokość równą 32 bity.

Po wykonaniu powyższych czynności należy wygenerować netlistę opisanego projektu, menu: Tools/Generate Netlist.

Następnie należy wygenerować pliki na potrzeb symulacji VDHL, menu: Tools/Generate Simulation HDL Files. Przed tym jednak upewnij się czy została wybrana opcja: Option/Project Option/HDL and Simulation/Simulation Models/Structural. Dodatkowo należy określić ścieżki *EDK library* oraz *Xilinx library*. Ścieżki te mogą być dowolnie określone – np. w katalogu roboczym EDK w podkatalogu *lib* (C:\My_Design\tutorial\lib).

Uwaga (tylko do przeczytania): W przypadku uruchamiania własnego projektu bardzo często zdarzają się błędy podczas generacji netlisty generowany jest błąd, którego źródło nie jest dokładnie podane. Dlatego w takim przypadku zaleca się aby projektowany moduł nie miał generowanej netlisty – nie był syntezowany. Aby tego dokonać należy w pliku *.mpd danego modułu zmienić wyrażenie na: *OPTION IMP_NETLIST= FALSE*. Następnie należy powtórnie wygenerować netlistę oraz przeprowadzić symulację VHDL. (Procedura opisana w dalszej części tego tutorialu.) Należy zaznaczyć, że wymieniony moduł musi być symulowany na poziomie funkcjonalnym – czyli wszystkie pliki źródłowe VHDL muszą być skopiowane do katalogu symulatora. Przeprowadzenie symulacji najczęściej umożliwia znalezienie błędu we własnym module i poprawne przeprowadzenie syntezy. Należy pamiętać, że po

poprawieniu błędu w pliku VHDL należy przywrócić wyrażenie w pliku *.mpd: *OPTION IMP_NETLIST= TRUE* oraz powtórnie wygenerować netlistę.

Symulacja VHDL

Pierwszym zadaniem jest symulacja na poziomie języka VHDL naszego modułu opb_mem wraz z modułami pomocniczymi (opb_epp, itd.). W tym celu otwórz nowy projekt ActiveHDL. Projekt ActiveHDL ma być projektem pustym oraz narzędzie do syntezy i implementacji ma być pozostawione puste (nie wybrane). Następnie skopiuj do katalogu roboczego ActiveHDL (src) następujące pliki z katalogu roboczego EDK:
hdl\system.vhd – plik nadrzędny opisujący cały system (cały układ FPGA). W przypadku EDK 6.1 należy ten plik skopiować z katalogu simulation\structural\system.hdl.
hdl\opb_mem_0_wrapper.vhd – plik opisujący parametry układu opb_mem na poziomie języka HDL
hdl\opb_epp_0_wrapper.vhd – plik przed syntezą opisujący działanie układu opb_epp dla podanych parametrów
simulation\opb_v_20_0_wrapper.vhd – opisujący działanie magistrali oraz arbitra OPB (po syntezie)
pcores\opb_mem\hdl\vhdl*.vhd – wszystkie pliki opisujące moduł opb_mem (przed syntezą)
pcores\opb_epp\hdl\vhdl*.vhd – wszystkie pliki opisujące moduł opb_epp (przed syntezą)
tut_opb_mem.zip\test.vhd – plik nadrzędny do symulacji, symulujący środowisko zewnętrzne pliku system.vhd (wymuszający sygnał clk oraz sygnały portu równoległego EPP). Katalog tutorial to domyślnie dane do tutorialu dostępne na sieci obok tekstu tego tutoriala.
pcores\opb_epp\others\epp_model.vhd – plik symulujący działanie portu równoległego EPP na podstawie skryptu (plik apsi.txt) czytanego przez program apsi.exe
apsi.zip\apsi.exe – program interpreter instrukcji – przesłań przez port równoległy.
tut_opb_mem.zip\apsi.txt – domyślny skrypt, komendy dla programu apsi.exe.
tut_opb_mem.zip\test.hex – przykładowy plik z danymi.

Skopiowane pliki *.vhd należy dodać do projektu a następnie skompilować.
Uwaga 1: Podczas kompilacji modułów VHDL należy ręcznie kasować następujące nieistniejące biblioteki pokazywane przez ActiveHDL:
LIBRARY opb_epp_v1_00_a;
USE opb_epp_v1_00_a.ALL;
Uwaga 2: Aby symulować moduł funkcjonalnie (na poziomie pisanego kodu VHDL) należy skopiować jego opis parametrów zewnętrznych czyli plik hdl\nazwa_modulu_wrapper.vhd oraz opis samego modułu czyli pliki: pcores\nazwa_modulu\hdl\vhdl*.vhd do katalogu roboczego ActiveHDL. Aby zasymulować moduł na poziomie po syntezie należy zamiast wyżej wspomnianych plików skopiować jeden plik: simulation\nazwa_modulu_wrapper.vhd. Jeżeli element jest elementem własnym (np. opb_mem) to jego opis znajduje się bezpośrednio w katalogu pcores (jak to opisano powyżej). Jeżeli element jest elementem dostarczonym przez firmę Xilinx to zamiast katalogu pcores należy użyć katalogu EDK_Path\hw\iplib\pcores, niestety niektóre projekty są zakodowane i można tylko korzystać z wersji po syntezie (np. MicroBlaze). Ogólna zasada jest taka, że w katalogu simulation znajdują się kompletne pliki po syntezie a w katalogu hdl znajdują się pliki hdl, które wymagają dołączenia dodatkowych plików źródłowych znajdujących się np. w katalogu pcores.

Wymuszenia (Teoria)

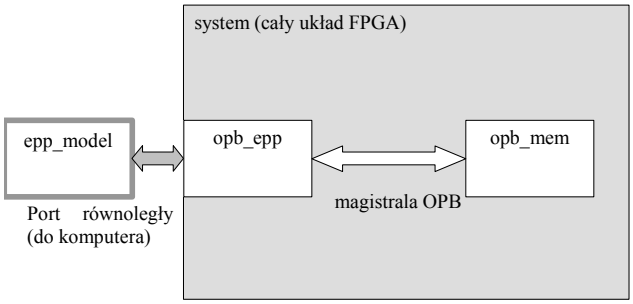
Naturalna i najczęściej stosowaną przez studentów metodą symulacji jest podawanie wymuszeń podczas symulacji w programie ActiveHDL. Jest to jednakże metoda prymitywna i dlatego stosowana tylko dla prostych projektów.

Następną bardziej rozbudowaną metodą podawania wymuszeń częściowo stosowaną podczas tego projektu jest nadawanie wymuszeń na poziomie języka VHDL. W tym celu napisano specjalny moduł test.vhd, w którym osadzono moduł testowany system.vhd (moduł nadrzędny). Otwórz plik test.vhd i zwróć uwagę na następujący fragment modułu:

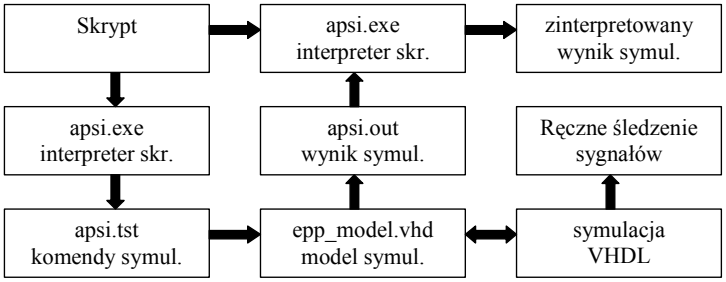
```
process begin
        wait for 10 ns;
        clk<= '0';
        wait for 10 ns;
        clk<= '1';
end process;
```

Powyższy fragment powoduje generacje sygnału zegarowego. Przykład jest bardzo prosty (można go również zaimplementować na poziomie symulacji ActiveHDL) ale może być w prosty sposób rozbudowany do bardzo skomplikowanego.

Przykładem bardziej zaawansowanej symulacji jest plik *epp_model.vhd*, który współpracuje z programem *apsi.exe* i umożliwia symulowanie działania portu równoległego EPP oraz działanie programu *apsi.exe*. Moduł ten należy umieścić w pliku nadrzędnym do symulacji (*test.vhd*) w miejscu gdzie fizycznie znajduje się port równoległy pokazuje to poniższy rysunek.



Dzięki temu rzeczywiste działania wykonywane przez program *apsi.exe* mogą być symulowane w ActiveHDL bez potrzeby czasochłonnego pisania osobnych skomplikowanych wymuszeń działania portu równoległego na poziomie języka VHDL. Schemat blokowy całego procesu symulacji jest przedstawiony poniżej:



Poszczególne elementy powyższego rysunku są następujące:

1) Skrypt

Pierwszą czynnością jest określenie komend jakie ma wykonywać program APSI, czyli napisanie odpowiedniego skryptu. Jedyną czynnością dodatkową podczas symulacji w

porównaniu ze standardowym wykonywaniem jest dodanie na początku skryptu dodatkowej instrukcji *vhdlsim*.

2) Uruchomienie programu *apsi.exe*

Ten krok służy generacji wymuszeń dla symulatora. Ponieważ w skrypcie umieszczono instrukcje *vhdlsim,* program *apsi.exe* przestanie się komunikować z płytą XSV (XSB) a zacznie generować wymuszenia dla symulatora w postaci pliku o roboczej nazwie *apsi.tst*.

3) Komedy symulacyjne *apsi.tst*

Pliku *apsi.tst* służący do komunikacji pomiędzy interpreterem skryptu (*apsi.exe)* a symulatorem VHDL.

4) Model symulacyjny portu równoległego *epp_model.vhd*

Model ten należy osadzić w pliku symulacyjnym (testbench) w miejscu gdzie fizycznie znajduje się port równoległy.

5) Symulacja VHDL

Podczas symulacji możliwe jest śledzenie przebiegów sygnałów wewnętrznych i zewnętrznych w podobny sposób jak to się odbywa podczas standardowej symulacji.

6) Wynik symulacji *apsi.out*

Wynik symulacji można analizować bezpośrednio podczas symulacji, jednakże bardzo często liczba przebiegów praktycznie uniemożliwia sprawdzenie poprawności działania układu. Dlatego dodatkowo podczas symulacji wszystkie dane odczytane na porcie równoległym są zapisywane przez model symulacyjny *epp_model.vhd* do pliku *apsi.out*.

6) Powtórne uruchomienie programu *apsi.exe*

Podczas powtórnego uruchomienia interpreter dysponuje wynikiem symulacji zapisanym w pliku *apsi.out* i na podstawie tego pliku zachowuje się tak jakby dane zapisane w tym pliku zostały w rzeczywistości odczytane z portu równoległego. Podsumowując, podczas powtórnego uruchomienia skrypt zachowuje się tak jakby w rzeczywistości kontaktował się z płytą XSV podłączoną do portu równoległego.

Wymuszenia (Czynności do wykonania)

Aby zasymulować działanie naszego układu należy wykonać pewne czynności na pamięci opb_mem. Będą to następujące czynności wykonywane przez port równoległy i opisane w skrypcie *apsi.txt* programu *apsi.exe*. Dokładny opis instrukcji skryptu znajduje się w pliku *apsi.doc* w module APSI.

vhdlsim // program epp.exe przechodzi w tryb symulacji
config system // konfiguracja układu FPGA (instrukcja nie symulowana)
writebyte 1 AA // zapisz pojedynczy bajt pod adres 1
readbyte 1 // odczytaj pojedynczy bajt spod adresu 1 - można porównać odczyt z zapisem
writeblock test.hex 0 F // zapisz 16 bajtów znajdujących się w pliku test.hex pod adres 0-F
readblock test.tmp 0 F // odczytaj to co poprzednio zapisane, wynik zapisz do pliku test.tmp
filecomp test.hex test.tmp // porównaj pliki zapisane i odczytane

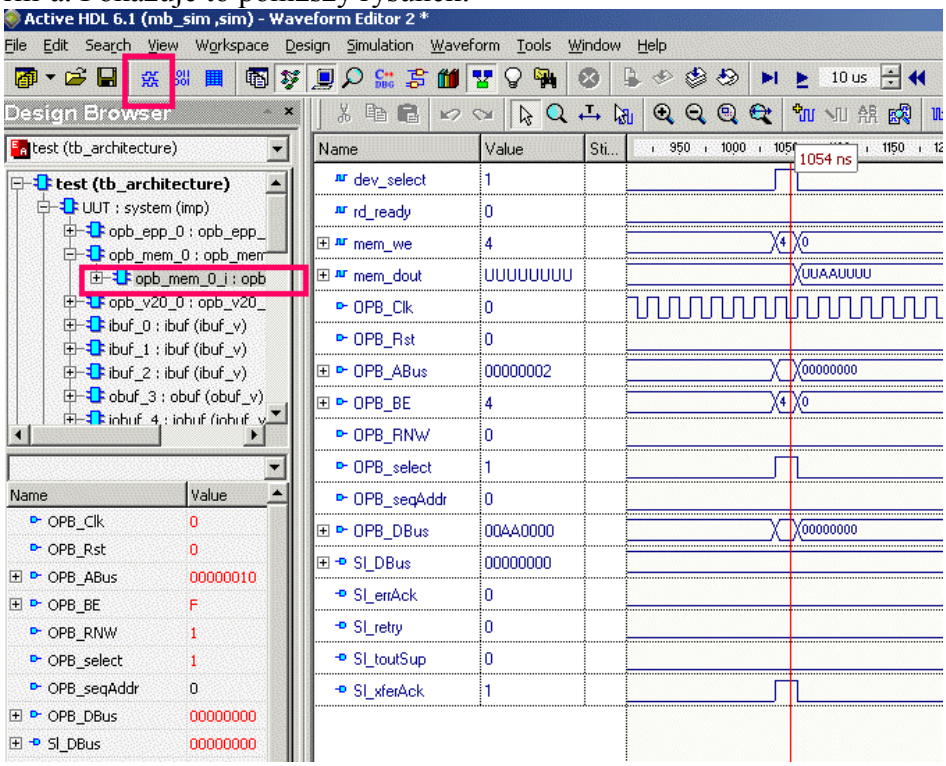
Przykład skryptu apsi.txt

Aby zasymulować działanie powyższego skryptu uruchom program *apsi.exe*, pliki *apsi.exe*, *apsi.txt* oraz *test.hex* powinny być skopiowane do katalogu roboczego ActiveHDL'a (*src*). Podczas pierwszego uruchomienia programu *apsi.exe* wynik odczytu jest inny od zapisu ponieważ nie przeprowadzono jeszcze symulacji VHDL (standardowo wyświetlana jest dana 0x03 świadcząca o braku przeprowadzenia symulacji).

Symulacja VHDL właściwa

Następnym etapem jest przeprowadzenie symulacji projektu w programie ActiveHDL.

Pierwszą rzeczą jest wybranie elementu nadrzędnego, którym jest test (tb_architecure). Następnie należy zainicjalizować symulację, menu: Simulation/Initialize Simulation. Teraz należy wybrać sygnały, które chcemy oglądnąć poprzez stworzenie nowego waveform'a (4 ikona od lewej) wybranie modułu opb_mem oraz przeciągnięcie go do na nowo stworzony waveform'a. Pokazuje to poniższy rysunek:

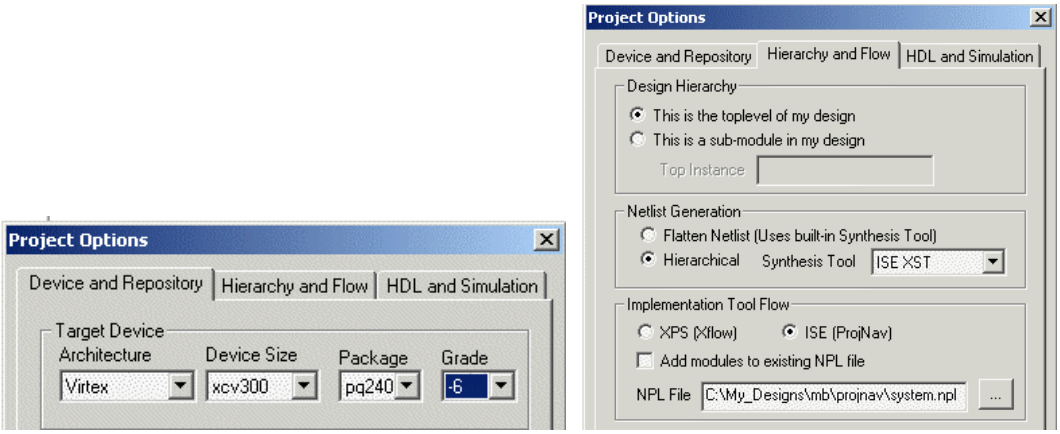


W ten sposób możemy oglądać wszystkie sygnały które znajdują się w bloku opb_mem.vhd. Czas symulacji powinien być duży (np. 10ms), ponieważ symulacja jest automatycznie kończona w momencie kiedy zostaną wykonane wszystkie komendy skryptu APSI. W sytuacji zakończenia symulacji pojawia się komunikat w ActiveHDL'u*: FAILURE: O.K. End of simulation (by epp_model, no further commands to execute).* Następnie uruchom symulacje. Pytanie kontrolne: co pokazuje powyższy rysunek w chwili czasowej 1054ns?

Teoretycznie możliwe jest sprawdzenie czy dane odczytane z pamięci są tożsame z danymi zapisanymi na podstawie oglądanych przebiegów symulacyjnych. Jednakże jest to dość czasochłonne i możliwa jest tylko wybiórcza kontrola. Dlatego aby przekonać się czy dane odczytane są takie same jak zapisane należy powtórnie uruchomić program *apsi.exe*. Tym razem program ten dysponuje już danymi odczytanymi z portu równoległego podczas symulacji i jest w stanie stwierdzić porównując dane zapisane i odczytane czy układ działa poprawnie – komenda filecomp jest zakończona tekstem: pliki identyczne.

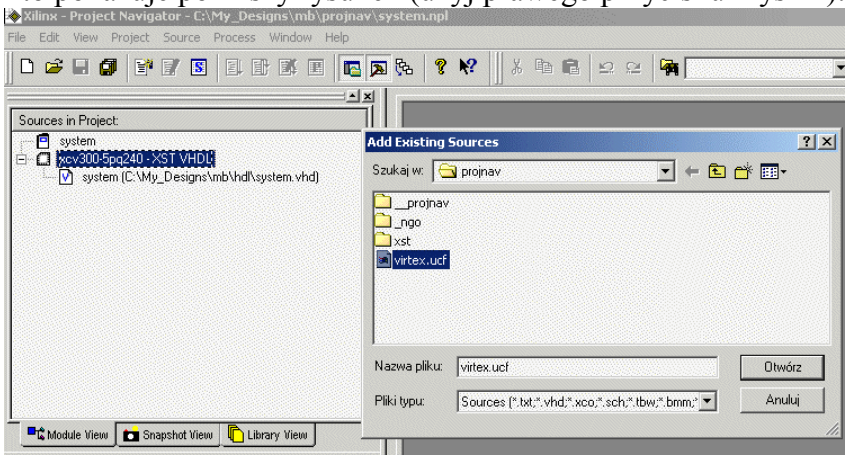
Implementacja

Następnym etapem jest rzeczywista weryfikacja systemu, czyli sprawdzenie czy układ działa poprawnie w rzeczywistości. Dlatego należy wrócić do programu EDK i wyeksportować projekt do programu Xilinx'a Project Navigator poprzez wybranie menu: Tools/Export to ProjNav. Aby to było możliwe należy wybrać opcje w menu: Option/Projekt Option oraz wybrać układ Virtex 300 oraz projekt ISE jak to pokazuje poniższy rysunek:

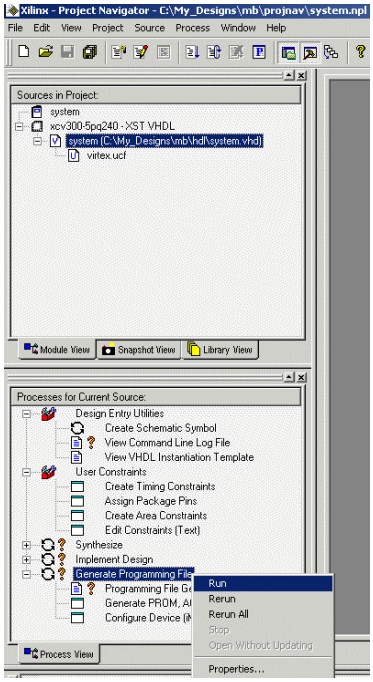


Ścieżka robocza projektu programu Project Navigator zostanie ustawiona w opcjach Hierarchy and Flow/NPL File (zob. powyższy rysunek) i do tego katalogu zostanie wyeksportowany projekt z EDK.

Następnie należy otworzyć program Project Navigator oraz projekt który przez chwilą został wygenerowany przez EDK. Do tego projektu należy dołączyć plik virtex.ucf, który między innymi opisuje lokalizacje końcówek układu virtex. Plik virtex.ucf należy skopiować z danych tutorialu do katalogu roboczego Project Navigator. Plik virtex.ucf należy dołączyć do projektu jak to pokazuje poniższy rysunek (użyj prawego przycisku myszki):

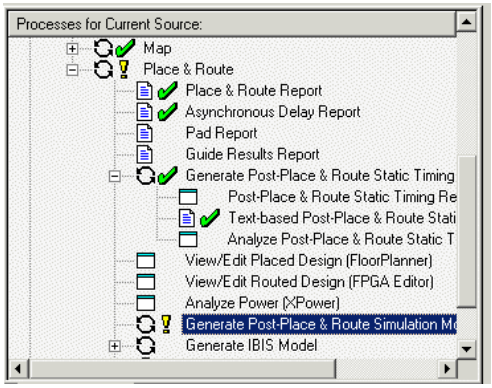


Następnie uruchom program generujący bit'a (plik konfigurujący układ FPGA), jak to pokazuje poniższy rysunek:



Symulacja czasowa

Należy pominąć ten punkt w przypadku braku czasu. Aby przeprowadzić symulacje czasową (uwzględniającą rzeczywiste opóźnienia czasowe w układzie FPGA) należy w programie Project Navigator wybrać Gennerate Post Place & Route jak to pokazuje poniższy rysunek:



W ten sposób został wygenerowany plik system_timesim.vhd w katalogu roboczym Project Navigator. Następnie skopiuj ten plik do katalogu roboczego ActiveHDL. Plik ten powinien podmienić istniejący opis vhdl pliku system.vhd. Można tego dokonać np. przez utworzenie nowego projektu (design i skopiowanie do niego wszystkich plików oprócz system.vhd) lub skasowanie biblioteki znajdującej się w okienku Design Browser w zakładce Files oraz powtórną kompilacje następujących plików: epp_model.vhd, system_timesim.vhd, test.vhd. Symulacja czasowa przebiega w podobny sposób jak dla pozostałych opisanych uprzednio symulacji, jednakże decydującą różnicą jest to, że sygnały wewnętrzne są zakodowane. Dlatego podczas tej symulacji najlepiej jest oglądać tylko sygnały zewnętrzne układu virtex. Sygnały wewnętrzne są dostępne w sposób trudny do rozszyfrowania, z praktycznego punktu widzenia możliwe jest tylko oglądanie sygnałów będących wyjściem rejestrów. Dlatego w sytuacjach gdy symulacja czasowa nie przebiega poprawnie (wynik jest niepoprawny) bardzo często sygnały które chcemy oglądnąć należy po prostu wyprowadzić na zewnątrz.

Symulacja czasowa jest czasochłonna i często nie pokazuje dokładnie problemu w opóźnieniach czasowych dlatego zaleca się użycie programu ISE Post-Place & Route **Static**

Timing Analyzer, który analizuje opóźnienia czasowe w układzie i pokazuje najgorsze opóźnienia. **Zaleca się skorzystanie z tego programu w razie jakichkolwiek niezgodności pomiędzy wynikiem rzeczywistym a symulacją**.

Rzeczywista weryfikacja

Plik *system.bit* wygenerowany przez Project Navigator należy teraz skopiować do katalogu roboczego ActiveHDL. Następnie otwórz plik *apsi.txt* i usuń (lub daj w komentarz - znak // - jak w języku C) linię vhdlsim – co spowoduje że program apsi.exe przejdzie w tryb rzeczywistej pracy i komunikacji z płytą XSV. Następnie powtórnie uruchom program *apsi.exe*.

Analizator stanów Logicznych*

W niektórych sytuacjach symulacja czasowa przebiega poprawnie, również Static Timing Analyzer pokazuje, że maksymalne opóźnienia w normie, jednakże rzeczywisty układ nie działa. Przyczyną takiego stanu rzeczy może być to, że modele symulacyjne są tylko modelami i nie uwzględniają rzeczywistych cech układu lub też układ nie został gruntownie przesymulowany (nie uwzględniono rzeczywistych wymuszeń). Czasami pojawiają się zakłócenia lub też wadliwa praca układu, które z definicji nie są symulowane. Ponadto proces symulacji jest dość czasochłonny, np. aby przesymulować działania układu przez czas 1s potrzebna jest wielogodzinna symulacja. Dlatego czasami sensowna jest rejestracja rzeczywistych przebiegów jakie występują w układzie FPGA. Do tego służy np. analizator stanów logicznych, który został wbudowany w element opb_epp. Przykład zastosowania analizatora został omówiony w osobnym tutorialu (tutorialu: Symulacja i testowanie ...)