Calculatoare și Tehnologia Informației
2023-2024

Universitatea Politehnica din Timișoara

SISTEM DIGITAL DE VIZUALIZARE PRIN INTERMEDIUL VGA FOLOSIND COMUNICAREA PRIN INTERFAȚĂ SERIALĂ

Candidat: Denisa-Alexandra VEREȘ

Coordonator științific: Conf.dr.ing. Alexandru AMĂRICĂI-BONCALO

Sesiunea: Iunie 2024

REZUMAT

Lucrarea descrie proiectarea, realizarea, validarea și implementarea unui sistem digital de vizualizare prin intermediul VGA folosind comunicarea prin interfața serială UART. Designul este bazat pe transmisia datelor corespunzătoare culorilor dintr-o interfață serială UART într-un modul VGA pentru afișarea acestora. Sistemul propus este modular, compus din entități iindependente, configurabile și reutilizabile. Designul a fost realizat folosind un limbaj de descriere hardware, verificat folosind metodologia UVM și validat folosind aserții. Pentru simulare au fost folosite aplicațiile QuestaSim și ModelSim, iar pentru implementare a fost folosită aplicația Lattice Diamond.

CUPRINS

LISTA FIGURILOR

LISTA TABELELOR

1. INTRODUCERE

1.1. INTRODUCERE GENERALĂ

În cadrul acestei lucrări am urmărit proiectarea, realizarea și validarea unui sistem digital de vizualizare prin intermediul VGA folosind comunicarea prin interfața serială UART. Au fost urmăriți pașii de realizare a unui design digital validat, de la definirea cerințelor și specificațiilor la validarea sistemului și verificarea acestuia pe placă.

1.2. DOMENIILE ABORDATE

Pentru proiectarea digitală am folosit principiile de bază care susțin crearea sistemelor modulare, independente și reutilizabile. Pentru verificarea hardware am urmat metodologia UVM, standardul universal acceptat în domeniu, pentru a crea medii de verificare modulare și ușor de întreținut.

În realizarea lucrării am folosit aplicațiile: EasyEda și draw.io pentru proiectare și realizarea diagramelor, Wavedrom pentru realizarea formelor de undă ideale, Visual Studio Code pentru implementarea sistemelor, ModelSim și QuestaSim pentru simulare și Lattice Diamond pentru implementare pe FPGA.

1.3. SCOPUL ȘI OBIECTIVELE GENERALE

Scopul acestui proiect este proiectarea, realizarea și validarea unui sistem digital de vizualizare prin intermediul VGA folosind comunicarea prin interfața serială UART.

Obiectivele propuse sunt:

- proiectarea și implementarea unor module standardizate, reconfigurabile și reutilizabile pentru protocoalele UART și VGA;
- proiectarea și implementarea unui mediu de verificare modular, ușor de întreținut și modificat, cu care să se verifice și valideze atât sistemul propus, cât și modulele componente.

1.4. STRUCTURA LUCRĂRII

1. INTRODUCERE – prezentarea scopului, obiectivelor și a domeniului
2. STUDIU BIBLIOGRAFIC – prezentarea cercetării actuale în domeniu
3. FUNDAMENTAREA TEORETICĂ – prezentarea principiilor și standardelor folosite
4. SOLUȚIA PROPUSĂ – proiectarea sistemului digital
5. IMPLEMENTARE – realizarea sistemului de verificare
6. REZULTATE EXPERIMENTALE – rezultatele simulărilor
7. CONCLUZII – concluziile finale și direcțiile de dezvoltare
8. BIBLIOGRAFIE – sursele bibliografice

2. STUDIU BIBLIOGRAFIC

Proiectarea digitală este un domeniu bine definit, care a început de la dezvoltarea tehnologiei CMOS. Trendurile actuale în domeniu se axează pe “utilizarea tehnicilor de inteligență artificială pentru automatizarea proiectării [1]”. Una dintre preocupările din domeniu este reprezentată de “lipsa de interoperabilitate și comparabilitate a cercetării bazate pe inteligența artificială în proiectarea circuitelor digitale [1]”.

Există numeroase cercetări care descriu diferite moduri de implementare a standardelor UART și VGA. Majoritatea lucrărilor se axează pe folosirea standardelor preconfigurate, fără a avea posibilitatea de a fi reconfigurate în timpul folosirii.

Standardul UART este folosit în ”modulele digitale care nu necesită o viteză rapidă de comunicare, precum modulele SIM, Bluetooth, GPS [2]”. ”Scopul principal al protocolului UART este de a oferi rezultate consistente și de înaltă calitate [3]”.

Într-o cercetare recentă ”arhitectura transmițătorului UART are un generator de baud rate, un generator de paritate, un FSM și un registru cu intrare paralelă și ieșire serială (PISO) [4]”. Pe de altă parte, ”receptorul UART are un generator de baud rate, un detector de margine negativă, un verificator de paritate, un FSM și un registru cu intrare serială și ieșire paralelă (SIPO) [4]”. După cum am menționat mai devreme, structura unui frame de UART este predefinită și nu poate fi schimbată, deoarece acest lucru ar necesita refacerea arhitecturii.

Într-o altă cercetare se dorește negarea dezavantajului folosirii mai multor ”periferice cu viteză redusă este reducerea eficienței magistralelor de date și a performanței procesoarelor [2]”. O direcție de dezvoltare abordată este crearea unui ”design multi canal UART care utilizează standardul APB eficient [2]”.

Datorită dezvoltării tehnologie, VGA nu mai este standardul pentru dispozitivele moderne. Acesta a fost înlocuit de HDMI, care acceptă rezoluții și viteze mai mari decât VGA. Dar asta nu înseamnă că standardul VGA nu mai este folosit. Datorită simplității și modularității, standardul este încă preferat în sistemele integrate și pentru aplicațiile care nu necesită rezoluții sau viteze mari.

Scopul unei cercetări recente este ”distribuirea datelor VGA pe un afișaj mai mare în format de segment [5]”. Această tehnologie ar fi rentabilă și ar face compatibile ecranele mari cu standardul VGA.

3. FUNDAMENTARE TEORETICĂ

3.1. PROIECTAREA UNUI DESIGN DIGITAL

Realizarea unui proiect presupune respectarea pașilor de proiectare în paralel cu verificarea acestuia. Astfel, se vor respecta pașii descriși în următorul tabel:

Tabelul 1. Pașii de realizare a unui design digital

Design	Verificare
Definirea cerințelor: În această etapă se stabilesc cerințele sistemului.	Verificarea proiectului: În această etapă se verifică funcționarea întreg sistemului pe placă.
Definirea specificațiilor: În această etapă se definesc specificațiile sistemului.	Verificarea funcțională: În această etapă se realizează verificarea funcțională a întregului proiect. Se urmărește ca designul să fie conform cerințelor.
Proiectarea topului: În această etapă se realizează designul la nivel de top. Modulele principale, independente sunt conectate la nivel înalt. Acest design oferă o vedere abstractă a întregului sistem.	Verificarea sistemului: În această etapă se realizează verificarea topului. Se urmărește atingerea valorilor de prag pentru coverage pentru a putea valida sistemul.
Proiectarea modulelor: În această etapă se realizează designul pentru fiecare modul principal. Aceste module sunt de obicei compuse din mai multe unități conectate între ele.	Verificarea integrării: În această etapă se realizează verificarea modulelor. Aceasta este prima etapă în care folosirea unui mediu de verificare este crucială, pentru că oferă o acoperire mult mai vastă a cazurilor de test posibile.
Proiectarea unităților: În această etapă se realizarea designul pentru cele mai mici unități (ex. counter, flip-flop, FSM). Aceste unități sunt de obicei testate rudimentar din cauza simplității lor.	Verificarea unităților: În această etapă se realizează verificarea fiecărei unități în parte. Această verificare se face de obicei folosind teste simple, fără randomizare, coverage sau scoreboard.

Proiectarea și verificarea trebuie să funcționeze în paralel pentru a economisi resurse, mai ales timp. Se dorește ca orice bug să fie descoperit cât mai repede în realizarea proiectului, pentru a evita situațiile neplăcute în care designul trebuie să fie refăcut complet.

3.2. DESIGN

În ceea ce privește designul, acesta este trebuie să fie specific pentru o anumită aplicație, dar asta nu înseamnă că și componentele sale trebuie să fie la fel. Componentele generale, care respectă o structură acceptată universal, precum anumite protocoale (ex. VGA, EtherNet), trebuie să fie independente și reutilizabile.

Există anumite principii care trebuie respectate când se realizează un design:

- Refolosirea: Aceasta permite economisirea timpului necesar reimplementării acelorași protocoale de fiecare dată.
- Independența: Componentele generale nu trebuie să depindă de nicio altă componentă.
- Redundanța: Părțile critice ale sistemului trebuie să fie dublate sau realizate în mai multe moduri pentru a se asigura funcționarea în caz de defectare.
- Integrarea metodelor de detecție și observare a erorilor: Este indispensibilă adăugarea unui mod de depanare pus la dispoziție atât utilizatorilor, cât și creatorilor, pentru a face posibilă repararea sistemelor și după faza de proiectare.

Este important ca designul să fie acompaniat de formele de undă ideale, numite și golden model, care prezintă funcționalitatea presupusă a unei componente. Acest golden model este folosit pentru validarea sistemului și pentru înțelegerea mai bună a designului.

3.3. VERIFICARE

“Verificarea este un proces folosit să demonstreze că scopul unui design este păstrat în implementarea sa [6]“. În cazul designului logic se folosesc testbenchuri pentru a verifica simularea codului prin folosirea unor secvențe predefinite.

Verificarea trebuie să se realizeze în paralel cu designul pentru a avea cele mai bune rezultate. Dacă verificarea ar începe după finalizarea designului, atunci foarte multe resurse ar fi irosite pentru a detecta, găsi și soluționa diverse erori. De exemplu, pot apărea situații în care un bug este cauzat de alt bug, mai multe buguri afectează același modul, două sau mai multe buguri se elimină reciproc. Într-o astfel de situație este aproape imposibil să se valideze designul la standardele necesare.

Luând în considerare funcționalitatea care trebuie verificată, se va decide:

- strategia de verificare;
- nivelul de granularitate;
- nivelul de abstractizare:
  - nivel înalt de abstractizare: “mai puțin control asupra sincronizării și coordonării, dar mai mult timp de rulare și mai mulți stimuli [6]”;
  - nivel scăzut de abstractizare: control detaliat;
- automatizarea unot cazuri de test.

Primul pas în realizarea verificării este realizarea unui plan de verificare. Sunt identificate toate caracteristicile care trebuie verificate, în special interfețele, funcțiile și

"cazurile limită implicate de arhitectură [6]". Fiecare caracteristică trebuie să aibă o scurtă descriere, care trebuie verificată în documentul de specificații a designului.

Verificarea se realizează folosind metodologia UVM (Universal Verification Methodology). UVM „este o metodologie standardizată pentru verificare designurilor digitale și a sistemelor pe chip (SoC) în industria semiconductoarelor [7]."

UVM se folosește în industrie datorită următoarelor motive:

- Standardizarea: UVM a fost proiectat ca o metodologie standardizată, cu un standard bine definit. Acest aspect este foarte important, deoarece înainte de introducerea metodologiei UVM, fiecare companie putea folosi orice metodologie pentru verificare, ceea ce făcea ca integrarea să fie foarte dificilă.
- Flexibilitatea: UVM a fost proiectat să suporte atât TLM, cât și RTL.
- Refolosirea: UVM a fost proiectat modular, cu o separare clară între testbench și DUT. De asemenea, UVM oferă un set predefinit de clase și module pentru realizarea testbenchului.
- Mentenanța: UVM a fost proiectat să fie mai ușor de întreținut, având o separare clară între implementare și metodologie.

Componentele de bază al unui mediu de verificare sunt:

- Item: Obiectele sunt entitatea de bază a unui mediu de verificare. Acestea reprezintă stimulii care sunt aplicați unui DUT și monitorizați de la DUT. Obiectele sunt folosite în agenți, secvențe, coverage, scoreboard și interfețe, fiind singurele entități indispensabile pentru întregul mediu de verificare. Aceste obiecte sunt de cele mai multe ori randomizate într-o anumită gamă de valori.
- Sequence: Secvențele sunt compuse din unul sau mai multe obiecte create într-o ordine predefinită. Secvențele reprezintă un șir de obiecte care împreună realizează o funcție. De aceea, există secvențe specifice pentru o anumită funcționalitate, dar și secvențe generalizate.
- Agent: Agenții sunt clasa de bază care încapsulează un driver, un monitor și un sequencer. Un agent este conectat la o singură interfață virtuală, pe care trimite și primește date de la DUT.
- Driver: Driverul este o entitate activă, care are rolul de a conduce stimulii la DUT.
- Monitor: Monitorul are scopul de a colecta pasiv datele provenite de la DUT. Acesta trimite mai departe spre scoreboard și coverage datele relevante. Monitorul este o componentă pasivă.
- Coverage: Coverageul este o metrică definită de designer, care spune cât la sută din design a fost verificat funcțional. În cazul coverageului se definește și un prag minim de care ar trebui să treacă pentru ca designul să fie considerat verificat.
- Scoreboard: Scoreboardul este componentă în care se verifică funcționarea corectă a sistemului.

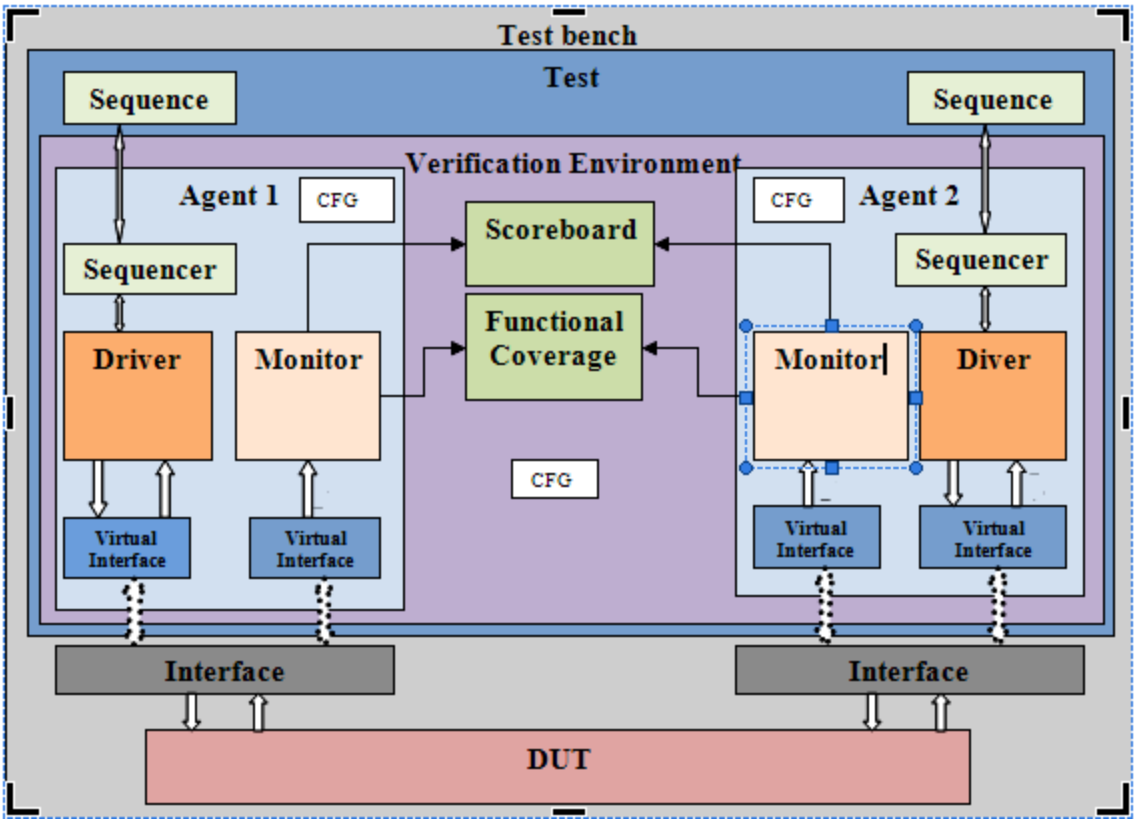

Figura 1. Componentele unui testbench conform UVM (sursa: Vitankar, Pankaj & Kureshi, A.K.. (2016). UVM ARCHITECTURE FOR VERIFICATION. International Journal of Electronics and Communication Engineering & Technology. 7. pp. 29-37 [8])

Interfețele nu fac parte din mediul UVM, acestea sunt doar entitatea de legătură dintre DUT și environment. Interfețele sunt entități simple, care au un rol bine definit de a despacheta și a împacheta datele în obiecte. Interfețele sunt reutilizabile și permit abstractizarea la nivel înalt.

În mediile de testare UVM sunt folosite 3 tipuri diferite de agenți:

- pasiv: Acest tip de agent conține doar monitorul. Scopul lui este să monitorizeze ieșirile din DUT.
- activ: Acest tip de agent conține monitorul, driverul și sequencerul. Secvențele sunt create și începute din test. Scopul acestui agent este să conducă stimulii spre DUT.
- reactiv: Acest tip de agent conține monitorul, driverul și sequencerul. În acest tip de agent, obiectele monitorizate sunt trimise înapoi spre o secvență virtuală, care declanșează o altă secvență pe același sau pe alt agent.

Fiecare proiect de verificare trebuie să aibă un coverage și un modul de scoreboard / aserții pentru a se asigura că majoritatea cazurilor de test au fost acoperite și că designul funcționează conform cerințelor.

O clasă de coverage conține unul sau mai multe covergroupuri bine definite, care la rândul lor conțin numeroase coverpointuri și crossuri dintre aceste puncte. De asemenea, un caz care trebuie mereu acoperit este cazul biților de toggle. Acest caz este de obicei realizat direct din interfața grafică a programului folosit pentru verificare, în cazul

de față QuestaSim. Verifcarea bazată pe coverage este mai eficientă, deoarece cazurile limită care mai trebuie verificate sunt mult mai ușor de identificat.

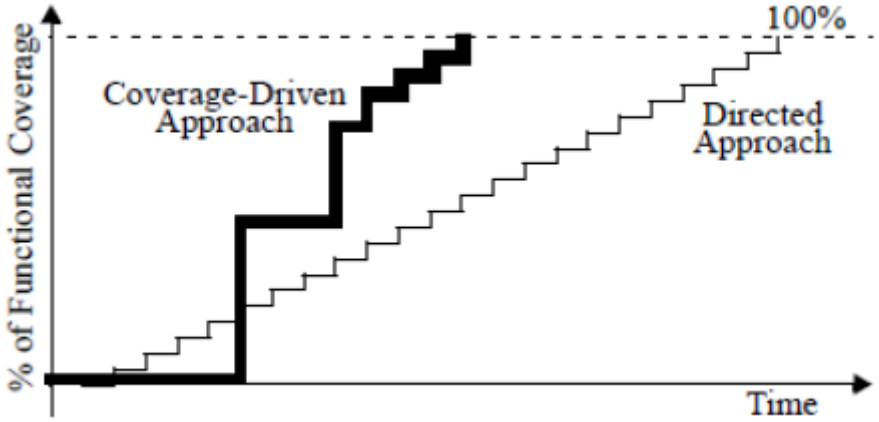

Figura 2. Progresul unui testbench bazat pe coverage (sursa: Janick Bergeron, “Writing Testbenches Using SystemVerilog”, ISBN-10: 0-387-29221-7 [12])

Funcționalitatea proiectului este verificată cu un scoreboard sau un modul de aserții. În proiectele vaste se implementează ambele moduri de verificare a funcționalității, deoarece unele funcțiuni pot fi verificate mai simplu cu aserții sau invers.

Un scoreboard primește modelul ideal al proiectului, realizat de designeri, și îl compară cu obiectele primite de monitoarele agenților. De multe ori funcționalitatea designului trebuie reimplementată în modulul de scoreboard.

Un modul de aserții verifică, în special, sincronizarea semnalelor. Există 2 tipuri de aserții, imediate și concurente. Aserțiile imediate sunt folosite în puține cazuri, cel mai prevalent caz fiind verificarea randomizării obiectelor. “Aserțiile concurente sunt cele mai valoroase tipuri de aserții [10]”. Acestea sunt „monitoare aflate într-un bloc de cod care probează periodic și testează semnalele și generează mesaje de eroare dacă aserția eșuează [10]”.

Un aspect foarte important al metodologiei UVM este reprezentat de existența fazelor de pre rulare, post rulare și în timpul rulării.

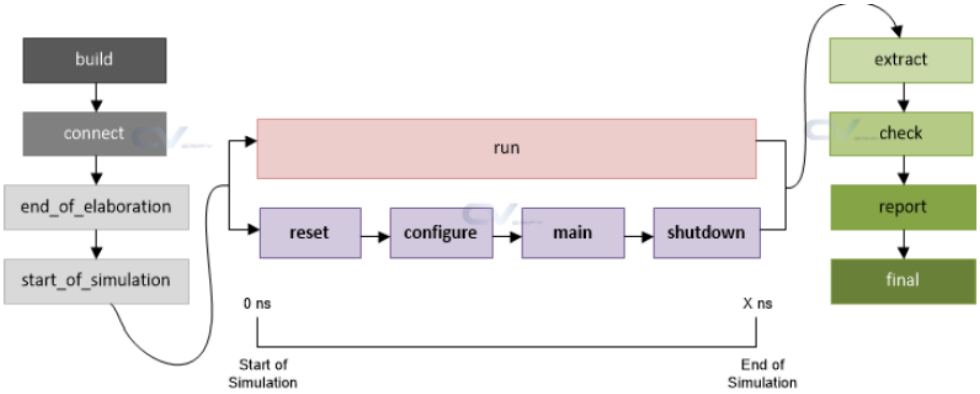

Figura 3. Fazele UVM (sursa: https://www.chipverify.com/uvm/uvm-phases [11])

O componentă are toate fazele implementate, dar programatorul alege dacă acestea trebuie suprascrise sau completate. Acest aspect duce la crearea unor teste individuale, dar standardizate.

Tabelul 2. Fazele UVM

Fază UVM	Descriere
build	Se creează structura test benchului. - instanțierea tuturor componentelor; - instanțierea modelului de registru; -scrierea și preluarea configurațiilor din baza de date; La final toate componentele au fost instanțiate.
connect	Se conectează compenentele prin TLM. - conectarea porturile TLM; La final toate porturile componentelor au fost conectate.
end of elaboration	Se finalizează testbenchul. - afișarea topologiei; - deschiderea fișierelor; - ”definirea configurațiilor suplimentare pentru componente [12]”;
start of simulation	Se pregătește rularea testelor. - afișarea topologiei; - setarea breakpointurilor pentru debugger; - setarea configurațiilor pentru faza următoare;
run	Se rulează testele. Aceasta este singura fază care consumă timp. - toate ”componentele implementează comportamentul necesar pe toată durata fazei, în toate fazele de run_time [12] ”. Această fază se poate termina în 2 feluri: ▪ toate obiecțiile sunt coborâte; ▪ expiră timpul de rulare (”Timpul de timeout default este 9200s [12]”).
extract	Se extrag datele din componente. - extrage orice date rămase din scoreboard și alte componente; - verifică DUT-ul pentru informațiile legate de starea finală și le afișează; - calculează statisticile și realizează rezumatul; - închide toate fișierele; La final toate datele legate de rulare au fost strânse.
check	Se verifică funcționarea sistemului. - verifică ca toate datele să fie colectate; La final se știe dacă testul a trecut sau nu.
report	Se raportează rezultatele. - raportarea rezultatelor; - scrierea rezultatelor în transcript;
final	Se finalizează testbenchul. - închiderea tuturor fișierelor. La final se iese din simulator.

Implementarea unui mod de prioritizare a mesajelor a îmbunătățit vizibil lizibilitatea verificării. În UVM există 6 nivele de verbozitate care definesc importanța unui mesaj.

Tabelul 3. Nivelele de verbozitate în UVM

Verbozitate	Valoare	Descriere
UVM_NONE	0	Este folosit pentru toate mesajele critice.
UVM_LOW	100	Este folosit pentru mesajele rare, care se întâmplă o singură dată pe durata rulării testului.
UVM_MEDIUM	200	Este folosit pentru mesajele din secvențe, care se întâmplă o dată pe rulare.
UVM_HIGH	300	Este folosit pentru afișarea detaliilor importante din rulare.
UVM_FULL	400	Este folosit pentru afișarea tuturor detaliilor din rulare.
UVM_DEBUG	500	Este folosit pentu depanarea problemelor.

3.4. UART

Protocolul de comunicare serială asincronă UART este una dintre cele mai comune protocoale de comunicare între dispozitive. Acesta are 2 linii, pentru RX și TX, pe care sunt puse date serial, una după alta. Una dintre diferențele majore dintre UART și alte protocoale de comunicare este că acesta nu are un semnal de tact. Se bazează pe comunicare asincronă și folosește semnale de tact interne pentru a reface semnalul.

Un frame de UART este format din 4 categorii de biți:

- Start: Bitul de start simbolizează începutul unei tranzacții și este întotdeauna 0.
- Data: Pot exista 5-9 biți de date într-un frame.
- Parity: Bitul de paritate este opțional și poate fi par, impar sau inexistent. Acesta este bitul de verificare, care reprezintă numărul de intrări primite pe 1 sau 0 logic.
- Stop: Pot exista 1, 1.5 sau 2 biți de stop. Aceștia simbolizează sfârșitul unei tranzacții și este întotdeauna 1.

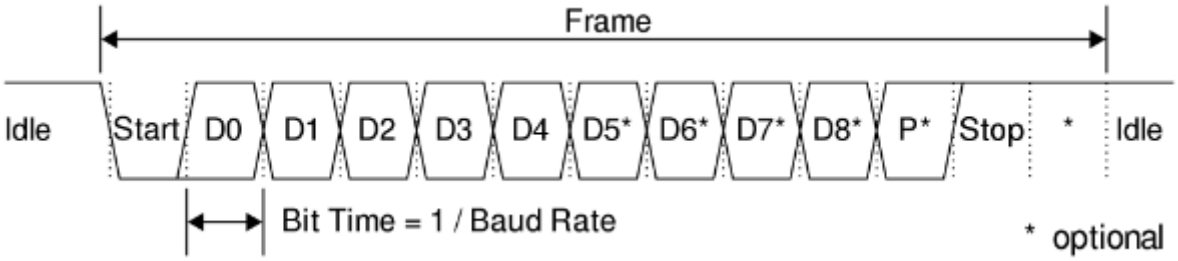

Figura 4. Structura unui frame UART (sursa: https://digilent.com/blog/uart-explained/ [13])

Un frame UART este mereu încadrat între biți de idle, de cele mai multe ori aceștia fiind 1 logic. Un parametru important pentru UART este numit baud rate, care reprezintă frecvența unui bit. Dispozitivele care comunică între ele trebuie să aibă același baud rate și structură a frameului.

3.5. VGA

„VGA (Video Graphics Array) este o serie de standarde vizuale prezentate în perioada anilor 1980 de IBM în computerele lor personale și este susținută pe scară largă în echipamentele de ilustrații și ecrane [14].”

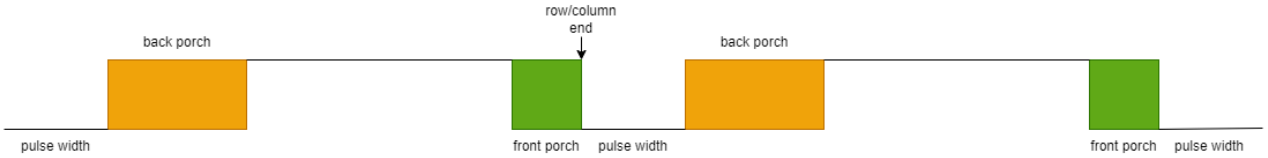

Figura 5. Explicație SYNC

Protocolul VGA se bazează pe trimiterea datelor de culoare pentru fiecare pixel în funcție de semnalele de sincronizare numite frecvent HSYNC și VSYNC.

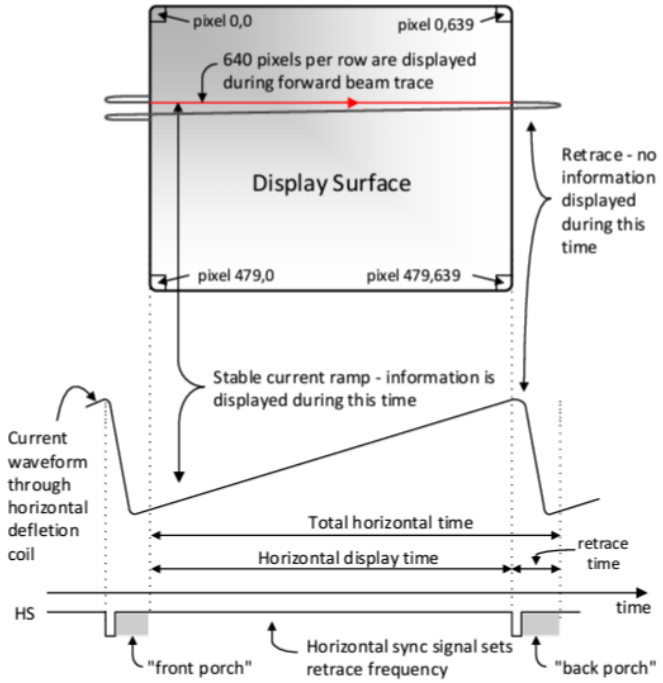

Figura 6. Detalii ecran (sursa: Nexys-4 FPGA: Technical report, September 2014 [15])

Datele indispensabile de care are nevoie standardul VGA pentru a funcționa sunt cele 3 culori de bază RGB (roșu, verde și albastru), fiecare pe 4 biți și semnalele de control VSYNC și HSYNC. Pinii rămași pot fi folosiți pentru a detecta tipul monitorului. Cei 12 biți de culoare permit afișarea a 4096 de culori diferite.

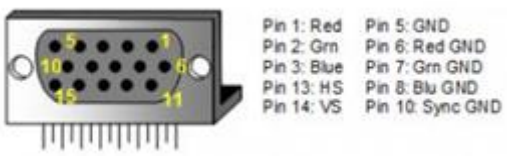

Figura 7. Conector VGA (sursa: Nexys A7™ FPGA Board Reference Manual, Octombrie 2019 [16])

4. SOLUȚIA PROPUSĂ

4.1. DEFINIREA CERINȚELOR

Acest proiect propune realizarea unui sistem de afișare a culorilor pe un ecran prin VGA, însoțit moduri de verificare și depanare a modulelor de comunicare UART și ale ecranelor cu port VGA.

4.2. DEFINIREA SPECIFICAȚIILOR

Utilizatorul aplicației are mai multe moduri de a interacționa cu sistemul:

1. Utilizatorul folosește comutatoarele de pe placă pentru a schimba forma afișajului de pe ecran.
2. Utiliizatorul folosește comutatoarele de pe placă pentru a trece la modurile de depanare pentru standarde.
3. Utilizatorul trimite comenzi pentru a schimba culorile de pe ecran.
4. Utilizatorul trimite comenzi pentru a schimba forma afișajului de pe ecran.

Rezultatul sistemului în urma interacțiunilor:

1. Pe LED-uri se vor afișa:
    a. erori (UART, sincronizare, configurare, afișare);
    b. configurația actuală;
    c. frameuri de UART;
2. Pe ecran se vor afișa culori în diferite configurații.

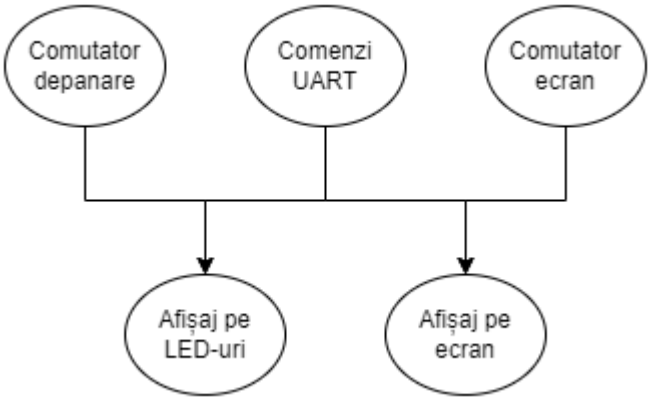

Figura 8. Diagrama simplificată a parcurgerii sistemului

4.3. PROIECTAREA TOPULUI

Sistemul este compus din 6 module principale, independente și modulare. În cazul de față topul se va numi Color Show (CS).

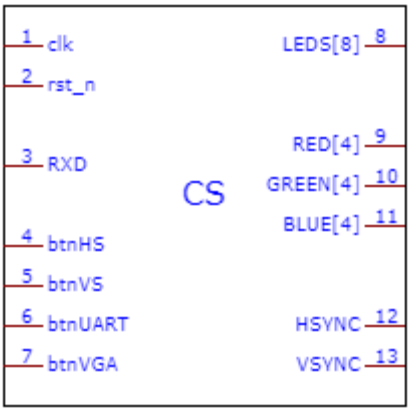

Figura 9. Componenta Top Module

Pe lângă modulele principale (clock divider, UART, VGA, color manager, led manager și debouncer) se mai pot observa 4 instanțe ale sincronizatoarelor.

La implementarea pe placă a designului au fost folosite PLL-uri pentru a ajunge la semnalul de tact dorit pentru VGA și UART.

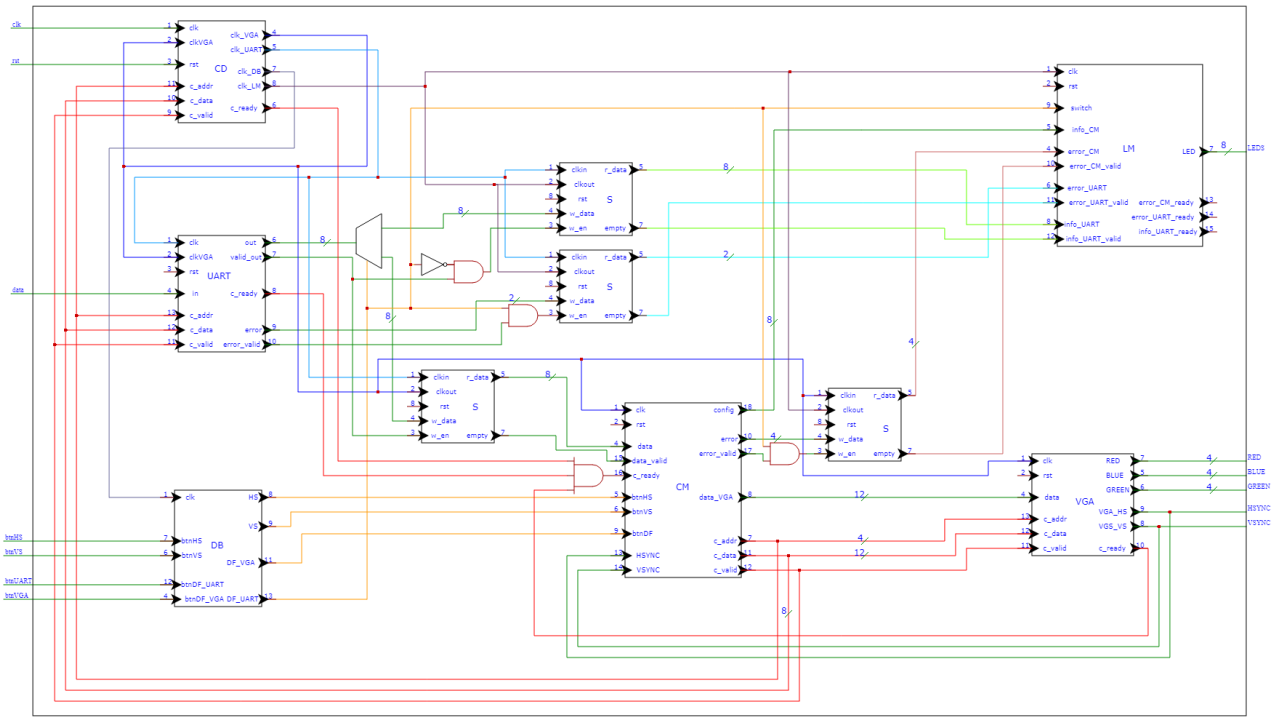

Figura 10. Design Top Module

Pinii din tabelul următor corespund pinilor plăcii Zedboard.

Tabelul 5. Intrările/Ieșirile sistemului

Nume	Tip	Mărime [bit]	Descriere	Pin
clk	input	1	Valoarea semnalului de tact al plăcii.	Y9
rst	input	1	Semnalul de reset general.	H19
btnHS	input	1	Valoarea butonului pentru împărțirea pe orizontală.	F22
btnVS	input	1	Valoarea butonului pentru împărțirea pe verticală.	G22
btnUART	input	1	Valoare butonului pentru activarea modului debug UART.	H22
btnVGA	input	1	Valoare butonului pentru activarea modului debug VGA.	F21
data	input	1	Data primită pe UART.	C14
RED	output	4	Canalul culorii roșu pe ieșirea de VGA.	V20 U20 V19 V18
GREEN	output	4	Canalul culorii verde pe ieșirea de VGA.	AB22 AA22 AB21 AA21
BLUE	output	4	Canalul culorii albastru pe ieșirea de VGA.	Y21, Y20, AB20, AB19
HSYNC	output	1	Semanlul de sincronizare pe orizontală pentru VGA.	AA19
VSYNC	output	1	Semnalul de sincronizare pe verticală pentru VGA.	Y19
leds	output	8	Valorile celor 8 leduri.	T22 T21 U22 U21 V22 W22 U19 U14

Această proiectare a fost aleasă în urma observării plăcii Zedboard / MachXO3D. Au existat alte 3 variante posibile de design care au fost luate în considerare.

În prima variantă de design au fost utilizate 16 LED-uri pentru a depana mai ușor codul, deoarece mai multe informații puteau fi afișate constant.

O altă variantă precedentă conținea un modul HDMI pe lângă modulul VGA pentru a asigura redundanța în cazul defectării sau inexistenței unui port.

Ultima variantă a fost implementată în acest proiect prin adăugarea modulului de debounce pentru butoane. Astfel, în designul final pot fi conectate atât comutatoare, cât și butoane.

4.4. PROIECTAREA MODULELOR

4.4.1. CLOCK DIVIDER (CD)

Tabelul 6. Intrările/Ieșirile modulului Clock Divider

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	c_ready	1
rst	1	clk_VGA	1
c_addr	4	clk_UART	1
c_data	8	clk_DB	1
c_valid	1	clk_LM	1
clkinVGA	1		

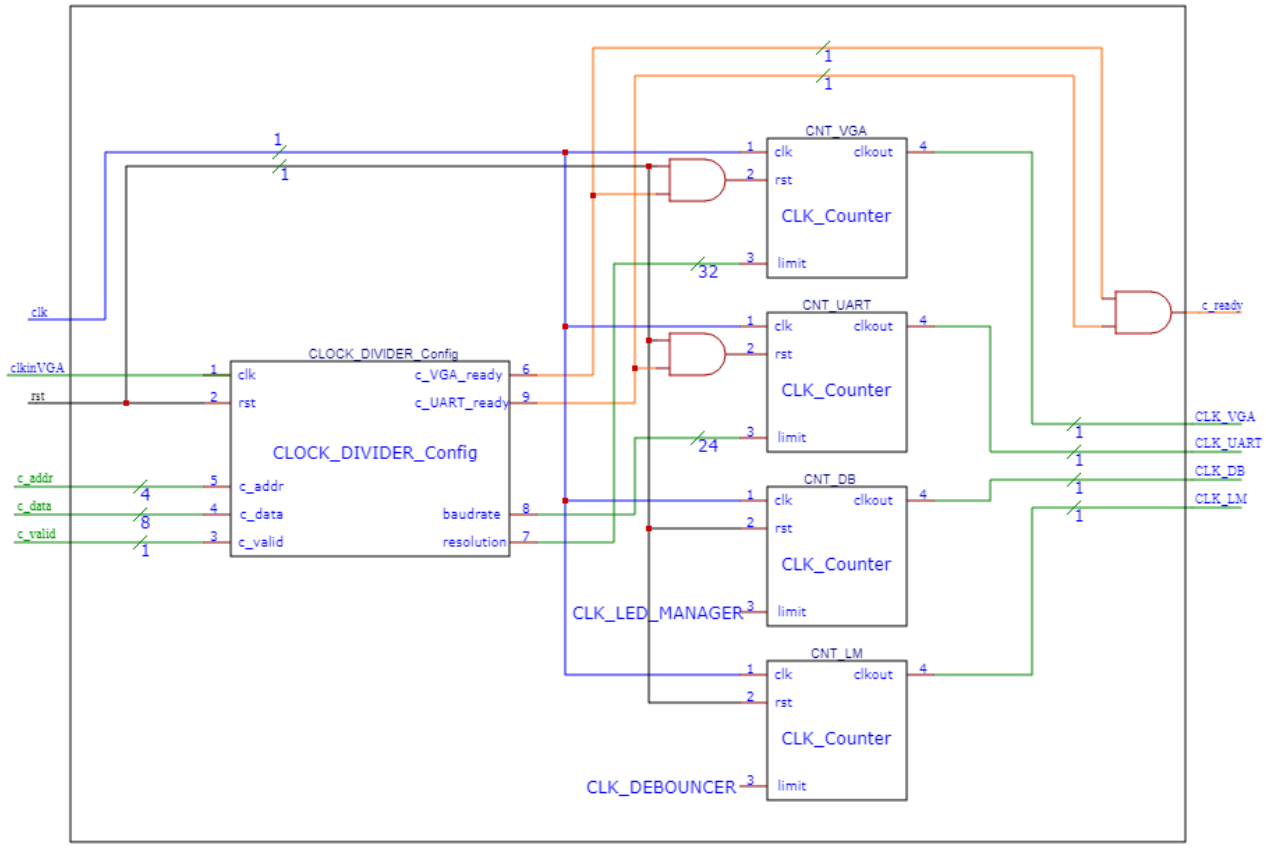

Figura 11. Modulul Clock Divider

Modulul Clock Divider generează impulsuri cu diferite frecvențe în funcție de configurația modulelor VGA și UART și de frecvența de intrare. Acest modul este specific pentru aplicație și nu poate fi refolosit. Singurele unități reutilizabile sunt numărătoarele reconfigurabile.

Pentru a se modifica limitele semnalul c_valid trebuie să fie activ, iar după modificarea configurației, semnalul c_ready va fi reactivat. Pentru oricare altă adresă limitele numărătoarelor nu se vor modifica și numărătoarele nu vor fi resetate.

În următoarea diagramă de semnale se pot observa 4 modificări successive ale configurației modulelor UART și VGA:

- BAUDRATE: duce la resetarea semnalului clock_UART și la modificarea limitei numărătorului CNT_UART;
- PARITY BIT: nu apar modificări, deoarece doar baudrate-ul si rezoluția influențează semnalele de clock pentru modulele UART și VGA;
- Rezoluție: duce la resetarea semnalului clock_UART, modificarea limitei numărătorului CNT_UART, resetarea semnalului clock_VGA și la modificarea limitei numărătorului CNT_VGA.

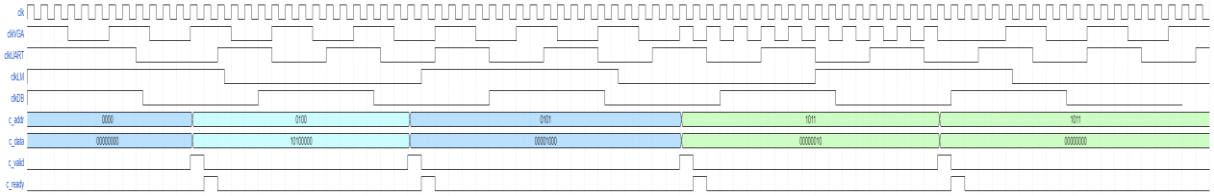

Figura 12. Golden model pentru reconfigurare

Modulul este format dintr-o unitate de configurare și 4 numărătoare, unul pentru fiecare frecvență de ieșire:

- CNT_VGA: limitele conform tabelului;
- CNT_UART: limitele conform tabelului;
- CNT_DB: limita este CLK_DEBOUNCER;
- CNT_LM: limita este CLK_LED_MANAGER.

Ieșirile:

✓ clkVGA: frecvența: 25Mhz – 65MHz
  - în funcție de rezoluția modulului VGA, frecvența este:
    - 640x480 – 25MHz
    - 800x600 – 40MHz
    - 1024x768 – 65MHz
  - frecvența inițială este cea corespunzatoare rezoluției de 640x480, adică 25MHz.

✓ clkUART: frecvența: 38.4kHz – 921.6kHz
  - în funcție de baudrate din modulul UART, frecvența este:
    - 2400 –38.4kHz
    - 4800 – 76.8kHz
    - 9600 – 153.6kHz
    - 19200 – 307.2kHz
    - 57600 – 921.6kHz
    - 11200 – 1843.2kHz
  - frecvența initială este cea corespunzatoare unui baudrate de 9600, adică 9.6kHz

✓ clkCM: frecvența: 25MHz – 65MHz
  - în funcție de rezoluția modulului VGA, specificată la punctul anterior

✓ clkDEB: frecvența 20Hz

✓ clkLED: frecvența 1HZ

4.4.2. DEBOUNCERS (DB)

Tabelul 7. Intrările/Ieșirile modulului Debouncers

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	HS	1
rst	1	VS	1
btnHS	1	DF_UART	1
btnVS	1	DF_VGA	1
btnDF_UART	1		
btnDF_VGA	1		

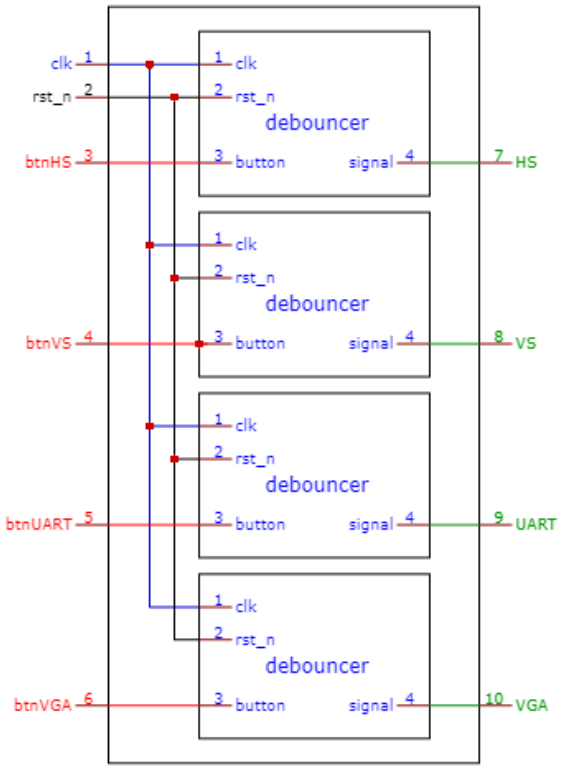

Figura 13. Modulul Debouncers

Modulul Debouncers este format din 4 module debouncer independente care primesc ca parametru limita minim acceptată pentru a valida un semnal.

4.4.3. UART

Tabelul 8. Intrările/Ieșirile modulului UART

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	c_ready	1
rst	1	error	2
clkVGA	1	valid_error	1
in	1	out	8
c_addr	4	valid_out	1
c_data	8		
c_valid	1		

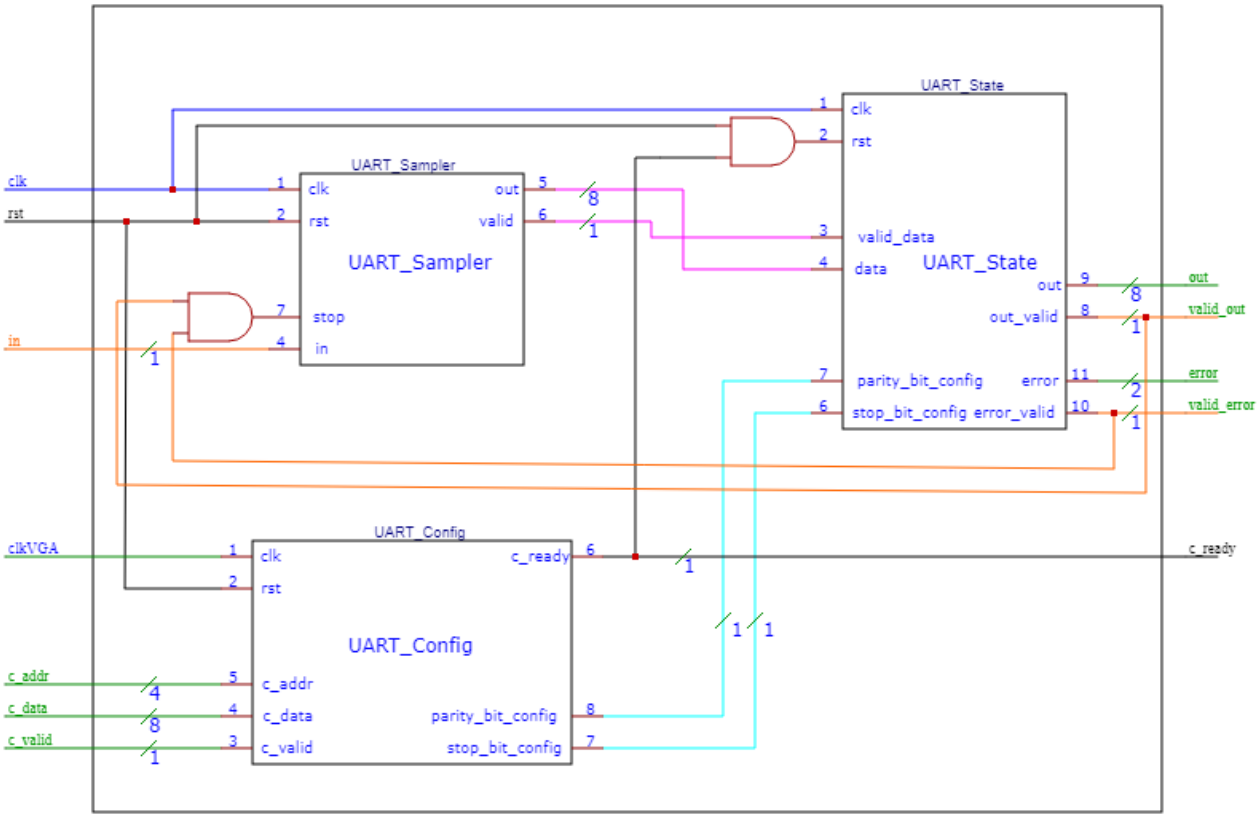

Figura 14. Modulul UART

Modulul configurabil UART are în strucutra sa registrii în care sunt salvate valorile configurabile și este inițializat cu BAUDRATE 9600, PARITY BIT none, 1 STOP BIT, 8 DATA BITS și cu frecvența de 153.6kHz.

Parametrii BAUDRATE, PARITY BIT și STOP BITS sunt configurabili, dar DATA BITS nu se poate modifica. Parametrul BAUDRATE influențează modulul CD, iar parametrii PARITYBIT și STOPBITS influențează modulul UART.

Modificarea configurației:

- c_valid trebuie să fie activ pentru a semnaliza primirea unei noi configurații;
- c_addr si c_data, formate din 4 și 8 biți sunt codificate conform următorului tabel;
- c_ready este reactivat după reconfigurare;
- după configurare se resetează valorile;
- orice transmisie în curs de efecuare este întreruptă.

Tabelul 9. Codificarea parametrilor configurabili ai modulului UART

Adresă	Cod	Date	Cod
UART_PARITY_ADDR	0101	PARITYBIT_NONE	00
		PARITYBIT_ODD	11
		PARITYBIT_EVEN	10
UART_STOP_ADDR	0110	STOPBITS_1	x0
		STOPBITS_2	x1

În următoarele diagrame se poate observa o comunicare reușită și o comunicare nereușită datorită nerespectării protocolului UART pentru bitul de STOP.

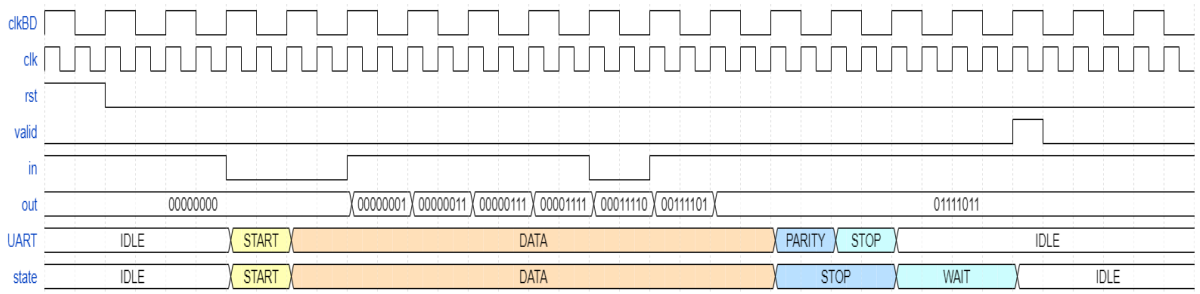

Figura 15. Golden model pentru o comunicare validă prin UART

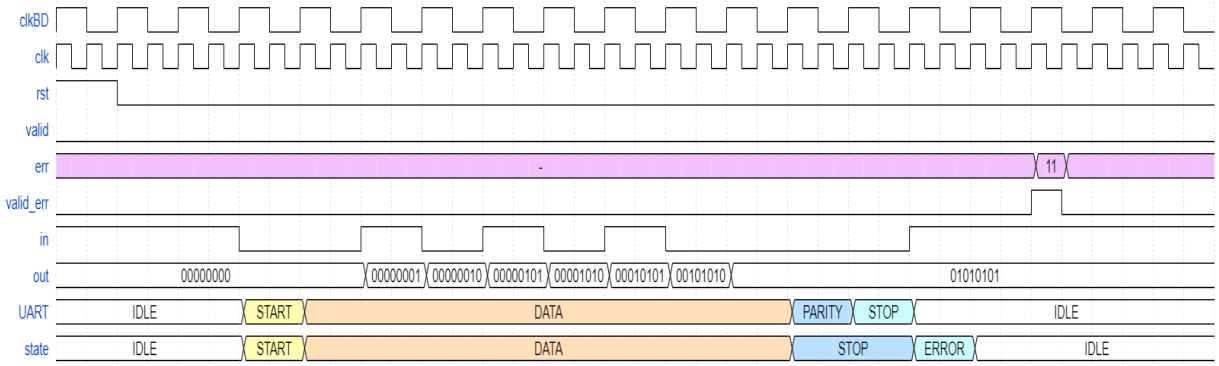

Figura 16. Golden model pentru o comunicare invalidă prin UART

Erorile trimise de modulul UART au următoarea codificare:

- 00 – eroare bit de STOP;
- 01 – eroare bit de PARITY;
- 10 – eroare bit de IDLE.

4.4.4. COLOR MANAGER (CM)

Tabelul 10. Intrările/Ieșirile modulului Color Manager

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	C_addr	5
rst	1	C_Data	15
RxD_Data	8	C_Valid	1
Empty	1	CM_Err	4
C_ready	1	Valid_Err	4
Vertical_Split	1	VGA_Not	4
Horizontal_Split	1	Valid_VGA_Not	4
VGA_Debugg	1	Data_VGA	12
HSync	1		
VSync	1		

Modulul Color Manager primește ca intrare comenzile date de utilizator și transmite mai departe culorile care trebuie afișate pe ecran, erori, notificări și configurații noi.

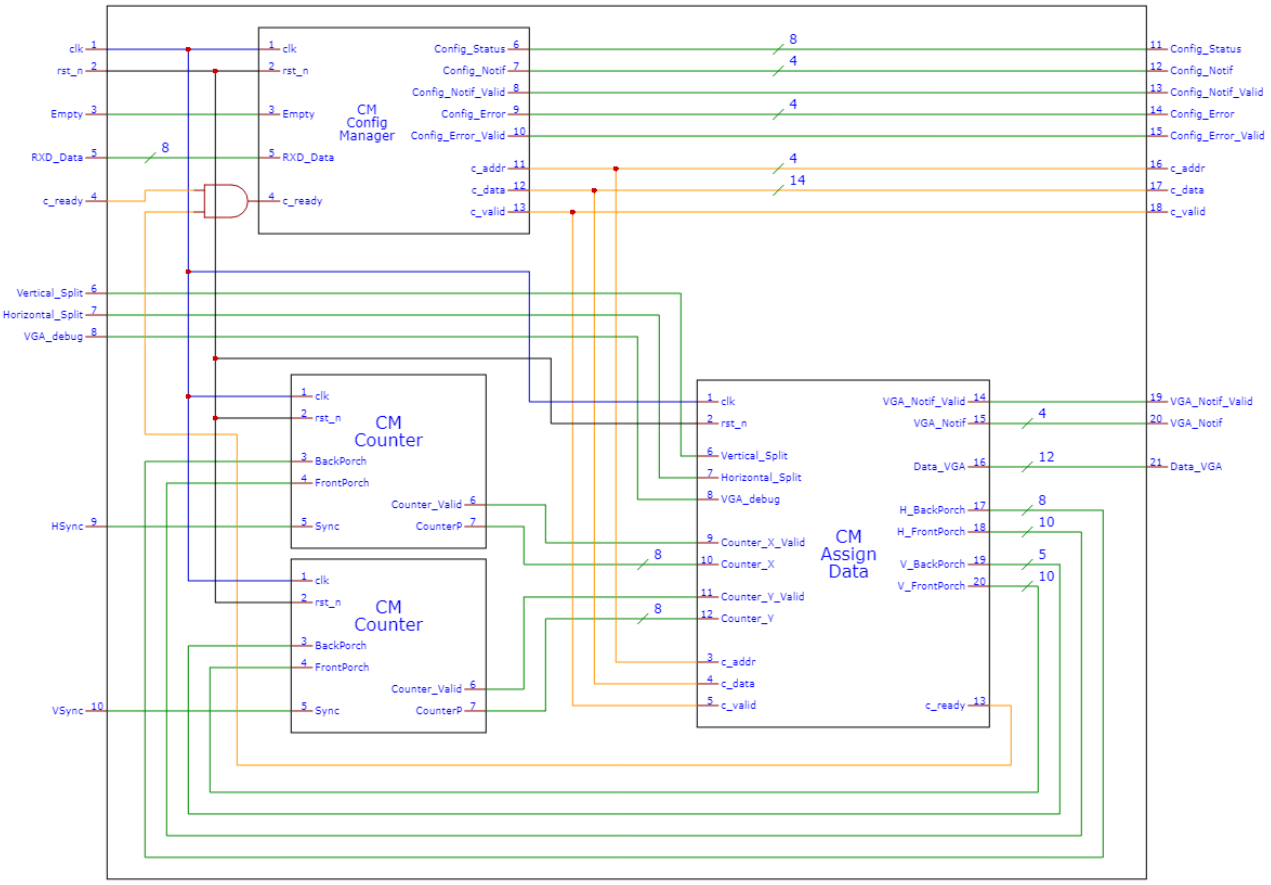

Figura 17. Modulul Color Manager

Tabelul 11. Descrierea cuvântului de comandă

Poziție bit	Descriere configurație	Cod	Descriere culoare	Cod
15	Selecție configurație / culoare	1	Selecție configurație / culoare	0
14	Adresa unității configurabile	01/10	-	0
13			Poziția cadranului: sus/jos	0/1
12	Adresa registrului configurabil	x	Poziția cadranului: stânga/dreapta	0/1
11		x	Codul culorii pe 12 biți	x
10	Codificarea configurației	x		x
9		x		x
8		x		x
7	-	0		x
6		0		x
5		0		x
4		0		x
3		0		x
2		0		x
1		0		x
0		0		x

Tabelul 12. Codificarea modulelor configurabile

Unitate	Valoare		Descriere
UART	0	0	Configurare registru BaudRate
	0	1	Configurare registru bit de paritate
	1	0	Configurare registru bit de stop
VGA	0	0	Culoare nouă pentru VGA prin UART
	1	0	Configurare cadran nou VGA
	1	0	Configurare registru rezoluție VGA

Tabelul 13. Codificarea configurărilor

Descriere	Valori			Codificare
Configurare registru BaudRate	0	0	0	2400
	0	0	1	4800
	0	1	0	9600
	0	1	1	19200
	1	0	0	57600
	1	0	1	112000
Configurare registru bit de paritate	x	0	0	fără paritate
	x	1	1	impară
	x	1	0	pară
Configurare registru bit de stop	x	x	0	1 bit de stop
	x	x	1	2 biți de stop
Configurare registru rezoluție VGA	x	0	0	640x480
	x	0	1	800x600
	x	1	0	1024x768
Configurare cadran nou VGA	x	0	1	VS
	x	1	0	HS
	x	1	1	VS & HS
	* comutatoarele VS și HS trebuie să fie inactive			

4.4.5. VGA

Tabelul 14. Intrările/Ieșirile modulului VGA

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	red	4
rst	1	green	4
c_addr	2	blue	4
c_data	2	HSync	1
c_valid	1	VSync	1
data_in	12		

Modulul VGA controlează ieșirea pentru VGA, astfel încât avem cele 3 culori principale care vor forma culoarea finală, dată prin intrarea data_in.

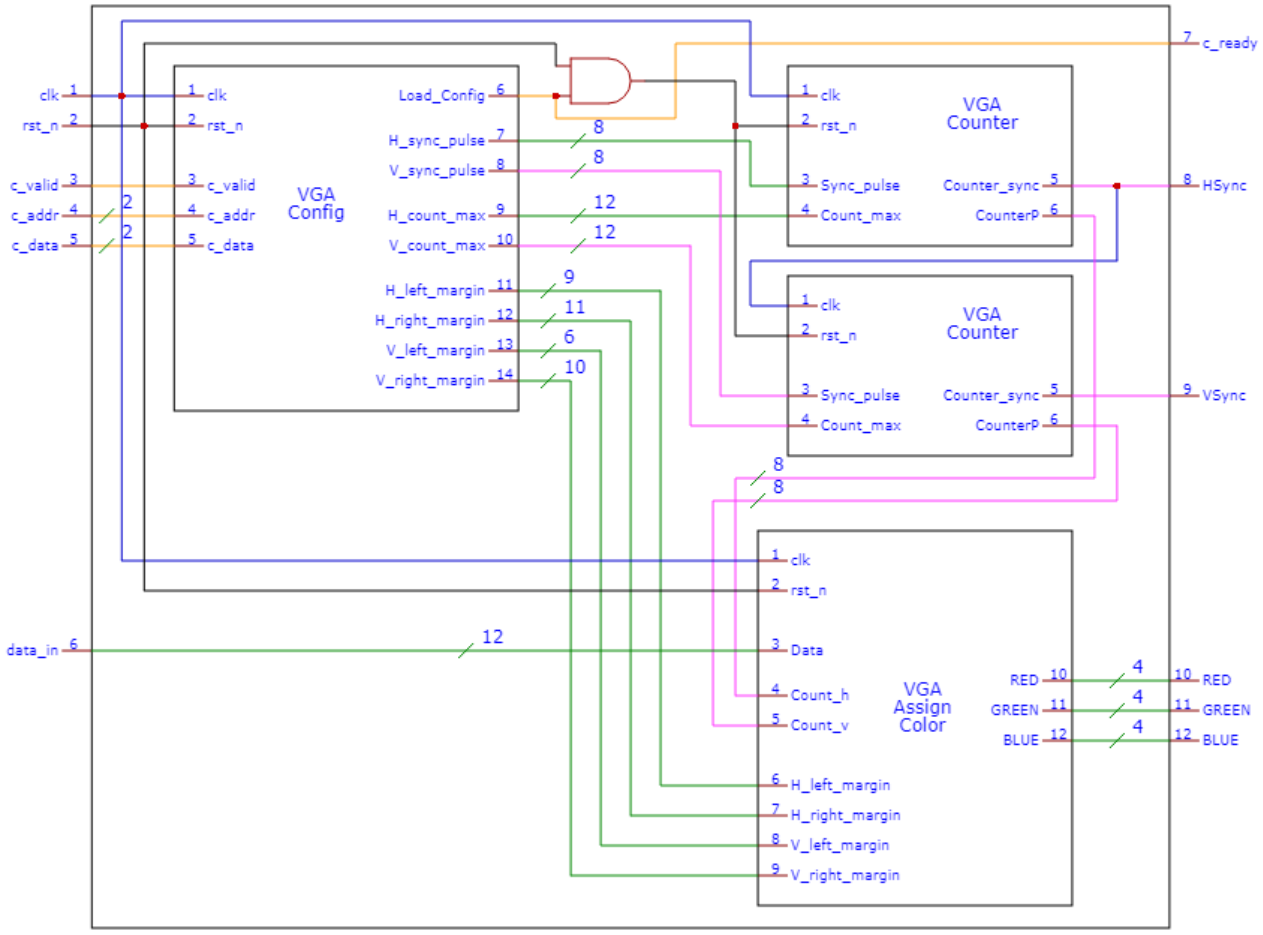

Figura 18. Modulul VGA

Se poate alege una dintre următoarele rezoluții:

- 640x480 (00 - default);
- 800x600 (01);
- 1024x768 (10).

VSync:

- 0 – moment inoportun pentru mutarea cursorului la noua coloană;
- 1 – moment oportun pentru mutarea cursorului la noua coloană.

HSync:

- 0 – moment inoportun pentru mutarea cursorului la un nou frame;
- 1 – moment oportun pentru mutarea cursorului un nou frame.

În cazului unei rezoluții noi, numărătoarele sunt resetate și limitele acestora sunt schimbate în funcție de tabelul de valori. O rezoluție nouă presupune un reset local, dar păstrarea noii configurații. Resetul local al numărătoarelor este asigurat de combinația logică dintre semnalul global de reset și semnalul intern de load config.

4.4.6. LED MANAGER (LM)

Tabelul 15. Intrările/Ieșirile modulului Led Manager

Intrări	Mărime [bit]	Ieșiri	Mărime [bit]
clk	1	leds	8
rst	1		
UART_data_debug_switch	1		
UART_data	8		
UART_errors	2		
CM_errors	4		
UART_data_valid	1		
UART_errors_valid	1		
CM_errors_valid	1		
config_notification	8		

Modulul Led Manager funcționează la o frecvența de 1Hz. Acesta primește informații de la celelalte module (actualizări, erori) și le transpune în output vizual, pe cele 8 LED-uri.

Modulul primește date de la 3 surse diferite: modulul principal CS, UART și CM. Fiecare informație activează o anumită combinație de LED-uri.

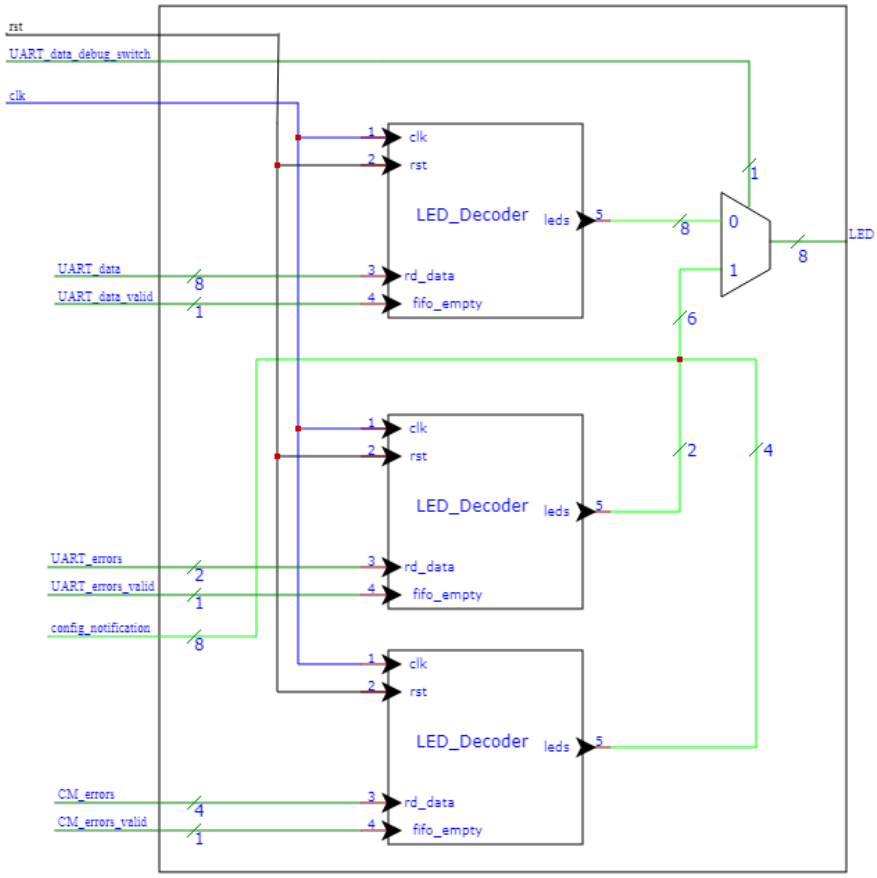

Figura 19. Modulul Led Manager

Tabelul 16. Datele provenite de la modulul UART

Eroare	Cod
NO_ERRORS	00
FAILED_STOP_BITS	11
FAILED_PARITY_BIT	01
FAILED_IDLE_BITS	10

Tabelul 17. Datele provenite de la modulul CM

Eroare	Cod
NO_ERRORS	0000
FAILED_CONFIGURATION_ADDRESS	0011
FAILED_BAUDRATE_CONFIGURATION	0100
FAILED_QUADRAN_CONFIGURATION	0101
FAILED_STOPBITS_CONFIGURATION	0110
FAILED_RESOLUTION_CONFIGURATION	0111
CORRECT_BAUDRATE_CONFIGURATION	1100
CORRECT_PARITYBIT_CONFIGURATION	1101
CORRECT_STOPBITS_CONFIGURATION	1110
CORRECT_RESOLUTION_CONFIGURATION	1111
WRONG_QUADRANT	0010
STATE0SPLIT_CHANGE	1000
STATE2VERTICAL_CHANGE	1001
STATE2HORIZONTAL_CHANGE	1010
STATE4SPLIT_CHANGE	1011

Pentru cazul în care UART_data_debug_switch nu este activ, LED-urile folosite pentru afișarea notificărilor și a erorilor se vor aprinde și vor rămâne aprinse până la primirea altor informații din aceeași categorie.

Pentru cazul în care UART_data_debug_switch este activ, LED-urile corespunzatoare frameului de UART se vor aprinde și vor rămâne aprinse până la primirea altor informații din aceeași categorie.

Tabelul 18. Formatul LED-urilor

LED	Valoare depanare	Valoare standard
U14	UART_data[7]	RESET
U19	UART_data[6]	UART_data_debug_switch
W22	UART_data[5]	EROARE UART
V22	UART_data[4]	
U21	UART_data[3]	EROARE CM
U22	UART_data[2]	
T21	UART_data[1]	
T22	UART_data[0]	

4.5. PROIECTAREA UNITĂȚILOR

4.5.1. CLOCK DIVIDER (CD)

Modulul este format din 2 unități: Clock Divider Config și Clock Divider Counter.

Modulul Clock Divider Counter este un numărător cu limita superioară variabilă.

Modulul Clock Divider Config se ocupă de reconfigurarea configurațiilor și controlează resetarea tactului. Modulul primește noile configurații, transmite configurațiile actuale ale modulelor UART și VGA spre numărătoare, pe care le și resetează.

Tabelul 19. Valorile configurabile ale clock dividerului

Adresă	Cod	Date	Cod	Limită
UART BAUDRATE ADDR	0100	BAUDRATE_2400	000	CLK_BAUDRATE 2400
		BAUDRATE_4800	001	CLK_BAUDRATE 4800
		BAUDRATE_9600	010	CLK_BAUDRATE 9600
		BAUDRATE_19200	011	CLK_BAUDRATE 19200
		BAUDRATE_57600	100	CLK_BAUDRATE 57600
		BAUDRATE_112000	101	CLK_BAUDRATE 112000
VGA RESOLUTION ADDR	1000	VGA_640x480	x00	CLK_VGA 640x480
		VGA_800x600	x01	CLK_VGA 800x600
		VGA_1024x768	x10	CLK_VGA 1024x768

4.5.2. DEBOUNCER (DB)

Frecvența modulului este 20Hz si nu este configurabil.

Modulul Debouncer primește ca intrare un semnal instabil și îl transformă într-un semnal stabil, folosind un numărător cu limită variabilă. Se așteaptă stabilizarea semnalului de intrare pentru ca semnalul de ieșire să se modifice.

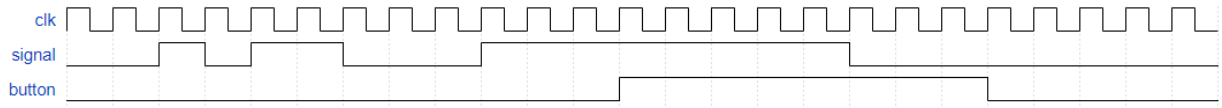

Figura 20. Golden model pentru debouncer

4.5.3. UART

Modulul UART este format din 3 module independente:

- UART_Sampler: Se ocupă de eșantionarea semnalului (un counter care numără câți biți de 0 și 1 sunt eșantionați în interiorul semnalului și un comparator care stabilește bitul de ieșire).
  - Pentru a evita situația de sincronizare greșită a modulului UART, în care este posibil să se transmită date greșite și să rămână nesincronizat, biții primiți sunt supraeșantionați.
  - Se iau câte 16 eșantioane pentru fiecare bit primit, din care primele și ultimele 3 eșantioane sunt ignorate, iar din celelalte eșantioane se calculează valoarea de ieșire.

- UART_Config: La primirea unei noi configurații modulul își actualizează registrele cu valorile parametrilor modificați și resetează modulului UART_State pentru a întrerupe orice comunicație neîncheiată.
- UART_State: Verifică dacă informația primită respectă protocolul UART.
    - Primul bit eșantionat reprezintă bitul de START, care trebuie să fie 0. În caz contrar se consideră că a fost un start fals și se reîncepe transmisia;
    - Următorii 8 biți eșantionați reprezintă biții de DATE, care pot lua valoarea 0 sau 1;
    - În cazul în care paritatea este activată, următorul bit reprezintă bitul de PARITY, care este verificat. În cazul în care paritatea nu este validă, se semnalează eroarea, se consideră că datele transmise au fost corupte și se reîncepe transmisia;
    - Următorul bit/biți reprezintă bitul/biții de STOP, care trebuie să fie pe 1. În caz contrar se semnalează eroarea, se consideră că datele transmise nu au fost eșantionate corect și se reîncepe transmisia;
    - Dacă toți biții verificați (START, PARITY, STOP) sunt valizi, atunci counicarea a avut succes.

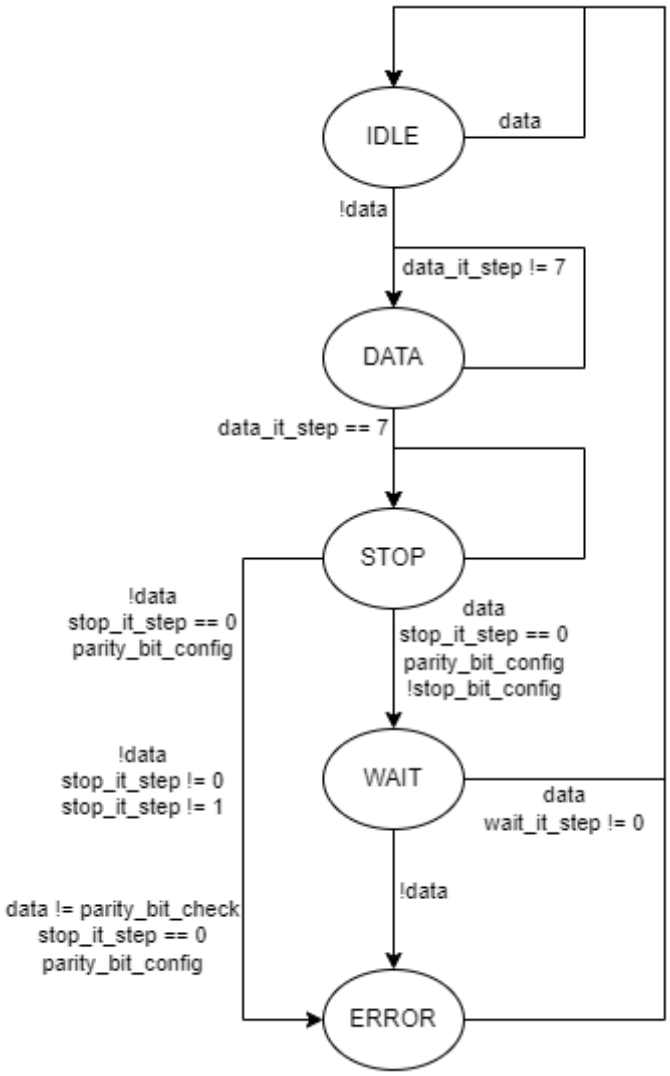

Figura 21. Diagrama de stări finite a modulului UART

4.5.4. COLOR MANAGER (CM)

Modulul Color Manager este alcătuit din 3 module: numărătoare, FSM și managerul de configurații.

La intrare acesta primește serial date de la modulul UART. O tranzacție completă este formată din 2 frameuri de UART.

Tabelul 20. Format primit de la UART

Poziție	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Valoare	1	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0

- addr_module: id unitate configurabilă, trimis prin c_addr[3:2];
- addr_reg: id registru din unitatea configurabilă (doar UART) trimis prin c_addr[1:0];
- c_data: noua configurație codificată: 00 0000 0000 0xxx.

Format primit de la UART: 00xx xxxx xxxx xxxx;

- c_addr: 10 00;
- c_data: xx xxxx xxxx xxxx.

Pentru un cuvânt de date, ultimii biți, c_data[11:0] semnifică biții de culoare.

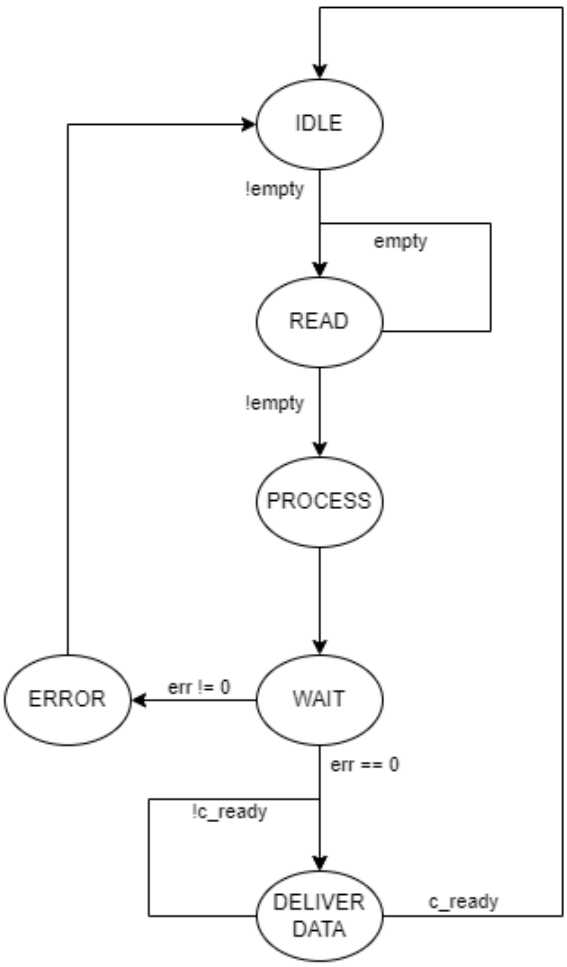

Figura 22. Diagrama de stări finite a managerului de configurații

În cazul unui cadran:

- bitul 14 arată poziția cadranului sus(0) / jos(1)
- bitul 13 arată poziția cadranului: stânga(0) / dreapta(1)

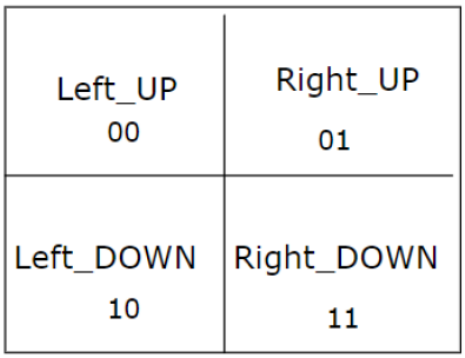

Figura 23. Poziția cadranelor

Tabelul 21. Formatul statusului de configurație

Poziția biților	7 6 5	4 3	2	1 0
Conținut	Baud rate	Parity	Stop Bit	VGA Resolution

Parametrul pentru reset este DEFAULT_CONFIG = 8'b01000000, BoudRate default 9600, niciun bit de parity, un bit de stop si rezoluția 640x480.

Color Manager primește numărătorul de pixeli, rezoluția actuală, starea culorii și transmite culoarea pentru VGA, dar și configurația VGA actuală spre numărătoarele interne.

Dacă nu este activat comutatorul modului de depanare VGA, Color Manager verifică datele primite de la UART și le salvează în regiștrii interni. Primirea unei culori pentru un cadran inactiv se rezumă la salvarea culorii în registrul corespunzător.

Config Notification este trimis doar când se primește o configurație pe care există codificare, adică se află în intervalul definit pentru codificarea datelor primite de la UART.

Config Error este trimis doar când se primește o configurație pe care nu există codificare, adică nu se află în intervalul definit pentru codificarea datelor primite de la UART.

În cazul în care se primește o configurație nouă, aceasta intră în viguare instant, din punctul de vedere al unui observator uman.

Transmiterea culorilor pentru VGA când acesta este în zona activa se face în funcție de rezoluție și coordonatele CounterX si CounterY (Counter_X si Counter_Y sunt cu un ciclu de clock în fața celor din VGA)

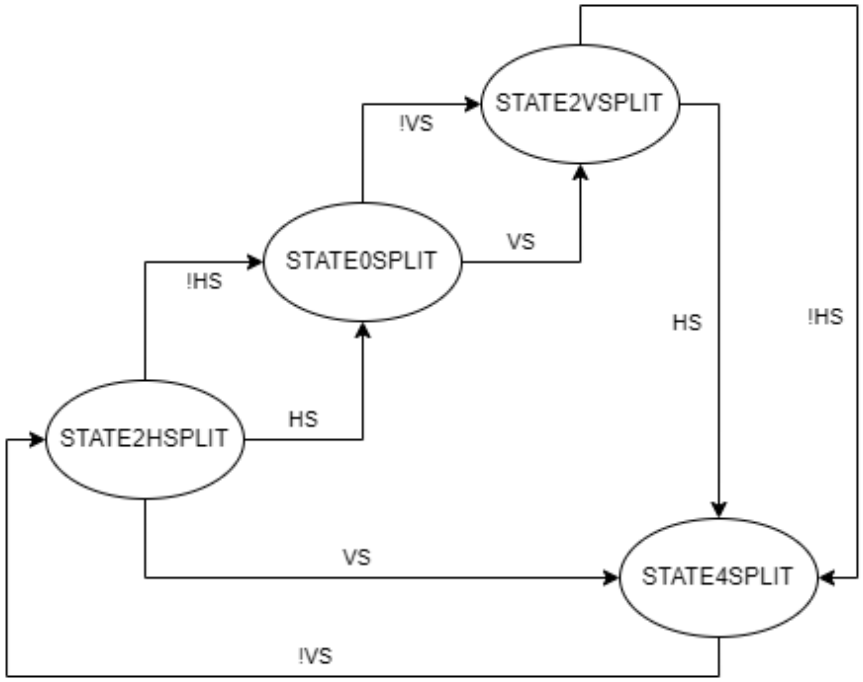

Figura 24. Diagrama de stări finite a managerului de cadrane

4.5.5. VGA

Modulul VGA este alcătuit din 3 tipuri de unități:

- Config: Rolul unității de configurare este să distribuie mai departe datele primite de la magistrala de configurații;
- Counter: Există 2 instanțieri ale unității, una pentru porțiunea verticală cu ieșirea VSync, alta pentru porțiunea orizonatală cu ieșire HSync. Aceste numărătoare primesc ca intrare și valoarea maximă admisă, definită în funcție de rezoluția selectată. Semanlul de HSync este conectat la intrarea de tact al numărătorului pentru VSync.
- Assign_color: Unitatea trimite mai departe culorile primite la intrarea modului VGA. Această transmisie este determinată de numărătorul pixelului, care trebuie să se afle în zona activă.

4.5.6. LED MANAGER (LM)

Modulul Led Manager este format din 3 decodificatoare pentru datele provenite de la modulul UART și erorile provenite de la modulele UART și CM.

5. IMPLEMENTARE

Designul realizat la pasul anterior trebuie să fie funcțional la nivel de unitate și cluster. Astfel, au fost realizate teste de bază pentru modulele simple și un environment de verificare randomizată pentru modulele complexe și pentru cluster.

Scenariile, cazurile de test și acoperirea au fost stabilite pentru a se asigura funcționarea robustă a proiectului.

5.1. VERIFICAREA UNITĂȚILOR

Primul pas în verificarea oricărui design este verificarea unităților. Acest pas este realizat construind testbenchuri simple pentru fiecare unitate. Aceste testbenchuri testează anumite funcționalități de bază în cazurile ideale și limită și nu sunt foarte complexe.

Modulul de sincronizare a fost testat în câteva cazuri predefinite, pentru că acesta nu ține de niciunul dintre modulele principale și este o singură unitate individuală.

5.2. VERIFICAREA MODULELOR

În această etapă se testează funcționarea unui modul compus din mai multe unități interconectate. Pentru această etapă se creează câte un environment de verificare pentru fiecare modul, respectând metodologia UVM.

5.2.1. INTERFACE

Interfețele sunt folosite pentru a accesa intrările și ieșirile DUT-ului. Acestea sunt componente standardizate, care conțin un clocking block pentru driver, unul pentru monitor, o funcție pentru recepția datelor pentru monitor și un task pentru trimiterea datelor pe driver.

Pentru verificarea tuturor modulelor au fost create 11 interfețe. O altă posibilitate era crearea a 6 interfețe, una pentru fiecare modul, dar acestea nu ar fi putut fi reutilizabile și ar fi trebuit create secvențe, itemi și teste individualizate.

După cum se poate observa în tabelul următor, interfețele CD și LM nu au un clocking block pentru driver, deoarece aceste 2 interfețe vor fi folosite exclusiv pentru monitorizare pasivă.

Tabelul 22. Interfețele de verificare

Modul	Nr. interfețe	Nr. utilizări	driver	monitor
CD	1	1	-	X
CM	2	3	X	X
CONF	2	8	X	X
DB	1	2	X	X
LM	1	1	-	X
UART	2	3	X	X
VGA	2	3	X	X

Semnalul de clock a fost primit ca parametru de toate interfețele. Acesta este creat din testbench și nu poate fi modificat pe parcursul unui test.

Semnalul de reset nu a fost conectat deloc la interfețe. Acest lucru ușurează verificarea. Resetarea mediului de verificare va fi făcută în faza de resetare a UVM. Resetarea DUT-ului va fi făcută direct din testbench. De aceea, în faza de resetare trebuie să se adauge o întârziere virtuală pentru a nu semnaliza începerea fazei principală înainte ca resetarea DUT-ului să fie finalizată.

5.2.2. ITEM

Obiectele sunt cea mai primitivă componentă a sistemului de verificare. Toate obiectele conțin variabile de tip logic, randomizabile sau nu, care corespund intrărilor și ieșirilor din interfețe. De aceea, există un tip de obiect pentru fiecare interfață.

Toate tipurile de obiecte conțin și următoarele funcții:

- void copy(item t): înlocuiește obiectul curent cu cel dat ca parametru;
- bit compare(item t): compară obiectul curent și obietul primit ca parametru și returnează 1 dacă sunt egali și 0 altfel;
- void setDefault(): resetează obiectul la valorile default;
- bit equalDefault(): compară obiectul curent cu valorile default și returnează 1 dacă este default și 0 altfel;
- string convert2string(): returnează obiectul pentru afișare.

După cum a fost menționat anterior, obiectele majorității modulelor sunt compuse din variabile care corespund interfețelor. Singurul modul care nu respectă această regulă este obiectul de intrare UART. Acesta este compus dintr-un frame configurabil de UART, care urmează să fie despachetat în interfață.

5.2.3. SEQUENCE

Secvențele sunt entități parametrizabile compuse din obiecte. Acestea sunt setate din test și sunt trimise serial spre sequencerul din agent.

Mediile de configurare conțin agenți reactivi, care au nevoie de secvențe și sequenceri virtuali pentru a funcționa. Sequencerul virtual este instanțiat în environment și conține sequencerul agentului de configurare. Acesta este conectat mai departe la secvența virtuală.

În această secvență se creează câte un thread pentru fiecare obiect primit. Aceste threaduri monitorizează permanent FIFO-ul conectat la monitor printr-o funcție care blochează rularea codului. După primirea obiectului din FIFO se începe noua secvență de configurare.

Sequencerii virtuali pot conține toți sequencerii agenților, astfel încât orice tip de secvență poate fi începută din secvența virtuală. În acest design nu a fost aleasă această metodă, deoarece complexitatea proiectului nu necesită implementări suplimentare.

5.2.4. TEST

Testele sunt folosite pentru a verificare funcționalitățile standard și specifice ale modulelor.

Tabelul 23. Teste

Test	Unitate	Descriere
CD standard	CD	Funcționarea modulului când sunt primite configurări randomizate la viteze scăzute.
CD no config		Funcționarea la viteze scăzute fără reconfigurări.
CM standard	CM	Funcționarea standard.
CS standard	CS	Funcționarea standard.
DB standard	DB	Funcționarea la viteze scăzute.
LM standard	LM	Funcționarea cu valori imposibile în practică.
UART standard	UART	Funcționarea standard.
UART error		Funcționarea modulului la primirea frame-urilor de UART eronate.
UART no config		Funcționarea standard fără reconfigurări.
VGA standard	VGA	Funcționarea modulului când sunt primite configurări randomizate.
VGA no config		Funcționarea standard fără reconfigurări.
VGA no data		Funcționarea pasivă a modulului.

Fazele UVM suprascrise în interiorul testului sunt:

- build: În această fază sunt construite și setate obiectele de configurare, care sunt apoi asociate unui environment prin baza de date. Secvențele simple și virtuale sunt create tot în această fază și sunt parametrizate cu specificațiile testului.
- start of simulation: Se afișează topologia testului.
- main: La începutul fazei se setează timpul de drenaj. Această fază este controlată de ridicarea și coborârea obiecțiilor. În cele mai multe situații se folosește o variantă de fork pentru a crea mai multe threaduri pentru rularea paralelă a secvențelor. În cazul agenților reactivi folosirea threadurilor este obligatorie.

5.2.5. AGENT

Agenții sunt conectați la interfețe și fac legătura dintre secvențele trimise din test și DUT. Fiecare interfață este conectată la un agent, astfel există 11 tipuri diferite de agenți, cu 11 tipuri de monitoare și 9 tipuri de drivere. Nu toți agenții au nevoie de drivere, deocare unii agenți pot fi doar pasivi. Toți agenții au o structură asemănătoare, singurele diferențe reieșiind din tiputile specifice de sequenceri și de posibilitatea agentului de a fi activ / reactiv / pasiv.

Toate acțiunile întreprinse de agent au loc în timpul fazelor UVM predefinite. Dintre aceste faze se suprascriu fazele de build, connect și run.

Tabelul 24. Fazele UVM ale agentului

Fază UVM	Entitate	Descriere
build_phase	agent	În această fază se construiește agentul. Primul pas este preluarea interfeței și obiectului de configurare din baza de date. Dacă nu se pot prelua, atunci o eroare fatală va fi executată și rularea va înceta. Următorul pas este crearea monitorului și setarea interfeței pentru monitor. Monitorul este creat indiferent de tipul de agent. În ultimul rând se verifică dacă agentul este activ, caz în care se creează un sequencer, un driver și se conectează interfața și la driver.
	driver	În această fază se preia interfața virtuală din baza de date. Dacă nu se poate realizare preluare se trimite o eroare fatală și rularea încetează.
	monitor	În această fază se preia interfața virtuală din baza de date. Dacă nu se poate realizare preluare se trimite o eroare fatală și rularea încetează. Obiectele și portul de export sunt create și obiectul anterior este setat la valorile default.
connect_phase	agent	În această fază se conectează portul de import al riverului la portul de export al sequencerului dacă agentul este activ.
run_phase	monitor	În această fază se monitorizează permanent ieșirile DUT-ului. Aceasta este prima fază care consumă timp și trebuie controlată prin clocking blockul moitorului. În interiorul buclei de monitorizare se primește obiectul de la interfață. Dacă acest obiect este diferit de obiectul anterior sau nu este default, atunci obiectul este preluat de monitor, afișat și trimis mai departe prin portul de export. La final se rescrie obiectul anterior cu obiectul curent.
reset_phase	driver	În această fază se resetează toate variabilele de tip input din interfață și se adaugă întârzierea virtuală. Înainte de începerea resetării se ridică obiecția, pentru a anunța că orice altă acțiune este oprită până la coborârea ei, care se ntâmplă după resetare.
main_phase	driver	În această fază se trimit permanent intrările DUT-ului. Când driverul primește de la sequencer un item, acesta este trimis prin interfață la DUT. Driverul trimite un semnal înapoi spre sequencer, pentru a anunța că este liber să primească alt obiect.

Fiecare agent conține:

- o interfață virtuală;
- un sequencer care primește ca parametru tipul obiectului;
- un driver definit, care conține:
  - o interfață virtuală;
  - un obiect de tipul primit ca parametru;
- un monitor definit, care conține:
  - o interfață virtuală;
  - un port de export care primește ca parametru tipul obiectului;
  - 2 obiecte pentru a stoca datele curente și anterioare;
- o configurație de agent.

5.2.6. ENVIRONMENT

Mediul de verificare este compus din mai mulți agenți și o componentă de coverage. Astfel, pentru fiecare modul ce trebuie verificat există un environment.

Aceste medii de verificare trec prin 2 faze UVM:

- build: În această fază se preia obiectul de configurare a environmentului din baza de date, după care fiecare environment este constuit diferit.
  - Agenții care pot fi doar pasivi sunt construiți direct.
  - Restul agenților au nevoie de un obiect de configurare în care se precizează tipul agentului (pasiv, activ). Acest obiect este apoi scris în baza de date și asociat agentului corespunzător. În final sunt creați agenții.
- connect: Componenta de coverage este conectată la monitoarele corecpunzătoare și în cazul agenților reactivi, sequencerul agentului reactiv este conectat la sequencerul virtual.

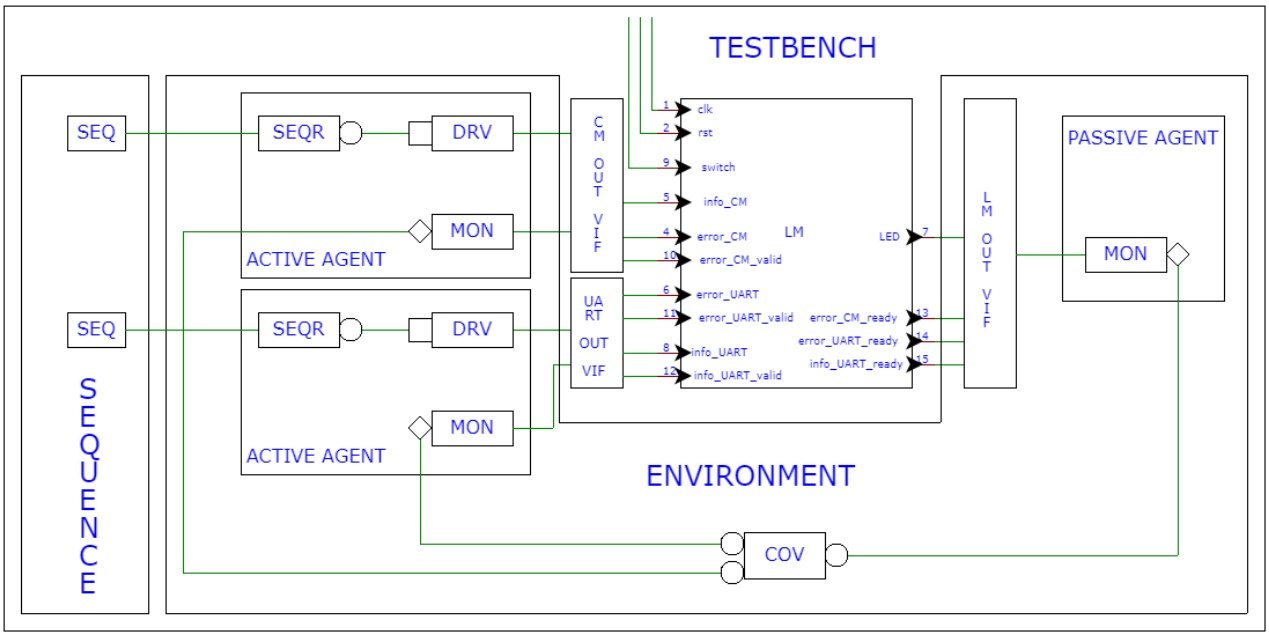

Figura 25. Design verificare LM

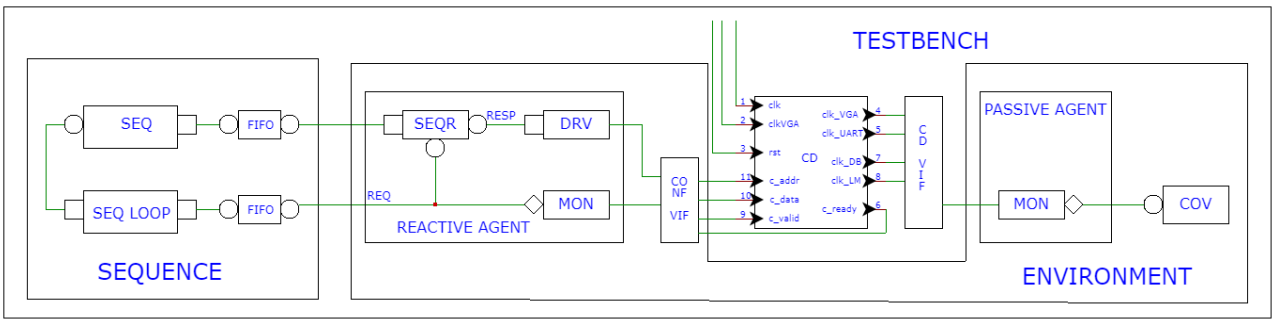

Figura 26. Design verificare CD

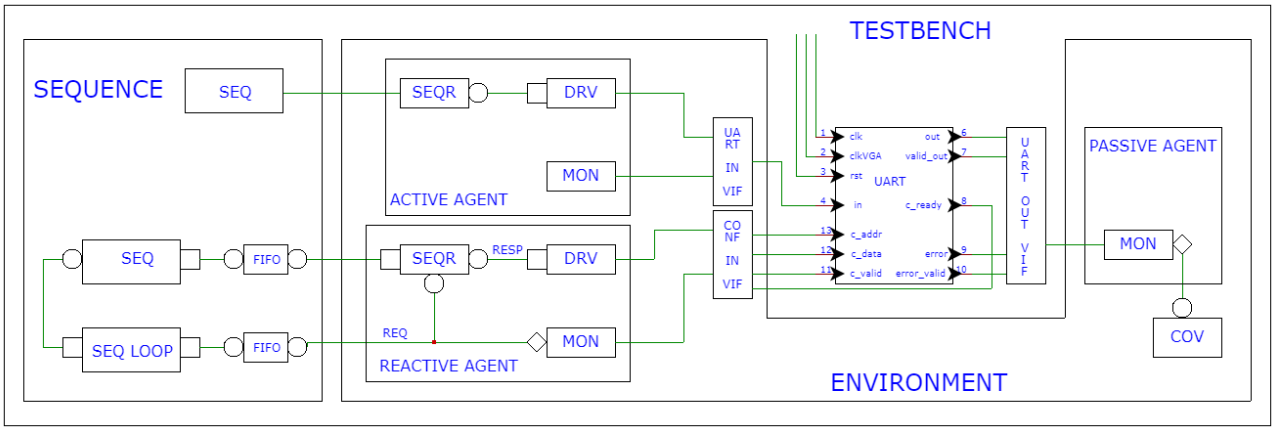

Figura 27. Design verificare UART

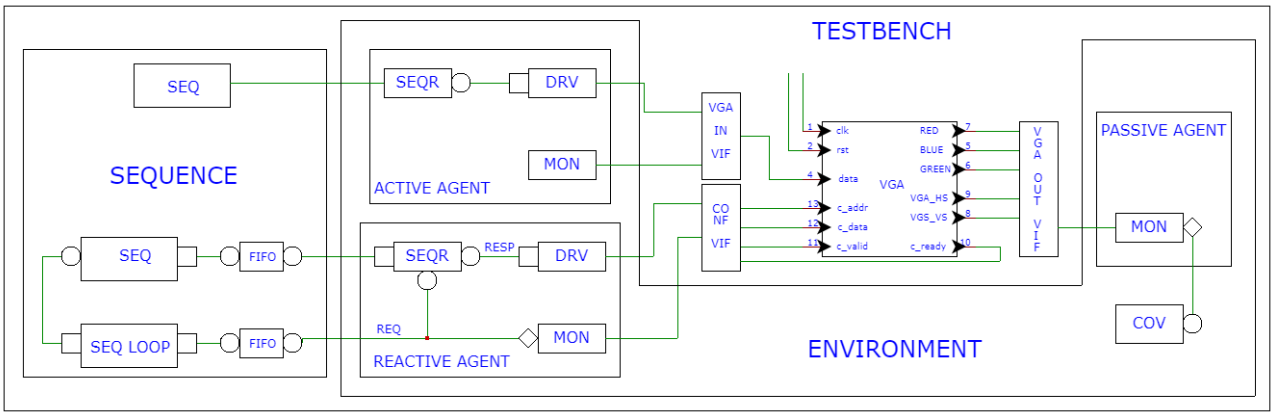

Figura 28. Design verificare VGA

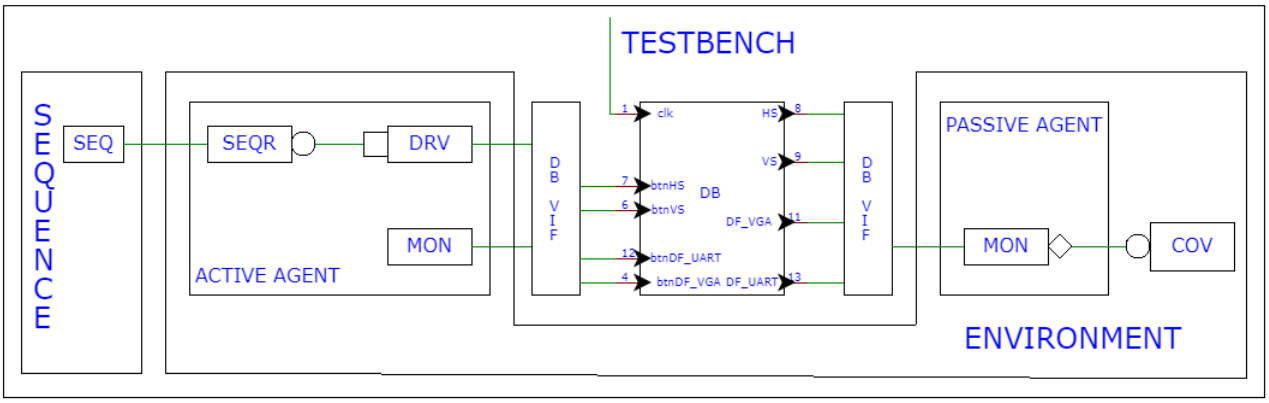

Figura 29. Design verificare DB

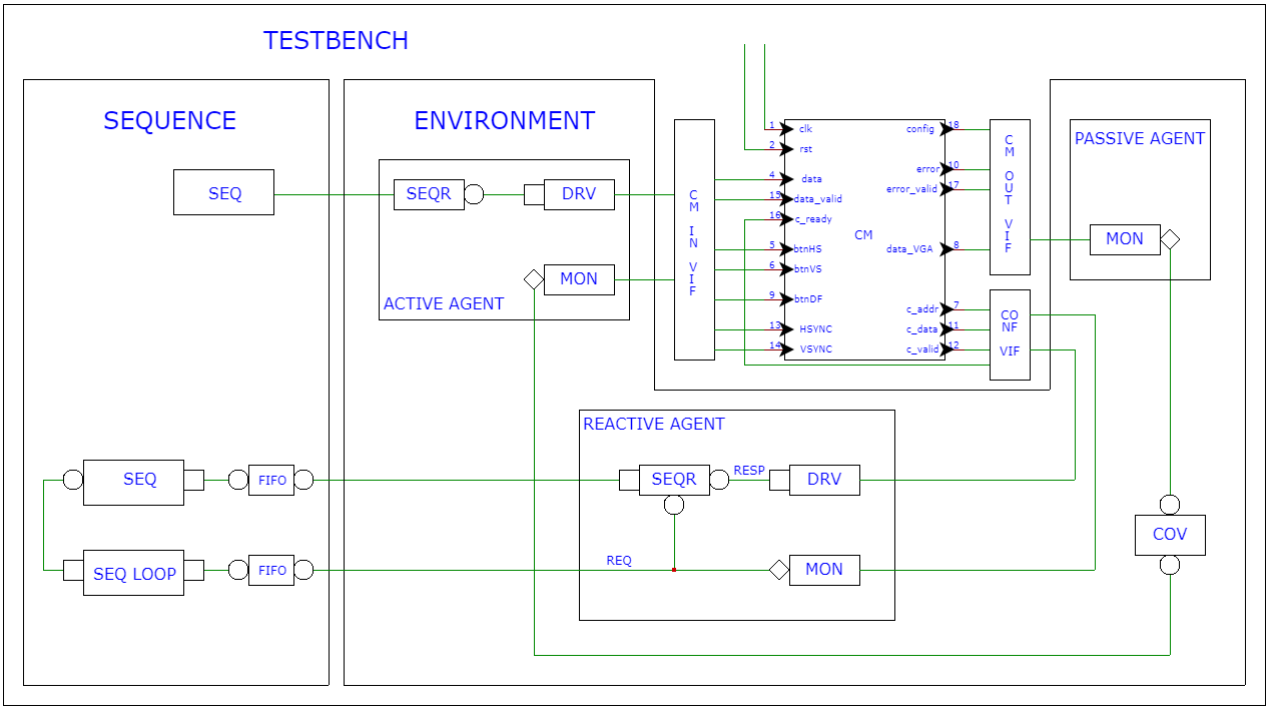

Figura 30. Design verificare CM

De precizat că în toate figurile anterioare nu a fost reprezentată entitatea test, care ar fi trebuit să fie cuprinsă în testbench și să conțină secvențele și environmentul. Pentru reprezentarea testului era necesară detalierea secvențelor și ordinii acestora pentru fiecare test în parte. Așadar, a fost aleasă varianta de reprezentare fără teste.

```
# Name                                Type                                Size  Value
# ------------------------------------------------------------------------------------
# uvm_test_top                        UART_test                           -     @483
#   env                               UART_environment                    -     @494
#     CONF_agent_h                    CONF_input_agent                    -     @639
#       CONF_input_driver             CONF_input_driver                   -     @832
#         rsp_port                    uvm_analysis_port                   -     @847
#         seq_item_port               uvm_seq_item_pull_port              -     @839
#       CONF_input_monitor            CONF_input_monitor                  -     @857
#         an_port                     uvm_analysis_port                   -     @874
#       CONF_input_seqr               CONF_input_sequencer                -     @668
#         export_port                 uvm_analysis_export                 -     @777
#         fifo                        uvm_tlm_analysis_fifo #(T)          -     @785
#           analysis_export           uvm_analysis_imp                    -     @824
#           get_ap                    uvm_analysis_port                   -     @816
#           get_peek_export           uvm_get_peek_imp                    -     @800
#           put_ap                    uvm_analysis_port                   -     @808
#           put_export                uvm_put_imp                         -     @792
#         rsp_export                  uvm_analysis_export                 -     @675
#         seq_item_export             uvm_seq_item_pull_imp               -     @769
#         arbitration_queue           array                               0     -
#         lock_queue                  array                               0     -
#         num_last_reqs               integral                            32    'd1
#         num_last_rsps               integral                            32    'd1
#     CONF_input_virtual_sequencer    CONF_input_virtual_sequencer        -     @520
#       rsp_export                    uvm_analysis_export                 -     @527
#       seq_item_export               uvm_seq_item_pull_imp               -     @621
#       arbitration_queue             array                               0     -
#       lock_queue                    array                               0     -
#       num_last_reqs                 integral                            32    'd1
#       num_last_rsps                 integral                            32    'd1
```

Figura 31. Topologia testului

În figura anterioară se poate observa topologia unui test specific. Primele 2 coloane reprezintă numele și tipul entităților. Datorită metodologiei UVM toate entitățile au un nume și un tip de bază, care trebuie suprascris sau folosit direct.

Componentele suprascrise sunt testul, environmentul, coverageul, agenții, monitoarele și driverele. Sequencerele sunt folosite cu implementarea UVM. Acestea sunt suprascrise doar în cazul agenților reactivi.

5.2.7. COVERAGE

Există câte un coverage pentru fiecare dintre cele 6 module. Modulul de coverage este foarte important pentru a verifica acoperirea cazurilor de test. Un coverage se va considera valid dacă depășește o acoperire de 80%.

Clasa coverage este formată din unul sau mai multe porturi de import conectate la monitoarele din agenți. Dacă există un singur monitor, atunci conexiunea se va realiza direct. Dacă este nevoie să se conecteze mai multe monitoare la același coverage se vor declara porturi de import diferite, folosing comenzile predefinite din UVM.

Pentru fiecare port va exista o funcție, numită write. Inexistența acestei funcții duce la emiterea unei erori și la încetarea rulării. În fiecare funcție de write se va primi obiectul de la monitor și se va face sample la covergroupul corespunzător.

În ceea ce privește fazele UVM, una singură este suprascrisă, faza de report. În această fază se afișează procentul de acoperire pe fiecare covergroup în parte.

Tabelul 25. Covergroup CD

Coverpoint	Item	Bini	Nr. Bini	Conținut
clk_VGA_cvp	clk_VGA	value_binary	2	0, 1
clk_UART_cvp	clk_UART	value_binary	2	0, 1
clk_LM_cvp	clk_LM	value_binary	2	0, 1
clk_DB_cvp	clk_DB	value_binary	2	0, 1
clks_cross	clk_VGA_cvp clk_UART_cvp clk_LM_cvp clk_DB_cvp	-	16	-

Tabelul 26. Covergroup DB

Coverpoint	Item	Bini	Nr. Bini	Conținut
HS_cvp	HS	value_binary	2	0, 1
VS_cvp	VS	value_binary	2	0, 1
DF_UART_cvp	DF_UART	value_binary	2	0, 1
DF_VGA_cvp	DF_VGA	value_binary	2	0, 1
HS_VS_cross	HS_cvp VS_cvp	-	4	-
UART_VGA_cross	DF_UART_cvp DF_VGA_cvp	-	4	-

Tabelul 27. Covergroup CM

Coverpoint	Item	Bini	Nr. Bini	Conținut
HSync_cvp	HSync	value_binary	2	0, 1
VSync_cvp	VSync	value_binary	2	0, 1
Vertical_Split_cvp	Vertical_Split	value_binary	2	0, 1
Horizontal_Split_cvp	Horizontal_Split	value_binary	2	0, 1
Empty_cvp	Empty	value_binary	2	0, 1
VGA_debug_cvp	VGA_debug	value_binary	2	0, 1
RXD_Data_cvp	RXD_Data	inter_values	3	['h00 : 'h55], ['h56 : 'hAA], ['hAB : 'hFF]
Config_Notif_Valid_cvp	Config_Notif_Valid	value_binary	2	0, 1
VGA_Notif_Valid_cvp	VGA_Notif_Valid	value_binary	2	0, 1
Error_Valid_cvp	Error_Valid	value_binary	2	0, 1
Data_VGA_cvp	Data_VGA	values05	8	'h000, 'h005, 'h050, 'h055, 'h500, 'h505, 'h550, 'h555
		values0A	8	'h000, 'h00A, 'h0A0, 'h0AA, 'hA00, 'hA0A, 'hAA0, 'hAAA
		values0F	8	'h000, 'h00F, 'h0F0, 'h0FF, 'hF00, 'hF0F, 'hFF0, 'hFFF
		values5A	8	'h555, 'h55A, 'h5A5, 'h5AA, 'hA55, 'hA5A, 'hAA5, 'hAAA
		values5F	8	'h555, 'h55F, 'h5F5, 'h5FF, 'hF55, 'hF5F, 'hFF5, 'hFFF
		valuesAF	8	'hAAA, 'hAAF, 'hAFA, 'hAFF, 'hFAA, 'hFAF, 'hFFA, 'hFFF

Tabelul 28. Covergroup UART

Coverpoint	Item	Bini	Nr. Bini	Conținut
error_cvp	error	values	3	0, 1, 2
out_cvp	out	limit_values	4	'h00, 'h55, 'hAA, 'hFF
		inter_values	3	['h00 : 'h54], ['h56 : 'hA9], ['hAB : 'hFE]
valid_error_cvp	valid_error	value_binary	1	1
valid_out_cvp	valid_out	value_binary	1	1
data_cross	valid_out_cvp out_cvp	-	7	-
error_cross	valid_error_cvp error_cvp	-	3	-

Tabelul 29. Covergroup VGA

Coverpoint	Item	Bini	Nr. Bini	Conținut
RED_cvp	RED	limit_values	4	'h0, 'h5, 'hA, 'hF
		inter_values	3	['h0 : 'h4], ['h6 : 'h9], ['hB : 'hE]
GREEN_cvp	GREEN	limit_values	4	'h0, 'h5, 'hA, 'hF
		inter_values	3	['h0 : 'h4], ['h6 : 'h9], ['hB : 'hE]
BLUE_cvp	BLUE	limit_values	4	'h0, 'h5, 'hA, 'hF
		inter_values	3	['h0 : 'h4], ['h6 : 'h9], ['hB : 'hE]
HSync_cvp	HSync	value_binary	2	0, 1
VSync_cvp	VSync	value_binary	2	0, 1

Tabelul 30. Covergroup LM

Coverpoint	Item	Bini	Nr. Bini	Conținut
leds0_cvp	leds[0]	value_binary	2	0, 1
leds1_cvp	leds[1]	value_binary	2	0, 1
leds2_cvp	leds[2]	value_binary	2	0, 1
leds3_cvp	leds[3]	value_binary	2	0, 1
leds4_cvp	leds[4]	value_binary	2	0, 1
leds5_cvp	leds[5]	value_binary	2	0, 1
leds6_cvp	leds[6]	value_binary	2	0, 1
leds7_cvp	leds[7]	value_binary	2	0, 1

După cum reiese din tabelele anterioare, există 172 cazuri care trebuie acoperite:

- CD: 24 (8 coverpoint + 16 cross);
- DB: 16 (8 coverpoint + 8 cross);
- CM: 69 (69 coverpoint);
- UART: 22 (12 coverpoint + 10 cross);
- VGA: 25 (25 coverpoint);
- LM: 16 (16 coverpoint).

Pentru a valida proiectul este necesar ca 80% din binuri să fie acoperite. Luând în considerare faptul că fiecare bin are o importanță egală înseamnă că 138 de binuri trebuie să fie acoperite.

Este important de precizat că numărul binurilor trebuie să fie relativ mic pentru a nu adăuga complexitate nedorită verificării. De aceea, binurile sunt stabilite încă de la începutul proiectului.

Rapoatele de coverage sunt salvate individual pentru fiecare test folosind scriptul de rulare a testelor UVM cu opțiuni speciale pentru simulare. Aceste opțiuni suplimentare se referă la salvarea tuturor detaliile referitoare la coverage, precum opțiunile specifice, statisticile individuale ale punctelor și intersecțiilor, numărul de lovituri ale unui bin.

5.3. VERIFICAREA SISTEMULUI

În această etapă se va verifica întreg designul, în alte cuvinte, clusterul.

Pentru realizarea testbenchului și mediului de verificare se vor refolosi interfețele, environmentele și agenții creați anterior pentru verificarea modulelor.

Această integrare este foarte eficientă, deoarece nu mai necesită rescrierea specificațiilor și implementarea acestora. Dacă mediile au fost făcute compatibile cu nivelul de cluster, atunci integrarea propriu zisă este foarte simplă.

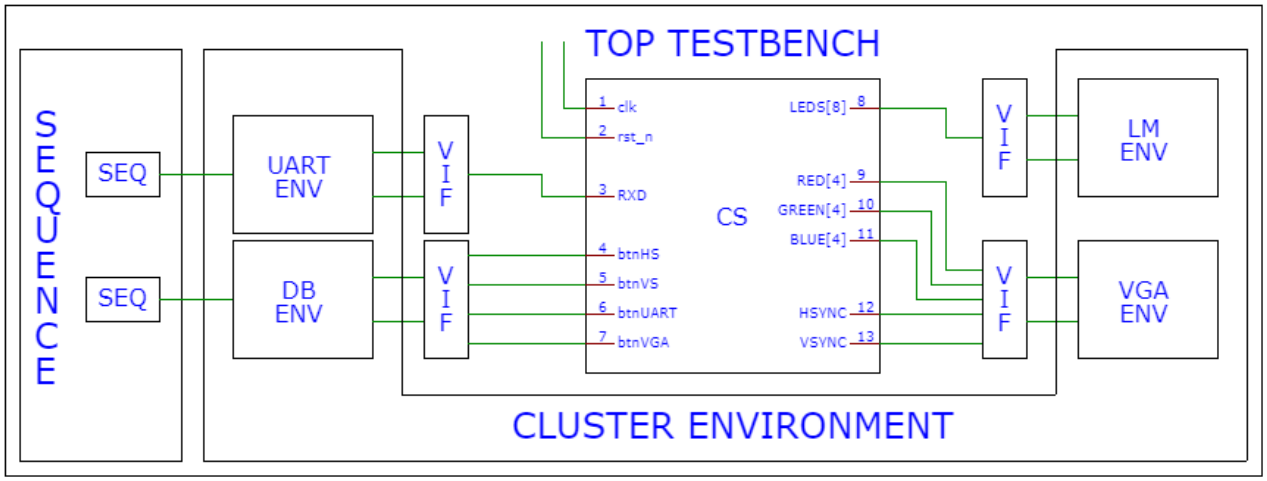

Figura 32. Design verificare CS

Mediul este compus din 4 environmente pentru modulele UART, DB, LM și VGA. La verificarea la nivel de cluster nu mai este nevoie de crearea agenților pasivi pentru modulele din interiorul sistemului. Numai intrările și ieșirile contează.

Este de așteptat ca orice bug găsit la această etapă să fie mult mai greu de depanat. Contrar așteptărilor, la această verificare sunt găsite cele mai multe buguri din cauze multiple, printre care sincronizarea defectuasă și nerealizarea modulelor conform cerințelor.

5.4. VERIFICAREA FUNCȚIONALĂ

Aserțiile sunt folosite pentru validarea automată a codului. Aceste module sunt legate de DUT prin instrucțiunea specială bind. De precizat că numele variabilelor create în testbench trebuie să aibă același nume ca intrările din modulul de aserții.

Tabelul 31. Aserții UART

Nr.	Nume	Descriere
1	VALID OUT	După primirea semnalul de confirmare a datelor primit din logica suplimentară se va activa într-un ciclu viitor ieșirea valid out.
2	VALID ERROR	După primirea semnalul de confirmare a erorii din logica suplimentară se va activa într-un ciclu viitor ieșirea valid error.

A fost folosită logică suplimentară pentru a ajuta la formarea aserțiilor. Așadar, în blocurile always suplimentare a fost folosit un FSM și numărătoare pentru a urmări primirea datelor pe UART. Au fost folosite 2 registre pentru a salva semnalele de confirmare. Aceste semnale sunt apoi folosite în aserții.

Tabelul 32. Aserții DB

Nr.	Nume	Descriere
1	ENABLE HS	Dacă semnalul de intrare btnHS este activ de 4 ori la rând, atunci semnalul de ieșire HS va fi activ și el.
2	DISABLE HS	Dacă semnalul de intrare btnHS este inactiv de 4 ori la rând, atunci semnalul de ieșire HS va fi inactiv și el.
3	ENABLE VS	Dacă semnalul de intrare btnVS este activ de 4 ori la rând, atunci semnalul de ieșire VS va fi activ și el.
4	DISABLE VS	Dacă semnalul de intrare btnVS este inactiv de 4 ori la rând, atunci semnalul de ieșire VS va fi inactiv și el.
5	ENABLE UART	Dacă semnalul de intrare btnDF_UART este activ de 4 ori la rând, atunci semnalul de ieșire DF_UART va fi activ și el.
6	DISABLE UART	Dacă semnalul de intrare btnDF_UART este inactiv de 4 ori la rând, atunci semnalul de ieșire DF_UART va fi inactiv și el.
7	ENABLE VGA	Dacă semnalul de intrare btnDF_VGA este activ de 4 ori la rând, atunci semnalul de ieșire DF_VGA va fi activ și el.
8	DISABLE VGA	Dacă semnalul de intrare btnDF_VGA este inactiv de 4 ori la rând, atunci semnalul de ieșire DF_VGA va fi inactiv și el.

Tabelul 33. Aserții LM

Nr.	Nume	Descriere
1	WRITE CM ERROR	LED-urile 3:0 vor prelua valoarea anterioară de la CM errors, dacă în ciclul anterior întrerupătorul de debug UART este inactiv și eroarea primită este validă.
2	WRITE UART ERROR	LED-urile 5:4 vor prelua valoarea anterioară de la UART errors, dacă în ciclul anterior întrerupătorul de debug UART este inactiv și eroarea primită este validă.
3	WRITE UART DATA	LED-urile 7:0 vor prelua valoarea anterioară de la UART data, dacă în ciclul anterior întrerupătorul de debug UART este inactiv și data primită este validă.
4	WRITE RESET	LED-ul 7 va afișa constant valoarea semnalului de reset.
5	WRITE SWITCH	LED-ul 6 va afișa constant valoarea semnalului de debug UART.

Tabelul 34. Aserții VGA

Nr.	Nume	Descriere
1	POS HS	Se verifică sinronizarea semnalului de sincronizare pe orizontală.
2	NEG HS	Se verifică sinronizarea semnalului de sincronizare pe orizontală.
3	INACTIVE DATA	Cei 12 biți de date trebuie să fie inactivi în momentele de sincronizare.
4	POS VS	Se verifică sinronizarea semnalului de sincronizare pe verticală.
5	NEG VS	Se verifică sinronizarea semnalului de sincronizare pe verticală.

În cazul aserțiilor 1, 2, 4 și 5 din tabelul aserțiilor pentru VGA trebuie specificat detaliat începutul aserției, deoarece, în caz contrar, vor fi începute aserții în fiecare ciclu de clock și simularea va dura foarte mult sau nu se va putea efectua până la capăt.

Au fost construite aserții pentru 4 module: UART, VGA, LM și DB. Modulele CD și CM au fost validate manual. Fiecare modul de aserții este compus din mai multe proprietăți și aserții cu același nume.

5.5. VERIFICAREA PROIECTULUI

Pentru verificarea pe placă a sistemului a fost folosită placa MachXO3D, alături de aplicația Diamond de la Lattice Semiconductors și aplicația open source PulseView, cu ajutorul plăcii Raspberry Pi Pico. La placa MachXO3D a fost conectat un modul UART prin Bluetooth și un conector VGA.

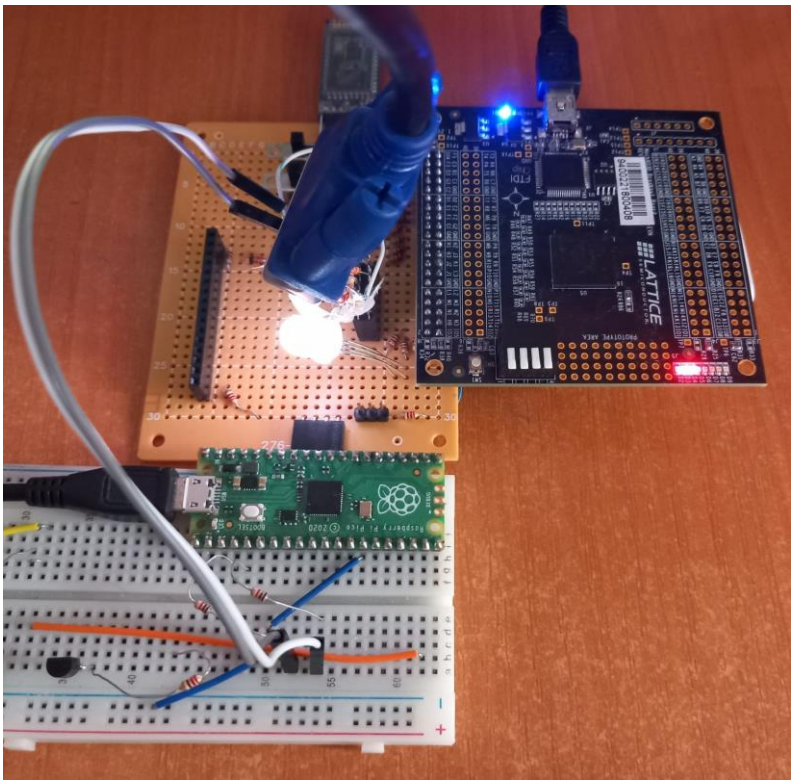

Figura 33. Plăcile MachXO3D și Raspberry Pi Pico

În primul rând, am folosit aplicația Pulseview pentru a vizualiza semnalele de HSync și VSync pentru VGA.

Figura 34. HSYNC în PulseView

Figura 35. VSYNC în PulseView

Pentru transmisia datelor prin Bluetooth am folosit aplicația mobilă Serial Bluetooth Terminal, care are setate valorile standard pentru transmisia UART de 8N1.

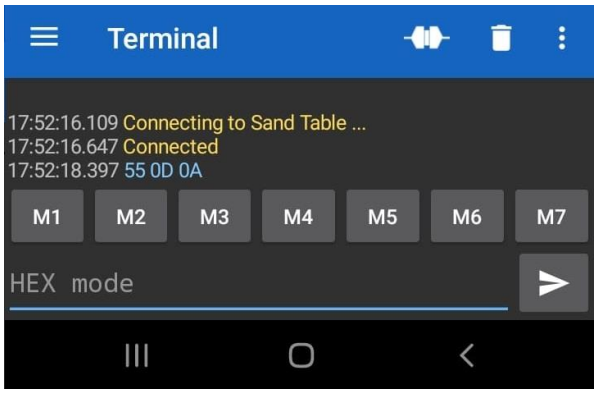

Figura 36. UART 8N1 HEX 00 prin Bluetooth

Aplicația PulseView oferă și standarde implementate, pentru a depana mai ușor transmisia datelor. Unul dintre aceste standarde este UART.

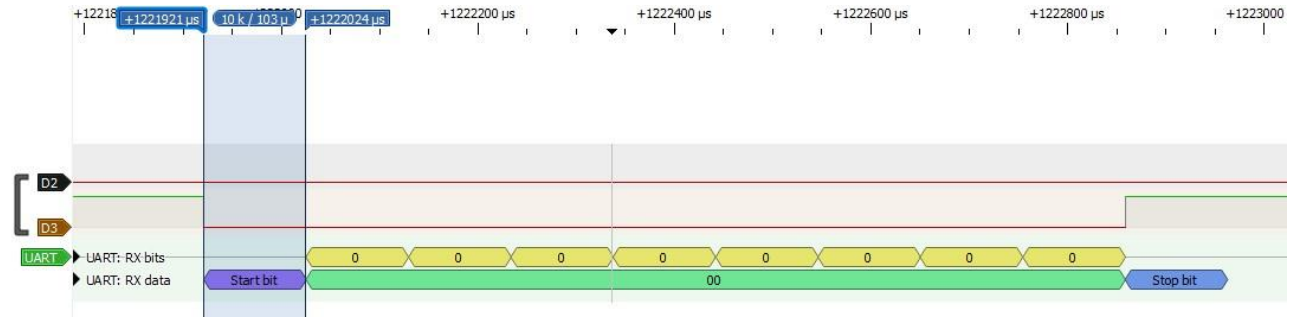

Figura 37. UART 8N1 HEX 00 în PulseView

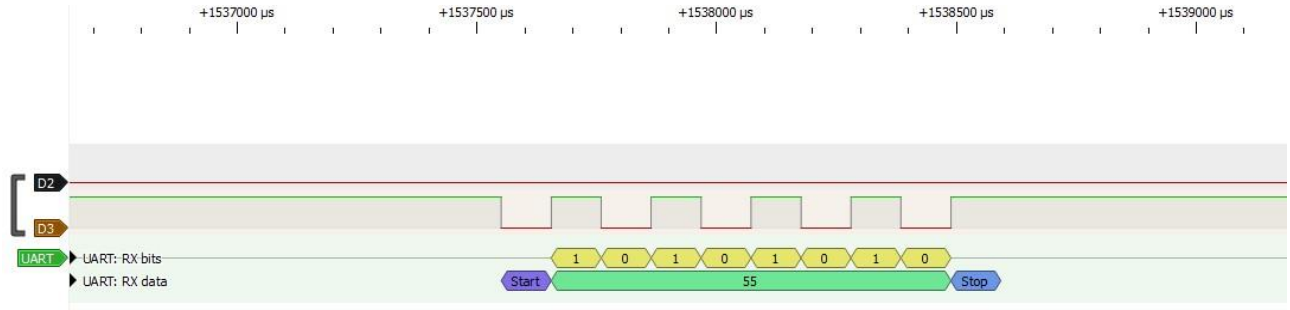

Figura 38. UART 8N1 HEX 55 în PulseView

6. REZULTATE EXPERIMENTALE

Rezultatele experimentale se pot împărți în 3 categorii distincte: coverage (acoperirea valorilor esențiale), simulare (comportamentul designului) și validare (funcționarea corectă a designului).

6.1. COVERAGE

În următoarele figuri se pot observa headerul grupului de cover și detalliile despre un coverpoint din raportul de coverage pentru testul simplu de UART.

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------
Covergroup                                              Metric       Goal    Status

----------------------------------------------------------------------------------------
 TYPE /VGA_coverage_pack/VGA_covergroup                100.00%        100    Covered
    covered/total bins:                                     25         25
    missing/total bins:                                      0         25
    % Hit:                                              100.00%        100
    type_option.weight=1
    type_option.goal=100
    type_option.comment=
    type_option.strobe=0
    type_option.merge_instances=auto(0)
```

Figura 39. Header covergroup

```
  Coverpoint VGA_covergroup::RED_cvp                    100.00%        100    Covered
      covered/total bins:                                     7          7
      missing/total bins:                                     0          7
      % Hit:                                              100.00%        100
      type_option.weight=1
      type_option.goal=100
      type_option.comment=
```

Figura 40. Rezumat coverpoint

```
  Coverpoint RED_cvp                                    100.00%        100    Covered
      covered/total bins:                                     7          7
      missing/total bins:                                     0          7
      % Hit:                                              100.00%        100
      option.weight=1
      option.goal=100
      option.comment=
      option.at_least=1
      option.auto_bin_max=64
      option.detect_overlap=0
      bin limit_values[0]                                 60666          1    Covered
      bin limit_values[1]                                 17948          1    Covered
      bin limit_values[2]                                 10256          1    Covered
      bin limit_values[3]                                  8333          1    Covered
      bin inter_values[0]                                 78614          1    Covered
      bin inter_values[1]                                 18589          1    Covered
      bin inter_values[2]                                 26281          1    Covered
```

Figura 41. Detalii coverpoint

Acest covergroup are o acoperire de 100%, cu toate cele 25 de binuri acoperite. Fiecare bin are o greutate de 1 și necesită cel puțin o lovitură. Se poate observa că fiecare bin a fost lovit între 8333 și 78614 ori.

În urma simulării s-a ajuns la un procent de coverage general de 96.26%, conform tabelului următor.

Tabelul 35. Coverage final

Unitate	Test	Procent [%]	Procent final / unitate [%]
CD	CD	100.00	100.00
	CD no config	100.00	
CM	CM	100.00	88.28
	CM	76.56	
DB	DB	100.00	100.00
LM	LM	100.00	100.00
UART	UART	82.53	89.68
	UART error	80.15	
	UART no config	50.79	
VGA	VGA	100.00	100.00
	VGA no config	100.00	
	VGA no data	57.17	

6.2. SIMULARE

Testele au fost rulate pentru a verifica și valida DUT-ul. Pentru fiecare modul testat a fost creat un format de waveform pentru a ușura vizualizarea datelor, în cazul testelor care necesită validare manuală.

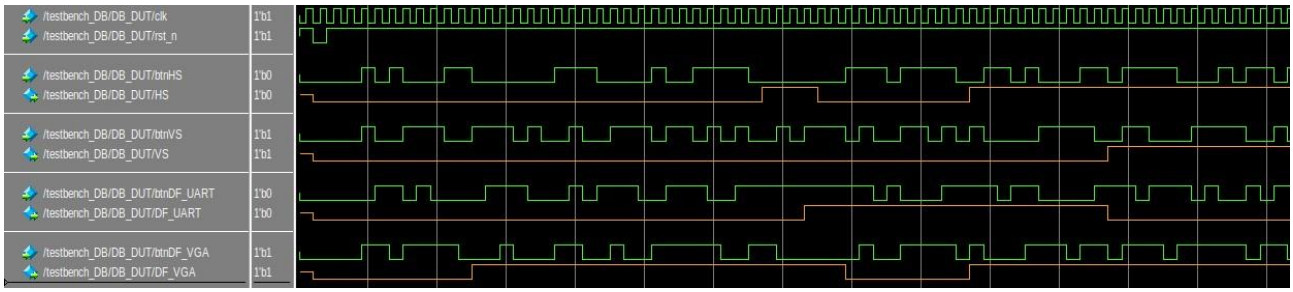

Figura 41. Waveform DB

În simularea modulului DB intrările sunt reprezentate de culoarea verde, iar ieșirile sunt portocalii. Pentru testare au fost folosite limite mai mici decât în raelitate.

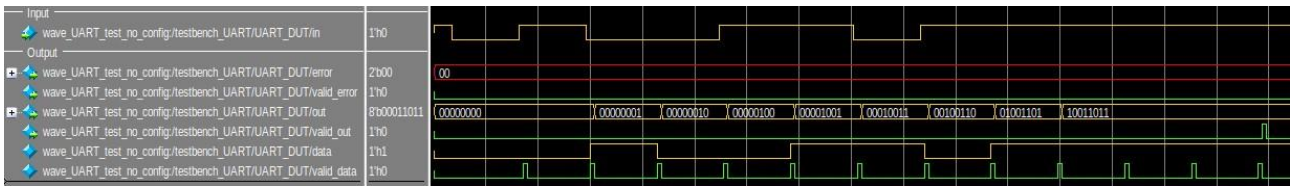

Figura 42. Waveform frame UART valid

Modulul UART a fost verificat prin 3 teste distincte. Datele de intrare și ieșire sunt reprezentate de galben, erorile sunt roșii și biții de confirmare, configurare și semnalele de tact au culoarea verde.

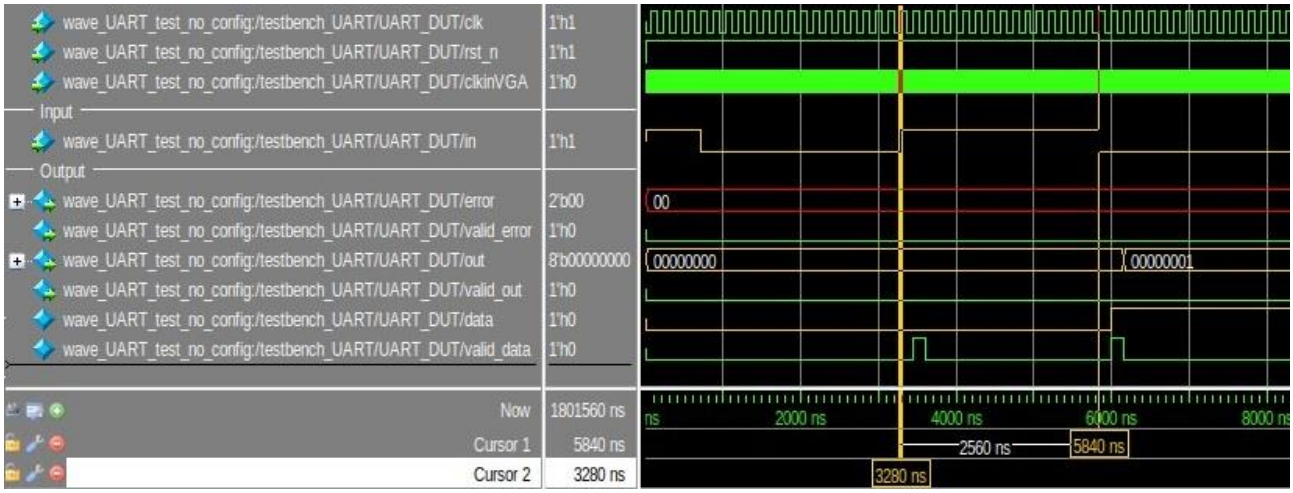

Figura 43. Waveform supraeșantionare UART

Supra eșantionarea se efectuează cu o viteză de 16 ori mai mare decât frecvența modulului UART. După numărarea eșantioanelor se validează bitul.

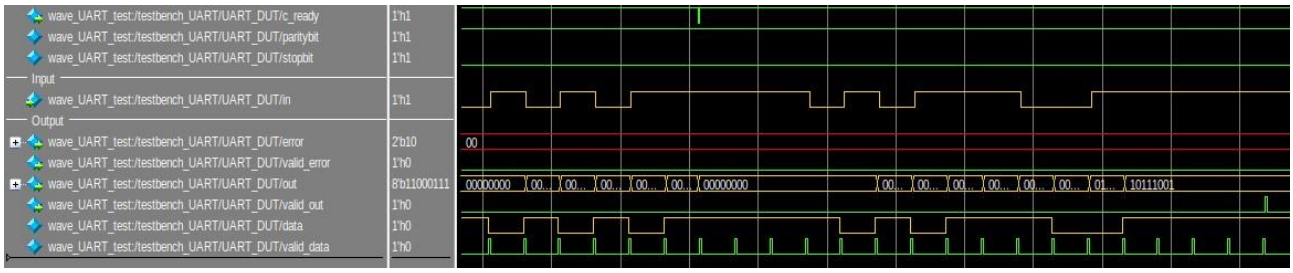

Figura 44. Waveform reconfigurare UART

Bitul de confirmare a reconfigurării resetează transmisia UART și poate cauza transmisia datelor fals validate.

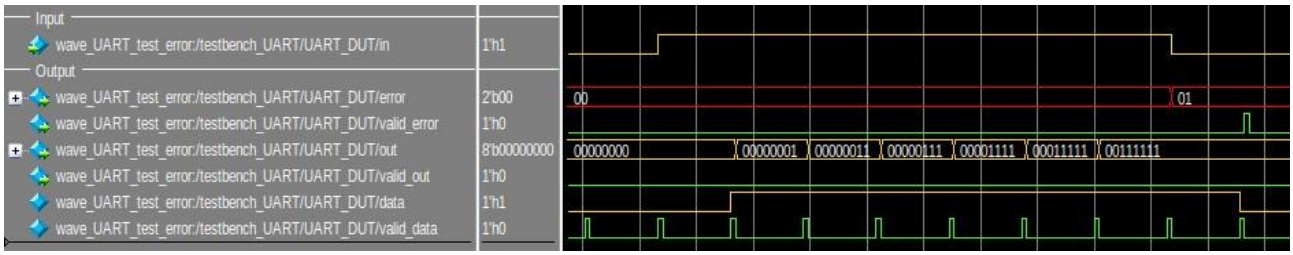

Figura 45. Waveform eroare UART

În forma de undă anterioară se poate observa cazul de eroare de bit de stop.

În urma simulării s-a constatat că nu se pot face transmisii back to back, deoarece unitatea FSM a modulului UART nu este suficient de rapidă. Astfel, este recomandată o pauză corespunzătoarel timpului de transmisie necesar pentru 3 biți de date.

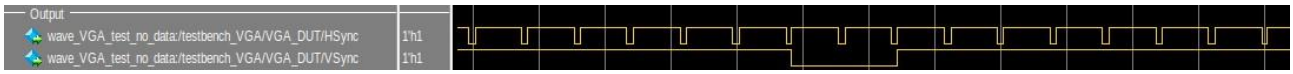

Figura 46. Waveform VGA HSYNC

Figura 47. Waveform VGA VSYNC

În figurile anterioare se pot observa semnalele de sinronizare ale modulului VGA.

Figura 48. Waveform date VGA

Fiecare pixel este reprezentat de 12 biți, 4 pentru fiecare culoare RGB.

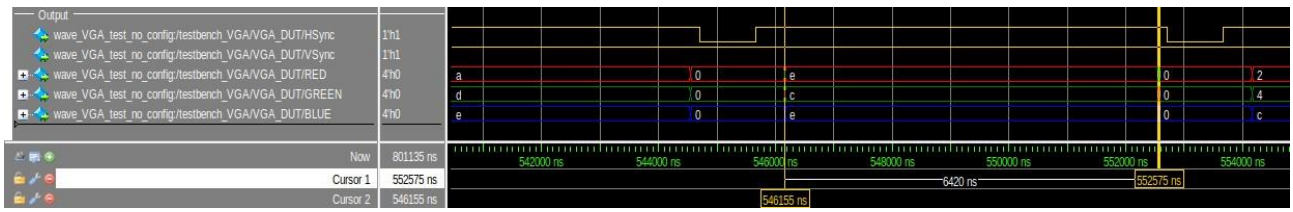

Figura 49. Waveform sincronizare VGA

Culoarea pixelului este trimisă doar pe durata activă a sincronizării. În timpul sincronizării orizontale sau verticale nu se trimit date.

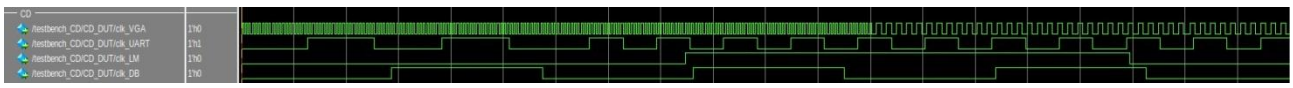

Figura 50. Waveform CD

În figura anterioară se pot observa cele 4 semnale de tact default folosite de sistem. Valorile limită ale numărătoarelor nu sunt cele folosite în practică.

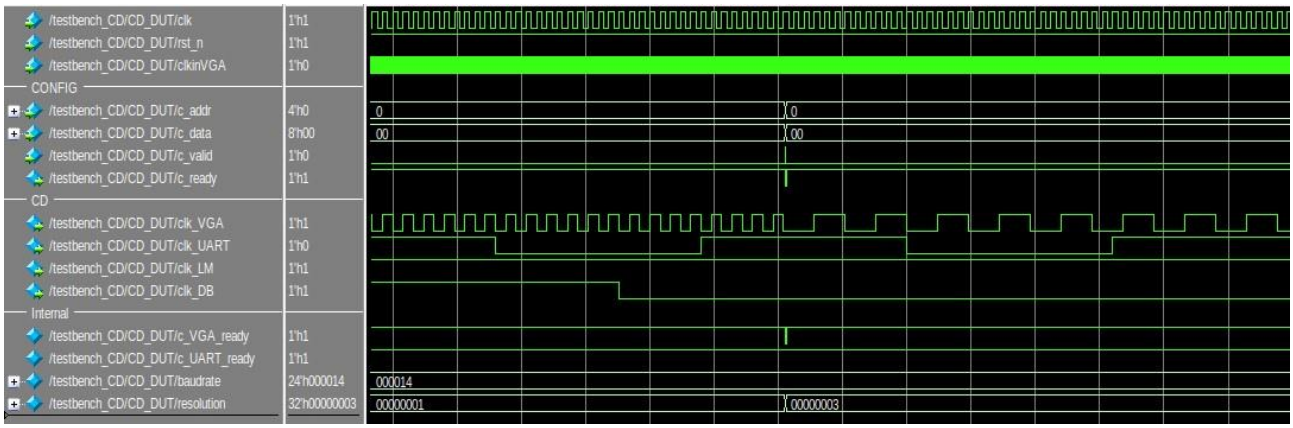

Figura 51. Waveform reconfigurare CD

Primirea unei noi configurații valide presupune resetarea ciclului de tact. În exemplul de mai sus se poate observa reconfigurarea frecvenței VGA.

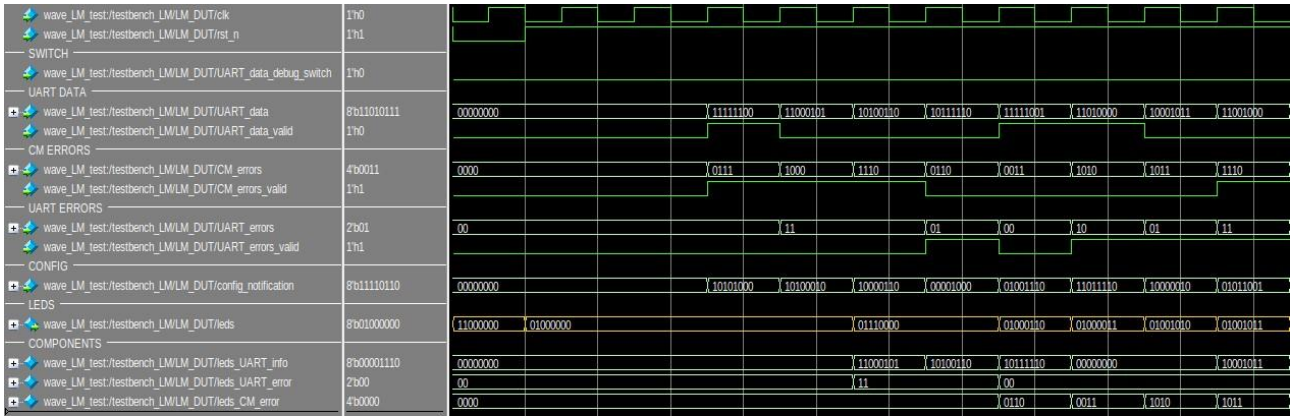

Figura 52. Waveform LM

Modulul LM are ieșirea reprezentată de culoarea portocalie și toate intrătile sunt verzi. Există o întârziere de doar un tact pentru datele primite.

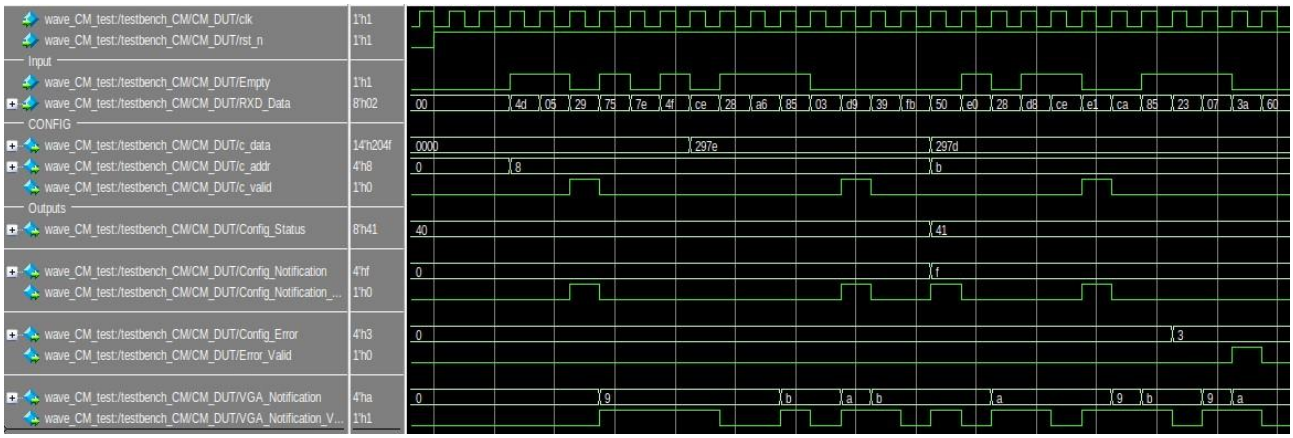

Figura 53. Waveform CM

Modulul CM este controlat în totalitate de modulele UART și DB. S-a constatat că modulul este rezistent resetărilor multiple, permite primirea datelor fără pauză de la UART și acceptă schimbările frecvente ale întrerupătoarelor.

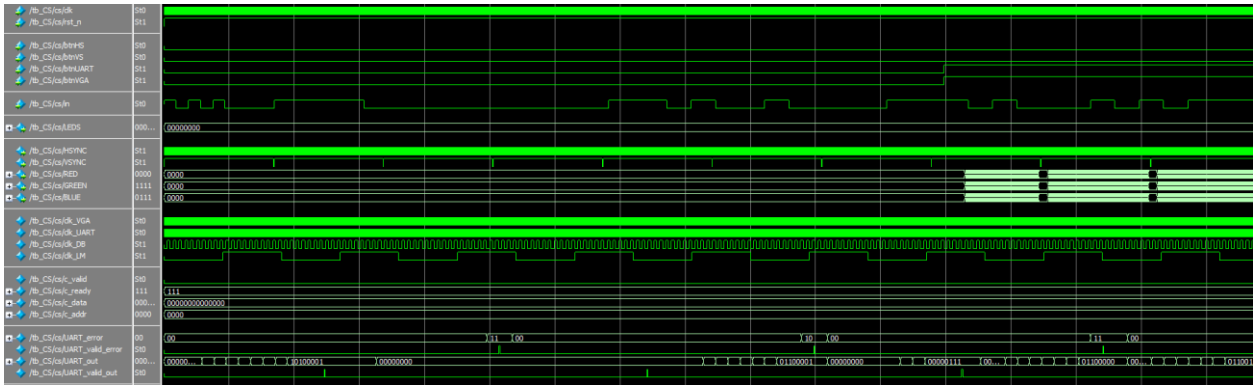

Figura 54. Waveform CS detaliat

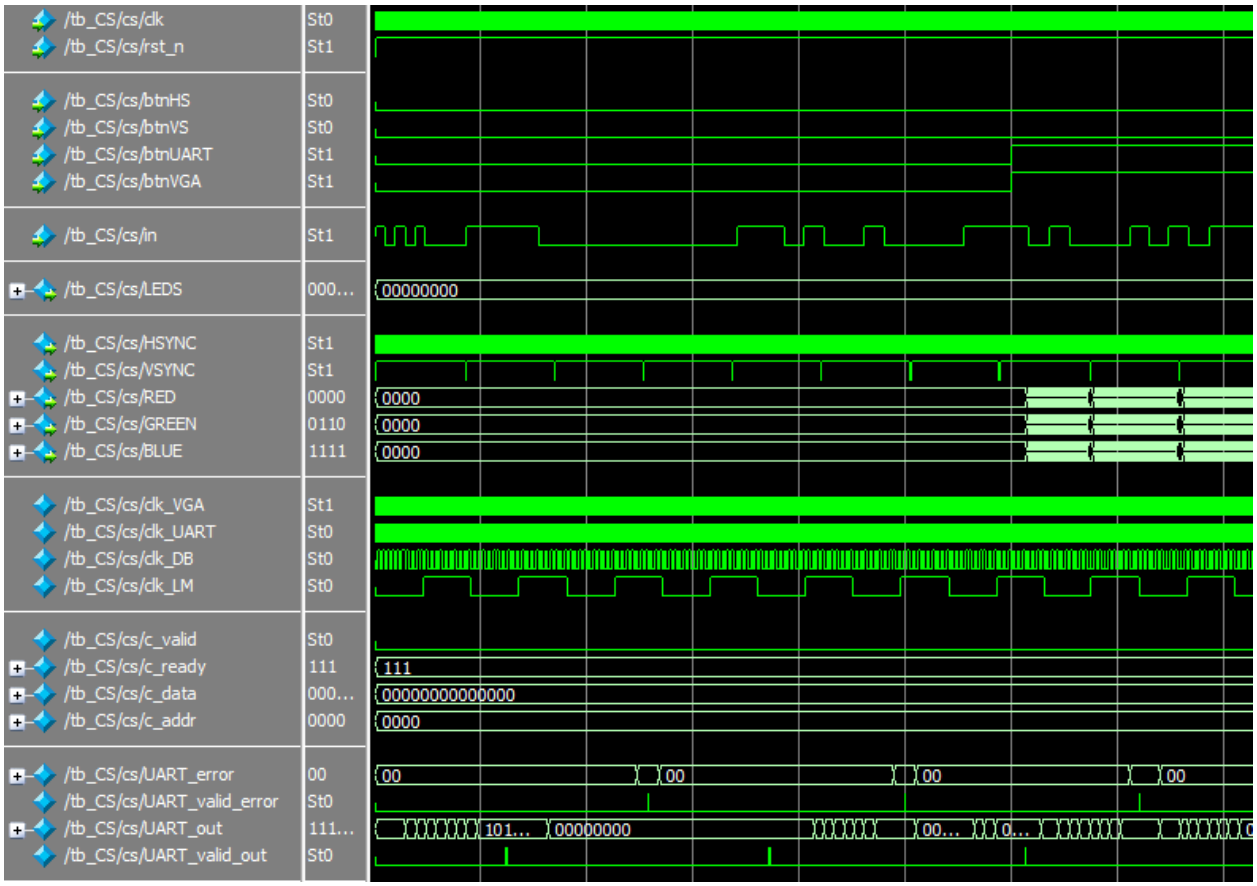

Figura 55. Waveform CS

În figurile precedente se poate observa funcționarea întreg sistemului creat. Sunt transmise constant informații pe interfața serială, care sunt apoi transpuse în comenzi pentru manager și trimise mai departe sub formă de culori spre VGA.

6.3. ASERȚII

În urma rulării testelor s-a confirmat corectitudinea codului celor 4 module. Pentru fiecare aserție începută există un mesaj de informare în transcript. În cazul unei erori se afișează mesajul de eroare și se contorizează eroarea la finalul testului UVM.

Modulul UART a fost singurul modul problematic. În cazul reconfigurărilor succesive, tranzacțiile începute erau oprite, dar în cazul în care exista un bit de 0 la intrare după reconfigurare, acesta era considerat bit de start

Aserțiile sunt afișate în fișierul transcript individual corespunzător testului rulat.

```
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA: ASSERTION 3: INACTIVE_DATA!
#    Time: 24075 ns Started: 24065 ns  Scope: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA File: testbench/VGA/assertion_VGA.sv Line: 55
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_1_POS_HS: ASSERTION 1: POS_HS!
#    Time: 25025 ns Started: 24065 ns  Scope: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_1_POS_HS File: testbench/VGA/assertion_VGA.sv Line: 49
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA: ASSERTION 3: INACTIVE_DATA!
#    Time: 25035 ns Started: 25025 ns  Scope: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA File: testbench/VGA/assertion_VGA.sv Line: 55
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_2_NEG_HS: ASSERTION 2: NEG_HS!
#    Time: 32075 ns Started: 25025 ns  Scope: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_2_NEG_HS File: testbench/VGA/assertion_VGA.sv Line: 52
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA: ASSERTION 3: INACTIVE_DATA!
#    Time: 32085 ns Started: 32075 ns  Scope: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_3_INACTIVE_DATA File: testbench/VGA/assertion_VGA.sv Line: 55
# ** Info: PASS ASSERTION: testbench_VGA.VGA_DUT.ass_VGA.ASSERTION_1_POS_HS: ASSERTION 1: POS_HS!
```

Figura 56. Informații aserții în transcript

7. CONCLUZII

7.1. CONCLUZII GENERALE

În concluzie, sistemul propus a fost proiectat, realizat și validat conform cerințelor. În realizarea proiectului s-au urmat pașii de construire a unui sistem digital, de la definirea cerințelor și specificațiior până la validarea sistemului în simulare și la implementarea acestuia pe placă. Ultimul pas de validare a sistemului fizic nu a fost atins.

Proiectarea sistemului folosind un limbaj de descriere hardware a asigurat existența unui design modular, ușor de extins. Specificațiile sistemului sunt bine documentate și ușor de înțeles. Validarea folosind metodologia UVM a ajutat la construirea unui mediu de verificare generalizat și specific în același timp, care poate fi folosit pentru verificarea individuală a modulelor refolosite în alte proiecte. Au fost testate cazurile limită și randomizate, pentru a asigura funcționarea robustă a sistemului.

7.2. PERSPECTIVE DE DEZVOLTARE

O posibilă direcție de dezvoltare a proiectului este folosirea modulelor într-un sistem de afișare a datelor provenite de la un microcontroler pe ecran. În prezent, pentru afișarea datelor de pe microcontrolere sunt folosite module precum LCD, 7 segmente sau OLED. Acestea au o dimensiune relativ mică și nu pot afișa multe informații. Dacă s-ar conecta direct un modul VGA, acesta ar ocupa mulți dintre pinii microncontrolerului. Sistemul prezentat ar rezolva această problemă, deoarece datele ar fi trimise prin UART și ar ocupa doar un pin. O altă variantă ar fi conectarea unor module Blutooth pentru a nu fi necesară conexiunea fizică dintre microcontroler și ecran.

O altă direcție de dezvoltare este reprezentată de folosirea sistemului ca aparat de depanare a ecranelor. În acest sens se poate verifica funcționarea pixelilor și capacitatea ecranului de a funcționa la anumite rezoluții sau frecvențe. De asemenea, la sistem se mai poate adăuga ultimul pin al conectorului VGA pentru a determina tipul ecranului.

O ultimă direcție de dezvoltare propusă este implementarea anumitor jocuri primitive pe FPGA. Majoritatea plăcilor de dezvoltarea au puse la dispoziție butoane și întrerupătoare, care permit controlul diferitelor acțiuni.

7.3. SINTEZA CONTRIBUȚIILOR

- proiectarea sistemului digital
- realizarea sistemului folosind un limbaj de descriere hardware
- proiectarea unui sistem de verificare și validare digitală
- realizarea sistemului urmând metodologia UVM
- conectarea și validarea sistemului digital folosind mediile de verificare
- implementarea sistemului pe FPGA

8. BIBLIOGRAFIE

1. P. Shrestha, A. Aversa, S. Phatharodom and I. Savidis, ”EDA-schema: A Graph Datamodel Schema and Open Dataset for Digital Design Automation”, Iunie 2024, pp. 69–77, GLSVLSI '24: Proceedings of the Great Lakes Symposium on VLSI 2024, ISBN: 9798400706059, https://doi.org/10.1145/3649476.3658718
2. N. Pham-Thai, B. Ho-Ngoc, T. Do-Duy, P. Q. Truong and V. C. Phan, ”A novel multichannel UART design with FPGA-based implementation”, Aprilie 2022, pp 358-369, ISSN: 0952-8091. https://doi.org/10.1504/IJCAT.2021.122350
3. P. Sharma, A. Kumar and N. Kumar, "Analysis of UART Communication Protocol," 2022 International Conference on Edge Computing and Applications (ICECAA), Tamilnadu, India, 2022, pp. 323-328, doi: 10.1109/ICECAA55415.2022.9936199.
4. A. K. Gupta, A. Raman, N. Kumar and R. Ranjan, "Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART)," 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2020, pp. 295-300, doi: 10.1109/SPIN48934.2020.9070856.
5. A. Chinchanikar, P. H. Chandankhede and A. Titarmare, "VGA Controller Design & Implementation on FPGA," 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2023, pp. 1-6, doi: 10.1109/GCAT59970.2023.10353298.
6. J. Bergeron, “Writing Testbenches Using SystemVerilog”, Springer, Library of Congress Control Number: 2005938214, ISBN-10: 0-387-29221-7.
7. https://www.chipverify.com/tutorials/uvm, accesat în 08.06.2024.
8. Vitankar, Pankaj & Kureshi, A.K.. (2016). UVM ARCHITECTURE FOR VERIFICATION. International Journal of Electronics and Communication Engineering & Technology. 7. pp. 29-37
9. J. Bergeron, “Writing Testbenches Using SystemVerilog”, ISBN-10: 0-387-29221-7
10. Clifford E. Cummings, SystemVerilog Assertions Design Tricks and SVA Bind Files, SNUG 2009, Sunburst Design World Class Verilog & SyystemVerilog Training, http://www.sunburst-design.com/papers/CummingsSNUG2009SJ_SVA_Bind.pdf
11. https://www.chipverify.com/uvm/uvm-phases, accesat în 08.06.2024.
12. https://verificationacademy.com/verification-methodology-reference/uvm/docs_1.1c/html/files/base/uvm_common_phases-svh.html, accesat în 08.06.2024.
13. https://digilent.com/blog/uart-explained/, accesat în 04.05.2024.
14. N. Lal, K. Pandey and M. Sharma, "Design and Implementation of a VGA Controller Using Complex Programmable Logic Devices," 2018 International Conference On Advances în Communication and Computing Technology (ICACCT), Sangamner, India, 2018, pp. 633-636, doi: 10.1109/ICACCT.2018.8529621.
15. Nexys-4 FPGA: Technical report, September 2014. https://digilent.com/reference/_media/reference/programmable-logic/nexys-4-ddr/nexys4ddr_rm.pdf
16. Digilent. Nexys A7™ FPGA Board Reference Manual, Octombrie 2019. https://digilent.com/reference/nexys_vga/refmanual

DECLARAŢIE DE AUTENTICITATE A LUCRĂRII DE FINALIZARE A STUDIILOR*

Subsemnatul VERES DENISA-ALEXANDRA,
legitimat cu C.I seria TZ nr. 572877,
CNP 6020125250030
autorul lucrării sistem digital de vizualizare prin intermediul VGA folosind comunicarea prin interfață serială
elaborată în vederea susţinerii examenului de finalizare a studiilor de LICENȚĂ organizat de către Facultatea DE AUTOMATICĂ ȘI CALCULATOARE din cadrul Universităţii Politehnica Timişoara, sesiunea IUNIE 2024 a anului universitar 2023 - 2024, coordonator Conf.dr.ing. ALEXANDRU AMĂRICĂI-BONCALO luând în considerare art. 34 din *Regulamentul privind organizarea și desfășurarea examenelor de licență/diplomă și disertație*, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în cazul constatării ulterioare a unor declaraţii false, voi suporta sancțiunea administrativă prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale;
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului;
- sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor;
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație;
- În elaborarea lucrării ~~am utilizat~~ instrumente specifice inteligenței artificiale (IA) și anume _____ (denumirea) _____ (sursa), pe care le-am citat în conținutul lucrării/nu am utilizat instrumente specifice inteligenței artificiale (IA)[1].

Declar că sunt de acord ca lucrarea să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Timişoara,

Data	Semnătura
23.06.2024	

*Declaraţia se completează de student, se semnează olograf de acesta și se inserează în lucrarea de finalizare a studiilor, la sfârșitul lucrării, ca parte integrantă.

[1] Se va păstra una dintre variante: 1 - s-a utilizat IA și se menționează sursa 2 – nu s-a utilizat IA