FACULTATEA: Automatică și Calculatoare
SPECIALIZAREA: Calculatoare și Tehnologia Informației
DISCIPLINA: Proiectarea sistemelor numerice
PROIECT: TITLUL PROIECTULUI VOSTRU

Îndrumător laborator **Realizator**

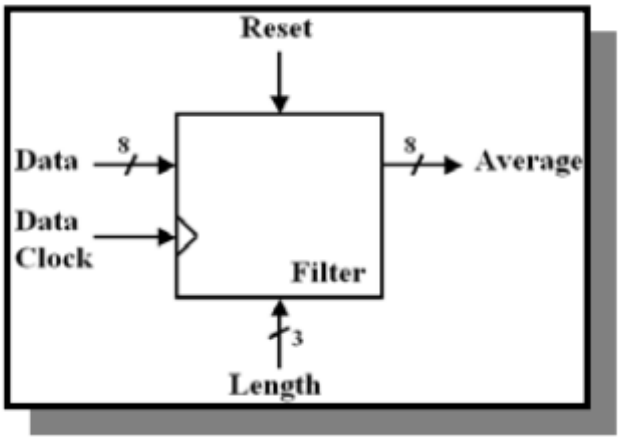
Ing. Călugăr Gabriel-Cătălin **Lucacel Andrei, Svegler Alexa**

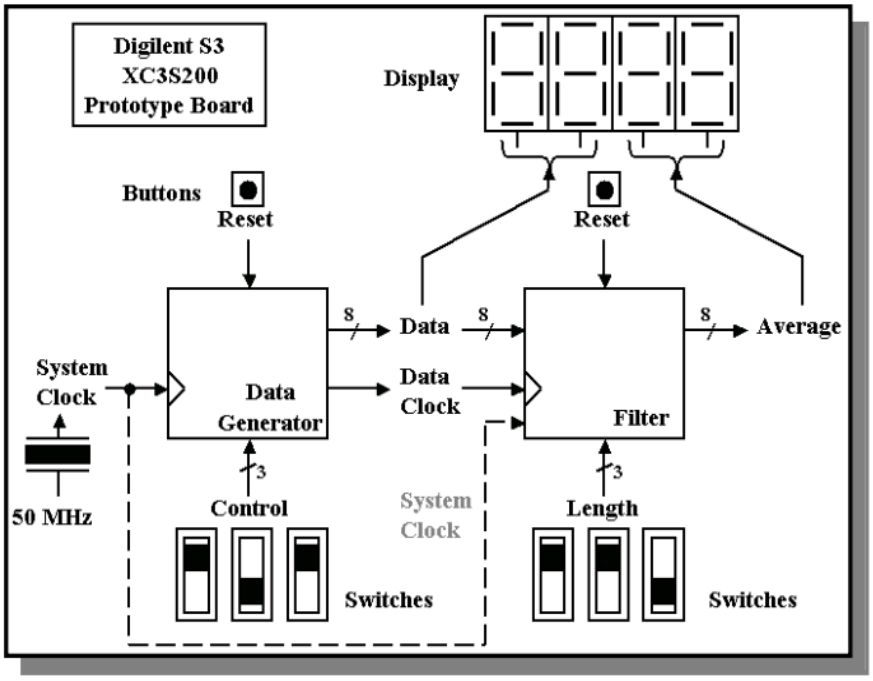
Cuprins

1. Specificația proiectului

Sa se realizeze un sidpozitiv de calcul al mediei unui set de numere, implementabil in dispozitiv FPGA, conform documentatiei din fisierul LabTaskUTCN.pdf. Proiectul va fi realizat de 2 studenti.

2. Schemă bloc, schemă detaliată



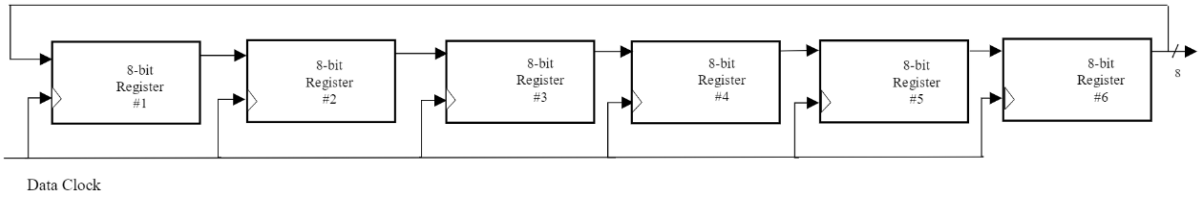


3. Componente

Proiectul este impartit in 2 componente mari: Data Generator si Filtrul.

Data Generator:

A.Memorii (pentru fiecare student)

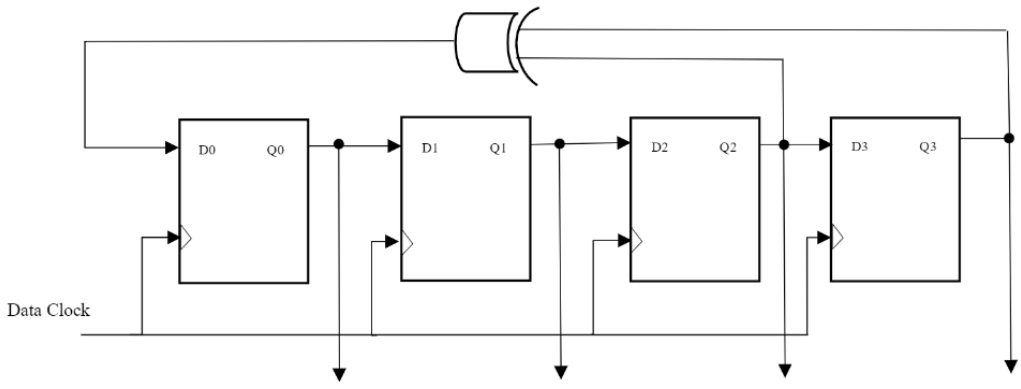


Memoriile sunt formate din 6 registre, folosite pentru a pastra numerele (cate 6 pt fiecare student, fiecare pe 8-biti).Fiecare registru primeste data pastrata in registrul anterior, primul registru primind-o pe cea de la final,formand o bucla.

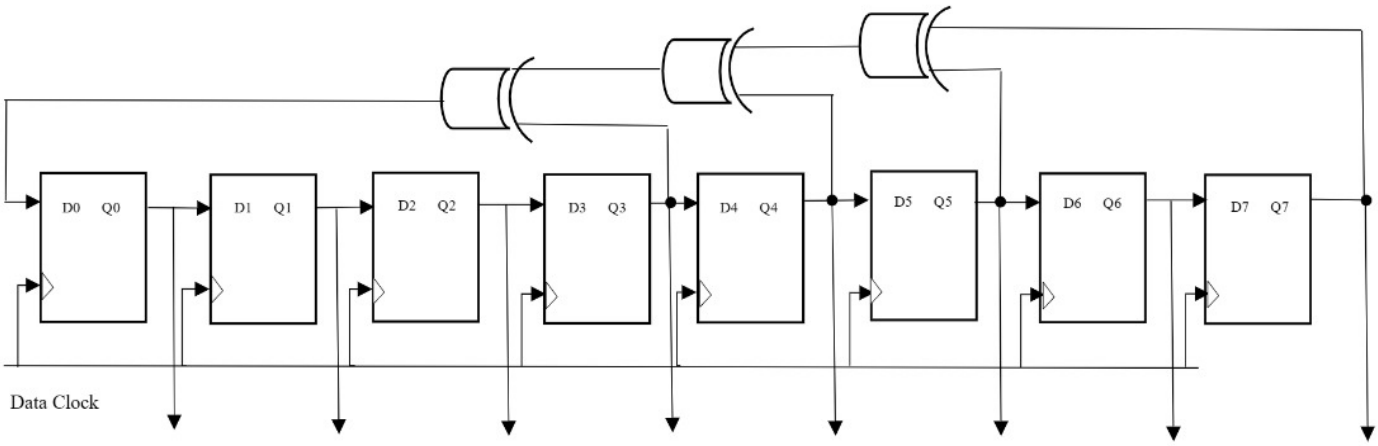
Am realizat două entități diferite, care au ca intrare un CLK și ca ieșire un semnal DATA pe 8 biți. Ambele arhitecturi conțin un proces dependent de ceasul de date de 1 Hz. În ele, am declarat un tip definit de utilizator – memorie1/memorie2 - o matrice formata din 6 vectori de tip vector_logic_standard de 8 biți. De asemenea, am declarat o variabilă de tip StudentMem numită memorie1 și respectiv memorie2 și le-am inițializat cu:1,2,3,4,5,6 iar pentru al doilea elev avem numerele: 7,8,9,10,11,12.

Cu ajutorul variabilei temp am reusit sa mutam numerele din fiecare registru cu o pozitie la dreapta, ultimul devenind primul.

B. Linear Feedback Shift Register



Această diagramă prezintă un registru de deplasare cu feedback liniar pe 4 biți (LFSR), care este format din 4 bistabile de tip D si o poarta Sau-Exclusiv, acționând împreună pentru a produce o secvență binară pseudoaleatorie (PRBS).Putem produce un PRBS de lungime 16-1 = 15, o secvență de lungime maximă care include toate numerele posibile de 4 biți, excluzând numarul cu toate zerourile.



Acesta este, de asemenea, un registru cu deplasare liniară, dar este pe 8 biți, pentru că dorim o gamă mai mare de valori.

Ambele LFRS au fost realizate în diferite entități care au ca intrări CLK și ca ieșiri datele pe 8 biți. În proiectarea acestora am folosit aceleași principii de deplasare a registrelor, în care avem un semnal intern numit OP_INT, pe 4 și respectiv 8 biți, ambii inițializați cu valori diferite de zero.

Avem un proces care depinde de un CLK și atribuim interschimbam valorile din bistabilele 3-1 cu valorile din bistabilele 2-0 ,iar primul flip-flop va primi rezultatul(feedback-ul) in urma operatiunii XOR. Pentru al doilea LFRS, operațiunile includ semnale auxiliare care reține feedback-ul XOR. De asemenea, deoarece avem o ieșire de 8 biți, dar un LFSR de 4 biți pentru primul caz, atribuim celor 4 biți cei mai semnificativi „0000”, iar celorlalți 4 rezultatele din registre. În al doilea LFRS nu este cazul.

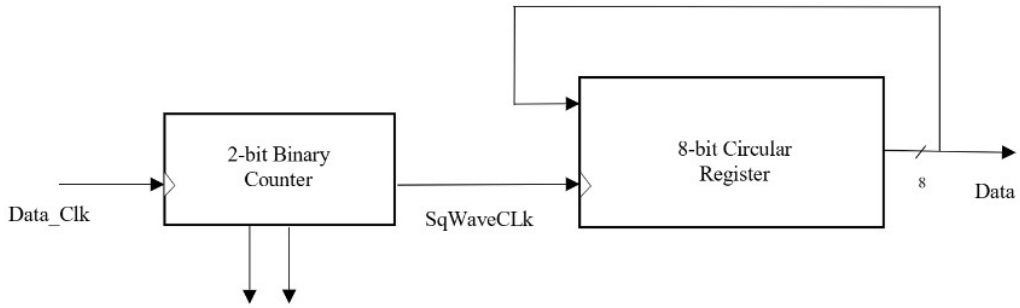
C. Square Wave Generator

Componenta square wave-ului are ca date de intrare Data_Clock și ca ieșire datele reprezentate de semnalul OP.

Prin urmare, am realizat o pereche entitate/arhitectură pentru generatorul de unde square wave-uri în care am declarat și un semnal de ieșire intern OP_INT de 8 biți încărcat cu valoarea „00001000” care va reprezenta iesirea modificata in urma deplasarii circulare aplicate.Operațiile sunt efectuate într-un proces dependent de CLK, în domeniul secvențial, și reprezintă în principal mișcările de deplasare, deoarece biților 6-0 li se atribuie biții 7 -1, al șaptelea primind valoarea primului si formandu-se o bucla.

De asemenea, în zona declarativă a procesului am declarat un vector inițializat cu 0, ceea ce ne va ajuta să întârziem clock-ul de patru ori, la o rată de 0,25 Hz.

In cadrul procesului, pe frontul ascendent al ceasului, numărul crește cu unu și dacă se atinge valoarea 3 atunci se efectuează operațiile de schimbare. Deoarece procesul este sensibil la clock, numărul va fi reinițializat cu 0. Pentru aceasta am folosit pur și simplu declarația if ... then.



2. FILTRUL

Filtrul, așa cum este descris în documentația LabTaskUTCN sau average_computer,cum l-am numit noi, trebuie să aibă mai multe intrări:

*Lenght-indică numărul de elemente pentru care vom calcula media după cum este descris mai jos:

Off - Off - Off (0 - 0 - 0)	Stop - Hold Value
On - Off - Off (1 - 0 - 0)	2 Sample Average
On - Off - On (1 - 0 - 1)	4 Sample Average
On - On - Off (1 - 1 - 0)	8 Sample Average
On - On - On (1 - 1 - 1)	16 Sample Average

Number-este o intrare de 8 biți furnizată de generatorul de date descris anterior.

*Data Clock- este generat de divizorul de frecvență descris mai devreme, care convertește ceasul intern de 100 MHz al plăcii Nexys4 într-un clock de 1 Hz care va fi folosit pentru filtru.

*Reset- este folosit direct de utilizator și șterge întreg modul Filter, încărcând *00000000" pe toate cele 16 registre care stochează numerele noastre.

Avem o singură ieșire când vine vorba de calcularea mediei, și anume Average. Aceasta este media numerelor stocate în modulul nostru în funcție de lungimea dorită de utilizator. Această ieșire va fi folosită de modulul nostru de afișare cu 7 segmente.

Primim datele de intrare pe 8 biți de la generatorul de date. Stocăm cel mult 16 numere de 8 biți în matricea noastră de 16 registre de memorie. Când un număr nou este primit de la generator, ultimul numar stocat în matricea noastră este eliminat.

Calculăm suma ultimelor două numere introduse și stocăm rezultatul în registrul denumit „AVG 2”,apoi shiftam AVG2 la dreapta cu o pozitie pentru a rezulta media aritmetica.Dupa, calculăm suma primelor 4 numere, o deplasăm la dreapta cu două poziții pentru a calcula media acelor patru numere și o stocăm în registrul numit „AVG 4”. Facem același lucru cu primele 8 numere, deplasând rezultatul cu trei poziții și stocarea acestuia în registrul numit „AVG 8”. În cele din urmă, adăugăm toate cele 16 numere, deplasăm rezultatul cu 4 poziții și îl stocăm în registrul „AVG 16”.

Inputul Clock Enable va dezactiva intrarea data_clk atunci când pe selecția Length este 000 și nu vom mai stoca numerele primite de la DataGenerator.

De asemenea avem nevoie de un multiplexor, care are ca selectie intrarea Length. In functie de selectie vom afisa fie media ultimelor 2,4,8 sau 16 numere generate sau sirul “00000000”.

Fiecare registru de memorie are, de asemenea, o intrare Data_clk. Acesta are o frecvență de 1 Hz, deci vectorul cu 16 numere va fi deplasat cu o poziție în fiecare secundă => media celor 2,4,8,16 numere va fi recalculata in fiecare secunda.

In realizarea acestei componente am declarat mai multe tipuri care nu fac parte din libraria standard. Toate aceste tipuri contin vectori de 12 elemente. Motivul pentru care am ales lungimea de 12 este faptul ca in momentul in care adunam toate cele 16 numere fiecare pe 8 biti, suma rezultata are o lungime de cel mult 12 biti. Din moment ce VHDL nu permite adunarea numerelor de lungimi diferite am considerat ca o posibila rezolvare a acestei probleme ar fi ca fiecare numar sa aiba aceeasi lungime maxima posibila.

Primul lucru pe care il facem in proces este sa verificam daca butonul de Reset este activ. In caz afirmativ asignam valoarea “ 00000000“ fiecarui element din vector, iar media ce va fi afisata va fi de asemenea “00000000”.

Apoi trecem la calcularea sumelor. De exemplu dacă vectorul number_array arată astfel number_array = [A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15], atunci vectorii sumelor vor fi:

sum2 = [A0+A1, A2+A3, A4+A5, A6+A7, A8+A9, A10+A11, A12+A13, A14+A15]

sum4 = [(A0+A1) + (A2+A3), (A4+A5) + (A6+A7), (A8+A9) + (A10+A11), (A12+A13) + (A14+A15)]

sum8 =[(A0+A1+A2+A3)+(A4+A5+A6+A7),(A8+A9+A10+A11) + (A12+A13+A14+A15)]

sum16 = [(A0+A1+A2+A3+A4+A5+A6+A7) + (A8+A9+A10+A11+A12+A13+A14+A15)]

După ce vectorul de sume este finalizat trecem la împărțiri. Vom lua primul element din sum2 îl vom shifta la dreapta pentru a face împărțirea, iar apoi vom stoca rezultatul. Procedăm la fel la fiecare vector de sumă, îi shiftăm cu 2,3 respectiv 4 poziții. Deoarece rezultatul va fi mereu pe 8 biti, vectorul ce stochează mediile va fi de tip std_logic_vector pe 8 biți.

La final avem funcția case care selectează rezultatul în funcție de intrarea Length.

3.Display

La realizarea afișorului am hotărât să avem o componentă mare Dislay, ce conține 2 componente: Convertor și FreqDivider7Seg.

Display ia ca intrări numerele și sumele cele mai recent generate. Ieșirea acestei componente este ANOD -ul, indicând care LED de pe placa Nexys4 să afișeze numărul, iar a doua ieșire este CATOD-ul, care reprezintă numărul în sine ce trebuie afișat. Pentru fiecare număr ce trebuie afișat i-am atașat câte 2 anoduri, numerele fiind între 0-255 în zecimal și avem nevoie de 2 anoduri ca să le afișăm in hexazecimal.

Number si average sunt intrările pe 8 biți. Împărțim fiecare număr în 2 numere separate pe 4 biți, pe care le vom trece prin componenta Convertor. Ieșirea va fi un vector pe 8 biți care indică ledurile ce vor fi aprinse pentru fiecare catod pentru a afișa numărul dorit.

Componenta Convertor este format dintr-un proces care contine funcția case. Această funcție desemnează pentru fiecare număr pe 4 biți o secvență parfticulară pe 7 biți, ceea ce va duce la echivalentul numărului nostru pe 4 biți în hexazecimal.

Din moment ce anozii cu respectivii lor catozi se aprind secvențional, noi trebuie să le aprindem unul după altul la o frecvență foarte mare astfel încât să nu putem observa că nu se aprind deodată. Pentru acesta avem un registru pe 4 biți(lucrăm cu doar 4 anozi) care conține 3 de '1' și unu de '0'. Apoi rotim registrul la o frecvență de 200Hz folosind FreqDivider7Seg,ceea ce înseamnă ca îl rotim de 200 de ori pe secundă, făcând să pară că cei 4 anozi se aprind deodtă.

Anozii sunt activi pe 0 logic, deci când registrul indică "1011" înseamnă că al doilea anod este activ.

Componenta FreqDivider7Seg este realizat cu un proces care are o variabilă ce numără pâna la 500 000. Variabila este incrementată de fiecare dată când clock-ul intern al placii trece

printr-un ciclu. Când variabila are o valoare între 0 si 250 000 iesirea componentei este 0. Când variabila este între 250 000 si 499 999 iesirea este 1. Când variabila ajunge la 500 000 se resetează la 0. Acest proces crează un semnal de clock cu frecvența de 200Hz.

4. 1Hz Frequency Divider

Placa Nexys4 are o frecvență de 100Mhz, dar noi trebuie să rulăm programul nostru în timp real deci pentru a sincroniza filter-ul cu data generator-ul avem nevoie de acelasi clock cu frecvența de 1Hz.

Pentru acest lucru am implementat componenta frequency_divider, care are ca intrare CLK și ca ieșire CLK_OUT, care va fi clockul cu frecvența de 1Hz.

Procesul numit DIV are o variabilă counter care numără până la 99 de milioane.Pe frontul ascendent al clock-ului dat variabila crește cu 1. Dacă valoarea contorului este între 0 și 49,5 milioane atunci ieșirea va fi „1”. În caz contrar, va menține „0”. Când contorul a ajuns la valoarea de 99 de milioane, va lua valoarea „0”.

4. Cod VHDL

Componenta mare a acestui proiect este *average.*

```
entity average is
    port (
    CLK: in std_logic;
    RST: in std_logic;
    Length: in std_logic_vector (2 downto 0);
    Control: in std_logic_vector (2 downto 0);
    ANOD: out std_logic_vector (7 downto 0);
    CATOD: out std_logic_vector (7 downto 0)
    );
end average;
```

CLK-clock-ul plăcii Nexys4 cu frecvența de 100Mhz

Control-selecția pe 3 biți pentru componenta DataGenerator

Length-selecția pe 3 biți pentru componenta average_computer(filter)

ANOD- semnal pe 8 biți care arată ce led al afișoruli va afișa informații

CATOD- semnal pe 8 biți folosit pentru fiecare anod

În arhitectură am declarat cele 4 componente principale.

```
17  architecture average_architecture of average is
18  component DataGenerator is
19      port (
20      CLK, RST: in std_logic;
21      Control: in std_logic_vector (2 downto 0);
22      Data1: out std_logic_vector (7 downto 0)
23      );
24  end component;
25
26  component frequency_divider is
27      port(
28      CLK: in std_logic;
29      CLK_OUT: out std_logic
30      );
31  end component ;
32
33
35  component average_computer is
36       port (
37           number: IN std_logic_vector (7 downto 0);
38           reset: IN std_logic;
39           data_clk: IN std_logic;
40           length: IN std_logic_vector (2 downto 0);
41           average: OUT std_logic_vector (7 downto 0)
42       );
43  end component ;
44
45  component DISPLAY is
46       port(CLK: in std_logic;
47       number: in std_logic_vector (7 downto 0);
48       average: in std_logic_vector (7 downto 0);
49       ANOD: out std_logic_vector(3 downto 0);
50       CATOD: out std_logic_vector(7 downto 0));
51  end component;
52
```

```
54  signal Data_internal: std_logic_vector (7 downto 0);
55  signal DATA_CLK: std_logic;
56  signal display_internal: std_logic_vector (7 downto 0);
57  signal average_internal: std_logic_vector (7 downto 0);
58  begin
59      DG: DataGenerator port map(CLK=>CLK, RST=>RST,Control=> Control,Data1=> Data_internal);
60      FDiv: frequency_divider port map(CLK=>CLK,CLK_OUT=> DATA_CLK);
61      Filter: average_computer port map(number=>Data_internal, reset=>RST,data_clk=> DATA_CLK, length=>length, average=> average_internal);
62      Displayer: DISPLAY port map(CLK=>CLK,number=> Data_internal,average=> average_internal,ANOD=> ANOD(3 downto 0), CATOD=>CATOD);
63      ANOD(7 downto 4) <= "1111";
64
65  end average_architecture;
66
```

5. Utilizare și semnificația notațiilor

Sistemul Principal

Intrări:

- **CLK:** Ceasul cu frecventa de 100 MHz de pe Nexys4.
- **CONTROL:** Selectia pe 3 biți pentru generatorul de date.
- **LENGTH:** Selectia pe 3 biți pentru media/filtrul.
- **RST:** resetare.

Ieșiri:

- **ANOD:** Semnal pe 8 biți care arată care LED al afișajului va afișa informația.
- **CATOD:** Semnal pe 8 biți folosit pentru fiecare anod pentru a reprezenta informația.

Divizor de Frecvență

Intrări:

- **CLK:** Ceasul sistemului.

Ieșiri:

- **DataClock:** Ceas cu frecvență de 1 Hz.

Generator de Date

Intrări:

- **CONTROL:** Selectie pe 3 biți pentru generatorul de date.
- **CLK:** Ceasul sistemului.
- **RST:** Resetare.

Ieșire:

- **DATA:** Flux de date pe 8 biți care va fi primit de filtru.

Pentru toate subcomponentele – generator de SquareWave, memoria unu și doi, shiftmic 0-15 și shiftmare 0-255 – avem ca intrări DataClock și ca ieșiri un semnal numit OP pe 8 biți care reprezintă ieșirea specifică pentru fiecare componentă.

Filtru

Intrări:

- **DataClock:** Ceas cu frecvență de 1 Hz.
- **LENGTH:** Select pe 3 biți pentru media/filtru.
- **RST:** Resetare.

- **NUMBER:** Date pe 8 biți primite de la generatorul de date.

Ieșire:

- **AVERAGE:** Media rezultată pe 8 biți.

Display

Intrări:

- **CLK:** Ceasul sistemului.
- **AVERAGE:** Media rezultată pe 8 biți.
- **DATA:** Flux de date pe 8 biți.

Ieșiri:

- **ANOD:** Semnal pe 4 biți deoarece folosim doar 4 anozi; acesta se va lega la semnalul ANOD pe 8 biți din modulul principal (primele 4 anode fiind dezactivate).
- **CATOD:** Semnal pe 8 biți folosit pentru fiecare anod pentru a reprezenta informația.

Divizor de Frecvență pentru Display

Intrări:

- **CLK:** Ceasul sistemului.

Ieșiri:

- **CLK_OUT:** Ceas de 200 Hz folosit pentru afișare.

Convertor

Intrări:

- **Number:** Număr pe 4 biți pentru conversie.

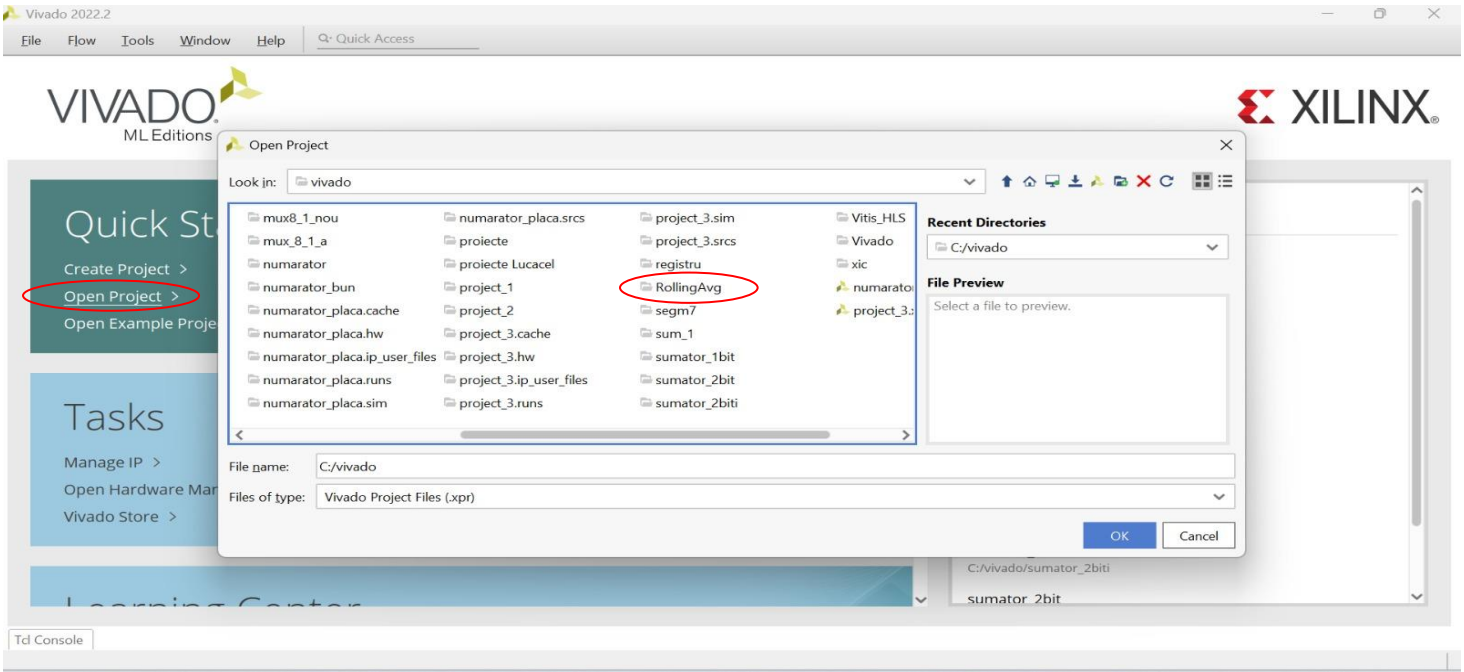
Ieșire:

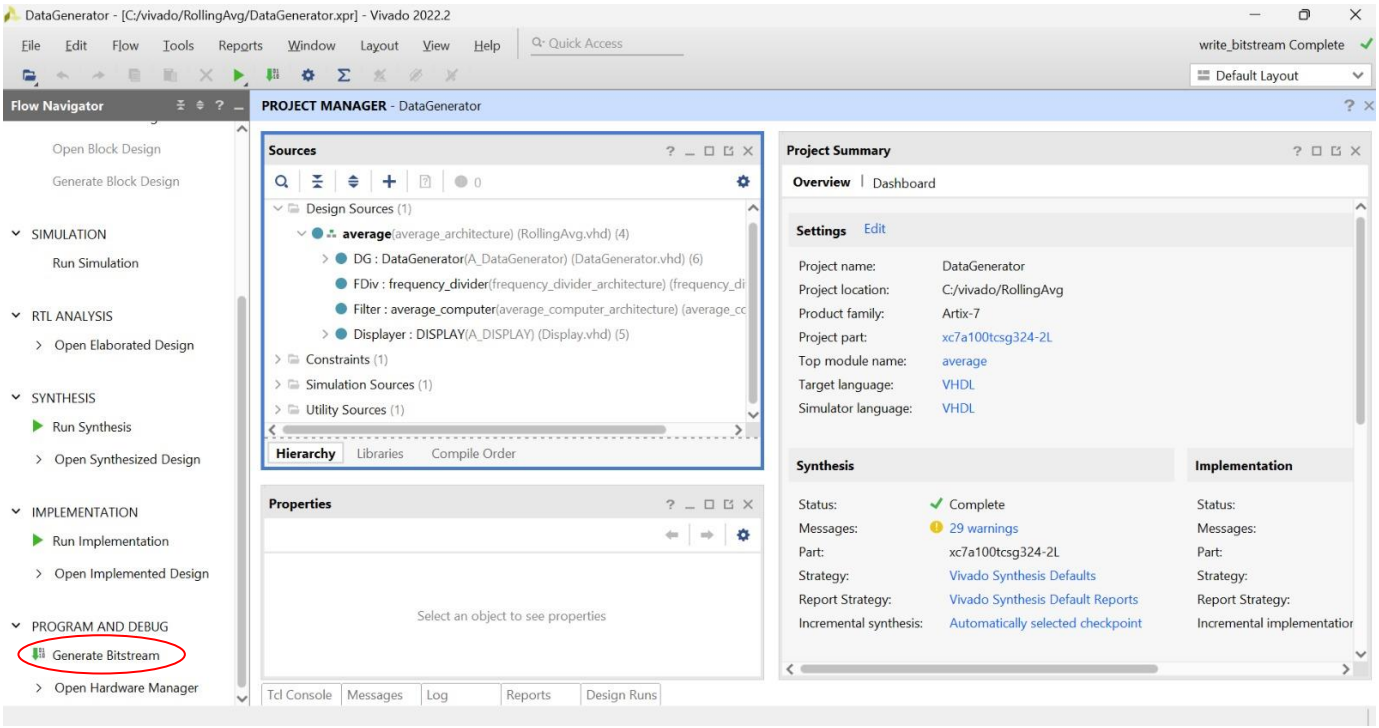
- **Converted:** Număr pe 8 biți care indică starea pe catode.

Fisier de constrangeri:

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { RST }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { Length[0] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { Length[1] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { Length[2] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { Control[0] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { Control[1] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { Control[2] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
```

```
##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { CATOD[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { CATOD[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { CATOD[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { CATOD[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { CATOD[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { CATOD[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { CATOD[6] }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { CATOD[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { ANOD[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { ANOD[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { ANOD[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { ANOD[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { ANOD[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { ANOD[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { ANOD[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { ANOD[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

##CPU Reset Button
```





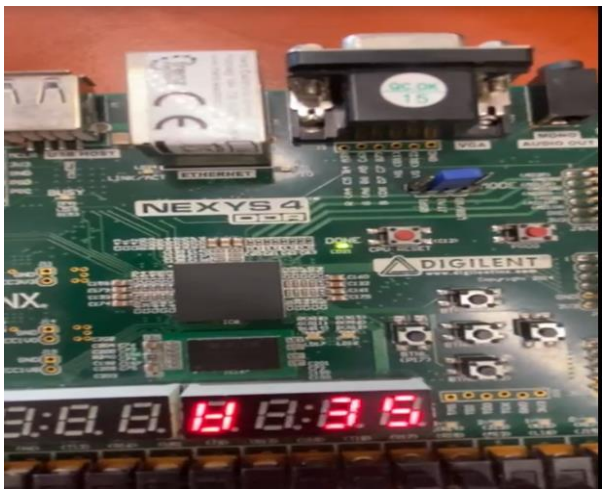
- Dupa generarea bitstream-ului, se deschide Hardware Manager-ul si se apasa optiunea de Auto Connect(placa trebuie sa fie conectata la PC prin cablu)
- Se apasa Program Device si se testeaza proiectul pe placuta,rezultatele fiind afisate pe SSD.

6. Justificarea soluției alese

Pentru rezolvarea problemei prima oară am încercat să o înțelegem printr-o abordare top-down, începând cu baza întregului sistem în minte și apoi împărțind-o în subprobleme mai mici pe care apoi am încercat să le rezolvăm. Apoi am început să implementăm totul într-o manieră bottom-up, începând de la subprobleme și ajungând la componenta mare. Proiectul nostru are astfel în total 4 componente principale.

Tehnica pe care am ales-o consta in stocharea si adaugarea ultimului imput la rezultatul curent.Totodata, se va scadea ultimul numar stocat in buffer din rezultat, iar rezultatul se va shifta cu un anumit numar de biti pentru a conferi o divizare eficienta(shiftare cu 1 bit= impartire cu 2,etc.)

7. Rezultate



8. Oportunități de dezvoltare

O îmbunătățire ar putea fi să extindem varietatea opțiunilor din care utilizatorul poate să aleagă. De exemplu am putea avea numere mai mari decât numerele pe 8 biți. În acest fel am putea extinde sistemul de calcul la numere mai mari decât 255, mărindu-i aplicabilitatea.