



---

**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

---

Universitatea Tehnică din Cluj-Napoca  
Facultatea de Automatică și Calculatoare  
Domeniul: Calculatoare și Tehnologia Informației  
Grupa: 30239

# **COMUNICAREA DINTRE UN FPGA ȘI UN MOUSE**

Boncea Natalia Georgiana

11 ianuarie 2024

## CUPRINS

<b>Capitolul 1 – INTRODUCERE</b>	<b>3</b>
1.1. Context	3
1.2. Soluția propusă	3
1.2.1. Cerința problemei	3
1.2.2. Decizii de proiectare	3
1.3. Plan de proiect	4
<b>Capitolul 2 – STUDIUL BIBLIOGRAFIC</b>	<b>5</b>
2.1. Prezentare generală	5
2.2. Protocoale de comunicare	5
2.2.1. Comunicația serială	5
2.2.2. Protocolul de comunicare PS/2	6
2.2.3. Protocolul de comunicare UART	7
2.2.4. Protocolul de comunicare I2C	8
2.2.5. Protocolul USB-HID	8
2.3. Interfețe de comunicare	9
2.3.1. USB	10
2.3.2. VGA	11
<b>Capitolul 3 – ANALIZĂ</b>	<b>13</b>
3.1. Prezentare generală	13
3.2. Modul de funcționare al protocolului PS/2	13
3.3. Protocolul de comunicare Host-to-PS/2-device	15
3.4. Sincronizarea cu portul VGA	16
3.4.1. Sincronizarea orizontală	16
3.4.2. Sincronizarea verticală	18
3.4.3. Calcularea timpului pentru semnalele de sincronizare VGA	20
<b>Capitolul 4 – PROIECTARE</b>	<b>21</b>
4.1. Diagramele bloc	21
4.1.1. Diagrama arhitecturii sistemului	21
4.1.2. Diagramele bloc pentru protocolul de comunicare PS/2	21



4.1.2.1. Componenta de transmitere (PS2_TX) .....	21
4.1.2.2. Componenta de receptare (PS2_RX) .....	22
4.1.2.3. Componenta de execuție (PS2_RXTX).....	23
4.1.2.4. Componenta de comandă (Mouse).....	25
4.1.3. Diagramele bloc pentru modulul VGA .....	25
4.1.3.1. Componenta VGA_Sync .....	25
4.1.3.2. Componenta RGB_VGA.....	26
4.1.3.3. Componenta Cursor .....	27
4.1.4. Diagramele bloc pentru componentele auxiliare.....	28
4.1. Organigrame.....	29
4.1.1. Organigrama componentei PS2_TX .....	30
4.1.2. Organigrama componentei PS2_RX .....	31
4.1.3. Organigrama automatului de stări finite pentru modul Mouse .....	32
<b>Capitolul 5 – IMPLEMENTARE .....</b>	<b>33</b>
5.1. Modul Mouse .....	33
5.1.1. PS2_TX.....	33
5.1.2. PS2_RX.....	37
5.1.3. PS2_RXTX .....	40
5.2. Modul VGA .....	44
5.2.1. VGA_SYNC (Circuitul de sincronizare VGA).....	44
5.2.2. CURSOR(Circuitul de desenare a cursorului) .....	48
5.2.3. RGB_VGA(Circuitul modulului VGA).....	50
5.3. Modul Mouse_to_VGA .....	51
5.4. Componente adiționale .....	54
5.4.1. Counter enable .....	54
<b>Capitolul 6 – TESTARE &amp; VALIDARE .....</b>	<b>55</b>
6.1. Testarea modulului pentru mouse .....	55
6.2. Testarea modulului VGA .....	57
<b>Capitolul 7 – CONCLUZII .....</b>	<b>60</b>
<b>BIBLIOGRAFIE.....</b>	<b>61</b>



## Capitolul 1 – INTRODUCERE

### 1.1. Context

**Descriere generală:** Un FPGA (Field Programmable Gate Array) este un circuit integrat care poate fi programat și reconfigurat după producție. Acesta conține o rețea de blocuri logice programabile (PLD) și interconexiuni, permițând utilizatorilor să implementeze circuite logice complexe.

Plăcile FPGA sunt prevăzute cu diverse componente care îi permit să funcționeze și să fie programată pentru diverse aplicații. Printre aceste componente se numără porturile și interfețele de comunicare, care permit plăcilor să comunice cu alte dispozitive periferice. În acest fel, placa FPGA poate deveni un intermediar, un centru de comandă și control între un dispozitiv de intrare, precum un mouse, și un dispozitiv de ieșire, precum ecranul unui calculator.

### 1.2. Soluția propusă

#### 1.2.1. Cerința problemei

Realizează o aplicație care realizează comunicarea dintre o placă Basys 3 și un mouse USB-HID.

**Specificații:** Aplicația va fi proiectată utilizând limbajul de descriere hardware VHDL. Implementarea se va face pe placa Basys 3. Aceasta prevede afișarea cursorului și a situației butoanelor apăsate într-o aplicație pe calculator. Aplicația de afișare a cursorului va fi implementată în limbajul C/C++ sau ceva similar.

#### 1.2.2. Decizii de proiectare

Înainte de a începe descrierea fazei de proiectare și proiectarea propriu-zisă, trebuie stabilite câteva decizii privind particularitățile aplicației de comunicare între placa FPGA și mouse. Acestea ne vor ajuta să avem un obiectiv clar în minte, ceea ce va rezulta într-un plan de proiect riguros și elocvent.

#### Decizii de proiectare:

1. Pentru comunicarea între mouse și placa FPGA se va utiliza interfața de comunicare PS/2;
2. Aplicația pe calculator se va realiza prin intermediul modulului VGA;
3. Aplicația pe calculator va fi scrisă în limbaj VHDL;
4. Exemplificarea interacțiunii corecte dintre placa FPGA și mouse se va face prin modificarea culorii ecranului;
5. Culoarea ecranului se va modifica prin click-stânga pe mouse;



6. Va exista un set de 8 culori pentru ecran și schimbarea culorii se va face în funcție de ordinea în care acestea vor fi hardcodate.

### 1.3. Plan de proiect

Faza de lucru	Perioada de lucru	Descriere
<b>Studiul bibliografic</b>	Săptămânile 2-3	Studierea modului de funcționare a protocoalelor de comunicare USB și VGA și acumularea de informații necesare pentru proiectarea aplicației
<b>Analiză</b>	Săptămâna 4	Descrierea funcționării aplicației și alegerea practicilor adecvate pentru stabilirea conexiunilor cu dispozitivele periferice
<b>Proiectare</b>	Săptămânile 5-7	Proiectarea protocoalelor de comunicare USB și VGA, a funcționalităților pentru butoanele mouse-ului și a interfeței grafice pentru aplicație. Proiectarea bancurilor de test pentru componentele care necesită acest gen de testare.
<b>Implementare</b>	Săptămânile 8-9	Implementarea protocoalelor de comunicare USB și VGA, a funcționalităților pentru butoanele mouse-ului și a interfeței grafice pentru aplicație. Proiectarea bancurilor de test pentru componentele care necesită acest gen de testare.
<b>Testare &amp; Validare</b>	Săptămâna 10	Testarea fiecărei componente din cadrul proiectului în mod individual și rezolvarea problemelor care ar putea apărea. Conectarea tuturor componentelor la un loc și testarea funcționării aplicației prin rularea programului scris.



## Capitolul 2 – STUDIUL BIBLIOGRAFIC

### 2.1. Prezentare generală

În acest capitol se va descrie pe larg design-ul aplicației și modul de funcționare a fiecărei componente în parte. Vom vedea cum funcționează protocoalele de comunicare pentru porturile USB-HID și VGA, cum se transmit datele între dispozitivele periferice și placa FPGA, precum și alte aspecte ce ne vor fi utile pentru aplicația noastră.

### 2.2. Protocoale de comunicare

Protocoalele de comunicare sunt seturi de reguli convenționale care stabilesc modul în care dispozitivele sau entitățile comunică între ele într-o rețea de calculatoare sau într-un sistem de comunicații. Aceste protocoale definesc formatele datelor, metodele de transmitere și procedurile de gestionare a comunicațiilor pentru a asigura o înțelegere comună între părțile implicate în comunicare.

De obicei, protocoalele de comunicare sunt stratificate în nivele, fiecare nivel având responsabilități specifice. Acest model de stratificare facilitează dezvoltarea, întreținerea și extinderea rețelelor și sistemelor de comunicații. Exemple de protocoale de comunicare include:

- TCP/IP (Transmission Control Protocol/Internet Protocol),
- HTTP (Hypertext Transfer Protocol),
- SMTP (Simple Mail Transfer Protocol),
- Comunicația serială,
- UART (Universal Asynchronous Receiver/Transmitter),
- ISC,
- USB-HID, etc.

În esență, protocoalele de comunicare sunt precum limbajul comun folosit de dispozitive pentru a se înțelege reciproc în cadrul unei rețele sau a unui sistem.

În cele ce urmează, vor fi prezentate mai în detaliu câteva dintre protocoalele de comunicare cele mai des utilizate, modul lor de funcționare, caracteristicile principale, în ce situații sunt utilizate și ce presupun. În urma acestei analize a setului de protocoale pus la dispoziție, le vom alege pe cele care ne vor fi de folos în realizarea aplicației noastre.

#### 2.2.1. Comunicația serială

Comunicația serială metodă de comunicare care are ca principiu de bază transmisia sau recepția datelor bit cu bit, un singur bit fiind transmis/recepționat la un moment dat. Deoarece, în sistemele de calcul, datele sunt reprezentate pe octet (sau multiplu), se folosește un port serial cu rolul de a converti fiecare octet într-un șir de biți (0 sau 1) și viceversa.

Numeroase cabluri pe care le folosim în viața de zi cu zi pentru a interacționa cu calculatoarele noastre sau cu alte dispozitive funcționează pe baza unui protocol de comunicație serială. Cel mai des întâlnit exemplu este cablul USB (Universal Serial Bus).

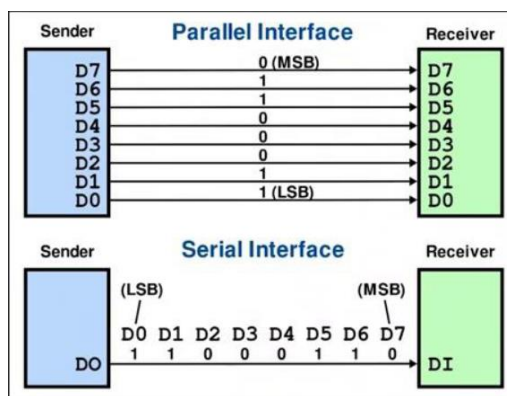


Fig. 1: Comparație între comunicația serială și cea paralelă

La o primă vedere, comunicația paralelă ar părea cea mai avantajoasă metodă în multe cazuri. Cu toate acestea, în multe situații, comunicația serială se dovedește a fi mult mai rapidă și mai eficientă în transmiterea datelor și în reducerea complexității și a conexiunilor decât cea paralelă. Viteza la care datele seriale sunt transmise poate fi modificată prin ceea ce se numește **rata Baud**. Rata Baud reprezintă numărul de biți transmiși pe secundă. Valorile uzuale pentru Baud rate sunt 2400, 4800, 9600 și 19200.

Există mai multe protocoale de comunicație serială, ca de exemplu UART, I2C și USB-HID, despre care vom discuta în continuare.

### 2.2.2. Protocolul de comunicare PS/2

PS/2 este un interfață de comunicare specializată pentru mouse și tastatură. Cu toate că majoritatea mouse-urilor și tastaturilor moderne sunt prevăzute cu interfețe de comunicare USB-HID, transmiterea datelor se realizează tot conform protocolului de comunicare pentru interfața PS/2. În continuare, vom studia principiul de funcționare al portului PS/2 pentru mouse.

Un mouse de calculator este conceput în principal pentru a detecta mișcarea bidimensională pe o suprafață. Circuitul său intern măsoară distanța relativă a mișcării și verifică starea butoanelor. Pentru un mouse cu o interfață PS2, această informație este ambalată în trei pachete și trimisă la gazdă prin portul PS2. În modul de flux, un mouse PS2 trimite pachetele continuu la o rată de eșantionare pre-stabilită.

Comunicația portului PS2 este bidirecțională, iar gazda poate trimite o comandă tastaturii sau mouse-ului pentru a seta anumite parametri. La început, un mouse este setat să fie în modul

non-continuu după pornire și nu trimite niciun fel de date. Gazda trebuie să trimită mai întâi o comandă mouse-ului pentru a-l inițializa și a activa modul de flux. Acest comportament subliniază deosebirea între comunicarea cu un mouse și cea cu o tastatură, în cazul căreia comunicarea este limitată la o singură direcție, și anume de la tastatură la calculator.

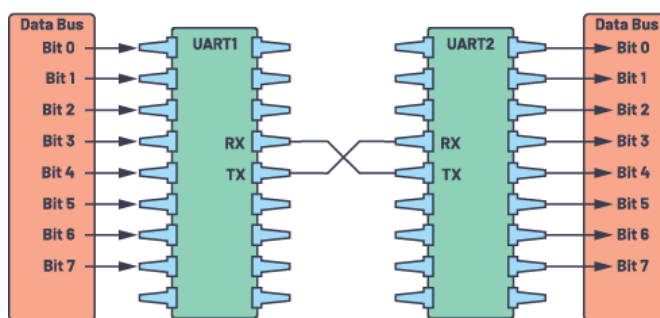
### 2.2.3. Protocolul de comunicare UART

UART (Universal Asynchronous Receiver/Transmitter) este un protocol de comunicație serială asincronă, ceea ce înseamnă că datele sunt transmise fără un semnal de ceas comun. Circuitul electronic UART din portul serial se ocupă de conversia efectivă a fiecărui octet într-un șir de biți (0 și 1) și viceversa.

Comunicarea UART implică două semnale principale: unul pentru transmiterea (ieșirea) de date (TX) și unul pentru primirea (intrarea) de date (RX). Principalul scop al liniei de transmisie și recepție pentru fiecare dispozitiv este să transmită, respectiv să primească date seriale ce intenționează a fi utilizate în comunicarea serială. Transmisia și recepția se pot efectua simultan (full duplex).

Fiind asincron, intervalul dintre pachetele de date poate fi nedefinit. Intervalul de timp dintre biți, pe de altă parte, este fix și trebuie cunoscut de ambele părți. Alte două aspecte care trebuie să fie cunoscute de ambele părți la transmiterea pachetelor sunt lungimea acestora și reprezentarea biților de START și STOP.

Transmițătorul UART este conectat la o magistrală de control de date care trimite datele în paralel. De aici, datele vor fi transmise pe căi seriale, bit cu bit, până la receptorul UART. Acesta, în schimb, va trimite datele seriale într-o formă paralelă către dispozitivul receptor.



**Fig. 2:** Calea de date în comunicarea UART

Acest protocol de comunicare ne va fi de folos în comunicare cu mouse-ul. Dacă un mouse folosește USB, există un convertor sau un controller în interiorul mouse-ului care traduce semnalele USB în comunicare serială și viceversa. Acesta este motivul pentru care, deși interfața fizică este USB, există încă o formă de comunicare serială între mouse și computer.



#### 2.2.4. Protocolul de comunicare I2C

Abrevierea de I2C vine de la Inter-Integrated Circuit. Acesta este un protocol de comunicație serială sincronă, dezvoltat de Philips (acum NXP Semiconductors). Protocolul I2C funcționează pe principiul master-slave. Master-ul generează semnalul de ceas și adresele, iar slave-ul primește semnalul de ceas și adresele. Rolul de master și slave se poate schimba pentru același dispozitiv.

Circuitul electronic I2C are la bază două căi principale: SDA, prin intermediul căruia se transmit datele, respectiv SCL, prin care se transmit semnalele de tact. Asemenea protocolului UART, și de această dată vom avea o parte de transmitere a semnalelor și una de receptare a acestora. Un aspect diferit între cele două protocoale, însă, este ceea ce trebuie să se cunoască de ambele părți, la I2C fiind necesare lungimea pachetului de date și frecvența.

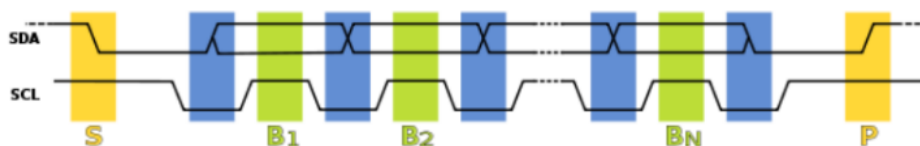


Fig. 3: Diagrama de timp în comunicarea I2C

Acesta este utilizat pentru a permite comunicarea între dispozitive integrate pe un singur bus de comunicație, pe distanțe scurte. Printre aplicațiile cele mai des întâlnite care folosesc acest protocol de comunicare se numără și comunicarea cu monitorul de calculator prin portul VGA sau HDMI.

#### 2.2.5. Protocolul USB-HID

USB-HID (Universal Serial Bus - Human Interface Device) este un protocol specific USB care se referă la dispozitivele care facilitează interacțiunea între utilizator și computer. Acest protocol este frecvent folosit pentru dispozitive precum tastaturi, mouse, gamepads, joystick-uri, dispozitive de control al prezentărilor, și altele care oferă o interfață umană pentru manipularea sau introducerea datelor.

Dispozitivele HID au la bază două concepte fundamentale: un descriptor de raport, respectiv mai multe rapoarte. Rapoartele sunt datele propriu-zise care sunt transmise între dispozitiv și client. Descriptorul de raport descrie formatul datelor și traduce datele de la dispozitivul suport.

Caracteristicile principale ale USB-HID includ:

- **Plug-and-Play:** Dispozitivele USB-HID sunt proiectate pentru a fi recunoscute și configurate automat de către sistemul de operare, permițând utilizatorilor să le conecteze și să le utilizeze fără a necesita driver-e suplimentare în majoritatea cazurilor.
- **Descriptori HID:** Dispozitivele USB-HID sunt definite prin intermediul descripătorilor HID, care furnizează informații despre funcționalitățile și caracteristicile dispozitivului. Acești descriptori sunt incluși în pachetul USB al dispozitivului și sunt utilizați pentru a informa sistemul de operare despre modul în care trebuie tratat dispozitivul.
- **Raportare de Date:** Dispozitivele USB-HID raportează date în cadrul unor structuri numite "rapoarte." Aceste rapoarte conțin informații despre starea dispozitivului, evenimente precum apăsarea unui buton sau mișcarea mouse-ului, și altele.
- **Compatibilitate cu Diverse Dispozitive:** Protocolul USB-HID este versatil și poate fi folosit pentru o gamă largă de dispozitive de interfață umană, ceea ce face posibilă conectarea și utilizarea facilă a acestor dispozitive în diverse medii și aplicații.

### 2.3. Interfețe de comunicare

O interfață de comunicare este un set de reguli și specificații care permit dispozitivelor sau sistemelor să comunice între ele. Aceasta furnizează o modalitate standardizată pentru schimbul de informații între entități diferite, cum ar fi dispozitive hardware, software sau sisteme complexe.

Interfețele de comunicare pot să fie fizice, precum conectorii și cablurile care permit transferul de date între dispozitive, sau logice, cum ar fi protocoalele de comunicație care definesc formatul și modul în care datele sunt transmise și interpretate.

Un exemplu comun de interfață de comunicare este USB (Universal Serial Bus), care include atât aspecte fizice (conectori și cabluri), cât și protocoale de comunicație care permit transferul de date între dispozitive.

În esență, o interfață de comunicare servește ca "punte" între entități diferite, facilitând schimbul de informații într-un mod standardizat și eficient.

În cele ce urmează, vom prezenta mai detaliat interfețele de comunicare cu care vom lucra în cadrul acestui proiect, și anume USB și VGA.

### 2.3.1. USB

USB (Universal Serial Bus) este un standard de interfață de comunicație dezvoltat pentru a facilita conexiunile între diferite dispozitive electronice.

Placa FPGA este echipată cu port USB type-A care face posibilă conectarea cu un mouse sau o tastatură.

Microcontrolerul Auxiliary Function (Microchip PIC24FJ128) asigură plăcii Basys 3 cu capacitatea de gazdă prin portul USB HID. După pornire, microcontrolerul se află în modul de configurare, fie descărcând un bitstream în FPGA, fie așteptând să fie programată din alte surse. Odată ce FPGA este programat, microcontrolerul comută în modul aplicație, care în acest caz este modul USB HID Host. Firmware-ul din microcontroler poate comanda un mouse sau o tastatură conectată la conectorul USB de tip A de la J2 etichetat "USB". Suportul pentru hub-uri nu este disponibil în prezent, așa că poate fi utilizat doar un singur mouse sau o singură tastatură. PIC24 conduce mai multe semnale în FPGA - două dintre ele sunt utilizate pentru a implementa o interfață PS/2 standard pentru comunicarea cu un mouse sau o tastatură, iar celelalte sunt conectate la portul de programare serială cu două fire al FPGA, astfel încât FPGA poate fi programat dintr-un fișier stocat pe un stick USB.

Microcontrolerul Auxiliary Function ascunde protocolul USB HID de FPGA și emulează un bus PS/2 de tip vechi. Microcontrolerul se comportă la fel ca o tastatură sau un mouse PS/2. Acest lucru înseamnă că noile proiecte pot reutiliza nucleele IP PS/2 existente. Mouse-urile și tastaturile care utilizează protocolul PS/2 folosesc un bus serial cu două fire (ceas și date) pentru a comunica cu o gazdă. Pe Basys 3, microcontrolerul emulează un dispozitiv PS/2, în timp ce FPGA joacă rolul de gazdă. Atât mouse-ul, cât și tastatura utilizează cuvinte pe 11 biți care includ un bit de pornire, un octet de date (LSB primul), paritate impară și bit de oprire, dar pachetele de date sunt organizate diferit, iar interfața tastaturii permite transferuri de date bidirecționale (astfel încât dispozitivul gazdă poate aprinde LED-urile de stare de pe tastatură).

În acest sens, pentru stabilirea conexiunii dintre mouse și placa FPGA vom implementa protocolul de comunicare PS/2, cu posibile părți corespondente altor protocole de comunicare.

Așa cum se poate observa din Figura 4, semnalele PS2\_CLK și PS2\_DAT (în codul nostru vor fi denumite ps2c, respectiv ps2d) ale microcontroller-ului sunt folosite pentru a implementa interfața PS/2 de comunicare cu un mouse sau o tastatură. Mouse-ul trimite semnale pe clock și data către placă atunci când este mișcat, iar în caz contrar, datele sunt menținute pe modul HIGH.

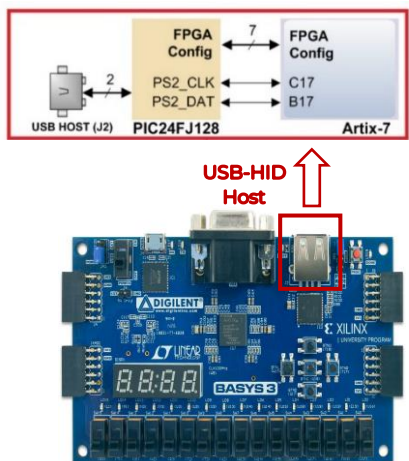


Fig. 4: Portul USB-HID

### 2.3.2. VGA

VGA (Video Graphics Array) este un standard de interfață video care facilitează conexiunea între o placă video și un monitor sau alt dispozitiv de afișare. VGA definește modul în care informațiile video sunt transmise între aceste dispozitive.

Caracteristicile principale ale interfeței VGA includ:

- **Semnalul Analogic:** VGA utilizează un semnal video analogic pentru a transmite informațiile despre culorile și intensitățile pixelilor din imagine.
- **Conectorul VGA:** Un conector specific, de obicei un conector DB-15 (cu 15 pini), este folosit pentru a realiza conexiunea fizică între dispozitive. Acesta include pini pentru semnalele de culoare (roșu, verde, albastru), sincronizare și altele.
- **Rezoluția și Ratele de Refresh:** VGA definește diferite rezoluții și rate de refresh posibile pentru a afișa imagini pe ecranele monitorului.
- **Sincronizarea Orizontală și Verticală:** VGA utilizează semnale separate de sincronizare orizontală și verticală pentru a coordona modul în care monitorul afișează informațiile video.

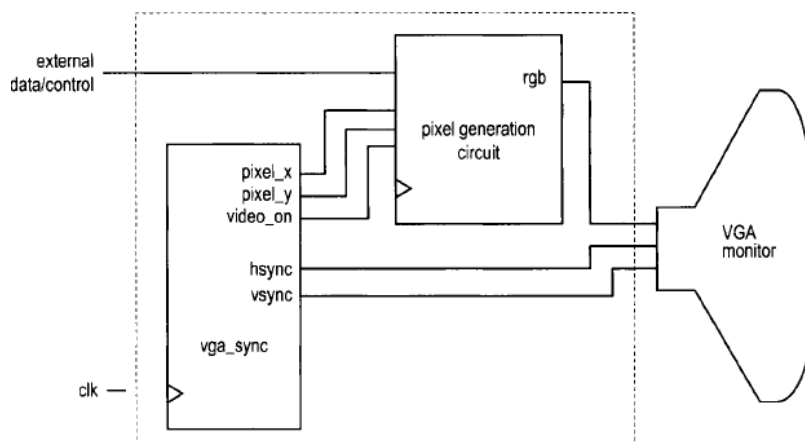
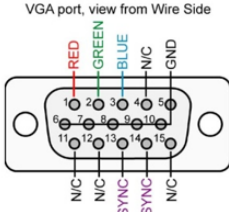



Fig. 5: Video controller pentru port VGA

Un video controller generează semnalele de sincronizare și transmite serial pixelii de date prin portul VGA al plăcii FPGA. Circuitul generator de semnale de sincronizare (*vga\_sync*) produce semnalele de timp și de sincronizare. Semnalele *hsync* și *vsync* sunt conectate la portul VGA pentru a controla scanarea pe orizontală și verticală. Cele două semnale sunt decodificate din numărătoarele interne, al căror output sunt semnalele *pixel\_x* și *pixel\_y*. Semnalele *pixel\_x* și *pixel\_y* indică poziția relativă a scanării și specifică locația curentă a pixelului. Circuitul *vga\_sync* generează semnalul *video\_on* pentru a indica dacă să activeze sau să dezactiveze afișajul. Circuitul de generare a pixelilor produce cele trei semnale video, denumite colectiv semnalul *rgb*. Culoarea este obținută ținând cont de coordonatele curente ale pixelului (semnalele *pixel\_x* și *pixel\_y*), de controlul extern și desemelele de date.

Placa FPGA Basys 3 pe care am utilizat-o pentru acest proiect are un port VGA integrat cu 10 semnale active, precum hsync, vsync, si semnalele video – 4bit red, 4bit green, 4 bit blue.

DB-15 Connector	VGA Signals	
	Vertical Sync(VS)	
	Horizontal Sync(HS)	
	Blue	
	Green	
	Red	

**Fig. 6:** Portul VGA

## Capitolul 3 – ANALIZĂ

### 3.1. Prezentare generală

În acest capitol, vom descrie modul de funcționare al aplicației noastre și modul prin care se va stabili conexiunea cu dispozitivele periferice, și anume mouse-ul, respectiv monitorul. Însă, pentru a înțelege mai bine modul de proiectare al aplicației noastre, vom începe cu o scurtă explicație a procesului de transmitere a datelor prin cele două interfețe de comunicare.

### 3.2. Modul de funcționare al protocolului PS/2

Microcontroller-ul se comportă ca o magistrală PS/2, iar mouse-ul conectat prin portul USE-HID utilizează cele 2 fire ale magistralei (clock și data) pentru a comunica cu placa FPGA. Mouse-ul trimite semnale pe clock și data către placă atunci când este mișcat, iar în caz contrar datele sunt menținute pe modul HIGH. La transmiterea unui octet, PS/2 transmite mai întâi bitul de START, urmat de 8 biți de date, urmați de bitul de STOP. Protocolul e repetat pentru fiecare octet din secvența care trebuie trimisă.

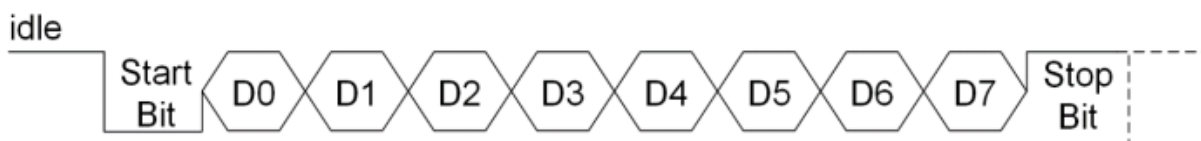


Fig. 7: Pachetul de date pentru comunicația serială

Câteva aspecte importante:

- Dacă nu este începută transmiterea, linia serială este în starea IDLE, pe 1
- Bitul START este întotdeauna 0, iar biții de date sunt transmiși în ordinea inversă a semnificației, primul este LSB iar ultimul este MSB
- Bitul STOP este întotdeauna 1
- Durata bitului STOP poate avea mai multe valori: 1, 1.5 sau 2 perioade de bit (1/ baud rate)
- Pe lângă biți de START și STOP se va folosi un bit adițional de paritate, împreună cu datele, pentru a detecta eventuale erori de transmisie.

Un mouse PS2 standard raportează mișcarea pe axa x (dreapta/stânga) și pe axa y (sus/jos) și starea butonului stâng, a butonului din mijloc și a butonului drept. Cantitatea fiecărei mișcări este înregistrată în contorul intern al mouse-ului. Atunci când datele sunt transmise către gazdă, contorul este resetat la zero și se reia numărătoarea. Conținutul contorului reprezintă un număr întreg cu semn pe 9 biți în care un număr pozitiv indică mișcarea spre dreapta sau în sus, iar un număr negativ indică mișcarea spre stânga sau în jos.

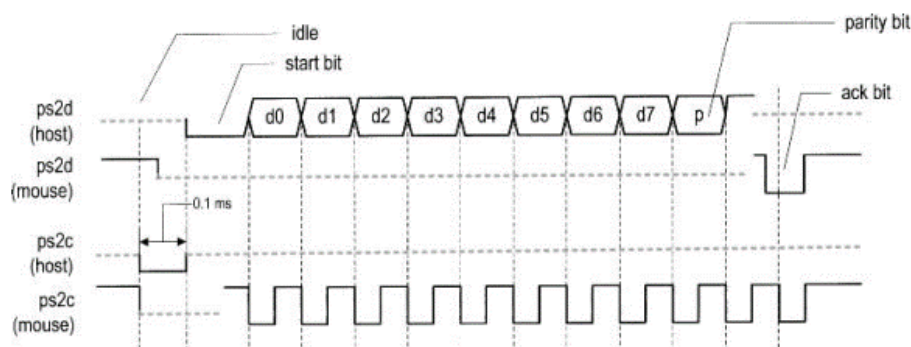


Fig. 8: Diagrama de timp pentru comunicarea gazdă – mouse în cazul protocolului PS/2

Relația dintre distanțele fizice este definită de parametrul de rezoluție al mouse-ului. Valoarea implicită a rezoluției este de patru numere pe milimetru. Atunci când un mouse se mișcă în mod continuu, datele sunt transmise într-un ritm regulat. Rata este definită de parametrul ratei de eșantionare a mouse-ului. Valoarea implicită a ratei de eșantionare este de 100 de eșantioane pe secundă. Dacă un mouse se mișcă prea repede, valoarea mișcării în timpul perioadei de eșantionare poate depăși intervalul maxim al contorului. Contorul este setat la magnitudinea maximă în direcția corespunzătoare. Doi biți de depășire sunt utilizați pentru a indica condițiile. Mouse-ul raportează mișcarea și activitățile butoanelor în 3 octeți, care sunt încorporați în trei pachete PS/2. Formatul detaliat al datelor de 3 octeți este prezentat Figura 9.

Pachetul de date transmis de mouse la un moment dat conține 33 de biți de informație în care sunt specificate butonul apăsător și direcția de deplasare a mouse-ului. Acești 33 de biți sunt partiționați în grupări de câte 11 biți. Aceste grupări încep cu un bit de start '0', urmat de 8 biți de informație, un bit de paritate impară și bitul de stop '1'. Transferul celor 33 de biți se poate observa în figura următoare:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							

Fig. 9: Pachetul de date de la mouse

Pentru început, vom analiza conținutul primului pachet de 11 biți. Așa cum se poate observa și din figura 9, primii biți primiți (bit 0 – 2) indică apăsarea butoanelor stânga, dreapta și mijloc ale mouse-ului. Următorii 2 biți primiți reprezintă biții de semn a coordonatelor de deplasare a mouse-ului pe axa X, respectiv pe axa Y, ceea ce ne va arăta direcția în care urmează să se miște mouse-ul. Ultimii 2 biți indică overflow-ul la mișcare.

Următoarele două pachete de câte 11 biți descriu coordonatele de mișcare a mouse-ului pe axa X, respectiv pe axa Y.

Astfel, aplicația noastră va trebui să primească toate cele 3 pachete de biți de informație de la mouse, să recunoască și să clasifice fiecare bit în funcție de scopul acestuia, și în cele din urmă, să proceseze toți biții pentru a transmite mai departe datele de intrare corecte către restul componentelor programului.

### 3.3. Protocolul de comunicare Host-to-PS/2-device

Protocolul de comunicare de la gazdă la dispozitivul PS/2 implică un schimb de date bidirecțional. Liniile de date și clock ale mouse-ului sunt, de fapt, circuite open-collector. În scopurile noastre de proiectare, le tratăm ca linii tri-state. Diagrama de sincronizare de bază a transmiterii unui pachet de la gazdă la un dispozitiv PS/2 este prezentată în Figura 8, în care semnalele de date și ceas sunt etichetate ps2d și ps2c. În scopul clarității, diagrama este împărțită în două părți pentru a arăta ce activități sunt generate de gazdă (adică cipul FPGA) și ce activități sunt generate de dispozitiv (adică mouse-ul).

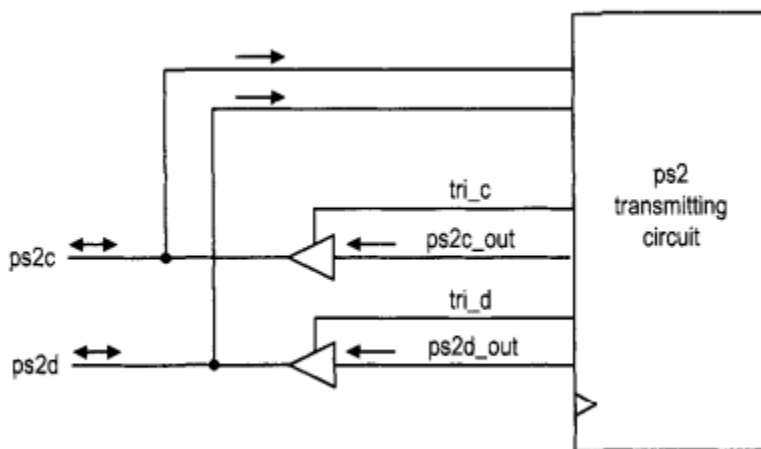


Fig. 10: Buffer-ele tri-state în subsistemul de transmisie PS/2

Secvența de operații de bază care are loc în cazul acestui protocol este:

1. Gazda forțează semnalul ps2c să fie '0' timp de cel puțin 100  $\mu$ s pentru a inhiba orice activitate a mouse-ului. Se poate considera că gazda solicită trimiterea unui pachet.
2. Gazda forțează semnalul ps2d să fie '0' și dezactivează linia ps2c (adică o face înaltă impedanță). Acest pas poate fi interpretat ca gazda trimițând un bit de start.



3. Dispozitivul PS/2 preia acum linia ps2c și este responsabil pentru generarea viitoare a semnalului de ceas PS/2. După detectarea bitului de start, dispozitivul PS2 generează o tranziție de la '1' la '0'.
4. Odată ce detectează tranziția, gazda deplasează cel mai puțin semnificativ bit de date pe linia ps2d. Menține această valoare până când dispozitivul PS/2 generează o tranziție de la '1' la '0' în linia ps2c, ceea ce confirmă practic recuperarea bitului de date.
5. Se repetă pasul 4 pentru ceilalți 7 biți de date și 1 bit de paritate.
6. După trimiterea bitului de paritate, gazda dezactivează linia ps2d (adică o face înaltă impedanță). Dispozitivul PS2 preia acum linia ps2d și confirmă finalizarea transmisiei aducând semnalul ps2d la '0'. Dacă este necesar, gazda poate verifica această valoare la ultima tranziție de la '1' la '0' în linia ps2c pentru a confirma că pachetul a fost transmis cu succes.

### 3.4. Sincronizarea cu portul VGA

Circuitul de sincronizare video generează semnalul *hsync*, care specifică timpul necesar pentru a parcurge (scana) o linie, și semnalul *vsync*, care specifică timpul necesar pentru a parcurge (scana) întregul ecran. Discuțiile ulterioare se bazează pe un ecran VGA de 640x480 cu o rată de pixeli de 25 MHz, ceea ce înseamnă că sunt procesați 25 de milioane de pixeli într-o secundă. Această rezoluție este cunoscută și sub denumirea de mod VGA.

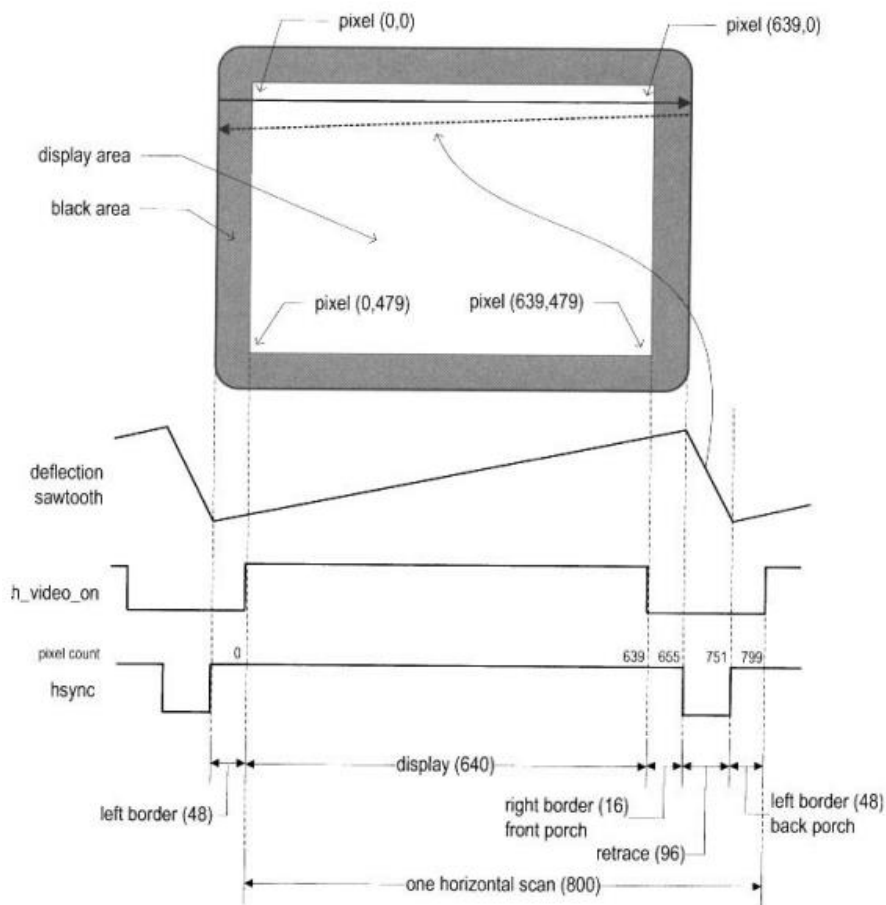
Ecranul unui monitor CRT include de obicei o mică margine neagră, așa cum este arătat în partea de sus a Figurii 11. Dreptunghiul din mijloc reprezintă porțiunea vizibilă. Coordonata axei verticale crește în jos. Coordonatele colțurilor stânga-sus și dreapta-jos sunt (0, 0) și (639, 479), respectiv.

#### 3.4.1. Sincronizarea orizontală

În Figura 11 este reprezentată o diagramă de timp detaliată pentru o scanare orizontală. O perioadă a semnalului *hsync* conține 800 de pixeli și poate fi divizată în patru regiuni:

- **Afisaj:** regiunea în care pixelii sunt efectiv afișați pe ecran. Lungimea acestei regiuni este de 640 de pixeli.
- **Retragere (retrace):** regiunea în care fasciculele de electroni se întorc la marginea stângă. Semnalul video ar trebui să fie dezactivat (adică negru), iar lungimea acestei regiuni este de 96 de pixeli.
- **Margine dreaptă:** regiunea care formează marginea dreaptă a regiunii de afișare. Este cunoscută și sub numele de *front porch* (porch before retrace). Semnalul video ar trebui să fie dezactivat, iar lungimea acestei regiuni este de 16 pixeli.

- **Margine stângă:** regiunea care formează marginea stângă a regiunii de afișare. Este cunoscută și sub numele de *back porch* (porch after retrace). Semnalul video ar trebui să fie dezactivat, iar lungimea acestei regiuni este de 48 de pixeli.



**Fig. 11:** Diagrama de timp a unei scanări orizontale

De reținut faptul ca lungimile marginilor din stânga și din dreapta pot varia pentru diferite mărci de monitoare.

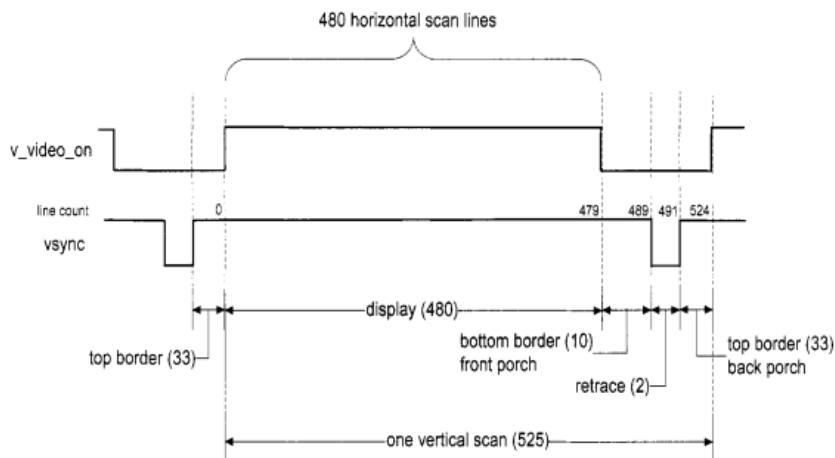
Semnalul *hsync* poate fi obținut cu ajutorul unui numărator special mod-800 și un circuit de decodificare. Timpii de numărare sunt marcați deasupra semnalului *hsync* în Figura 11. Numărătorul începe să numere intenționat de la începutul regiunii de afișare. Acest lucru ne permite să utilizăm ieșirea număratorului ca și coordonată orizontală (axa x). Această ieșire constituie semnalul pixel-x. Semnalul *hsync* devine '0' atunci când ieșirea contorului se află între 656 și 751.

De reținut faptul că monitorul CRT ar trebui să fie negru în marginile dreaptă și stângă și în timpul retragerii. Folosim semnalul *h\_video\_on* pentru a indica dacă coordonata orizontală curentă se află în regiunea afișabilă. Acesta este '1' doar atunci când numărul de pixeli este mai mic de 640.

### 3.4.2. Sincronizarea verticală

În timpul scanării verticale, fasciculele de electroni se deplasează treptat de sus în jos și apoi se întorc în partea de sus. Acest lucru corespunde timpului necesar pentru a reîmprospăta întregul ecran. Formatul semnalului *vsync* este similar cu cel al semnalului *hsync*, așa cum este arătat în Figura 12. Unitatea de timp a mișcării este reprezentată în termeni de linii de scanare orizontale. Un interval al semnalului *vsync* este de 525 de linii și poate fi împărțit în patru regiuni:

- **Afișaj:** regiunea în care liniile orizontale sunt efectiv afișate pe ecran. Lungimea acestei regiuni este de 480 de linii.
- **Retragere (retrace):** regiunea în care fasciculele de electroni se întorc în partea de sus a ecranului. Semnalul video ar trebui să fie dezactivat, iar lungimea acestei regiuni este de 2 linii.
- **Margine inferioară:** regiunea care formează marginea de jos a regiunii de afișare. Este cunoscută și sub numele de *front porch* (porch before retrace). Semnalul video ar trebui să fie dezactivat, iar lungimea acestei regiuni este de 10 linii.
- **Margine superioară:** regiunea care formează marginea de sus a regiunii de afișare. Este cunoscută și sub numele de *back porch* (porch after retrace). Semnalul video ar trebui să fie dezactivat, iar lungimea acestei regiuni este de 33 de linii.



**Fig. 12:** Diagrama de timp a unei scanări verticale

La fel ca în cazul scanării orizontale, lungimile marginilor superioare și inferioare pot varia pentru diferite mărci de monitoare.



Semnalul *vsync* poate fi obținut cu ajutorul unui numărator special mod-525 și un circuit de decodificare. Din nou, în mod intenționat, începem număratoarea de la începutul regiunii de afișare. Acest lucru ne permite să utilizăm ieșirea contorului ca și coordonată verticală (axa *y*). Această ieșire constituie semnalul *pixel\_y*. Semnalul *vsync* devine '0' atunci când numărul liniei este 490 sau 491.

La fel ca în scanarea orizontală, folosim semnalul *v\_video\_on* pentru a indica dacă coordonata verticală curentă se află în regiunea afișabilă. Este afirmat doar atunci când numărul liniei este mai mic de 480.

Pentru alte rezoluții vom avea alte valori pentru frecvența pixelilor, pentru margini și pentru valorile de retrace. Toate aceste valori se pot observa în tabelul de mai jos, tabel care ne va fi de folos la implementarea modului VGA:

Resolution	Refresh Rate (Hz)	Pixel Clock (MHz)	Horizontal (pixel clocks)				Vertical (rows)				Hsync Polarity	Vsync Polarity
			Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch		
640x350	70	25.175	640	16	96	48	350	37	2	60	p	n
640x350	85	31.5	640	32	64	96	350	32	3	60	p	n
640x400	70	25.175	640	16	96	48	400	12	2	35	n	p
640x400	85	31.5	640	32	64	96	400	1	3	41	n	p
640x480	60	25.175	640	16	96	48	480	10	2	33	n	n
640x480	73	31.5	640	24	40	128	480	9	2	29	n	n
640x480	75	31.5	640	16	64	120	480	1	3	16	n	n
640x480	85	36	640	56	56	80	480	1	3	25	n	n
640x480	100	43.16	640	40	64	104	480	1	3	25	n	p
720x400	85	35.5	720	36	72	108	400	1	3	42	n	p
768x576	60	34.96	768	24	80	104	576	1	3	17	n	p
768x576	72	42.93	768	32	80	112	576	1	3	21	n	p
768x576	75	45.51	768	40	80	120	576	1	3	22	n	p
768x576	85	51.84	768	40	80	120	576	1	3	25	n	p
768x576	100	62.57	768	48	80	128	576	1	3	31	n	p
800x600	56	36	800	24	72	128	600	1	2	22	p	p
800x600	60	40	800	40	128	88	600	1	4	23	p	p
800x600	75	49.5	800	16	80	160	600	1	3	21	p	p
800x600	72	50	800	56	120	64	600	37	6	23	p	p
800x600	85	56.25	800	32	64	152	600	1	3	27	p	p
800x600	100	68.18	800	48	88	136	600	1	3	32	n	p
1024x768	43	44.9	1024	8	176	56	768	0	8	41	p	p
1024x768	60	65	1024	24	136	160	768	3	6	29	n	n
1024x768	70	75	1024	24	136	144	768	3	6	29	n	n
1024x768	75	78.8	1024	16	96	176	768	1	3	28	p	p
1024x768	85	94.5	1024	48	96	208	768	1	3	36	p	p
1024x768	100	113.31	1024	72	112	184	768	1	3	42	n	p
1152x864	75	108	1152	64	128	256	864	1	3	32	p	p
1152x864	85	119.65	1152	72	128	200	864	1	3	39	n	p
1152x864	100	143.47	1152	80	128	208	864	1	3	47	n	p
1152x864	60	81.62	1152	64	120	184	864	1	3	27	n	p
1280x1024	60	108	1280	48	112	248	1024	1	3	38	p	p
1280x1024	75	135	1280	16	144	248	1024	1	3	38	p	p
1280x1024	85	157.5	1280	64	160	224	1024	1	3	44	p	p
1280x1024	100	190.96	1280	96	144	240	1024	1	3	57	n	p

Fig. 13: Tabelul valorilor pentru variațiile rezoluției



### 3.4.3. Calcularea timpului pentru semnalele de sincronizare VGA

Așa cum am menționat anterior, vom considera frecvența pixelilor egală cu 25MHz. Aceasta este determinată de trei parametri:

- $p$ : numărul de pixeli pe linia orizontală de scanare. Pentru o rezoluție 640 x 480, aceasta este:

$$p = 800 \frac{\text{pixels}}{\text{line}}$$

- $l$ : numărul de linii dintr-un ecran (adică dintr-o scanare verticală). Pentru o rezoluție 640 x 480, aceasta este:

$$l = 525 \frac{\text{lines}}{\text{screen}}$$

- $s$ : numărul de afișaje de ecran pe secundă. Pentru o funcționare fără clipire, îl putem seta la:

$$s = 60 \frac{\text{screens}}{\text{second}}$$

Parametrul  $s$  specifică cât de rapid ar trebui să fie reîmprospătat ecranul. Pentru ochiul uman, rata de reîmprospătare trebuie să fie de cel puțin 30 de afișaje de ecran pe secundă pentru a face ca mișcarea să apară continuă. Pentru a reduce clipirea, monitorul are în mod obișnuit o rată mult mai mare, cum ar fi specificația de 60 de ecrane pe secundă menționată mai sus. Rata de pixeli poate fi calculată cu ajutorul celor trei parametri:

$$\text{pixel rate} = p * l * s \approx 25M \frac{\text{pixels}}{\text{seconds}}$$

Rata de pixeli pentru alte rezoluții și rate de reîmprospătare poate fi calculată într-un mod similar. Evident, rata crește pe măsură ce rezoluția și rata de reîmprospătare cresc.

## Capitolul 4 – PROIECTARE

### 4.1. Diagramele bloc

#### 4.1.1. Diagrama arhitecturii sistemului

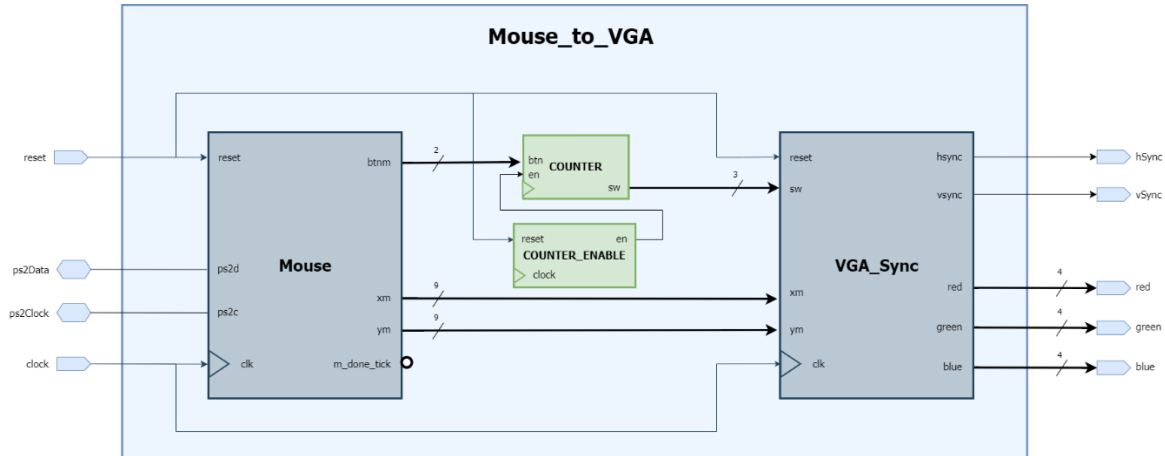


Fig. 14: Diagrama arhitecturii sistemului

#### 4.1.2. Diagramele bloc pentru protocolul de comunicare PS/2

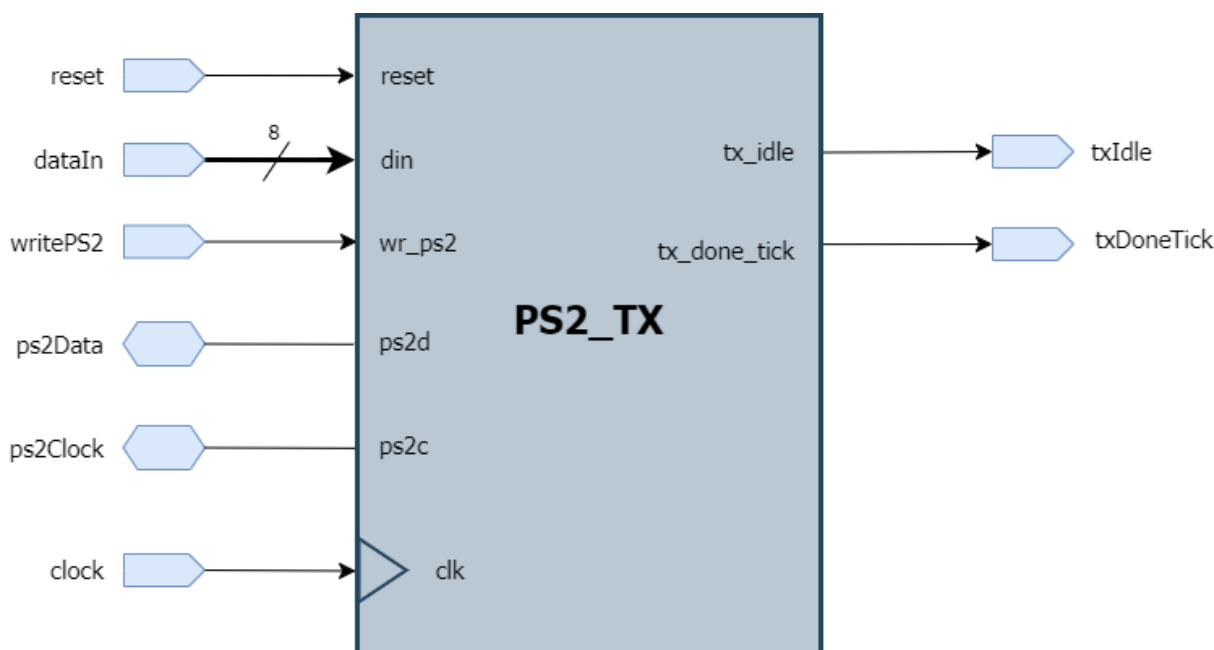
Pe baza explicațiilor din capitolul anterior, putem observa faptul că, pentru implementarea protocolului de comunicare PS/2, vom avea nevoie de:

- componentă de transmitere de date către mouse
- componentă de receptare a datelor provenite de la mouse
- componentă care să țină loc de unitate de execuție pentru cele 2 componente precizate anterior
- componentă care să țină loc de unitate de comandă pentru cele 2 componente precizate anterior

În acest sens, vom continua cu prezentarea structurii componentelor care asigură funcționarea conexiunii dintre mouse și placa Basys 3, fără a intra însă în detalii de implementare.

##### 4.1.2.1. Componenta de transmitere (PS2\_TX)

În figura următoare, se poate observa diagrama bloc a componentei PS2\_TX, al cărei rol este de a transmite date dinspre placa Basys 3 către mouse.



**Fig. 15:** Diagrama bloc – PS2\_TX

Din această figură, vedem că această componentă are două semnale inout, și anume *ps2d*, respectiv *ps2c*, consecință a faptului că modalitatea de comunicare dintre placă și mouse este bidirecțională, nu unidirecțională. Funcționarea acestei componente nu ține cont doar de semnalul de ceas intern al plăcii, ci și de cel al mouse-ului. În același timp, această componentă trebuie să forțeze și să suprascrie semnalele de clock și de date aparținând mouse-ului, pentru a-l anunța că dispozitivul gazdă este pregătit să primească date și că poate începe.

Pe lângă acestea, mai avem și un semnal de intrare *din* care ne furnizează date de intrare de la componenta de receptare. Semnalul de intrare *wr\_ps2* ne anunță dacă putem scrie date pe semnalele de clock și de date de la mouse, în funcție de comanda pe care o primim de la componenta care joacă rol de unitate de comandă. Nu în cele din urmă, avem un semnal de intrare *reset* care ne permite să resetăm toate semnalele.

Ca semnale de ieșiri, avem *tx\_idle*, care ne anunță dacă placa așteaptă să transmită date, respectiv *tx\_done\_tick*, care ne anunță dacă am terminat de transmis date.

#### 4.1.2.2. Componenta de receptare (PS2\_RX)

În figura următoare, se poate observa diagrama bloc a componentei PS2\_RX, care se ocupă cu receptarea datelor provenite de la mouse.

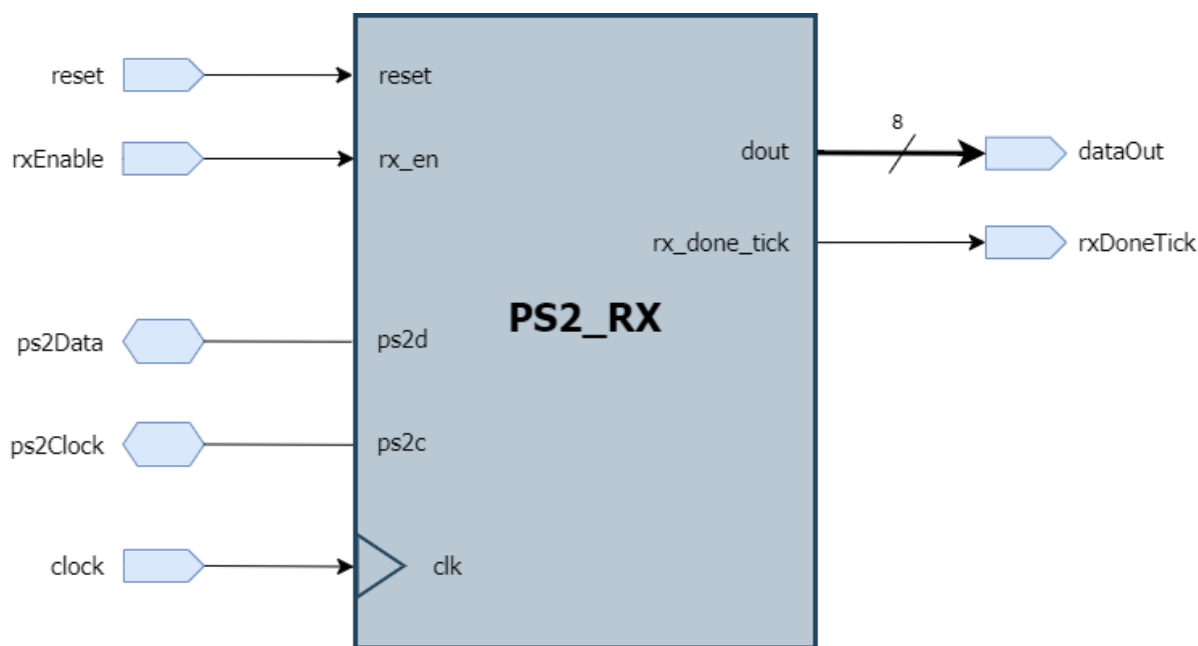


Fig. 16: Diagrama bloc – PS2\_RX

Pentru început, observăm cele două semnale inout, *ps2d*, respectiv *ps2c*, despre care am discutat în descrierea componentei anterioare și care au și aici același scop, cu excepția faptului că în cadrul acestei componente semnalele nu vor mai fi modificate de placă. Astfel, putem vedea aceste două semnale pur și simplu ca semnale de intrare.

Semnalul de intrare *rx\_en* ne anunță dacă putem citi semnale care provin de la mouse de pe liniile de date și clock și este controlat de componenta care joacă rol de unitate de comandă. Semnal de intrare *reset*, la fel ca în cazul componentei anterioare, ne permite să resetăm toate semnalele. Tot ca semnal de intrare avem și semnalul *clk*, prin care este furnizat semnalul de ceas intern al plăcii.

Ca semnale de ieșire, avem o magistrală *dout*, prin care sunt transmise mai departe datele filtrate ce provin de la mouse, adică care nu conțin și semnalele de start, stop și paritate, respectiv *rx\_done\_tick*, care ne anunță dacă am terminat de primit și prelucrat date.

#### 4.1.2.3. Componenta de execuție (PS2\_RXTX)

În figura următoare, se poate observa diagrama bloc a componentei PS2\_RXTX, care joacă rol de unitate de execuție pentru cele două componente menționate mai sus.



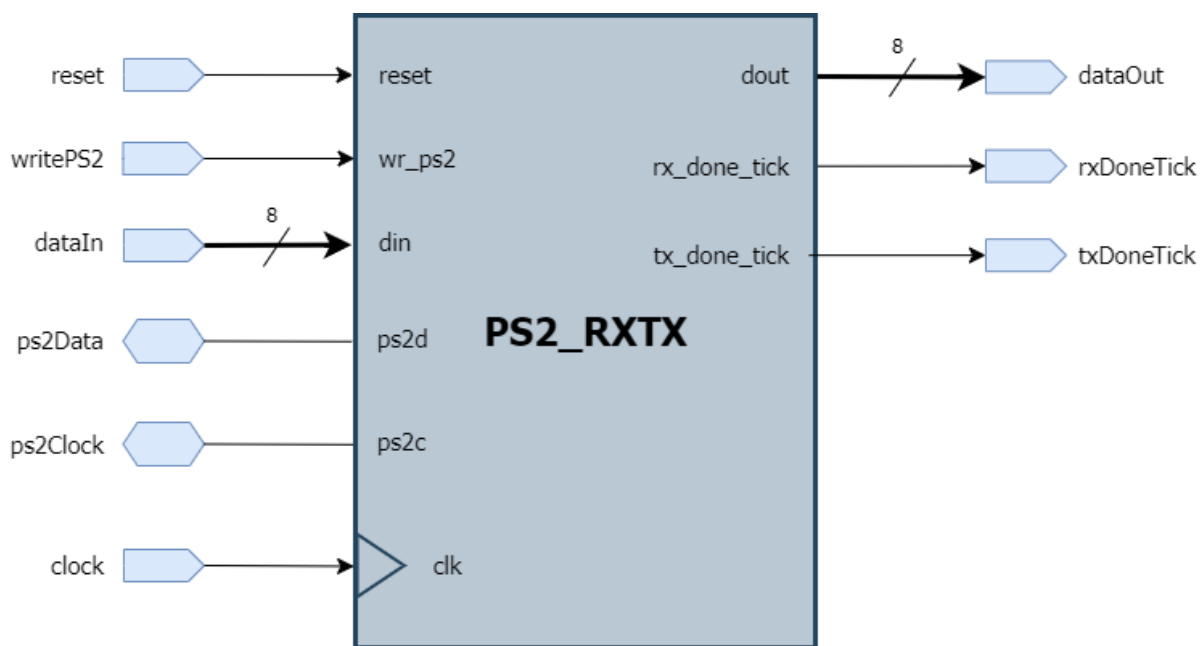


Fig. 17: Diagrama bloc – PS2\_RXTX

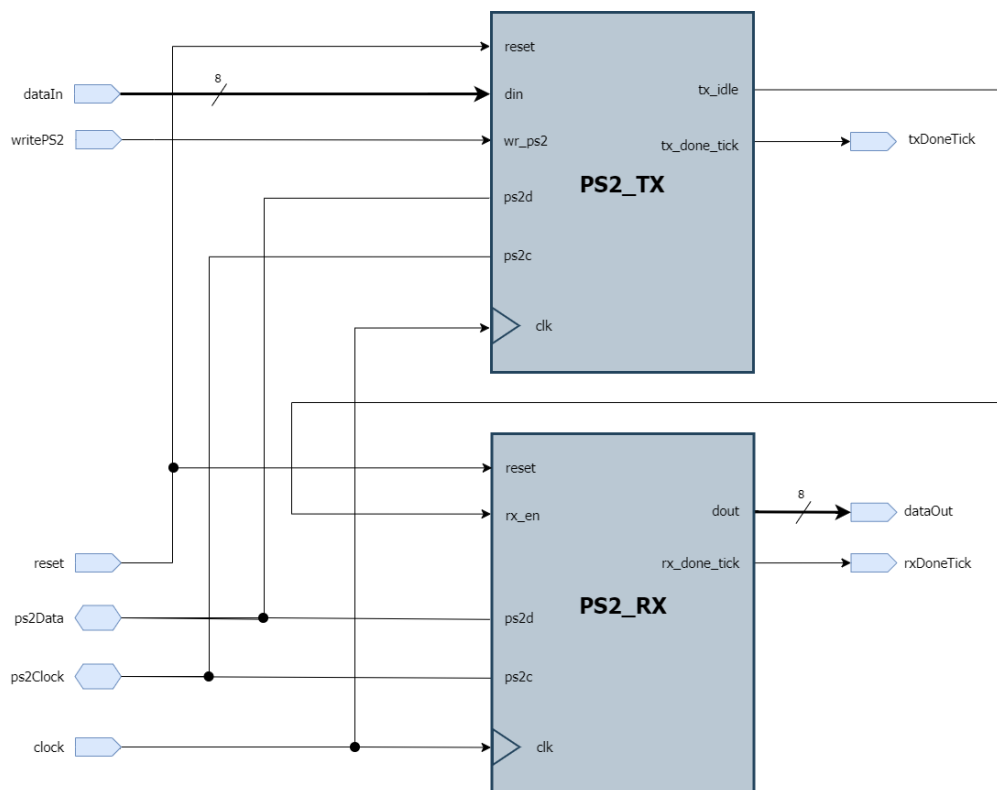


Fig. 18: Diagrama bloc a interfeței bidirecționale PS/2

#### 4.1.2.4. Componenta de comandă (Mouse)

În figura următoare, se poate observa diagrama bloc a componentei Mouse, care joacă rol de unitate de comandă pentru comunicarea dintre placă și mouse.

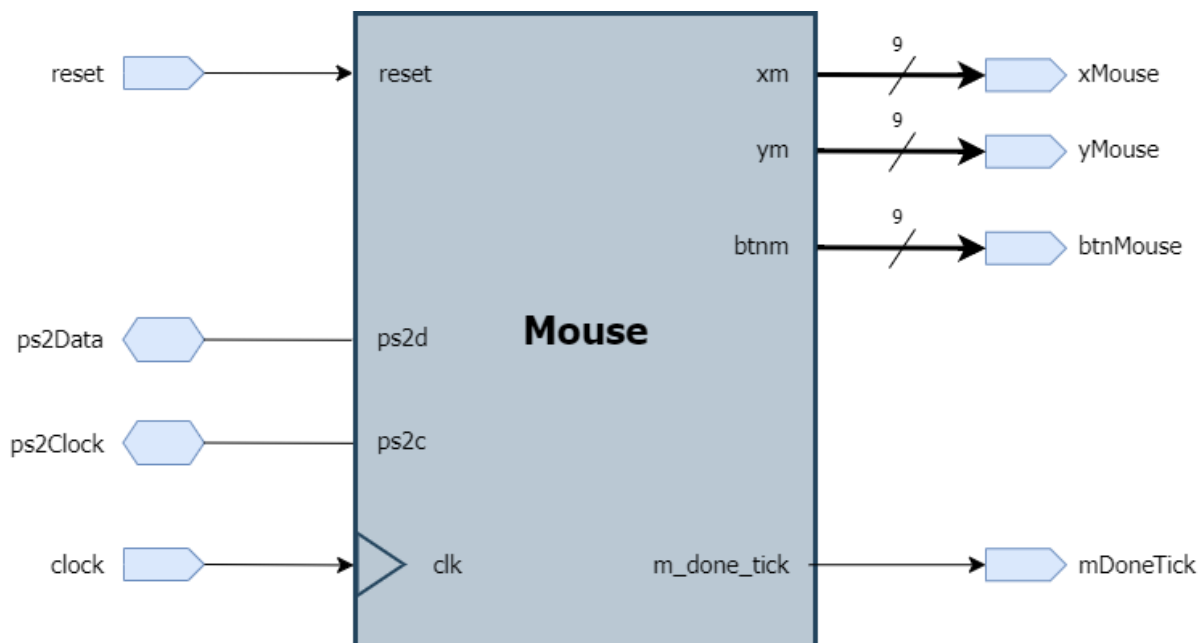


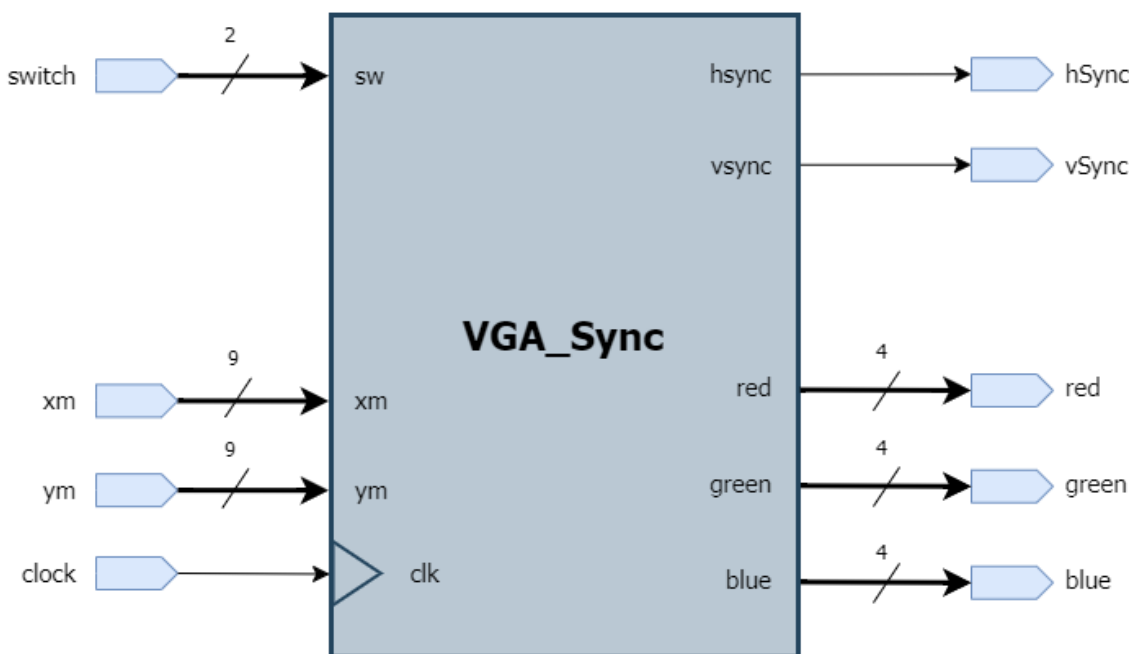
Fig. 19: Diagrama bloc – Mouse

Această componentă se folosește de toate componentele menționate mai sus pentru a primi și prelucra datele de la mouse, de care vom avea nevoie pentru modulul VGA. Ieșirile *xm*, *ym*, și *btnm* ne furnizează mai departe coordonatele de mișcare ale mouse-ului pe axa x și pe axa y și stările butoanelor mouse-ului. Semnalul de ieșire *m\_done\_tick* ne anunță când unitatea a terminat de procesat un nou pachet de informație de la mouse.

### 4.1.3. Diagramele bloc pentru modulul VGA

#### 4.1.3.1. Componenta VGA\_Sync

În figura următoare, se poate observa diagrama bloc a componentei VGA\_Sync, care pregătește semnalele ce urmează să fie transmise prin portul VGA și activează pixelii care vor afișa ceva din programul nostru.



**Fig. 20:** Diagrama bloc – VGA\_Sync

Ca semnale de intrare pentru această componentă avem  $xm$  și  $ym$ , care provin de la modulul mouse-ului. Ele redau coordonatele relative la poziția precedentă ale mouse-ului pe axa x, respectiv pe axa y. Semnalul de intrare  $switch$  are rolul de a selecta culoarea pentru întregul ecran, exceptând cursorul mouse-ului.

Semnalele de ieșire  $hSync$  și  $vSync$  au rolul de a activa pixelii vizibili ai ecranului în funcție de poziția lor pe orizontală și verticală. Culoarea pixelilor este transmisă adaptorului VGA prin semnalele de ieșire  $red$ ,  $green$  și  $blue$ .

Sursa semnalului de  $clock$  în cazul acestui modul este atipică față de celelalte module. Așa cum se poate observa în tabelul de la **Fig. 13** din capitolul “Analiză”, pentru o afișare corectă a cadrelor la o rată de refresh de 60Hz, având rezoluția de 1280x1024, avem nevoie de un semnal de clock pentru pixeli cu o frecvență de 108MHz. Fiindcă avem nevoie de un semnal de clock cu o frecvență mai mare decât cea de care dispune placa FPGA, trebuie să ne generăm un **Clocking Wizard** din IP Catalog. În Vivado, Clocking Wizard este o unealtă care te ajută să generezi și să configurezi blocuri de gestionare a semnalului de ceas într-un proiect FPGA. Cu această unealtă ne putem crea un semnal de clock cu o frecvență mai mare decât cea de care dispune componenta hardware.

#### 4.1.3.2. Componenta RGB\_VGA

În figura următoare, se poate observa diagrama bloc a componentei RGB\_VGA, care stabilește conexiunile între modulul VGA\_Sync și Clock Wizard-ul de 108MHz generat în aplicația Vivado.

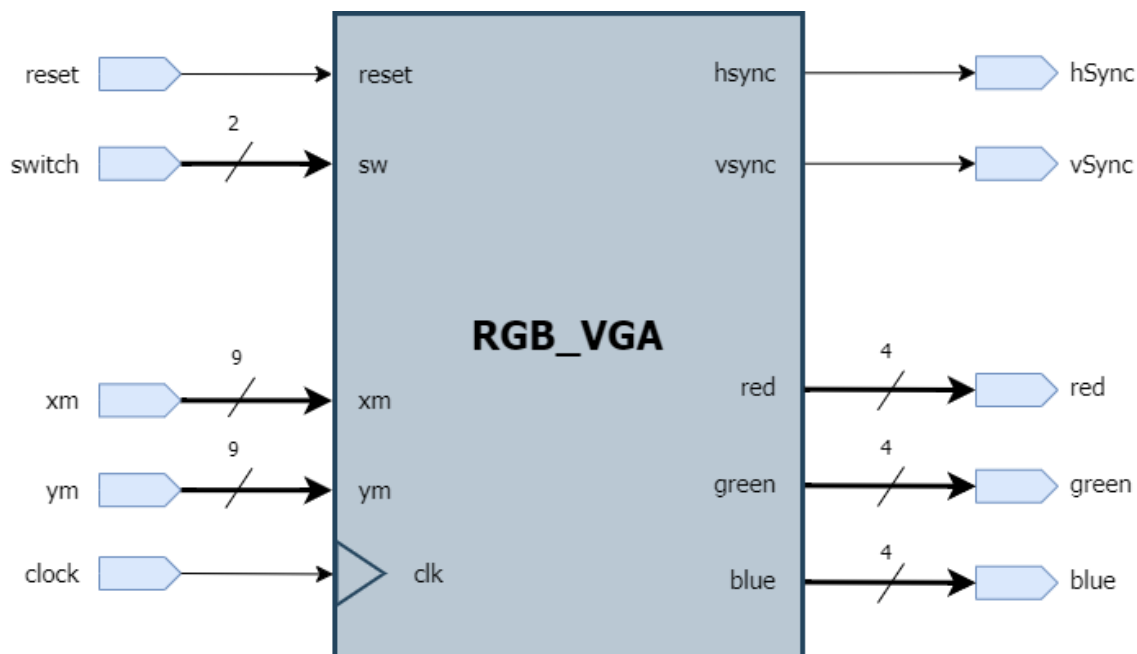


Fig. 21: Diagrama bloc – RGB\_VGA

După cum se poate observa, intrările și ieșirile coincid cu cele ale modulului VGA\_Sync, acesta din urmă fiind în componența modulului RGB\_VGA. Singurul semnal distinct este semnalul *reset*, această componentă fiind responsabilă de resetarea sistemului de procesare al semnalelor către portul VGA. O altă diferență este că semnalul de clock al acestui modul este semnalul intern de clock al plăcii FPGA, de 100MHz. În interiorul acestui modul se face conversia de frecvență pentru semnalul de ceas, de la 100MHz la 108MHz.

#### 4.1.3.3. Componenta Cursor

În figura următoare, se poate observa diagrama bloc a componentei Cursor, care este responsabilă de desenarea cursorului pe ecran.

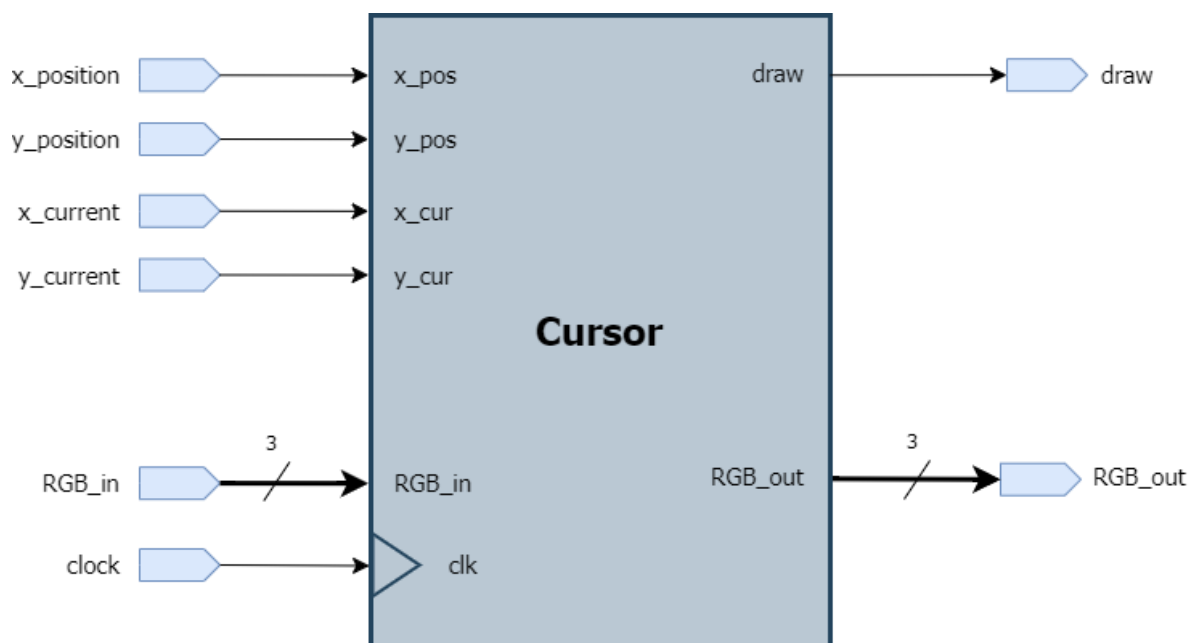


Fig. 21: Diagrama bloc – Cursor

Pentru a desena cursorul pe ecran, avem nevoie, mai întâi, de coordonatele unde ar trebui să fie desenat acesta. Aceste date sunt furnizate prin semnalele de intrare  $x\_pos$  și  $y\_pos$ . Semnalele de intrare  $x\_cur$  și  $y\_cur$  transmit coordonatele curente pe axa orizontală și pe axa verticală ale pixelului care urmează a fi desenat. Mai avem nevoie de asemenea de culoarea ecranului pentru completarea pixelilor din jurul cursorului, informație transmisă prin intrarea  $RGB\_in$ . Mai avem și un semnal de  $clock$ , a cărui frecvență este de 108MHz, acesta provenind de la modulul VGA\_Sync.

Prin ieșirea  $RGB\_out$ , se transmite culoarea fiecărui pixel care intră în componența cursorului către modulul VGA\_Sync. Semnalul de ieșire  $draw$  stabilește care pixeli trebuie să preia culoarea transmisă de la modulul Cursor.

#### 4.1.4. Diagramele bloc pentru componentele auxiliare

Pe lângă componentele prezentate anterior, mai avem și o serie de componente ajutătoare, mai mici, care fac operații mai simple, dar esențiale pentru o comunicare corectă între modulul mouse și modulul VGA.

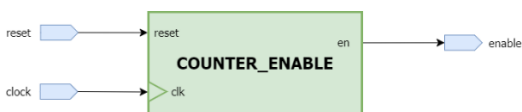


Fig. 22: Diagrama bloc – Counter enable

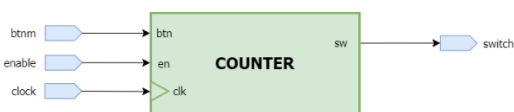


Fig. 23: Diagrama bloc – Counter



Counter este componenta care stabilește culoarea care pe care o va lua ecranul. Incrementarea se face odată cu apăsarea butonului stâng. Pentru că o simplă apăsare de buton poate transmite foarte multe semnale către acest modul, ceea ce ar duce în mod normal la un comportament neașteptat, avem semnalul de enable care permite incrementarea la o anumită perioadă de timp.

Counter\_enable stabilește activarea semnalului de enable pentru componenta counter. O dată la aproximativ 0.3s, semnalul en este activat pentru o perioadă de ceas.

#### 4.1. Organigrame

În cazul componentelor care implementează protocolul PS/2, implementarea acestora este structurată sub forma unor automate cu stări finite. În continuare sunt prezentate organigramele automatelor cu stări finite care descriu comportamentul fiecărui modul care participă la comunicarea dintre mouse și placa FPGA.

#### 4.1.1. Organigrama componentei PS2\_TX

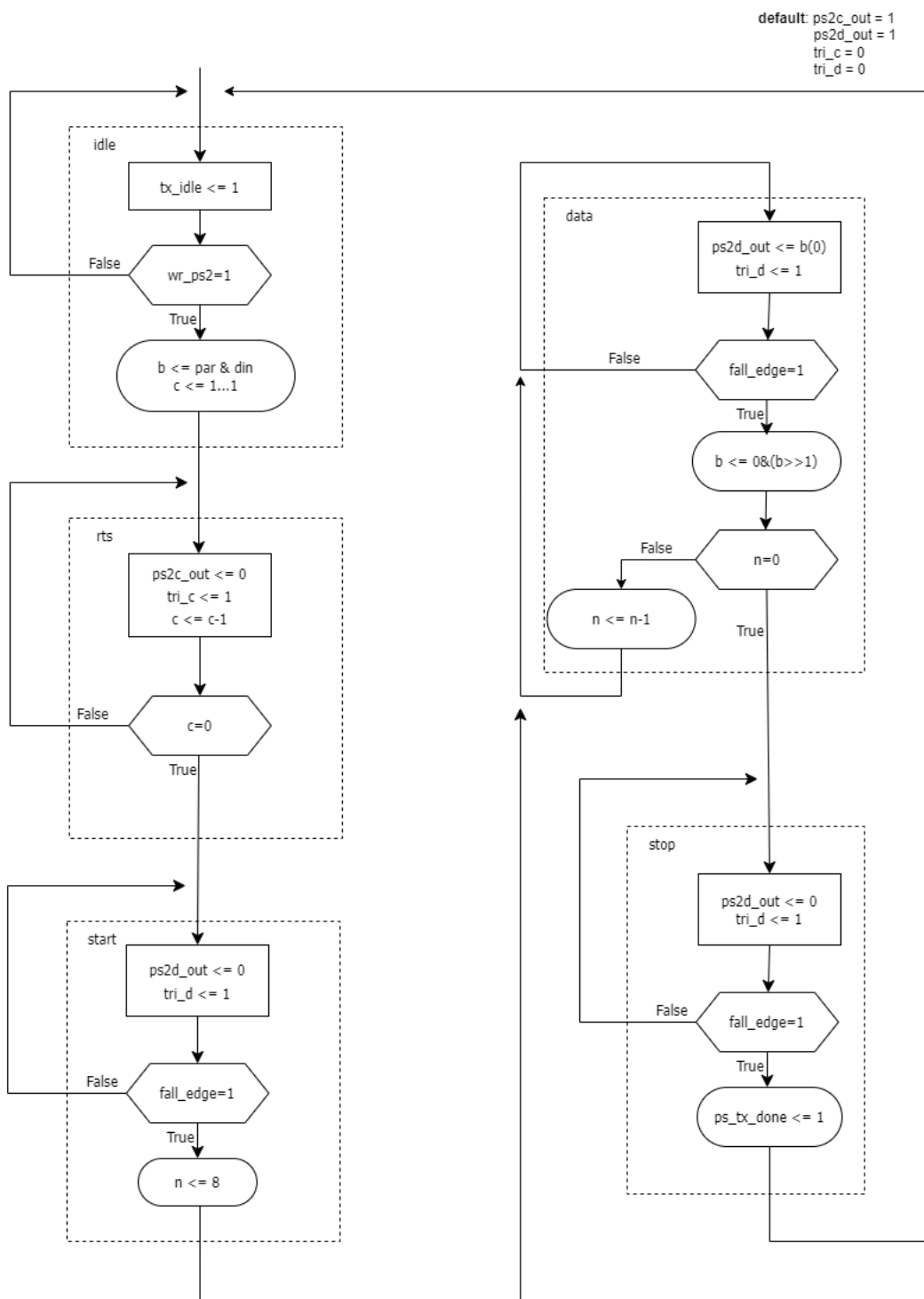


Fig. 24: Organigramă – PS2\_TX

#### 4.1.2. Organigrama componentei PS2\_RX

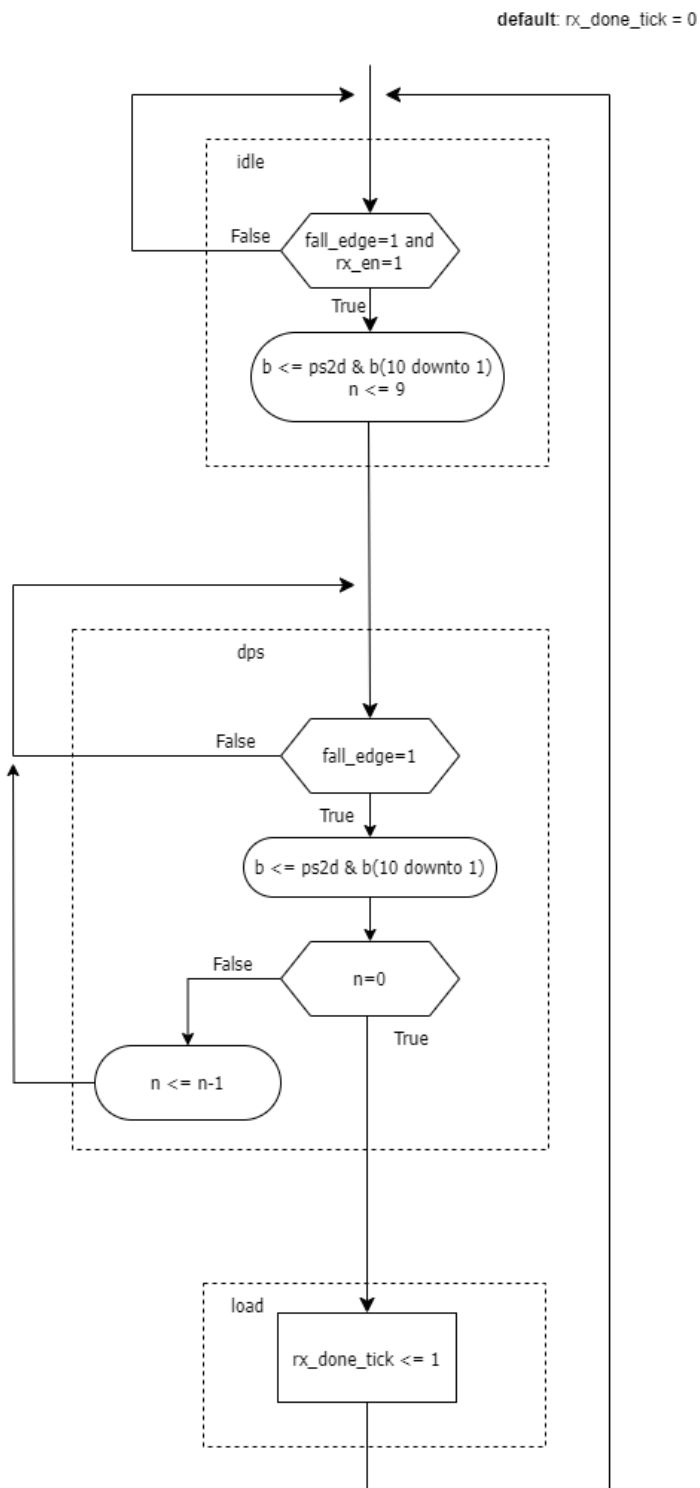


Fig. 25: Organigramă – PS2\_RX



#### 4.1.3. Organigrama automatului de stări finite pentru modul Mouse

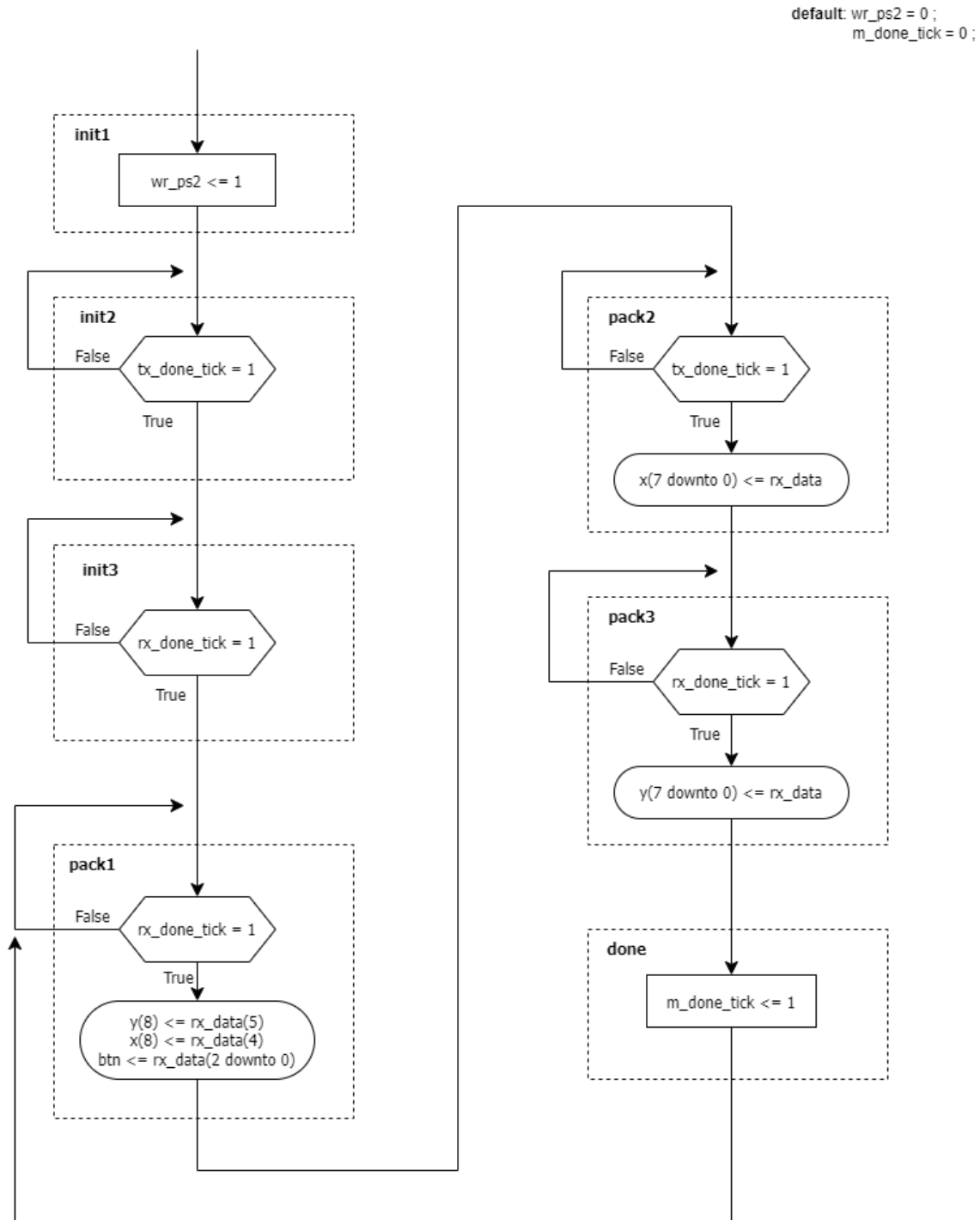


Fig. 26: Organigramă – FSM Mouse

## Capitolul 5 – IMPLEMENTARE

### 5.1. Modul Mouse

Pentru a descrie implementarea modului care asigură conexiunea cu mouse-ul, vom parcurge mai întâi componentele mai mici, pentru ca mai apoi să ne îndreptăm pas cu pas spre imaginea de ansamblu (vom merge dinspre “interior” înspre ”exterior”).

#### 5.1.1. PS2\_TX

În figura următoare este prezentată entitatea modului PS2\_TX. Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

##### Cod 1: Entitate PS2\_TX

```
entity ps2_tx is
  port (
    clk, reset      : in std_logic;
    din             : in std_logic_vector (7 downto 0) ;
    wr_ps2          : std_logic;
    ps2d, ps2c       : inout std_logic ;
    tx_idle         : out std_logic;
    tx_done_tick    : out std_logic
  ) ;
end ps2_tx;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

##### Cod 2: Arhitectura PS2\_TX

```
architecture arch of ps2_tx is
  type statetype is (idle, rts, start, data, stop); --FSM cu 5 stari
  signal state_reg , state_next: statetype;
  signal filter_reg , filter_next : std_logic_vector ( 7 downto 0 ) ;
  signal f_ps2c_reg ,f_ps2c_next: std_logic;
  signal b_reg , b_next : std_logic_vector (8 downto 0 ) ;
  signal c_reg , c_next : unsigned (12 downto 0) ;
  signal n_reg ,n_next : unsigned (3 downto 0) ;
  signal par: std_logic;
  signal tri_c , tri_d: std_logic ;
```



```
signal fall_edge : std_logic ;
signal ps2c_out , ps2d_out : std_logic;
begin
    -- filter and falling - edge tick generated
    -- on for ps2c
    -- .....
    .
    .
    .
    process (clk , reset) --acelasi lucru ca la rx
    begin
        if reset= '1' then
            filter_reg <= ( others => '0' ) ;
            f_ps2c_reg <= '0' ;
        elsif (clk'event and clk= '1' ) then
            filter_reg <= filter_next;
            f_ps2c_reg <= f_ps2c_next;
        end if;
    end process;

    filter_next <= ps2c & filter_reg ( 7 downto 1);
    f_ps2c_next <= '1' when filter_reg = "11111111" else
        '0' when filter_reg = "00000000" else
        f_ps2c_reg ;
    fall_edge <= f_ps2c_reg and ( not f_ps2c_next ) ;

    -- fsm d
    -- registers
    process ( clk , reset) --trecere in starea urmatoare la clock-ul
        procesorului si resetare
    begin
        if reset = '1' then
            state_reg <= idle ;
            c_reg <= ( others => '0' ) ;
            n_reg <= ( others => '0' ) ;
            b_reg <= ( others => '0' ) ;
        elsif ( clk'event and clk = '1' ) then
            state_reg <= state_next ;
            c_reg <= c_next ;
            n_reg <= n_next ;
            b_reg <= b_next ;
        end if ;
    end process ;

    -- odd parity bit
    par <= not ( din (7) xor din (6) xor din (5) xor din (4) xor
        din (3) xor din (2) xor din (1) xor din (0) ); --calculeaza
        bit-ul de paritate care ar trebui sa fie primit

    -- fsm d next - state logic and data path logic
    process ( state_reg , n_reg , b_reg , c_reg , wr_ps2 , din , par ,
        fall_edge )
    begin
        state_next <= state_reg ; --pregatim semnalele next cu cele
        saluate in registru
```



```
c_next <= c_reg ;
n_next <= n_reg ;
b_next <= b_reg ;
tx_done_tick <= '0' ;           --setam flag-ul pentru tx_done_tick
ps2c_out <= '1' ;               --trimitem 1 pe clock catre ps2
ps2d_out <= '1' ;               --trimitem 1 pe data catre ps2
tri_c <= '0' ;
tri_d <= '0' ;
tx_idle <= '0' ;                 --presupun ca tx nu este in asteptare
case state_reg is
  when idle =>
    tx_idle <= '1' ;             --tx este in starea idle(asteptare)
    if wr_ps2 = '1' then         --daca trebuie sa scriem
      b_next <= par & din ;      --trimitem datele de intrare si
bitul de paritate
      c_next <= ( others => '1' ) ; -- 2 ^ 13 - 1
      state_next <= rts ;        --trecem in starea urmatoare
    end if ;
  when rts => -- request to send
    ps2c_out <= '0' ;           --pe ps2 clock punem 0
    tri_c <= '1' ;
    c_next <= c_reg - 1 ;        --se forțează semnalul ps2c să fie '0'
timp de aprox. 100 µs
    if(c_reg = 0 ) then
      state_next <= start ;
    end if ;
  when start => -- assert start bit
    ps2d_out <= '0' ;           --trimitem bit-ul de start
    tri_d <= '1' ;
    if fall_edge = '1' then
      n_next <= "1000" ;
      state_next <= data ;
    end if ;
  when data => -- 8 data + 1 parity
    ps2d_out <= b_reg (0) ;
    tri_d <= '1' ;
    if fall_edge = '1' then
      b_next <= '0' & b_reg ( 8 downto 1); --pune 8 biti de
date de si bitul de paritate 0
      if n_reg = 0 then
        state_next <= stop ;
      else
        n_next <= n_reg - 1 ;
      end if ;
    end if ;
  when stop => -- punem high impedance pe ps2d
    if fall_edge = '1' then
      state_next <= idle ;
      tx_done_tick <= '1' ;
    end if ;
end case ;
end process ;
```

```
--tri_state_buffers
ps2c <= ps2c_out when tri_c = '1' else 'Z' ; --trimitem clock pe ps2
      daca tri_c este 1, altfel high impedance(no driver)
ps2d <= ps2d_out when tri_d = '1' else 'Z' ; --trimitem date pe ps2
      daca tri_d este 1, altfel high impedance(no driver)

end arch ;
```

Pentru început, se declară semnalele interne care ne vor ajuta pe parcurs și se declară stările automatului finit TX (a cărui organigramă a fost ilustrată în capitolul anterior). Pentru o parte din semnale avem un semnal *reg* și un semnal *next*. Acest lucru se datorează faptului că ne dorim să sincronizăm într-un fel clock-ul intern al plăcii cu cel de la mouse. În acest sens, semnalele din registrele interne (*reg*) se vor modifica doar pe frontul crescător al clock-ului intern și vor lua valoarea semnalelor *next*, în timp ce semnalele *next* se pot modifica la orice moment de timp.

Semnalele interne din acest modul sunt:

- *filter\_reg*, *filter\_next* : ajută la filtrarea corectă a semnalului de clock de la mouse și sincronizarea corectă a celor 2 semnale de clock;
- *f\_ps2c\_reg*, *f\_ps2c\_next*: același rol ca semnalul anterior și ajută la recunoașterea frontului descrescător al clock-ului de la mouse;
- *b\_reg*, *b\_next* : stocarea biților de date care vin de la mouse și a celor care trebuie trimiși înapoi;
- *c\_reg*, *c\_next* : ajută la contorizarea perioadei de aproximativ 100μs;
- *n\_reg*, *n\_next* : ajută la numărarea biților transmiși;
- *state\_reg*, *state\_next*: setează starea automatului la un moment de timp;
- *par*: reține bitul de paritate;
- *tri\_c*, *tri\_d*: three state buffer pentru clock și date;
- *fall\_edge* : activ când suntem pe frontul descrescător al clock-ului de la mouse.

În primul proces și în instrucțiunile până la al doilea proces se descrie comportamentul semnalelor de filtrare la activarea semnalului de *reset* și pe frontul crescător al semnalului de clock intern al plăcii. De asemenea, se face activarea semnalului pentru recunoașterea frontului descrescător al clock-ului de la mouse.

În al doilea proces se descrie comportamentul celorlalte semnale din regiștri la activarea semnalului de *reset* și pe frontul crescător al semnalului de clock intern al plăcii. După aceea, se calculează bitul de paritate cu datele primite pe linia *din*.

După aceea, urmează procesul care descrie comportamentul automatului de stări finite:

- *idle* : se activează semnalul *tx\_idle* și dacă primim semnal că trebuie să scriem, se pregătesc semnalele *b* și *c* și se trece în starea următoare;
- *request to send (rts)* : forțăm linia de clock a mouse-ului să fie 0 pentru aprox. 100μs;
- *start* : trimitem bitul de start pe linia de date și inițializăm semnalul *n* cu 8, deoarece avem 9 biți de date de transmis (8 de date + 1 de paritate);

- *data* : trimitem biții de date și cel de paritate;
- *stop* : trimitem bitul de stop și semnalăm că am terminat de transmis prin activarea semnalului *tx\_done\_tick*;

În final, se descrie comportamentul three state bufferului care poate fi ori activ pe 1 când se trimit forțează liniile de date și clock, ori în stare de înaltă impedanță.

### 5.1.2. PS2\_RX

În figura următoare este prezentată entitatea modulului PS2\_RX. Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

#### Cod 3: Entitate PS2\_RX

```
entity ps2_rx is
  port (
    clk, reset      : in std_logic;
    ps2d, ps2c      : in std_logic; -- keydata, keyclock
    rx_en           : in std_logic ;
    rx_done_tick    : out std_logic;
    dout            : out std_logic_vector (7 downto 0)) ;
end ps2_rx;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

#### Cod 4: Arhitectura PS2\_RX

```
architecture arch of ps2_rx is
  type statetype is (idle, dps, load); --FSM cu 3 stari
  signal state_reg , state_next: statetype;
  signal filter_reg , filter_next : std_logic_vector ( 7 downto 0 ) ; --
  -- filtru de 8 clock-uri (la fiecare clock de procesor sunt evaluati
  -- ultimii 8 biti de clock de la ps2c)
  signal f_ps2c_reg ,f_ps2c_next: std_logic; --ps2 clock filtrat (daca
  -- toti ultimele 8 semnale de clock sunt 1 atunci 1, pt toti 0 - 0,
  -- altfel semnalul de dinainte; ajuta doar la perceperea momentului de
  -- fall_edge)
```



```
signal n_reg , n_next : unsigned (3 downto 0) ;    --pentru
numararea bitilor care se salveaza
signal fall_edge : std_logic ;
signal ps2c_out , ps2d_out : std_logic;
begin
    -- filter and falling - edge tick generat
    i o n f o r p s 2 c
    . . . . .
    . . . . .
    process (clk , reset)  --acelasi lucru ca la rx
    begin
        if reset= '1' then
            filter_reg <= ( others => '0' ) ;
            f_ps2c_reg <= '0' ;
        elsif (clk'event and clk= '1' ) then
            filter_reg <= filter_next; --registul filter primeste ce
            este in filter_next pe risign edge la clock-ului procesorului
            f_ps2c_reg <= f_ps2c_next; --registul f_ps2c primeste ce
            este in f_ps2c_next pe risign edge la clock-ului procesorului
        end if;
    end process;

    filter_next <= ps2c & filter_reg ( 7 downto 1);
    f_ps2c_next <= '1' when filter_reg = "11111111" else
        '0' when filter_reg = "00000000" else
        f_ps2c_reg ;
    fall_edge <= f_ps2c_reg and ( not f_ps2c_next ) ;

    -- f s m d t o e x t r a c t t h e 8 - b i t d a t a
    -- registers
    process ( clk , reset)  --trecere in starea urmatoare la clock-ul
                                procesorului si resetare
    begin
        if reset = '1' then
            state_reg <= idle ;
            n_reg <= ( others => '0' ) ;
            b_reg <= ( others => '0' ) ;
        elsif ( clk'event and clk = '1' ) then
            state_reg <= state_next ;
            n_reg <= n_next ;
            b_reg <= b_next ;
        end if ;
    end process ;

    -- f s m d n e x t - s t a t e   l o g i c a n d d a t a p a t h   l o
g i c
    process ( state_reg , n_reg,b_reg, fall_edge , rx_en , ps2d )
    begin
        rx_done_tick <= '0' ;
        state_next <= state_reg ;  --state_next primeste state_reg
cand se schimba ceva din lista de sensibilitate
        n_next <= n_reg ;
        b_next <= b_reg ;
```

```

case state_reg is
  when idle =>
    if fall_edge = '1' and rx_en = '1' then --daca suntem in
fall_edge si rx e enable, atunci luam date de la ps2d in b_next
      --s h i f t   i n   s t a r t   b i t
      b_next <= ps2d & b_reg ( 10 downto 1); --salvam bitul de
start
      n_next <= "1001" ; --pregatim cei 10 biti de date (8
data + 1 parity + 1 stop) sa fie salvati, cred
      state_next <= dps ; --trecem la starea urmatoare de
salvare a datelor
    end if ;
    when dps => -- 8 d a t a + 1 p a r i t y + 1 s t o p
      if fall_edge = '1' then
        b_next <= ps2d & b_reg ( 10 downto 1 ) ; --chiar daca
noi i-am dat n=9, se fac 10 evaluari, astfel ca se salveaza 10 biti de
date
        if n_reg = 0 then --daca am terminat de salvat date
          state_next <= load ; --trecem in starea urmatoare
        else --altfel
          n_next <= n_reg - 1; --decrementam numarul de biti
care mai trebuie primiti
        end if ;
      end if ;
    when load =>
      -- 1 e x t r a   c l o c k   t o   c o m p l e t e   t h e   l a s t
s h i f t
      state_next <= idle ; --trecem inapoi la starea de asteptare
      rx_done_tick <='1'; --anuntam ca am terminat de salvat un
pachet de date
    end case ;
  end process ;

  -- o u t p u t
  --trimitem mai departe doar bitii de la 1 la 8 (adica fara start,
stop si parity)
  dout <= b_reg ( 8 downto 1); -- d a t a b i t s

end arch ;

```

Pentru început, se declară semnalele interne care ne vor ajuta pe parcurs și se declară stările automatului finit RX (a cărei organigramă a fost ilustrată în capitolul anterior). Pentru o parte din semnale avem un semnal *reg* și un semnal *next*, al căror scop este același ca în cazul celor din modulul anterior.

Semnalele interne din acest modul sunt:

- *filter\_reg*, *filter\_next* : ajută la filtrarea corectă a semnalului de clock de la mouse și sincronizarea corectă a celor 2 semnale de clock;
- *f\_ps2c\_reg*, *f\_ps2c\_next*: același rol ca semnalul anterior și ajută la recunoașterea frontului descrescător al clock-ului de la mouse;



- *b\_reg* , *b\_next* : stocarea biților de date care vin de la mouse și a celor care trebuie trimiși înapoi;
- *n\_reg* , *n\_next* : ajută la numărarea biților transmiși;
- *state\_reg* , *state\_next*: setează starea automatului la un moment de timp;
- *fall\_edge* : activ când suntem pe frontul descrescător al clock-ului de la mouse.

În primul proces și în instrucțiunile până la al doilea proces se petrece același lucru ca în modulul PS2\_TX, prezentat anterior.

În al doilea proces se descrie comportamentul celorlalte semnale din regiștri la activarea semnalului de *reset* și pe frontul crescător al semnalului de clock intern al plăcii.

După aceea, urmează procesul care descrie comportamentul automatului de stări finite:

- *idle* : dacă suntem pe frontul descrescător al clock-ului de la mouse și putem recepționa date, se inițializează semnalul *n* cu 9 și *b* primește bitul de start;
- *data parity stop (dps)*: reținem biții de date, bitul de paritate și bitul de stop;
- *load* : semnalăm că am terminat de transmis prin activarea semnalului *rx\_done\_tick* și ne întoarcem în starea *idle*;

În final, se pun biții de pe magistrala *b* pe ieșirea *dout*.

### 5.1.3. PS2\_RXTX

În figura următoare este prezentată entitatea modulului PS2\_RXTX. Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

#### Cod 5: Entitate PS2\_RXTX

```
entity ps2_rxtx is
  port (
    clk, reset      : in std_logic;
    wr_ps2          : std_logic;
    din             : in std_logic_vector ( 7 downto 0 ) ;
    dout            : out std_logic_vector ( 7 downto 0 ) ;
    rx_done_tick    : out std_logic ;
    tx_done_tick    : out std_logic ;
    ps2d, ps2c      : inout std_logic
  ) ;
end ps2_rxtx;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

#### Cod 6: Arhitectura PS2\_RXTX

```
architecture arch of ps2_rxtx is
    signal tx_idle : std_logic;
begin
    ps2_tx_unit : entity work.ps2_tx
        port map(clk=>clk, reset=>reset , wr_ps2=>wr_ps2,
            din=>din, ps2d=>ps2d, ps2c=>ps2c,
            tx_idle=>tx_idle, tx_done_tick=>tx_done_tick);
    ps2_rx_unit : entity work.ps2_rx(arch)
        port map(clk=>clk , reset=>reset , rx_en=>tx_idle,
            ps2d=>ps2d, ps2c=>ps2c,
            rx_done_tick=>rx_done_tick, dout=>dout);
end arch;
```

Așa cum am precizat și în capitolul anterior, această componentă joacă rolul de unitate de execuție pentru modulul mouse-ului. Rolul ei este doar de a instanția cele două componente RX și TX și de a face conexiunile între ele.

#### 5.1.4. MOUSE (Automatul de stări finite al modulului Mouse)

În figura următoare este prezentată entitatea modulului Mouse, care reprezintă unitatea de comandă a implementarea protocolului PS/2. Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

#### Cod 7: Entitate MOUSE

```
entity mouse is
    port (
        clk, reset      : in std_logic;
        ps2d, ps2c      : inout std_logic ;
        xm, ym          : out std_logic_vector ( 8 downto 0 ) ;
        btnm            : out std_logic_vector (2 downto 0 ) ;
        m_done_tick     : out std_logic) ;
end mouse;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.



### Cod 8: Arhitectura MOUSE

```
architecture arch of mouse is

    constant STRM: std_logic_vector ( 7 downto 0 ) := "11110100" ;
    -- stream command F4
    type state_type is (init1, init2, init3, pack1 , pack2, pack3, done) ;
    signal state_reg , state_next : state_type;
    signal rx_data: std_logic_vector ( 7 downto 0 ) ;
    signal rx_done_tick , tx_done_tick: std_logic;
    signal wr_ps2 : std_logic;
    signal x_reg , y_reg : std_logic_vector ( 8 downto 0 ) ;
    signal x_next , y_next : std_logic_vector ( 8 downto 0 ) ;
    signal btn_reg , btn_next : std_logic_vector ( 2 downto 0 ) ;

begin
    -- i n s t a n t i a t i o n
    ps2_rxtx_unit : entity work.ps2_rxtx (arch)
    port map(clk=>clk , reset=>reset , wr_ps2=>wr_ps2,
            din=>STRM , dout=>rx_data ,
            ps2d=>ps2d, ps2c=>ps2c,
            rx_done_tick=>rx_done_tick,
            tx_done_tick=>tx_done_tick);
    -- s t a t e a n d d a t a r e g i s t e r s
    process (clk, reset)
    begin
        if reset = '1' then
            state_reg <= init1;
            x_reg <= ( others => '0' ) ;
            y_reg <= ( others => '0' ) ;
            btn_reg <= ( others => '0' ) ;
        elsif (clk'event and clk='1') then
            state_reg <= state_next ;
            x_reg <= x_next;
            y_reg <= y_next;
            btn_reg <= btn_next ;
        end if ;
    end process ;
    -- n e x t - s t a t e l o g i c
    process (state_reg ,rx_done_tick, tx_done_tick, x_reg, y_reg,
            btn_reg, rx_data)
    begin
        wr_ps2 <= '0' ;
        m_done_tick <= '0' ;
        x_next <= x_reg;
        y_next <= y_reg;
        btn_next <= btn_reg ;
        state_next <= state_reg;
        case state_reg is
            when init1 =>
                wr_ps2 <= '1';
                state_next <= init2 ;
```

```
when init2 => -- wait for send to complete
  if tx_done_tick = '1' then
    state_next <= init3 ;
  end if ;
when init3 => -- wait for acknowledge packet
  if rx_done_tick = '1' then
    state_next <= pack1 ;
  end if ;
when pack1 => -- wait for 1st data packet
  if rx_done_tick = '1' then
    state_next <= pack2 ;
    y_next ( 8 ) <= rx_data ( 5 ) ;
    x_next (8) <= rx_data ( 4 ) ;
    btn_next <= rx_data ( 2 downto 0 ) ;
  end if ;
when pack2 => -- wait for 2nd data packet
  if rx_done_tick = '1' then
    state_next <= pack3 ;
    x_next ( 7 downto 0 ) <= rx_data ;
  end if ;
when pack3 => -- wait for 3rd data packet
  if rx_done_tick = '1' then
    state_next <= done ;
    y_next ( 7 downto 0 ) <= rx_data ;
  end if ;
when done =>
  m_done_tick <= '1' ;
  state_next <= pack1 ;
end case ;
end process ;
xm <= x_reg ;
ym <= y_reg ;
btnm <= btn_reg ;
end arch ;
```

Pentru început, se declară semnalele interne care ne vor ajuta pe parcurs și se declară stările automatului finit de stări pentru unitatea de comandă a protocolului PS/2 (a cărui organigramă a fost ilustrată în capitolul anterior). Pentru o parte din semnale avem un semnal *reg* și un semnal *next*, al căror scop este același ca în cazul celor din modulele anterioare.

Semnalele interne din acest modul sunt:

- *rx\_data* : îi este asignat semnalul care vine de la componenta RX;
- *rx\_done\_tick*, *tx\_done\_tick* : semnalează dacă recepționarea sau transmiterea datelor s-a efectuat;
- *wr\_ps2* : este folosit pentru a anunța dispozitivul gazdă că poate transmite date la mouse;
- *x\_reg*, *x\_next* : rețin biții care formează coordonata pe axa x;
- *y\_reg*, *y\_next* : rețin biții care formează coordonata pe axa y;

- *btn\_reg, btn\_next* : rețin biții care arată starea butoanelor de pe mouse;
- *state\_reg, state\_next*: setează starea automatului la un moment de timp;

În continuare, este instanțiat modulul PS2\_RXTX. Ce este de observat la acest lucru este faptul că pe linia de date *din* s-a asignat constanta *STRM*. Aceasta va pune pe linia de date codificarea aferentă comenzii F4.

Următorul pas este descrierea comportamentului semnalelor la apăsarea butonului de *reset*, respectiv pe frontul crescător al semnalului de ceas al plăcii FPGA.

Mai apoi, urmează procesul care descrie comportamentul automatului de stări:

- *init1* : se activează semnalul *wr\_ps2*, ceea ce va porni trimiterea datelor de la placa FPGA la mouse;
- *init2* : se așteaptă până la finalizarea completă a trimiterii;
- *init3* : se așteaptă până la primirea primului pachet de date de la mouse;
- *pack1* : se primește primul pachet de date de la mouse și se rețin biții de semn pentru coordonatele de poziție și starea butoanelor de pe mouse;
- *pack2* : se primește al doilea pachet de date de la mouse și se rețin coordonatele fără semn pe axa x;
- *pack3* : se primește al treilea pachet de date de la mouse și se rețin coordonatele fără semn pe axa y;
- *done* : se semnalează faptul că s-a primit cu succes un pachet întreg de date de la mouse și execuția se întoarce în starea *pack1* pentru a începe primirea următorului pachet.

## 5.2. Modul VGA

Pentru a descrie implementarea modulului VGA, vom face aceeași parcurgere a componentelor care îl alcătuiesc, dinspre “interior” înspre ”exterior”.

### 5.2.1. VGA\_SYNC (Circuitul de sincronizare VGA)

În figura următoare este prezentată entitatea modulului circuitului de sincronizare VGA. Semnalele din entitate sunt aceleași cu cele din diagrama componente din capitolul anterior.

### Cod 9: Entitate VGA\_SYNC

```
entity vga_sync is
  port (
    clk      : IN STD_LOGIC;
    hsync     : OUT STD_LOGIC;
    vsync     : OUT STD_LOGIC;
    xm       : IN STD_LOGIC_VECTOR(8 downto 0) ;
    ym       : IN STD_LOGIC_VECTOR(8 downto 0) ;
    r        : OUT STD_LOGIC_VECTOR(3 downto 0);
    g        : OUT STD_LOGIC_VECTOR(3 downto 0);
    b        : OUT STD_LOGIC_VECTOR(3 downto 0);
    s        : IN STD_LOGIC_VECTOR(2 downto 0)) ;
end vga_sync;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

### Cod 10: Arhitectura VGA\_SYNC

```
architecture arch of vga_sync is
  -----1280x1024 @ 60 Hz pixel clock 108 MHz
  signal RGB : STD_LOGIC_VECTOR(2 downto 0);
  signal CURS_X : INTEGER range 0 to 1688:=1048;
  signal CURS_Y : INTEGER range 0 to 1688:=554;
  signal DRAW : STD_LOGIC := '0';
  signal HPOS : INTEGER range 0 to 1688:=0;
  signal VPOS : INTEGER range 0 to 1066:=0;

begin

  cursor_draw_gen : entity WORK.cursor port map (
    clk => clk,
    x_cur => hpos,
    y_cur => vpos,
    x_pos => curs_x,
    y_pos => curs_y,
    RGB_in => s,
    RGB_out => rgb,
    draw => draw);
```



```
process(clk)
begin
  if(clk'event and clk='1')then
    if(draw = '1') then
      r <= (others => rgb(0));
      g <= (others => rgb(1));
      b <= (others => rgb(2));
    else
      r <= (others => s(0));
      g <= (others => s(1));
      b <= (others => s(2));
    end if;
    if(hpos<1688)then
      hpos<=hpos+1;
    else
      hpos<=0;
      if(vpos<1066)then
        vpos<=vpos+1;
      else
        vpos<=0;
        if(xm(8)='0' and xm>"0000011")then
          curs_x <= curs_x+conv_integer(xm(7 downto 0));
        elsif (xm(8)='1' and not(xm(7 downto 0)-1) > "0000011")
          then
          curs_x <= curs_x-conv_integer(not(xm(7 downto 0)-1));
        end if;
        if(ym(8)='0' and ym > "0000011")then
          curs_y <= curs_y-conv_integer(ym(7 downto 0));
        elsif (ym(8)='1' and not(ym(7 downto 0)-1) > "0000011")
          then
          curs_y <= curs_y+conv_integer(not(ym(7 downto 0)-1));
        end if;
      end if;
    end if;
    if((hpos>0 and hpos<408) or (vpos>0 and vpos<42))then
      r<=(others=>'0');
      g<=(others=>'0');
      b<=(others=>'0');
    end if;
    if(hpos>48 and hpos<160)then----hsync
      hsync<='0';
    else
      hsync<='1';
    end if;
    if(vpos>0 and vpos<4)then-----vsync
      vsync<='0';
    else
      vsync<='1';
    end if;
  end if;
end process;
```

La începutul arhitecturii modulului sunt declarate semnalele interne:

- *rgb* : reține semnalul care descrie culoarea transmisă de modulul Cursor
- *curs\_x, curs\_y* : păstrează coordonatele mouse-ului relative la ecran;
- *draw* : este activ atunci când trebuie să desenăm cursorul;
- *hpos* : reține poziția pe orizontală a pixelului curent;
- *vpos* : reține poziția pe verticală a pixelului curent;

Urmează instanțierea modulului Cursor, responsabil cu desenarea cursorului.

În continuare, este descris procesul care stabilește activarea pixelilor și colorarea lor. Acest proces este dependent de semnalul de clock de 108MHz. Mai întâi, se stabilește culoarea fiecărui pixel. În cazul în care semnalul *draw* este activ, pixelul face parte din cursor și ia informațiile din semnalul *rgb*, adică de la modulul Cursor. Altfel, pixelul face parte din fundal și va primi date de la semnalul *sw*.

Condiția următoare stabilește limitele semnalelor *hpos* și *vpos*, limite descrise de dimensiunea ecranului măsurată în pixeli. Semnalul *hpos* poate crește până la valoare 1687, după care se întoarce la 0. Odată cu resetarea semnalului *hpos* se incrementează semnalul *vpos*, care poate ajunge până la valoare 1066. După parcurgerea întregului ecran, adică atunci când se resetează componenta *vpos*, se actualizează coordonatele mouse-ului relative la ecran. Actualizarea se face prin adăugarea, respectiv scăderea coordonatelor relative la poziția anterioară, provenite de la mouse, în funcție de semnul coordonatei (bitul 8). De asemenea, am observat că se rețin anumite zgomote în semnalele de la mouse. Această presupunere se bazează pe faptul că semnalele care descriu deplasarea mouse-ului sunt diferite de 0 chiar și la staționare. Pentru a diminua acest comportament, am verificat ca valoarea semnalelor care descriu deplasarea mouse-ului să nu fie mai mică de 3, o valoare destul de mică, de multe ori insesizabilă. Această constrângere, însă, afectează precizia mouse-ului.

După aceea, este descris comportamentul semnalelor pentru fiecare regiune. Pentru rezoluția de 1280x1080, regiunile au următoarele dimensiuni:

- **pe orizontală:**
  - *zona vizibilă(afișaj)*: 1280 pixeli;
  - *front porch*: 48 pixeli;
  - *back porch*: 248 pixeli;
  - *retrace*: 112 pixeli;
- **pe verticală:**
  - *zona vizibilă(afișaj)*: 1024 pixeli;
  - *front porch*: 1 pixeli;
  - *back porch*: 38 pixeli;
  - *retrace*: 3 pixeli;

Mai întâi, se verifică dacă pixelul curent este în regiunea afișajului. Dacă nu, atunci acesta va fi negru. După aceea, se verifică dacă ne aflăm în regiunea de retragere(*retrace*). Dacă răspunsul este afirmativ, atunci semnalele *hsync* și *vsync* vor fi 0. Altfel, se activează aceste semnale.



### 5.2.2. CURSOR(Circuitul de desenare a cursorului)

În figura următoare este prezentată entitatea modulului circuitului de desenare a cursorului, Cursor. Semnalele din entitate sunt aceleași cu cele din diagrama componente din capitolul anterior.

#### Cod 11: Entitate CURSOR

```
entity cursor is
    port (
        clk           : IN STD_LOGIC;
        x_cur, y_cur   : IN INTEGER;
        x_pos, y_pos   : IN INTEGER;
        RGB_in         : IN STD_LOGIC_VECTOR(2 downto 0);
        RGB_out        : OUT STD_LOGIC_VECTOR(2 downto 0);
        draw           : OUT STD_LOGIC
    );
end cursor;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

#### Cod 12: Arhitectura CURSOR

```
architecture Behavioral of cursor is

    type displayrom is array(0 to 255) of std_logic_vector(1 downto 0);
    signal mouserom: displayrom := (
        "00", "00", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "01", "00", "00", "00", "00", "11", "11", "11", "11", "11",
        "00", "01", "01", "01", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "01", "00", "00", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11", "11",
        "00", "00", "11", "11", "00", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11",
        "00", "11", "11", "11", "00", "01", "01", "00", "11", "11", "11", "11", "11", "11", "11",
        "11", "11", "11", "11", "11", "00", "01", "01", "00", "11", "11", "11", "11", "11", "11",
        "11", "11", "11", "11", "11", "00", "01", "01", "00", "11", "11", "11", "11", "11", "11",
        "11", "11", "11", "11", "11", "11", "00", "00", "11", "11", "11", "11", "11", "11", "11"
    );
end;
```

```

constant OFFSET: INTEGER := 16;    -- 16
signal rom_pos: INTEGER range 0 to 255 := 0;    -- 16
signal en: STD_LOGIC;

begin
  en <= '1' when (x_cur >= x_pos and x_cur < (x_pos + OFFSET)
    and y_cur >= y_pos and y_cur < (y_pos + OFFSET))
    else '0';

  draw <= en;

  rom_pos <= (y_cur - y_pos) * 16 + (x_cur - x_pos)
    when (x_cur >= x_pos and x_cur < (x_pos + OFFSET)
    and y_cur >= y_pos and y_cur < (y_pos + OFFSET))
    else 0;

  process(en, rom_pos)
  begin
    if en = '1' then
      case mouserom(rom_pos) is
        when "01" => rgb_out <= "111";
        when "00" => rgb_out <= "000";
        when "11" => rgb_out <= rgb_in;
        when others => rgb_out <= rgb_in;
      end case;
    else
      rgb_out <= rgb_in;
    end if;
  end process;
end Behavioral;

```

Mai întâi, se creează o memorie ROM de 256 de vectori de 2 biți pentru a realiza harta de biți cu forma cursorului. Cursorul va avea dimensiunea de 16x16 pixeli. Există 3 combinații de culori disponibile:

- **00** – negru;
- **01** – alb;
- **11** – transparent(culoarea fundalului rgb\_in);
- **Altă valoare** - transparent(culoarea fundalului rgb\_in).

După care, am creat o constantă *OFFSET* egală cu 16, dimensiunea cursorului în pixeli, care va participa în calculul coordonatelor vârfulor imaginii create. Semnalul *rom\_pos* va reține poziția în harta de biți a pixelului curent. Semnalul *en* este echivalentul semnalului *draw*.

Începem prin a atribui o valoare semnalului *en*, în funcție de poziția pixelului curent. Dacă acesta se află între pixelii care ar încadra marginile poziției cursorului, atunci *en* ia valoarea 1. Altfel, ia valoarea 0. Aceasta valoare se atribuie ulterior semnalului de ieșire *draw*.

Mai apoi, calculăm poziția în harta de biți a pixelului curent *rom\_pos*, însă doar dacă pixelul îndeplinește condiția menționată anterior. Altfel, acesta va lua valoarea 0.

Urmează să stabilim culoarea pixelului, în funcție de poziția lui în harta de biți. Dacă *en* este activ, atunci semnalul de ieșire va lua valoarea corespunzătoare codului de culori stabilit mai sus. În caz contrar, culoarea pixelului va coincide cu culoarea fundalului, transmisă prin semnalul *rgb\_in*.

### 5.2.3. RGB\_VGA(Circuitul modulului VGA)

În figura următoare este prezentată entitatea modulului RGB\_VGA . Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

#### Cod 13: Entitate RGB\_VGA

```
entity rgb_vga is
  port (
    clk, reset      : IN STD_LOGIC;
    hsync           : OUT STD_LOGIC;
    vsync           : OUT STD_LOGIC;
    xm              : IN STD_LOGIC_VECTOR(8 downto 0 ) ;
    ym              : IN STD_LOGIC_VECTOR(8 downto 0 ) ;
    r               : OUT STD_LOGIC_VECTOR(3 downto 0);
    g               : OUT STD_LOGIC_VECTOR(3 downto 0);
    b               : OUT STD_LOGIC_VECTOR(3 downto 0);
    s               : IN STD_LOGIC_VECTOR(2 downto 0)) ;
end rgb_vga;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

#### Cod 14: Arhitectura RGB\_VGA

```
architecture Behavioral of rgb_vga is
  signal x_reg: std_logic_vector (11 downto 0 ) := ( others => '0' );
  signal y_reg: std_logic_vector (11 downto 0 ) := ( others => '0' );
  signal VGACLK:STD_LOGIC;

  component clk_wiz_0
  port
  (-- Clock in ports
  -- Clock out ports
    clk_out1          : out    std_logic;
```

```
-- Status and control signals
reset          : in      std_logic;
clk_in1        : in      std_logic
);
end component;

begin

C: clk_wiz_0 port map (
  -- Clock out ports
  clk_out1 => VGACLK,
  -- Status and control signals
  reset => reset,
  -- Clock in ports
  clk_in1 => clk
);

C1: entity work.vga_sync PORT MAP(
  CLK => VGACLK,
  HSYNC => hsync,
  VSYNC => vsync,
  xm => xm,
  ym => ym,
  R => red,
  G => green,
  B => blue,
  S => sw
);

end Behavioral;
```

Rolul acestui modul este de a stabili conexiunea între modulul VGA\_Sync și Clocking Wizard(clk\_wiz\_0), astfel că el cuprinde doar instanțieri ale celor 2 componente și semnale prin care comunică componentele.

### 5.3. Modul Mouse\_to\_VGA

În figura următoare este prezentată entitatea modulului Mouse\_to\_VGA, componenta principală care cuprinde întreaga aplicație . Semnalele din entitate sunt aceleași cu cele din diagrama componentei din capitolul anterior.

#### Cod 15: Entitate MOUSE\_TO\_VGA

```
entity mouse_to_vga is
    port (
        clk, reset      : in std_logic;
        ps2d, ps2c       : inout std_logic;
        hsync, vsync     : out std_logic;
        red              : out std_logic_vector(3 downto 0);
        green            : out std_logic_vector(3 downto 0);
        blue             : out std_logic_vector(3 downto 0);
    );
end mouse_to_vga;
```

Rolul fiecărui semnal din entitate a fost dezvoltat în capitolul anterior. În continuare, ne vom axa pe felul în care funcționează această componentă și cum sunt prelucrate semnalele pentru a ajunge la rezultatul dorit.

#### Cod 16: Arhitectura RGB\_VGA

```
architecture Behavioral of mouse_to_vga is

    signal xm: std_logic_vector(8 downto 0) ;
    signal ym: std_logic_vector(8 downto 0) ;
    signal m_done_tick: std_logic;
    signal btnm: std_logic_vector(2 downto 0) := (others=>'0');
    signal x_reg, x_next: std_logic_vector(11 downto 0) := (others=>'0');
    signal y_reg, y_next: std_logic_vector(11 downto 0) := (others=>'0');
    signal sw: std_logic_vector(2 downto 0) := (others=>'0');
    signal en: std_logic;
    type RAM is array (0 to 7) of STD_LOGIC_VECTOR(2 downto 0);
    signal color : RAM := (
        "000",
        "001",
        "010",
        "011",
        "100",
        "101",
        "110",
        "111");
begin

    mouse_gen : entity work.mouse(arch) port map(
        clk => clk,
        reset => reset,
        ps2d => ps2d,
```

```
ps2c => ps2c,  
xm => xm,  
ym => ym,  
btnm => btnm,  
m_done_tick => m_done_tick  
);  
  
vga_gen : entity work.rgb_vga port map(  
    clk => clk,  
    reset => reset,  
    sw => sw,  
    hsync => hsync,  
    vsync => vsync,  
    xm => x_reg,  
    ym => y_reg,  
    red => red,  
    green => green,  
    blue => blue  
);  
  
counter_en : entity work.counter_enable port map (  
    Clock => clk,  
    RESET => reset,  
    one_second_enable => en  
);  
  
process (clk, btnm, reset)  
begin  
    if reset = '1' then  
        sw <= (others => '0');  
    elsif clk = '1' and clk'event and en = '1' then  
        if btnm(0) = '1' then  
            if sw >= 7 then  
                sw <= "000";  
            else  
                sw <= sw + 1;  
            end if;  
        end if;  
    end if;  
end process;  
  
x_reg <= x_reg when m_done_tick= '0' else  
    xm(8) & "000" & xm(7 downto 0);  
y_reg <= y_reg when m_done_tick= '0' else  
    ym(8) & "000" & ym(7 downto 0);  
  
end Behavioral;
```

Pentru început, sunt declarate semnalele interne ale modulului, ale căror roluri sunt identice cu ale celor din componentele anterioare având nume identic. În plus, se creează și o memorie ROM cu 8 vectori, reprezentând cele 8 culori care pot apărea pe ecran.

După aceea, sunt instanțiate componentele pentru mouse și VGA și se stabilesc legăturile între cele două. Mai apare, de asemenea, o instanțiere a componentei counter\_enable, al cărei scop a fost precizat în capitolul anterior.

Componenta Counter este reprezentată de următorul proces. Acesta incrementează valoarea semnalul *sw* în funcție de semnalul de *clock*, semnalul *en* și semnalul *btnm(0)*, care conține informații legate de apăsarea butonului stâng al mouse-ului. Acest semnal stabilește culoarea care va fi afișată pe ecran, ținând cont de valorile hardcodate în memoria ROM.

În final, se atribuie valori semnalelor *x\_reg* și *y\_reg* în funcție de valoarea semnalului *m\_done\_tick*.

## 5.4. Componente adiționale

### 5.4.1. Counter enable

#### Cod 17: Entitate COUNTER\_ENABLE

```
entity mouse_to_vga is
    port (
        clk, reset    : in std_logic;
        enable        : out std_logic;
    );
end mouse_to_vga;
```

#### Cod 18: Arhitectura COUNTER\_ENABLE

```
architecture Behavioral of counter_enable is
    signal counter: STD_LOGIC_VECTOR (27 downto 0);

begin

    process( reset,Clock)
    begin
        if(reset='1') then
            counter <= (others => '0');
        elsif(rising_edge(Clock)) then
            if(counter>x"1F5E000") then
                counter <= (others => '0');
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
    enable <= '1' when counter=x"1F5E000" else '0';

end Behavioral;
```

## Capitolul 6 – TESTARE & VALIDARE

Pentru a ușura procesul de debugging și de corecție a erorilor, dar și pentru a asigura o funcționare cât mai corectă a codului, am testat modulele componentelor mari individual. Testarea acestor s-a făcut atât prin testbench, cât și prin testare funcționării codului printr-un cod simplu, în funcție de caz.

În această secțiune voi prezenta codul pentru testare, alături de rezultatele testării. Aceste rezultate vor fi susținute de capturi ale formelor de undă în testbench sau de fotografii ilustrând funcționarea corectă.

### 6.1. Testarea modului pentru mouse

Datorită faptului că pentru realizarea comunicării cu mouse-ul am avut de implementat protocolul PS/2, este foarte dificil să simulăm printr-un banc de test întregul proces de comunicare între cele două dispozitive, chiar dacă vorbim și despre transmiterea unui singur pachet întreg de date de la mouse. Pentru a testa, totuși, corectitudinea implementării protocolului și a conexiunilor făcute, am realizat un cod destul de simplu, prin care se poate vizualiza mișcarea mouse-ului pe fiecare axă și funcționarea butoanelor mouse-ului prin intermediul led-urilor plăcii FPGA.

#### Cod 19: MOUSE\_LED

```
entity mouse_led is
    port ( clk, reset: in std_logic;
          sa: in std_logic; -- select axes
          ps2d, ps2c : inout std_logic;
          led: out std_logic_vector ( 7 downto 0)
    );
end mouse_led;

architecture arch of mouse_led is
    signal p_reg , p_next : unsigned (9 downto 0 ) ;
    signal xm: std_logic_vector (8 downto 0 ) ;
    signal ym: std_logic_vector (8 downto 0 ) ;
    signal cm: std_logic_vector (8 downto 0 ) ;
    signal m_done_tick: std_logic;
    signal btnm: std_logic_vector (2 downto 0) ;
begin
    -- instantiation
    mouse_unit: entity work.mouse(arch)
        port map(clk=>clk, reset=>reset ,
        ps2d=>ps2d, ps2c=>ps2c,
        xm=>xm, ym=>ym , btnm=>btnm ,
        m_done_tick=>m_done_tick);

    -- select axes
    cm <= xm when sa='0' else ym;
```



```
-- r e g i s t e r
process (clk , reset)
begin
    if reset= '1' then
        p_reg <= ( others => '0' ) ;
    elsif (clk'event and clk='1') then
        p_reg <= p_next;
    end if ;
end process ;

-- c o u n t e r
p_next <= p_reg when m_done_tick= '0' else
    "0000000000" when btnm(0)='1' else -- l e f t b u t t o n
    "1111111111" when btnm(1)='1' else -- r i g h t b u t t o n
    p_reg + unsigned(cm(8) & cm);

with p_reg(9 downto 7) select
    led <= "10000000" when "000",
           "01000000" when "001",
           "00100000" when "010",
           "00010000" when "011",
           "00001000" when "100",
           "00000100" when "101",
           "00000010" when "110",
           "00000001" when others;

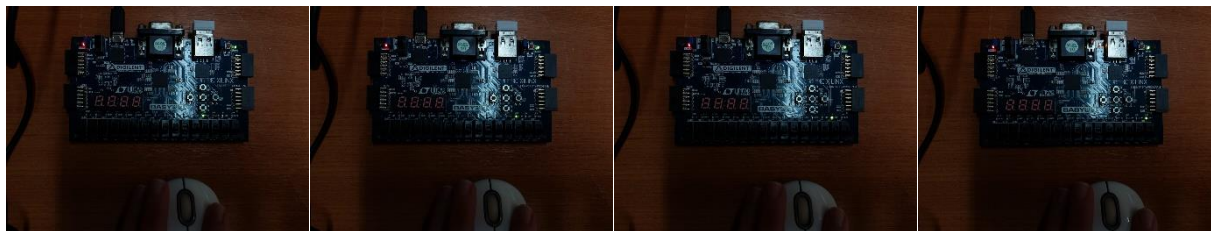
end arch;
```

Entitatea acestui modul de testare este prevăzută cu semnale pentru *clock* și *reset*, pentru intrările de date și de clock de la mouse, cu o intrare *sa* controlată de un switch, care selectează axa care să fie testată, și 8 led-uri.

În arhitectura modulului se instanțiază componenta Mouse, se selectează axa de coordonate pentru care se va face testarea (0 pentru axa x, 1 pentru axa y) și se face interpretarea datelor de la mouse pentru cele 8 led-uri. Modul de funcționare corect al codului ar trebui să fie următorul:

- Luminile de pe placa FPGA se vor aprinde pe rând, în funcție de direcția în care se mișcă mouse-ul, respectiv de butonul apăsat;
- La deplasarea mouse-ului spre stânga (sau în sus pentru axa y), luminile se vor aprinde pe rând dinspre dreapta înspre stânga. Când se face o depășire, se mută lumina din nou la cel mai din dreapta led;
- La deplasarea mouse-ului spre dreapta (sau în jos pentru axa y), luminile se vor aprinde pe rând dinspre stânga înspre dreapta. Când se face o depășire, se mută lumina din nou la cel mai din stânga led;
- La apăsare butonului din stânga al mouse-ului, se va aprinde cel mai din stânga led;
- La apăsare butonului din dreapta al mouse-ului, se va aprinde cel mai din dreapta led;

În urma testării codului de mai sus, se poate constata că modulul protocolului PS/2 funcționează conform așteptărilor. Mai jos sunt prezentate câteva fotografii care ilustrează modul de funcționare al codului de testare.

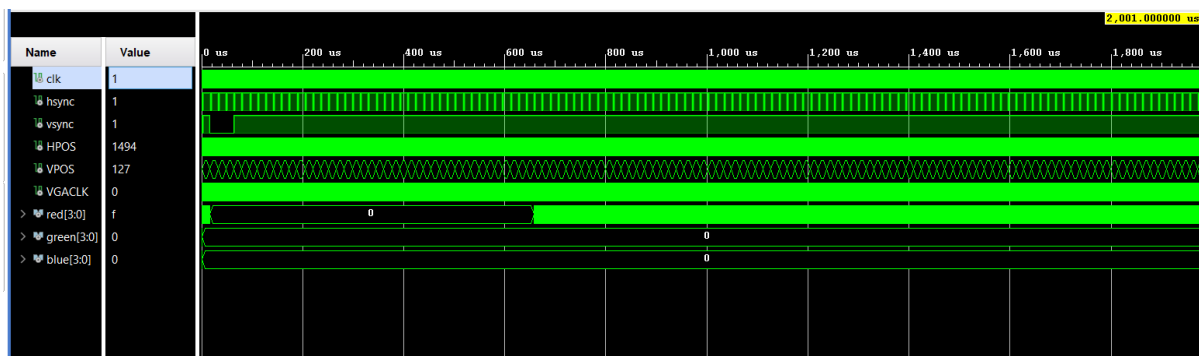


**Fig. 27:** Testarea comunicării cu mouse-ul

## 6.2. Testarea modulului VGA

Pentru testarea modulului VGA am realizat un banc de testare pentru a putea observa dacă se asignează corect semnalele *hsync* și *vsync* și dacă se numără corect pixelii pe orizontală și pe verticală.

În continuare voi pune câteva imagini cu formele de undă ale semnalelor semnificative pentru noi în cadrul testării prin banc de test.



**Fig. 28:** TestBench – vizualizarea semnalelor în ansamblu



Dezactivarea  
semnalului vsync  
în zona de retrace

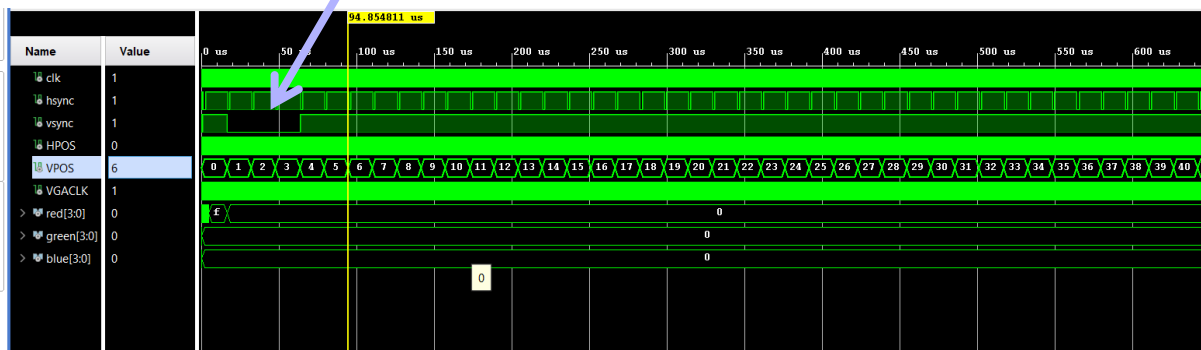


Fig. 29: TestBench – incrementarea semnalului *vpos*

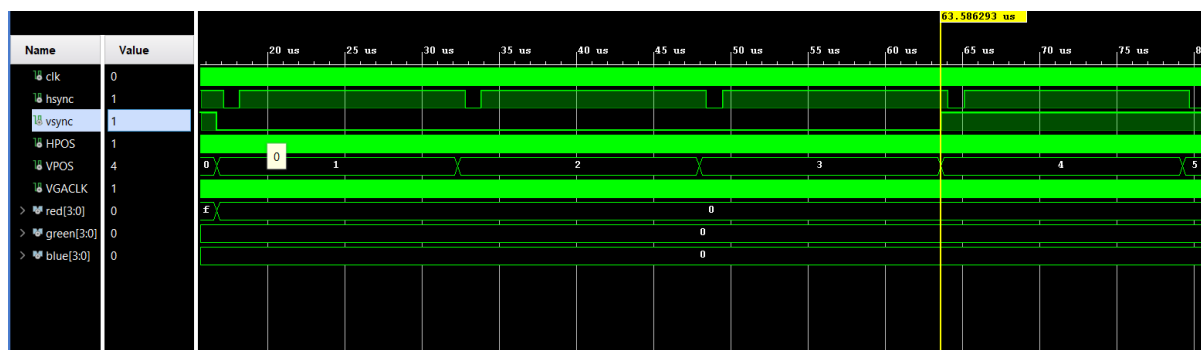


Fig. 30: TestBench – vizualizarea regiunii de retrace a semnalului

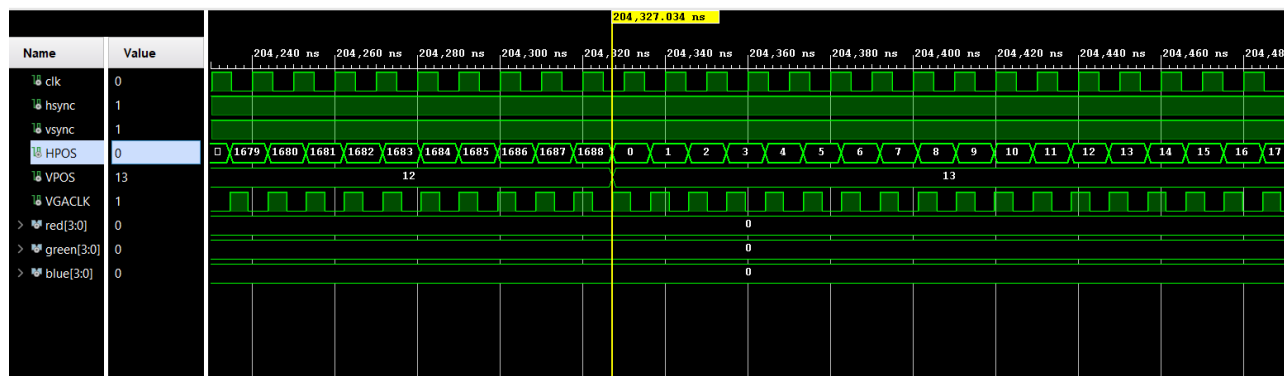


Fig. 31: TestBench – incrementarea și resetarea semnalului *hsync*

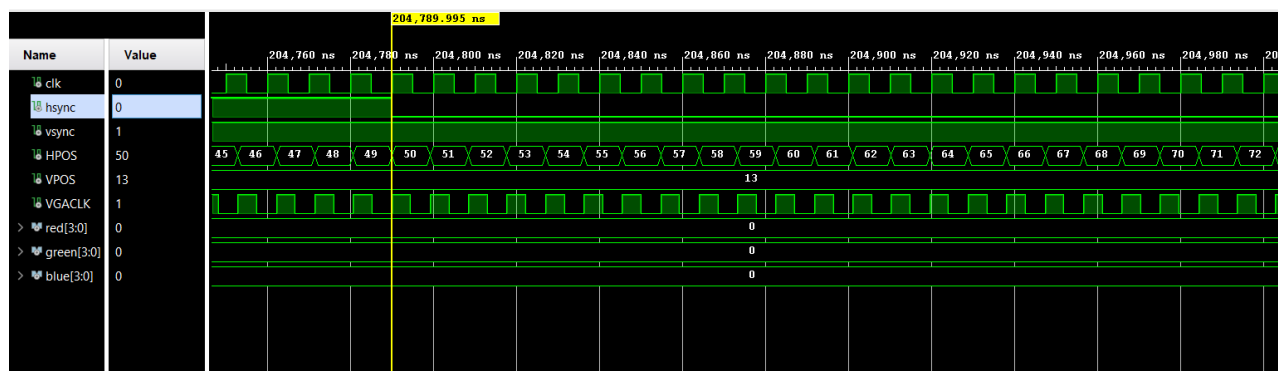


Fig. 32: TestBench – vizualizarea regiunii de retrace a semnalului *hsync*

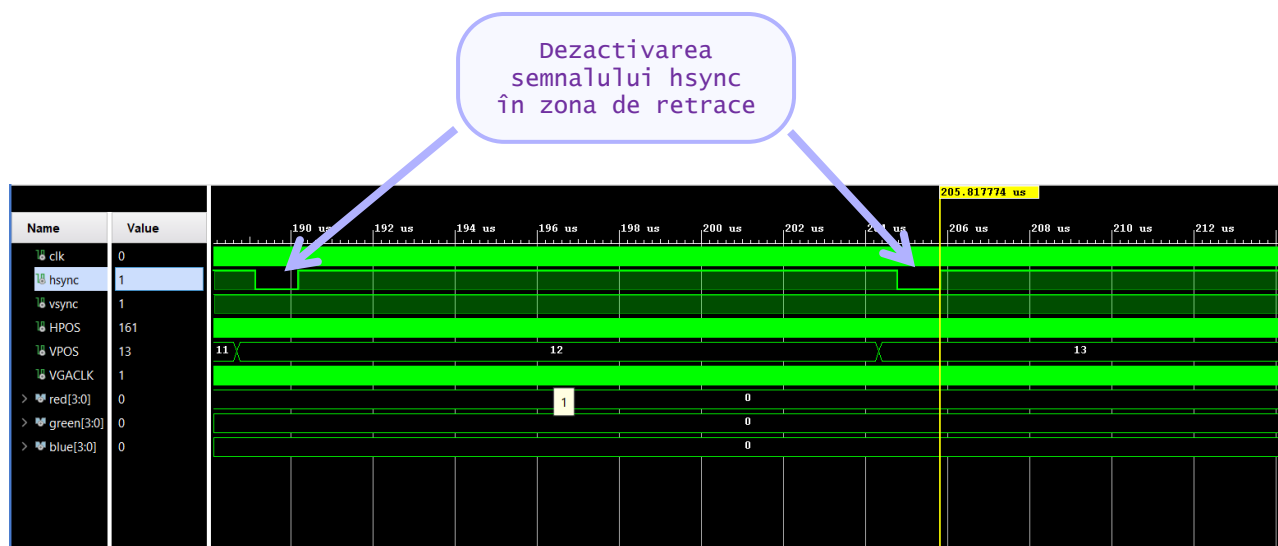


Fig. 33: TestBench – vizualizarea regiunilor de retrace ale semnalului *hsync*

În urma analizării bancului de test, se poate observa că modulul VGA are comportamentul așteptat.



## Capitolul 7 – CONCLUZII

Proiectarea unei aplicații de comunicare între o placă FPGA și două dispozitive externe nu este o muncă ușoară. Aceasta presupune foarte multă documentație și experimentare. Cu toate acestea, un astfel de proiect te poate ajuta să acumulezi cunoștințe valoroase de programare și să înțelegi mai bine principiul de funcționare al dispozitivelor cu care interacționează placa și îți oferă posibilitatea de a învăța să le folosești pentru dezvoltarea unor proiecte spectaculoase și cu aplicabilitate în viața cotidiană.

Aplicația pe care am făcut-o exemplifică destul de bine comunicarea între mouse și placa FPGA, respectiv între placa FPGA și monitor. Cu toate acestea, precizia de detectare a schimbărilor de stare a mouse-ului ar avea nevoie de îmbunătățiri, acesta fiind unul din dezvoltările pe care mi-aș dori să le aduc proiectului în viitor. De asemenea, se pot observa mici întârzieri în ceea ce privește actualizarea imaginii de pe ecran, așa că îmi propun ca în viitor să aduc modificări la timpul de răspuns și actualizare a imaginii pe ecran.

Proiectul acesta poate reprezenta o bază consistentă pentru alte proiecte mai interesante, ca de exemplu pentru un program de desenare sau un joc precum Ping-Pong și DX Ball.



## BIBLIOGRAFIE

- <https://www.fpga4student.com/2017/12/how-to-interface-mouse-with-FPGA.html>
- [https://github.com/AngeloJacobo/FPGA\\_Book\\_Experiments/tree/main/13.7.10%20Mouse\\_Scribble%20%5BVIDEO%5D](https://github.com/AngeloJacobo/FPGA_Book_Experiments/tree/main/13.7.10%20Mouse_Scribble%20%5BVIDEO%5D)
- [https://github.com/AngeloJacobo/FPGA\\_Book\\_Experiments/tree/main/13.7.9%20Mouse\\_Pointer\\_Circuit%20%5BVIDEO%5D](https://github.com/AngeloJacobo/FPGA_Book_Experiments/tree/main/13.7.9%20Mouse_Pointer_Circuit%20%5BVIDEO%5D)
- <http://quitoart.blogspot.com/2017/07/vhdl-ps2-mouse-fpga-board-interface.html>
- <http://quitoart.blogspot.com/2017/08/vhdl-pong-game-using-vga-module-xilinx.html>
- <https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html>
- <https://www.geeksforgeeks.org/i2c-communication-protocol/>
- <https://www.totalphase.com/blog/2019/06/what-you-must-know-about-these-5-serial-communication-protocols/>
- <https://digitalsystemdesign.in/interfacing-vga-display-with-fpga/>
- <https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-vga-test-pattern-with-mouse-overlay/start>
- <https://www.youtube.com/watch?v=lyGwvGzrqp8>
- <https://www.youtube.com/watch?v=4xBjrE9csxg>
- <https://learn.microsoft.com/en-us/windows-hardware/drivers/hid/>
- <https://users.utcluj.ro/~negrum/wp-content/uploads/2019/12/Curs06.pdf>
- <https://users.utcluj.ro/~negrum/wp-content/uploads/2019/12/Curs07.pdf>
- "Arhitectura Calculatoarelor. Îndrumător de laborator", Florin Oniga, Mihai Negru, Editura UTPRESS Cluj-Napoca, 2019
- <https://www.youtube.com/watch?v=4enWoVHCykl>
- [https://www.burtonsys.com/ps2\\_chapweske.htm](https://www.burtonsys.com/ps2_chapweske.htm)
- [https://users.utcluj.ro/~baruch/media/sie/labor/PS2/PS-2\\_Mouse\\_Interface.htm#Footnotes](https://users.utcluj.ro/~baruch/media/sie/labor/PS2/PS-2_Mouse_Interface.htm#Footnotes)
- [http://ebook.pldworld.com/\\_eBook/FPGA/HDL-Examples-/interfacing%20mouse%20with%20VHDL.pdf](http://ebook.pldworld.com/_eBook/FPGA/HDL-Examples-/interfacing%20mouse%20with%20VHDL.pdf)
- [http://ebook.pldworld.com/\\_eBook/FPGA/HDL-Examples-/VGA%20controller%20-%20graphical%20based.pdf](http://ebook.pldworld.com/_eBook/FPGA/HDL-Examples-/VGA%20controller%20-%20graphical%20based.pdf)
- <https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual>
- <https://www.youtube.com/watch?v=WK5FT5RD1sU>