UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Structura sistemelor de calcul

Comunicația dintre placa Nexys 4 DDR și un mouse USB HID

10.10.2021

Îndrumator de proiect:
Cristi Mocan

Student:
Găvră Anamaria

Grupa: 30236

Cuprins

Introducere

Acest proiect are ca scop dezvoltarea unei aplcatii simple care sa aiba la baza o conexiune intre o placa Nexys 4 DDR, un mouse USB si un monitor. Atat miscarea mouse-ului, cat si starea butoanelor sunt informatiile transmise placutei de dezvoltare, care interactioneaza cu monitorul cu ajutorul caruia sunt afisate aceste informatii.

Principalul obiectiv al proiectului este acela de a realiza proiectarea aplicatiei cu ajutorul mediului de dezvoltare Vivado Design Suite si utilizand limbajul de descriere hardware VHDL. Aceasta etapa este urmata de implementarea codului calculator pentru afisarea cursorului si a starii butoanelor mouseu-lui.

VHDL (VHSIC Hardware Description Language) este unul dintre cele mai utilizate limbaje de descriere hardware, fiind destinat proiectarii sistemelor electronice digitale prin descrierea comportamentala sau structurala a unui proiect.

Baza teoretica utilizata in elaborarea lucrarii este prezentata la sectiunea ,,Fundamentare teoretica” alaturi de descrierea resurselor utilizate. In capitolul ,,Proiectare si implementare” sunt cuprinse etapele percurse pentru indeplinirea obiectivelor enumerate anterior, motivarea solutiei alese, prezentarea fiecarei entitati in parte, dar si a relatiei dintre acestea. Acesta mai contine si pasii care trebuie urmati pentru a rula proiectul, fiind cea mai importanta parte a raportului.

Rezultatele obtinute in urma verificarii modului de functionare a proiectului si interpretarea acestora, prezentarea mediului de dezvoltare, a limbajului de programare utilizat, procedura de testare, dificultatile intampinate si modul de rezolvare a acestora sunt cuprinse in cea de a treia sectiune denumita sugestiv ,,Rezultate experimentale”. Alte observati cu privire la rezultatele obtinute se gasesc in capitolul ,,Concluzii”, alaturi de cateva aplicatii si sugestii pentru dezvoltari ulterioare ale acestui proiect. La sectiunea ,,Anexe” se gasesc portiuni din codul implementat pe placuta FPGA, cateva scheme si grafice.

Fundamentare teoretica

Acest proiect constituie un exemplu simplu de utilizare a conexiunii dintre placuta de dezvoltare Nexys 4 DDR si doua periferice: unul de iesire care are rolul de a face vizibile rezultatele pe ecran, utilizans ieşirea VGA a placutei, şi unul de intrare care capteaza informatia din exterior si o transmite placutei fiind conectat la intrarea USB.

Mouse-ul USB HID

1 Generalitati

Mouse-ul este unul dintre cele mai utilizate dispozitive periferice pentru introducerea comenzilor. Senzorul care este situat in partea inferioara a mouse-ului spre suprafata pe care este asezat, are rolul de a detectata miscarea acestuia, de a o preluata şi de a o transmite ulterior dispozitivului la care este conectat (in acest caz, placa FPGA), care proceseaza informatiile primite de la acest dispozitiv.

2 Principiu de functionare

Informatia de miscare a mouse-ului este convertita in miscarea unui cursor pe ecranul unui dispozitiv de afisare conectat la portul VGA al celui care realizeaza conversia. De fiecare data cand mouse-ul este mutat, sunt trimise trei cuvinte de 11 biti de la mouse la dispozitivul gazda, fiecare set de date transmis continand 33 de biti. Fiecare cuvant are urmatoarele elemente:

- un bit de pornire setat pe „0” logic;
- 8 biti de date, primul fiind cel mai putin semnificativ;
- un bit de paritate impar
- un bit de oprire setat pe „1” logic.

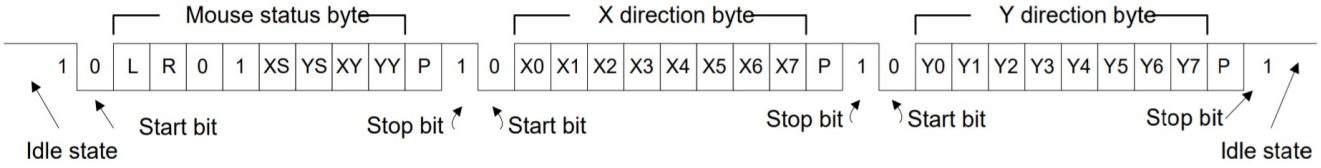

Figure 1

Dupa cum se observa in figura 1, biiti de pornire sunt bitii 0, 11 și 22 , iar cei de oprire sunt bitii 10, 21 și 32 . Cele trei campuri de date de 8 biți conțin date de mișcare. Validarea datelor transmise se realizeaza pe frontul descrescator al semnalului de ceascare are o frecventa cuprinsa intre 20 Khz si 30 Khz.

3 Determinarea coordonatelor

Mouse-ul se deplaseaza intr-un sistem de coordonate, miscarile acestuia determinand valorile din grupul celor 8 biti de date; astfel deplasarea mouse-ului spre dreapta genereaza un numar pozitiv in campul X; iar o miscare spre stanga determina aparitia unei valori negative pe aceesi pozitie. De asemenea, miscarea mouse-ului in fata este sugerata de aparitia unui numar pozitiv in campul Y, in timp ce deplasarea in spate este reprezentata printr-un numar negativ pe pozitia respectiva.

Bitii XS si YS din octetul de stare (primul set de informatii trimis catre placa vhdl) sunt bitii de semn pentru coordonatele mouse-ului pe cele doua axe. Prezenta valorii „1” logic pe una dintre pozitii indica un numar negativ si, mai exact, o deplasare in stanga, respectiv in spate. Daca valoarea unuia dintre aceste campuri este zero, este sugerata o valoare pozitiva; adica o deplasare spre dreapta sau in fara a mouse-ului.

4 Starea butoanelor

Primii doi biti din octetul de stare (L si R) specifica starea butonului din stanga, respectiv a butonului drept al mouse-ului. Daca unul dintre butoane a fost apasat, bitii corespunzatori primesc valoarea unu logic, in caz contrar, valoarea acestor camputi ramane zero. Dupa apasare, valoarea bitilor redevine zero.

5 Viteza de miscare

Valoarea numerelor $X_0 \rightarrow X_7$ si $Y_0 \rightarrow Y_7$, reprezentate in cel de-al doilea(X) si cel de-al treilea cuvant transmis (Y), sunt folosite pentru a determina viteza de miscare a mouse-ului. Cu cat valoarea acestor campuri este mai mare, cu atat mouse-ul se deplaseaza mai repede. Pentru a indica o depasire a valorii care se poate reprezenta pe numarul de biti alocat acestor campuri, se utilizeaza bitii Xv si Yv din octetul de stare. Cand are loc o depasire a unei valori a vitezei de deplasare, valoarea bitului ce corespunde depasirii devine unu.

6 Comunicatia dintre placa FPGA si mouse-ul USB

Comunicatia intre fpga si periferice este bidirectionala, ceea ce inseamna ca gazda poate atat sa primeasca date de la mouse, cat si sa trimita comenzi. Spre deosebire de tastatura, mouse-ul are diferite moduri de operare. Dupa ce este pornit, se afla in modul de configurare si nu transmite date pana nu primeste o comanda specifica, dupa care intra in modul stream si incepe sa transmita date de miscare si starea butoanelor. La pornire, efectueaza un test de pornire intern. Mouse-ul trimite un octet de date cu valoarea hexazecimala “AA”, care indică faptul ca testul a fost efectuat cu success, și apoi trimite id-ul mouse-ului, care are valoarea “00”,(id-ul unui mouse standard PS2). Gazda poate sa transmita date pentru resetarea, verificarea sau configurarea prorietatilor mose-ului ca viteza de transmitere a datelor. Dupa trimiterea unei comenzi, mouse-ul raspunde cu “FE” in cazul in care comanda a fost receptionata fara erori.

Valoarea fiecarei deplasari este inregistrata în contorul intern al mouse-ului . Cand datele sunt transmise, contorul este resetat si reincepe sa numere. Numaratorului are o capacitate de 9 biti care formeaza un numar pozitiv pentru cazul in care deplasarea s-a realizat in sus sau in dreapta, iar negativ pentru deplasarea in jos sau la stanga. Relația dintre distanta si valoarea numarului este data de rezoluția mouseului. In mod impicit, are valoarea de patru numarari/mm. Daca contorul depaseste valoarea maxima, este setat bitul de overflow.

UART

UART (Universal Asynchronous Receiver - Transmitter) este un dispozitiv hardware pentru comunicatii seriale asincrone in care formatul datelor si vitezele de transmisie sunt configurabile. Trimite biti de date unul cate unul, de la cel mai putin semnificativ la cel mai semnificativ, incadrati de biti de pornire si de oprire, astfel incat canalul de comunicatie gestioneaza sincronizarea precisa.

In cazul comunicatiei seriale asincrone, fiecare caracter transmis este precedat de un bit de START, pentru care linia de comunicatie are nivelul logic 0, si este urmat de cel putin un bit de STOP, pentru care linia de comunicaţie are nivelul logic 1. Deci, bitii de START si de STOP incadreaza fiecare caracter transmis. Intervalul de timp intre transmisia a douz caractere succesive este variabil, pe durata acestui interval linia de comunicaţie avand nivelul logic 1. Atunci cand receptorul detecteaza bitul de START care indică începutul unui caracter, porneşte un oscilator de ceas local, care permite masurarea intervalului de timp corespunzător unui bit, interval care depinde de debitul binar. Acest oscilator permite esantionarea corectă a biţilor individuali ai caracterului., aceasta realizandu-se aproximativ la mijlocul intervalului corespunzator fiecarui bit.

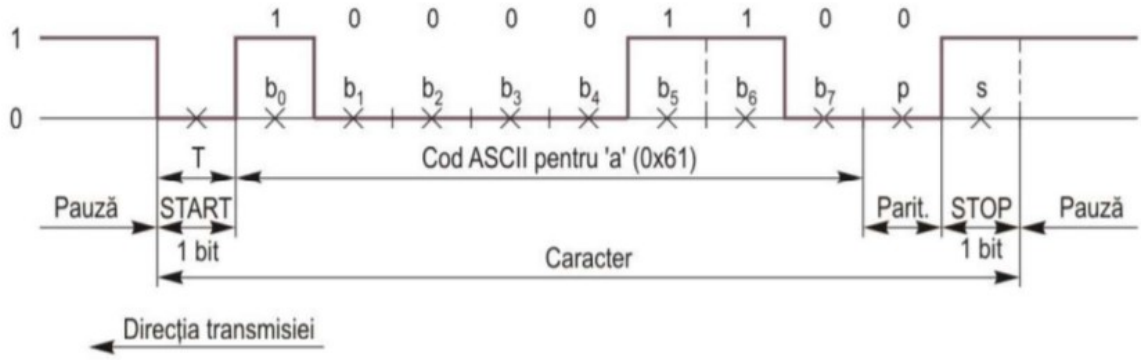

Figure 2

Transmitatorul serial va permite specificarea vitezei de comunicatie printr-un generic. Dimensiunea datelor transmise este de 9 biti, fara bit de paritate, urmate de un singur bit de stop. Acesta este implementat cu ajutorul unui registru de deplasare cu incarcare paralela si iesire seriala. Cealalta componenta a transmitatorului este unitate de control care genereaza semnalul de iesire txRdy si semnalele de comanda. Aceasta va include si numărătoare pentru contorizarea biţilor transmişi şi pentru măsurarea duratei unui bit, în funcţie de debitul binar specificat. Intrările mo-dulului sunt semnalul de ceas, semnalul de resetare sincronă, vectorul de date care trebuie transmise, şi semnalul de control Start. Ieşirile mo-dulului sunt linia Tx pe care sunt transmise datele în mod serial si semnalul TxRdy, care indica finalizarea transmisiei de date.

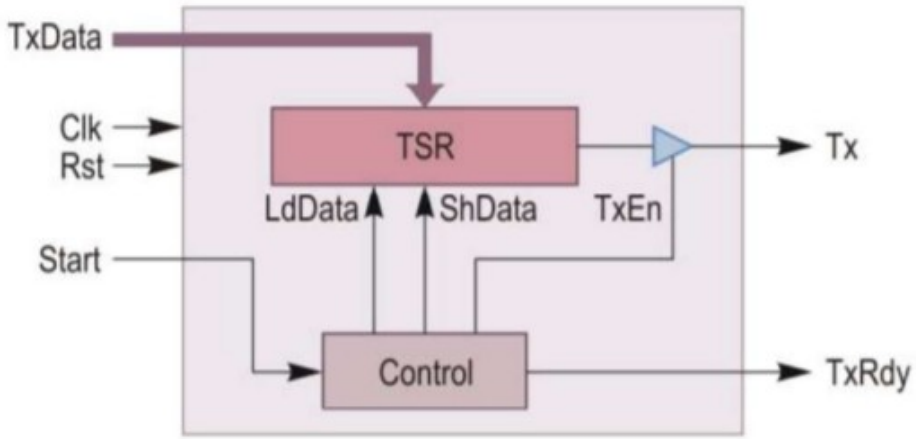

Figure 3

Placa Nexys 4 DDR

Informatia pe care o prelueaza mouse-ul este transmisa placutei FPGA, aceasta fiind componenta care are cel mai important rol din acest proiect. Placa FPGA este un circuit integrat conceput pentru a fi configurat de un proiectant dupa fabricare, utilizand un limbaj de descriere hardware. Computerul realizeaza sinteza codului pe care il implmenteaza pe placa. Pentru aceste operatii se utilizeaza un mediu de dezvoltare a aplicatiilor.

Componente

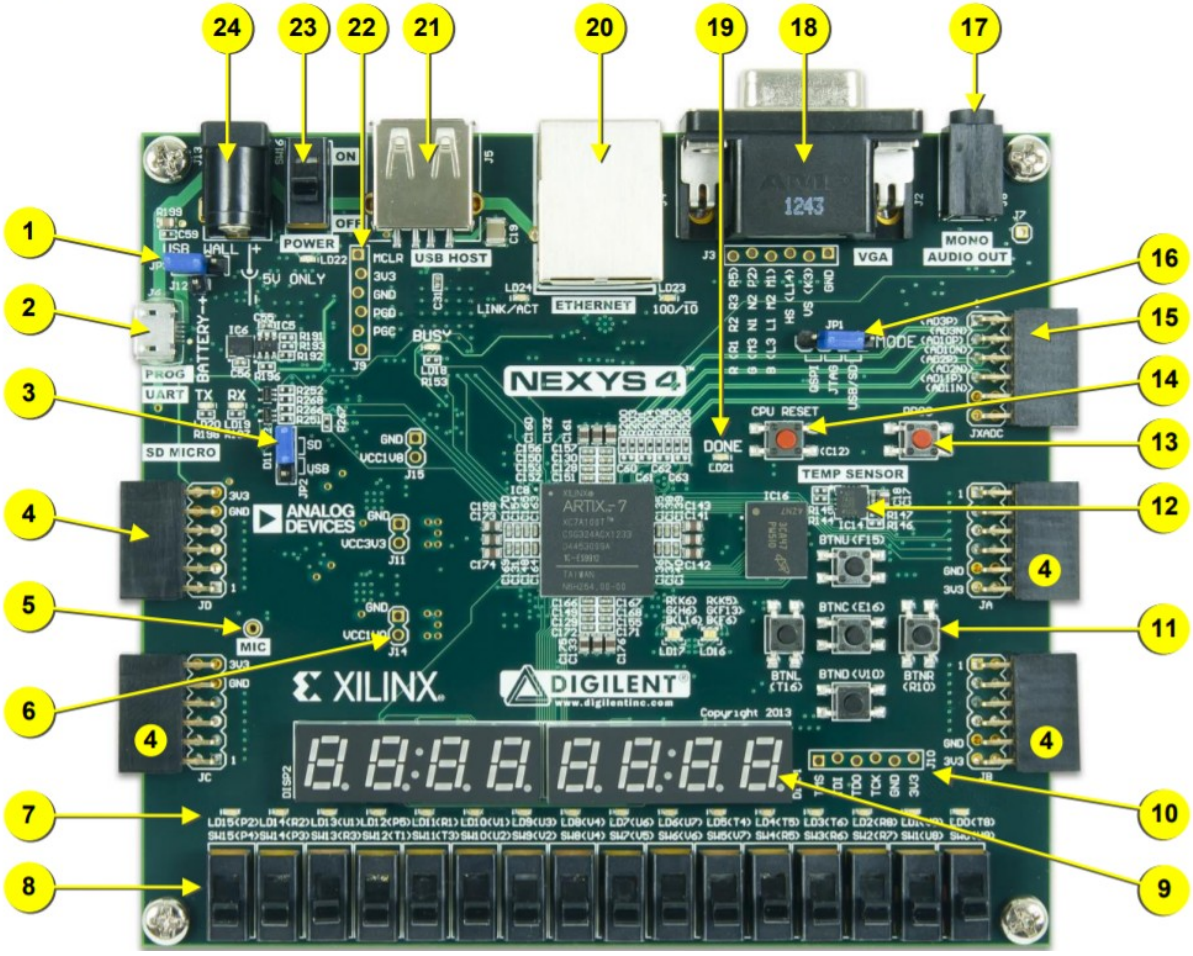

Figure 5

Placuta de dezvoltare Nexys4 contine urmatoarele elemente principale care sunt vizibile si in figura 2:

Un jumper este alcatuit din perechi de pini conductivi utilizati pentru a inchide un circuit electronic pentru a ajuta la selectarea unei anumite caracteristici a circuitului folosind un capac care sa acopere pinii pentru unirea acestora.

Placa Nexys 4 poate primi energie de la **portul USB-JTAG (2)** sau de la o sursa de alimentare externa care se conecteaza la placuta prin **mufa de alimentare(24)**. **Jumper-ul de selectare a sursei de curent -JP3-(1)** de langa mufa de alimentare determina care este sursa utilizata. Toate sursele de alimentare Nexys 4 pot fi pornite si oprite printr-un singur **comutator de alimentare(23)**. Cand placuta este alimentata, ledul *„Power”* este aprins.

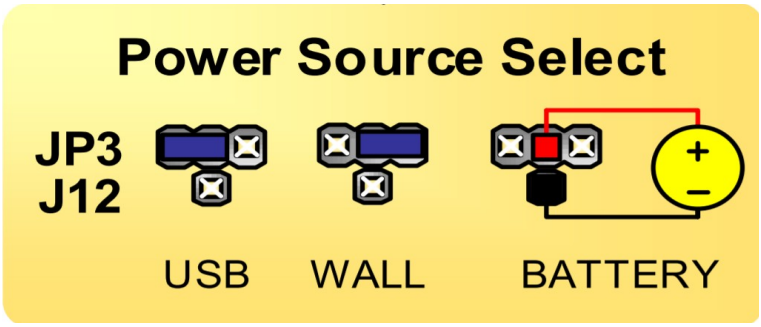

Figure 6

Dupa pornire, placa trebuie configurata inainte de a putea indeplini orice functie. Se poate configura in unul dintre urmatoarele moduri:

1. Se poate configura cu ajutorul unui computer prin portul **USB UART/JTAG(2)**.
2. Un fisier stocat in dispozitivul flash serial nevolatil poate fi transferat la FPGA utilizand portul SPI.
3. Un fisier de programare poate fi transferat pe FPGA de pe un card micro SD.
4. Un fisier de programare poate fi transferat de pe un stick de memorie USB atasat la **portul USB HID (21)**.

Pentru a decide care dintre aceste moduri de configurare este folosit, se utilizeaza **jumper-ul pentru modul de programare-JP1-(16)** si cel **de configurare externa-JP2- (3),** ca in figura alaturata.

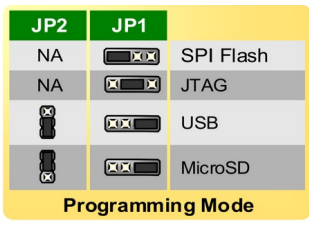

Figure 7

USB-UART Bridge

Nexys4 DDR include o punte USB-UART (atasata la conectorul J6) care va permite sa utilizati aplicatii pentru computer pentru comunicatia cu placa. Este, de asemenea, folosit ca si controler pentru circuitele Digilent USB-JTAG, dar functiile USB-UART si USB-JTAG se comporta complet independent unele de altele. Combinatia acestor doua caracteristici intr-un singur dispozitiv permite ca DDR-ul Nexys4 sa fie programat si alimentat de la un computer atasat cu un singur cablu Micro USB.

USB HID Host

Microcontrolerul funcției auxiliare furnizeaza capacitatea de compatibilitate HID cu USB Nexys4 DDR. Firmware-ul din microcontroler poate fi montat pe un mouse si o tastatura atasata la conectorul A USB. Numai tastaturile și mouse-urile care suportă interfața HID de boot sunt acceptate. PIC24 conduce mai multe semnale in FPGA - doua sunt folosite pentru a implementa o interfață standard PS/2 pentru comunicarea cu un mouse sau tastatura, iar altele sunt conectate la portul de programare serial cu două fire FPGA, astfel incat FPGA poate fi programata si stocata pe o unitate USB sau pe o cartela microSD.

Monitorul

Informatiile miscarilor mouse-ului si starea butoanelor procesate de placuta Nexys 4, determina miscarea cursorului vizibil pe ecran. Monitorul este un periferic de iesire pentru afisare grafica a datelor sub forma caracterelor si a simbolurilor. Placa Nexys 4 utilizeaza 14 semnale FPGA pentru a crea un port VGA cu 4 biti pe culoare si cele doua semnale de sincronizare standard (HS - Sincronizare orizontala si VS - Sincronizare verticala).

Solutia propusa

Dupa ce este pornit, mouse-ul efectueaza un test intern si in cazul in care a fost efectuat cu succes, este returnat id-ul mouse-ului. Dupa primirea comenzii de pornire, mouse-ul incepe sa transmita date de pozitie care sunt “curatate” cu ajutorul unui debouncer dupa care sunt receprionate de componenta PS2Com si transmise mai departe la Mouse Control, care interpreteaza datele primite si trimite comenzi in functie de starea in care se afla. Iesirile acestuia sunt conectate la un afisor SSD, dar si la iesirea tx, prin care sunt transmise semnalele la computer, urmand sa fie afisate folosind software-ul TeraTerm.

Cazuri de utilizare

Deplasarea mouse-ului

Pentru incarcarea proiectului, se ruleaza codul cu ajutorul aplicatiei de dezvoltare Vivado Design Suite facand dublu-click pe fisierul cu extensia *„xpr”*. Pentru a obtine bitstream-ul se apasa pe butonul *„Generate Bitstream”* din sectiunea *„Program and debug*” a meniului *„Flow Navigator*”. Daca nu au aparut erori, se genereaza un fisier cu extensia „.bit”. Se selecteaza optiunea *„Open Hardware Manager”* din sectiunea *„Program and Debug”* a aceluiasi meniu. Se apasa pe iconita Auto Connect din panoul *„Hardware Manager*”, dupa care se apasa pe butonul Program Device din partea de sus a ferestrei. Se selecteaza fisierul cu extensia .bit generat anterior dupa care se apasa butonul *„Program”*.

In urma incarcarii proiectului pe placa, a conectarii mouse-ului la portul USB si a al placutei Nexys4, utilizatorul poate misca mouseul pe o suprafata plana, cu senzorul inspre acea suprafata. Pe monitor, dar si pe primele sase cifre ale afisajului de pe placuta, se observa coordonatele mouse-ului. Pe cifra numarul sapte este afisata pozitia rotitei. Dupa terminarea utilizarii se deconecteaza mouse-ul si monitorul de la placa de dezvoltare FPGA, iar alimentarea acesteia se opreste de la shitch-ul de pornire/oprire.

Apasarea butoanelor

Pentru a testa aceasta functionalitate trebuie urmati pasii prezentati la cazul anterior pentru conectarea perifericelor si pentru incarcarea proiectului pe placa Nexys4 . Prin apasarea butonului din stanga, pe ultima cifra a afisajului apare cifra unu., daca este apasat butonul din drapta, cifra doi, iar pentru butonul din mijloc, cifra patru. La apasarea simultana a mai multor butone, apare supa valorilor corespunzatoare fiecarui buton.

Proiectare si implementare

Arhitectura

Schema bloc

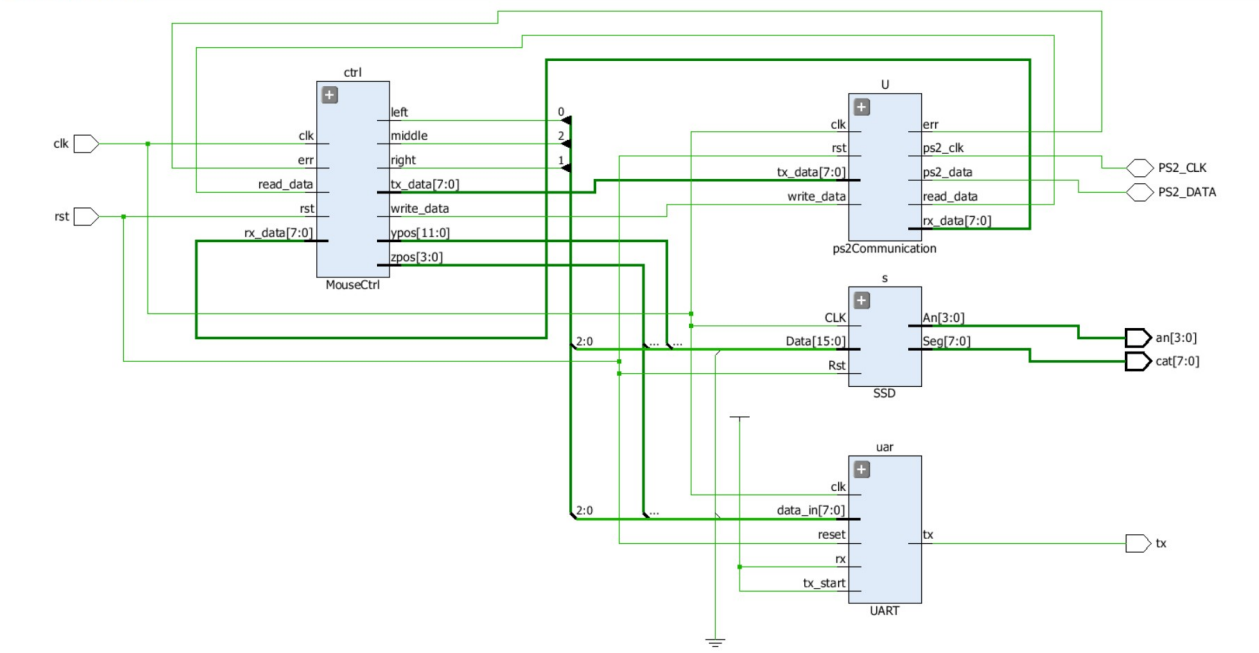

Figure 8

Descriere generala

Dupa cum se poate observa si in figura de mai sus, proiectul are ca intrari semnalul de clock primit de la placuta Nexys4 (clk) si un semnal de reset sincron *rst*, iar ca iesiri, doi vectori de cate opt biti (An si Cat) si un semnal tx. Iesirea An este folosita pentru a determina care anod al afisajului este activ in fiecare perioada de ceas prin dezactivarea bitului corespunzator acestuia, restul bitilor avand valoarea logica „1”. Vectorul Cat este folostita pentru a activa catozii anodului activ astfel incat sa fie afisata cifra corespunzatoare. Semnalul tx are rolul de a transmite datele de la placuta la calculator pentru a le afisa pe ecran. Pe langa aceste intrari si iesiri, proiectul mai are doua linii de magistrala prin care se pot primi, dar si transmite date. Acestea sunt semnalul de clock primit de la mouse-ul USB (PS2_clk) si datele transmise serial de mouse (PS2_data).

Arhitectura proiectului are urmatoarele elemente principale, care, la randul lor, sunt alcatuite din alte componente mai mici: debouncer, PS2Communication, transmitatorul UART, Mouse Control si un afisajului cu sapte segmente (SSD). Componenta PS2Communication are rolul de a recepriona datele transmise in mod serial de catre mouse si de a le transmite mai departe unitatii MouseCtrl, transformandu-le totodata in date paralele (rx_data), dar si de a transmite in mod sertial comenzile primite paralel de la MouseCtrl catre mouse, utilizand magistrala. MouseCtrl este responsabila de interactiune cu mouse-ul, transmitand comenzi si interpretand raspunsurile primite. Trimite date legate de starea butoanelor, pozitia mouse-ului si a rotitei catre afisajul SSD si catre transmitatorul UART, care trimite aceste infotmatii calculatorului, prin semnalul de iesire *tx*.

PS2Communication

RTL

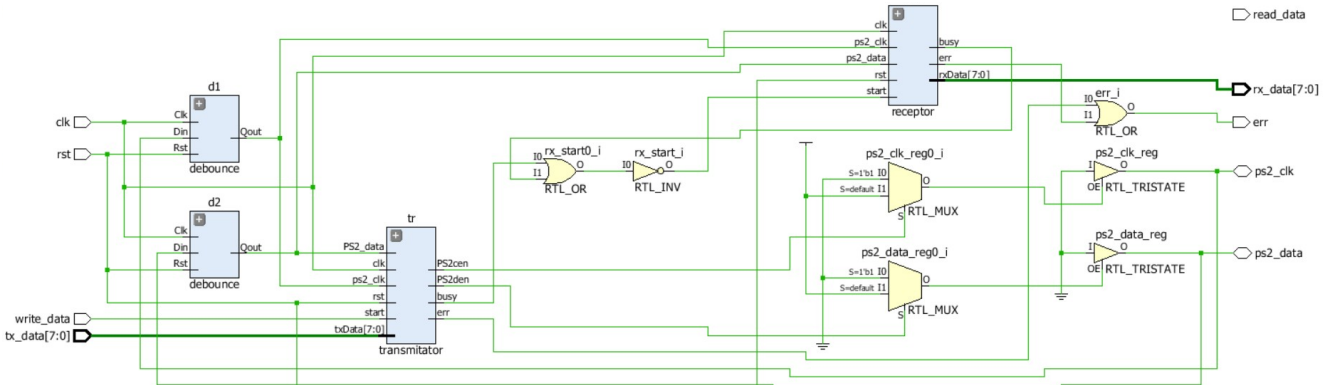

Figure 9

Descriere

Aceasta componenta are rolul de a transmite bidirectional datele intre mouse si unitatea MouseCtrl utilizand magistrala de date. Intrarile componentei sunt semnalul de clock primit de la placuta, un semnal de reset sincron, semnalul de control Write_Data, si un vector de dimensiune opt (tx_data), care contine datele ce trebuie transmise de la MouseCtrl la mouse. Iesirile sunt: semnalul *readData* care este activ in momentul in care sunt disponibile datele citite pe vectorul de opt biti rx_data, si semnalul de eroare err care este activat in momentul in care apare o problema la transmiterea sau receptionarea datelor. Pe langa acestea, mai exista

doua semnale de tipul inout,prin care sunt transmise datele (PS2_data) si semnalul de clock al mouse-ului (ps2_clk) intre mouse si componenta in ambele sensuri.

Arhitectura acestuia contine urmatoarele componente descrise in continuare: un debouncer pentru filtrarea semnalelor primite de la mouse, un transmitator care trimite datele de la *Mouse Control* catre mouse si un receptor care cireste datele primite serial de la mouse si le transmite mai departe.

Rol

Aceasta componenta este utilizata pentru a realiza comunicatia dintre componenta MouseCtrl si dispozitivul periferic, trimitand comenzile primite de la MouseCtrl, datele si raspunsurile transmise de mouse.

Debouncer

RTL

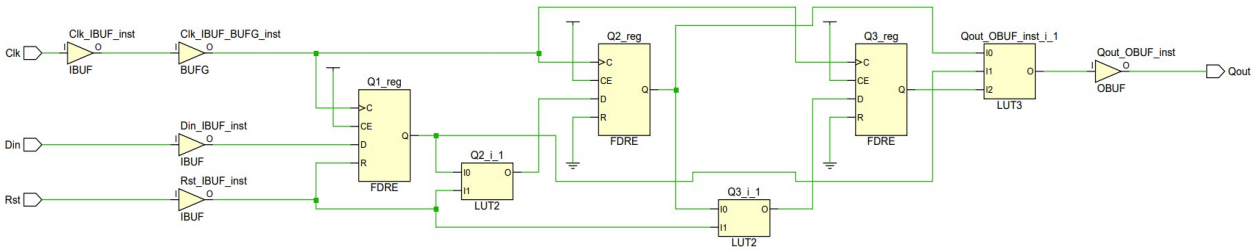

Figure 10

Descriere

Acest dispozitiv contine trei bistabile de tip D si trei porti logice. Intrarile acestuia sunt semnalul de clock, un semnal rst pentru resetarea sincrona a circuitului si datele de intrare care trebuie procesate (Din). Bistabilele functioneaza ca un registru de deplasare, iesirea fiecarui bistabil fiind legat la intrarea urmatorului. La intrarea primului bistabil este conectata intrarea Din. Pentru a determina valoarea iesirii, se utilizeaza o poarta *SI* a carei intrari sunt iesirea primelor doua bistabile si iesirea negata a celui de al treilea.

Rol

La orice circuite electrice, la apasarea butoanelor se genereaza semnale multiple din cauza contactului imperfect; de aceea, avem nevoie de un dispozitiv care sa asigure ca se trimite un singur semnalul. Acesta se leaga la intrarea generata de mouse pentru a asigura ca la apasarea butonului semnalul nu oscileaza.

Receptor

Figure 11:

Descriere

Aceasta componenta are ca intrari semnalul de clock de la placa (clk), un semnal de reset sincron (rst), semnalul de control *start* care porneste receptarea, datele primite de la mouse (PS2_Data) si *semnalul de clock al mouse-ului* (*PS2_clk*), iar iesirile sunt: semnalul *busy*, activ in timpul citirii datelor, semnalul de eroare *err* si date le receptionate sub forma unui vector de dimensiune opt, *RX*.

La inceput, unitatea de control se afla in starea "*Ready*", cand se asteapta activarea intrarii "*Start"*, dupa care trece in starea "Wait bit" in care asteapta aparitia fiecarui bit. Daca bitul receptionat este acela de start iar valoarea acestuia nu este zero, automatul de stare se reseteaza, revenint in prima stare, iar daca valoarea este corecta, se trece la starea "StartB". Asemanator se procedeaza si in cazul bitului de *stop*, dar, de aceasta data, valoarea asteptata este unu, iar trecerea automatului este in starea "stop". In momentul in care este receptionat un bit de date, se trece in starea DataB. Trecerea la oricare dintre aceste stari se face pe frontul descrescator, acesta fiind frontul pe care se actualizeaza datele. In starile "DataB" si "StartB", sunt memorate datele in vectorul de iesire. Din acestea, automatul revine in starea "WaitB" pe frontul crescator

(frontul pe care sunt citite datele). Din starea “Stop”, se revine in starea initiala si se asteapta aparitia unui nou set de date.
In ultimul proces sunt atribuite valorile corespunzatoare iesirilor in functie de starea curenta. La aparitia unei probleme la receptionarea datelor, semnalul err este activat, iar semnalul busy este actin pe perioada citirii datelor si inactiv in starea initiala, cand se asteapta aparitia semnalului *start*.

Rol

Aceasta componenta are rolul de a receptiona semnalele primite serial de la mouse si de a le transmite sub forma unui vector. Un alt rol pe care il are receptorul este acela de a verifica corectitudinea datelor de intrare si de a informa utilizatorul de apatitia unor nereguli ca: o valoare neasteptata pentru bitul de pornire, sau pentru cel de oprire, durata prea mica a semnalului de pornire sau paritate gresita, prin activarea unui semnal de eroare.

Transmitator

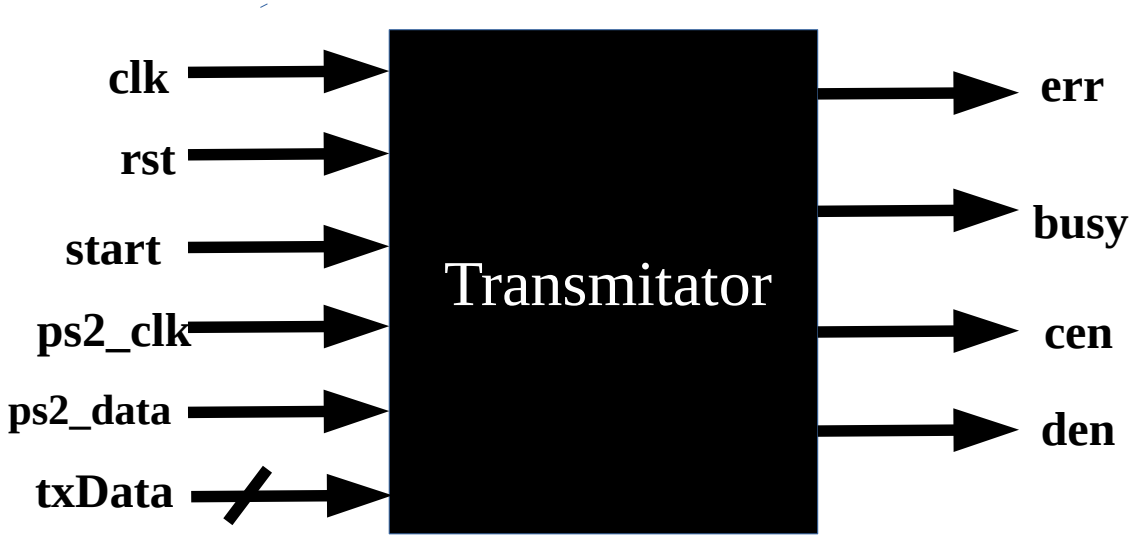

Figure 12:

Descriere

Dupa cum se poate observa in figura de mai sus, transmitatorul are ca intrari semnalul de clock al placii (clk), un semnal de reset sincron (rst), semnalul de control *start* care porneste transmisia, datele primite de la mouse (PS2_Data), *semnalul de clock al* mouse*-ului* (*PS2_clk*),

si un vector de dimensiune opt ce contine *d*atele care trebuie trimise (txData*)*, iar iesirile sunt: semnalul *busy*, activ in timpul trimiterii datelor, semnalul de eroare *err*.

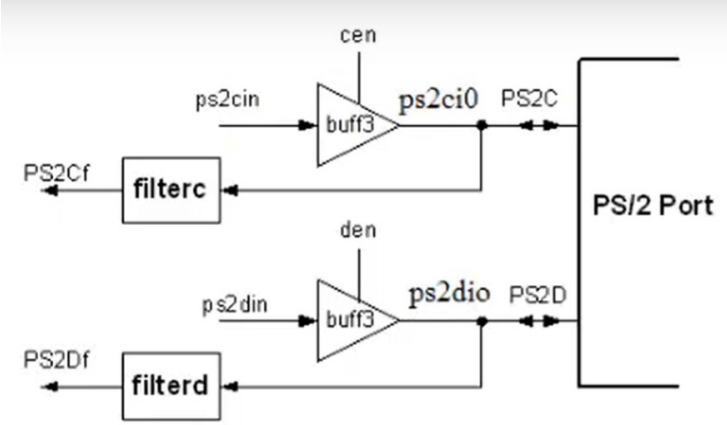

Figure 13

Pe langa acestea, mai exista doua iesiri *“cen”* si *“den”* care au rolul de a controla bufferele de la cele doua intraripentru a intrerupe transmisia datelor de pe magistrala, dupa cum se observa in figura de mai sus. Astfel, cand acestea au valoarea zero, transmisia este oprita, iar PS2_clk si Ps2_data au valoarea zero.

Pentru a transmite date de la gazda la mouse, trebuie urmati urmatorii pasi:

1. Se aduce linia pentru semnalul de clock la valoarea zero pentru 100 microsecunde,
2. Se aduce linia de date la valoarea zero,
3. Este eliberata linia pentru clock,
4. Se transmite bitul de start
5. Se asteapta frontul crescator al semnalului de ceas
6. Se transmite fiecare bit in parte controland linia de date prin intermediul semnalului cen,
7. Se asteapta ca dispozitivul periferic sa transmita zero pe ambele linii de date ca raspuns la datele primite.
8. Sunt eliberate ambele linii.

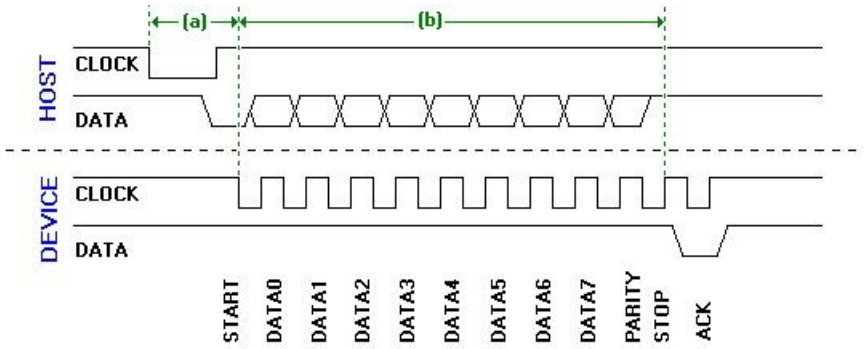

Figure 14

La inceput, unitatea de control se afla in starea "*Ready*", cand se asteapta activarea intrarii "*Start"*, dupa care trece in starea "Wait bit" in care asteapta aparitia fiecarui bit. Daca bitul receptionat este acela de start iar valoarea acestuia nu este zero, automatul de stare se reseteaza, revenint in prima stare, iar daca valoarea este corecta, se trece la starea "StartB". Asemanator se procedeaza si in cazul bitului de *stop,* dar, de aceasta data, valoarea asteptata este unu, iar trecerea automatului este in starea "stop". In momentul in care este receptionat un bit de date, se trece in starea DataB. Trecerea la oricare dintre aceste stari se face pe frontul descrescator, acesta fiind frontul pe care se actualizeaza datele. In starile "DataB" si "StartB", sunt memorate datele in vectorul de iesire. Din acestea, automatul revine in starea "WaitB" pe frontul crescator (frontul pe care sunt citite datele). Din starea "Stop", se revine in starea initiala si se asteapta aparitia unui nou set de date.
In ultimul proces sunt atribuite valorile corespunzatoare iesirilor in functie de starea curenta. La aparitia unei probleme la receptionarea datelor, semnalul err este activat, iar semnalul busy este actin pe perioada citirii datelor si inactiv in starea initiala, cand se asteapta aparitia semnalului *start*.

Rol

Aceasta componenta are rolul de a transmite comenzile de la componenta "*Mouse Control" l*a dispozitivul periferic prin modificarea valorilor semnalelor "cen" si "den" si de a anunta utilizatorul de aparitia unor erori la transmiterea datelor.

Transmitator UART

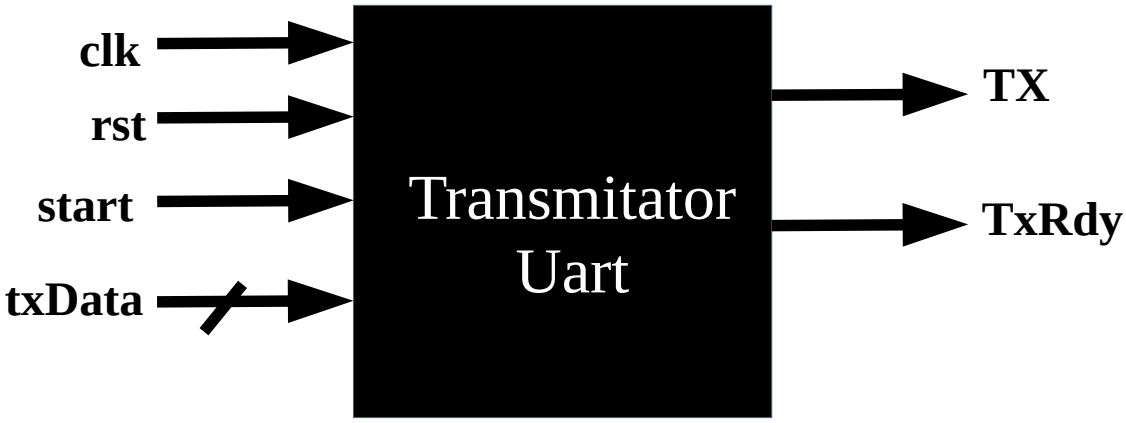

Figure 15:

Descriere

Un UART este un circuit utilizat pentru comunicatii seriale pe un port serial al dispozitivului periferic. Intrarile componentei sunt semnalul de clock, un semnal de reset sincron, semnalul de control, S*tart,* si un vector de noua elemente, reprezentand datele transmise. Iesirile sunt semnalul *txRdy* care este activ in momentul in care au fost transmise toate datele de intrare si *tx* prin care sunt transmise datele de intrare in mod serial. Pentru implementarea componentei au fost descrise patru procese: trei pentru unitatea de control (primul pentru descrierea registrului de stare, al doilea pentru determinarea starii urmatoare, iar ultimul pentru actualizarea semnalelor de control) si unul pentru registrul de deplasare.

Rol

Aceasta componenta este utilizata pentru a transmite datele de intrare in mod serial si, mai ales, pentru a face posibila configurarea formatului datelor transmise, dar si a vitezei de transmitere a acestora.

Mouse Control

1 RTL

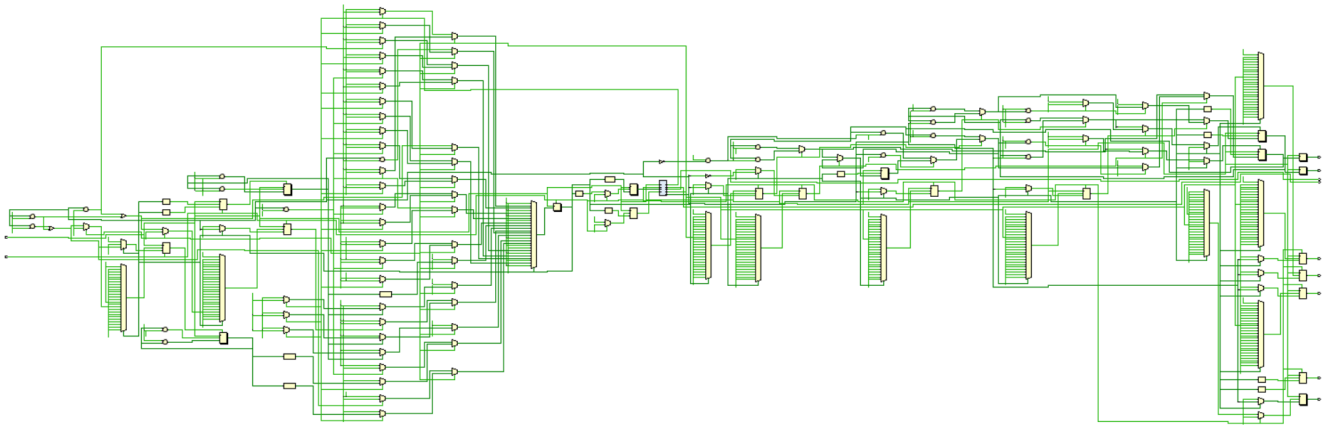

Figure 16

2 Descriere

Dupa cum se observa si in schema de mai sus, intrarile sisremului sunt: semnalul de clock al placutei Nexys4 (clk), semnalul de reset sincron (rst), Semnalul de eroare err primit de la UART, care semnaleaza aparitia unei erori la transmiterea/receptionarea datelor; caz in care este resetat, semnalul read_data, care este activat in momentul in care sunt disponibile datele primite de la mouse, si vectorul rx_data care contine datele citite. Iesirile sunt: semnalul tx_data activat cand se trnsmit comenzi catre mouse, write_data ce contine comanda transmisa, coordonata pe axele orizontala (x_pos), respectiv pe axa verticala(ypos) a mouse-ului, miscarea rotitei (zpos) starea celor trei butoane: stang (left), drept (right) si mijlociu (middle). La cele doua tipuri de semnale se adauga cele doua semnale primite de la mouse.

3 Rol

MouseCtrl are scopul de a interactiona cu mouse-ul prin intermediul componentei PS2Com, transmitand comenzi specifice si interpretand raspunsurile si datele primite de la acesta.

Rezultate experimentale

Debouncer

Simulare

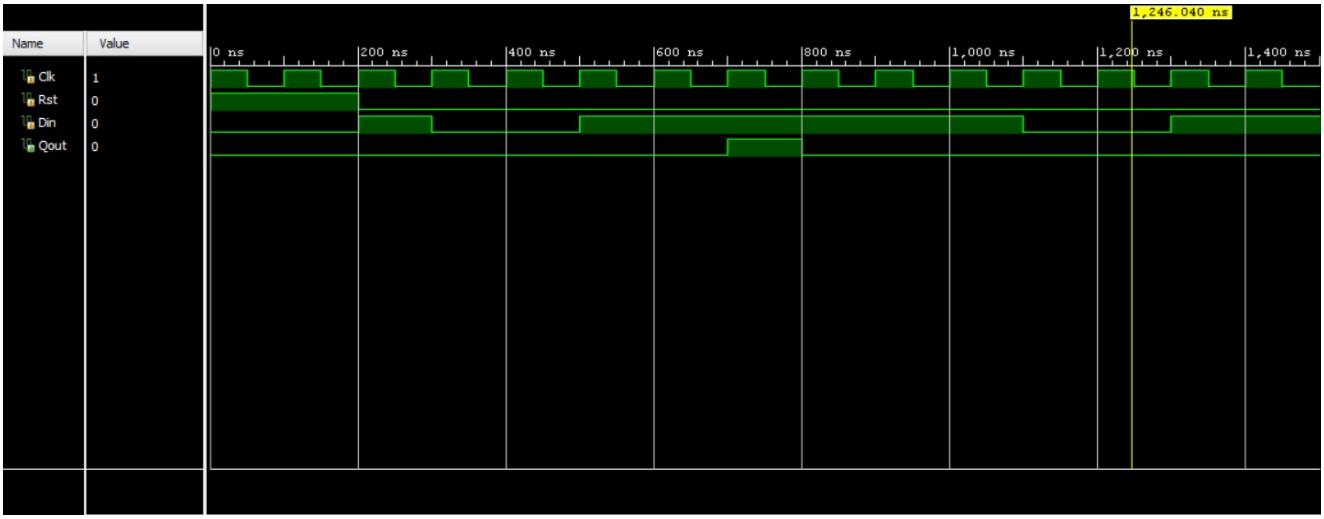

Figure 17

Transmitator UART

Figure 18

Transmitator

Figure 19

Receptor

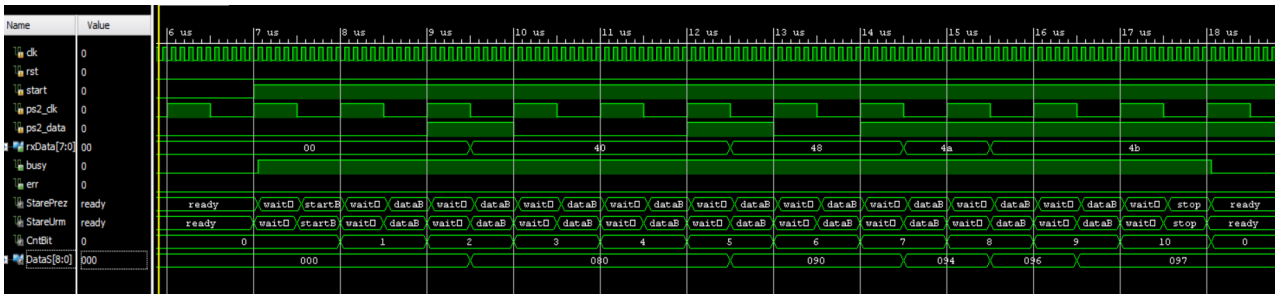

Figure 20

Bibliografie

[sursa online noiembrie 2021] https://digilent.com/reference/_media/reference/programmable-logic/nexys-4/nexys4_rm.pdf

[sursa online noiembrie 2021] https://viaembedded.github.io/docs/vab1000-user-manual/jumpers.html

[sursa online noiembrie 2021] https://users.utcluj.ro/~baruch/ssc/labor/Testare-Depanare.pdf

[sursa online noiembrie 2021] http://www.burtonsys.com/ps2_chapweske.htm

Anexa 1

Debouncer

```
entity debounce is
    Port ( Clk : in STD_LOGIC;
           Rst : in STD_LOGIC;
           Din : in STD_LOGIC;
           Qout : out STD_LOGIC);
end debounce;

architecture Behavioral of debounce is
signal Q1, Q2, Q3 : std_logic;
begin

process(Clk)
begin
  if (Clk'event and Clk = '1') then
    if (Rst = '1') then
      Q1 <= '0';
      Q2 <= '0';
      Q3 <= '0';
    else
      Q1 <= Din;
      Q2 <= Q1;
      Q3 <= Q2;
    end if;
  end if;
end process;

Qout <= Q1 and Q2 and (not Q3);
end Behavioral;
```

Anexa 2

Transmitator

```
architecture Behavioral of transmitator is
 type TIP_STARE is (ready, PS2Clk0, PS2Data0, relclk, down_edge, clk_low, up_edge, clk_h,
stop_bit, wait_ack, ack);
   signal  StarePrez, StareUrm : TIP_STARE;
   signal LdEn : std_logic := '0';
   signal ShEn : std_logic := '0';
   signal TSR : std_logic_vector(10 downto 0) := (others => '0');
   signal cntBit : integer := 0;
   signal cnt,cnt20 : integer := 0;
begin

proc1: process (Clk)
 begin
    if RISING_EDGE (Clk) then
       if (Rst = '1') then
          StarePrez <= ready;
       else
           StarePrez <= StareUrm;
       end if;
     end if;
 end process proc1;

 busy <= '0' when StarePrez = ready else '1';
 shEn <= '1' when StarePrez = clk_low else '0';
 ldEn <= '1' when StarePrez = Ps2clk0 else '0';

 proc_control: process (StarePrez, start, clk)
 begin


    case StarePrez is
       when ready =>
          err <= '0';
          Cnt <= 0;
          Cnt20 <= 0;
          CntBit <= 0;
          if (Start = '1') then
             StareUrm <= PS2Clk0;
          end if;
```

```
when PS2Clk0 =>
   ps2cen <= '0';
   if(cnt = 2000) then
      StareUrm <= PS2Data0;
      cnt <= 0;
   elsif(rising_edge(clk)) then
        cnt <= cnt+1;
     end if;

when ps2Data0 =>
   ps2cen <= '0';
   ps2den <= '0';
   if(cnt20 = 400) then
       stareUrm <= relclk;
       cnt20 <= 0;
    elsif(rising_edge(clk)) then
       cnt20 <= cnt20 + 1;
    end if;

when relclk =>
    ps2cen <= '1';
    ps2den <= '0';
    stareUrm <= down_edge;

  --clk nu este restabilit imediat => asteptam
when down_edge =>
    ps2den <= '0';
    if(ps2_clk = '0') then
        stareUrm <= clk_low;
    end if;

when clk_low =>
    if rising_edge(clk) then
       cntbit <= cntbit + 1;
    end if;
    ps2den <= TSR(0);
    stareUrm <= up_edge;

when up_edge =>
    ps2den <= TSR(0);
    if(cntBit =  10) then
```

```
            ps2den <= '1';
            stareUrm <= stop_bit;
          elsif(ps2_clk = '1') then
            stareUrm <= clk_h;
          end if;

        when clk_h =>
          ps2den <= TSR(0);
          if(ps2_clk = '0') then
            stareUrm <= clk_low;
          end if;

        when stop_bit =>
          ps2den <= '1';
          if(ps2_clk = '1') then
            stareUrm <= wait_ack;
          end if;

        when wait_ack =>
          if(ps2_clk = '0') then
            stareUrm <= ack;
            if(ps2_data /= '0') then
               err <= '1';
            end if;
          end if;

        when ack =>
          if(ps2_clk = '1' and ps2_data = '1') then
            stareUrm  <= ready;
          end if;

       when others =>
          StareUrm <= ready;
    end case;
 end process proc_control;

reg: process(Clk)
  begin
   if(rising_edge(Clk)) then
      if (rst = '1') then
          TSR <= (others => '0');
      else
```

```
        if (ldEn = '1') then
           TSR(8 downto 1) <= txData;
           TSR(0) <= '0';
           TSR(10) <= '1';
           TSR(9) <= txData(0) xor txData(1) xor txData(2) xor txData(3) xor txData(4) xor txData(5)
xor txData(6) xor txData(7);
        else
           if ( shEn = '1') then
              TSR <= '1' & TSR(10 downto 1);
           end if;
         end if;
      end if;
   end if;
 end process;
end Behavioral;
```

Anexa 3

Receptor UART

```
architecture Behavioral of receptor is
    type TIP_STARE is (ready, startB, waitBit, dataB, stop);
    signal  StarePrez, StareUrm : TIP_STARE;
    signal CntBit:INTEGER:=0;
    signal DataS:STD_LOGIC_VECTOR(8 DOWNTO 0):= (others => '0');
begin


proc1: process (Clk)
 begin
    if RISING_EDGE (Clk) then
       if (Rst = '1') then
          StarePrez <= ready;
       else
          StarePrez <= StareUrm;
       end if;
     end if;
 end process proc1;
```

```
proc2: process (StarePrez, ps2_data,ps2_clk, start, clk)
 begin
    case StarePrez is
       when ready =>
          CntBit <= 0;
          if (Start = '1' and ps2_data = '0') then
            StareUrm <= WaitBit;
          end if;

        when WaitBit=>
         if falling_edge(ps2_clk) then
              if cntBit = 0 then
                 if ps2_Data /= '0' then
                       StareUrm <= ready;
                 else
                       StareUrm <= startB;
                end if;
              elsif cntBit <= 9 then
                 StareUrm <= dataB;
                 DataS(9-CntBit) <= ps2_Data ;
              elsif cntBit = 10 then
                 StareUrm <= stop;
              end if;
           end if;

         when startB =>
             if rising_edge(ps2_clk) then
                 cntBit <= cntBit + 1;
                 StareUrm <= waitBit;
             end if;

       when dataB =>
           if rising_edge(ps2_clk) then
             CntBit <= CntBit + 1;
             StareUrm <= WaitBit;
           end if;

       when stop=>
           if rising_edge(ps2_clk) then
             StareUrm <= ready;
           end if;
```

```
        when others =>
            stareUrm <= ready;
     end case;
  end process;

proc3: process (StarePrez)
 begin
    case StarePrez is
       when ready => busy <= '0'; err<= '0';
       when startB => busy <= '1'; err <= ps2_data;
       when waitBit => busy <= '1'; err <= '0';
       when dataB => busy <= '1';  err <= '0';
       when stop => busy <= '1'; err <= not(ps2_data);
       when others => busy <= '1'; err <= '1';
    end case;
 end process proc3;
RxData <= dataS(8 downto 1);
end Behavioral;
```

Anexa 4

PS2Communication

```
architecture Behavioral of UART is
signal tx : std_logic := '1';
signal tx_rdy : std_logic := '0';
signal rx_start : std_logic := '1';
signal b1, b2 : std_logic := '0';
signal err1, err2 : std_logic := '0';
signal ps2clk_deb, ps2data_deb : std_logic := '0';
signal cen, den : std_logic := '1';
begin

d1: debounce port map (clk, rst, ps2_clk, ps2clk_deb);

d2: debounce port map (clk, rst, ps2_data, ps2data_deb);

tr:transmitator Port map (clk, rst, write_data, tx_Data, ps2clk_deb, PS2data_deb, cen, den, b1, err1);

rec:receptor port map ( clk, rst, rx_start, ps2clk_deb, ps2data_deb, rx_data, b2, err2);
```

```
ps2_clk <= 'Z' when cen = '1' else '0';
ps2_data <= 'Z' when den = '1' else '0';
ps2_data <= ps2_data and tx;

rx_start <= not( b1 or b2);
err <= err1 or err2;
end Behavioral;
```

Anexa 5

Mouse Control

```
architecture Behavioral of MouseCtrl is

constant CHECK_PERIOD_MS     : integer := 500;
constant TIMEOUT_PERIOD_MS   : integer := 100;
constant FA: std_logic_vector(7 downto 0) := "11111010"; -- 0xFA(ACK)
constant FF: std_logic_vector(7 downto 0) := "11111111"; -- 0xFF(RESET)
constant AA: std_logic_vector(7 downto 0) := "10101010"; -- 0xAA(START)
constant OO: std_logic_vector(7 downto 0) := "00000000"; -- 0x00(=>ID)

constant READ_ID          : std_logic_vector(7 downto 0) := x"F2";
constant ENABLE_REPORTING : std_logic_vector(7 downto 0) := x"F4";
constant SET_RESOLUTION   : std_logic_vector(7 downto 0) := x"E8";
constant RESOLUTION       : std_logic_vector(7 downto 0) := x"03"; -- (8 counts/mm)
constant SET_SAMPLE_RATE  : std_logic_vector(7 downto 0) := x"F3";
constant SAMPLE_RATE      : std_logic_vector(7 downto 0) := x"28"; -- (40 samples/s)
constant MAX_X : std_logic_vector(11 downto 0) := x"4FF"; -- 1279
constant MAX_Y : std_logic_vector(11 downto 0) := x"3FF";-- 1023

 signal haswheel: std_logic := '0';
signal x_pos, y_pos: std_logic_vector(11 downto 0) := (others => '0');
signal x_overflow, y_overflow: std_logic := '0';
signal x_sign, y_sign: std_logic := '0';

type fsm_state is
(
   reset, reset_wait_ack, start, wait_id,
   Sread_id, read_id_wait_ack, read_id_wait_id,
   Sset_resolution, set_resolution_wait_ack,
   send_resolution, send_resolution_wait_ack,
   set_rate, set_rate_wait_ack, send_rate, send_rate_wait_ack,
```

```
   en_reporting, en_reporting_wait_ack,
   read_byte_1,read_byte_2,read_byte_3,read_byte_4,
   check_id, check_id_wait_ack, check_id_wait_id,
   Snew_event
);
signal state: fsm_state := reset;
signal read_data  : std_logic;
signal err  : std_logic;
signal rx_data: std_logic_vector (7 downto 0);
signal tx_data: std_logic_vector (7 downto 0);
signal write_data : std_logic;
constant CHECK_PERIOD_CLOCKS   : integer :=
((CHECK_PERIOD_MS*SYSCLK_FREQUENCY_HZ)/(1000));
signal periodic_check_cnt       : integer := 0;
signal reset_periodic_check_cnt  : STD_LOGIC := '0';
constant TIMEOUT_PERIOD_CLOCKS : integer :=
((TIMEOUT_PERIOD_MS*SYSCLK_FREQUENCY_HZ)/(1000));
signal timeout_cnt       : integer range 0 to (TIMEOUT_PERIOD_CLOCKS - 1) := 0;
signal reset_timeout_cnt  : STD_LOGIC := '0';

begin

U: UART PORT MAP(ps2_clk, ps2_data, clk, rst, tx_data, write_data, rx_data, read_data, err);

-------------------------------------------------------------------------------------------------------------------------
-----
-------------------------------------------------------------->  CHECK CNT
<---------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-----

Count_periodic_check: process (clk, periodic_check_cnt, reset_periodic_check_cnt)
begin
   if clk'EVENT AND clk = '1' then
      if reset_periodic_check_cnt = '1' then
         periodic_check_cnt <= 0;
      elsif periodic_check_cnt < (CHECK_PERIOD_CLOCKS - 1) then
         periodic_check_cnt <= periodic_check_cnt + 1;
      end if;
   end if;
end process Count_periodic_check;
```

```
-------------------------------------------------------------------------------------------------------------------------
-----
-------------------------------------------------------------->  TIMEOUT CNT
<---------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-----

Count_timeout: process (clk, timeout_cnt, reset_timeout_cnt)
begin
   if clk'EVENT AND clk = '1' then
      if reset_timeout_cnt = '1' then
         timeout_cnt <= 0;
      elsif timeout_cnt < (TIMEOUT_PERIOD_CLOCKS - 1) then
         timeout_cnt <= timeout_cnt + 1;
      end if;
   end if;
end process Count_timeout;


   xpos <= x_pos when rising_edge(clk);
   ypos <= y_pos when rising_edge(clk);


-------------------------------------------------------------------------------------------------------------------------
-----
-------------------------------------------------------------->  SET X
<-------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
  set_x: process(clk)
   variable x_inter: std_logic_vector(11 downto 0);
  begin
      if(rising_edge(clk)) then
         if(state = read_byte_2) then
            if(x_sign = '1') then
               if(x_overflow = '1') then
                  x_inter := x_pos + "111000000000";-- inc is -256
               else
                  x_inter := x_pos + ("1111" & rx_data);
               end if;
               if(x_inter(11) = '1') then
                  x_pos <= (others => '0');
               else
```

```
            x_pos <= x_inter;
          end if;
        else
          if(x_overflow = '1') then
            x_inter := x_pos + "000100000000"; -- inc is 256
          else
            x_inter := x_pos + ("0000" & rx_data);
          end if;
          if(x_inter > ('0' & MAX_X)) then
            x_pos <= MAX_X;
          else
            x_pos <= x_inter;
          end if;
        end if;
      end if;
    end if;
  end process set_x;

-------------------------------------------------------------------------------------------------------------------------
-----
-------------------------------------------------------------->  SET Y
<-------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-----

  set_y: process(clk)
  variable y_inter: std_logic_vector(11 downto 0);
  begin
    if(rising_edge(clk)) then
      if(state = read_byte_3) then
        if(y_sign = '1') then
          if(y_overflow = '1') then
            y_inter := y_pos + "111100000000"; -- inc is -256
          else
            y_inter := y_pos + ("1111" & ((not rx_data) + "00000001"));
          end if;
          if(y_inter(11) = '1') then
            y_pos <= (others => '0');
          else
            y_pos <= y_inter;
          end if;
        else
```

```
            if(y_overflow = '1') then
              y_inter := y_pos + "000100000000"; -- inc is 256
            else
                if(rx_data /= X"00") then
                     y_inter := y_pos + ("0000" & ((not rx_data) + "00000001"));
                end if;
            end if;
            if (y_inter > (MAX_Y)) then
              y_pos <= MAX_Y;
            else
              y_pos <= y_inter;
            end if;
          end if;
        end if;
      end if;
    end process set_y;

-------------------------------------------------------------------------------------------------------------------------
-----
-------------------------------------------------------------->   FSM
<-------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
    manage_fsm: process(clk,rst)
    begin
      if(rst = '1') then
        state <= reset;
        haswheel <= '0';
        x_overflow <= '0';
        y_overflow <= '0';
        x_sign <= '0';
        y_sign <= '0';
        new_event <= '0';
        left <= '0';
        middle <= '0';
        right <= '0';
        reset_periodic_check_cnt <= '1';
        reset_timeout_cnt <= '1';


      elsif(rising_edge(clk)) then
        write_data <= '0';
        case state is
```

```
        -- powered-up/reset/error then => RESET state. RESET command (FF) is sent to the mouse
        -- From here the FSM transitions to a series of states that perform the mouse initialization
procedure.

-------------------------------------------------------------------->  RESET
        when reset =>
          haswheel <= '0';
          x_overflow <= '0';
          y_overflow <= '0';
          x_sign <= '0';
          y_sign <= '0';
          left <= '0';
          middle <= '0';
          right <= '0';
          tx_data <= FF;
          write_data <= '1';
          reset_periodic_check_cnt <= '1';
          reset_timeout_cnt <= '1';
          state <= reset_wait_ack;

        when reset_wait_ack =>
          if(read_data = '1') then
            if(rx_data = FA) then
              state <= start;
            else
              state <= reset;
            end if;
          elsif(err = '1') then
            state <= reset;
          else
            state <= reset_wait_ack;
          end if;

-------------------------------------------------------------------->  START
        when start =>
          if(read_data = '1') then
            if(rx_data = AA) then
              state <= wait_id;
            else
              state <= reset;
            end if;
```

```
            elsif(err = '1') then
              state <= reset;
            else
              state <= start;
            end if;

----------------------------------------------------------------->  WAIT ID
          when wait_id =>
            if(read_data = '1') then
              if(rx_data = OO) then
                state <= Sread_id;
              else
                state <= reset;
              end if;
            elsif(err = '1') then
              state <= reset;
            else
              state <= wait_id;
            end if;

          when Sread_id =>
            tx_data <= READ_ID;
            write_data <= '1';
            state <= read_id_wait_ack;

          when read_id_wait_ack =>
            if(read_data = '1') then
              if(rx_data = FA) then
                state <= read_id_wait_id;
              else
                state <= reset;
              end if;
            elsif(err = '1') then
              state <= reset;
         else
              state <= read_id_wait_ack;
            end if;

          when read_id_wait_id =>
            if(read_data = '1') then
              if(rx_data = x"00" or rx_data = x"03") then
                haswheel <= rx_data(0);-- = 0 pt data = 0 \ =1 pt data = 3
```

```
            state <= Sset_resolution;
          else
            state <= reset;
          end if;
        elsif(err = '1') then
          state <= reset;
        else
          state <= read_id_wait_id;
        end if;

-------------------------------------------------------------->  RESOLUTION
      when Sset_resolution =>
        tx_data <= SET_RESOLUTION;
        write_data <= '1';
        state <= set_resolution_wait_ack;

      when set_resolution_wait_ack =>
        if(read_data = '1') then
          if(rx_data = FA) then
            state <= send_resolution;
          else
            state <= reset;
          end if;
        elsif(err = '1') then
          state <= reset;
        else
          state <= set_resolution_wait_ack;
        end if;

      when send_resolution =>
        tx_data <= RESOLUTION;
        write_data <= '1';
        state <= send_resolution_wait_ack;

      when send_resolution_wait_ack =>
        if(read_data = '1') then
          if(rx_data = FA) then
            state <= set_rate;
          else
            state <= reset;
          end if;
        elsif(err = '1') then
```

```
            state <= reset;
          else
            state <= send_resolution_wait_ack;
          end if;

----------------------------------------------------------------->   SET SAMPLE RATE
        when set_rate =>
          tx_data <= SET_SAMPLE_RATE;
          write_data <= '1';
          state <= set_rate_wait_ack;

        when set_rate_wait_ack =>
          if(read_data = '1') then
            if(rx_data = FA) then
              state <= send_rate;
            else
              state <= reset;
            end if;
          elsif(err = '1') then
            state <= reset;
          else
            state <= set_rate_wait_ack;
          end if;

        when send_rate =>
          tx_data <= SAMPLE_RATE;
          write_data <= '1';
          state <= send_rate_wait_ack;

        when send_rate_wait_ack =>
          if(read_data = '1') then
            if(rx_data = FA) then
              state <= en_reporting;
            else
              state <= reset;
            end if;
          elsif(err = '1') then
            state <= reset;
          else
            state <= send_rate_wait_ack;
          end if;
```

```
------------------------------------------------------------------>  ENABLE REPORTING
        when en_reporting =>
          tx_data <= ENABLE_REPORTING;
          write_data <= '1';
          state <= en_reporting_wait_ack;

        when en_reporting_wait_ack =>
          if(read_data = '1') then
            if(rx_data = FA) then
              state <= read_byte_1;
            else
              state <= reset;
            end if;
          elsif(err = '1') then
            state <= reset;
          else
            state <= en_reporting_wait_ack;
          end if;

------------------------------------------------------------------>  BYTE 1
        when read_byte_1 =>
          reset_periodic_check_cnt <= '0';
          new_event <= '0';
          zpos <= (others => '0');
          if(read_data = '1') then
            left <= rx_data(0);
            middle <= rx_data(2);
            right <= rx_data(1);
            x_sign <= rx_data(4);
            y_sign <= not rx_data(5);
            x_overflow <= rx_data(6);
            y_overflow <= rx_data(7);
            state <= read_byte_2;
          elsif periodic_check_cnt = (CHECK_PERIOD_CLOCKS - 1) then -- Check periodically if
the mouse is present
            state <= check_id;
          else
            state <= read_byte_1;
          end if;


------------------------------------------------------------------>  BYTE2
```

```
        when read_byte_2 =>
          if(read_data = '1') then
            state <= read_byte_3;
          elsif periodic_check_cnt = (CHECK_PERIOD_CLOCKS - 1) then
            state <= check_id;
          elsif(err = '1') then
            state <= reset;
          else
            state <= read_byte_2;
          end if;

-----------------------------------------------------------------> BYTE3
        when read_byte_3 =>
          if(read_data = '1') then
            if(haswheel = '1') then
              state <= read_byte_4;
            else
              state <= Snew_event;
            end if;
          elsif periodic_check_cnt = (CHECK_PERIOD_CLOCKS - 1) then
            state <= check_id;
          elsif(err = '1') then
            state <= reset;
          else
              state <= read_byte_3;
          end if;

-----------------------------------------------------------------> BYTE4
        -- only reached when mouse is in scroll mode wait for the fourth byte to arrive
        when read_byte_4 =>
          if(read_data = '1') then
            zpos <= rx_data(3 downto 0);
            state <= Snew_event;
          elsif periodic_check_cnt = (CHECK_PERIOD_CLOCKS - 1) then
            state <= check_id;
          elsif(err = '1') then
            state <= reset;
          else
            state <= read_byte_4;
          end if;

-----------------------------------------------------------------> CHECK
```

```
when check_id =>
  reset_timeout_cnt <= '0';
  tx_data <= READ_ID;
  write_data <= '1';
  state <= check_id_wait_ack;

when check_id_wait_ack =>
  if(read_data = '1') then
    if(rx_data = FA) then
      state <= check_id_wait_id;
    else
      state <= reset;
    end if;
  elsif(err = '1') then
    state <= reset;
  elsif (timeout_cnt = (TIMEOUT_PERIOD_CLOCKS - 1)) then
    state <= reset;
  else
    state <= check_id_wait_ack;
  end if;

when check_id_wait_id =>
  if(read_data = '1') then
    if(rx_data = "000000000") or (rx_data = "00000011") then
      reset_timeout_cnt <= '1';
      state <= read_byte_1;
    else
      state <= reset;
    end if;
  elsif(err = '1') then
    state <= reset;
  elsif (timeout_cnt = (TIMEOUT_PERIOD_CLOCKS - 1)) then
    state <= reset;
  else
    state <= check_id_wait_id;
  end if;


----------------------------------------------------------------->   NEW EVENT
when Snew_event =>
  new_event <= '1';
  state <= read_byte_1;
```

```
          when others =>
            state <= reset;

        end case;
      end if;
    end process manage_fsm;

end Behavioral;
```

Anexa 6

```
architecture Behavioral of Pr36 is
component SSD is
Port ( Clk  : in  STD_LOGIC;
          Rst  : in  STD_LOGIC;
          Data : in  STD_LOGIC_VECTOR (31 downto 0);
          An   : out STD_LOGIC_VECTOR (7 downto 0);
          Seg  : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component MouseCtrl is
generic
(
   SYSCLK_FREQUENCY_HZ : integer := 100000000
);
port(
   clk         : in std_logic;
   rst         : in std_logic;
   xpos        : out std_logic_vector(11 downto 0);
   ypos        : out std_logic_vector(11 downto 0);
   zpos        : out std_logic_vector(3 downto 0);
   left        : out std_logic;
   middle      : out std_logic;
   right       : out std_logic;
   new_event   : out std_logic;
   ps2_clk     : inout std_logic;
   ps2_data    : inout std_logic
);
end component;

signal data: std_logic_vector(31 downto 0) := (others => '0');
```

```
begin

ctrl: MouseCtrl Port map (clk, rst, data(31 downto 20), data(19 downto 8), data(7 downto 4),data(0),
data(2) ,data(1), open, ps2_clk, ps2_data);

s: ssd port map(clk, rst, data, an, cat);

end Behavioral;
```

Transmitator UART

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

entity UART_tx is
 generic( fr: integer := 100_000_000;
          bRate: integer := 8680_555);
  port( clk            : in  std_logic;
        rst          : in  std_logic;
        start        : in  std_logic;
        txData       : in  std_logic_vector (7 downto 0);
        tx     : out std_logic;
        TxRdy : out STD_LOGIC);
end UART_tx;

architecture Behavioral of uart_tx is
 type TIP_STARE is (ready, load, send, waitBit, shift);
 signal StarePrez, StareUrm : TIP_STARE;
 signal LdData : std_logic := '0';
 signal ShData : std_logic := '0';
 signal TxEn : std_logic := '0';
 signal TSR : std_logic_vector(9 downto 0) := (others => '0');
 signal cntBit : integer := 0;
 signal cntRate : integer := 0;
 constant T_BIT : integer := fr/bRate;
begin
proc1: process (Clk)
 begin
    if RISING_EDGE (Clk) then
       if (Rst = '1') then
          StarePrez <= ready;
       else
```

```
          StarePrez <= StareUrm;
        end if;
    end if;
 end process proc1;

proc2: process (StarePrez, start, clk, cntRate, cntBit)
 begin
    case StarePrez is
       when ready =>
          CntRate <= 0;
          CntBit <= 0;
          if (Start = '1') then
             StareUrm<= load;
          end if;
       when load =>
          StareUrm<= send;
       when send =>
          if RISING_EDGE (Clk) then
             CntBit <= CntBit + 1;
          end if;
          StareUrm<= waitbit;
       when waitbit =>
          if RISING_EDGE (Clk) then
             CntRate <= CntRate + 1;
          end if;
          if (CntRate = T_BIT-3) then
             CntRate <= 0;
             StareUrm<= shift;
          end if;
       when shift =>
          if (CntBit = 10) then
             StareUrm<= ready;
          else
             StareUrm<= send;
          end if;
       when others =>
          StareUrm <= ready;
    end case;
 end process proc2;

proc3: process (StarePrez, TSR)
 begin
```

```
    case StarePrez is
       when ready => LdData <= '0'; ShData <= '0'; TxEn <= '0'; Tx <= '1'; TxRdy <= '1';
       when load => LdData <= '1'; ShData <= '0'; TxEn <= '0'; Tx <= '1'; TxRdy <= '0';
       when send => LdData <= '0'; ShData <= '0'; TxEn <= '1'; Tx <= TSR(0); TxRdy <= '0';
       when waitBit => LdData <= '0'; ShData <= '0'; TxEn <= '1'; Tx <= TSR(0); TxRdy <= '0';
       when shift => LdData <= '0'; ShData <= '1'; TxEn <= '1'; Tx <= TSR(0); TxRdy <= '0';
    end case;
 end process proc3;

reg: process(Clk)
 begin
    if(rising_edge(Clk)) then
       if (rst = '1') then
          TSR <= (others => '0');
       else
          if (ldData = '1') then
             TSR(8 downto 1) <= txData;
             TSR(0) <= '0';
             TSR(9) <= '1';
          else
             if (shData = '1') then
                TSR <= '0' & TSR(9 downto 1);
             end if;
          end if;
       end if;
    end if;
end process reg;
end Behavioral;
```

SSD

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

entity SSD is
Port ( CLK  : in  STD_LOGIC;
       Rst  : in  STD_LOGIC;
       Data : in  STD_LOGIC_VECTOR (31 downto 0);
       An   : out STD_LOGIC_VECTOR (7 downto 0);
       Seg  : out STD_LOGIC_VECTOR (7 downto 0));
```

```
end SSD;

architecture Behavioral of SSD is

signal Num       : STD_LOGIC_VECTOR (19 downto 0) := (others => '0');
signal LedSel    : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
signal Hex       : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');

begin

-- Divizor
divclk: process (Clk)
   begin
   if (Clk'event and Clk = '1') then
      if (Rst = '1') then
         Num <= (others =>'0');
      elsif (Num = x"FFFFF") then
         Num <= (others =>'0');
      else
         Num <= Num + 1;
      end if;
   end if;
   end process;

   LedSel <= Num (19 downto 17);

-- Selectia anodului activ
 An <= "11111110" when LedSel = "000" else
       "11111101" when LedSel = "001" else
       "11111011" when LedSel = "010" else
       "11110111" when LedSel = "011" else
       "11101111" when LedSel = "100" else
       "11011111" when LedSel = "101" else
       "10111111" when LedSel = "110" else
       "01111111" when LedSel = "111" else
       "11111111";

-- Selectia cifrei active
  Hex <= Data (3  downto  0) when LedSel = "000" else
         Data (7  downto  4) when LedSel = "001" else
         Data (11 downto  8) when LedSel = "010" else
         Data (15 downto 12) when LedSel = "011" else
         Data (19 downto 16) when LedSel = "100" else
         Data (23 downto 20) when LedSel = "101" else
         Data (27 downto 24) when LedSel = "110" else
         Data (31 downto 28) when LedSel = "111" else
         X"0";

-- Activarea/dezactivarea segmentelor cifrei active
  Seg <= "11111001" when Hex = "0001" else
         "10100100" when Hex = "0010" else
```

```
        "10110000" when Hex = "0011" else
        "10011001" when Hex = "0100" else
        "10010010" when Hex = "0101" else
        "10000010" when Hex = "0110" else
        "11111000" when Hex = "0111" else
        "10000000" when Hex = "1000" else
        "10010000" when Hex = "1001" else
        "10001000" when Hex = "1010" else
        "10000011" when Hex = "1011" else
        "11000110" when Hex = "1100" else
        "10100001" when Hex = "1101" else
        "10000110" when Hex = "1110" else
        "10001110" when Hex = "1111" else
        "11000000";

end Behavioral;
```