

# Proiect la Structura Sistemelor de Calcul

Proiect:

Comunicația dintre placa Nexys 4 DDR și o tastatură USB HID

Nume profesor îndrumator:

Lisman Florin

Nume Student: Vasilache Maria

Grupa: 30238

-2022-

## Contents

|  |    |
|--|----|
| 1. Rezumat .....   | 4  |
| 2. Introducere .....   | 4  |
| 3. Fundamentare teoretică .....  | 5  |
| 1.2.2 USB HID Host.....  | 5  |
| 1.2.3 HID Controller .....   | 5  |
| 1.2.4 Bus Timing.....  | 5  |
| 4. Proiectare și implementare.....                                     | 6  |
| 1.2.5 Conectarea tastaturii la FPGA.....                               | 7  |
| 1.2.6 Tastatura-codificare taste.....                                  | 8  |
| 1.2.7. Codificare taste .....  | 8  |
| 1.2.8. PS2-KeyBoard.....   | 9  |
| 1.2.9. Afisare SSD( seven segment display, afisor sapte segmente)..... | 10 |
| 1.3 Schema bloc .....  | 10 |
| 1.3. Codificare cod ASCII pentru toate tastele:.....                   | 13 |
| 1.3.1 Taste speciale.....  | 13 |
| 1.3.2 Paranteze: .....   | 13 |
| 1.3.3 Litere MARI – mici.....  | 13 |
| 1.3.4 Cifre-codificare .....   | 14 |
| 1.3.5. Afisare pe SSD- Reprezentare grafica: Codificare:hgfedcba ..... | 14 |
| 1.3.6 Afisare pe SSD-Schema bloc .....                                 | 16 |
| 1.3.7. Codificare VHDL .....   | 17 |
| 1.3.8 Afisare pe SSD- Codificare VHDL .....                            | 17 |
| 2. Implementare in VHDL.....   | 18 |
| 2.1 Implementare debouncer .....                                       | 18 |
| 2.2 Implementare Afisare taste apasate .....                           | 19 |
| 2.3 Controller: PS/2.....  | 19 |
| 2.3.4. Generare intrerupere .....                                      | 21 |
| 2.4.1. Controlarea tastelor apasate=>generarea codurilor ASCII .....   | 21 |
| 2.4.2. Retinerea tastelor apasate.....                                 | 21 |
| 4. Legatura intre cod si componente:.....                              | 21 |
| 5. Rezultate experimentale.....  | 22 |
| 6. Concluzii .....   | 25 |

|                     |    |
|---------------------|----|
| Bibliografia: ..... | 26 |
|---------------------|----|

# 1. Rezumat

Proiectarea proiectului se realizeaza utilizând limbajul VHDL. Intrucat tema impune conectarea unei tastaturi la placa urmand sa fie apasata la un moment dat o tasta, care, se va afisata tasta apăsată pe afişajul cu 7 segmente al plăcii Nexys 4 DDR.

Tema se va implementare pe placa Digilent Nexys 4 DDR.

Pentru acest proiect este necesara cunoaserea modului in care se conecteaza tastatura la Nexys 4. De asemenea, etapa de identificare a tastei apasate, urmand sa se afiseze valoarea acesteia pe SSD.

Tastatura foloseşte protocolul standard IBM pentru a comunica cu computerul. Acest protocol are scopul de a trimite *scan code-ul* tastei care este apăsată către PC şi de a primi o comandă de răspuns de la acesta. Astfel, avem de-a face cu un protocol bi-direcţional deoarece fiecare componentă(PC/tastatură) trimite şi primeşte comenzi

Pasii pe care trebuie sa-i urmam sunt:

1. Conectarea placutei la tastatura prin interfaţa PS2
  2. Codificarea tastei apasate
  3. Decodificarea tastei
  4. Afisarea pe SSD a valorii corespunzatoare
- 

## 2. Introducere

Proiectul presupune transmitia de date preluate de la tastatura, decodificare si afisarea pe SSD. Intrucat avem doar 7 segmente, este imposibila afisarea tuturor literelor existente, insa pentru o implementare cat mai interactiva, voi afisa pe SSD atat litere mari, cat si litere mici.

Modul de functionare, respecta modelul real, in care pentru afisarea literelor mari se apasa simultan tastele Shift + Tasta.

De altfel, pentru utilizatorii care folosesc in mod frecvent tasta CapsLock, pentru litere mari, am pus la dispozitie existenta aceste functionalitati, prin urmare, se pot folosi de ele.

De remarcat este faptul ca nu s-au putut afisa toate literele mari, din acest motiv, se va afisa 0 pe SSD daca nu exista reprezentare.

Limitarea este data de numarul de afisaje digitale. Pentru o implementare viitoare, care ar pune la dispozitie un SSD mai generos, codul poate identifica apasarea oricarei litere sau combinatii, iar in viitor se pot modifica liniile de cod ale reprezentarii pe Anazi.

### 3. Fundamentare teoretică

Pentru analizarea cerinței, a fost nevoie de o documentare atentă, suportul pe care l-am luat ca punct de referință, a fost documentația de la Digilent.

În această sursă am găsit informații cu privire la PS/2, iar prin prisma aprofundării acestui subiect, mi-am făcut o imagine de ansamblu cu privire la pașii pe care trebuie să-l fac pentru a contura proiectul

Protocolul va prelua comunicarea. Acest lucru se face folosind liniile de date și de clock. Aceste linii sunt **HIGH** atunci când nu are loc comunicarea (idle). Comunicarea tastatură-gazdă și gazdă-tastatură sunt ușor diferite, astfel vor fi explicate separat. Rețineți că, frecvența ceasului este de aproximativ 20-30Khz și este generată de tastatură, dar numai atunci când o transmisie are loc.

Tastatura este liberă să trimită date către gazdă, atunci când ambele linii de date și de clock sunt ținute **HIGH**. Tastatura va lua nivelul **LOW** de date (Bitul de start) și apoi începe să genereze impulsuri de ceas de pe linia de clock

#### 1.2. Descrierea componentelor utilizate (din cadrul plăcuței)

##### 1.2.2 USB HID Host

Placa este prevăzută cu USB HID Host, odată ce FPGA este programat, microcontrolerul trece în modul aplicație, care este USB HID Host în acest caz. Firmware-ul din microcontroler poate conduce mouse-ul sau o tastatură atașată la conectorul USB.

Sunt acceptate doar tastaturile și mouse-urile care acceptă interfața BootHID.

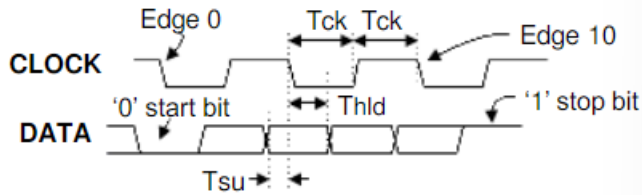
##### 1.2.3 HID Controller

Microcontrolerul cu funcție auxiliară ascunde protocolul USB HID de FPGA și emulează un PS/2 bus de stil vechi. Microcontrolerul se comportă exact ca o tastatură sau un mouse PS/2. Aceasta înseamnă că noile modele pot reutiliza nucleele PS/2 IP existente. Mouse-urile și tastaturile care utilizează protocolul PS/2 folosesc o magistrală serială cu două fire (ceas și date) pentru a comunica cu o gazdă.

Pe Nexys4 DDR, microcontrolerul emulează un dispozitiv PS/2, în timp ce **FPGA joacă rolul gazdei**. Atât mouse-ul, cât și tastatura folosesc cuvinte de **11 biți** care includ un **bit de pornire**, un **octet de date** (LSBfirst), **paritate impară** și **un bit de oprire**, dar pachetele de date sunt organizate diferit,

Interfața tastaturii permite **transferuri de date bidirecționale** (deci dispozitivul gazdă poate aprinde LED-urile de stare de pe tastatură).

##### 1.2.4 Bus Timing



| 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 |

Ceasul și semnalele de date sunt conduse numai **atunci când au loc transferuri de date**; în caz contrar, ele sunt menținute în starea de înaltă impedanță (driver-driver deschis).

Acest lucru necesită ca atunci când semnalele PS/2 sunt utilizate într-un design, pull-up-urile interne trebuie să fie activate în FPGA pe pinii de date și de ceas. (Pull-up: **1**, **când nu e** apasat, **0** **când e** apasat).

Semnalul ceasului este în mod normal condus de dispozitiv, dar poate fi ținut la nivel scăzut de gazdă în cazuri speciale. Timer-ele definesc cerințele de semnal pentru comunicațiile mouse-gazdă și **comunicațiile bidirecționale de la tastatură**. Un circuit de interfață PS/2 poate fi implementat în FPGA pentru a crea o interfață de tastatură sau mouse.

## 5. Proiectare și implementare

### 5.1. Descrierea prin diagrama de stare

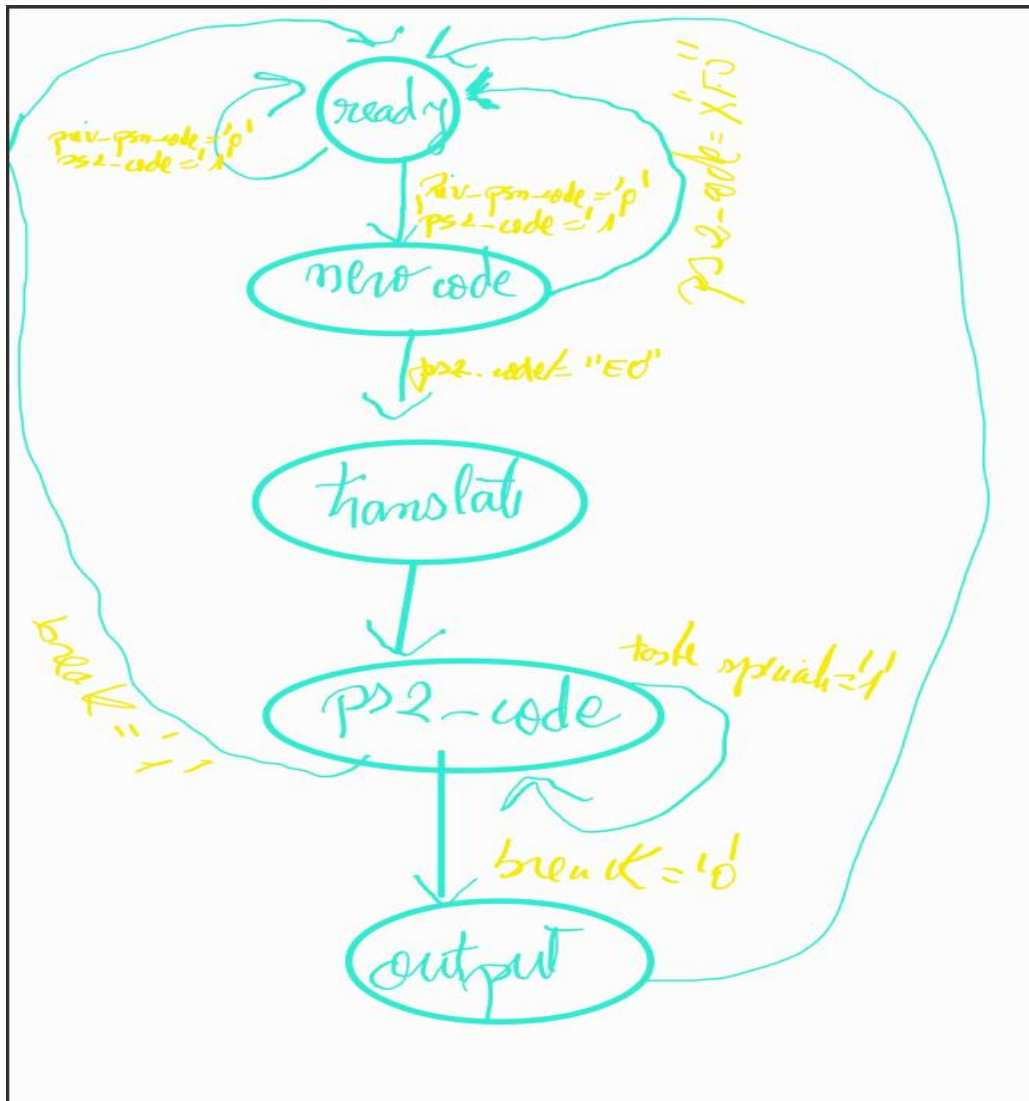
Se poate vedea evoluția automatului, până la start până la ultima, fiind precizate cele mai importante semnale de comandă.

```

TYPE machine IS (ready, new_code, translate, output);
SIGNAL state : machine;
--stările în care se afla dispozitivul nostru

```

FSM – pentru tastatura



### 1.2.5 Conectarea tastaturii la FPGA

Când o tastatură sau un mouse **este conectat** la Nexys4 DDR, se trece la **“self-test passed”**, comanda **“0xAA”** este setată la gazdă.

După aceasta, pot fi emise comenzi către dispozitiv. Deoarece atât tastatura, cât și mouse-ul folosesc același port PS/2, se poate spune tipul de dispozitiv conectat folosind ID-ul dispozitivului.

### 1.2.6 Tastatura-codificare taste

PS/2 foloseste coduri atunci cand se apasa o tasta, codificarea se face in cod ASCII, care este unic pentru fiecare tasta.

Daca tasta este **held-down**, codul va fi transmis in mod repetat la fiecare **100ms**.

Cand se apasa o tasta, se transmite codul F0 key-up code, urmat de codul tastei propriu-zise.

Cand shift este apasata, se produce un nou caracter(litera mare), Majuscula trimite un cod additional

Pentru ca gazda(placuta) **este conectata la ceas**, se verifica in permanenta daca s-a apasat o tasta, mai precis, daca s-a detectat **un cod specific tastelor**

## 1.2.7. Codificare taste

Frecventa: 20-30KHz

Tastatura trimite **date** pe **11 biti**:

→**bit de start**: '0'

→**8 biti**: codul scanat

→**1 bit**: de paritate (impara)

→**1 bit**: bit de stop

Tastatura genereaza 11 tranzitii(20-30 KHz).

Datele sunt validate pe: **falling edge** al ceasului. In cod VHDL, vom folosi urmatoarea sintaxa:

**falling\_edge(clk)**

|                 |           |             |           |            |           |           |           |           |                   |               |           |                        |
|-----------------|-----------|-------------|-----------|------------|-----------|-----------|-----------|-----------|-------------------|---------------|-----------|------------------------|
| ESC<br>76       | F1<br>05  | F2<br>06    | F3<br>04  | F4<br>0C   | F5<br>03  | F6<br>0B  | F7<br>83  | F8<br>0A  | F9<br>01          | F10<br>09     | F11<br>78 | F12<br>07              |
| ~<br>0E         | 1 !<br>16 | 2 @<br>1E   | 3 #<br>26 | 4 \$<br>25 | 5 %<br>2E | 6 ^<br>36 | 7 &<br>3D | 8 *<br>3E | 9 ( )<br>46<br>45 | - _<br>4E     | = +<br>55 | BackSpace<br>← 66      |
| TAB<br>0D       | Q<br>15   | W<br>1D     | E<br>24   | R<br>2D    | T<br>2C   | Y<br>35   | U<br>3C   | I<br>43   | O<br>44           | P<br>4D       | [ {<br>54 | ] }<br>5B<br>\  <br>5D |
| Caps Lock<br>58 | A<br>1C   | S<br>1B     | D<br>23   | F<br>2B    | G<br>34   | H<br>33   | J<br>3B   | K<br>42   | L<br>4B           | ::<br>4C      | ' "<br>52 | Enter<br>↵ 5A          |
| Shift<br>12     | Z<br>1Z   | X<br>22     | C<br>21   | V<br>2A    | B<br>32   | N<br>31   | M<br>3A   | , <<br>41 | > .<br>49         | / ?<br>4A     | ↑<br>59   | Shift<br>59            |
| Ctrl<br>14      | Alt<br>11 | Space<br>29 |           |            |           |           |           |           | Alt<br>E0 11      | Ctrl<br>E0 14 |           |                        |

Figure 9. Keyboard scan codes.



### 1.2.8. PS2-KeyBoard

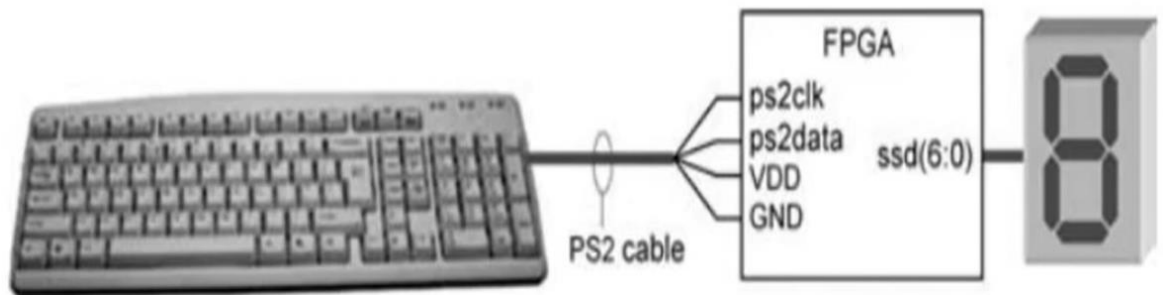
Aceasta este o componentă importantă pentru înțelegerea comunicării computerelor cu perifericele, mai cu seama comunicarea calculator-tastatură.

Pentru interfața PS2, codul se va implementa folosind limbajul VHDL.

Din punct de vedere al apariției Interfața PS2, aceasta a fost introdusă de IBM în 1987 pentru conectarea computerelor la dispozitive cu tastatură și mouse. Chiar dacă acum dă loc interfeței USB (universal serial bus), studiul său ajută la înțelegerea modului în care computerelor comunică cu perifericele.

Sistemul este reprezentat în figură, care arată o tastatură PS2 conectată la un FPGA, care, la rândul său, alimentează un SSD.

#### Design of a PS2 Keyboard Interface



Dorim să proiectăm un circuit capabil să citească tastele numerice (0, 1, ..., 9) și litere de la tastatură și să le afișăm pe SSD atunci când aceste taste sunt apășate.

#### 1.2.8 Schema bloc PS/2 + intrări explicate

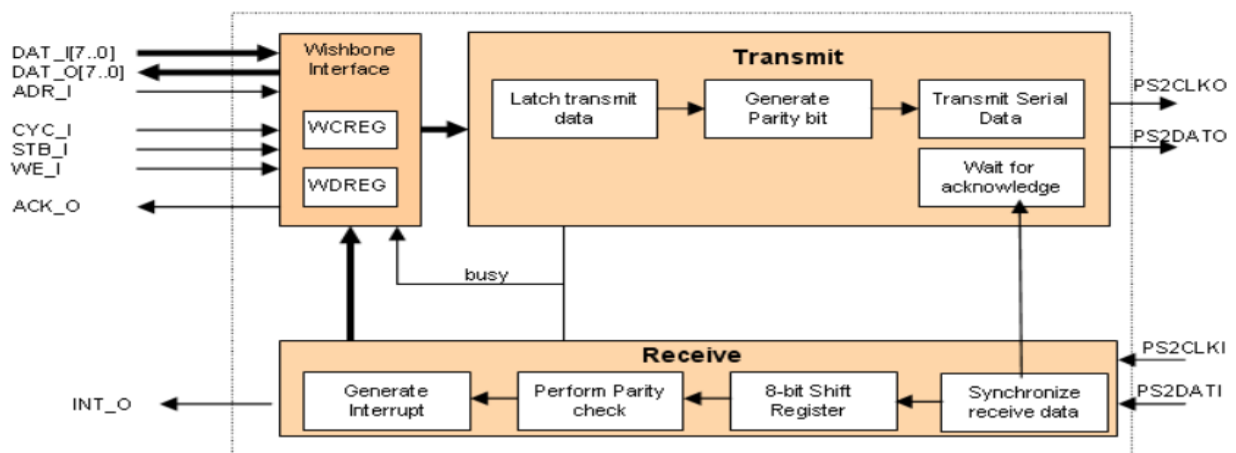


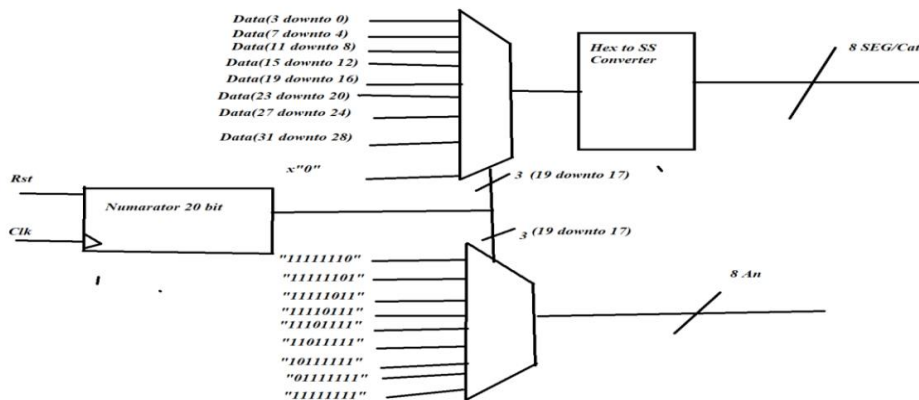
Figure 1. PS2\_W block diagram.

| PS/2 Interface Signals |   |     |   |
|------------------------|---|-----|---|
| PS2CLKTRI              | O | Low | Tri-state enable signal for the PS2CLK bidirectional buffer       |
| PS2CLKO                | O | -   | PS2 clock output  |
| PS2CLKI                | I | -   | PS2 clock input   |
| PS2DATATRI             | O | Low | Tri-state enable signal for the PS2DATA bidirectional buffer      |
| PS2DATAO               | O | -   | PS2 data output (data from the PS2 Controller to the PS/2 device) |
| PS2DATAI               | I | -   | PS2 data input (data from the PS/2 device to the PS2 Controller)  |

### 1.2.9. Afisare SSD( seven segment display, afisor sapte segmente)

Placa dispune de 8 afisaje, insa acestea, au catod comun, iar anodul este separat, prin urmare, avem 4 anodi si 8 catodi. Atat catodul, cat si anodul, sunt active pe '0'. Din acest motiv, pentru codificare, se va folosi logica negativa pentru activarea unui semnal.

Aceste afisaje nu se pot aprinde simultan, avand anodul separate, de aceea se programeaza la o frecventa de 62,5 Hz, cu o rata de refresh la 16ns, fiecare cifra fiind active 2ns. Acest mecanism va fi perceput de ochiul uman ca o afisare simultana a cifrelor.



## 1.3 Schema bloc

Observatie: FPGA-ul este bidirectional, acesta va primi comenzile de la apasarea unei taste, urmand sa afiseze pe SSD valoarea acesteia

Design of a PS2 Keyboard Interface

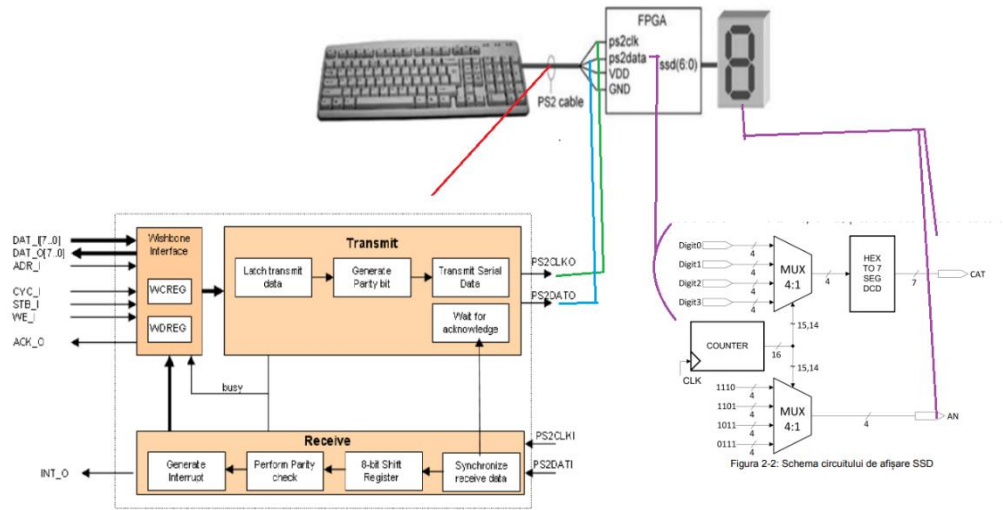
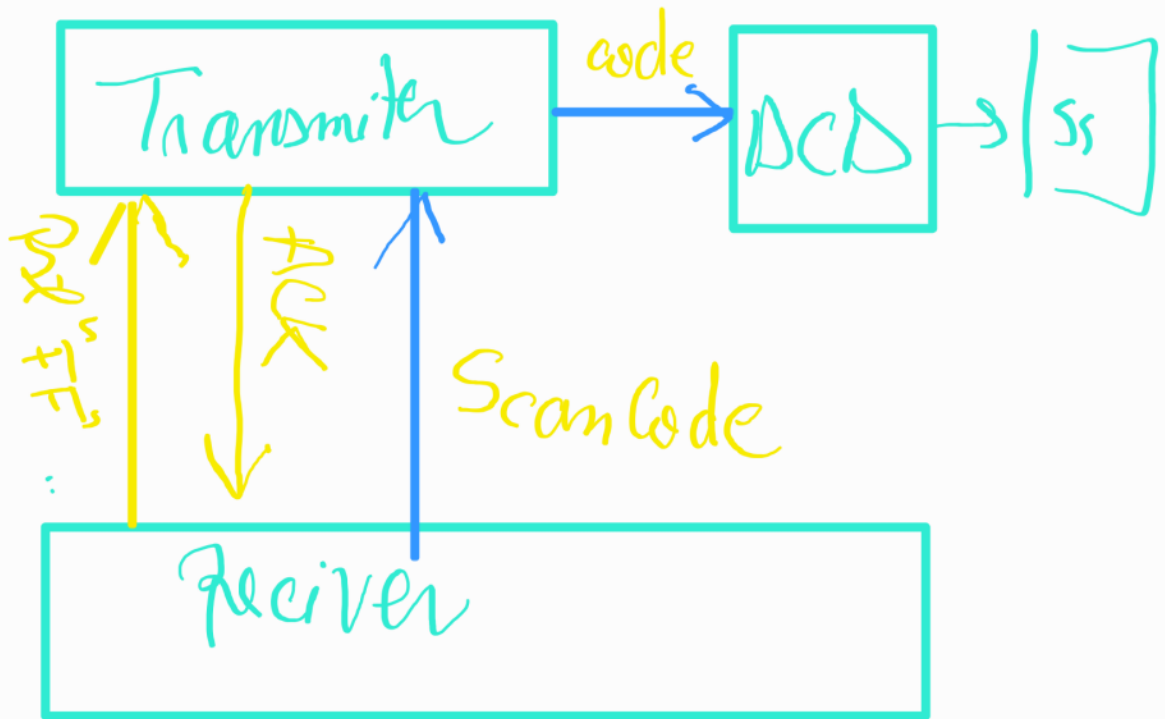


Figure 1. PS2\_W block diagram.

Figura 2-2: Schema circuitului de afisare SSD

PS/2



Observatie:

Cand tastatura este conectata, se va transmite de la reciver, codul 0xFF, care semnifica faptul ca este in starea de primire a semnalului. Cand Transmitter-ul va detecta codul 0xFF, se va transmite codul de activare "0xAA", care semnifica faptul ca se pot receptiona coduri de la taste, respective combinatii de taste.

Dupa aceasta initializare, se vor putea accepta coduri pentru CapsLock si pentru Ctrl, Alt...

## 1.3. Codificare cod ASCII pentru toate tastele:

Observație, întrucât afișajul dispune de 8 segmente, nu se vor putea reprezenta toate tastele, de aceea am ales doar tastele ce urmează să fie afișate, însă pentru documentare, am pus codurile pentru toate tastele.

### 1.3.1 Taste speciale

x"12" - left shift  
code

x"59" -right shift  
code

### 1.3.2 Paranteze:

|            |            |
|------------|------------|
| x"54" --^[ | x"4E" --_  |
| x"5B" --^] | x"4E" --^_ |

### 1.3.3 Litere MARI – mici

|               |              |              |
|---------------|--------------|--------------|
| x"1C" --A/a   | x"33" --^H/h | x"44" --^O/o |
| x"32" --B/b   | x"43" --^I/i | x"4D" --^P/p |
| x"21" --C/c   | x"3B" --^J/j | x"15" --^Q/q |
| x"23" --^D/d  | "42" --^K/k  | x"2D" --^R/r |
| x"24" --^E/e  | x"4B" --^L/l | x"2C" --^T/t |
| x"2B" --^F/f  | x"3A" --^M/m | x"3C" --^U/u |
| x"34" --^G /g | x"31" --^N/n | x"2A" --^V/v |

x"1D" --^W/w

x"35" --^Y/y

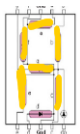
x"22" --^X /x

x"1A" --^Z/z

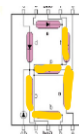
### 1.3.4 Cifre-codificare

|   |           |
|---|-----------|
| 1 | 16 (F016) |
| 2 | 1E (F01E) |
| 3 | 26 (F026) |
| 4 | 25 (F025) |
| 5 | 2E (F02E) |
| 6 | 36 (F036) |
| 7 | 3D (F03D) |
| 8 | 3E (F03E) |
| 9 | 46 (F046) |
| 0 | 45 (F045) |

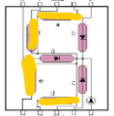
### 1.3.5. Afisare pe SSD- Reprezentare grafica: Codificare:hgfedcba



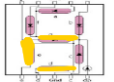
A: 0000010 (1)



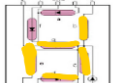
B: 1100000 (1)



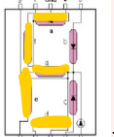
C: 0110001 (1)



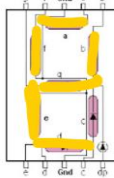
c: 1110010 (1)



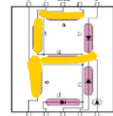
d: 1000010 (1)



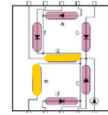
E: 0110000 (1)



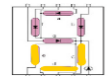
e: 0010000 (1)



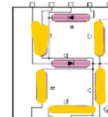
F: 0111000 (1)



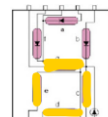
r: 1111010 (1)



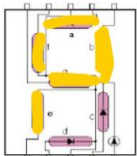
u: 1100011 (1)



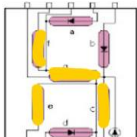
U: 100001 (1)



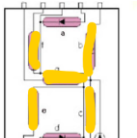
o: 1100010 (1)



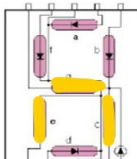
P: 0011000 (1)



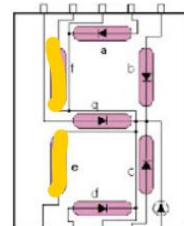
h: 1101000 (1)



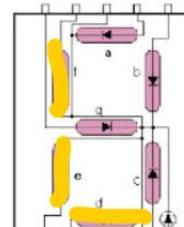
H: 1001000 (1)



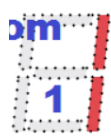
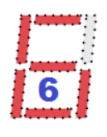

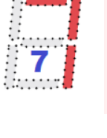






n: 1101010 (1)



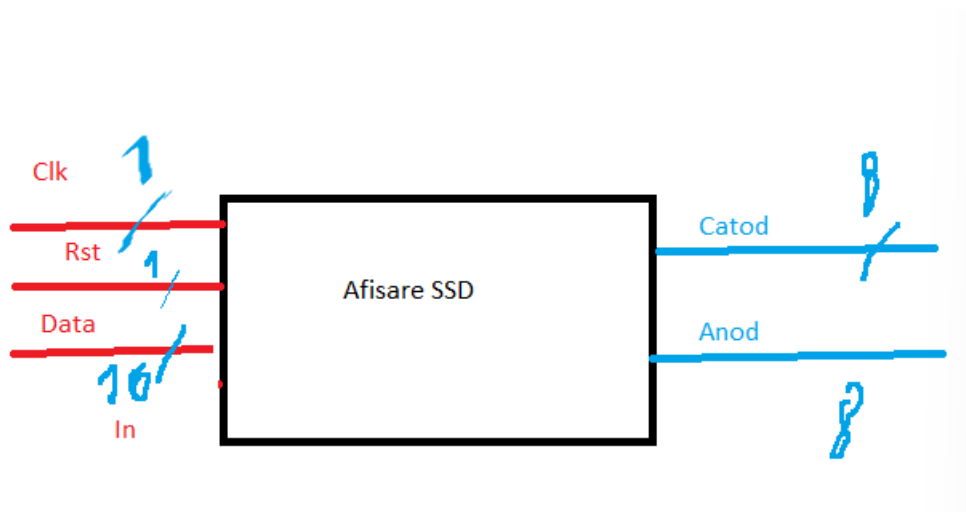
I: 1111001 (1)



L: 1110001 (1)

|   |   |
|---|---|
|    |    |
| 1001111   | 0100000   |
|    |    |
| 0010010   | 0001111   |
|    |    |
| 0000110   | 0000000   |
|    |    |
| 1001100   | 0000100   |
|  |  |
| "0100100  | 00000011  |

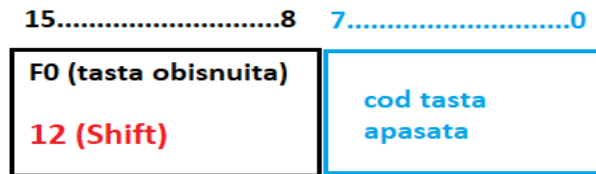
### 1.3.6 Afisare pe SSD-Schema bloc





### 1.3.7. Codificare VHDL

Codificare intrare



### 1.3.8 Afisare pe SSD- Codificare VHDL

```

Cat <= "11111001" when cathodes_aux = x"31" else -- 1
      "10100100" when cathodes_aux = x"32" else -- 2
      "10110000" when cathodes_aux = x"33" else -- 3
      "10011001" when cathodes_aux = x"34" else -- 4
      "10010010" when cathodes_aux = x"35" else -- 5
      "10000010" when cathodes_aux = x"36" else -- 6
      "11111000" when cathodes_aux = x"37" else -- 7
      "10000000" when cathodes_aux = x"38" else -- 8
      "10010000" when cathodes_aux = x"39" else -- 9
      "10001000" when cathodes_aux = x"41" else -- A
      "10000011" when cathodes_aux = x"62" else -- b
      "11000110" when cathodes_aux = x"43" else -- C
      "10100111" when cathodes_aux = x"63" else -- c
      "10100001" when cathodes_aux = x"64" else -- d
      "10000110" when cathodes_aux = x"45" else -- E
      "10000100" when cathodes_aux = x"65" else -- e
      "10101111" when cathodes_aux = x"72" else -- r
      "10001110" when cathodes_aux = x"46" else -- F
      "11100011" when cathodes_aux = x"75" else -- u
      "11000001" when cathodes_aux = x"55" else -- U
      "10100011" when cathodes_aux = x"6F" else -- o
      "10001100" when cathodes_aux = x"50" else -- P
      "10001011" when cathodes_aux = x"68" else -- h
      "10001001" when cathodes_aux = x"48" else -- H
      "10101011" when cathodes_aux = x"6E" else -- n
      "11001111" when cathodes_aux = x"6C" else -- l
      "11000111" when cathodes_aux = x"4C" else -- L
      "10100100" when cathodes_aux = x"53" else -- S
      "10111111" when cathodes_aux = x"2D" else -- -
      "11110111" when cathodes_aux = x"5F" else -- .
      "01111111" when cathodes_aux = x"2E" else -- '
      "11011111" when cathodes_aux = x"27" else -- dreapta(pt cea de langa shift)
      "11111101" when cathodes_aux = x"60" else -- stanga(pt cea de langa 1)
      "11000110" when cathodes_aux = x"5B" else -- [
      "11110000" when cathodes_aux = x"5D" else -- ]
      "10110111" when cathodes_aux = x"3D" else -- =
      "11000000"; -- 0

```

Pe input vom primi 16 biti, care reprezinta: tipul de tasta apasata (primii 8 biti), urmat de codificarea tastei.

HGFEDCBA: codificare valori

## 2. Implementare in VHDL

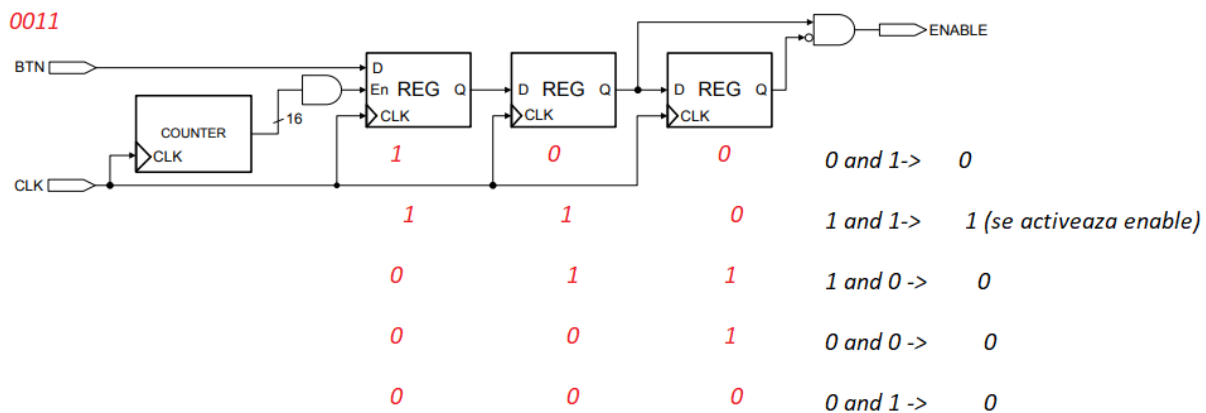
### 2.1 Implementare debouncer

Debouncer-ul numit si circuit monoimpuls, este folosit pentru butoanele care sunt apasate. Fiindca noi vom apasa taste, este posibil ca acestea sa fie uzate, fapt ce va determina un tren de impulsuri, cand, in realitate s-a apasat o singura data.

Debouncer-ul este alcatuit dintr-un numerator, care va fi implementat pe un numar cat mai mare de biti, in functie de uzura butonului, acesta reprezinta perioada de esantionare.

In lista de elemente care contribuie la realizarea circuitului, se numara si un set de 3 bistabile, care propaga un tren de impulsuri. Cand se ajunge la 11...1 pe numerator, se preia valoarea butonului. Cand se ajunge la ultimul bistabil, se face o operatie de SI intre penultimul bit introdus, cu ultimul bit din bistabil negativ.

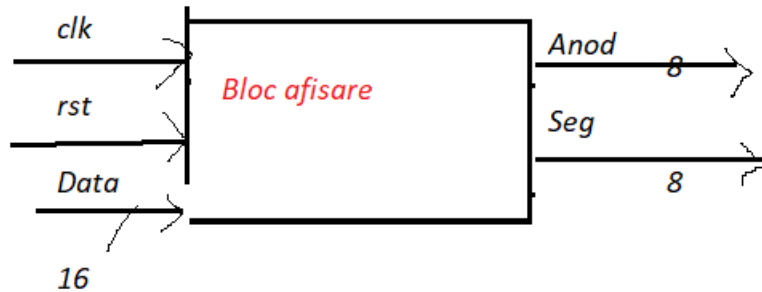
Logica pentru SI, are drept scop activarea o singura data a circuitului, chiar daca se va mai detecta in viitor un 1. Aceasta valoare de 1, nu va face o noua activare a circuitului.



## 2.2 Implementare Afisare taste apasate

Pentru implementarea afisarii pe SSD, m-am folosit de faptul ca pe acesta se va afisa reprezentarea unei singure litere/cifre. Din acest motiv, vom folosi un singur Anod si Catozii pe care ii avem.

Ca input, vom primi un semnal pe 16 biti, reprezentand cele 4 cifre care transmit codificarea tastei apasate. Primele 2 cifre, reprezinta tipul de tasta(normala/shift), iar a doua grupare de 8 biti, reprezinta codificarea propriu-zisa a tastei.



Din punctul de vedere al afisarii, am codificat bitii corespunzatori care vor fi plasati anozilor, pentru a se afisa litera dorita, pentru acest lucru, am folosit un case, urmat de codificarea corespunzatoare.

## 3. Controller: PS/2

### 2.3.1. Synchronize recive data

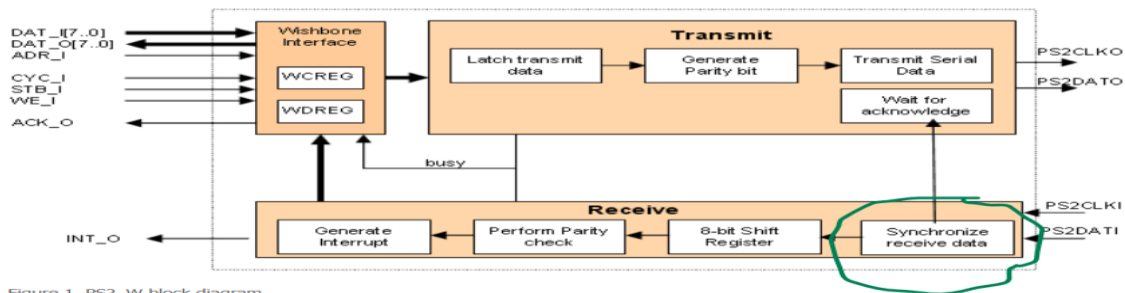
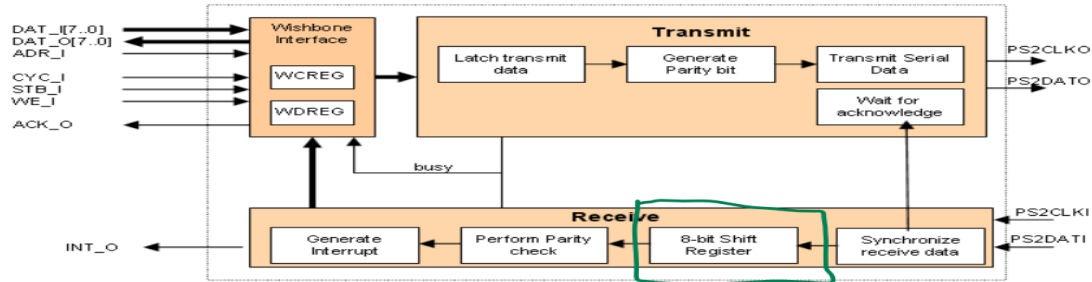


Figure 1. PS2\_W block diagram.

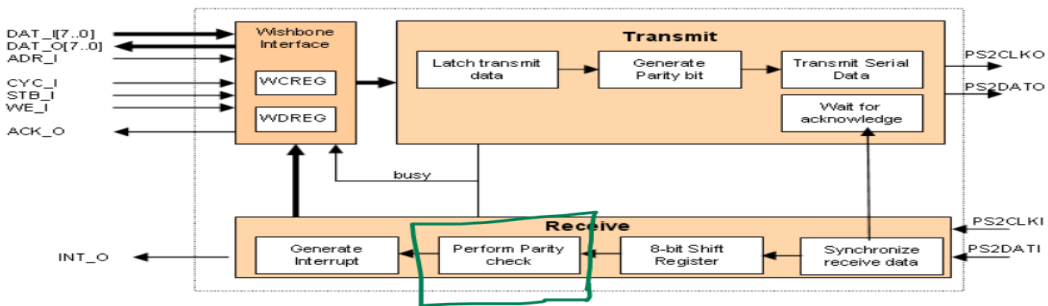
```
--sincronizare filp-flor
PROCESS(clk)
BEGIN
  IF(clk'EVENT AND clk = '1') THEN
    sync_ffs(0) <= ps2_clk;
    sync_ffs(1) <= ps2_data;
  END IF;
END PROCESS;
```

### 2.3.2. Datele primite prin PS/2 -pe front descrescator



```
--datele primite in PS2- pe front descrescator
PROCESS(ps2_clk_int)
BEGIN
  IF(falling_edge(ps2_clk_int)) THEN --falling edge of PS2 clock
    ps2_word <= ps2_data_int & ps2_word(10 DOWNTO 1); --se shiftaza in registrul din PS2 date
  END IF;
END PROCESS;
```

### 2.3.3. Se verifica daca: bitul de paritate, stop, start sunt corecti



Regula pentru validare:

```
--Se verifica daca: bitul de paritate, stop, start sunt corecti
--bit start nu e 0->eroare (0)
--bit stop: este 1-->eroare (11)
--paritatea=>cu xor intre 2 biti adiacenti
--xor 11->0 00 ->0 10->0 01->1 =>0 paritate para, 1 paritate impara

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)));
```

When sending data from the device to the Controller, the PS2 frame consists of 11 fields. The order of these fields is shown in Table 2.

Table 2. PS2 Frame (from PS/2 device to Controller).

| Start | Data 0 | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Parity | Stop |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|------|
|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|------|

- Start - indicates the start of a transmitted frame. This is always a '0'.
- Data0-Data7 - the actual data bits. The least significant bit (Data0) is always sent first.
- Parity - used for error detection. Odd parity is used. Therefore, if the number of '1's in Data0-7 is even, the Parity bit is set (1). If the number of '1's is odd, the Parity bit is cleared (0).
- Stop - indicates the end of the frame currently being transmitted. This is always a '1'.

## Transmisia datelor pe front descrescator

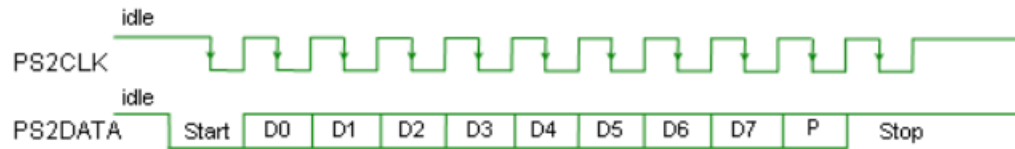
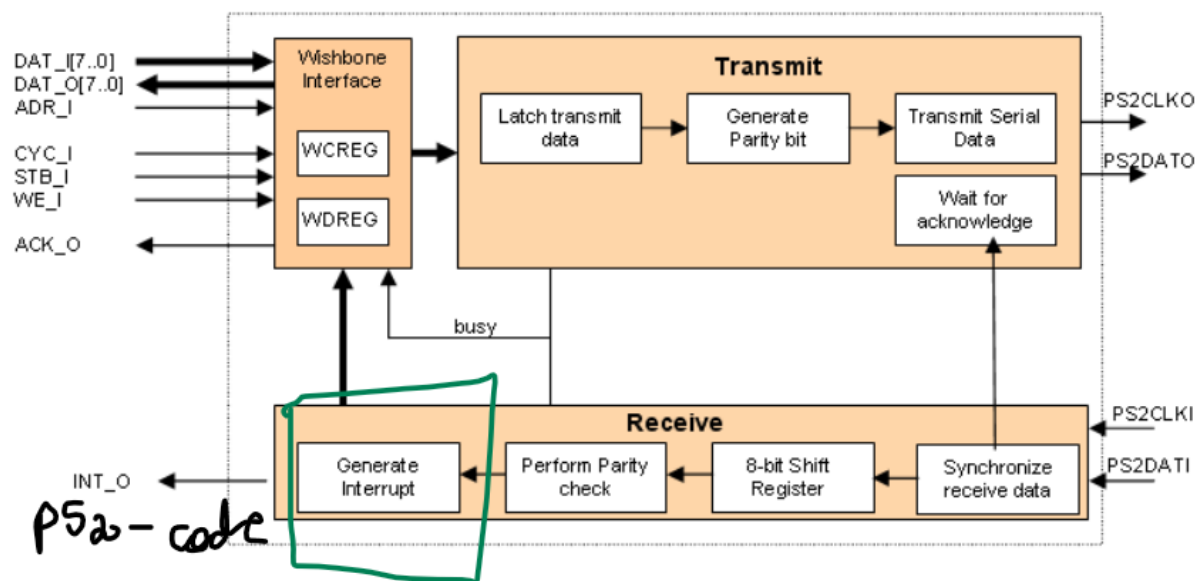


Figure 1. Transmission of data from PS/2 device to Controller.

## 2.3.4. Generare intrerupere



## 2.4.1. Controlarea tastelor apasate=>generarea codurilor ASCII

## 2.4.2. Retinerea tastelor apasate

Intrucat exista posibilitatea sa se apese simultan 2 taste, ex: Shift+A, trebuie sa avem doua variabile, in care sa se memoreze valoarea anterioara(Shift)+ Tasta nou apasata(A)

```
IF(rising_edge(clk)) THEN
    prev_ps2_code_new <= ps2_code_new;    --1
```

# 4.Legarura intre cod si componente:

PS2- descries in ps2\_keyboard.vhdl

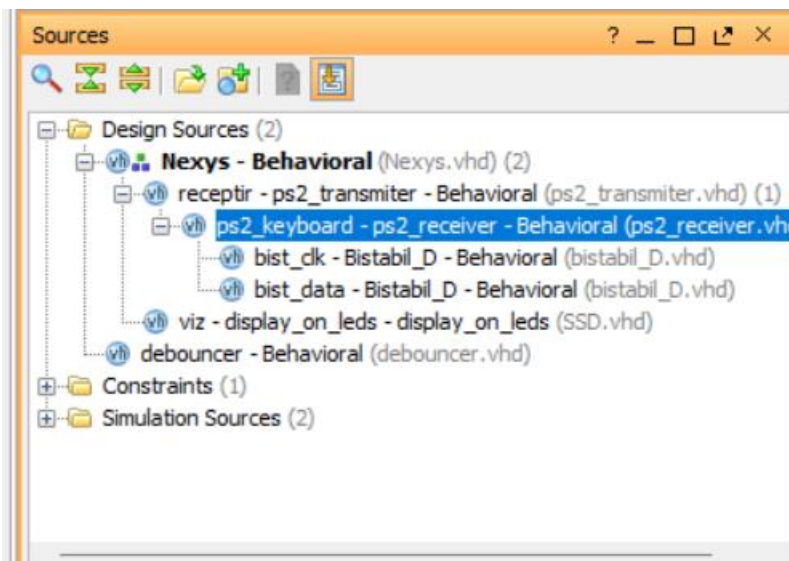


## 6. Detalii implementare

Pentru implementare, vom folosi clk-ul placii si cel al ps2. Cel al placii este de 18\_000 de ori mai mare, din acest motiv pe fiecare ciclu de ceas al placii vom analiza ce comanda s-a transmis. Dupa 8 cicluri de ceas, vom avea noul cod de control, dupa ce se transmite scan\_code-ul, de pe clk-ul placii, se va analiza/transmite, pe clk-ul ceasului

Intrerupere: in mod implicit, la apasarea unui cCTRL, Shift, se genereaza coduri de intrerupere. Se foloseste in acest sens un fleg pentru a mentine informatia de break.

### 6.1. Ierarhia de fisiere



## 6.2. Receiver

```
entity ps2_receiver is
  Port ( ps2_data: in std_logic; --bitul care vine serial, de la tastatura
        ps2_clk: in std_logic; --clockul tastaturii
        CLK: in std_logic;      --clockul placutei
        character_code: out std_logic_vector(7 downto 0); --codul caracterului pe 8 biti
        enable: out std_logic); --new code available flag );
end ps2_receiver;
```

Are rolul de a captat cate un cod primit, verifica codurile, validandu-se tipul de paritate, iar apoi transmite rezultatul obtinut, dupa formarea cuvantului de 11 biti.

Clk-ul placutei are o frecventa mai mare decat clk-ul ps2-ului, din acest motiv, trebuie sa folosim bistabile D, pentru a mentine valoarea corecta pana aceasta va fi citita de PS2. (frecventa ceas: 100\_000\_000, iar ps2=20\_000.

Pentru a determina paritatea si eroarea, se face un xor intre bitii adiacenti, mai precis: --bitul de start=0, bitul de stop=1, verificarea cu parity bit, avem paritate impara, altfel, eroare.

Se va putea trece la urmatorul bit citit, daca, s-a ajuns la 5\_000ns  $\Leftrightarrow$  100\_000\_000/20\_000, altfel, se mentine citirea.

Dupa citire, se activeaza "enable", care anunta citirea complete.

## 6.3. Transmitter

```
5
6 entity ps2_transmitter is
7   Port (
8     clk      : IN  STD_LOGIC;
9     ps2_clk   : IN  STD_LOGIC;      --clock pt PS2
10    ps2_data  : IN  STD_LOGIC;      --datele de pe PS2
11    ascii_new : OUT STD_LOGIC;      --arata daca avem
12    ascii_code : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --ASCII value
13 end ps2_transmitter;
```

In aceasta parte, se verifica ce taste au fost apasate, in functie de scan code-urile generate.

Am folosit in implementare un FSM, pentru a decide care in ce stare va intra automatul in cazul in care s-a primit/generat un cod corect.

Cand tastatura este conectata, se va transmite de la receiver, codul 0xFF, care semnifica faptul ca este in starea de primire a semnalului. Cand Transmitter-ul va detecta codul 0xFF, se va transmite codul de activare "0xAA", care semnifica faptul ca se pot receptiona coduri de la taste, respective combinatii de taste.

Dupa aceasta initializare, se vor putea accepta coduri pentru CapsLock si pentru Ctrl, Alt...

Cazul in care se transmite codul:

“59” - s-a detectat un shift, vom activa un flag, care va fi de ajutor la decodificarea caracterelor.

“12” -shift....

In functie de flag-uri determinam codurile ASCII, care sunt in documentatie, pentru fiecare caracter.

**Obs:** Eu am decodificat toate valorile, pentru o posibila implementare viitoare.

Daca nu se mai citeste nimic, atunci, se activeaza un flag: break, pentru a se decodifica valoarea.

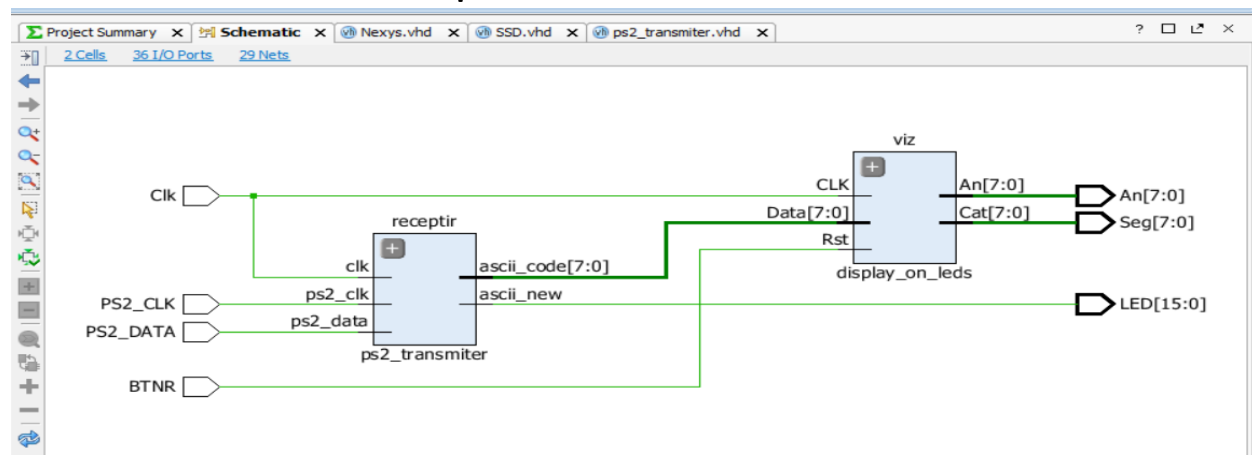
Se va transmite la iesire, codul ASCII corespunzator literei citite.

## 6.4. Afisaj

```
entity display_on_leds is
    port (
        CLK: in std_logic;
        Rst : in STD_LOGIC;
        Data : in STD_LOGIC_VECTOR (7 downto 0);
        An: out std_logic_vector(7 downto 0);
        Cat: out std_logic_vector(7 downto 0)
    );
end;
```

Vom afisa caracterele primite pe afisajul SSD, in functie de codul ASCII, primit pe intrarea Data.

# 7. Rezultate experimentale



Functionalitatea a fost testata pe placa Nexys4.

Problemele aparute pe parcurs, au fost la generarea perioadei pe care se proceseaza tastele, astfel incat sa se citeasca tot scandode-ul.



La testarea finala, rezultatele au fost cele asteptate.

In cadrul laboratorului/proiectului, s-au testat toate cazurile implementate, se pot consulta imaginile de mai jos.

---

### 7.1. Simulare-video

Pentru a observa functionalitatea in timp real, se poate urmări videoclipul realizat in timpul laboratorului:

<https://drive.google.com/file/d/16O4uqwVNLrvX-2C1hvPnq8CWIF6T9bAB/view?usp=sharing>

---

### 7.2 Etape testare

Am avut nevoie de o tastatura, pentru simulare, acest lucru a ridicat un set de probleme, intrucat doua dintre tastaturile pe care am testat proiectul nu transmiteau semnalul de ps2, iar pe afisaj nu se putea vedea nimic, fapt ce a condus catre o cercetare mai profunda a codului si modificari ale acestuia in cazul in care nu era nevoie.

Proiectul se comporta conform asteptarilor, dar doar daca se foloseste o anumita tastatura, de pilda, cele pe care le detin eu, nu se potrivesc cu placuta, doar pe cea din laborator am putut face simularea, in care se observa rezultatele asteptate.

---

## 8.Imbunatatiri ulterioare

Pe viitor s-ar putea analiza apasarea tastelor F, a combinatiilor de taste si a tastelor pentru tastatura numerica.

De asemenea s-ar putea implementa logica pentru caracterele de sub tastele F, de exemplu, semnele de punctuatie, @, %, & si altele.

S-ar putea implementa afisarea tuturor caracterelor citite, in ordinea in care au fost procesate. Pentru interactivitate, s-ar putea implementa logica pentru sageti, care sa modifice pozitia unui caracter afisat.

## 9.Concluzii

---

Pentru implementare a fost necesara citirea documentatiei si transpunerea in cod VHDL.

In urma documentarii, implementarii si testarii, am obtinut rezultatul asteptat.

Proiectul respecta toate cerintele mentionate si are o functionalitate corecta, putandu-se vedea in materialul de mai sus imaginile din timpul testarii.

De remarcat este faptul ca nu s-au putut afisa toate literele mari, din acest motiv, se va afisa 0 pe SSD daca nu exista reprezentare.

Limitarea este data de numarul de afisaje digitale. Pentru o implementare viitoare, care ar pune la dispozitie un SSD mai generos, codul poate identifica apasarea oricarei litere sau combinatii, iar in viitor se pot modifica liniile de cod ale reprezentarii pe Anazi.

---

## Bibliografia:

1. <https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-keyboard-demo/start>
2. [https://www.academia.edu/32460779/Nexys4\\_DDR\\_FPGA\\_Board\\_Reference\\_Manual](https://www.academia.edu/32460779/Nexys4_DDR_FPGA_Board_Reference_Manual)
3. <https://www.win.tue.nl/~aeb/linux/kbd/scancodes-10.html#keyboardid>
4. <https://techdocs.altium.com/display/FPGA/PS2+W++Wishbone+PS2+Controller>
5. <https://techdocs.altium.com/display/FPGA/PS2+W++Transmission+Protocols>
6. <https://techdocs.altium.com/display/FPGA/PS2+Commands>
7. <https://www.fpga-key.com/tutorial/section924>
8. [https://wiki.osdev.org/PS/2\\_Keyboard](https://wiki.osdev.org/PS/2_Keyboard)