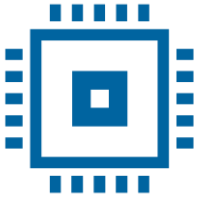

Universitatea
Transilvania
din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ
ȘI ȘTIINȚA CALCULATOARELOR

PROIECT DE DIPLOMĂ

Conducător științific:
Prof. Dr. Ing. Dan Nicula

Absolvent:
Laszlo Zsolt

BRAȘOV, 2025

Departamentul Electronică și Calculatoare
Programul de studii: Calculatoare

LASZLO Zsolt

Mediu de verificare a unui controller I2C configurabil prin APB utilizând UVM

Conducător științific:
Prof. Dr. Ing. Dan Nicula

Cuprins

Cuprins

LISTA DE FIGURI, TABELE ȘI CODURI SURSĂ

FIGURI

TABELE

CODURI SURSĂ

LISTA DE ACRONIME

ACK - Acknowlage
AMBA – Advanced Microcontroller Bus Arhitecture
APB – Advanced Peripheral Bus;
BCL – Base Class Libraryș
CFG - Configurationș
DUT – Design/Device Under Test;
FIFO – First In First Out
FSM – Finite State Machine
I2C – Inter-Integrated Circuit;
IRQ – Interrupt Request
NACK – Not Acknowlage
R - Read
RTL – Register Transfer Level
RW – Read/Write
RX – Receive
SCL – Serial clock
SDA – Serial data
SVA – SystemVerilog Assertion
TLM – Transaction Level Modeling
TX – Transmit
UVM – Universal Verification Methodology;
W - Write

1 INTRODUCERE

1.1 Tema proiectului

Tema proiectului este realizarea unui mediu cât mai parametrizabil de verificare pentru un sistem de transmisie-recepție I2C având regiștrii de configurare accesibili prin protocolul APB. Acest sistem transmite date dintr-o memorie FIFO prin protocolul I2C și în direcție inversă, suportând configurații de tip (Master / Slave), frecvența de transmisie, adresa țintă sau de dispozitiv. Dispozitivul generează o întrerupere mascabilă în momentul finalizării transferului prin I2C, semnalând eventualele erori întâmpinate pe parcursul transferului.
Procesul de verificare este bazat pe stimuli generați si transmiși către DUT, monitorizarea, colectarea și verificarea tranzacțiilor specifice protocolului respectiv de comunicare, ulterior trimiși către monitorul global numit si scoreboard pentru a verifica integritatea datelor si caracteristice specifice sistemului verificat.

1.2 SCOPUL

Verificarea în circuitele integrate este asigurarea că sistemul proiectat se comportă corespunzător cerințelor și specificațiilor date [1], proces ce consumă timp semnificativ în dezvoltarea circuitelor integrate, prin urmare se dorește a avea componente cat mai versatile, reutilizabile si parametrizabile întrucât să se reducă timpul de dezvoltare a mediilor de verificare.

Componentele de verificare aparținătoare protocoalelor specifice de comunicații APB respectiv I2C sunt realizate astfel încât să fie universale și complete, nefiind dependente de DUT, putând fi folosite ulterior in proiecte viitoare. Fiecare componenta specifică protocolului său este responsabilă de verificarea integrității semnalelor de control, semnalarea erorilor și extragerea acoperirii funcționale specifice protocolului respectiv, astfel reducând complexitatea monitorului global și focusarea acestuia pe atributele specifice sistemului verificat.

1.3 OBIECTIVE

Dezvoltarea unui mediu de verificare pentru un circuit integrat se poate despărții în mai multe etape, printre primele fiind analiza specificațiilor dispozitivului și crearea unui plan de implementare. În acest plan se definesc caracteristicile dispozitivului ce se doresc a fi testate,

acoperirea funcțională pentru asigurarea că DUT-ul a fost testat în fiecare stare/ipostază dorită și scenariile de verificare prin care dorim să ducem dispozitivul la limitele performanței acestuia.

Prin dezvoltarea metricilor respectivi se creează o imagine de ansamblu a mediului de verificare și specificațiilor acestuia înainte de a-l construi. Acest lucru este benefic inginerilor de verificare întrucât cunoscând caracteristicile necesare pentru verificarea eficientă a dispozitivului, implementarea mediului devine semnificativ mai ușoară.

1.3.1 Verificarea funcționalității

Scopul principal în dezvoltarea unui mediu de verificare este ca aceasta să asigure funcționalitatea dispozitivului în ipostaze gândite și negândite în momentul proiectării acestuia. Această verificare se dorește a fi cât mai automatizată, necesitând cât mai puțină interacțiune cu utilizatorul.

Pentru a putea realiza acest lucru, este necesară o înțelegere deplină a specificațiilor sistemului cât și a protocoalelor de comunicații utilizate de acesta. Procesul de dezvoltare a metricilor, deși lent, este un proces esențial pentru a asigura integritatea mediului de verificare ce urmează a fi dezvoltat.

1.3.2 Acoperire funcțională

Acoperirea funcțională este o metrică definită de către inginerul de verificare ce măsoară procentajul obiectivelor de verificare atinse pe parcursul simulării. Această metrică permite vizualizarea procentului de caracteristici verificate pe parcursul rulării testelor cât și caracteristicile însăși, astfel ajutând utilizatorul prin vizualizarea caracteristicilor ce mai trebuise a fi verificate.

În acest mediu, acoperirea funcțională se împarte în 2 categorii, cea de protocol și cea specifică dispozitivului verificat. Prin această implementare se crește nivelul de versatilitate a agenților de magistrală permițând reutilizarea acestora și în alte medii de verificare ce utilizează protocolul respectiv de comunicare.

1.3.3 Acoperire de cod

Acoperirea de cod se definește a fi procentul codului utilizat pe parcursul simulării de către modulele RTL. Această metrică este creată de către simulator, verificând starea semnalelor pe parcursul simulării și eșantionând liniile de cod utilizate, expresiile atinse și stările automatelor FSM în cazul existenței lor.

Scopul acestei statistici este de a verifica eficiența și performanța dispozitivului la nivel de cod verificând existența fragmentelor de cod inaccesibile sau redundante. Se dorește de a obține un procentaj de 100% în ce privește codul utilizat sau o valoare ce tinde spre aceasta după rularea unei regresii.

1.4 STADIUL ACTUAL AL PROIECTULUI

1.4.1 Evoluția metodologiei standard de verificare în industrie

Înaintea implementării bibliotecii UVM în limbajul de verificare SystemVerilog, au existat mai multe încercări de a aduce la un nivel standardizat metodologia de verificare în industria de semiconductoare prin intermediul bibliotecilor cu clase de bază (BCL), însă aceste biblioteci erau dependente de simulatorul utilizat, implicit de comercianții acestora [11].

Pentru rezolvarea acestei probleme organizația Accellera a format un subcomitet pentru implementarea unei metodologii universale de verificare care să nu fie dependent de simulator sau de comercianții acestuia, astfel în anul a 2010 au creat biblioteca UVM. În urma analizelor s-a estimat că această metodologie va ajunge standardul în industrie pentru o vreme [11].

Relația dintre limbajul SystemVerilog și biblioteca UVM a devenit din ce în ce mai apropiată, întrucât termenii de ”verificare prin SystemVerilog” și ”verificare prin UVM” ajung să fie interschimbabile în contextul industriei. Aplicate împreună, SystemVerilog și UVM oferă soluționarea eficientă a problemelor ce implică verificarea hardware la nivel de design.

1.4.2 Migrarea de la simulare la emulare

Pentru a sigura reutilizabilitatea mediilor de simulare și emulare, mediul trebuie să respecte anumite principii fundamentale precum partiția acestuia în secții distincte din punct de vedere al consumului timpului de simulare.

Componentele precum driverul și modele funcționale de magistrală (Bus Functioning Model), plasate în categoria consumătoare de timp, comunicarea cu dispozitivul verificat trebuie realizată exclusiv prin acestea. Componentele responsabile de verificarea propriu zisă sunt plasate în categoria neconsumătoare de timp, acesta funcționând la un nivel mai înalt de abstractizare, încapsulând datele referitoare la comportamentul dispozitivului, permițând verificarea mai eficientă a acestora [12].

1.4.3 Implementări similare

În articolul [13] este prezentat o metodologie de verificare a protocolului I2C, axându-se în principal pe atributul de clock stretching. Autorii au implementat un mediu de verificare utilizând limbajul SustemVerilog și biblioteca UVM, prin care se verifică un dispozitiv I2C slave, mediul lor emulând dispozitivul master. Suportarea fenomenului de clock stretching însă este singura caracteristică specială a agentului, articolul nemenționând atribute multimaster precum verificarea eliberării magistralei sau suportarea pierderii unei arbitrări.

Implementarea componentei de verificare responsabile protocolului I2C prezentat în publicația [14] este structurată sub forma a 2 agenți distincți în funcție de tip (master/slave). Metodologia respectivă limitează versatilitatea și refolosirea componentelor specifice, ele fiind dezvoltate independent și având într-o anumită măsură dependență de proiectul pentru care au

fost create. Pe lângă acest fapt, componentele respective suportă configurări limitate de către utilizator, nefiind ușor implementate în proiecte viitoare.

Metodologia implementată în lucrarea [15] în verificarea protocolului APB este structurată având principala unealtă de verificare aserțiile limbajului SystemVerilog. Pe baza acestora se pot face simulări fără a fi necesara rularea interfeței grafice ale simulatorului. Rezultatele autorilor au constat într-o acoperire funcțională de 100% structurate pe 1030 de bin-uri și de o acoperire de cod de 97%.

1.4.4 Încadrarea implementării alese în domeniu

Metodologia de implementare a mediului de verificare are ca principalele puncte forte structura modularizată, configurabilitatea accentuată și permiterea reutilizării a componentelor ce alcătuiesc mediul, obținând o versatilitate ridicată.

Caracteristicile acesteia permit utilizarea componentelor și în alte medii de verificare, permițând dezvoltarea rapidă și eficientă acestora, nefiind necesară implementarea verificării de protocol și axarea inginerului de verificare pe caracteristicile dispozitivului testat.

2 METODOLOGIA DE VERIFICARE ȘI PROTOCOALELE DE COMUNICAȚII

UVM
Protocolul APB
Protocolul I2C

2.1 UVM – STRUCTURĂ ȘI SPECIFICAȚII

Odată cu dezvoltarea rapidă a industriei semiconductoarelor și a tehnologiei de proiectare a circuitelor integrate, complexitatea circuitelor crește făcând verificarea acestora din ce în ce mai dificilă. Conform statisticilor aproximativ 70% din timpul consumat pentru dezvoltarea unui circuit integrat este consumat de verificare, fapt ce face ca metodele tradiționale de verificare să nu fie suficiente cerințelor actuale [2].

UVM este o bibliotecă ce extinde limbajul SystemVerilog aducând clase, metode și structuri de date predefinite într-un mod ierarhic bazat pe conceptul programării orientate pe obiecte (OOP) simplificând dezvoltarea mediului de verificare având un șablon pentru structura acestuia. Arhitectura ierarhică este împărțită în 4 nivele: nivelul de test, nivelul secvență, nivelul de driver și monitorizare, și nivelul de DUT [3].

2.1.1 Componentele principale UVM

1. **Agent:** Agentul UVM se comportă ca o interfață între DUT și mediul de verificare. Are ca și rol executarea și monitorizarea transferurilor de date prin interfața și protocolul specific asignat lui și trimiterea acestora către scoreboard.
2. **Scoreboard:** Este un monitor global care colectează tranzacțiile de la fiecare agent și verifică integritatea și comportamentul sistemului verificat asigurându-se că acesta respectă cerințele proiectantului. De asemenea eșantionează tipurile de tranzacții și modul în care DUT-ul a fost configurat pentru a crea un raport de acoperire funcțională care va fi salvat la finalul unei simulări.
3. **Environment:** Este mediul unde sunt create, configurate și conectate componentele de verificare (agenții și monitorul global) asigurând liniile de comunicare între acestea utilizând porturi TLM.
4. **Sequencer:** Arbitrează, randomizează și lansează secvențe către driverul agentului corespunzător ca acesta să execute tranzacțiile indicate. Comunicarea cu driverul este una bidirecțională, sequencer-ul așteaptă ca driverul să specifice când este eligibil pentru a executa o tranzacție nouă.

5. **Test:** Aflat în vârful ierarhiei mediului de verificare, este responsabil de configurarea specifică a întregului mediu pentru a satisface scenariile ce necesită a fi verificate prin intermediul secvențelor.

2.1.2 Fazele unei simulări în UVM

O caracteristică a simulării folosind UVM este că simularea este despărțită in mai multe faze ce se parcurg secvențial. Fazele principale ale simulării în ordine de execuție sunt următoarele:

1. **Build phase:** se execută la începutul simulării, creând și configurând obiectele și componentele necesare executării testului respectiv. Se execută in stil top-down, din componenta test se creează componenta environment, din environment se creează agenții și așa mai departe din vârful ierarhiei în jos.
2. **Connect phase:** în această fază se realizează conexiunile dintre componente, se creează canalele TLM și se setează pointerii resurselor comune ale componentelor. Această fază se execută in ordine inversă față de build phase, adică de jos în sus ierarhic.
3. **Run phase:** se generează și se transmit stimulii către DUT, se monitorizează integritatea transferurilor, se semnalează eventualele erori întâmpinate pe parcurs și se eșantionează acoperirea funcțională si cea de cod. În această fază are loc verificarea propriu zisă, fiind singura fază care consumă timp de simulare.
4. **Report phase:** se execută la sfârșitul simulării și are ca scop raportarea numărului de erori întâmpinate pe parcurs, verificarea dacă DUT-ul a picat testul rulat și în caz contrar salvarea acoperirii funcționale și de cod într-o bază de date.

2.1.3 Modelul de regiștrii UVM

Biblioteca UVM are la îndemână un model de regiștrii care permite inginerului de verificare să creeze o clonă high-level a regiștriilor din DUT, ușurând metodologia de verificare și acoperirea funcțională. Structura de date este configurabilă ca si lățime, acces și valoare inițială și permite structurarea acesteia în mai multe câmpuri accesibile independent.

Acest model vine cu un adaptor de magistrală care translatează tranzacția executată pe interfața cu DUT-ul și actualizează valoarea clonei high-level pe parcursul simulării, astfel încât valoarea scrisă in regiștrii DUT-ului să fie identici cu valoare din model. În cazul în care accesul regiștriilor din DUT este limitat doar la citire, modelul permite prezicerea valorii lor după caz. În momentul citirii unui registru modelul compară automat valoarea citită din design cu valoarea din model sau valoarea prezisă, semnalând ca și eroare diferența valorilor [4].

Modelul de regiștrii se poate lega și la sequencer-ul unui agent pentru a lansa secvențele de scriere și citire utilizând metodele predefinite ale modelului.

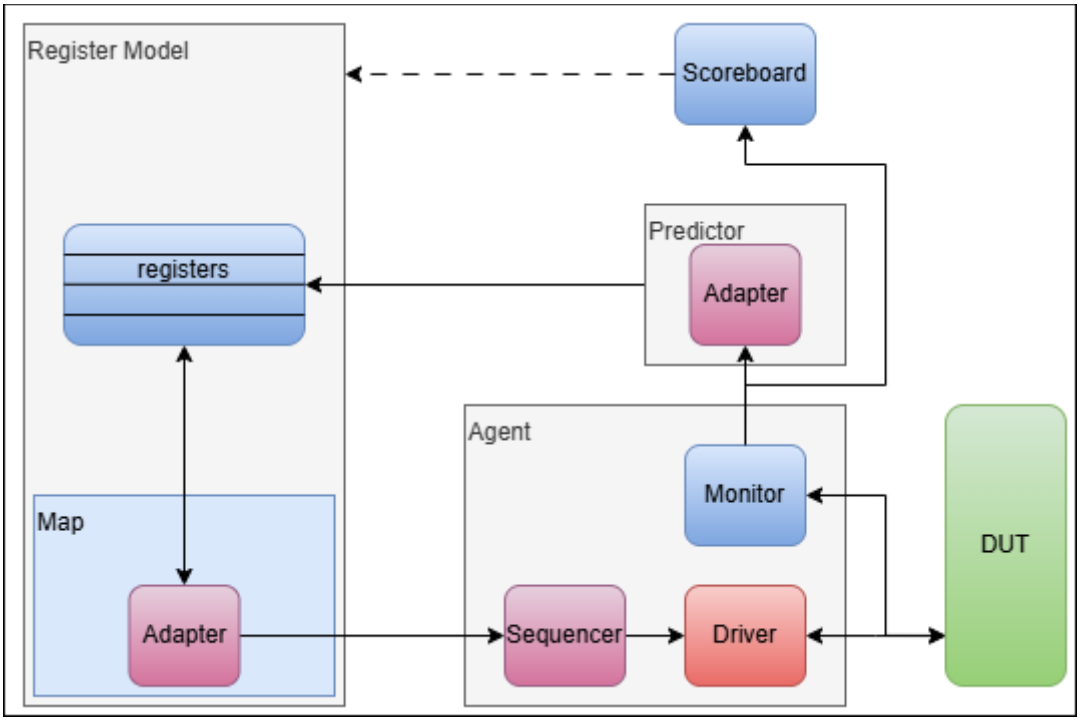

Figura 1 Schema bloc a integrării modelului de regiștrii

2.2 PROTOCOLUL APB

APB este un protocol de comunicații de cost și complexitate redusă, dezvoltat de către ARM, făcând parte din grupul AMBA. Protocolul este unul sincron half-duplex, nesuportând pipelining-ul unde fiecare tranzacție durează cel puțin 2 perioade de ceas. Interfața a fost creată cu scopul accesări regiștrilor de control a dispozitivelor periferice [5].

2.2.1 Tipuri de tranzacții APB

Figura 2 ilustrează tranzacțiile de scriere APB. Tranzacția poate fi despărțită în două faze: faza de setup și faza de acces. În faza de setup se pun datele și adresa pe magistralele corespunzătoare, se pune semnalul PWRITE în 1 logic alături de semnalul PSEL, indicând începerea comunicației cu dispozitivul selectat. Faza de acces constă în punerea semnalului PENABLE în 1 logic și așteptând răspunsul de la periferic, adică semnalul PREADY, însoțit de semnalul PSLVERR în cazul în care s-a detectat o eroare în tranzacție sau lipsit de acesta in cazul efectuării cu succes a transferului. Semnalele de control, adresa și datele de pe magistrale trebuie să fie stabile pe tot parcursul transferului.

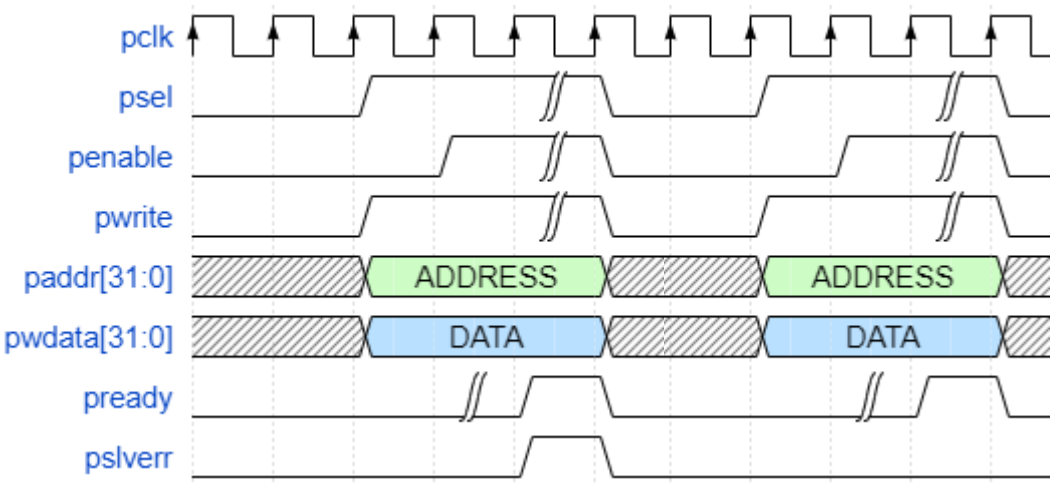

Figura 2 Formele de undă a tranzacției de scriere APB

În figura 3 se poate observa operația de citire APB. Faza de setup este similară cu cea de la scriere, mai puțin faptul că nu se mai pun date pe magistrala de date de scriere și semnalul PWRITE care de data aceasta este în 0 logic indicând operația de citire dispozitivului. În faza de acces se așteaptă răspunsul perifericului indicând validitatea datelor de pe magistrala de citire sau semnalarea erorii de transfer.

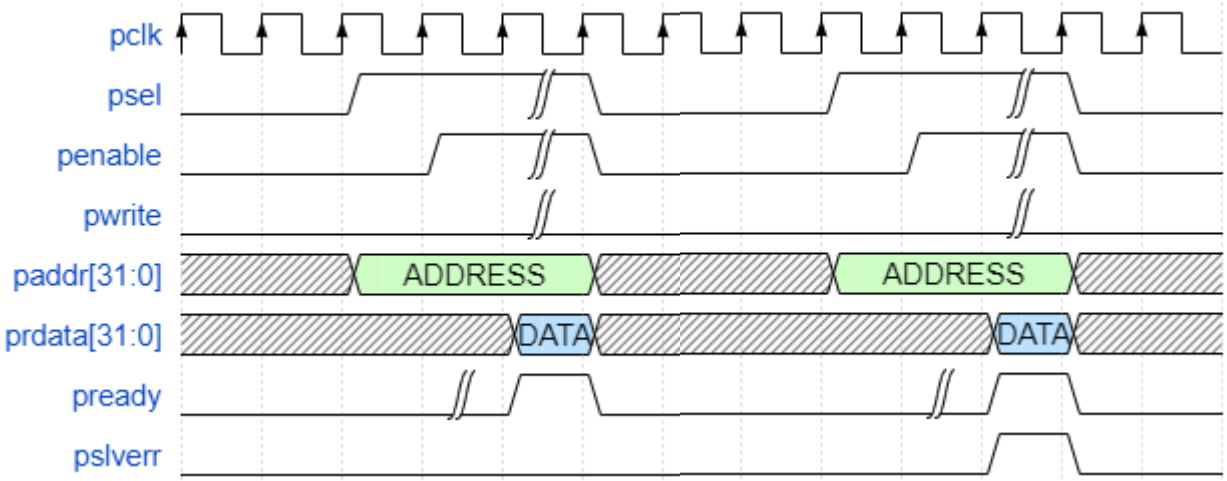

Figura 3 Formele de undă a tranzacției de citire APB

2.3 PROTOCOLUL I2C

I2C este un protocol de comunicații serial sincron pe 2 fire, folosit în dispozitive care operează la o viteză relativ redusă. Comunicarea între dispozitive este inițiată de către master pornind oscilația semnalului de ceas și transmițând adresa slave-ului cu care vrea să inițieze un canal de comunicare pe magistrală urmat de bitul de operație, așteptând semnalarea recepționării acestuia, după care se transmit datele în direcția indicată de către bitul de operație [6].

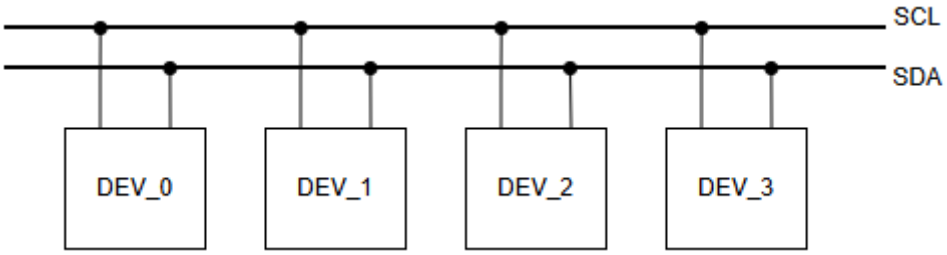

Figura 4 Dispozitive legate la magistrala I2C

Adresarea se realizează pe 7 sau 10 biți depinzând de dispozitiv, iar datele se transmit in pachete de câte 8 biți, necesitând câte un semnal de recepționare la fiecare byte transferat. În cazul operației de scriere dispozitivul slave este responsabil pentru semnalarea recepționării, iar in cazul de citire semnalarea recepționării de către master indică și faptul că acesta dorește să continue transferul cu încă un octet.

Starea inițială a semnalelor SCL și SDA este în 1 logic, condiția de începere a unui transfer este frontul negativ al semnalului SDA cât timp SCL se află în 1 logic iar condiția de încheiere este frontul pozitiv al semnalului SDA de asemenea cât timp SCL este în 1 logic. Pe tot parcursul transferului semnalul SDA trebuie comutat doar pe palierul negativ al semnalului de ceas, eșantionarea acestuia având loc pe frontul pozitiv al acestuia. Pe lângă asta semnalul SDA trebuie să fie menținut stabil pe tot parcursul palierului pozitiv al ceasului, excepție fiind condițiile de start si de stop [7].

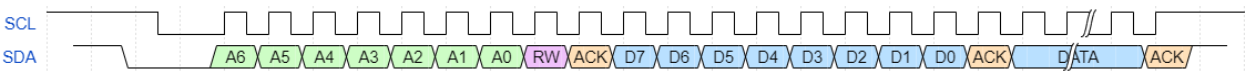

Figura 5 Formele de undă a unui transfer prin I2C

2.3.1 Condiția repeated start

De multe ori în operațiile ce utilizează I2C este necesară scrierea unei comenzi apoi citirea deîndată a răspunsului, dar cum o asemenea operație necesită 2 cadre de transfer există posibilitatea ca un alt master să ocupe magistrala între cele 2 cadre, forțând primul dispozitiv să aștepte eliberarea acesteia, riscând ca datele să nu mai aibă relevanță, neștiind durata timpului până la eliberarea magistralei.

Soluția la problema respectivă este condiția de repeated-start, adică la finalul operației masterul în loc să semnaleze o condiție de stop, acesta indică încă o condiție de start, urmată de adresa si bitul de operație, având control total asupra magistralei și elimină șansa ocupației acesteia de către celelalte dispozitive. Oricâte condiții de start consecutive s-ar transmite pe magistrală, la finalul tranzacției se transmite doar o singură condiție de stop [8].

Această condiție de repeated-start trebuie suportată de ambele dispozitive care comunică prin magistrala I2C.

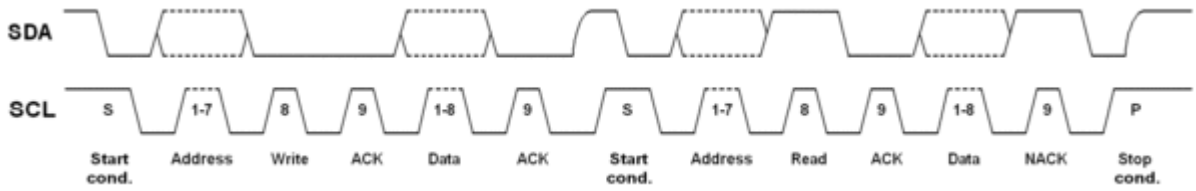

Figura 6 Exemplu de condiție de repeated-start [8]

2.3.2 Clock stretching

Cum în comunicarea prin I2C masterul este cel care asigură ceasul pe care este sincronizat transferul există posibilitatea ca dispozitivul slave să nu poată ține pasul cu rata de transfer cerută de către master, acesta fiind prea mare [8].

Soluționarea problemei constă în preluarea provizorie a controlului semnalului de ceas SCL și menținerea acesteia în 0 logic de către dispozitivul slave până când acesta este eligibil pentru a transmite sau de recepționa date. Dispozitivul master are ca rol detectarea acestui fenomen și de a aștepta să i se redobândească controlul acestuia asupra semnalului de ceas.

Acest fenomen este independent de direcția transferului, eșantionarea semnalului SDA se va efectua în momentul eliberării ceasului de către slave.

2.3.3 Arbitrare între 2 masteri

Unele dispozitive I2C master suportă conectarea la magistrală împreună cu alte dispozitive master, fără ca aceștia să interfereze transferurile celorlalte dispozitive, aceștia au denumirea de multimaster.

Multimasterii au atributul de a putea verifica dacă magistrala este ocupată de alt dispozitiv și pentru a nu perturba transferul acestuia așteaptă eliberarea acesteia, prin detectarea unei condiții de stop de pe aceasta, înainte de a începe transferul propriu.

În cazul in care 2 dispozitive multimaster încep câte o tranzacție în același timp va avea loc o arbitrare astfel: după comutarea semnalului SDA în 1 logic fiecare multimaster citește semnalul pentru a se asigura că într-adevăr semnalul a fost comutat în 1. În caz contrar acesta constată că un alt dispozitiv ține semnalul legat la masă, așadar cedează controlul pe magistrală dispozitivului respectiv iar în funcție de dispozitivul care a pierdut arbitrarea, aceasta poate relua transferul după eliberarea magistralei sau să abandoneze complet tranzacția respectivă.

3 SPECIFICAȚIILE SISTEMULUI VERIFICAT

Structura și accesul regiștrilor
Atribute specifice I2C ale sistemului
Fluxul transferului de date

3.1 STRUCTURA REGIȘTRILOR

Comunicarea unității centrale cu controllerul I2C se realizează prin intermediul bancului de regiștrii al acestuia, accesibil prin interfața APB. DUT-ul are la dispoziție 9 regișrtii cu lățime de 32 de biți împărțite pe câmpuri, permițând stocarea informațiilor din mai multe surse în același registru. Câmpurile nu sunt accesibile individual, scrierea și citirea trebuie efectuată pe întregul registru decodificând ulterior informația.

Acești regiștrii sunt împărțiți 3 categorii funcționale:

- **Regiștrii de date:** regiștrii prin care unitatea centrală furnizează datele de transmisie sau accesează datele recepționate de către controller.
- **Regiștrii de configurare:** prin acestea se configurează controllerul, indicând caracteristicile transferului I2C. Configurațiile determină modul controllerului (master/slave), adresa țintă și de dispozitiv, frecvența ceasului etc.
- **Regiștrii de status:** accesibili doar prin operația de citire, acestea oferă informații despre fluxul transferului I2C în timp real, permițând monitorizarea acestuia de către unitatea centrală.

Tabelul 1.Regiștrii dispozitivului

Adresa	Nume	Valoare inițială	Acces	Descriere
0x00	TX_FIFO_DATA	0x00000000	W	Prin intermediul acestui registru se accesează memoria FIFO de transmitere, umplerea acesteia se realizează prin scrieri repetate în acest registru
0x04	RX_FIFO_DATA	0x00000000	R	Prin intermediul acestui registru se accesează memoria FIFO de recepție, citirile repetate incrementează pointerul de adrese al memoriei
0x08	ADDR	0x00000000	RW	Împărțit în 2 câmpuri, acest registru conține adresa țintă de transmisie/recepție în cazul în care controllerul este de tip master și adresa la care răspunde dacă este configurat în mod slave
0x0C	CTRL	0x00000000	RW	Registrul de control, prin acesta se indică dispozitivului starea (activ sau pasiv) și numărul de octeți ce se dorește a fi transmise/recepționate pe magistrala I2C în cazul configurării de tip master

0x10	CMD	0x00000000	W	Registrul comandă, acesta este folosit pentru a șterge întreruperea, resetarea pointerilor memoriilor FIFO sau inițierea tranzacției pe magistrala I2C în modul master
0x14	STATUS	0x00000000	R	Acest registru ne indică starea dispozitivului pe parcursul transferului I2C, precum numărul de octeții transmiși/recepționați, dacă s-a pierdut arbitrarea pe magistrală și dacă tranzacția încă este în desfășurare
0x18	IRQ	0x00000000	R	Conține vectorul de întreruperi, acesta necesită a fi citit după apariția întreruperii pentru a lua la cunoștință cauza acestei
0x0C	IRQ_MASK	0x00000000	RW	Conține masca vectorului de întreruperi, fiecare câmp se poate masca independent, 0 logic pe câmpul respectiv reprezintă mascarea întreruperii
0x20	DIVIDER	0x0000FFFF	RW	Prin acest registru se setează frecvența ceasului de transmisie I2C printr-o formulă internă

Accesarea adreselor nemapate va induce ridicarea semnalului PSLVERR de pe interfața APB, iar datele de pe magistrale nu vor fi înregistrate de către sistem. Orice acces de scriere în regiștrii este blocată de către sistem pe tot parcursul unei tranzacții pe magistrala I2C, astfel evitând întâmpinările erorilor software, cum ar fi schimbarea configurațiilor în mijlocul tranzacției.

3.2 ATRIBUTE SPECIFICE I2C ALE SISTEMULUI

3.2.1 Modul master

Modul master al controllerului are anumite atribute ale unui dispozitiv I2C multimaster, dar într-o manieră mai limitată. Acesta verifică dacă magistrala este ocupată de către un alt dispozitiv, și semnalează acest lucru în registrul status, dar așteptarea eliberării acesteia este soluționată prin software, adică nepornind un transfer până la semnalarea eliberării.

Dispozitivul suportă arbitrare, semnalând în registrul de status dacă acesta a fost pierdută. În momentul în care arbitrarea a fost pierdută, transferul este abandonat, necesitând intervenție software pentru a-l relua, însă există șansa de a pierdere date din memoriile FIFO.

Masterul suportă și fenomenul de clock stretching, acesta așteptând eliberarea semnalului SCL de către slave, continuând transferul. Apariția acestui fenomen pe parcursul transferului nu este raportat unității centrale.

3.2.2 Modul slave

Dispozitivul în modul slave suportă condiția de repeated-start, însă acesta este recepționat ca fiind 2 transferuri separate, ca și cum ar fi înregistrat o condiție de stop înainte de cea de start. O întrerupere va fi generată la finalul primului transfer, necesitând mascarea acesteia sau a citi periodic registrul de status pentru a determina sfârșitul transferului.

3.3 FLUXUL UNUI TRANSFER

Inițierea unui transfer de date pe magistrala I2C de către controller se face în primul rând prin configurarea acestuia în modul și configurațiile de funcționare dorite, prin scrierea acestora în regiștrii corespunzători.

În cazul în care dispozitivul a fost configurat ca să transmită date, pe magistrala I2C vor ajunge datele stocate în memoria FIFO de transmitere încărcată de către unitatea centrală prin scrierea acestora la adresa registrului TX, iar în cazul în care controllerul recepționează date, acesta le va stoca într-o memorie FIFO similară cu cea te transmisie accesibil prin citirea repetată a registrului RX.

Finalul tranzacției pe magistrala I2C, golirea memoriei de recepție sau umplerea memoriei de transmisie generează o întrerupere care poate fi mascată. Generarea întreruperii indică schimbarea unui câmp din registrul care conține vectorul de întreruperi. Citind acest registru se poate determina cauza apariției întreruperii generate de către dispozitiv. Cauzele ce țin de transferul prin I2C sunt: cererea de transfer nu a fost acceptată de către slave, semnalarea nerecepționării datelor de către slave, încercarea de a citi din memoria goală, încercarea de a scrie într-o memorie plină sau decurgerea cu succes a transferului.

4 STRUCTURA MEDIULUI DE VERIFICARE

Caracteristica principală a mediului de verificare este modularitatea, fiecare componentă având rolul și funcționalitățile proprii, simplificând implementarea. Structura mediului are la bază șablonul ierarhic oferit de biblioteca UVM, cu mici alterații specifice, pentru a păstra integritatea modularității.

Componentele active ale mediului pot fi configurate prin intermediul bazei de date de configurații uvm_config_db din biblioteca UVM. Acesta este încărcată la începutul simulării, de sus în jos din punct de vedere ierarhic pentru a asigura distribuirea configurațiilor componentelor înainte ca acestea să fie create. Fiecare componentă configurabilă prin această bază de date o accesează în build_phase, citind parametrii necesari configurării și oprind simularea în cazul întâmpinării lipsei configurațiilor sau incompatibilitatea acestora.

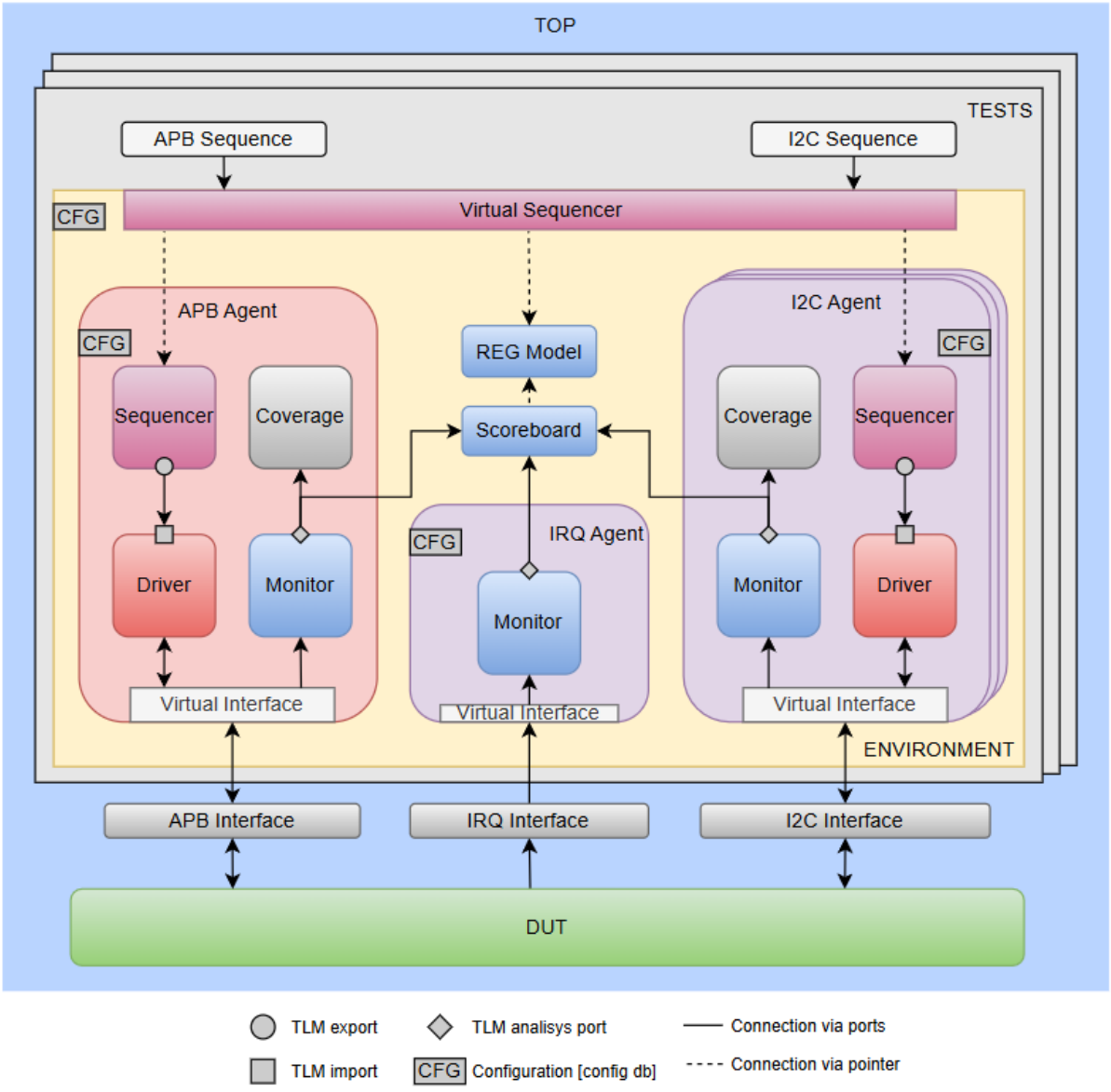

Figura 7 Schema bloc a mediului de verificare

În figura 7 se poate observa schema bloc a mediului de verificare, conexiunile și domeniul lor de aplicare. Componenta responsabilă pentru semnalelor de ceas și reset asincron nu sunt prezente în diagramă, prezența ei fiind subînțeleasă prin sincronitatea protocoalelor de comunicații.

În următoarele subcapitole se vor prezenta specificațiile, scopul și aplicabilitatea componentelor ce alcătuiesc mediul.

4.1 INTERFEȚE

Noțiunea de interfață în SystemVerilog se referă la încapsularea datelor și a anumitor funcționalități ce aparțin unui anumit protocol. Interfețele din mediul de verificare sunt responsabile de asigurarea unui canal de comunicare între mediu și DUT, și menținerea integrității protocolului utilizat.

Interfețele au ca și intrare semnalele sistemului (ceas și reset), permițând sincronizarea acestuia cu componentele de verificare și dispozitivul verificat. Această sincronizare se realizează prin intermediul grupurilor de sincronizare (clocking block), restricționând momentul comutării semnalelor (în cazul nostru pe frontul pozitiv al ceasului de sistem), permiterea introducerii unui timp de propagare și asigurând direcția semnalelor (intrarea sau ieșire).

```
//master clocking block
  clocking mst_cb @(posedge pclk);
    input  prdata ;
    input  pready ;
    input  pslverr;
    output psel   ;
    output penable;
    output pwrite ;
    output paddr  ;
    output pwdata ;
  endclocking: mst_cb
```

Codul 1 Exemplu clocking block pentru interfața APB master

Asigurarea integrității protocolului se realizează prin intermediul aserțiilor (SVA), monitorizând semnalele din interfața respectivă pe tot parcursul simulării, asigurându-se că semnalele se comportă conform protocolului utilizat și semnalarea erorilor in caz contrar. Proprietățile ce trebuie respectate de semnalele interfeței sunt definite de către utilizator, inclusiv gradul de severitate a violărilor de protocol sau momentul eșantionării semnalelor.

```
property not_unknown(signal,dsbl);
  @(posedge pclk) disable iff(dsbl)
    !$isunknown(signal);
endproperty

assert property (not_unknown(psel,~preset_n)) else $error("PSEL must not
be unknown while reset is not asserted"   );
```

Codul 2 Exemplu de SVA

În fragmentul de cod 2 se poate observa proprietatea `not_unknown` ce funcționează similar cu o funcție cu 2 parametrii. Cuvântul cheie `assert` urmat de proprietatea `not_unknown` atribuie proprietatea descrisă parametrilor săi pe tot parcursul simulării, afișând mesajul de eroare din funcția de sistem `$error` în cazul in care semnalele nu îndeplinesc proprietatea respectivă. Eșantionarea are loc la fiecare front pozitiv al semnalului de ceas, indicat de structura `@(posedge pclk)`.

4.2 AGENTUL APB

Agentul APB emulează un dispozitiv care folosește protocolul respectiv, dând stimuli DUT-ului și analizând tranzacțiile realizate cu acesta. Integritatea protocolului este asigurată de aserțiile interfeței APB, semnalând violările de protocol făcute de către ambele partide.

Agentul permite configurarea de tip (master/slave), dacă să eșantioneze sau nu acoperirea funcțională și dacă rolul acestuia să fie activ sau pasiv. Prin pasivitatea agentului se înțelege crearea doar a componentei de monitorizare, permițând agestuia să fie legat între două componente RTL fără perturbarea comunicării acestora.

4.2.1 Interfața virtuală

Accesul agentului către interfața APB se realizează printr-o interfață virtuală, adică un pointer. Prin acest pointer agentul poate accesa semnalele interfeței prin intermediul grupului de sincronizare (clocking block).

Interfața APB are la dispoziție 3 asemenea grupuri, pentru funcționalitățile modului master și slave, precum și a monitorizării obiective a semnalelor. Grupurile dedicate funcționării modurilor master și slave sunt desemnate driverului, acestea având direcția semnalelor (intrare/ieșire) conform tipului acestora. Configurarea de tip (master/slave) a agentului APB indică folosirea grupului respectiv de sincronizare de către driverul acestuia.

Deoarece monitorul este un element pasiv, necomutând semnalele din interfață, acestuia i se atribuie un grup de sincronizare având direcția semnalelor ca și intrare, astfel asigurând o eșantionarea uniformă pe toate semnalele, acestea având sursele externe și timpul de propagare identic.

În afara semnalului de resetare a sistemului, toate semnalele au comutarea și eșantionarea sincronizată pe blocul corespunzător de sincronizare, care la rândul lui este sincronizat pe ceasul de sistem. Astfel unitatea atomică de timp suportată de agentul APB este o perioadă a ceasului de sistem. Perioada de timp între 2 fronturi pozitive ale ceasului de sistem nu este eșantionată iar schimbarea valorii semnalelor de către agent implică și așteptarea frontului înainte de a comuta.

4.2.2 Obiectul de tranzacție

În domeniul acesta de aplicare, obiectul de tranzacție este structura de date ce abstractizează caracteristicile transferului, indiferent de protocolul de comunicații folosit.

Obiectele pot conține de exemplu adresa și datele transferate, dar și atribute mai subtile precum numărul de perioade de ceas între 2 tranzacții sau timpul de răspuns al dispozitivului.

Tabelul 2.Obiectul de tranzacție APB

Nume	Lățime	tip	descriere
kind	1	apb_trans_kind_t	operația tranzacției pe magistrală (scriere/citire)
addr	AW	bit	adresa accesată
data	DW	bit	datele scrise/citite în funcție de tipul agentului și de operația executată pe magistrală
delay	32	bit	numărul perioadelor de ceas între 2 tranzacții
delay_kind	3	apb_delay_kind_t	categorii de întârzieri între tranzacții. (null, short, medium, long, max)
resp	1	apb_trans_resp_t	răspunsul dat de către slave (error/ok)
ready delay	32	bit	numărul perioadelor din faza de acces înainte de răspunsul slave-ului

În tabelul 1 se pot observa atributele obiectului de tranzacție APB. Tipurile de date apb_trans_kind_t, apb_delay_kind_t și apb_trans_resp_t sunt definite de către inginerul de verificare, codificând valoarea numerică în format text pentru a simplifica referința la valoarea acestora pe parcursul dezvoltării mediului de verificare. Valorile AW și DW sunt parametrii care permit schimbarea lățimilor respective însă necesită recompilarea codului.

Aceste date sunt randomizate respectând anumite constrângeri specifice testului rulat, ulterior transmise către driver pentru a fi translatate la nivel de protocol. Constrângerile pot fi de 2 feluri: hard și soft. Constrângerile de tip soft permit suprascrierea lor totală de către un alt element a mediului de verificare, iar cele hard permit doar adăugarea unor noi atribute fără perturbarea celor originale.

Obiectele tranzacției sunt folosite atât în generarea de stimuli cât și în monitorizarea transferurilor, driverul utilizând datele din tranzacție pentru a genera stimulii corespunzători pe interfață iar monitorul colectând atributele transferului de pe interfață în obiect pentru a face transmiterea acestora mai ușoară între componentele de verificare.

4.2.3 Sequencer

Sequencer-ul este componenta de verificare care manipulează secvențele ce conțin stimulii necesari pentru a realiza tranzacțiile, făcând arbitrare între ele și trimițând tranzacțiile din aceștia prin intermediul unui port TLM către driver pentru ca acesta să execute tranzacția pe interfață, ulterior semnalând finalizarea acesteia și permiterea trimiterii unor noi tranzacții.

Secvențele se primesc din vârful ierarhic, adică din test, iar randomizarea tranzacțiilor se realizează utilizând constrângerile specifice testului sau în cazul absențelor acestora se utilizează cele de bază predefinite în clasa tranzacție.

4.2.4 Driver

Driverul este componenta activă cea mai apropiată de nivelul RTL, acesta fiind responsabil de generarea stimulilor pentru a realiza comunicarea între dispozitivul verificat și mediul de verificare prin intermediul interfeței și a protocolului de comunicare utilizat.

Acesta primește o tranzacție de către sequencer, extrage informația din aceasta și execută transferul pe interfață conform atributelor de tranzacție primite. În funcție de tipul agentului (master/slave), driverul extrage date diferite din tranzacția primită de către sequencer și manipulează semnale diferite din interfață.

Metoda de rulare principală a driverului APB este despărțită în 2 fire de execuție după primirea tranzacției. Un fir este dedicat manipulării semnalelor în condiții optime de transfer iar celălalt așteaptă semnalul de resetarea a sistemului, și la momentul apariției acestuia întoarce valoarea semnalelor din interfață la valoarea lor inițială. Finalizarea oricărui fir de execuție implicit omoară celălalt fir, ulterior semnalând finalul transferului către sequencer.

4.2.5 Monitor

Responsabilitatea monitorului constă în colectarea tranzacțiilor și a atributelor acestora de pe interfață, ulterior trimițându-le componentelor responsabile eșantionării acoperirii funcționale și către monitorul global pentru a putea fi verificate integritatea specificațiilor sistemului verificat.

Monitorul utilizează o instanță diferită a obiectului tranzacție a agentului, datele și atributele transferului sunt stocate în acest obiect, încapsularea lor simplificând transferul acestora. Transferul obiectelor de tranzacție re realizează prin porturi TLM de analiză (TLM analisys port) ce permit accesul a mai multor componente simultan la obiectul respectiv.

Deoarece obiectul din portul de analiză se consideră a fi resursă comună între componentele conectate la aceasta există posibilitatea apariției fenomenului numit race condition, ce indică alterarea obiectului de către o componentă înainte citirii acesteia de către o altă componentă, făcând ca datele din obiect să nu mai fie relevante. Pentru a evita acest fenomen se recomandă clonarea locală a obiectului din port de către fiecare componentă astfel fiecare alterație a obiectului nu părăsește domeniul de aplicare a componentei.

Similar cu driverul APB, structura monitorului este pe 2 fire de execuție, unul dedicat colectării și trimiterii tranzacțiilor iar celălalt dedicat așteptării semnalului de resetarea a sistemului. La apariția semnalului de reset variabilele locale sunt readuse în starea lor inițială și dacă o tranzacție în parcurs de colectare acesta este abandonată.

Cum această componentă cu rol de monitorizarea este singura care are acces la interfață, acoperirea funcțională ce ține de semnalul de reset și momentul apariției acesteia sunt prelucrate în monitor și nu în componenta responsabilă acoperirii funcționale de protocol.

La sfârșitul simulării monitoarele raportează numărul de tranzacții colectate, număr ce poate fi comparat cu numărul de tranzacții inițiate de către driver pentru a autoverifica corectitudinea transmiterii și colectării tranzacțiilor

4.2.6 Coverage

Această componentă are ca principal scop eșantionarea acoperirii funcționale ale protocolului APB prin intermediului grupurilor și punctelor de acoperire (covergroup/coverpoint) definite de către inginerul de verificare.

Tranzacțiile și atributele acestora ajung la componentă prin portul TLM implicit clasei, moștenită din clasa părinte uvm_subscriber având metoda write, care este apelată automat în momentul trimiterii tranzacției prin port de către monitor. După primirea tranzacției, se creează o clonă locală a obiectului de tranzacție și se apelează funcția de eșantionare a grupului de acoperire salvând atributele tranzacției respective și combinațiile acestora.

```
read_write_cov: coverpoint trans.kind{
      bins write = {APB_WRITE};
      bins read  = {APB_READ };
    }

resp_kind_cov: coverpoint trans.resp{
      bins okay  = {APB_OKAY};
      bins error = {APB_ERROR};
    }
```

Codul 3 Exemplu de puncta de acoperire în componenta APB

În fragmentul de cod 3 se pot observa 2 puncte de acoperire ce eșantionează tipul tranzacției și răspunsul primit de către slave. Ambele puncte de acoperire fac parte din același grup de acoperire, ele fiind eșantionate simultan.

La finalul simulării aceste eșantioane sunt salvate într-o bază de date ce poate fi interpretată de diverse aplicații software, interpretând datele și formând o imagine de ansamblu asupra obiectivelor testate, fiind vizibil și procentajul acoperirii grupului, evidențiind punctele și combinațiile atributelor neatinse.

4.3 Agentul I2C

Agentul I2C este componenta responsabilă în generarea de stimuli și monitorizarea tranzacțiilor de pe magistrala respectivă. Acesta permite configurări de tip (master/slave) precum și adresa la care să răspundă dispozitivul în cazul în care este configurat în modul slave.

Numărul maxim al agenților I2C suportați în mediul de verificare este 128 indiferent de tipul acestora, însă este necesară o măsură de precauție pentru a nu permite ca mai mulți agenți slave și/sau dispozitivul verificat să aibă aceeași adresă de acces.

Agentul are toate caracteristicile unui dispozitiv multimaster I2C acesta asigurându-se că magistrala este inactivă înainte de a executa un transfer pe aceasta și cedarea controlului pe magistrală în momentul sesizării pierderii unei arbitrări. Pe lângă aceasta agentul suportă și fenomenul de clock stretching inițiat de către un dispozitiv slave, prin citirea constantă a semnalului de ceas SCL după comutarea acestuia în 1 logic și așteptarea eliberării semnalului de către slave în momentul în care s-a observat că semnalul nu s-a comutat corespunzător.

4.3.1 Interfața virtuală

Interfața virtuală este mediul prin care agentul I2C accesează magistrala, pentru a realiza un canal de comunicare cu dispozitivul verificat. Acesta conține 2 semnale bidirecționale SCL și SDA care sunt manipulate alternativ de către DUT și driverul agentului.

Cum ambele semnale sunt bidirecționale (DUT-ul suportând fenomenul de clock stretching) interfața nu necesită grupuri de sincronizare diferite pentru tipul agentului (master/slave). Semnalele fiind de tipul triand, în cazul manipulării diferite ale acestora din mai multe surse, semnalul va prelua valoarea 0 logic în cazul în care cel puțin unul din sursele acestuia are valoarea respectivă și 1 logic când toate sursele acesteia au valoare aceasta, similar cu o poartă logică și.

wand/triand	0	1	X	Z
0	0	0	0	0
1	0	1	X	1
X	0	X	X	X
Z	0	1	X	Z

Figura 8 caracteristicile semnalului de tip traind [10]

Interfața are ca și intrare semnalul de resetarea a sistemului și un ceas funcțional cu o frecvență mai mare decât frecvența maximă a semnalului SCL, cu ajutorul căreia se manipulează și eșantionează semnalele de pe interfață de către agent. Astfel unitatea atomică de timp a agentului I2C este perioada ceasului funcțional.

Asigurarea integrității protocolului este minimal în interfață, acesta având aserții pentru cunoașterea stării semnalelor pe parcursul simulării și că semnalul SCL nu comută în afara cadrului de tranzacție.

4.3.2 Obiectul de tranzacție

Obiectul de tranzacție a agentului I2C, similar cu cel al agentului APB reprezintă o abstractizare a atributelor transferului ce urmează a fi executat pe magistrală sau a fost colectat de pe aceasta.

Tabelul 3 Obiectul de tranzacție I2C

Nume	Lățime	tip	descriere
kind	1	i2c_trans_kind_t	operația tranzacției pe magistrală (scriere/citire)
addr	7	bit	adresa accesată
data	8	bit [$]	datele scrise/citite în funcție de tipul agentului și de operația executată pe magistrală
delay	32	bit	numărul perioadelor de ceas între 2 tranzacții
delay_kind	3	i2c_delay_kind_t	categorii de întârzieri între tranzacții. (null, short, medium, long, max)
Resp	1	i2c_trans_resp_t [$]	răspunsul primit după transmiterea adresei sau a unui octet (ACK/NACK)
repeated_start	1	bit	prezența fenomenului de repeated start în tranzacție
clock_strech	1	bit	prezența fenomenului de clock stretch start în tranzacție
clock_period	32	bit	perioada semnalului SCL în perioade de ceas funcțional
arb_lost	1	bit	semnalarea pierderii arbitrajului

În tabelul 3 se pot observa atributele ce construiesc obiectul de tranzacție I2C. Similar cu obiectul tranzacție există tipuri de date definite de inginerul de verificare.

Simbolul $ reprezintă structura de date de tip coadă, având la dispoziție metodele size, push și pop, folosite de driverul și monitorul agentului, simplificând semnificativ colectarea octeților de pe magistrală și eficientizând utilizarea memoriei de către simulator, acestea fiind alocate dinamic.

4.3.3 Driver

Driverul agentului I2C este componenta activă în comunicarea cu dispozitivul verificat prin intermediul stimulilor generați și transmiși pe magistrală utilizând interfața virtuală. După cum a fost menționat anterior interfața utilizează o singură structură de sincronizare pentru ambele tipuri de agenți (master/slave) ce permite oricărui agent să monitorizeze activitatea pe magistrală chiar dacă agentul respectiv nu face parte din transferul respectiv.

Prin această metodă de implementare driverul master citește semnalele de pe magistrală pentru detectarea condițiilor de start și de stop care delimitează cadrul de transfer, și în cazul în care o secvență a fost lansată după detectarea condiției de start, acesta așteaptă detecția condiției de stop înainte de a executa tranzacția.

Similar în manipularea semnalului de ceas SCL după fiecare comutare din 0 în 1 logic de se așteaptă o perioadă de ceas funcțional și se citește valoarea acesteia pentru a determina că într-adevăr semnalul s-a comutat. În cazul în care acesta nu s-a comutat, driverul detectează fenomenul de clock stretching, cedând controlul asupra semnalului și așteptând eliberarea acestuia de către slave. Cum cadrul de transfer este sincronizat pe frontul crescător al semnalului SCL, semnalul de date SDA rămâne constant până la eliberarea semnalului de ceas.

Detectarea accesării simultane a magistralei de către 2 dispozitive master se realizează într-un mod similar cu detectarea fenomenului de clock stretching, doar că pe semnalul SDA. De fiecare dată când aceasta este comutat în 1 logic pe palierul negativ al semnalului SCL se așteaptă momentul eșantionării semnalului, adică frontul pozitiv al semnalului de ceas SCL, se recitește semnalul și dacă acesta se află în 0 logic, driverul sesizează că magistrala este controlată de un alt dispozitiv și acesta cedează controlul asupra ei, semnalează pierderea arbitrajului și încheie tranzacția dând semnalul de item done sequencerului. Tranzacția nu este reluată la eliberarea magistralei.

Fluxul driverului master este structurat pe mai multe fire de execuție, similar ca la driverul APB unul din acestea este dedicat verificării semnalului de resetare a sistemului. Față de driverul APB, acesta necesită mai multe fire de execuție pentru transferul normal, fiind nevoit de a manipula 2 semnale în paralel și se poate ajunge la încheierea transferului din mai multe cauze, acestea fiind ne receptarea datelor sau a adresei de către slave, pierderea arbitrajului sau execuția transferului în condiții optime. La întâmpinarea oricăruia din aceste fenomene se trece în starea de încheiere a transferului semnalând o condiție de stop sau de repeated start (mai puțin la pierderea arbitrajului) și semnalarea sequenserului că tranzacția a fost încheiată.

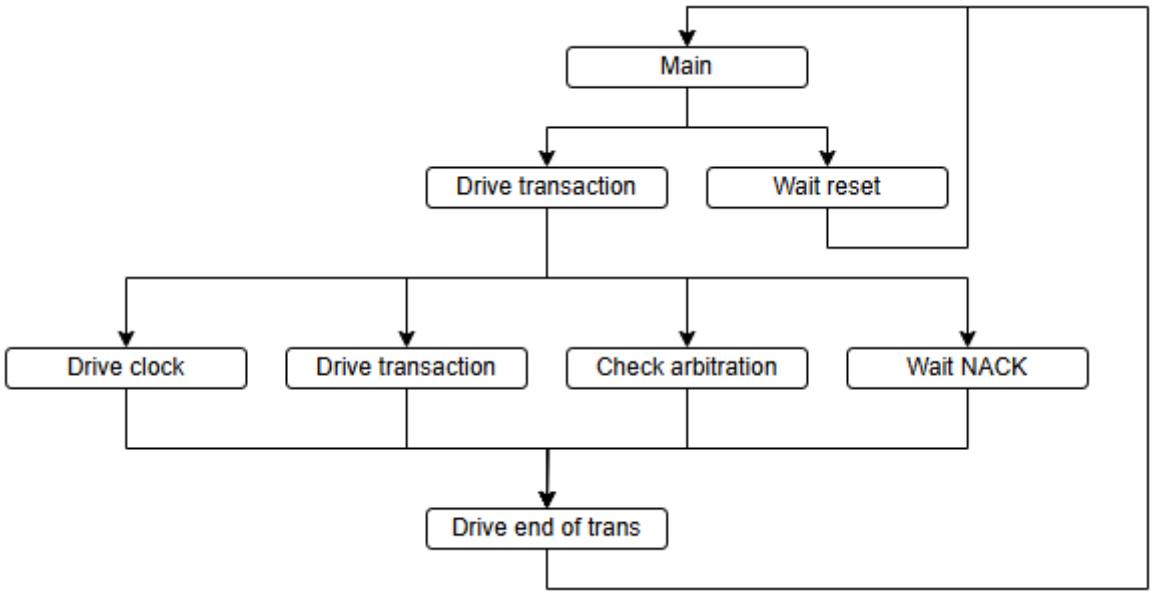

Figura 9 Structura firelor de execuție în driverului I2C master

Fluxul driverului slave este împărțit în 2 faze: faza de adresă și faza de date. În faza de adresă se eșantionează bit cu bit adresa, comparând-o cu cea a dispozitivului și dacă acestea sunt identice, pe baza bitului de operație, se trece în faza de transmisie sau recepție a datelor. În cazul în care adresa recepționată nu coincide cu cea de dispozitiv, driverul comută semnalul SDA în 1 logic, însemnând cedarea controlului de pe semnal și transmisia răspunsului NACK. Faza de date este de 2 feluri și este determinată de bitul de operație. În cazul operației de scriere driverul așteaptă 8 perioade de SCL după care ia controlul semnalului SDA pentru a oferi un răspuns, iar în cazul operației de citire driverul furnizează datele pe magistrală și așteaptă răspunsul dispozitivul master.

4.3.4 Monitor

Similar cu agentul APB, interfața virtuală I2C conține un grup de sincronizare separat pentru monitor, având direcția tuturor semnalelor ca și intrare pentru ca acestea să aibă același timp de propagare, evitând eventualele probleme de eșantionare.

Monitorul are ca principal scop colectarea tranzacțiilor de pe magistrală, dar și verificarea anumitor aspecte ale acestora, aspecte care necesită o complexitate mare pentru a fi verificare prin intermediul aserțiilor din interfață.

Fluxul monitorizării magistralei constă în primul rând în detectarea condiției de start pe magistrală, urmat de colectarea adresei și a bitului de operație, eșantionând semnalul SDA și concatenând biții transmiși ulterior salvându-le în obiectul tranzacție al monitorului. Cum lungimea transferului nu este indicată în antetul tranzacției, monitorul este nevoit să colecteze datele și recepționările acestora până la detectarea condiției de stop. Acest lucru se realizează prin execuția monitorizării pe mai multe fire de execuție, unul pentru colectarea octeților și stocarea lor în coadă, celelalte așteptând una din condițiile de finalizare ale tranzacției. La apariția uneia din condițiile acestea, firul de execuție al monitorizării este ucis, iar monitorul intră

în faza de trimitere a tranzacției către monitorul global (scoreboard) și către componentele responsabile acoperirii funcționale.

Câmpul `repeated_start` al obiectului de tranzacție este completat de către monitor în funcție de modul finalizării transferului (condiție de start/stop). Implementarea aceasta împarte tranzacțiile ce conțin repeaded start în mai multe tranzacții a căror număr este determinat de numărul condițiilor de start detectate.

Câmpul `clock_stretch` și perioada ceasului SCL sunt completate numărând perioadele de ceas funcțional între 2 fronturi negative ale semnalului SCL în faza trimiterii adresei de către master (perioada semnalului SCL fiind constantă în această fază), salvând valoarea perioadei ca model de referință. Pe parcursul transferului se execută similar numărarea perioadei semnalului SCL, comparând-o cu modelul de referință. În momentul sesizării că perioada numărată este mai mare decât cea de referință indică apariția fenomenului de clock stretching.

```
task collect_clock_period(output int per);
    per = 0;
    @(negedge vif.scl);

    fork: count_period
      forever begin
        @(vif.mon_cb);
        per++;
      end
      @(negedge vif.scl);
    join_any
    disable count_period;
  endtask
```

Codul 4 funcția ce numără perioada semnalului SCL

Cum toți agenții sunt legați la aceeași interfață monitoarele acestora au posibilitatea de a colecta tranzacțiilor neaparținătoare agenților lor și trimiterea acestora către scoreboard, asftel acesta primește de la fiecare agent câte o tranzacție identică simultan. Pentru a combate suprasolicitarea scoreboardului și a putea extrage o statistică validă din acoperirea funcțională din test (evitarea eșantionării multiple a tranzacției), doar portul TLM al agentului cu indexul 0 este legat la scoreboard.

4.3.5 Coverage

Acoperirea funcțională specifică protocolului I2C este structurată pe 2 grupuri, una fiind dedicată adresei accesate și a datelor de control ale transferului precum operația, lungimea și diverse combinații ale acestora, iar cealaltă pentru asigurarea comutarea semnalului SDA la fiecare bit pe parcursul transferului datelor (ex. bitul 7 din octetul transmit a avut starea 0 și 1 logic).

Prin separarea celor 2 grupuri se asigură eliminarea eșantionării redundante, oferind date precise legate de numărul eșantioanelor a stărilor atinse pe parcursul unei simulări, pentru crearea unei statistici cât mai precise.

4.4 AGENTUL IRQ

Agentul de întrerupere este unul pasiv, acesta având un singur semnal de intrare pe interfață, și care nu necesită generator de stimuli, reducându-se la o singură componentă de monitorizare.

4.4.1 Monitor

Monitorul agentului de întreruperi are ca scop detectarea fronturilor semnalului IRQ și semnalarea acestora componentelor de verificare specifice dispozitivului verificat. Fluxul acestuia se desfășoară pe un singur fir de execuție, acesta așteptând frontul pozitiv al întreruperii, semnalarea acestuia, apoi similar cu frontul negativ. Prin acest mod de semnalare se ajunge la o eficiență sporită de simulare, reducând eșantionarea la punctele cheie ale semnalului, adică comutarea acestuia.

4.5 VIRTUAL SEQUENCER

Sequencer-ul virtual se comportă ca o graniță între componenta test și mediul de verificare, acesta având pointeri spre secvențiatoarele fiecărui agent, cum și la modelul de regiștrii. Prin această componentă secvențele din test sunt distribuite agenților corespunzători, permițând componentei de test controlarea fluxului de simulare prin tipul secvențelor lansate și momentul lansării acestora.

Având pointer spre modelul de regiștrii, acesta permite utilizarea metodelor predefinite din biblioteca UVM pentru accesarea, prezicerea și compararea valorilor din model și valorilor din DUT, cum și permiterea lansării secvențelor APB.

4.6 SCOREBOARD

Scotreboard-ul este componenta principală în verificarea caracteristicelor specifice DUT-ului și de a eșantiona acoperirea funcțională a acestuia. Începând de al acest nivel componentele își pierd din versatilitate și încep să depindă din ce în ce mai mult de DUT.

Principalele funcții ale componentei sunt automatizarea verificării caracteristicelor și colectarea acoperirii funcționale specifice dispozitivului verificat. Acesta colectează tranzacții de la fiecare agent și compară atributele acestora cu specificațiile configurării actuale pentru a verifica integritatea și comportamentul dispozitivului.

4.6.1 Verificarea funcționalității

Utilizarea modelului de regiștrii UVM simplifică verificarea semnificativ, salvând valorile scrise în regiștrii DUT-ului cunoscând pe tot parcursul simulării configurările acestuia. Bazat pe

aceste configurări, scoreboard-ul verifică diferite aspecte ale dispozitivului, ignorând caracteristicile ce nu sunt relevante în configurarea actuală.

Majoritatea verificării se reduce la interpretarea configurațiilor DUT-ului și prezicerea comportamentului acestuia prin intermediul tranzacțiilor primite de la agenți. Tot prin intermediul tranzacțiilor se prezic valorile regiștrilor care se pot accesa doar prin citire, permițând metodei mirror să compare valoare din registru cu valoare prezisă automat în momentul citirii registrului.

4.6.2 Acoperire funcțională

Scoreboard-ul este componenta responsabilă și de eșantionarea acoperirii funcționale specifice dispozitivului verificat. La momentul primirii unei tranzacții de către un agent, se creează o copie după aceasta și i se eșantionează atributele.

Diferența față de componentele de acoperire funcțională ale agenților este că acesta nu este limitată la un singur tip de tranzacție, eșantionând combinațiile atributelor de tranzacții I2C și APB, astfel obținându-se un procentaj referitor volumului de caracteristici verificate.

Grupurile de acoperire funcțională referitoarea regiștrilor sunt eșantionate doar în momentul accesării și schimbării valorii acestora, eliminând posibile eșantionări redundante astfel obținând o statistică precisă la finalul simulării referitoare la stările și configurațiile verificare.

```
covergroup status_reg_cov @(status_acces_e);
  option.per_instance = 1;
  byte_cnt_tog: coverpoint byte_cnt {
      bins minim     = {0};
      bins maxim     = {256};
      bins range[10] = {[1:255]};
    }
    arb_lost_tog: coverpoint arb_lost;
    nack_tog:     coverpoint nack    ;
    bsy_tog:      coverpoint bsy     ;
    tip_tog:      coverpoint tip     ;
  endgroup
```

Codul 5 exemplu de eșantionare în momentul accesării regitrului

În fragmentul de cod 5 se poate observa grupul de acoperire a câmpurilor registrului de status. Acesta este eșantionat automat în momentul apariției evenimentului `status_acces_e` care este emis în momentul citirii acestui registru, determinat de atributul adresă din obiectul de tranzacție APB primit.

4.7 REG MODEL

Modelul de regiștrii este structurat astfel încât să mimice precis comportamentul regiștrilor din DUT, creând un așa numit golden model, care reduce verificarea la compararea modelului cu DUT-ul.

Actualizarea valorilor din model se realizează în 2 feluri: prin adaptorul de magistrală sau prezicerea valorilor.

4.7.1 Adaptor

Adaptorul de magistrală este componenta modelului responsabilă pentru translatarea tranzacțiilor de pe interfață, în cazul actual APB, și actualizarea corespunzătoare a valorii regiștrilor. Această translatare are loc în metoda `bus2reg` a adaptorului.

Adaptorul are posibilitatea de a face această translație în ambele direcții, permițând inițierea tranzacțiilor pe interfața APB prin metoda `reg2bus`, astfel simplificând accesarea regiștrilor din DUT prin simpla accesare a modelului.

4.7.2 Predictor

Predictorul modelului de regiștrii este componenta responsabilă pentru actualizarea valorilor regiștrilor ce pot fi accesați doar prin citire. În momentul apelării funcției predict, acesta actualizează valoarea din model cu cel dat de către utilizator, fără de a efectua o tranzacție pe magistrală, astfel permițând verificarea folosind metoda mirror.

4.8 SECVENȚE

Secvențele sunt mediul prin care componenta test realizează comunicarea cu agenții mediului pe parcursul simulării. Aceste secvențe sunt specifice tipului de agent (I2C/APB) și se pot lansa secvențial sau în paralel. În cazul lansării mai multor secvențe prin același sequencer, acesta creează o coadă în ordinea sosirii acestora sau realizează o arbitrare între acestea după caz pentru a determina ordinea executării acestora.

Modul de dezvoltare a secvențelor a avut ca principal scop reducerea numărului necesar de secvențe pentru abordarea scenariilor de verificare impuse, permițând utilizarea acestora de mai multe teste. O asemenea secvență este cea de configurare, prin intermediul căreia se trimit tranzacții pe interfața APB, accesând regiștrii din DUT și configurându-l pe acesta după scenariul specific de verificare.

4.9 TEST

Componenta test este vârful ierarhiei mediului de verificare prin intermediul căreia se coordonează simularea și configurarea specifică a mediului. Componenta utilizează secvențele pentru a realiza o comunicare cu agenții mediului, implicit cu DUT-ul. Momentul și timpul lansării secvențelor și prin sequencer-ul cărui agent are loc lansarea determină caracteristicile principale ale unui anumit test.

Structura testelor este realizată prin metoda de moștenire OOP, având un test de bază care creează structura de bază a mediului de verificare, moștenitorii acestuia configurând ulterior mediul pentru a respecta cerințele necesare parcurgerii testului.

Testele se pot împărții în mai multe categorii în funcție de scopul acestuia (ex. test de citire, test de arbitrare etc.) însă se dorește a avea teste cât mai puțin restrictive, cu o acoperire cât mai marea a atributelor verificate într-o singură rulare. Verificând acoperirea funcțională, se poate crea un test specific care să verifice părțile neacoperite ce se dovedesc a fi greu sau poate chiar imposibil de atins prin testele aleatoare.

4.10 Top

Modulul top este mediul în care se instanțiază DUT-ul, interfețele și testele, și de unde pornește simularea prin selectarea testului ce urmează să fie rulat. Modulul asigură o comunicare cu utilizatorul prin linia de comandă prin intermediul căreia se selectează caracteristicile simulării respective, caracteristici precum testul ce se dorește a fi rulat, valoarea de referință a randomizării (seed) sau verbozitatea simulării respective.

5 UTILIZAREA SIMULATORULUI ȘI RULAREA TESTELOR

Utilizarea simulatorului
Parametrii unei simulări
Încapsularea comenzilor în Makefile

5.1 UTILIZAREA ȘI CARACTERISTICILE SIMULATORULUI

Rularea testelor specifice sau a regresiilor se realizează prin intermediul terminalului unei interfețe Linux, utilizând comenzile specifice simulatorului Questa. Pentru compilarea ușoară a fișierelor de design și a codului sursă a mediului de verificare, s-au structurat fișiere ce includ calea spre fișierele ce necesită a fi compilate. Calea spre fișiere este realizată în stil bottom up, pentru a sigura vizibilitatea instanțelor și a tipurilor de date definite de către utilizator.

Simulatorul permite utilizarea acestuia cu sau fără interfață grafică, în funcție de necesitatea de performanță sau de versatilitate, interfața grafică oferind posibilitatea de a viziona mai în detaliu a simulării, inclusiv a formelor de undă, dar tot odată utilizând mai multe resurse în realizarea acestor aspecte.

Bazele de date referitoare la acoperirea funcțională și cea de cod sunt salvate într-un fișier cu extensia ucdb, permițând manipularea acestora de către simulator, privind vizualizarea datelor și combinarea a mai multor baze de date având aceeași extensie de fișier, permițând vizualizarea în ansamblu a datelor colectate pe parcursul a mai multor simulări.

5.2 IMPLEMENTAREA COMENZILOR SIMULATORULUI ÎNTR-UN MAKEFILE

Pentru simplificarea utilizării simulatorului și manipularea directoarelor și fișierelor generate de către simulator, se utilizează scripturi shell folosind limbajul bash. Prin intermediul acestor scripturi se formează directoare pentru a împărții fișierele generate de către simulator în diverse categorii, ușurând utilizatorului accesarea acestora. În cazul rulării unei regresii, utilizarea acestor scripturi permit utilizatorului de accesa fișierele fiecărei simulări în parte, vizionând caracteristicile și rezultatele acestora.

Comenzile simulatorului împreună cu aceste scripturi sunt încapsulate într-un Makefile, permițând parametrizarea și condiționarea acestora. Încapsularea se realizează prin definirea mai multor comenzi ce se doresc a fi executate secvențial și atribuirea unor termene cheie acestor structuri, iar prin apelarea din terminal a comenzii `make` urmat de termenul respectiv se execută structura de comenzi. Această apelare poate conține și suprascrierea parametrilor predefiniți din fișier, oferind o versatilitate ridicată și ușurință de utilizare.

```
.PHONY: run_cov_multiple

run_cov_multiple:
      for i in {1..${NUM_SEEDS}}; do \
            make run_cov TOOL=${TOOL} TESTNAME=${TESTNAME} SEED=$$RANDOM;\
      done\
```

Codul 6 Exemplu de încapsulare a unei structuri de comenzi

Fragmentul de cod 6 exemplifică o structură încapsulată sub termenul cheie `run_cov_multiple` utilizând parametrii referitori la simulatorul utilizat, numele testului rulat și numărul rulărilor, utilizând seed-uri aleatorii. Prin acest mod de implementare se realizează o utilizare simplificată a mediului de verificare, având cele mai des utilizate combinații de comenzi încapsulate sub o singură comandă.

6 REZULTATE OBȚINUTE ȘI CONCLUZIE

Implementarea mediului folosind structura UVM a permis automatizarea verificării efectuate de aceasta, reducând interacțiunea cu utilizatorul și permițând acestuia de a trata scenariile de verificare impuse prin intermediul secvențelor de simulare.

Rularea testelor individual cu seed-ul de randomizare predefinit a permis recrearea scenariilor care au generat erori de funcționare, ca acestea să poată fi revizuite de către proiectant și de a putea rezolva erorile respective într-un mod eficient. Pe de altă parte, rularea testelor prin intermediul unei regresii, permite ca mai multe teste să fie rulate consecutiv, salvând datele ce țin de acoperirea funcțională și de cod a dispozitivului într-o manieră automată și eficientă.

În urma rulării regresiei ce conține testele de bază, s-au găsit mai multe erori de funcționare a dispozitivului verificat, de exemplu actualizarea întârziată sau eronată a registrului status, ce au fost semnalate proiectantului pentru remedierea acestora. După remediere și trecerea testelor de către dispozitiv, s-a colectat acoperirea funcțională și cea de cod ale dispozitivului utilizând metodele menționate.

Covergroups	Bins	Hits	Misses	Goal	Coverage
Search...	Search...	Search...	Search...	Search...	Search...
/src_package/coverage/coverage__1/tx_fifo...	16	16	0	100	100%
/src_package/coverage/coverage__1/rx_fifo...	16	16	0	100	100%
/src_package/coverage/coverage__1/addr_r...	80	46	34	100	71.87%
/src_package/coverage/coverage__1/ctrl_re...	30	23	7	100	80%
/src_package/coverage/coverage__1/cmd_r...	12	10	2	100	85.71%
/src_package/coverage/coverage__1/status...	20	15	5	100	66.66%
/src_package/coverage/coverage__1/irq_reg...	68	50	18	100	85.62%
/src_package/coverage/coverage__1/irq_ma...	12	12	0	100	100%
/src_package/coverage/coverage__1/divider...	18	14	4	100	77.77%

Figura 10 Raport de acoperire funcțională referitoare la caracteristicile specific dispozitivului

Regresia cu teste de bază a adus rezultate promițătoarea, prin intermediul acesteia s-a obținut o acoperire funcțională de 85,29% și acoperire de cod de 80,87%, urmând să fie testate caracteristicile mai complexe ale dispozitivului pentru a realiza un procent de acoperire cât mai complet.

S-a constatat și faptul că unele aspecte referitoare la acoperirea funcțională a protocoalelor nu pot fi eșantionate, deoarece modul în care dispozitivul a fost proiectat nu permite ca aceasta să ajungă în stările respective. Din acest motiv s-a creat un fișier de excluderi care ignoră aspectele menționate anterior în momentul formării procentajului, oferind o viziune mai clară asupra nivelului de verificare efectuat.

În concluzie, după rezultatele obținute în momentul de față raportate la metoda de implementare utilizată, mediul de verificare s-a dovedit a fi eficient și robust în verificarea integrității datelor și a semnalelor de control, cât și în colectarea acoperirii funcționale

referitoare la protocoalele specifice de comunicare utilizate de agenții acestuia. Nivelul de configurare suportat de către agenții respectivi, permite implementarea acestora cu ușurință în proiecte viitoare, reducând semnificativ timpul de dezvoltare a mediilor de verificare, atribut care are tinde treptat în a deveni o necesitate în industrie.

7 BIBLIOGRAFIE

[1] Kropf, Thomas. *Introduction to formal hardware verification*. Springer Science & Business Media, 1999.

[2] D. Wang, J. Yan and Y. Qiao, "Research on Chip Verification Technology Based on UVM," 2021 6th International Symposium on Computer and Information Processing Technology (ISCIPT), Changsha, China, 2021, pp. 117-120

[3] S. N. Chauhan and G. K. Andurkar, "Development of UVM Testbench for I3C protocol," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-4

[4] A. Jain and R. Gupta, "Scaling the UVM_REG Model towards Automation and Simplicity of Use," 2015 28th International Conference on VLSI Design, Bangalore, India, 2015, pp. 164-169

[5] armDeveloper. *AMBA APB Protocol Specification*. https://developer.arm.com/documentation/ihi0024/c/

[6] P. Bagdalkar and L. Ali, "Hardware Implementation of I2C Controller on FPGA and Validation Through Interfacing with Low-Cost ADC," 2020 Fourth International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2020, pp. 887-891

[7] I2C-bus specification and user manual" (UM10204) by NXP Semiconductors

[8] I²C-Bus.org. (n.d.). *I²C – What's That?* In *I²C-Bus.org*. Retrieved June 8, 2025, from https://www.i2c-bus.org

[9] IEEE Computer Society. (2018, February 22). *IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language* (IEEE Std 1800-2017). IEEE.

[10] ChipVerify. "Verification – Assertion Based Verification (ABV), Gate Level Simulation & More." *ChipVerify*, 2015–2025. Available at https://www.chipverify.com (Accessed June 13, 2025).

[11] J. Bromley, "If SystemVerilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language," Proceedings of the 2013 Forum on specification and Design Languages (FDL), Paris, France, 2013, pp. 1-7.

[12] K. Salah, "Revisiting UVM," 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, 2024, pp. 1-4

[13] L. M. Kappaganthu, M. D. Prakash and A. Yadlapati, "I2C protocol and its clock stretching verification using system verilog and UVM," 2017 International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2017, pp. 478-480

[14] L. Shrinivasan, R. Verma, D. Koppad and M. R, "Development of Verification Environment for I2C Serial Communication Protocol using UVM," 2024 5th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2024, pp. 1-5

[15] P. Dwivedi, N. Mishra and A. Singh-Rajput, "Assertion & Functional Coverage Driven Verification of AMBA Advance Peripheral Bus Protocol Using System Verilog," *2021 International*

Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), Bhilai, India, 2021, pp. 1-6

REZUMAT

În această lucrare s-a implementat un mediu de verificare cât mai configurabil și reutilizabil utilizând biblioteca UVM a limbajului SystemVerilog, pentru un controller I2C. Controllerul este configurabil prin protocolul APB, ce permite schimbarea inclusiv a tipului de dispozitiv (master/slave). Din acest motiv componenta responsabilă în realizarea comunicației cu dispozitivul prin interfața I2C necesită a putea fi cât mai configurabilă, permițând verificarea dispozitivului din mai multe unghiuri utilizând mai multe instanțe a unui singur agent.

Pornind de la această idee, implementarea întregului mediu a avut ca și scop dezvoltarea componentelor, ca ele să fie cât mai universale și versatile, și să nu fie structurate doar pentru a satisface cerințele dispozitivului verificat, putând fi ulterior implementate în proiecte viitoare.

În primele 2 capitole sunt prezentate necesitatea componentelor universale în verificarea hardware și noțiuni teoretice despre protocoalele și metodologiile utilizate. În capitolul 3 se prezintă caracteristice de funcționare a dispozitivului verificat, iar capitolul 4 abordează structura și modul în care mediul de verificare a fost implementat. În capitolul 5 se prezintă uneltele prin intermediul cărora se realizează o simulare și cum se colectează datele ce țin de funcționalitate și acoperire funcțională discutate în capitolul 6.

ABSTRACT

In this paper, a highly configurable and reusable verification environment was implemented using the UVM library of the SystemVerilog language, for an I2C controller. The controller is configurable via the APB protocol, which also allows changing the type of device (master/slave). For this reason, the component responsible for establishing communication with the device through the I2C interface needs to be highly configurable, allowing verification of the device from multiple perspectives using several instances of a single agent.

Based on this idea, the implementation of the entire environment aimed to develop components to be as universal and versatile as possible, rather than being structured solely to meet the requirements of the verified device, thus enabling their use in future projects.

The first two chapters present the need for universal components in verification and theoretical concepts about the protocols and methodology used. Chapter 3 describes the operating characteristics of the verified device, while Chapter 4 addresses the structure and implementation of the verification environment. Chapter 5 presents the tools used for simulation and how data related to functionality and functional coverage are later discussed in chapter 6.