Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.31
DOI: https://doi.org/10.24867/15BE11Mihajlovic

PROJEKTOVANJE UNIVERZALNE VERIFIKACIONE KOMPONENTE ZA APB PROTOKOL PRIMJENOM UVM METODOLOGIJE

DEVELOPING UNIVERSAL VERIFICATION COMPONENT FOR APB PROTOCOL USING UVM METODOLOGY

Jovana Mihajlović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Zadatak rada jeste projektovanje verifikacione komponente za APB protokol korišćenjem UVM metologije. Programski jezik je SystemVerilog. U radu je dat uvod u SystemVerilog jezik i UVM metodologiju. Na kraju su prikazani rezultati testiranja i funkcionalni coverage.*
Ključne reči: *UVM, SystemVerilog, APB protocol, UVC, functional verification*
Abstract: *The task of this paper is developing the verification component for APB protocol using UVM metodology. Programming language is SystemVerilog. The paper contains inroduction into SystemVerilog and UVM. Also, results of test scenarios and functional coverage are presented.*
Ključne reči: *UVM, SystemVerilog, APB protocol, UVC, functional verification*

1. UVOD

Pri projektovanju savremenih digitalnih sistema, kao posljedica povećanja njihove kompleksnosti funkcionalna verifikacija je postala jedan od najzahtjevnijih koraka.
Funkcionalna verifikacija ima zadak da provjeri funkcionalnu tačnost dizajna u odnosu na specifikaciju. Pri tome, funkcionalna verifikacija se ne bavi tačnošću specifikacije koja se već smatra tačnom.
Tokom poslednjih godina uvedeni su novi standardi i metodologije za verifikaciju, kao što je UVM metodologija (Universal Verification Methodology).
U ovom master radu će biti prikazana primjena UVM metodologije na primjeru projektovanja univerzalne verifikacione komponente za APB (Advanced Peripheral Bus) protokol. Cilj rada je izrada verifikacione komponente koja se kasnije može upotrijebiti prilikom verifikacije proizvoljnog sistema sa AMBA (Advanced Microcontroller Bus Architecture) magistralom koji koristi APB protokol. Univerzalna verifikaciona komponenta će biti opisana korišćenjem SystemVerilog jezika.
Specifičnosti SystemVerilog jezika koji se koristi i u verifikaciji i za dizajn biće predstavljene u drugom poglavlju.
U ovom radu biti će predstavljena i sama UVM metodologija.
U četvrtom poglavlju će biti opisan APB protokol.
Nakon toga će biti predstavljena APB UVC komponenta i rezultati testiranja.

NAPOMENA:
Ovaj rad proistekao je iz master rada, čiji mentor je bio prof. dr Darko Marčetić.

2. SystemVerilog

SystemVerilog predstavlja prvi industrijski HDVL (Hardware Design and Verification Language) jezik. Koristi se kako za modelovanje i dizajn, tako i za verifikaciju u industriji poluprovodnika. On čini kombinaciju jezika za dizajniranje kao što su Verilog i VHDL sa mogućnostima jezika specijalizovanih za verifikaciju (OpenVera, e), uz uobičajene programske jezike C i C++. Zasnovan je na Verilogu.
2005. godine postaje IEEE (Institute of Electrical and Electronics Engineers) standard – 1800 IEEE, a 2009. je dodatno unaprijeđen.
Kako vrijeme prolazi, ovaj objektno orjentisani jezik pronalazi sve veću primjenu u RTL (Register Transfer Level) dizajniranju, verifikaciji zasnovanoj na assertionima kao i u izgradnji okruženja za constrained random verifikaciju.
Operatori u SystemVerilog-u su mješavina Verilog i C operatora. U oba jezika tip i veličina operatora su fiksni, pa je tako i u SystemVerilog-u.
Za provjeru protokola i ponašanja sistema u SystemVerilog jeziku se koriste assertion-i. Oni se uglavnom koriste za provjeru dizajna, ali mogu biti korišćeni i za prikupljanje functional coverage-a.
Njihova prednost je što umanjuju vrijeme potrebno za debug i brže dovode do njihovog otkrivanja, imaju mogućnost interagovanja sa C funkcijama i imaju podesiv "severity" ili nivo značaja.
Mogu biti korišćeni za:

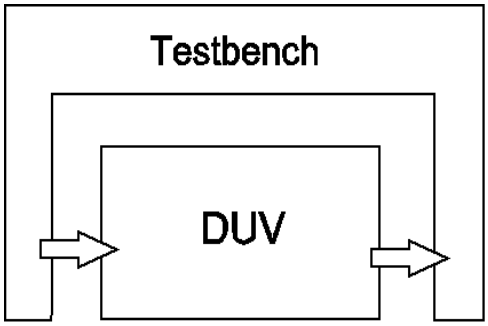
- provjeru nekih logičkih uslova
- provjeru ispravnosti sekvenci korišćenjem vremenskih izraza "temporal expressions"
- provjeru nedefinisanih stanja signala (X,Z)
- "onehot" provjera …

Assertions mogu da se podijele u dvije grupe: "immediate assertions" i "concurrent assertions".

3. UVM METODOLOGIJA

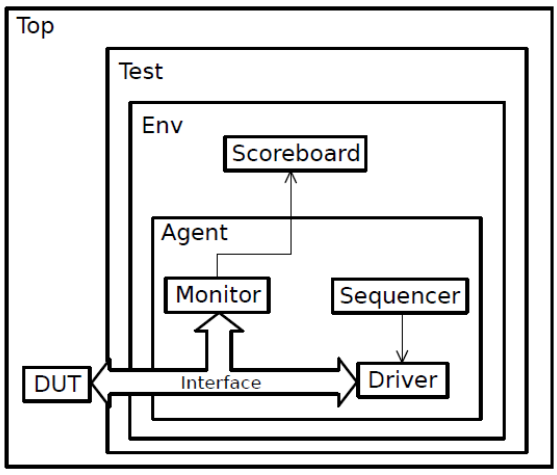
Prilikom dizajniranja elektronskih kola se koriste HDL (Hardware Description Language) jezici, od kojih su najpoznatiji Verilog i VHDL, koji omogućavaju da se apstraktni modeli i tražena funkcionalnost prevedu na „pravi" hardware. Međutim kada se ovaj deo posla završi, postavlja se pitanje da li implemetirano zaista radi ono što je specifikacijom predviđeno. Odgovor treba da da proces verifikacije, čiji početak u praksi nikada ne čeka kraj dizajniranja, već se ta dva procesa odvijaju paralelno.
Proces verifikacije se obavlja kreiranjem virtuelnog okruženja oko dizajna (DUV – Device Under Verifica-

tion), koje omogućava slanje stimulusa i analizu njegovog ponašanja na isti. Ovo okruženje se naziva *testbench.*



Slika.3.1 DUV i Testbench

Proces „funkcionalne verifikacije“ zahteva da se u testbench implementira ista funkcionalnost koja se očekuje od dizajna i da se na osnovu poređenja ponašanja utvrdi njegova ispravnost.
Tradicionalni pristup je bio pisanje direktnih testova, koji proveravaju neku tačno određenu situaciju. Ovakav način testiranja ima linearno povećanje potrebnog vremena sa povećanjem kompleksnosti dizajna. Kako kompleksnost elektronskih kola raste iz godine u godinu vreme za verifikaciju ovom metodom više nije prihvatljivo i bilo je neophodno osmisliti način koji će moći brže da dovede do rezultata.
Rešenje je pronađeno u obliku „constraint random verification“ metoda, koje se zasnivaju na slanju nasumičnih stimulusa u okviru nekih, unaprijed isplaniranih ograničenja. Da bi ovo bilo omogućeno, neophodno je automatizovati proces generisanja stimulusa, kao i blokova koji će pratiti pruozrokovane izlaze, blokova koji predviđaju kakvi izlazi treba da budu i blokova koji će vršiti analizu i poređenje.
Takođe, zbog same prirode nasumičnog saobraćaja, neophodno je i nekako pratiti koje situacije su testirane.
Još jedna prednost constraint random verification metoda jeste to što generisanjem nasumičnog saobraćaja omogućavaju pogađanje situacija koje eventualno nisu predviđene, a koje se mogu ispostaviti kao značajne.
Da bi bilo omogućeno ponovno korišćenje testbencha sa minimalnim promenama pri prelasku sa verifikacije jednog, na drugi dizajn, neophodno je standardizovati implementaciju svih neophodnih blokova.
Zasnovan na objektno-orjentisanom jeziku, SystemVerilog-u, UVM poseduje čitav niz klasa koje omogućavaju implementaciju cijelog testbench-a, koji će se od projekta do projekta uglavnom razlikovati samo po broju instanciranih UVC-eva (Universal Verification Component) i po dodavanju koda koji je specifičan za konkretnu specifikaciju/protokol koji je potrebno verifikovati.
Svi blokovi koji se koriste pri izgradnji testbench-a nastaju korišćenjem klasa koje su izvedene od neke od osnovnih, i dodavanjem koda koji će je učiniti primenljivom za konkretan slučaj zadate specifikacije.
Klase mogu biti podeljene u tri glavne grupe, uvm_component koje su korišćene za izgradnju hijerarhiske strukture testbench-a, uvm_object koje se koriste za strukture podataka za konfigurisanje testbench-a, i uvm_transaction koje se koriste za generisanje stimulusa i analizu i prikupljanje coverage-a.



Slika.3.2 Struktura testbench-a

Svaka od ovih klasa već ima ugrađene metode u skladu sa namjenom za koju su predviđene, a moguće je dodavanje novih u izvedenim klasama.
„Top level“ klasa u UVM testbench-u je označena kao test klasa, i ona je odgovorna za konfigurisanje testbench-a, iniciranje izgradnje sledećeg nivoa u hijerarhiji i iniciranje slanja stimulusa pokretanjem glavne sekvence. Uobičajeno nastaje nasleđivanjem test_base klase.
Sama struktura UVM testbench-a je modularno izgrađena da bi omogućila ponovno iskorišćenje koda i u drugim projektima, „horizontal reuse“, ili i u višem nivou hijerarhije unutar istog projekta, „vertical reuse“. Dvije osnovne komponente koje omogućavaju ponovno korišćenje su samo okruženje – env, i agent (koji bi trebalo da opisuje ponašanje nekog protokola ili modula).

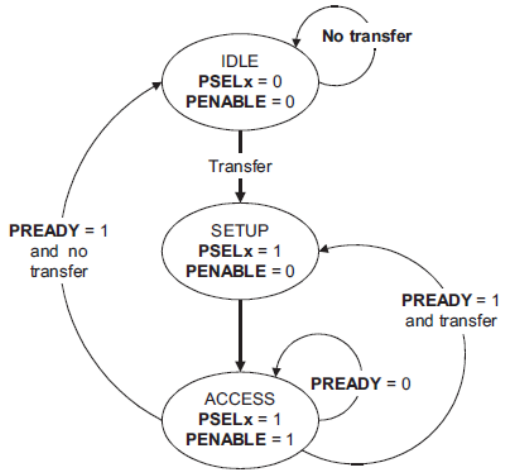
4. APB PROTOKOL

Advanced Peripheral Bus (APB) je dio Advanced Microcontroller Bus Architecture (AMBA) familije protokola. On definiše low-cost interfejs koji je optimizovan za minimalnu potrošnju energije i predstavlja interfejs smanjene kompleksnosti.
APB ne posjeduje pipeline, i predviđen je za low-bandwidth periferije koje ne zahtjevaju visoke performanse AXI protokola.
Da bi se olakšala integracija APB periferija u bilo koji dizajn, on je predviđen da na svaku uzlaznu ivicu takta obavlja po jednu promjenu. Za svaki transfer su neophodna bar dva takta.
APB može da se poveže sa:

- AMBA Advanced High-performance Bus (AHB)
- AMBA Advanced High-performance Bus Lite (AHB-Lite)
- AMBA Advanced Extensible Interface (AXI)
- AMBA Advanced Extensible Interface Lite (AXI4-Lite)

Kao primjer, može se koristiti za pristup programabilnim kontrolnim registrima perifernih uređaja.
Sam protokol je unaprijeđen par puta u odnosu na prvu verziju predstavljenu 1998. godine, a verzija koja će biti korišćena u ovom radu je verzija 2.0, koja u trenutku pisanja predstavlja najnoviju verziju.
APB protokol poseduje dve nezavisne magistrale podataka, od kojijh se jedna koristi za čitanje, a druga za upis podataka. Magistrale mogu biti maksimalne širine 32

bita. Pošto poseduju zajedničke handshake signale, nije moguće istovremeno prosleđivanje podataka na obje strane.



Slika 4.1 Prelazi mašine stanja

Mašina stanja prolazi kroz naredna stanja:

- IDLE - Ovo je inicijalno stanje, u koje se prelazi kada nema transfera.
- SETUP - Iniciranje transfera inicira prelaz u ovo stanje. U ovom stanju se postavlja odgovarajući selekcioni signal, PSELx. Magistrala ostaje u ovom stanju samo jedan ciklus takta i odmah zatim prelazi u ACCESS stanje na sledeću uzlaznu ivicu takta.
- ACCESS - Signal dozvole, PENABLE, se postavlja tokom ovog stanja. Adresa, kontrolni signali i podatak za upis moraju biti stabilni tokom ovog prelaza iz SETUP stanja.
- Prelaz iz ovog stanja se kontroliše signalom PREADY:
- Ukoliko je PREADY na niskom nivou, magistrala ostaje u ACCESS stanju.
- Ukoliko slave uređaj postavi PREADY na visoku vrijednost, onda se izlazi iz ACCESS stanja i prelazi u IDLE stanje ukoliko nema više zahtjeva za transferima. Ako postoji zahtjev za novim transferom, prelazi se direktno u SETUP stanje.

5. PROJEKTOVANJE UNIVERZALNE VERIFIKACIONE KOMPONENTE

Agent je osnovna komponenta u kojoj se kreiraju sve podkomponente:

- konfiguracija,
- interface,
- monitor,
- driver,
- sequencer.

Za prosleđivanje konfiguracije podkomponentama je iskorišćen konfiguracioni objekat.

hdh_apb_config m_cfg;

On je postavljen u bazu podataka od strane neke komponente više u hijerarhiji (u ovom konkretnom primjeru, u env komponenti), a ovdje se dohvata zarad konfigurisanja samog agenta.

Ukoliko konfiguracija svakog agenta (master i slave) nije setovana, podrazumijevaće se default konfiguracija.

Interface nije predstavljen na slici 5.1 budući da predstavlja komponentu koju je potrebno instancirati odvojeno u testbench-u. Nakon toga se dodijeli određenom agentu, te se na taj način povezuju.

virtual hdh_apb_if HDH_APB;

Pošto je predviđeno da ovaj UVC osim monitorisanja ispravnosti saobraćaja (pasivan UVC) bude u mogućnosti i da šalje saobraćaj (aktivan UVC), osim monitor-a neophodno je kreirati i driver i sequencer komponente.

Za sve agente u verifikacionom okruženju (VE), odgovarajući konfiguracioni objekat (tipa hdh_apb_config class) se kreira u build fazi. Isti objekat se koristi i za master i za slave agenta. Nakon što su konfiguracioni objekti kreirani, konfiguracionim poljima se dodijeljuju određene vrijednosti.

U virtuelni interface su dodati signali definisani protokolom.

Pomoću define-ova su definisane širine signala na magistrali. Oni se postavljaju u hdh_apb_types.svh fajlu.

Interface sadrži dva input porta: PRESETn i PCLK. Potrebno je da testbench drive- uje PRESETn kao active low kako bi saobraćaj bio moguć.

Za monitorisanje APB magistrale kreirane su dvije monitor komponente – master i slave monitor. Master monitor provjerava ispravnost signala koje slave postavlja i obrnuto, slave monitor provjerava ispravnost signala koje master postavlja i kontroliše promijene vrijednosti na magistrali.

Master monitor sačeka da PSEL bude postavljen, a zatim i PREADY . Pročita vrijednosti ostalih signala i upakuje ih u odgovarajuću transakciju. Transakcija se preko analysis port- a šalje u scoreboard.

Sve ovo se dešava u forever begin ... end petlji što omogućava hvatanje svih transakcija na magistrali.

Korišćenjem fork join petlje u paraleli se izvršavaju taskovi za provjeru ispravnosti saobraćaja.

Slave monitor sačeka da PSEL bude postavljen, a zatim provjerava da li vrijednost PSEL-a odgovara slave_id – u. Ukoliko odgovara, nastavlja se monitoring jer to znači da je on taj slave kojeg je master selektovao. Nakon toga sačeka da PREADY bude postavljen, te kao i master monitor pročita vrijednosti ostalih signala i upakuje ih u odgovarajuću transakciju. Transakcija se preko analysis port- a šalje u scoreboard.

U oba monitora se sve ovo dešava u forever begin ... end petlji što omogućava hvatanje svih transakcija na magistrali.

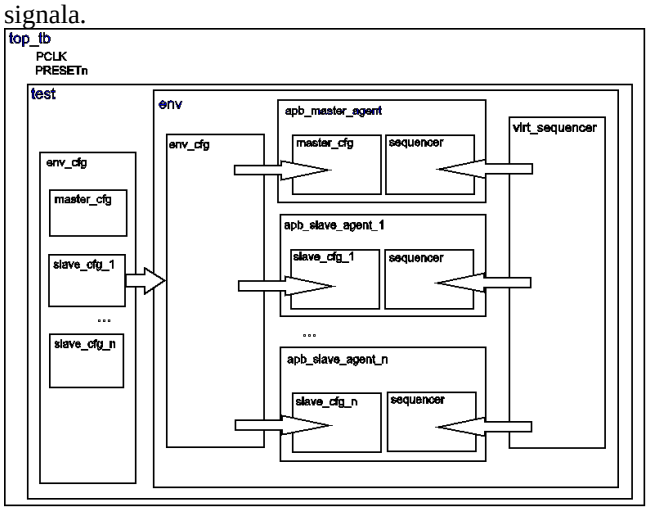
Korišćenjem fork join petlje u paraleli se izvršavaju taskovi za provjeru ispravnosti saobraćaja.

Kreiraju se dvije driver klase, jedna za master, druga za slave uređaje, gdje se pri samom kreiranju UVC-a u zavisnosti od konfiguracije kreira driver jednog ili drugog tipa.

U slučaju da uređaj radi kao master, prvo se inicijalizuje magistrala postavljanjem nekativnih vrijednosti kontrolnih signala. A zatim se u forever begin..end petlji dohvati sequence_item i pokrene samo postavljanje signala na magistralu u skladu sa protokolom.

Da bi uporedo bio podržan i reset sistema, prethodno opisan proces je ubačen u fork..join_any strukturu kao proces koji se izvršava u paraleli zajedno sa taskom

reset_signals. Ovaj task obavlja reset vrijednosti na žicama magistrale u slučaju da PRESETn signal bude postavljen na aktivnu vrijednost.
U slučaju kada driver radi kao slave, prvo se inicijalizuje vrijednost signala koje postavlja slave uređaj. A zatim se na sličan način kao na master strani u forever begin..end petlji u paraleli, korišćenjem fork..join_any strukture pokreću procesi za reset i za postavljanje signala na magistrali.
Pošto slave uređaj ne inicira saobraćaj, već praktično samo odgovara na zahtev od master uređaja, driver čeka da preko PSEL signala prepozna da je on slave kome je zahtjev poslat, a zatim u zavisnosti od tipa transakcije (read/write) odgovara na zahtjev. U slučaju write transakcije vrijednost sa PWDATA signala upisuje u asocijativni niz koji služi da simulira memoriju, i nasumično postavlja kontrolne signale kao odgovor.
U slučaju read transakcije, provjerava da li za zadatu adresu već ima unijet podatak u asocijatvni niz (što bi značilo da je u nekom prethodnom trenutku poslata write transakcija sa istom adresom) i ukoliko ima, tu vrijednost postavlja na PRDATA signale. Ukoliko ne postoji prethodni unos, na PRDATA signale postavlja nasumično izabran podatak (koji bi mogao da simulira vrijednost na memorijskoj lokaciji na koju nikada nije izvršen upis nekog smislenog podatka).
Da bi testiranje bilo omogućeno neophodno je postojanje testbench fajla. U njemu se napravi modul u kome bi bio instanciran DUT i povezani njegovi signali na virtuelni interface, ili u ovom slučaju u nedostatku pravog RTL modula kreira se samo virtuelni interface preko koga će biti povezani master i slave APB agenti.
Kreirani virtuelni interface se postavlja u bazu podataka da bi komponente niže u hijerarhiji mogle da ga dohvate.
U testbench- u se obavlja i generisanje clock i reset signala.



Slika 5.1. Dijagram okruženja za testiranje

Kako bi se pokrila funkcionalnost i omogućilo pogađanje situacija zadatih preko coverage- a, kreirani su testovi:

- u2u_read_test
- u2u_write_test
- u2u_slverr_test
- u2u_random_test
- u2u_multislave_test.

Svi testovi koriste hdh_apb_base_sequence.
Za pokretanje sekvenci u testu je neophodno kreirati samu sekvencu, obaviti dodjelu nasumičnih vrijednosti variablama i na kraju pozivom start metode pokrenuti sekvencu.
Pošto se coverage sakuplja samo kada postoji validna transakcija nije bilo moguće pogoditi bin PSEL = 0, iako je takva situacija očigledno postojala. Kada neka situacija nije pogođena u coverage-u, neophodno je to „opravdati“ u dokumentaciji inače se proces verifikacije ne može smatrati potpunim.
U ovom slučaju to je opravdano, i ova vrijednost bin-a bi mogla da bude isključena iz računanja konačnog coverage-a. Isključenje bin-a je ovdje uradjeno iz samog koda. Druga opcije jeste iskljucenje bin-a pomocu samog alata.
Takođe, iz koda je isključen bin cross coverage-a koji provjerava da li je tokom READ transfera PSTRB imao vrijedsnost koja nije nula. Opravdano je isključenje ovog bin-a budući da nije po protokolu. Druga opcija je da se napiše kao illlegal bin, te bi u slučaju narušavanja protokola simulacija bila završena. Pošto već imamo provjeru za PSTRB tokom READ transfera, ovdje to nije urađeno.

6. ZAKLJUČAK

U ovom radu dat je primjer izrade jedne reusable komponente primjenom UVM metodologije.
UVM metodologija predstavlja jedan od rezultata razvoja constrained random verifikacije. Random verifikacija omogućava lakše pokrivanje šireg spektra scenarija.
Ovakav pristup omogućava ubrzavanje procesa verifikacije, te je dominantan u poslednjih 20 godina.
APB UVC implementira APB protokol i sve njegove feature. Podržana je potpuna APB protokol specifikacija: jedan master i vise slave- ova, transakcije upisa i čitanja sa i bez čekanja, slave error response, zaštita pristupa, strobe signal. Definisanje metrike za praćenje napretka verifikacije omogućeno je coverage- em.
Funkcionalna pokrivenost definisana je u master komponenti. Rezultati coverage-a pokazuju 100% pokrivenost. Takođe, rezultati regresije pokazuju prolaznost svih testiranih scenarija.
Verifikacioni plan je urađen pomoću vPlan programa. U njemu se nalazi spisak svih test scenarija, coverage item-a i checkera. Svi item- i su mapirani u odnosu na specifikaciju.

7. LITERATURA

[1] www.verificationacademy.com
[2] https://www.elektronika.ftn.uns.ac.rs/funkcionalna-verifikacija-hardvera/
[3] www.doulos.com
[4] Universal Verification Methodology (UVM) 1.1 User’s Guide
[5] Universal Verification Methodology (UVM) 1.1 Class Reference
[6] SystemVerilog 3.1a Language Reference Manual
[7] AMBA APB Protocol Version 2.0 Specification