Universitatea Politehnica Timișoara
Facultatea de Automatică și Calculatoare
Secția de Calculatoare și Tehnologia Informației
Anul universitar 2023-2024

PROIECTAREA, REALIZAREA ȘI VALIDAREA UNUI SISTEM DIGITAL PENTRU AFIȘAREA PRIN VGA A DATELOR PRIMITE PE UART

LUCRAREA DE DIPLOMĂ

Candidat: Vereș Denisa-Alexandra

Profesor coordonator: Conf.dr.ing. Alexandru Amăricăi-Boncalo

Sesiunea: Iunie 2024

CUPRINS

*** pune diacritice

LISTA FIGURILOR

LISTA TABELELOR

1. INTRODUCERE

Scopul acestui proiect este proiectarea, realizarea și validarea unui sistem digital .

Obiectivele propuse sunt:

- realizarea unor module standardizate reconfigurabile și refolosibile pentru protocoalele UART și VGA;
- proiectarea unui
- realizarea unui mediu de verificare modular, ușor de întreținut și modificat, care să valideze atât protocoalele standard, cât și sistemul întreg.

Pentru realizarea proiectului se vor urma etapele de documentare, design, implementare, testare și se va finaliza prin testarea pe placă.

Aplicatii folosite în realizarea acestui proiect:

- ModelSim & QuestaSim: compilare si simulare
- Visual Studio Code: editare cod
- Wavedrom: editare modele ideale
- EasyEda: editare design
- Vivado/Diamond: implementare placa
- Microsoft Excel: verificare
- Microsoft Word: editare text
- Microsoft PowerPoint: editare prezentare

*** fa o introducere a proiectului – descrie pe scurt ce face (nu uita de modurile de debug)

*** refa golden model pentru VGA

!!!!!!!!!!!!! Verifica bitul de paritate !!!!!!!!!!!!!!!

2. STUDIU BIBLIOGRAFIC

Designul digital 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 transmiterului 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, ”receiverul UART are un generator de baud rat, 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. PAȘII DE REALIZARE A UNUI DESIGN DIGITAL

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

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.
Designul topului: În această etapă se realizarea 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 și scoreboard pentru a putea valida sistemul.
Designul modulelor: În această etapă se realizarea 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.
Designul 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.

Designul 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 niciun alt component.
- Redundanța: Părțile critice ale sistemului trebuie să fie dublate sau realizate în mai multe moduri pentru a se adigura 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 funcționalitate presupusă a unei componente. Acest golden model este folosit pentru respectarea scoreboardului 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 [1]”;
  - 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, spre deosebire de predecesorii săi.
- 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 componenta 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 componente 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 importantă. 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.

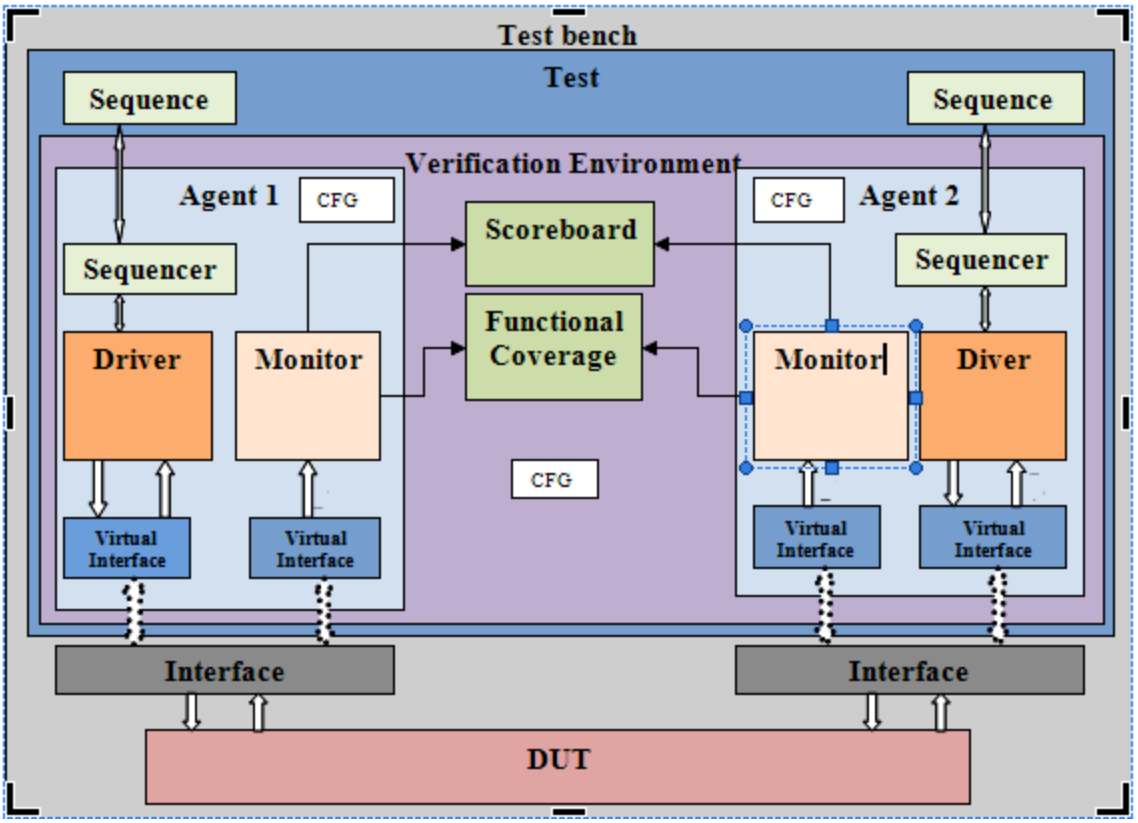

Figura 1. Componentele unui testbench conform UVM (sursa: Vitankar, Pankaj & Kureshi, A.K.. (2016). UVM ARCHITECTURE FOR VERIFICATION. International Journal of Electronics and Communication Engineering & Technology. 7. pp. 29-37)

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

În mediile de testare UVM sunt folosiți 3 tipuri diferiți 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 grupuri de acoperire (covergroups) bine definite, care la rândul lor conțin numeroase puncte de acoperire (coverpoints) și intersecții (cross) 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.

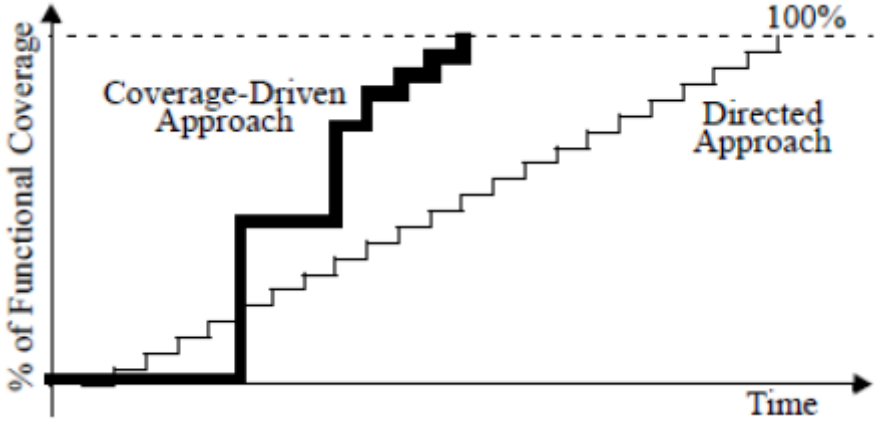

Figura 2. Progresul unui testbench bazat pe coverage (sursa: Janick Bergeron, “Writing Testbenches Using SystemVerilog”, ISBN-10: 0-387-29221-7)

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, folosind logică.

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 [8]”. Acestea sunt „monitoare aflate într-un bloc de cod care probează periodic și testează semnalele și generează mesaje de eroare dacă aserția eșuează [8]”.

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

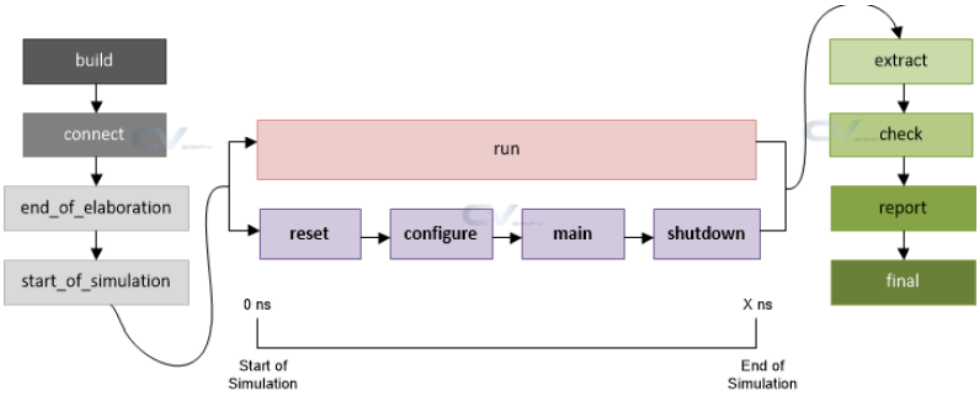

Figura 3. Fazele UVM (sursa: https://www.chipverify.com/uvm/uvm-phases)

O componentă are toate fazele implementate, dar programatorul alege dacă acestea trebuie rescrise 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 [9]”;
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 [9] ”. Această fază se poate termina în 2 feluri: ▪ toate obiecțiile sunt coborâte; ▪ expiră timpul de rulare (”Timpul de timeout default este 9200s [9] ”).
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 fist 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_HIGH	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 biți, 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.

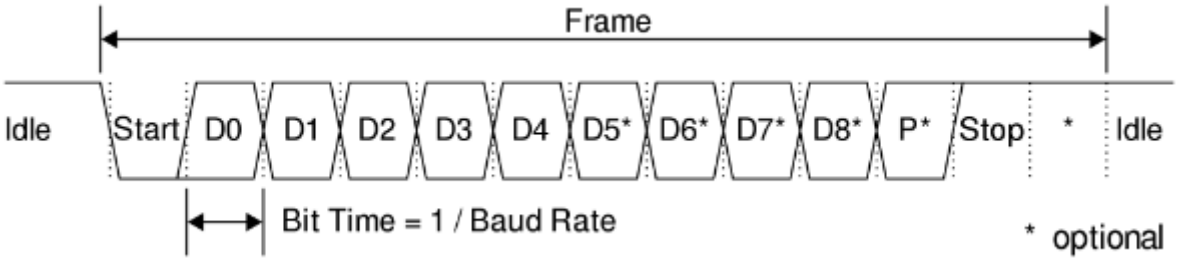

Figura 4. Structura unui frame UART (sursa: https://digilent.com/blog/uart-explained/)

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 BaudRate, care reprezintă frecvența unui bit. Dispozitivele care comunică între ele trebuie să aibă același baud rate setat și să aibă aceiași 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 [10].”

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

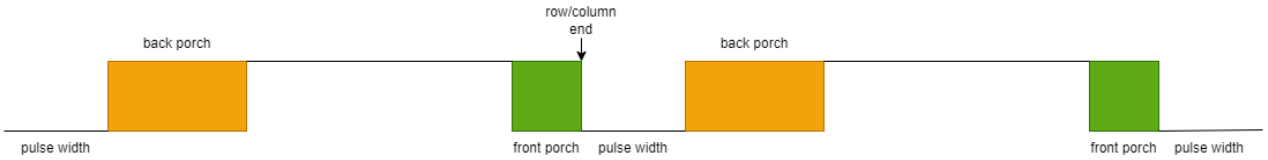

Figura 5. Explicație SYNC

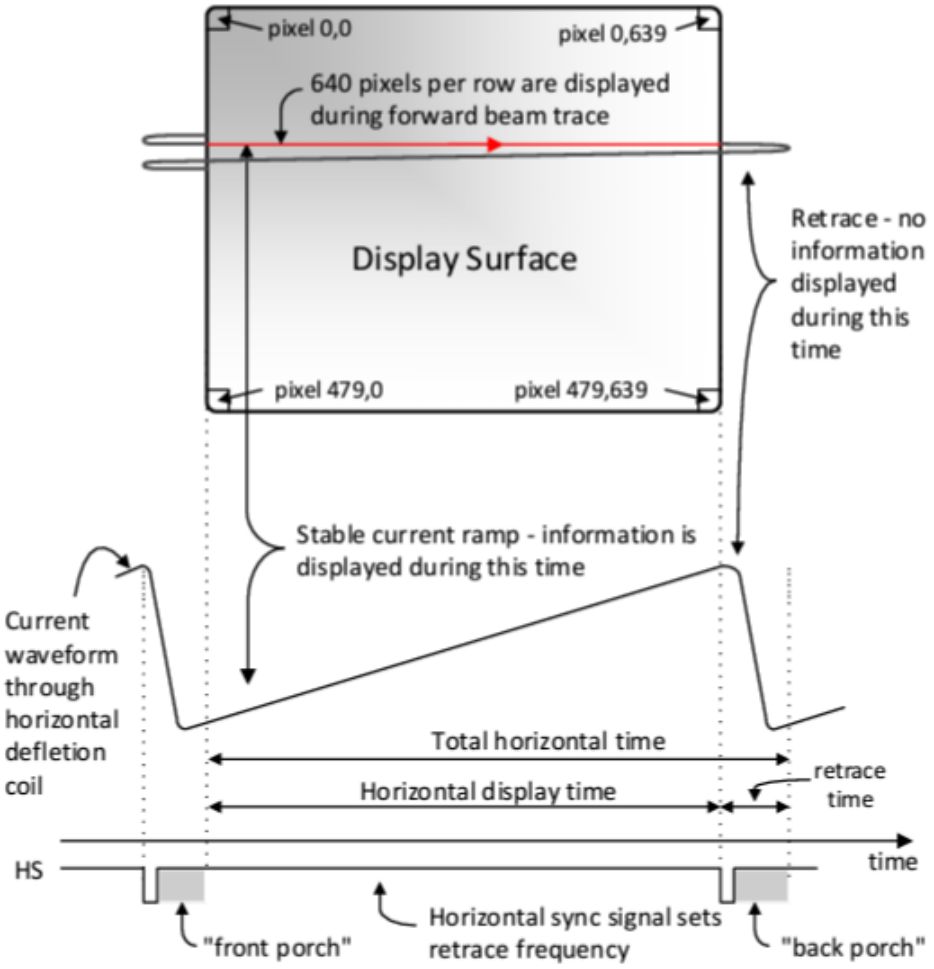

Figura 6. Detalii ecran (sursa: Xilinx. Nexys-4 DDR FPGA: Technical report, September 2014)

Datele indispensabile de care are nevoie standardul VGA pentru a funcționa sunt cele 3 culori de bază (roșu, verde și albastru), fiecare pe 4 biți și semnalele de control VSYNC și HSYNC. Pinul rămas poate fi folosit pentru a detecta tipul monitorului. Cei 12 bi'i de culoare permit afi;area a 4096 de culori diferite.

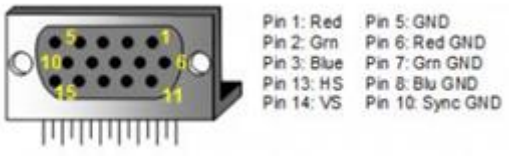

Figura 7. Conector VGA (sursa: Nexys A7™ FPGA Board Reference Manual, Octombrie 2019)

Actual, deși a trecut mult timp de la standardizarea protocolului VGA, acesta este și va rămâne un standard foarte răspândit datorită simplității și compatibilității lui. "Ecranele moderne cu VGA acceptă diferite rezoluții și un controler VGA dictează rezoluția producând semnale de sincronizare [11]".

Tabelul 4. Parametrii rezoluție (sursa: https://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml)

Format	Horizontal (in Pixels)				Vertical (in Lines)			
	Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	640	16	96	48	480	11	2	31
800x600, 60Hz	800	40	128	88	600	1	4	23
1024x768, 60Hz	1024	24	136	160	768	3	6	29

4. SOLUȚIA PROPUSĂ

4.1. DESIGN

4.1.1. DEFINIREA CERINȚELOR

Acest proiect propune realizarea unui sistem de verificare și depanare a modulelor de comunicare UART și ale ecranelor cu port VGA.

4.1.2. 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.

Rezultatul sistemului în urma interacțiunilor:

1. Pe LED-uri se vor afișa:
    a. erori
    b. configurația actuală
    c. datele de la UART
2. Pe ecran se vor afișa culori în diferite configurații.

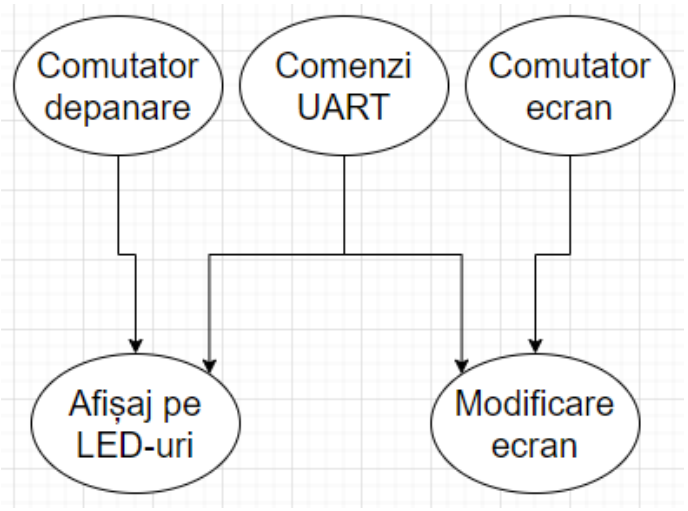

Figura 8. Diagrama simplificată a parcurgerii sistemului

4.1.3. DESIGNUL TOPULUI

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

Pinii din tabelul următor sunt pinii plăcii Zedboard.

Tabelul 5. Intrările/Ieșirile sistemului

Nume	Tip	Mărime [bit]	Descriere	PINI
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.	
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

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, UART și LED-uri.

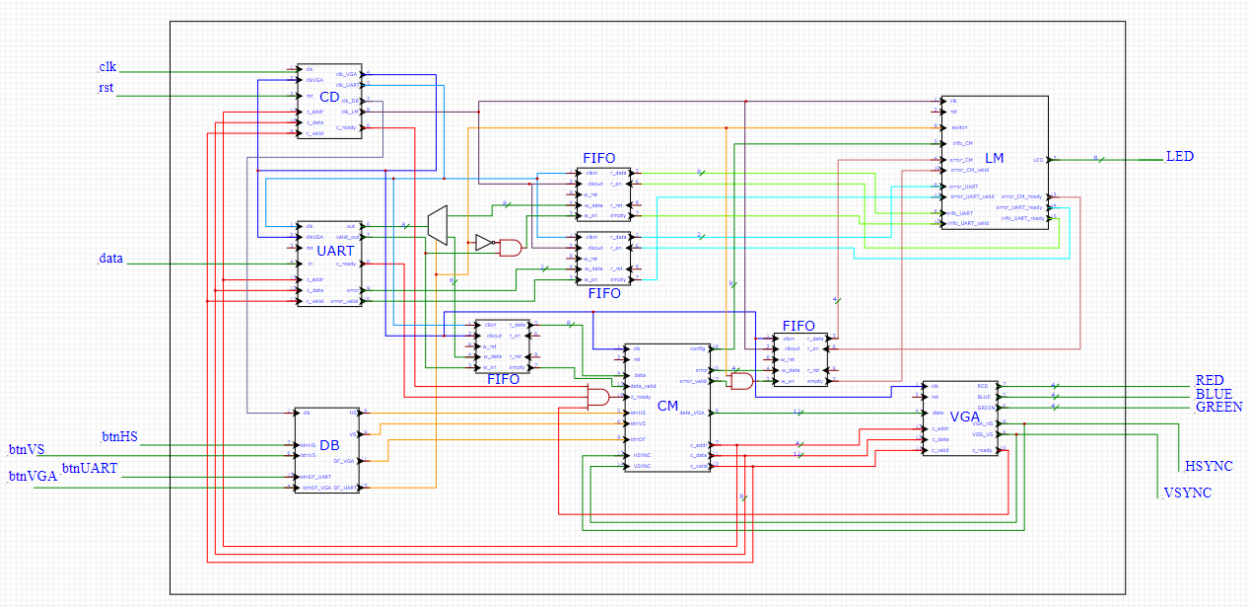

Figura 9. Top Module

4.1.4. DESIGNUL MODULELOR

4.1.4.1. CLOCK DIVIDER (CD)

Tabelul 6. Intrările/leșirile 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		

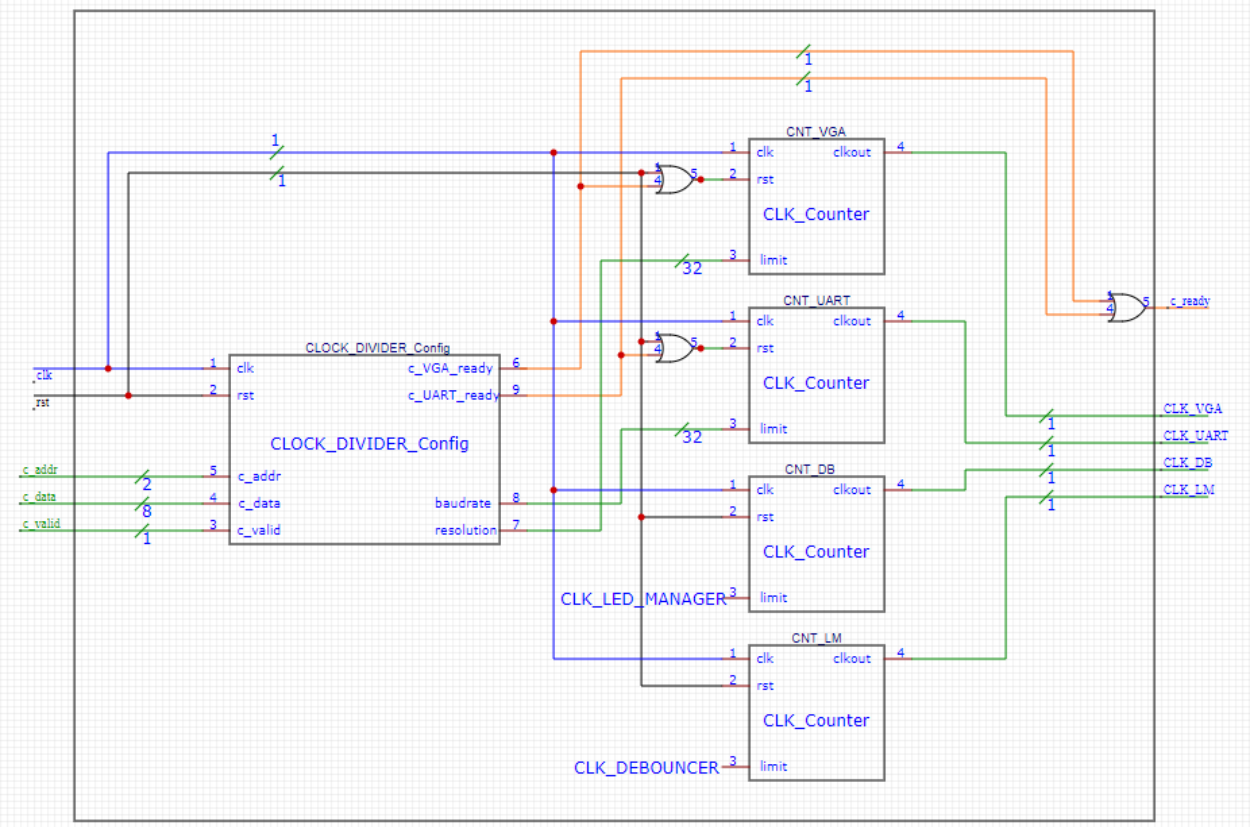

Figura 10. Modulul Clock Divider

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

Ieșirile:

- ✓ clkVGA – frecvența: 25Mhz – 65MHz
  - o în funcție de rezoluția modulului VGA, frecvența este:
    - 640x480 – 25MHz
    - 800x600 – 40MHz
    - 1024x768 – 65MHz
  - o frecvența inițială este cea corespunzatoare rezoluției de 640x480, adică 25MHz.
- ✓ clkUART – frecvența: 38.4kHz – 921.6kHz
  - o 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
  - o frecvența initială este cea corespunzatoare unui baudrate de 9600, adică 9.6kHz
- ✓ clkCM – frecvența: 25MHz – 65MHz
  - o în funcție de rezoluția modulului VGA, specificată la punctul anterior
- ✓ clkDEB – frecvența 20Hz
- ✓ clkLED – frecvența 1HZ

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

- CNT_VGA – limitele conform tabelului din 4.1.5.1.
- CNT_UART – limitele conform tabelului din 4.1.5.1.
- CNT_DB – limita este CLK_DEBOUNCER
- CNT_LM – limita este CLK_LED_MANAGER

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 urmatoarea 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 și la modificarea limitei numărătorului CNT_UART
- Rezoluție – duce la resetarea semnalului clock_VGA și la modificarea limitei numărătorului CNT_VGA

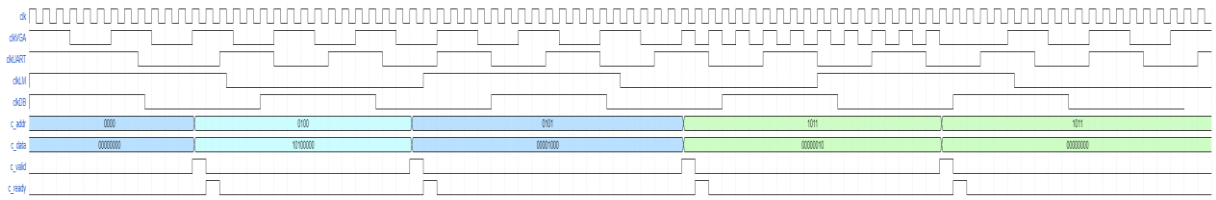

Figura 11. Golden model pentru reconfigurare

4.1.4.2. 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		

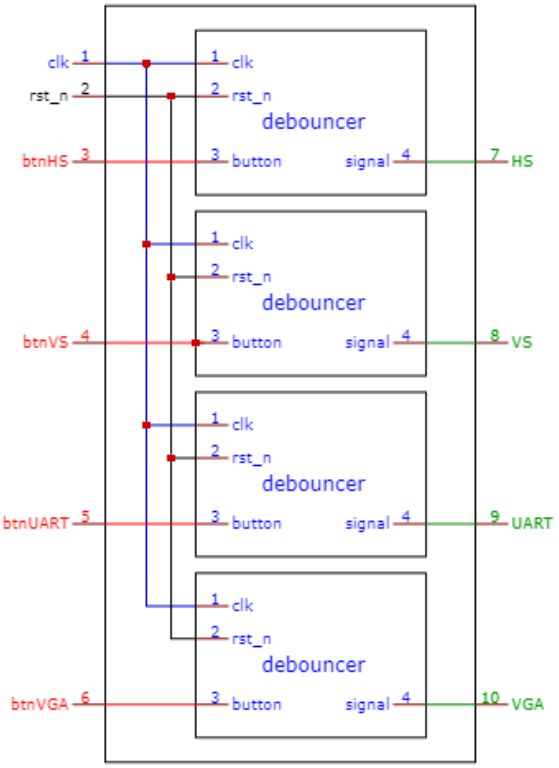

Figura 12. Modulul Debouncers

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

4.1.4.3. 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		

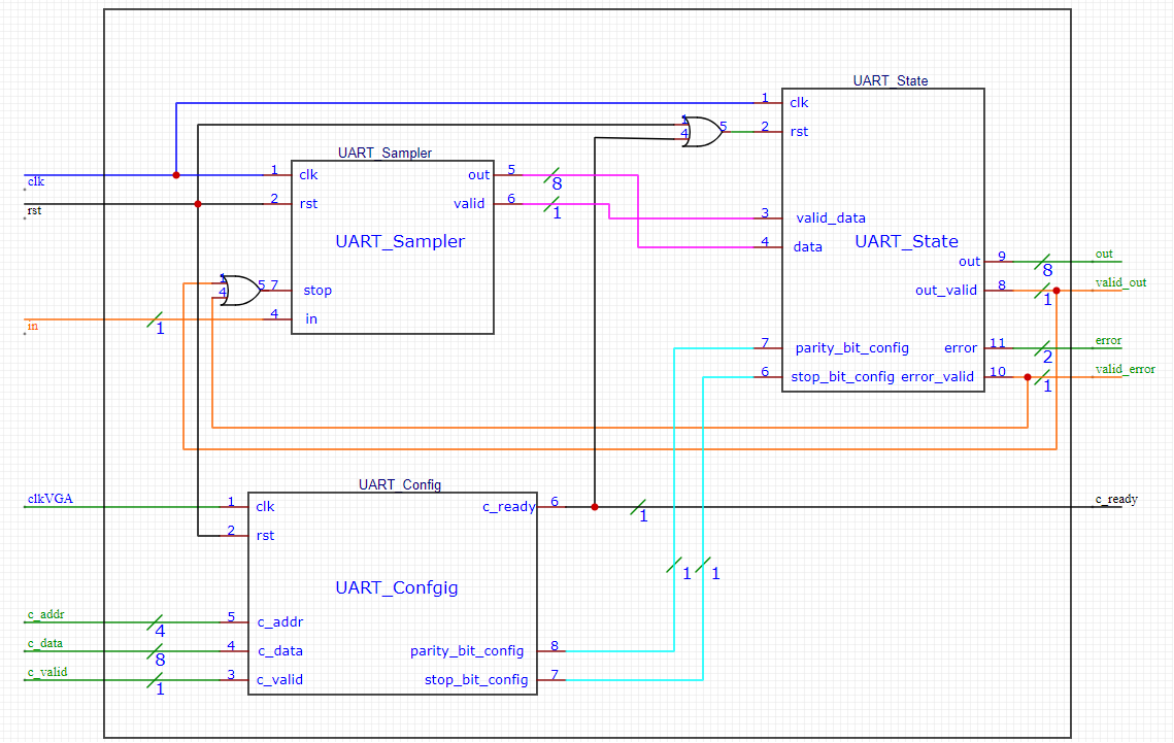

Figura 13. Modulul UART

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 si 8 biți sunt codificate conform următorului tabel;
- c_ready este reactivat după reconfigurare
- după configurare se resetează valorile și orice transmisie în curs de efecuare este întreruptă

Tabelul 9. Codificarea parametrilor configurabili ai modulului UART

c_addr	COD	c_data[1:0]	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 urmatoarele diagrame se poate observa o comunicare reușită și o comunicare nereușită datorită nerespectării protocolului UART pentru bit-ul de STOP.

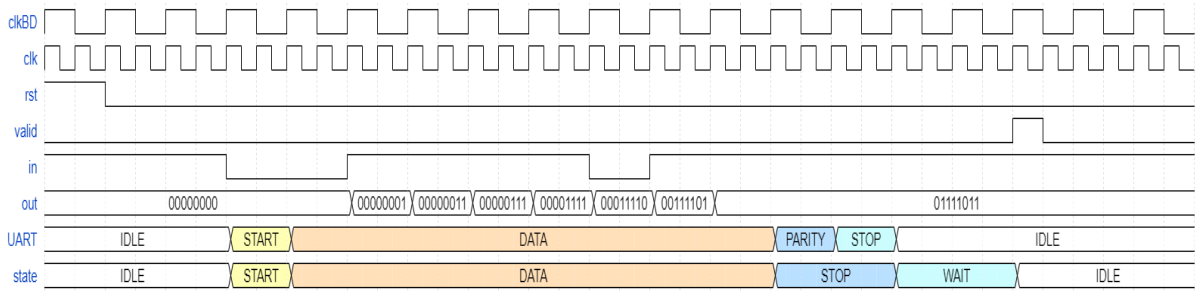

Figura 14. Golden model pentru o comunicare validă prin UART

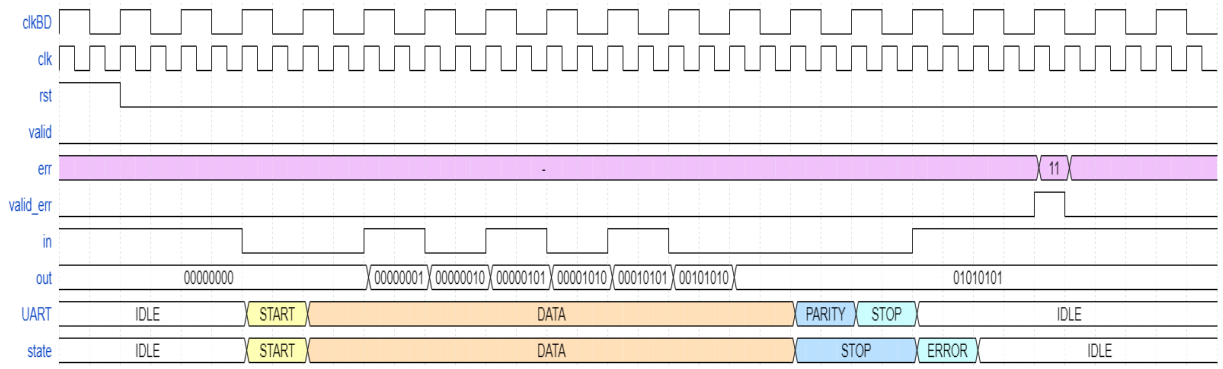

Figura 15. Golden model pentru o comunicare invalidă prin UART

Erorile trimise de modulul UART au următoarea codificare:

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

4.1.4.4. 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		

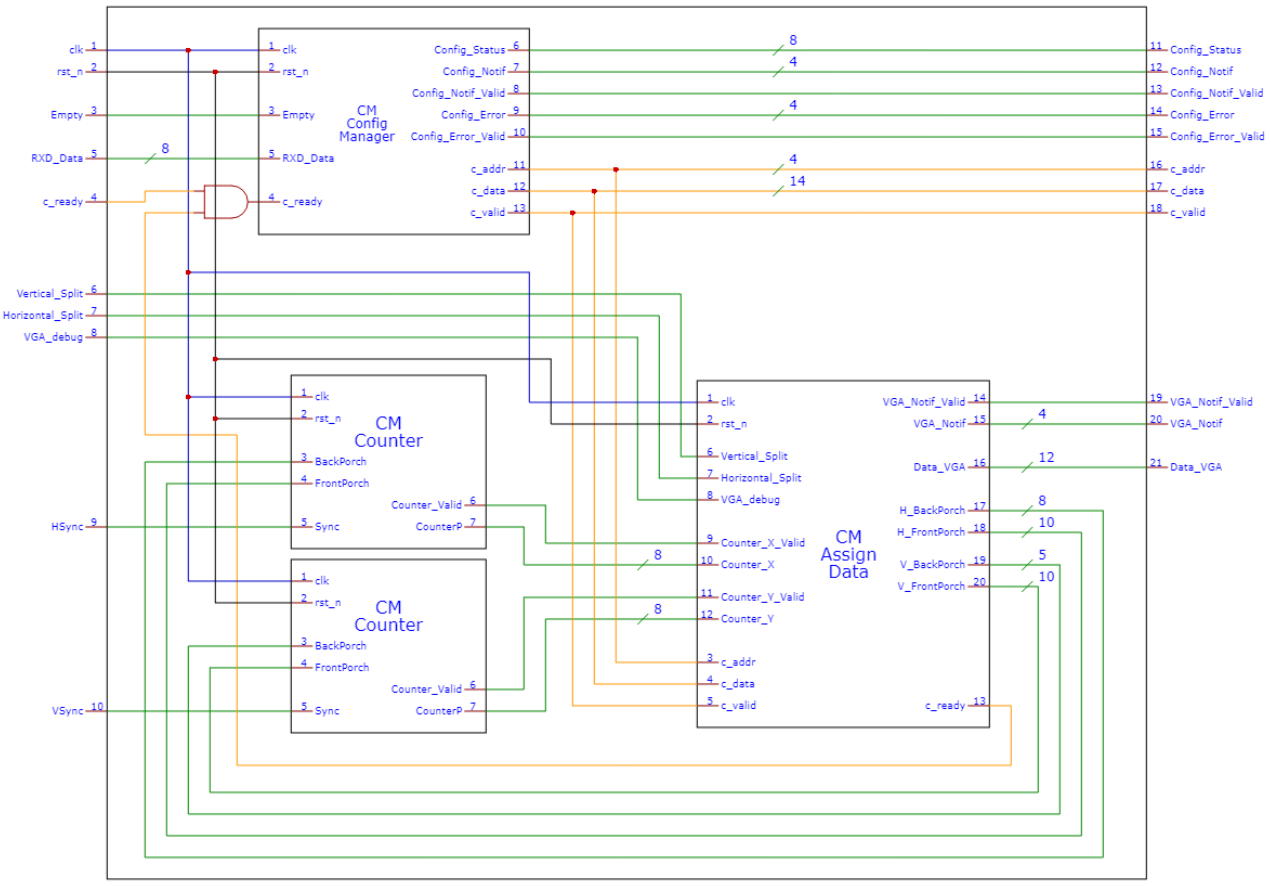

Figura 16. Modulul Color Manager

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.

Tabelul 11. Descrierea cuvântului de comandă

Pozitie bit	Descriere Configuratie	Configuratie	Descriere Culoare	Culoare	Pozitie bit
15	Selectie Configuratie/ Culoare	1	Selectie Configuratie/Culoare	0	15
14	Adresa unitate configurabila 01 UART 10 VGA	01/10	-	0	14
13			Arata pozitia cadranului sus(0)/jos(1)	0/1	13
12	Adresa registru din unitate *tabel registru	x	Locatia cadranului: stanga(0) /dreapta(1)	0/1	12
11		x	Codul culorii convertit din hexazecimal Site pentru aflarea codului	x	11
10	Codificare configuratie *tabel config	x		x	10
9		x		x	9
8		x		x	8
7	Biti nefolisiti	0		x	7
6		0		x	6
5		0		x	5
4		0		x	4
3		0		x	3
2		0		x	2
1		0		x	1
0		0		x	0

Tabelul 12. Codificarea modulelor configurabile

Table registru			
Unitate	Valoare		Ce face?
UART	0	0	Configurare BoudRate
	0	1	Configurare Parity
	1	0	Configurare Bit Stop
VGA	0	0	Culoare pentru VGA prin UART
	1	0	Configurare cadran nou VGA
	1	1	Configurare rezolutie pentru VGA

Tabelul 13. Codificarea configurărilor

Tabel configuratii				
Ce face?	Valori			Valori codificate
Configurare BoudRate	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 Parity	X	0	0	no parity
	X	1	1	odd
	X	1	0	even
Configurare Bit Stop	X	x	0	1 stop bits
	X	x	1	2 stop bits
Configurare rezolutie pentru VGA	X	0	0	640x480 default
	X	0	1	800x600
	X	1	0	1024x768
Configurare cadran nou VGA	X	0	1	vertical split
	X	1	0	horizontal split
	X	1	1	vertical and orizontal split
	*only when split switches are disabled			

4.1.4.5. 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		

Acest modul comandă ieșirile pentru portul VGA.

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

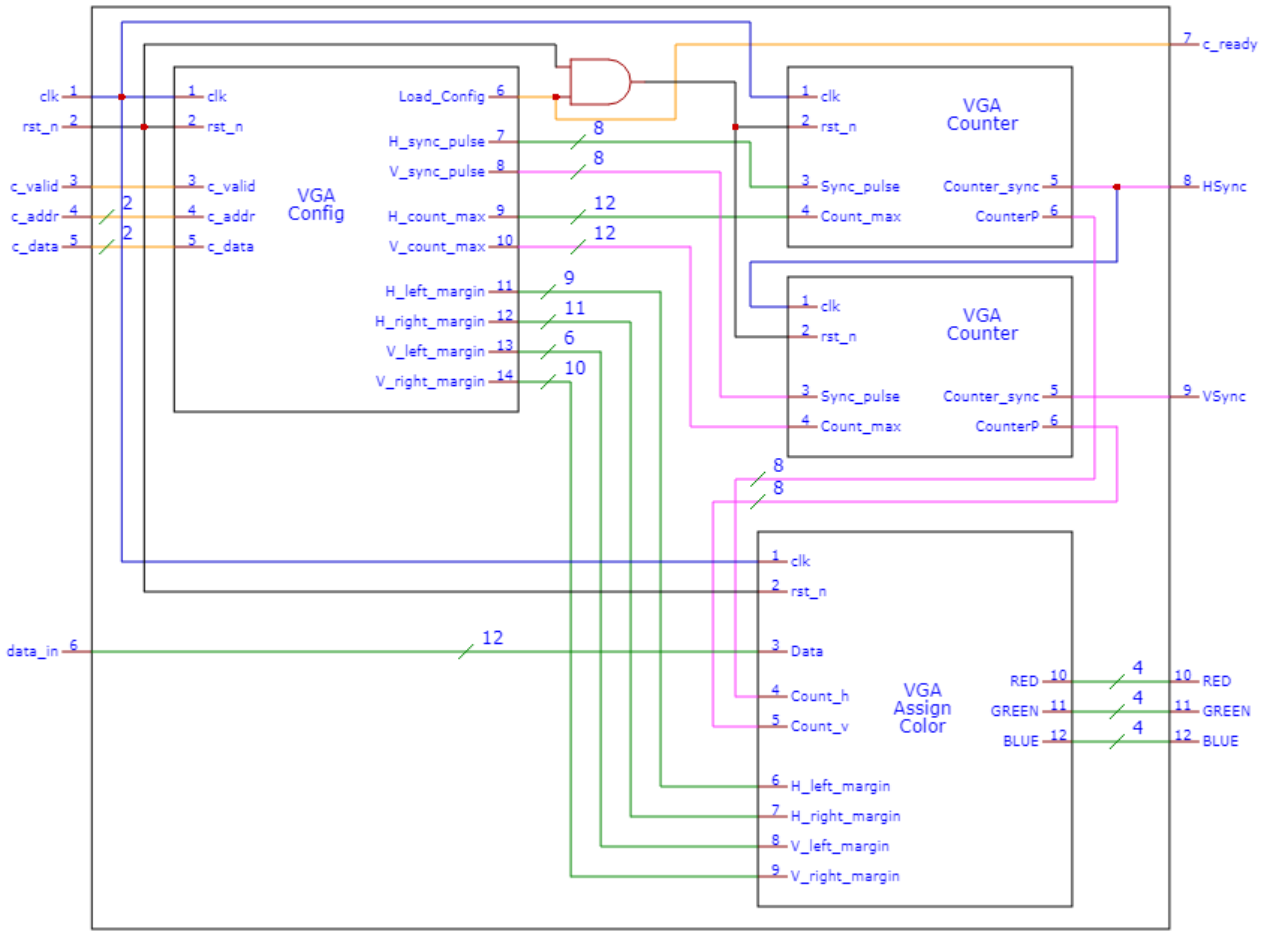

Figura 17. Modulul VGA

Se pot alege dintre rezoluțiile:

- 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 valorile calculate. 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 Load_Config.

4.1.4.6. 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.

Modulul 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.

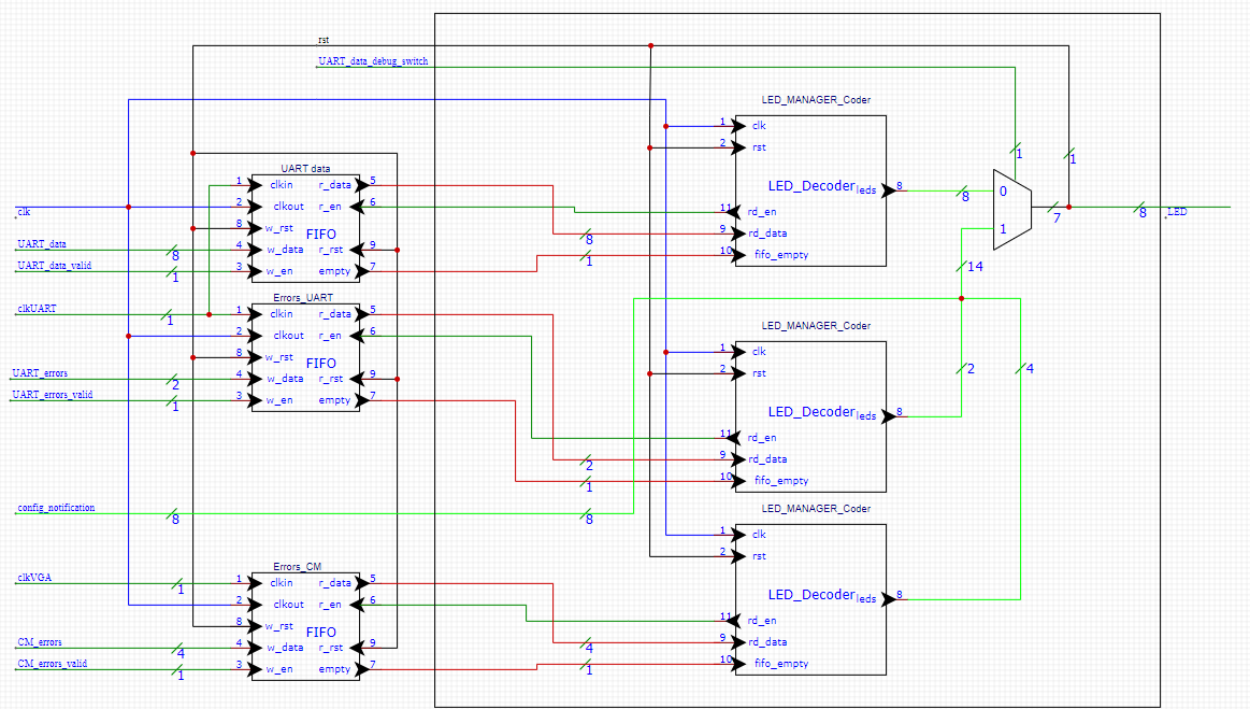

Figura 18. Modulul Led Manager

Pentru cazul în care UART_data_debug_switch nu este activ, LED-urile vor fi aprinse conform următorului tabel. LED-urile folosite pentru afisarea notificarilor si a erorilor, se vor aprinde timp de 1 secunda, dupa care se vor stinge.

Tabelul 16. Formatul LED-urilor în funcționarea normală

LED	VALOARE
U14	RESET (1)
U19	UART_data_debug_switch (1)
W22 V22	EROARE UART
U21 U22 T21 T22	EROARE CM

Pentru cazul în care UART_data_debug_switch este activ, LED-urile vor fi aprinse conform următorului tabel. LED-urile corespunzatoare valorii UART_data se vor aprinde timp de 1 secunda, după care se vor stinge.

Tabelul 17. Formatul LED-urilor în modul de depanare

LED	VALOARE
U14	UART_data[7]
U19	UART_data[6]
W22	UART_data[5]
V22	UART_data[4]
U21	UART_data[3]
U22	UART_data[2]
T21	UART_data[1]
T22	UART_data[0]

Tabelul 18. Datele provenite de la modulul UART

EROARE	CODIFICARE
NO_ERRORS	00
FAILED_STOP_BITS	11
FAILED_PARITY_BIT	01
FAILED_IDLE_BITS	10

Tabelul 19. Datele provenite de la modulul CM

EROARE	CODIFICARE
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

4.1.5. DESIGNUL UNITĂȚILOR

4.1.5.1. Clock Divider

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 20. Valorile configurabile ale clock dividerului

Adresă	Cod	Data	Cod	Limită	Valoare
UART BAUDRATE ADDR	0100	BAUDRATE_2400	000	CLK_BAUDRATE 2400	5859
		BAUDRATE_4800	001	CLK_BAUDRATE 4800	2930
		BAUDRATE_9600	010	CLK_BAUDRATE 9600	1465
		BAUDRATE_19200	011	CLK_BAUDRATE 19200	732
		BAUDRATE_57600	100	CLK_BAUDRATE 57600	244
		BAUDRATE_112000	101	CLK_BAUDRATE 112000	122
VGA RESOLUTION ADDR	1000	VGA_640x480	x00	CLK_VGA 640x480	9
		VGA_800x600	x01	CLK_VGA 800x600	6
		VGA_1024x768	x10	CLK_VGA 1024x768	3

Tabelul 21. Valorile neconfigurabile ale clock divider-ului

limit	value
clk_DB	11 250 000
clk_LM	225 000 000

4.1.5.2. Debouncer

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.

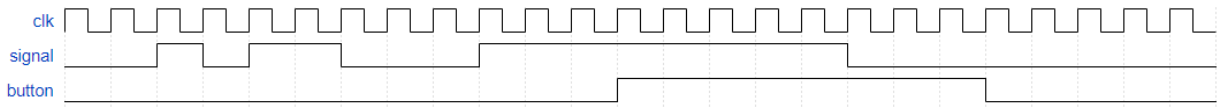

Figura 19. Golden model pentru debouncer

4.1.5.3. UART

Modulul UART este format din 3 module independente:

- UART_Sampler: se ocupa de esantionarea semnalului (un counter care numara cati biti de 0 si 1 sunt esantionati în interiorul semnalului si un comparator care stabileste bit-ul de iesire)
  - Pentru a evita situatia de sincronizare gresita a modulului UART, în care este posibil sa se transmita date gresite si sa ramana nesincronizat, bitii primiti sunt esantionati.
  - Se iau cate 16 esantioane pentru fiecare bit primit, din care primele si ultimele 3 esantioane sunt ignorate, iar din celelalte esantioane se calculeaza valoarea de iesire.

- UART_Config: La primirea unei noi configuratii modulul isi actualizeaza registrele cu valorile parametrilor modificati si reseteaza modulului UART_State pentru a intrerupe orice comunicatie neincheiata

- UART_State: Verifica daca informatia primita respecta protocolul UART:
  - Primul bit esantionat reprezinta bit-ul de START, care trebuie sa fie 0. În caz contrar se considera ca a fost un start fals si se reincepe transmisia.

- Urmatorii 8 biti esantionati reprezinta bitii de DATE, care pot lua valoare de 0 sau 1.
- În cazul în care paritatea este activata, urmatorul bit reprezinta bit-ul de PARITY, care este verificat. În cazul în care paritatea nu este valida, se semnaleaza eroarea, se considera ca datele transmise au fost corupte si se reincepe transmisia.
- Urmatorul bit/biti reprezinta bit-ul/bitii de STOP, care trebuie sa fie pe 1. În caz contrar se semnaleaza eroarea, se considera ca datele transmise nu au fost esantionate corect si se reincepe transmisia.
- Daca toti bitii verificati (START, PARITY, STOP) sunt valizi, atunci counicaea a avut succes.

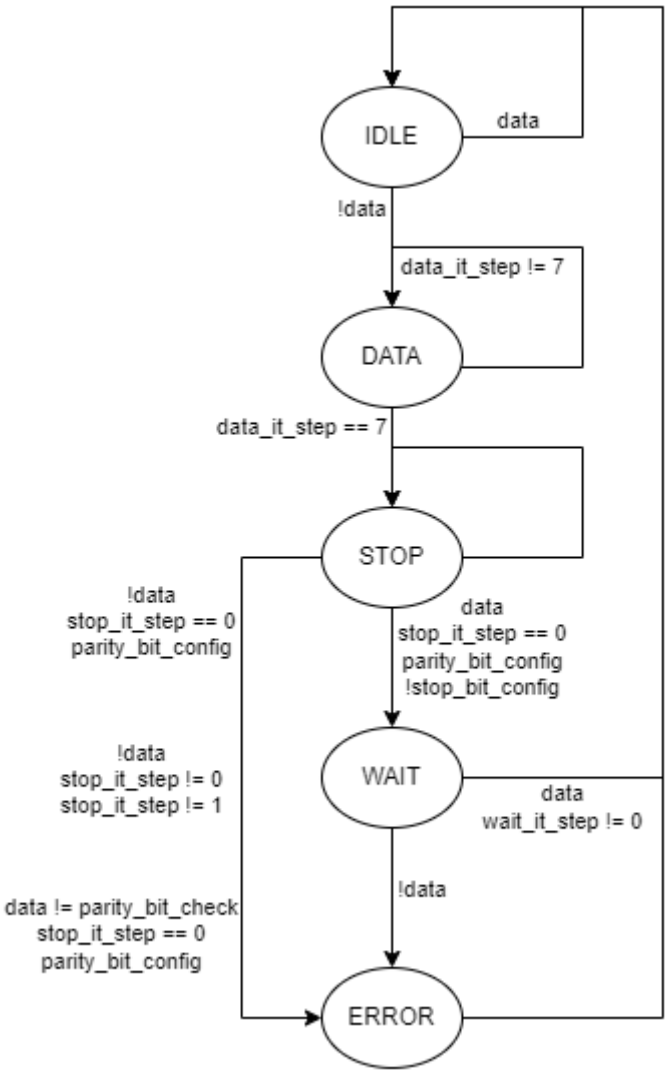

Figura 20. Diagrama de stări finite a modulului UART

După cum se poate observa în diagramă, numărul biților de date este predefinit și nu poate fi modificat. Așadar, modulul UART acceptă doar 8 biți de date.

4.1.5.4. Color Manager

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

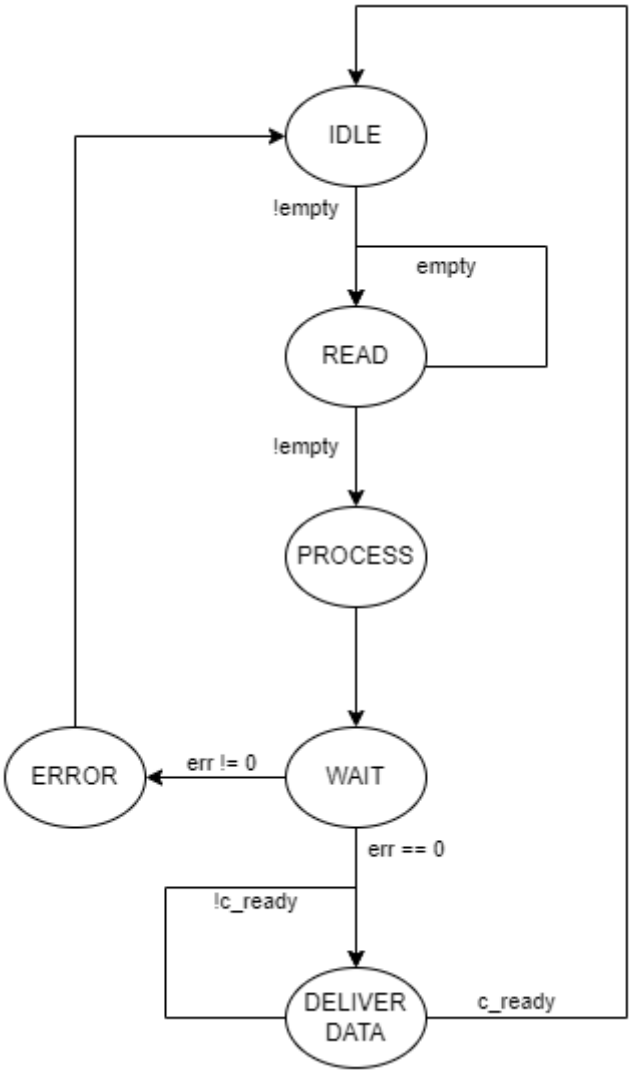

Figura 21. Diagrama de stări finite a managerului de configurații

La intrare acesta primește pe rând date de la modulul UART. O tranzacție completă este formată din 2 frame-uri de UART.

Tabelul 22. Format primit de la UART

Pozitie	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 tot prin C_Addr, adică 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.

În cazul unui cadran:

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

Tabelul 23. Formatul statusului de configurație

Pozitia bitilor	**7 6 5**	**4 3**	**2**	**1 0**
Continut	Boudrate	Parity	Stop Bit	VGA Rezolution

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 respective.

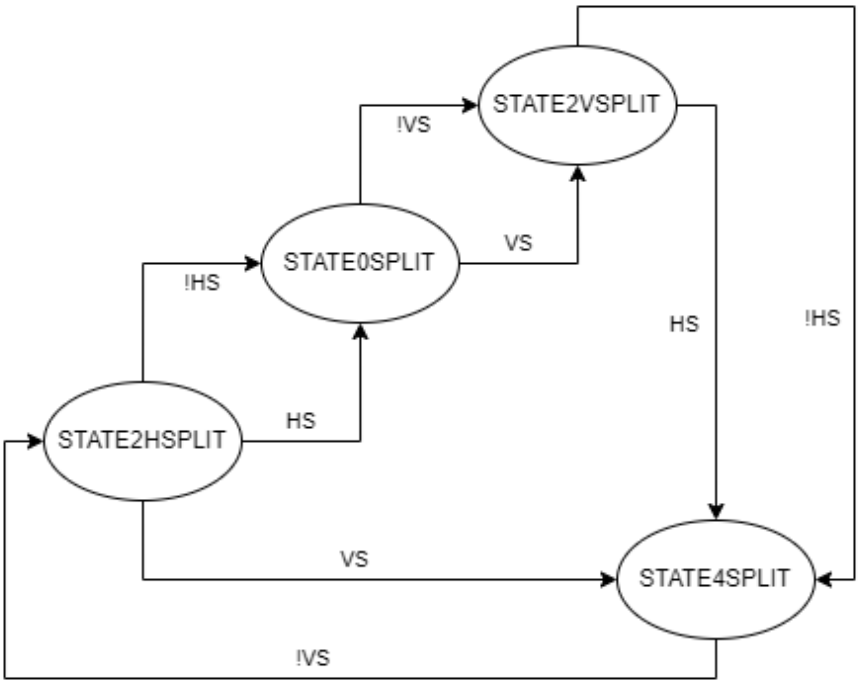

Figura 22. Diagrama de stări finite a managerului de cadrane

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 primeste o configurație nouă, aceasta intră în viguare instant, din punctul de vedere al unui observator uman.

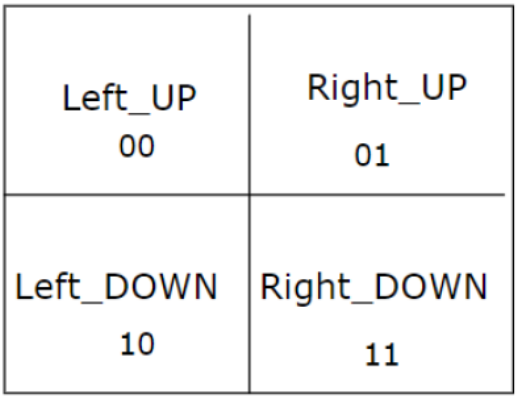

Figura 23. Poziția cadranelor

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

4.1.5.5. 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.

4.1.5.6. Led Manager

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

4.2. VERIFICARE

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.

4.2.1. 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 testate în câteva cazuri predefinite, pentru că acesta nu ține de niciunul dintre modulele principale și este o singură unitate individuală.

4.2.2. 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.

4.2.2.1. 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 receptarea 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 refolosibile și ar fi trebuit craete 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 24. 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 începe faza principală înainte ca resetarea DUT-ului să fie finalizată.

4.2.2.2. 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.

4.2.2.5. 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 sequencers și de posibilitatea agentului de a fi activ/reactiv.

Toate acțiunile întreprinse de agent au loc în timpul fazelor UVM predefinite.

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 predefinit care primește ca parametru tipul obiectului sau un sequencer definit (în cazul agenților reactivi);
- 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.

4.2.2.6. 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_phase: 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_phase: Componenta de coverage este conectată la monitoarele corecpunzătoare și în cazul agenților reactivi, sequencerul agentului reactiv este conectat la sequencerul virtual.

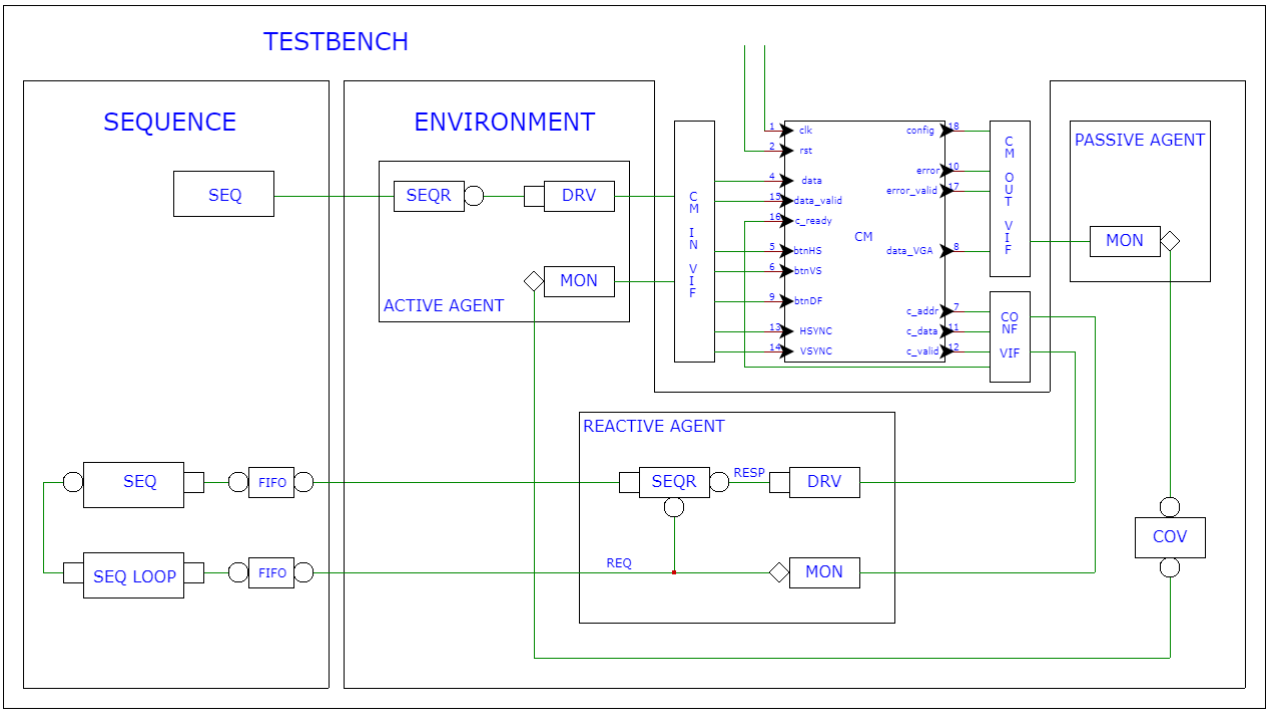

Figura 22. Design verificare Color Manager

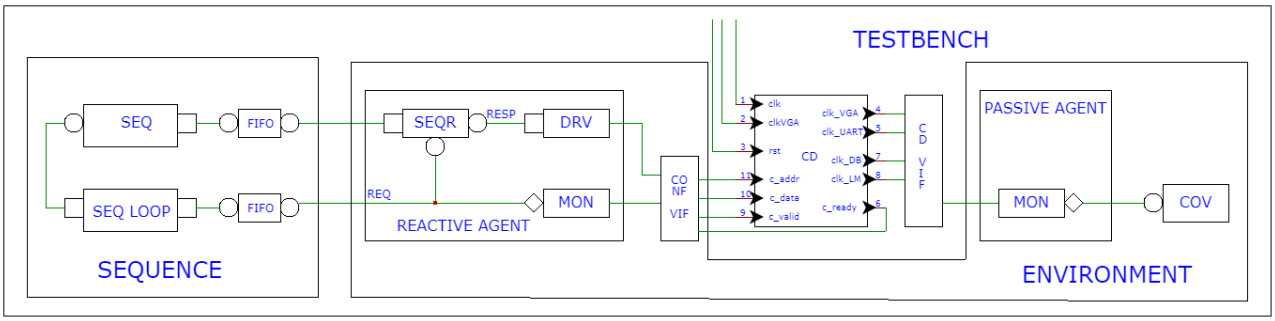

Figura 23. Design verificare Clock Divider

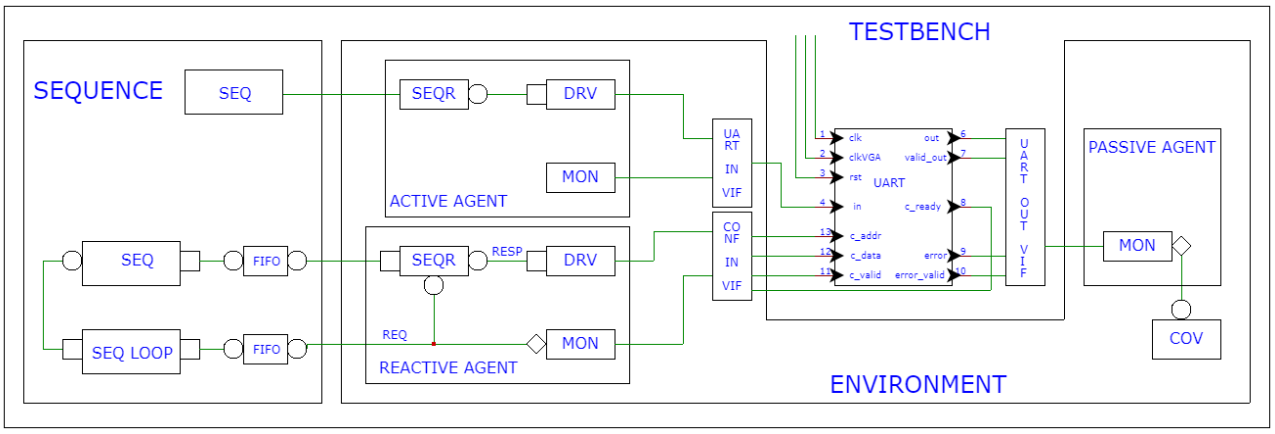

Figura 24. Design verificare UART

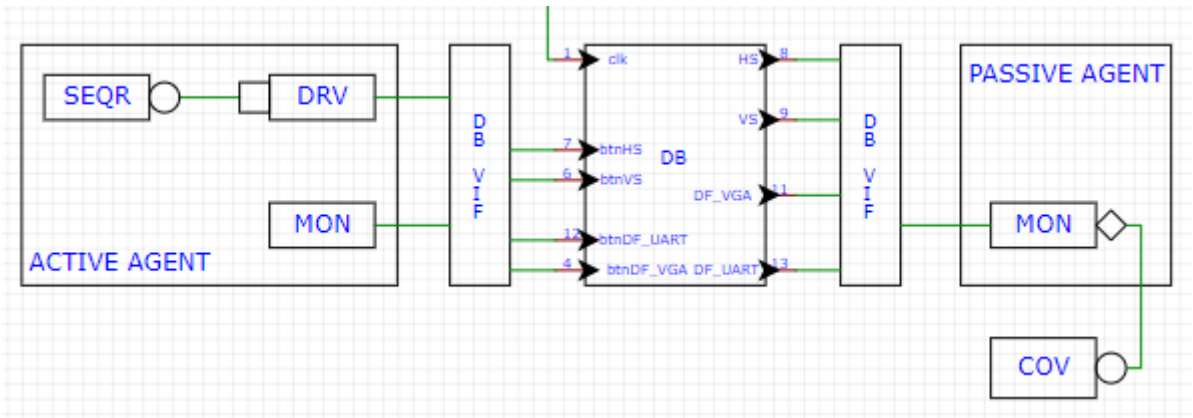

Figura 25. Design verificare Debouncers

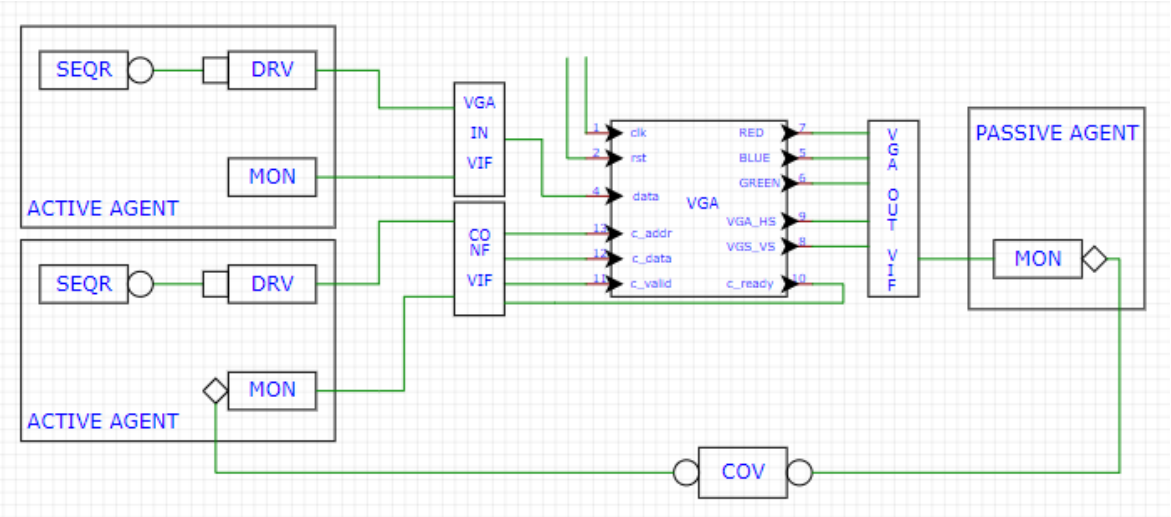

Figura 26. Design verificare VGA

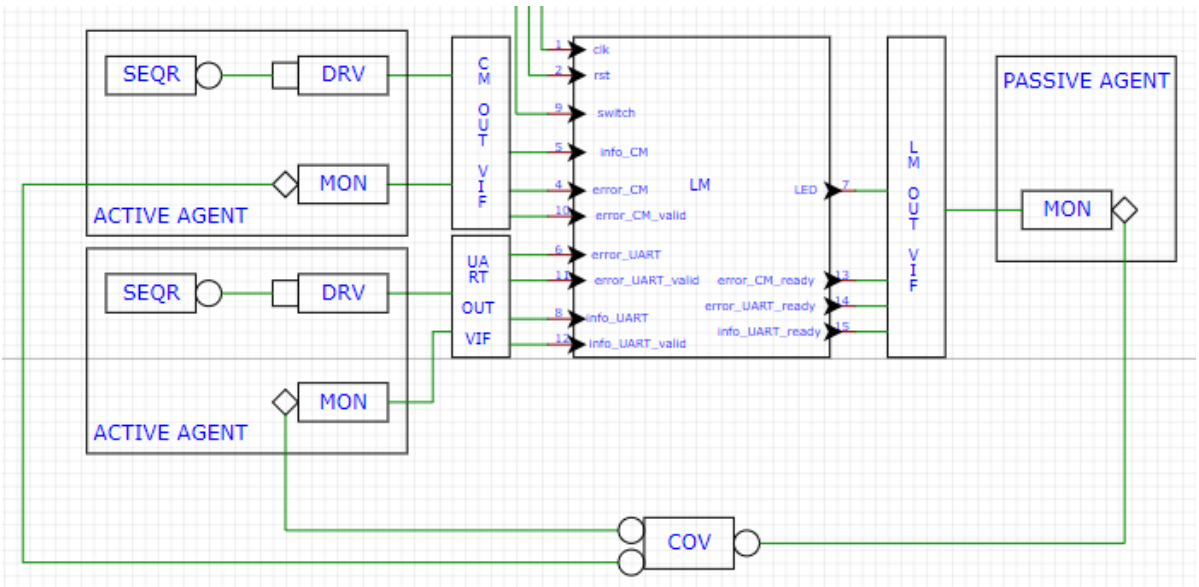

Figura 27. Design verificare Led Manager

4.2.2.7. 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.

O clasă de coverage mai conține unul sau mai multe obiecte, care au rolul de a ține datele primite pe port, și unul sau mai multe covergroupuri, în funcție de cerințe, care vor avea ca parametru obiectul corespunzător fiecăruia. Covergroupurile pot fi incluse în clasa de coverage, dar nu vor mai putea fi refolosite.

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 rescrisă, faza de report. În această fază se afișează procentul de acoperire pe fiecare covergroup în parte.

Cele 6 clase de coverage au în compoziția lor 6 cover grupuri.

Coverpoint	Item	Bins	Nr. Bins	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	-

Coverpoint	Item	Bins	Nr. Bins	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]

Coverpoint	Item	Bins	Nr. Bins	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	-

Coverpoint	Item	Bins	Nr. Bins	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

Coverpoint	Item	Bins	Nr. Bins	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	-

Coverpoint	Item	Bins	Nr. Bins	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

După cum reiese din tabelele anterioare, există 118 cazuri care trebuie acoperite (CD: 24; CM: 15; DB: 16; LM: 16; UART: 22; VGA: 25). 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 95 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.

4.2.3. 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 și environmentele create 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.

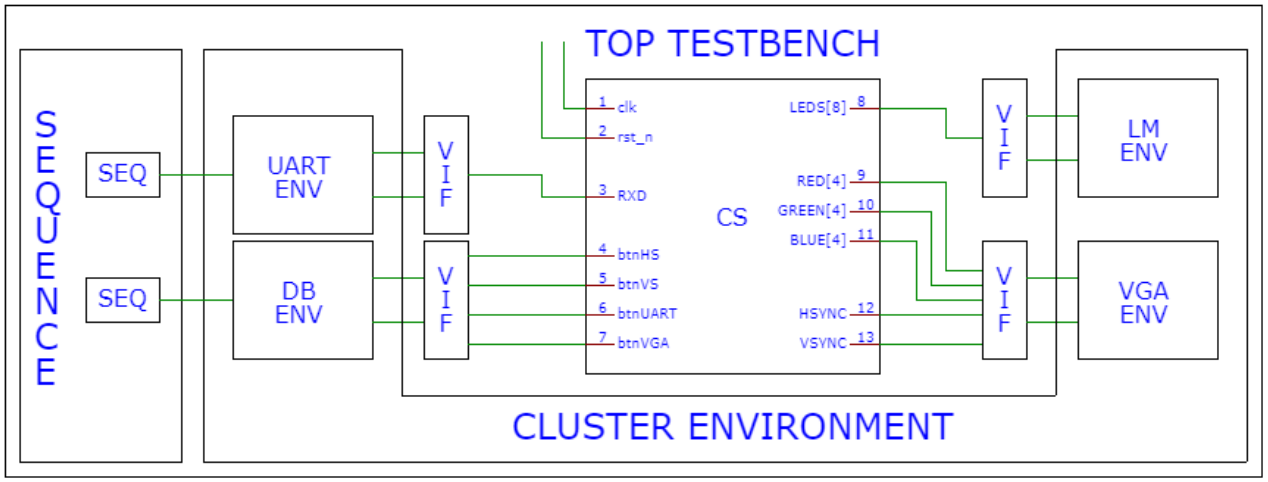

Figura 28. Design verificare Color Show

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.

4.2.4. VERIFICAREA FUNCȚIONALĂ

*** pune poze cu logurile pentru coverage si asertii

4.2.5. VERIFICAREA PROIECTULUI

Figura . HSYNC

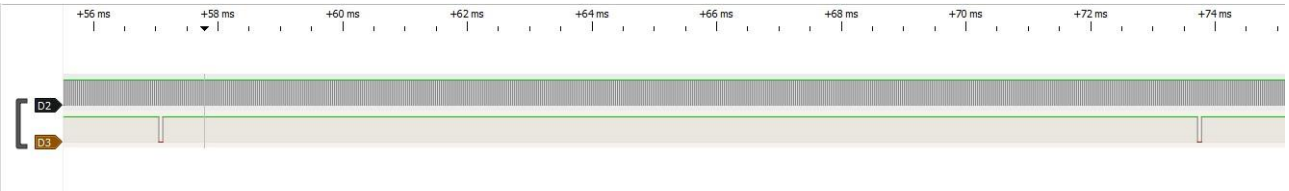

Figura . VSYNC

Figura . UART 8N1 HEX 00 prin Bluetooth

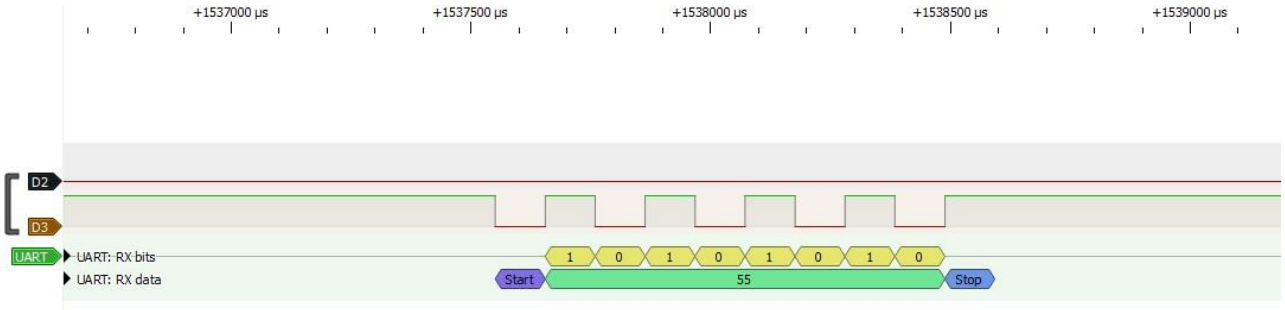

Figura . UART 8N1 HEX 55 prin Bluetooth

5. IMPLEMENTARE

Pentru implementare se va folosi placa de dezvoltare ZedBoard.

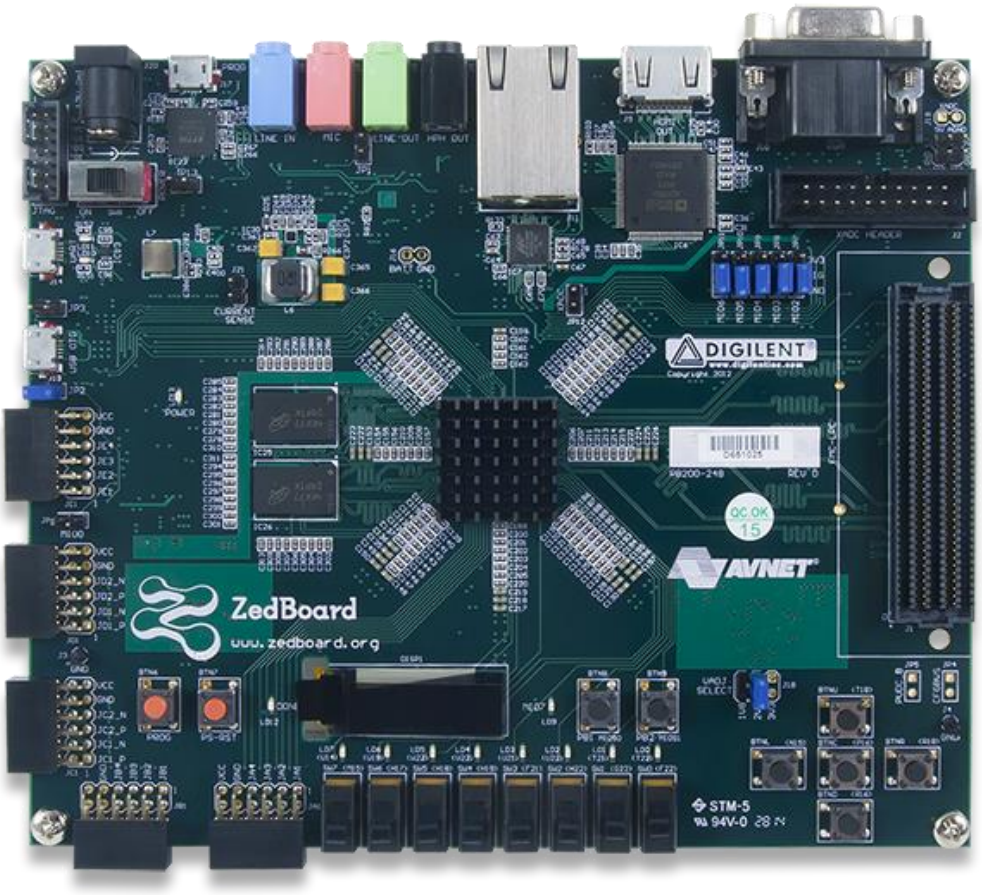

Figura 29. ZedBoard (sursa: https://digilent.com/reference/programmable-logic/zedboard/start)

*** pune poze din vivado si din realitate

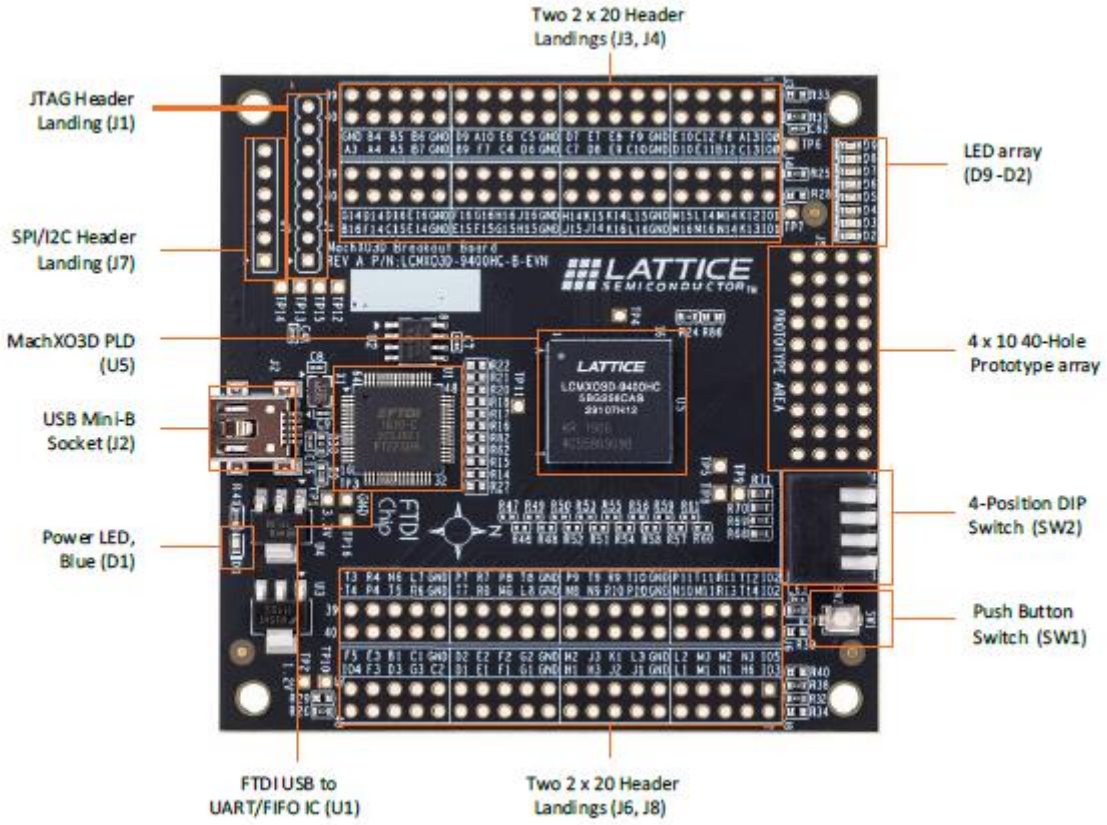

Figura 30. Placa MachXO3D Breakout

6. CONCLUZII

În concluzie, sistemul realizat

În proiectarea, realizarea și validarea sistemului au fost urmăriți pașii de realizare a unui design logic. Au fost definite cerințele și specificațiile designului, după care a fost proiectat sistemul, de la top la unități. Unitățile au fost implementate, verificate și integrate în modulele principale. A fost proiectat mediul de verificare pentru module și cluster, pentru a valida sistemul. Au fost testate cazurile limită și randomizate, pentru a asigura funcționarea robustă a sistemului.

O posibilă direcție de dezvoltare a proiectului este folosirea modulelor într-un sistem de afișare a datelor provenite de la un microcontroler pe ecran. Actual, 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. Dar 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. Astfel, ar putea fi proiectate diferite jocuri pentru una sau mai multe persoane. Se pot adăuga și module de comunicare fără fir, precum Bluetooth, care ar permite realizarea unui joc multi player la distanță.

Așadar, direcțiile de dezvoltare ale proiectului sunt utilizarea acestuia ca periferic pentru sisteme integrate, depanator de ecrane sau implementarea unor jocuri pe FPGA.

7. 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
8. 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
9. https://verificationacademy.com/verification-methodology-reference/uvm/docs_1.1c/html/files/base/uvm_common_phases-svh.html
10. 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.
11. Digilent. Nexys A7™ FPGA Board Reference Manual, Octombrie 2019. https://digilent.com/reference/nexys_vga/refmanual
12.