Universitatea Tehnică din Cluj-Napoca

Catedra de Calculatoare și Tehnologia Informației – Secția română

Comunicația dintre o placă Nexys 4 DDR și o tastatură USB HID

Proiectul este realizat de către studenta Stoian Iulia Tia, aparținând grupei 30231, Seria A.

Îndrumător de proiect: Mocan Cristi.

Data: 15 octombrie 2020.

Cuprins

[**1.** **Rezumat** 3](#_Toc60933258)

[**2.** **Introducere** 4](#_Toc60933259)

[**3.** **Fundamentare teoretică** 5](#_Toc60933260)

[**3.1** **Tehnologii utilizate** 5](#_Toc60933261)

[**3.1.1** **Interfață USB-UART (Port Serial)** 5](#_Toc60933262)

[**3.1.2** **Host USB HID** 5](#_Toc60933263)

[**3.1.3** **Controller USB HID** 5](#_Toc60933264)

[**3.1.4** **Tastatură** 6](#_Toc60933265)

[**3.1.5** **Afișor de 7 segmente** 6](#_Toc60933266)

[**4.** **Proiectare și implementare** 8](#_Toc60933267)

[**4.1** **Metoda experimentală utilizată** 8](#_Toc60933268)

[**4.2** **Soluția aleasă** 8](#_Toc60933269)

[**4.3** **Arhitectura generală a sistemului** 9](#_Toc60933270)

[**4.4** **Modulele sistemului** 9](#_Toc60933271)

[**4.4.5** **Modulul transmițătorului serial** 11](#_Toc60933272)

[**4.5** **Detalii de implementare** 13](#_Toc60933273)

[**4.6** **Manual de utilizare** 14](#_Toc60933274)

[**5.** **Rezultate experimentale** 18](#_Toc60933275)

[**6.** **Concluzii** 18](#_Toc60933276)

[**7.** **Bibliografie** 19](#_Toc60933277)

[**8.** **Anexe** 20](#_Toc60933278)

[8.1 debounce\_clasic.vhd 20](#_Toc60933279)

[8.2 uart\_tx.vhd 20](#_Toc60933280)

[8.3 displ7seg.vhd 23](#_Toc60933281)

[8.4 scan2ascii.vhd 24](#_Toc60933282)

[8.5 ps2\_keyboard.vhd 25](#_Toc60933283)

[8.6 debouncer.vhd 28](#_Toc60933284)

[8.7 modulul\_principal.vhd 29](#_Toc60933285)

[8.8 Fișierul de constrângeri 31](#_Toc60933286)

[8.9 Codul aplicației software 32](#_Toc60933287)

# **Rezumat**

Proiectul realizat este „Comunicația dintre o placă Nexys 4 DDR și o tastatură USB HID”. Placa Nexys 4 DDR este o platformă completă pentru dezvoltarea circuitelor digitale, fiind bazată pe cel mai recent tip de circuit integrat digital configurabil (FPGA-Field Programmable Gate Array), mai precis Artix7TM de la Xilinx®. Au fost folosite doar câteva dintre funcționalitățile de care dispune Nexys4 DDR, care vor fi menționate în următoarele capitole. Acestea sunt indispensabile în vederea realizării proiectului propus. Una dintre caracteristici este existența unui Host USB HID (Human Interface Device) pentru mouse-uri, tastaturi și stick-uri de memorie. Posibilitatea transmiterii și primirii seriale a fluxurilor de date (între FPGA și un dispozitiv periferic, în cazul acesta – o tastatură și PC) se datorează existenței unei interfețe USB-UART. Tototdată, cele două afișoare de 7 segmente care sunt puse la dispoziție, fiecare cu posibilitatea de afișare a câte 4 cifre [1, p.1], au fost folosite cu scopul de a afișa informații relevante cu privire la operațiile care se vor efectua asupra sistemului.

# **Introducere**

Obiectivul acestui proiect este exploatarea acestor caracteristici pentru realizarea cu succes a unei conexiuni între o placă Nexys 4 DDR și o tastatură cu port USB.

În primul rând, se dorește ca în momentul în care se apasă o tastă de la tastatura conectatată la circuitul FPGA, caracterul corespunzător să fie afișat pe unul dintre afișoarele de 7 segmente de care dispune această placă de dezvoltare. Se cere ca proiectarea să se realizeze în limbajul VHDL (Very High Description Language) de descriere a hardware-ului. Proiectul va fi creat în mediul de dezvoltare Vivado 2019.2 Design Suite de la Xilinx®.

În al doilea rând, se dorește să se implementeze (într-un limbaj de programare dorit) o aplicație software, care dispune de o interfață grafică și oferă utilizatorului posibilitățile de realizare a unei conexiuni la un port serial și de afișare într-un terminal a caracterului tastat.

Dat fiind faptul că nu se dispune de una dintre resursele necesare pentru realizarea acestui proiect (mai exact, circuitul integrat Nexys 4 DDR), se va folosi pe post de substituent placa de dezvoltare Basys 3. Această placă de dezvoltare pune la dispoziție toate tehnologiile necesare, fiind de asemenea bazată tipul de FPGA Artix7TM de la Xilinx®.

În continuare, părțile teoretice vor fi descrise pentru Nexys 4 DDR, iar cele practice pentru Basys 3.

# **Fundamentare teoretică**

## **Tehnologii utilizate**

### **Interfață USB-UART (Port Serial)**

Nexys4 DDR include o interfață USB-UART, atașată la conectorul J6, care permite comunicația între PC, FPGA și alte dispozitive periferice [1, p. 10]. Transmisia de date serială este o tehnologie de comunicație în care biți de date sunt transferați succesiv de-a lungul unui canal de comunicare [2]. În cazul Nexys4 DDR, schimbul de date se face prin intermediul portului serial format din două linii de comunicare: TXD (Transmitted Data) și RXD (Received Data).

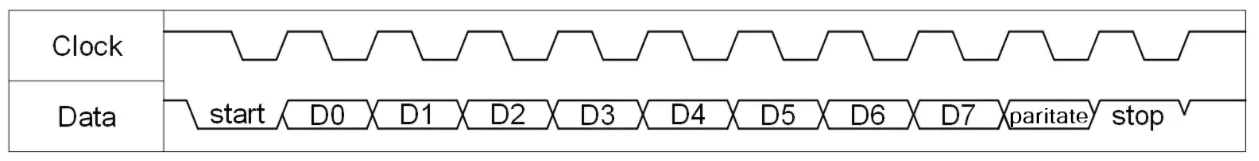
### **Host USB HID**

Nexys4 DDR nu dispune de un port PS/2, motiv pentru care „microcontroller-ul auxiliar” face capabilă găzduirea unui USB Embedded HID (Human Interface Device) prin configurația sa ca un host USB HID. Firmware-ul din microcontroller poate controla un mouse sau o tastatură atașate la conectorul USB type A la portul J5 denumit „USB Host” [1, p. 10]. De menționat este faptul că acest FPGA nu permite conectarea mai multor dispozitive perifice de tip USB HID în acelați timp.

### **Controller USB HID**

„Microcontroller-ul auxiliar” funcționeaza ca un „translator de protocol de comunicare” între un mouse sau o tastatură USB și placa de dezvoltare. Datorită configurației sale explicate la subcapitolul 3.1.2, Nexys4 DDR dispune de un port PS/2 emulat și poate genera un flux emulat de date de tip PS/2 dinspre un mouse sau o tastatură atașate la conectorul USB type A existent pe placa de dezvoltare [4].

Atât mouse-ul, cât și tastatura utilizează șiruri de 11 biți, care includ un bit de start, un octet de date (LSB first – Least Significant Bit First, adică prioritatea se stabilește în funcție de bitul cel mai nesemnificativ) , un bit de paritate (odd) și un bit de stop. Sincronizarea benzilor de comunicație este prezentată în figura de mai jos [5, Figure 2.]:



**Figura 3.1.** Diagrama de sincronizare a liniilor de comunicație a unui dispozitiv PS/2 înspre gazdă

Semnalul tactului de ceas și cel de date sunt „conduse” (adică trece curentul prin ele) doar atunci când are loc un transfer de date. Altfel, ele sunt menținute în starea inactivă „idle” la o impedață mare.

### **Tastatură**

Tastaturile tip PS/2 folosesc coduri de scanare pentru a comunica informație despre apăsarea tastelor. Fiecărei taste îi este asignat un cod de scanare care este transmis oricând aceasta este apăsată. Dacă o tastă se menține apăsată, codul de scanare va fi transmis în mod repetat, o dată la aproximativ 100ms. Când o tastă este eliberată, se transmite un cod de tastare F0, urmat de codul de scanare al tastei eliberate. Codurile de scanare pentru majoritatea tastelor sunt arătate în figura de mai jos [1, p. 12]:

A picture containing text, electronics, keyboard

Description automatically generated

**Figura 3.2.** Codurile de scanare ale unei tastaturi

Spre exemplu, dacă se apasă tasta „A” a tastaturii conectate la placa Nexys 4 DDR, se va transmite codul de scanare „1C”. În momentul în care se eliberează tasta „A”, se va transmite codul de scanare „F01C”.

### **Afișor de 7 segmente**

Placa Nexys 4 DDR conține două afișoare LED de 7 segmente cu anod comun, fiecare a câte 4 cifre. Afișoarele sunt configurate să se comporte ca un singur afișor a 8 cifre. Fiecare dintre aceste 8 cifre este compusă din 7 segmente aranjate într-un șablon al formei cifrei 8. Segmentele LED pot fi iluminate individual, astfel încât există 128 de șabloane posibile pentru configurația unei singure cifre. Dintre aceste 128 de șabloane posibile, cele care corespund cifrelor 0-9 sunt următoarele [3]:

A picture containing chart

Description automatically generated

**Figura 3.3.** Șabloanele corespunzătoare cifrelor 0-9

De asemenea, reprezentarea prin intermediul celor 7 segmente este posibilă pentru majoritatea literelor latine [3]:

Application, calendar

Description automatically generated

**Figura 3.4.** Șabloanele corespunzătoare pentru majoritatea literelor latine

# **Proiectare și implementare**

## **Metoda experimentală utilizată**

Pentru a verifica dacă transmisiile codurilor de scanare se realizează cu succes, se verifică configurația de pe afișor după apăsarea unei taste. Dacă acea configurație nu coincide cu cea așteptată (și corectă), atunci există două posibile erori: prima eroare poate avea legătură cu transmisia codului de scanare, iar a doua variantă poate avea legătură cu corelația dintre caracter-șablon 7 segmente. Folosind un emulator de terminal, se stabilește conexiunea serială între placă și PC. Pentru a afișa în terminal tasta apăsată de la tastatură, trebuie să se apese butonul BTNU (butonul reprezentând direcția – sus). Caracterul de pe afișor cu cel din terminal trebuie să coincidă, altfel este evident că există erori, fie în transmiterea codului ascii, fie în translatarea din cod scanare în cod ascii.

## **Soluția aleasă**

În ceea ce privește schimbul serial de date, linia de comunicare RX va fi folosită pentru transmiterea codurilor de scanare dinspre tastatură înspre placa de dezvoltare. Linia de comunicare TX va fi folosită pentru transmiterea codurilor de scanare (primite anterior la apăsarea unei taste) dinspre placa de dezvoltare înspre PC. Aceste coduri de scanare vor fi translatate în codul ASCII înainte de transmisie, cu scopul de a afișa pe ecranul PC-ului caracterele ASCII corespunzătoare. De menționat că dintre toate tastele din figura 3.2, numai cele care sunt evidențiate (colorate) vor avea implementată această funcționalitatea de afișare, atât pe afișor, cât și în terminalul aplicației software care va realiza conexiunea serială dintre placă și PC prin portul Windows COM cu numele „COM4”. Mai mult, având în vedere faptul că literele latine K, M, V, W, X și Z nu au o reprezentare standard în sistemul de 7 segmente, s-au ales configurații cât de cât rezonabile pentru afișarea acestora. Modul în care se va reprezenta alfabetul pe un afișor de 7 segmente (de o cifră) este prezentat în figura de mai jos [3]:



**Figura 4.1.** Șabloanele corespunzătoare literelor A-Z

Se poate observa că unele configurații coincid pentru mai multe caractere. În implementarea acestui proiect nu se va ține cont de diferența între majuscule sau minuscule, tocmai pentru a evita o frecvență mai mare a acestor repetiții de configurații identice.

Aplicația software se va implementa în limbajul de programare Python 2.7, iar pentru primirea serială de date dinspre placa de dezvoltare înspre PC se va importa pachetul denumit „serial”. Acest pachet este un framework cu ajutorul căruia se poate realiza conexiunea la portul serial și citirea datelor transmise pe acestă linie de comunicare.

## **Arhitectura generală a sistemului**

Arhitectura generală a sistemului este formată din 6 module, mai precis: modulul transmițătotului serial (uart\_tx), modulul receptorului serial (ps2\_keyboard), modulul translatorului din cod de scanare în cod ascii (scan2ascii), modulul afișorului de 7 segmente (displ7seg) și modulele pentru cele 2 tipuri de debounce (debounce – pentru sincronizarea semnalelor de la tastatură – ps2\_clock și ps2\_data – și debounce\_clasic – pentru semnalul de la butonul de la circuitul FPGA). Mai jos, se poate observa schema arhitecturii generale ale sistemului, iar în continuare se va explica funcționalitatea fiecărui modul în parte.

Diagram

Description automatically generated

**Figura 4.2.** Schema arhitecturii generale ale sistemului

## **Modulele sistemului**

* + 1. **Modulul receptorului serial**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

Chart, box and whisker chart

Description automatically generated

**Figura 4.3.** Cutia neagră a modulului receptorului serial

Rolul receptorului serial este acela de recepta bit cu bit codul de scanare al unei taste a tastaturii PS/2, în momentul în care aceasta este apăsată. Câte un bit din acest cod este primit ca semnal de intrare pe linia de date „ps2\_data”. De asemenea, semnalele „clk” și „ps2\_clk” sunt tot de intrare (fiecare de câte un bit), primul reprezentând semnalul de ceas al sistemului (care are o frecvență de 100 MHz), iar al doilea – semnalul de ceas al tastaturii PS/2. După ce a fost receptat întregul octet, valoarea sa este transmisă mai departe prin semnalul de ieșire „ps2\_code” de 8 biți, care reprezintă codul de scanare a tastei apăsate anterior.

* + 1. **Modulul de debounce pentru semnalele de la tastatură**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

Chart, box and whisker chart

Description automatically generated

**Figura 4.4.** Cutia neagră a modului de debounce pentru semnalele de la tastatură

Rolul modulului de debounce pentru semnalele transmise de la tastatura PS/2 este acela de a aduce semnalele într-o stare stabilă, iar valorile lor de ieșire să rămână neschimbate. Astfel, se evită apariția unor ambiguități temporare care ar putea avea efecte conflictuale asupra funcționării corecte a transmisiei. Semnalele „clk” și „button” sunt ambele semnale de intrare pe 1 bit, unde primul reprezintă semnalul de ceas al sistemului (care are o frecvență de 100 MHz), iar al doilea – semnalul pentru care se aplică funcționalitatea de debounce. Semnalul „result” este un semnal de ieșire pe 1 bit, care reprezintă valoarea semnalului „button”, după aplicarea asupra acestuia a funcționalității de debounce.

* + 1. **Modulul translatorului din cod de scanare în cod ascii**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

Chart

Description automatically generated

**Figura 4.5.** Cutia neagră a modului translatorului din cod de scanare în cod ascii

Rolul modulului translatorului din cod de scanare în cod ascii este destul de evident, datorită numelui său foarte sugestiv. Cu alte cuvinte, acesta are ca scop decodificarea din codurile standard de scanare ale tastelor unei tastaturi de tip PS/2, în codul ASCII corespunzător caracterului tastei apăsate. „ps2\_code” este semnalul de intrare pe 8 biți care primește codul de scanare, iar „ascii\_code” este semnalul de ieșire, de asemenea pe 8 biți, care transmite codul ascii rezultat în urma decodificării.

* + 1. **Modulul afișorului de 7 segmente**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

A picture containing diagram

Description automatically generated

**Figura 4.6.** Cutia neagră a modului afișorului de 7 segmente

Rolul modulului afișorului de 7 segmente este cel de a configura segmentele celei mai din dreapta cifre ale afișorului, în așa fel încât să afișeze cifra corespunzătoare codului de scanare transmis ca semnal de intrare pe linia de date de 8 biți „data” . Semnalele „an” și „seg” sunt ambele semnale de ieșire pe 1 bit, unde primul reprezintă configurația anozilor celor 4 cifre de pe afișor (această configurație are valoarea setată de către constanta „1110”, însemnând că doar cea mai din dreapta cifră este activă), iar al doilea – configurația celor 8 segmente pentru cifra activată de către configurația anozilor (7 segmente pentru afișarea caracterului și un segment pentru led-ul „dp”, care reprezintă led-ul care are forma unui punct; acesta fiind tot timpul inactiv – bit-ul cel mai puțin semnificativ).

### **Modulul transmițătorului serial**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

Chart

Description automatically generated

**Figura 4.7.** Cutia neagră a modulului transmițătorului serial

Rolul transmițătorului serial este acela de transmite codul ascii al unei taste către PC. Acest cod este primit ca semnal de intrare pe linia de date de 8 biți „TxData”. De asemenea, semnalele „Clk”, „Rst” și „Start” sunt tot de intrare (dar fiecare de câte un bit), primul reprezentând semnalul de ceas al sistemului (care are o frecvență de 100 MHz), al doilea – semnalul sincron de resetare al sistemului, iar al treilea – semnalul care în starea sa activă permite începerea transmisiei octetului, prin tranziția automatului de stare din starea „ready” de așteptare, în starea „load” de încărcare a biților.

* + 1. **Modulul de debounce pentru semnalele de la FPGA**

În figura de mai jos, se poate observa cutia neagră a acestui modul:

Chart

Description automatically generated

**Figura 4.8.** Cutia neagră a modului de debounce pentru semnalele de la FPGA

Rolul modulului de debounce pentru semnalele transmise de la FPGA este același cu cel al modulului de debounce pentru semnalele de la tastatură. Semnalele „Clk”, „Rst” și „Din” sunt toate semnale de intrare pe 1 bit, unde primul reprezintă semnalul de ceas al sistemului (care are o frecvență de 100 MHz), al doilea – semnalul sincron de resetare al sistemului, iar al treilea – semnalul pentru care se aplică funcționalitatea de debounce. Semnalul „Qout” este un semnal de ieșire pe 1 bit, care reprezintă valoarea semnalului „Din”, după aplicarea asupra acestuia a funcționalității de debounce.

## **Detalii de implementare**

Modulele menționate la subcapitolul anterior sunt interconectate, astfel încât să se obțină rezultatele dorite. Pentru a descrie mai amănunțit comunicarea dintre aceste module, se vor descrie pe rând modulele care participă în efectuarea recepției seriale, respectiv transmisiei seriale.

Receptorul serial „ps2\_keyboard” preia codul de scanare transmis de către tastatura de tip PS/2 la apăsarea unei taste, efectuându-se sincronizările (bit cu bit) necesare prin intermediul modulului „debounce”. Acel semnal de date sincronizat, va fi trimis mai departe înspre modulul „displ7seg”, pentru afișarea caracterului corespunzător pe afișorul plăcii de dezvoltare (cifra activă fiind cea mai din dreapta).

Comunicare dintre modulele „ps2\_keyboard”, „debounce” și „displ7seg” este ilustrată mai jos:

Chart

Description automatically generated

**Figura 4.9.** Comunicația dintre modulele „ps2\_keyboard”, „debounce” și „displ7seg”

Transmițătorul serial preia de asemenea acest cod „ps2\_code” sincronizat, îl translatează în cod ascii prin intermediul modulului „scan2ascii”, după care așteaptă activarea semnalului de „start” (semnal atașat butonului BTNU – conectorul J8; asupra acestui semnal se va aplica funcționalitatea de debounce prin intermediul modulului „debounce\_clasic”) pentru a-l transmite înspre PC.

Comunicarea dintre modulele „ps2\_keyboard”, „debounce”, „scan2ascii” și „uart\_tx” și „debounce\_clasic” este ilustrată mai jos:

Chart

Description automatically generated

**Figura 4.10.** Comunicația dintre modulele „ps2\_keyboard”, „debounce”, „scan2ascii”, „decounce\_clasic” și „uart\_tx”

În coninuare, se va detalia implementarea aplicației de efectuare a unei conexiuni seriale și de emulare a unui terminal, pentru afișarea caracterelor transmise la apăsarea tastelor de la tastatura PS/2. Partea de BACK-END: pentru realizarea conexiunii seriale, a fost importată biblioteca „serial”. Acesta bibliotecă pune la dispoziție o multitudine de funcții ajutătoare, unele dintre acestea fiind folosite pentru proiectul acesta (de exemplu, funcția de conectare sau de citire a datelor prin conexiunea serială realizată).

Partea de Front-END: interfață grafică cu care interacționează utilizatorul a fost creată folosind expansiva bibliotecă „Tkinter”. Ferestrele aplicației sunt widget-uri Tkinter, iar la evenimentele de apăsare a butoanelor existente pe aceste ferestre sunt atașate funcții (de exemplu, pentru a realiza conexiunea serială, se apasă butonul „CONNECT”).

Detaliu de implementare: afișarea în terminalul aplicației este condiționată de selectarea butonului „RECEIVE DATA”, existent pe fereastra terminalului. Dacă acest buton se selectează, fără a fi transmise date de la tastatura PS/2, atunci în terminal nu se va afișa nimic.

Funcționarea aplicației software este detaliată în manualul de utilizare (sunt incluse și imagini). Codul aplicației software, cât și cele ale modulelor, sunt incluse în capitolul „Anexa”.

## **Manual de utilizare**

Pentru afișarea caracterelor corespunzătoare tastelor apăsate (la tastatura conectata la circuitul FPGA) pe ecranul PC-ului, s-a implementat o aplicație software intermediară. Rolurile acesteia sunt cel de creare a unei conexiuni seriale și cel de emulare a unui terminal. Aplicația a fost implementată în limbajul Python 2.7, folosind mediul de dezvoltare PyCharm Community Edition 2020.3. Pentru realizarea montajului și configurarea circuitului FPGA, se parcurg următorii pași:

1. Se conectează tastatura la placa de dezvoltare Basys 3 prin portul USB HID al circuitului FPGA (J2);
2. Se conectează capătul corespunzător al unui cablu Micro USB Type A la portul UART/JTAG USB (J4) al circuitului FPGA, iar celălalt capăt al cablului se conectează la PC prin port-ul USB „COM4” al PC-ului;
3. Se pornește alimentarea plăcii de dezvoltare de la Power Switch (SW16);
4. Se deschide mediul de dezvoltare Vivado 2019.2 Design Suite de la Xilinx®.
5. Se deschide proiectul „big” creat în mediul de dezvoltare;
6. În meniul „Program and Debug”, se selectează opțiunea „Generate Bitstream” pentru generarea șirului de biți;
7. După finalizarea generării, tot din meniul „Program and Debug”, se selectează opțiunea „Open Hardware Manager” pentru a deschide ferestra „Hardware”;
8. În fereastra „Hardware”, conexiunea la circuitul FPGA se realizează prin selectarea butonului „Auto Connect” ;
9. Se configurează circuitul FPGA, prin programarea pe placa de dezvoltare a șirului de biți creat anterior cu numele „modulul\_principal.bit”. Pentru aceasta, se selectează opțiunea „Program Device” din meniul de tip Drop-Down al opțiunii „Open Hardware Manager”, unde se alege placa de dezvoltare conectată (cu numele „xc7a35t\_0”) din lista afișată pe ecran;
10. Se poate începe testarea și observare funcționării cu succes a operațiilor implementate.

Pentru afișarea caracterelor pe cea mai din dreapta cifră a afișorului de 7 segmente al circuitului FPGA, se va apăsa tasta corespunzătoare caracterului care se dorește să se afișeze. Se poate apăsa fiecare tastă în parte (A-Z, 0-9) pentru a se testa și observa afișarea corectă a caracterelor corespondente.

Pentru afișarea caracterelor în fereastra terminalului, se parcurg următorii pași:

1. Se deschide mediul de dezvoltare PyCharm Community Edition 2020.3;
2. Se deschide proiectul „serial-port-connection-app” creat în mediul de dezvoltare;
3. Se expandează proiectul pentru a putea vedea fișierul „gui.py”.
4. Se selectează fișierul „gui.py” prin click-dreapta;
5. Se selectează opțiunea „Run” din meniul afișat pe ecran pentru a rula aplicația;
6. Se apasă butonul „CONNECT” de pe fereastra principală a aplicației. Conexiunea se realizează la portul serial „COM4” (se poate folosi și alt port, dar acesta a fost ales din motive de organizare a spațiului de lucru pentru montaj[[1]](#footnote-1)).

Graphical user interface, text, application

Description automatically generated

**Figura 4.12.** Ferestra principală a aplicației software implementate

1. Indiferent dacă s-a realizat sau nu conexiunea, va apărea pe ecran o fereastră cu un mesaj sugestiv.

Graphical user interface

Description automatically generated

**Figura 4.13.** Mesajele sugestive posibile

1. După ce conexiunea s-a realizat cu succes, va apărea pe ecran fereastra terminalului unde vor fi afișate caracterele transmise;
2. Pentru a afișa un caracter în terminal, în primul rând, se apasă tasta corespunzătoare caracterului respectiv (la tastatura conectata la circuitul FPGA; caracterul va fi afișat pe cea mai din dreapta cifră a afișorului de 7 segmente al plăcii de dezvoltare);
3. În al doilea rând, se apasă butonul BTNU (T18; butonul care activează semnalul Start și mută automatul de stare al transmițătorului serial UART în starea „load”), pentru a începe transmisia octetului pe linia de comunicare TX;
4. În ultimul rând, se apasă butonul „RECEIVE DATA” pus la dispoziție în fereastra terminalului aplicației, acțiune ce rezultă în afișarea caracterului respectiv în terminal. Acest proces se repetă pentru fiecare tastă în parte, pentru a se testa și observa funcționarea corectă a transmițătorului serial UART TX.

Graphical user interface, application, Word

Description automatically generated

**Figura 4.14.** Fereastra terminalului

# **Rezultate experimentale**

Pentru a vizualiza un videoclip care surpinde testarea tuturor funcționalităților acestui proiect, se poate accesa următorul link: <https://github.com/iuliastoian/Structura-sistemelor-de-calcul> .

Placa de dezvoltare utilizată pentru realizarea proiectului este circuitul FPGA Basys 3, de la Digilent. Tastatura de tip PS/2 utilizată are codul MY8518, de la Myria. Sistemul de operare al PC-ului utilizat este Windows 10 Pro.

Mai jos se poate observa tabelul instrumentelor de proiectare utilizate pentru realizarea proiectului:

|  |  |  |
| --- | --- | --- |
| **Instrument de proiectare** | **Versiune** | **Unde a fost utilizat?** |
| Xilinx: Vivado Design Suite | 2019.2 | Programarea hardware-ului. |
| PyCharm Community Edition | 2020.3 | Programarea aplicației software de efectuare a unei conexiuni seriale și de emulare a unui terminal. Limbajul de programare utilizat – Python 3.9.1. |

**Tabel 5.1.** Instrumente de proiectare

# **Concluzii**

Realizând acest proiect, am învățat cum funcționează transmisia și recepția serială de date prin intermediul liniilor de comunicare RX și TX, cum să implementez aceste funcționalități în limbaj VHDL, cum să realizez o aplicație software care realizează o conexiune serială și comunică, prin intermediul acesteia, cu circuitul FPGA care este conectat la PC.

# **Bibliografie**

1. <https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-4-ddr/nexys4ddr_rm.pdf>
2. <https://ro.wikipedia.org/wiki/Transmisie_de_date_serial%C4%83>
3. <https://en.wikipedia.org/wiki/Seven-segment_display>
4. Pong, P. Chu. „FPGA Prototyping by Systemverilog Examples: Xilinx Microblaze MCS SoC Edition”. 2018, Section 18.5. Web.
5. <https://www.digikey.com/eewiki/pages/viewpage.action?pageId=28278929>
6. <https://users.utcluj.ro/~baruch/ssc/labor/Testare-Depanare.pdf>

# **Anexe**

## debounce\_clasic.vhd

Acest debounce a fost preluat din resursele oferite la orele de laborator.

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** debounce\_clasic **is**

**Port** **(** Clk **:** **in** STD\_LOGIC**;**

Rst **:** **in** STD\_LOGIC**;**

Din **:** **in** STD\_LOGIC**;**

Qout **:** **out** STD\_LOGIC**);**

**end** debounce\_clasic**;**

**architecture** Behavioral **of** debounce\_clasic **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;

## uart\_tx.vhd

Acest transmițător serial a fost realizat practic la orele de laborator în săptămâna a 11-a, ca parte introductivă a capitolului 7, adică „Proiectarea unei interfețe de intrare/ieșire”.

-- modulul transmitatorului

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**std\_logic\_unsigned**;**

**entity** uart\_tx **is**

**generic** **(**BitRate**:** integer **:=** 9\_600**);** -- rata de biti;

**port** **(**Clk**:** **in** std\_logic**;** -- semnalele de intrare

Rst**:** **in** std\_logic**;**

TxData**:** **in** std\_logic\_vector**(**7 **downto** 0**);** -- octetul care trebuie transmis

Start**:** **in** std\_logic**;** -- comanda de incepere a transmisiei

-- semnalele de iesire

Tx**:** **out** std\_logic**;** -- linia pe care sunt transmise datele in mod serial

TxRdy**:** **out** std\_logic -- semnalul de stare

**);**

**end** uart\_tx**;**

**architecture** Behavioral **of** uart\_tx **is**

-- starile automatului

**type** STATE\_TYPE **is** **(**ready**,** load**,** send**,** waitbit**,** shift**);**

-- starea curenta

**signal** St**:** STATE\_TYPE **:=** ready**;**

-- semnal pentru contorizarea bitilor transmisi pe linia seriala

**signal** CntBit**:** integer **:=** 0**;**

-- semnal pentru contorizarea ciclurilor de ceas

**signal** CntRate**:** integer **:=** 0**;**

-- frecventa semnalului de ceas 100 MHz

**constant** CLK\_FREQ**:** integer **:=** 100\_000\_000**;**

-- numarul de cicluri de ceas corespunzatori duratei unui bit

**constant** T\_BIT**:** integer **:=** CLK\_FREQ **/** BitRate**;**

-- semnalele de comanda

**signal** LdData**,** ShData**,** TxEn**:** std\_logic**;**

-- semnalul pentru registrul de deplasare

**signal** TSR**:** std\_logic\_vector**(**9 **downto** 0**)** **:=** **(others** **=>** '0'**);**

-- atribut pentru a evita modificarea numelor semnalelor dupa etapa de sinteza

**attribute** keep**:** string**;**

**attribute** keep **of** St**,** CntRate**,** CntBit**,** TSR**:** **signal** **is** "TRUE"**;**

**begin**

-- proces pentru registrul de deplasare TSR

**process** **(**Clk**,** Rst**)**

**begin**

**if** **rising\_edge(**Clk**)** **then**

**if** Rst **=** '1' **then**

TSR **<=** **(others** **=>** '0'**);**

**elsif** LdData **=** '1' **then**

TSR **<=** '1' **&** TxData **&** '0'**;** -- 0 bitul de START si 1 bitul de STOP

**elsif** ShData **=** '1' **then**

TSR **<=** '0' **&** TSR**(**9 **downto** 1**);** -- deplasare la dreapta cu 1 bit

**end** **if;**

**end** **if;**

**end** **process;**

-- Automat de stare pentru unitatea de control a transmitatorului serial

proc\_control**:** **process** **(**Clk**)**

**begin**

**if** **RISING\_EDGE** **(**Clk**)** **then**

**if** **(**Rst **=** '1'**)** **then**

St **<=** ready**;**

**else**

**case** St **is**

-- se indica terminarea transmiterii unui octet

**when** ready **=>**

CntRate **<=** 0**;**

CntBit **<=** 0**;**

**if** **(**Start **=** '1'**)** **then**

St **<=** load**;**

**end** **if;**

-- se incarca registrul de deplasare cu octetul care trebuie transmis

**when** load **=>**

St **<=** send**;**

-- se transmite 1 bit

**when** send **=>**

St **<=** waitbit**;**

-- se asteapta trecerea intervalului de timp egal cu durata unui bit

**when** waitbit **=>**

CntRate **<=** CntRate **+** 1**;**

**if** **(**CntRate **=** T\_BIT**)** **then**

CntRate **<=** 0**;**

St **<=** shift**;**

**end** **if;**

-- registrul de deplasare se deplaseaza la dreapta cu 1 bit

**when** shift **=>**

CntBit **<=** CntBit **+** 1**;**

**if** **(**CntBit **=** 10**)** **then**

St **<=** ready**;**

**else**

St **<=** send**;**

**end** **if;**

**when** **others** **=>**

St **<=** ready**;**

**end** **case;**

**end** **if;**

**end** **if;**

**end** **process** proc\_control**;**

-- Setarea semnalelor de comanda

LdData **<=** '1' **when** St **=** load **else** '0'**;**

ShData **<=** '1' **when** St **=** shift **else** '0'**;**

TxEn **<=** '0' **when** St **=** ready **or** St **=** load **else** '1'**;**

-- Setarea semnalelor de iesire

Tx **<=** TSR**(**0**)** **when** TxEn **=** '1' **else** '1'**;**

TxRdy **<=** '1' **when** St **=** ready **else** '0'**;**

**end** Behavioral**;**

## displ7seg.vhd

Acest modul primește pe linia de date „data” un octet, reprezentând codul de scanare a tastei apăsate. În funcție de litera pe care o reprezintă acest cod de scanare, se vor configura cele 7 segmente după modelul prezentat în [figura 4.1](#_Soluție_aleasă). Până în momentul de față, proiectul a fost realizat astfel încât să se folosească doar cea mai din dreapta cifră, motiv pentru care configurația este condusă de către o constantă.



**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** displ7seg **is**

**port** **(**

-- semnalele de intrare

data **:** **in** std\_logic\_vector**(**7 **downto** 0**);** -- linia de date pentru o cifra (cifra cea mai din dreapta)

-- semnalele de iesire

an **:** **out** std\_logic\_vector **(**3 **downto** 0**);** -- selectia anodului activ (in cazul acesta este activ anodul 4)

seg **:** **out** std\_logic\_vector **(**7 **downto** 0**));** -- selectia catozilor (segmentelor) cifrei active

**end** displ7seg**;**

**architecture** behavioral **of** displ7seg **is**

**begin**

**process(**data**)**

**begin**

**case** data **is**

**when** x"45" **=>** seg **<=** "00000011"**;** -- "0"

**when** x"16" **=>** seg **<=** "10011111"**;** -- "1"

**when** x"1e" **=>** seg **<=** "00100101"**;** -- "2"

**when** x"26" **=>** seg **<=** "00001101"**;** -- "3"

**when** x"25" **=>** seg **<=** "10011001"**;** -- "4"

**when** x"2e" **=>** seg **<=** "01001001"**;** -- "5"

**when** x"36" **=>** seg **<=** "01000001"**;** -- "6"

**when** x"3d" **=>** seg **<=** "00011111"**;** -- "7"

**when** x"3e" **=>** seg **<=** "00000001"**;** -- "8"

**when** x"46" **=>** seg **<=** "00001001"**;** -- "9"

**when** x"1C" **=>** seg **<=** "00010001"**;** -- a

**when** x"32" **=>** seg **<=** "11000001"**;** -- b

**when** x"21" **=>** seg **<=** "01100011"**;** -- c

**when** x"23" **=>** seg **<=** "10000101"**;** -- d

**when** x"24" **=>** seg **<=** "01100001"**;** -- e

**when** x"2b" **=>** seg **<=** "01110001"**;** -- f

**when** x"34" **=>** seg **<=** "01000011"**;** -- g

**when** x"33" **=>** seg **<=** "11010001"**;** -- h

**when** x"43" **=>** seg **<=** "11011111"**;** -- i

**when** x"3b" **=>** seg **<=** "10000111"**;** -- j

**when** x"42" **=>** seg **<=** "01010001"**;** -- k

**when** x"3a" **=>** seg **<=** "01010101"**;** -- l

**when** x"4b" **=>** seg **<=** "11100011"**;** -- m

**when** x"31" **=>** seg **<=** "11010101"**;** -- n

**when** x"44" **=>** seg **<=** "11000101"**;** -- o

**when** x"4d" **=>** seg **<=** "00110001"**;** -- p

**when** x"15" **=>** seg **<=** "00011001"**;** -- q

**when** x"2d" **=>** seg **<=** "11110101"**;** -- r

**when** x"1b" **=>** seg **<=** "01001001"**;** -- s

**when** x"2c" **=>** seg **<=** "11100001"**;** -- t

**when** x"3c" **=>** seg **<=** "10000011"**;** -- u

**when** x"2a" **=>** seg **<=** "11000111"**;** -- v

**when** x"1d" **=>** seg **<=** "10101001"**;** -- w

**when** x"22" **=>** seg **<=** "10010001"**;** -- x

**when** x"35" **=>** seg **<=** "10001001"**;** -- y

**when** x"1a" **=>** seg **<=** "00100101"**;** -- z

**when** **others** **=>** seg **<=** "11111111"**;** -- toate segmentele stinse

**end** **case;**

**end** **process;**

an **<=** "1110"**;** -- se activeaza cifra cea mai din dreapta

**end** behavioral**;**

## scan2ascii.vhd

În acest modul se realizează translatarea din cod scanare în cod ascii.

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**entity** scan2ascii **is**

**port(**

ps2\_code **:** **in** std\_logic\_vector**(**7 **downto** 0**);**

ascii\_code **:** **out** std\_logic\_vector**(**7 **downto** 0**));** -- ascii value

**end** scan2ascii**;**

**architecture** behavioral **of** scan2ascii **is**

**begin**

**process(**ps2\_code**)**

**begin**

**case** ps2\_code **is**

**when** x"1c" **=>** ascii\_code **<=** x"41"**;** --a

**when** x"32" **=>** ascii\_code **<=** x"42"**;** --b

**when** x"21" **=>** ascii\_code **<=** x"43"**;** --c

**when** x"23" **=>** ascii\_code **<=** x"44"**;** --d

**when** x"24" **=>** ascii\_code **<=** x"45"**;** --e

**when** x"2b" **=>** ascii\_code **<=** x"46"**;** --f

**when** x"34" **=>** ascii\_code **<=** x"47"**;** -- g

**when** x"33" **=>** ascii\_code **<=** x"48"**;** -- h

**when** x"43" **=>** ascii\_code **<=** x"49"**;** -- i

**when** x"3b" **=>** ascii\_code **<=** x"4a"**;** -- j

**when** x"42" **=>** ascii\_code **<=** x"4b"**;** -- k

**when** x"4b" **=>** ascii\_code **<=** x"4c"**;** -- l

**when** x"3a" **=>** ascii\_code **<=** x"4d"**;** -- m

**when** x"31" **=>** ascii\_code **<=** x"4e"**;** -- n

**when** x"44" **=>** ascii\_code **<=** x"4f"**;** -- o

**when** x"4d" **=>** ascii\_code **<=** x"50"**;** -- p

**when** x"15" **=>** ascii\_code **<=** x"51"**;** -- q

**when** x"2d" **=>** ascii\_code **<=** x"52"**;** -- r

**when** x"1b" **=>** ascii\_code **<=** x"53"**;** -- s

**when** x"2c" **=>** ascii\_code **<=** x"54"**;** -- t

**when** x"3c" **=>** ascii\_code **<=** x"55"**;** -- u

**when** x"2a" **=>** ascii\_code **<=** x"56"**;** -- v

**when** x"1d" **=>** ascii\_code **<=** x"57"**;** -- w

**when** x"22" **=>** ascii\_code **<=** x"58"**;** -- x

**when** x"35" **=>** ascii\_code **<=** x"59"**;** -- y

**when** x"1a" **=>** ascii\_code **<=** x"5a"**;** -- z

**when** x"45" **=>** ascii\_code **<=** x"30"**;** --0

**when** x"16" **=>** ascii\_code **<=** x"31"**;** --1

**when** x"1e" **=>** ascii\_code **<=** x"32"**;** --2

**when** x"26" **=>** ascii\_code **<=** x"33"**;** --3

**when** x"25" **=>** ascii\_code **<=** x"34"**;** --4

**when** x"2e" **=>** ascii\_code **<=** x"35"**;** --5

**when** x"36" **=>** ascii\_code **<=** x"36"**;** --6

**when** x"3d" **=>** ascii\_code **<=** x"37"**;** --7

**when** x"3e" **=>** ascii\_code **<=** x"38"**;** --8

**when** x"46" **=>** ascii\_code **<=** x"39"**;** --9

**when** **others** **=>** ascii\_code **<=** x"20"**;** -- blank space

**end** **case;**

**end** **process;**

**end** behavioral**;**

## ps2\_keyboard.vhd

Acest modul se ocupă de primire codurilor de scanare primite prin linia de date RX [5].

--------------------------------------------------------------------------------

--

-- FileName: ps2\_keyboard.vhd

-- Dependencies: debounce.vhd

-- Design Software: Quartus II 32-bit Version 12.1 Build 177 SJ Full Version

--

-- HDL CODE IS PROVIDED "AS IS." DIGI-KEY EXPRESSLY DISCLAIMS ANY

-- WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT

-- LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A

-- PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY

-- BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL

-- DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF

-- PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS

-- BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),

-- ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

--

-- Version History

-- Version 1.0 11/25/2013 Scott Larson

-- Initial Public Release

--

--------------------------------------------------------------------------------

**LIBRARY** ieee**;**

**USE** ieee**.**std\_logic\_1164**.all;**

**ENTITY** ps2\_keyboard **IS**

**GENERIC(**

clk\_freq **:** INTEGER **:=** 100\_000\_000**;** -- system clock frequency in Hz

debounce\_counter\_size **:** INTEGER **:=** 9**);** -- set such that (2^size)/clk\_freq = 5us (size = 8 for 50MHz)

**PORT(**

clk **:** **IN** STD\_LOGIC**;** -- system clock

ps2\_clk **:** **IN** STD\_LOGIC**;** -- clock signal from PS/2 keyboard

ps2\_data **:** **IN** STD\_LOGIC**;** -- data signal from PS/2 keyboard

ps2\_code\_new **:** **OUT** STD\_LOGIC**;** -- flag that new PS/2 code is available on ps2\_code bus

ps2\_code **:** **OUT** STD\_LOGIC\_VECTOR**(**7 **DOWNTO** 0**));** -- code received from PS/2

**END** ps2\_keyboard**;**

**ARCHITECTURE** logic **OF** ps2\_keyboard **IS**

**SIGNAL** sync\_ffs **:** STD\_LOGIC\_VECTOR**(**1 **DOWNTO** 0**);** -- synchronizer flip-flops for PS/2 signals

**SIGNAL** ps2\_clk\_int **:** STD\_LOGIC**;** -- debounced clock signal from PS/2 keyboard

**SIGNAL** ps2\_data\_int **:** STD\_LOGIC**;** -- debounced data signal from PS/2 keyboard

**SIGNAL** ps2\_word **:** STD\_LOGIC\_VECTOR**(**10 **DOWNTO** 0**);** -- stores the ps2 data word

**SIGNAL** error **:** STD\_LOGIC**;** -- validate parity, start, and stop bits

**SIGNAL** count\_idle **:** INTEGER **RANGE** 0 **TO** clk\_freq**/**18\_000**;** -- counter to determine PS/2 is idle

**BEGIN**

-- synchronizer flip-flops

**PROCESS(**clk**)**

**BEGIN**

**IF(**clk'**EVENT** **AND** clk **=** '1'**)** **THEN** -- rising edge of system clock

sync\_ffs**(**0**)** **<=** ps2\_clk**;** -- synchronize PS/2 clock signal

sync\_ffs**(**1**)** **<=** ps2\_data**;** -- synchronize PS/2 data signal

**END** **IF;**

**END** **PROCESS;**

-- debounce PS2 input signals

debounce\_ps2\_clk**:** **entity** work**.**debounce

**GENERIC** **MAP(**counter\_size **=>** debounce\_counter\_size**)**

**PORT** **MAP(**clk **=>** clk**,** button **=>** sync\_ffs**(**0**),** result **=>** ps2\_clk\_int**);**

debounce\_ps2\_data**:** **entity** work**.**debounce

**GENERIC** **MAP(**counter\_size **=>** debounce\_counter\_size**)**

**PORT** **MAP(**clk **=>** clk**,** button **=>** sync\_ffs**(**1**),** result **=>** ps2\_data\_int**);**

-- input PS2 data

**PROCESS(**ps2\_clk\_int**)**

**BEGIN**

**IF(**ps2\_clk\_int'**EVENT** **AND** ps2\_clk\_int **=** '0'**)** **THEN** -- falling edge of PS2 clock

ps2\_word **<=** ps2\_data\_int **&** ps2\_word**(**10 **DOWNTO** 1**);** -- shift in PS2 data bit

**END** **IF;**

**END** **PROCESS;**

-- verify that parity, start, and stop bits are all correct

error **<=** **NOT** **(NOT** ps2\_word**(**0**)** **AND** ps2\_word**(**10**)** **AND** **(**ps2\_word**(**9**)** **XOR** ps2\_word**(**8**)** **XOR**

ps2\_word**(**7**)** **XOR** ps2\_word**(**6**)** **XOR** ps2\_word**(**5**)** **XOR** ps2\_word**(**4**)** **XOR** ps2\_word**(**3**)** **XOR**

ps2\_word**(**2**)** **XOR** ps2\_word**(**1**)));**

-- determine if PS2 port is idle (i.e. last transaction is finished) and output result

**PROCESS(**clk**)**

**BEGIN**

**IF(**clk'**EVENT** **AND** clk **=** '1'**)** **THEN** -- rising edge of system clock

**IF(**ps2\_clk\_int **=** '0'**)** **THEN** -- low PS2 clock, PS/2 is active

count\_idle **<=** 0**;** -- reset idle counter

**ELSIF(**count\_idle **/=** clk\_freq**/**18\_000**)** **THEN** -- PS2 clock has been high less than a half clock period (<55us)

count\_idle **<=** count\_idle **+** 1**;** -- continue counting

**END** **IF;**

**IF(**count\_idle **=** clk\_freq**/**18\_000 **AND** error **=** '0'**)** **THEN** -- idle threshold reached and no errors detected

ps2\_code\_new **<=** '1'**;** -- set flag that new PS/2 code is available

ps2\_code **<=** ps2\_word**(**8 **DOWNTO** 1**);** -- output new PS/2 code

**ELSE** -- PS/2 port active or error detected

ps2\_code\_new **<=** '0'**;** -- set flag that PS/2 transaction is in progress

**END** **IF;**

**END** **IF;**

**END** **PROCESS;**

**END** logic**;**

## debouncer.vhd

Acest debouncer este preluat din aceeași sursă cu ps2\_keyboard.vhd [5].

--------------------------------------------------------------------------------

--

-- FileName: debounce.vhd

-- Dependencies: none

-- Design Software: Quartus II 32-bit Version 11.1 Build 173 SJ Full Version

--

-- HDL CODE IS PROVIDED "AS IS." DIGI-KEY EXPRESSLY DISCLAIMS ANY

-- WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT

-- LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A

-- PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY

-- BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL

-- DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF

-- PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS

-- BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),

-- ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

--

-- Version History

-- Version 1.0 3/26/2012 Scott Larson

-- Initial Public Release

--

--------------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** debounce **is**

**generic** **(**

counter\_size **:** integer **:=** 20**);** -- dimensiunea numaratorului (20 de biti dau un timp de 10.5ms

-- in care intrarea este stabila cu un clock de 100mhz)

**port(**

-- semnalele de intrare

clk **:** **in** std\_logic**;** -- intrare clock 100mhz

button **:** **in** std\_logic**;** -- intrare semnal pentru care trebuie facut debounce

-- semnalul de iesire

result **:** **out** std\_logic**);** -- semnalul pentru care s-a facut debounce

**end** debounce**;**

**architecture** logic **of** debounce **is**

**signal** flipflops **:** std\_logic\_vector**(**1 **downto** 0**);** -- bistabil

**signal** counter\_set **:** std\_logic**;** -- sincronizeaza reset la zero

**signal** counter\_out **:** std\_logic\_vector**(**counter\_size **downto** 0**)** **:=** **(others** **=>** '0'**);** -- iesire numarator

counter\_set **<=** flipflops**(**0**)** **xor** flipflops**(**1**);** -- determina cand sa porneasca/reseteze numaratorul

**process(**clk**)**

**begin**

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

flipflops**(**0**)** **<=** button**;**

flipflops**(**1**)** **<=** flipflops**(**0**);**

**if(**counter\_set **=** '1'**)** **then** -- reseteaza numaratorul pentru ca intrarea se schimba

counter\_out **<=** **(others** **=>** '0'**);**

**elsif(**counter\_out**(**counter\_size**)** **=** '0'**)** **then** -- timpul in care intrare este stabila nu s-a scurs inca

counter\_out **<=** counter\_out **+** 1**;**

**else** -- timpul in care intrarea este stabila s-a scurs

result **<=** flipflops**(**1**);**

**end** **if;**

**end** **if;**

**end** **process;**

**end** logic**;**

## modulul\_principal.vhd

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**entity** modulul\_principal **is**

**generic(**

clk\_freq **:** integer **:=** 100\_000\_000**;** -- clock de 100mhz al sistemului

ps2\_debounce\_counter\_size **:** integer **:=** 9**);** -- se seteaza astfel incat 2^size/clk\_freq = 5us (size = 9 pentru 100mhz)

**port(**

-- semnalele de intrare

clk **:** **in** std\_logic**;** -- intrare de clock a sistemului

start **:** **in** std\_logic**;**

rst **:** **in** std\_logic**;** -- intrare de resetare a sistemului

ps2\_clk **:** **in** std\_logic**;** -- semnal de clock de la tastatura ps2

ps2\_data **:** **in** std\_logic**;** -- semnal de date de la tastatura ps2

-- semnalele de iesire

an **:** **out** std\_logic\_vector **(**3 **downto** 0**);** -- anozii afisorului

seg **:** **out** std\_logic\_vector **(**7 **downto** 0**);** -- catozii afisorului

Tx **:** **out** std\_logic**;**

TxRdy **:** **out** std\_logic**);**

**end** modulul\_principal**;**

**architecture** behavioral **of** modulul\_principal **is**

**signal** ps2\_code\_new **:** std\_logic**;** -- flag care indica transmisia unui nou cod din componenta ps2\_keyboard

**signal** ps2\_code **:** std\_logic\_vector**(**7 **downto** 0**);** -- codul transmis de catre componenta ps2\_keyboard

**signal** ascii\_code **:** std\_logic\_vector**(**7 **downto** 0**);** -- codul transmis de catre componenta ps2\_keyboard

**signal** startD **:** std\_logic**;**

**signal** resetD **:** std\_logic**;**

--signal data : std\_logic\_vector(7 downto 0); -- configuratia segmentelor in functie de codul primit de la tastatura

-- semnalele interne pentru citirea Tx si TxRdy

--signal tempTx: std\_logic := '1';

--signal tempTxRdy: std\_logic := '0';

**begin**

-- se instantiaza logica interfetei ps2 cu tastatura

kb**:** **entity** work**.**ps2\_keyboard

**generic** **map(**clk\_freq **=>** clk\_freq**,** debounce\_counter\_size **=>** ps2\_debounce\_counter\_size**)**

**port** **map(**

clk **=>** clk**,**

ps2\_clk **=>** ps2\_clk**,**

ps2\_data **=>** ps2\_data**,**

ps2\_code\_new **=>** ps2\_code\_new**,**

ps2\_code **=>** ps2\_code**);**

s2a**:** **entity** work**.**scan2ascii

**port** **map(**

ps2\_code **=>** ps2\_code**,**

ascii\_code **=>** ascii\_code**);**

-- maparea pentru modulul transmitatorului serial

uart\_tx**:** **entity** WORK**.**uart\_tx

**generic** **map** **(**BitRate **=>** 9\_600**)**

**port** **map** **(**Clk **=>** clk**,**

Rst **=>** resetD**,**

TxData **=>** ascii\_code**,**

Start **=>** startD**,**

Tx **=>** Tx**,**

TxRdy **=>** TxRdy**);**

-- debounce pentru semnalul butonului de start

startBtn**:** **entity** work**.**debounce\_clasic

**PORT** **MAP(**Clk **=>** clk**,** Rst **=>** rst**,** Din **=>** start**,** Qout **=>** startD**);**

-- debounce pentru semnalul butonului de reset

resetBtn**:** **entity** work**.**debounce\_clasic

**PORT** **MAP(**Clk **=>** clk**,** Rst **=>** rst**,** Din **=>** rst**,** Qout **=>** resetD**);**

-- se instantiaza afisorul de sapte segmente

ssd**:** **entity** work**.**displ7seg

**port** **map** **(**

clk **=>** clk**,**

rst **=>** resetD**,**

data **=>** ps2\_code**,**

an **=>** an**,**

seg **=>** seg**);**

--data <= ps2\_code;

**end** behavioral**;**

## Fișierul de constrângeri

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

##7 segment display

set\_property PACKAGE\_PIN W7 [get\_ports {seg[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[7]}]

set\_property PACKAGE\_PIN W6 [get\_ports {seg[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[6]}]

set\_property PACKAGE\_PIN U8 [get\_ports {seg[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[5]}]

set\_property PACKAGE\_PIN V8 [get\_ports {seg[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[4]}]

set\_property PACKAGE\_PIN U5 [get\_ports {seg[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[3]}]

set\_property PACKAGE\_PIN V5 [get\_ports {seg[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[2]}]

set\_property PACKAGE\_PIN U7 [get\_ports {seg[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[1]}]

set\_property PACKAGE\_PIN V7 [get\_ports seg[0]]

set\_property IOSTANDARD LVCMOS33 [get\_ports seg[0]]

set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]

set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]

set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]

##Buttons

set\_property PACKAGE\_PIN T18 [get\_ports start]

set\_property IOSTANDARD LVCMOS33 [get\_ports start]

set\_property PACKAGE\_PIN U17 [get\_ports rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports rst]

##USB-RS232 Interface

set\_property PACKAGE\_PIN A18 [get\_ports Tx]

set\_property IOSTANDARD LVCMOS33 [get\_ports Tx]

##USB HID (PS/2)

set\_property PACKAGE\_PIN C17 [get\_ports ps2\_clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports ps2\_clk]

set\_property PULLUP true [get\_ports ps2\_clk]

set\_property PACKAGE\_PIN B17 [get\_ports ps2\_data]

set\_property IOSTANDARD LVCMOS33 [get\_ports ps2\_data]

set\_property PULLUP true [get\_ports ps2\_data]

## Codul aplicației software

**import** serial

**from** Tkinter **import** **\***

**import** tkMessageBox

**from** serial **import** PARITY\_ODD**,** EIGHTBITS**,** STOPBITS\_ONE

**global** start\_window

**global** fpga

**def** start\_window**():**

**global** start\_window

start\_window **=** Tk**()**

start\_window**.**title**(**"Serial Port Terminal Emulator by Stoian Iulia Tia"**)**

start\_window**.**geometry**(**"600x400"**)**

# connect button

Button**(**text**=**"CONNECT"**,** width**=**"30"**,** height**=**"2"**,** command**=**connect**).**pack**(**pady**=**185**)**

start\_window**.**mainloop**()**

**def** connect**():**

**global** fpga

**try:**

fpga **=** serial**.**Serial**(**port**=**"com4"**,** baudrate**=**9600**,** timeout**=**0**,** bytesize**=**EIGHTBITS**,** parity**=**PARITY\_ODD**,**

stopbits**=**STOPBITS\_ONE**)**

tkMessageBox**.**showinfo**(**"Success!"**,** "Serial connection at port COM4 established."**)**

**print(**"Serial connection at port COM4 established."**)**

**except:**

tkMessageBox**.**showinfo**(**"Error!"**,** "The connection could not be established."**)**

**print(**"The connection could not be established."**)**

**return**

terminal\_window **=** Toplevel**(**start\_window**)**

terminal\_window**.**title**(**"Terminal for Received Data through RX communication Line"**)**

terminal\_window**.**geometry**(**"600x400"**)**

**def** receive**():**

data **=** fpga**.**read**(**size**=**2**)**

output\_textbox**.**insert**(**END**,** data**[**0**:**1**]** **+** " "**)**

# receive button

Button**(**terminal\_window**,** text**=**"RECEIVE DATA"**,** width**=**"30"**,** height**=**"2"**,** command**=**receive**).**pack**()**

output\_textbox **=** Text**(**terminal\_window**,** width**=**600**,** height**=**370**,** wrap**=**WORD**)**

output\_textbox**.**pack**()**

start\_window**()**

1. Pentru a folosi alt port, este necesar ca numele acestuia să fie specificat în codul aplicației la linia cu numărul 26:

   **Figura 4.11.** Linia 26 din codul aplicației software [↑](#footnote-ref-1)