



Universitatea Tehnică din Cluj-Napoca

Catedra de Calculatoare

Utilizarea portului VGA al plăcii Nexys 4 DDR pentru afișarea unor imagini

Student: **Bîrluțiu Claudiu-Andrei**

Grupa 30236

Indrumător de proiect: Dr. Ing. Cristi Mocan

Data 4.10.2021



Cuprins

Cuprins	2
1.Rezumat	5
2.Introducere	6
2.1 Tema și contextul proiectului	6
2.2 Problema și obiectivele principale ale proiectului	6
2.3 Terminologia de bază și tehniciile folosite în realizarea proiectului	7
2.4 Conținutul lucrării	8
3.Fundamentare teoretică	10
3.1 Placa de dezvoltare Nexys 4 DDR	10
3.1.1 Aspecte generale	10
3.1.2 Portul VGA al plăcii de dezvoltare Nexys 4 DDR	11
3.1.3 Slotul micro SD	12
3.1.4 Mediul de proiectare	12
3.2 Cum lucrează monitoarele VGA	13
3.3 Exemple de controlare VGA	17
3.4 Card Micro SD	18
3.4.1 Descriere generală	19
3.4.2 Comunicare între micro SD și host	19
3.4.3 Pinii cardului micro SD	20
3.4.4 Despre interfața SPI	21
3.4.5 Modul SPI pentru citirea cardurilor micro SD	21
3.5 Alte module	22
3.5.1 Divizorul de frecvență	22
3.5.2 Circuite de debounce	22
3.5.3 Automate de stare	23
3.6 Metode	23
3.6.1 Metode folosite pentru aplicarea unor efecte imaginilor proiectate	23
3.7 Soluția propusă	24
3.8 Scenarii de utilizare propuse	25
4. Proiectare și implementare	26
4.1 Arhitectura	26
4.2 Control Unit	30
4.3 Divizor de frecvență	32
4.4 Sync_generator	32
4.5 Modulul MPG	36
4.6 Modulul Offset Counter	37
4.7 Modulul Memory_address	39



4.8 Modulul SDInitialise	40
4.9 Modulul SDController	45
4.10 Modulul RAM	48
4.11 Modulul ImageCounter	49
4.12 Modulul ModulMicroSD	50
4.13 Modulul EffectGenerator	52
4.14 Modulul ControllerVGA	54
5. Rezultate experimentale	56
5.1 Control Unit	56
5.2 Divizor de frecvență	56
5.3 Sync generator	57
5.4 Modul MPG	59
5.5 Offset counter	60
5.6 Memory_address	60
5.7 SDInitialise	61
5.8 SDControllerRead	62
5.9 Modul RAM	63
5.10 Modul ImageCounter	64
5.11 Modulul ModulMicroSD	64
5.12 Modulul EffectGenerator	66
5.13 Modulul ControllerVGA	66
5.14 Cod python de extragere biți imagine	67
6. Rezultate și mod de utilizare	68
7. Concluzii	72
 7.1 Aspecte generale despre rezultatele obținute	72
Bibliografie	73
Anexa 1 Inițializare card în modul SPI	74
Anexa 2 Cod vhdl pentru Control Unit	75
Cod modul principal ControlUnit.vhd	75
Modul de testare control unit ControlUnit_tb.vhd	76
ANEXA 3 Cod VHDL pentru Divizorul de frecvență	79
Codul modulului principal	79
Modulul de testare DivizorFrecventa_25_tb.vhd	79
Codul pentru divizor de frecvență generic	80
ANEXA 4 Codul VHDL pentru Sync_generator	82
Modulul principal Sync_generator.vhd	82
CODUL pentru fsm_horizontal_sync.vhd	83
Codul pentru fsm_vertical_sync.vhd	85



Codul pentru testare sync_generator_tb.vhd.....	86
ANEXA 5 Codul pentru modulul MPG.....	88
Codul modulului mpg.vhd	88
Codul pentru modulul de test mpg_tb.vhd.....	89
ANEXA 6 Codul pentru modulul Offset Counter.....	91
Codului modulului offset_counter.vhd.....	91
Codul pentru modulul de testare offset_counter_tb.vhd.....	92
ANEXA 7 Modulul Memory address	95
Codul pentru memory_address.vhd	95
Codul pentru testare memory_address_tb.vhd.....	96
ANEXA 8 Modulul SDInitialise	98
Codul pentru SDInitialise.vhd.....	98
ANEXA 9 Modulul SDControllerRead	104
Codul pentru SDController.vhd	104
ANEXA 10 Modulul ModulMicroSD.....	109
Codul pentru ModulMicroSD.vhd	109
ANEXA 11 Modulul RAM	115
Codul pentru RAM.vhd.....	115
Codul pentru RAM_tb.vhd	116
ANEXA 12 Modulul ImageCounter	118
Codul pentru ImageCounter.vhd	118
Codul pentru ImageCounter_tb.vhd.....	119
ANEXA 13 Modulul Effect_generator	121
Codul pentru Effect_generator.vhd	121
Codul pentru Effect_generator_tb.vhd.....	122
ANEXA 14 Modulul ControllerVGA	124
Codul pentru ControllerVGA.vhd.....	124
Codul pentru ControllerVGA_tb.vhd	125
ANEXA 15 Modulul ExecutionUnit.....	127
Codul pentru ExecutionUnit.vhd	127
ANEXA 16 Modulul BlackBox	131
Codul pentru BlackBox.vhd.....	131
ANEXA 17 Pachet VGA.....	133
VgaPackage.....	133
ANEXA 18 Fișierul de constrângeri	134
ANEXA 19 Cod python obținere fișier binar.....	143
Cod	143



1. Rezumat

Înainte de a posta o fotografie pe rețelele de socializare de multe ori cauți un filtru pentru a modifica imaginea inițială, iar principalul scop este de a obține aprecieri în plus, nu-i aşa? În principiu vrei să modifici culorile imaginii, fie ca imaginea să aibă culori mai deschise sau mai închise. Prezentul raport descrie un controller VGA prin care mai multe imagini stocate pe un card SD sunt citite și afișate pe rând pe un monitor cu ajutorul placii de dezvoltare Nexys 4 DDR. Asupra imaginii proiectate pe monitor se pot aplica câteva filtre prestabilite sau utilizatorul poate să deschidă sau să închidă culorile imaginii. În prezentul document se aduc la cunoștință informații legate de etapele realizării acestui proiect și sunt descrise principalele tehnici de proiectare și implementare, tehnici precum SPI (Serial Peripheral Interface) pentru citirea datelor de pe cardul micro SD, FSM (Finite-State Machine) pentru implementarea unității de control. Codul sursă este scris în limbajul VHDL, un limbaj de descriere hardware. Mediul de dezvoltare folosit este Vivado Xilinx, cu ajutorul căruia codul sursă VHDL a fost sintetizat.



2. Introducere

2.1 Tema și contextul proiectului

Tema principală a proiectului se referă la realizarea unui controller VGA implementat pe placa de dezvoltare Nexys 4 DDR care are rolul de a prelua imagini de pe un card micro SD și de a le projecția pe un monitor. Pentru a include o parte interactivă, controllerul VGA ar trebui să permită și modificarea culorilor imaginii în timp real, fie prin aplicarea unor filtre predefinite, fie prin modificarea intensității culorilor prin intermediul butoanelor puse la dispoziție utilizatorului. În cadrul controllerelor VGA prezентate în lucrările de referință [1], [4] imaginea de proiectat este fie direct generată sau calculată prin prelucrarea pixelilor în momentul în care aceștia sunt proiecțiați pe ecran, fie stocată în memoria placii de dezvoltare și apoi preluată și proiectată pe monitor (prin pixel înțelegându-se cel mai mic element adresabil al unei imagini rasterizate). Aceste abordări sunt limitate din punct de vedere al spațiului de memorie disponibil, iar generarea directă a imaginii necesită anumite strategii de calcul prin care se determină configurația pixelilor. De asemenea, algoritmii care vor fi folosiți pentru determinarea pixelilor imaginii de proiectat vor duce la arhitecturi complexe ce trebuie configurate pe placa de dezvoltare și implicit sunt limitate din punct de vedere a numărului de blocuri (LUT-uri) destinate plasării și rutării arhitecturilor. În [4] sunt descrise câteva metode prin care să se reducă dimensiunea memoriei necesare stocării imaginii de afișat, dar această abordare va duce la scăderea calității imaginii afișate, deoarece pixelul din memorie îi vor corespunde mai mulți pixeli ai ecranului (*title-mapped scheme*). În cazul proiectului de față, un card microSD poate să stocheze un număr mai mare de imagini și se pot citi pe rând datele acestor imagini într-o memorie RAM și apoi afișate pe ecranul monitorului prin controllerul VGA, fără a fi necesar ca toate imaginile să fie stocate simultan în memoria plăcuței FPGA.

2.2 Problema și obiectivele principale ale proiectului

Problema de rezolvat se referă la utilizarea portului VGA disponibil pe placa de dezvoltare Nexys4 DDR pentru a afișa pe ecranul unui monitor mai multe imagini preluate de pe un card microSD. Maparea directă a imaginilor pe placa FPGA și afișarea lor ulterioră ar genera un număr mare de arhitecturi ce trebuie configurate, iar acest lucru ar necesita un număr mai mare de resurse și de asemenea ar exista riscul că sistemul să nu poată fi implementat pe placa de dezvoltare. Astfel, pentru a avea un număr mai mare de imagini care pot fi afișate pe ecran fără a-și pierde din calitate, se recurge la folosirea unei memorii externe cum este un card microSD unde să se stocheze imaginile, iar apoi să fie preluate și aduse în memoria RAM pentru a fi procesate și afișate pe ecranul monitorului.

Obiectivele controller-ului VGA constau în:

1. afișarea unor imagini preluate de pe un card microSD pe ecranul monitorului
2. aplicarea unor filtre predefinite imaginilor de către utilizator, creșterea și scăderea intensității culorilor prin intermediul butoanelor disponibile pe placă
3. posibilitatea utilizatorului de a schimba imaginea de afișat



2.3 Terminologia de bază și tehniciile folosite în realizarea proiectului

FPGA-urile (“Field-programmable gate array”) sunt circuite integrate care sunt proiectate după cum le spune și numele, pentru a fi configurate ulterior de către utilizatori. Acestea conțin un set de blocuri logice programabile și o ierarhie de interconexiuni reconfigurabile ce permit legarea blocurilor între ele. De subliniat este faptul că aceste blocuri sunt limitate, iar arhitecturile complexe sunt greu sau imposibil de configurați, un exemplu bun fiind cazul unui controller VGA în care se încearcă algoritmi complecsi pentru generarea unor imagini ce vor fi proiectate pe ecranul monitorului. Limbajele cele mai des folosite pentru configurația astfelor de circuite sunt de tipul HDL (“hardware description language”) care sunt utilizate pe scară largă datorită mai multor avantaje dintre care amintim: posibilitatea descrierii funcționale a sistemelor la nivel înalt fără a detalia structuri ale porțiilor elementare (adică o abordare structurală a sistemului) și de asemenea oferă posibilitatea modificării mai ușoare a sistemului. Limbajul HDL folosit în realizarea proiectului de față este VHDL.

Utilitarul folosit pentru simulare, sinteză, implementarea și generarea ‘bitsream-ului’ este Vivado 2019.1 de la compania Xilinx. Acest IDE oferă utilizatorilor un mediu prietenos, intuitiv și interactiv pentru astfel de proiecte. De asemenea, mediul Vivado dispune de un simulator in-built denumit ISIM cu ajutorul căruia se pot testa înainte de implementare componentele și sistemul în ansamblu pentru a nu apărea complicații în momentul implementării. Simularea sistemului înainte de implementarea efectivă a acestuia va reduce costurile și numărul de erori care pot apărea în timpul implementării.

Prin VGA (“Video Graphics Array”) se înțelege un standard video introdus la sfârșitul anilor '80 în calculatoarele IBM și este suportat de majoritatea componentelor grafice ale PC-urilor și monitoarelor de astăzi. În [2] este descris principiul de funcționare a monitoarelor CRT ce se bazează pe fascicule de electroni de amplitudine modulară sau raze catodice pentru afișarea informațiilor pe un ecran phosphor-coated. Acest tip de ecrane se folosește mai puțin în zilele noastre datorită apariției ecranelor de tip de tip LCD (“liquid-crystal display”). Acestea din urmă au un alt mod de funcționare folosind un șir de comutatoare ce pot impune un voltaj asupra cristalelor lichide, schimbând astfel permisibilitatea luminii prin aceste cristale pixel cu pixel. În partea de fundamentare teoretică va fi descris în principiu modul de funcționare a monitoarelor CRT, dar noțiunile de sincronizare și datele pixels sunt valabile și pentru LCD.

Pentru stocarea imaginilor se va folosi un card microSD de 2GB. Cardul va fi formatat cu ajutorul unui editor hexazecimal și va conține pixelii imaginilor sub forma unor siruri de biti. Cardurile de memorie SD sunt special proiectate pentru a asigura securitatea, performanța și alte cerințe necesare stocării fișierelor în device-urile electronice. Avantajele utilizării cardurilor de memorie sunt: portabilitatea, tensiuni mici de alimentare (maximele între 2.7 și 3.6 V), viteza de citire a datelor, mecanism ușor de stergere a datelor, dimensiuni mici vs capacitate mare de stocare a memoriei.

Placa de dezvoltare Nexys 4 DDR este o platformă de dezvoltare completă care aparține clasei de FPGA Artix-7TM, Xilinx. Aceasta placă dispune de diferite periferice și componente, putând fi folosită într-o gama largă de proiecte fără a necesita alte resurse. În continuare sunt amintite principalele caracteristici ale plăcii de dezvoltare necesare realizării controllerului VGA. Alte



specificații despre placa Nexys 4 DDR se regăsesc în [2]. Nexys 4 DDR dispune de 14 semnale FPGA pentru crearea portului VGA: 4 biți pentru fiecare dintre cele trei culori (rosu, verde, albastru) și două semnale pentru timpii de sincronizare (horizontal sync și vertical sync- aprofundarea celor 2 concepte în capitolul urmator). Având în vedere faptul că pentru fiecare culoare avem 4 biti, se pot determina astfel 16 nivele de semnal pentru fiecare din cele 3 culori, iar combinatia acestora va duce la 4096 de culori diferite care se pot afisa. Un controller VGA va avea misiunea de a asocia semnalele de culoare și cele de sincronizare amintite mai sus la momentele de timp corespunzătoare pentru a determina poziția și configurația culorii pixelilor afișați pe ecran.

Nexys 4 DDR conține și un slot pentru carduri de memorie microSD atât pentru configurarea plăcii FPGA care va prelua ‘bitstream-ul’ din memoria cardului, cât și pentru accesul utilizatorilor. În proiectul dat, pe cardul microSD vor fi încărcate fișiere binare cu codificarea imaginilor ce vor fi proiectate pe ecranul monitorului (bitii corespunzători pixelilor imaginii). Pentru vizualizarea fișierelor și fomatarea cardului se va folosi editorul *HxD Free ware and Disk Editor* disponibil pe sistemul de operare Windows.

Pentru descrierea hardware a sistemului s-au folosit mai multe tehnici. Pentru modulul microSD s-a recurs la interfața SPI (Serial Peripheral Input), o interfață simplă și care permite comunicația între microcontrolere și diferite circuite periferice din sistemele înglobate (în proiectul de față fiind folosită pentru legarea modulului microSD la controllerul VGA). Pentru unitatea de control a sistemului se va folosi un automat de stare.

2.4 Conținutul lucrării

În capitolul 2, *Fundamentare teoretică* vor fi prezentate în detaliu concepțile și tehniciile folosite pentru realizarea proiectului și de asemenea va conține referințe la resursele bibliografice, unde sunt descrise și alte metode pentru implementarea unui astfel de controller VGA. Vor fi detaliate concepțile principalelor modele arhitecturale folosite pentru realizarea sistemului. Se vor explica concepțile legate de sincronizare (timing) și configurația pixelilor ce vor fi mapați pe ecranul monitorului și modul în care se transmit aceste semnale prin intermediul portului VGA pus la dispozitie de către placa de dezvoltare. De asemenea, va fi descris modul de citire de pe un card micro SD folosindu-se interfața SPI. Se va explica și modul de aplicare a unor filtre imaginilor de afișat pe ecran (în timp real).

În capitolul 3, *Proiectare și implementare* vor fi prezentate în detaliu etapele parcurse pentru realizarea obiectivelor proiectului. Va fi descrisă soluția aleasă pentru implementarea sistemului și motivul pentru care s-a recurs la o astfel de abordare. Se vor descrie schema bloc și cea arhitecturală cu toate modulele hardware și software folosite, însotite de diferiți algoritmi, precum aplicarea de filtre imaginilor. De asemenea se va regăsi și un manual de utilizare, cu ajutorul căruia utilizatorul va putea să interacționeze cu controllerul VGA. Se vor regăsi diagramele de stare ale unității de control a controllerului VGA și a interfeței SPI pentru modulul de citire a cardului micro SD.

În capitolul 4, *Rezultate experimentale* se vor prezenta simulările sistemului și validitatea rezultatelor obținute în urma implementării. Se vor prezenta principalele instrumente de proiectare și testare folosite, precum și rapoartele de implementare și performanța din punct de vedere a resurselor folosite în realizarea circuitului. Vor fi incluse capturi de ecran cu simulările diferitelor module și se



vor face anumite comparații între mai multe implementări ale aceluiași modul. Rezultatele vor fi prezentate sub forma tabelară sau grafică pentru a fi vizualizate mai ușor.

În capitolul 5, *Concluzii* se va prezenta un sumar al proiectului care include principalele observații cu privire la dificultățile întâmpinate pe parcursul realizării proiectului și interpretările proprii despre rezultatele obținute. Este exprimată opinia proprie a autorului despre noțiunile dobândite și sunt amintite propunerile și îmbunătățiri pentru dezvoltări ulterioare a sistemului. De asemenea, vor fi prezentate principalele avantaje și dezavantaje ale soluției propuse pentru problema dată și vor fi conturate contribuțiiile originale ale autorului.

3.Fundamentare teoretică

3.1 Placa de dezvoltare Nexys 4 DDR

3.1.1 Aspecte generale

Nexys 4 DDR reprezintă o platformă de dezvoltare pentru circuite integrate bazată pe cea mai recentă tehnologie de Artix-7™ Field Programmable Gate Array(FPGA) al echipei Xilinx. Datorită resurselor numeroase pe care le pune la dispoziție (porturi USB, Ethernet, VGA, memorii externe , slot micro SD , senzor de temperatură, microfon etc), această platformă permite integrarea unor circuite atât simple combinaționale, cât și mai complexe precum procesoare embedded. Având un număr mare de componente, se pot proiecta o gamă largă de sisteme fără a avea nevoie de alte resurse.

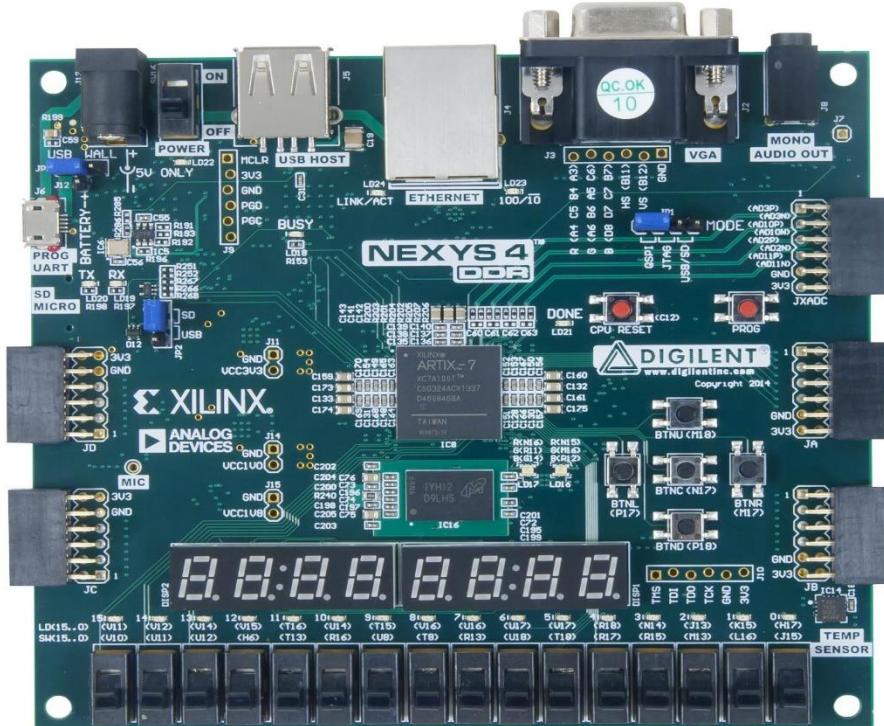


Figura 3.1.1 Placa de dezvoltare Nexy 4 DDR

Această clasă de produse, Artix-7 FPGA, oferă o performanță înaltă atât din punct de vedere al capacitații de memorie cât și al resurselor puse la dispoziție. Principalele specificații ale plăcii de dezvoltare Nexys 4 DDR se referă la numărul mare de slice-uri (15 850 la număr) fiecare având 4 LUT-uri și 8 bistabile de tip flip-flops. Acest număr mare de slice-uri oferă capacitatea implementării unor sisteme cu arhitecturi complexe, precum și posibilitatea de mapare și rutare a componentelor cât mai optim. De asemenea, placa dispune de 4860 Kbits de memorie RAM de viteză ridicată și un convertor on-chip analogic-digital (XADC).

Din colecția de porturi și periferice vom aminti : 16 switch-uri , 16 led-uri , două display-uri 4-digit 7-segment, port de ieșire 12-bit VGA, 5 butoane, conector pentru card micro SD , senzor de

temperatură , port USB-JTAG pentru programare și comunicare. Mai multe informații despre resursele disponibile ale plăcii de dezvoltare se regăsesc în [2].

FPGA-urile sunt un tip de circuit logic programabil (format dintr-o matrice de blocuri logice configurabile, CLB-uri, ce sunt conectate prin interconexiuni programabile), iar pentru a-l configura se folosește în general un limbaj de descriere hardware (HDL). Dintre cele mai importante operații ce sunt executate atunci când se utilizează un circuit FPGA pentru implementare sunt operațiile de plasare și rutare. Plasarea se referă la selecția blocurilor sau modulelor circuitului programabil care vor fi utilizate pentru implementarea diferitelor funcții ale sistemului digital, pe când rutarea constă în interconectarea blocurilor.

Nexys 4 DDR dispune de o memorie DDR2 SDRAM de 128 MiB, iar procesul prin care se încarcă informațiile de configurare în memoria circuitului se numește configurare, termen folosit pentru acest tip de circuite programabile ce sunt bazate pe memorii volatile. Sistemul configurat pe placă dezvoltare se va sterge din memorie în momentul opririi alimentării.

De asemenea, placă conține un oscilator de cuarț ce generează un semnal de ceas de 100MHz.

3.1.2 Portul VGA al plăcii de dezvoltare Nexys 4 DDR

Portul VGA al plăcii de dezvoltare Nexys 4 DDR este determinat de 14 semnale principale: 4 semnale pentru fiecare din cele 3 culori primare (roșu, verde și albastru) și 2 semnale de sincronizare (sincronizare pe orizontală și sincronizare pe verticală). În figura de mai jos (preluată din [2]) este ilustrat un astfel de port.

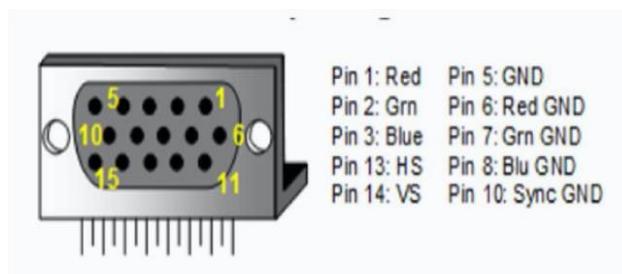


Figura 3.1.2 Port VGA cu pinii corespunzători

Având în vedere faptul că se pot transmite câte 4 biti pentru fiecare semnal de culoare, se pot obține astfel 16 nivele de semnal pentru o singură culoare, în limbaj natural înțelegându-se 16 nuanțe diferite ale culorii respective. Deoarece culoarea pixelului va fi dată prin combinarea celor 3 semnale de culoare, va rezulta o gamă de 4096 de culori diferite pentru un singur pixel.

Misiunea unui controller VGA se rezumă la transmiterea spre monitor a configurației pixelului ce trebuie afișat la momentul de timp bine determinat prin semnalul *pixel rate* și de asemenea transmiterea celor două semnale de sincronizare (HS și VS).

Pozitia pixelilor se va determina cu ajutorul unor numărătoare ce sunt incrementate de pixel - rate-ul de 25MHz.Pentru o rezoluție de 640x480, primul numărător va număra în intervalul 0:639, și



va determina coloana de pe ecran unde s-a ajuns cu procesul de scanare, iar cel de-al doilea numărător va număra în intervalul 0:479, incrementându-se în momentul în care se trece la o nouă linie din gridul de rasterizare, oferind astfel poziția pe axa verticală a pixelului.

3.1.3 Slotul micro SD

Placa Nexys 4 DDR pune la dispozitie un slot micro SD ce va permite doua funcționalități: fie este folosit pentru configurarea plăcii (încărcarea bitstream-ului), fie pentru accesul utilizatorilor (citirea/ scrierea unor date ce vor fi procesate de sistemul configurat). Proiectarea unui controler pentru lucrul cu micro SD va avea în vedere cele 2 interfețe prin care se poate comunica cu un card micro SD: bus mode și SPI. Pentru a alimenta slotul, semnalul SD_RESET trebuie să fie dezactivat de placa FPGA.

3.1.4 Mediul de proiectare

Pentru proiectarea sistemelor digitale se utilizeaza pachete de programe de proiectare asistată de calculator CAD. Aceste programe oferă mai multe funcționalități: descrierea sistemului digital, sinteza descrierii (translatarea descrierii sistemului într-o listă de conexiuni), simularea funcțională (folosită înainte de implementarea propriu zisă pentru evitarea erorilor), implementarea sistemului (constă în adaptarea listei de conexiuni pentru o utilizare eficientă a resurselor disponibile) și configurarea circuitului.

Nexys 4 DDR este compatibil cu mediul de dezvoltare Xilinx Vivado Design Suite ce integrează toate utilitarele pentru proiectarea sistemelor digitale ce pot fi programate pe această placă. Aceste utilitare vor permite realizarea funcționalităților amintite mai sus, funcționalități ce pot fi executate cu ajutorul unei interfețe grafice intuitive pusă la dispozitie utilizatorului, numita Vivado Integrated Design Environment(IDE).

Alte facilități pe care le pune la dispozitie acest mediu se referă la analizele de performanță ale sistemului prin simularea logică a diferitelor etape ale procesului de proiectare, verificarea regulilor de proiectare, definirea costrângerilor (de sinteză și implementare), analiza puterii consumate, vizualizarea structurii proiectului, schema RTL a sistemului etc.

Descrierea RTL a sistemului de proiectat poate fi specificată în limbaje de descriere hardware cum este limbajul VHDL. Cu ajutorul acestui limbaj se poate descrie comportamentul și structura sistemelor digitale la multiple nivele de abstractizare.

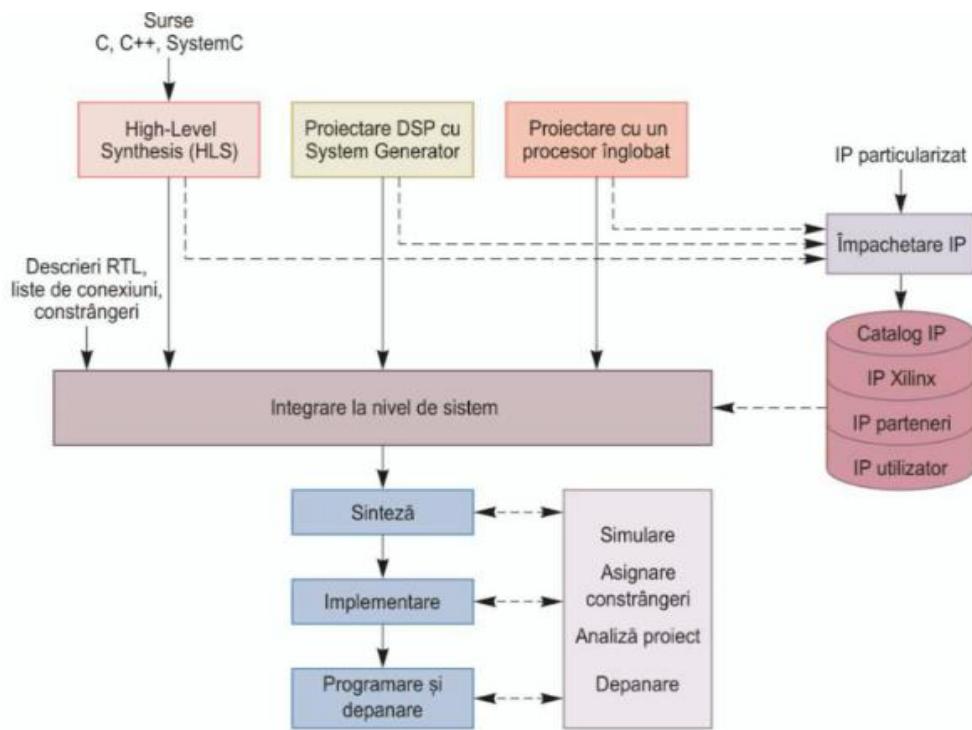


Figura 3.1.3 Procesul de proiectare utilizand mediul de proiectare

Xilinx Vivado Design Suite [7]

3.2 Cum lucrează monitoarele VGA

VGA (“Video Graphics Array”) este o interfață analogică care a fost extrem de folosită înainte de apariția interfețelor digitale cum ar fi DVI și HDMI. Diferențele între cele două tehnologii sunt pe larg explicate în capitolul 16 (*VHDL Designs of DVI Video Interfaces*) din [5]. În momentul de față tot mai puține laptopi și computere sunt prevăzute cu porturi VGA, datorită faptului că se vrea o performanță cât mai bună, obținută prin cadrul interfețelor mai noi, amintite mai sus, în ceea ce privește transmiterea datelor spre monitoare și care asigură o calitate înaltă a imaginii. Ecranul monitorului pentru un VGA standard are rezoluția de 640x480 ceea ce înseamnă un număr destul de mic de pixeli. Prin aceasta rezoluție se înțelege că vor exista 640 coloane X 480 linii de elemente ale imaginii, numite pixeli. Ecranul poate fi văzut ca un grid (rețea de linii și coloane) sau ca o matrice de pixeli, astfel că fiecare element al imaginii îi corespunde un loc bine determinat pe ecran. De fapt, o imagine este afisată pe monitor prin aprinderea și stingerea pixelilor individuali ai acestuia. Monitorul se va afla într-o continuă scanarea a întregului său ecran, determinând fiecare pixel la un moment de timp bine specificat. Această trecere prin matricea de pixeli se va realiza extrem de rapid, astfel încât ochiul uman este înselat, având clara impresie că toți pixelii sunt aprinși în același timp pe tot ecranul.

În figura 3.2.1 este prezentat faptul că procesul de scanare a ecranului pornește din punctul $(0,0)$ al ecranului care este de fapt colțul stânga-sus, și se deplasează spre dreapta până când va atinge ultima coloană de pe linia respectivă, moment în care se trece la capătul stâng al liniei următoare.

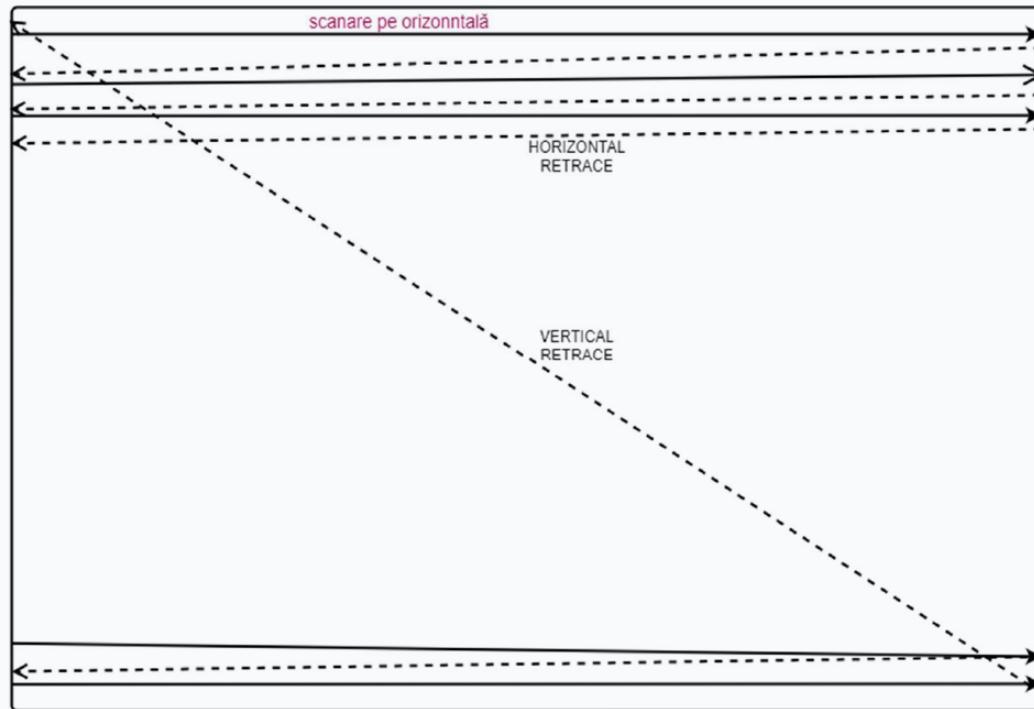


Figura 3.2.1 Scanare ecran CRT

Când se ajunge cu scanarea la ultimul pixel al matricii (pixelul din colțul dreapta jos), scanarea se va relua de la început, adică de la primul pixel. Pentru a se evita fenomenul de „pâlpâire” a ecranului se dorește ca gridul ecranului să fie scanat de cel puțin 60 de ori pe secundă (ceea ce înseamnă o frecvență de 60 Hz).

Pentru controlul monitorului VGA este nevoie de 5 semnale esențiale în definirea pixelilor de pe ecran: 3 semnale pentru culorile primare roșu, verde și albastru, și 2 timpi de sincronizare (sincronizare pe orizontală și sincronizare pe verticală). Cele 3 semnale dedicate celor 3 culorilor de bază sunt folosite pentru a configura culoarea pixelului a cărui poziție pe ecran este determinată cu ajutorul celor doi timpi de sincronizare. Datorită faptului că cele 3 semnale dedicate culorilor sunt de tip analogic, prin varierea tensiunii acestora în intervalul $0 - 0.7$ V va varia implicit și intensitatea culorilor pixelilor, obținându-se astfel diferite nuanțe de culoare.

Timpul ratei de scanare a ecranului va fi dat de cele două semnale de sincronizare. HS (timpul de sincronizare pe verticală) va determina timpul necesar scanării unei linii, iar VS (timpul de sincronizare pe orizontală) va determina timpul de scanare a întregului ecran. Mai multe detalii despre timpii necesari și fazele prin care trece scanarea ecranului se regăsesc în capitolul 10.3.3 („VGA Monitor Controller”) din [1].



Având în vedere principiul de funcționare a monitoarele CRT (cathode ray tube) prezentat pe larg atât în capitolul 12 („VGA Controller I: Graphic”) din [4], cât și în secțiunea 9.1 („VGA System Timing”) din [2], există nișe timpi „morti” în procesul de scanare a ecranului în care niciun pixel al imaginii nu se afișează pe ecran, aşa numita perioadă de „blanking”. Acest fenomen se datorează faptului că în cadrul procesului de scanare a ecranelor catodice se folosește un fascicul de electroni care lovește suprafața de phosphor de pe ecran în punctele corespunzătoare pixelilor imaginii de proiectat. Fasciculul parcurge ecranul catodic ca în figura 3.1. O observație foarte prețioasă care ne ajută la înțelegerea semnalelor de timing este aceea că pe ecran este afișată informația doar când fascicul „înaintează” (deci se mișcă de la stânga la dreapta și de sus în jos), ci nu și în timpul în care fasciculul se resetează, se întoarce la marginea din stânga sau de sus a display-ului. Deci, în acel timp de resetare ecranul își pierde din potențialul de afișare, regăsindu-se într-o perioadă de „blanking” până când fasciculul e stabilizat și poate să înceapă un nou pas pe orizontală sau pe verticală. Modul de funcționare a ecranelor LCD („liquid-crystal display”) diferă de cel prezentat pentru CRT și se bazează pe un șir de comutatoare ce pot impune un voltaj asupra cristalelor lichide, schimbând astfel permisibilitatea luminii prin aceste cristale pixel cu pixel. Cu toate acestea, pentru controllerul VGA pe care dorim să îl implementăm, acei timpi „morti” prezentați mai sus sunt valabili și în cadrul ecranelor LCD și se pot urmări în figurile

Din diagrama de timp (figura 3.2.2) pentru scanarea pe orizontală se observă că este nevoie de 800 de pași pentru parcurgerea unei linii din gridul ecranului. Acești 800 de pași sunt împărțiți în 4 regiuni astfel:

1. **HD** - Zona de display (640 pixeli) : regiunea în care pixeli imaginilor apar pe ecran
2. **HR** - Horizontal retrace (96 pixeli) : reprezintă perioada de timp în care scanarea începe din

nou de la marginea din stânga. În acest timp, niciun pixel al imaginii nu trebuie afișat pe ecran

3. **HB** - Back Porch (48 pixeli): este regiunea din partea stângă a zonei de display în care se ajunge după retrace; și în această perioadă nu se va afisa niciun pixel al imaginii
4. **HF** - Front porch (16 pixeli) : reprezintă regiunea din parte dreaptă a zonei de display care precedă horizontal retrace-ul; de asemenea niciun pixel transmis pentru afisare

Scanarea pe verticală a ecranului are în vedere parcurgerea a 525 de pași corespunzători numărului de linii ale gridului și ai perioadei de vertical retrace. Aceasta corespunde timpului necesar de refresh a întregului ecran. Trebuie să luăm în vedere că trecerea la următorul pas pe verticală este condiționat de scanarea pe orizontală și această scanare este compusă din 4 regiuni conform diagramei de timp ilustrată în figura 3.2.3.

1. **VD** - Zona de display (480 linii): reprezintă regiunea în care liniile orizontale sunt afișate pe ecran, deci singura perioadă în care este permisa afisarea pixelilor imaginii. Pentru toate celelalte regiuni, nu se va mai transmite niciun pixel al imaginii pentru a fi afișat pe ecran.
2. **VR** - Vertical Retrace (2 linii): reprezintă perioada în care fasciculul de electroni se întoarce în partea de sus a ecranului, deci se începe noul ciclu de scanare.

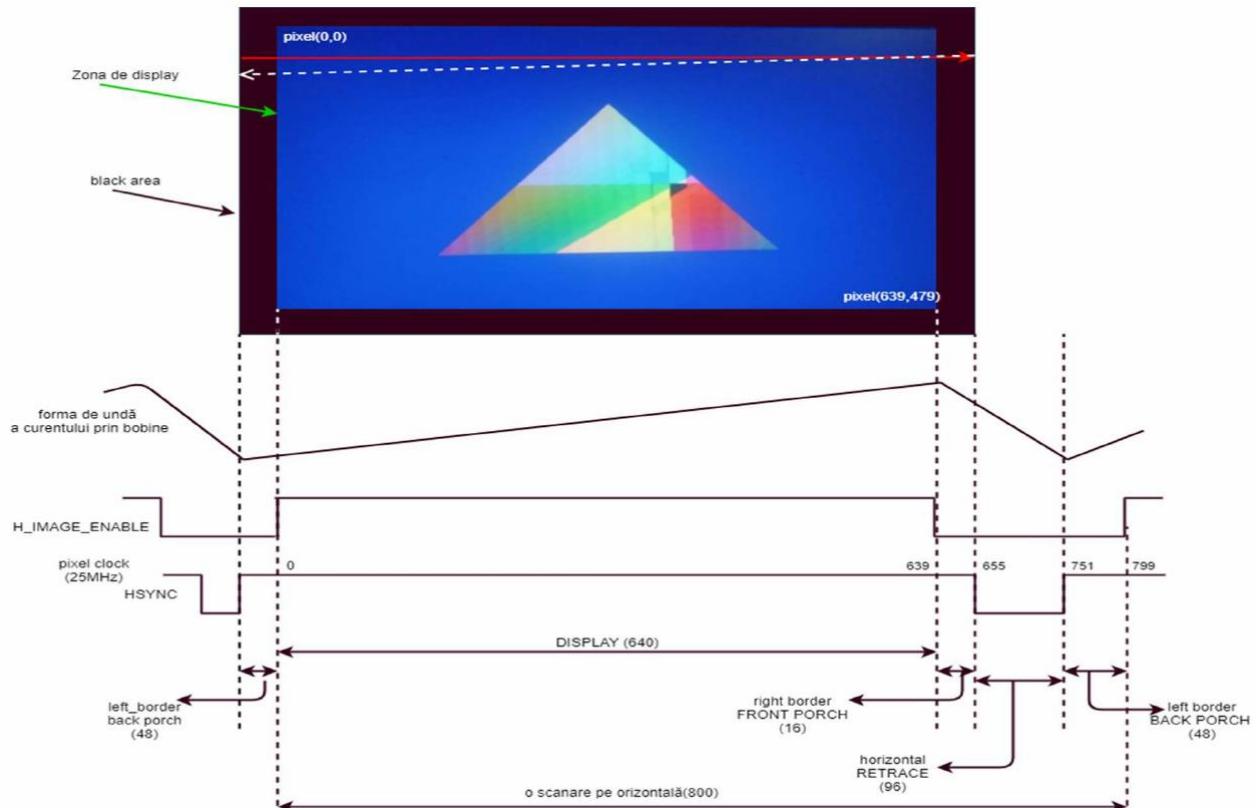


Figura 3.2.2 Timing diagram pentru o scanare pe orizontală

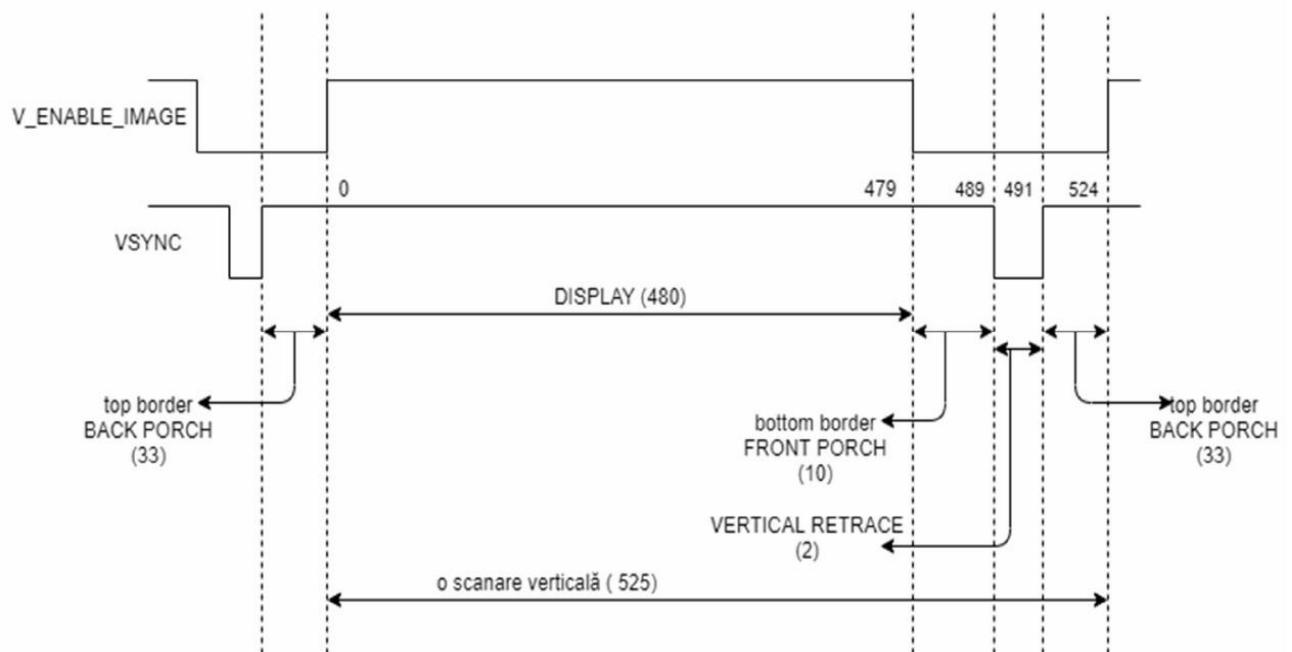


Figura 3.2.3 Timing diagram pentru o scanare pe verticală



3. **VF** - Front Porch (10 linii) : este regiunea din partea de jos a zonei de display care precedă retrace-ul pe verticală.

4. **VB** - Back Porch (33 linii) : este regiunea din partea de sus a ecranului în care se ajunge după retrace-ul pe verticală;

Sunt de reținut aceste regiuni în care se poate afla procesul de scanare a ecranului deoarece se pot concretiza în stări ale automatului de stare (FSM) folosit pentru unitatea de control a controllerului VGA. Se recurge la o astfel de abordare, deoarece în fazele/ regiunile enumerate mai sus se pot genera semnale specifice pentru a putea transmite configurația pixelilor imaginii la momentul portivit pe ecranul monitorului.

În capitolul dedicat portului VGA din [2] este specificat faptul că pentru o rezoluție de 640x480 este nevoie de o rată de refresh de 60 Hz. Pentru o astfel de rată de refresh prin care este evitat fenomenul de „pâlpâire” a ecranului este nevoie de determinarea frecvenței de parcursere a pixelilor de pe ecran în procesul de scanare, așa numitul *pixel rate*. Aceasta se calculează foarte ușor, plecând de la datele pe care le cunoaștem. Acest pixel rate este influențat de trei parametri : numărul de pixeli dintr-o scanare orizontală(C), numărul de linii ale ecranului dintr-o scanare verticală(L) și frecvența de refresh a ecranului(F = 60Hz = 60 ecrane/s).

$$C = 800 \text{ pixels/line}$$

$$L = 525 \text{ pixels/screen (din diagramele de timing)}$$

$\text{Pixel_rate} = C * L * F = 25\text{MHz}$ (aproximativ)- ceea ce înseamnă ca 25 de milioane de pixeli sunt procesați într-o secundă.

3.3 Exemple de controlere VGA

În continuare sunt descrise câteva exemple de controlere VGA implementate în limbajul VHDL, exemple ce se regăsesc pe larg explicate în cartea de referință [4]. Aceste exemple descriu o metodă aparte de generare a imaginilor proiectate pe ecran și asupra acestui aspect se vor concentra remarcile de mai jos. Modul de transmitere a semnalelor de culoare și sincronizare prin portul VGA sunt bazate pe principiul descris anterior, dar se va pune accentul pe metoda de prelucrare a pixelilor imaginii de afișat pe ecranul monitorului.

În capitolul 12.3 („*Overview of the pixel generation circuit*“) din [4] sunt descrise câteva metode prin care se determină configurația pixelilor prin cele 3 semnale RGB ale portului VGA. O primă metodă prezentată este aceea de a folosi o memorie pentru a stoca datele ce urmează a fi afișate pe ecran (bitii pentru fiecare pixel în parte). Aceasta poartă denumirea de bit-mapped scheme, ceea ce înseamnă că fiecare pixel al ecranului este mapat direct într-un cuvânt de memorie, adică va exista o relație de 1 la 1. Adresarea memoriei se va realiza la momente bine determinate de timp, calculate prin intermediul pixel-rate-ului și a semnalelor de sincronizare. Principala problemă a acestei abordări

constă în dimensiunea mare a memoriei, deoarece pentru un cuvânt sunt necesari 12 biti (bitii semnalelor de culoare ai pixelilor), iar pentru o imagine de rezoluție 640x480 va fi nevoie de o memorie de $640 \times 480 \times 12 = 3600$ Kb. Se va limita astfel numărul de imagini ce pot fi proiectate pe ecran din cauza resurselor de memorie disponibile pe placa de dezvoltare unde se integrează circuitul. Pentru a reduce spațiul de memorie, se pot folosi mai multe tehnici care vor duce de fapt la scăderea calității imaginii. Una dintre aceste tehnici, denumită tile-mapped scheme, se referă la grupări de pixeli ai ecranului cărora să le corespundă un singur cuvânt de memorie, deci nu va mai exista o relație 1:1. O altă tehnică este de a mapa doar anumite obiecte în memoria internă a placii de dezvoltare, obiecte ce vor fi proiectate pe ecran doar pe anumite zone, nu pe toată suprafața de 640x480 (exemplu în figura 3.3.1)

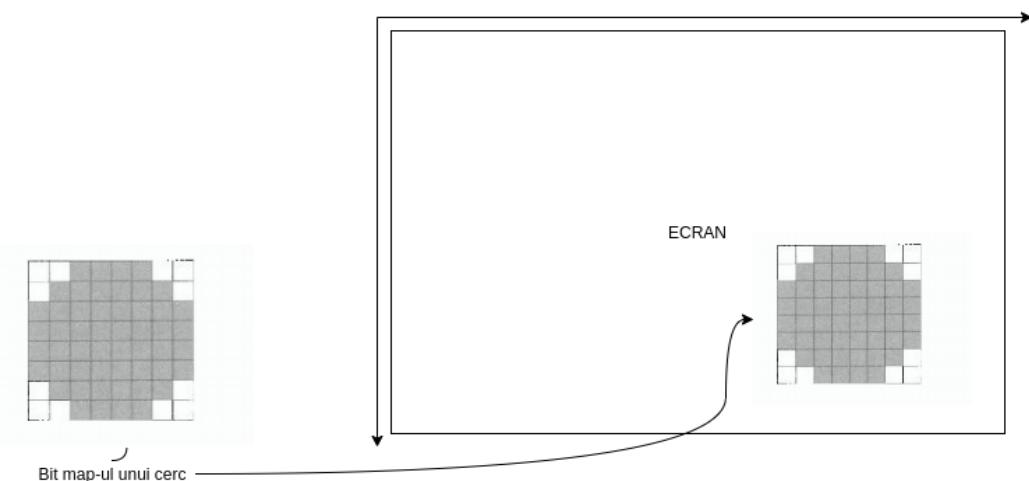


Figura 3.3.1 Maparea unui obiect în memorie și plasarea lui într-o zonă a ecranului

O altă metodă prin care să se genereze obiecte pentru a fi afișate se bazează pe descrierea coordonatelor limită de pe ecran. Adică se vor crea anumiți algoritmi de calcul pentru a determina dacă poziția curentă de pe ecran (poziție dată de procesul de scanare) corespunde interiorului sau exteriorului obiectului de proiectat. În capitolul 12.4.1 al lucrării de referință [4] este prezentat un model de joc denumit *pong*, ale căruia obiecte implicate în scenă sunt generate prin calcule ale limitelor de coordonate ce definesc obiectele pe ecran. Principalul dezavantaj al acestei abordări este faptul că acei algoritmi de calcul folosiți în determinarea obiectelor vor duce la arhitecturi hardware complexe care vor ocupa un număr mare de blocuri ale placii de dezvoltare FPGA și vor necesita interconexiuni complexe, ceea ce duce la riscul de a nu putea fi configurate și implementate.

3.4 Card Micro SD

Controllerul VGA propus pentru proiectul de față implică folosirea unui card micro SD pentru stocarea imaginilor ce se doresc să fie afișate pe ecran. Spre deosebire de metodele amintite mai sus, aceasta abordare permite aducerea datelor corespunzătoare pixelilor imaginii de pe cardul micro SD

într-o memorie RAM a circuitului, iar din aceasta memorie controllerul VGA va adresa configurația pixelilor, formând astfel cele 3 semnale de culori.

3.4.1 Descriere generală

Conform cu lucrarea de referință [3], cardurile de memorie SD sunt special construite pentru a respecta securitatea, capacitatea, performanța și cerințele necesare pentru consumatoarele audio/video ale dispozitivelor electronice. Astfel de carduri se folosesc în zilele noastre pentru mai multe facilități ce le pun la dispoziție, dintre care amintim dimensiuni mici vs. capacitatea de memorie mare, tensiuni de alimentare mici (valori maxime între 2.7 și 3.6 V), viteza de citire și scriere a datelor destul de mare de ordinul 104 MB/s pentru cele mai performante. Alte specificații cu privire la proprietățile diferitelor tipuri de carduri se regăsesc în capitolul 2 din [3].

3.4.2 Comunicare între micro SD și host

Ceea ce ne interesează în cadrul proiectului este cum se realizează comunicarea între cardul micro SD și host (în cazul nostru placă de dezvoltare nexys 4 DDR în cadrul sistemului controller VGA). Aceasta comunicare se bazează pe fluxuri de comenzi și date care sunt pornite și opriate prin biti de setare.

Elementele principale ale procesului de comunicare:

» **comanda:** un semn prin care se cere pornirea unei operații specifice; această comandă este inițiată de către host către un card (sau mai multe); transmiterea comenzi se realizează serial prin linia CMD

» **răspuns:** este semnalul transmis de card către host în urma receptionării unei comenzi; tot transmitere serială prin linia CMD

» **date:** au sens bidirectional: pot fi transmise de la card spre host și vice versa; sunt transmise prin liniile de date.

Transferul de date se realizează în blocuri. Blocurile sunt însoțite de o serie de biți (CRC) prin care se vor depista eventualele erori și informații despre procesul de citire sau scriere. Operația de citire de pe cardul SD se poate vizualiza în figura 3.4.1, preluată din capitolul 3.6 (*Bus protocol*). Tot în acest capitol sunt ilustrate și formatul token-ului de comandă și a celui de răspuns.

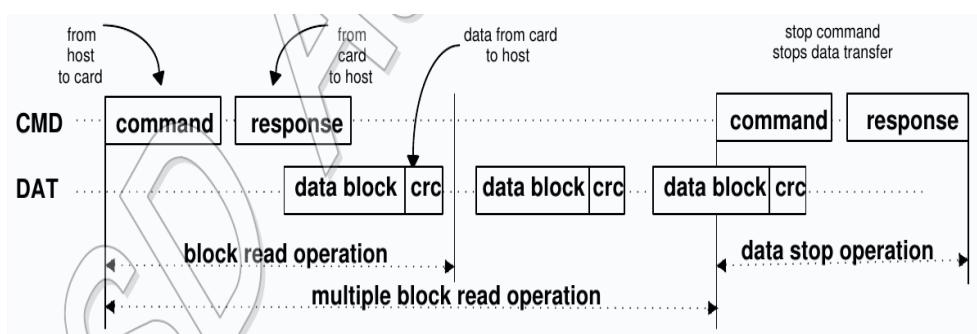


Figura 3.4.1 Read Operation

Pentru pornirea comunicării între host și cardul de memorie, hostul trebuie se reseteze cardul prin oprirea și repornirea tensiunii de alimentare, sau prin comanda GO_IDLE (CMD0). Principalul motiv pentru care se realizează aceasta operație este pentru a verifica dacă tensiunea furnizată de host este acceptată de cardul micro SD, de aceea trebuie consultată specificația cardului înainte de utilizarea acestuia.

Host-ul, în procesul de comunicare, reprezintă circuitul master, acesta trimite comenzi cardului de memorie.

3.4.3 Pinii cardului micro SD

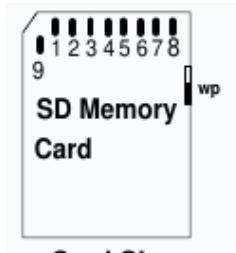


Figura 3.4.2 Pinii cardului micro SD

Tabel 3.1 Denumirea și descriere pinilor cardului micro SD

	Modul SD		Modul SPI	
Pin	Nume	Descriere	Nume	Descriere
1	CD/DAT3	Card Detect/ Data Line [bitul 3]	CS	Chip Select
2	CMD	Linia de comanda/răspuns	DI	Data In
3	VSSI	GND	VSS	GND
4	VDD	Tensiune alimentare	VDD	Tensiune alimentare
5	CLK	Clock	SCLK	Clock
6	VSS2	GND 2	VSS2	GND 2
7	DAT0	Data line [bit 0]	DO	Data out
8	DAT1	Data line [bit 1]	RSV	
9	DAT2	Data line [bit 2]	RSV	

3.4.4 Despre interfață SPI

Modul de comunicație al unei interfețe SPI este duplex, adică datele se pot transmite simultan în ambele direcții. Arhitectura interfeței este de tipul *master-slave*, existând un singur master (host-ul -controllerul VGA în cazul nostru) și mai multe dispozitive slave (în cazul proiectului prezentat în această lucrare există un singur slave, cardul micro SD). Masterul este responsabil pentru inițierea transferurilor și este cel care furnizează semnalul de ceas pentru sincronizare.

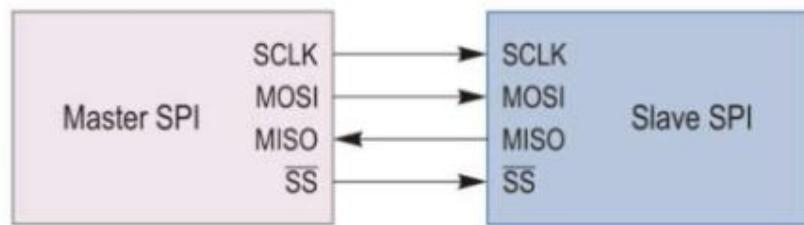
Principalele semnale ale interfeței SPI sunt:

»**SCLK** (Serial Clock) – semnalul de ceas generat de către host (master) și se folosește pentru recepția și transmisia datelor de către dispozitivul slave. O remarcă destul de importantă pe care o regăsim și în lucrarea de referință [7] este că dispozitivul master trebuie să cunoască numărul ciclurilor de ceas pe care trebuie să le genereze, astfel încât, dacă dispozitivul slave (cardul) trebuie să returneze date, acesta să poată returna în întregime toate datele cerute.

»**MOSI** (Master Output, Slave Input) – folosit pentru transmisia datelor master->slave

»**MISO** (Master Input, Slave Output) – utilizat pentru transmisia datelor slave->master

»**SS** (Slave Select) – utilizat de dispozitivul master pentru selecția dispozitivului slave cu care va comunica. În cazul proiectului prezentat, există doar un singur dispozitiv slave, cardul micro SD. Acest semnal este activ pe zero.



Următoare figura este preluată din [7] și reprezintă un tip de configurație SPI.

Figura 3.4.3 Configuratie a interfeței SPI cu un singur dispozitiv slave

Comunicarea între cele două dispozitive se realizează prin transferul serial al bițiilor între cele două componente. Pe perioada unui ciclu de ceas, dispozitivul master va transmite un bit pe linia MOSI, iar slave-ul citește de pe aceeași linie un bit. Tot în aceeași perioadă de ceas, slave-ul va transmite un bit, iar master-ul va citi un bit pe linia MISO.

3.4.5 Modul SPI pentru citirea cardurilor micro SD

Implementarea unui modul SPI pentru comunicarea cu un card micro SD va folosi doar o submulțime din protocolul cardului și din mulțimea de comenzi. Avantajul acestei interfețe este faptul că se va reduce la un efort minim comunicarea dintre cele două entități (host<→card de memorie). În capitolul 7 din lucrarea de referință [3] este descris pe larg SPI Bus Protocol și conține principalele comenzi folosite pentru comunicare.



Canalul SPI este byte oriented, ceea ce arată că fiecare comandă sau bloc de date este construit din 8-bit bytes și sunt aliniate la semnalul CS (mai precis, lungimea acestora este multiplă de 8 cicluri de ceas). În cazul cardurilor SDHC și SDXC, lungimea blocului de date este de 512 bytes.

În momentul pornirii, cardurile SD trec în modul SD, iar pentru a trece în modul SPI este nevoie ca semnalul CS să fie activat (0 logic) în timp ce se receptionează comanda de resetare CMD0. În momentul trecerii în modul SPI, cardul va transmite răspunsul R1.

Modul SPI suportă operații single block read și multiple block read (CMD17 sau CMD18). Toate comenziile au o lungime de 6 bytes. În subcapitolul 7.3 („*SPI Mode Transactions Packets*”) sunt specificate comenziile disponibile în modul SPI cu descrierile aferente ale acestora. Mai multe detalii despre comenziile disponibile interfetei SPI sunt prezentate în ANEXA 1 care este un extras din lucrarea [3].

Pentru o funcționare optimă a cardului micro SD, semnalul de ceas SCLK trebuie să aibă o frecvență cuprinsă între 100 și 400 kHz. Există mai multe formate pentru răspunsuri (R1, R2, R4, R5, R7) ce apar în momentul în care se iese vreo eroare. Configurarea acestor tipuri de răspunsuri este prezentată în subcapitolul 7.3.2 *Responses* din [3].

3.5 Alte module

Dintre modulele necesare realizării sistemului amintim:

3.5.1 Divizorul de frecvență

Circuit care are ca intrare un semnal de o anumită frecvență și furnizează un semnal de ieșire de o frecvență mai mică decât al aceluia de intrare. Un astfel de divizor se construiește cu ajutorul unui numărător și a factorului de scalare.

$$\text{Scale} = f(\text{in}) / f(\text{out}); \quad f(\text{in}) - \text{frecvența semnal intrare}$$
$$f(\text{out}) - \text{frecvența semnal ieșire}$$

Factorul de scalare va da numărul de impulsuri ale semnalului de intrare pentru o perioadă a semnalului de ieșire. Pentru a obține un semnal de ieșire cu factor de umplere 50% (jumate din perioadă să este ‘0’ logic, iar cealaltă jumătate este ‘1’ logic), număratorul intern al carui semnal de clock este semnalul de divizat, va schimba la fiecare scale/2 stări valoarea semnalului de ieșire (se va face toggle pe semnalul de ieșire).

De exemplu, pentru *pixel rate* o să fie nevoie de un astfel de circuit pentru a diviza semnalul de clock de 100MHz dat de placă Nexys 4 DDR la 25 MHz cât este nevoie pentru semnalul de procesarea a pixelilor.

3.5.2 Circuite de debounce

Cele mai cunoscute circuite de debounce sunt cele de tipul Mono Pulse Generator. Aceste tipuri de circuite asigură un singur semnal de enable în momentul apasării unui buton, pentru



eliminarea hazardurilor ce pot apărea din cauza degradării în timp a butoanelor. Se elimină astfel situația acelor zone critice în care butonul este sau nu apăsat și care ar produce mai multe impulsuri nedorite.

3.5.3 Automate de stare

Sunt circuite care descriu diagrama de stare a unui sistem. Sunt folosite în special pentru logica de control a sistemelor.

Sunt 2 tipuri de automate de stare:

- Automate Moore: ieșirile sunt funcții doar de starea prezenta
- Automate Mealy : au iesiri care pot să fie funcții atât de starea prezenta, cât și de intrările prezente.

3.6 Metode

3.6.1 Metode folosite pentru aplicarea unor efecte imaginilor proiectate

Având în vedere noțiunile prezentate mai sus despre portul VGA și semnalele ce vor transmise monitorului prin intermediul acestuia, știm că informația de culoare este formată din 12 biti, câte 4 biti pentru fiecare din cele 3 culori principale (roșu , verde și albastru). Aplicarea unor filtre imaginilor ce vor fi proiectate se rezumă la modificarea acestui sir de biti pentru obținerea efectelor dorite. Urmatoarele informații sunt preluate din [8].

Pentru obținerea efectului de grayscale avem 2 metode :

1. Media aritmetică a valorilor

$$\text{Grayscale} = (R + G + B)/3$$

sau

$$\text{Grayscale} = R/3 + G/3 + B/3;$$

E o metodă simplă, dar nu funcționează aşa cum ne aşteptăm datorită percepției diferite a ochiului uman pentru lungimile de undă a celor 3 tipuri de culoare.

2. Metoda ponderată

$$\text{Grayscale} = 0.299R + 0.587G + 0.114B;$$

Această abordare are în vedere ponderea culorilor în funcție de lungimea lor de undă.

Pentru obținerea efectului sepia, care da un aspect monocromatic de maro se poate obține astfel:

$$\begin{aligned}\text{sepiaRed} &= .393 * \text{originalRed} + .769 * \text{originalGreen} + .189 * \text{originalBlue} \\ \text{sepiaGreen} &= .349 * \text{originalRed} + .686 * \text{originalGreen} + .168 * \text{originalBlue} \\ \text{sepiaBlue} &= .272 * \text{originalRed} + .534 * \text{originalGreen} + .131 * \text{originalBlue}\end{aligned}$$



Se vor rotunji valorile obținute pentru a vea numere întregi.

Reflexia imaginii -se vor inversa pixelii imaginii astfel încât imaginea rezultat să fie oglindită.

3.7 Soluția propusă

În continuare este descrisă soluția pe care am propus-o pentru afișarea unor imagini pe monitor cu ajutorul plăcuței de dezvoltare Nexys 4 DDR. Datorită constrângerilor de memorie și complexitate a algoritmilor de procesare a pixelilor în cadrul arhitecturii sistemului, pentru ca sistemul să permită afișarea mai multor imagini la comanda utilizatorului, se recurge la folosirea unei memorii externe cum este un card microSD unde să se stocheze informația pixelilor imaginilor. Nexys 4 DDR dispune de un slot micro SD, iar interfata de comunicare dintre cele două entități este cea SPI. Citirea informației de pe cardul micro SD va fi stocată temporar într-o memorie RAM, ca apoi să fie procesată și transmisă spre portul VGA, monitor.

Alegerea interfeței SPI pentru comunicarea dintre placa de dezvoltare și cardul micro SD a constat în simplitatea acestei abordări. Sistemul master va fi reprezentat de placa de dezvoltare, iar slave-ul va fi cardul micro SD.

S-a ales o rezoluție a imaginii de 640x480 cu frecvența de refresh 60 Hz. În cazul acestei rezoluții se va avea în vedere frecvența de procesarea a pixelilor, pe care am determinat-o ca fiind de aproximativ 25 MHz. Descrierea Hardware pentru modulul ce va genera semnalele de sincronizare (hsync și vsync) va conține constante și generice pentru reutilizarea codului în cazul în care se dorește o rezoluție mai mare a imaginii. S-a ales rezoluția aceasta pentru siguranța sintetizării și configurației memoriei RAM, responsabilă cu stocarea temporară a imaginii citite de pe cardul micro SD.

Pentru a scoate configurația culorii de 12 biti per pixel a imaginilor se va folosi un script în limbajul de programare Python. Acest script va lua o imagine de tipul .png sau .jpg și va genera un fisier .bin cu biții imaginii. Acest fisier .bin va fi adăugat pe cardul micro SD, iar cu ajutorul programului HxD se va formata cardul microSD și se va urmări adresa blocului de început a imaginilor stocate pe acesta.

Pentru a include o parte interactivă, sistemul va avea și un modul de procesare a pixelilor, prin care se pot aplica anumite filtre predefinite imaginii. Se vor aplica urmatoarele filtre: grayscale și sepia și în plus cu ajutorul switch-urilor de pe placa vor putea crește gradul de culoare aplicat pixelilor. De asemenea, utilizatorul va putea să deplaceze imaginea pe ecran pe cele 2 coordonate (x și y) cu ajutorul butoanelor.

Pentru descrierea logicii de comandă a sistemului, a generării semnalelor de sincronizare și pentru realizarea interfeței SPI se vor folosi automate de stare, deoarece este mai ușor de înțeles și descrie sistemul din punct de vedere al stărilor în care se poate afla la un moment dat.

3.8 Scenarii de utilizare propuse

Etapele corespunzătoare procesului de determinarea a cazurilor de utilizare constau în elicitarea cerințelor (descoperirea cerintelor sistemului) și specificarea cerintelor (implica modelarea riguroasa a cerintelor : diagrame UML, use-case).

Principalele cerințe pe care trebuie să le îndeplinească sistemul pe baza problemei puse sunt:

- proiectarea unor imagini pe ecranul unui monitor cu ajutorul placii de dezvoltare NEXY 4 DDR
- folosirea unui card microSD pentru stocarea imaginilor
- construirea ierarhică a sistemului

Figura următoare este o reprezentare use-case UML care capturează grafic actorii sistemului, cazurile de utilizare și relațiile dintre acestea. Actorii sunt orice tip de utilizator.

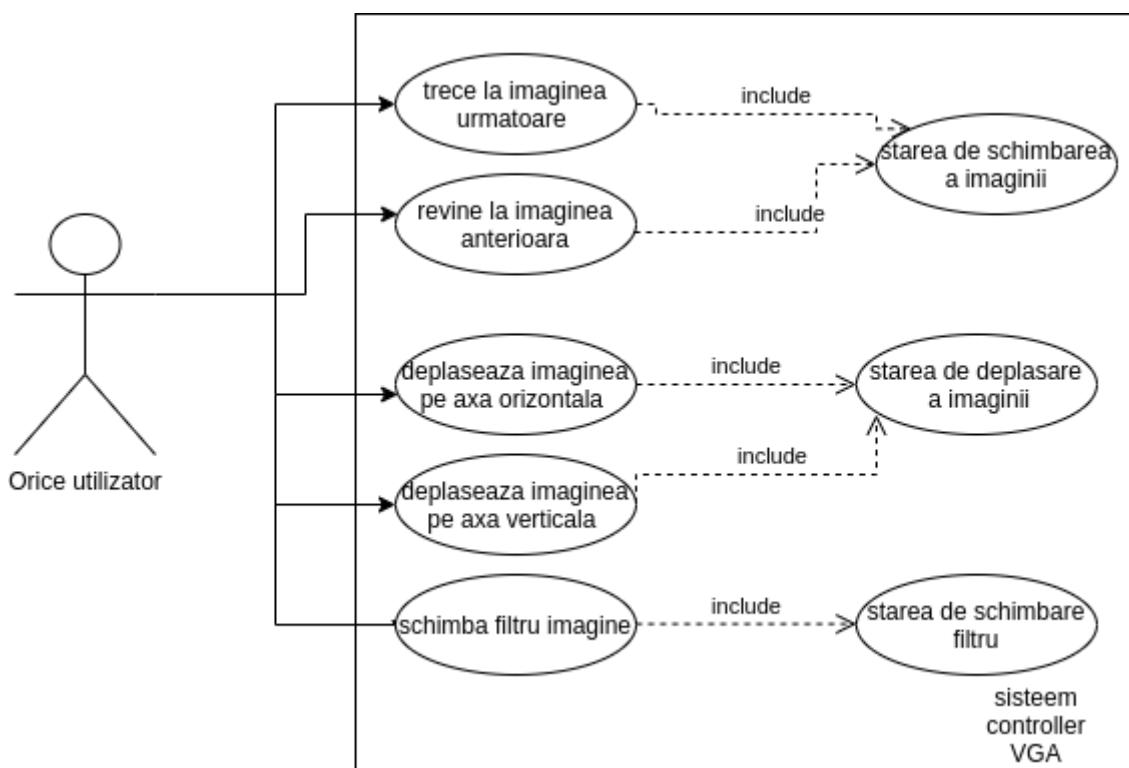


Figura 3.8.1 Use-Case

Scenarii de utilizare ale sistemului

Use case: schimbă imaginea

Actor principal: utilizator

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul din mijloc până când ledul 0 de pe placă este aprins. Aceasta va indica faptul că sistemul se află în starea în care utilizatorul poate schimba imaginea de afișat.



2. Utilizatorul apasă butonul din stânga pentru a trece la imaginea anterioară sau butonul din dreapta pentru a trece la următoarea imagine.

Use case: deplasează imaginea pe axa orizontală

Actor principal: utilizator

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul din mijloc până când ledul 1 de pe placă este aprins. Aceasta va indica faptul că sistemul se află în starea în care utilizatorul poate să deplaseze imaginea pe ecranul monitorului.
2. Utilizatorul apasă butonul din stânga pentru a mișca imaginea spre stânga sau butonul din dreapta pentru a deplasa imaginea în partea dreaptă

Use case: deplasează imaginea pe axa verticală

Actor principal: utilizator

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul din mijloc până când ledul 1 de pe placă este aprins. Aceasta va indica faptul că sistemul se află în starea în care utilizatorul poate să deplaseze imaginea pe ecranul monitorului.
2. Utilizatorul apasă butonul up pentru a mișca imaginea în sus pe ecran sau butonul down pentru a deplasa imaginea în josul ecranului.

Use case: schimbare efect imagine

Actor principal: utilizator

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul din mijloc până când ledul 2 de pe placă este aprins. Aceasta va indica faptul că sistemul se află în starea în care utilizatorul poate să aplice un filtru imaginii de proiectat.
2. Pentru a aplica filtre predefinite, utilizatorul va putea activa primele 3 switch-uri. Primul switch – va aplica grayscale, al doilea switch va aplica sepia, iar cu urmatoarele 12 switch-uri se va putea mari aportul de culoare RGB.

Din cauza resurselor limitate (doar 4 butoane disponibile) a fost necesară separarea condițiilor de funcționare în stări diferite, chiar dacă se putea să se facă toate operațiile în aceeași stare (să schimbi imaginea și apoi să îi schimbi filtrul fără a necesita apăsarea butonului pentru a ajunge la o stare validă de modificare a filtrului).

4. Proiectare și implementare

4.1 Arhitectura

Corelând cerințele problemei cu scenariile de utilizare propuse, se poate contura o imagine de ansamblu a sistemului în funcție de semnalele de intrare și ieșire. Numim *black box* această imagine



de ansamblu pe care o avem asupra sistemului. Din scenariile de utilizare menționate în capitolul anterior remarcăm necesitatea unor resurse de pe placa Nexys 4 DDR : cele 5 butoane (pentru operațiile de modificare a poziției imaginilor, de navigare printre imaginile disponibile) , cel puțin 3 switch-uri (pentru aplicarea filtrelor), portul VGA (pentru conexiunea cu monitorul), 3 led-uri (ce indică starea sistemului cu operațiunea ce se poate execuța), slotul micro SD pentru conectarea cardului de memorie în care vor fi stocate imaginile.

Urmărind fișierul de constrângeri al plăcii, precum și manualul de referință al acesteia se pot determina semnalele de ieșire ale portului VGA (2 semnale de sincronizare și 3 semnale de 4 biți pentru cele trei culori) și de asemenea semnalele de ieșire pentru slotul micro SD prin care se va realiza comunicarea de tip SPI.

Black box -ul sistemului descris în documentul de față este:

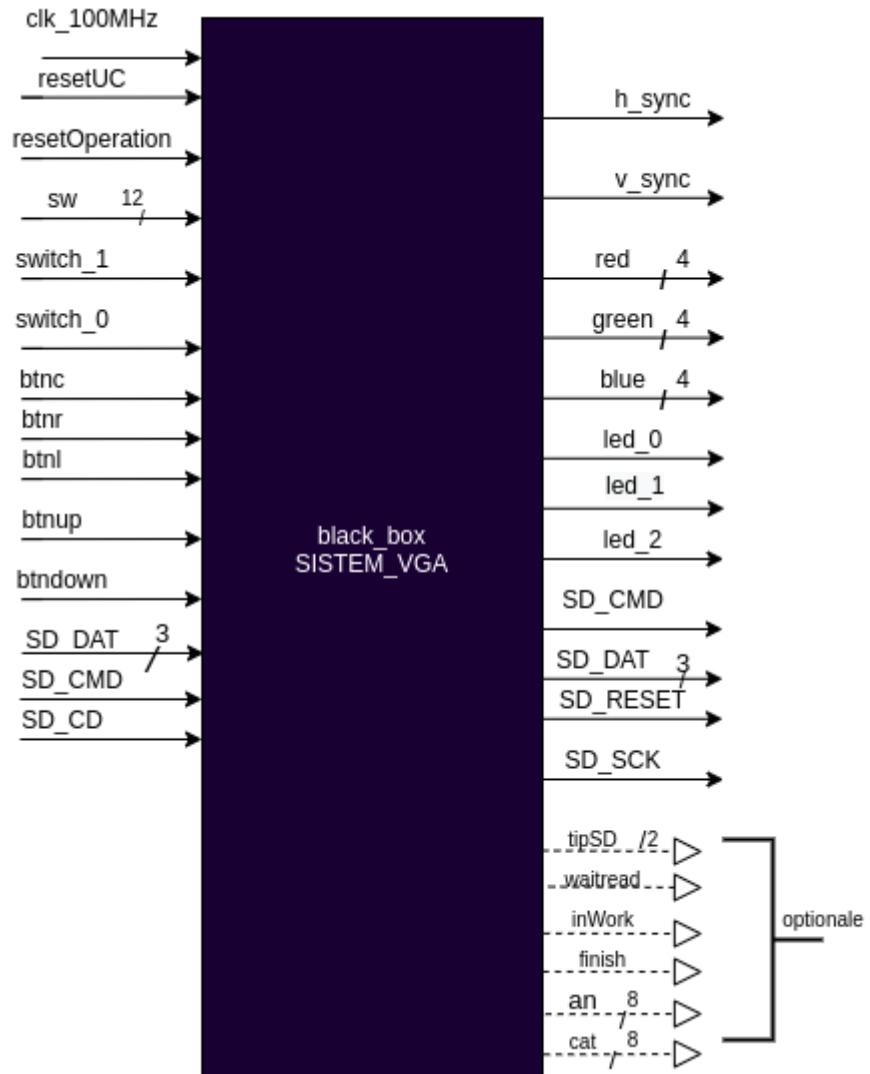


Figura 4.1 Black box-ul sistemului

Prin definirea mai multor cazuri de utilizare a sistemului VGA se poate observa necesitatea unei logici de control prin care sistemul să își cunoască starea curentă și în funcție de aceasta să se execute anumite operații. Astfel, sistemul poate fi împărțit în două unități: unitatea de control și unitatea de execuție. Aceasta descompunere ierarhică a sistemului simplifică logica de proiectare și implementare a sistemului, precum crește gradul de reutilizare și posibilitatea de modificare în cazul care se dorește introducerea unor noi funcționalități, sau modificarea celor curente. Trecerea dintr-o stare în alta a unității de control se realizează prin apăsarea butonului **btnc**. Ieșirile unității de control vor fi semnale de validare a unor operații ce se vor executa în unitatea de execuție, iar starea în care se află sistemul va fi semnalizată prin aprinderea ledului asociat. Unitatea de execuție va cuprinde principalele module ce se vor ocupa de generarea semnalelor de sincronizare pentru portul VGA ,

determinarea culorii pixelilor la momente de timp bine definite, implementarea interfeței de comunicare SPI între placă și cardul micro SD.

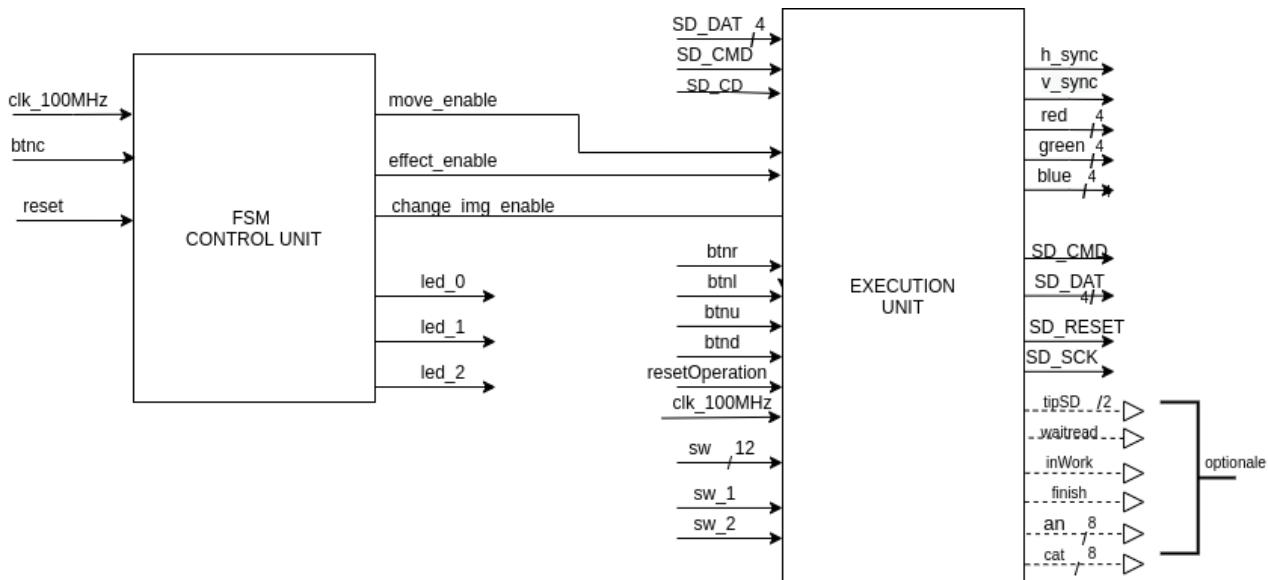


Figura 4.2 Unitate de control și unitatea de execuție a sistemului

Unitatea de control se poate implementa cu un automat de stare de tip Moore. În cadrul unității de execuție vor exista mai multe module care vor fi prezentate în detaliu în următoarele subcapitole. Din figura 4.3 se pot determina principalele module ale unității de execuție:

1. **Sync_generator** – este circuitul responsabil pentru generarea semnalelor de sincronizare h_sync și v_sync pentru a determina timpii corecți în procesul de scanare a monitorului. Acest modul este cât mai generic implementat pentru a se putea realiza cât mai ușor modificările ulterioare, precum schimbarea rezoluției de lucru. Sunt folosite automate de stare pentru descrierea scanării pe verticală și pe orizontală, stările fiind reprezentate de regiunile posibile ale procesului, tinând cont de timpii fiecarei regiuni în parte. Circuitul va genera un semnal de *enable_image*, semnal care va fi activ doar în momentul în care ambele automate de stare sunt în starea ce descrie regiunea de display, atât pe verticală, cât și pe orizontală. Cu ajutorul numărătoarelor interne, ale căror semnal de ceas are frecvența pixel_rate (25 MHz pentru rezolutia de 640x480), se va putea determina poziția pixelului ce trebuie afisat pe ecran (*addr_x* și *addr_y*).

2. **Offset_counter** – acest circuit are ca intrări butoanele left, right, up și down și semnalul de move_enable de la unitatea de control. Acesta va fi format din două numărătoare, un numărător pentru fiecare dintre cele două axe de coordonate. Prin incrementarea și decrementarea celor două numărătoare prin apasarea butoanelor, se va determina un offset pentru axa X și un offset pentru axa Y, ce vor reprezenta valorile cu care se va deplasa imaginea pe ecran și de asemenea direcția.

3. **Memory_address** – acest modul are drept rol combinarea semnalelor venite de la sync_generator (poziția curentă din procesul de scanare a ecranului), cu valorile de offset venite de la offset_counter și va determina adresa ce trebuie accesată din memoria RAM pentru a lua culoarea pixelului ce va fi afișat la momentele de timp corespunzătoare. În acest modul se face și reflexia imaginii.

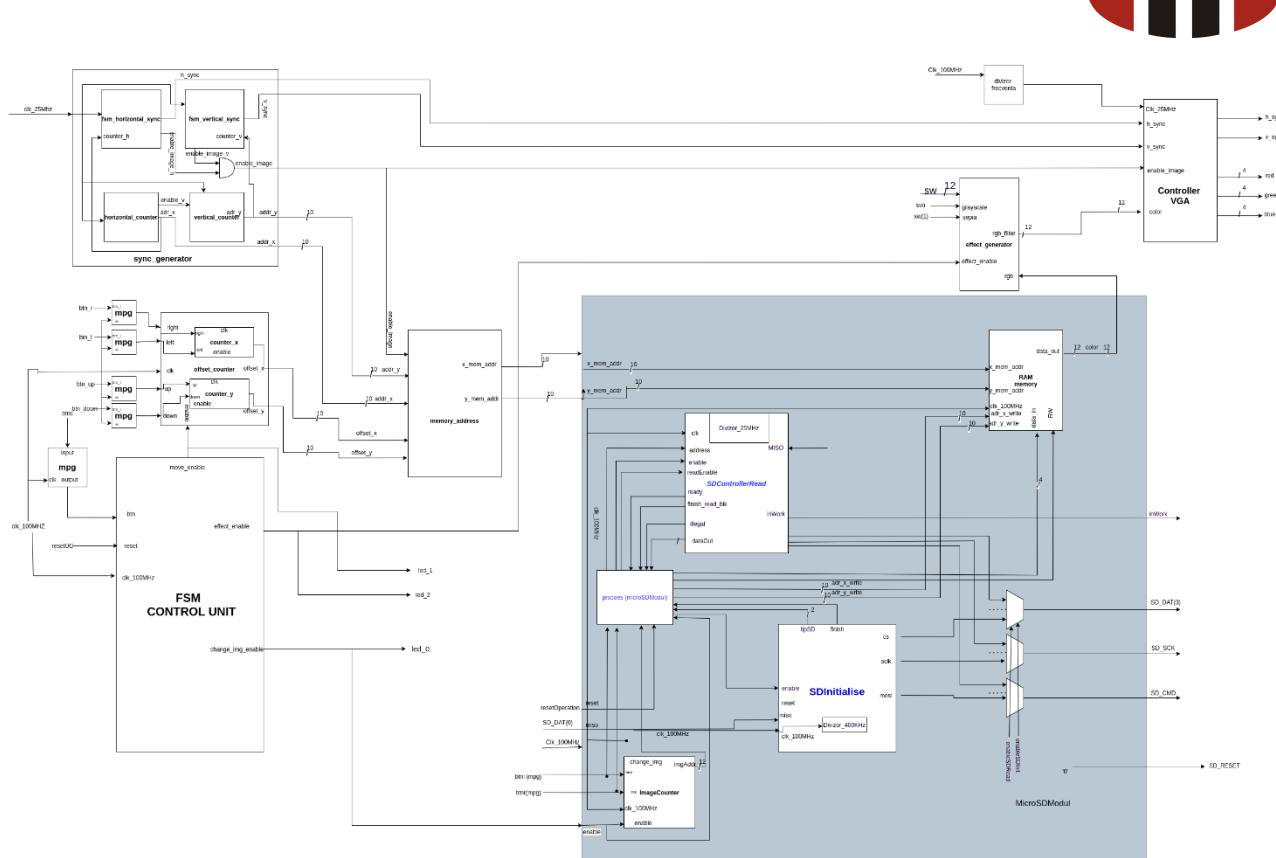


Figura 4.3 Schema bloc a sistemului

4. Effect_generator – scopul acestui modul este de a aplica efecte prestabilite prin selectarea celor 2 switch-urii sau aportului de culoare rgb prin activarea a 12 switch-urii.

5. Controller_Vga – are ca intrări semnalul de ceas de 25 MHz , h_sync, v_sync și enable_image de la sync_generator și semnalul de culoare pe care o să le treacă printr-un circuit de pipeline cu scopul de a anula problemele de timing.

6. MicoSDModul – va fi unul dintre cele mai importante module ale sistemului. Acesta se va ocupa cu citirea imaginilor de pe cardul micro sd și stocarea lor temporară în memoria RAM. Pentru implementarea acestui modul va fi nevoie de un fsm special care va comanda operațiile dorite pe microSD. În primă fază, fsm-ul trebuie să se ocupe de inițializarea cardului microSD în modul SPI. Pentru aceasta s-a creat o entitate separată care se va ocupa de această operație, **SDInitialise**. După inițializarea cardului și verificarea eventualelor erori, fsm-ul se va ocupa de logica de citire a unei imagii din memoria SD, astfel că va comanda o entitate nouă care va citi blocuri de memorie de pe cardul microSD, iar apoi acea informație va fi salvata temporar într-o memorie **RAM** care va stoca configurația imaginii curente afișată pe ecran. **SDControllerRead** este entitatea care va citi un bloc de memorie și va informa host-ul când se va termina această operație cu scopul prelucrării datelor citite.

4.2 Control Unit

Acest circuit va furniza logica de control a sistemului. Potrivit scenariilor de utilizare sistemul are trei funcționalități principale: schimbarea imaginii, modificarea poziției imaginii pe ecran și aplicarea unor efecte asupra imaginilor. Pentru ca sistemul să execute una dintre aceste operații este nevoie de o logică de control prin care se vor determina semnale de enable ce vor activa execuția unor module specialize pentru operațiunea dorită.

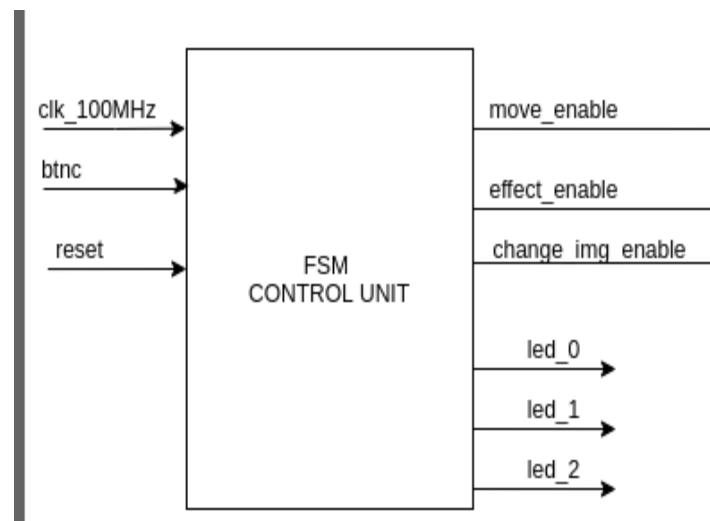


Figura 4.2.1 Schema bloc a unitatii de control

Intrări:

- semnalul de clk de 100 MHz de la placa Nexys 4 DDR, pentru a sincroniza trecerea dintr-o stare în alta
- semnalul btnc : prin activarea acestui semnal se va trece la starea următoare. Acest semnal va fi sincronizat cu semnalul de ceas deoarece vine de la un circuit MPG (Monopulse Generator) prin care se preia doar un singur impuls de la butonul plăcii la apăsarea acestuia.
- semnalul reset (resetUC) : reprezintă semnalul de reset care aduce sistemul în starea inițială. Resetarea automatului se va face sincron.

Ieșiri:

-*move_enable*: este semnalul care va activa numărătoarele din modulul offset_counter prin care se va modifica poziția imaginii pe ecran. Acestui semnal îi corespunde și ieșirea led_1 prin care se va aprinde ledul corespunzător acestei operații de pe placă.

-*change_img_enable*: este semnalul care va activa numărătorul counter_img_address din modulul image_controller și prin care se va permite schimbarea imaginii de afisat pe ecran. Led_0 este semnalul asociat.

-*effect_enable*: este semnalul care va activa modulul effect_generator pentru posibilitatea aplicării unor efecte imaginii proiectate. Acestui semnal îi corespunde și ieșirea led_2

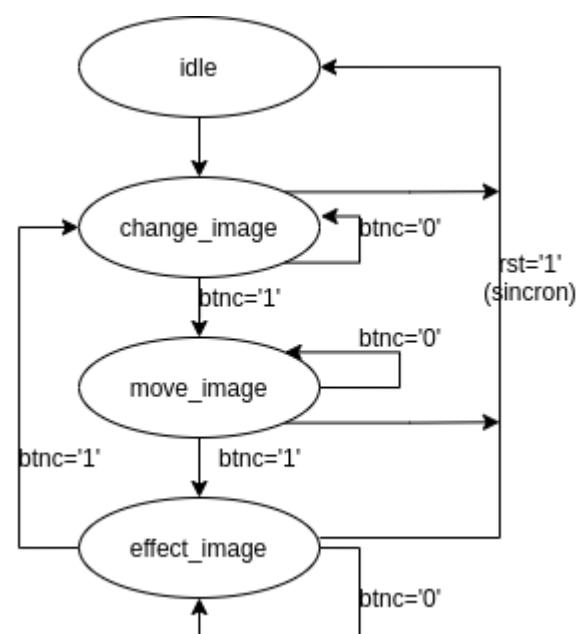


Figura 4.2.2 Automatul de stare al unității de control

Această unitate se va implementa cu ajutorul unui automat de stare care are ca stări principale: change_image, move_image și effect_image. Ieșirile depend numai de starea curentă ceea ce arată că automatul este unul de tip Moore.

Tabel 4.1 Tabelul stărilor și ieșirilor

Stare	<i>change_im_g_enable</i>	<i>led_0</i>	<i>move_enable</i>	<i>led_1</i>	<i>effect_enabl_e</i>	<i>led_2</i>
<i>idle</i>	0	0	0	0	0	0
<i>change_image</i>	1	1	0	0	0	0
<i>move_image</i>	0	0	1	1	0	0
<i>effect_image</i>	0	0	0	0	1	1

Implementarea acestui modul s-a realizat prin definirea unui tip enumerat pentru stări, iar pentru tranziția dintr-o stare în alta s-a folosit un singur process. Pentru logica ieșirilor s-a folosit de asemnea un process ce are ca sensibilitate starea curentă și în funcție de aceasta se determină valorile ieșirilor.

4.3 Divizor de frecvență

Această componentă are ca rol divizarea frecvenței semnalului de intrare și furnizarea noului semnal spre ieșire. În sistemul VGA este nevoie de o asemenea componentă pentru a diviza semnalul de ceas de la placa Nexys 4 DDR care are o frecvență de 100 MHz pentru obținerea unui semnal de ceas de frecvență 25 MHz, ce reprezintă pixel_rate-ul necesar procesului de scanare a ecranului. În corespondență cu aspectele teoretice menționate în capitolul 3, pentru construcția unui astfel de circuit avem nevoie de o valoare numită *scale* care se calculează astfel:

$$\text{scale} = f(\text{in}) / f(\text{out}); \quad f(\text{in}) = 100\text{MHz} \quad f(\text{out}) = 25\text{MHz}$$

$$\Rightarrow \text{scale} = 100/25 = 4;$$

Aceasta conduce la faptul că vor fi necesare 4 impulsuri ale semnalului de intrare pentru o perioadă a semnalului de ieșire. Pentru a obține un semnal de ieșire cu factor de umplere 50% (jumate din perioada sa este ‘0’ logic, iar cealaltă jumătate este ‘1’ logic), se va face toggle pe semnalul de ieșire la $4/2=2$ impulsuri de ceas ale semnalului de intrare.

Implementarea unui astfel de divizor se poate face printr-un numărător pe 2 biti, al cărui semnal de ceas este clock-ul de 100 MHz. Deoarece stările prin care trece numărătorul sunt (00->01->10->11->00...) se observă că bitul cel mai semnificativ

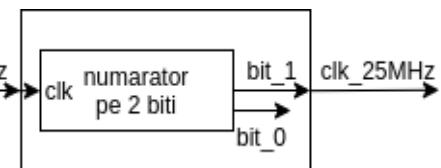


Figura 4.3.1 Schema RTL a divizorului

se va schimba din două în două impulsuri de ceas. Având această proprietate, se poate lua direct acest bit ca semnal de ieșire.

S-a creat și un divizor de frecvență general al cărui cod se află în anexa 3.

4.4 Sync_generator

Rolul acestei componente este de a furniza semnalele de sincronizare pe verticală și orizontală (vsync și hsync), dar și determină poziția pixelilor ce urmează a fi transmisi ecranului la momentele de timp corespunzătoare. În continuare va fi explicat modul de construcție al acestui modul cu componente sale. Realizarea acestuia are ca bază aspectele teoretice amintite în capitolul 3, unde s-au descris fazele sau regiunile prin care trece procesul de scanare a ecranului, care vor fi esențiale pentru determinarea poziției pixelilor pe ecran, dar și pentru generarea semnalelor de hsync și vsync. Amintim pentru orizontală că pixelii imaginii trebuie să fie transmiși numai când scanarea se află în *regiunea de display*, iar în zonele *horizontal retrace*, *back porch* și *front porch* nu se transmite niciun pixel. Semnalul **h_sync** va primi polarizarea specifică rezoluției numai în perioada de horizontal retrace. Similar cu scanarea pe orizontală, se comportă și scanarea pe verticală, în care transmiterea unui pixel de imagine este condiționată de *regiunea de display*, în fazele *vertical retrace*, *front porch* și *back porch* fiind dezactivată această opțiune. Polarizarea specifică rezoluției pentru semnalul **vsync** va fi aplicată numai în regiunea de vertical retrace.

Intrări:

-semnalul de ceas divizat pentru rezoluția de lucru (pentru 640/480 semnalul are o frecvență de 25 MHz).

Ieșiri:

-**vsync** -pentru polarizare în timpul scanării pentru pe verticală. Acest semnal va fi transmis la portul VGA

- **hsync** -pentru polarizare în timpul scanării pentru pe orizontală. Acest semnal va fi transmis la portul VGA

-**addr_x** – va da poziția de pe ecran a pixelului curent (de fapt, va da evoluția pe orizontală a scanării, care include și timpii morți în care nu se afișează nimic, această situație fiind rezolvată prin semnalul enable_image)

-**addr_y** – va da poziția de pe ecran a pixelului curent

-**enable_image**: este semnalul care validează transmiterea unui pixel spre ecran

Schema circuitului:

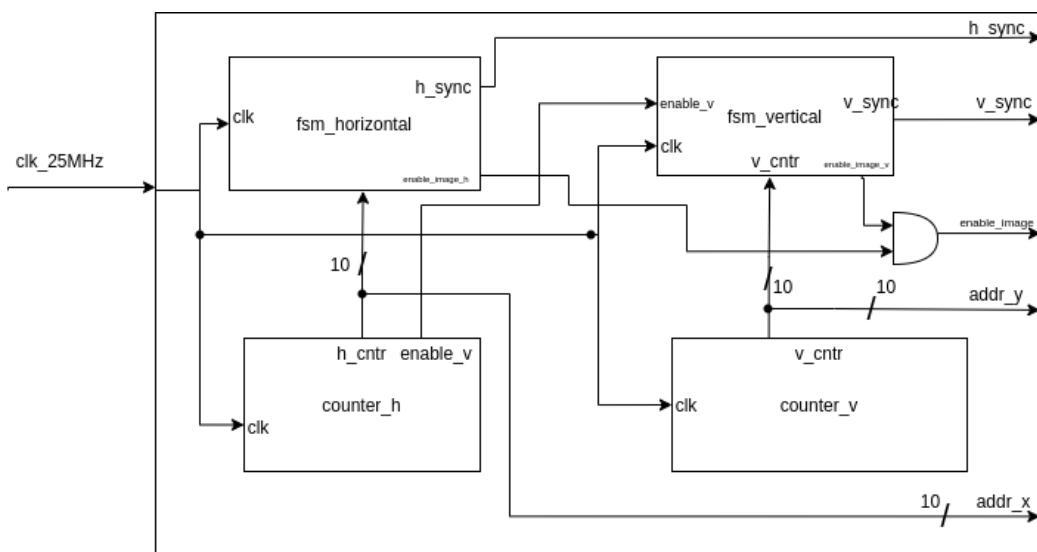


Figura 4.4.1 Schema RTL pentru Sync Generator

Idea realizării circuitului constă în procesul de scanare a ecranului, timpul de procesare al unui pas din scanare fiind dat de pixel rate-ul rezoluției, în cazul nostru fiind de 25 MHz. Se folosesc două numărătoare prin care se vor determina regiunile menționate mai sus pentru scanarea pe orizontală și scanarea pe verticală.

Pentru descrierea zonelor de scanare s-au folosit două automate de stare, ale căror stări vor descrie zonele din timpul scanării pe orizontală, și respectiv pe verticală. În figura 4.4.2 este prezentat automatul de stare pentru scanarea pe verticală. Se observă că tranziția între stări se face când numărătorul pentru scanare depășește limita impusă pentru fiecare regiune.

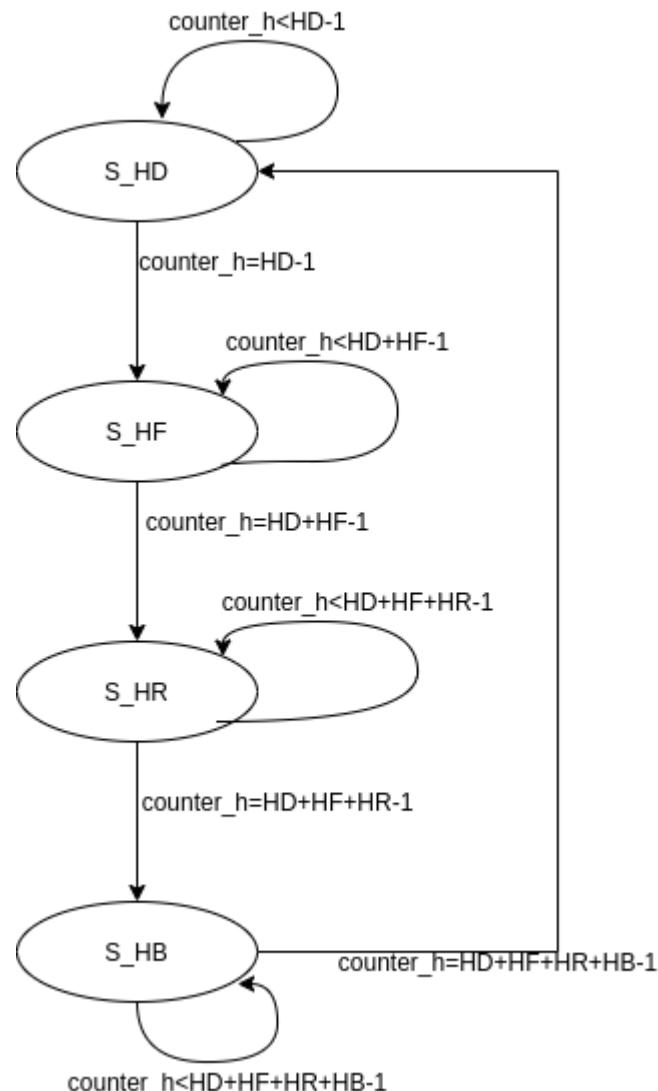


Figura 4.4.2 Automat stare pentru orizontală

HD, HF, HR și HB reprezintă timpii fiecărei regiuni. Aceștia diferă în funcție de rezoluția aleasă, de aceea am creat un pachet special pentru definirea acestor timpi ca și constante **VgaPackage.vhd**, pentru eventuale modificări și actualizări ale sistemului. Semnalele de ieșire hsync și enable_image depind de starea în care se află scanarea pe orizontală. Astfel, hsync va fi polarizat cu valoarea H_pol specifică rezoluției doar în regiunea S_HR (horizontal retrace), iar în rest valoarea acestuia va fi not(H_pol). Semnalul enable_image va fi condiționat de regiunea de display dată de starea S_HD, în celelalte regiuni nu trebuie transmiși pixeli spre ecran deoarece sunt timpii morți în care scanarea trece de la un rând la altul.

Tabel 4.4.1 Tabelul iesirilor

Stare	<i>hsync</i>	<i>enable_image_h</i>
S_HD	<i>not(H_POL)</i>	1
S_HF	<i>not(H_POL)</i>	0
S_HR	<i>H_POL</i>	0
S_HB	<i>not(H_POL)</i>	0

Numărătorul asociat scanării pe verticală este condiționat de starea numărătorului pe orizontală. Incrementarea acestuia se realizează în momentul în care se termină un rând de scanat, adică în momentul în care numărătorul pe orizontală atinge limita maximă și se resetează. Pentru delimitarea regiunilor pe verticală s-a folosit din nou un automat de stare figura 4.4.3.

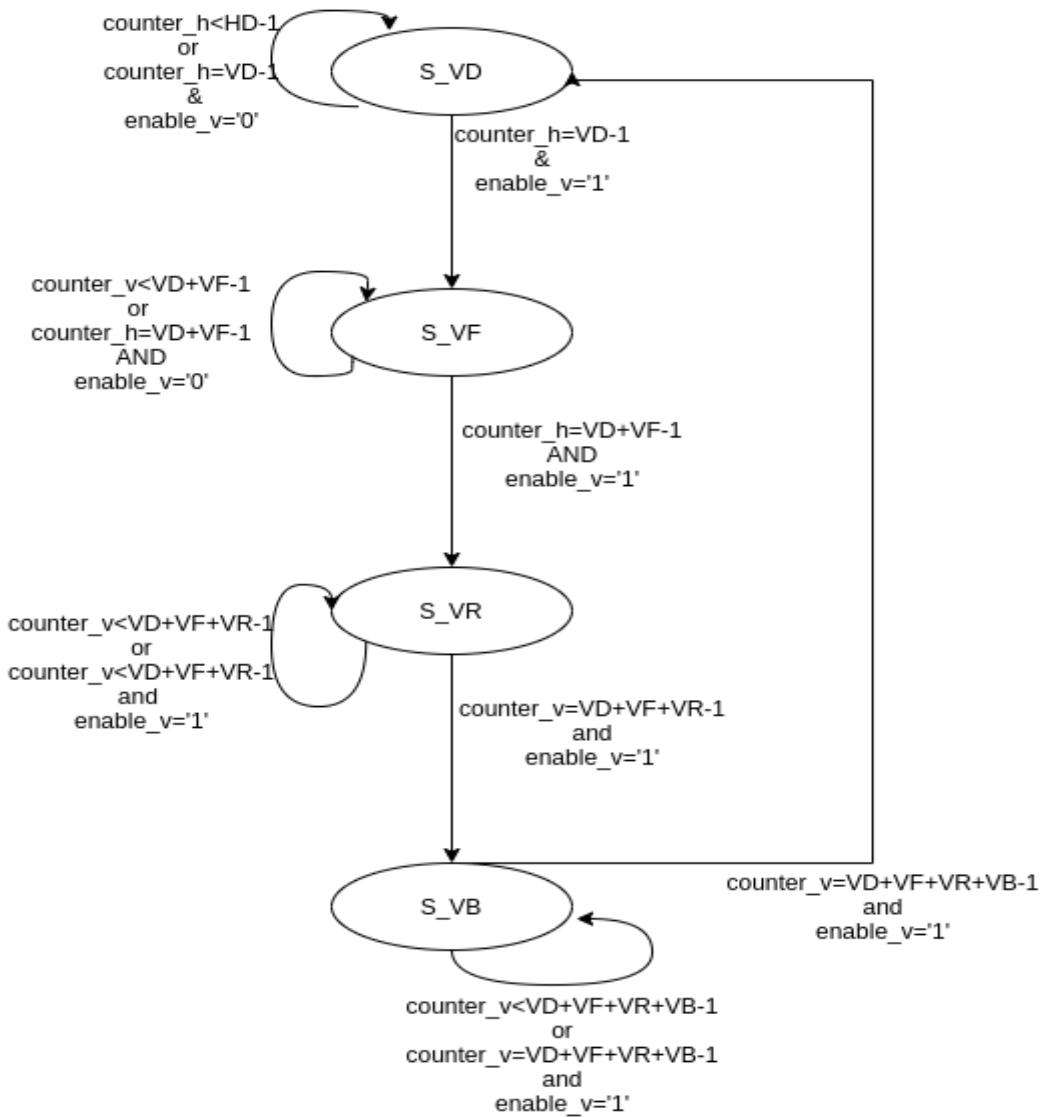


Figura 4.4.3 Automat stare pe verticală



În plus fata de logica folosită pentru automatul de stare pe orizontală avem semnalul *enable_v* venit de la numaratorul pe orizontală. Acesta va fi activ în momentul în care se ajunge la limita superioară pentru acest numarator. De ce e nevoie de un astfel de semnal și nu e dde ajuns sa punem doar condițiile pe *counter_v*? Răspunsul este ca pentru ca o stare a numaratorului pe verticală vor trece mai multe impulsuri de ceas (pană când se procesează un rând din scaanare), iar tranzitia va fi data de acel semnal de *enable_v*. Logica iesirilor este similară cu cea a automatului pe orizontală. Transmiterea pixelilor spre monitor este validată doar în regiunea de display (S_VD), iar polarizarea semnalului vsync cu valoarea specifică rezoluției alese se va aplica doar în regiunea vertical retrace (S_VR). În continuare e prezentat tabelul iesirilor în funcție de stare

Tabel 4.4.1 Tabelul ieșirilor

Stare	<i>vsync</i>	<i>enable_image_v</i>
S_VD	<i>not(H_POL)</i>	1
S_VF	<i>not(H_POL)</i>	0
S_VR	<i>H_POL</i>	0
S_VB	<i>not(H_POL)</i>	0

Semnalul *enable_image* va fi activ doar în momentul în care ambele automate de stare se află în starea specifică regiunii de display. Astfel acest semnal se poate obține printr-o poartă SI cu intrările *enable_image_v* și *enable_image_h*. Semnalele *addr_x* și *addr_y* sunt pe 10 biți deoarece pentru o rezoluție de 640/480 avem nevoie de cel puțin 10 biti pentru a reprezenta aceste valori în binar.

4.5 Modulul MPG

Acest modul reprezintă un circuit de debounce, pentru a anula anomaliiile ce pot apărea în momentul apăsării unui buton. Dacă butonul este foarte uzat, apăsarea lui va genera mai multe impulsuri decât ne-am dori, deoarece el poate oscila între valori de ‘1’ și ‘0’ până când se va stabiliza. Pentru a corecta un astfel de comportament, s-a decis crearea unui circuit special care se va ocupa de acest aspect, astfel încât, de la buton se va lua doar un impuls de la buton, iar valoarea acestui semnal va dura doar o perioadă de ceas.

Intrări

-**clock**: semnalul de clock de 100 MHz

-**btn**: semnalul venit de la buton, de la care se ia doar un singur impuls egal cu o perioadă de ceas

Iesiri

-**enable**: semnalul extras de la apăsarea butonului, care rămâne activ doar o perioadă de ceas chiar dacă se menține apăsat butonul

Schema RTL

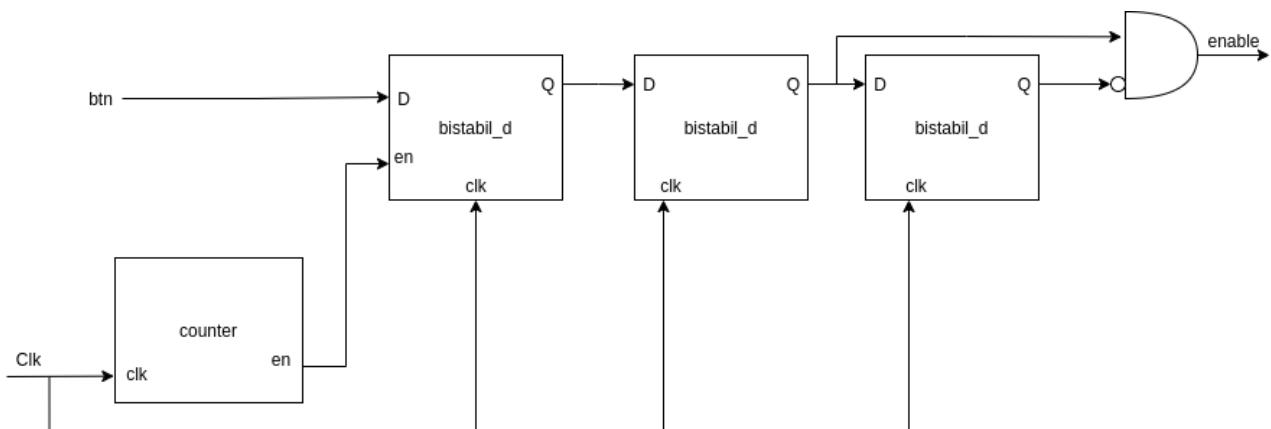


Figura 4.5.1 Schema RTL a circuitului

Pentru realizarea acestui modul este nevoie de 3 bistabile și un numărător. Numărătorul este pe 16 biti, și va genera un semnal de enable doar în starea xFFFF, ceea ce determină faptul că va verifica starea butonului doar la 655.350 ns (la un semnal de clock de 100MHz). Astfel reduce riscul de a se face citiri ale butonului când acesta oscilează între stări. Folosirea celor trei bistabile va duce la generarea unui singur impuls cu o durată egală cu a semnalului de clock în momentul în care se citește valoarea de la buton. De exemplu, la o tranziție a butonului de la starea '0' la '1', care va fi prinsă de numărător, în primul bistabil se va încărca valoarea '1'. Apoi la următorul ciclu de ceas, pe bistabilul 2 se va încărca valoarea '1', iar enable-ul va fi '1' în acest caz, fiindcă bistabilul 3 e în starea '0' logic. La următorul impuls de ceas se schimbă lucrurile, fiindcă valoarea de '1' ajunge pe ultimul bistabil ce va cauza o intrare de 0 logic pentru poarta SI în care se determină semnalul enable-ul. Această remarcă descrie comportamentul circuitului, în care se va păstra doar un ciclu de ceas semnalul venit de la buton, și care va fi sincronizat cu semnalul de ceas.

4.6 Modulul Offset Counter

Rolul acestei componente este de a seta offset-ul cu care se poate deplasa imaginea pe ecran cu ajutorul celor 4 butoane disponibile pe placă Nexys4 DDR. Este formată din două numărătoare pentru deplasare pe orizontală și deplasare pe verticală, iar acestea vor număra doar în intervalul regiunilor de display (orizontal – HD stări, vertical -VD stări). Semnalele de la butoane vor fi trecute mai întâi print-un MPG, iar efectul lor asupra numaratoarelor va fi decrementarea sau incrementarea valorilor cu pe orizontală și pe verticală.

Intrări

- clk**: semnalul de clock de 100 MHz pentru numărătoare
- left, right** : semnalele venite de la butoane pentru a determina incrementarea sau decrementarea numărătorului asociat offsetului pe orizontală
- up, down** : semnalele venite de la butoanele up și down, vor da incrementarea sau decrementarea numărătorului asociat offsetului pe verticală
- enable**: este semnalul venit de la unitatea de control și care va permite operația de deplasare , adică doar când acesta e activ se vor putea modifica valorile numărătoarelor

Iesiri

- offset_x** : va fi valoarea numărătorului intern pentru deplasarea pe orizontală și va da numărul de pixeli cu cât va fi deplasată imaginea pe orizontală
- offset_y**: va fi valoarea numărătorului intern pentru deplasarea pe verticală și va da numărul de pixeli cu cât va fi deplasată imaginea pe verticală

Schema RTL

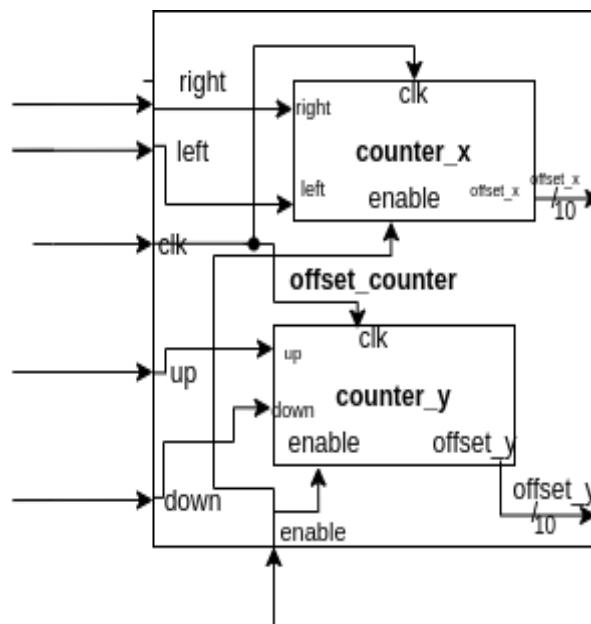


Figura 4.6.1 Schema RTL pentru Offset Counter

Implementarea logicii acestui modul s-a realizat printr-un proces care tratează cazurile în care se apasă un buton sau altul, iar validarea incrementării și decrementării este dată de semnalul de enable. La apăsarea butonului left se incrementează valoarea offsetului pe orizontală deoarece prin deplasare la stânga se va grăbi afişarea pixelului, relativ cu poziţia reală a acestuia data de sync generator.

4.7 Modulul Memory_address

Acest modul se va ocupa cu calculul locației din memoria RAM de unde se va lua pixelul imaginii și va fi transmis semnalelor RGB spre monitor. Funcționarea acestuia constă în adunarea adresei unde ar fi pixelul, data de sync_generator în funcție de *pixel_rate* (deci, cea dată în urma scanării ecranului) cu valoarea de deplasare, iar asupra rezultatului se efectuează modulo pentru a rămâne în intervalul de display. Prin operația de modulo, când se depășește limita superioară, rezultatul va începe de la limita inferioară ;Exemplu pentru scanare pe orizontală $640 \% 640 = 0 \rightarrow$ deci, pixelul va fi afișat în partea stângă a ecranului. De asemenea, în momentul în care control unit oferă posibilitatea de a reflecta imaginea, s-a realizat o logica suplimentară prin care se va inversa poziția pixelilor. De asemenea, din cauza resurselor limitate ale plăcii Nexys4DDR și imposibilitatea creării unei memorii RAM care să stocheze întreaga configurație a unei imagini cu rezoluția 640/480, s-a implementat o logică suplimentară prin care se aplică o scalare a pixelilor imaginii. Astfel, imaginea stocată pe care va avea o rezoluție de 160/120 și se va scala cu un factor de 4 pentru a obține o rezoluție de 640/480.

Intrări:

-enable: semnalul venit de la sync_generator prin care se comunică dacă la un moment de timp scanarea se află în regiunea de display și se pot transmite pixeli spre ecranul monitorului

-addr_x: semnal ce arată poziția reală a pixelului pe orizontală în timpul scanării; acest semnal vine de la sync generator

-addr_y: semnal ce arată poziția reală a pixelului pe verticală în timpul scanării; acest semnal vine de la sync generator

-offset_x: semnal ce vine de la modulul offset counter și reprezintă deplasamentul pe axa x

-offset_y: semnal ce vine de la modulul offset counter și reprezintă deplasamentul pe axa y a pixelului ce va fi afișat pe ecran din memoria RAM

Ieșiri:

-x_mem_addr: poziția pe coloana din memoria RAM de la care se transmite pixelul curent portului VGA, sau în cazul în care semnalul de enable e inactiv , atunci acest semnal va avea valoarea 0

-y_mem_addr: poziția pe linie din memoria RAM de la care se transmite pixelul curent portului VGA, sau în cazul în care semnalul de enable e inactiv , atunci acest semnal va avea valoarea 0

Schema RTL

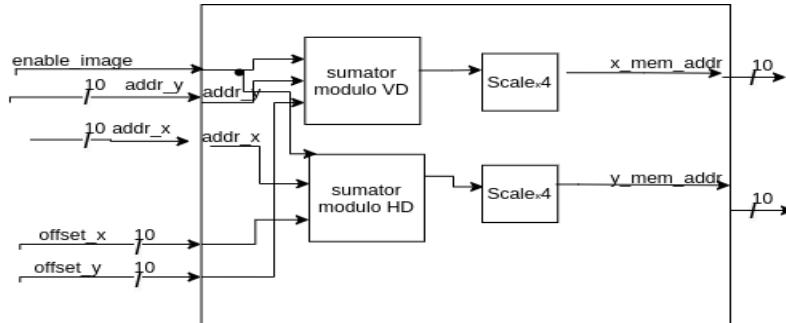


Figura 4.7.1 Schema circuitului memory address



4.8 Modulul SDInitialise

Acest modul este responsabil cu inițializarea cardului micro SD pentru a funcționa în modul SPI. Pentru efectarea acestei operații trebuie transmise comenzi speciale cardului micro SD, pentru a intra în acest mod, deoarece el inițial va intra în SD mode. Protocolul SPI va prezenta un canal de comunicație byte oriented. Fiecare comandă sau bloc de date este construit pe 8-bit bytes și e aliniat cu semnalul **CS** (Chip Select), adică lungimea e multiplu de 8 cicluri de ceas. Cardul incepe să numere semnalele de ceas doar după aserțiunea semnalului CS. Pentru cardurile standard: un bloc de date poate fi oricât de mare, cel puțin de un byte, iar operațiile pe blocuri parțiale read/write sunt permise. În cazul SDHC, lungimea blocului este fixă (512 bytes), iar citirile parțiale nu sunt permise.

Intrări

- clk100Mhz**: semnalul de ceas de 100 MHz primit de la placă, pe care o să îl divizeze la 400 KHz, pentru a putea genera un semnal de ceas SCK acceptat de cardul micro SD
- reset** : semnal de reset ce va reîncepe procesul de inițializare a cardului
- enable**: semnal care va valida executia procesului din fsm- ul de inițializare
- MISO**: acest semnal vine de la magistrala DAT(0) a cardului, primindu-se date de la card (răsunuri și comenzi sau user data)

Ieșiri

- SCK** – va reprezent semnalul de ceas transmis cardului pentru transmisia comenzi și primirea răsunurilor acest semnal va avea o frecvență de 200 KHz
- MOSI** – prin acest semnal se va trimite cardului biții comenzi, acesta fiind legat la magistrala **SD_CMD** a conectorului micro SD
- CS**: Chip Select, semnal care va activa sau dezactiva cardul. Semnalul e activ pe 0 logic.
- tipSD**: acest semnal va da tipul cardului, îm urma procesului de inițializare:
 - 00-nu s-a terminat inițializare,
 - 01-SDHC,
 - 11-standard,
 - 10- eroare
- finish**: acest semnal va marca încheierea procesului de inițializare. Va fi esențial în entitatea mare, unde va prezenta o continuare de trecere într-o nouă stare în **fsm-ului Hostului**.

Schema RTL a modulului

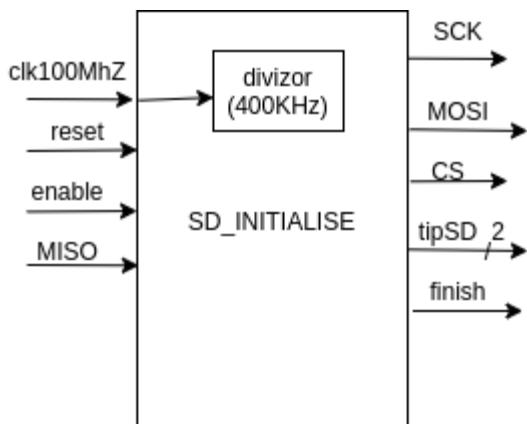


Figure 4.8.1 Schema RTL SDInitialise

Pentru implementarea acestui circuit s-a creat un automat de stare care să descrie șirul de comenzi pe care trebuie să le transmită hostul către cardul micro SD, pentru a intra în modul SPI. De asemenea, se va determina și tipul de card folosit deoarece este nevoie pentru logica de citire, care diferă pentru cele două tipuri de carduri (standard și SDHC).

Forma unei comenzi

Pozitie	47	46	45:40	39:8	7:1	0
Lungime	1	1	6	32	7	1
Valoare	'0'	'1'	xxx..x	xxx..x	xx...x	'1'
Descriere	Bit de start	Bit transmisie	Indexul comenzi	argumentul comenzi	CRC	Bit de end

Forme valide pentru răspunsuri

Răspuns de tip R1

7	6	5	4	3	2	1	0
'0'							-

- 6: eroare din partea parametrului
- 5: eroare din cauza adresei
- 4: eroare la ștergere
- 3: eroare CRC
- 2: comanda ilegală
- 1: eroare la ștergeri
- 0: in idle state - folositoare pentru a verifica dacă s-a terminat procesul de initializare după trimiterea comenzi ACMD41



Răspuns de tip R3

Acest tip de răspuns este format din 5 bytes.

R1		OCR	
39		32	31
'0'			0

Răspuns de tip R7

Acest tip de răspuns este format din 5 bytes.

R1			Versiunea comenzi			biti rezervați			Voltaj acceptat			Echo-back					
39	...	32	31	...	28	27	...	12	11		8	7					0
‘0’			(0001)			...					

Acest tip de răspuns se va întârzi la transmiterea comenzi CMD8, prin care se verifică dacă voltajul dat de host este acceptat de către cardul micro SD. Placa FPGA va da un voltaj de 2.7 – 3.6 V , iar bitii 11..8 din răspunsul R7 la CMD8 va transmite dacă acest volaj e acceptat de card.

Descrierea procesului de initializare în modul SPI și comenziile folosite:

Trebuie setat SPI clock-ul (SCK) la o frecvență între 100 KHz și 400 KHz. În implementare s-a folosit un divizor de ceas care să genereze un semnal de ceas de 400 KHz, iar acest semnal de ceas s-a folosit pentru procesul FSM-ului, iar **SCK** va avea jumate din frecvența acestui semnal divizat. La început se vor trimite 80 de semnale de clock de frecvență 200 KHz cardului, dar cu CS=1, adică va fi deselectat.

Se va trimite apoi CMD0. Pentru a trece în SPI mode , trebuie ca semnalul CS să fie 0 logic în timp ce se trimite de la host CMD0. Răspunsul la aceasta comandă este R1 și în cazul in care acest semnal are forma x"01" atunci se va trece la următorul pas din procesul de inițializare. Formatul aceste comenzi este **b"01_000000" & x"00000000" & x"95"** . De remarcat este faptul că pentru comenzile folosite la inițializare este necesar un CRC valid, pentru celealte comenzi în modul SPI nu e nevoie ca acest CRC să fie valid și se va folosi xFF pentru acesta. Fiecare comandă transmisă pe magistrala va fi protejată de biții CRC. Interfața SPI este by default CRC off mode, adică nu vor conta acești biți asociați comenzilor, cu excepția comenzilor de resetare (CMD0 trebuie sa aibă CRC valid). De asemenea și pentru CMD8 trebuie să se respecte configurația bună a CRC-ului, în caz contrar se va întoace o eroare în răspunsul venit de la cardul micro SD.

Următorul pas constă în verificarea voltajului suportat de cardul micro SD, iar pentru aceasta se va trimite CMD8 cu formatul **b"01 001000" & x"000001aa" & x"87"** (...1aa – voltaj cuprins



intre 2.7 - 3.6V). Răspunsul la această comanda este de tip R7, iar dacă biții 11:8="0001" atunci voltajul e acceptat de micro SD și se poate continua initializarea.

Comanda CMD58 va fi transmisă inițial tot pentru a verifica range-ul Vdd acceptat de card. Aceasta are formatul **b"01_111010" & x"00000000" & x"ff"**. Dacă totul este în regulă se va trece la ciclul de comenzi CMD55-ACMD41.

Comanda ACMD41 (CMD55+ACMD41) este folosită pentru a porni inițializarea și verifică dacă microSD-ul este inițializat complet. Este obligatoriu de a trimite CMD8 înainte de primul ACMD41. Dacă răspunsul la comanda ACMD41, R1 are pentru bitul **in_idle_state** valoarea 1, înseamnă că nu s-a terminat inițializarea cardului. Gazda limite ACMD41 repetat până când acest bit este setat cu 0. După ce inițializarea s-a efectuat, gazda verifică valoarea lui CCS în răspunsul comenzi CMD58. Dacă acest semnal este 1 atunci cardul folosit este SDHC sau SDXC, iar dacă valoarea lui este '0' atunci se va lucra cu un micro SD standard. Formatul comenzi *CMD55 b"01_110111" & x"00000000" & x"ff"*, iar formatul comenzi *ACMD41* este **b"01_101001" & x"40000000" & x"ff"**.

Modul SPI permite citirea uneia sau mai multor blocuri de memorie. Vor fi astfel două tipuri de operații de citire: single block read și multiple block read. Comenzile pentru citire sunt CMD17 și CMD18

După recepția comenzi de citire, cardul va răspunde cu un token de răspuns, urmat de un token de date al cărui lungime este setată prin comanda CMD16 în cadrul unei carduri SD, sau 512 bytes pentru cardurile de tip SDHC sau SDXC. De asemenea, un bloc de date valid va fi urmat de CRC pe 16 biti (2 octeti).

Dacă accesul parțial la memorie este activat pentru cardurile standard (parametrul CSD din READ_BL_PARTIAL este egal cu 1) – lungimea blocului poate fi orice număr din intervalul 1:512 (bytes). Adresa de start poate fi oricare byte address în range-ul valid al cardului. Fiecare bloc ar trebui să fie cuprins într-un singur sector fizic al cardului.

În cadrul cardurilor SDHC pot fi suportate doar blocuri de 512 bytes. Iar adresa de start trebuie să fie aliniată cu limitele blocurilor. Mai multe detalii se vor regăsi în subcapitolul dedicat modului de citire a cardului micro SD, dar în cadrul fsm-ului de inițializare am adăugat și transmiterea comenzi CMD16 pentru a seta o lungime a blocului de citit de 512 bytes și pentru cardurile standard. Formatul comenzi CMD16 este **b"01_010000" & x"00000200" & x"FF"**.

În momentul în care va apărea o eroare se va trece într-o stare *ILLEGAL* unde se va seta valoarea semnalului tipSD cu "10" pentru ca hostul să vadă dacă s-a generat o eroare în procesul de inițializare.

In figura 4.8.2 este redat FSM-ul cu procesul de inițializare. Pentru transmiterea comenziilor și recepția răspunsurilor s-au creat stări ce joacă rolul unor subroutines. În momentul în care se intră în aceste stări, este ținută minte starea din procesul de initializare în care se va trece după terminarea execuției rutinei apelate.

Semnalul de **finish** va semnaliza încheierea procesului de inițializare, iar acesta va fi condiția din automatul de stare a hostului care va determina trecerea la starea de citire a cardului, dacă totul a decurs bine.

Codul pentru SDInitialise se regăsește în ANEXA 8.

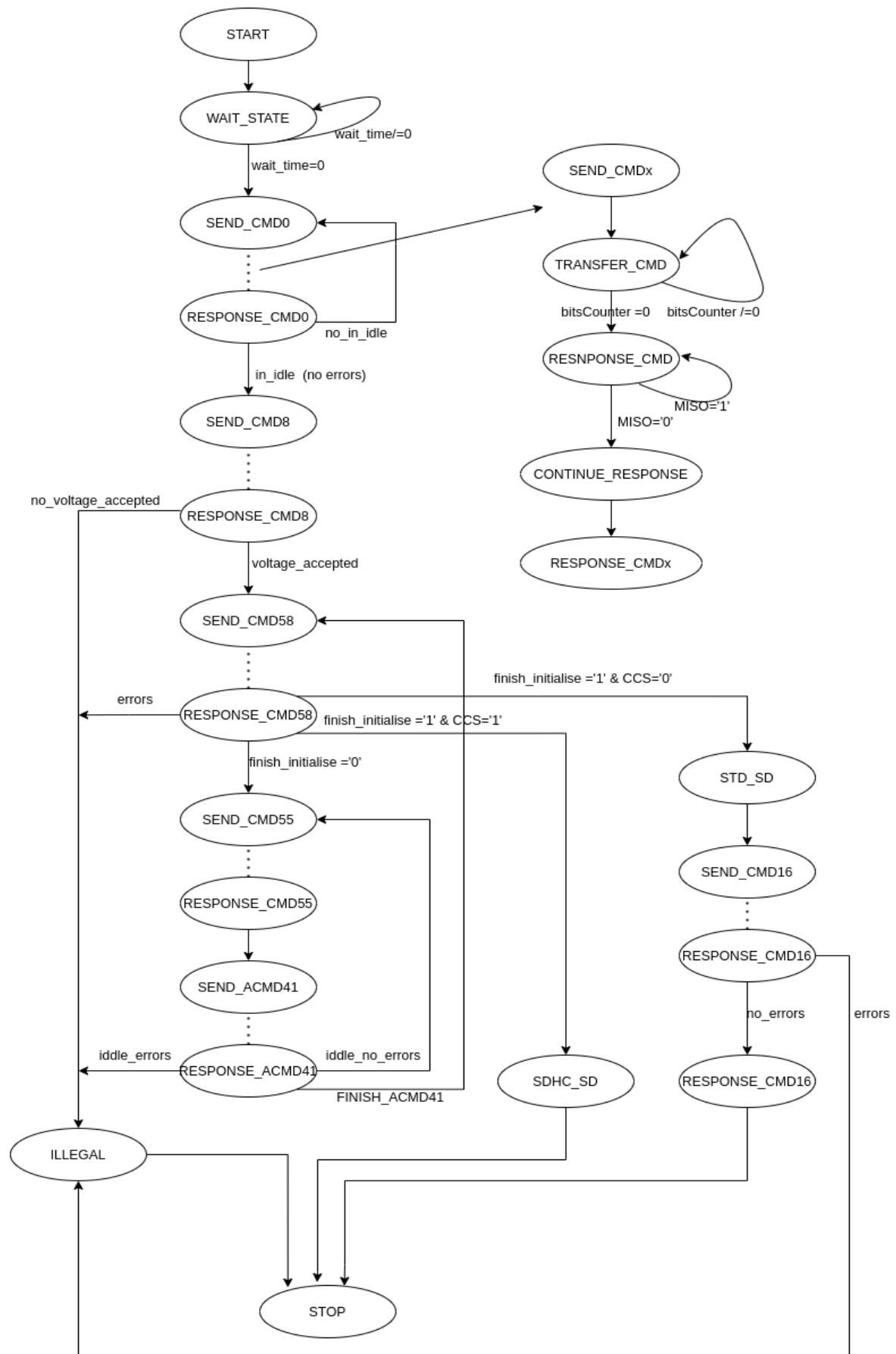


Figura 4.8.2 FSM Initializare card mod SPI



4.9 Modulul SDController

Rolul acestui modul este de a trimite comenzi de citire cardului micro SD. Citirea a fost implementată pentru un card SDHC, astfel blocul de date citit are o lungime de 512 bytes. De altfel, acest modul va transmite informația citită byte cu byte, hostul fiind atenționat în momentul în care s-a citit un byte de memorie. Există o comunicare între această unitate și modulul dedicat host-ului, astfel că în momentul în care este valid un byte de date, controller trimit o notificare spre host și nu va continua citirea până când nu va primi informarea de la host că a reușit să prelucreze datele citite. La rândul lui hostul va trimite o notificare cum că a prelucrat datele, iar atunci controller va știi să continuie procesul de citire până când consumă toți cei 512 bytes de citit, iar apoi va trece în starea de așteptare. Controller-ul va ieși din starea de așteptare când primește de la host semnalul de *read_enable* și de asemnea adresa blocului de citit. Cum s-a menționat în subcapitolul anterior, în cadrul citirii cardurilor SDHC pot fi suportate doar blocuri de 512 bytes. Iar adresa de start trebuie să fie aliniată cu limitele blocurilor.

Intrări

- clk100Mhz**: semnalul de ceas de 100 MHz primit de la placă, pe care o să îl divizeze la 50MHz, pentru a putea genera un semnal de ceas SCK acceptat de cardul micro SD pentru citire (25 MHz)
- reset** : semnal de reset ce va reîncepe procesul citire
- enable**: semnal care va valida execuția procesului din fsm- ul de inițializare
- MISO**: acest semnal vine de la magistrala DAT(0) a cardului, primindu-se date de la card (răspunsuri la comenzi sau user data)
- address**: adresa blocului care se dorește a fi citit
- read_enable**: semnal ce va cere citirea unui bloc de date de către controller

Ieșiri

- data_out**: va reprezenta blocul de date citit de către controller din memoria cardului și pe care îl transmite host-ului; acesta va avea lungimea de 512 bytes
- **SCK** – va reprezenta semnalul de ceas transmis cardului pentru transmisia comenzielor, primirea răspunsurilor și citirea datelor; acest semnal va avea o frecvență de 25 MHz
- MOSI** – prin acest semnal se va trimite cardului biți comenzi, acesta fiind legat la magistrala **SD_CMD** a conectorului micro SD
- **CS**: Chip Select, semnal care va activa sau dezactiva cardul. Semnalul e activ pe 0 logic.
- in_work**: semnal ce va fi activ în momentul în care se fac operații cu cardul. Când fsm-ul controllerului e în starea de așteptare acesta semnal va fi dezactivat
- ready**: semnal ce va fi activ în momentul în care se intră pentru prima dată în starea de așteptare, în cadrul fsm-ului controllerului. În aceste condiții hostul poate să înceapă să ceară citiri de date
- finish_read_blk**: acest semnal va semnala terminarea citirii unui bloc de date
- illegal** : semnal ce va raporta o eroare

Schema RTL

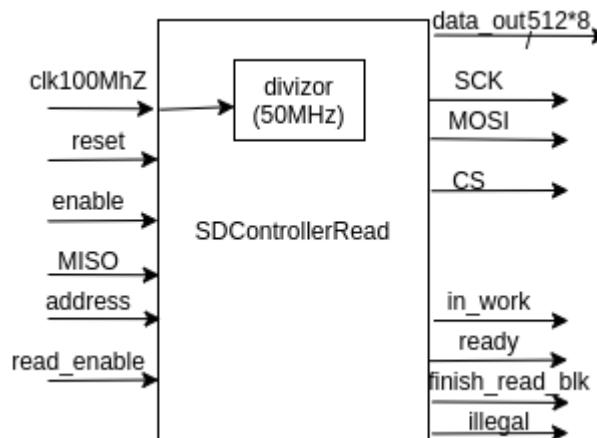


Figure 4.9.1 Schema RTL a modulului SDController

În figura 4.9.3 este prezentat fsm-ul acestui modul. În starea START se vor inițializa semanlele și apoi se va trece în starea de așteptare WAIT_STATE. În WAIT_STATE se va aștepa semnalul read_enable. În momentul activării acestui semnal de către host, se va porni citirea blocului de date de la adresa address. Pentru a porni citirea trebuie trimisă comanda CMD17, al cărei format este **b"01_010001" & address & x"FF"**. Adresa va fi pe 32 de biți. După transmiterea comenzi CMD17 și primirea răspunsului se va citi blocul de date care va fi precedat de un data token de forma xFE , apoi se vor citi 512 bytes, iar la final se va citi CRC-ul care e pe 2 bytes. Cum se poate observa în figura 4.9.2

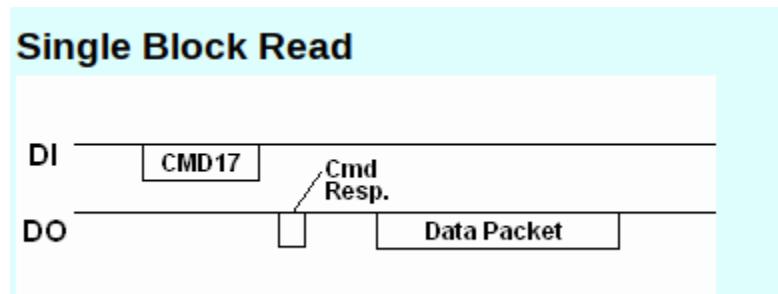


Figure 4.9.2 Citirea unui bloc de date

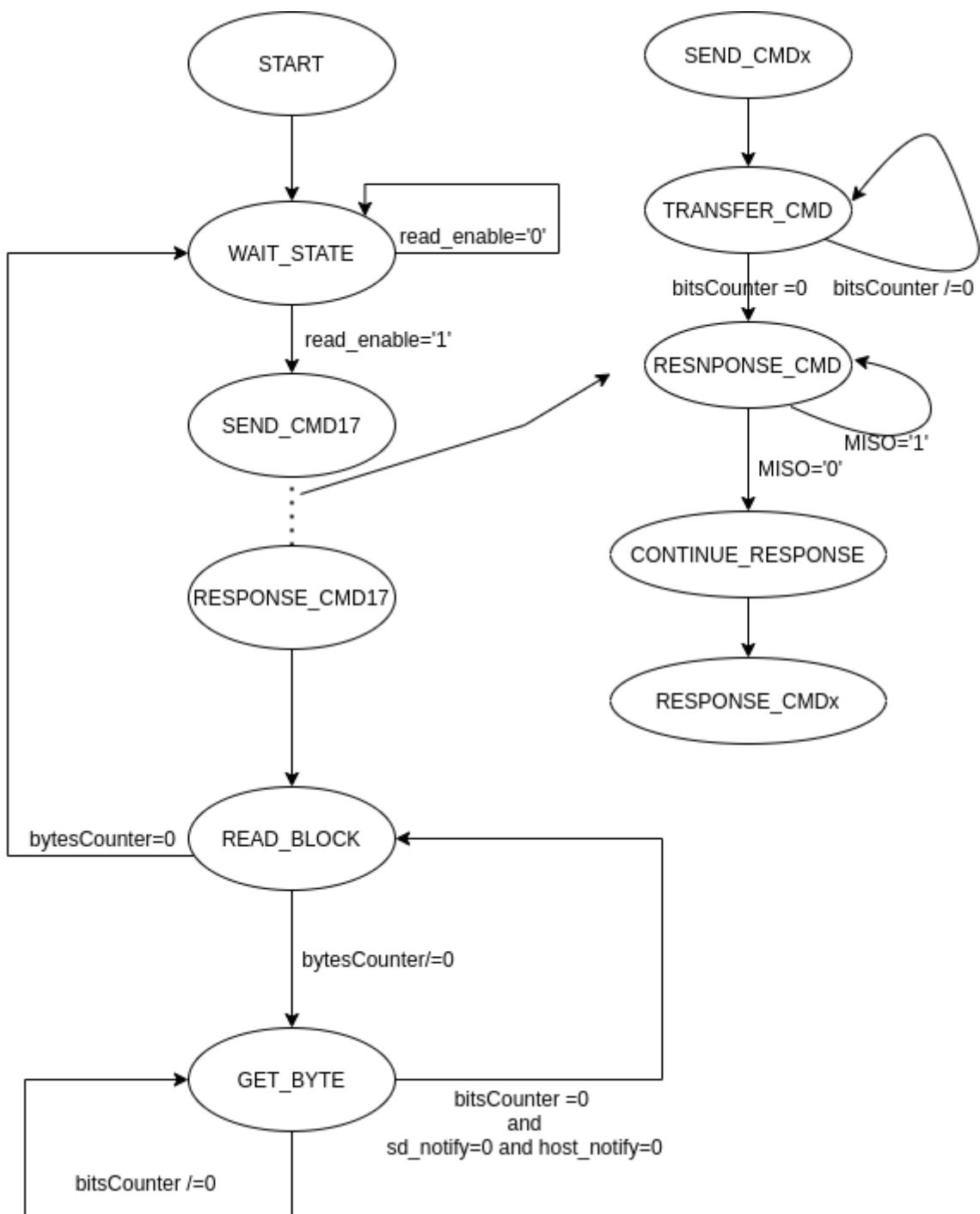


Figure 4.9.3 FSM-ul modulului SDController

4.10 Modulul RAM

Rolul acestui modul este de a stoca temporar imaginea curentă afişată pe ecranul monitorului și care a fost citită de pe cardul microSD. Pentru implementarea acestei memorii s-a declarat un nou tip de semnal, care este de forma unui vector de STD_LOGIC_VECTOR. Dimensiunea memoriei este de $120 \times 160 \times 12 = 230.400$ bits. Lungimea unui cuvânt este de 160×12 și va reprezenta configurația pixelilor de pe o linie. Cu alte cuvinte, imaginea citită are o rezoluție de $160/120$, ceea ce înseamnă că memoria RAM creată va conține 120 de vectori (linii) ce conțin configurația pixelilor a 160 de coloane.

Scrierea în memoria RAM se va face la nivelul a 4 biți dintr-un cuvânt. Acei patru biți vor reprezenta o parte din configurația unui pixel RGB. Ieșirea memoriei RAM va fi pe 12 biți, ceea ce va da configurația întreagă a unui pixel în momentul în care este preluat și afișat pe ecran în procesul de scanare al acestuia.

Intrări

- **clk**: semnalul de ceas de 25 MHz
- **r_addrx**: coordonata x a adresei unde se face citirea
- **r_addrx**: coordonata y a adresei unde se face citirea
- **w_addrx**: coordonata x a adresei unde se face scrierea
- **w_addrx**: coordonata y a adresei unde se face scrierea
- **RW**: semnal ce va activa scrierea în memoria RAM
- **data_in**: semnal pe 4 biti și care reprezintă cantitatea unei culori din configurația RGB a unui pixel

Ieșiri

- **data_out**: semnal cu lungimea de 12 biți ce reprezintă configurația RGB a pixelului curent ce va fi afișat pe ecran în procesul de scanare a acestuia

RTL

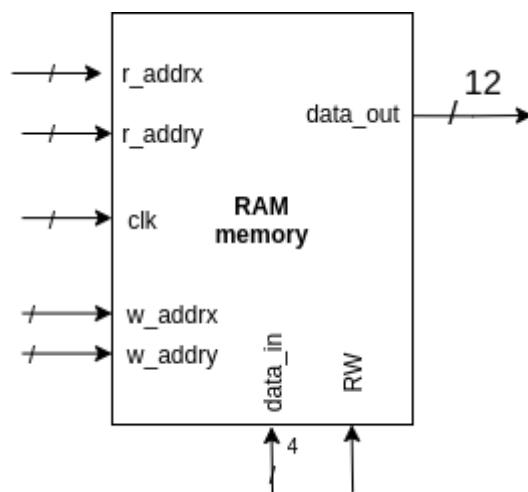


Figure 4.10.1 RTL RAM

4.11 Modulul ImageCounter

Rolul acestui modul este de a hardcoda într-o memorie de tip ROM adresele blocurilor din memoria fizică a cardului unde se află imaginile. De asemenea, în interiorul acestui modul există un numărător care la apăsarea butoanelor left și right (semnale trecute prin circuitul de debounce) va decrementa sau incrementa un contor ce dă numărul imaginii ce va fi afișată pe ecran. Cu acest contor se va adresa memoria ROM și se va transmite mai departe adresa primului bloc a imaginii ce urmează să fie citită și apoi afișată pe monitor.

Intrări

- clk: semnal de ceas (100MHz) pentru procesul ce implementează numărătorul de imagini
- reset: semnal de reset
- enable : change_enable venit de la UC
- btndl : semnal venit de la butonul left (trecut prin mpg) și care va permite întoarcerea la imaginile anterioare
- btngr : semnal venit de la butonul right (trecut prin mpg) și care va permite trecerea la următoarea imagine

Ieșiri

- imgAddr: partea mai puțin seminificativă a adreselor blocurilor de început ale imaginilor. semnal pe 12 biti.

S-a hotărât pentru început stocarea a 9 imagini pe cardul microSD. Acest modul l-am creat cu un generic pentru numărul de imagini, astfel că se pot face ușor modificări ulterioare ale programului.

RTL

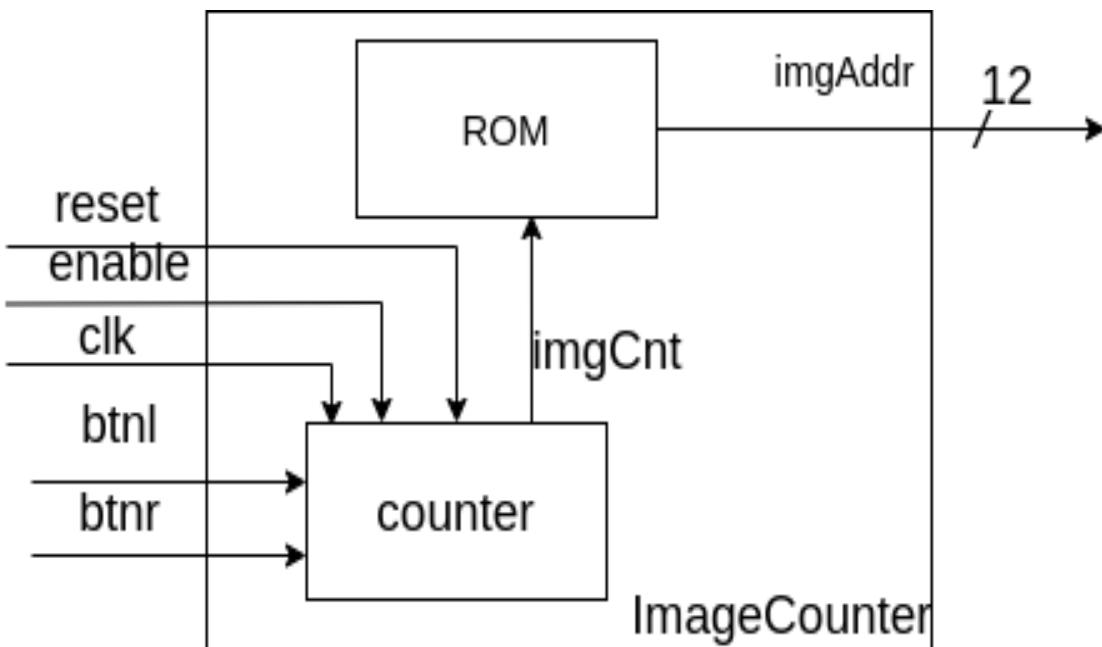


Figure 4.11.1 RTL ImageCounter



4.12 Modulul ModulMicroSD

Rolul acestei componente este de a implementa logica de citire a imaginilor ce urmează a fi afișate pe ecran din memoria cardului microSD. Acesta va instanția componentetele: *SDInitialise*, *SDControllerRead*, *ImageCounter* și *RAM*. S-a creat un fsm care să descrie pașii pe care trebuie să îi facă sistemul pentru citirea imaginilor de pe cardul microSD și aducerea lor temporară în memoria RAM.

În prima stare pe care o descrie fsm-ul acestui modul (START) se va pregăti inițializarea cardului microSD. În starea INIT_CARD se va aștepta semnalul venit de la componenta SDInitialise prin care se comunică faptul că inițializarea cardului microSD s-a terminat. Se va verifica dacă inițializarea s-a realizat cu succes sau nu. În cazul în care inițializarea nu s-a putut realiza cu succes sistemul va trece într-o stare numită ERROR_STATE în care se va bloca și va anunța utilizatorul că nu se mai poate continua procesul. În cazul în care inițializarea a avut succes se va trece într-o stare de așteptare până când componenta *SDControllerRead* comunică prin semnalul *ready* faptul că sistemul e pregătit pentru citirea de pe cardul microSD.

Starea WAIT_READ va reprezenta starea în care se va aștepta un semnal de la utilizator (schimbarea imaginii de afișat- bnl sau bnr) pentru a da comanda de citire a unei imagini din memoria cardului. De asemenea, se va trata cazul inițial în care la pornire se va afișa prima imagine. Odată cu primirea semnalului de citire a unei imagini se trece într-o stare de pregătire a acestei comenzi în care se formează adresa primului bloc a imaginii ce se dorește a fi citită. Informația de adresă se poate determina cu ajutorul componentei *ImageCounter* care oferă cei mai puțini 12 biți ai adresei blocului (*readAddress <= x"00000" & imgAddress*).

Odată cu pregătirea comenzi de citire, se va trece în starea de READ_DATA în care se așteaptă semnalul *finish_read_blk* de la **SDControllerRead** prin care se semnalează terminarea citirii unui bloc de 512 bytes. Apoi se trece în stările WAIT_WRITE și WRITE_IN_MEMORY, stări în care se va scrie blocul citit în memoria RAM. S-au folosit variabile auxiliare prin care s-au contorizat bytes procesați din blocul citit.

O imagine este stocată pe $160*120*3/512 = 113$ blocuri. Se va lua o variabilă contor ce va monitoriza numărul blocurilor citite. Astfel procesul va oscila între stările de citire bloc din memoria cardului și scrierea acestuia în memoria RAM dedicată stocării temporare a imaginii curente până când se vor epuiza cele 113 blocuri. În momentul în care se termină de procesat toate blocurile imaginii se va trece în starea de WAIT_READ.

Intrări

-**clk100MHZ**: semnal de ceas (100MHz) pentru procesul ce implementează numărătorul de imagini

-**clk25MHZ**: semnal de ceas cu frecvență de 25 MHz. Cu această frecvență se executa procesul dedicat fsm-ului acestui modul

-**reset**: semnal de reset

-**enable** : change_enable venit de la UC

-**bnl** : semnal venit de la butonul left (trecut prin mpg) și care va permite întoarcerea la imaginile anterioare și trecerea din starea WAIT_READ

-**btr** : semnal venit de la butonul right (trecut prin mpg) și care va permite trecerea la următoarea imagine și trecerea din starea WAIT_READ

-**SD_RESET**: pentru conectorul microSD. Va fi hardcodat pe valoarea 0.

-**SD_CD**: pentru conectorul microSD

- **SD_SCK**: pentru conectorul microSD și reprezintă semnalul de ceas generat de master pentru transmiterea comenzi și primirea răspunsului.
- **SD_CMD**: pentru conectorul microSD, linia de comenzi
- **SD_DAT**: pentru conectorul microSD, linia de date

Ieșiri

- **tipSD**: tipul cardului microSD. Semnal transmis mai departe de la componenta SDInitialise
- **error**: semnal ce indică o eroare apărută în cadrul sistemului
- **waitRead**: semnal ce specifică faptul că sistemul așteaptă o comandă de citire de la utilizator(în principiu schimbarea imaginii de afișat)
- **finish**: semnal ce indică terminarea citirii unui bloc de date
- **inWork**: semnal ce indică faptul că se face citire de pe cardul microSD, să se fac operații pe acesta
- **addrx**: ajută la calcularea adresei pixelului din memoria RAM ce urmează a fi transmis portului VGA
- **addry**: ajută la calcularea adresei pixelului din memoria RAM ce urmează a fi transmis portului VGA
- **data_out**: semnal pe 12 biți ce vine de la memoria RAM și reprezintă configurația pixelului ce este afișat pe ecran la un moment bin definit

RTL

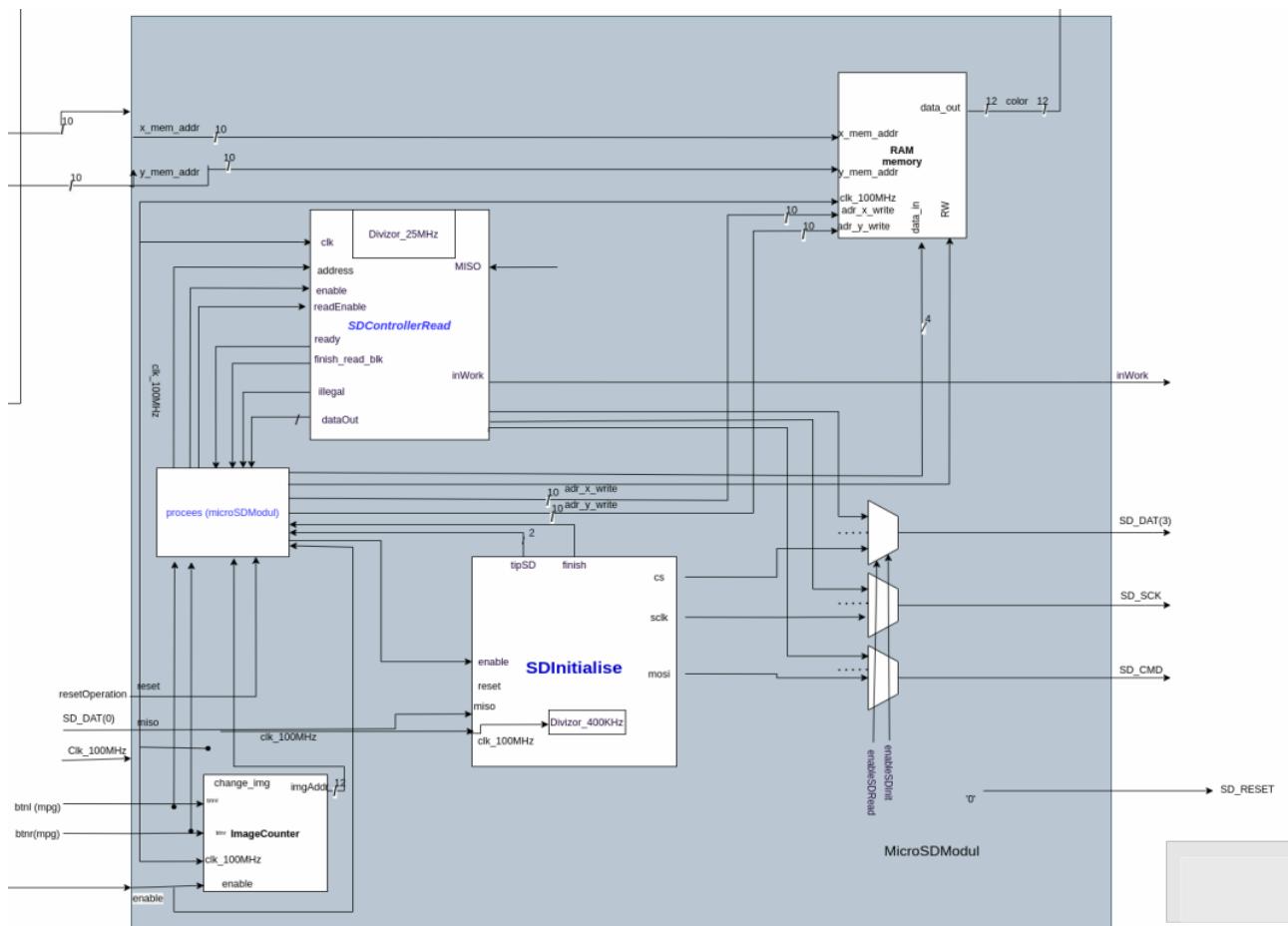


Figure 4.12.1 RTL ModulMicroSD

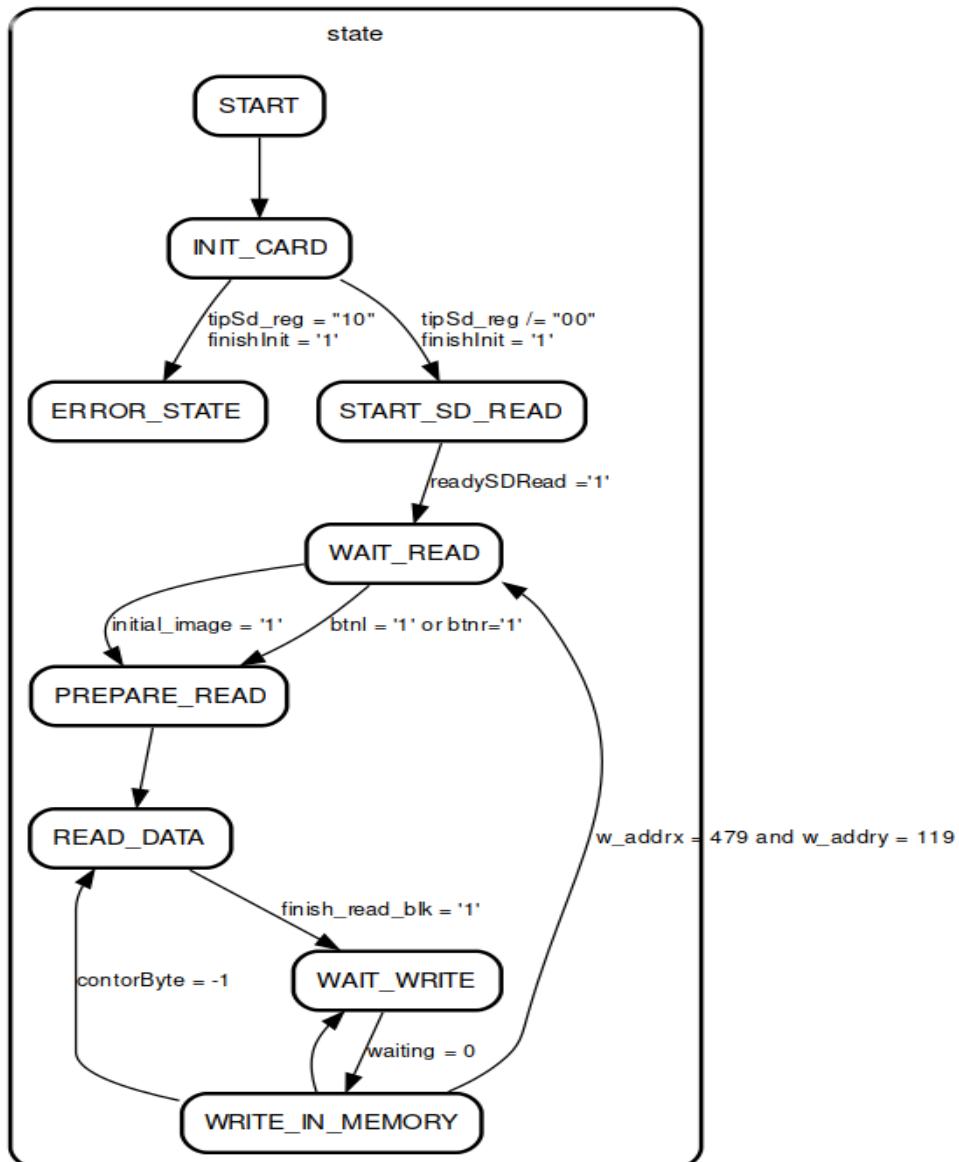


Figure 4.12.2 FSM ModulMicroSD

4.13 Modulul EffectGenerator

Această componentă se va ocupa cu aplicarea unor efecte asupra pixelilor imaginilor. Înainte de a fi transmisă configurația RGB portului VGA, acel semnal de 12 biti de la memoria RAM va trece prin această componentă și va suferi niște modificări. Implementarea acestei componente constă într-un proces combinațional, al căruia semnal de enble vine de la UC. Se va aplica un effect de grayscale dacă primul switch de pe placă este activ, dacă cel de-al doilea switch e activ și primul nu atunci se va aplica un efect de sepia asupra pixelilor sau în caz contrar se pot incrementa valorile pentru fiecare culoare din configurația RGB cu ajutorul switch-urilor (13 -2). De exemplu, $red_final = red_initial + switch(13 \text{ downto } 11)$.

Intrări

- effect_enable**: activarea aplicării efectelor asupra pixelilor imaginii
- sw**: intrare pe 12 biți pentru valorile primite de la switch-urile cu care se incrementează aportul de culoare din RGB
- grayscale**: aplicare filtru grayscale
- sepia**: aplicare effect prestabilit sepia
- rgb**: configurația RGB pe 12 biți a pixelului venită de la memoria RAM

Ieșiri

- rgb_out**: configurația RGB pe 12 biți a pixelului după aplicarea efectelor dacă s-au activat sau în caz contrar chiar valoare rgb inițială

RTL

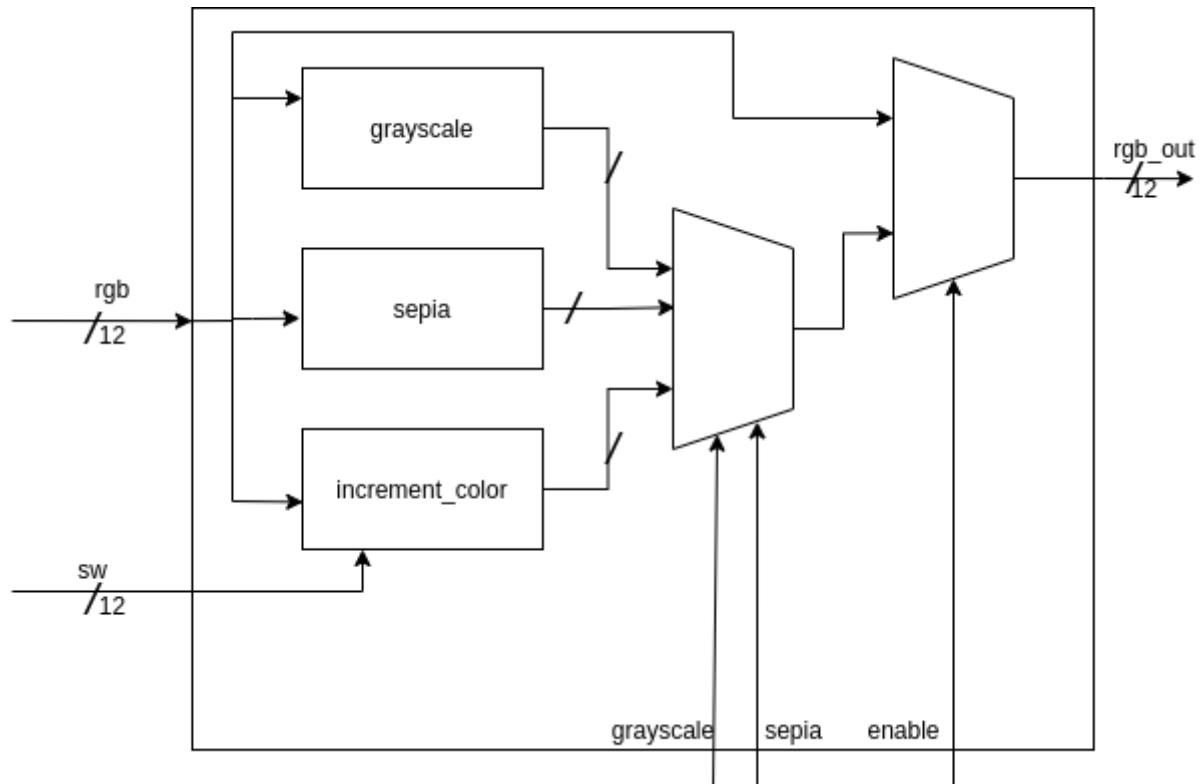


Figure 4.13.1 RTL EffectGenerator

4.14 Modulul ControllerVGA

Rolul acestui circuit este de a introduce un etaj pipeline pentru a sincroniza toate semnalele cu semnalul de ceas de frecvență 25MHZ, înainte de a le transmite portului VGA. Deci, se va introduce o întârziere de perioadă de ceas înainte de transmiterea datelor *hsync*, *vsync*, *blue*, *green* și *red*.

Intrări

- clock_25MHz**: semnalul de clock de 25 MHz ce reprezintă pixel rate-ul rezoluției de 640/480
- hsync_clock**: semnalul de sincronizare pe orizontală
- vsync_clock**: semnalul de sincronizare pe verticală
- enable_image**: semnalul de enbare al imaginii, procesul de scanare a ecranului se află în zona de display
- color**: configurația RGB a pixelului de afișat

Ieșiri

Se vor trimite portul VGA al plăcii.

- red**: semnal pe 4 biți ce descrie aportul de culoare roșie
- green**: semnal pe 4 biți ce descrie aportul de culoare verde
- blue**: semnal pe 4 biți ce descrie aportul de culoare albastră
- hsync**: semnalul de sincronizare pe orizontală
- vsync**: semnalul de sincronizare pe verticală

RTL

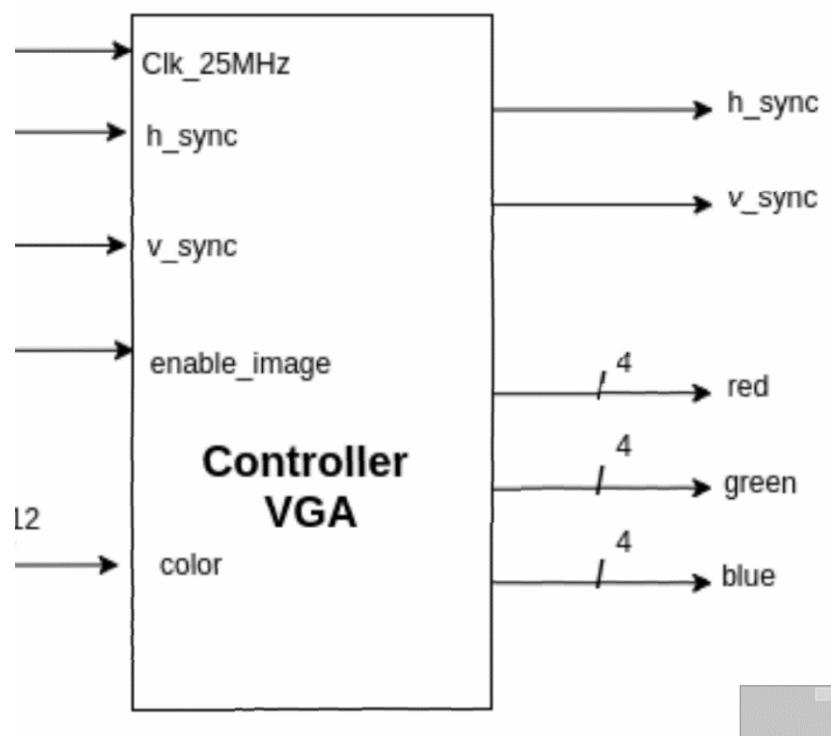


Figure 4.14.1 RTL ControllerVGA



4.15 Alte componente

În componenta UE au fost instanțiate toate componentele ce descriu execuția sistemului VGA.

Componenta BlackBox va instația unitatea de execuție și unitatea de control. Componenta Display7seg este optională și s-a folosit explicit pentru testarea datelor citite de pe cardul microSD.

5. Rezultate experimentale

5.1 Control Unit

Pentru această componentă s-a realizat un modul de test care va trece automatul de stare prin toate stările lui posibile în funcție de intrări și s-au verificat valorile semnalelor de ieșire. Pentru simulare s-a definit un semnal de ceas, iar obținerea impulsurilor acestuia s-a făcut prin crearea unui proces în care se va asigna pentru jumate de perioadă valoarea 0 logic, iar pentru celalătă jumate de perioadă valoarea 1 logic. Perioada acestui semnal de clock este de 10 ns, la fel ca al oscilatorului de cuarț al placii NEXY 4 DDR. Având în vedere că tranziția între stări e posibilă în momentul apăsării unui buton (btnc), s-a definit un semnal pentru acesta. De remarcat este faptul că acest buton trebuie să rămână activ doar o perioadă de ceas, iar tranzițiile lui trebuie să fie sincrone cu cele ale semnalului de ceas, deoarece în cadrul sistemului, semnalele de la butoane vor fi trecute prin un circuit MPG care va asigura un singur impuls al semnalului de la buton și durata în care este activ să fie doar o perioadă de ceas (chiar dacă butonul e mentinut apăsat).

Pentru simulare, în prima fază s-a resetat automatul unității de control prin aplicarea unui '1' logic pe semnalul de reset (switch_4) -> se verifică dacă ieșirile sunt corecte. Apoi se setează switch-ul pe 0 logic și se verifică dacă sistemul a trecut în starea de schimbare a imaginii. Se aplică apoi '1' logic pentru o perioadă de ceas pentru btnc și se urmărește dacă sistemul a trecut în starea move_image. Pe aceeași logică se verifică și celelalte tranziții între stări. Mai jos se regăsește o captură de ecran cu simularea. Se observă că simularea a decurs cu succes, iar tranzițiile sunt corecte, conform figurii 4.2.2.



Figura 5.1.1 Simulare unitate de control

5.2 Divizor de frecvență

Pentru simularea acestui circuit s-a luat un semnal de ceas de perioadă 10 ns pentru clk_100MHz. S-a folosit un proces pentru determinarea impulsurilor de ceas pentru clk_100MHz.

Pe simulare se observă că perioada semnalului de ieșire (clock_25MHz) este de 40 ns => o frecvență de $1/(40 * 10^{-9}) = 10^8/4 \text{ Hz} = 25 * 10^6 \text{ Hz} = 25 \text{ MHz}$.

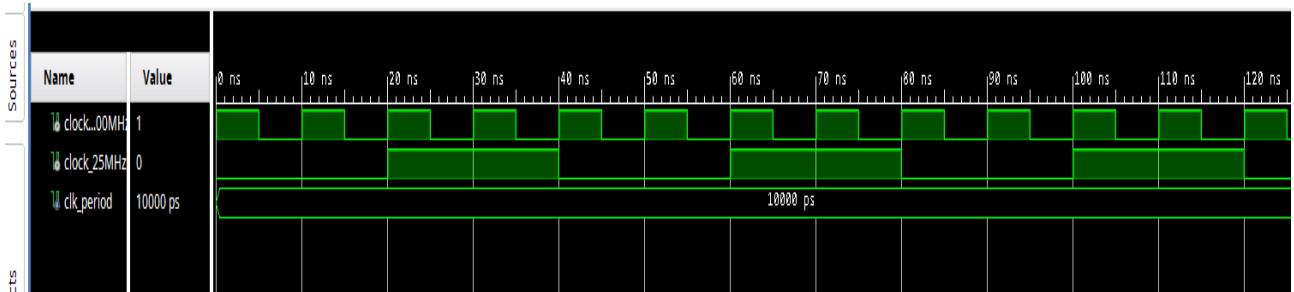


Figura 5.2.1 Simulare divizor de ceas de 25 MHz

5.3 Sync generator

Pentru testarea acestui modul s-a creat fișierul *sync_generator_tb* în care s-a instanțiat aceasta componentă și s-a creat semnalul de clock, singura intrare posibilă. Crearea semnalului de clock s-a făcut concurrent cu ajutorul expresiei *after*, iar perioada acestuia s-a setat ca fiind de 10 ns, pentru a se putea urmări mai ușor evolutia sistemului. S-a rulat simularea pentru 0.1 milisecunde și s-a urmărit evolutia numărătoarelor și a stărilor în care se află cele două automate. O primă verificare a constat în urmărire stărilor automatului pe orizontală. În figura 5.3.1 se observă tranziția din starea *S_HD* în starea *S_HF* în momentul în care numărătorul ajunge la valoarea $(280)_{16} = 640$, iar când numărătorul ajunge la valoare $(290)_{16}=656$ se trece în starea *S_HR*. Se observă că *enable_image* e activ când automatele sunt în starea de display, iar când se trece în starea *S_HF*, se dezactivează acest semnal. O remarcă importantă se referă la evoluția semnalului **hsync**, care se polarizează când se trece în starea *S_HR*. (polarizarea pentru rezoluția 640/480 este *H_pol* = '0').

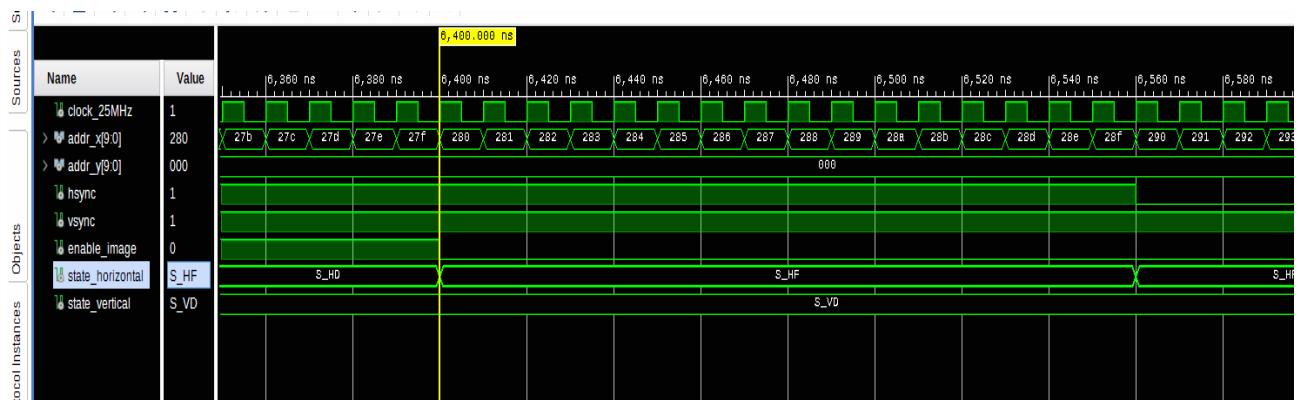


Figura 5.3.1 Simulare sync generator capture 1

Se continuă cu evoluția automatului de stare pe orizontală. În figura 5.3.2 este surprinsă trecerea sistemului din starea de *horizontal retrace* în *starea de back porch* cand numărătorul ajunge la valoare $(2f0)_{16}$. Se observă că se anulează polarizarea semnalului *hsync*. În figura 5.3.3 Se observă cazul în care se termină de procesat un rând, și se resetează numărătorul pe orizontală. De asemenea, se va incrementa numărătorul pe verticală cu o unitate la acest pas.

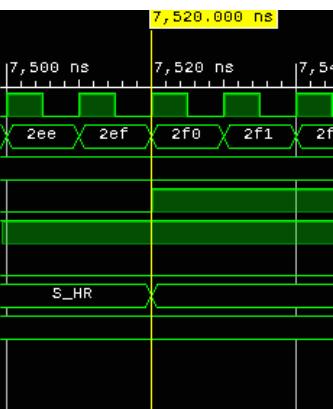
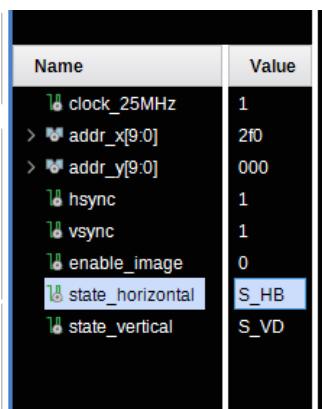
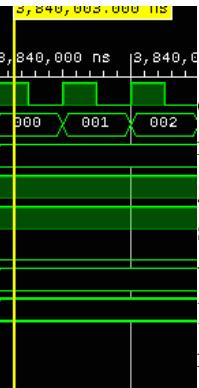
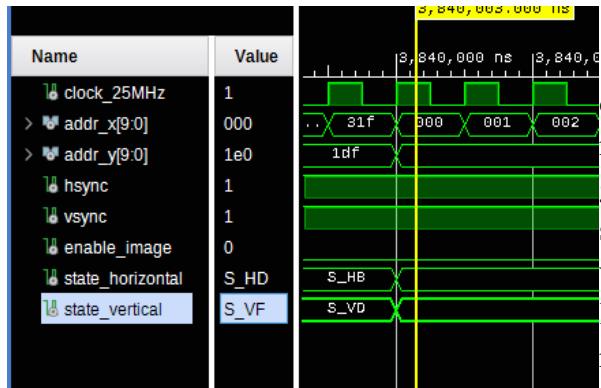


Figura 5.3.2 Simualare sync generator capture 2



Figura 5.3.3 Simualare sync generator capture 3

În continuare sunt prezentate câteva cazuri din evoluția automatului asociat scanării pe verticală.



În această captură observăm ca trecerea din starea S_VD în starea S_VF se realizează în momentul în care s-a procesat ultima linie din gridul de pixeli ($1df$)₁₆=479. În acest moment, scanarea trebuie să revină din colțul stânga sus, dar vor exista niște tempi morți pana când se va reveni la aceasta poziție. Acești tempi vor fi dați de regiunile din automatul de stare pe verticală. O remarcă importantă, este că incrementarea numărătorului pe verticală se face doar când se atinge limita superioară a celui pe

horizontal. În acest moment, se va efectua o trecere din starea S_VF în starea S_VD și se va reinicia numărătorul pe verticală. Acestea vor fi urmărite de tempi morți pana când se va reveni la aceasta poziție. Acești tempi vor fi dați de regiunile din automatul de stare pe verticală. O remarcă importantă, este că incrementarea numărătorului pe verticală se face doar când se atinge limita superioară a celui pe

Figura 5.3.4 Simualare sync generator capture 4 orizontală.

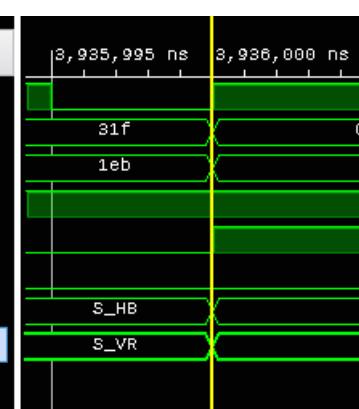
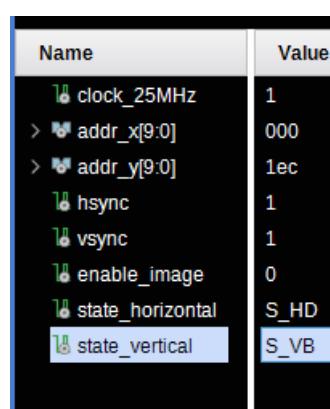
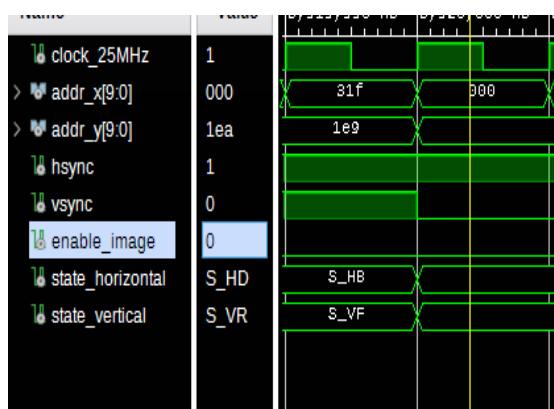


Figure 5.3.5 Simualare sync generator capture 5

Figure 5.3.6 Simualare sync generator capture 6

În figura 5.3.5 se ilustrează trecerea din starea S_VF în starea S_VR care este urmată de polarizarea semnalului v_sync cu valoarea specifică rezoluției de lucru alese. În figura 5.3.6 se

observă că la trecerea din starea S_VR în starea S_VB se depolarizează semnalul **vsync**. De asemenea, se observă că semnalul de **enable_image** este dezactivat pentru că nu suntem în zona de display.

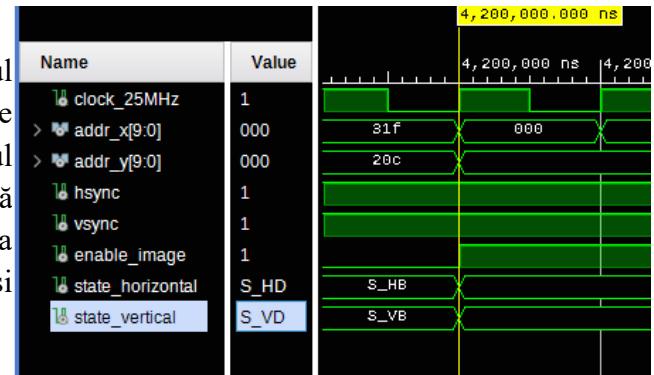


Figura 5.3.7 Simualare sync generator capture 7

5.4 Modul MPG

Pentru testarea modului MPG s-a creat fișierul *mpg_tb*, în care s-a instantiat circuitul **mpg** și a fost descris un scenariu de funcționare într-un proces. S-a simulațat cazul în care butonul e apăsat și menținut apăsat, observându-se în figura 5.3.1 cum doar un singur impuls egal cu o perioadă de ceas va fi generat pentru semnalul de *enable* și acesta numai după un interval de timp (ffff).

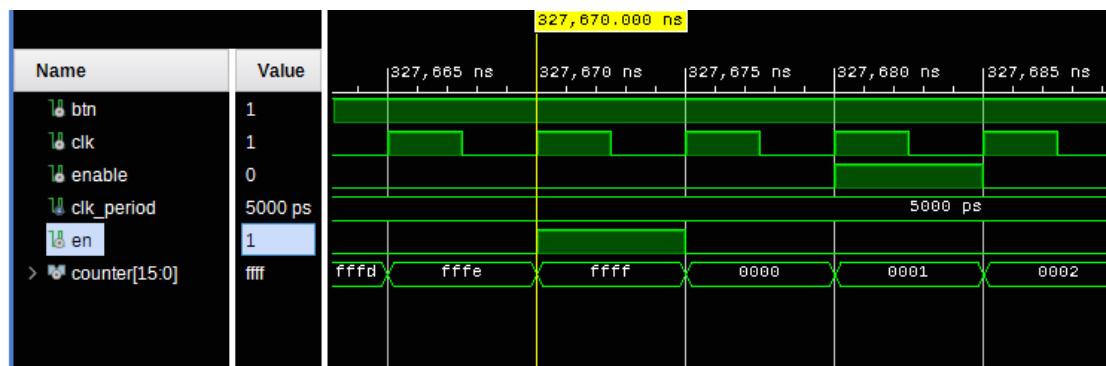


Figura 5.3.1 Simulare MPG

Chiar dacă butonul e apăsat, nu se va genera niciun semnal de enable. În momentul în care numărătorul intern ajunge la starea x”ffff”, observăm că semnalul *en* va fi activ și va permite citirea valorii semnalului de la buton. Semnalul de *enable* va fi activat doar pentru o singură perioadă de ceas, iar apoi dezactivat.

5.5 Offset counter

Pentru testarea acestui modul s-a folosit fișierul *offset_counter_tb.vhd* în care s-a instantiat acest modul și s-au generat câteva intrări posibile și s-au verificat valorile de pe iesiri. Semnalul de ceas a fost generat prin intermediul unui proces, iar perioada acestuia este de 10 ns. În primă fază s-a testat dacă la început offsetul este setat la 0 pe ambele direcții, iar apoi s-a încercat simularea deplasării pe orizontală și pe verticală prin setarea de-a lungul unei perioade de ceas a unui semnal dintre *up*, *down*, *left* și *right*. S-au pus *if*-uri pentru a testa dacă valorile obținute sunt la fel cu cele așteptate. De asemenea, e tratat și unul din cazurile limită în care se ajunge la limita inferioară și se pornește de la cea superioară dacă se încearcă decrementarea.

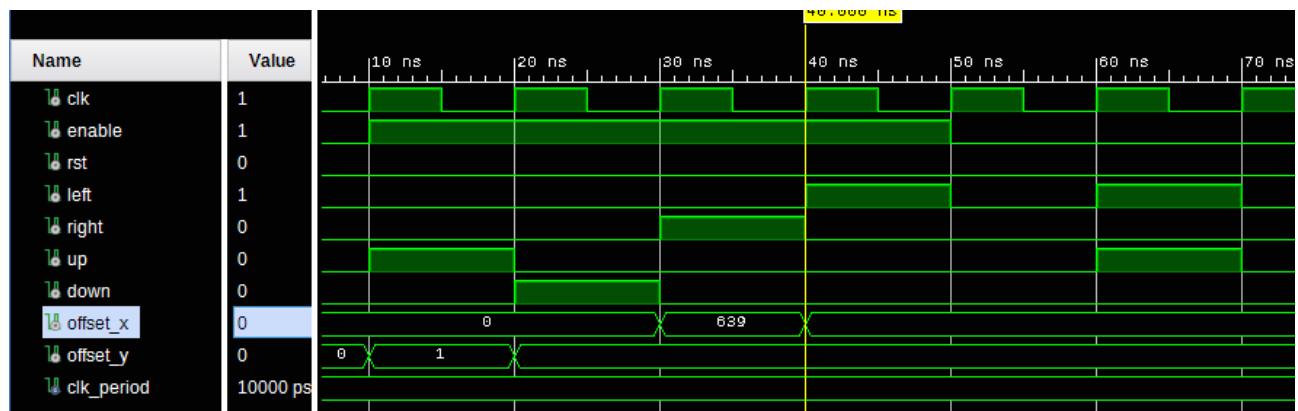


Figura 5.5.1 Simulare offset counter

În figura 5.5.1 este surprinsă o parte din simularea circuitului, în care se observă comportamentul numărătoarelor de offset la aplicarea semnalelor venite de la butoane. De asemenea, este surprins și rolul semnalului de *enable*, care dacă este dezactivat va opri orice operație pe cele două numărătoare.

5.6 Memory_address

Pentru testarea acestui modul s-a creat fișierul de test *memory_address_tb.vhd* prin care s-au aplicat diferite semnale de offset și s-a observat logica de computație a adresei finale de memorie, de la care se va transmite semnalele rgb la portul vga. De asemenea, din simulare se observă că în momentul în care se dezactivează semnalul de enable, adresa va fi 0.

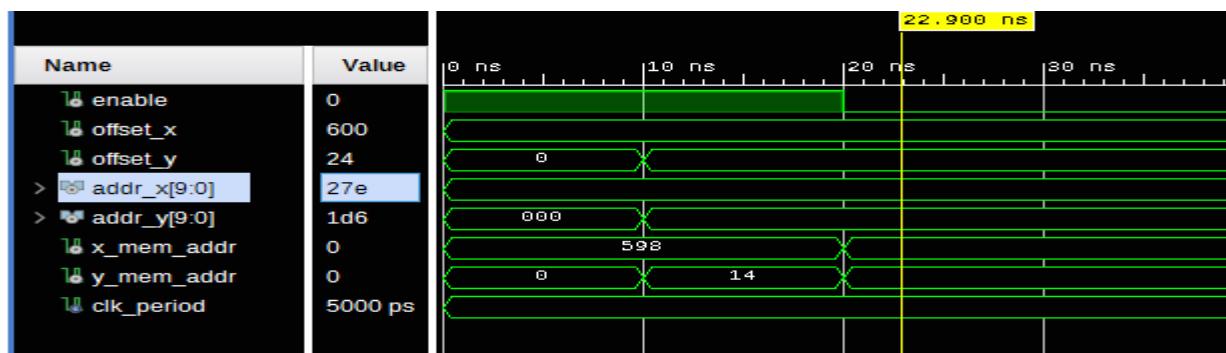


Figura 6.6.1 Simulare memory address

5.7 SDInitialise

Pentru testarea acestui modul s-a realizat un modul principal și s-a generat bitstream-ul. Semnalul tipSD s-a legat la primele 2 led-uri și s-au introdus 2 carduri diferite: standard și SDHC, iar ledurile s-au aprins conform așteptărilor. S-a încercat crearea unui modul ILA pentru testarea sistemului VGA, unde au fost adăugate semnale interne ale acestui modul, dar nu s-a putut realiza implementarea din cauza numărului prea mare a LUT-urilor.

În simulatorul pus la dispoziție de mediul de dezvoltare Vivado s-a încercat simularea manuală, configurându-se semnalele necesare procesului de simularea ale acestui modul.

Pentru inceput s-a analizat faptul că se intră în starea de așteptare în care se trimit 80 de impulsuri de ceas SCK către cardul microSD.

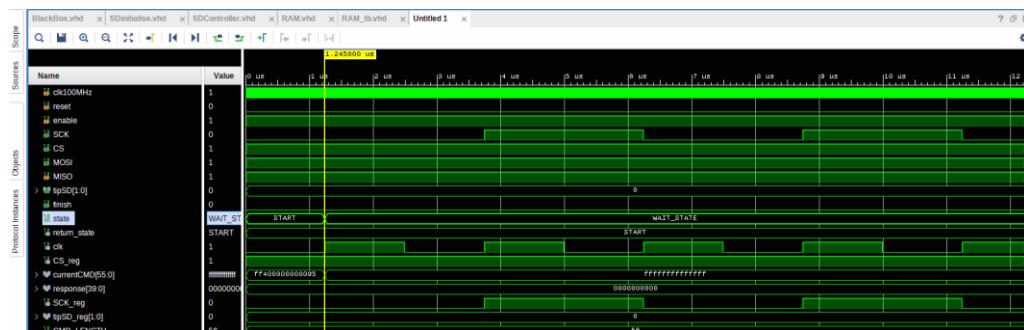


Figura 5.7.1 Starea de trimitere a celor 80 de impulsuri

După trimiterea celor 80 de impulsuri se va trimite comanda CMD0 spre cardul microSD. Trimitera comenzii se realizează prin apelul acelei subroutine de care aminteam special construită pentru trimitera unei comenzi cardului microSD, iar apoi recepționarea răspunsului venit de la acesta.

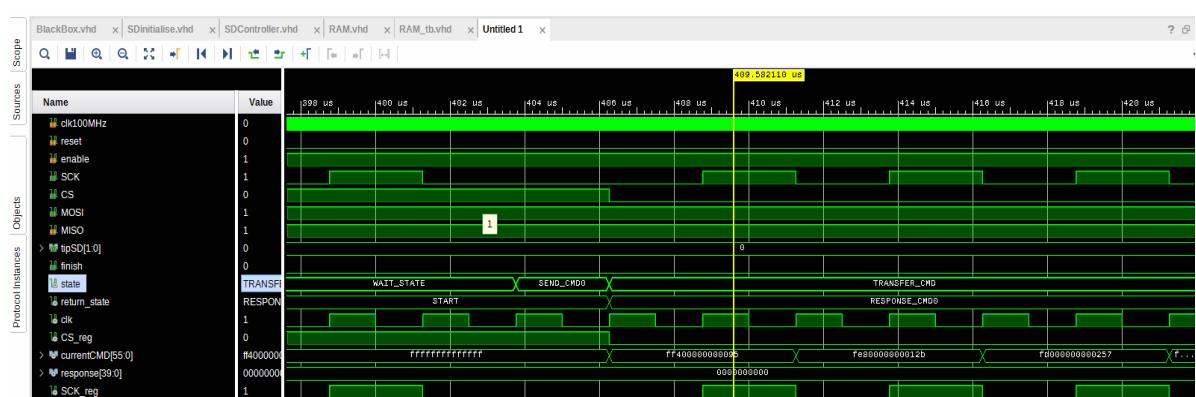


Figura 5.7.2 Transmitere CMD0

În figura 5.7.2 se observă trecerea în starea SEND_CMD0, stare care va comanda încărcarea bițiilor specifici comenzi CMD0 în registrul semnal currentCMD. Apoi se va trece în starea de transmitere a comenzi.

În figura 5.7.3 se observă că după transmiterea comenzi se va trece în starea de așteptare a răspunsului venit de la card la primirea comenzi. Se va trece în starea GET_RESPONSE, stare în



care se va aștepta primul bt al răspunsului are ar trebui să fie 0. Dacă pe linia MISO se primesc doar biți de 1 se va aștepta până când se va primi un 0 logic.

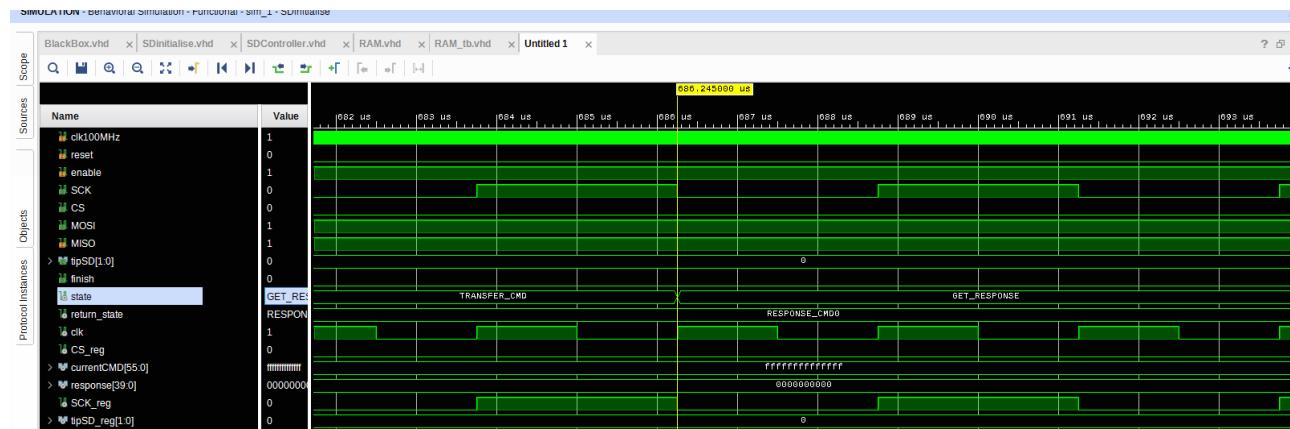


Figura 5.7.3 Așteptare primire răspuns pentru CMD0

După primirea primului bit de 0 din răspuns, se va trece în starea CONTINUE_RESPONSE în care se vor citi ceilalți biți ai răspunsului. În figura 5.7.4 se observă faptul că dacă răspunsul la CMD0 este format numai din biți de 0, atunci se va reîncerca trimitera lui CMD0 până cand se va primi răspunsul că microSD-ul se află în starea de idle.

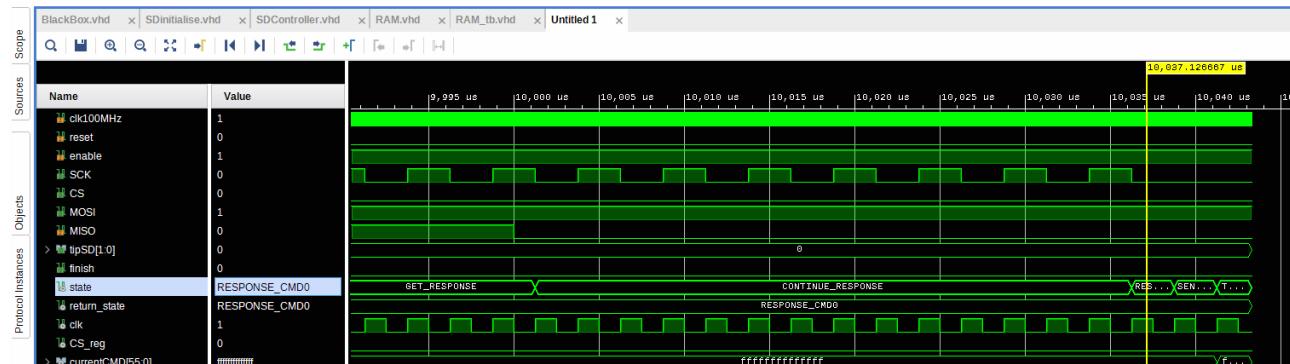


Figura 5.7.3 Așteptare primire răspuns pentru CMD0

Logica pentru transmiterea celorlalte comenzi este similară.

5.8 SDControllerRead

Nu am reușit să fac un test bench. Am făcut un modul în care să citesc anumite date de la diferite adrese din memoria sd cardului și afișarea lor pe mod pe afișoare. Monitorizarea datelor citite s-a făcut cu ajutorul programului HxD disponibile pe sistemul de operare Windows și care permite vizualizarea memoriei fizice a cardului.

În simulatorul pus la dispoziție de mediul de dezvoltare Vivado s-a încercat simularea manuală, configurându-se semnalele necesare procesului de simularea ale acestui modul.

În figura 5.8.1 se observă trecerea în starea SEND_CMD17, în momentul în care atât semnalele de enable și read_enable sunt active. De asemnea, starea aceasta va duce la încărcarea comenții CMD17 în semnalul/ registrul currentCMD17. De asemnea, se observă formarea comenții cu ajutorul adresei primite ca intrare. Acea adresă primită ca intrare va reprezenta de fapt adresa blocului de date ce va fi citit de pe cardul SDHC.



Figura 5.8.1 Trimiterea comenzii CMD17 și formarea acesteia

În figura 5.8.2 se observă recepționarea răspunsului de către master, răsăuns venit de la card.

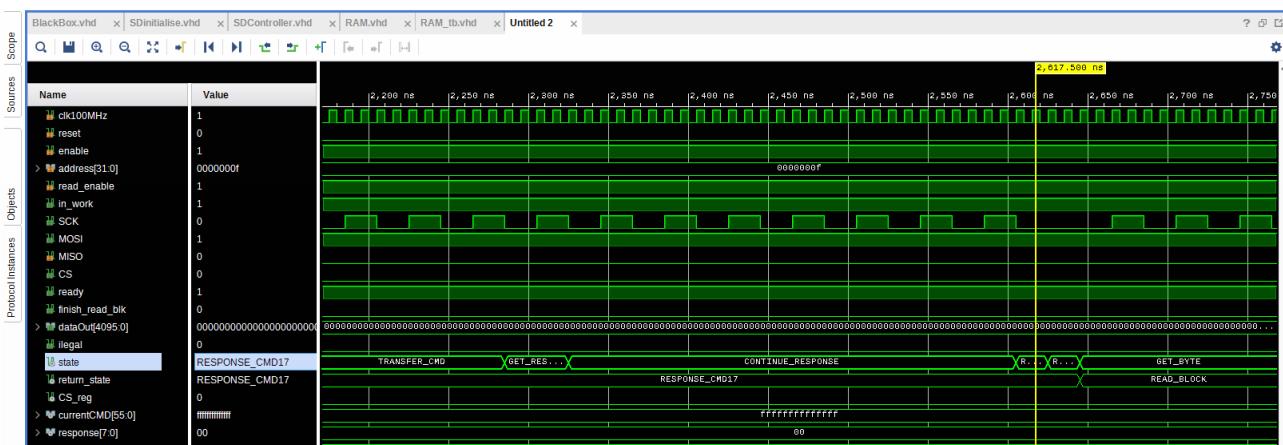


Figura 5.8.2 Primirea răspunsului de cardul microSD

5.9 Modul RAM

Pentru testarea memoriei RAM s-a creat un fișier test bench care va trata câteva cazuri din modul de funcționare a memoriei. Pentru început se dorește a se scrie în memorie la adresa data de coordonatele $w_addr=0$ și $w_addrx=1$ valoarea $data_in = x"A"$. Aceasta înseamnă că se va scrie valoarea pentru culoarea green a primului pixel al imaginii stocate în memoria RAM. De asemenea se observă faptul că scrierea se realizează doar când semnalul RW este activ, are valoarea 1 logic. Pe $data_out$ se va regăsi configurația RGB a primului pixel al imaginii, deoarece avem $r_addr = 0$ și $r_addrx = 0$. Observăm că după operația de scriere apare modificarea corespunzătoare a culorii green pentru a acestui pixel. În figura 5.9.1 se poate observa cele discutate anterior.

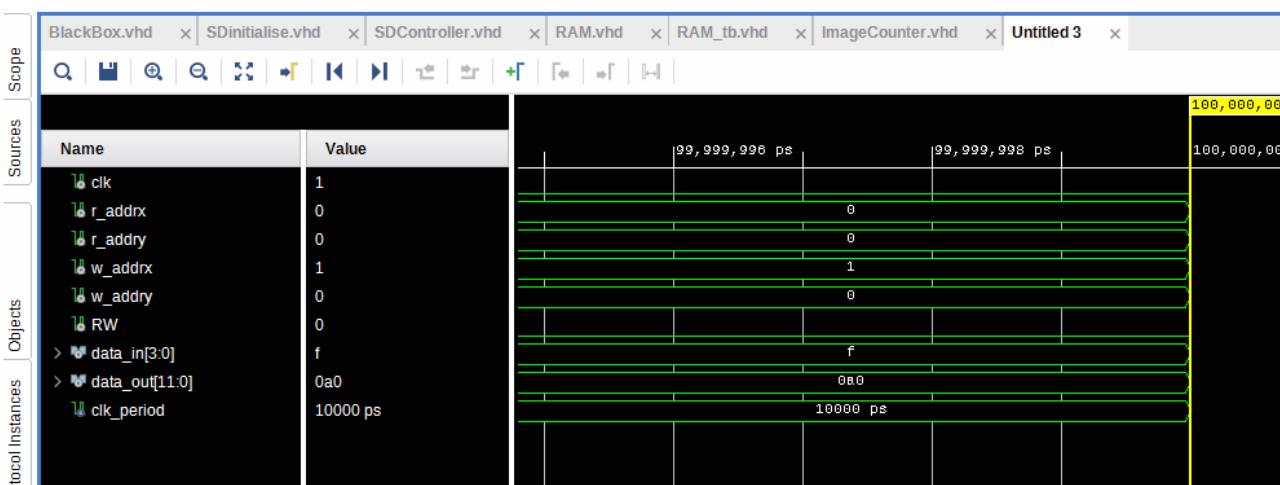


Figura 5.9.1 Simulare memorie RAM

5.10 Modul ImageCounter

Pentru testarea acestui modul s-a creat un fișier test bench prin care se generează un semnal de clock cu perioada de 10 ns și s-au generat semnalele ce simulează activitatea numărătorului, btl1 și bttr pentru a se observa adresarea memoriei ROM cu adresele blocurilor hardcodate. În figura 5.10.1 se poate observa evoluția sistemului testat.

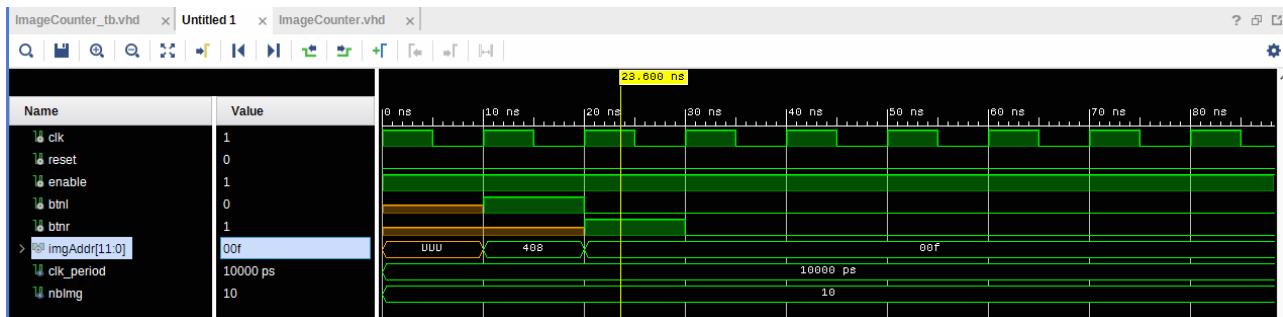


Figura 5.10.1 Simulare ImageCounter

La apăsarea butonului left se va decrementa valoare numărătorului de imagini și se va adresa valoarea din memoria ROM corespunzătoare ultimei imagini de afișat. La apăsarea butonului right se observă că se ajunge iar la adresa primei imagini de afișat.

5.11 Modulul ModulMicroSD

Pentru testarea acestui modul în simulatorul pus la dispoziție de mediul de dezvoltare Vivado s-a încercat simularea manuală, configurându-se semnalele necesare procesului de simularea ale acestui modul.

În figura 5.11.1 se observă trecerea din starea START în starea INIT_CARD. Din starea INIT_CARD se ieșe în momentul când se primește semnalul *finishInit*, și se va verifica tipul cardului dat de semnalul *tip_sd_reg*. Dacă tipul cardului este SDHC atunci se trece în starea



START_SD_READ, stare în care se va activa componenta SDControllerRead pentru citirea datelor de pe card.

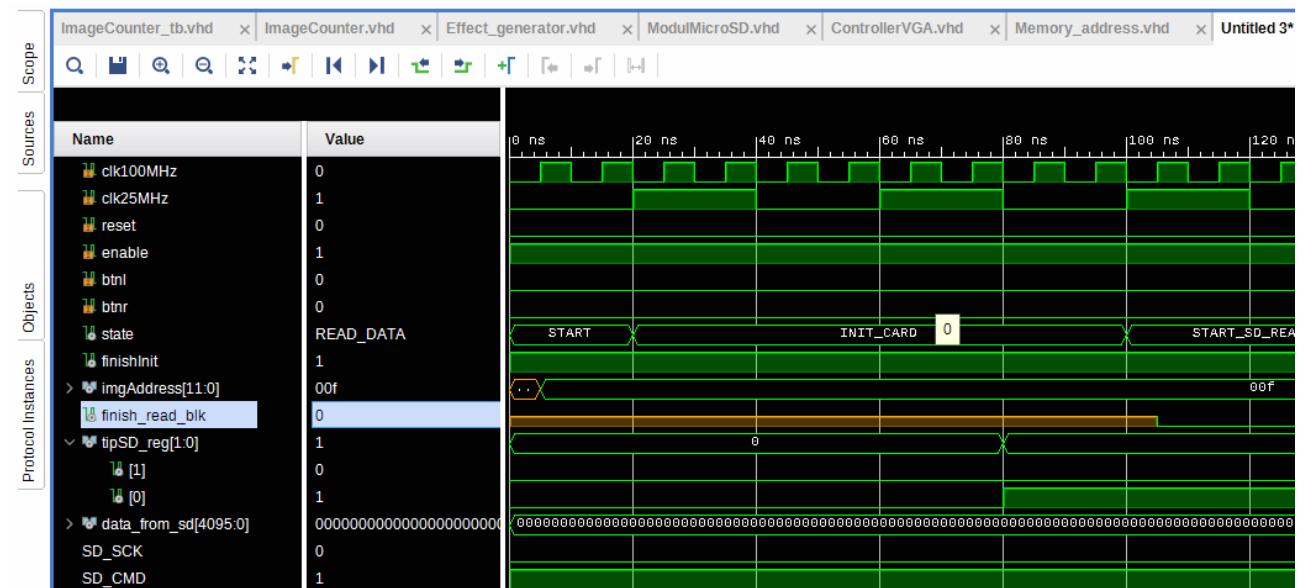


Figura 5.11.1 Simulare ModulMicroSD

Având în vedere faptul că la pornirea sistemului trebuie să se afișeze prima imagine din memoria cardului, autoomatul de stare al modulului va trece în starea de citire a blocurilor imaginii. Se observă în figura 5.11.2 tranziția între stări.

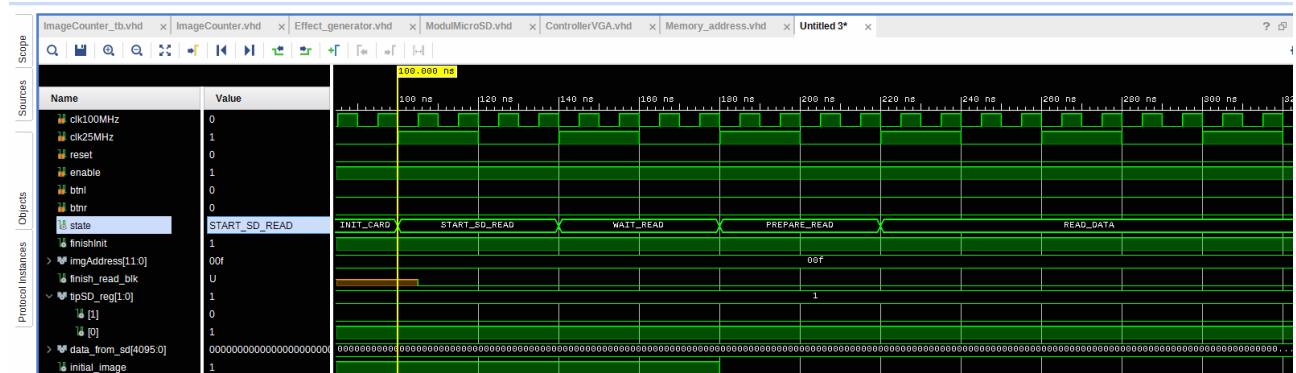


Figura 5.11.2 Simulare ModulMicroSD citire prima imagine

În momentul când se primește informația că s-a citit un bloc al imaginii se pornește procesul de scriere în memoria RAM. Starea WAIT_WRITE este o stare intermediară în care se așteaptă câteva perioade de ceas până când se vor stabiliza valorile în memoria RAM.

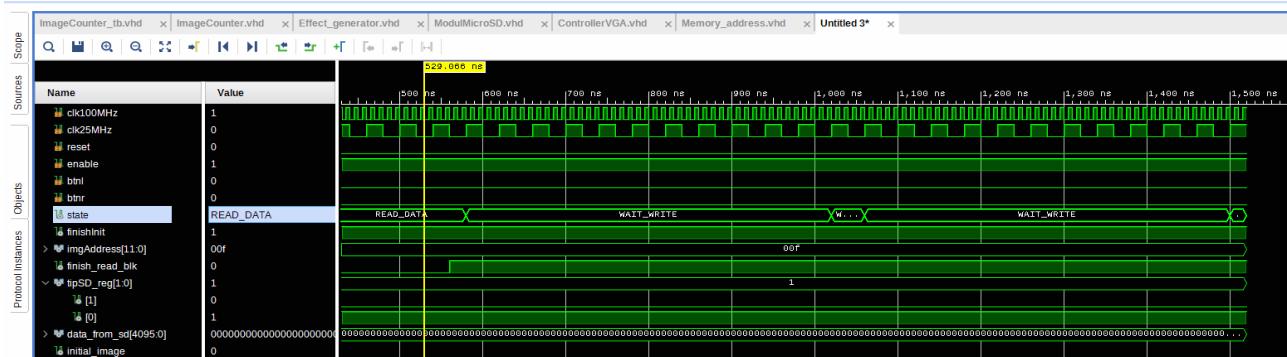


Figura 5.11.3 Simulare ModulMicroSD scriere în memoria RAM

5.12 Modulul EffectGenerator

Simularea acestui modul s-a realizat cu ajutorul unui fișier test bench. În memoria RAM configurația pixelilor e salvată sub formă *bgr*, iar ieșirea **rgb_out** va fi inversă, adică *rgb*. S-au aplicat pe rând efectele disponibile.

În figura următoare se observă aplicarea filtrului sepia, apoi grayscale, iar pe urmă incrementarea aportului de culoare cu ajutorul switch-urilor. Calculele efectuate sunt cele descrise în fundamentarea teoretică.

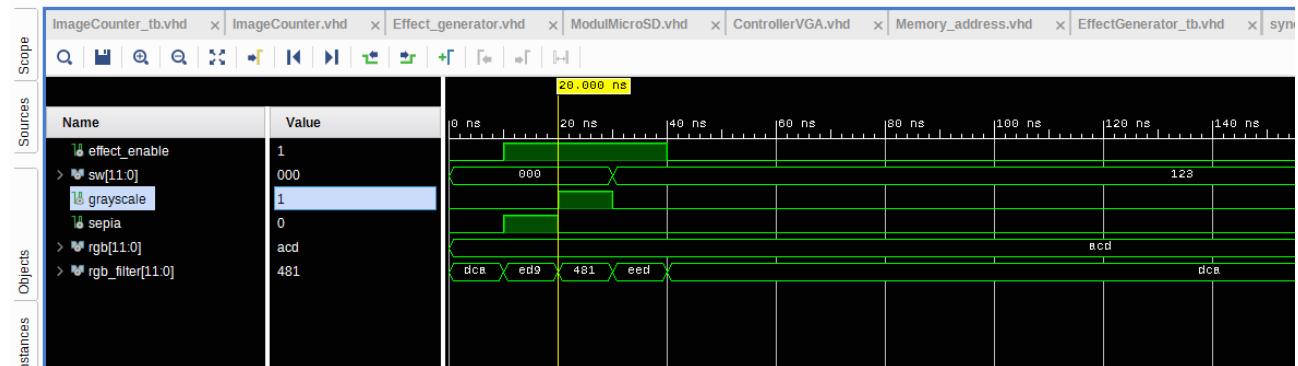


Figura 5.12.1 Simulare EffectGenerator

5.13 Modulul ControllerVGA

Pentru acest modul s-a creat un test bench simplu prin care să se observe întârzierea de un clock a semnalelor duse spre portul VGA cu scopul eliminării problemelor de timing ce pot apărea.

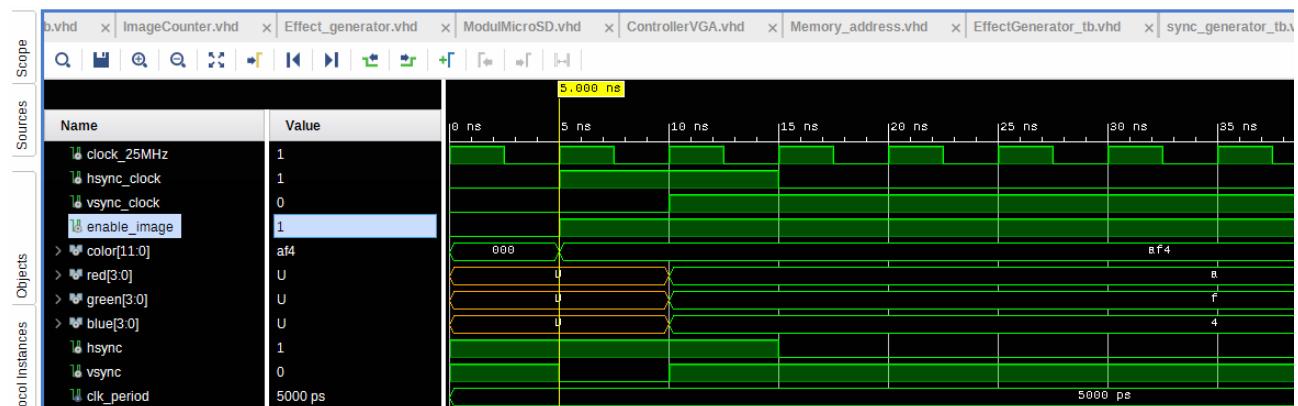


Figura 5.13.1 Simulare ControllerVGA

Pentru BlackBox s-a testat direct pe placă funcționarea circuitului. S-a încercat testarea cu ILA, dar nu s-a putut implementa sistemul din cauză numărului mare de LUT-ruri necesare.

Tabel Statististici

Total power	LUT-uri	Bistabile	Run Strategy
57.137	20097	2775	Flow Runtime_Optimized

5.14 Cod python de extragere biți imagine

S-a creat un cod python care folosește biblioteca PIL pentru extragerea configurației pixelilor unei imagini. Valorile pentru cele trei culori (R, G și B) se regăsesc în intervalul 0..255, iar pentru a le aduce în intervalul 0-15 (pe 4 biti) s-a folosit regula de trei simplă. Valorile obținute se vor scrie pe un fișier binar. Un byte din memoria cardului unde e stocată o imagine reprezintă informația unei culori din RGB-ul unui pixel. Pentru un pixel de imagine sunt necesari astfel 3 bytes din memoria cardului. Codul va genera un fișier .bin cu imaginea, iar informația din acest fișier va fi copiată în memoria cardului cu ajutorul programulu Hxd.



Figure 5.14.1 Fișierul binar obținut în urma conversiei

6. Rezultate și mod de utilizare

Modul de utilizare al sistemului VGA este destul de intuitiv. Utilizatorii au posibilitatea de a naviga între 9 imagini ce sunt disponibile pe cardul microSD. Se pot adăuga și alte imagini pe card, dar trebuie făcute modificările aferente în memoria ROM a adreselor.

Pe lângă schimbarea imaginii afișate pe ecran, utilizatorii mai au 2 opțiuni disponibile precum deplasarea imaginii pe cele două axe cu ajutorul butoanelor, dar și aplicarea de efecte asupra imaginii afișate.

La pornire, se va afișa prima imagine dde pe cardul microSD. Primul led de pe placa va indica faptul că utilizatorul poate naviga între imagini prin apăsarea butoanelor left și right de pe placa Nexys4DDR.

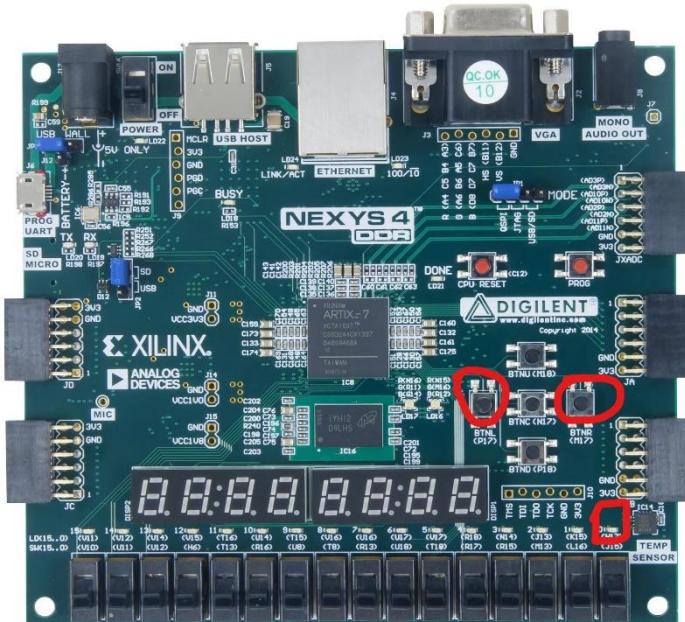


Figura 6.1 Modul de schimbare imagine

Cele 10 poze afișate pe ecran sunt:



Figura 6.2 Imagine 1



Figura 6.3 Imagine 2



Figura 6.4 Imagine 3



Figura 6.5 Imagine 4



Figura 6.6 Imagine 5



Figura 6.7 Imagine 6



Figura 6.8 Imagine 7



Figura 6.9 Imagine 8



Figura 6.10 Imagine 9



Figura 6.11 Imagine 10

Pentru neavigarea între modurile de funcționare, utilizatorul va apăsa pe *butonul din centru*. Al doilea mod este cel de deplasare imagine pe ecran. Acest mod va fi semnalat prin aprinderea celui de-al doilea led de pe placă după ce s-a apăsat butonul din centru.

Deplasarea imaginii se poate face pe cele două axe de coordonate x și y cu ajutorul butoanelor disponibile pe placa nexys 4 DDR.

Un exemplu de deplasare a imaginii se poate observa în figura 6.13

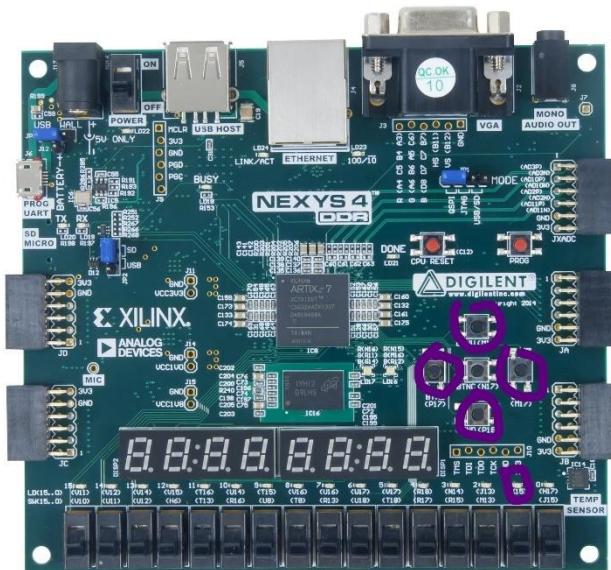


Figura 6.12 Modul de deplasare imagine

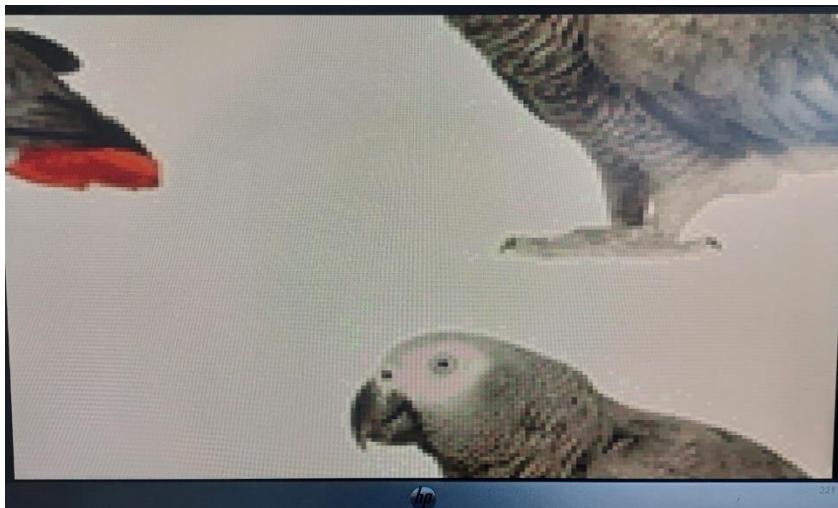


Figura 6.13 Deplasare imagine pe axa x și y

Un alt mod de funcționare este acela de a aplica anumite efecte asupra imaginii curente. Acest mod este semnalat de aprinderea celui de-al treilea led al plăcii și se ajunge în el prin apăsarea butonului din centru. Dacă ne aflăm în acest mod, se pot aplica efecte cu ajutorul switch-urilor. Primul switch (cel mai din dreapta) va aplica un efect de grayscale spre verzui asupra imaginii, al doilea switch va aplica un efect de sepia asupra imaginii, iar dacă nicunul dintre cele 2 efecte nu e aplicat se poate mări aportul de culoare din RGB-ul pixelilor cu ajutorul a 12 switch-uri de pe plăcuță. În figura 6.14 se observă distribuția switch-urilor.

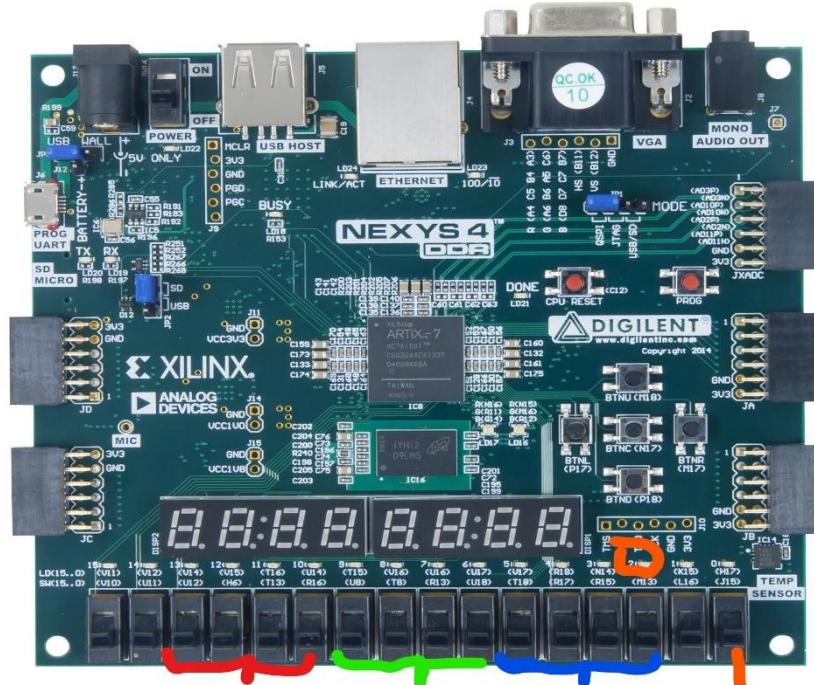


Figura 6.14 Mod aplicare efecte



Figura 6.15 „Grayscale”



Figura 6.16 Sepia



Figura 6.17 Efect obținut prin combinarea celor 12 switch-uri

Ultimul switch (rimul din stânga) va fi resetul aplicat pe operația curentă. De exemplu dacă suntem în modul de deplasare a imaginii, prin activarea reset-ului se va reveni la poziția inițială, iar dacă suntem în modul de schimbare imagine se va reveni la imaginea curentă. Al doilea switch este



resetul dus la unitatea de control și va aduce sistemul la operația inițială, cea de schimbare a imaginii.

7. Concluzii

7.1 Aspecte generale despre rezultatele obținute

Soluția propusă și implementată rezolvă situația în care se pot afișa mai multe imagini pe ecran cu ajutorul plăcuței FPGA. Deoarece resursele plăcii nexys4 DDR sunt limitate, nu se pot stoca în memoria internă a acesteia mai multe imagini și apoi să fie afișate. Folosirea unui card de memorie microSD pentru stocarea imaginilor și citirea lor pe rând pentru a fi afișate limitează considerabil necesitatea de resurse.

Rezultatul obținut nu e în totalitatea ceea ce mi-am dorit, deoarece a trebuit să fac acea scalare pentru a afișa imaginile pe ecran și astfel am pierdut din claritatea lor. Marele avantaj al sistemului implementat este faptul că este simplu de utilizat și de asemnea s-a încercat o implementarea cât mai generală prin folosirea de generice cu scopul de a înlătura modificările ulterioare ale proiectului. S-a creat chiar un pachet special dedicat specificațiilor rezoluției de lucru, iar în implementarea logicii controllerului VGA s-au folosit constante pentru acei timpi ai procesului de scanare a ecranului.

În urma proiectului mi-am îmbunătățit abilitățile de proiectare.

7.2 Dezvoltări ulterioare

În continuare vor fi enumerate câteva idei de dezvoltare a sistemului implementat:

- Mărirea rezoluției de lucru
- Renunțarea la scalare și folosirea memorie DDR a plăcuței FPGA pentru a stoca temporar imaginea de afișat pe ecran
- Implementarea unei logici prin care să nu se hardcodeze adresele imaginilor, ci să se determine automat locația tuturor imaginilor prezente pe cardul microSD
- Adăugarea cursorului mouse-ului și schimbarea imaginilor cu ajutorul mouse-ului
- Stocarea imaginilor modificate prin aplicarea de efecte în memoria cardului



Bibliografie

[1] Hwang, E. O., *Digital Logic and Microprocessor Design With VHDL*, Editura Brooks/Cole, 2005 , pp. 356- 366

[2] *Nexys4 DDR™ FPGA Board Reference Manual*, [Online], Disponibilă: <https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual>

[3] “SD Specifications Part 1 Physical Layer Simplified Specification Version 8.00”, Technical Committee SD Card Association, [Online], Disponibilă: <https://www.sdcards.org/>

[4] Chu, P. P., *FPGA Prototyping by VHDL examples Xilinx Spartan™ -3 Version*, Editura Wiley-Interscience, New Jersey, 2008

[5] Penroni, V. A., *Circuit Design and Simulation with VHDL Second Edition*, Editura The MIT Press, Londra, 2010

[6] Wilson, P. , *Design Recipes for FPGAs Using Verilog and VHDL Second Edition*, Editura Elsevier, 2016

[7] Baruch, Z. F., *Structura sistemelor de calcul*, Editura Albastră, Cluj-Napoca, 2005

[8] *This was CS50*, Havard University, Fall 2019, [Online], Filter - CS50 (harvard.edu) , accesat 21.10.2021

Anexa 1 Inițializare card în modul SPI

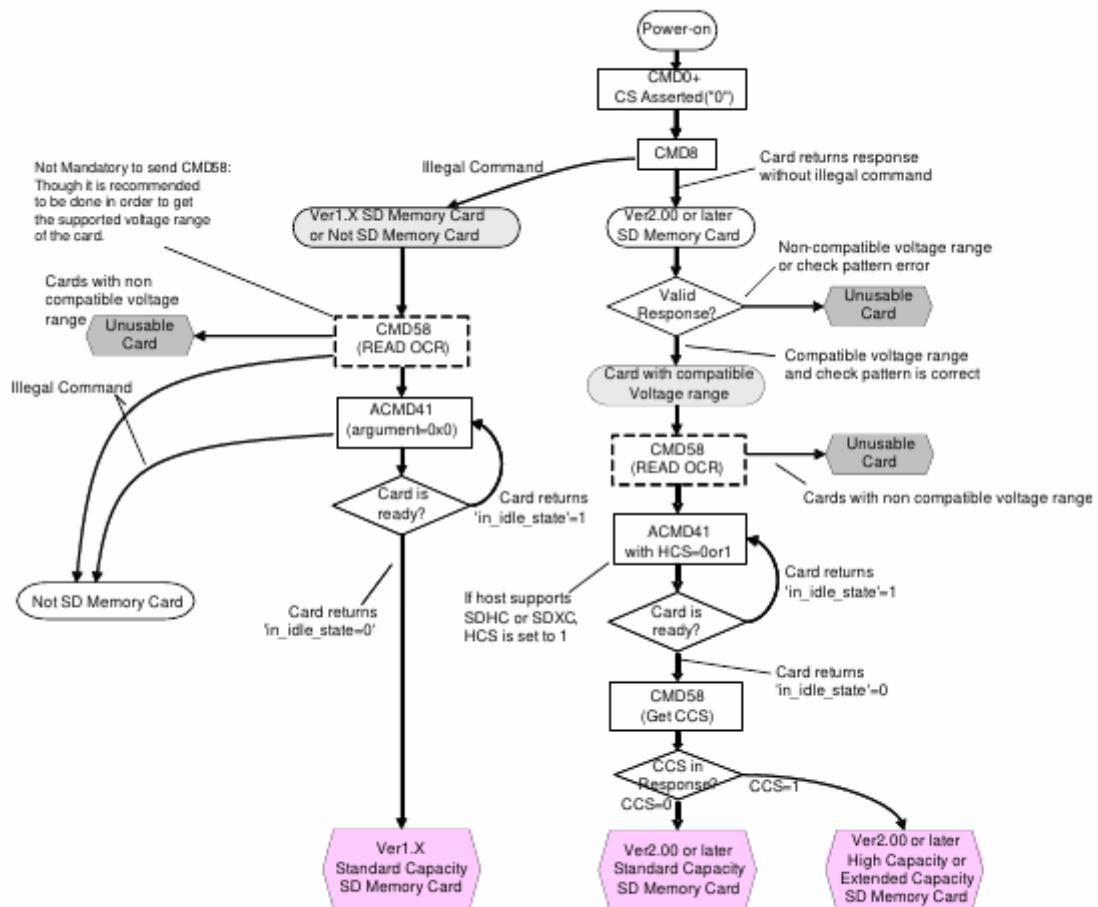


Figure 7-2 : SPI Mode Initialization Flow

Anexa 2 Cod vhdl pentru Control Unit

Cod modul principal ControlUnit.vhd

```
--  
-- Name: Birlutiu Claudiu-Andrei  
-- UTCN CTI-ro  
-- Modul: Unitatea de control  
-----  
--  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
entity ControlUnit is  
    Port ( clk_100MHz :          in STD_LOGIC;  
           btnc :              in STD_LOGIC;  
           reset :             in STD_LOGIC;  
           move_enable :        out STD_LOGIC;  
           effect_enable :      out STD_LOGIC;  
           change_img_enable :  out STD_LOGIC;  
           led_0 :              out STD_LOGIC;  
           led_1 :              out STD_LOGIC;  
           led_2 :              out STD_LOGIC;  
    );  
end ControlUnit;  
  
architecture Behavioral of ControlUnit is  
    --declararea unui tip enumerat pentru stari  
    type STATE_TYPE is (idle, change_image, move_image, effect_image);  
    signal state : STATE_TYPE:=idle;  
begin  
  
    --procesul de trecere in urmatoarea stare  
    gen_next_state: process(clk_100MHz)  
    begin  
        if rising_edge(clk_100MHz) then  
            if reset='1' then          --reset sincron  
                state <=idle;  
            else  
                case state is  
                    when idle =>  
                        state <=change_image;  
                    when change_image =>  
                        if btnc='1' then      --pentru a evita cazul in care  
                            buttonul sta mai mult activ decat perioada de ceas; ar fi trecut prin mai multe  
                            stari  
                            state <= move_image;  
                        end if;  
                    when move_image =>  
                        if btnc='1' then  
                            state <= effect_image;  
                        end if;  
                    when effect_image =>  
                        if btnc='1' then  
                            state <= change_image;  
                        end if;  
                    when others => state<=idle;  
                end case;  
            end if;  
        end if;  
    end process;
```



```
    end if;
end process;

--procesul de determinarea a semnalelor de iesire
generate_outputs: process(state)
begin
    move_enable      <='0';          --ne folosim de proprietatea procesului
care va asigna valoarea semnalului doar la ultima expresie de asignare
    effect_enable    <='0';
    change_img_enable <='0';
    led_0            <='0';
    led_1            <='0';
    led_2            <='0';
    case state is
        when change_image => change_img_enable<='1'; led_0 <='1';
        when move_image   => move_enable<='1';           led_1 <='1';
        when effect_image => effect_enable<='1';         led_2 <='1';
        when others => null ;
    end case;
end process;

end Behavioral;
```

Modul de testare control unit ControlUnit_tb.vhd

```
-----
-- Name: Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Modul: Unitatea de control test bench
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ControlUnit_tb is
end;

architecture bench of ControlUnit_tb is

component ControlUnit
    port (
        clk_100MHz : in STD_LOGIC;
        btnc : in STD_LOGIC;
        switch_4 : in STD_LOGIC;
        move_enable : out STD_LOGIC;
        effect_enable : out STD_LOGIC;
        change_img_enable : out STD_LOGIC;
        led_0 : out STD_LOGIC;
        led_1 : out STD_LOGIC;
        led_2 : out STD_LOGIC
    );
end component;
```



```
-- Clock period
constant clk_period : time := 10 ns;
-- Generics

-- Ports
signal clk_100MHz : STD_LOGIC;
signal btnc : STD_LOGIC:='0';
signal switch_4 : STD_LOGIC;
signal move_enable : STD_LOGIC;
signal effect_enable : STD_LOGIC;
signal change_img_enable : STD_LOGIC;
signal led_0 : STD_LOGIC;
signal led_1 : STD_LOGIC;
signal led_2 : STD_LOGIC;

begin

    DUT : ControlUnit
    port map (
        clk_100MHz => clk_100MHz,
        btnc => btnc,
        switch_4 => switch_4,
        move_enable => move_enable,
        effect_enable => effect_enable,
        change_img_enable => change_img_enable,
        led_0 => led_0,
        led_1 => led_1,
        led_2 => led_2
    );

clk_process : process
begin
    clk_100MHz <= '1';
    wait for clk_period/2;
    clk_100MHz <= '0';
    wait for clk_period/2;
end process clk_process;

-- vom genera semanle pentru btnc; acestea vor oscila intre 0 si 1 dupa o perioada de ceas
gen_signal : process
begin
    switch_4 <='1';
    wait for clk_period;
    if led_0 /= '0' or led_1 /= '0' or led_2 /= '0' or move_enable /= '0' or
change_img_enable /= '0' or effect_enable /= '0' then
        report "FAIL for idle" severity ERROR;
    end if;
    wait for clk_period;
    switch_4 <='0';
    wait for clk_period;
    -----state change_image
    if led_0 /= '1' or led_1 /= '0' or led_2 /= '0' or move_enable /= '0' or
change_img_enable /= '1' or effect_enable /= '0' then
        report "FAIL for change_image" severity ERROR;
    end if;
    wait for clk_period;
    -----state move_image-----
    btnc <='1';
    wait for clk_period;
    if led_0 /= '0' or led_1 /= '1' or led_2 /= '0' or move_enable /= '1' or
change_img_enable /= '0' or effect_enable /= '0' then
```



```
    report "FAIL for move_image" severity ERROR;
end if;

btnc <='0';
wait for clk_period;
-----state effect_image
btnc <='1';
wait for clk_period;
if led_0 /= '0' or led_1 /= '0' or led_2 /= '1' or move_enable /= '0' or
change_img_enable /= '0' or effect_enable /= '1' then
    report "FAIL for effect_image" severity ERROR;
    end if;

btnc <='0';
wait for clk_period;
-----tansition effect_image -> state change_image
btnc <='1';
wait for clk_period;
if led_0 /= '1' or led_1 /= '0' or led_2 /= '0' or move_enable /= '0' or
change_img_enable /= '1' or effect_enable /= '0' then
    report "FAIL for tansition effect_image -> state change_image" se-
verity ERROR;
    end if;
btnc <='0';
wait;
end process;
end;
```



ANEXA 3 Cod VHDL pentru Divizorul de frecventa

Codul modulului principal

```
--  
--Name: Birlutiu Claudiu-Andrei  
--UTCN CTI -ro  
--Divizor de frecventa 25 MHZ  
--  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;           --pentru a folosi tipul std logic  
use ieee.std_logic_unsigned.all;  
use IEEE.NUMERIC_STD.ALL;  
  
entity DivizorFrecventa_25 is  
    Port ( clock_100MHz : in STD_LOGIC  
          , clock_25MHz : out STD_LOGIC  
        );  
end DivizorFrecventa_25;  
  
architecture Behavioral of DivizorFrecventa_25 is  
signal clk_divider: unsigned (15 downto 0) := (others=>'0'  
  
begin  
    process(clock_100Mhz)  
    begin  
        if rising_edge(clock_100Mhz)  
            then clk_divider <= clk_divider + 1;  
        else clk_divider <= clk_divider;  
        end if;  
    end process;  
    clock_25MHz <= clk_divider(1);           --semnalului de  
                                                --iesire ii va fi asignat bitul cel mai semnificativ al numaratorului  
end Behavioral;
```

Modulul de testare DivizorFrecventa_25_tb.vhd

```
--  
--Name: Birlutiu Claudiu-Andrei  
--UTCN CTI -ro  
--Divizor de frecventa 25 MHZ test bench  
--  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
  
entity DivizorFrecventa_25_tb is  
end;  
  
architecture bench of DivizorFrecventa_25_tb is  
  
component DivizorFrecventa_25
```



```
port (
    clock_100MHz : in STD_LOGIC;
    clock_25MHz : out STD_LOGIC
);
end component;

-- Clock period
constant clk_period : time := 10 ns;
-- Generics

-- Ports
signal clock_100MHz : STD_LOGIC;
signal clock_25MHz : STD_LOGIC;

begin

DivizorFrecventa_25_inst : DivizorFrecventa_25
    port map (
        clock_100MHz => clock_100MHz,
        clock_25MHz => clock_25MHz
    );

clk_process : process
begin
    clock_100MHz <= '1';
    wait for clk_period/2;
    clock_100MHz <= '0';
    wait for clk_period/2;
end process clk_process;

end;
```

Codul pentru divizor de frecventa generic

```
-----
-- 
-- Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Divizor de frecventa general
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity DivizorFrecventa_SD is
    generic ( fout: REAL:=0.4);
    Port ( clk100Mhz : in STD_LOGIC;
            clkDivised : out STD_LOGIC);
end DivizorFrecventa_SD;

architecture Behavioral of DivizorFrecventa_SD is
```



```
signal scale: NATURAL:=integer( 100.0 / fout);
signal counter: NATURAL:=0;
signal clk_reg : STD_LOGIC:='0';
begin

div: process(clk100MHz)
begin
  if rising_edge(clk100MHz) then
    if counter = scale/2 -1 then
      clk_reg <= not clk_reg;
      counter <=0;
    else
      counter <= counter+1;
    end if;
  end if;
end process;

clkDivised <= clk_reg;

end Behavioral;
```

ANEXA 4 Codul VHDL pentru Sync_generator

Modulul principal Sync_generator.vhd

```

--  

-- Birlutiu Claudiu-Andrei  

-- UTCN CTI-ro  

-- GENERATOR DE ADRESE  

--  

--  

library IEEE;  

use IEEE.STD_LOGIC_1164.ALL;          --contine tipul std_logic  

use IEEE.STD_LOGIC_ARITH.ALL;          --pentru calcule signal anumite conversii  

use IEEE.std_logic_unsigned.all;        --permite adunari directe pe vectori de  

biti/std_logic  

use work.VgaPackage.all;               --AM ALES O REZOLUTIE DE 640/480  

--aceasta componenta genereaza semnalele de timp si sincronizare ale Portului VGA  

entity sync_generator is  

  Port (  

    clock_25MHz : in STD_LOGIC;  

    addr_x : out STD_LOGIC_VECTOR (9 downto 0);  

    addr_y : out STD_LOGIC_VECTOR (9 downto 0);  

    hsync : out STD_LOGIC;  

    vsync : out STD_LOGIC;  

    enable_image: out STD_LOGIC);  

end sync_generator;  

--obs datele output despre pixeli se dau serial  

architecture Behavioral of sync_generator is  

--diagramele de timing detaliate se gasesc in documentatie  

--am folosit constante pentru a fi usor modificar rezolutie ca metoda de dez-  

voltare ulterioara a programului  

-- numaratoare pentru pozitia pe orizontala, respectiv pe verticala; sunt ini-  

tializate cu 0  

signal h_cntr: std_logic_vector(9 downto 0) := (others =>'0');  

signal v_cntr : std_logic_vector(9 downto 0) := (others =>'0');  

--semnalul de enable al imaginii care este initializat cu 1  

signal enable_image_h: std_logic := '1';  

signal enable_image_v: std_logic := '1';  

signal enable_v: STD_LOGIC:='0';  

begin
  --numaratorul pentru pozitia orizontala
  horizontal_counter: process (clock_25MHz)
    --  

procesul depinde doar de clk_25, adica inaintam cu numaratorul in functie doar  

de acest pixel rate de 25MHz
    begin
      if (rising_edge(clock_25MHz)) then
        if (h_cntr = (H_max - 1)) then           --in momentul in care ajunge
la limita din drepta (799) acesta se reseteaza
          h_cntr <= (others =>'0');
        else

```



```
        h_cntr <= h_cntr + 1;
    end if;
end if;
end process;

--semnalul de enable pentru numaratorul pe verticala
enable_v <= '1' when h_cntr=H_max - 1 else '0';
--numaratorul pe verticala;
-- acesta merge in paralel (concurrent) cu processul pentru orizontala
vertical_counter: process (clock_25MHz)
begin
    if (rising_edge(clock_25MHz)) then
        if enable_v = '1' then
            if v_cntr = V_max-1 then
                v_cntr <= (others =>'0');
            else
                v_cntr <= v_cntr + 1;
            end if;
        end if;
    end if;
end process;

fsm_horizontal: entity work.fsm_horizontal_sync PORT MAP(
    clk_25MHz => clock_25MHz,
    counter_h => h_cntr,
    h_sync => hsync,
    enable_image_h =>enable_image_h
);
fsm_vertical: entity work.fsm_vertical_sync PORT MAP(
    clk_25MHz => clock_25MHz,
    counter_v => v_cntr,
    enable_v => enable_v,
    v_sync => vsync,
    enable_image_v =>enable_image_v
);

--semnalul intern pentru validare afisare pixe daca se afla in zona de
display; in acest caz se compara pozitia unui pixel cu limitele VD si HD
    enable_image <='1' when enable_image_h='1' and enable_image_v ='1'
                           else '0';

--se asigneaza porturilor de iesire semnalele interne corespunzatoare
addr_x  <= h_cntr;
addr_y  <= v_cntr;

end Behavioral;
```

CODUL pentru fsm_horizontal_sync.vhd

```
--  
-- Birlutiu Claudiu -Andrei  
-- Fsm pentru scanarea pe orizontala
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```

use work.VgaPackage.all;

entity fsm_horizontal_sync is
  Port ( clk_25MHz :      in STD_LOGIC;
         counter_h :      in STD_LOGIC_VECTOR(9 downto 0);
         h_sync :          out STD_LOGIC;
         enable_image_h : out STD_LOGIC);
end fsm_horizontal_sync;

architecture Behavioral of fsm_horizontal_sync is
  signal state_horizontal : STATE_H:=S_HD;
  signal h_sync_reg :        STD_LOGIC:= not(H_pol);
  signal enable_image_h_reg: STD_LOGIC:=not(H_POL);
begin
  --procesul pentru starea urmatoare
  gen_horizontal_state: process(clk_25MHz)
  begin
    if rising_edge(clk_25MHz) then
      case state_horizontal is
        when S_HD =>
          if counter_h < HD-1 then
            state_horizontal <= S_HD;
          else
            state_horizontal <= S_HF;
          end if;
        when S_HF =>
          if counter_h < HD+HF-1 then
            state_horizontal <= S_HF;
          else
            state_horizontal <= S_HR;
          end if;
        when S_HR =>
          if counter_h < HD+HF+HR-1 then
            state_horizontal <= S_HR;
          else
            state_horizontal <= S_HB;
          end if;
        when S_HB =>
          if counter_h < HD+HF+HR+HB-1 then
            state_horizontal <= S_HB;
          else
            state_horizontal <= S_HD;
          end if;
        when others=> state_horizontal <= S_HD;
      end case;
    end if;
  end process gen_horizontal_state;

  --procesul pentru generare iesiri
  gen_outputs_h: process(state_horizontal)
  begin
    case state_horizontal is
      when S_HR =>
        h_sync_reg <= H_pol;
        enable_image_h <='0';
      when S_HD =>
        enable_image_h <='1';
        h_sync_reg <=not(H_pol);
      when others =>
        h_sync_reg <=not(H_pol);
        enable_image_h <='0';
    end case;
  end process gen_outputs_h;

```



```
    h_sync <= h_sync_reg;  
  
end Behavioral;
```

Codul pentru fsm_vertical_sync.vhd

```
--  
-- Birlutiu Claudiu-Andrei  
-- UTCN CTI-ro  
-- Fsm pentru scanare pe verticala  
--  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
use work.VgaPackage.all;  
  
entity fsm_vertical_sync is  
    Port ( clk_25MHz : in STD_LOGIC;  
           counter_v : in STD_LOGIC_VECTOR(9 downto 0);  
           enable_image_v : out STD_LOGIC;  
           enable_v       : in STD_LOGIC;  
           v_sync         : out STD_LOGIC);  
end fsm_vertical_sync;  
  
architecture Behavioral of fsm_vertical_sync is  
    signal state_vertical      : STATE_V:=S_VD;  
    signal v_sync_reg : STD_LOGIC:= not(V_POL);  
  
begin  
  
    --procesul pentru starea urmatoare  
    gen_next_state_vertical: process(clk_25MHz)  
        begin  
            if rising_edge(clk_25MHz) then  
                case state_vertical is  
                    when S_VD =>  
                        if counter_v = VD-1 and enable_v='1' then  
                            state_vertical <= S_VF;  
                        else  
                            state_vertical <= S_VD;  
                        end if;  
                    when S_VF =>  
                        if counter_v = VD+VF-1 and enable_v='1' then  
                            state_vertical <= S_VR;  
                        else  
                            state_vertical <= S_VF;  
                        end if;  
                    when S_VR =>  
                        if counter_v = VD+VF+VR-1 and enable_v='1' then  
                            state_vertical <= S_VB;  
                        else  
                            state_vertical <= S_VR;  
                        end if;  
                    when S_VB =>  
                        if counter_v = VD+VF+VR+VB-1 and enable_v='1' then  
                            state_vertical <= S_VD;
```



```
        else
            state_vertical <= S_VB;
        end if;
    end case;
end if;
end process gen_next_state_vertical;

--procesul pentru iesiri
gen_outputs_v: process(state_vertical)
begin
    case state_vertical is
        when S_VR =>
            v_sync_reg <= V_pol;
            enable_image_v <='0';
        when S_VD =>
            enable_image_v <='1';
            v_sync_reg <=not(V_pol);
        when others =>
            v_sync_reg <=not(V_pol);
            enable_image_v <='0';
    end case;
end process gen_outputs_v;

v_sync <= v_sync_reg;

end Behavioral;
```

Codul pentru testare sync_generator_tb.vhd

```
-- 
-- Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- modul de de testare sync_generator
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sync_generator_tb is
-- Port ();
end sync_generator_tb;

architecture Behavioral of sync_generator_tb is
    signal clock_25MHz : std_logic:='1'; --clockul este
de 25MHz , pixel rate-ul pentru o rezolutie de 640/480; aceasta inseamna ca 25 M
pixels sunt procesati intr-o secunda
    signal addr_x : std_logic_vector (9 downto 0):=(others=>'0');
    signal addr_y : std_logic_vector (9 downto 0):=(others=>'0');
    signal hsync : std_logic:='0'; --hsync si
vsync reprezinta semnalele care controleaza scanarea monitorului pe orizontala,
respectiv verticala
    signal vsync : std_logic:='0';
    signal enable_image : std_logic:='0';
begin

    clock_25MHz<= not(clock_25MHz) after 5 ns;
    DUT: entity work.sync_generator PORT MAP (
        clock_25MHz=>clock_25MHz,
```



```
addr_x => addr_x,  
addr_y => addr_y,  
hsync => hsync,  
vsync => vsync,  
enable_image=>enable_image  
) ;  
  
end Behavioral;
```

ANEXA 5 Codul pentru modulul MPG

Codul modulului mpg.vhd

```
-----
-- Name:Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Monopulse generator
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mpg is
    Port (btn:    in STD_LOGIC;
          clk:    in STD_LOGIC;
          enable: out STD_LOGIC);
end mpg;

architecture Behavioral of mpg is
    signal counter: STD_LOGIC_VECTOR(15 downto 0):=x"0000";
    signal en: STD_LOGIC:='0';
    signal Q1: STD_LOGIC:='0';
    signal Q2: STD_LOGIC:='0';
    signal Q3: STD_LOGIC:='0';
begin
    --un counter pe 16 biti
    counter1: process(clk)
        begin
            if rising_edge(clk) then
                counter<=counter+1;
            end if;
        end process;
    --enable-ul de la acest counter se primeste la ultima stare a lui
    en<='1' when counter=x"ffff" else '0';

    --in primul bistabil se pune valoarea de la buton
    reg1: process(clk)
        begin
            if rising_edge(clk) then
                if en='1' then
                    Q1<=btn;
                end if;
            end if;
        end process;

    reg2: process(clk)
    begin
        if rising_edge(clk) then
            Q2<=Q1;
        end if;
    end process;

    reg3: process(clk)
        begin

```

```

        if rising_edge(clk) then
            Q3<=Q2;
        end if;
    end process;
    --va genera un singur impuls sincron cu semnalul de ceas, chiar daca se
    mentine butonul apasat
    enable<=Q2 AND not Q3;

end Behavioral;

```

Codul pentru modulul de test mpg_tb.vhd

```

-- 
-- Name:Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Monopulse generator
-- 

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mpg_tb is
end;

architecture bench of mpg_tb is

component mpg
    port (
        btn : in STD_LOGIC;
        clk : in STD_LOGIC;
        enable : out STD_LOGIC
    );
end component;

-- Clock period
constant clk_period : time := 5 ns;
-- Generics

-- Ports
signal btn : STD_LOGIC:='0';
signal clk : STD_LOGIC:='0';
signal enable : STD_LOGIC:='0';

begin

mpg_inst : mpg
    port map (
        btn => btn,
        clk => clk,
        enable => enable
    );

clk_process : process
begin
    clk <= '1';
    wait for clk_period/2;
    clk <= '0';
    wait for clk_period/2;

```



```
end process clk_process;

gen_test: process
begin
    btn <='1';
    wait for 2 ns;
    btn<='0';
    wait for 2 ns;
    if enable ='1' then
        report "ERROR for first test" severity ERROR;
    end if;
    btn <='1';
    wait for 2**16 * 10 ns;

    if enable ='1' then
        report "ERROR";
    end if;
    wait;
end process gen_test;
end;
```

ANEXA 6 Codul pentru modulul Offset Counter

Codului modulului offset_counter.vhd

```

-----
-- Birlutiu Claudiu-Andrei
-- Utcn CTI-ro
-- Numaratoarele de offset
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;          --pentru folosirea tipului std_logic
use IEEE.STD_LOGIC_ARITH.ALL;         --pentru operatii matematice
use IEEE.std_logic_unsigned.all;       --pentru a putea a face operatii de adunare/scadere etc. pe vectori de biti/std_logic
use ieee.math_real.all;
use work.VgaPackage.all;

entity offset_counter is
    Port ( clk :      in STD_LOGIC;
           enable :     in STD_LOGIC;
           rst :        in STD_LOGIC;
           left :       in STD_LOGIC;
           right :      in STD_LOGIC;
           up :         in STD_LOGIC;
           down :       in STD_LOGIC;
           offset_x :   out natural range 0 to HD-1;
           offset_y :   out natural range 0 to VD-1);
end offset_counter;

architecture Behavioral of offset_counter is

--ne luam semnale interne echivalente unor numaratoare modulo 640, respectiv
480; aceste se initializeaza cu 0
signal x_pos: natural range 0 to HD-1 := 0;
signal y_pos: natural range 0 to VD-1 := 0;

begin
    horizontal_counter: process(clk)
    begin
        if enable = '1' then
            if rising_edge(clk) then
                if rst = '1' then
                    x_pos <= 0;
                else
                    if left = '1' then
                        if x_pos = 639 then
                            x_pos <= 0;
                        else
                            x_pos <= x_pos + 1;           --se incrementeaza
deoarece prin deplasare la stanga se GRABESTE afisarea pixelului, relativ cu
pozitia reala a acestuia data de generatorul de adrese
                    end if;
                elsif right = '1' then

```



```
        if(x_pos = 0) then
            x_pos <= 639;
        else
            x_pos <= x_pos - 1;
        end if;
    end if;
end if;
end process;

vertical_counter: process(clk)
begin
if enable = '1' then
    if rising_edge(clk) then
        if rst = '1' then
            y_pos <= 0;
        elsif down = '1' then
            if y_pos = 0 then
                y_pos <= 479;
            else
                y_pos <= y_pos - 1;           --in acest caz se face decrementare deoarece se INTARZIE afisarea pixelului pe verticala
            end if;
        elsif up = '1' then
            if y_pos = 479 then
                y_pos <= 0;
            else
                y_pos <= y_pos + 1;
            end if;
        end if;
    end if;
end if;
end process vertical_counter;

--se asigneaza semnalele interne porturilor corespunzatoare
offset_x <= x_pos;
offset_y <= y_pos;

end Behavioral;
```

Codul pentru modulul de testare offset_counter_tb.vhd

```
--  
-- Birlutiu Claudiu-Andrei  
-- Utcn CTI-ro  
-- Numaratoarele de offset  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
use work.VgaPackage.all;  
entity offset_counter_tb is  
end;  
  
architecture bench of offset_counter_tb is  
  
component offset_counter  
port (
```

```

clk :      in STD_LOGIC;
enable :   in STD_LOGIC;
rst :      in STD_LOGIC;
left :     in STD_LOGIC;
right :    in STD_LOGIC;
up :       in STD_LOGIC;
down :     in STD_LOGIC;
offset_x : out natural range 0 to 639;
offset_y : out natural range 0 to 479
);
end component;

-- Clock period
constant clk_period : time := 10 ns;
-- Generics

-- Ports
signal clk : STD_LOGIC:='0';
signal enable : STD_LOGIC:='0';
signal rst : STD_LOGIC:='0';
signal left : STD_LOGIC:='0';
signal right : STD_LOGIC:='0';
signal up : STD_LOGIC:='0';
signal down : STD_LOGIC:='0';
signal offset_x : natural range 0 to 639;
signal offset_y : natural range 0 to 479;

begin

offset_counter_inst : offset_counter
port map (
    clk => clk,
    enable => enable,
    rst => rst,
    left => left,
    right => right,
    up => up,
    down => down,
    offset_x => offset_x,
    offset_y => offset_y
);

clk_process : process
begin
    clk <= '1';
    wait for clk_period/2;
    clk <= '0';
    wait for clk_period/2;
end process clk_process;

gen_inputs: process
begin
    if offset_x /=0 or offset_y/=0 then
        report "EROARE LA INITIALIZARE" severity ERROR;
    end if;
    wait for clk_period;
    -----
    up<='1';
    enable<='1';
    wait for clk_period;
    if offset_y/=1 then
        report "EROARE LA Incrementare verticala" severity ERROR;
    end if;

```

```

-----  

up<='0';  

down<='1';  

wait for clk_period;  

if offset_y/=0 then  

  report "EROARE LA decrementare verticala" severity ERROR;  

end if;  

-----  

down<='0';  

right<='1';  

wait for clk_period;  

if offset_x/= HD-1 then  

  report "EROARE LA decrementare orizontala" severity ERROR;  

end if;  

-----  

right<='0';  

left <='1';  

wait for clk_period;  

if offset_x/= 0 then  

  report "EROARE LA incrementare orizontala" severity ERROR;  

end if;  

left<='0';  

enable<='0';  

wait for clk_period;  

left<='1'; up<='1';  

wait for clk_period;  

if offset_x/= 0 and offset_y/= 0 then  

  report "EROARE LA semnalul enable" severity ERROR;  

end if;  

left<='0'; up<='0';  

wait;  

end process;  

end;  


```



ANEXA 7 Modulul Memory address

Codul pentru memory_address.vhd

```
-----
--Birlutiu Claudiu-Andrei
--UTCN CTi-ro
--modul folosit pentru calculul adreselor
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.std_logic_unsigned.all;
use ieee.math_real.all;
use work.VgaPackage.all;

--ACEASTA COMPOENTA COMBINA INFORMATIILE DE LA NUMARATORUL DE OFFSET SI GENERATORUL DE ADRESE
--aceasta da informatia cand sa se afiseze un pixel
entity memory_address is
    Port ( enable : in STD_LOGIC;
            offset_x : in natural range 0 to HD-1;
            offset_y : in natural range 0 to VD-1;
            addr_x :      in STD_LOGIC_VECTOR (9 downto 0);
            addr_y :      in STD_LOGIC_VECTOR (9 downto 0);
            x_mem_addr : out natural range 0 to 159;
            y_mem_addr : out natural range 0 to 119);

end memory_address;

architecture Behavioral of memory_address is
--pentru generalitate se folosesc constante
    constant Height: natural := VD;
    constant Width: natural := HD;
begin
    --se aduna adresa unde ar fi pixelul, data de generatorul de adrese in functie de pixel_rate (deci, cea data in urma scanari ecranului) cu valoarea de deplasare si se face modulo pentru a ramane in intervalul de display
    --aceasta se realizeaza cand sistemul e in zona de display
    --prin operatia de modulo, cand se depasese limitea superioara, rezultatul va incepe de la limita inferioara ; 640%640=0 -> deci, pixelul va fi afisat in partea stanga a ecranului
    x_mem_addr <= ((conv_integer(addr_x) + offset_x) mod Width)/4 when enable = '1' else 0;
    y_mem_addr <= ((conv_integer(addr_y) + offset_y) mod Height)/4 when enable = '1' else 0;
end Behavioral;
```

Codul pentru testare memory_address_tb.vhd

```

-- 
--Birlutiu Claudiu-Andrei
--UTCN CTi-ro
--modul folosit pentru calculul adreselor
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use work.VgaPackage.all;

entity memory_address_tb is
end;

architecture bench of memory_address_tb is

component memory_address
    port (
        enable : in STD_LOGIC;
        offset_x : in natural range 0 to HD-1;
        offset_y : in natural range 0 to VD-1;
        addr_x : in STD_LOGIC_VECTOR (9 downto 0);
        addr_y : in STD_LOGIC_VECTOR (9 downto 0);
        x_mem_addr : out natural range 0 to 639;
        y_mem_addr : out natural range 0 to 479
    );
end component;

-- Clock period
constant clk_period : time := 5 ns;
-- Generics

-- Ports
signal enable : STD_LOGIC:='0';
signal offset_x : natural range 0 to HD-1:=0;
signal offset_y : natural range 0 to VD-1:=0;
signal addr_x : STD_LOGIC_VECTOR (9 downto 0):=(others =>'0');
signal addr_y : STD_LOGIC_VECTOR (9 downto 0):=(others =>'0');
signal x_mem_addr : natural range 0 to 639:=0;
signal y_mem_addr : natural range 0 to 479:=0;

begin

memory_address_inst : memory_address
    port map (
        enable => enable,
        offset_x => offset_x,
        offset_y => offset_y,
        addr_x => addr_x,
        addr_y => addr_y,
        x_mem_addr => x_mem_addr,
        y_mem_addr => y_mem_addr
    );

```

```

gen_inputs: process
begin
    enable <='1';
    offset_x <= 600;
    addr_x <= conv_std_logic_vector(638,10);
    wait for 10 ns;
    if x_mem_addr /= (600 + 638) mod 640 then
        report "EROARE in timpul calcularii adresei pe orizontala" severity
error;
    end if;

    offset_y <= 24;
    addr_y <= conv_std_logic_vector(470,10);
    wait for 10 ns;
    if y_mem_addr /= (24 + 470) mod 480 then
        report "EROARE in timpul calcularii adresei pe verticala" severity
error;
    end if;

    enable <='0';
    wait for 10 ns;
    if y_mem_addr /=0 or x_mem_addr /=0 then
        report "Eroare la dezactivarea semnalului de enable; rezultatul ar
fi trebuit sa fie 0 pentru ambele adrese";
    end if;
    wait;
end process;
end;

```

ANEXA 8 Modulul SDInitialise

Codul pentru SDInitialise.vhd

```
--  
-- Birlutiu Claudiu-Andrei  
-- UTCN CTI-ro  
-- Modul initializare card in modul SPI  
-----  
--  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
entity SDinitialise is  
    Port ( clk100MHz : in STD_LOGIC;  
            reset :      in STD_LOGIC;  
            enable:      in STD_LOGIC;  
  
            SCK:        out std_logic;  
            CS :        out std_logic;  
            MOSI:       out std_logic;  
            MISO:       in  std_logic;  
  
            tipSD:      out STD_LOGIC_VECTOR(1 downto 0);  
            finish:     out STD_LOGIC  
  
        );  
end SDinitialise;  
  
architecture Behavioral of SDinitialise is  
    type INIT_STATE is (  
        START,           --stare initiala  
        WAIT_STATE,      -- stare in care se asteapta cel putin 74 de impulsuri  
        SEND_CMD0,       -- trimitere CMD0  
                        --verificare raspuns CMD0  
        RESPONSE_CMD0,   --trimitere comanda CMD8  
        SEND_CMD8,       --verificare raspuns CMD8  
        RESPONSE_CMD8,   --SEND_CMD55,  
                        --verificare raspuns CMD55  
        SEND_CMD55,      --SEND_ACMD41,  
                        --verificare raspuns ACMD41  
        RESPONSE_ACMD41, --SEND_CMD58,  
                        --verificare raspuns la comanda CMD58  
        SEND_CMD58,      --SDHC_SD,  
                        --card SDHC  
        STD_SD,          --card standard  
        SEND_CMD16,  
        RESPONSE_CMD16,  
        TRANSFER_CMD,    --stare in care se transmit bitii de comanda  
        GET_RESPONSE,    --stare in care se primeste raspunsul de la card  
        CONTINUE_RESPONSE, --continuare primire raspuns  
        ILLEGAL,  
        STOP  
    );  
  
    --definirea semenalelor de stare  
    signal state: INIT_STATE:=START;          --starea curenta  
    signal return_state: INIT_STATE:=START;    --starea ce urmeaza dupa executia  
unei rutine de transfer sau primire de date
```



```
constant CMD_LENGTH: INTEGER:=56;
--DEFINIRE COMENZI DE TRIMIS; inainte se pune un octet xFF pentru siguranta
transmiterii comenziilor
constant CMD0 : STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_000000" & x"00000000" & x"95";
constant CMD8: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_001000" & x"000001aa" & x"87";
constant CMD55: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_110111" & x"00000000" & x"01"; --nu conteaza CRC-ul aici
constant ACMD41: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_101001" & x"40000000" & x"01"; --nu conteaza CRC-ul aici
constant CMD58: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_111010" & x"00000000" & x"ff"; --nu conteaza CRC-ul aici
constant CMD16 : STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_010000" & x"00000200" & x"FF"; --nu conteaza crc-ul; va seta blocul de
date citit; 512- bytes

--definire semnal process automat de stare
signal clk : STD_LOGIC:='0';
signal CS_reg: STD_LOGIC:='1';

--semnal ce descrie comanda curenta
signal currentCMD: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=CMD0;

--declarare semnal pentru primirea raspunsului de la card
constant MAX_LENGTH_RESP: INTEGER:=40;
signal response : STD_LOGIC_VECTOR(MAX_LENGTH_RESP-1 downto
0):=(others=>'0');

--constant valid responses
constant IDLE_NO_ERRORS_C0 : STD_LOGIC_VECTOR(7 downto 0) := "00000001"; -
- Normal R1 code after CMD0.
constant FINISH_ACMD41 : STD_LOGIC_VECTOR(7 downto 0) := x"00"; -
-Terminarea initializarii

--semnale auxiliare pentru cele doua ieisiri
signal SCK_reg : STD_LOGIC:='0';
signal tipSD_reg : STD_LOGIC_VECTOR(1 downto 0):="00";
begin

init_process: process(clk)
variable bitsCounter : INTEGER:=0;
variable wait_time_init : INTEGER:= 160;
variable type_response : STD_LOGIC:='0';
variable finish_inialise : STD_LOGIC:='1';

begin
if enable='1' then
if rising_edge(clk) then
if reset = '1' then
SCK_reg <= '0';
state <= START;
else
case state is
when START =>
SCK_reg <= '0';
currentCMD <= (others => '1');
CS_reg <= '1';
wait_time_init := 160;
finish_inialise:='0';
state <= WAIT_STATE;
```



```
when WAIT_STATE =>
    if wait_time_init = 0 then
        state <= SEND_CMD0;
    else
        wait_time_init := wait_time_init - 1;
        SCK_reg <= not SCK_reg;
    end if;

when SEND_CMD0 =>
    CS_reg <= '0';
    bitsCounter := CMD_LENGTH-1;
    currentCMD <= CMD0;
    type_response := '0';
    state <= TRANSFER_CMD;
    return_state <= RESPONSE_CMD0;

when RESPONSE_CMD0 =>
    if response(7 downto 0) = IDLE_NO_ERRORS_C0 then
        state <= SEND_CMD8;
    else
        state <= SEND_CMD0;
    end if;

when SEND_CMD8 =>
    CS_reg <= '0'; -- se va activa CS_reg-ul
    bitsCounter := CMD_LENGTH-1;
    type_response := '1';
    currentCMD <= CMD8;
    state <= TRANSFER_CMD;
    return_state <= RESPONSE_CMD8;

when RESPONSE_CMD8 =>
    if response(34) ='1' then
        state <= ILLEGAL;
    elsif response(15 downto 8) /= x"01" then
        state <= ILLEGAL;
    else
        state <= SEND_CMD58;
    end if;

when SEND_CMD58 =>
    CS_reg <= '0'; -- se va activa CS_reg-ul
    bitsCounter := CMD_LENGTH-1;
    type_response := '1';
    currentCMD <= CMD58;
    state <= TRANSFER_CMD;
    return_state <= RESPONSE_CMD58;

when RESPONSE_CMD58 =>
    if response(34) ='1' then
        state <= ILLEGAL;
    elsif finish_inialise ='1' then
        if response(30) = '1' then
            state <= SDHC_SD;
        else
            state <= STD_SD;
        end if;
    else
        state <= SEND_CMD55;
    end if;
```



```
when SEND_CMD55 =>
    CS_reg <= '0';                      -- se va activa CS_reg-ul
    bitsCounter := CMD_LENGTH-1;
    type_response := '0';                --R1 rapsuns
    currentCMD <= CMD55;
    state <= TRANSFER_CMD;
    return_state <= RESPONSE_CMD55;

when RESPONSE_CMD55 =>
    state <= SEND_ACMD41;

when SEND_ACMD41 =>
    CS_reg <= '0';                      -- se va activa CS_reg-ul
    bitsCounter := CMD_LENGTH-1;
    type_response := '0';                --R1 rapsuns
    currentCMD <= ACMD41;
    state <= TRANSFER_CMD;
    return_state <= RESPONSE_ACMD41;

when RESPONSE_ACMD41 =>
    if response(7 downto 0) = IDLE_NO_ERRORS_C0 then
        state <= SEND_CMD55;
    elsif response(7 downto 0) = FINISH_ACMD41 then
        finish_initialise := '1';
        state <= SEND_CMD58;
    else
        state <= ILLEGAL;
    end if;

when TRANSFER_CMD =>
    if SCK_reg = '1' then
        if bitsCounter = 0 then
            state <= GET_RESPONSE;
        else
            bitscounter := bitscounter -1;
            currentCMD <= currentCMD(CMD_LENGTH-2 downto
0) & '1';
        end if;
    end if;
    SCK_reg <= not SCK_reg;

when GET_RESPONSE =>
    if SCK_reg = '1' then
        if MISO = '0' then
            response <= response(MAX_LENGTH_RESP-2
downto 0) & MISO;
            if type_response = '0' then
                bitsCounter := 6;
            else
                bitsCounter := 38;
            end if;
            state <= CONTINUE_RESPONSE;
        end if;
    end if;
    SCK_reg <= not SCK_reg;

when CONTINUE_RESPONSE =>
    if SCK_reg = '1' then
```



```
        response <= response(MAX_LENGTH_RESP-2 downto 0)
& MISO;
      if bitsCounter = 0 then
        --CS_REG <='1';           --se va deselecta cardul
        state <= return_state;
      else
        bitsCounter := bitsCounter - 1;
      end if;
    end if;
    SCK_reg <= not SCK_reg;

    when SDHC_SD =>
      tipSD_reg <= "01";
      state <= STOP;
    when STD_SD =>
      state <= SEND_CMD16;

    when SEND_CMD16 =>
      CS_reg <= '0';
      bitsCounter := CMD_LENGTH-1;
      currentCMD <= CMD16;
      type_response := '0';
      state <= TRANSFER_CMD;
      return_state <= RESPONSE_CMD16;

    when RESPONSE_CMD16 =>
      if response(2) = '1' then
        state <= ILLEGAL;
      else
        tipSD_reg <= "11";
        state <= STOP;
      end if;

    when ILLEGAL =>
      tipSD_reg <= "10";
      state <= STOP;

    when STOP => null;
    when others => state <= START;
  end case;

  end if;
end if;
end if;
end process init_process;

tipSd <= tipSD_reg;
finish <= '1' when state = STOP else '0';

--setarea semnalelor de iesire cu cele interne pentru conectorul de card
SCK <= SCK_reg;
MOSI <= currentCMD(CMD_LENGTH-1);
CS <= CS_reg;

--devizor de ceas pentru obtinerea unui semnal cu frecventa de 400KHz- dublul
frecventei semnalului SCK
DivizorFrecventa_SD_inst : entity work.DivizorFrecventa_SD
  generic map (
    fout => 0.4
  )
  port map (
```



```
clk100Mhz => clk100Mhz,  
clkDivised => clk  
);  
  
end Behavioral;
```

ANEXA 9 Modulul SDControllerRead

Codul pentru SDController.vhd

```

-- 
-- Birlutiu Claudiu-Andrei
-- UTCN CTI -ro
-- Modul SD controller
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SDControllerRead is
    Port ( clk100MHz      : in STD_LOGIC;      --semnal de ceas de 100MHz
           reset          : in STD_LOGIC;
           enable          : in STD_LOGIC;
           address         : in STD_LOGIC_VECTOR(31 downto 0);
           read_enable     : in STD_LOGIC;

           in_work         : out STD_LOGIC;

           --semmale specifice conectorului microSD
           SCK            : out STD_LOGIC;        --clock transmis cardului
           MOSI           : out STD_LOGIC;
           MISO           : in STD_LOGIC;
           CS             : out STD_LOGIC;

           ready           : out STD_LOGIC;
           finish_read_blk: out STD_LOGIC;
           dataOut         : out std_logic_vector(512*8-1 downto 0);
           ilegal          : out std_logic:='0');
end SDControllerRead;

architecture Behavioral of SDControllerRead is
    type SD_STATE is (
        START,                --stare initiala
        WAIT_STATE,           --stare in care se asteapta semnalul de read de la host
        ca se poate realiza citirea
        SEND_CMD17,            --comanda pentru citirea unui block
        RESPONSE_CMD17,        --primirea raspunsului R1 pentru CMD17
        READ_BLOCK,            --se va incepe citirea blocului
        GET_BYTE,
        TRANSFER_CMD,          --stare in care se transmit bitii de comanda
        GET_RESPONSE,          --stare in care se primeste raspunsul de la card
        CONTINUE_RESPONSE,     --continuare citire raspuns
        ILLEGAL
    );
    constant CMD_LENGTH: INTEGER:=56;
    constant CMD17      : STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=x"FF" &
b"01_010001" & x"00000000" & x"FF";  --va seta adresa de la care se porneste
citirea

    --declarare semnale pentru stari
    signal state : SD_STATE:= START;
    signal return_state: SD_STATE:= START;

```



```
--semnal auxiliar pentru chip select
signal CS_reg: std_logic:='1';

--semnal ce descrie perioada curenta
signal currentCMD: STD_LOGIC_VECTOR(CMD_LENGTH-1 downto 0):=(others =>'1');

--definire semnal de raspuns
constant MAX_LENGTH_RESP : INTEGER:=8;
signal response : STD_LOGIC_VECTOR(MAX_LENGTH_RESP- 1 downto 0):=(others=>'0');

--definire semnal pe 7 biti primit ca date de la card
signal byteVector: STD_LOGIC_VECTOR(7 downto 0):=(others=>'0');
--semnalul de date auxiliar din blocul de date citit
signal data_out: STD_LOGIC_VECTOR(512*8-1 downto 0):=(others=>'0');

--definire size-ul blocului de citit
--data token (1 byte) + Data (512 bytes) + CRC (2 bytes)
constant BLK_SIZE : INTEGER:= 515;

--definire semnal clk divizat
signal clk : STD_LOGIC:='0';

--definire semnal de ceas auxiliar pentru SCK
signal SCK_reg : std_logic:='0';

--semnal intern pentru semnalul de notificare CARD pregatit pentru comenzi
citire
signal ready_reg: STD_LOGIC:='0';

begin

state_proces: process(clk)
    variable bitsCounter      : INTEGER:=0;
    variable bytesCounter     : INTEGER:=0;
    variable returnData       : std_logic:='0';
begin
    if enable = '1' then
        if rising_edge(clk) then
            if reset= '1' then
                state <= START;
                ready_reg <= '0';
            else
                case state is

                    when START =>
                        SCK_reg <= '0';
                        finish_read_blk <= '0';
                        CS_reg <='1';
                        state <= WAIT_STATE;
                        ready_reg <= '1';

                    when WAIT_STATE =>
                        if read_enable = '1' then
                            state <= SEND_CMD17;
                            finish_read_blk <='0';
                        end if;

                    when SEND_CMD17  =>
                        CS_reg <= '0';
                        SCK_reg <= '0';

                end case;
            end if;
        end if;
    end if;
end process;

```



```
bitsCounter := CMD_LENGTH-1;
currentCMD <= CMD17(CMD_LENGTH-1 downto 40) & address & x"FF";
state <= TRANSFER_CMD;
return_state <= RESPONSE_CMD17;

when RESPONSE_CMD17 =>
  if response(2)='1' then
    state <= ILLEGAL;
  else
    state <= READ_BLOCK;
    bytesCounter := BLK_SIZE;

  end if;

when READ_BLOCK =>
  CS_reg <= '0';
  returnData := '0';
  bitsCounter := 7;
  state <= GET_BYTE;
  return_state <= READ_BLOCK;

  if bytesCounter = BLK_SIZE then
    bytesCounter := bytesCounter - 1;
  elsif bytesCounter = BLK_size - 1 then
    if byteVector = x"FF" then
      null;
    elsif byteVector = x"FE" then
      returnData := '1';
      bitsCounter := 512 * 8 -1;
      bytesCounter := 2;
    else
      state <= ILLEGAL;
    end if;
  elsif bytesCounter = 2 then
    bytesCounter := bytesCounter - 1;
  elsif bytesCounter = 1 then
    bytesCounter := bytesCounter - 1;
  else
    SCK_reg <='0';
    CS_reg <='1';
    state <= WAIT_STATE;
    finish_read_blk <='1';
  end if;

when GET_BYTE =>
  if SCK_reg ='1' then
    byteVector <= byteVector(6 downto 0) & MISO;
    if returnData = '1' then
      data_out <= data_out(512*8-2 downto 0) &
MISO;
    end if;
    if bitsCounter = 0 then
      state <= return_state;
      if returnData = '1' then
        data_out <= data_out(512*8-2 downto 0) &
MISO;
      end if;
    else
      bitsCounter := bitsCounter - 1;
    end if;
  end if;
```



```
sck_reg <= not SCK_reg;

when TRANSFER_CMD =>
    if SCK_reg = '1' then
        if bitsCounter = 0 then
            state <= GET_RESPONSE;
        else
            bitscounter := bitscounter -1;
            currentCMD <= currentCMD(CMD_LENGTH-2
downto 0) & '1';
        end if;
    end if;
    SCK_reg <= not SCK_reg;

when GET_RESPONSE =>
    if SCK_reg ='1' then
        if MISO = '0' then
            response <= response(MAX_LENGTH_RESP-2
downto 0) & MISO;
            bitsCounter := 6;           --raspuns de tip R1
            state <= CONTINUE_RESPONSE;
        end if;
    end if;
    SCK_reg <= not SCK_reg;

when CONTINUE_RESPONSE =>
    if SCK_reg ='1' then
        response <= response(MAX_LENGTH_RESP-2 downto 0)
& MISO;
        if bitsCounter = 0 then
            state <= return_state;
        else
            bitsCounter := bitsCounter - 1;
        end if;
    end if;
    SCK_reg <= not SCK_reg;

when ILLEGAL => illegal <='1';
when others => state<=WAIT_STATE;
end case;
end if;
end if;
end if;
end process;

--asignarea semnalelor interne
ready      <= ready_reg;
dataOut    <= data_out;
in_work    <= '0' when state = WAIT_STATE or state = ILLEGAL or state = START
else '1';

--semnale pentru conectorul microSD
MOSI <= currentCMD(CMD_LENGTH-1);
CS   <= CS_reg;
SCK  <= SCK_reg;

---instantiere divizor de ceas de 50 MHZ _dublu fata de SCK-----
DivizorFrecventa_SD_inst : entity work.DivizorFrecventa_SD
generic map (
    fout => 50.0  --divizor de frecventa 50 MHz=> SCK va fi de 25 MHz
```



```
)  
port map (  
    clk100Mhz => clk100Mhz,  
    clkDivised => clk  
);  
  
end Behavioral;
```

ANEXA 10 Modulul ModulMicroSD

Codul pentru ModulMicroSD.vhd

```
-----
-- Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Modul micro SD
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity ModulMicroSD is
  Port ( clk100MHz : in STD_LOGIC;
         clk25MHz : in STD_LOGIC;
         reset :      in STD_LOGIC;
         enable :     in STD_LOGIC;
         btnl:        in STD_LOGIC;
         btnr:        in STD_LOGIC;
         --semnale card
         SD_RESET:out std_logic;
         SD_CD:   in std_logic;
         SD_SCK:  inout std_logic;
         SD_CMD:  inout std_logic;
         SD_DAT:  inout std_logic_vector(3 downto 0);

         --semnale pentru afisor
         an:   out std_logic_vector(7 downto 0);
         cat:  out std_logic_vector(7 downto 0);

         --semnal ce indica tipul microSD-ului; 00 sau 10 -EROARE , 01-SDHC,
11-Standard
         tipSD:    out STD_LOGIC_VECTOR(1 downto 0);
         error:    out STD_LOGIC:='0';
         waitRead: out STD_LOGIC:='0';

         finish:   : out STD_LOGIC:='0';
         inWork:   : out STD_LOGIC:='0';

         --semnale acces memorie
         addrx :   in NATURAL:=0;
         addry :   in NATURAL:=0;
         data_out : out STD_LOGIC_VECTOR (11 downto 0):=(others=>'0')
      );
end ModulMicroSD;

architecture Behavioral of ModulMicroSD is

  --semnal pentru activarea fsm-ului de initializare card
  signal enableSDInit : STD_LOGIC:='0';
  signal CS_init : STD_LOGIC:='1';
  signal MOSI_init : STD_LOGIC:='1';
  signal MISO_init : STD_LOGIC:='1';
  signal SCK_init : STD_LOGIC:='0';
  signal tipSD_reg: STD_LOGIC_VECTOR(1 downto 0):="00";
  signal finishInit: STD_LOGIC:='0';

```



```
--semnal pentru activarea fsm-ului de citire card
signal enableSDRead : STD_LOGIC:='0';
signal CS_read : STD_LOGIC:='1';
signal MOSI_read : STD_LOGIC:='1';
signal MISO_read : STD_LOGIC:='1';
signal SCK_read : STD_LOGIC:='0';
signal readAddress: STD_LOGIC_VECTOR(31 downto 0):=(others=>'0');
--semnale de notificare
signal read_enable: STD_LOGIC:='0';
signal sd_in_work: STD_LOGIC:='0';
signal finish_read_blk : STD_LOGIC:='0';
signal readySDRead : STD_LOGIC:='0';
--declarare semnal pentru octetul primit de la sd
signal data_from_sd : STD_LOGIC_VECTOR(512*8 - 1 downto 0):=(others=>'0');

--siignal datatodisplay
signal dataToDisplay : STD_LOGIC_VECTOR(31 downto 0):= (others=>'0');

--definire stari pentru host
type HOST_STATE is (START, INIT_CARD, START_SD_READ , WAIT_READ, PRE-
PARE_READ, READ_DATA, WRITE_IN_MEMORY, WAIT_WRITE, ERROR_STATE);
signal state: HOST_STATE:=START;

--semnal pentru prima imagine
signal initial_image : STD_LOGIC:='1';

-----
signal w_addrx : NATURAL:=0;
signal w_addr y : NATURAL:=0;
signal RW : STD_LOGIC:='0'; --semnal de enable scriere
signal data_in : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal illegal : std_logic :='0';

--image adress
signal imgAddress: STD_LOGIC_VECTOR(11 downto 0):=x"000";

begin

-----FSM ITINITIALIZARE CARD-----
init_SD_modul: entity work.SDinitialise PORT MAP(
    clk100MHz => clk100MHz,
    reset => '0',
    enable => enableSDInit,
    SCK => SCK_init,
    CS => CS_init,
    MOSI => MOSI_init,
    MISO => MISO_init,
    tipSd => tipSd_reg,
    finish => finishInit
);

-----FSM CITIRE CARD-----
read_SD_modul: entity work.SDControllerRead PORT MAP(
    clk100MHz => clk100MHz,
    reset => '0',
    enable => enableSDREAD,
    address => readAddress,
```



```
    read_enable => read_enable,           -- validarea citirii de la host
    in_work      => sd_in_work,          -- semnal ce specifica faptul ca
microSD face operatii
    --semnale pentru conectorul sd
    SCK          => SCK_read,
    MOSI         => MOSI_read,
    MISO         => MISO_read,
    CS           => CS_read,

    --semnale ce indica starea autmotuui de citire
    finish_read_blk => finish_read_blk,
    dataOut          => data_from_sd,
    ilegal           => ilegal,
    ready            => readySDRead
);
----- IMAGE COUNTER-----
----- Image_counter: entity work.ImageCounter generic map (nbImg => 10)
port map (clk =>clk100MHz,
           reset =>reset,
           enable => enable,
           btnl =>btnl,
           btnr => btnr,
           imgAddr=> imgAddress);

----- SEMNALE CONNECTO MICRO SD-----
----- SD_SCK <= SCK_init when enableSDInit = '1' and enableSDRead = '0' else
SCK_read when enableSDRead = '1' else '0';
    SD_DAT(3) <= CS_init when enableSDInit = '1' and enableSDRead = '0' else
CS_read when enableSDRead = '1' else '1';
    SD_CMD <= MOSI_init when enableSDInit = '1' and enableSDRead = '0' else
MOSI_read when enableSDRead = '1' else '1';
    MISO_init <= SD_DAT(0);
    MISO_read <= SD_DAT(0);
    SD_RESET <= '0';

-----AUTOMATUL DE STARE AL HOSTULUI-----
----- state_machine_host: process(clk25MHz)
variable contorByte: INTEGER:=0;      --contor bytes prelucrati
variable waiting     : INTEGER:=4;    --contor pentru timpul de asteptare
begin
    if enable = '1' then           --doar cand semnalul de enable e
pornit se va realiza procesul hostului
        if rising_edge(clk25MHz) then
            if reset = '1' and state /=ERROR_STATE then
                RW <='0';
                enableSDRead <= '1';
                enableSDInit <= '0'; --sa nu se mai reseteaze cardul
                initial_image <= '1';
                state <= START_SD_READ;

            else
                case state is
                    when START =>
                        RW <='0';
                        enableSDInit <= '1';
                        state <= INIT_CARD;

                    when INIT_CARD =>
                        if finishInit = '1' then
```



```
enableSDInit <='0';
if tipSd_reg = "10" then
    state <= ERROR_STATE;
elsif tipSd_reg /= "00" then
    enableSDRead <= '1';
    state <= START_SD_READ;
end if;
end if;

when START_SD_READ =>
if readySDRead ='1' then
    state <= WAIT_READ;
end if;

when WAIT_READ =>
read_enable <= '0';
w_addrx <= 0;
w_addrx <= 0;
if initial_image = '1' then
    state <= PREPARE_READ;
    initial_image <= '0';
elsif btnl = '1' or btnr='1' then
    state <=PREPARE_READ;
end if;

when PREPARE_READ =>
read_enable<='1';
readAddress <= x"00000" & imgAddress;
state <= READ_DATA;

when READ_DATA =>
if sd_in_work ='1' then
    read_enable <='0';
end if;
if finish_read_blk = '1' then
    state <= WAIT_WRITE;
    waiting :=10;
    contorByte :=511;
    RW<='1'; --se activeaza citirea
    data_in <= data_from_sd(8*(contorByte+1)-1-4
downto 8*contorByte);
end if;

when WRITE_IN_MEMORY =>
state <= WAIT_WRITE;
w_addrx <= (w_addrx + 1) mod 480;
if w_addrx = 479 and w_addrx < 119 then
    w_addrx <= w_addrx + 1;

elsif w_addrx = 479 and w_addrx = 119 then
    RW <= '0';
    state <= WAIT_READ;
end if;

contorByte := contorByte - 1;
if contorByte = -1 then
    RW<='0';
    read_enable<='1';
    readAddress <= readAddress + 1;
    state <= READ_DATA;
else
```



```
        data_in <= data_from_sd(8*(contorByte+1)-1-4
downto 8*contorByte);
                end if;

        when WAIT_WRITE =>
            if waiting = 0 then
                waiting :=10;
                state <= WRITE_IN_MEMORY;
            else
                waiting := waiting- 1;
            end if;

        when ERROR_STATE =>
            error <='1';
        when others =>
            state <= START;
        end case;
    end if;
end if;
end process;

-- -----COUNTER IMAGE ADDRESS-----
-- -----
-- counter_adresa: process(clk100MHz)
begin
    if rising_edge(clk100MHz) then
        if state = WAIT_READ then
            if btnr_db = '1' then
                readAddress <= readAddress + 1;
            elsif btnl_db = '1' then
                readAddress <= readAddress - 112;
            end if;
        end if;
    end if;
end process;
-- -----INSTANTIARE MEMORIE VRAM-----
-- -----
VRAM_MEMORY: entity work.RAM
port MAP( clk => clk25MHz,
          r_addrx => addrx,
          r_addrv => addrv,
          w_addrx => w_addrx,
          w_addrv => w_addrv,
          RW      => RW,
          data_in => data_in,
          data_out => data_out);

--     dataToDisplay <= data_from_sd(512*8-1 downto 512*8-32);

--     Diplay7seg: entity work.displ7seg PORT MAP(
--         clk  => clk100MHz,
--         rst  =>reset,
--         data => dataToDisplay,
--         an   =>an,
--         seg   =>cat
--     );

-----semnale iesire led-uri-----
tipSD <= tipSd_reg;                                --tip card
```



```
finish <= finish_read_blk;      -- terminare operatie de citit card
inWork <= sd_in_work;          --cardul este prin in procesul de initializare
sau citire
  waitRead <= '1' when state =WAIT_READ else '0'; --starea de asteptare a unei
comenzi

end Behavioral;
```

ANEXA 11 Modulul RAM

Codul pentru RAM.vhd

```
-----
-- Birlutiu Claudiu-Andrei
-- UTCN CTI-ro
-- Modulul ram pentru stocarea configuratiei imaginii de citit
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RAM is
    Port ( clk : in STD_LOGIC;
            r_addrx : in NATURAL;
            r_addr y : in NATURAL;
            w_addrx : in NATURAL;
            w_addr y : in NATURAL;
            RW : in STD_LOGIC; --semnal de enable scriere
            data_in : in STD_LOGIC_VECTOR (3 downto 0);
            data_out : out STD_LOGIC_VECTOR (11 downto 0));
end RAM;

architecture Behavioral of RAM is

type TYPE_RAM is array(0 to 119) of STD_LOGIC_VECTOR(160*12-1 downto 0);

signal ram_memory : TYPE_RAM:=(others=>(others=>'0'));
begin

ram_process: process(clk)
begin
    if rising_edge(clk) then
        if RW = '1' then
            ram_memory(w_addrx)(4*(w_addrx+1)-1 downto 4*(w_addrx+1)-4) <=
data_in;

        end if;
    end if;
end process;
data_out <= ram_memory(r_addrx)(12*(r_addrx+1)-1 downto 12*(r_addrx+1)-12);

end Behavioral;
```

Codul pentru RAM_tb.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RAM_tb is
end;

architecture bench of RAM_tb is

component RAM
port (
clk : in STD_LOGIC;
r_addrx : in NATURAL;
r_addr : in NATURAL;
w_addrx : in NATURAL;
w_addr : in NATURAL;
RW : in STD_LOGIC;
data_in : in STD_LOGIC_VECTOR (3 downto 0);
data_out : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

-- Clock period
constant clk_period : time := 10 ns;
-- Generics

-- Ports
signal clk : STD_LOGIC:='0';
signal r_addrx : NATURAL:=0;
signal r_addr : NATURAL:=0;
signal w_addrx : NATURAL:=0;
signal w_addr : NATURAL:=0;
signal RW : STD_LOGIC:='0';
signal data_in : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal data_out : STD_LOGIC_VECTOR (11 downto 0);

begin

RAM_inst : RAM
port map (
clk => clk,
r_addrx => r_addrx,
r_addr => r_addr,
w_addrx => w_addrx,
w_addr => w_addr,
RW => RW,
data_in => data_in,
data_out => data_out
);

clk_process : process
begin
clk <= '1';
wait for clk_period/2;
clk <= '0';
wait for clk_period/2;
end process clk_process;

process
begin

```



```
    wait for 10 ns;
    RW<='1';
    data_in<="1010";
    w_addrx<=1;
    wait for 10 ns;
    RW<='0';
    data_in<="1111";
    wait for 10 ns;
    r_addrx <= 0;
    r_addrw <=0;
    wait for 10 ns;
    wait;

end process;

end;
```

ANEXA 12 Modulul ImageCounter

Codul pentru ImageCounter.vhd

```
--Birlutiu Claudiu-Andrei
--UTCN CTI-ro
-----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity ImageCounter is
    generic ( nbImg: NATURAL:=10);
    Port ( clk :      in STD_LOGIC;
            reset :     in STD_LOGIC;
            enable :    in STD_LOGIC;
            btnl :      in STD_LOGIC;
            btrn :      in STD_LOGIC;
            imgAddr :   out STD_LOGIC_VECTOR (11 downto 0)
        );
end ImageCounter;

architecture Behavioral of ImageCounter is
    --definire memorie rom cu partea mai putin seminifcativa a adreselor
    --blocurilor
    --de inceput ale imaginilor
    type TYPE_ROM is array(0 to nbImg-1) of STD_LOGIC_VECTOR(11 downto 0);
    constant IMG_ROM : TYPE_ROM:=(x"00f",
    x"080",
    x"0f1",
    x"162",
    x"1d3",
    x"244",
    x"2b5",
    x"326",
    x"397",
    x"408",
    others=>x"000");
begin
    process(clk)
        variable imgCnt      : INTEGER range 0 to 9:=0;
    begin
        if rising_edge(clk) then
            if enable='1' then
                if reset = '1' then
                    imgCnt := 0;
                elsif btrn ='1' then
                    if imgCnt = nbImg-1 then
                        imgCnt :=0;
                    else
                        imgCnt := imgCnt +1;
                    end if;
                elsif btnl = '1' then
                    if imgCnt = 0 then
                        imgCnt :=nbImg-1;
                    else
                        imgCnt := imgCnt -1;
                    end if;
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```



```
        end if;
    end if;
    imgAddr <= IMG_ROM(imgCnt);
end if;
end process;

end architecture Behavioral;
```

Codul pentru ImageCounter_tb.vhd

```
-----
--Birlutiu Claudiu-Andrei
--UTCN CTI-ro
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ImageCounter_tb is
end;

architecture bench of ImageCounter_tb is

component ImageCounter
generic (
    nbImg : NATURAL
);
port (
    clk : in STD_LOGIC;
    reset : in STD_LOGIC;
    enable : in STD_LOGIC;
    btnl : in STD_LOGIC;
    btnr : in STD_LOGIC;
    imgAddr : out STD_LOGIC_VECTOR (11 downto 0)
);
end component;

-- Clock period
constant clk_period : time := 10 ns;
-- Generics
constant nbImg : NATURAL := 10;

-- Ports
signal clk : STD_LOGIC:='1';
signal reset : STD_LOGIC;
signal enable : STD_LOGIC;
signal btnl : STD_LOGIC;
signal btnr : STD_LOGIC;
signal imgAddr : STD_LOGIC_VECTOR (11 downto 0);

begin

ImageCounter_inst : ImageCounter
generic map (
    nbImg => nbImg
```



```
)  
port map (  
    clk => clk,  
    reset => reset,  
    enable => enable,  
    btnl => btnl,  
    btnr => btnr,  
    imgAddr => imgAddr  
) ;  
  
clk_process : process  
begin  
    clk <= '1';  
    wait for clk_period/2;  
    clk <= '0';  
    wait for clk_period/2;  
end process clk_process;  
  
generare_semnale: process  
begin  
    reset<='0';  
    enable<='1';  
    wait for clk_period;  
    btnl <='1';  
    wait for clk_period;  
    btnl<='0';  
    btnr<='1';  
    wait for clk_period;  
    btnr<='0';  
    wait;  
  
end process;  
  
end;
```

ANEXA 13 Modulul Effect_generator

Codul pentru Effect_generator.vhd

```

--Birlutiu Claudiu -Andrei
-- UTCN CTI-ro
-- Proiect Controller VGA
-- Effect generator
-----
--



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Effect_generator is
    Port (
        effect_enable : in STD_LOGIC;
        sw: in STD_LOGIC_VECTOR(11 downto 0);
        grayscale : in STD_LOGIC;
        sepia : in STD_LOGIC;
        rgb : in STD_LOGIC_VECTOR (11 downto 0);
        rgb_filter : out STD_LOGIC_VECTOR (11 downto 0));
end Effect_generator;

architecture Behavioral of Effect_generator is
begin

process(effect_enable, grayscale, sepia, sw, rgb)
    variable red:      NATURAL;
    variable green:    NATURAL;
    variable blue:     NATURAL;
begin
    red  := conv_integer(rgb(3 downto 0));
    green := conv_integer(rgb(7 downto 4));
    blue := conv_integer(rgb(11 downto 8));
    if effect_enable = '1' then
        if grayscale = '1' then
            red := red/3;
            green := 2 *green/3;
            blue := 1 * blue/10;
        elsif sepia = '1' then
            red := red/3 + 3*green/4 + blue/10;
            green := red/3 + 2*green/3 + blue/10;
            blue := red/5 + green/2 + blue/10;
        else
            red := red + conv_integer(sw(11 downto 8));
            green:= green + conv_integer(sw(7 downto 4));
            blue := blue + conv_integer(sw(3 downto 0));
        end if;
    end if;
    rgb_filter <= conv_std_logic_vector(red,4) & conv_std_logic_vector(green,4) &conv_std_logic_vector(blue,4);
end process;

end Behavioral;

```



Codul pentru Effect_generator_tb.vhd

```
--  
-- Birlutiu Claudiu-Andrei  
-- UTCN CTI-ro  
-- Modul testare efect generator  
--  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
  
entity Effect_generator_tb is  
end;  
  
architecture bench of Effect_generator_tb is  
  
component Effect_generator  
port (  
    effect_enable : in STD_LOGIC;  
    sw : in STD_LOGIC_VECTOR(11 downto 0);  
    grayscale : in STD_LOGIC;  
    sepia : in STD_LOGIC;  
    rgb : in STD_LOGIC_VECTOR (11 downto 0);  
    rgb_filter : out STD_LOGIC_VECTOR (11 downto 0)  
);  
end component;  
  
-- Ports  
signal effect_enable : STD_LOGIC:='0';  
signal sw : STD_LOGIC_VECTOR(11 downto 0):=(others=>'0');  
signal grayscale : STD_LOGIC:='0';  
signal sepia : STD_LOGIC:='0';  
signal rgb : STD_LOGIC_VECTOR (11 downto 0):=(others=>'0');  
signal rgb_filter : STD_LOGIC_VECTOR (11 downto 0):=(others=>'0');  
  
begin  
  
Effect_generator_inst : Effect_generator  
port map (  
    effect_enable => effect_enable,  
    sw => sw,  
    grayscale => grayscale,  
    sepia => sepia,  
    rgb => rgb,  
    rgb_filter => rgb_filter  
);  
  
process  
begin  
    rgb<=x"acd";  
    wait for 10 ns;  
    effect_enable<='1';  
    sepia <='1';      --aplicare efect sepia  
    wait for 10 ns;  
    sepia<='0';  
    grayscale<='1';   --aplicare efect grayscale  
    wait for 10 ns;  
    grayscale<='0';   --aplicare efect de pe switch-uri  
    sw <= x"123";  
end process;
```



```
wait for 10 ns;
effect_enable <='0';
wait;

end process;
end;
```



ANEXA 14 Modulul ControllerVGA

Codul pentru ControllerVGA.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--aceasta componenta preia informatiile legate de culoare si sincronizare; elimina problemele eventuale de timing prin procedeul numit pipelineing
--SE ELIMINA intarzierile rezultante pe caile de date combinationale
entity controller_vga is
    Port ( clock_25MHz : in STD_LOGIC;
           hsync_clock : in STD_LOGIC;
           vsync_clock : in STD_LOGIC;
           enable_image : in STD_LOGIC;
           color : in STD_LOGIC_VECTOR (11 downto 0);
           red : out STD_LOGIC_VECTOR (3 downto 0);
           blue : out STD_LOGIC_VECTOR (3 downto 0);
           green : out STD_LOGIC_VECTOR (3 downto 0);
           hsync : out STD_LOGIC;
           vsync : out STD_LOGIC);
end controller_vga;

architecture Behavioral of controller_vga is
--se iau semnale interne sincronziare scanare ecran
signal h_sync_reg : std_logic := '1';
signal v_sync_reg : std_logic := '1';

--se iau semnale interne pentru vectorii de culori
signal vga_red_reg : std_logic_vector(3 downto 0);
signal vga_green_reg : std_logic_vector(3 downto 0);
signal vga_blue_reg : std_logic_vector(3 downto 0);

--semnalele in urma procesului de pipelining
signal img_red : std_logic_vector(3 downto 0);
signal img_green: std_logic_vector(3 downto 0);
signal img_blue : std_logic_vector(3 downto 0);

begin
    --se elimina intarzierile
    process (clock_25MHz)
    begin
        if (rising_edge(clock_25MHz)) then
            img_red    <= color(11 downto 8);
            img_green  <= color(7 downto 4);
            img_blue   <= color(3 downto 0);
            h_sync_reg <= hsync_clock;
            v_sync_reg <= vsync_clock;
        end if;
    end process;

    process (clock_25MHz )
    begin
        if (rising_edge(clock_25MHz )) then
            if enable_image = '1' then
                vga_red_reg  <= img_red;
                vga_blue_reg <= img_blue;
                vga_green_reg <= img_green;
            else

```



```
        vga_red_reg  <= x"0";           --in cazul in care suntem in
afara zonei de display, se pune pe iesiri 0->negru sau perioada de #blanking#
        vga_blue_reg <= x"0";
        vga_green_reg <= x"0";
    end if;
end if;
end process;

-- se fac asignarile corespunzatoare
hsync  <= h_sync_reg;
vsync  <= v_sync_reg;
red    <= vga_red_reg;
green  <= vga_green_reg;
blue   <= vga_blue_reg;

end Behavioral;
```

Codul pentru ControllerVGA_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity controler_vga_tb is
end;

architecture bench of controler_vga_tb is

component controler_vga
port (
clock_25MHz : in STD_LOGIC;
hsync_clock : in STD_LOGIC;
vsync_clock : in STD_LOGIC;
enable_image : in STD_LOGIC;
color : in STD_LOGIC_VECTOR (11 downto 0);
red : out STD_LOGIC_VECTOR (3 downto 0);
blue : out STD_LOGIC_VECTOR (3 downto 0);
green : out STD_LOGIC_VECTOR (3 downto 0);
hsync : out STD_LOGIC;
vsync : out STD_LOGIC
);
end component;

-- Clock period
constant clk_period : time := 5 ns;
-- Generics

-- Ports
signal clock_25MHz : STD_LOGIC:='1';
signal hsync_clock : STD_LOGIC:='0';
signal vsync_clock : STD_LOGIC:='0';
signal enable_image : STD_LOGIC:='0';
signal color : STD_LOGIC_VECTOR (11 downto 0):=(others=>'0');
signal red : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal blue : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal green : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal hsync : STD_LOGIC:='0';
```



```
signal vsync : STD_LOGIC:='0';

begin

controler_vga_inst : controler_vga
  port map (
    clock_25MHz => clock_25MHz,
    hsync_clock => hsync_clock,
    vsync_clock => vsync_clock,
    enable_image => enable_image,
    color => color,
    red => red,
    blue => blue,
    green => green,
    hsync => hsync,
    vsync => vsync
  );

clk_process : process
begin
clock_25MHz <= '1';
wait for clk_period/2;
clock_25MHz <= '0';
wait for clk_period/2;
end process clk_process;

generare_semnale: process
begin
  wait for clk_period;
  enable_image <='1';
  color <= x"af4";
  hsync_clock<='1';
  vsync_clock<='0';
  wait for clk_period;
  color <= x"af4";
  hsync_clock<='1';
  vsync_clock<='1';
  wait for clk_period;
  color <= x"af4";
  hsync_clock<='0';
  vsync_clock<='1';
  wait for clk_period;
  wait;
end process;

end;
```

ANEXA 15 Modulul ExecutionUnit

Codul pentru ExecutionUnit.vhd

```

-- 
-- Birlutiu Claudiu-Andrei
-- UTCN CTI
-- Unitate de executie
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ExecutionUnit is
  PORT(
    clk100MHz : in STD_LOGIC;
    reset :      in STD_LOGIC;
    sw : in STD_LOGIC_VECTOR(11 downto 0);
    move_enable:      in STD_LOGIC;
    change_img_enable : in STD_LOGIC;
    effect_enable:      in STD_LOGIC;
    btnl:      in STD_LOGIC;
    btnr:      in STD_LOGIC;
    btnu:      in STD_LOGIC;
    btnd:      in STD_LOGIC;
    --switch-uri
    sw_1:      in STD_LOGIC;
    sw_2:      in STD_LOGIC;
    --semnale card
    SD_RESET:out std_logic;
    SD_CD:   in std_logic;
    SD_SCK:  inout std_logic;
    SD_CMD:  inout std_logic;
    SD_DAT:  inout std_logic_vector(3 downto 0);

    --semnale pentru afisor
    an:  out std_logic_vector(7 downto 0);
    cat: out std_logic_vector(7 downto 0);

    --semnal ce indica tipul microSD-ului; 00 sau 10 -EROARE , 01-SDHC,
11-Standard
    tipSD:      out STD_LOGIC_VECTOR(1 downto 0);
    error:      out STD_LOGIC:='0';
    waitRead:  out STD_LOGIC:='0';

    finish     : out STD_LOGIC:='0';
    inWork     : out STD_LOGIC:='0';

    --semnale culori
    red   : out STD_LOGIC_VECTOR(3 downto 0);
    blue  : out STD_LOGIC_VECTOR(3 downto 0);
    green : out STD_LOGIC_VECTOR(3 downto 0);
    hsync : out STD_LOGIC;
    vsync : out STD_LOGIC);

end ExecutionUnit;

```



```
architecture Behavioral of ExecutionUnit is

    signal clk25MHz      : STD_LOGIC:='0';
    signal addr_x        : STD_LOGIC_VECTOR(9 downto 0);
    signal addr_y        : STD_LOGIC_VECTOR(9 downto 0);

    signal r_addrx       : NATURAL:=0;
    signal r_addr_y       : NATURAL:=0;
    signal enable_image   : STD_LOGIC:='0';

    signal rgb_filter    : STD_LOGIC_VECTOR(11 downto 0);
    signal rgb            : STD_LOGIC_VECTOR(11 downto 0);
    signal h_sync_reg     : STD_LOGIC:='0';
    signal v_sync_reg     : STD_LOGIC:='0';

    --semnale interne pentru offset
    signal offset_x       : INTEGER:=0;
    signal offset_y       : INTEGER:=0;

    --semnal debounce btl si btnc
    signal btl_db          : STD_LOGIC:='0';
    signal btr_db          : STD_LOGIC:='0';
    signal bts_db          : STD_LOGIC:='0';
    signal btd_db          : STD_LOGIC:='0';

begin

    MicroSDModul: entity work.ModulMicroSd PORT MAP(
        clk100MHz => clk100MHz,
        clk25MHz    => clk25MHz,
        reset       => reset,
        enable      => change_img_enable,
        btr         => btr_db,
        btl         => btl_db,
        --semnale card
        SD_RESET    =>SD_RESET,
        SD_CD       =>SD_CD,
        SD_SCK      =>SD_SCK,
        SD_CMD      =>SD_CMD,
        SD_DAT      =>SD_DAT,
        --semnale pentru afisor
        an           =>an,
        cat          =>cat,
        --semnal ce indica tipul microSD-ului; 00 sau 10 -EROARE , 01-SDHC,
11-Standard
        tipSD        =>tipSD,
        error        =>error,
        waitRead    =>waitRead,
        finish       =>finish,
        inWork       =>inWork,
        addrx       => r_addrx,
        addr_y      => r_addr_y,
        data_out    =>rgb
    );

    SyncGenerator: entity work.sync_generator PORT MAP(
        clock_25MHz  => clk25MHz,
        addr_x       => addr_x,
```



```
addr_y      => addr_y,
hsync       => h_sync_reg,
vsync       => v_sync_reg,
enable_image => enable_image
);

DivizorFrecventa: entity work.DivizorFrecventa_25 PORT MAP(
    clock_100MHz => clk100MHz,
    clock_25MHz   => clk25MHz
);

MemoryAddress: entity work.memory_address PORT MAP (
    enable  => enable_image,
    offset_x => offset_x,
    offset_y => offset_y,
    addr_x  => addr_x,
    addr_y  => addr_y,
    x_mem_addr => r_addrx,
    y_mem_addr => r_addry
);

ControllerVGA: entity work.controler_VGA PORT MAP(
    clock_25MHz  => clk25MHz,
    hsync_clock  => h_sync_reg,
    vsync_clock  => v_sync_reg,
    enable_image =>enable_image,
    color        => rgb_filter,
    red          => red,
    blue         => blue,
    green        => green,
    hsync        => hsync,
    vsync        => vsync
);

OffsetCounter: entity work.offset_counter port map(
    clk100MHz     => clk100MHz,
    enable        => move_enable,
    rst           => reset,
    left          => btnl_db,
    right         => bt(nr_db,
    up            => bt(nu_db,
    down          => bt(nd_db,
    offset_x     => offset_x,
    offset_y     => offset_y);

EffectGenerator: entity work.Effect_generator port map(
    effect_enable => effect_enable,
    sw  => sw,
    grayscale => sw_1,
    sepia    => sw_2,
    rgb      =>rgb,
    rgb_filter=>rgb_filter
);
-----DEBOUNCER-----
DebouncerLeft: entity work.mpg PORT MAP (
    btn => btl,
    clk => clk100MHz,
    enable => btl_db
);
DebouncerRight: entity work.mpg PORT MAP (
    btn => bt(nr,
    clk => clk100MHz,
```



```
    enable => btnr_db
);
DebouncerUp: entity work.mpg PORT MAP (
    btn => btnu,
    clk => clk100MHz,
    enable => btnu_db
);
DebouncerDown: entity work.mpg PORT MAP (
    btn => btnd,
    clk => clk100MHz,
    enable => btnd_db
);
-----
end Behavioral;
```

ANEXA 16 Modulul BlackBox

Codul pentru BlackBox.vhd

```
-----
--Bitlutiu Claudiu-Andrei
--UTCN CTI -ro
--BlackBox Controller VGA cu preluare imagini de pe card
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BlackBox is
  port(
    clk100MHz : in STD_LOGIC;
    resetOperation : in STD_LOGIC; --reset
    resetUC: in STD_LOGIC;
    btnl: in STD_LOGIC;
    btnr: in STD_LOGIC;
    btnc: in STD_LOGIC;
    btnu: in STD_LOGIC;
    btnd: in STD_LOGIC;
    sw_1: in STD_LOGIC;
    sw_2: in STD_LOGIC;
    sw : in STD_LOGIC_VECTOR(11 downto 0);
    --semnale card
    SD_RESET:out std_logic;
    SD_CD: in std_logic;
    SD_SCK: inout std_logic;
    SD_CMD: inout std_logic;
    SD_DAT: inout std_logic_vector(3 downto 0);

    an : out STD_LOGIC_VECTOR(7 downto 0);
    cat: out STD_LOGIC_VECTOR(7 downto 0);
    --semnal ce indica tipul microSD-ului; 00 sau 10 -EROARE , 01-SDHC, 11-
Standard
    tipSD: out STD_LOGIC_VECTOR(1 downto 0);
    error: out STD_LOGIC:='0';
    waitRead: out STD_LOGIC:='0';

    --semnale leduri
    led_0 : out STD_LOGIC:='0'; --change image state
    led_1 : out STD_LOGIC:='0'; --move image state
    led_2 : out STD_LOGIC:='0'; --effect image state
    finish : out STD_LOGIC:='0';
    inWork : out STD_LOGIC:='0';

    --semnale culori
    red : out STD_LOGIC_VECTOR(3 downto 0);
    blue : out STD_LOGIC_VECTOR(3 downto 0);
    green : out STD_LOGIC_VECTOR(3 downto 0);
    hsync : out STD_LOGIC;
    vsync : out STD_LOGIC);
end BlackBox;
```



```
architecture Behavioral of BlackBox is
    signal move_enable :      STD_LOGIC:='0';
    signal effect_enable :    STD_LOGIC:='0';
    signal change_img_enable : STD_LOGIC:='0';
begin
    UC: entity work.ControlUnit port map (
        clk100MHz => clk100MHz,
        btnc => btnc,
        reset => resetUC,      --button reset
        move_enable => move_enable,
        effect_enable => effect_enable,
        change_img_enable => change_img_enable,
        led_0 => led_0,
        led_1 => led_1,
        led_2 => led_2
    );
    UE: entity work.ExecutionUnit port map(
        clk100MHz => clk100MHz,
        reset      => resetOperation,
        move_enable      => move_enable,
        effect_enable     => effect_enable,
        change_img_enable => change_img_enable,
        btnl      => btnl,
        btnr      => btnr,
        btnu      => btnu,
        btnd      => btnd,
        sw_1      => sw_1,
        sw_2      => sw_2,
        sw       => sw,
        --semnale card
        SD_RESET => SD_RESET,
        SD_CD    => SD_CD,
        SD_SCK   => SD_SCK,
        SD_CMD   => SD_CMD,
        SD_DAT   => SD_DAT,
        --semnale pentru afisor
        an       => an,
        cat      => cat,
        --semnal ce indica tipul microSD-ului; 00 sau 10 -EROARE , 01-SDHC, 11-
Standard
        tipSD      => tipSD,
        error      => error,
        waitRead   => waitRead,
        finish     => finish,
        inWork     => inWork,
        --semnale culori
        red       => red,
        blue      => blue,
        green     => green,
        hsync     => hsync,
        vsync     => vsync );
end Behavioral;
```



ANEXA 17 Pachet VGA

VgaPackage

```
--  
--Birlutiu Claudiu-Andrei  
-- UTCN CTI-ro  
-- pachet cu constante VGA  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
package VgaPackage is  
    --diagramele de timing detaliate se gasesc in documentatie  
    --am folosit constante pentru a fi usor modificarea rezolutiei ca metoda de  
    dezvoltare ulterioara a programului  
  
    --CONSTANTELE DE TIMING PENTRU 640/480  
    --PENTRU AXA ORIZONTALA  
    constant HD : integer := 640;      --zona de display pe orizontala  
    constant HF : integer := 16;       --front porch - regiunea inainte de retrace  
    constant HB : integer := 48;       --back porch - regiunea dupa retrage  
    constant HR : integer := 96;       --retrace (se intoarce la prima coloana )  
    constant H_max : integer := 800;    --reprezinta timpul total pentru orizontala  
->(scanarea pe orizontala)  
    --vertical  
    constant VD : integer := 480;     --display area  
    constant VF : integer := 10;       --front porch - regiunea inainte de retrage  
    constant VB : integer := 33;       --back porch - regiunea dupa retrage  
    constant VR : integer := 2;        --vertical retrage (se intoarce la prima linie)  
    constant V_max : integer := 525;   --reprezinta timpul total pentru verticala ->  
    scanarea pe verticala  
  
    --polaritate 0 la 640x480 @60Hz, la alte rezolutii aceasta e 1  
    constant H_pol : std_logic := '0';  
    constant V_pol : std_logic := '0';  
  
    --definirea starilor pentru automotul pentru sincronizare pe orizontala si verticala  
    type STATE_H is ( S_HD, S_HF, S_HR, S_HB);  
    type STATE_V is ( S_VD, S_VF, S_VR, S_VB);  
end package;
```



ANEXA 18 Fișierul de constrângeri

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in
##   the project

## Clock signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports
{ clk100MHz }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk100MHz}];

##Switches

set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { sw_1 }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { sw_2 }];
#IO_L3N_T0_DQS_EMCCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { sw[2] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { sw[3] }];
#IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { sw[4] }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { sw[5] }];
#IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { sw[6] }];
#IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { sw[7] }];
#IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { sw[8] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { sw[9] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { sw[10] }];
#IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { sw[11] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { resetUC }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports
{ resetOperation }];
#IO_L21P_T3_DQS_14 Sch=sw[15]
```



LEDs

```
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { led_0 }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { led_1 }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { led_2 }];
#IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { inWork }];
#IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports
{ LED[2] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports
{ LED[3] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports
{ LED[4] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports
{ LED[5] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports
{ LED[6] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { finish }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { inWork }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { error }];
#IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports
{ waitRead }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { tipSD[0] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { tipSD[1] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

#set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVCMOS33 } [get_ports
{ LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 } [get_ports
{ LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports
{ LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports
{ LED17_B }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11 IOSTANDARD LVCMOS33 } [get_ports
{ LED17_G }]; #IO_0_14 Sch=led17_g
#set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVCMOS33 } [get_ports
{ LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r
```

##7 segment display



```
set_property -dict { PACKAGE_PIN T10  IOSTANDARD LVCMOS33 } [get_ports { cat[0] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10  IOSTANDARD LVCMOS33 } [get_ports { cat[1] }];
#IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16  IOSTANDARD LVCMOS33 } [get_ports { cat[2] }];
#IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13  IOSTANDARD LVCMOS33 } [get_ports { cat[3] }];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15  IOSTANDARD LVCMOS33 } [get_ports { cat[4] }];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11  IOSTANDARD LVCMOS33 } [get_ports { cat[5] }];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18  IOSTANDARD LVCMOS33 } [get_ports { cat[6] }];
#IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15  IOSTANDARD LVCMOS33 } [get_ports { cat[7] }];
#IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17  IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18  IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9   IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14  IOSTANDARD LVCMOS33 } [get_ports { an[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14  IOSTANDARD LVCMOS33 } [get_ports { an[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14  IOSTANDARD LVCMOS33 } [get_ports { an[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2   IOSTANDARD LVCMOS33 } [get_ports { an[6] }];
#IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13  IOSTANDARD LVCMOS33 } [get_ports { an[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]
```

##Buttons

```
#set_property -dict { PACKAGE_PIN C12  IOSTANDARD LVCMOS33 } [get_ports
{ CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17  IOSTANDARD LVCMOS33 } [get_ports { btnc }];
#IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18  IOSTANDARD LVCMOS33 } [get_ports { btnu }];
#IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17  IOSTANDARD LVCMOS33 } [get_ports { btnl }];
#IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17  IOSTANDARD LVCMOS33 } [get_ports { btnr }];
#IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18  IOSTANDARD LVCMOS33 } [get_ports { btnd }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```



##Pmod Headers

##Pmod Header JA

```
#set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { JA[1] }];
#IO_L20N_T3_A19_15 Sch=ja[1]
#set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { JA[2] }];
#IO_L21N_T3_DQS_A18_15 Sch=ja[2]
#set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { JA[3] }];
#IO_L21P_T3_DQS_15 Sch=ja[3]
#set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { JA[4] }];
#IO_L18N_T2_A23_15 Sch=ja[4]
#set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { JA[7] }];
#IO_L16N_T2_A27_15 Sch=ja[7]
#set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { JA[8] }];
#IO_L16P_T2_A28_15 Sch=ja[8]
#set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports { JA[9] }];
#IO_L22N_T3_A16_15 Sch=ja[9]
#set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { JA[10] }];
#IO_L22P_T3_A17_15 Sch=ja[10]
```

##Pmod Header JB

```
#set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports { JB[1] }];
#IO_L1P_T0_AD0P_15 Sch=jb[1]
#set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports { JB[2] }];
#IO_L14N_T2_SRCC_15 Sch=jb[2]
#set_property -dict { PACKAGE_PIN G16 IOSTANDARD LVCMOS33 } [get_ports { JB[3] }];
#IO_L13N_T2_MRCC_15 Sch=jb[3]
#set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { JB[4] }];
#IO_L15P_T2_DQS_15 Sch=jb[4]
#set_property -dict { PACKAGE_PIN E16 IOSTANDARD LVCMOS33 } [get_ports { JB[7] }];
#IO_L11N_T1_SRCC_15 Sch=jb[7]
#set_property -dict { PACKAGE_PIN F13 IOSTANDARD LVCMOS33 } [get_ports { JB[8] }];
#IO_L5P_T0_AD9P_15 Sch=jb[8]
#set_property -dict { PACKAGE_PIN G13 IOSTANDARD LVCMOS33 } [get_ports { JB[9] }];
#IO_O_15 Sch=jb[9]
#set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { JB[10] }];
#IO_L13P_T2_MRCC_15 Sch=jb[10]
```

##Pmod Header JC

```
#set_property -dict { PACKAGE_PIN K1 IOSTANDARD LVCMOS33 } [get_ports { JC[1] }];
#IO_L23N_T3_35 Sch=jc[1]
#set_property -dict { PACKAGE_PIN F6 IOSTANDARD LVCMOS33 } [get_ports { JC[2] }];
#IO_L19N_T3_VREF_35 Sch=jc[2]
#set_property -dict { PACKAGE_PIN J2 IOSTANDARD LVCMOS33 } [get_ports { JC[3] }];
#IO_L22N_T3_35 Sch=jc[3]
```



```
#set_property -dict { PACKAGE_PIN G6   IOSTANDARD LVCMOS33 } [get_ports { JC[4] }];
#IO_L19P_T3_35 Sch=jc[4]
#set_property -dict { PACKAGE_PIN E7   IOSTANDARD LVCMOS33 } [get_ports { JC[7] }];
#IO_L6P_T0_35 Sch=jc[7]
#set_property -dict { PACKAGE_PIN J3   IOSTANDARD LVCMOS33 } [get_ports { JC[8] }];
#IO_L22P_T3_35 Sch=jc[8]
#set_property -dict { PACKAGE_PIN J4   IOSTANDARD LVCMOS33 } [get_ports { JC[9] }];
#IO_L21P_T3_DQS_35 Sch=jc[9]
#set_property -dict { PACKAGE_PIN E6   IOSTANDARD LVCMOS33 } [get_ports { JC[10] }];
#IO_L5P_T0_AD13P_35 Sch=jc[10]
```

##Pmod Header JD

```
#set_property -dict { PACKAGE_PIN H4   IOSTANDARD LVCMOS33 } [get_ports { JD[1] }];
#IO_L21N_T3_DQS_35 Sch=jd[1]
#set_property -dict { PACKAGE_PIN H1   IOSTANDARD LVCMOS33 } [get_ports { JD[2] }];
#IO_L17P_T2_35 Sch=jd[2]
#set_property -dict { PACKAGE_PIN G1   IOSTANDARD LVCMOS33 } [get_ports { JD[3] }];
#IO_L17N_T2_35 Sch=jd[3]
#set_property -dict { PACKAGE_PIN G3   IOSTANDARD LVCMOS33 } [get_ports { JD[4] }];
#IO_L20N_T3_35 Sch=jd[4]
#set_property -dict { PACKAGE_PIN H2   IOSTANDARD LVCMOS33 } [get_ports { JD[7] }];
#IO_L15P_T2_DQS_35 Sch=jd[7]
#set_property -dict { PACKAGE_PIN G4   IOSTANDARD LVCMOS33 } [get_ports { JD[8] }];
#IO_L20P_T3_35 Sch=jd[8]
#set_property -dict { PACKAGE_PIN G2   IOSTANDARD LVCMOS33 } [get_ports { JD[9] }];
#IO_L15N_T2_DQS_35 Sch=jd[9]
#set_property -dict { PACKAGE_PIN F3   IOSTANDARD LVCMOS33 } [get_ports { JD[10] }];
#IO_L13N_T2_MRCC_35 Sch=jd[10]
```

##Pmod Header JXADC

```
#set_property -dict { PACKAGE_PIN A14  IOSTANDARD LVDS   } [get_ports { XA_N[1] }];
#IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
#set_property -dict { PACKAGE_PIN A13  IOSTANDARD LVDS   } [get_ports { XA_P[1] }];
#IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
#set_property -dict { PACKAGE_PIN A16  IOSTANDARD LVDS   } [get_ports { XA_N[2] }];
#IO_L8N_T1_AD10N_15 Sch=xa_n[2]
#set_property -dict { PACKAGE_PIN A15  IOSTANDARD LVDS   } [get_ports { XA_P[2] }];
#IO_L8P_T1_AD10P_15 Sch=xa_p[2]
#set_property -dict { PACKAGE_PIN B17  IOSTANDARD LVDS   } [get_ports { XA_N[3] }];
#IO_L7N_T1_AD2N_15 Sch=xa_n[3]
#set_property -dict { PACKAGE_PIN B16  IOSTANDARD LVDS   } [get_ports { XA_P[3] }];
#IO_L7P_T1_AD2P_15 Sch=xa_p[3]
#set_property -dict { PACKAGE_PIN A18  IOSTANDARD LVDS   } [get_ports { XA_N[4] }];
#IO_L10N_T1_AD11N_15 Sch=xa_n[4]
#set_property -dict { PACKAGE_PIN B18  IOSTANDARD LVDS   } [get_ports { XA_P[4] }];
#IO_L10P_T1_AD11P_15 Sch=xa_p[4]
```



##VGA Connector

```
set_property -dict { PACKAGE_PIN A3 IOSTANDARD LVCMOS33 } [get_ports { red[0] }];
#IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4 IOSTANDARD LVCMOS33 } [get_ports { red[1] }];
#IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5 IOSTANDARD LVCMOS33 } [get_ports { red[2] }];
#IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4 IOSTANDARD LVCMOS33 } [get_ports { red[3] }];
#IO_L8P_T1_AD14P_35 Sch=vga_r[3]

set_property -dict { PACKAGE_PIN C6 IOSTANDARD LVCMOS33 } [get_ports { green[0] }];
#IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5 IOSTANDARD LVCMOS33 } [get_ports { green[1] }];
#IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6 IOSTANDARD LVCMOS33 } [get_ports { green[2] }];
#IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6 IOSTANDARD LVCMOS33 } [get_ports { green[3] }];
#IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]

set_property -dict { PACKAGE_PIN B7 IOSTANDARD LVCMOS33 } [get_ports { blue[0] }];
#IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7 IOSTANDARD LVCMOS33 } [get_ports { blue[1] }];
#IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7 IOSTANDARD LVCMOS33 } [get_ports { blue[2] }];
#IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8 IOSTANDARD LVCMOS33 } [get_ports { blue[3] }];
#IO_L4P_T0_35 Sch=vga_b[3]

set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports { hsync }];
#IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12 IOSTANDARD LVCMOS33 } [get_ports { vsync }];
#IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

##Micro SD Connector

```
set_property -dict { PACKAGE_PIN E2 IOSTANDARD LVCMOS33 } [get_ports
{ SD_RESET }];
#IO_L14P_T2_SRCC_35 Sch=sd_reset
set_property -dict { PACKAGE_PIN A1 IOSTANDARD LVCMOS33 } [get_ports { SD_CD }];
#IO_L9N_T1_DQS_AD7N_35 Sch=sd_cd
set_property -dict { PACKAGE_PIN B1 IOSTANDARD LVCMOS33 } [get_ports { SD_SCK }];
#IO_L9P_T1_DQS_AD7P_35 Sch=sd_sck
set_property -dict { PACKAGE_PIN C1 IOSTANDARD LVCMOS33 } [get_ports
{ SD_CMD }];
#IO_L16N_T2_35 Sch=sd_cmd
set_property -dict { PACKAGE_PIN C2 IOSTANDARD LVCMOS33 } [get_ports
{ SD_DAT[0] }];
#IO_L16P_T2_35 Sch=sd_dat[0]
set_property -dict { PACKAGE_PIN E1 IOSTANDARD LVCMOS33 } [get_ports
{ SD_DAT[1] }];
#IO_L18N_T2_35 Sch=sd_dat[1]
set_property -dict { PACKAGE_PIN F1 IOSTANDARD LVCMOS33 } [get_ports
{ SD_DAT[2] }];
#IO_L18P_T2_35 Sch=sd_dat[2]
```



```
set_property -dict { PACKAGE_PIN D2  IOSTANDARD LVCMOS33 } [get_ports  
{ SD_DAT[3] }]; #IO_L14N_T2_SRCC_35 Sch=sd_dat[3]
```

##Accelerometer

```
#set_property -dict { PACKAGE_PIN E15  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_MISO }]; #IO_L11P_T1_SRCC_15 Sch=acl_miso  
#set_property -dict { PACKAGE_PIN F14  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_MOSI }]; #IO_L5N_T0_AD9N_15 Sch=acl_mosi  
#set_property -dict { PACKAGE_PIN F15  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_SCLK }]; #IO_L14P_T2_SRCC_15 Sch=acl_sclk  
#set_property -dict { PACKAGE_PIN D15  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_CSN }]; #IO_L12P_T1_MRCC_15 Sch=acl_csn  
#set_property -dict { PACKAGE_PIN B13  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_INT[1] }]; #IO_L2P_T0_AD8P_15 Sch=acl_int[1]  
#set_property -dict { PACKAGE_PIN C16  IOSTANDARD LVCMOS33 } [get_ports  
{ ACL_INT[2] }]; #IO_L20P_T3_A20_15 Sch=acl_int[2]
```

##Temperature Sensor

```
#set_property -dict { PACKAGE_PIN C14  IOSTANDARD LVCMOS33 } [get_ports  
{ TMP_SCL }]; #IO_L1N_T0_AD0N_15 Sch=tmp_scl  
#set_property -dict { PACKAGE_PIN C15  IOSTANDARD LVCMOS33 } [get_ports  
{ TMP_SDA }]; #IO_L12N_T1_MRCC_15 Sch=tmp_sda  
#set_property -dict { PACKAGE_PIN D13  IOSTANDARD LVCMOS33 } [get_ports  
{ TMP_INT }]; #IO_L6N_T0_VREF_15 Sch=tmp_int  
#set_property -dict { PACKAGE_PIN B14  IOSTANDARD LVCMOS33 } [get_ports  
{ TMP_CT }]; #IO_L2N_T0_AD8N_15 Sch=tmp_ct
```

##Omnidirectional Microphone

```
#set_property -dict { PACKAGE_PIN J5  IOSTANDARD LVCMOS33 } [get_ports { M_CLK }];  
#IO_25_35 Sch=m_clk  
#set_property -dict { PACKAGE_PIN H5  IOSTANDARD LVCMOS33 } [get_ports  
{ M_DATA }]; #IO_L24N_T3_35 Sch=m_data  
#set_property -dict { PACKAGE_PIN F5  IOSTANDARD LVCMOS33 } [get_ports  
{ M_LRSEL }]; #IO_0_35 Sch=m_lrsel
```

##PWM Audio Amplifier

```
#set_property -dict { PACKAGE_PIN A11  IOSTANDARD LVCMOS33 } [get_ports  
{ AUD_PWM }]; #IO_L4N_T0_15 Sch=aud_pwm  
#set_property -dict { PACKAGE_PIN D12  IOSTANDARD LVCMOS33 } [get_ports  
{ AUD_SD }]; #IO_L6P_T0_15 Sch=aud_sd
```

##USB-RS232 Interface



```
#set_property -dict { PACKAGE_PIN C4 IOSTANDARD LVCMOS33 } [get_ports  
{ UART_TXD_IN }]; #IO_L7P_T1_AD6P_35 Sch=uart_txd_in  
#set_property -dict { PACKAGE_PIN D4 IOSTANDARD LVCMOS33 } [get_ports  
{ UART_RXD_OUT }]; #IO_L11N_T1_SRCC_35 Sch=uart_rxd_out  
#set_property -dict { PACKAGE_PIN D3 IOSTANDARD LVCMOS33 } [get_ports  
{ UART_CTS }]; #IO_L12N_T1_MRCC_35 Sch=uart_cts  
#set_property -dict { PACKAGE_PIN E5 IOSTANDARD LVCMOS33 } [get_ports  
{ UART_RTS }]; #IO_L5N_T0_AD13N_35 Sch=uart_rts
```

##USB HID (PS/2)

```
#set_property -dict { PACKAGE_PIN F4 IOSTANDARD LVCMOS33 } [get_ports  
{ PS2_CLK }]; #IO_L13P_T2_MRCC_35 Sch=ps2_clk  
#set_property -dict { PACKAGE_PIN B2 IOSTANDARD LVCMOS33 } [get_ports  
{ PS2_DATA }]; #IO_L10N_T1_AD15N_35 Sch=ps2_data
```

##SMSC Ethernet PHY

```
#set_property -dict { PACKAGE_PIN C9 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_MDC }]; #IO_L11P_T1_SRCC_16 Sch=eth_mdc  
#set_property -dict { PACKAGE_PIN A9 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_MDIO }]; #IO_L14N_T2_SRCC_16 Sch=eth_mdio  
#set_property -dict { PACKAGE_PIN B3 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_RSTN }]; #IO_L10P_T1_AD15P_35 Sch=eth_rstn  
#set_property -dict { PACKAGE_PIN D9 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_CRSDV }]; #IO_L6N_T0_VREF_16 Sch=eth_crsv  
#set_property -dict { PACKAGE_PIN C10 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_RXERR }]; #IO_L13N_T2_MRCC_16 Sch=eth_rxerr  
#set_property -dict { PACKAGE_PIN C11 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_RXD[0] }]; #IO_L13P_T2_MRCC_16 Sch=eth_rxd[0]  
#set_property -dict { PACKAGE_PIN D10 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_RXD[1] }]; #IO_L19N_T3_VREF_16 Sch=eth_rxd[1]  
#set_property -dict { PACKAGE_PIN B9 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_TXEN }]; #IO_L11N_T1_SRCC_16 Sch=eth_txen  
#set_property -dict { PACKAGE_PIN A10 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_TXD[0] }]; #IO_L14P_T2_SRCC_16 Sch=eth_txd[0]  
#set_property -dict { PACKAGE_PIN A8 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_TXD[1] }]; #IO_L12N_T1_MRCC_16 Sch=eth_txd[1]  
#set_property -dict { PACKAGE_PIN D5 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_REFCLK }]; #IO_L11P_T1_SRCC_35 Sch=eth_refclk  
#set_property -dict { PACKAGE_PIN B8 IOSTANDARD LVCMOS33 } [get_ports  
{ ETH_INTN }]; #IO_L12P_T1_MRCC_16 Sch=eth_intn
```

##Quad SPI Flash

```
#set_property -dict { PACKAGE_PIN K17 IOSTANDARD LVCMOS33 } [get_ports  
{ QSPI_DQ[0] }]; #IO_L1P_T0_D00_MOSI_14 Sch=qspi_dq[0]  
#set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMOS33 } [get_ports  
{ QSPI_DQ[1] }]; #IO_L1N_T0_D01_DIN_14 Sch=qspi_dq[1]
```



```
#set_property -dict { PACKAGE_PIN L14  IOSTANDARD LVCMOS33 } [get_ports  
{ QSPI_DQ[2] }]; #IO_L2P_T0_D02_14 Sch=qspi_dq[2]  
#set_property -dict { PACKAGE_PIN M14  IOSTANDARD LVCMOS33 } [get_ports  
{ QSPI_DQ[3] }]; #IO_L2N_T0_D03_14 Sch=qspi_dq[3]  
#set_property -dict { PACKAGE_PIN L13  IOSTANDARD LVCMOS33 } [get_ports  
{ QSPI_CSN }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_csn
```



ANEXA 19 Cod python obținere fișier binar

Cod

```
from PIL import Image

im = Image.open('poza.jpg', 'r')

pix_val = list(im.getdata())


pix_val2 = [ (int(r*15/255), int(g*15/255), int(b*15/255)) for (r, g, b) in
pix_val]

"""Scriere intr-un fisier text"""
new_file=open("newfileTXT.txt",mode="a+", encoding="utf-8")

i=0
for pixel in pix_val2:
    i+=1
    new_file.write(f"{{pixel[0]}, {pixel[1]}, {pixel[2]}} ")
    if i!=307200 and i%480==0:
        new_file.write("\n")

new_file.close()

"""Scriere intr-un fisier binar"""
new_file=open("newfileBIN.bin",mode="wb")

for pixel in pix_val2:
    new_file.write(pixel[0].to_bytes(1,'big'))
    new_file.write(pixel[1].to_bytes(1,'big'))
    new_file.write(pixel[2].to_bytes(1,'big'))

new_file.close()
```