Paweł DUC
EVATRONIX SA
ul. Przybyły 2,43-300 Bielsko-Biała

Sprzętowa weryfikacja funkcjonalna magistrali AMBA® AXI

Mgr inż. Paweł DUC

Ukończył studia na wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki Akademii Górniczo-Hutniczej w Krakowie. Jest projektantem układów scalonych w firmie EVATRONIX SA w Bielsku-Białej. Jego zainteresowania naukowe koncentrują się wokół nowoczesnych metod weryfikacji funkcjonalnej i formalnej bloków IP oraz systemów on-chip jak również sprzętowej realizacji algorytmów fuzzy logic.

e-mail: pawel.duc@evatronix.com

Streszczenie

W artykule przedstawiono moduł monitora magistrali AMBA® AXI, umożliwiający weryfikację poprawności oraz weryfikację *functional coverage* protokołu AXI w systemach ko-emulacji sprzętowo-programowej układów SoC (System-on-Chip). Układ monitora składa się z syntezowalnej części sprzętowej oraz części programowej. Część sprzętowa służy do bezpośredniej obserwacji stanu magistrali i zawiera podstawowe elementy weryfikacyjne, zaś część programowa umożliwia komunikację części sprzętowej z programowym środowiskiem weryfikacyjnym.

Słowa kluczowe: AMBA, AXI, Sce-Mi, sprzętowa weryfikacja funkcjonalna, functional coverage.

Functional hardware verification of AMBA® AXI bus

Abstract

The currently observed increase in SoC (System-on-Chip) system complexity determines evolution of the verification methods to ensure complete and as fast as possible verification of the whole system correctness. One of the main direction in development of the complex SoC design verification methodology is implementation of hardware accelerated systems in the verification process. There is a number of ways used in this kind of verification. One is the transaction based hardware-software co-emulation, that support high level software test environment to control and observe the hardware implementation of design under test. This paper presents the AMBA® AXI bus monitor for using in co-emulation systems, with particular attention paid to the Sce-Mi based systems. The monitor architecture has two parts, hardware and software. The synthesizable hardware part is implemented in a programmable device of the emulator system and is used to direct bus observation through basic checkers. The task of the monitor software part is to enable proper configuration of the hardware part, to receive verification status information, to perform more sophisticated checking and to report verification results. Communication between the hardware and software parts is based on exchange of message vectors through a message channel known from the co-emulation Sce-Mi standard.

Keywords: AMBA, AXI, Sce-Mi, functional hardware verification, functional coverage.

1. Wstęp

Rosnąca złożoność układów scalonych, wynikająca z rozwoju produktów elektronicznych przeznaczonych dla wydajnych aplikacji, jak na przykład aplikacje multimedialne, wymusza równoczesny rozwój metod i narzędzi pozwalających na ich wyczerpującą weryfikację na etapie projektowania. Obecne systemy on-chip (SoC) integrują w swojej strukturze kompletne platformy sprzętowe, zarządzane przez wydajne procesory korzystające z bardzo pojemnych pamięci oraz komunikujące się z otoczeniem za pomocą dużej ilości zewnętrznych interfejsów dostępnych poprzez moduły peryferyjne. Ich skomplikowanie połączone z równoczesnym dążeniem do jak najkrótszego czasu potrzebnego na przejście drogi od projektu do produktu sprawia, że klasyczne symulacyjne metody weryfikacji stały się niewystarczające.

Przyspieszenie procesu weryfikacji uzyskuje się stosując sprzętowe metody akceleracji. Jedną z nich jest zastosowanie transakcyjnych środowisk ko-emulacji bazujących na platformach sprzętowo-programowych. Umożliwiają one implementację weryfikowanego układu w strukturach programowalnych, przy równoczesnym zapewnieniu kontroli nad procesem weryfikacji z poziomu środowiska programowego [1].

Celem artykułu jest przedstawienie elementu programowo-sprzętowego, umożliwiającego lepsze wykorzystanie zalet transakcyjnych środowisk ko-emulacji poprzez przeniesienie weryfikacji interfejsu AMBA AXI do środowiska sprzętowego.

2. Magistrala systemowa AMBA AXI

Kluczową rolę w systemach on-chip odgrywa magistrala systemowa. Istnieje wiele różnych standardów definiujących zbiór cech charakterystycznych magistrali, takich jak zestaw sygnałów czy protokół wymiany danych. Jednym z najbardziej rozpowszechnionych jest wprowadzony przez firmę ARM standard AMBA® (ang. *Advanced Microcontroller Bus Architecture*). Określa on sposób komunikacji w systemach on-chip, wprowadzając zbiór standardowych protokołów, przeznaczonych dla różnych podsystemów bazującego na nim układu [2]:

- AHB (*Advanced High-performance Bus*),
- ASB (*Advanced System Bus*),
- APB (*Advanced Peripheral Bus*).

W roku 2003 firma ARM zdefiniowała w ramach standardu AMBA nowy interfejs oraz bazujący na nim protokół wymiany danych na wewnątrz-układowych magistralach systemowych, nazwany AXI - *Advanced eXtensible Interface*. Standard protokołu AXI przeznaczony jest dla szybkich i wydajnych systemów scalonych, wymagających dużych przepustowości przy niewielkich opóźnieniach (ang. *latency*) [3].

Główne cechy charakteryzujące protokół AXI, to[3]:

- oddzielenie kanałów przesyłania adresów i danych kontrolnych od kanałów danych,
- wsparcie dla transferów danych przy użyciu adresów niewyrównanych do szerokości magistrali danych, poprzez użycie linii aktywujących poszczególne bajty słowa,
- transakcje typu *burst* jako podstawowy typ transferu, z przesyłaniem jedynie adresu początkowego,
- oddzielne kanały do zapisu i odczytu danych, w celu umożliwienia implementacji ograniczonych modułów DMA, wykorzystujących jeden kierunek przesyłu danych,
- transakcje typu *outstanding*.

W roku 2010 wydana została druga wersja protokołu AXI, wprowadzająca pewne zmiany wynikające z dotychczasowych doświadczeń. Najważniejsze z nich dotyczą rozszerzenia maksymalnej długości transakcji *burst* z 16 do 256 transferów oraz usunięcia transakcji zapisu o zmienionej kolejności wykonywania (ang. *write interleaving*).

3. Standard modelowania interfejsu ko-emulacji Sce-Mi

Standard modelowania interfejsu w systemach ko-emulacji Sce-Mi (and. *Standard Co-Emulation Modeling Interface*)[4] został zdefiniowany przez organizację Accellera w celu ujednolicania sposobów komunikacji w systemach emulacji sprzętowo-programowej.

Specyfikacja Sce-Mi definiuje interfejs wielokanałowej komunikacji, pozwalającej na połączenie wysokopoziomowego programowego środowiska weryfikacyjnego ze sprzętową implemen-

tacją weryfikowanego układu. Sce-Mi określa trzy modele mechanizmów wymiany danych:

- model przekazywania wiadomości oparty na makrach – ang. *Macro-based Message Passing Use Model,*
- model komunikacji oparty na funkcjach – ang. *Function-based Use Model,*
- model komunikacji strumieniowej – ang. *Pipes-based Use Model.*

Podstawowym modelem używanym w ko-emulacji jest model wymiany wiadomości oparty na makrach. Są to tzw. wiadomości bezczasowe (ang. *untimed messages*), co oznacza, że z punktu widzenia czasu emulowanego, moment wysłania wiadomości odpowiada momentowi jej odebrania po drugiej stronie kanału komunikacyjnego. Niezbędnym elementem środowiska wykorzystującego ten model komunikacji są sprzętowe transaktory, odpowiednio sterujące interfejsami sygnałowymi emulowanego układu.

W celu stworzenia kanałów komunikacji pomiędzy środowiskiem programowym a transaktorami, w modelu sprzętowym instancjonowane są tzw. makra. Są to zdefiniowane przez standard bloki funkcjonalne, umożliwiające dostęp do kanałów komunikacyjnych oraz sterowanie sygnałami zegarowymi i zerującymi infrastruktury sprzętowej. Makra Sce-Mi są zastępowane elementami realizującymi ich funkcjonalności przez narzędzie programowe, określone w specyfikacji Sce-Mi jako *infrastrukture linker*. Standard definiuje cztery rodzaje makr:

- *SceMiMessageInPort* – port wejściowy służący do odbierania wiadomości od środowiska programowego,
- *SceMiMessageOutPort* – port wyjściowy służący do wysyłania wiadomości do środowiska programowego,
- *SceMiClockPort* – generator kontrolowanego sygnału zegarowego oraz kontrolowanego sygnału reset,
- *SceMiClockControl* – moduł sterowania zegarem kontrolowanym.

Interfejs bloków dostępowych kanałów komunikacyjnych, implementujących funkcjonalność portów wiadomości, bazuje na protokole *dual-ready*. Oprócz magistrali danych, która w tym przypadku ma szerokość równą szerokości wektora wiadomości w kanale, posiada on dwie jednobitowe linie *ready*. Jedna z linii jest sterowana z modułu nadającego wiadomość, zaś druga z modułu odbierającego. Transfer wektora wiadomości uznaje się za przeprowadzony w cyklu zegara niekontrolowanego, w którym obydwie linie przyjęły aktywny stan wysoki.

W systemie może zostać zaimplementowana większa ilość kontrolowanych sygnałów zegarowych. Zatrzymanie jednego z nich powoduje zatrzymanie pozostałych. Kontrolowanie sygnałów zegarowych umożliwia wstrzymywanie pracy części sprzętowej na czas realizacji komunikacji z programową części środowiska.

Istotnym elementem części sprzętowej są tzw. niekontrolowane sygnały zegarowy i zerujący. Niekontrolowany sygnał zegarowy jest to sygnał dostarczany przez infrastrukturę Sce-Mi za pomocą funkcjonalności makra *SceMiClockControl*. Służy on do taktowania interfejsów wymiany wektorów wiadomości i nigdy nie jest blokowany. Niekontrolowany sygnał zerujący jest generowany automatycznie na początku emulacji i służy do prawidłowej inicjalizacji pracy modułów komunikacyjnych oraz transaktorów.

Od strony programowej realizację komunikacji zapewnia zdefiniowany przez Sce-Mi i implementowany w języku C++ interfejs programowania aplikacji (API). Umożliwia on dostęp do kanałów komunikacyjnych z częścią sprzętową. Implementację programowego interfejsu Sce-Mi, podobnie jak implementację elementów sprzętowych, tworzy dostawca środowiska Sce-Mi.

4. Architektura monitora magistrali AXI

W celu umożliwienia weryfikacji poprawności protokołu magistrali AXI w systemach sprzętowych, powstał blok monitora tej magistrali przystosowany do implementacji w środowiskach ko-emulacji sprzętowo-programowej.

Blok monitora AXI został zaimplementowany w sposób umożliwiający dostosowanie go do użycia w różnych systemach ko-emulacji, zapewniających jednak pewne wymagane minimum cech. Sprzętowa część modułu monitora posiada dwa porty wymiany wiadomości zgodne z protokołem dual-ready oraz linię gotowości na sygnał zegarowy taktujący magistralę AXI, nie zawiera jednak żadnych makr Sce-Mi. Dzięki temu może zostać użyty również w systemach sprzętowych nie korzystających ze standardu Sce-Mi, umożliwiających jednak komunikację za pomocą protokołu dual-ready oraz blokowanie sygnału zegarowego magistrali AXI. W przypadku stosowania monitora w systemie opartym na standardzie Sce-Mi, odpowiednie makra powinny zostać umieszczone w module łączącym, tzw. *bridge*.

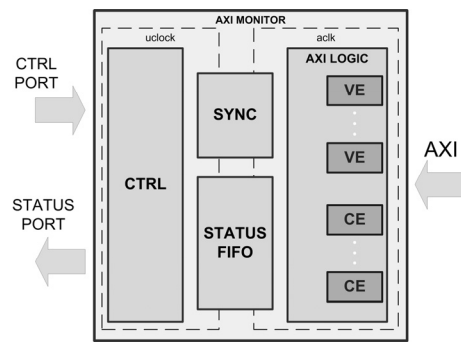
Część programowa modułu została zaimplementowana jako klasa języka C++, która powinna być wykorzystana jako obiekt wewnętrzny, używany za pomocą zewnętrznego wrappera, dostosowanego do konkretnego systemu ko-emulacji.

Część sprzętowa

Sprzętową część monitora stanowi syntezowalny moduł, opisany w języku Verilog. Jego schemat funkcjonalny został przedstawiony na rys. 1.

W skład syntezowalnego bloku monitora wchodzą cztery główne bloki funkcjonalne:

- CTRL – moduł kontrolny,
- AXI LOGIC – blok weryfikacyjny magistrali AXI,
- STATUS FIFO – pamięć FIFO,
- SYNC – blok synchronizacji międzydomenowej.



Rys. 1. Schemat funkcjonalny sprzętowej części monitora AXI
Fig. 1. Functional diagram of AXI monitor

Moduł kontrolny CTRL odpowiada za komunikację z częścią programową poprzez odbieranie wiadomości kontrolnych z portu wejściowego CTRL PORT, konfigurację modułu weryfikacyjnej w module AXI LOGIC oraz wysyłanie informacji na temat przebiegu weryfikacji za pomocą portu wyjściowego STATUS PORT. Blok ten działa w domenie zegara niekontrolowanego *uclock* infrastruktury Sce-Mi. W przypadku implementacji monitora w systemie bazującym na innym sposobie komunikacji, sygnał zegarowy taktujący moduł kontrolny musi zapewniać prawidłowe funkcjonowanie interfejsów CTRL PORT i STATUS PORT.

Moduł kontrolny posiada sygnał wyjściowy służący do blokowania sygnału zegarowego magistrali AXI. Przyjmuje ona stan niski, kiedy monitor nie jest gotowy do pracy, np. gdy pamięć FIFO danych statusowych jest pełna i nie może przyjąć kolejnych zapisów, zanim moduł kontrolny nie odbierze i nie prześle przynajmniej części z dotychczas zapisanych w niej przez moduł weryfikacyjny, informacji. W systemach opartych na standardzie Sce-Mi sygnał ten jest podawany na wejście gotowości na zegar kontrolowany modułu *SceMiClockControl*. W innych systemach może być użyty w analogiczny sposób, w najprostszym przypadku do bramkowania sygnału zegarowego magistrali AXI.

Moduł AXI LOGIC stanowi główną część bloku monitora. Na podstawie parametrów konfiguracyjnych monitoruje stan poszczególnych kanałów magistrali AXI i w przypadku wykrycia błędu protokołu umieszcza w kolejce STATUS FIFO odpowiednią informację statusową, która następnie jest wysyłana przez moduł kontrolny do środowiska programowego.

Moduł logiki weryfikacyjnej zawiera elementy weryfikacyjne VE (ang. *checkers*), które sprawdzają poprawność konkretnych cech protokołu AXI. Przykładem takiej cechy jest wymóg, aby ustawiony w stan wysoki sygnał gotowości danych adresowych w kanale adresowym zapisu (AWVALID) pozostał w stanie wysokim dopóty, dopóki sygnał gotowości zapisu (AWREADY) również nie przyjmie stanu wysokiego [3]. W przypadku stwierdzenia przez element weryfikacyjny naruszenia tego wymogu, w kolejce informacji statusowych zostaje umieszczona odpowiednia informacja, która zostaje odebrana przez moduł kontrolny i wysłana wyjściowym kanałem komunikacyjnym do części programowej monitora. Tam informacja ta zostaje zdekodowana, a na konsoli lub w pliku raportu pojawia się informacja:

```
AMBA AXI Monitor Checker Failed at xx ACLK cycle after ARESETN
    AXI_EM_AWVALID_STABLE:  AWVALID must stay asserted until
                            AWREADY is asserted
```

gdzie `xx` jest numerem cyklu zegara magistrali AXI *aclk* licząc od zbocza dezaktywującego sygnał zerowania *areset*, w którym wykryte zostało naruszenie reguły, zaś `AXI_EM_AWVALID_STABLE` jest nazwą elementu weryfikacyjnego.

Prócz elementów weryfikacyjnych, umożliwiających sprawdzenie poprawności protokołu transakcji na magistrali AXI, monitor zawiera zbiór liczników pokrycia (ang. *coverage*) CE, liczących ilość wystąpień zdefiniowanych poprawnych transakcji na monitorowanym interfejsie. Przykładem takiej transakcji jest transfer adresu początkowego i danych kontrolnych inicjujących transakcję typu *burst fixed read* w kanale adresowym odczytu. Przesłanie danych z elementów CE do środowiska programowego odbywa się po wysłaniu do modułu sprzętowego wiadomości kontrolnej z żądaniem wysłania danych *coverage*. Najczęściej operację tę wykonuje się po zakończeniu testu funkcjonalnego. Odebrane dane są zapisywane w pliku wyjściowym w odpowiednim formacie. Raport pokrycia może też zostać wyświetlony na konsoli.

Po przeprowadzeniu wszystkich testów, raporty pokrycia, wygenerowane po każdym z nich, są analizowane przez zewnętrzną aplikację, która łączy wyniki z wszystkich testów i generuje raport końcowy z procesu weryfikacji. Raport ten pokazuje, które z funkcjonalności magistrali AXI, badane przez elementy CE, zostały wykryte podczas procesu weryfikacji. Umożliwia to zbadanie tzw. *functional coverage*, czyli stwierdzenie faktu czy wszystkie z badanych funkcjonalności opisujących magistralę AXI, zaistniały podczas testów. Dla przykładowej funkcjonalności transakcji w kanale adresowym odczytu odpowiednia linia raportu będzie wyglądać następująco:

```
Fixed read address transactions            : xx
```

gdzie xx oznacza liczbę prawidłowo zainicjalizowanych transakcji odczytu spod stałego adresu.

Moduł STATUS FIFO służy do gromadzenia informacji o naruszeniach protokołu AXI wykrytych przez moduł weryfikacyjny, przed wysłaniem ich przez moduł kontrolny do środowiska programowego. Umożliwia ona płynną pracę części sprzętowej monitora zapewniając prawidłową synchronizację danych pomiędzy domenami zegarów *uclock* i *aclk*. W przypadku wystąpienia dużej liczby naruszeń w krótkim czasie, zawartość pamięci FIFO przekroczy poziom określony przez jej parametr *almost full* co spowoduje zablokowanie sygnału zegarowego magistrali AXI *aclk* przez moduł kontrolny. Dzięki temu będzie on mógł zwolnić miejsce w kolejce poprzez przesłanie zgromadzonych informacji do środowiska programowego, a następnie odblokować zegar AXI i umożliwić dalszą pracę magistrali.

Blok synchronizacji międzydomenowej SYNC zapewnia prawidłową synchronizację sygnałów kontrolnych pomiędzy domenami *uclock* i *aclk*. Zastosowanie monitora w systemach opartych na standardzie Sce-Mi nie wymaga stosowania takiej synchronizacji ze względu na fakt, iż zbocza narastające zegarów aclk i uclock są ze sobą zsynchronizowane. Blok synchronizacji został zaimplementowany w celu umożliwienia zastosowania monitora AXI w systemach innych niż te, bazujące na Sce-Mi.

Część programowa

Jak to już zostało wspomniane, część programową monitora stanowi klasa języka C++ o nazwie *saes_axi_monitor*, której implementacja może zostać ukryta w bibliotece łączonej z systemem. Dzięki zastosowaniu w niej natywnych konstrukcji języka C++, monitor może być wykorzystany zarówno w środowiskach korzystających z biblioteki SystemC, jak również w typowo programowych środowiskach implementowanych w języku C++. Użycie monitora umożliwia dostarczany plik nagłówkowy deklarujący klasę monitora oraz typ wyliczeniowy *saes_axi_mode* służący do konfiguracji trybu pracy monitora. Tryb pracy monitora określa podzbiory aktywnych elementów weryfikacyjnych. Dzięki temu możliwe jest wykorzystanie monitora do ograniczonej weryfikacji interfejsu AXI, np. interfejsu korzystającego tylko z kanałów odczytu danych. Dezaktywacja niektórych elementów weryfikacyjnych sprawia, że wynik ich działania jest pomijany w raportowaniu.

W zależności od docelowego systemu, w którym ma pracować monitor, obiekt klasy *saes_axi_monitor* musi zostać zaimplementowany jako część obiektu zewnętrznego (*wrappera*), który posiada dostęp do infrastruktury komunikacyjnej ko-emulatora, np. elementów programowych Sce-Mi, oraz udostępnia klasie *saes_axi_monitor* wskaźniki do strumieni wyjściowych raportowania. Klasa *saes_axi_monitor* z kolei udostępnia obiektom zewnętrznym swoje pola i metody, umożliwiające prawidłową pracę monitora.

W celu przeprowadzenia testu funkcjonalnego z użyciem Monitora AXI, obiekt zewnętrzny (wrapper) w pierwszej kolejności konfiguruje obiekt klasy *saes_axi_monitor* poprzez aktualizację pól składowych odpowiednimi wartościami oraz przesyła do części sprzętowej wektory kontrolne wypełnione w wewnętrznej tablicy wiadomości wejściowych obiektu klasy *saes_axi_monitor* za pomocą metod *SetParameters* i *SetMode*. Po zakończeniu testu wysyła wiadomość kontrolną żądania przesłania danych *coverage* sformowaną za pomocą metody *SetCovReq*.

Po każdorazowym odebraniu wektora wiadomości z kanału statusu, obiekt wrappera wpisuje go do tablicy wiadomości wyjściowych obiektu klasy *saes_axi_monitor* i wywołuje metodę *OutService*. Mechanizm ten działa niezależnie od rodzaju wiadomości statusowej, gdyż w wiadomościach statusowych zawarta jest informacja o rodzaju wiadomości, zaś metoda *OutService* rozpoznaje ją i wykonuje zadania odpowiednie do otrzymanego rodzaju informacji.

5. Podsumowanie

Zastosowanie opartej na transakcjach ko-emulacji sprzętowo-programowej umożliwia znaczne skrócenie procesu testowania systemów on-chip lub złożonych bloków IP. Przedstawiony monitor, opracowany przez Autora w firmie EVATRONIX, pozwala na sprawdzenie poprawności protokołu interfejsu AXI wewnątrz implementacji sprzętowej weryfikowanych modeli. Dotychczas tego typu weryfikacja była domeną funkcjonalnej weryfikacji symulacyjnej lub weryfikacji formalnej. Dzięki prezentowanemu rozwiązaniu, możliwe jest ograniczenie długotrwałego procesu symulacji złożonych modeli i jeszcze lepsze wykorzystanie zalet systemów ko-emulacji.

6. Literatura

[1] Garcia L., Vreelland R.: Transaction-based Co-Modeling Changes SoC Verification Methodology. Mentor Graphics Technical Library, 2003.
[2] AMBA® Specification, Rev 2.0, ARM, 1999.
[3] AMBA® AXI Protocol ver. 2.0 Specification, ARM, 2010.
[4] Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual ver. 2.0, Accellera, 2007.

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*