Podłączenie własnego modułu do magistrali OPB w VHDL z wykorzystaniem modułów opb_master i opb_slave

Autor: Ernest Jamro
Aktualizacja: 14.10.2005

Spis treści:

1 Wstęp

Zadaniem tego tutorialu jest zilustrowanie jak należy podłączyć własne urządzenie do magistrali OPB. W ramach urządzenia zostaną użyte gotowe moduły: opb_slave1, opb_master oraz fifo. Tutorial ten w odróżnieniu do tutorialu: tut_opb_mem („Projektowanie własnego modułu na przykładzie modułów OPB_MEM oraz OPB_EPP") będzie się skupiał tylko na projektowaniu kodu VHDL – wnętrza własnego modułu. Dlatego dodatkowo aby wykonać cały kompletny projekt należy zapoznać się również z tutorialem: tut_opb_mem. Ponadto cały gotowy projekt VHDL znajduje się w tutorialu tut_opb_mem dlatego zaleca się również zapoznanie się z kodami VHDL tego tutorialu. Przed przystąpieniem do powyższego projektu należy zapoznać się ze standardem magistrali OPB, który jest zamieszczony na stronie OPB (http://galaxy.uci.agh.edu.pl/~jamro/opb).

Istnieją dwie metody dołączania swojego modułu do magistrali OPB:

- Jako urządzenie podrzędne (slave) – w tym przypadku należy skorzystać z gotowego modułu opb_slave.
- Jako urządzenie nadrzędne (master) - w tym przypadku należy skorzystać z gotowego modułu opb_master.

Każde urządzenie może mieć wiele niezależnych magistral OPB, dlatego można np. projektować urządzenie z dwoma magistralami master i jedną magistralą slave, itd.

Moduły opb_master, opb_slave, fifo można wykorzystywać wielokrotnie w tym samym projekcie. Nie należy zmieniać ich nazwy, tak aby nazwy się różniły. Nie należy ich również modyfikować. W przypadku błędnie działających modułów należy o tym powiadomić prowadzącego. W pracy zespołowej bardzo ważne jest aby poszczególne elementy były wykonywane przez różne osoby, dlatego dalsza modyfikacja wspomnianych

modułów jest bardzo prosta pod warunkiem zachowania powyższych zasad. Wystarczy że jedna osoba zmodyfikuje np. moduł *opb_master* a już wszystkie moduły będą mogły współpracować z nowszą i ulepszoną wersją interface'u. W przypadku istnienia kilku wersji tego samego modułu należy zastosować moduł młodszy.

2 Parametry (generic) podstawowe modułu opb_slave i opb_master:

Każdy projekt jest sparametryzowany. Parametry te należy ustawiać w zależności od potrzeb w programie EDK (w menu: Project/Add_Edit Cores/Parameters). Wyjątek stanowi parametr C_OPB_DWIDTH i C_OPB_AWIDTH, które są automatycznie ustawiane w programie EDK. Aby można było ustawić parametr w programie EDK musi on być również uwzględniony w kodzie VHDL każdego modułu za pomocą słowa kluczowego *generic.* W opisie vhdl, każdego modułu powinny znaleźć się następujące parametry:

C_OPB_AWIDTH – określa szerokość magistrali adresowej OPB (parametr niemożliwy do zmienienia w EDK, zawsze 32).

C_OPB_DWIDTH – określa szerokość magistrali danych (parametr niemożliwy do zminienia w EDK, zawsze 32)

C_BASEADDR – (dotyczy tylko urządzenia slave) określa adres bazowy urządzenia.

C_HIGHADDR – (tylko slave) określa adres końcowy urządzenia. Uwaga: ADDR_SIZE= C_HIGHADDR - C_BASEADDR + 1 powinien być potęgą liczby 2. Ponadto C_BASEADDR powinien być całkowitą wielokrotnością ADR_SIZE. Za pomocą parametrów C_BASEADDR i C_HIGHADDR określa się na jakie lokacje adresowe dane urządzenie *slave* odpowiada. Stała ADDR_SIZE= C_HIGHADDR+1- C_BASEADDR określa przestrzeń adresową użytkownika (czyli przestrzeń adresową używaną wewnątrz modułu), szerokość magistrali adresowej AWIDTH użytkownika jest równa log2(ADDR_SIZE)

C_AWIDTH – określa (w podobny sposób jak C_DWIDTH) szerokość magistrali adresowej użytkownika. W przypadku urządzenia Slave szerkość magistrali adresowej jest obliczana lokalnie według wzoru: awidth= log2(C_HIGHADDR + 1 - C_HIGHADDR). W przypadku urządzenia master szerokość magistrali adresowej *awidth* jest określona poprzez użytkownika lub wyprowadzana jako parametr w EDK.

C_REAL_DWIDTH (lub **c_nazwaMagistrali_dwidth**). Rzeczywista szerokość danych magistrali OPB. Domyślnie szerokość magistrali danych magistrali OPB (C_OPB_DWIDTH) jest równa 32 bity i nie może być zmieniona w środowisku EDK przez użytkownika. Dlatego w przypadku gdy użytkownik potrzebuje mniejszą szerokość magistrali danych wprowadzono dodatkowo ten parametr. Dzięki temu można zdefiniować rzeczywistą szerokość danych na magistrali OPB, pozostałe młodsze bity (o indeksie od *c_real_dwidth* do *c_opb_dwidth-1*) są na stałe podłączone do masy. Umożliwia to łatwe komunikowanie się z urządzeniami, które są podłączone do magistrali 32-bitowej jednak w rzeczywistości używają mniejszej liczby bitów (0÷c_dwidth-1). Dozwolone wartości tego parametru to 1-16, 32 bity. Warto zwrócić uwagę, że w przypadku kiedy dane urządzenie używa paru magistral zamiast słowa *real* występuje nazwa magistrali np. dla urządzenia, które posiada jedną magistralę slave i jedną magistralę master, szerokość *c_real_dwidth* po stronie slave jest nazywana *c_sl_dwidth*, a po stronie master *c_m_dwidth*. Wszystkie moduły podłączone do tej samej magistrali OPB muszą mieć ustawioną **taką samą wartość** parametru *c_real_dwidth* w przeciwnym wypadku

wystąpią błędy podczas transmisji – co jest komunikowane odpowiednim ostrzeżeniem podczas symulacji – szerokość magistrali po stronie master nie jest taka sama jak po stronie slave. W przypadku modułów dostarczonych przez firmę Xilinx szerokość *real_dwidth* jest zawsze 32-bity, w szczególności dotyczy to procesora MicroBlaze. Uwaga: Dla c_real_dwidth<=4 (lub c_dwidth<=4) zmianie ulega także szerokość magistrali adresowej, czyli adres po stronie użytkownika może być wielokrotnością adresu po stronie magistrali OPB lub na odwrót.

C_DWIDTH – wewnętrzna szerokość magistrali danych użytkownika, może przyjmować wartości 1..16 lub 32. Na przykład: użytkownik używa wewnętrznej pamięci 8-bitowej. W tym przypadku szerokość użytkownika *c_dwidth* jest 8-bitów. Jednakże moduł użytkownika jest podłączony do magistrali OPB o szerokości np. *real_dwidth*=16-bitów. *M*agistrala OPB *opb_dwidth* ma szerokość 32-bity. W tym wypadku jest dokonywana konwersja szerokości magistrali z *c_real_dwidth* na *dwidth*. Zob. parametr user_full_transfer. Zaleca się aby stosunek c_dwidth/c_real_dwidth (lub odwrotność) był potęgą liczby 2, w przeciwnym wypadku adresowanie sekwencyjne może nie działać poprawnie.

Zaleca się aby szerokość danych użytkownika ***c_dwidth*** **była taka sama jak szerokość** ***c_real_dwidth,*** ponieważ dzięki temu układ zajmuje mniejszą powierzchnię i działa szybciej - nie jest konieczne stosowanie dodatkowego skomplikowanego układu konwersji szerokość danych. Dlatego projektując system należy odpowiednio określić parametry *real_dwidth*. Co więcej w przypadku, kiedy na jednej magistrali występuje wiele modułów 32-bitowych i wiele modułów np. 8-bitowych zaleca się użycia pojedynczego układu konwersji szerokości danych *real_dwidth* (moduł opb2opb_dwidth), zamiast użycia wielu lokalnych modułów konwersji danych.

Podczas dokonywania konwersji szerokości danych (*real_dwidth != dwidth*) należy zwrócić szczególną uwagę na parametr *use_be*, *fifo_wr_awidth*, *fifo_rd_awidth*, *use_seqAddr* sygnał *seqAddr*. Warto podkreślić, że aby konwersja szerokości danych przebiegała poprawnie muszą być zastosowane dodatkowe przerzutniki (*fifo_??_awidth=1*) lub bufor fifo (*fifo_??_awidth>1*), w przeciwnym przypadku układ nie zawsze działa poprawnie. Bufory są w niektórych przypadkach dodawane automatycznie, sytuacje te podane są w poniższych sytuacjach wyjątkowych:

- Fifo_??_awidth=0, use_seqAddr=1 – powyższa konfiguracja jest zalecana w momencie kiedy moduł slave transferuje duże ilości danych oraz nie jest potrzebny dodatkowy bufor FIFO. W tym wypadku kiedy *real_dwidth != dwidth* parametry zostaną wewnętrznie zmienione na Fifo_??_awidth=4, co umożliwi poprawną zmianę szerokości magistrali danych. Warto zwrócić uwagę, że powyższa konfiguracja może powodować dodatkowe niewymuszone odczyty (fifo_rd_awidth>1) co może powodować błędy w niektórych układach.
- fifo_??_awidth=0, use_seqAddr=0, use_be=0 – powyższa konfiguracja nie działa poprawnie w przypadku kiedy *real_dwidth != dwidth* i dlatego zostanie wewnętrznie zastąpiona przez: fifo_??_awidth=1, use_seqAddr=1. Powyższa konfiguracja jest zalecana w momencie kiedy moduł slave nie musi przesyłać dużej ilości danych i służy tylko np. do konfiguracji. W tym wypadku dodatkowe rejestry służące do konwersji zostaną dołożone tylko w przypadku kiedy *real_dwidth != dwidth.*
- fifo_??_awidth=0, use_seqAddr=0, use_be=1 – w tym przypadku konwersja polega na dokonaniu pojedynczego transferu po obu stronach, czyli np. pojedynczy transfer na magistrali 16-bitowej powoduje tylko pojedynczy transfer na magistrali 8-bitowej. Konfiguracja ta może więc powodować błędy, np. w momencie kiedy liczba przesłanych bajtów po jednej stronie jest większa od drugiej, dlatego należy odpowiednio sterować sygnałami ByteEnable.

Parametry zaawansowane

W przypadku modułu *opb_master/opb_slave* istnieje wiele parametrów i wyprowadzenia, które bardzo często nie są wykorzystywane, dlatego wystarczy zapoznać się z podstawowym działaniem tych modułu, poniższe parametry i wyprowadzenia bardzo często można przyjąć jako domyślne.

C_SPEEDRATE - określa szybkość pracy układu *opb_slave*. Wartość 0 powoduje, że układ opb_slave1 działa najszybciej (brak opóźnień potokowych), odbywa się to jednak kosztem zwiększonych opóźnień (czasów propagacji) w układzie FPGA. W przypadku kiedy czasy propagacji są zbyt duże (większe niż okres zegara) należy zwiększyć ten parametr do '1' jeśli to nie pomoże to do '2'. W konsekwencji powoduje to zastosowanie architektury potokowej oraz zmniejszenie czasu propagacji kosztem zwiększenia liczby taktów zegara potrzebnych do dokonania transferu. Poniższa tabelka pokazuje dodatkowe opóźnienie (w taktach zegara) wnoszone przez moduł *opb_slave1* dla różnych parametrów *c_speedrate*

c_speedrate	seqAddr='0'	seqAddr='1'
0	0	0
1	1	0
2	1	1

transfer_size=4 (tylko master)– w przypadku transferu sekwencyjnego generowany jest sygnał Mn_busLock, który powoduje, że magistrala OPB jest zarezerwowana tylko dla tego urządzenia. Dlatego aby uniemożliwić zablokowanie magistrali na zbyt długi okres czasu po *transfer_size* przetransmitowanych danych oraz w przypadku gdy inne urządzenie żąda dostępu do magistrali, magistrala jest zwalniana. Ustawienie większej wartości tego parametru powoduje, że transfer danych przebiega szybciej (transfer blokowy jest z reguły szybszy) ale może to powodować zbyt długie przywłaszczenie magistrali. Dlatego zaleca się określenie tego parametru w granicach: 4-16, w szczególnych przypadkach więcej.

fifo_wr_awidth=0 – rozmiar wewnętrznego bufora FIFO (first-in first-out) służącego do zapisu (OPB_RNW='0'). Uwaga: Dla modułu opb_slave zobacz rozdział: podłączenie FIFO do modułu opb_slave. Dla
Moduł *opb_master/opb_slave* działa następująco dla tego parametru:
<0 – logika zapisu jest w ogóle nie używana – zaleca się ustawić *fifo_wr_awidth*<0 w momencie kiedy urządzenie *opb_master* dokonuje tylko operacje odczytu
=0 – bezpośrednie podłączenie – brak bufora FIFO – najczęściej stosowana konfiguracja, zajmująca mało zasobów układu FPGA, ale charakteryzująca się stosunkowo dużym czasem propagacji. Dlatego ta konfiguracja jest zalecana pod warunkiem, że czasy propagacji wewnątrz układu FPGA nie są zbyt duże. Warto podkreślić, że w przypadku kiedy logika danych do zapisu *dwr* pochodzi z innej magistrali OPB zaleca się ustawienie tego parametru na >0.
=1 – zamiast bufora FIFO stosowany jest pojedynczy rejestr (przerzutnik typu D), który powoduje odseparowanie czasowe logiki użytkownika od logiki magistrali OPB. Zaletą tej konfiguracji w porównaniu z *fifo_wr_awidth=0* jest zmniejszenie czasów propagacji i przez to zwiększenie maksymalnej częstotliwości zegara systemowego. Wadą natomiast więcej zajmowanych zasobów układu FPGA oraz większe opóźnienie potokowe (latency). Dla przykładu operacja zapisu zajmuje co najmniej 3 takty zegara. Omawiana konfiguracja jest zalecana dla wykonywania pojedynczych transferów (*seqAddr*=0). W przypadku adresowania

sekwencyjnego (seqAddr=1)dla fifo_wr_awidth=1 występuje mała przepustowość około 3 takty zegara na pojedynczy transfer.
>=2 – implementacja bufora FIFO o szerokości adresowej pamięci FIFO równej *fifo_wr_awidth*, czyli liczba zgromadzonych danych jest równa $2^{fifo\_wr\_awidth}$. Konfiguracja ta zajmuje więcej zasobów niż dla *fifo_wr_awidth=1* ale charakteryzuje się większymi możliwościami: możliwość magazynowania większej liczy danych, szybszy transfer blokowy (seqAddr='1'). Warto w tym miejscu podkreślić, że bufor FIFO jest w pełni wykorzystywany tylko w ramach pojedynczego transferu blokowego czyli dla sygnału *seqAddr='1'*. W przypadku braku transferu sekwencyjnego (*seqAddr='0'*) bufor może pamiętać tylko pojedynczą daną i zachowuje się tak samo jak dla parametru *fifo_wr_awidth=1* a dodatkowo zajmuje więcej zasobów FPGA. Dlatego w przypadku braku adresowania sekwencyjnego zalecane jest ustawienie tego parametru na 1. Dodatkową zaletą dla *fifo_wr_awidth>1* jest to, że transfer blokowy może odbywać się bardzo szybko – 1 transfer na jeden takt zegara. **Zalecaną wartością tego parametru jest 4 i aktualnie wartość ta nie może być większa niż 4**. Zwiększenie wartości tego parametru ponad 4 powoduje duży wzrost zajmowanych zasobów FPGA. Dla przykładu dla *fifo_wr_awidth=5*, bufor FIFO zajmuje 2 razy więcej powierzchni niż dla *fifo_wr_awidth=4*.

Dla *fifo_wr_awidth>1* bufor FIFO jest opróżniany w momencie, kiedy liczba zapisanych w nim danych jest *>= transfer_size*. Jest to spowodowane tym, że bufor próbuje łączyć poszczególne zapisy w jeden transfer blokowy. Aby temu zaradzić można albo ustawić transfer_size=1 lub *fifo_wr_awidth=1* lub też najlepiej ustawić seqAddr='0' w momencie kiedy chcemy opróżnić bufor zapisu.

fifo_rd_awidth=0 – podobnie jak dla *fifo_wr_awidth* ale dotyczy logiki odczyty (OPB_RNW=1).

single_master=0 – w przypadku kiedy do wybranej magistrali OPB jest podłączony tylko jeden moduł *master*, nie jest konieczne stosowanie dodatkowej logiki arbitrażu magistrali. W konsekwencji możliwa jest znaczna redukcja zasobów sprzętowych zajmowanych przez urządzenie *master*. Dlatego w tym przypadku zaleca się (ale nie jest to konieczne) ustawienie tego parametru na 1. W przypadku kiedy podłączone są 2 lub więcej urządzeń master konieczne jest **ustawienie tego parametru na 0** – w przeciwnym wypadku układ nie będzie działał poprawnie.

use_seqAddr=1 – użycie logiki adresowania sekwencyjnego (transferu blokowego). Dla:
=0 - logika *seqAddr* nie jest używana (OPB_seqAddr=0). Zaleca się (ale nie jest to konieczne) ustawienie tego parametru na *0* w momencie kiedy nie używa się logiki adresowania sekwencyjnego. W wyniku dodatkowo zmniejsza się zasoby układu FPGA zajmowane przez moduł *opb_master*.
=1 - logika adresowania sekwencyjnego jest używana stosownie do sygnału *seqAddr*. Warto podkreślić, że sygnał seqAddr może być aktywny w momencie kiedy sygnał sel='0' i nie kończy to sekwencji adresowania sekwencyjnego.
=2 – logika adresowania sekwencyjnego ignoruje stan sygnału seqAddr, a adresowanie sekwencyjne odbywa się na zasadzie porównywania z ostatnim adresem i sprawdzeniem czy aktualny adres jest większy od niego o 1. Nie zaleca się nadmiernego korzystania z tej opcji ponieważ logika detekcji adresowania sekwencyjnego zajmuje dodatkową powierzchnię oraz powoduje dodatkowe opóźnienie w układzie zarówno w sensie czasu propagacji jaki i dodatkowy jeden (lub więcej w przypadku zapisu) cykl martwy w stosunku do logiki użytkownika w przypadku pierwszego cyklu niesekwencyjnego. Dlatego zaleca się

stosowanie buforów FIFO w przypadku stosowania use_seqAddr=2. Dla *c_use_addr_counter>0* , c_use_seqAddr=2 nie działa prawidłowo.
Warto podkreślić, że moduł *opb_master* zakłada adresowanie sekwencyjne nawet w momencie kiedy sygnał sel='0'. Może mieć to poważne konsekwencje, ponieważ moduł *opb_master* próbuje połączyć poszczególne transfery po stronie użytkownika w jeden duży transfer blokowy po stronie magistrali OPB. Dlatego bufor zapisu zostanie opróżniony (dane zapisane na magistrali OPB) dopiero w momencie, kiedy sygnał seqAddr='0'. W przypadku *use_seqAddr=2,* kiedy ignorowany jest sygnał *seqAddr* konieczne jest zaadresowanie modułu *opb_master* nowym niesekwencyjnym adresem aby opróżnić poprzedni zapis. Alternatywnym rozwiązaniem jest ustawienie transfer_size=1 lub fifo_wr_awidth=1, co jednak pogarsza parametry transmisji danych oraz nie rozwiązuje problemu dla dwidth<bus_dwidth i use_be>0.

use_be=1 – używanie logiki Byte Enable. W przypadku kiedy dokonywane są tylko pełne transfery danych (wykorzystywana jest pełna szerokość *c_dwidth*) zaleca się (ale nie jest to konieczne) ustawienie tego parametru na 0. W przeciwnym wypadku ustaw *use_be=1,* czyli zostanie użyta logika Byte Enable co wymaga dodatkowych zasobów FPGA.

use_addr_counter =0. Ustawienie tego parametru na 1 powoduje, że zostanie użyty wewnętrzny licznik adresu. Normalnie (use_addr_couner=0) adres na magistrali OPB *OPB_ABus* jest powiązany bezpośrednio z wejściem adresowym użytkownika *a.* Bardzo często jednak adres *OPB_ABus* jest wyjściem licznika, który jest odpowiednio inkrementowany po każdym wykonanym transferze. W takim przypadku zamiast używania dodatkowego licznika, można użyć licznik wewnętrzny modułu *opb_master* poprzez ustawienie parametru *use_addr_counter=1.* Licznik ten jest ładowany synchronicznie (przy narastającym sygnale zegarowym *clk*) stanem wejścia adresowego *adr* przy sygnale *load_addr='1'.* Licznik jest automatycznie inkrementowany po każdym wykonanym transferze na magistrali OPB. Dodatkowo dla sygnału load_addr='1' zapamiętywany jest kierunek transmisji (sygnał *rnw*). Pewnym problemem jest praca tego modułu w trybie tylko do odczytu z buforem FIFO (*fifo_wr_awidth<0* oraz *fifo_rd_awidth>0*) dla którego odczyt w tym trybie jest dokonywany przed wystawieniem sygnału *sel=1*. Otóż dla *use_addr_counter=1,* moduł rozpoczyna odczyt na magistrali OPB zaraz po tym jak sygnał *load_adr='1'* i kończy w momencie wypełnienia bufora FIFO.
Dla ***use_addr_counter=2*** moduł *opb_master* nie czeka na sygnał *load_adr='1'* i rozpoczyna odczyt zaraz po konfiguracji układu FPGA od adresu 0. Można więc założyć, ze parametr *use_addr_counter=2* należy użyć w momencie kiedy licznik adresu nie jest w ogóle ładowany sygnałem *load_addr,* tylko licznik działa w kółko w pełnym zakresie 2^{awidth} lokacji adresowych.

W przypadku kiedy poprzednią operacją była operacja zapisu (**rnw='0'**) sygnał **load_adr='1'** może wystąpić tylko wtedy kiedy bufor fifo jest pusty: **fifo_empty='1'**. W przeciwnym wypadku dane zapisane do bufora FIFO zostaną utracone. W przypadku dokonywania odczytu zapis licznika adresowego może wystąpić w dowolnym momencie. Dodatkowo wejście *seqAddr* podczas transferu powinno być na stałe ustawione na '1' dla *use_adr_counter>0* ponieważ licznik z definicji dokonuje adresowania sekwencyjnego. Dla *use_be*>0 po zakończeniu transferu wejście *seqAddr* powinno być wyzerowane co spowoduje, że bufor zapisu zostanie opróżniony (dla odczyt nie ma to znaczenia). Jest to ważne tylko dla use_be>0 ponieważ, dla tej opcji moduł *opb_master* zapisuje tylko pełne dane na magistrali OPB (OPB_BE= 11..1), czyli moduł czeka aż zostaną dopisane kolejne dane lub też kiedy zostanie zakończone adresowanie sekwencyjne.

wr_pipeline_latency=0 – parametr ten określa opóźnienie potokowe pomiędzy logiką użytkownika na magistrali danych a magistralą adresową. Parametr ten może być wykorzystywany w momencie kiedy pomiędzy urządzeniem *opb_slave* a urządzeniem *opb_master* znajduje się skomplikowana logika użytkownika, która wprowadza duże opóźnienie na magistrali danych oraz nie wprowadza żadnego opóźnienia na magistrali adresowej. W tym wypadku, konieczne jest użycie architektury potokowej na magistrali danych co z kolei komplikuje logikę kontrolną. W konsekwencji wprowadzono ten parametr, który bardzo upraszcza całą logikę kontrolną. A mianowicie, użytkownik łączy urządzenia *opb_slave* i *opb_master* tak jakby w bloku danych nie było żadnego opóźnienia potokowego. Jedyną różnicą jest odpowiednie ustawienie parametru *wr_pipeline_latency*, czyli podanie liczby etapów potokowości występujących na magistrali danych pomiędzy modułem *opb_slave* i *opb_master*. Warto podkreślić że parametr ten może być wykorzystywany tylko w przypadku dokonywania operacji zapisu – czyli dane są transmitowane tylko w jednym kierunku z moduł *opb_slave* do modułu *opb_master*. Aby powyższy parametr był wydajnie wykorzystywany konieczne jest wykorzystywanie adresowania sekwencyjnego. Ponadto $2^{fifo\_wr\_awidth} > wr\_pipeline\_latency$ w przeciwnym wypadku układ nie będzie działał poprawnie.

user_full_transfer – w przypadku kiedy real_dwidth nie jest wielokrotnością dwidth (lub na odwrót), po którejś ze stron musi wystąpić niepełny transfer. Dla user_full_transfer= 1 pełny transfer występuje po stronie użytkownika, a po stronie magistrali OPB pewne bity są zero lub też są ignorowane. Przykładem może być real_dwidth= 16 oraz dwidth=20. W tym przypadku istnieją dwa przypadki: user_full_transfer=1 – w tym przypadku magistrala OPB potrzebuje 2 pełne transfery aby dokonać jeden pełny transfer po stronie użytkownika. Dla user_full_transfer=0, magistrala OPB potrzebuje tylko jeden transfer aby dokonać pojedynczego transferu na magistrali danych użytkownika. Wartością domyślną i najczęściej stosowaną jest user_full_transfer=1 – kiedy to po stronie użytkownika wszystkie bity danych są w pełni wykorzystywane.

3 Reset asynchroniczny

Układy FPGA posiadają dedykowany reset asynchroniczny, który jest aktywny zaraz po konfiguracji układu FPGA. Dlatego zalecane jest użycie dedykowanego modułu: ROC (Reset On Configuration), który aktywuje sygnał *arst* (asynchroniczny reset) zaraz po konfiguracji. Każdy przerzutnik używany w układzie FPGA powinien być zerowany (ustawiany) asynchronicznie tym sygnałem. Sygnał *arst* jest sygnałem wewnętrznym generowanym przez sam układ FPGA, dlatego aby go uzyskać należy użyć poniższej składni:
Deklaracja komponentu:

```
component ROC -- reset on Configuration
  port (O : out std_logic);
 end component;
```

Umiejscowienie komponentu najlepiej w module: opb_mój_moduł_nadrzędny:

```
global_reset: ROC
        port map (O=> arst);
```

Warto podkreślić, że element ROC jest dostępny w bibliotece unisim, którą należy dołączyć do projektu poprzez następującą komendę:

```
library UNISIM;
use unisim.all;
```

4 Urządzenie Slave

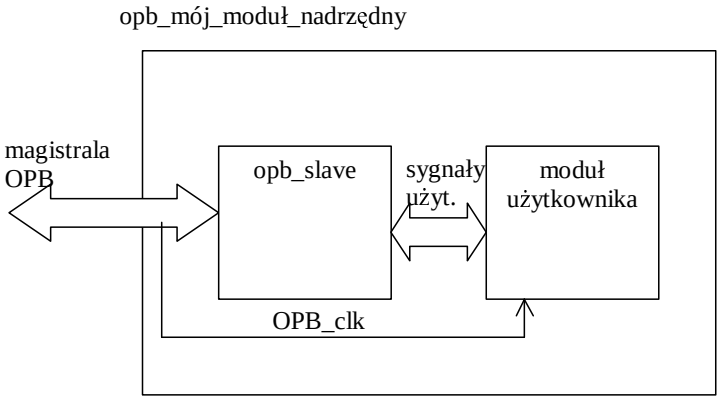
4.1 Opis modułu opb_slave.

Użytkownik powinien podłączyć bezpośrednio sygnały OPB_* oraz Sl_* do odpowiednich zewnętrznych wyprowadzeń modułu opb_slave1. Dla urządzenia Slave są to następujące sygnały magistrali OPB:

- **wejścia:** OPB_clk, OPB_rst, OPB_ABus, OPB_BE, OPB_DBus, OPB_RNW, OPB_select, OPB_seqAddr
- **wyjścia:** sl_Dbus, sl_errAck, sl_retry, sl_toutSup, sl_xferAck – wyjście może być aktywne tylko wtedy kiedy sygnał OPB_select jest aktywny oraz kiedy adres podany na magistrali OPB_ABus jest w lokacji adresowej tego urządzenia.

Dokładniejszy opis wymaganych sygnałów dla innych urządzeń znajduje się w pliku (Embeded System Tools Guide) est_guide.pdf w rozdziale Microprocessor Peripherial Discription.
Uwaga: Moduł *opb_slave* wykorzystuje wewnętrznie (bez wiedzy użytkownika) moduł *opb_master*, dlatego aby moduł *opb_slave* działał poprawnie konieczne jest dołączenie również modułu *opb_master* do projektu w Active HDL i oraz odpowiednia edycja pliku *.pao).

Połączenie modułu opb_slave w ramach modułu nadrzędnego użytkownika pokazuje poniższy rysunek. Przykład całego połączenie znajduje się np. w module opb_mem.



Moduł *opb_slave* odpowiednio przetwarza sygnały magistrali OPB i wyprowadza dalej w postaci sygnałów użytkownika. Sygnały użytkownika należy dalej odpowiednio wykorzystać w swoim module. Znaczenie tych sygnałów jest podobne jak w przypadku sygnałów OPB_* i Sl_* – różnice są podane dokładniej w opisie. Są to następujące sygnały:

sel – podobnie jak OPB_select ale jest aktywny tylko wtedy kiedy zostanie wybrane urządzenie użytkownika (adres na magistrali OPB jest w zakresie c_baseaddr÷c_highaddr). Dlatego użytkownik nie musi już dalej dekodować starsze bity magistrali adresowej, może się skupić tylko na młodszych liniach adresowych *adr* użytych w projekcie.

ack – użytkownik odpowiada na sygnał *sel* w momencie kiedy jego urządzenie jest gotowe do zapisu lub odczytu (podobnie jak sygnał Sl_xferAck).

rnw – tak samo jak *OPB_RNW* (1 – podczas cyklu odczytu, 0 – podczas cyklu zapisu)

adr – adres – podobnie jak OPB_ABus, ale kopiowane są tylko najmniej znaczące bity – ważne dla tego urządzenia. Szerokość magistrali adresowej *adr* jest określona za pomocą funkcji: *awidth*= log2(C_HIGHADDR + 1 - C_HIGHADDR).

drd – dane odczytywane z urządzenia użytkownika (ważne dla *rnw*='1'), są one następnie transmitowane do urządzenia master poprzez magistralę Sl_Dbus. Szerokość tej magistrali definiuje parametr: *dwidth* (*c_real_dwidth*).

dwr – zapisywane dane do urządzenia slave (ważne dla *rnw*='0'). Szerokość tej magistrali definiuje parametr: *dwidth* (*c_real_dwidth*).

be – Byte Enable – podobnie jak OPB_BE ale o szerokości (*c_dwidth+7)/8*.

seqAddr – sygnał taki sam jak OPB_seqAddr

Podczas osadzania modułu opb_slave w swoim projekcie należy dodatkowo obliczyć szerokość magistrali *adr* (magistrali adresowej użytkownika): *awidth*. Do tego celu stosuje się następujący kod dodany do modułu: opb_mój_moduł_nadrzędny:

```
function log2(arg: integer) return integer is
   variable ret: integer:= -1;
  begin
        if arg=0 then return 0; end if;
        for i in 0 to 32 loop
          if 2**i = arg then
                ret:= i;
                return i;
          end if;
        end loop;
        assert false -- always assert when loop does not finish the function
        report "Address space is not a power of 2"
        severity failure;
        return -1;
end log2;

  constant adr_space: integer:= conv_integer(c_highaddr - c_baseaddr + 1);
  constant awidth: integer:= log2(adr_space) - log2(8/real_dwidth) + log2(8/dwidth); -- user address
width
```

Moduł opb_slave osadza się w następujący sposób w module opb_mój_moduł_nadrzędny:

```
slave: OPB_slave
  generic map( C_OPB_AWIDTH=> C_OPB_AWIDTH, C_OPB_DWIDTH=> C_OPB_DWIDTH,
    C_BASEADDR=> C_BASEADDR, C_HIGHADDR=> C_HIGHADDR,
     C_SPEEDRATE=> C_SPEEDRATE, C_REAL_DWIDTH=> C_REAL_DWIDTH,
      -- user interface data and address width
     DWIDTH=> C_DWIDTH, AWIDTH=> awidth)
  port map(
    clk=> OPB_clk, arst=> arst,
    -- OPB signals
    OPB_ABus=> OPB_ABus, OPB_BE=> OPB_BE, OPB_RNW=> OPB_RNW,
    OPB_select=> OPB_select, OPB_seqAddr=> OPB_seqAddr,
    OPB_DBus=> OPB_DBus, Sl_DBus=> Sl_DBus,
```

```
   Sl_errAck=> Sl_errAck, Sl_retry=> Sl_retry,
   Sl_toutSup=> Sl_toutSup, Sl_xferAck=> Sl_xferAck,
         -- user signals
         sel=> sel, ack=> ack, rnw=> rnw, adr=> adr,
         drd=> drd, dwr=> dwr, be=> be, seqAddr=> seqAddr);
```

Oczywiście wcześniej należy zadeklarować component opb_slave oraz sygnały użytkownika.

4.2 Dołączenie rejestru

Dołączenie pojedynczego rejestru do modułu opb_slave1 można dokonać tego w następujący sposób:

Dołączenie pojedynczego rejestru 8-bitowego lub bez dekodowania logiki Byte Enable:

```
-- zapis do rejestru o szerokości 16-bitów
process(clk, arst) begin
  if arst='1' then reg<= (others=> '0');
  elsif clk'event and clk='1' then
         if sel='1' and rnw='0' then
                reg<= dwr;
         end if;
  end if;
end process;

drd<= reg; -- dane odczytywane
ack<= sel; -- rejestr jest zawsze gotowy do transferu
```

Dołączenie czterech rejestrów 16-bitowych (dwidth=16, awidth=2): rejestr: reg(0 to 3)(0 to 15)

```
-- zapis do rejestru
process(clk, arst) begin
  if arst='1' then reg<= (others=> '0');
  elsif clk'event and clk='1' then
         if sel='1' and rnw='0' then
           for a in 0 to 3 loop – dla każdego adresu
                if adr(awidth-3 to awidth-1)=a then – ignorowana jest najmłodsza linia adresowa
                   for i in 0 to 1 loop—Byte enable
                        if be(i)='1' then
                          reg(a)(8*i to 8*i+7)<= dwr(8*i to 8*i+7)
                        end if;
                   end loop;
                end if;
           end loop;
         end if;
  end if;
end process;

-- dane odczytywane (logika kombinacyjna)
process(adr, reg) begin
 for a in 0 to 3 loop
    if adr(awidth-3 to awidth-1)=a then – ignorowana jest najmłodsza linia adresowa: adr(awidth-1)
         drd<= reg(a);
   end if;
 end loop;
end process;
```

```
ack<= sel; -- rejestr jest zawsze gotowy do transferu
```

Warto zwrócić uwagę że podczas podłączania magistrali adresowej ignorowany jest najmłodszy bit magistrali adresowej adr(awidth-1) ponieważ rejestr jest 16-bitowy (założenie że *dwidth*=16). W przypadku rejestru 32-bitowego ignorowane byłyby dwa najmłodsze bity magistrali adresowej. Ponadto aby układ był poprawnie kompilowany należy dołączyć bibliotekę:

```
library IEEE;
use IEEE.std_logic_unsigned.all;
```

4.3 Dołączenie pamięci

Dołączenie pamięci blokowej mem (szerokość magistrali danych dwidth=8), szerokości magistrali adresowej (awidth=8). Czas oczekiwania podczas zapisu: 0 –cykli zegara, czas oczekiwania podczas odczytu 1-cykl zegara.

Podłączenie pamięci:

```
  mema: mem -- BRAM module
    generic map (AWidth=> awidth-2, DWidth=> 8)
    port map (clk=> opb_clk, we=> mem_we,
        adr=> adr(0 to awidth),
         din=> dwr, dout=> drd);
```

Logika sterująca:

```
mem_we<= sel and not rnw and BE(0);
-- przerzutnik rd_ready (sygnał wykorzystywany tylko podczas odczytu i świadczy o tym, że stan pamięć jest gotowa do odczytu
rd_rdy: process(OPB_clk, arst) begin
  if arst='1' then
        rd_ready<= '0';
 elsif opb_clk='1' and opb_clk'event then
        rd_ready<= sel and rnw and not rd_ready;
 end if;
end process;
ack<= sel when RNW='0' -- no wait state for a write
         else rd_ready; -- one wait state for a read
```

W przypadku podłączenia pamięci BRAM logika sterująca jest już bardziej zaawansowana, ponieważ w przypadku odczytu konieczne są dwa takty zegara. W tym celu dołączono dodatkowy rejestr *rd_ready*, który podczas pierwszego cyklu odczytu jest równy zero, natomiast podczas drugiego cyklu odczytu jest równy ‘1’. Sygnał *ack* (gotowości) jest aktywny natychmiast podczas cyklu zapisu (tak jak sygnał *stb*) oraz aktywny po jednym takcie zegara podczas cyklu odczytu (tak jak sygnał: *rd_ready*).

4.4 Podłączenie bufora FIFO (First-In First-Out)

Moduł *opb_slave* posiada wewnętrzny bufor FIFO, który można użyć ustawiając fifo_wr_awidth>1 i/lub fifo_rd_awidth>1. Jednakże bufor FIFO ustawiony w ten sposób powoduje tylko przyspieszenie operacji transferu na magistrali OPB. Nie może być on używany jako dodatkowa logika magazynująca dane użytkownika przed ich wysłaniem (odczytem) przez magistralę OPB.

W niektórych sytuacjach konieczne jest użycie bufora FIFO w celu magazynowania danych przed ich wysłaniem. Przykładem może być moduł klawiatury, UART, które to

moduły poprzez odpowiednie odpytywanie lub też poprzez generacje sygnału przerwania wymuszają na odpowiednim module master odczyt danych. Jednakże pomiędzy generacją przerwania a odczytem danych może upłynąć dodatkowy czas, który spowoduje, że następne dane pochodzące z klawiatury UART'u będą nadpisywały poprzednie nie odczytane jeszcze dane. W tym celu należy stosować zewnętrzny bufor FIFO, którego podłączenie zostanie poniżej opisane. W tym miejscu należy podkreślić, że w przypadku zapisu (OPB_RNW=0) możliwe jest użycie wewnętrznego bufora FIFO, jednak powoduje to, że w momencie kiedy bufor FIFO nie jest pusty, ścieżka odczytu jest niedostępna. Dodatkowym problemem jest to, że bufor FIFO jest wypełniany tylko w ramach pojedynczego transferu blokowego (zob. use_seqAddr).

Aby podłączyć zewnętrzny moduł FIFO należy użyć gotowego modułu, który ma następujące parametry i wyprowadzenia:

```
entity fifo is
  generic( adr_width: integer:= 4; -- internal memory address width - 4 is preferable for distributed
rams, 0- direct connection
  data_width: integer:= 8); -- data bus width
  port (clk: in std_logic; -- clock
    arst: in std_logic; -- global asynchronous reset
    rst: in std_logic; -- synchronous reset for the whole device
    almost_full, almost_empty: out std_logic; -- only one data can be accepted (transfered in or out)
    full: buffer std_logic;
    half_full: out std_logic; -- fifo is half full
    -- data input signals (slave)
  datI_I: in std_logic_vector(0 to data_width-1);
  stbI_I: in std_logic; -- strobe similar as OPB_select (master ready to transfer data)
 ackI_O: out std_logic; -- acknowledge
-- data output signals (master)
 datO_O: out std_logic_vector(0 to data_width-1);
stbO_O: out std_logic; -- FIFO has valid data when stbO_O='1'
ackO_I: in std_logic);
end fifo;
```

Moduł FIFO stosuje następującą sygnały kontrolne: **stb** – aktywny podobnie jak sygnał *sel* na magisrali OPB, kiedy urządzenie master jest gotowe do wysłania danych, **ack** – (podobnie jak OPB_xferAck) aktywny w momencie kiedy urządzenie *slave* jest gotowe do pobrania danych – jest to reakcja na aktywny sygnał *stb*. Moduł FIFO jest modułem *slave* po stronie wejściowej (dane są zapisywane do FIFO) oraz modułem *master* po stronie wyjściowej (dane są transmitowane na zewnątrz urządzenia FIFO). Sygnał *stbO_O* w stanie '1' świadczy, że w module FIFO są zapisane jakieś dane, w stanie '0' świadczy że moduł FIFO jest pusty. Sygnał *ack_I_O* po stronie wejściowej jest aktywny tylko wtedy kiedy sygnał *stbI_I='1'* oraz FIFO nie jest pełne.

Dodatkowo dodano sygnały *full*- aktywny wtedy kiedy FIFO jest zapełnione i nie jest w stanie przyjąć więcej danych; *almost_full* – aktywny wtedy kiedy fifo jest w stanie przyjąć co najwyżej jedną daną lub też FIFO jest pełne; *almost_empty* – w buforze FIFO znajduje się tylko jedna dana; *empty*- (sygnał tożsamy z zanegowanym sygnałem *stbO_O*) bufor FIFO jest pusty; *half_full*- bufor FIFO jest w połowie (lub więcej) zapełniony.

Tylko odczyt

Aby podłączyć bufor FIFO do modułu *opb_slave* w celu dokonania tylko odczytu przez magistralę OPB należy zastosować następującą logikę: Moduł opb_slave jest podłączony do modułu FIFO (część master) w następujący sposób:
opb_slave1 ⇔ FIFO
drd <= datO_O
ack <= sel and rnw and stbO_O
ackO_I<= ack

Tylko zapis
Z reguły należy stosować moduł FIFO wewnętrzny (fifo_wr_awidth>1). Ale w niektórych sytuacjach konieczne jest użycie FIFO zewnętrznego również do zapisu. Aby podłączyć bufor FIFO do modułu *opb_slave* w celu dokonania tylko zapisu przez magistralę OPB, należy zastosować następującą logikę: Moduł opb_slave1 jest podłączony do modułu FIFO (część slave) w następujący sposób:
FIFO ⇔ opb_slave1
datI_I<= dwr
stbI_I<= sel and not rnw (and be)
ackI_O=> ack

Część master FIFO powinna być podłączona do dalszej części logiki użytkownika.
Dodatkowo można generować sygnał OPB_retry aby uniknąć dodatkowych cykli oczekiwania w momencie kiedy FIFO jest pełne. Można to zrobić stosując dodatkową logikę:
retry<= full

5 Urządzenie Master

5.1 Opis modułu opb_master

Moduł *opb_master* należy podłączyć w podobny sposób jak urządzenie *opb_slave1*. Od strony użytkownika urządzenie *opb_master* posiada następujące sygnały:

sel – (wejście) częściowo odpowiednik sygnału *Mn_select*. Sygnał jest ustawiany w stan '1' przez użytkownika w momencie kiedy użytkownik chce dokonać transmisji na magistrali OPB. Ustawienie sygnału *sel* powoduje, że urządzenie master wystawi żądanie przyznania magistrali OPB. W przypadku ustawienia sygnału *sel* użytkownik musi wystawić ważny adres na magistrali *adr*, wielkość transferu – logika Byte Enable *be*, ważny sygnał *seqAddr* oraz ważne dane na magistrali *dwr* w przypadku zapisu lub też być gotowym na odebranie danych z magistrali *drd* w przypadku dokonywania cyklu odczytu.

ack – (wyjście) odpowiednik *OPB_xferAck*. Sygnał świadczący o tym, że aktualny cykl magistrali zostanie zakończony pomyślnie i na magistrali OPB zostanie w tym cyklu zegarowym dokonany transfer danych. Jest to odpowiedź urządzenia *slave* (po drugiej stronie magistrali OPB) na aktywny sygnał *sel* wystawiony przez użytkownika.

rnw – (wejście) odpowiednik *OPB_RNW*. Sygnał wystawiany przez użytkownika i świadczący o rodzaju transferu jaki ma zostać wykonany: (1 - podczas odczytu, 0 - podczas zapisu)

adr – (wejście) podobnie jak *OPB_ABus*. Adres wystawiany przez użytkownika, pod który ma być dokonany zapis lub odczyt.

do – (wyjście) dane odczytywane z magistrali OPB. Ważne tylko podczas cyklu odczytu *rnw='1'* oraz kiedy *ack='1'*. Szerokość tej magistrali definiuje parametr: *dwidth* (c_real_dwidth).

di – (wejście) dane wystawiane przez użytkownika oraz zapisywane przez urządzenie master do urządzenia slave,. Szerokość tej magistrali definiuje parametr: *dwidth* (c_dwidth).

be – (wejście) Byte Enable – definiuje szerokość aktywnego transferu, sygnał wystawiany przez użytkownika podczas aktywnego sygnału *sel*.

seqAddr – (wejście) podobnie jak *OPB_seqAddr* definiuje czy użytkownik chce dokonać transferu blokowego z adresowaniem sekwencyjnym. Wykonanie transferu blokowego wydatnie poprawia szybkość transmisji i dlatego powinno być używane w przypadku przesyłania dużej ilości danych. Ustawienie *seqAddr* powoduje również zarezerwowanie magistrali OPB poprzez wystawienie sygnału OPB_busLock. Wewnętrzna logika modułu *opb_master* kontroluje jednak czas posiadania magistrali OPB, dlatego sygnał seqAddr może być przetrzymywany dowolnie długo w stanie 1. Podobnie wewnętrzna logika kontroluje magistralę adresową i rozpoznaje stan przepełnienia, kiedy to adr(0 to awidth-1)= 111..1 i przechodzi w stan 0. W tym wypadku seqAddr może być cały czas równy '1' ponieważ wewnętrzna logika wykryje stan przepełnienia i odpowiednio wysteruje sygnał M_seqAddr. Logika ta działa dla awidth<=24.

msb_adr – Magistrala *OPB_ABus* jest 32-bitowa a przestrzeń adresowa używana przez dane urządzenie (określona parametrem awidth) jest bardzo często mniejsza np. 18-bitowa w przypadku obrazu monochromatycznego 512×512. Dlatego konieczne jest podanie adresu bazowego (najbardziej znaczących bitów adresu – msb_adr), ponieważ np. obraz może być zapisany pod adres: 0_0000 ÷ 3_FFFF lub 4_0000 ÷ 7_FFFF, itd.

Zaawansowane sygnały urządzenia master

Sygnały:
load_addr:= 0 – wejście służące do załadowania licznika adresu. Sygnał ten jest używany tylko dla *use_addr_counter*>0.

fifo_empty, fifo_full – wyjścia pokazujące stan wewnętrznego bufora FIFO, *fifo_empty=1* gdy bufor FIFO jest pusty, fifo_full=1 w momencie kiedy bufor FIFO jest pełny i nie może przyjąć więcej danych. Zob. **fifo_wr_awidth oraz fifo_rd_awidth**

fifo_almost_empty, fifo_almost_full – wyjścia pokazujące stan wewnętrzny bufora FIFO. fifo_almost_empty – pokazuje, że w buforze FIFO znajduje się co najwyżej 1 dana, czyli FIFO może być odczytane tylko raz. fifo_almost_full pokazuje, że bufor jest prawie pełny i może być przyjęta co najwyżej jedna dana. Sygnały te mogą być użyte do wystawienia linii seqAddr innego urządzenia master – czyli w momencie kiedy FIFO jest prawie pusty to sygnał seqAddr powinien być 0 podczas odczyty z FIFO, podobnie sygnał seqAddr=0 gdy sygnał fifo_almost_full=1 podczas zapisu FIFO.

5.2 Pojedynczy zapis

Dokonanie pojedynczego zapisu na magistrali OPB:
Ustaw: sel=1, rnw=0, seqAdr=0, be, adr, di
Czekaj aż pojawi się sygnał ack

5.3 Zapis blokowy

Ustaw sel=1, rnw=0, seq_adr=1, be, adr, di
Czekaj aż pojawi się sygnał ack.
W następnym cyklu zegara: ustaw sel=1, rnw=0, seqAdr=1, adr+1(2, lub 4), di, be
Czekaj aż pojawi się sygnał ack.
W przypadku dokonywania ostatniego zapisu zaleca się aby seqAdr=0.

Licznik adresu jest inkrementowany w zależności od wielkości transferu: w przypadku transferu 1-bajtowago o 1, w przypadku transferu 16-bitowego o 2, w przypadku transferu 32-bitowego o 4. Zob. parametr *use_addr_conter*.

5.4 Pojedynczy odczyt

Ustaw sel=1, rnw=1, seqAdr=0, adr,
Czekaj aż pojawi się sygnał *ack* i zapisz dane *do* wraz z narastającym sygnałem zegarowym

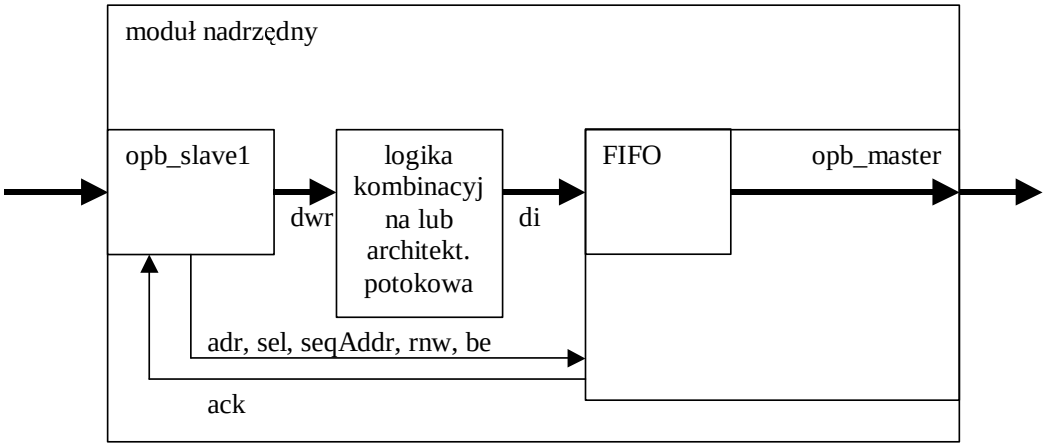
5.5 Użycie wewnętrznego licznika adresów

W przypadku kiedy urządzenie opb_master powinno zapisywać/odczytywać po kolei wszystkie lokacje adresowe z inkrementacją co 1 należy użyć parametru use_addr_counter>0 co spowoduje użycie wewnętrznego licznika z wpisem równoległym.
Dla use_addr_counter>0 procedura postępowania jest następująca:

1) Podaj na magistralę adresową *a* adres początkowy oraz wystaw sygnał load_addr='1' (wejście synchroniczne). W ten sposób nastąpi załadowanie licznika adresowego przy narastającym sygnale *clk*. Procedura ta może być pominięta w przypadku kiedy licznik zawsze liczy od zera oraz parametr use_addr_counter=2. Pamiętaj, że wystawienie sygnału load_addr=1 powoduje wyzerowanie bufora FIFO do zapisu.
2) Dokonuj odczytów/zapisów podobnie jak w poprzednich punktach, jedyną różnicą jest to, że stan magistrali adresowej *a* jest ignorowany – adres jest podawany z wewnętrznego licznika adresu inkrementowanego odpowiednio do rozmiaru transferu (o 1 dla dwidth=8, o 2 dla dwidth=16 lub o 4 dla dwidth=32)

5.6 Podłączenie urządzenia Slave i Master za pomocą dodatkowej logiki użytkownika (tylko zapis)

Bardzo często urządzenie *opb_slave* i urządzenie *opb_master* mają być podłączone ze sobą poprzez logikę kombinacyjną lub architekturę potokową. Przykładem takiego układu jest np. mnożenie przez stały współczynnik, dla którego dana wejściowa jest podawana przez urządzenie opb_slave a wynik mnożenia jest wysyłany poprzez urządzenie opb_master. Warto podkreślić, że zaleca się do tego typu połączenia stosowanie kolejki FIFO znajdującej się wewnątrz modułu *opb_master* ze względu na bardzo duże czasy propagacji – sygnał danych i adresów propaguje z magistrali OPB wejściowej (po stronie opb_slave), poprzez logikę użytkownika do magistrali OPB wyjściowej (po stronie modułu opb_master).



Sposób połączenia modułów opb_slave i opb_master, dla których transfer danych jest wykonywany tylko od urządzenia opb_slave do urządzenia opb_master (czyli tylko zapis), jest następujący:

```
sl: OPB_slave1
  generic map (C_OPB_AWIDTH=> C_OPB_AWIDTH, C_OPB_DWIDTH=> C_OPB_DWIDTH,
    C_BASEADDR=> C_BASEADDR,   C_HIGHADDR=> C_HIGHADDR
    C_SPEEDRATE=> C_SPEEDRATE,  USE_SeqAddr=> C_USE_SeqAddr
    DWIDTH=> C_DWIDTH,  AWIDTH=>AWIDTH) -- awidth=log2(c_highaddr+1-c_baseaddr)
  port map ( clk=> OPB_clk,  arst=>arst-- (od modułu ROC – asynchroniczny reset)
    -- OPB signals (nie umieszczone tutaj)
   .....
    -- user signals
  sel=> sel,  ack=>ack,  rnw=> rnw,  adr=> adr,
  drd=> drd,  dwr=> zeros,  be=>be, seqAddr=> seqAddr );

log_kombin: user_logic
  (din=> dwr, dout=> di)

mn: opb_master
  generic map(opb_dwidth=> c_opb_dwidth, dwidth=> c_dwidth,
    opb_awidth=> c_opb_awidth, awidth=> awidth
    transfer_size=> c_transfer_size, use_be=>0,
   fifo_wr_awidth=4, fifo_rd_awidth=>-1, wr_pipeline_latency=> user_pipeline_latency)
 port map(clk=>opb_clk, arst=> arst,
-- OPB Master interface (nie umieszczony tutaj)
…….
   -- internal simplified bus
 di=> md, do=> open,   be=> be,
 a=> adr, msb_adr=> c_baseAddr,
 rnw=> rnw, sel=> sel, seqAddr=> seqAddr, ack=> ack);
```

Wszystkie stałe z przedrostkiem C_* są wyprowadzane na zewnątrz modułu nadrzędnego użytkownika za pomocą słowa kluczowego *generic* i mogą być ustawiane w programie EDK.

W przypadku kiedy logika użytkownika posiada dodatkowe rejestry potokowe na ścieżce danych jedyna dodatkową czynnością jaką należy wykonać jest ustawienie parametru

wr_pipeline_latency odpowiednio do liczby etapów potokowości (liczby rejestrów) umieszczonych w logice użytkownika. Dodatkowa logika modułu *opb_master* odpowiednio zsynchronizuje opóźnione dane wejściowe *di* z magistralę adresową oraz wejściami/wyjściami kontrolnymi. Warto podkreślić, że aby FIFO działało poprawnie wydajnie się używania adresowania sekwencyjnego.