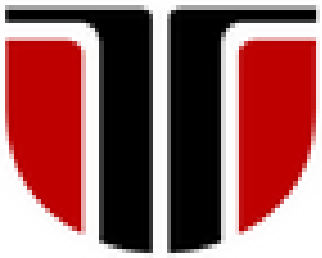
Universitatea Tehnica din Cluj-Napoca



Catedra de Calculatoare

Conectarea tastaturii cu FPGA si transmitere seriala de date

Studenti:
Irimus Ileana Maria
Voicu Diana-Georgiana

Indrumator de proiect:
Prof. Fati Daniela

Grupa 30233

Data: 7.11.2020

Cuprins

1. Rezumat 3
2. Introducere 4
2.1 Continutul sectiunilor 4
3. Fundamentare teoretică 5
3.1. Basys3 5
3.2 USB-UART Bridge (Port Serial) 7
3.3 USB HID Host 8
3.4. Tastatura USB 9
3.5 Seven Segment Display 10
4. Proiectare şi implementare 11
4.1. Afisarea pe display-ul cu 7 segmente a datelor transmise de tastatura USB HID 11
4.2. Afisarea in consola HTerm a datelor transmise de Basys3 prin portul serial UART 14
5. Rezultate experimentale 16
6. Concluzii 17
Bibliografie 18
Anexe 19

1. Rezumat

In cazul FPGA-urilor, pe langa intelegerea functionalitatii placii propriu zise, este foarte important sa putem observa si sa intelegem modul in care sunt realizate conexiunile cu alte dispozitive. Astfel, in acest proiect am urmarit realizarea conexiunii unei tastaturi USB HID cu o placa de dezvoltare prin portul USB HIID, precum si transmiterea mai departe a informatiei prin portul UART pe consola calculatorului. In realizarea acestuia, am utilizat FPGA-ul Basys 3, aplicatia Hterm, precum si tastatura necesara. Proiectul a fost dezvoltat in limbajul VHDL. Rezultatele obtinuite au fost cele asteptate: afisarea pe placa cu ajutorul 7 segments, a tastelor apasate, precum si transmiterea utilizand un buton de Start, a respectivei taste pe consola calculatorului(HTerm). Asadar, proiectul este complet functional, reusind sa expuna intr-un mod concret, coerent comunicarea dintre FPGA si alte dispozitive, subliniand utilitatea si functionalitatea porturilor USB HID, UART.

2. Introducere

Un sistem de calcul reprezinta un ansamblu de componente hardware (dispozitive fizice) si software (programe) ce permit solutionarea unor probleme a caror rezolvare se poate descrise sub forma unui algoritm.[1]

Interfata dintre utilizator si sistemul de calcul, respectiv dintre sistemul de calcul si alte sisteme fizice este asigurata de dispozitivele periferice de intrare(tastatura, mouse, scanner), iesire(imprimanta, monitor), intrare-iesire(modem, placa de retea) si de stocare(hard disk, floppy disk, banda magnetica). Printre componentele de baza ale unui sistem de calcul se numara si sistemul de intrare/iesire si structurile de interconectare a acestora. Acest subsistem este reprezentat de magistrale care transporta informatie(date, instructiuni, semnale de control), realizand legatura dintre sistemul de calcul si echipamentele periferice.[2] Magistralele la care ne vom referi in continuare sunt magistrale I/O externe, un ansamblu format din controller impreuna cu interfata dintre acesta si echipamentul periferic (PS/2 – tastatura, USB).[3]

Tema principala a proiectului consta in realizarea unei implementari hardware pe o placuta de dezvoltare FPGA a unui program care sa realizeze conectarea unei tastaturi USB HID cu o placuta FPGA si care prin intermediul unei componente ce poarta denumirea de UART sa afiseze in consola pe calculator tastele apasate.

Limbajul folosit pentru realizarea acestui proiect este VHDL in mediul de dezvoltare Xilinx Vivado 2020.1, iar ca placa de dezvoltare am folosit o placuta FPGA Basys3, care ne va pune la dispozitie portul USB necesar. Ne propunem ca pe langa dezvoltarea acestui program sa ne imbogatim cunostintele legate atat de programarea in limbajul VHDL, cat si de componente hardware precum UART, registre, memorii, programe aditionale ce vor juca rolul consolelor: HTerm.

In continuare, se vor descrie fundamentele teoretice legate de proiect precum metode, modele si tehnologii utilizate, urmate de detalii despre proiectare si implementare cu etapele parcuse pentru realizarea obiectivelor proiectului. Mai apoi, in sectiunea “Rezultate experimentale”, se va demonstra ca sistemul proiectat a fost implementat cu success si ca rezultatele obtinute sunt valide. In final se va prezenta un sumar al raportului proiectului, dar si posibile implementari ulterioare.

3. Fundamentare teoretica

3.1. Basys3

Un FPGA (Field Programmable Gate Array) este un circuit integrat digital configurabil, de catre utilizator, dupa ce a fost fabricat, fiind un tip de circuit logic programabil. FPGA-urile sunt alcatuite din blocuri logice configurabile (programabile) legate intre ele de o serie de conexiuni configurabile la randul lor. Basys3 reprezinta o placa de dezvoltare a circuitelor digitale, avand la baza o platforma bazata pe FPGA, dezvoltata de Xilinx.[4]

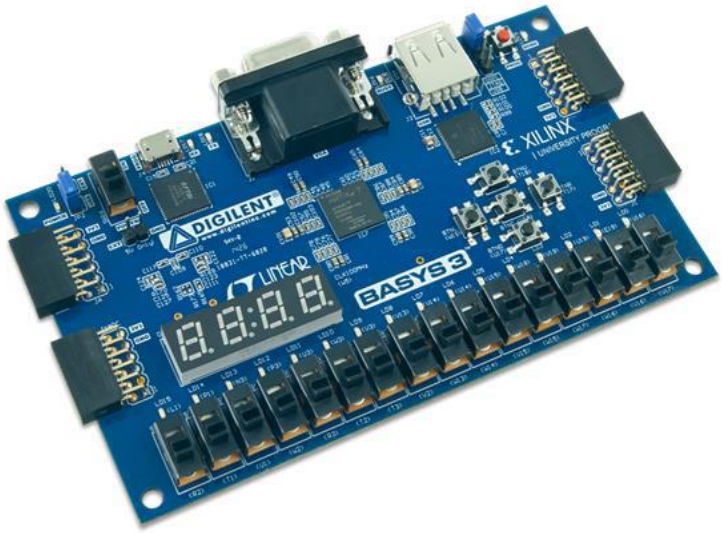


Figura 3.1: Placa de dezvoltare Basys3

Statistici:

- Device/IC: Xilinx Artix-7 FPGA (XC7A35T-1CPG236C)

Conectori:

- USB A
- USB micro-B
- 4 porturi Pmod 12-pin
- VGA

Programare:

- Proiectat exclusiv pentru Vivado Design Suite

Caracteristici:

- Dispune de Xilinx Artix-7 FPGA: XC7A35T-1CPG236C
- 33,280 logic cells in 5200 slices
- 1,800 Kbits de memorie RAM
- 5 clock management tiles, fiecare avand un phase-locked loop (PLL)
- 90 DSP slices
- Internal clock speeds care depasesc 450 MHz
- Convertor analog-digital on-chip (XADC)
- Digilent USB-JTAG port pentru programare si comunicare FPGA
- Serial Flash
- USB-UART Bridge
- 12-bit VGA output
- USB HID Host for mouse-uri, tastaturi si stick-uri de memorie
- 16 user switches (comutatoare)
- 16 user LEDs
- 5 user pushbuttons (butoane)
- 4-digit 7-segment display (afisor cu 4 cifre si 7 segmente)
- 4 porturi Pmod: 3 porturi Pmod Standard 12-pin, 1 semnal XADC cu scop dublu/port standard Pmod

3.2 USB-UART Bridge (Port Serial)

Basys 3 include o punte USB-UART FTDI FT2232HQ (atasata la conectorul J4) care permite utilizarea aplicatiilor pe PC pentru a comunica cu placa, folosind comenzi standard de port Windows COM Port. Datele portului serial sunt interschimbate cu FPGA folosind un port serial cu două fire (TXD / RXD). Dupa generarea bitstream.ului, se cauta conexiunea placutei la PC, urmand mai apoi ca prin intermediul acestei conexiuni prin portul USB sa fie programata placa si sa se execute implementarea

Două LED-uri de stare ofera feedback vizual asupra traficului/datelor care circulă prin port: LED-ul de transmisie (LD18) și LED-ul de recepție (LD17).

FT2232HQ este, de asemenea, utilizat drept controller pentru circuitele Digilent USB-JTAG, dar functiile USB-UART și USB-JTAG(pentru comunicarea dintre FPGA si Vivado) se comporta complet independente una de cealalta. Combinarea acestor doua caracteristici intr-un singur dispozitiv permite ca Basys 3 sa fie programat, comunicat prin UART si alimentat de la un PC atasat cu un singur cablu Micro USB.[4]

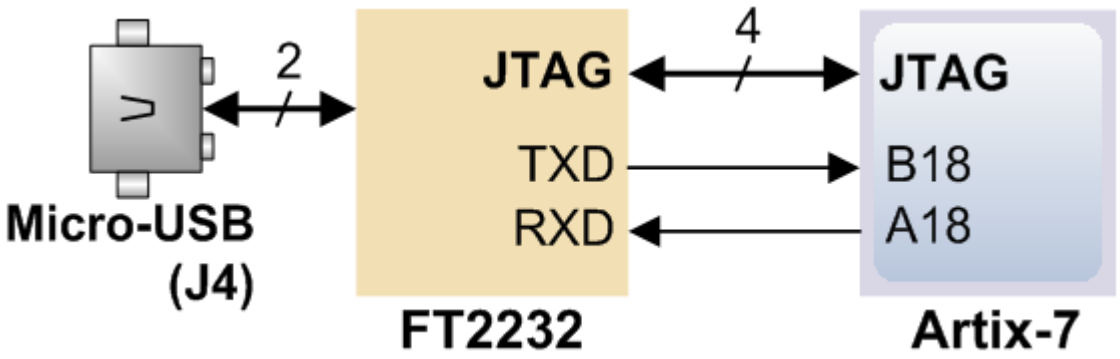


Figura 3.2: Conexiunile dintre FT2232HQ si Artix-7

3.3 USB HID Host

Microcontrolerul cu functie auxiliara (Microchip PIC24FJ128) asigura Basys 3 cu USB HID host capability.

Dupa pornire, microcontrolerul este in modul de configurare, fie descarcand un flux de biti pe FPGA, fie asteptand sa fie programat din alte surse. Odata ce FPGA-ul este programat, microcontrolerul comuta in modul aplicatie, care in acest caz este modul USB HID Host. Firmware-ul din microcontroler poate conduce un mouse sau o tastatura atasata la conectorul USB de tip A la J2 etichetat „USB”. PIC24 conduce mai multe semnale in FPGA - doua sunt folosite pentru a implementa o interfata PS/2 standard pentru comunicarea cu mouse-ul sau tastatura, iar celelalte sunt conectate la portul de programare serial cu doua fire FPGA, astfel incat FPGA-ul poate fi programat dintr-un fisier stocat pe un USB pen drive.[4]

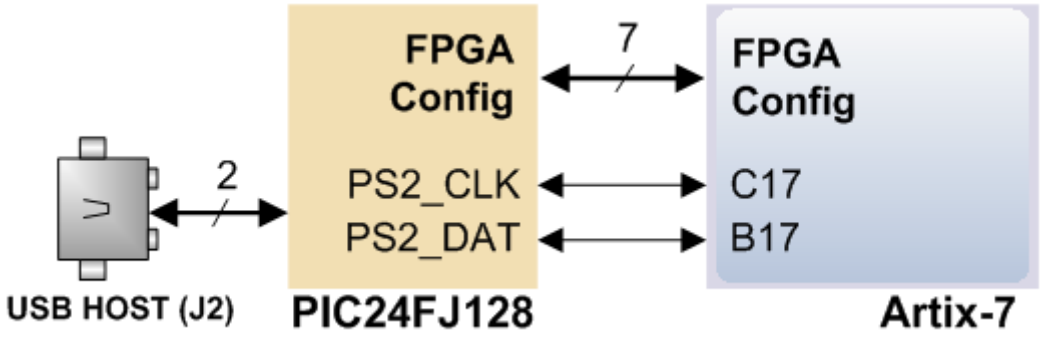


Figura 3.3: Conexiunile dintre Basys3 si PIC24

3.4. Tastatura USB

Tastatura folosește drivere open-collector, astfel incat tastatura sau un dispozitiv gazda atasat pot conduce magistrala cu doua fire (daca dispozitivul gazda nu va trimite date la tastatura, atunci gazda poate utiliza porturi numai pentru intrare).

Tastaturile in stil PS/2 folosesc coduri de scanare pentru a comunica datele de apasare a tastelor. Fiecarei chei i se atribuie un cod care este trimis ori de cate ori este apasata tasta. Daca tasta este mentinuta apasata, codul de scanare va fi trimis in mod repetat aproximativ o data la fiecare 100ms. Cand este eliberata o cheie, se trimite un cod de tastare F0, urmat de codul de scanare al cheii eliberate. Daca o cheie poate fi mutata pentru a produce un caracter nou (cum ar fi o majuscula), apoi un caracter de schimbare este trimis in plus fata de scanare, codul si gazda trebuie sa stabileasca ce caracter ASCII sa utilizeze. Unele taste, numite taste extinse, trimit un E0 inainte de codul de scanare (si pot trimite mai multe coduri de scanare). Cand este eliberata o cheie extinsa, un E0 F0 se trimite codul de tastare, urmat de codul de scanare. [4]

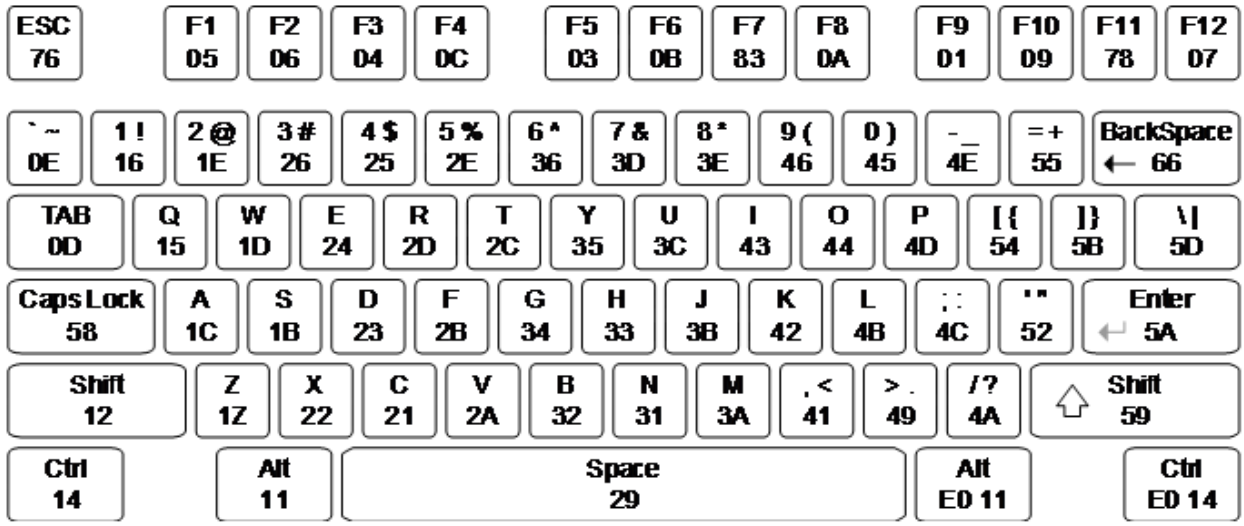


Figura 3.4: Codurile de scanare pentru majoritatea tastelor

3.5 Seven Segment Display

Placa Basys3 contine un afisor LED cu patru cifre din anod comun, cu sapte segmente. Fiecare dintre cele patru cifre este compusa din sapte segmente aranjate intr-un model ca in figura 3.5.1, cu un LED incorporat in fiecare segment.

Figura 3.5.1: Un afisor sapte segmente neiluminat si noua modele de iluminare corespunzatoare cifrelor zecimale si nod comun al circuitului anodic

Anozii celor sapte LED-uri care formeaza fiecare cifra sunt legati impreuna intr-un nod de circuit „anod comun”, dar catozii LED raman separati. Semnalele anodului comun sunt disponibile ca patru semnale de intrare „activare cifra” la afisorul din 4 cifre. Catozii de segmente similare de pe toate cele patru afisaje sunt conectati in sapte noduri de circuit etichetate CA prin CG (astfel, de exemplu, cei patru catozi „D” din cele patru cifre sunt grupati impreuna intr-un singur nod de circuit numit „CD”). Aceste sapte semnale catodice sunt disponibile ca intrari pe afisorul cu 4 cifre. [4]

Pentru a ilumina un segment, anodul ar trebui sa fie condus sus, in timp ce catodul este condus jos. Cu toate acestea, deoarece Basys3 foloseste tranzistoare pentru a conduce suficient curent in punctul comun al anodului, activarea anodului este inversata. [4]

Un circuit de control al afisajului de scanare poate fi utilizat pentru a afisa un numar din patru cifre pe acest afisaj. Acest circuit conduce semnalele anodice si modelele de catod corespunzatoare ale fiecarei cifre intr-o succesiune continua repetata la o rata de actualizare care este mai rapida decat poate detecta ochiul uman. Fiecare cifra este iluminata doar o patrime din timp, dar pentru ca ochiul nu poate percepe intunecarea unei cifre inainte de a fi luminata din nou, cifra apare continuu luminata.[4]

4. Proiectare si implementare

Pentru implementarea proiectului s-a ales o abordare hardware.

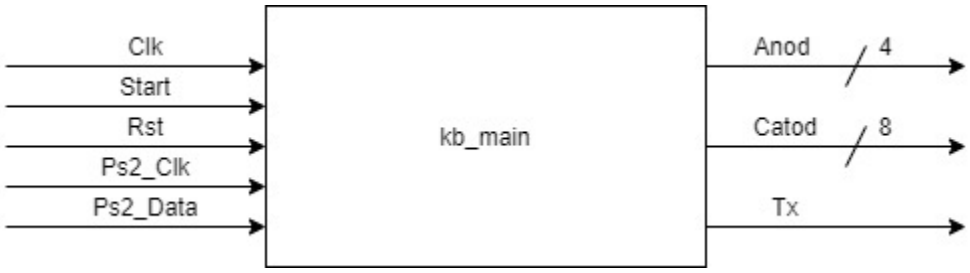


Figura 4.1. Cutia neagra a circuitului

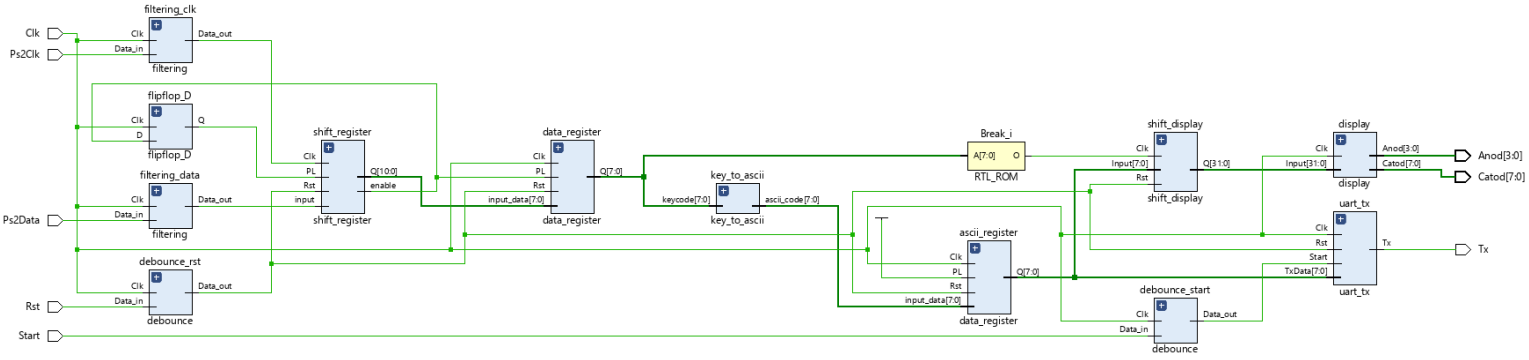


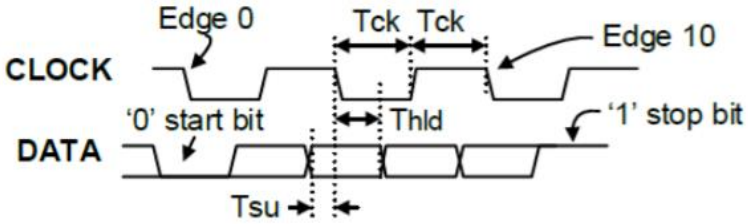
Figura 4.2. Schema bloc a circuitului

4.1. Afisarea pe display-ul cu 7 segmente a datelor transmise de tastatura USB HID

Pentru afisarea pe display a datelor transmise, mai intai am realizat legatura dintre tastatura si placuta Basys3 prin portul USB HID. Stiind ca portul transmite datele in mod serial, am ales sa utilizam un **registru de deplasare** in care la fiecare clock se va memora data transmisa prin port, bucata cu bucata, pana cand se ajunge la transmiterea totala a informatiei si obtinerea vectorului de date.

Tastatura ofera atat datele de care avem nevoie, cat si un clock propriu, insa in implementare trebuie sa luam in considerare faptul ca atat aceste date, cat si clock-ul sunt active pe nivelul logic “high” si inactive pe nivelul logic “low”. Datele vor fi transmise incepand cu bitul de Start pe logic “low”, urmand sa transmita byte-ul de date, un bit de paritate, iar la final un bit de Stop pe logic “high”. Astfel, transmiterea datelor incepe cu cel mai putin semnificativ bit (LSB). Fiecare bit trebuie citit pe falling edge-ul semnalului de clock. [6] Asadar, registrul de deplasare va face shiftare la dreapta pe falling edge, memorand data transmisa.

La fel ca in cazul butoanelor, unde avem nevoie de **debouncer** pentru a obtine un singur semnal corect in locul multiplelor semnale generate din cauza imperfectiunilor hardware ale placutei, vom avea nevoie de un debouncer numit “**filter**” pentru clock-ul si datele(tastele) tastaturii. Implementarea acestuia a fost realizata cu ajutorul informatiilor prezentate in [6].



Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

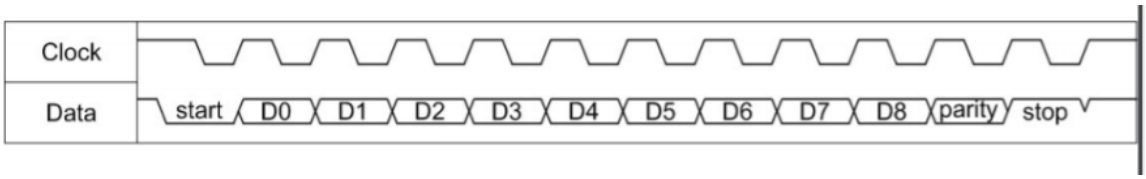


Figura 4.1.1. Transmiterea seriala prin USB HID.

Odata completa transmiterea unui byte si obtinerea vectorului de 11 biti, atat semnalul de clock cat si semnalul de date se reintorc la nivelul logic “high”. [6] Deci in urma transmisiei, vectorul trebuie resetat pentru a astepta un nou byte. Astfel, in registrul nostru de deplasare am folosit un semnal de enable, a carui valoare este data de negarea bitului ‘0’ (bitul de Start). Daca bitul respectiv va mentine valoarea logica ‘1’ inseamna ca data transmisa nu este completa si semnalul PL va continua sa fie ‘0’, dar in momentul in care transmiterea byte-ului a luat sfarsit si pe pozitia 0 se gaseste valoarea ‘0’, atunci semnalul PL va deveni’1’, moment in care tot vectorul va fi resetat la valoarea “high” (un vector doar cu valori de ‘1’).

Din acest vector de 11 biti noi trebuie sa retinem stric valoarea datelor (8 biti) care urmeaza a fi decodificate. Pentru aceasta am utilizat un alt **registru de date** care memoreaza cei 8 biti necesari. Astfel, in acest registrul cand clock-ul este active si semnalul PL=’1’ inseamna ca datele din registrul de deplasare sunt complete si cei 8 biti care ne intereseaza pot fi adaugati in registrul de date. Avand in vedere concurenta componentelor, este necesar un **bistabil D** intre cele doua componente pentru a realiza intarzierea informatiei, astfel incat sa se transmita correct datele.

Datele sunt transmise mai apoi la **decodificator**, componenta ce converteste codurile de pe tastatura in coduri ascii ce pot fi usor citite de componentele viitoare. Toate aceste date vor fi memorate intr-un **al doilea registru de date** pentru a preveni pierderea informatiei la transmiterea intre componente. Datele salvate ajung la componenta de **deplasare a display-ului**, componenta ce asigura o vizualizare mai placuta pentru utilizator, deplasand la stanga fiecare octet primit anterior, pentru ca utilizatorul sa poata vedea ultimele 4 taste apasate, folosind toata capacitatea afisorului.

Vectorul de date deplasate va fi transmis componentei de **display**, care se va asigura ca totul va fi vizibil pe afisoare, decodificand din hexazecimal in 7 segments DCD pentru a aprinde segmentele corespunzatoare. Daca se doreste resetarea afisorului/informatiei in general, se poate utiliza butonul de Reset (U18- butonul din mijloc), care va readuce toate componentele in starea initiala.

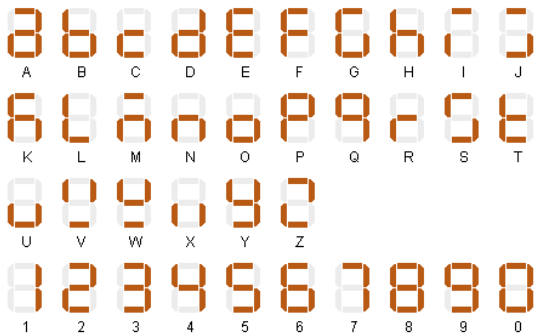


Figura 4.1.2. 7-Segment Alphabet Siekoo

4.2. Afisarea in consola HTerm a datelor transmise de Basys3 prin portul serial UART

Pentru realizarea transmisiei datelor prin portul serial UART, am implementat schema bloc din figura 4.2.1.

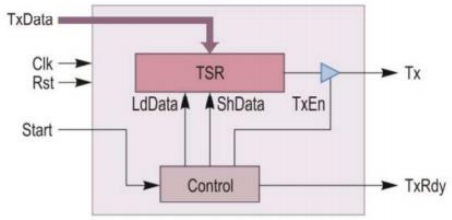


Figura 4.2.1 Schema bloc a transmitatorului serial UART.

Datele mentinute in cel de-al doilea reigstru de date ce memoreaza codurile ascii, vor fi transmise simultan si la componenta **UART**. Aceasta are rolul de a transmite serial aceasta informatie prin portul UART aplicatiei “HTerm” de pe calculator la apasarea butonului de Start.

La fel ca in cazul portului USB HID, aceasta componenta are rolul de a transmite bitii de date unul cate unul de la cel mai putin semnificativ pana la cel mai semnificativ, avand la capete bitii de Start si Stop, astfel incat canalul comunicarii sa fie optim. UART contine registru de deplasare, metoda fundamentala de conversie intre formele seriale si paralele. Conexiunea UART-ului poate fi receiver(primeste informatia), transmiter(transmite informatia mai departe) sau alternativ(receiver/transmiter). In implementarea noastra am avut nevoie de o transmitere receiver pentru datele receptionate de la Basys3.

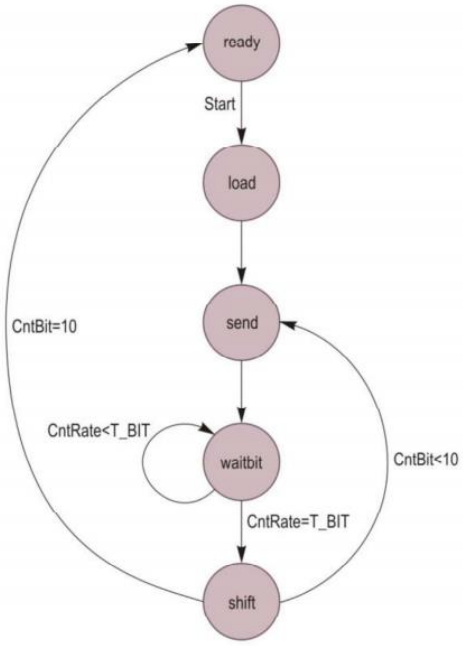


Figura 4.2.2. Diagrama de stare a unitatii de control pentru transmitatorul serial UART.

Manual de utilizare

Pentru inceput este nevoie de instalarea celor doua IDE pe care le vom utiliza, precum si de Basys3 si o tastatura USB care foloseste un protocol compatibil cu placuta (un model mai vechi):

- Instalare Xilinx Vivado Design Suite [7]
- Instalare HTerm [8]
- Compatibilitate tastatura: ledul portocaliu de langa portul usb lumineaza scurt si intrerupt

Se deschide aplicatia Vivado, se deschide aplicatia HTerm, se conecteaza placuta la calculator si tastatura la placuta, se incarca fisierul de biti pe Basys3.

Se apasa tasta dorita de la tastatura si se observa rezultatul pe afisorul placutei. Se apasa butonul de Start de pe Basys3 (butonul de sus) pentru afisarea in HTerm a tastei introduse. Se observa atata in HTerm, cat si pe placa datele transmise. Se procedeaza la fel pentru fiecare tasta introdusa. Pentru resetare se apasa butonul de Reset de pe placuta (butonul din mijloc).

5. Rezultate experimentale

In urma realizarii proiectului, am reusit transmiterea si afisarea cu success a datelor de la tastatura USB HID atat pe Basys3, cat si in consola HTerm. Am utilizat mediul de dezvoltare Vivado 2020.1, limbajul folosit este VHDL, iar sistemul de operare este Windows 10.

Astfel, tastele apasate pe tastatura sunt afisate pe display-ul cu 7 segmente conform figurii 4.1.2.

Dupa apasarea tastei A:

Dupa apasarea tastei E:

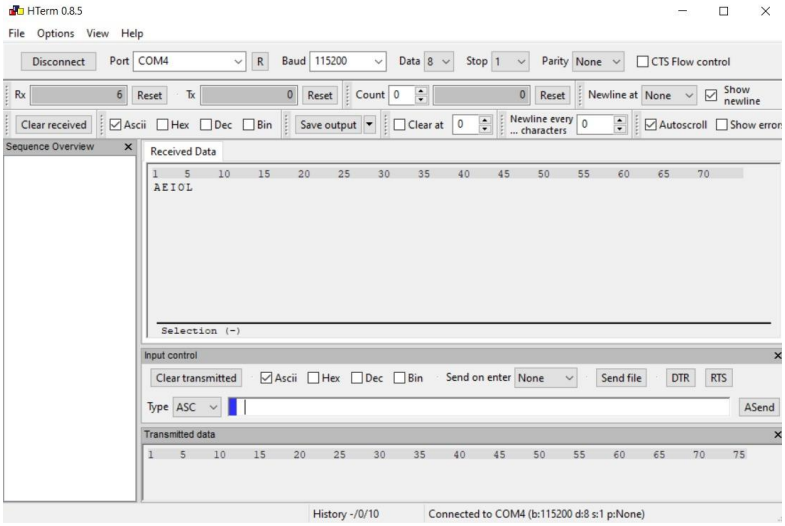
Dupa apasarea tastei I:

Dupa apasarea tastei O:

Dupa apasarea tastei L:

Dupa apasarea tastei SPACE:

In terminalul HTerm, pentru aceleasi date introduse obtinem:



6. Concluzii

Proiectul prezentat implementeaza functionalitatea de citire, transmitere si afisare a tastelor apasate de la o tastatura USB pe afisorul cu 7 segmente de pe Basys3 si mai apoi in terminalul HTerm.

Avantajele acestei implementari sunt:

- ✓ Transmiterea seriala a datelor si vizualizarea pe afisor a ultimelor 4 taste apasate;
- ✓ Ultilizarea unui terminal usor de inteles si folosit (HTerm);
- ✓ Conversia standard a codurilor tastaturii in cod ascii pentru a nu creea confuzii;
- ✓ Transmiterea in seria a informatiilor digitale printr-un singur fir (UART) deoarece este mai putin costisitoare decat transmisia paralela prin mai multe fire.

Un dezavantaj al acestei implementari ar fi faptul ca de fiecare data cand dorim afisarea tastei apasate in terminalul HTerm este nevoie sa apasam butonul de Start de pe placuta.

Realizarea proiectului a reprezentat atat o oportunitate pentru dezvoltarea si aprofundarea cunostintelor legate de limbajul VHDL, cat si de a intelege functionalitatea portului serial UART si USB HID.

O posibila dezvoltare ulterioara ar fi renuntarea la butonul de Start pentru afisarea tastei apasate in aplicatia HTerm.

Bibliografie

[1] C. Patrascu, “Structura sistemelor de calcul”, **http://www.cs.ucv.ro/staff/cpatrascu/SIE/Structura%20sistemelor%20de%20calcul.pdf**

[2] “Arhitectura sistemelor de calcul”, cs.ubbcluj.rowww.cs.ubbcluj.ro › ~forest › Curs › Arhitectura sistemelor de calcul

[3] Ozten Chelai, “Arhitectura calculatoarelor (suport de curs şi laborator)”, https://fmidragos.files.wordpress.com/2012/07/arhitectura-sistemelor-de-calcul.pdf

[4] Digilent, “Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users”, https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/

[4] Constraints for Basys3, **https://github.com/Digilent/Basys3/blob/master/Projects/XADC_Demo/src/constraints/Basys3_Master.xdc**

[5] Dr. Baruch Zoltan Francisc, “Capitolul 6 – Testarea si depanarea proiectelor VHDL”, **Aplicatii-Proiectare-Digitala.pdf (utcluj.ro)**

[6] Richard E. Haskell Darrin M. Hanna LBE Books, “Digital Design Using Digilent FPGA Boards – Third Edition”, 2014, **http://tempus-desire.eu/meetings/2016-06_mc-armenia/downloads/presentations/Basys3%20interfaces.pdf?fbclid=IwAR1hyAb2a9AXRiHtylJ6jG0HaHhajBpIOxAxT44BosexKQ8X_y6J1fPBaRU**

[7] Instalare Xilinx Vivado Design Suite, **https://www.youtube.com/watch?v=DIOll3P65hg**

[8] Instalare HTerm, **https://www.electronoobs.com/eng_arduino_tut17_lib_1.php**

Anexa A

COD SURSA

debounce.vhd

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

entity debounce is
	port(Clk: in std_logic;
	    Data_in: in std_logic;
	    Data_out: out std_logic);
end debounce;

architecture Behavioral of debounce is
signal temp_out: std_logic_vector(2 downto 0);

begin

process(Clk, Data_in)
begin
	if(Rising_Edge(Clk)) then
		temp_out(2) <= Data_in;
		temp_out(1) <= temp_out(2);
		temp_out(0) <= temp_out(1);
	end if;
end process;

Data_out <= temp_out(2) and temp_out(1) and (not temp_out(0));

end architecture;
```

filtering.vhd

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

entity filtering is
        port(Clk: in std_logic;
            Data_in: in std_logic;
            Data_out: out std_logic);
end filtering;

architecture Behavioral of filtering is
signal temp_out: std_logic_vector(7 downto 0);

begin

process(Clk, Data_in)
begin
        if(Rising_Edge(Clk)) then
                temp_out(7) <= Data_in;
                temp_out(6 downto 0) <= temp_out(7 downto 1);
        end if;
end process;

Data_out <= '1' when temp_out = x"FF" else '0';

end architecture;
```

flipflop_D.vhd

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

entity flipflop_D is
  Port (Clk: in std_logic;
              D: in std_logic;
                    Q: out std_logic);
end flipflop_D;

architecture Behavioral of flipflop_D is
signal Q_temp: std_logic;

begin

process (Clk) begin
   if rising_edge(Clk) then
      Q_temp <= D;
   end if;
end process;

Q <= Q_temp;

end Behavioral;
```

shift_register.vhd

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

-- Registru de shiftare cu rst si pl asincron, si clock pe falling edge.

entity shift_register is
   Port ( Clk: in std_logic;
                Rst: in std_logic;
                PL: in std_logic;
                input: in std_logic;
                enable: out std_logic;
                        Q: out std_logic_vector(10 downto 0));
end shift_register;

architecture Behavioral of shift_register is

signal Q_temp: std_logic_vector(10 downto 0);

begin

process (Clk, Rst)
begin
   if (Rst = '1') then
      Q_temp <= (others => '0');
   elsif (PL = '1') then
      Q_temp <= (others => '1');
   elsif (falling_edge(Clk)) then
            Q_temp <= input & Q_temp(10 downto 1);
          end if;
end process;

Q <= Q_temp;
enable <= not Q_temp(0);

end Behavioral;
```

data_register.vhd

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

entity data_register is
  Port (Clk: in std_logic;
        Rst: in std_logic;
        PL: in std_logic;
        input_data: in std_logic_vector(7 downto 0);
        Q: out std_logic_vector(7 downto 0));
end data_register;

architecture Behavioral of data_register is
signal temp_Q: std_logic_vector(7 downto 0) := (others => '0');

begin

process(Clk, Rst, PL, input_data)
begin
    if(Rst = '1') then
        temp_Q <= (others => '0');
    elsif(Rising_Edge(Clk)) then
        if(PL = '1') then
            temp_Q <= input_data;
        end if;
    end if;
end process;

Q <= temp_Q;

end Behavioral;
```

key_to_ascii.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--decoder key 2 ascii
entity key_to_ascii is
  Port (keycode : in std_logic_vector (7 downto 0) ;
        ascii_code: out std_logic_vector(7 downto 0) );
end key_to_ascii;

architecture Behavioral of key_to_ascii is
signal temp: std_logic_vector(7 downto 0);
begin
with keycode select
temp <=
"00110000" when "01000101" , -- 0
"00110001" when "00010110", -- 1
"00110010" when "00011110", -- 2
"00110011" when "00100110", -- 3
"00110100" when "00100101", -- 4
"00110101" when "00101110", -- 5
"00110110" when "00110110", -- 6
"00110111" when "00111101", -- 7
"00111000" when "00111110", -- 8
"00111001" when "01000110", -- 9
```

```
"01000001" when "00011100", -- A
"01000010" when "00110010", -- B
"01000011" when "00100001", -- C
"01000100" when "00100011", -- D
"01000101" when "00100100", -- E
"01000110" when "00101011", -- F
"01000111" when "00110100", -- G
"01001000" when "00110011", -- H
"01001001" when "01000011", -- I
"01001010" when "00111011", -- J
"01001011" when "01000010", -- K
"01001100" when "01001011", -- L
"01001101" when "00111010", -- M
"01001110" when "00110001", -- N
"01001111" when "01000100", -- 0
"01010000" when "01001101", -- P
"01010001" when "00010101", -- Q
"01010010" when "00101101", -- R
"01010011" when "00011011", -- S
"01010100" when "00101100", -- T
"01010101" when "00111100", -- U
"01010110" when "00101010", -- V
"01010111" when "00011101", -- W
"01011000" when "00100010", -- X
"01011001" when "00110101", -- Y
"01011010" when "00011010", -- Z
```

```
"01100000" when "00001110", -- '
"00101101" when "01001110", -- -
"00111101" when "01010101", -- =
"01011011" when "01010100", -- [
"01011101" when "01011011", -- ]
"01011100" when "01011101", -- \
"00111011" when "01001100", -- ;
"00100111" when "01010010", -- '
"00101100" when "01000001", -- ,
"00101110" when "01001001", -- .
"00101111" when "01001010", -- /

"00100000" when "00101001", -- (space)
"00001101" when "01011010", -- (enter, cr)
"00001000" when "01100110", -- (backspace)
"00101010" when others ; --
ascii_code <= temp;
end Behavioral;
```

shift_display.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shift_display is
   port (Clk: in std_logic;
         Rst: in std_logic;
         Input: in std_logic_vector(7 downto 0);
         Q: out std_logic_vector(31 downto 0));
end shift_display;
architecture Behavioral of shift_display is
signal temp_Q : std_logic_vector(31 downto 0);
 begin
    process (Clk, Rst, Input)
    begin
        if Rst = '1' then
            temp_Q <= (others => '1');
        elsif Rising_Edge(Clk) then
            temp_Q(31 downto 24) <= temp_Q(23 downto 16);
            temp_Q(23 downto 16) <= temp_Q(15 downto 8);
            temp_Q(15 downto 8) <= temp_Q(7 downto 0);
            temp_Q(7 downto 0) <= Input;
         end if;
    end process;
Q <= temp_Q;
end Behavioral;
```

display.vhd

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


entity display is
        port (Clk: in std_logic;
                Input: in std_logic_vector(31 downto 0);
                Anod: out std_logic_vector(3 downto 0);
                Catod: out std_logic_vector(7 downto 0));
end display;


architecture Behavioral of display is
signal HEX: std_logic_vector(7 downto 0);
signal S: std_logic_vector(15 downto 0) := "0000000000000000";
begin
--counter up
process(clk)
begin
if clk'event and clk='1' then
   S <= S+1;
end if;
end process;
```

```
--primul mux 4:1
process(S, Input)
begin
   case S(15 downto 14) is
      when "00" => HEX<= Input(7 downto 0);
      when "01" => HEX<= Input(15 downto 8);
      when "10" => HEX<= Input(23 downto 16);
      when others => HEX<= Input(31 downto 24);
   end case;
end process;
--hex to 7 seg dcd
with HEX SELect
  catod <=  "11000000" when "00110000",   --0
                "11111001" when "00110001",   --1
          "10100100" when "00110010",   --2
          "10110000" when "00110011",   --3
          "10011001" when "00110100",   --4
          "10010010" when "00110101",   --5
          "10000010" when "00110110",   --6
          "11111000" when "00110111",   --7
          "10000000" when "00111000",   --8
          "10010000" when "00111001",   --9

          "10001000" when "01000001",   --A
          "10000011" when "01000010",   --b
          "10100111" when "01000011",   --c
```

```
"10100001" when "01000100",   --d
"10000110" when "01000101",   --E
"10001110" when "01000110",   --F
"11000010" when "01000111",   --G
 "10001001" when "01001000",   --H
"11101110" when "01001001",   --i
"11100001" when "01001010",   --j
"10001010" when "01001011",   --k
"11000111" when "01001100",   --L
"10101010" when "01001101",   --m
"10101011" when "01001110",   --n
"10100011" when "01001111",   --o
"10001100" when "01010000",   --P
"10011000" when "01010001",   --q
"10101111" when "01010010",   --r
"11010010" when "01010011",   --S
"10000111" when "01010100",   --t
"11100011" when "01010101",   --u
"11010101" when "01010110",   --V
"10010101" when "01010111",   --w
"11101011" when "01011000",   --x
"10010001" when "01011001",   --y
"11100100" when "01011010",   --Z
"11111101" when "01100000",   -- ' (apostrof)
"10111111" when "00101101",   -- - (minus)
"10110111" when "00111101",   -- = (egal)
```

```
            "11000110" when "01011011",   -- [
            "11110000" when "01011101",   -- ]
            "11101101" when "01011100",   -- \
            "11010011" when "00111011",   -- ;
            "11011111" when "00100111",   -- ' (tilda, sub esc)
            "11110011" when "00101100",   -- ,
            "11111110" when "00101110",   -- .
            "11011011" when "00101111",   -- /
            "11111111" when "00100000",    -- (space)
            "11111111" when "00001101",    -- (enter, cr)
            "11111111" when "00001000",    -- (backspace)
            "11111111" when others ;


--al doilea mux 4:1
process(S)
begin
   case S(15 downto 14) is
      when "00" => anod <= "1110";
      when "01" => anod <= "1101";
      when "10" => anod <= "1011";
      when others => anod <= "0111";
   end case;
end process;
end Behavioral;
```

uart_tx.vhd

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

entity uart_tx is
  generic (N: integer := 115_200);
  Port (Clk: in std_logic;
        Rst: in std_logic;
        TxData: in std_logic_vector(7 downto 0);
        Start: in std_logic;
        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 St : TIP_STARE;

constant Clk_frequency : integer := 100_000_000;
constant T_BIT : integer := Clk_frequency/N;

signal CntRate : integer := 0;
signal CntBit : integer := 0;
```

```
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');

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

begin

 proc_control: process (Clk)
 begin
   if RISING_EDGE (Clk) then
      if (Rst = '1') then
          St <= ready;
      else
         case St is
            when ready =>
               CntRate <= 0;
               CntBit <= 0;
               if (Start = '1') then
                  St <= load;
               end if;
            when load =>
               St <= send;
```

```
            when send =>
               CntBit <= CntBit + 1;
               St <= waitbit;
            when waitbit =>
               CntRate <= CntRate + 1;
               if (CntRate = T_BIT-3) then
                  CntRate <= 0;
                  St <= shift;
               end if;
            when shift =>
               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';


 shift_register: process(Clk, Rst)
 begin
   if (Rising_Edge(Clk)) then
      if Rst = '1' then
         TSR <= (others => '0');
      elsif LdData = '1' then
         TSR(8 downto 1) <= TxData;
         TSR(0) <= '0';
         TSR(9) <= '1';
      elsif ShData = '1' then
         TSR <= '0' & TSR(9 downto 1);
      end if;
   end if;
 end process;
end Behavioral;
```

kb_main.vhd

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


entity kb_main is
        port (Clk: in std_logic;
        Start: in std_logic;
        Rst: in std_logic;
        Ps2Clk: in std_logic;
        Ps2Data: in std_logic;
    Anod: out std_logic_vector (3 downto 0);
    Catod: out std_logic_vector (7 downto 0);
    Tx: out std_logic);
end kb_main;


architecture Behavioral of kb_main is
--debounce
signal Rst_out: std_logic;
--filters
signal Ps2Clk_out: std_logic;
signal Ps2Data_out: std_logic;
--shift_register
signal PL: std_logic;
signal PL_out: std_logic;
signal key_input: std_logic;
signal key_output: std_logic_vector(10 downto 0);
```

```
--data register
signal key_data: std_logic_vector(7 downto 0);
signal ascii_data: std_logic_vector(7 downto 0);
--uart register
signal Start_out: std_logic;
--ascii register
signal display_data: std_logic_vector(7 downto 0);
signal data: std_logic_vector(7 downto 0);
--break
signal Break: std_logic;
--shift_display
signal shifted_display: std_logic_vector(31 downto 0);
--uart
signal TxRdy: std_logic;
begin


   ------------------------------------------------------------------
   --                  Debouncere si Filtere
   ------------------------------------------------------------------


   -- debouncer pentru butonul Rst
          debounce_rst: entity WORK.debounce port map(
             Clk => Clk,
             Data_in => Rst,
             Data_out => Rst_out);
```

```
-- debouncer pentru butonul Start
 debounce_start: entity WORK.debounce port map(
    Clk => Clk,
    Data_in => Start,
    Data_out => Start_out);

-- filter pentru clk keybord
filtering_clk: entity WORK.filtering port map(
          Clk => Clk,
          Data_in => Ps2Clk,
          Data_out => Ps2Clk_out);

-- filter pentru data keyboard
filtering_data: entity WORK.filtering port map(
          Clk => Clk,
          Data_in => Ps2Data,
          Data_out => Ps2Data_out);
----------------------------------------------------------------------
--                  Bistabil si Registrii
----------------------------------------------------------------------
-- bistabil D pentru intarziere dintre registre => mentinerea datelor
flipflop_D: entity WORK.flipflop_D port map (
    Clk => Clk,
    D => PL,
          Q => PL_out);
```

```
-- serial input de pe keyboard
        shift_register: entity WORK.shift_register port map(
                Clk => Ps2Clk_out,
                Rst => Rst_out,
                PL => PL_out,
                input => Ps2Data_out,
                enable => PL,
                        Q => key_output);

        -- registru care memoreaza datele
        data_register: entity WORK.data_register port map (
            Clk => Clk,
    Rst => Rst_out,
    PL => PL,
    input_data => key_output(8 downto 1),
    Q => key_data);

-- registru care pastreaza codurile ascii
        ascii_register: entity WORK.data_register port map (
            Clk => Clk,
    Rst => Rst_out,
    PL => '1',
    input_data => ascii_data,
    Q => data);
```

```
-----------------------------------------------------------------
--                Decodificator pentru Keys
-----------------------------------------------------------------

        -- decodificator catre transforma datele primite de pe keys in coduri ascii
        key_to_ascii: entity WORK.key_to_ascii port map (
          keycode => key_data,
  ascii_code => ascii_data);

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

        -- break signal to signalize when to stop
        Break <= '1' when key_data = x"F0" else '0';

        -----------------------------------------------------------------
        --                Componente pentru Display
        -----------------------------------------------------------------

        --componenta pentru shiftarea pe display a carcterelor afisate
shift_display : entity WORK.shift_display port map (
    Clk => Break,
    Rst => Rst_out,
    Input => data,
    Q => shifted_display);
```

```
  --afisor cu 7 segmente
  display : entity WORK.display port map (
     Clk => Clk,
     Input => shifted_display,
                  Anod => Anod,
                  Catod => Catod);


  ----------------------------------------------------------------------
  --                    Componenta UART
  ----------------------------------------------------------------------


  --uart pentru transmiterea seriala catre PC
  uart_tx: entity WORK.uart_tx port map (
     Clk => Clk,
     Rst => Rst_out,
     TxData => data,
     Start => Start_out,
     Tx => Tx,
     TxRdy => TxRdy);


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


end Behavioral;
```