Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.3
DOI: https://doi.org/10.24867/01BE10Lasu

PROJEKTOVANJE VERIFIKACIONOG OKRUŽENJA ZA APB2SPI NA UVM METODOLOGIJI

DESIGN VERIFICATION ENVIRONMENT FOR APB2SPI BASED ON UVM METHODOLOGY

Čongor Lašu, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Zadatak master rada jeste projektovanje verifikacionog okruženja za modul koji je APB2SPI, zasnovano na UVM metodologiji, programski jezik je SystemVerilog. Zadatak je pisanje dokumenata koji su verifikacioni plan i verifikaciona arhitektura, razvijanje UVC-eva, i razvijanje testova koji proveravaju osobine dizajna.*

Ključne reči: *Funkcionalna verifikacija, APB2SPI, UVM, pokrivenost*

Abstract – *The task of master thesis is developing verification environment for the module which is APB2SPI, based on UVM methodology and SystemVerilog programming language is used. To make documents which are verification plan and verification architecture, developing UVCs and developing tests which will check the features of design.*

Keywords: *Functional verification, APB2SPI, UVM, coverage*

1. UVOD

Verifikacija kao zadatak ima da potvrdi da logika dizajna radi usklađeno sa dizajniranom specifikacijom. Prevedeno, verifikacija pokušava da da odgovor na pitanje: "Da li ovaj predloženi dizajn radi ono što ste mislili?" Ovaj zadatak je složen, i oduzima dosta vremena i napora u većini projekata tokom projektovanja elektronskih sistema.

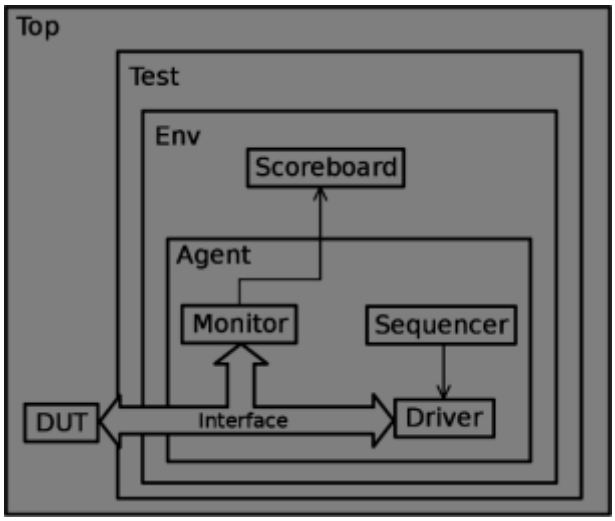
Funkcionalna verifikacija je definisana kao proces provere da li RTL dizajn (Verilog, VHDL, SystemVerilog) ispunjava svoje specifikacije iz funkcionalne perspektive. RTL verifikacija se obično deli na dve diskretne oblasti, funkcionalna verifikacija i fizička verifikacija. Funkcionalna verifikacija utvrđuje da *DUT* (Design Under Test) pravilno implementira funkcionalnost specifikacije. Fizička verifikacija proverava da sinteza, implementacija i protok protokola održavaju istu funkcionalnost u svakom nivou apstrakcije. Funkcionalna verifikacija obično predstavlja jednu od najizazovnijih oblasti u dizajnu čipa.

Kako se veličina dizajna povećava, tako se povećava i složenost. Zbog ogromnog broja potencijalnih stanja dizajna, u velikim dizajnima, funkcionalna verifikacija obično nije u mogućnosti da detaljno proveri dizajn. Iz pomenutog razloga uvedena je nova metrika, koja se zove pokrivenost (eng. Coverage).

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Rastislav Struharik, vanredni profesor.

2. UVM (Universal Verification Methodology)

Universal Verification Methodology (UVM) je standardizovana metodologija za verifikaciju dizajna integrisanih kola. UVM je izvedena uglavnom iz OVM-a (Open Verification Methodology). UVM je jedna SystemVerilog biblioteka, koja omogućava lako kreiranje fleksibilnih, i ponovo upotrebljivih verifikacionih komponenti. Verifikaciono okruženje se sastoji od velikog broja klasa, gde se svaka klasa koristi sa specifičnom svrhom, koje su grupisane tako da se povećava modularnost i portabilnost celog okruženja. Ove SystemVerilog klase su implementirane unutar UVM biblioteke i predstavljaju verifikacione komponente kao što su agenti, monitori, sekvence itd. Više informacija se može naći u knjigama [1,2]. Jednostavan primer verifikacionog okruženja, koje poštuju UVM pravila, je moguće videti na *Slika 1*. Kao što se može videti, sve komponente se nalaze na nižem nivou, dok se test klasa, koja obuhvata sve komponente, nalazi na najvišem nivou.



Slika 1 : *UVM verifikaciono okruženje*

U nastavku će biti opisane glavne komponente verifikacionog okruženja:

- Sequencer, sequences - Sequencer predstavlja jedan napredni stimulus generator, koji obezbeđuje sekvence (sequence item) driver-u na izvršenje.

Sequencer vrši kontrolu generisanja slučajnog stimulusa izvršavajući sekvence. Jednostavna sekvenca će generisati jedan ili više slučajnih data item-a. Kompleksne sekvence mogu sadržati informacije o tajmingu, parametrima i dodatnim ograničenjima.

- Driver - Driver je jedna aktivna komponenta koja oponaša logiku koja pokreće DUT. Tipični driver uzima podatke generisanih od strane sequencer-a, i prosleđuje ih DUT-u i scoreboard-u na dalju obradu.

- Monitor - Monitor je jedna pasivna komponenta koja sakuplja vrednosti sa izlaza DUT-a. Ova komponenta je zadužena za sakupljanje informacija o pokrivenosti, i proveru protokola. Nakon obrade primljenih podataka, monitor vrši pakovanje ovih podataka, i preko analysis port-ova prosleđuje scoreboard komponenti.
- Agent - Agent vrši enkapsulaciju sequencer-a, driver-a i monitor-a u smislenu celine. Verifikaciono okruženje može sadržati više agent-a. Neki agent-i su aktivni, i vrše instanciranje transakcija u DUT, dok pasivni agent-i samo posmatraju ponašanje DUT-a. Agent-i treba da budu konfigurabilni, kako bi mogli biti ili pasivni ili aktivni.
- Interface – Služi kao stvarna veza između DUT-a i verifikacionog okruženja. Skup mreže ili žice.
- Scoreboard - Scoreboard predstavlja mehanizam, koji se koristi za dinamičko predviđanje odziva DUT-a, i radi upoređivanje ovih odziva naspram posmatranih odziva na izlazu DUT-a. Ona u suštini sadrži funkcionalnost DUT-a, tj. Referentni model, koje je bitan za trenutni test scenario.
- Environment - UVM komponenta koja objedinjuje jedan ili više agent-a (ili UVC-a), zajedno sa konfiguracionom objektom. Svrha environment-a je da instancira prethodno pomenute komponente, konfiguriše ih i da ostvari konekcije između njih. Konfiguraciona svojstva zapravo vrše prilagođavanje topologije i ponašanja, i na ovaj način omogućavaju da se okruženje ponovo koristi.
- Test - Test predstavlja posebnu klasu koja služi za formiranje specifičnog verifikacionog scenarija. U ovoj klasi se instanciraju verifikacione komponente i vrši se njihova konfiguracija. Konfiguracija se podešava na način koji je osmišljen za test koji se planira izvršiti. U slučaju korišćenja TLM port-ova, potrebno je uraditi povezivanje ovih port-ova unutar test klase.

3. DUT (Design Under Test)

APB2SPI modul je veza između serijske i paralelne komunikacije, APB (Advanced Peripheral Bus) je paralelni deo, a SPI (Serial Peripheral Interface) je serijski. Modul se sastoji od konfiguracionih i statusnih registara, koji su mapirani u APB adresnom prostoru. Preko njih je moguće konfigurisati modul. Modul može biti konfigurisan da radi u master ili u slave modu, veličina podataka koja se šalje ili prima, smer komunikacije, redosled podataka takođje su konfigurabilni, i pored toga postoji i slave select manipulacija.

4. VERIFIKACIONI PLAN

Prvi korak u verifikaciji je izrada verifikacionog plana, koja služi kao putokaz u toku procesa verifikacije U verifikacionom planu su napisani SVA assertion-e za APB i SPI interfejs, potrebni checker-i za proveru dizajna i pokrivenost. U *Tabela 1* moguće je videti assertion-e za APB interfejs.

Tabela 1 : Assertion-e za APB interfejs

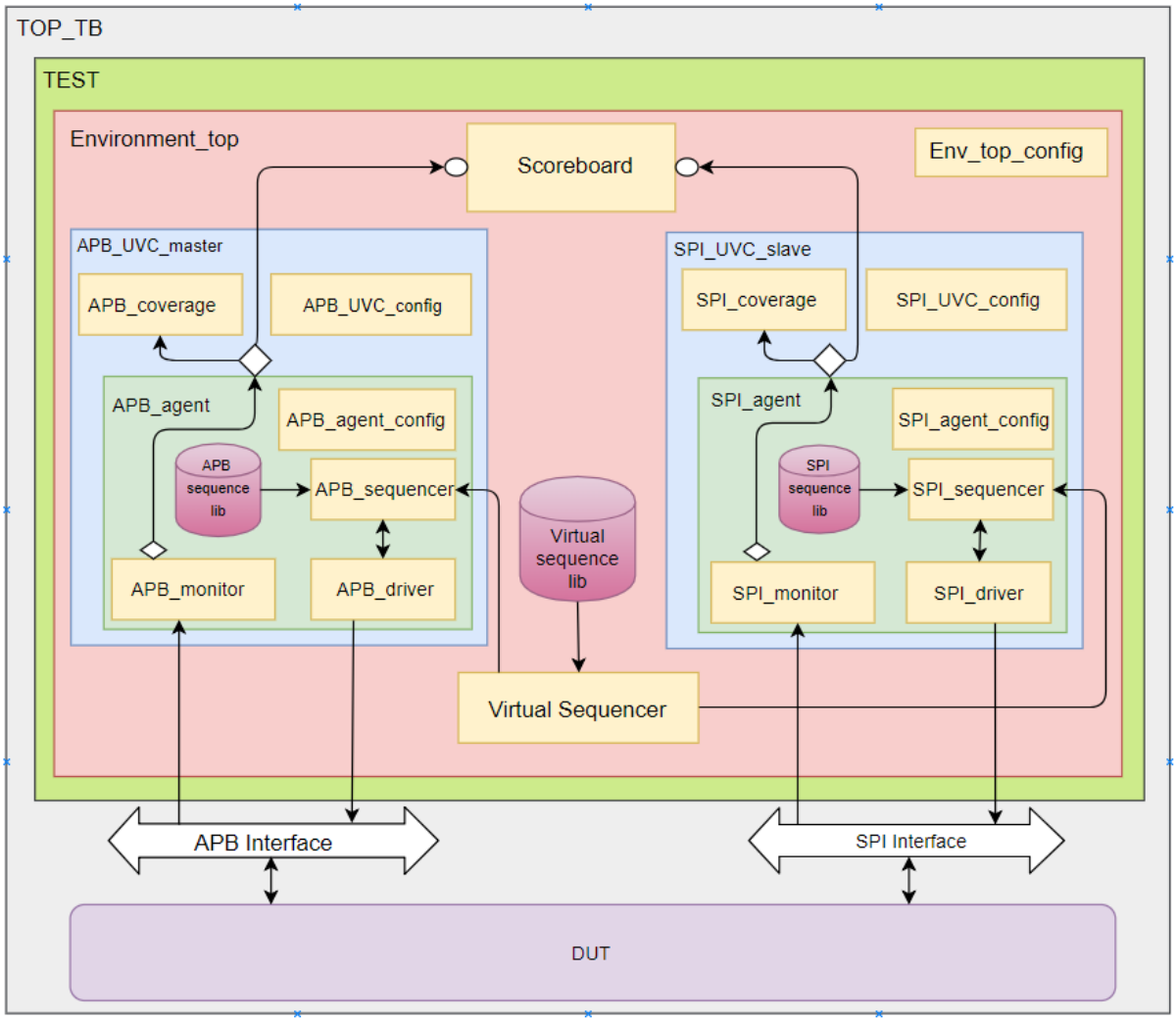
ID	Ime	Opis
1	penable_assert_chk	Enable signal mora biti postavljen posle jednog clock takta psel signala
2	penable_deassert_chk	Enable signal mora biti deaktiviran istovremeno sa signalom pready
3	penable_stability_chk	Ako se signal penable aktivira, onda on mora biti aktivan dok signal pready neće biti deaktiviran
4	pstrb_in_phase_read_chk	Tokom tranzakcije čitanja, signal pstrb mora biti deaktiviran
5	pwdata_stabilty_chk	Ako se signal psel aktivira i signal pwrite ima vrednost WRITE, onda signal pwdata mora biti stabilan dok signal pready neće se deaktivirati
6	paddr_stabilty_chk	Ako se signal psel aktivira, onda signal paddr mora biti stabilan dok se ne deaktiviše signal pready
7	pwrite_unknown_chk	Signal pwrite ne sme imati nepoznate vrednosti
8	pready_unknown_chk	Singal pready ne sme imati nepoznate vrednosti
9	psel_stability_chk	Ako se signal psel aktivira, onda on mora biti aktivan dok signal pready neće biti deaktiviran

U *Tabela 2* možemo videti pokrivenost covergroup-a APB2SPI_REGS.

Tabela 2 : Pokrivenost APB2SPI_REGS-a

ID	Ime	Opis
1	PWRITE_cov	Pokriti vrste tranzakcije
2	PADDR_cov	Pokriti adrese
3	WRITE_x_ADDR_cov	Pokriti na kojim adresama smo probali pisati
4	READ_x_ADDR_cov	Pokriti iz kojeg adresa smo probali čitati

U *Tabela 3* možemo videti checker-i za Status 0 registar.



Slika 2 : Blok dijagram TESTBENCH-a

Tabela 3: Lista checker-a

Br	Ime checker-a	Opis
1	SPIxSTAT0_rst_val_chk	Nakon reset-a Status 0 registar mora imati vrednost 0x00000030
2	SPI_STAT0_rd_only_chk	Status 0 registar je read only. Bilo koji upis ne sme promeniti njihov vrednost
3	transmit_buffer_TFE_flag_chk	TFE flag postavljen je ako predajni bafer prazan
4	transmit_buffer_TNF_flag_chk	TNF flag postavljen je ako predajni bafer nije pun
5	transmit_buffer_TXBEC_cnt_chk	Brojač elemenata predajnog bafera, mora biti manja ili jednaka sa FDEPTH+1
6	receive_buffer_RFF_flag_chk	RFF flag postavljen je ako prijemni bafer pun
7	receive_buffer_RNE_flag_chk	RNE flag postavljen je ako prijemni bafer nije prazan
8	receive_buffer_RXBEC_cnt_chk	Brojač elemenata prijemnog bafera, mora biti manja ili jednaka sa FDEPTH+1

5. VERIFIKACIONA ARHITEKTURA

Testbench se sastoji od DUT-a, od dva interfejsa i od test-a. Blok dijagram testbench-a je prikazan na *Slika 2*.
U okviru testbench-a korišćena je dva interfejsa (APB i SPI).
Test component se sastoji od Environment_top-a, koji se sadrži konfiguracioni objekat i sledeće komponente: UVC-eve, scoreboard i virtualni sequencer.

6. REZULTATI TESTIRANJA

Nakon razvoja verifikacionog okruženja, počela je provera funkcionalnosti, na osnovu verifikacionog plana.

Slika 3 : Rezultat testiranja

Tokom simulacije scoreboard daje povratne informacije o tome, da li funkcionalnost dizajna radi po specifikaciji. U slučaju traženja bug-a, često se koristi i alat SimVision, koji nam daje grafički prikaz trenutne simulacije, koji u nekim slučajevima olakšava identifikaciju greške. Ovo je moguće videti na *Slika 3*.

7. POKRIVENOST

Jedan od težih zadataka u procesu verifikacije je odlučiti kada je verifikacija završena. Da bi smo došli do tog zaključka mora se dati odgovor na dva pitanja: da li su sve osobine dizajna, koje su identifikovane u verifikacionom planu, verifikovane? I, da li postoje delovi koda u dizajnu koji se nikad nisu koristili? Da bi smo dali odgovore na ova pitanja uvodi se nova metrika, pokrivenost (eng. Coverage). Više infomacija se može naći na web stranici [3]. Dve najčešće korišćene coverage metrike su:

- Strukturna pokrivenost (code coverage) - implicitna
- Funkcionalna pokrivenost (functional coverage) – eksplicitna

obe metrike, uz detaljno razrađen plan o prikupljanju pokrivenosti.

Code coverage

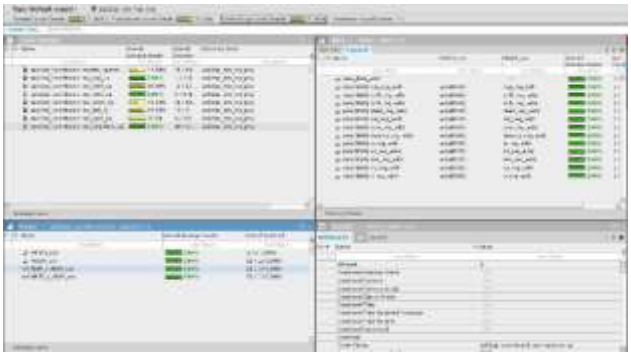
Strukturna pokrivenost ili pokrivenost koda daje informacije o stepenu aktivacije source koda tokom verifikacije čime se omogućava praćenje struktura koje se nikad ne aktiviraju. Glavna prednost ove metrike je što je implicitna odnosno kreiranje modela je automatsko. Za korišćenje pokrivenosti koda nije potrebno dodavati poseban kod i ne zahteva poseban pristup tokom verifikacije. Mana ovog tipa pokrivenosti je što je moguće imati 100% pokrivenosti, a da i dalje postoje greške u dizajnu.Postoji više tipova strukturne pokrivenosti:

- Toggle coverage
- Line coverage
- Statement coverage
- Final state machine coverage
- Branch coverage
- Expression coverage

Functional coverage

Cilj funkcionalne verifikacije je utvrditi da li dizajn implementira sve osobine i funkcioniše na način opisan u funkcionalnoj specifikaciji. Međutim do zaključka o tome da li je neka funkcionalnost stvarno implementirana i da li je verifikovana, ne može se doći na osnovu praćenja pokrivenosti koda. Zbog toga se uvodi nova, eksplicitna metrika - funkcionalna pokrivenost. Cilj ove metrike je merenje progresa verifikacije u odnosu na funkcionalne zahteve dizajna. Jedan od problema korišćenja constrained-random pristupa generisanja stimulus-a je što ne znamo tačno koje funkcionalnosti se verifikuju (šta je tačno dovedeno na ulaz DUT-a) bez da ručno analiziramo waveform-e tokom simulacije. Međutim, praćenje funkcionalne pokrivenosti nam omogućava upravo ovo – određivanje funkcionalnosti koje su verifikovane bez vizuelne analize samih signala. Mane ove metrike su što, pošto nije implicitna, ne može biti automatski implementirana. Implementacija dobrog modela pokrivenosti zahteva dosta vremena pošto je potrebno prvenstveno napraviti dobar plan, identifikovati sve osobine od interesa i odrediti način na koji će se prikupljati podaci o pokrivenosti.

Na *Slika 4* možemo videti analizu covergroup-a spi_registers_cg.



Slika 4 : Pokrivenost SPI_REGISTER_CG-a

7. ZAKLJUČAK

Glavni zadatak ovog rada je bio proučavanje modernih tehnika za verifikaciju, kako bi se napravio verifikacioni plan za modul koji je APB2SPI, i da se implementira i simulira planirani verifikaciono okruženje, koristeći UVM metodologiju i CADENCE alat za simulaciju. Verifikacioni plan i arhitektura su napravljeni uz pomoć projekt zadatka i dizajn specifikacije. Verifikaciono okruženje bazirana na modernim verifikacionim tehnikama, kao što su nasumično generisanje ulaznih promenljivih sa nekim ograničenjima, i funkcionalna pokrivenost. Tokom simulacije neke informacije su sakupljene, kako bi bilo moguće izvršiti analizu verifikovanog modula.

8. LITERATURA

[1] Sharon Rosenberg, Kathleen A Meade; A Practical Guide to Adopting the Universal Verification Methodology (UVM); Cadence Design Systems, Inc; San Jose, USA; 2010;

[2] Chris Spear, Greg Tumbush; SystemVerilog for Verification: A Guide to Learning the Testbench Language Features; Third Edition; Springer; New York, USA; 2012;

[3] http://www.elektronika.ftn.uns.ac.rs; Sajt katedra za elektroniku; Novi Sad; Predmet: Funkcionalna verifikacija hardvera; 06.09.2018;

Kratka biografija:

Čongor Lašu rođen je u Subotici 1993 god. Master rad na Fakultetu Tehničkih Nauka iz oblasti Elektrotehnike i računarstvo - Embedded sistemi i algoritmi odbranio je 2018 god.