



Implementarea prin hardware a unui algoritm de procesare a imaginilor

Autori: Apostol Cătălin Ovidiu

Zoican Denis Alexandru

Grupa : 30223

Indrumator: Fati Daniela

Data: 05/01/2020

Cuprins

1. Rezumat	3
2. Introducere	4
3. Fundamente teoretice	5
4. Proiectare si implementare	6
4.1. Metoda experimentală	6
4.2. Solutia aleasa	6
4.3. Arhitectura generala a sistemului.....	6
4.4. Algoritmi implementati.....	8
4.4.1. Modul VGA.....	8
4.4.2. Modul principal de procesare	11
4.5. Manual de utilizare	15
5. Rezultate experimentale.....	19
5.1. Instrumente de proiectare utilizate.....	19
5.2. Circuit utilizat pentru implementare	19
5.3. Procedura de testare utilizata	20
5.4. Capturi de ecran cu testarea rezultatelor.....	21
6. Concluzie.....	28
BIBLIOGRAFIE.....	29
ANEXA	30

1. Rezumat

Procesarea de imagini este din ce in ce mai intalnita in viata de zi cu zi, fiind regasita in telefoanele noastre mobile pana la tehnologiile avansate folosite de armata. Acest domeniu cuprinde foarte multi algoritmi complecsi si tehnologii noi, dar are la baza niste concepte ce vor fi prezentate in acest fisier.

Proiectul se axeaza pe implementarea unor algoritmi de procesare asupra unor imagini stocate pe placa Basys3 intr-o maniera hardware, rezultatul fiind afisat pe un monitor prin intermediul unei conexiuni VGA.

Proiectul este implementat utilizand limbajul de descriere hardware VHDL, utilizand cunostiinte dobandite la facultate ce au oferit posibilitate crearii unei diagrame de stare prin intermediul careia multiple componente create au fost conectate, interactionand intre ele.

Dupa finalizarea proiectului, rezultatele au fost unele favorabile, reusind sa evidentieze transformarile care au fost aplicate pe imagine, aceste rezultate fiind afisate pe un monitor.

Implementarea hardware a dat dovada de viteza sporita.

2. Introducere

În contextul dezvoltării constante a tehnologiilor, procesare de imagini a avut parte de o imensă evoluție în ultimii ani, fiind folosită în tot mai multe domenii (cum ar fi autonomus driving, aplicații pentru telefoane mobile, automatizarea armelor).[1]

Procesul de dezvoltare constantă a calității imaginilor are ca și consecințe creșterea dimensiunii imaginilor ceea ce aduce și la necesitatea de componente hardware cu o putere computațional tot mai mare. În acest context, capacitățile programării paralele de pe placă FPGA o fac să fie o opțiune atractivă pentru rezolvarea unor astfel de acțiuni repetitive.[1]

În cadrul proiectului ne-am hotărât să ne axăm pe domeniul precizat anterior, prin aplicarea unor algoritmi de procesare a imaginilor prin intermediul plăcii FPGA din familia ARTIX-7, Basys 3. Rezultatul acestor procesări va fi afișat pe un ecran conectat la placa FPGA prin intermediul portului VGA.

Folosindu-ne de placă FPGA am dezvoltat o interfață ușor de folosit de către utilizator, acesta putând aplica diferiți algoritmi de procesare pe o imagine prin intermediul butoanelor. Imaginea procesată rezultată după apăsarea unui buton va fi afișată pe ecran, utilizatorul putând observa schimbările realizate. Proiectul este împărțit în 2 module principale: Conectarea VGA și aplicarea algoritmilor de procesare pe imaginea din memorie. Aceste module au fost dezvoltate separat, ulterior fiind combinate în proiectul principal.

În capitolele ce urmează am prezentat în mod amănunțit pașii pe care i-am urmat în dezvoltarea acestui proiect, dar și rezultatele obținute și modul de utilizare al programului obținut. În capitolul “Fundamente teoretice” am prezentat sursele pe care le-am folosit pentru a cunoaște mai bine domeniul problemei și tehnologiile pe care le putem aborda în cadrul proiectului. În capitolul “Proiectare și Implementare” am explicat pașii pe care i-am urmat în procesul de implementare al acestui proiect, am descris fiecare componentă principală a proiectului, ideea de funcționare, dar și modul prin care le-am implementat în limbaj hardware, dar și un manual de utilizare al produsului final obținut. În capitolul “Rezultate experimentale” am prezentat ustensilele pe care le-am folosit pentru debugging și prezentarea rezultatelor obținute. Rezultatele obținute prin intermediul ecranului dar și simulări din timpul au fost capturate și atasate mai jos în această rubrică. Tot aici am prezentat și câteva probleme care au apărut și impedimente pe care le-am avut atât din pricina softului prin care am programat placa, cât și specificațiile tehnice ale acesteia. În capitolul Concluzii am explica pe larg ce am realizat în urma implementării, și extensii ce pot fi adăugate, sau implementări viitoare la acest proiect.

Tot codul folosit se poate vedea în secțiunea Anexa.

3. Fundamente teoretice

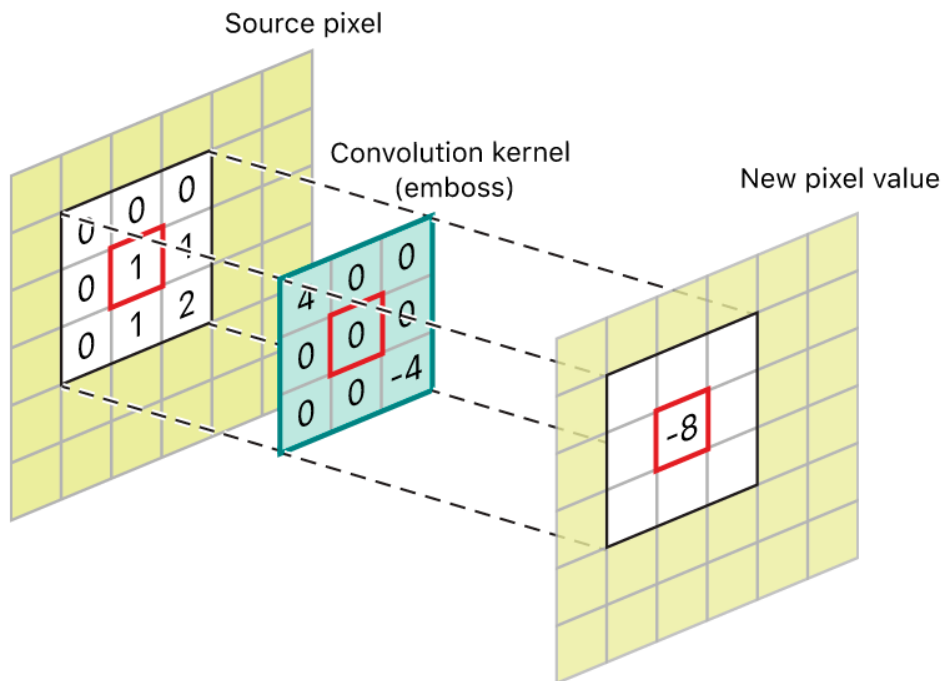
Înainte de implementarea efectivă a proiectului am început să ne documentăm citind multiple articole ce prezentau succint informații despre procesarea de imagini.

Un avantaj major în folosirea plăcilor FPGA pentru procesarea imaginilor îl reprezintă executia paralelă a instrucțiunilor, astfel fiind preferată o abordare hardware față de una software.[2]

Algoritmii folosiți au trebuit să fie adaptați unei abordări hardware, aceștia reprezentând algoritmi care stau la baza procesării de imagini: sobel edge detection, blur filter, greyscale transformation.[3]

Pentru implementarea algoritmilor menționați mai sus am utilizat matrice de convoluție Kernel (figura 1) care vor aplica un anumit filtru pe pixel, în funcție de valorile acestuia. Pentru fiecare tip de procesare există o matrice de convoluție diferită.

Figura 1) Aplicarea unei matrice de convoluție Kernel pe un pixel[6]



Conexiunea VGA reprezintă o modalitate rapidă și simplă pe care o putem folosi pentru a afișa imaginile stocate pe FPGA. Pentru ca monitorul să recepționeze datele dorite, aplicația trebuie să

trimite 5 semnale: R,G,B pentru a reda culoarea pixelului si respectiv HSync si VSync care ne ajuta la determinarea pozitiei.[4]

4. Proiectare si implementare

4.1. Metoda experimentală

Pentru verifica functionarea corecta a proiectului, initial am realizat un modul de simulare in mediul de dezvoltare Vivado in cadrul caruia am testat daca sunt trimise valorile corecte pe porturile R,G,B,HSync si VSync corespunzatoare pixelului procesat. La simulare am generat un semnal de clock , iar apoi am verificat ca pe fiecare front crescator al acestui clock sa se schimbe adresa corespunzator, si am verificat ca valoarea pixelului trimis sa fie corespunzatoare valorii pixelului din fisierul .coe.

Dupa observarea unor rezultate bune oferite de simulator, am inceput afisarea imaginii pe un dispozitiv fizic, monitor, prin intermediul conexiunii VGA.

4.2. Solutia aleasa

Din cauza limitarilor pe care le are placa FPGA pe care am folosit-o pentru proiect (Basys3) si anume cea mai mare limitare fiind memoria sa interna scazuta (1,800 Kbits of RAM), nu am putut stoca videouri pentru o procesare de imagini in timp real, ba chiar am fost nevoiti sa folosim imagini de dimensiuni mici.

Ne-am hotarat sa implementam 3 algoritmi de baza in procesarea de imagini si anume: Blur filter , GreyScale transformation si respectiv Sobel Edge detection.

- Blur filter: este folosit pentru a oferi un efect de blur pe toata suprafata imaginii.
- GreyScale transformation: transforma poza color intr-o imagine formata din nuante de gri.
- Sobel Edge detection: Afiseaza doar liniile puse in evidenta de contrastul dintre pixelii vecini.

4.3. Arhitectura generala a sistemului

Proiectul cuprinde 4 componente interconectate între ele: componenta periferică, monitorul, componenta de memorie ROM (care ține în interior imaginea înainte și după procesare), componenta VGA care ajută la realizarea conexiunii între componenta Main și monitor, și componenta main care conține atât algoritmi cât și state machine-ul după care programul funcționează.(Conform Figura 2)).

Figura 2a) Schema bloc generală a proiectului.

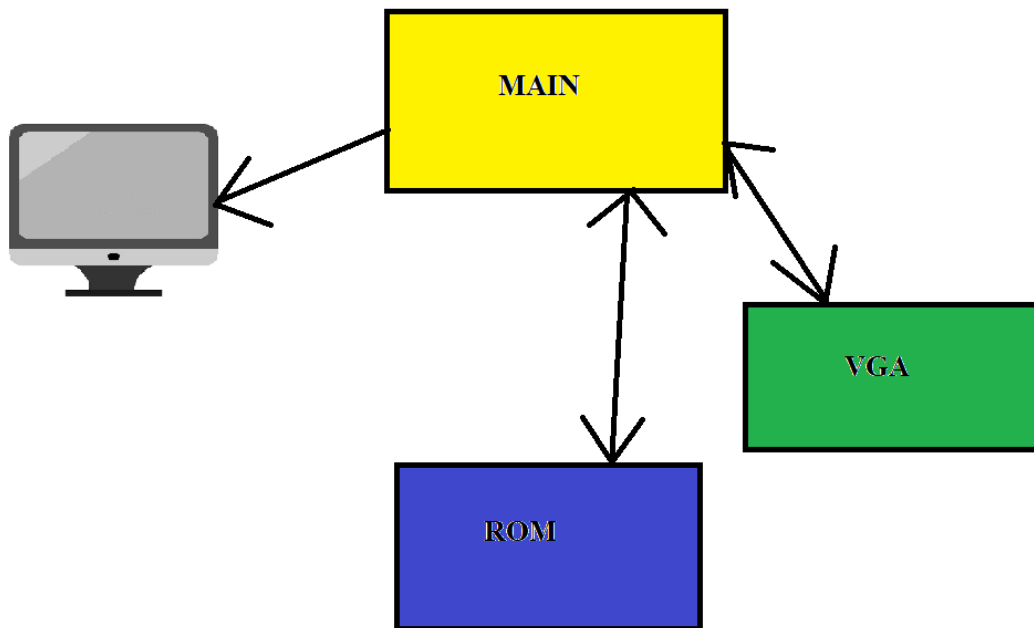
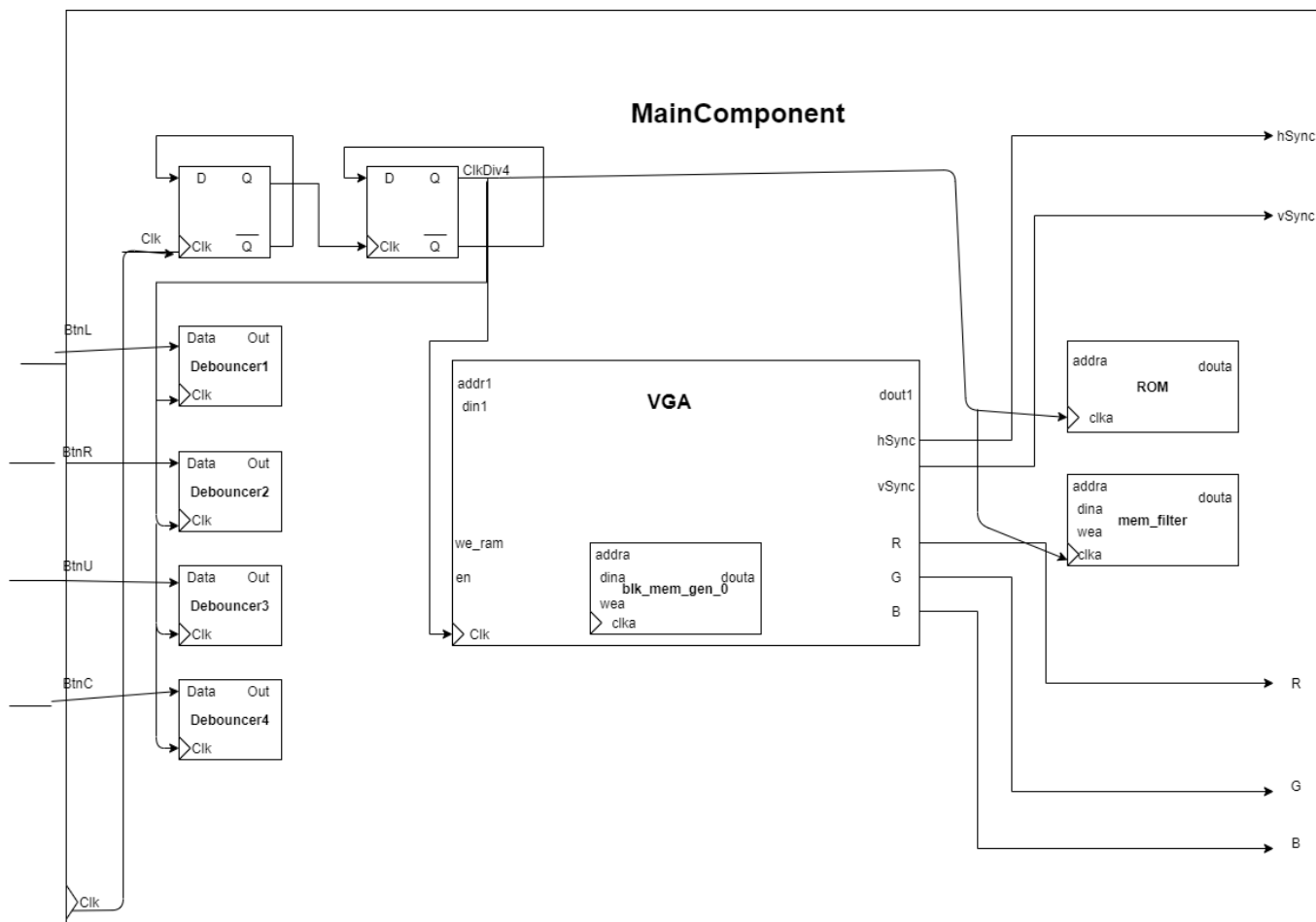


Figura 2b) Schema bloc prezentând I/O și semnalele de control



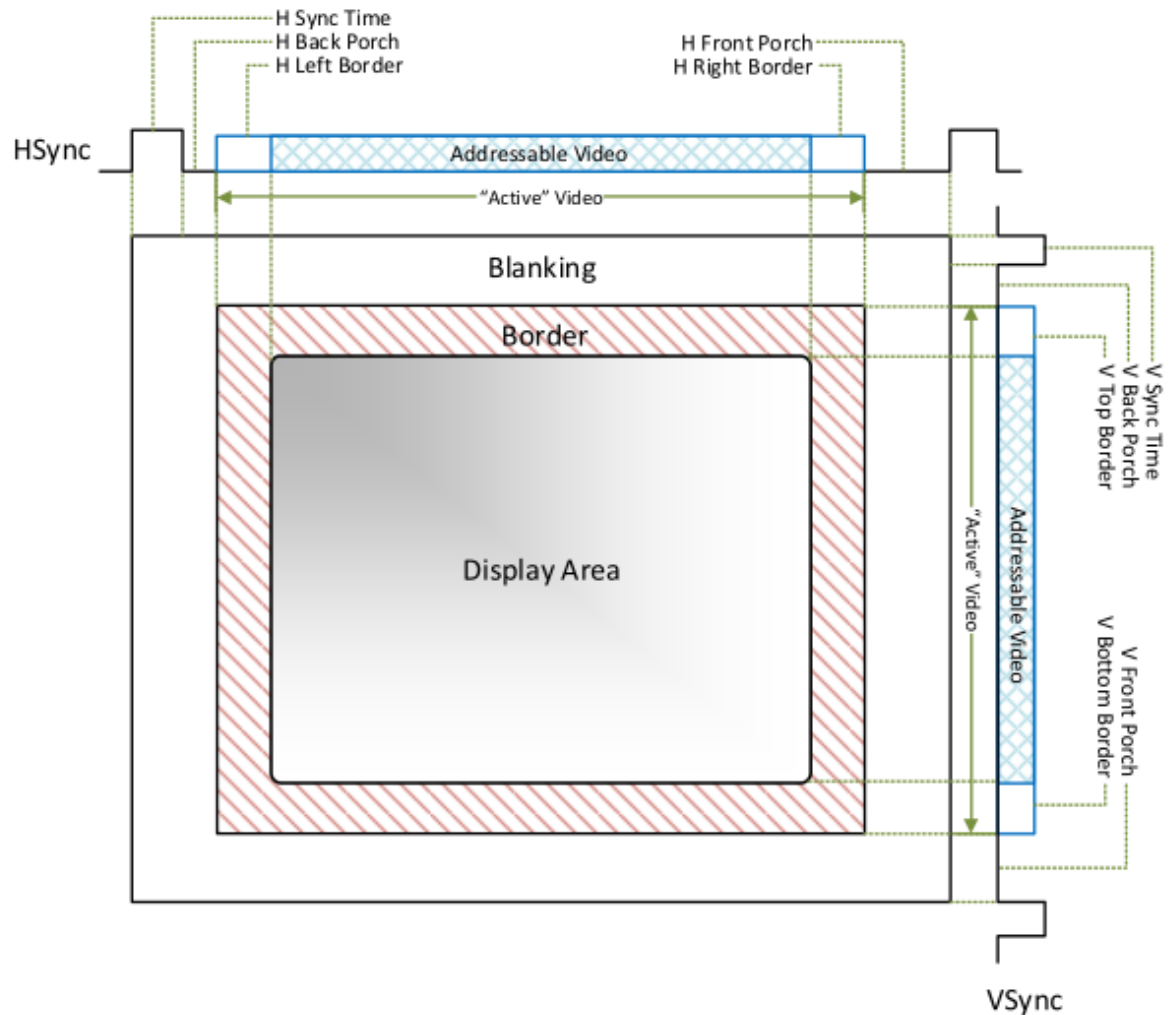
4.4. Algoritmi implementati

4.4.1. Modul VGA

Analizand Figura 3) putem observa cum este impartit ecranul, care este suprafata adresabila a acestuia, si modul in care putem verifica daca ne aflam in suprafata adresabila folosindu-ne de HSync si VSync.

Intervalele BackPorch, FrontPorch si respectiv Sync reprezinta momentele in care ecranul nu afiseaza informatia primita prin cablul VGA. Aceste intervale exista atat pentru directia orizontala (HBackPorch, HFrontPorch si HSync) care se activeaza dupa parcurgerea unei linii , cat si pentru directia verticala (VBackPorch, VFrontPorch si VSync) care se activeaza dupa parcurgea unei coloane. Parcurgerea pixelilor pe ecranul VGA se face incepand din coltul din stanga sus , parcurgandu-se pe rand fiecare linie.

Figura 3) Specificari de sincronizare pentru un ecran [4]



Imaginea afisata prin intermediul VGA este preluata dintr-o memorie ROM, aceasta fiind in format .coe. In fisierul .coe, fiecare bit este reprezentat in TrueColor, ceea ce inseamna ca fiecare componenta R,G,B este formata din 8 biti. Insa pinii R,G,B de pe portul VGA primesc doar 4 biti, asta inseamna ca paleta de culori este mult mai restransa. La conversie vom folosi doar cei mai semnificativi 4 biti.

Cel mai important obiectiv al acestui model este de seta semnalele HSync si VSync la momentele potrivite deoarece trebuie sa existe o sincronizare perfecta intre trimiterea pixelilor si pozitia pointerului de pe matricea de pixeli a ecranului. Acest lucru este realizabil prin folosirea mai multor procese sincronizate de acelasi semnal de ceas.

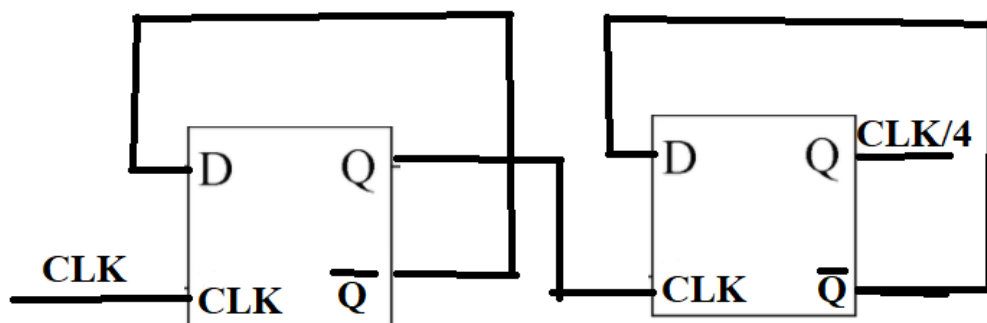
In functie de ecranul folosit, pentru implementarea proiectului a trebuit sa schimbam frecventa clock-ului pentru a fi compatibila cu rata de refresh a ecranului. In cadrul proiectului am folosit un ecran cu rezolutia de 640x480 si rata de refresh 60Hz, deci va trebui sa folosim un clock de 25 Mhz(Figura 4) pentru a sincroniza programul cu monitorul.

Figura 4) Clk-ul pixelilor in functie de rezolutie si rata de refresh a monitorului.[5]

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21

Pentru asta am folosit doua bistabile de tip D interconectate intre ele pentru a diviza clock-ul de pe placa (de 100 de Mhz) de 4 ori.(Figura 5)

Figura 5) Divizare clock utilizand 2 bistabile D.



Pentru a decide unde ne aflam pe ecran la momentul actual am folosit doua procese pentru pozitia orizontala si respectiv verticala. La fiecare semnal de clock se va incrementa pozitia hPos respectiv vPos , iar cand se ajunge in zona de HSync respectiv VSync, se vor reseta cele doua variabile pentru pozitie. Aceste variabile sunt utilizate pentru a verifica daca ne aflam in spatiul adresabil, in caz contrar nu vom desena pixelul in acea pozitie si nici nu vom inainte in memorie.

Informatiile despre pixelii ce vor fi desenate pe ecran sunt preluate din blocul de memorie alocat in care am incarcat in prealabil fisierul .coe ce reprezinta imaginea pe care am dorit sa o folosim.

Pentru stocarea imaginii am folosit un bloc de memorie generat folosind functia IP Catalog a mediului de dezvoltare Vivado , cu dimensiunea cuvintului de 24 de biti (reprezentand pixelii in TrueColor) si numarul de cuvinte egal cu 16384(care reprezinta numarul de pixeli ai pozei incarcate poza avand o dimensiune de 128x128). Am ales sa folosim o imagine de 128x128 deoarece am fost limitati de dimensiunea memoriei interne a placii si faptul ca avem nevoie de macar 2 copii ale imaginii pentru a realiza procesarea.

Pentru a genera fisierul .coe pe care l-am incarcat in memorii a trebuit sa facem urmatoarele transformari pe imagine: am redimensionat imaginea aleasa pentru a fi de 128x128 pixeli [7] . Am transformat imaginea jpg/png intr-o imagine binara folosind un program online [8] dupa care am folosit un executabil open source care ne-a generat fisierul .coe din fisierul binar [9] .

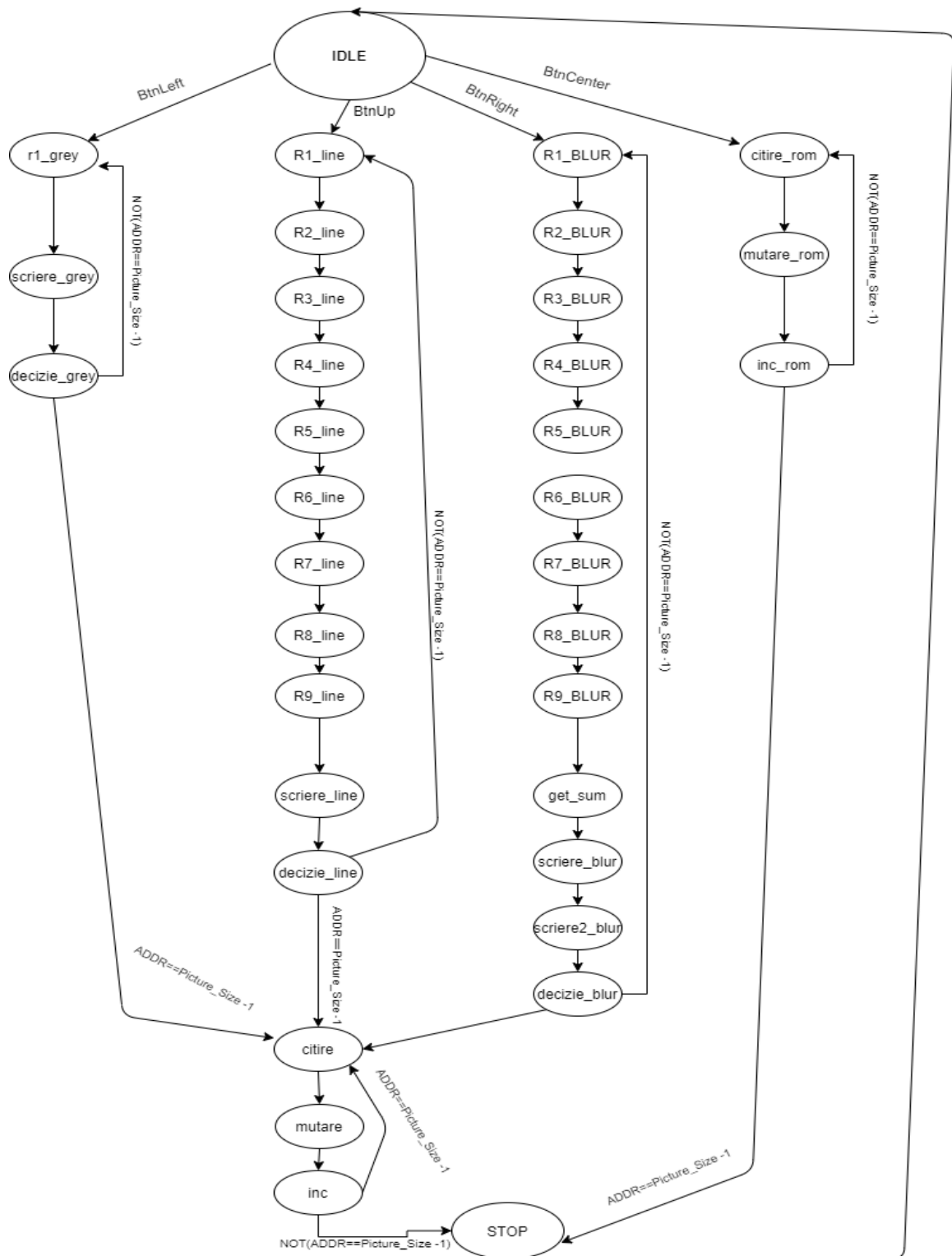
Pentru a aplica un filtru pe o imagine , avem nevoie de o memorie auxiliara deoarece la procesarea unui pixel nu vrem sa folosim pixelii deja procesati, avem nevoie doar de pixelii din imaginea originala. Am facut si o functie care sa reseteze imaginea la imaginea initiala(fara nici un filtru aplicat pe ea), pentru aceasta am folosit inca un bloc de memorie(Rom de data aceasta), si acesta initializat cu fisierul .coe reprezentand poza de afisat.In total am folosit 2 blocuri de memorie Ram si unul ROM toate de dimensiunea 24x16384.

4.4.2. Modul principal de procesare

Ideea principala pentru procesarea imaginilor o reprezinta aplicarea unei matrici de convolutie Kernel asupra fiecarui pixel pentru a-i schimba informatia in functie de vecinii acestuia.

Deoarece nu am putut aplica algoritmi de procesare intr-o maniera software, acestia au fost adaptati pentru aplicarea unei metode hardware care are la baza un state-machine.(Figura 6)).

Figura 6) Diagrama de stare modulul principal



Fiecare dintre cele 4 ramuri principale ale diagramei de stare realizeaza unul dintre cele 4 obiective principale ale proiectului:

- Ramura din dreapta, care incepe cu citire_Rom, pe aceasta ramura se va copia poza initiala in memoria pe care o folosim la afisare (reseteaza poza).
- Ramura din stanga, care incepe cu r1_grey, pe aceasta ramura convertim poza in greyScale. In r1_grey, se calculeaza media aritmetica a celor 3 canale de culoare, in scriere_grey, se scrie in memoria auxiliara (mem_filter) valoarea calculata anterior, iar in decizieGrey verificam daca am ajuns la finalul matricii, in acest caz trecem la starea citire, altfel ne intoarcem la r1_grey.
- Ramura a 3-a, R1_BLUR, aplica filtrul de blur pe imagine. In starile R1_Blur pana la R9_Blur vom citii pixeli vecini si ii vom inmulti cu valoarea corespunzatoare din matricea kernel (Figura 7)), acest proces se va face pentru fiecare canal de culoare in parte. In stare get_sum vom imparti cele 3 sume obtinute (sumR,sumG,sumB) la numarul de pixeli implicati in suma (9) si vom crea pixelul ce va trebui scris in memorie. In starile scriere_blur si scriere2_blur se realizeaza scriere pixelului format anterior in memoria mem_filter. In starea decizie_blur verificam daca am ajuns la finalul memorie, in acest caz trecem la starea citire. Altfel ne intoarcem la r1_grey.

Efectul de blur se obtine prin aplicarea acestei matrici deoarece si pixelii vecini contribuie la formarea culorii pixelului procesat.

Figura 7) Matrice de convolutie Kernel pentru filtrul Blur

1	2	1
2	4	2
1	2	1

- Ramura a (2) , R1_line . Pe aceasta ramura se realizeaza aplicarea filtrului de edge detection folosind un algoritm de tipul Sobel edge detection. R1_line pana la R9_line se aplica matricile kernel care detecteaza liniile orizontale si respectiv verticale pe fiecare dintre pixelii vecini, si se calculeaza suma acestoare (sumaX si sumaY) (Figura 8)). In starea scriere line vom calcula daca pixelul la care ne aflam se afla sau nu pe o linie (aceasta verificare se face utilizand o valoare prag pe care am dat-o in cod), si apoi se salveaza in matrice pixelul calculat astfel, cu o anumita intensitate. Daca punctul se afla atat pe o linie orizontala cat si una verticala va trebui sa combinam cele 2 valori de pe x si y. Cel mai eficient mod de combinare a acestor valori este folosind functia radical(sqrt) , insa este foarte costisitor sa facem o astfel de operatie, iar pentru cerintele noastre am ales sa folosim media aritmetica ca si metoda de combinare. Apoi in starea decizie_line se verifica daca am aplicat filtrul pe toti pixelii pozei , daca da se va trece la starea citire, altfel se va reveni la starea initiala(R1_line).

Figura 8) Matricile kernel pentru detectia liniilor verticale(Y) respectiv orizontale (X)

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Dupa ce am aplicat filtrele corespunzatoare, vom trece la stare Citire.In aceasta secventa de 3 stari se va face copierea memoriei auxiliare in memoria principala , din care se va face afisarea. In starea citire se citeste pixelul din memoria auxiliara, in stare mutare se va scrie in memoria principala valoarea citita anterior, in starea inc , daca nu am ajuns la finalul matricii auxiliare se incrementeaza adresa pixelului actual pentru a trece la urmatorul, altfel se trece in starea stop.

Din starea Stop se trece direct in starea IDLE.

Cat timp suntem in stare IDLE , vom face afisarea . Ramanem in starea IDLE pana cand se apasa unul din butoanele: BTNCenter, BTNUp, BTNLeft respectiv

BTNRright, care vor trece masina de stari intr-unul din cele 4 brate prezentate in Figura 6).

Diagrama de stare lucreaza direct cu semnalele de control care controleaza restul componentelor.

- EnableVGA. Aceste semnal are rolul de a permite functionarea modulului VGA doar in anumite situatii. Este setat pe 1 doar in Starea Idle. Pentru ca reprezinta momentul in care procesarea a luat sfarsit.
- Wea_MEM . Acesta este semnalul de WriteEnable pentru memoria auxiliara utilizata la procesarea imaginii. Acest semnal va fi activat cand vom avea nevoie sa scriem rezultatele calculate in memoria auxiliara (in starile scriere_grey, scriere_blur si scriere_line).
- We_RAM . Acesta este semnalul de WriteEnable pentru memoria principala , din care se face afisarea. Acest semnal este activ atunci cand dorim sa trecem datele din memoria auxiliara, sau din memoria ROM , in memoria principala.(in starile mutare si mutare_rom).
- Tot in procesul principal in starile de incrementare , se incrementeaza adresa la care ne aflam in o anumita memorie.

4.5. Manual de utilizare

Pentru a putea folosi proiectul implementat utilizatorul are nevoie de urmatoarele dispozitive hardware(Figura 9)):

- a. Placa FPGA Basys3
- b. Monitor cu port VGA
- c. Un laptop sau calculator cu medium de dezvoltare Vivado instalat , un port USB
- d. 1) Un Cablu VGA pentru a conecta placa FPGA la monitor
2) Un cablu USB->MicroUSB pentru a conecta placa la laptop/calculator.

Figura 9) Componente necesare implementarii proiectului.



a)



b)



c)



d1)



d2)

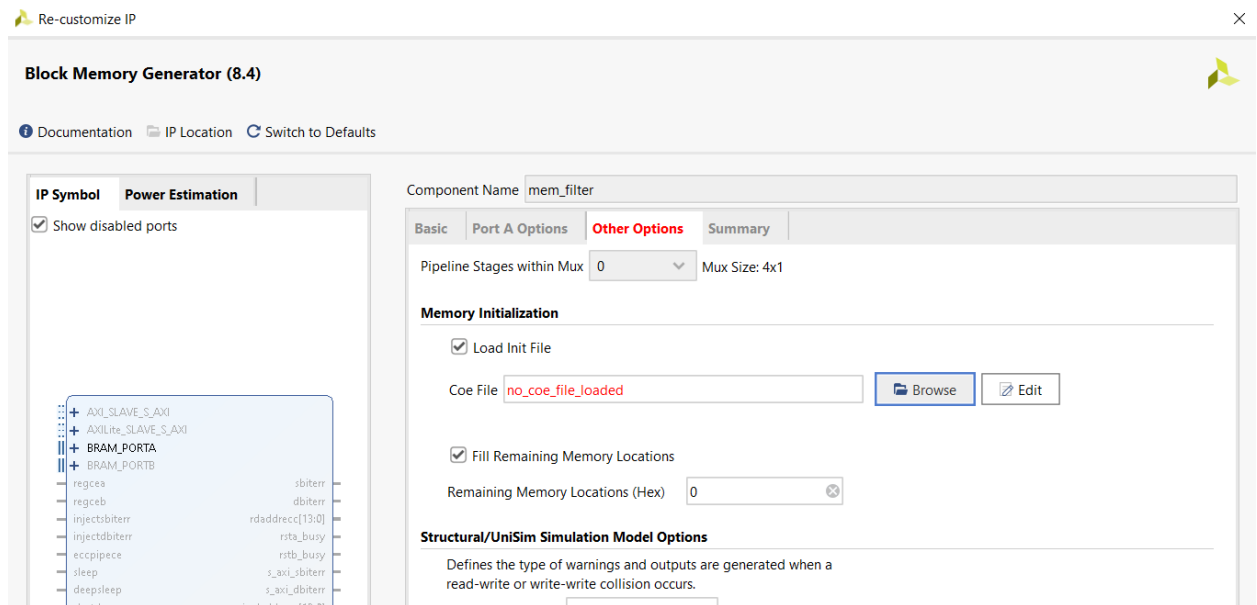
Primul pas este conectarea monitorului si laptopului la curent, dupa care se face conectarea monitorului la placa de dezvoltare, respectiv a laptopului la placa de dezvoltare prin cele 2 cabluri VGA si USB.

Al doilea pas: instalarea mediului de dezvoltare VIVADO.

Al treilea pas: deschiderea proiectului utilizand mediul de dezvoltare VIVADO.

Al patru-lea pas: incarcarea celor 3 blocuri de memorie cu fisierul .coe corespunzator pozei initiale .(conform Figura 10)

Figura 10) Incarcare fisiere .coe in blocul de memorie generat prin IP catalog.

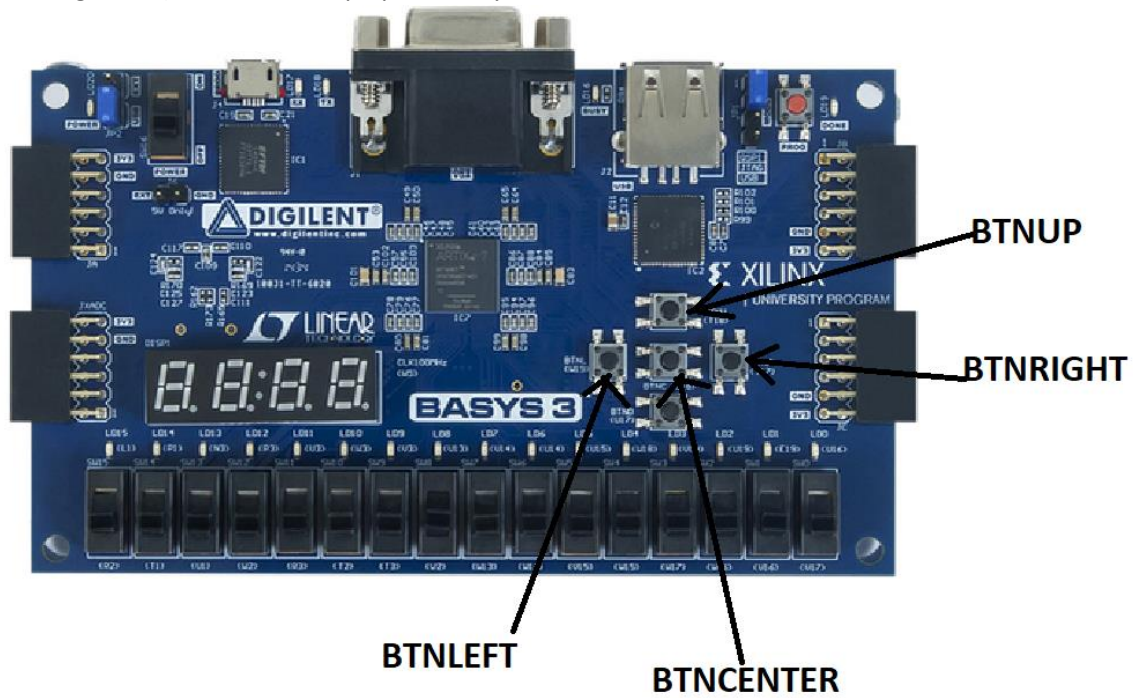


Al cinci-lea pas: generam bitstreamul.

Al sase-lea pas: incarcam bistreamul pe placa de dezvoltare.

Dupa incarcarea bistreamului utilizatorul ar trebui sa vada pe ecran poza incarcata de acesta in blocurile de memorie. Ulterior acesta poate sa aplice diferite filtre prin intermediul butoanelor de pe placa (Figura 11)).

Figura 11) Butoanele de pe placa Basys3.



Modul de utilizare al butoanelor :

- Butonul BTNLEFT aplica un filtru GreyScale pe imaginea curenta.
- Butonul BTNRIGHT aplica un filtru Blur pe imaginea curenta.
- Butonul BTNUP aplica un filtru edge detection pe imaginea curenta.
- Butonul BTNCENTER reseteaza imaginea in starea ei initiala.

Se poate realiza aplicarea filtrelor succesive pe imagini.

5. Rezultate experimentale

5.1. Instrumente de proiectare utilizate

Mediul de dezvoltare ales a fost XILINX VIVADO, un program software pentru sinteza si analiza proiectelor HDL, inlocuind XilinX ISE cu functii suplimentare pentru dezvoltarea unui sistem pe chip si sinteza la nivel inalt. Acesta este disponibil atat pe Windows cat si pe Linux. Folosim acest program atat la dezvoltarea cat si simularea proiectului.

Simulatorul Vivado este un simulator care are multe functii ce suporta ca si limbaje de programare Verilog, SistemVerilog si VHDL. Este folosit pentru a vizualiza modificarea diferitelor semnale la un anumit moment de timp din ciclul de functionare al proiectului.[10]










Limbajul de programare Hardware pe care l-am ales este VHDL. Limbaj folosit pentru automatizarea design-ului electronic si descrierea sistemelor digitale.

5.2. Circuit utilizat pentru implementare

Implementarea proiectului nostru a constatat in implementarea a doua functii principale: conectarea placii la VGA si afisarea unei imagini pe monitor si procesarea imaginilor prin intermediul unui program hardware.

Prima parte de care ne-am ocupat a fost conectarea placii la monitor prin portul VGA si afisarea imaginii ca si metoda de testare pentru aceasta componenta am folosit afisarea pe ecran, dupa ce am rezolvat problemele aparute, am implementat si modulul principal care aplica filtrele pe imagini.

Figura 12a) Utilizarea Luturilor , Registrilor si Blockurilor de RAM.

Name	^1	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
▼ N Main		726	321	291	726	27	19	2
 D1 (D_FlipFlop)		1	1	1	1	0	0	0
 D2 (D_FlipFlop_0)		1	1	1	1	0	0	0
 debouncer1 (debounce)		1	3	3	1	0	0	0
 debouncer2 (debounce_1)		12	3	10	12	0	0	0
 debouncer3 (debounce_2)		9	3	7	9	0	0	0
 debouncer4 (debounce_3)		5	3	6	5	0	0	0
>  Memorie (mem_filter)		20	4	11	20	9	0	0
>  ROmul (ROM)		26	9	12	26	9	0	0
>  VGA1 (VGA)		396	125	166	396	9	0	0

5.3. Procedura de testare utilizata

Ca si metode de testare si simulare am folosit atat simulatorul din mediul de dezvoltare Vivado cat si reprezentarea imaginii pe monitor prin portul VGA.

In Vivado am creat un banc de test in care am instantiat clasa noastra principala (main, in care am modificat pentru testare butoanele sub forma unor switchuri).

Am folosit bancul de test creat pentru a verifica daca imaginea este procesata corect, mai exact, daca valorile pentru canalele de culoare sunt generate corect. Putem observa ca pentru filtrul blur, valorile sumar, sumag si sumab au fost calculate luand in considerare valorile pixelilor vecini cat si valorile din matricea de convolutie kernel. In stările r1_blur, r2_blur, r3_blur, r4_blur, r5_blur, r6_blur, r7_blur, r8_blur, r9_blur sunt preluate valorile vecinilor pixelului procesat, inmultite si insumate pentru a fi impartite la numarul de pixeli din matricea de convolutie in stare get_sum. Starea scriere este folosita pentru a scrie noile valori calculate in matricea auxiliara creata in noul bloc de memorie.

Figura 12 b c d) Poze din modulul de simulare prezent in Vivado din timpul aplicarii Blur

Figura 12b)

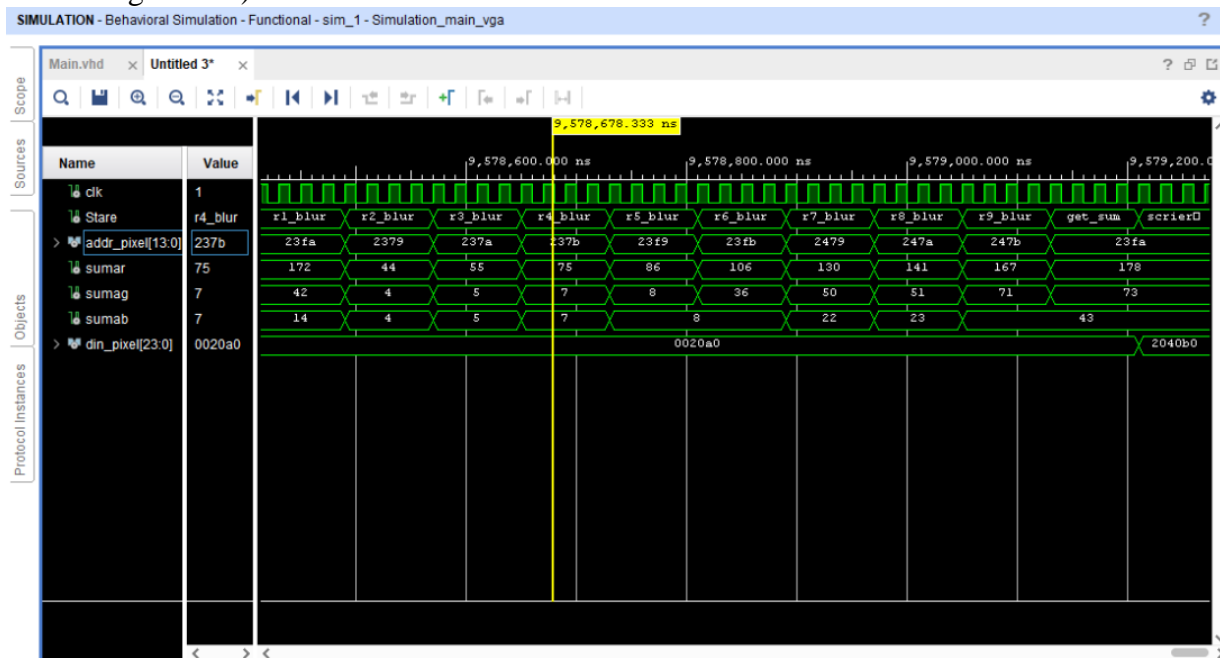


Figura 12c)

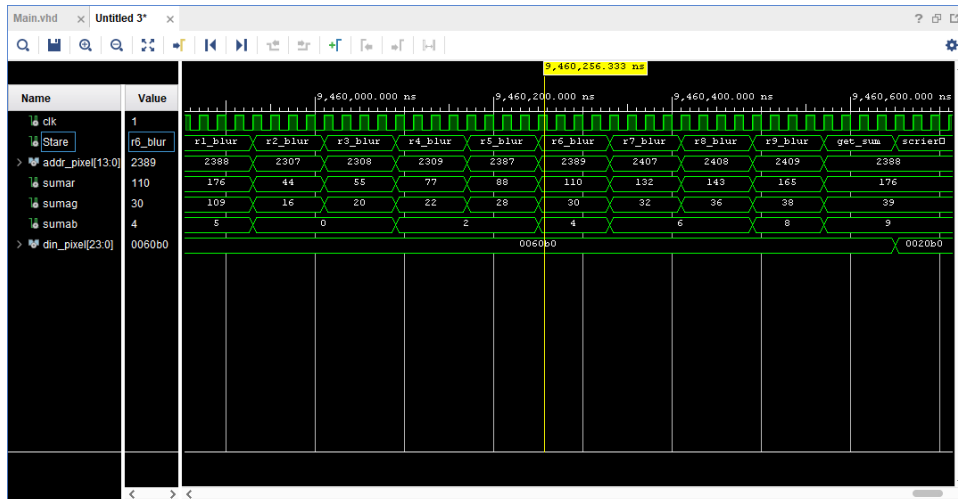
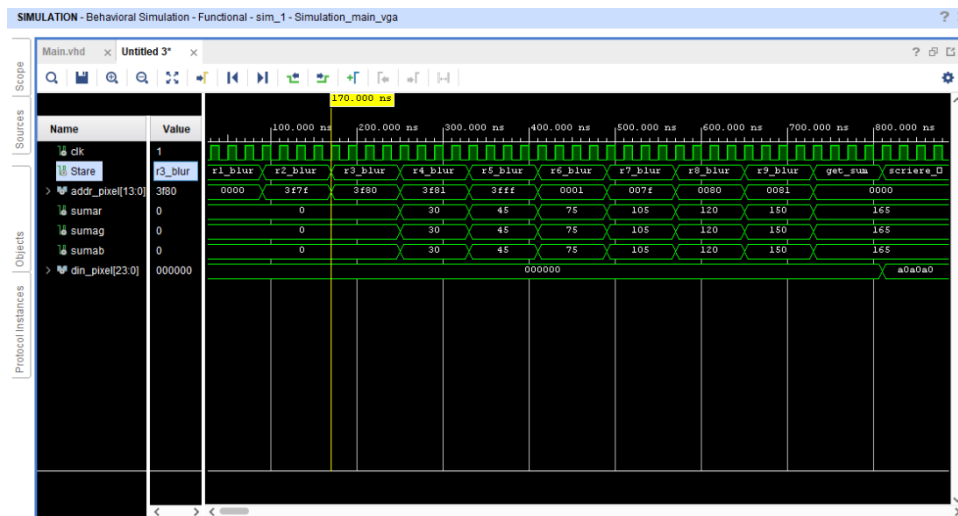


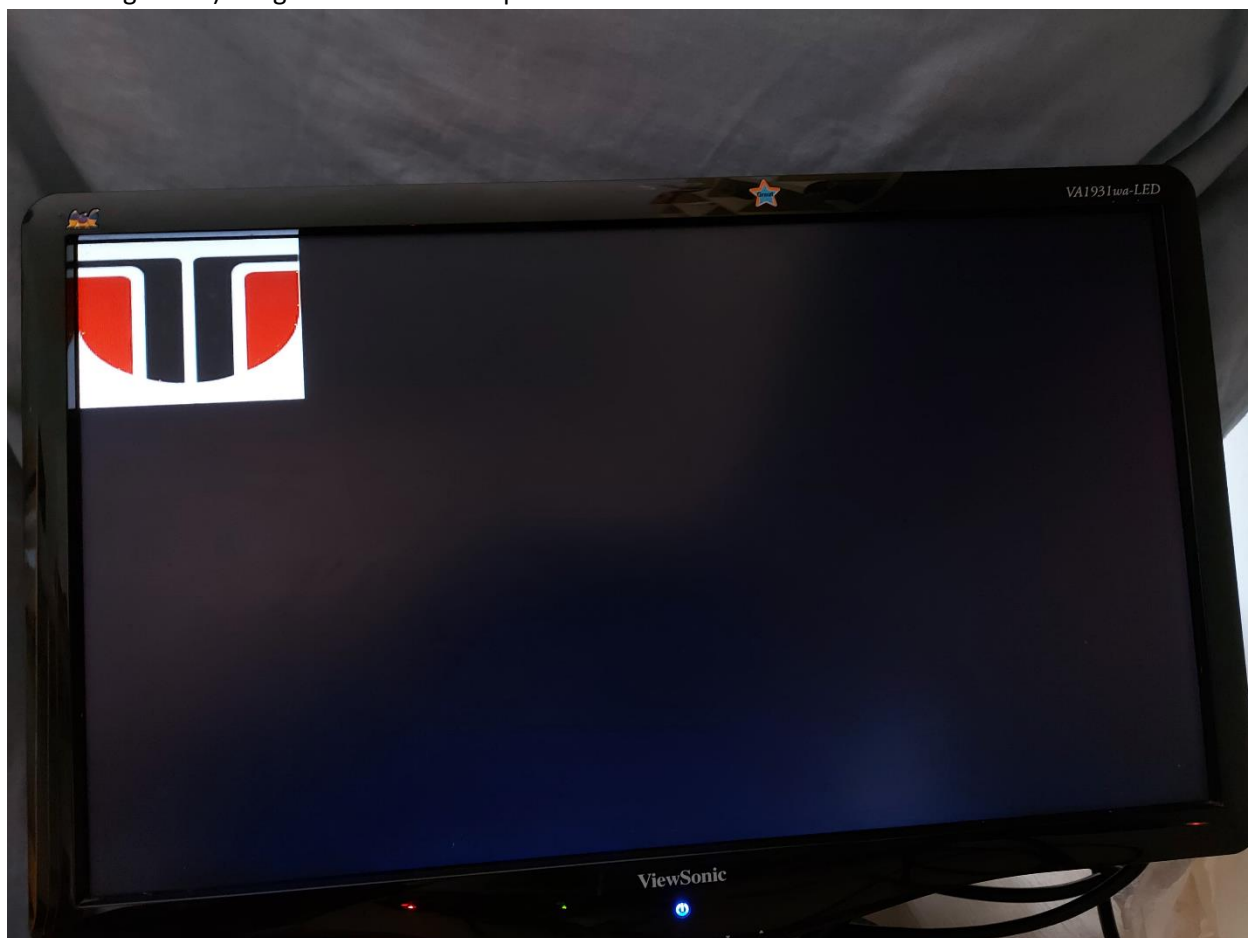
Figura 12d)



5.4. Capturi de ecran cu testarea rezultatelor

Dupa incarcarea fisierului .bit pe placa FPGA, utilizatorul va vedea pe ecran imaginea originala ce se afla pe placuta sub forma unui fisier .coe .

Figura 13) Imaginea afisata initial pe monitor



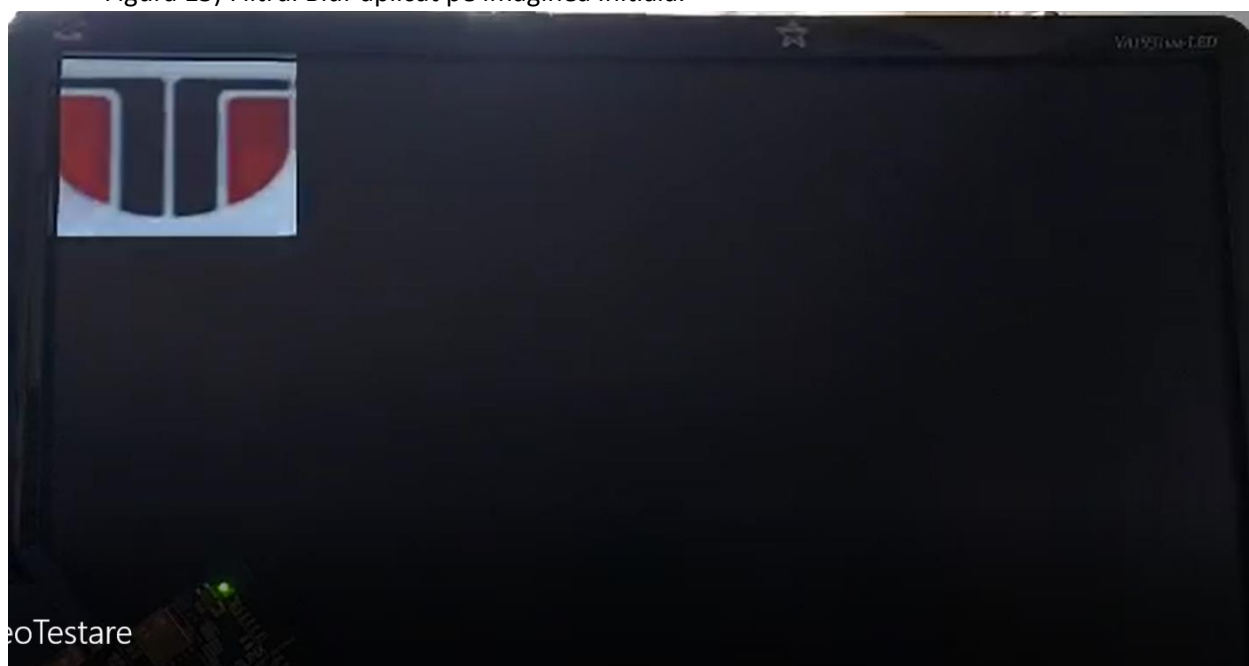
Urmarind manualul de utilizare prezentat anterior, putem aplica diferite filtre asupra imaginii afisate pe ecran. Prin apasarea butonului BTNLEFT se aplica un filtru GreyScale asupra imaginii.

Figura 14) Filtrul GreyScale aplicat pe imaginea initiala



Prin apasarea butonului BTNRIGHT se aplica un filtru Blur asupra imaginii.

Figura 15) Filtrul Blur aplicat pe imaginea initiala.



Prin apasarea butonului BTNUP se aplica un filtru EdgeDetection asupra imaginii.

Figura 16) Filtrul EdgeDetection aplicat pe imaginea initiala.



Prin apasarea butonului **BTNLEFT** se va aplica un filtru Greyscale, urmand sa apasarea butonului **BTNRIGHT** pentru aplicarea unui filtru de Blur.

Figura 17) Blur aplicat dupa GreyScale pe imagine.



Prin apasarea butonului **BTNLEFT** se va aplica un filtru Greyscale, urmand sa apasarea butonului **BTNUP** pentru aplicarea unui filtru de EdgeDetection.

Figura 18) GreyScale + EdgeDetection aplicat pe imagine.



6. Concluzie

Proiectul a reusit sa implementeze diferite filtre pentru procesarea imaginilor, creand o interfata usor de folosit de catre orice utilizator. Documentarea ampla in domeniul “Procesarii de imagini” a avut un rol crucial in atingerea obiectivelor dorite, adunand cunostiinte necesare pentru implementarea algoritmilor precizati anterior.

Implementarea algoritmilor dintr-un punct de vedere hardware a fost un impediment pe care am putut sa il depasim doar prin folosirea cunostiintelor dobandite in facultate, intelegand foarte bine cum putem implementa o diagrama de stare, ce componente hardware sa folosim si cum sa le conectam.

O alta provocare pe care am intalnit-o pe parcursul dezvoltarii proiectului, a fost gestionarea memoriei oferite de placa Basys3, fiind nevoiti sa stocam multiple poze pe aceasta. Aceasta problema a fost rezolvata prin realizarea unor compromisuri cand a venit vorba despre dimensiunea pozei folosite, trebuind sa folosim o poza de dimensiunea 128x128.

Sincronizarea a avut un rol enorm in proiectul nostru, deoarece a trebuit sa tinem cont atat de sincronizarea componentelor cat si sincronizarea acestora cu rata de refresh a ecranului. Acest aspect ne-a fortat sa cream multiple diagrame de stare, adaugand diferite stari cu scopul de a sincroniza perfect tot proiectul, atat partea de procesare a imaginilor cat si afisarea pe ecranul VGA.

Pe langa interfata usor de folosit, proiectul hardware prezinta viteza mai mare fata de un program software obisnuit, componentele lucrând in paralel. Dezavantajul major al proiectului este faptul ca are o memorie RAM destul de mica ce nu ne permite sa folosim poze de dimensiuni mari.

Procesarea de imagini este o tehnica folosita in multiple domenii, proiectul nostru putând fiind implementat in proiecte mult mai mari :

- Detectarea benzilor (Lane detection) de catre o masina – Filtrul de Edge Detection este prezent in majoritatea algoritmilor complexi de Lane Detection.
- Aplicatii pentru telefonul mobil – Un telefon mobil poate sa foloseasca filtrele mentionate pentru editarea pozelor.
- Afisarea fisierelor media pe un ecran VGA – Daca proiectul este implementat pe o placa mult mai avansata, ce dispune de mult mai multa memorie, proiectul poate fi adaptat foarte usor in afisarea pozelor de dimensiuni mult mai mari sau chiar a unor videoclipuri.

Proiectul poate fi dezvoltat din multiple puncte de vedere :

- Folosirea unei placi mai bune pentru a stoca fisiere media mai mari.
- Implementarea altor filtre decat cele prezentate anterior.
- Realizarea procesarii de imagini pe imagini primite de catre o camera web (Real Time Processing).
- Folosirea unui alt tip de conexiune decat cea VGA

BIBLIOGRAFIE

[1] Prof.(Dr.) P.K Dash, Prof. Shashank Pujari, Miss. Sofia Nayak . “Implementation of Edge Detection Using Fpga & Model Based Approach” ICICES2014

[2] Nazma Nausheen, Ayan Seal, Pritee Khanna, Santanu Haldar, A FPGA based Implementation of Sobel Edge Detection, Microprocessors and Microsystems (2017)

[3] https://www.southampton.ac.uk/~msn/book/new_demo/sobel/

[4] <https://reference.digilentinc.com/learn/programmable-logic/tutorials/vga-display-controller/start>

[5] <https://projectf.io/posts/video-timings-vga-720p-1080p/>

[6] https://developer.apple.com/documentation/accelerate/blurring_an_image

[7] <https://resizeimage.net/>

[8] <https://xem.github.io/3DShomebrew/tools/image-to-bin.html>

[9] <https://wornwinter.wordpress.com/2015/02/07/coegen-v0-01-generate-coe-files-from-binary-files-for-xilinx-fpga-block-ram/>

[10] https://en.wikipedia.org/wiki/Xilinx_Vivado

ANEXA

Componenta Main

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
USE ieee.numeric_std.ALL;
```

entity Main is

Port (

btnL : std_logic;

btnR : std_logic;

btnC : std_logic;

btnU : std_logic;

clk:std_logic;

hsync:out std_logic;

vsync:out std_logic;

R:out std_logic_vector(3 downto 0);

G:out std_logic_vector(3 downto 0);

B:out std_logic_vector(3 downto 0)

);

end Main;

architecture Behavioral of Main is

constant PICTURE_SIZE : Integer:=16384;

constant PICTURE_WIDTH : Integer:= 128;

CONSTANT PICTURE_HEIGHT : Integer :=128;

```

--Semnale processing
    signal addr_pixel : std_logic_vector(13 downto 0);
    signal addr_vga : std_logic_vector(13 downto 0);

    signal sumax : integer :=0;
    signal sumay : integer :=0;

    signal sumar : integer :=0;
    signal sumab : integer :=0;
    signal sumag : integer :=0;
    --Semnale VGA
    signal douta : std_logic_vector(23 downto 0);

```

```

TYPE State_type IS(

```

```

idle,
r1_blur,
r2_blur,
r3_blur,
r4_blur,
r5_blur,
r6_blur,
r7_blur,
r8_blur,
r9_blur,
r1_grey,
scriere_grey,
decizie_grey,

```

```

r1_line,
r2_line,
r3_line,
r4_line,
r5_line,
r6_line,
r7_line,
r8_line,
r9_line,
scriere_line,
decizie_line,
scriere_blur,
scriere_blur2,
decizie_blur,
get_sum,
citire,
mutare,
inc,
citire_ROM,
mutare_ROM,
inc_rom,
stop);
SIGNAL Stare : State_Type;

```

```

signal Rst:std_logic := '0';

```

```

signal EnableVGA: std_logic := '0';

```

```

--Filter Matrix

```



```
signal mat1_blur : integer :=1;  
signal mat2_blur : integer :=2;  
signal mat3_blur : integer :=1;  
signal mat4_blur : integer :=2;  
signal mat5_blur : integer :=4;  
signal mat6_blur : integer :=2;  
signal mat7_blur : integer :=1;  
signal mat8_blur : integer :=2;  
signal mat9_blur : integer :=1;
```

```
signal mat11 : integer :=-1;  
signal mat12 : integer :=0;  
signal mat13 : integer :=1;  
signal mat14 : integer :=-2;  
signal mat15 : integer :=0;  
signal mat16 : integer :=2;  
signal mat17 : integer :=-1;  
signal mat18 : integer :=0;  
signal mat19 : integer :=1;
```

```
signal mat21 : integer :=-1;  
signal mat22 : integer :=-2;  
signal mat23 : integer :=-1;  
signal mat24 : integer :=0;  
signal mat25 : integer :=0;  
signal mat26 : integer :=0;  
signal mat27 : integer :=1;  
signal mat28 : integer :=2;
```

```
signal mat29 : integer :=1;
```

```
signal din_pixel : std_logic_vector(23 downto 0):=X"000000";
```

```
-- Flip Flop
```

```
signal DFlipFlopOut1: STD_LOGIC := '0';
```

```
signal DFlipFlopOut1_NOT: STD_LOGIC := '1';
```

```
signal ClockDiv4: STD_LOGIC := '0'; -- 25 MHz Clock
```

```
signal ClockDiv4_NOT: STD_LOGIC := '1';
```

```
component D_FlipFlop is
```

```
Port (
```

```
clk:in std_logic;
```

```
D:in std_logic;
```

```
Q:out std_logic
```

```
);
```

```
end component;
```

```
component VGA is
```

```
Port (
```

```
addr1 : in std_logic_vector(13 downto 0);
```

```
dout1 : out std_logic_vector(23 downto 0);
```

```
en : in std_logic;
```

```
clk:in std_logic; --25 MHz
```

```
hsync:out std_logic;
```

```
vsync:out std_logic;
```

```
R:out std_logic_vector(3 downto 0);
```

```

G:out std_logic_vector(3 downto 0);
B:out std_logic_vector(3 downto 0);
din1:in std_logic_vector(23 downto 0);
we_ram:in std_logic_vector(0 downto 0)
);
end component;
signal we_ram : std_logic_vector(0 downto 0) := "0";

```

----MEMoria in care modificam !

```

signal douta_mem : std_logic_vector (23 downto 0);
signal wea_mem : std_logic_vector ( 0 downto 0):="0";
signal pixel_citit : std_logic_vector(23 downto 0);

```

component mem_filter IS

```

PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNT0 0);
    addra : IN STD_LOGIC_VECTOR(13 DOWNT0 0);
    dina : IN STD_LOGIC_VECTOR(23 DOWNT0 0);
    douta : OUT STD_LOGIC_VECTOR(23 DOWNT0 0)
);

```

END component;

component debounce is

```

Port (
    DATA: in std_logic;
    CLK : in std_logic;
    OP_DATA : out std_logic);
end component ;

```

--ROMMUL CARE VA PASTRA IMAGINEA INITIALA !

component ROM IS

PORT (

clka : IN STD_LOGIC;

addra : IN STD_LOGIC_VECTOR(13 DOWNT0 0);

douta : OUT STD_LOGIC_VECTOR(23 DOWNT0 0)

);

END component;

--semnala rom

signal addr_ROM : std_logic_vector(13 downto 0);

signal Romout : std_logic_vector(23 downto 0);

signal initializare : std_logic := '0';

signal greyscale : integer := 0;

signal vvs : integer := 0;

signal TRESHOLD : integer := 4;

-- butons

signal BLeft : std_logic := '0';

signal BRight : std_logic := '0';

signal BCenter : std_logic := '0';

signal BDown : std_logic := '0';

begin

debouncer1: debounce port map (Data=>BtnL,CLK=>CLockDiv4,OP_DATA=>BLeft);

debouncer2: debounce port map (Data=>BtnR,CLK=>CLockDiv4,OP_DATA=>BRight);

debouncer3: debounce port map (Data=>BtnC,CLK=>CLockDiv4,OP_DATA=>BCenter);

debouncer4: debounce port map (Data=>BtnU,CLK=>CLockDiv4,OP_DATA=>BDown);

ROmul : ROM port map (CLKA =>ClockDiv4,Addr=>addr_pixel,douta=>RomOut);

Memorie: mem_filter port map (

clka => ClockDiv4,

wea=>wea_mem,

addra=>addr_pixel,

dina=>din_pixel,

douta=>douta_mem

);

DFlipFlopOut1_NOT<=not DFlipFlopOut1;

ClockDiv4_NOT<= not ClockDiv4;

--Pass Main 100 MHz clock to 2 cascaded DFlipFlops to divide frequency by 4. Result frequency= 25 MHz.

D1: D_FlipFlop Port map (clk=> clk, D=> DFlipFlopOut1_NOT, Q=>DFlipFlopOut1);

D2: D_FlipFlop Port map (clk=> DFlipFlopOut1, D=> ClockDiv4_NOT, Q=>ClockDiv4);

--VGA

VGA1: VGA Port map (addr1=>addr_vga,

din1 => din_pixel,

dout1 => douta,

en => EnableVGA,

clk => ClockDiv4,

hsync => hsync,

vsync => vsync,

R => R,

```

G => G,
B => B,
we_ram=>we_ram
);

--processing
proc1: process (ClockDiv4)
begin
if RISING_EDGE (ClockDiv4) then
case Stare is

when idle =>
-- addr_pixel<=(others=>'0'); Am modificat aici
EnableVga<='1';
wea_mem <="0";
we_ram<="0";
if (BLeft = '1') then
--face Convert to greyscale.
addr_pixel<= "00"&x"000";
addr_vga<= "00"&x"000";
Stare<=r1_grey;
EnableVga<='0';
elsif ( BRight = '1') then
--Face Blur
addr_pixel<= "00"&x"000";
addr_vga<= "00"&x"000";
Stare<=r1_blur;
EnableVga<='0';

```

```

elsif (BDown = '1' ) then
--Face edge detection
  addr_pixel<= "00"&x"000";
  addr_vga<= "00"&x"000";
  Stare<=r1_line;
  EnableVga<='0';
elsif ( BCenter = '1') then
--face reset.
  addr_pixel<= "00"&x"002";
  addr_vga<= "00"&x"000";
  EnableVga<='0';
  Stare<=citire_Rom;
else
  Stare<=idle;
end if;

```

```

when R1_blur =>
sumaR<= ( mat5_blur * to_integer(unsigned(douta(7 downto 4))));
sumaG<= ( mat5_blur * to_integer(unsigned(douta(15 downto 12))));
sumaB<= ( mat5_blur * to_integer(unsigned(douta(23 downto 20))));
  addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) -
PICTURE_WIDTH - 1),addr_pixel'length));
  addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) -
PICTURE_WIDTH - 1),addr_vga'length));
  Stare <= R2_blur;

```

```

when R2_blur =>
sumaR<= sumaR + ( mat1_blur * to_integer(unsigned(douta(7 downto 4))));
sumaG<= sumaG + ( mat1_blur * to_integer(unsigned(douta(15 downto 12))));

```

```

sumaB<= sumaB + ( mat1_blur * to_integer(unsigned(douta(23 downto 20))));
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));
Stare<=R3_blur;

```

when R3_blur =>

```

sumaR<= sumaR + ( mat2_blur * to_integer(unsigned(douta(7 downto 4))));
sumaG<= sumaG + ( mat2_blur * to_integer(unsigned(douta(15 downto 12))));
sumaB<= sumaB + ( mat2_blur * to_integer(unsigned(douta(23 downto 20))));
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));
Stare<=R4_blur;

```

when R4_blur =>

```

sumaR<= sumaR + ( mat3_blur * to_integer(unsigned(douta(7 downto 4))));
sumaG<= sumaG + ( mat3_blur * to_integer(unsigned(douta(15 downto 12))));
sumaB<= sumaB + ( mat3_blur * to_integer(unsigned(douta(23 downto 20))));
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
PICTURE_WIDTH - 2),addr_vga'length));
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
PICTURE_WIDTH - 2),addr_pixel'length));
Stare<=R5_blur;

```

when R5_blur =>

```

sumaR<= sumaR + ( mat4_blur * to_integer(unsigned(douta(7 downto 4))));
sumaG<= sumaG + ( mat4_blur * to_integer(unsigned(douta(15 downto 12))));
sumaB<= sumaB + ( mat4_blur * to_integer(unsigned(douta(23 downto 20))));

```



```

    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
2),addr_vga'length));

    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
2),addr_pixel'length));

    Stare<=R6_blur;

when R6_blur =>

    sumaR<= sumaR + ( mat6_blur * to_integer(unsigned(douta(7 downto 4))));
    sumaG<= sumaG + ( mat6_blur * to_integer(unsigned(douta(15 downto 12))));
    sumaB<= sumaB + ( mat6_blur * to_integer(unsigned(douta(23 downto 20))));
    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
PICTURE_WIDTH -2),addr_vga'length));

    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
PICTURE_WIDTH -2),addr_pixel'length));

    Stare<=R7_blur;

when R7_blur =>

    sumaR<= sumaR + ( mat7_blur * to_integer(unsigned(douta(7 downto 4))));
    sumaG<= sumaG + ( mat7_blur * to_integer(unsigned(douta(15 downto 12))));
    sumaB<= sumaB + ( mat7_blur * to_integer(unsigned(douta(23 downto 20))));
    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));

    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));

    Stare<=R8_blur;

when R8_blur =>

    sumaR<= sumaR + ( mat8_blur * to_integer(unsigned(douta(7 downto 4))));
    sumaG<= sumaG + ( mat8_blur * to_integer(unsigned(douta(15 downto 12))));
    sumaB<= sumaB + ( mat8_blur * to_integer(unsigned(douta(23 downto 20))));

```

```

    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));

    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));

    Stare<=R9_blur;

when R9_blur =>

    sumaR<= sumaR + ( mat9_blur * to_integer(unsigned(douta(7 downto 4))));
    sumaG<= sumaG + ( mat9_blur * to_integer(unsigned(douta(15 downto 12))));
    sumaB<= sumaB + ( mat9_blur * to_integer(unsigned(douta(23 downto 20))));

    --addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) -
Picture_WIDTH - 1),addr_pixel'length));

    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) - Picture_WIDTH -
1),addr_vga'length));

    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) - Picture_WIDTH
- 1),addr_pixel'length));

    Stare<=get_sum;

when get_sum =>

din_pixel <=

std_logic_vector(to_unsigned(sumab/(mat1_blur+mat2_blur+mat3_blur+mat4_blur+mat5_blur+
mat6_blur+mat7_blur+mat8_blur+mat9_blur),4))

    &"0000"

    &
std_logic_vector(to_unsigned(sumag/(mat1_blur+mat2_blur+mat3_blur+mat4_blur+mat5_blur+
mat6_blur+mat7_blur+mat8_blur+mat9_blur),4))

    &"0000"

    &
std_logic_vector(to_unsigned(sumar/(mat1_blur+mat2_blur+mat3_blur+mat4_blur+mat5_blur+
mat6_blur+mat7_blur+mat8_blur+mat9_blur),4))

    &"0000";

    Stare<=scriere_blur;

```

```

when scriere_blur =>
    wea_mem <="1";
    stare <= scriere_blur2;

when scriere_blur2 =>
    stare <= decizie_blur;

when decizie_blur =>
    wea_mem <="0";
    we_ram<="0";
    if ( to_integer(unsigned(addr_vga)) = Picture_size - 1)then
        Stare<=citire;
        addr_pixel <="00"&x"002";
        addr_vga <="00"&x"000";
    else
        Stare<=r1_blur;
        addr_vga <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga)) +
1,addr_vga'length));
        addr_pixel <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel)) +
1,addr_pixel'length));
    end if;

when R1_grey =>
    greyscale <= (to_integer(unsigned((douta(7 downto 4))))+to_integer(unsigned((douta(15
downto 12))))+to_integer(unsigned((douta(23 downto 20)))))/3;
    wea_mem <="1";
    Stare<=scriere_Grey;

when scriere_grey =>
    din_pixel <=

```

```

std_logic_vector(to_unsigned(greyscale,4))
&"0000"
& std_logic_vector(to_unsigned(greyscale,4))
&"0000"
& std_logic_vector(to_unsigned(greyscale,4))
&"0000";
Stare<=decizie_grey;

when decizie_grey =>
    wea_mem <="0";
    we_ram<="0";
    if ( to_integer(unsigned(addr_pixel)) = (Picture_Size-1))then
        Stare<=citire;
        addr_pixel <="00"&x"002";
        addr_vga <="00"&x"000";
    else
        Stare<=r1_grey;
        addr_pixel <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel)) +
1,addr_pixel'length));
        addr_vga <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga)) +
1,addr_vga'length));
    end if;

when R1_line =>
    sumax<= ( mat15 * to_integer(unsigned(douta(7 downto 4))));
    sumay<= ( mat25 * to_integer(unsigned(douta(7 downto 4))));
    addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) -
PICTURE_WIDTH - 1),addr_pixel'length));
    addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) -
PICTURE_WIDTH - 1),addr_vga'length));

```

```
Stare<=R2_line;
```

```
when R2_line =>
```

```
sumax<=sumax+ ( mat11 * to_integer(unsigned(douta(7 downto 4))));
```

```
sumay<=sumay+ ( mat21 * to_integer(unsigned(douta(7 downto 4))));
```

```
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +  
1),addr_pixel'length));
```

```
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +  
1),addr_vga'length));
```

```
Stare<=R3_line;
```

```
when R3_line =>
```

```
sumax<=sumax+ ( mat12 * to_integer(unsigned(douta(7 downto 4))));
```

```
sumay<=sumay+ ( mat22 * to_integer(unsigned(douta(7 downto 4))));
```

```
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +  
1),addr_pixel'length));
```

```
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +  
1),addr_vga'length));
```

```
Stare<=R4_line;
```

```
when R4_line =>
```

```
sumax<=sumax+ ( mat13 * to_integer(unsigned(douta(7 downto 4))));
```

```
sumay<=sumay+ ( mat23 * to_integer(unsigned(douta(7 downto 4))));
```

```
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +  
PICTURE_WIDTH - 2),addr_pixel'length));
```

```
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +  
PICTURE_WIDTH - 2),addr_vga'length));
```

```
Stare<=R5_line;
```

```
when R5_line =>
```

```
sumax<=sumax+ ( mat14 * to_integer(unsigned(douta(7 downto 4))));
```

```

sumay<=sumay+ ( mat24 * to_integer(unsigned(douta(7 downto 4))));

addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
2),addr_pixel'length));

addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
2),addr_vga'length));

Stare<=R6_line;

when R6_line =>

sumax<=sumax+ ( mat16 * to_integer(unsigned(douta(7 downto 4))));
sumay<=sumay+ ( mat26 * to_integer(unsigned(douta(7 downto 4))));

addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
PICTURE_WIDTH -2),addr_pixel'length));

addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
PICTURE_WIDTH -2),addr_vga'length));

Stare<=R7_line;

when R7_line =>

sumax<=sumax+ ( mat17 * to_integer(unsigned(douta(7 downto 4))));
sumay<=sumay+ ( mat27 * to_integer(unsigned(douta(7 downto 4))));

addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));

addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));

Stare<=R8_line;

when R8_line =>

sumax<=sumax+ ( mat18 * to_integer(unsigned(douta(7 downto 4))));
sumay<=sumay+ ( mat28 * to_integer(unsigned(douta(7 downto 4))));

addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) +
1),addr_pixel'length));

addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) +
1),addr_vga'length));

```

```

Stare<=R9_line;

when R9_line =>
sumax<= sumax + ( mat19 * to_integer(unsigned(douta(7 downto 4))));
sumay<=sumay +( mat29 * to_integer(unsigned(douta(7 downto 4))));
addr_pixel<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel) - Picture_WIDTH
- 1),addr_pixel'length));
addr_vga<= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga) - Picture_WIDTH -
1),addr_vga'length));
Stare<=scriere_line;
wea_mem <="1";

when scriere_line =>
if( sumax> treshold or sumax < - treshold) then
    if ( sumay> treshold or sumay < - treshold) then

din_pixel <=
std_logic_vector(to_unsigned((sumax+sumay)/2,4))
&"0000"
& std_logic_vector(to_unsigned((sumax+sumay)/2,4))
&"0000"
& std_logic_vector(to_unsigned((sumax+sumay)/2,4))
&"0000";

else

din_pixel <=
std_logic_vector(to_unsigned(sumax,4))
&"0000"
& std_logic_vector(to_unsigned(sumax,4))

```

```

&"0000"
& std_logic_vector(to_unsigned(sumax,4))
&"0000";
end if;

elsif ( sumay> treshold or sumay < - treshold) then

din_pixel <=
std_logic_vector(to_unsigned(sumay,4))
&"0000"
& std_logic_vector(to_unsigned(sumay,4))
&"0000"
& std_logic_vector(to_unsigned(sumay,4))
&"0000";

else
    din_pixel<=x"000000";
end if;

stare <= decizie_line;

when decizie_line =>
    wea_mem <="0";
    we_ram<="0";
    if ( to_integer(unsigned(addr_pixel)) = (Picture_Size-1))then
        Stare<=citire;
        addr_pixel <="00"&x"002";
        addr_vga <="00"&x"000";
    else
        Stare<=r1_line;
    end if;
end when;

```



```
    addr_pixel <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel)) +
1,addr_pixel'length));
```

```
    addr_vga <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga)) +
1,addr_vga'length));
```

```
end if;
```

```
when citire =>
```

```
pixel_citit<=douta_mem;
```

```
Stare<=mutare;
```

```
when mutare =>
```

```
din_pixel<=pixel_citit;
```

```
we_ram<="1";
```

```
Stare<=inc;
```

```
when inc =>
```

```
if ( not(addr_pixel = ("00"&x"001")))
```

```
    addr_vga <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga)) +
1,addr_vga'length));
```

```
    addr_pixel <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel)) +
1,addr_pixel'length));
```

```
    Stare<=citire;
```

```
    we_ram<="0";
```

```
else
```

```
Stare<=stop;
```

```
we_ram<="0";
```

```
addr_pixel<=(others=>'0');
```

```
addr_vga<=(others=>'0');
```

```
end if;
```

```

when citire_ROM =>
    pixel_citit<=RomOut;
    Stare<=mutare_rom;

    when mutare_rom =>
        din_pixel<=pixel_citit;
        we_ram<="1";
        Stare<=inc_rom;

    when inc_rom =>
        if ( not(addr_pixel = ("11"&x"fff")))then
            addr_vga <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_vga)) +
1,addr_vga'length));
            addr_pixel <= std_logic_vector(to_unsigned(to_integer(unsigned(addr_pixel)) +
1,addr_pixel'length));
            Stare<=citire_rom;

            we_ram<="0";
        else
            Stare<=stop;
            we_ram<="0";
            addr_pixel<=(others=>'0');
            addr_vga<=(others=>'0');
        end if;

when stop =>
    stare<=idle;

```

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

```
end Behavioral;
```

Componenta VGA

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
  
entity VGA is  
Port (  
    addr1 : in std_logic_vector(13 downto 0);  
    dout1 : out std_logic_vector(23 downto 0);  
    din1 : in std_logic_vector(23 downto 0);  
    en : in std_logic;  
    clk:in std_logic; --25 MHz  
    hsync:out std_logic;  
    vsync:out std_logic;  
    R:out std_logic_vector(3 downto 0);  
    G:out std_logic_vector(3 downto 0);  
    B:out std_logic_vector(3 downto 0);  
    we_ram:in std_logic_vector(0 downto 0)  
);  
end VGA;
```

architecture Behavioral of VGA is

--Block RAM

signal addra : STD_LOGIC_VECTOR(13 DOWNT0 0):=(others=>'0');

signal dina : STD_LOGIC_VECTOR(23 DOWNT0 0):=(others=>'0');

signal douta : STD_LOGIC_VECTOR(23 DOWNT0 0):=(others=>'0');

-- Constants

constant PICTURE_SIZE : Integer:=16384; -- 300 Pixels* 300 Pixels picture= 90000 Pixels

constant PICTURE_WIDTH : Integer:= 128;

CONSTANT PICTURE_HEIGHT : Integer :=128;

signal H_DISPLAY:integer := 639; -- horizontal length

signal H_L_BORDER:integer := 48; -- horizontal left border

signal H_R_BORDER:integer := 16; -- horizontal right border

signal H_RETRACE:integer := 96; -- horizontal retrace

signal V_DISPLAY:integer := 479; -- vertical length

signal V_T_BORDER:integer := 10; -- vertical top border

signal V_B_BORDER:integer := 33; -- vertical bottom border

signal V_RETRACE:integer := 2; -- vertical retrace

signal vPos:integer := 0;

signal hPos:integer := 0;

signal videoOn:std_logic := '0';

component D_FlipFlop is

Port (

clk:in std_logic;

D:in std_logic;

Q:out std_logic

);

end component;

component blk_mem_gen_0 IS

PORT (

clka : IN STD_LOGIC;

wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);

addra : IN STD_LOGIC_VECTOR(13 DOWNTO 0);

dina : IN STD_LOGIC_VECTOR(23 DOWNTO 0);

douta : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)

);

END component;

begin

--Block RAM containing picture

U3: blk_mem_gen_0 port map (clka=>clk, wea=>we_ram, addra=>addra, dina=>din1,
douta=>douta);

Horizontal_position_counter:process(clk)

begin

```

if(en='1')then
if(clk'event and clk = '1')then
if (hPos = (H_DISPLAY + H_R_BORDER + H_RETRACE + H_L_BORDER)) then
hPos <= 0;
else
hPos <= hPos + 1;
end if;
end if;
else
hPos <= 0;
end if;
end process;

```

```

Vertical_position_counter:process(clk, hPos)
begin
if(en='1')then
if(clk'event and clk = '1')then
if(hPos = (H_DISPLAY + H_R_BORDER + H_L_BORDER + H_L_BORDER))then
if (vPos = (V_DISPLAY + V_T_BORDER + V_RETRACE + V_B_BORDER)) then
vPos <= 0;
else
vPos <= vPos + 1;
end if;
end if;
end if;
else
vPos <= 0;
end if;
end process;

```

```
Horizontal_Synchronisation:process(clk, hPos)
```

```
begin
```

```
if(en='1')then
```

```
if(clk'event and clk = '1')then
```

```
if((hPos <= (H_DISPLAY + H_R_BORDER)) OR (hPos > H_DISPLAY + H_R_BORDER +  
H_RETRACE))then
```

```
HSYNC <= '1';
```

```
else
```

```
HSYNC <= '0';
```

```
end if;
```

```
end if;
```

```
end if;
```

```
end process;
```

```
Vertical_Synchronisation:process(clk, vPos)
```

```
begin
```

```
if(en='1')then
```

```
if(clk'event and clk = '1')then
```

```
if((vPos <= (V_DISPLAY + V_T_BORDER)) OR (vPos > V_DISPLAY + V_T_BORDER +  
V_RETRACE))then
```

```
VSYNC <= '1';
```

```
else
```

```
VSYNC <= '0';
```

```
end if;
```

```
end if;
```

```
end if;
```

```
end process;
```

```
video_on:process(clk, hPos, vPos)
```

```

begin
if(en ='1')then
if(clk'event and clk = '1')then
if(hPos <= H_DISPLAY and vPos <= V_DISPLAY)then
videoOn <= '1';
else
videoOn <= '0';
end if;
end if;
end if;
end process;

```

```

dout1<=douta;

```

```

draw:process(clk, hPos, vPos, videoOn,en)

```

```

begin
if(en ='1')then
if(clk'event and clk = '1')then
if(videoOn = '1')then
    if (hpos <= picture_width and vpos < picture_height) then
        B<=douta(23 downto 20); G<=douta(15 downto 12); R<=douta(7 downto 4);
        if(to_integer(unsigned(addr)) = Picture_size-1) then
            addr <= "0000000000000000";
        else
            addr<=STD_LOGIC_VECTOR(unsigned(addr)+1);
        end if;
    else
        R<=(others=>'0');G<=(others=>'0');B<=(others=>'0');
    end if;
end if;
end process;

```



```

else
R<=(others=>'0');G<=(others=>'0');B<=(others=>'0');
end if;
end if;
else
    addra <= addr1;
end if;
end process;

end Behavioral;

```

Componenta D_Flip-Flop

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity D_FlipFlop is
Port (
clk:in std_logic;

```

```

D:in std_logic;
Q:out std_logic
);
end D_FlipFlop;

architecture Behavioral of D_FlipFlop is
signal aux:std_logic := '0';
begin
process(clk)
begin
if (clk'event and clk='1') then
    aux <= D;
end if;
end process;

Q <= aux;

end Behavioral;

```

Componenta Debouncer

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity debounce is
Port (
DATA: in std_logic;

```

```
CLK : in std_logic;  
OP_DATA : out std_logic);  
end debounce ;
```

architecture Behavioral of debounce is

```
signal Q1, Q2, Q3 : std_logic;
```

```
begin
```

```
process(CLK)
```

```
begin
```

```
    if (CLK'event and CLK = '1') then
```

```
        Q1 <= DATA;
```

```
        Q2 <= Q1;
```

```
        Q3 <= Q2;
```

```
    end if;
```

```
end process;
```

```
OP_DATA <= Q1 and Q2 and (not Q3);
```

```
end Behavioral;
```