MINISTERUL EDUCAȚIEI NAȚIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Dispozitiv de calcul al mediei unui set de numere

Studenți:
Horvath Andrea - Anett
Mihai Cristina - Mădălina

Profesor coordonator:
Prof. dr. ing. Octavian Creț

Facultatea de Automatică și Calculatoare
Specializarea Calculatoare și Tehnologia Informației
Anul I, Secția Engleză, Grupa 30414

Cuprins:

1. Specificația proiectului

- **Obiectivul proiectulu**i este dezvoltarea unui **sistem simplu de procesare a semnalelor** care va calcula **media unui flux paralel de date pe 8 biți** ca un exercițiu de proiectare a sistemelor numerice.
- Proiectul va fi implementat pe o **plăcuță Nexys4 DDR** Rev C board pentru a permite demonstrarea unui sistem funcțional.
- Sistemul va fi dezvoltat ca un **model VHDL** folosind **Xilinxs ISE WebPack 14.7** care include Modelisim Simulation Tools pentru verificarea designului.

Tema:

În cadrul sistemelor de procesare a semnalului este adesea nevoie să se calculeze valoarea medie numerică pentru un flux de date de intrare. Acesta implementează un filtru uniformizând schimbările rapide ale fluxului de date. Sistemul de filtrare va rula în "timp real" și va produce valoarea medie la aceeași rată ca și datele originale de intrare.

Cerința este de a dezvolta un model VHDL pentru sisteme de filtrare digitale combinat cu un generator de flux de date. Comutatoarele (switches), butoanele (buttons) și afisorul pe 7 segmente (display) aflate pe plăcuță Nexys4 DDR vor trebui incluse pentru a demonstra operația corectă.

Scopul lucrării este realizarea unui sistem funcțional care să îndeplinească specificațiile cu minimum de complexitate și resurse.

2. Schema bloc cu componentele principale

Schema bloc (Fig. 1) este împărțită în două componente majore: Generator de date (generator) și Filtru de date (filter), ambele legate la afișorul plăcuței (display) (Fig. 2).

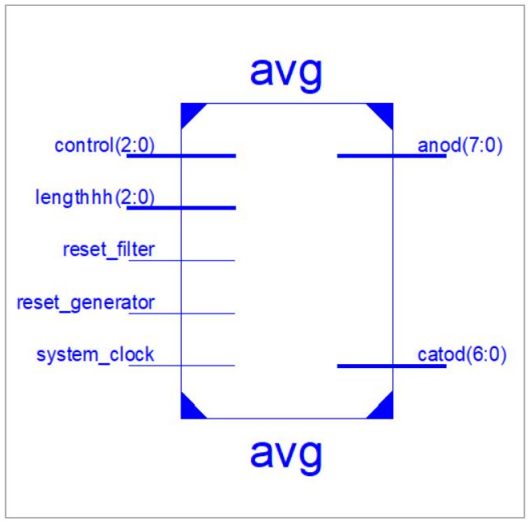

Fig. 1 *Schema bloc*

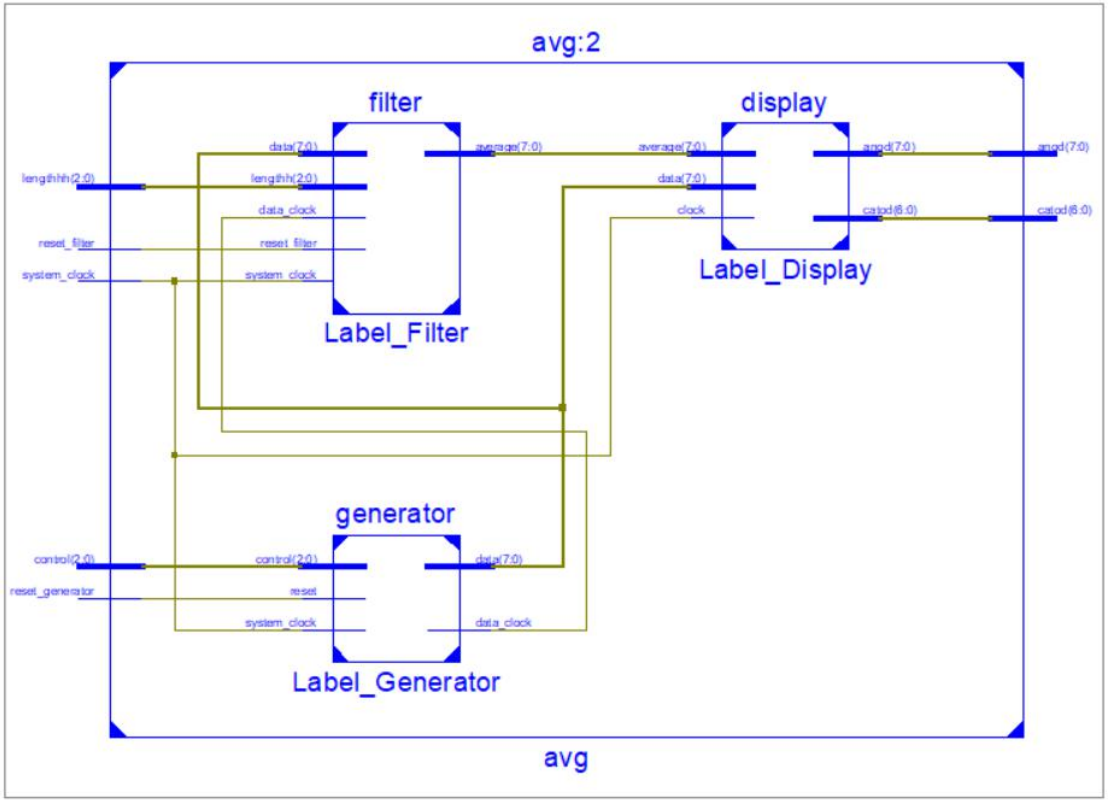

Fig. 2 *Componentele principale ale schemei bloc*

3. Etapele de proiectare

3.1. Componentele MSI utilizate

3.1.1. Numărător binar direct pe 27 de biți

Numărătoarele sunt circuite logice secvențiale (CLS) care contorizează numărul de impulsuri de tact (system_clock) aplicate la intrare.

Acest numărător (Fig. 3) s-a realizat prin conectarea a 27 de celule de memorie pentru a obține 2^{27} stări distincte. Este o componentă utilizată cu scopul divizării semnalului de tact intern al plăcuței suport.

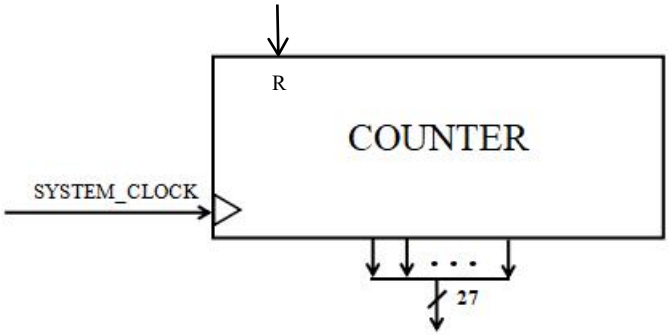

Fig. 3 *Numărător binar direct pe 27 de biți*

3.1.2. Comparator cu calea de date pe 27 de biți

Comparatoarele numerice sunt circuite logice combinaționale (CLC) care permit determinarea valorii relative a două numere binare.

Comparatorul prezent (Fig. 4) va fi utilizat cu scopul detectării numărului 49 999 999. De asemenea, este o componentă a divizorului de frecvență ce va fi prezentat ulterior.

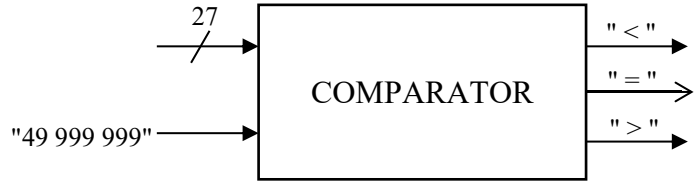

Fig. 4 *Comparator cu calea de date pe 27 de biți*

3.1.3. Numărător binar direct pe 3 biți

Acest numărător (Fig. 5) s-a realizat prin conectarea a 3 celule de memorie pentru a obține 2^3 stări distincte. Este o componentă utilizată cu scopul divizării semnalului de 1Hz (CLOCK). Funcționarea sa este detaliată în Tabelul 1:

Starea curentă (zecimal)	Starea curentă (binar)	Starea următoare (zecimal)	Starea următoare (binar)	CLOCK_OUT
0	000	1	001	0
1	001	2	010	0
2	010	3	011	0
3	011	4	100	0
4	100	5	101	1
5	101	6	110	1
6	110	7	111	1
7	111	0	000	1

Tabelul 1

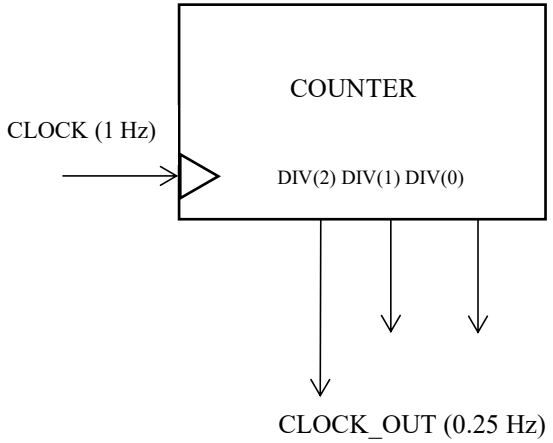

Fig. 5 *Numărător binar direct pe 3 biți*

3.1.4. Multiplexor 8:1 cu calea de date pe 1 bit

Multiplexoarele sunt CLC-uri care permit trecerea datelor de pe una din intrări la o ieșire unică. Selecția intrării se face printr-un cuvânt de cod de selecție, numită și adresă. Există deci 2 tipuri de intrări: intrări de date și intrări de selecție, și o singură ieșire.

Multiplexor din Fig. 6 are rolul de a alege unul dintre cele două clock-uri divizate. Conform switch-urilor de control (C2, C1, C0), impulsul de tact final (DATA_CLOCK) va lua valoarea clock-ului de 1 Hz (CLOCK), respectiv 0.25 Hz (CLOCK_SW).

Ieșirea multiplexorului este prezentată explicit în Tabelul 2:

C2	C1	C0	DATA_CLOCK
0	0	0	CLOCK
0	0	1	CLOCK_SW
0	1	0	CLOCK
0	1	1	CLOCK
1	0	0	CLOCK
1	0	1	CLOCK
1	1	0	CLOCK
1	1	1	CLOCK

Tabelul 2

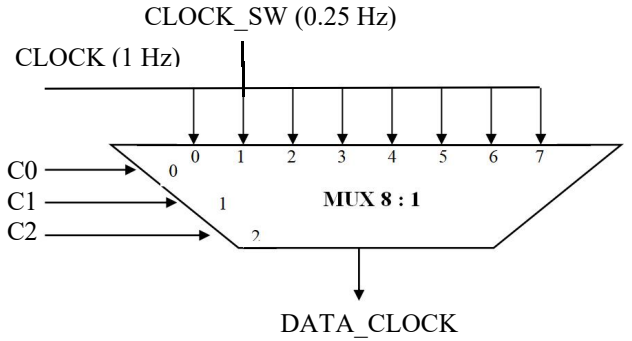

Fig. 6 *Multiplexor 8:1 cu calea de date pe 1 bit*

3.1.5. Registru de memorie

Registrele sunt CLS-uri care permit stocarea și/sau deplasarea informației codificate binar. Registrele de memorie memorează informația binară în celule de memorie binară.

Se vor folosi 6 astfel de registre (Fig. 6) pentru implementarea blocului generator de date pentru Student One și Student Two. Registrele vor fi înlănțuite și legate circular.

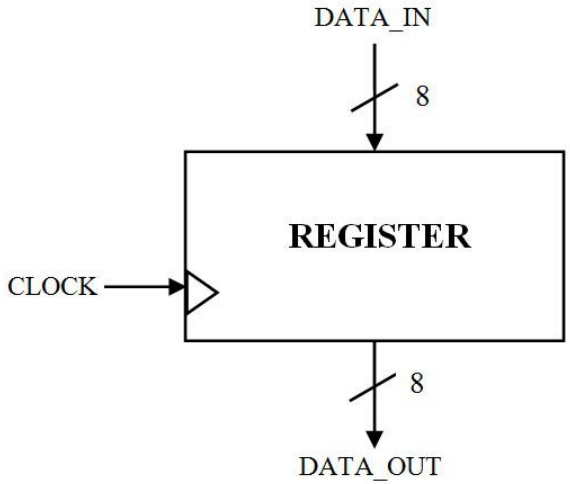

Fig. 7 *Registru de memorie*

3.1.6. Multiplexor 8:1 cu calea de date pe 8 biti

Multiplexorul din Fig. 8 va fi utilizat cu scopul selecției uneia dintre metodele de generare a fluxului de date.

Selecția multiplexorului este dată de switch-urile de control (C2, C1, C0), iar intrările sunt reprezentate de numere pe 8 biți generate de elementele următoare:

-SW (square wave);
-Stud1 (Secvență ciclică de 6 cifre “Studentul nr. 1”);
-Stud2 (Secvență ciclică de 6 cifre “Studentul nr. 2”);
-Prg1 (Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15]));
-Prg2 (Generator de secvențe pseudo-aleatoare pe 8 biți (interval [0;255])).

Ieșirea multiplexorului este prezentată în Tabelul 3:

C2	C1	C0	Data
0	0	0	"00000000"
0	0	1	Square wave
0	1	0	Student 1
0	1	1	Student 2
1	0	0	"00000000"
1	0	1	"00000000"
1	1	0	PRG 1
1	1	1	PRG 2

Tabelul 3

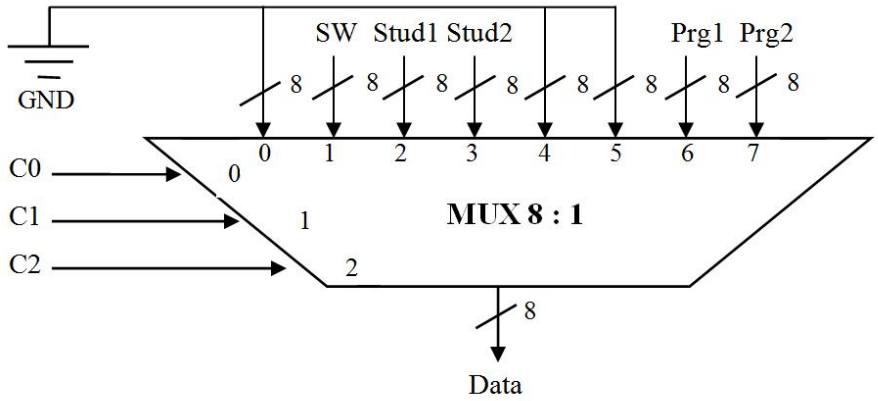

Fig. 8 *Multiplexor 8:1 cu calea de date pe 8 biti*

3.1.7. Sumator pe 8/9/10/11 biți

Sumatoarele sunt CLC-uri care realizează adunarea numerelor binare. Pentru proiectul curent, este necesară utilizarea sumatoarelor pe 8, 9, 10, respectiv 11 biți. În Fig. 9 se prezintă schema unui sumator pe n biți, unde n poate fi înlocuit oricare dintre variantele menționate. De menționat că bitul de trasport este concatenat cu suma finală.

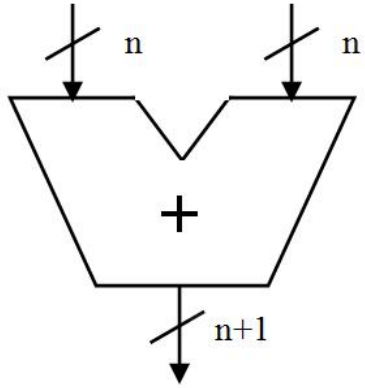

Fig. 9 *Sumator pe n biți*

3.1.8. Multiplexor 8:1 cu calea de date pe 12 biți

Multiplexorul din Fig. 10 va fi utilizat cu scopul selecției uneia dintre sume efectuate pentru 2, 4, 8 sau 16 numere. Selecția și ieșirea multiplexorului sunt prezentate explicit în Tabelul 4.

L2	L1	L0	Data
0	0	0	"000000000000"
0	0	1	"000000000000"
0	1	0	"000000000000"
0	1	1	"000000000000"
1	0	0	"000"&Sum2
1	0	1	"00"&Sum4
1	1	0	"0"&Sum8
1	1	1	Sum16

Tabelul 4

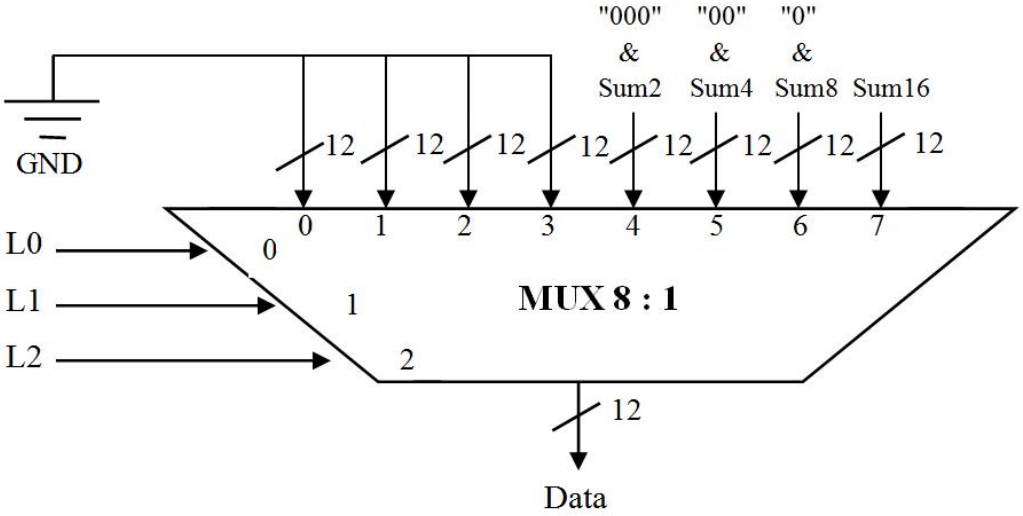

Fig. 10 *MUX 8:1 cu calea de date pe 12 biti*

3.2. Blocuri principale

3.2.1. Divizor de frecvență

Pentru realizarea divizorului de frecvență (Fig. 11) am folosit un numărător binar direct pe 27 de biți, un comparator cu calea de date pe 27 de biți și un “T” Flip-Flop. Numărătorul (COUNTER) primește ca impuls clock-ul intern al plăcuței (SYSTEM_CLOCK) cu o frecvență de 100MHz. Comparatorul primește ca intrări ieșirea numărătorului și numărul “49 999 999” (care se reprezintă tot pe 27 de biți). În momentul în care comparatorul indică o egalitate (“=”), bistabilul “T” primește un semnal de tact și, simultan, numărătorul va fi resetat. Astfel, numărătorul devine un modulo 49 999 999. Intrarea bistabilului este legată la GND astfel că, la fiecare semnal primit, ieșirea (CLOCK) se va schimba în ‘1’. La următorul impuls primit, ieșirea reia valoarea ‘0’ și ciclul se repetă. Astfel, frecvența CLOCK-ului va fi 1 Hz.

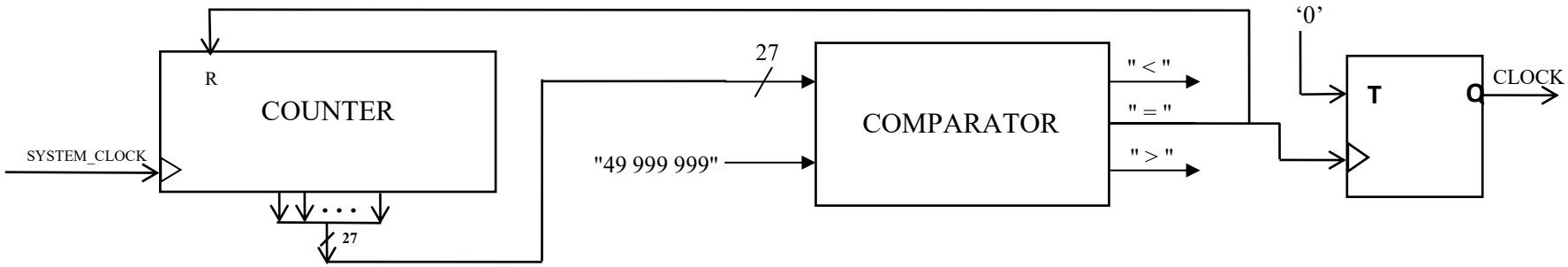

Fig. 11 *Divizor de frecvență*

3.2.2. Square Wave

Această componentă (Fig. 12) generează o secvență de 8 numere la o frecvență de 0.25 Hz prin rotația unei secvențe de 8 biți. În vederea obținerii unui nou semnal cu frecvență de 0.25 x 1 Hz, am utilizat un numărător binar direct pe 3 biți. Numărătorul primește ca impuls un semnal de tact cu frecvența de 1 Hz. Cel mai semnificativ bit de ieșire va reprezenta semnalul (CLOCK_OUT) cu noua frecvență (0.25 Hz).

Se utilizează 8 bistabile "D" cu scopul memorării și rotirii a 8 biți, care au fost inițializate anterior. La fiecare semnal de tact, conținutul bistabilelor se va roti și va genera un nou număr (S(7) - S(0)).

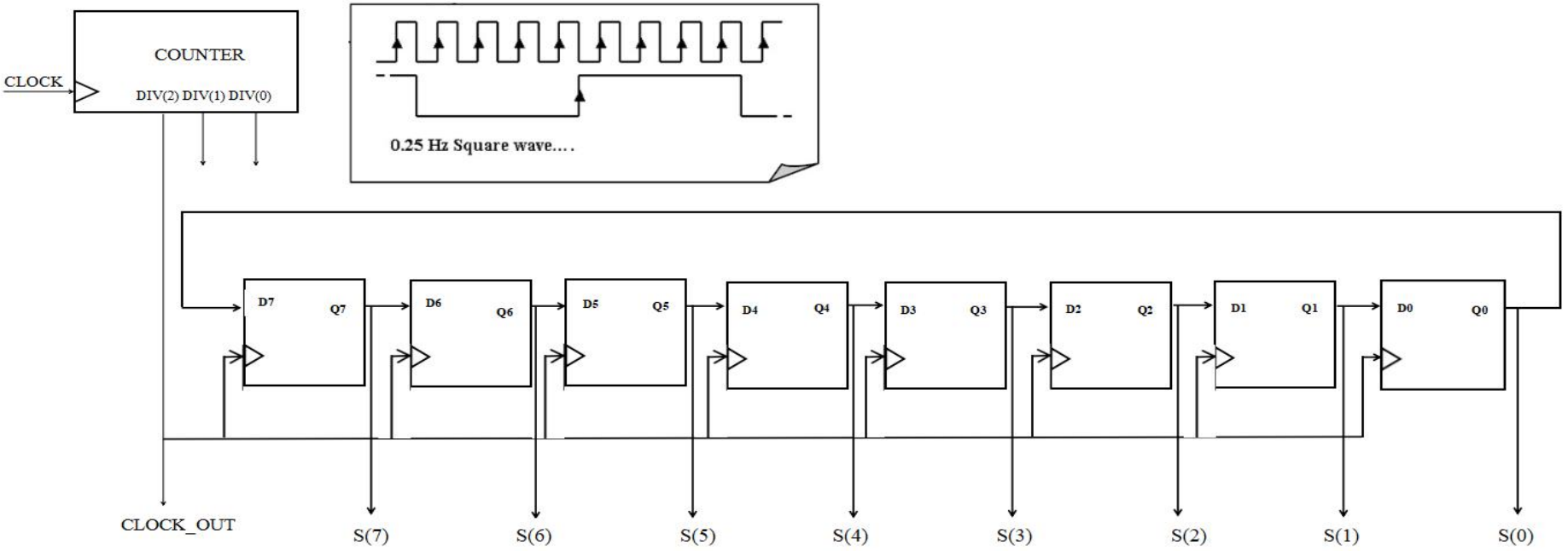

Fig. 12 *Square Wave*

3.2.3. Secvență ciclică de 6 cifre “Studentul nr. 1” și “Studentul nr. 2”

Această componentă înglobează o secvență de 6 registre de memorie conectate la același semnal de tact (CLOCK) cu frecvență de 1 Hz. Registrele sunt inițializate cu valorile alese de “Studentul nr. 1”, respectiv “Studentul nr. 2” și au același regim de funcționare. La fiecare semnal de tact, se generează un număr pe 8 biți (S(0)) și are loc rotația la dreapta a informației binare dintr-un registru în următorul.

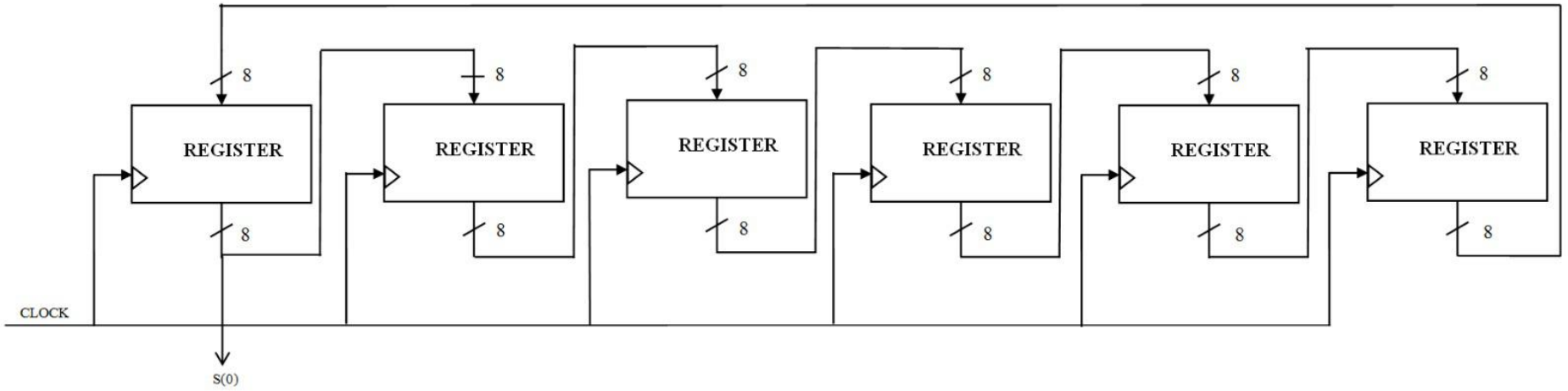

Fig. 13 *Secvență ciclică de 6 cifre*

3.2.4. Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15])

Fig. 14 prezintă un registru de deplasare cu feedback linear (Linear Feedback Shift Register). Pentru această componentă se folosesc 8 bistabile tip “D” și o poartă XOR. Intrările primelor 4 bistabile sunt legate la GND initializand primii 4 biți ai numărului generat cu “0000” forțând intervalul [0;15]. Prin alegerea corectă a intrărilor porții XOR, putem obține PRBS (Pseudo-Random Binary Sequence) de lungime maximă , care include toate combinațiile posibile, excluzând rezultatul “0000” (S(3)-S(0)). Numerele sunt generate la o frecvență de 1 Hz.

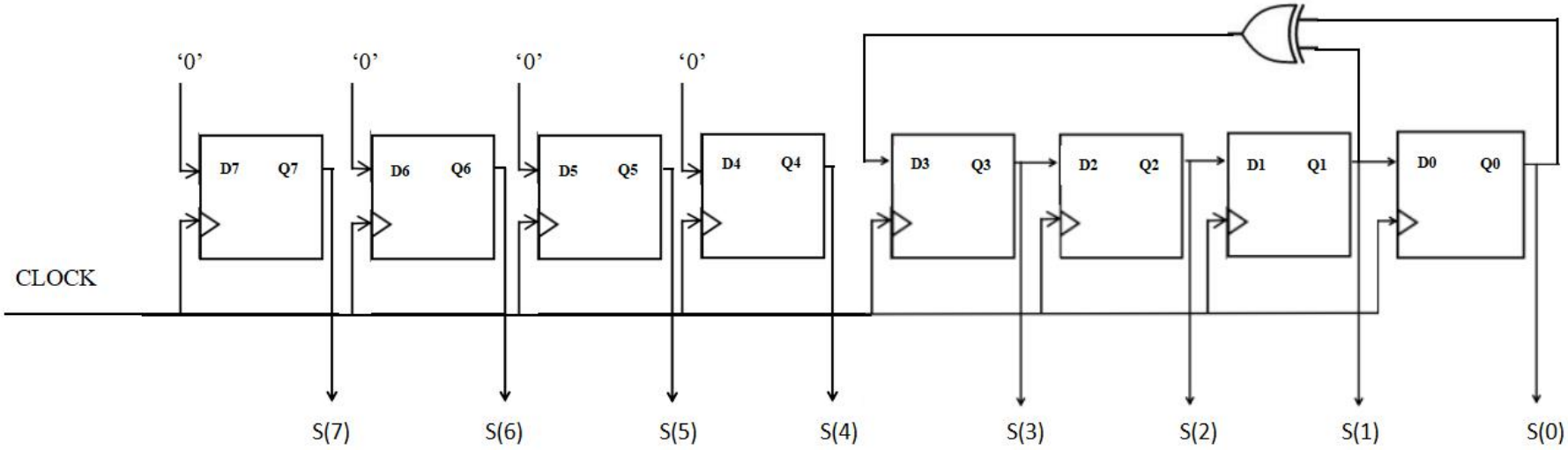

Fig. 14 *Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15])*

3.2.5 Generator de secvențe pseudo-aleatoare pe 8 biți (interval [0;255])

Fig. 15 prezintă un registru de deplasare cu feedback linear (Linear Feedback Shift Register). Pentru această componentă se folosesc 8 bistabile tip "D" și 3 porți XOR. Prin alegerea corectă a intrărilor porților XOR, putem obține PRBS (Pseudo-Random Binary Sequence) de lungime maximă $2^n - 1$, care include toate combinațiile posibile, excluzând rezultatul "00000000" (S(7)-S(0)). Numerele sunt generate la o frecvență de 1 Hz.

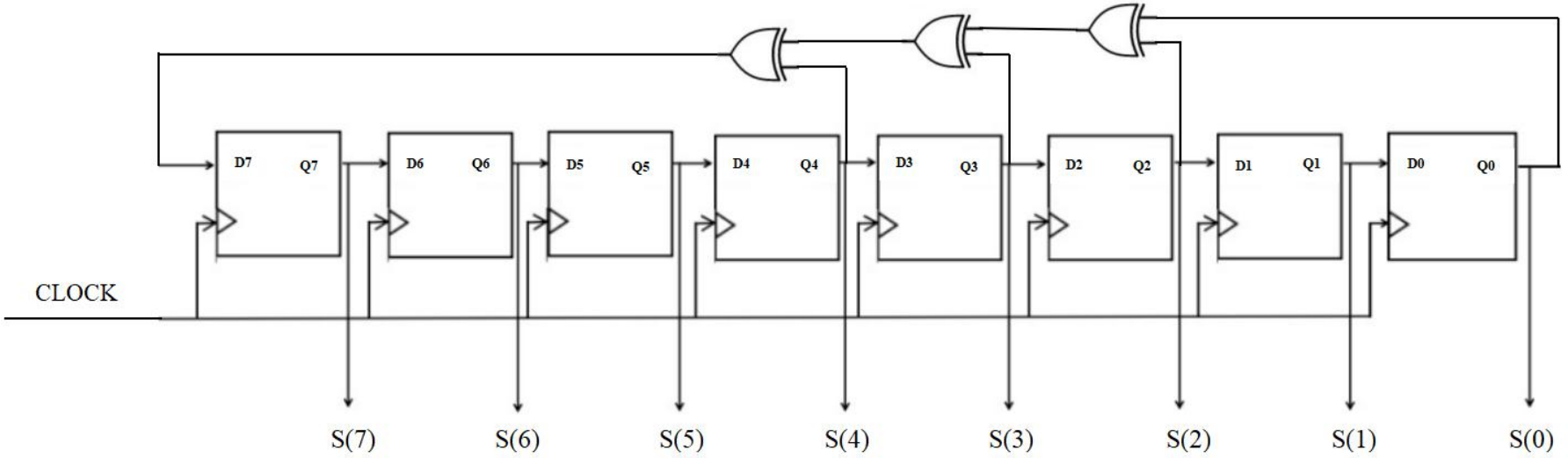

Fig. 15 *Generator de secvente pseudo-aleatoare pe 8 biți (interval [0;255])*

3.2.6. Generator

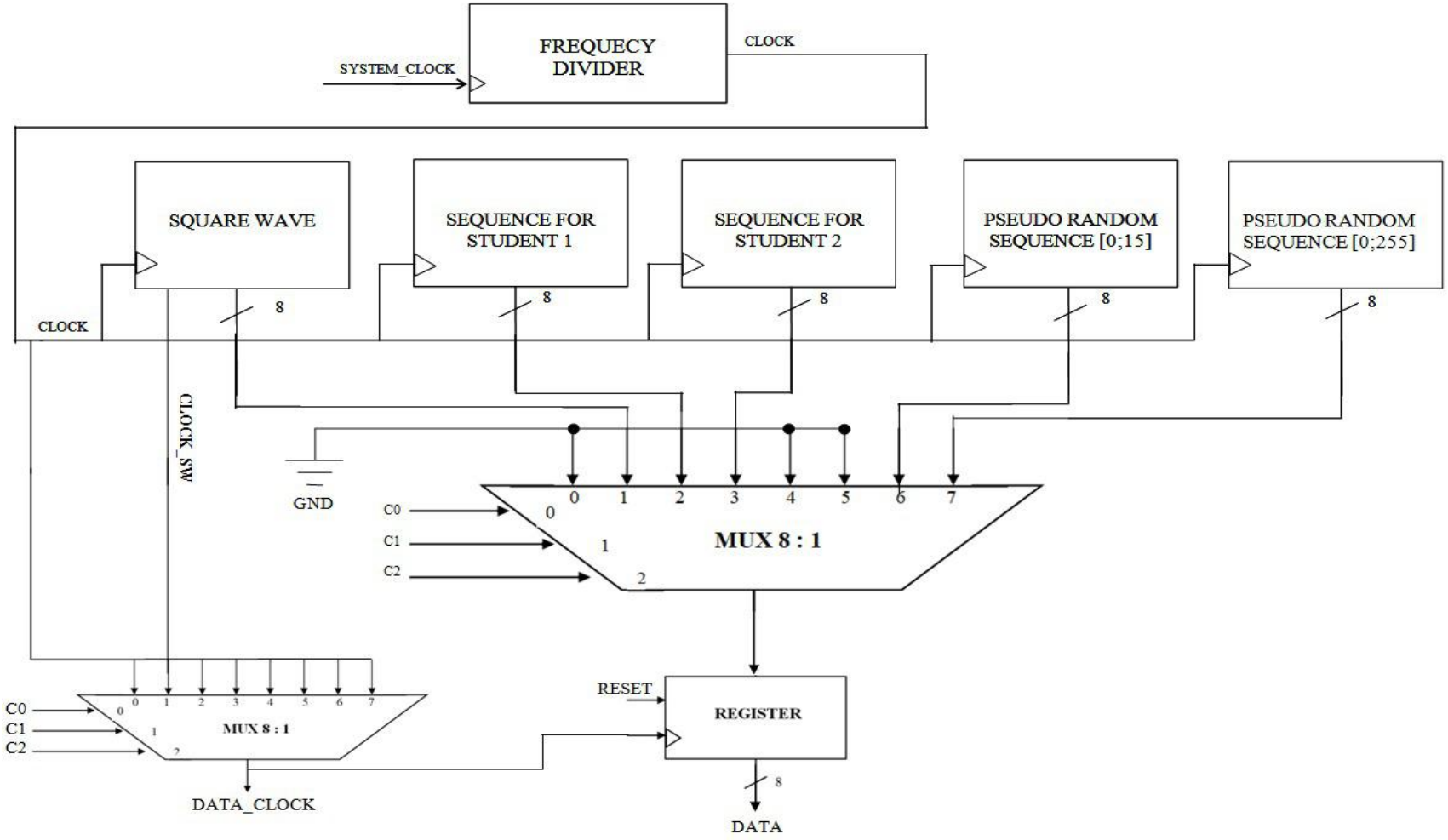

Fig. 16 *Generator - schemă detaliată*

Componenta rezentată (Fig. 16) reprezintă generatorul de numere și cuprinde :
- divizor de frecvență (3.2.1);
- Square wave (3.2.2);
- Secvența ciclică de 6 cifre “Studentul nr. 1” și “Studentul nr. 2” (3.2.3);
- Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15]) (3.2.4);
- Generator de secvențe pseudo-aleatoare pe 8 biți (interval [0;255]) (3.2.5);
- Multiplexor 8:1 cu calea de date pe 8 biți (3.1.6);
- Multiplexor 8:1 cu calea de date pe 1 bit (3.1.4).

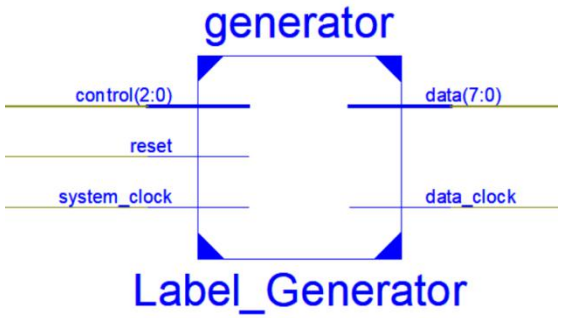

Fig. 17 *Generator - schema bloc*

Componenta primește pe intrare că semnal de tact clock-ul intern al plăcuței (SYSTEM_CLOCK) cu o frecvență de 100 Mhz. Acesta intră în divizorul de frecvență, generandu-se un nou semnal de tact (CLOCK) cu o frecvență de 1 Hz. Toate componentele generatoare de numere (Square Wave, Student 1, Student 2, PRG1, PRG2) vor funcționa pe același semnal de tact de 1 Hz. În funcție de switch-urile de control (C2, C1, C0), ieșirea multiplexorului (Tabelul 3) este stocată într-un registru de memorie și va fi conform Tabelului 5.

Generator de numere (Control)	**Numere generate**
Square Wave (“001”)	**216, 108, 54, 27, 141, 198, 99, 177**, 216, 108, 54 …
Student 1 (“010”)	**8, 6, 1, 0, 4, 5**, 8, 6, 1, 0, 4, 5 …
Student 2 (“011”)	**9, 7, 1, 4, 2, 3**, 9, 7, 1, 4, 2, 3 …
Pseudo-aleator [0;15] (“110”)	**13, 10, 5, 11, 7, 15, 14, 12, 8, 1, 2, 4, 9, 3, 6**, 13, 10, 5 …
Pseudo-aleatro [0;255] (“111”)	**45, 90, 180, 105, 210, 164, 72, 145, 34, 69, 138, 20, 41, 82** ...

Tabelul 5

Ieșirea DATA_CLOCK este asociată ieșirii multiplexorului cu calea de date pe 1 bit conform Tabelului 2 din 3.1.3. Registrul de memorie permite resetarea generatorului prin intrarea RESET. Ieșirea acestuia este asociată ieșirii DATA a generatorului (Fig. 17).

3.2.7. Filtru

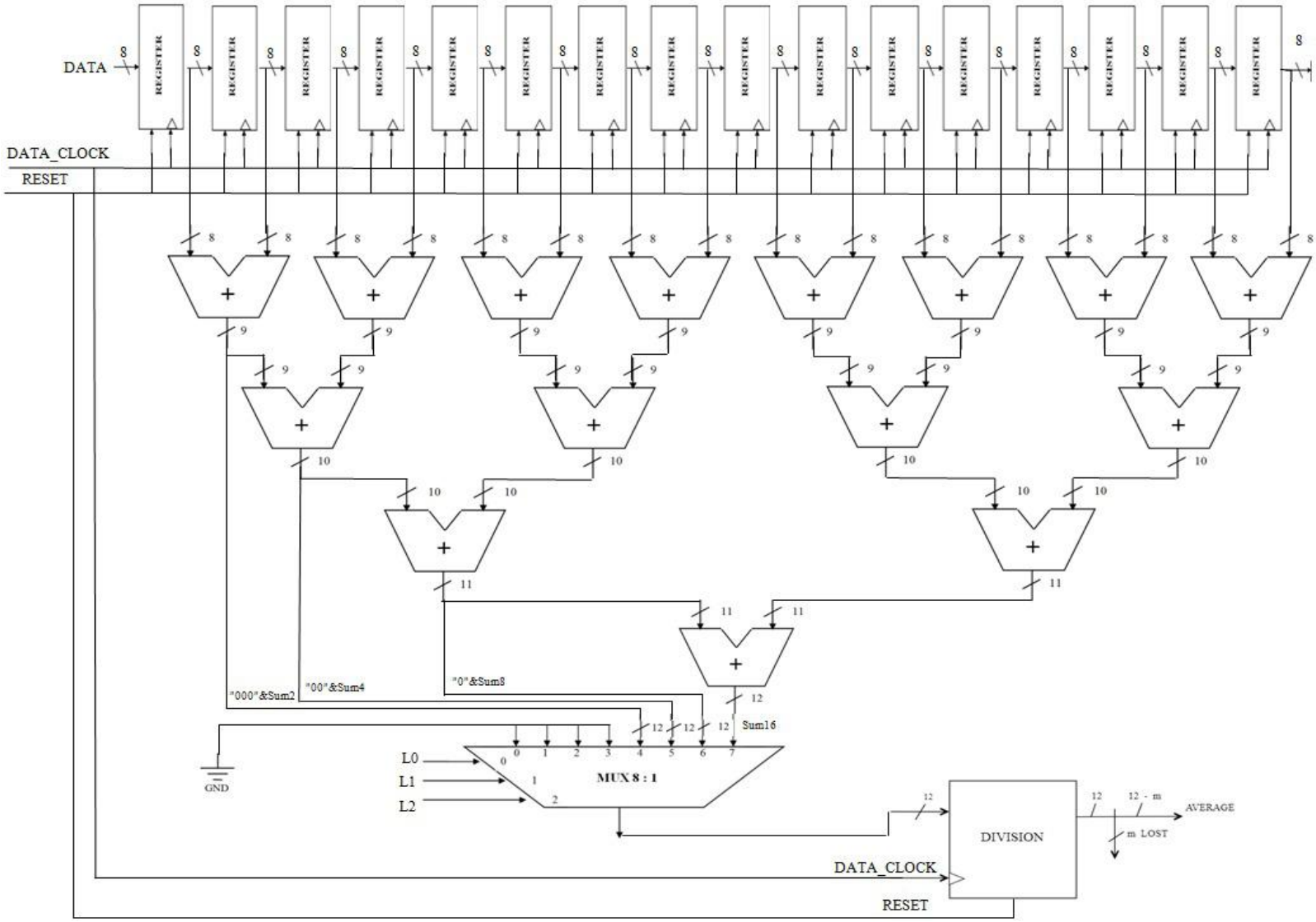

Fig. 18 *Filtru - schemă detaliată*

Componenta prezentată în Fig. 18 reprezintă filtrul și cuprinde:
- 16 registre de memorie;
- 8 sumatoare pe 8 biți;
- 4 sumatoare pe 9 biți;
- 2 sumatoare pe 10 biți;
- 1 sumator pe 11 biți;
- un MUX 8:1 cu calea de date pe 12 biți;
- un registru (DIVISION).

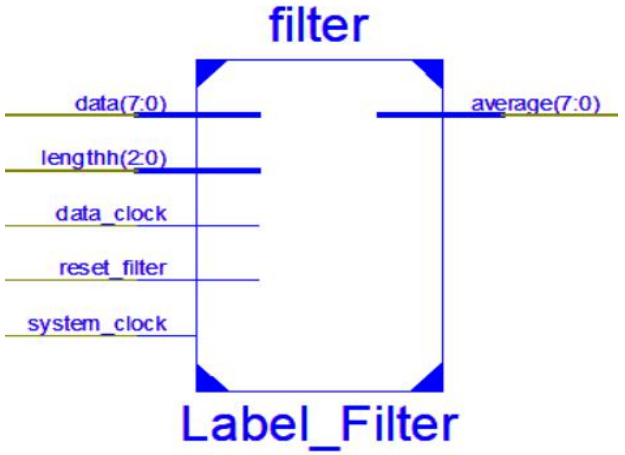

Fig. 19 *Filtru - schema bloc*

La fiecare impuls de tact (DATA_CLOCK), componenta primește pe intrarea DATA o secvență de 8 biți care se deplasează spre dreapta dintr-un registru în altul. Secvențele de biți ce ies din cele 16 registre intră în sumatoarele pe 8 biți două câte două. Numerele generate se adună două câte două până se ajunge un număr pe 12 biți. Rezultatele primelor sumatoare din fiecare tip vor servi ca intrări pentru multiplexorul 8:1 (Fig. 18). Ieșirea acestuia este detaliată în Tabelul 4 (3.1.8).

Registrul DIVISION stochează suma aleasă și are posibilitatea de RESET. Ieșirea registrului pierde m biți conform tabelului 6, obținându-se astfel media aritmetică (AVERAGE).

LENGTH (L2, L1, L0)	**m**
“100” (2 SAMPLES)	1
“101” (4 SAMPLES)	2
“110” (8 SAMPLES)	3
“111” (16 SAMPLES)	4

Tabel 6

3.2.8. Display

Pentru afișorul 7 segmente se va implenta inițial un divizor de frecvență (Fig. 20). Pentru realizarea acestuia am folosit un numărător binar direct pe 16 biți, un comparator cu calea de date pe 16 de biți și un “T” Flip-Flop. Numărătorul (COUNTER) primește ca impuls clock-ul intern al plăcuței (SYSTEM_CLOCK) cu o frecvență de 100MHz. Comparatorul primește ca intrări ieșirea numărătorului și numărul “25 000” (care se reprezintă tot pe 16 de biți). În momentul în care comparatorul indică o egalitate (“=”), bistabilul “T” primește un semnal de tact și, simultan, numărătorul va fi resetat. Astfel, numărătorul devine un modulo 25 000. Intrarea bistabilului este legată la GND astfel că, la fiecare semnal primit, ieșirea (CLOCK_DISPLAY) se va schimba în ‘1’. La următorul impuls primit, ieșirea reia valoarea ‘0’ și ciclul se repetă. Astfel, frecvența CLOCK_DISPLAY-ului va fi 2 000 Hz.

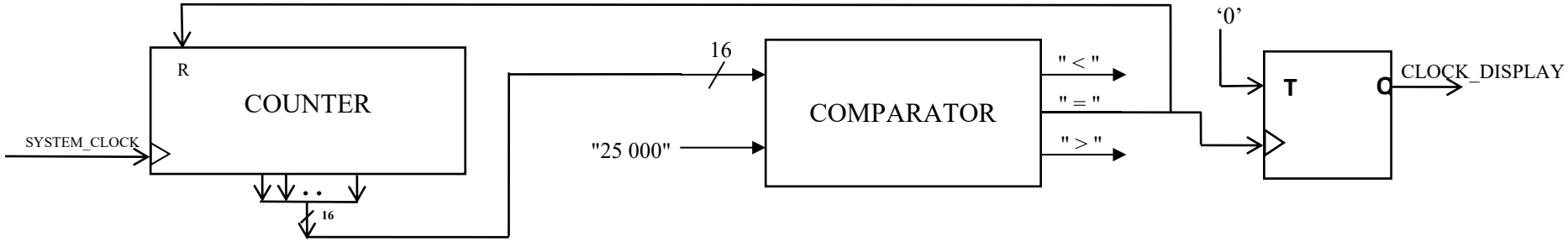

Fig. 20 *Divizor de frecvență*

În continuare, vom lucra cu noul impuls de tact pentru activarea secvențială a anozilor plăcuței. Acest lucru se realizează prin rotirea secvenței “011111” între bistabilele D6, D5, D4, D2, D1 și D0 (Fig. 21). Bistabilele D7 și D3 sunt legate la VCC și vor fi inactivi pe tot parcursul prezentării, întrucât anozii sunt activi pe ‘0’. Ieșirile bistabilelor vor reprezenta selecțiile multiplexorului 64:1 cu calea de date pe 7 biți. În funcție de anodul activ, ieșirea multiplexorului va lua valoarea cifrei codificată în “catod” conform Tabelului 7.

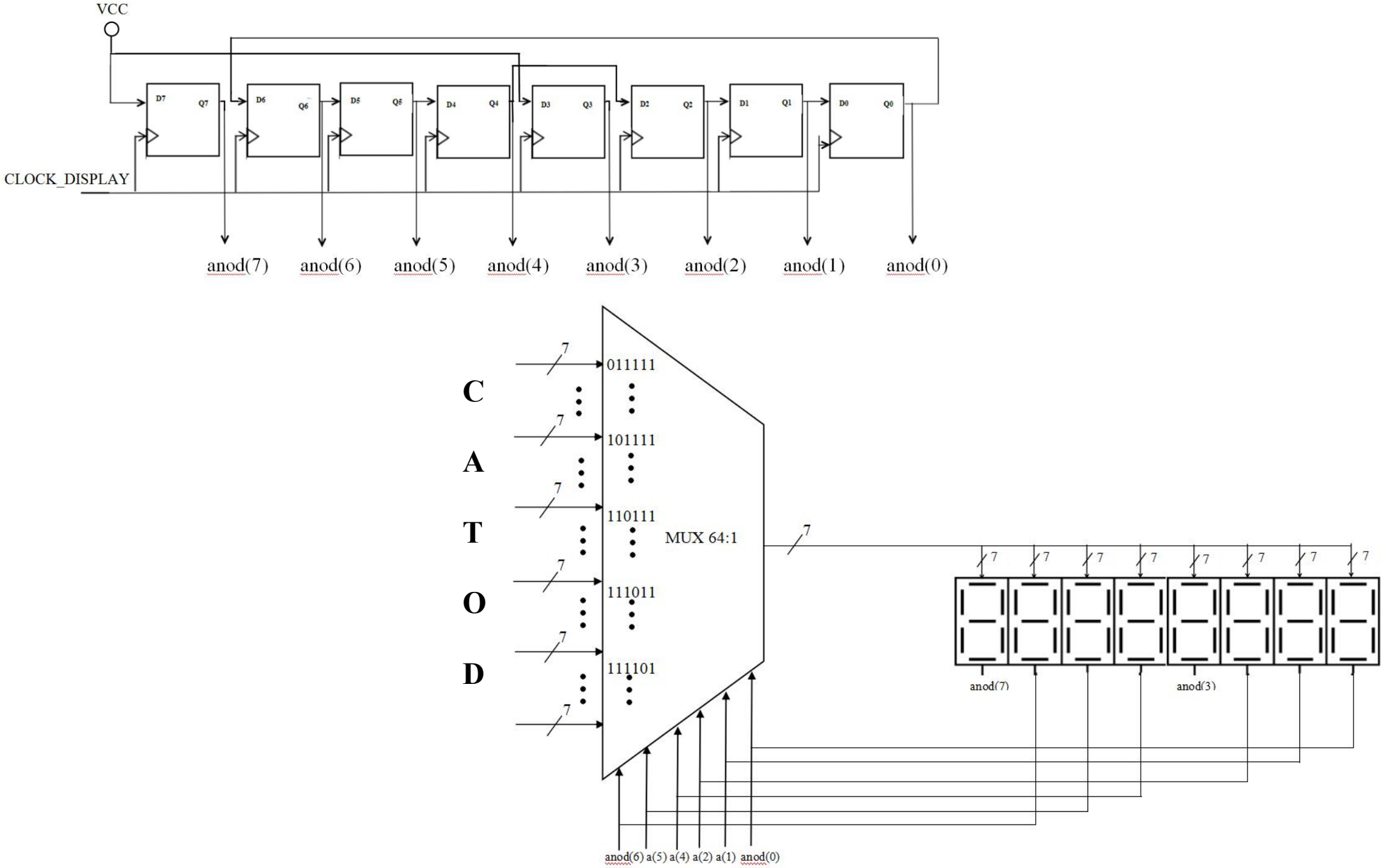

Fig. 21 *Afișorul 7 segmente*

CIFRA	CATOD						
	a	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
F	0	1	1	1	0	0	0

Tabel 7

4. Implementarea proiectului pe plăcuța FPGA

Se realizează corespondența dintre schema proiectului (Fig. 22) și plăcuța suport (Fig. 23) conform Tabelului 8.

I/O SCHEMĂ PROIECT	COMPONENTA ȘI PINII CORESPUNZĂTORI PLĂCUȚEI FPGA
RESET (GENERATOR)	BUTON “P17”
RESET (FILTER)	BUTON “M17”
SYSTEM CLOCK (100 MHz)	PIN “E3”
CONTROL SWICHTES	SWITCH-control ‘V10 U11 U12”
LENGTH SWITCHES	SWITCH-length “M13 L16 J15”
DATA	AN(6), AN(5), AN(4) “P14 T14 K2”
AVERAGE	AN(2), AN(1), AN(0) “T9 J18 J17”

Tabel 8

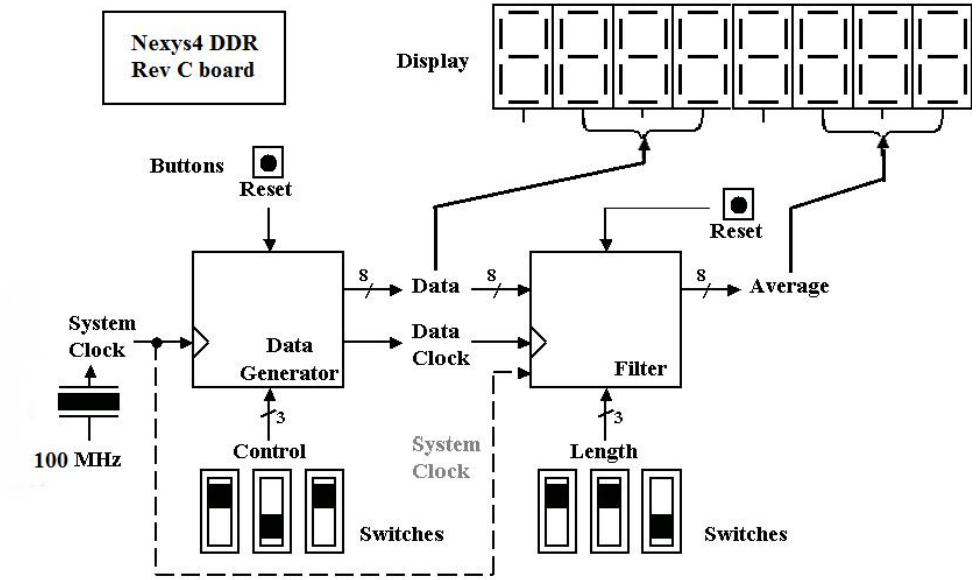

Fig. 22 *Schemă proiect*

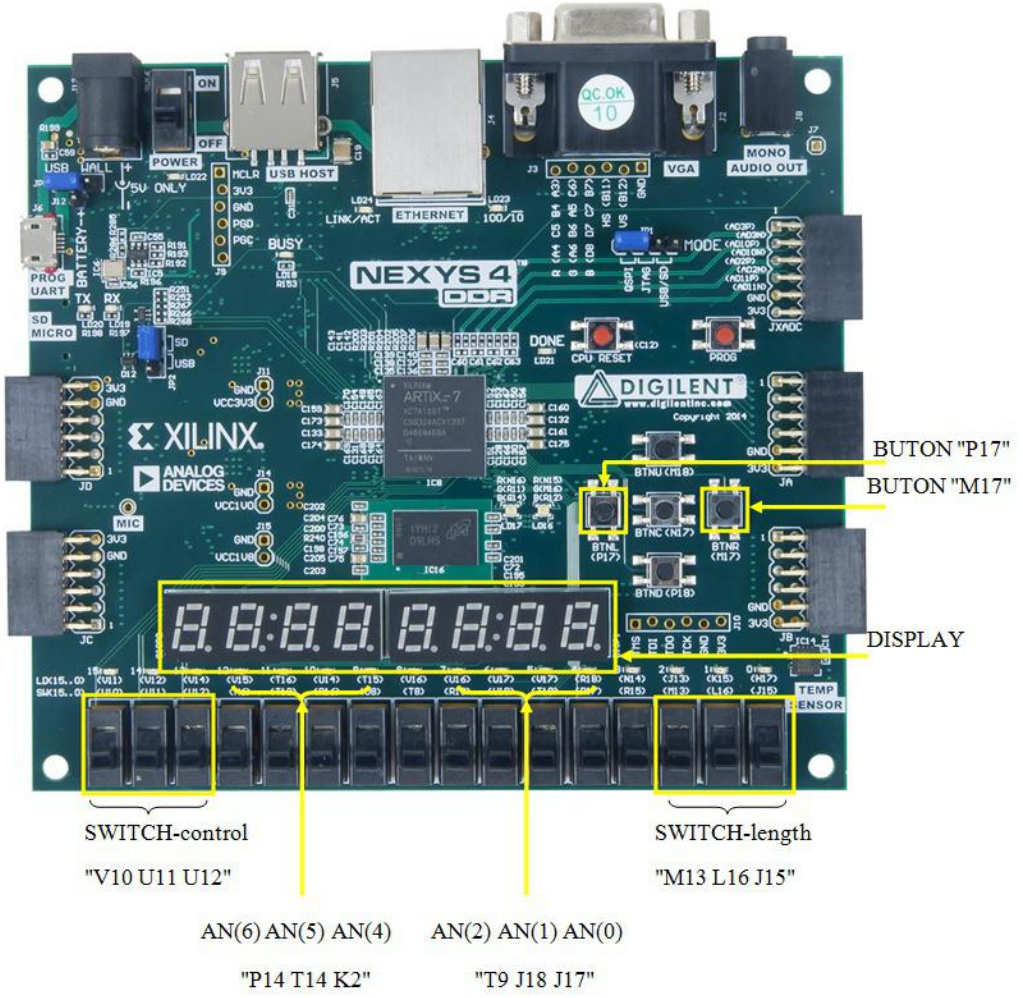

Fig. 23 *Plăcuța FPGA suport (Nexys4 DDR)*

5. Justificarea soluției alese

Cerința a fost abordată în manieră “bottom-up” implementând inițial componente idividuale MSI, urmând să le legăm treptat între ele. Astfel, se poate urmări și înțelege funcționalitatea fiecărei componente în parte, dar și în cadrul design-ului final.

Toate componentele sunt legate prin descriere structurală în blocul final, lucru convenabil pentru posibililele imbunatiri ulterioare. Am recurs la o implementare simplă și ușor de înțeles de orice utilizator interesat de funcționarea proiectului.

6. Instrucțiuni de utilizare

6.1. Utilizarea în Active - HDL

Pentru acest proiect este nevoie de mediul de proiectare **Active-HDL 6.3**. Se va urmări următoarea secvență de pași:

1. Se intră în meniul *File Explorer*.
2. Se deschid următoarele foldere în ordinea data:
- Local Disk (C:);
- My_Designs;
- *projectt*;
- *project*;
- *project.aws*.
3. Odată încărcat, proiectul va trebui compitlat și simulat. Compilarea a fost realizată anterior. Pentru simulare se creează un *Waveform* utilizând butonul *New Waveform* din bara de meniu. Se selectează fișierul *top-level "avg"* care se dorește a fi simulat. Se selectează opțiunea *Initialize simulation* din meniul *Simulation*. Se vor adăuga semnalele prin combinația de taste Ctrl + I. Se selectează toate semnalele prezente și se apasă *Add*. Se asociază stimuli fiecărui semnal de input prin click drepta și comanda *Simulators*. Pentru rularea simulării se apasă butonul *Run For (F5)*. Simulatorul va rula un interval de timp selectat din lista de lângă buton. Se vor selecta 100 de secunde. Dacă se dorește resetarea simulării, se va selecta *Restart Simulation* din meniul *Simulation*.

6.2. Utilizarea în ISE Design Suite și pe plăcuța FPGA

Pentru acest proiect este nevoie de programul **ISE Design Suite 14.7**. Se urmăresc pașii:

1. Se intră în meniul *File Explorer*.
2. Se deschid următoarele fordele în ordine:

-Local Disk (C:);
-ISE;
-final;
-final.xise.

3. Se selectează *top-level "avg"* din meniul *Design*.
4. Se deschide din *Processes Users Constraints: Floorplane Area/IO/Logic (PlanAhead)*.
5. Se va deschide o fereastră *ISE Project Navigator* și se dă click pe butonul *YES*.
6. Se deschide fișierul *.ucf* nou format din meniul *Design*.
7. Se va deschide o fereastră *ISE Project Navigator* și se dă click pe butonul *YES*.
8. Se revine la fișierul *top-level* și se selectează opțiunea *Configure Target Device*.
9. În fereastra nou deschisă, se selectează pe rând: *YES*, *Boundary Scan*, click dreapta în fereastra liberă, *Initialize Chain*, click drepta pe iconița apărută și *Program*.
10. Odată încărcat, proiectul este gata pentru a fi testat pe plăcuța *Nexys4*.

Se selectează opțiunile dorite pentru testare conform tabelelor 9 si 10. Funcționalitatea se poate urmări în Tabelul 11.

Setările comutatoarelor de "control":

Off - Off - Off	Modul Test / 0 (Zero)
Off - Off - On	Square Wave
Off - On - Off	Secvență ciclică de 6 cifre “Studentul nr. 1”
Off - On - On	Secvență ciclică de 6 cifre “Studentul nr. 2”
On - On - Off	Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15])
On - On - On	Generator de secvențe pseudo-aleatoare pe 8 biți (interval [0;255])

Tabelul 9

Setările comutatoarelor de "lungime":

Off - Off - Off	Stop - Păstrează valoarea
On - Off - Off	Medie pentru 2 numere
On - Off - On	Medie pentru 4 numere
On - On - Off	Medie pentru 8 numere
On - On - On	Medie pentru 16 numere

Tabelul 10

Square Wave	Medie 2	Medie 4	Medie 8	Medie 16
216	0	0	0	0
108	108	54	27	13
54	162	81	40	20
27	81	94	47	23
141	40	101	50	25
198	84	82	68	34
99	169	105	93	46
177	148	116	105	52
216	138	153	127	63
108	196	172	127	77
54	162	150	127	84
27	81	138	127	87
141	40	101	127	89
198	84	82	127	97
99	169	105	127	110
177	148	116	127	116

Tabelul 11

7. Posibilități de dezvoltare ulterioare

Posibilitățile de dezvoltare sunt restrânse, întrucât o îmbunătățire presupune o modificare parțială (nu totală) a codului. Astfel, imbunatatile pe care le-am luat în considerare sunt următoare:

1. Extinderea numărului de posibilități pentru “lungime”, deoarece numărul switch-urilor alocate acestei opțiuni și sumatorul generic existent (vezi Anexa) ne permit acest lucru (vezi tabelul 12).
2. Extinderea numărului de posibilități pentru “control”, deoarece numărul switch-urilor alocate acestei opțiuni ne permite acest lucru (vezi tabelul 13). Funcția MOD este descrisă astfel: f(x) = 27%x, iar funcția DIV este descrisă astfel: $f(x) = \frac{127}{x}$, unde x ia pe rând valorile 1, 2, 3, 4, 5, 6, 7 cu ajutorul unui numărător binar direct modulo 7.

Off - Off - Off	Stop - Păstrează valoarea
Off - Off -On	Medie pentru 2 numere
Off - On - Off	Medie pentru 4 numere
Off - On - On	Medie pentru 8 numere
On - Off - Off	Medie pentru 16 numere
On - Off - On	Medie pentru 32 numere
On - On - Off	Medie pentru 64 numere
On - On - On	Medie pentru 128 numere

Tabelul 12

Off - Off - Off	Modul Test / 0 (Zero)
Off - Off - On	Square Wave
Off - On - Off	Secvență ciclică de 6 cifre “Studentul nr. 1”
Off - On - On	Secvență ciclică de 6 cifre “Studentul nr. 2”
On - Off - Off	Generator de secvențe pseudo-aleatoare funcție MOD
On - Off - On	Generator de secvențe pseudo-aleatoare funcție DIV
On - On - Off	Generator de secvențe pseudo-aleatoare pe 4 biți (interval [0;15])
On - On - On	Generator de secvențe pseudo-aleatoare pe 8 biți (interval [0;255])

Tabelul 13

8. Anexă

Cod componentă finală:

```
library ieee;
use ieee.std_logic_1164.all;

--This is the final component (bottom-up implementation)

entity avg is
    port (reset_generator : in std_logic; --button
    reset_filter : in std_logic; --button
    system_clock : in bit; --100 MHz
    control : in std_logic_vector (2 downto 0); --switches
    lengthhh : in std_logic_vector (2 downto 0); --switches
    anod: out std_logic_vector(7 downto 0);
    catod : out std_logic_vector (6 downto 0));
end avg;

architecture avg of avg is

signal data_signal : std_logic_vector (7 downto 0);
signal data_clock : bit;
signal average_signal : std_logic_vector (7 downto 0);

component generator is
    port (system_clock : in bit;
    control : in std_logic_vector (2 downto 0);
    reset : in std_logic;
    data : out std_logic_vector (7 downto 0);
    data_clock : out bit);
end component generator;

component filter is
    port (data : in std_logic_vector (7 downto 0);
    data_clock : in bit;
    system_clock : in bit;
    lengthh : in std_logic_vector (2 downto 0);
    reset_filter : in std_logic;
    average : out std_logic_vector (7 downto 0) );
end component filter;
```

```
component display is
    port ( clock : in bit;
    data : in std_logic_vector (7 downto 0);
    average : in std_logic_vector ( 7 downto 0);
    anod: out std_logic_vector(7 downto 0);
    catod : out std_logic_vector (6 downto 0)
     );
end component display;

begin

    Label_Generator : generator port map (system_clock, control, reset_generator, data_signal, data_clock);
    Label_Filter : filter port map (data_signal, data_clock, system_clock, lengthhh, reset_filter, average_signal);
    Label_Display : display port map (system_clock, data_signal, average_signal, anod, catod);
end avg;
```

Cod Filter:

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;

--This component calculates the average

entity filter is
    port (data : in std_logic_vector (7 downto 0); --serial number on 8 bits
    data_clock : in bit; -- 1 Hz / 0.25 Hz
    system_clock : in bit; -- 100 MHz (never used)
    lengthh : in std_logic_vector (2 downto 0); --length switches
    reset_filter : in std_logic; --button
    average : out std_logic_vector (7 downto 0) );
end filter;
```

```
architecture filter of filter is

component sum is
    port ( data : in std_logic_vector (7 downto 0);
    clock : in bit;
    reset : in std_logic;
    Sum2 : out std_logic_vector(8 downto 0);
    Sum4 : out std_logic_vector(9 downto 0);
    Sum8  : out std_logic_vector(10 downto 0);
    Sum16 : out std_logic_vector(11 downto 0) );
end component sum;

signal sum_2 : std_logic_vector (8 downto 0);
signal sum_4 : std_logic_vector (9 downto 0);
signal sum_8 : std_logic_vector (10 downto 0);
signal sum_16 : std_logic_vector (11 downto 0);

signal avrg : std_logic_vector (7 downto 0);

begin
    Addition : sum port map (data, data_clock, reset_filter, sum_2, sum_4, sum_8, sum_16);


process (data_clock, lengthh, reset_filter) --computes the average according to the length
begin
    if (reset_filter = '1') then
        avrg <= (others => '0');
    elsif (data_clock'event and data_clock = '1') then
         case lengthh is
            when "000" => --stop-hold value
            avrg <= avrg;
            when "100" => -- 2 sample average
            avrg <= sum_2(8 downto 1); -- division by 2 (the last bit is lost)
            when "101" => -- 4 sample average
            avrg <= sum_4(9 downto 2); -- division by 4 (the last 2 bits are lost)
            when "110" => -- 8 sample average
            avrg <= sum_8(10 downto 3); -- division by 8 (the last 3 bits are lost)
            when "111" => -- 16 sample average
            avrg <= sum_16(11 downto 4); -- division by 16 (the last 4 bits are lost)
            when others => avrg <= (others => '0');
            end case;
    end if;
    average <= avrg;
end process;

end filter;
```

Cod Data Generator:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.all;

--This component generates digits/numbers according to the control switches

entity generator is
    port (system_clock : in bit; -- 100 MHz
    control : in std_logic_vector (2 downto 0); -- the control switches
    reset : in std_logic; --button
    data : out std_logic_vector (7 downto 0); --generated number
    data_clock : out bit); -- 1 Hz
end generator;

architecture generator of generator is

-- We instatiate the components needed to generate digits/numbers

component prg1 is
    port (clock : in bit;
    C : out std_logic_vector (7 downto 0));
end component prg1;

component prg2 is
    port (clock : in bit;
    C : out std_logic_vector (7 downto 0));
end component prg2;

component stud1 is
    port (clk : in bit;
    C : out std_logic_vector (7 downto 0));
end component stud1;
```

```
component stud2 is
    port (clock : in bit;
    O : out std_logic_vector (7 downto 0));
end component stud2;

component square_wave is
    port ( clock : in bit;
    clock_out : out bit;
    O : out std_logic_vector (7 downto 0));
end component square_wave;

component clk_div is
    port ( clock : in bit;
    clk : out bit );
end component clk_div;

type tip is array (4 downto 0) of std_logic_vector (7 downto 0);
signal s : tip;
signal data_signal : std_logic_vector (7 downto 0);
signal data_clock_signal : bit;
signal data_clock_signal_sqwave : bit;
signal clock : bit;

begin
    L0 : square_wave port map (data_clock_signal, data_clock_signal_sqwave, s(4));
    L1 : stud1 port map (data_clock_signal, s(0));
    L2 : stud2 port map (data_clock_signal, s(1));
    L3 : prg1 port map (data_clock_signal, s(2));
    L4 : prg2 port map (data_clock_signal, s(3));

    L5 : clk_div port map (system_clock, data_clock_signal);
```

```
process (data_clock_signal, reset, control)
begin
    if (reset = '1') then
        data_signal <= "00000000";
    elsif (data_clock_signal'event and data_clock_signal= '1') then
        case control is --choosing a method of generating digits/numbers
            when "001" => data_signal <= s(4); --square wave
            when "010" => data_signal <= s(0); --repeaed 6 digit sequence for student #1
            when "011" => data_signal <= s(1); --repeaed 6 digit sequence for student #2
            when "110" => data_signal <= s(2); --pseudo random sequence reduced range 0 to 15
            when "111" => data_signal <= s(3); --pseudo random sequence full range 0 to 255
            when others => data_signal <= "00000000"; --test mode
        end case;
    end if;
    end process;

process (control)
begin
    if (control = "001") then --when choosing the square wave the data_clock gets 0.25 Hz
        clock <= data_clock_signal_sqwave;
    else
        clock <= data_clock_signal; --otherwise, 1 Hz
        end if;
    end process;
    data <= data_signal;
    data_clock <= clock;
    end generator;
```

Sumator generic:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder is
    generic (n : integer := 8);
    port ( nr1, nr2 : in std_logic_vector (n-1 downto 0);
    sum : out std_logic_vector (n downto 0));
end adder;

architecture adders of adder is
begin
    sum <= ('0' & nr1) + ('0' & nr2);
end adders;
```