Department of Computer Science
Technical University of Cluj-Napoca

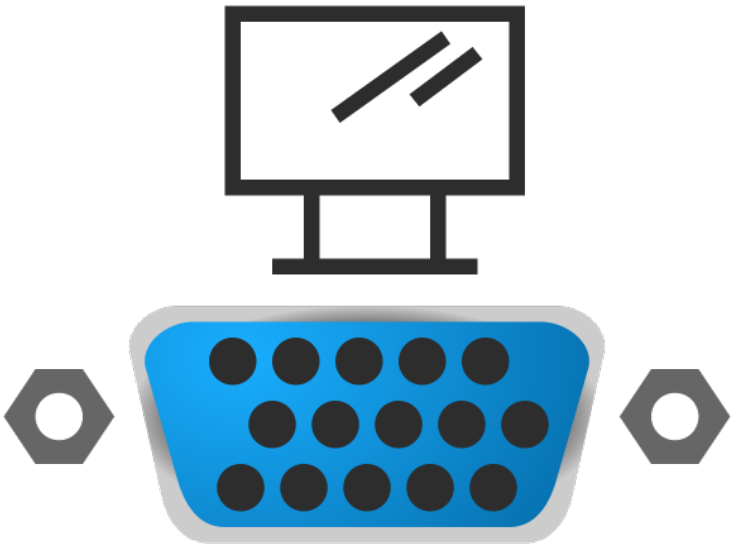
Structura Sistemelor de Calcul

Laboratory activity

Utilizarea portului VGA al plăcii Nexys 4 DDR pentru afișarea unor imagini și procesarea acestora.

Nume: Trufin Radu-Sebastian
Grupa: 30233
Email: trufin.radu@yahoo.com

Nume: Vîjaică Paul-Augustin
Grupa: 30233
Email: augustinvijaica@gmail.com



Data : 11/1/2021
Îndurmator : Fati Daniela

Cuprins

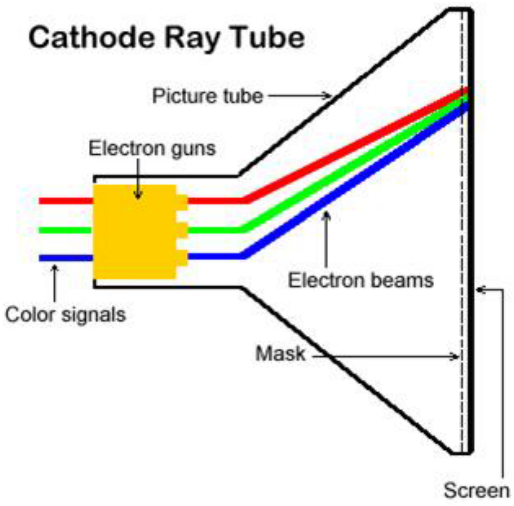
$***$

Introducere

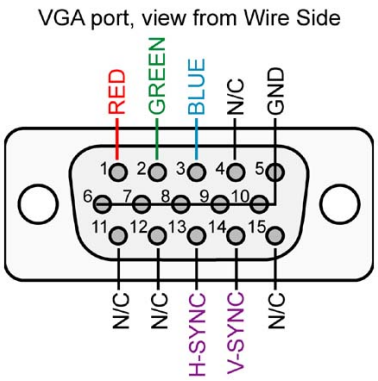
Acest document are ca scop general rezolvarea și implementarea problemei afișării unei imagini folosind portul VGA al plăcii Nexys 4 DDR folosind mediul de proiectare Vivado. Învățarea modului de a conduce direct un monitor VGA cu un FPGA deschide o fereastră pentru multe proiecte potențiale: jocuri video, procesare de imagini, o fereastră de terminal pentru un procesor personalizat și multe altele. Importanța și necesitatea procesării imaginilor digitale provine din două domenii principale de aplicare: prima fiind îmbunătățirea informațiilor picturale pentru interpretarea umană și a doua fiind prelucrarea datelor unei scene pentru o percepție autonomă a mașinii. Prelucrarea digitală a imaginii are o gamă largă de aplicații, cum ar fi teledetecția, stocarea imaginilor și a datelor pentru transmiterea în aplicații de afaceri, imagistica medicală si acustica. Astfel s-a ales ca extensie in cadrul proiectului posibilitatea de a procesa imaginea generata pe monitorul VGA. Utilizatorul poate schimba modul de vizualizare al imaginii generate cu ajutorul comutatoarelor plăcii. Modurile implementate pentru procesarea imaginii sunt următoarele : efect negativ, iluminare, întunecare și vizualizarea imaginii în format binar. Se va explica în detaliu funcționarea monitoarelor VGA și principiul afișării unei imagini pe aceste monitoare. Pentru realizarea acestui lucru vor trebui studiati pinii corespunzatori și realizată sincronizarea unor semnale importante. Componentele proiectului se introduc folosind librăriile din proprietatea intelectuală (IP) realizată de Xilinx. Blocul de memorie din cadrul proiectului folosește matricea generată de o funcție din Matlab, care primește ca input o imagine și returnează ca output valorile RGB corespunzatoare fiecărui pixel. Folosirea unui astfel de bloc de memorie ușurează munca programatorului și reduce compelxitatea codului și implicit a proiectului. În secțiunile următoare se descrie standardul VGA și modul său de funcționare, după care se detaliază componentele din cadrul proprietății intelectuale și atributele acestora. Apoi se parcurge codul Matlab, respectiv VHDL folosit pentru implementarea soluției. Obiectivul documentului este afișarea și procesarea pe un monitor VGA al următoarei imagini (pisica **Yuki**).

Fundamentare teoretică

Standardul VGA (Video Graphics Array) este un sistem analogic de transmisie a imaginii prin intermediul a trei componente (RGB). Din acest motiv, standardul este supus interferențelor care au loc mai ales în cazul transmisiilor la distanțe mai mari - față de corespondentele digitale ale acestuia, adică DVI sau HDMI. Standard VGA a apărut în anul 1987 pentru plăcile grafice instalate în calculatoarele de tip IBM PC. În prezent, standardul VGA se utilizează pentru specificația rezoluției 640×480 pixeli, indiferent de numărul de culori. Acest standard a fost realizat inițial pentru monitoarele cu tub catodic. Tubul cu raze catodice (CRT) este un tub de vid care conține unul sau mai multe tunuri electronice și un ecran fosforescent și este utilizat pentru a afișa imagini. Acesta modulează, accelerează și deviază fasciculul de electroni pe ecran pentru a crea imaginile. Imaginile pot reprezenta forme de undă electrice (osciloscop), imagini (televizor, monitor de calculator), ținte radar sau alte fenomene.

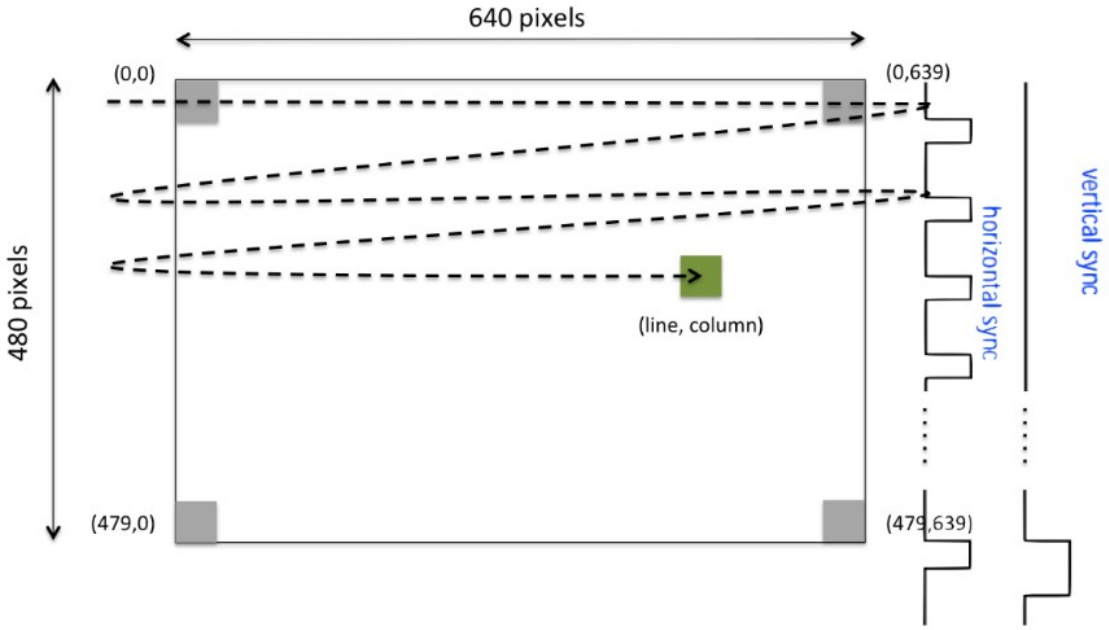


Un conector VGA prezintă 15 pini dinte care 6 sunt pini de împământare (GND) iar 4 dintre pini (4, 11, 12, 15) sunt folosiți rareori pentru comunicarea seriala a monitorului privind rezoluția, dar sunt toti opționali. Cei 5 pini care se studiează in detaliu sunt red, green, blue, **hsync** (horizontal synchronization) și **vsync** (vertical synchronization).



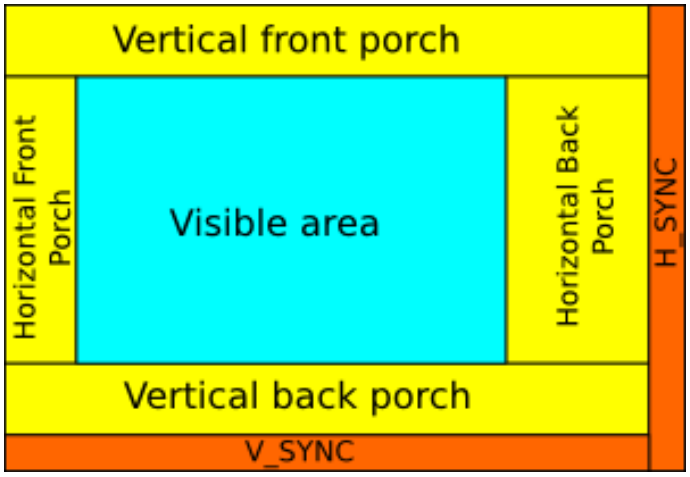
Fiecare pixel de pe monitor are o componentă de culoare roșie, verde și albastră. Semnalele de culoare roșu, verde și albastru de la portul VGA la monitor sunt analogice și sunt generate de convertoarele Digital to Analog cu rezistență de pe placa FPGA care primesc intrarea de la liniile dedicate I/O FPGA. Sistemul de culori VGA folosește palete bazate pe registre pentru a mapa culorile în diferite adâncimi de biți la gama sa de ieșire pe 18 biți.

Porturile VGA de plăcă utilizează două semnale de sincronizare, unul numit hsync (sincronizare orizontală) pentru a specifica timpul necesar scanării printr-un rând de pixeli, iar celălalt numit vsync (sincronizare verticală) pentru a specifica timpul necesar scanării prin întregul ecran cu rândurile sale de pixeli. Conducerea corectă a semnalelor hysnc și vsync este o sarcina de implementat.



Zona de afișare de pe ecran utilizând VGA are o rezoluție de 640 x 480 pixeli. Originea zonei de afișare este **(0,0)** și se află în colțul din stânga sus. Dimensiunea x crește de la stânga la dreapta, în timp ce y crește de sus în jos. Se ruleaza circuitul VGA la o rată de pixeli de 25 MHz, ceea ce înseamnă că 25 de milioane de pixeli vor fi scanati în fiecare secundă. Monitoarele includ, de obicei, mici margini negre care înconjoară zona de afișare.

Se împarte fiecare rând de scanare orizontală în **800 pixeli**, începând de la pixelul 0 și terminând la pixelul 799. Se scaneaza prin fiecare rând orizontal de pixeli la 25 MHz. Numărul de pixeli începe la începutul zonei de afișare și trece prin toți cei 640 de pixeli afișați, urmat de o margine dreaptă de 16 pixeli. Urmează o retragere de 96 de pixeli, care este de fapt o întârziere a monitoarelor CRT pe care trebuie să o includem și, în cele din urmă, o margine stânga de 48 de pixeli.



Fiecare linie a ecranului începe cu o regiune video activă, în care valorile (RGB) sunt afișate pentru fiecare pixel din linie. Regiunea activă este urmată de o regiune de golire, în care se transmit pixeli negri. În mijlocul intervalului de golire, se transmite un impuls de sincronizare orizontală. Intervalul de golire înainte de pulsul de sincronizare este cunoscut sub numele de '**front porch**', iar intervalul de golire după pulsul de sincronizare este cunoscut sub numele de '**back porch**'.

Zona vizibilă reprezintă 640 x 480 pixeli, dimensiunea totală fiind **800** (640 + 16 + 96 + 48) (Horizontal Display + Front Porch + Retrace + Back Porch) pe orizontală și **525** (480 + 10 + 2 + 33) (Vertical Display Front Porch + Retrace + Back Porch) pe verticală. Aceste valori au fost luate de pe site-ul auxiliar:

`http://tinyvga.com/vga-timing/640x480@60Hz`

Pe acest site auxiliar se prezintă valorile de sincronizare ale constantelor menționate pentru diverse rezoluții ale monitoarelor și se specifică faptul că polaritatea semnalului de sincronizare orizontală este negativă iar cea a semnalului de sincronizare verticală este pozitivă. În continuare se prezintă un tabel sugestiv care conține și valorile timpilor acestor intervale.

Horizontal timing (line)

Scanline part	Pixels	Time [µs]
Active video	640	25.422
Front porch	16	0.636
Sync pulse	96	3.813
Back porch	48	1.907
Whole line	800	31.778

Vertical timing (frame)

Frame part	Pixels	Time [µs]
Active video	480	15.253
Front porch	10	0.318
Sync pulse	2	0.0636
Back porch	33	1.049
Whole frame	525	16.683

Ecranele VGA moderne acceptă rezoluții multiple de afișare, iar controlerul VGA dictează rezoluție prin producerea de semnale de sincronizare pentru a controla modelele raster. Datele video provin de obicei dintr-o memorie de reîmprospătare video cu unul sau mai mulți octeți atribuiți fiecărui pixel locație. Controlerul prelucrează apoi și aplică date video pe afișaj exact în momentul în care fasciculul de electroni se deplasează pe un pixel dat. Controlerul VGA generează sincronizarea orizontală (HS) și sincronizarea verticală (VS) sincronizează semnalele și coordonează livrarea datelor video pe fiecare ceas de pixeli. Ceasul pixel definește timpul disponibil pentru afișarea unui pixel de informații. Semnalul VS definește frecvența de reîmprospătare a afișajului sau frecvența la care toate informațiile de pe afișaj sunt redesenate. Frecvența minimă de reîmprospătare este o funcție a intensității afișajului de fosfor și fascicul de electroni, cu frecvențe practice de reîmprospătare în intervalul 60 Hz - 120 Hz. Numărul de linii orizontale afișat la o anumită frecvență de reîmprospătare definește frecvența de revenire orizontală.

Proiectare și implementare

Proprietatea intelectuală (IP) se referă la funcțiile logice preconfigurate care pot fi utilizate în proiectare. Xilinx oferă o selecție largă de IP care este optimizată pentru FPGA Xilinx. Acestea pot include funcții livrate prin intermediul software-ului Xilinx CORE Generator, prin Xilinx Architecture Wizard sau prin System Generator. Xilinx și companiile sale partenere produc IP care variază în complexitate, de la simpli operatori aritmetici și elemente de întârziere până la blocuri complexe la nivel de sistem, precum filtre de procesare digitală a semnalului (DSP), multiplexoare, transformatoare și memorie. Xilinx IP este livrat prin următoarele instrumente și mecanisme:

CORE Generator IP : Software-ul Xilinx CORE Generator creează versiuni parametrizabile ale IP-ului 'soft' predefinit optimizat pentru FPGA-urile Xilinx. CORE Generator IP include memorii și FIFO, precum și procesare digitală a semnalului (DSP), functii matematice, interfețe standard de magistrale, logică standard și funcții de rețea.

Architecture Wizard IP : Arhitect Wizard IP pentru arhitectura Xilinx configurează caracteristici și module arhitecturale 'dure' FPGA, cum ar fi digital clock managers (DCM) sau blocurile DSP48. Architecture Wizard poate crea cu ușurință configurații care altfel ar putea necesită scrierea unui set mare de constrângeri sau atribute HDL.

Xilinx oferă un nucleu generator de memorie de bloc flexibil pentru a crea memorii compacte, de înaltă performanță, care rulează până la 450 MHz Generatorul de memorie Block LogiCORE IP automatizează crearea de memorii de blocuri optimizate pentru resurse și putere pentru FPGA Xilinx. Disponibil prin sistemul ISE Design Suite CORE Generator System, nucleul permite utilizatorilor să creeze funcții de memorie bloc pentru a se potrivi cu o varietate de cerințe. Cunoștințele încorporate despre arhitecturile dispozitivelor Xilinx îi permit să utilizeze caracteristicile arhitecturale FPGA specializate pentru a crea cea mai compactă soluție de înaltă performanță sau cu putere redusă.

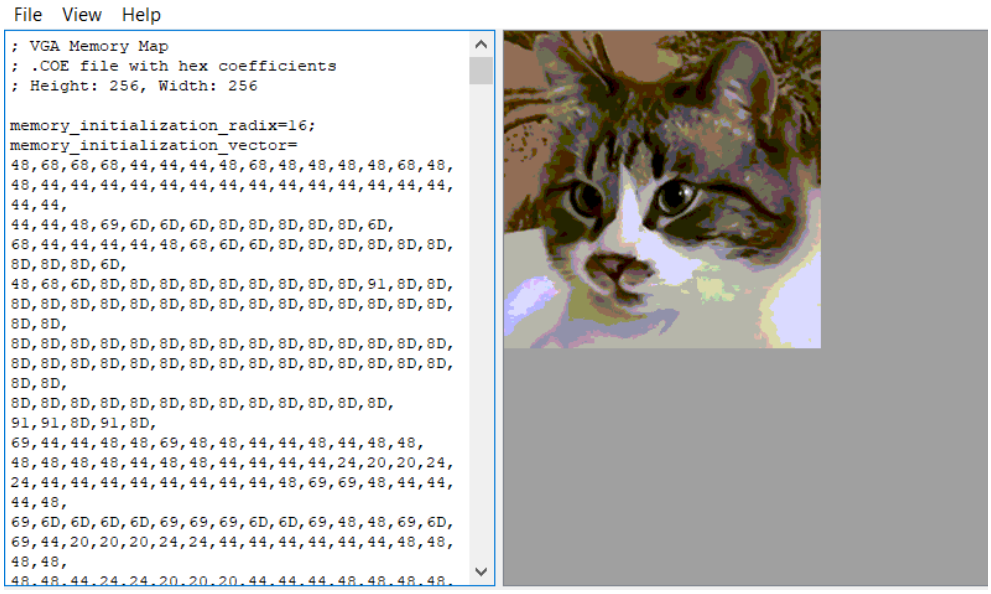
Unele nuclee IP, de exemplu, FIR Compiler, necesită specificarea mai multor valori de date, cum ar fi coeficienții de filtrare (FIR Compiler). Pentru a specifica valorile pentru aceste module, trebuie să se încarce un fișier cu extensia COE atunci când se creează nucleul IP. Se specifică locația făcând click pe butonul Load Coefficients din GUI de personalizare a nucleului IP. Când se efectuează click pe buton, apare o casetă de dialog a browserului pentru a încărca un fișier COE. Fișierul coe conține radacina (radix), lățimea coeficientului și coeficienții filtrului. Fișierul raportează coeficienții filtrului în ordine. Raza, lățimea coeficientului și coeficienții filtrului sunt setul minim de date necesare într-un astfel de tip. Pentru generarea unui astfel de fișier se utilizează o funcție auxiliară creată in Matlab. Se va crea in mediul

Vivado un bloc ROM folosind Core Generator unde datele inițiale sunt stocate într-un fișier .coe. Putem converti imaginea jpg în fișier într-un fișier corespunzător .coe numit utilizând funcția Matlab IMG2coe8 (imgfile, outfile) Această funcție este compatibila și cu alte formate de imagine standard, cum ar fi bmp, gif și tif. Fișierul .coe produs de această funcție conține un octet de 8 biți pentru fiecare pixel al imaginii cu formatul :

$$\text{color byte} = [\text{R2,R1,R0,G2,G1,G0,B1,B0}]$$

Imaginea originală citită în funcția Matlab va conține 8 biți de roșu, 8 biți de verde și 8 biți de albastru. Octetul de culoare pe 8 biți stocat în fișierul .coe va conține doar 3 biți superiori de roșu, 3 biți superiori de verde și 2 biți superiori de albastru. Acest lucru trebuie realizat, deoarece majoritatea plăcilor acceptă doar VGA pe 8 biți de culoare. Imaginea color rezultată pe 8 biți va fi de calitate redusă comparativ cu imaginea originală.

Funcția IMG2coe8 primește numele imaginii ca parametru și numele fișierului rezultat ce va conține matricea de octeți (RGB) și extensia COE. Se scriu în fișier primele două linii care inițializează memoria și se prelucrează imaginea. size(img) este un triplet ce reprezintă [înălțimea, lungimea, componenta RGB] a unei poze. Se parcurge fiecare pixel al imaginii de dimensiune height x width și se calculează intensitatea fiecărei culori pentru fiecare pixel. Se construiește un octet (Outbyte) care este format din primii 3 biți ai componentei roșii, primii 3 biti ai componentei verzi și primii 2 biti ai componentei albastre. Acest octet se scrie în fișierul de output. Adițional se afișează ';' în momentul în care se indică sfârșitul parcurgerii și se trece la o nouă linie din 32 in 32 de elemente, ceea ce inseamnă că fișierul final cu extensia COE va avea $(256 \cdot 256)/32 + 2$ (liniile inițiale) adică 2050 de linii, în cazul în care dimensiunea imaginii este de 256 x 256 pixeli. Output-ul pentru imaginea inițiala 'yukicat' va fi de forma următoare. Pentru revizualizarea imaginii reduse în calitate, se folosește resursa auxilia **coetool**, programata in Python.



coetool este un program gratuit GUI util pentru a converti din fișiere .coe (hartă de memorie VGA pentru un ROM într-un FPGA) în fișiere imagine și invers. De asemenea, este posibil să se mărească imaginile și să se exporteze atât fișierul .coe cât și imaginea. În continuare se prezintă codul Matlab al funcției IMG2coe8 :

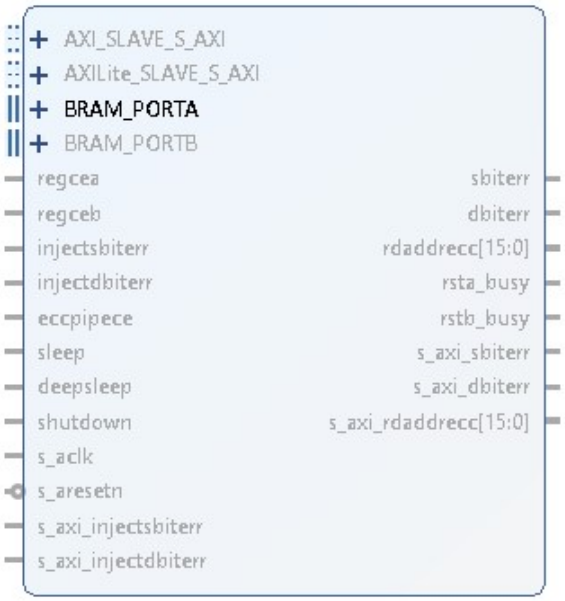
```
function img2 = IMG2coe8(imgfile, outfile)

imgfile = 'yukicat.jpg';
outfile = 'yukicat.coe';

img = imread(imgfile);
height = size(img, 1);
width = size(img, 2);

s = fopen(outfile,'wb');

fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);

fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');

cnt = 0;
img2 = img;

for r = 1:height
    for c = 1:width
        cnt = cnt + 1;
        R = img(r,c,1);
        G = img(r,c,2);
        B = img(r,c,3);
        Rb = dec2bin(double(R),8);
        Gb = dec2bin(double(G),8);
        Bb = dec2bin(double(B),8);
        img2(r,c,1) = bin2dec([Rb(1:3) '00000']);
        img2(r,c,2) = bin2dec([Gb(1:3) '00000']);
        img2(r,c,3) = bin2dec([Bb(1:2) '000000']);
        Outbyte = [Rb(1:3) Gb(1:3) Bb(1:2)];

        if (Outbyte(1:4) == '0000')
            fprintf(s,'0%X',bin2dec(Outbyte));
        else
            fprintf(s,'%X',bin2dec(Outbyte));
        end
        if ((c == width) && (r == height))
            fprintf(s,'%c',';');
        else
            if (mod(cnt,32) == 0)
                fprintf(s,'%c\n',',');
            else
                    fprintf(s,'%c',',');
            end
        end
    end
end

fclose(s);
```

Memoria bloc IP Xilinx LogiCORE (Block Memory Generator) este un constructor de memorii avansat care generează memorii performante folosind resurse RAM de bloc încorporate în Xilinx FPGA. Nucleul BMG acceptă atât interfețe Native, cât și AXI4. Configurarea Block Memory Generator a interfeței AXI4 este derivată din interfața nativă BMG și adaugă un standard industrial (interfață de protocol de magistrală). Două AXI4 sunt disponibile ca stil de interfață: AXI4 și AXI4-Lite. Pentru stocarea informațiilor din imagine, se va folosi o memorie cu interfață nativă. În continuare se prezintă câteva proprietăți ale acestei interfețe:

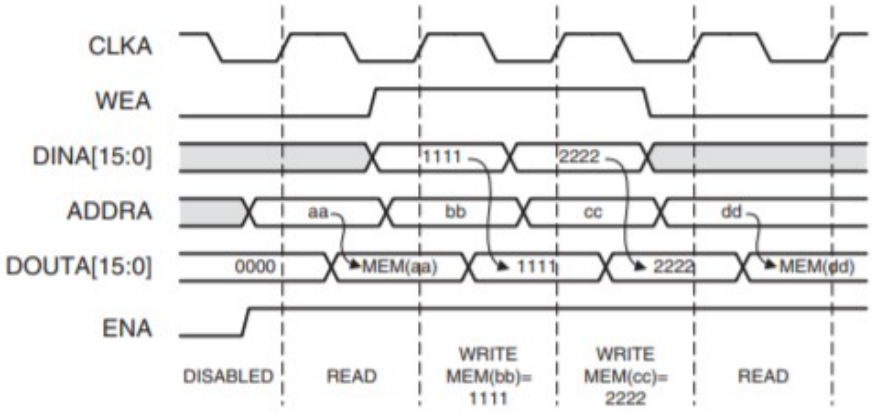
- Generează RAM/ROM cu un singur port sau RAM/ROM cu două porturi simple.
- Suporta dimensiuni de până la maximum 16 megaocteti. (16MB).
- Suportă capacitatea de corectare a erorilor Hamming (ECC).
- Performanță de până la 450 MHz.

Nucleul configurează blocul primitivelor RAM și le conectează împreună folosind unul dintre următorii algoritmi:

- Algoritm de arie minimă: memoria este generată utilizând numărul minim de blocuri de primitive RAM. Sunt utilizați atât biți de date, cât și biți de paritate.
- Algoritm de putere redusă: memoria este generată astfel încât numărul minim de blocuri de primitive RAM sunt activate în timpul unei operații de citire sau scriere.
- Algoritm primitiv fixat: memoria este generată folosind un singur tip de bloc RAM primitiv.

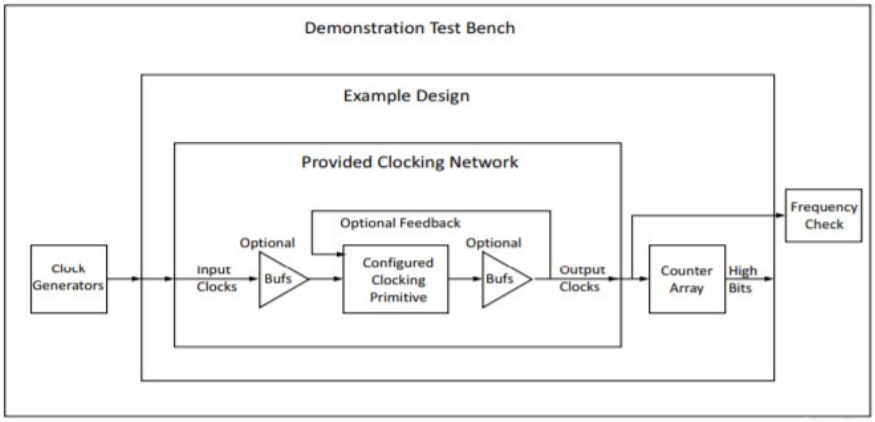


Pentru crearea unui block de memorie RAM, se selectează **IP** Catalog în panoul **Flow Navigator**. În noua fereastă din dreapta **(IP Catalog)** se caută și se selectează **Block Memory Generator**. În fereastra **Basic** tipul interfeței se alege **Native** și tipul memoriei **Single Port RAM**. Algoritmul selectat pentru configurarea blocului primitive RAM este **Minimum Area**. În fereastra **Port A Options** se configurează opțiunile unicului port al memoriei. Se selectează **Write Width** la 8 deoarece aceasta este dimensiunea unui octet de culoare = [R2,R1,R0,G2,G1,G0,B1,B0]. Automat se va completa cu 8 și câmpul **Read Width**. Câmpurile **Write Depth** și **Read Depth** vor fi reprezentate de dimensiunea imaginii și anume 65536. Modul de operare pentru fiecare port determină relația dintre interfețele de scriere și citire pentru acel port. Portul A și portul B pot fi configurate independent cu oricare dintre trei moduri de scriere: read first, write first sau modul fără schimbare. În modul write first, datele de intrare sunt scrise simultan în memorie și conduse la ieșire. Acest mod transparent oferă flexibilitatea utilizării magistralei de ieșire a datelor în timpul unei operații de scriere pe aceeași port.

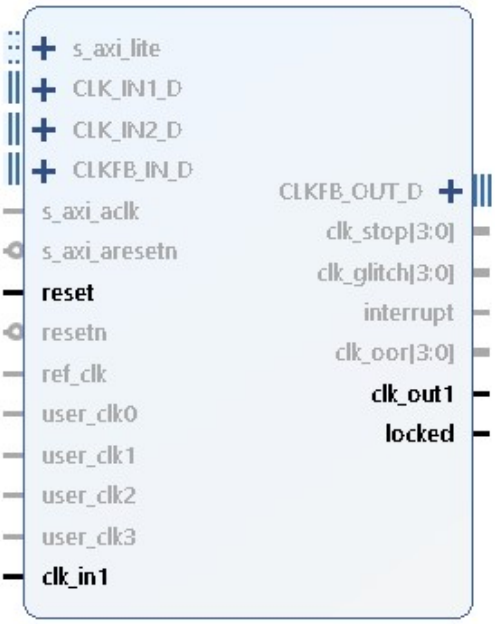


Se selecteaza opțiunea **Write First** și pentru câmpul Enable Port Type se selectează **Always Enabled** (astfel nu se vor mai adăuga semnale adiționale ENA sau ENB). În fereastra **Other Options** se încarcă fișierul cu extensia COE rezultat din funcția Matlab definită anterior și se generează blocul de memorie.

S-a menționat faptul că scanarea liniilor se face la rata de 25MHz. Asta înseamnă că semnalul de ceas al plăcii FPGA trebuie divizat deoarece majoritatea oscilatoarelor iau valori de 50 - 100 MHz. IP-ul Clocking Wizard LogiCORE simplifică crearea pachetelor de cod sursă HDL pentru circuite de ceas personalizate. Acest Wizard ghidează atributele adecvate pentru semnalul de ceas și permite înlocuirea oricărui parametru calculat. Sinteza de frecvență permite ceasurilor de ieșire să prezinte frecvențe diferite comparativ cu cele de intrare. Alinierea fazelor permite blocarea fazei a ceasului de ieșire la o referință, cum ar fi pinul ceasului de intrare pentru un dispozitiv.



Clocking Wizard ajută la crearea unui circuit de ceas pentru o frecvență, fază și factor de umplere dați de utilizator utilizând un manager de ceas în mod mixt (**Mixed Mode Clock Manager**) (MMCM) (E2 / E3 / E4) sau buclă blocată în fază (**Phase Locked Loop**) (PLL) (E2 / E3 / E4). De asemenea, ajută la verificarea frecventei ceasului output în simulare, oferind un exemplu de proiectare sintetizabil care poate fi testat pe hardware. Pentru crearea unui Wizard se selectează **IP** Catalog în panoul **Flow Navigator**. În noua fereastă din dreapta **(IP Catalog)** se caută și se selectează **Clocking Wizard**. În fereastra **Clocking Options** se bifează primitiva **PLL**. Un PLL este un sistem de control pentru feedback care ajustează automat faza unui semnal generat local pentru a se potrivi cu faza unui semnal de intrare. PLL-urile funcționează producând o frecvență a oscilatorului pentru a se potrivi cu frecvența unui semnal de intrare. În regiunea **Clocking Features** se bifează **Frequency Synthesis** și **Phase Alignment** iar in regiunea **Jitter Optimization** se bifează **Balanced** (se alege automat lățimea de bandă corectă pentru optimizarea instabilităților). În fereastra **Output Clocks** se setează frecvența de ieșire (Output Freq) la 25MHz cu un factor de umplere de 50 (%). În regiunea **Clocking Feedback** se bifează opțiunea **Automatic Control On-Chip** și se generează Clocking Wizard-ul.



În continuare se definește entitatea proiectului din fișierul main vga_driver în care se declară și se folosesc cele două componente IP descrise anterior:

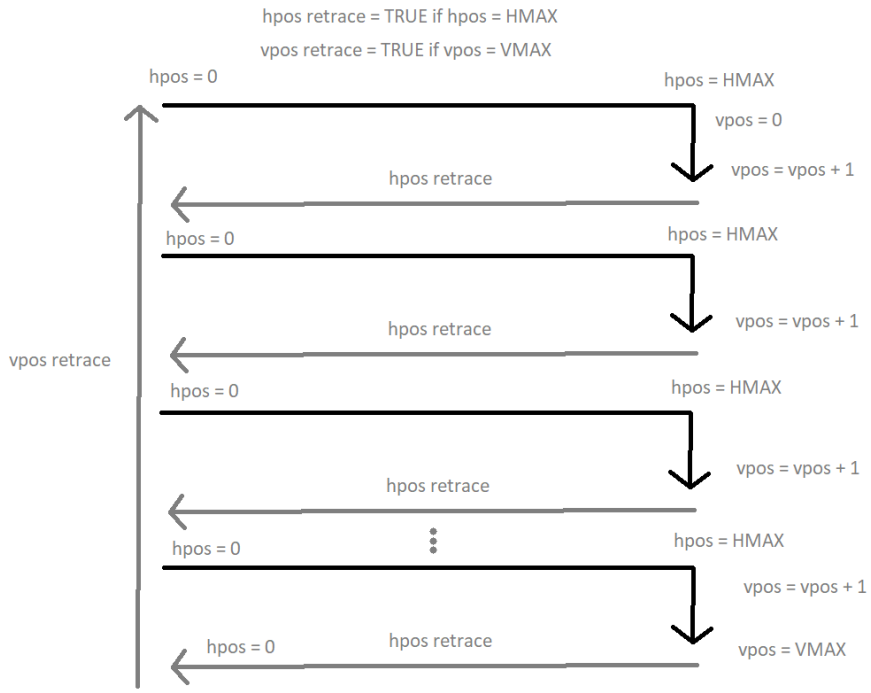
```
entity vga_driver is
     port (clk : in  std_logic;
           hsync : out std_logic;
           vsync : out std_logic;
           R,G : out std_logic_vector (2 downto 0);
           B : out std_logic_vector (1 downto 0));
end vga_driver;
```

Singurul semnal de intrare este reprezentat de semnalul de ceas al plăcii, care ulterior va fi divizat la 25MHz. Ieșirile sunt cele 5 elemente specifice fiecărui controler VGA: Semnalele de sincronizare orizontală și verticală și componenta RGB de 8 biți. În continuare se prezintă arhitectura acestui driver VGA, începând cu declararea constantelor, semnalelor și componentelor necesare în rezolvarea problemei și pe urmă se va discuta logica problemei.

```
architecture behavioral of vga_driver is

signal clk25 : std_logic := '0'; -- Semnalul de ceas de 25 MHz (frecventa pixelilor)

constant picture_size : integer := 65536;

-- Semnale pentru blocul de memorie RAM
signal wea : std_logic_vector(0 downto 0) := "0";
signal addra : std_logic_vector(15 downto 0) := (others => '0');
signal dina : std_logic_vector(7 downto 0) := (others => '0');
signal douta : std_logic_vector(7 downto 0) := (others => '0');

constant HD : integer := 639; -- Horizontal Display
constant HFP : integer := 16; --  Right border (front porch)
constant HSP : integer := 96; -- Sync pulse (retrace)
constant HBP : integer := 48; --  Left border (back porch)

-- HD + HFP + HSP + HBP = 800

constant VD : integer := 479;  -- Vertical Display (480)
constant VFP : integer := 10;  -- Right border (front porch)
constant VSP : integer := 2; -- Sync pulse (retrace)
constant VBP : integer := 33; --  Left border (back porch)

-- VD + VHP + VSP + VBP = 525

signal hPos : integer := 1;
signal vPos : integer := 0;

signal videoOn : std_logic := '0'; -- Semnal de desenare
```

```
component blk_mem_gen_0
      port (clka : in std_logic; -- Semnal de ceas
            wea : in std_logic_vector(0 downto 0); - Write Enable
            addra : in std_logic_vector(15 downto 0); -- Adrese pe 16 biti
            dina : in std_logic_vector(7 downto 0); -- Pixel cu RGB
            douta : out std_logic_vector(7 downto 0)); -- Pixel cu RGB
end component;

component clk_wizard
    port (clk_out1 : out std_logic; -- Ceasul cu frecventa 25MHz
          clk_in1 : in std_logic); -- Ceasul cu frecventa 100MHz
end component;

begin

clk_divider : clk_wizard
   port map (clk_out1 => clk25,
             clk_in1 => clk); -- clk va fi ceasul de 25MHz

picture_load : blk_mem_gen_0
    port map (clka => clk25,
              wea => wea,
              addra => addra,
              dina => dina,
              douta => douta);
```

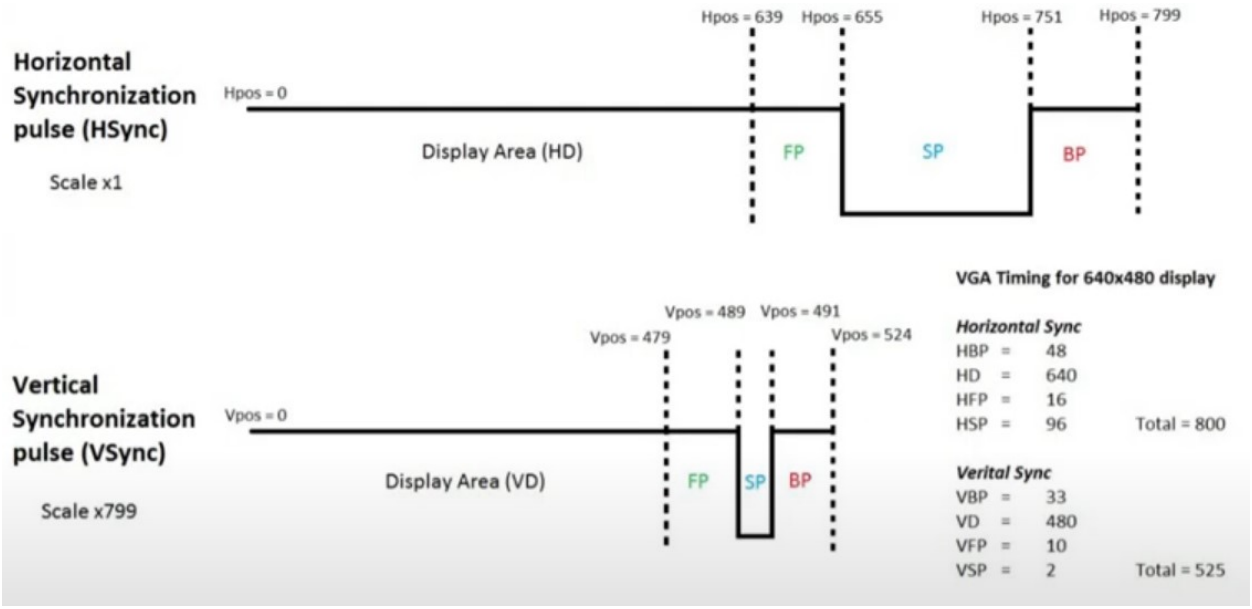
În schema următoare se prezintă modul în care se schimbă valorile semnalelor de poziție pentru desenarea imaginii. Se va descrie un proces în care se va prelucra contorul orizontal hPos, și anume pe frontul crescător de ceas dacă acesta a ajuns la valoarea maximă (se ține cont și de întârzierile create de semnalele porch) (**Horizontal Display + Horizontal Front Porch + Horizontal Sync Pulse + Horizontal Back Porch**) atunci se va reseta la 0, altfel se incrementeaza. Analog se va descrie un proces și pentru contorul vertical, unde diferă semnalul vPos și valoarea maximă posibilă (**Vertical Display + Vertical Front Porch + Vertical Sync Pulse + Vertical Back Porch**). În plus se ține cont și de valoarea semnalului hPos în lista de sensibilitate, deoarece pe verticală se realizează operația de vPos retrace cand s-a ajuns pe ultima linie și pe ultima coloană (se va reseta și semnalul hPos in celălalt proces).




```
HPC : process(clk25)
begin
        if clk25'event and clk25 = '1' then
                if hPos = 799 then
                        hPos <= 0;
                else
                        hPos <= hPos + 1;
                end if;
        end if;
end process;
```

```
VPC : process(clk25, hPos)
begin
        if clk25'event and clk25 = '1' then
                if hPos = 799 then
                        if vPos = 524 then
                                vPos <= 0;
                        else
                                vPos <= vPos + 1;
                        end if;
                end if;
        end if;
end process;
```

În figura următoare se prezintă diagramele de timp ale semnalelor de sincronizare. Se vor descrie două procese care vor seta pe 1 sau pe 0 aceste două semnale în funcție de valoarea contoarelor. Pentru ambele semnale se observă ca sunt setate pe '1' logic cand nu se află în momentul SP (Sync Pulse) (Adică aceste contoare se află fie la o valoare mai mică decât **Display + Front Porch**, fie mai mare decât **Display + Front Porch + Sync Pulse**.




```
HorizontalSync : process(clk25, hPos)
begin
        if clk25'event and clk25 = '1' then
                if((hPos <= (HD + HFP)) or (hPos > HD + HFP + HSP)) then
                        hsync <= '1';
                else
                        hsync <= '0';
                end if;
        end if;
end process;
```

```
VerticalSync: process(clk25, vPos)
begin
        if clk25'event and clk25 = '1' then
                if((vPos <= (VD + VFP)) or (vPos > VD + VFP + VSP)) then
                        vsync <= '1';
                else
                        vsync <= '0';
                end if;
        end if;
end process;
```

Următorul proces setează variabila videoOn (semnalul de desenare) care este activ pe '1' logic doar în momentul în care contoarele **hPos** și **vPos** se află în zona vizibilă a regiunii 800x525 și anume regiunea de display 640x480. Când aceste variabile se află în afara regiunii în momentele de porch sau sync nu se va desena pe ecran.

```
Video : process(clk25, hPos, vPos)
begin
        if clk25'event and clk25 = '1' then
                if hPos <= HD and vPos <= VD then
                        videoOn <= '1';
                else
                        videoOn <= '0';
                end if;
        end if;
end process;
```

Ultimul proces este cel care realizează desenarea imaginii pe ecran, în momentul când semnalul videoOn este setat pe '1' logic, se parcurge fiecare element al matricei 256x256 de **picture_size** 65536 și se prelucrează valorile RGB, după care se incrementează adresa pentru parcurgerea elementului următor din memorie.

```
draw : process(clk25, hPos, vPos, videoOn)
begin
        if clk25'event and clk25 = '1' then
                if videoOn = '1' then
            if unsigned(addra) < picture_size then
                R <= douta(7 downto 5);
                G <= douta(4 downto 2);
                B <= douta(1 downto 0);
                addra <= std_logic_vector(unsigned(addra)+1);
                if hPos = HD and vPos = VD then
                    addra <= (others => '0');
                end if;
                        else
                                    R <= (others => '0');
                                    G <= (others => '0');
                                    B <= (others => '0');
                        end if;
                        else
                R <= (others => '0');
                G <= (others => '0');
                B <= (others => '0');
                end if;
        end if;
end process;
end behavioral;
```

Pentru procesarea imaginii, se introduc în cadrul entității proiectului următoarele semnale de intrare :

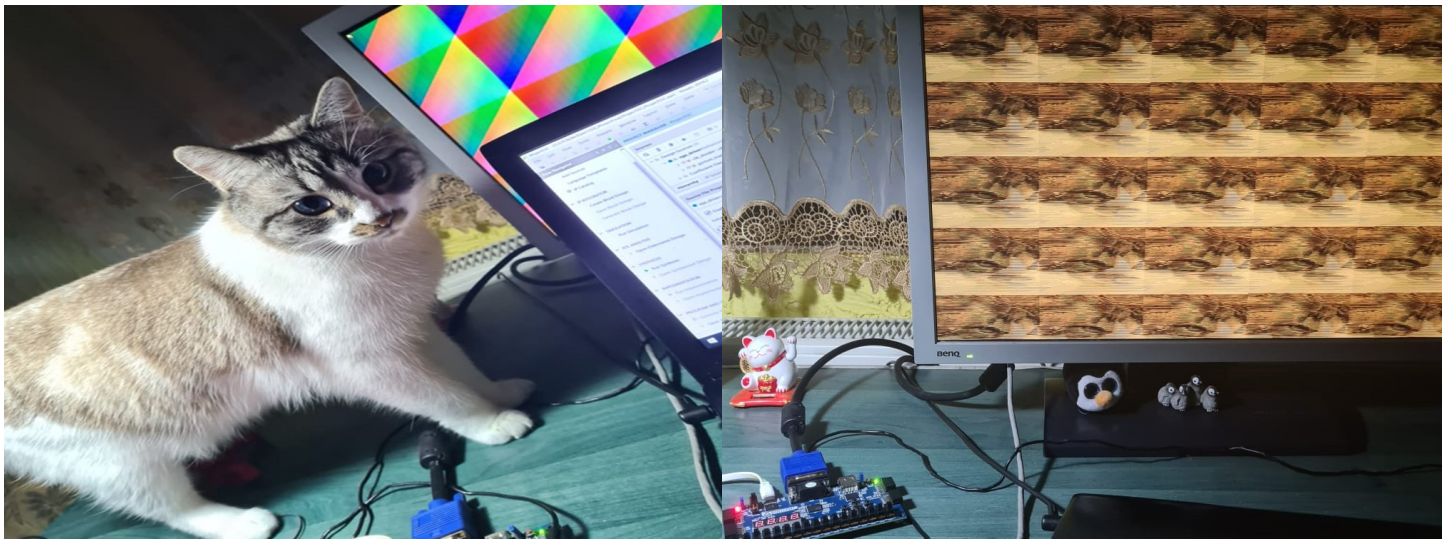
```
           invert : in std_logic;
           brighten : in std_logic;
           binary : in std_logic;
           darken : in std_logic;
```

În momentul când un semnal este setat pe '1' logic, imaginea generată pe ecran se va procesa dupa modul selectat : pentru invert, imaginea se va afișa în modul negativ; pentru brighten, imaginea va fi afișată cu o luminozitate mai ridicată; pentru darken, imaginea se afișează cu o luminozitate scăzută iar pentru binary, imaginea se afișează in modul binar. Pentru realizarea acestor operații, se adaugă următoarele linii de cod în ultimul proces descris.

```
if invert = '1' and brighten  = '0' and darken = '0' and bin = '0' then
    R <= "111" - douta(7 downto 5);
    G <= "111" - douta(4 downto 2);
    B <= "11" - douta(1 downto 0);
end if;
if brighten  = '1' and invert = '0' and darken = '0' and bin = '0' then
    if douta(7 downto 5) < "100" then
        R <= douta(6 downto 5) & "0" ;
    else
        R <= "111";
    end if;
    if douta(4 downto 2) < "100" then
        G <= douta(3 downto 2) & "0" ;
    else
        G <= "111";
    end if;
    if douta(1 downto 0) < "10" then
        B <= douta(0) & "0" ;
    else
        B <= "11";
    end if;
end if;
if invert = '0' and brighten  = '0' and darken = '1' and bin = '0' then
        R <= "0" &  douta(7 downto 6);
        G <= "0" & douta(4 downto 3);
        B <= "0" & douta(1);
end if;
if invert = '0' and brighten = '0' and darken = '0' and bin = '1' then
        threshold <= douta(7 downto 5) * "100000" + douta(4 downto 2)
        * "100" + douta(1 downto 0);
        if conv_integer(threshold) > 139 then
                R <= (others => '1');
                G <= (others => '1');
                B <= (others => '1');
        else
                R <= (others => '0');
                G <= (others => '0');
                B <= (others => '0');
        end if;
end if;
```

Pentru creșterea luminozității se înmulțește cu 2 fiecare valoare RGB. Pentru scăderea luminozității se împarte cu 2 fiecare valoare RGB. Pentru negativul imaginii, valorile RGB iau complementul acestora iar pentru a afișa imaginea în modul binar, se calculează o valoare de prag pentru fiecare pixel care va desena fie negru, fie alb.

Rezultate experimentale



(a) Înainte de proiect (b) După proiect

Figure 4.1: Afișarea pisicii pe ecran

(a) Bright (b) Dark (c) Negative (d) Binary

Figure 4.2: Modurile de procesare ale imaginii

În figurile de mai sus se ilustrează rezultatul obținut. S-a utilizat placa **Basys3 Artix7** și un monitor VGA **BenQ FP222WA** de rezoluțIie 1680x1050. În Anexă se găsesc fișierele de constrângeri atât pentru placa Basys3, cât și pentru placa **Nexys4**. Se prezintă un scurt manual de utilizare (Basys3) :

- Comutatorul V15 (SW5) : modul **invert**.
- Comutatorul W15 (SW4) : modul **luminos**.
- Comutatorul W17 (SW3) : modul **întunecat**.
- Comutatorul W16 (SW2) : modul **binar**.

Concluzii

Procesarea de imagini reprezintă o unealtă puternică în alte aplicații precum procesarea video sau pattern recognition. Am învățat din documentația VGA și Xilinx pentru a reuși desenarea imaginii pe ecran, respectiv a include blocurile de memorie și clocking wizard pentru realizarea proiectului. Posibilități de dezvoltare ale proiectului sunt includerea unor noi moduri de procesare ale imaginii (grayscale de exemplu) care este mai dificil de implementat și creșterea calității imaginii prin extinderea componentei RGB de la 8 la 24 de biți.

$* * *$

Rezumat

S-a reușit desenarea pisicii Yuki pe monitor, mai concis s-a rezolvat problema afișării unei imagini utilizând portul VGA al unei plăci FPGA și s-a implementat un mod de procesare al imaginilor afișate. Utilizând limbajul VHDL, s-a realizat un conector VGA care a stocat pe placă imaginea reprezentată sub forma unei matrici de componente RGB (rezultate din Matlab) și s-a extins un proces VHDL pentru a face posibilă procesarea imaginii. Rezultatele s-au obținut folosind placa Basys3.

$* * *$

Bibliografie

https://en.wikipedia.org/wiki/VGA_connector

https://components101.com/connectors/vga-connector-pinout-datasheet

http://tinyvga.com/vga-timing/640x480@60Hz

http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab3_19/rom_vivado.html

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug896-vivado-ip.pdf

http://jqm.io/files/coetool/

https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf

https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf

* * *

Anexă

Fișierul principal din design source Vivado :

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

entity vga_driver is
     port (clk : in  std_logic;
           hsync : out std_logic;
           vsync : out std_logic;
           R,G : out std_logic_vector (2 downto 0);
           invert : in std_logic;
           brighten : in std_logic;
           binary : in std_logic;
           darken : in std_logic;
           B : out std_logic_vector (1 downto 0));
end vga_driver;

architecture behavioral of vga_driver is
```

```
signal clk25 : std_logic := '0'; -- Semnalul de ceas de 25 MHz (frecventa pixelilor)

constant picture_size : integer := 65536;

--Signals for Block RAM
signal wea : std_logic_vector(0 downto 0) := "0";
signal addra : std_logic_vector(15 downto 0):= (others => '0');
signal dina : std_logic_vector(7 downto 0) := (others => '0');
signal douta : std_logic_vector(7 downto 0):= (others => '0');

constant HD : integer := 639; -- Horizontal Display
constant HFP : integer := 16; --  Right border (front porch)
constant HSP : integer := 96; -- Sync pulse (retrace)
constant HBP : integer := 48; --  Left border (back porch)

-- HD + HFP + HSP + HBP = 800

constant VD : integer := 479;  -- Vertical Display (480)
constant VFP : integer := 10;  -- Right border (front porch)
constant VSP : integer := 2; -- Sync pulse (retrace)
constant VBP : integer := 33; --  Left border (back porch)

-- VD + VHP + VSP + VBP = 525

signal hPos : integer := 1;
signal vPos : integer := 0;
```

```
signal inv : std_logic := '0';

signal bri : std_logic := '0';

signal dark : std_logic := '0';

signal bin : std_logic := '0';

signal threshold : std_logic_vector(8 downto 0) := (others => '0');


signal videoOn : std_logic := '0';


component blk_mem_gen_0

      port (clka : in std_logic;

            wea : in std_logic_vector(0 downto 0);

            addra : in std_logic_vector(15 downto 0);

            dina : in std_logic_vector(7 downto 0);

            douta : out std_logic_vector(7 downto 0));

end component;


component clk_wizard

    port (clk_out1 : out std_logic;

          clk_in1 : in std_logic);

end component;


begin


clk_divider : clk_wizard

   port map (clk_out1 => clk25,

             clk_in1 => clk);


picture_load : blk_mem_gen_0
```

```
    port map (clka => clk25,
              wea => wea,
              addra => addra,
              dina => dina,
              douta => douta);


HPC : process(clk25)
begin
        if clk25'event and clk25 = '1' then
                if hPos = 799 then
                        hPos <= 0;
                else
                        hPos <= hPos + 1;
                end if;
        end if;
end process;


VPC : process(clk25, hPos)
begin
        if clk25'event and clk25 = '1' then
                if hPos = 799 then
                        if vPos = 524 then
                                vPos <= 0;
                        else
                                vPos <= vPos + 1;
                        end if;
                end if;
        end if;
```

```
end process;


HorizontalSync : process(clk25, hPos)
begin
        if clk25'event and clk25 = '1' then
                if((hPos <= (HD + HFP)) OR (hPos > HD + HFP + HSP))then
                        hsync <= '1';
                else
                        hsync <= '0';
                end if;
        end if;
end process;


VerticalSync: process(clk25, vPos)
begin
        if clk25'event and clk25 = '1' then
                if((vPos <= (VD + VFP)) OR (vPos > VD + VFP + VSP))then
                        vsync <= '1';
                else
                        vsync <= '0';
                end if;
        end if;
end process;


Video : process(clk25, hPos, vPos)
begin
        if clk25'event and clk25 = '1' then
                if hPos <= HD and vPos <= VD then
```

```
                videoOn <= '1';
            else
                videoOn <= '0';
            end if;
        end if;
end process;

draw : process(clk25, hPos, vPos, videoOn)
begin
        if clk25'event and clk25 = '1' then
            if videoOn = '1' then
            if unsigned(addra) < picture_size then
                R <= douta(7 downto 5);
                G <= douta(4 downto 2);
                B <= douta(1 downto 0);

                if invert = '1' and brighten  = '0' and darken = '0' and bin = '0' then
                    R <= "111" - douta(7 downto 5);
                    G <= "111" - douta(4 downto 2);
                    B <= "11" - douta(1 downto 0);
                end if;

                if brighten  = '1' and invert = '0' and darken = '0' and bin = '0' then
                    if douta(7 downto 5) < "100" then
                        R <= douta(6 downto 5) & "0" ;
                    else
                        R <= "111";
                    end if;
```

```
        if douta(4 downto 2) < "100" then
            G <= douta(3 downto 2) & "0" ;
        else
            G <= "111";
        end if;
        if douta(1 downto 0) < "10" then
            B <= douta(0) & "0" ;
        else
            B <= "11";
        end if;
    end if;

    if invert = '0' and brighten  = '0' and darken = '1' and bin = '0' then
        R <= "0" &  douta(7 downto 6);
        G <= "0" & douta(4 downto 3);
        B <= "0" & douta(1);
    end if;

    if invert = '0' and brighten = '0' and darken = '0' and bin = '1' then
        threshold <= douta(7 downto 5) * "100000" + douta(4 downto 2) * "100"
        + douta(1 downto 0);
        if conv_integer(threshold) > 139 then
            R <= (others => '1');
            G <= (others => '1');
            B <= (others => '1');
        else
            R <= (others => '0');
            G <= (others => '0');
```

```
                        B <= (others => '0');
                    end if;
                end if;

                addra <= std_logic_vector(unsigned(addra)+1);
                if hPos = HD and vPos = VD then
                    addra <= (others => '0');
                end if;
                    else
                            R <= (others => '0');
                            G <= (others => '0');
                            B <= (others => '0');
                    end if;
                    else
                R <= (others => '0');
                G <= (others => '0');
                B <= (others => '0');
                end if;
        end if;
end process;

inv <= invert;
bri <= brighten;
dark <= darken;
bin <= binary;

end behavioral;
```

Fișierul de constrângeri pentru placa Basys3 :

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
  set_property IOSTANDARD LVCMOS33 [get_ports clk]
  create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
## Switches
set_property PACKAGE_PIN W16 [get_ports {binary}]
  set_property IOSTANDARD LVCMOS33 [get_ports {binary}]
set_property PACKAGE_PIN W17 [get_ports {darken}]
  set_property IOSTANDARD LVCMOS33 [get_ports {darken}]
set_property PACKAGE_PIN W15 [get_ports {brighten}]
  set_property IOSTANDARD LVCMOS33 [get_ports {brighten}]
set_property PACKAGE_PIN V15 [get_ports {invert}]
  set_property IOSTANDARD LVCMOS33 [get_ports {invert}]
## VGA Connector
set_property PACKAGE_PIN G19 [get_ports {R[0]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {R[0]}]
set_property PACKAGE_PIN H19 [get_ports {R[1]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {R[1]}]
set_property PACKAGE_PIN J19 [get_ports {R[2]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {R[2]}]
set_property PACKAGE_PIN N19 [get_ports {R[3]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {R[3]}]
set_property PACKAGE_PIN N18 [get_ports {B[0]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
set_property PACKAGE_PIN L18 [get_ports {B[1]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
set_property PACKAGE_PIN K18 [get_ports {B[2]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
set_property PACKAGE_PIN J18 [get_ports {B[3]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
set_property PACKAGE_PIN J17 [get_ports {G[0]}]
```

```
  set_property IOSTANDARD LVCMOS33 [get_ports {G[0]}]
set_property PACKAGE_PIN H17 [get_ports {G[1]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {G[1]}]
set_property PACKAGE_PIN G17 [get_ports {G[2]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {G[2]}]
set_property PACKAGE_PIN D17 [get_ports {G[3]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {G[3]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
  set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
  set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```

Fișierul de constrângeri pentru placa Nexys4 :

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }]
                                              ;
## Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { invert }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { brighten }]
                                                  ;
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { binary }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { darken }];
## VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { R[0] }];
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { R[1] }];
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { R[2] }];
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { R[3] }];
set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { G[0] }];
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { G[1] }];
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { G[2] }];
```

```
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { G[3] }];
set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { B[0] }];
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { B[1] }];
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { B[2] }];
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { B[3] }];
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { hsync }];
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { vsync }];
```

Fișierul Matlab IMG2coe8 :

```
function img2 = IMG2coe8(imgfile, outfile)
imgfile = 'yukicat.jpg';
outfile = 'yukicat.coe';
img = imread(imgfile);
height = size(img, 1);
width = size(img, 2);
s = fopen(outfile,'wb');
fprintf(s,'%s\n','; VGA Memory Map ');
fprintf(s,'%s\n','; .COE file with hex coefficients ');
fprintf(s,'; Height: %d, Width: %d\n\n', height, width);
fprintf(s,'%s\n','memory_initialization_radix=16;');
fprintf(s,'%s\n','memory_initialization_vector=');
cnt = 0;
img2 = img;


for r = 1:height
    for c = 1:width
        cnt = cnt + 1;
        R = img(r,c,1);
        G = img(r,c,2);
        B = img(r,c,3);
        Rb = dec2bin(double(R),8);
```

```
        Gb = dec2bin(double(G),8);
        Bb = dec2bin(double(B),8);
        img2(r,c,1) = bin2dec([Rb(1:3) '00000']);
        img2(r,c,2) = bin2dec([Gb(1:3) '00000']);
        img2(r,c,3) = bin2dec([Bb(1:2) '000000']);
        Outbyte = [Rb(1:3) Gb(1:3) Bb(1:2)];

        if (Outbyte(1:4) == '0000')
            fprintf(s,'0%X',bin2dec(Outbyte));
        else
            fprintf(s,'%X',bin2dec(Outbyte));
        end
        if ((c == width) && (r == height))
            fprintf(s,'%c',';');
        else
            if (mod(cnt,32) == 0)
                fprintf(s,'%c\n',',');
            else
                    fprintf(s,'%c',',');
            end
        end
    end
end
fclose(s);
```