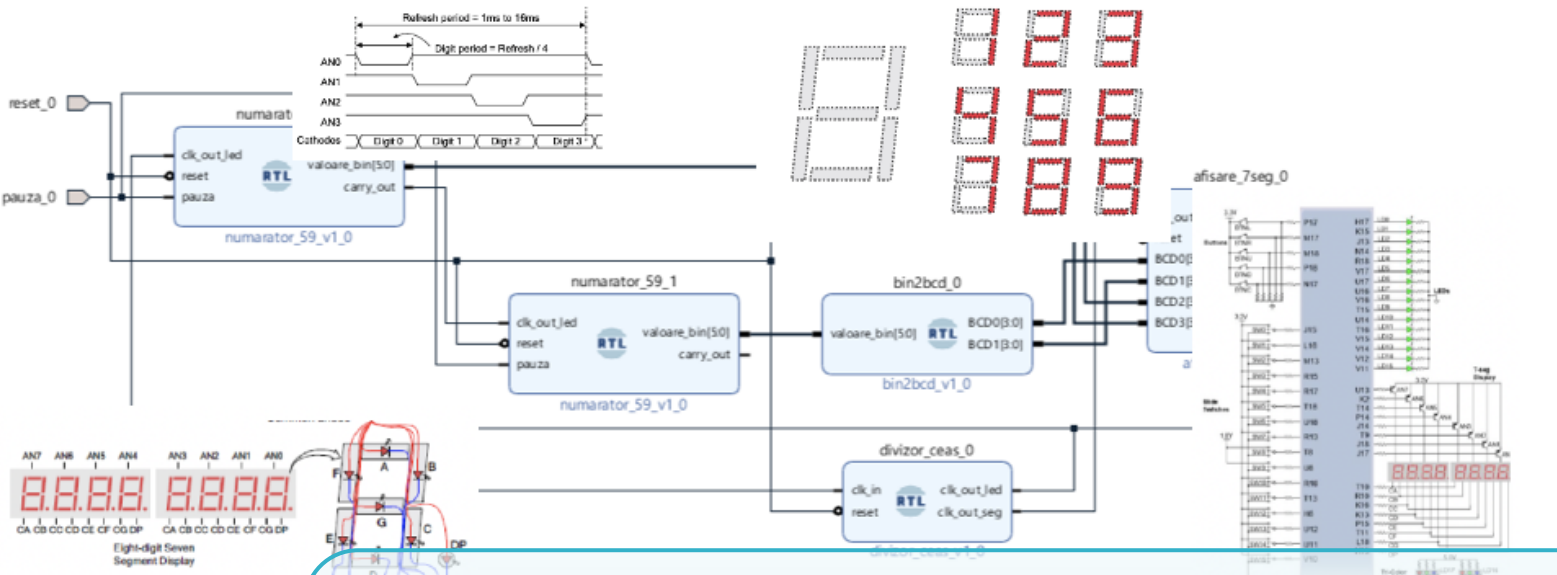

3. Circuitele secvențiale

3.1 Detaliile problemei

Circuitele secvențiale sunt circuitele a căror ieșire depinde de apariția frontului pozitiv sau negativ de ceas. În cadrul acestei lucrări de laborator vom dori să proiectăm și să implementăm un circuit care să simuleze un ceas cronometru.

Vom folosi dispozitivele 7 segmente existente pe placă pentru a afișa valoarea curentă a ceasului, 2 switch-uri pentru RESET și PAUZĂ și un led pentru a verifica funcționarea ceasului. Proiectarea necesită parcurgerea următorilor pași:

1. Determinarea semnalelor de I/O precum. În cazul nostru vom folosi semnalele din tabelul 3.1.

	Semnal	*Dimensiune*	*Descriere*
Intrări	**clk**	1 bit	semnal de ceas. Toate semnalele circuitului nostru vor fi sincronizate cu ceasul.
	pauza	1 bit	semnal folosit pentru oprirea ceasului cronometru
	reset	1 bit	orice circuit are nevoie de un semnal de reset care aduce circuitul în starea inițială în care toate semnalele sunt inițializate cu 0. De regulă acest semnal este 1 inițial pentru 20 ns și apoi el devine 0.
Ieșiri	**An**	8 biți	semnal care va explicat în secțiunea dedicată afișorului 7 segmente.
	Seg	8 biți	semnal care va fi explicat în secțiunea dedicată modulului de afișare 7 segmente
	DP	1 bit	semnal care va explicat în secțiunea dedicată modulului de afișare 7 segmente
	clik_out	1 bit	semnal de ieșire legat la un led pentru a observa funcționarea ceasului

Tabela 3.1: Semnalele de intrare - ieșire (I/O)

2. Determinarea modulelor componente:

- Din documentația plăcii se cunoaște faptul că frecvența de lucru a ceasului intern plăcii este de 50MHz sau de 100MHz. Ambele frecvențe sunt foarte mari și drept urmare fără existența unui divizor de ceas orice am încerca să afișăm, se va afișa mult prea repede ca să poată fi și observat. Ca atare vom proiecta în Verilog un divizor de ceas care să reducă frecvența de funcționare a ceasului de la 50MHz la 1Hz (aprox 1s) ca în figura 3.1:

$$clk_{out} = \frac{clk_{in}}{DIV} \tag{3.1}$$

```
1  module divizor_de_ceas(
2      input clk_in,
3      output clk_out
4      );
5
6      reg[27:0] numarator = 28'd0;
7      parameter DIV = 28'd2;
8
9  always @(posedge clk_in) begin
10     numarator <= numarator + 28'd1;
11     if(numarator >= (DIV - 1))
12         numarator <= 28'd0;
13 end
14
15 assign clk_out = (numarator < DIV / 2) ? 1'b0 : 1'b1;
16
17 endmodule
18
19
```

Figura 3.1: Divizorul de ceas

Pentru a obține o frecvență de 1Hz va trebui să facem o împărțire cu 50.000.000 valoare ce rezultă din conversie [1] sau se pot face conversiile din Hz direct în s^2.
În cadrul proiectării noastre vom avea o implementare a divizorului dea ceas un pic mai diferită deoarece vom prefera să facem un divizor de ceas care să primească ca intrare valoarea de ceas a plăcii (50MHz) și să ne ofere la ieșire 2 valori ale ceasului (una pentru a aprinde led-ul cu o frecvență observabilă și o a doua valoare care va aprinde circuitul de afișare cu 7 segmente). În concluzie, vom avea nevoie să proiectăm un divizor de ceas cu 2 intrări și 2 ieșiri (**clk_in**, **reset / clk_out_led**, **clk_out_seg**). Rezultatul proiectării în Verilog al acestui modul este prezentat în figura 3.2.
- Vom avea nevoie de proiectarea unui numărător binar care să numere de 0 la 59. Acest modul va avea ca intrări semnalele: **clk_out_led**, **reset**, **pauza** iar ca ieșiri **valoare_bin**, **carry_out**. Modulul trebuie proiectat și implementat în cadrul laboratorului.
- Un modul de conversie din binar în BCD este necesar pentru a face conversia numărului rezultat din numărător într-o valoare reprezentată în cod BCD (conversia BCD și operațiile BCD sunt prezentate mai jos). Acest modul va avea următoarele semnale de I/O: **valoare_bin / BCD0**, **BCD1**. Valorile de ieșire BCD0 și BCD1 vor corespunde valorii minutelor și vom mai avea nevoie de duplicarea acestui modul pentru a trata cazul secundelor.

[1] `https://www.rapidtables.com/convert/frequency/mhz-to-hz.html`
[2] `https://www.unitjuggler.com/convert-frequency-from-s(p)-to-Hz.html`

C:/Users/Decebal/Desktop/lab3_new/lab3_new.srcs/sources_1/imports/temp1/divizor_ceas.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
21 module divizor_ceas(
22     input clk_in, reset,
23     output clk_out_led, clk_out_seg
24     );
25
26 reg[32:0] counter;
27
28 always @(posedge clk_in)
29     if (!reset)
30         counter = 0;
31     else
32         counter <= counter + 1;
33
34     //assign clk_out_led = counter[27];
35     assign clk_out_led = counter[26];
36     assign clk_out_seg = counter[17];
37 endmodule
```

Figura 3.2: Proeictarea unui divizor de ceas

- Ultimul modul necesar este modulul de afișare care va prelua valorile BCD produse de cele două numărătoare și le va afișa pe un dispozitiv 7 segmente.

3. Generarea diagramei care conține toate modulele și semnalele necesare implementării proiectului.

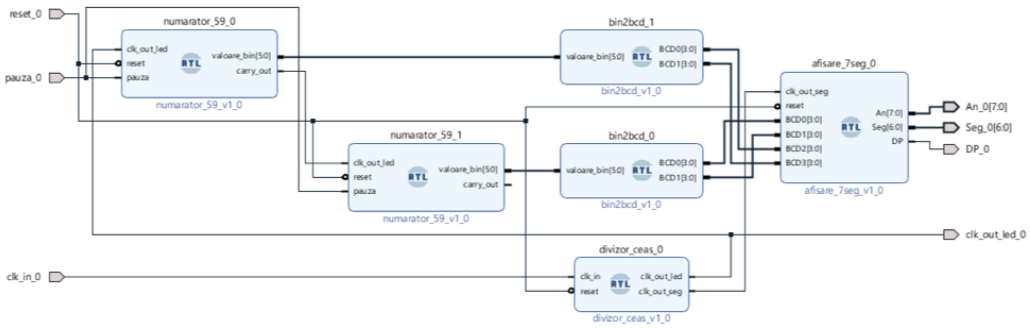

Figura 3.3: Diagrama circuitului

3.2 Numere ZCB

În calculatoarele numerice, informația memorată în format binar se regăsește în memorie sau în registrele procesorului. Registrele conțin date sau informații de control. Informația de control este formată dintr-un grup de biți utilizat în vederea specificării secvenței semnalelor de comandă necesare manipulării datelor în registre. Datele sunt numere şi alte informații codificate binar, care sunt prelucrate în vederea obținerii unui rezultat numeric.

Număr hexazecimal	*Hexazecimal codificat binar*	*Număr octal*	*Octal codificat binar*	*Număr ZCB*	*Număr zecimal*
0	0000	0	000	0000	0
1	0001	1	001	0001	1
2	0010	2	010	0010	2
3	0011	3	011	0011	3
4	0100	4	100	0100	4
5	0101	5	101	0101	5
6	0110	6	110	0110	6
7	0111	7	111	0111	7
8	1000			1000	8
9	1001			1001	9
A	1010				10
B	1011				11
C	1100				12
D	1101				13
E	1110				14
F	1111				15

Tabela 3.2: Reprezentarea numerelor zecimale în diferite sisteme de numerație

Toate tipurile de date, cu excepția numerelor binare, sunt reprezentate în registrele calculatorului în formă binară, deoarece acestea sunt alcătuite din bistabile, care pot memora informație 0 sau 1.

Utilizarea numerelor **binare codificate zecimal (ZCB)**. Este foarte important să nu se confunde conversia numerelor zecimale în binar si codificarea binară a numerelor zecimale. Când se realizează conversia numărului zecimal 99 într-un număr binar se obține șirul de biți 1100011, reprezentarea ZCB fiind 10011001. Singura diferență constă în simbolurile utilizate pentru reprezentarea biților.

În tabelul 3.2 se poate observa codificarea numerelor zecimale în diferite sisteme de numerație.

În calculatoarele numerice precum şi în diverse aplicații este necesar să se utilizeze alte coduri precum codul Gray, codul 2421, codul excess-3 Gray, etc. Acestea sunt prezentate în tabelul 3.3.

Digital zecimal	*Grey*	*BCD 8421*	*2421*	*Excess-3*	*Excess-3 Grey*
0	0000	0000	0000	0011	0010
1	0001	0001	0001	0100	0110
2	0011	0010	0010	0101	0111
3	0010	0011	0011	0110	0101
4	0110	0100	0100	0111	0100
5	0111	0101	1011	1000	1100
6	0101	0110	1100	1001	1101
7	0100	0111	1101	1010	1111
8	1100	1000	1110	1011	1110
9	1101	1001	1111	1100	1010

Tabela 3.3: Diferite coduri pentru cifrele zecimale

În majoritatea dispozitivelor numerice, operațiile de prelucrare a informațiilor se desfășoară în

binar. În multe aplicații este necesară o afișare directă, pe dispozitive cu șapte segmente (sau tuburi NIXIE), a informației prelucrate. Din acest motiv, este necesară o conversie a valorii binare, în zecimal codificat binar (ZCB). Codul ZCB permite o decodificare ușoară pentru dispozitive cu șapte segmente.

Convertoarele prezentate sunt de următoarele tipuri:

- ASINCRON – realizate numai cu rețele combinaţionale de porți logice. Tehnica de conversie se bazează pe metodele descrise de J.P. Couler;
- SINCRON – serie şi paralel, utilizând registre de deplasare şi sumatoare;
- SECVENŢIAL – utilizând numărătoare binare şi decadice.

3.2.1 Conversie din binar în ZCB

Un număr binar, se exprimă de obicei sub forma: $N_b = b_{n-1}b_{n-2}...b_1b_0$ unde n este numărul de biți și b_j este o valoare egală cu 0 sau 1. Valoarea zecimală a acestui număr este:

$$N_Z = b_{n-1} * 2^{n-1} + b_{n-2} * 2^{n-2} + ... + b_1 * 2^1 + b_0 * 2^0 \tag{3.2}$$

Sau sub formă de înmulțire cu 2 prin împachetare:

$$N_Z = (...((b_{n-1} * 2 + b_{n-2}) * 2 + b_{n-3}) * 2 + ... + b_1) * 2 + b_0 \tag{3.3}$$

Înmulțirea cu 2 prin împachetare, se poate realiza prin deplasare la stânga (exemplu în tabelul 3.4).

0	0	0	b_3	$N_Z = b_3$
0	0	b_3	b_2	$N_Z = b_3 * 2 + b_2$
0	b_3	b_2	b_1	$N_Z = (b_3 * 2 + b_2) * 2 + b_1$
b_3	b_2	b_1	b_0	$N_Z = ((b_3 * 2 + b_2) * 2 + b_1) * 2 + b_0$

Tabela 3.4: Exemplu: $N_Z = (((b_3 * 2 + b_2) + b_1) + b_0$ valoarea zecimală a numărului binar $N_B = b_3b_2b_1b_0$

Se observă că pentru a converti un număr binar N_B într-un număr ZCB, trebuie modificate ponderile corespunzătoare biților registrului, efectuând corecțiile necesare în timpul deplasării. În acest scop registrul, care va păstra numărul ZCB, va trebui împărțit în grupe de 4 biți (decade). Biții unei decade $d_8d_4d_2d_1$ vor avea ponderile 8, 4, 2, 1 şi valoarea reprezentată nu va depăși valoarea 9.

Orice deplasare la stânga va însemna o înmulțire cu 2. În momentul în care o decadă conține un număr mai mare decât 4, rezultatul deplasării va depăși valoarea 10 şi în acest caz va trebui făcută următoarea corecție: **înainte de deplasare, se adaugă valoarea 3, la acele decade care conțin un număr mai mare decât 4**. Corecția prezentată rezultă din următoarele constatări:

- dacă valoarea decadei este mai mică sau egală cu 4 prin deplasare la stânga cu o poziție se obține un număr mai mic decât 9. Prin urmare nu trebuie făcută nici o corecție
- dacă valoarea decadei este mai mare decât 4 şi mai mică sau egală cu 7, prin deplasarea la stânga cu o poziție se obține un număr mai mare decât 9. se poate scrie:

$$decada * 2 = (5 + (decada - 5)) * 2 = 10 + (decada - 5) * 2 \tag{3.4}$$

Se observă că înainte de deplasare se scade 5, iar după deplasare se adaugă 1 la poziția cea mai puțin semnificativă din decada următoare. Același lucru se poate realiza dacă se scade

valoarea 5 înainte de deplasare şi se forțează în bitul d8 al decadei valoarea 1 (echivalent cu a aduna valoarea 8 înainte de deplasare). Deci înainte de deplasare, se adaugă 8 şi se scade 5, ceea ce este echivalent cu a aduna 3 înainte de deplasare.

- dacă valoarea decadei este mai mare sau egală cu 8 şi mai mică sau egală cu 9, după deplasare, bitul cu ponderea 8 (egal cu 1) va trece în bitul cel mai puțin semnificativ al decadei următoare. Prin aceasta, valoarea se modifică de la 8 la 10 în loc să se modifice de la 8 la 16. Are loc o pierdere egală cu 6. Corecția se poate face, după deplasare, adunând 6 la valoarea decadei sau înainte de deplasare adunând 3.

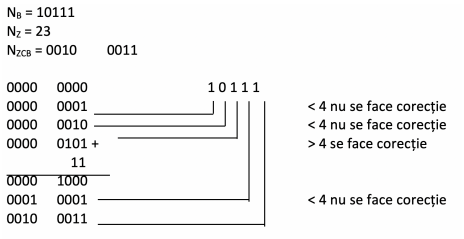

Figura 3.4: Exemplu conversie ZCB

Modul de conversie prezentat mai sus este serie-sincron şi necesită un transfer paralel – serie a informației la intrare. Conform teoriei prezentate, modulul Verilog de conversie binar-ZCB este prezentat în figura 3.5.

C:/Users/Decebal/Desktop/lab3_new/lab3_new.srcs/sources_1/imports/temp1/conversie_bin_BCD.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////...
21
22
23 module bin2bcd(
24     input [5:0]valoare_bin,
25      output [3:0]BCD0, BCD1
26     );
27
28     integer i;
29     reg [7:0]bcd;
30
31     //bloc Always - implementarea algoritmului Double Dabble
32      always @(valoare_bin)
33         begin
34             bcd = 0; //initializare bcd la 0.
35             for (i = 0; i < 6; i = i+1) //run for 8 iterations
36             begin
37                 bcd = {bcd[6:0], valoare_bin[5-i]}; //concatenation
38                 //daca un digit din 'BCD' este mai mare decat 4, adunam 3 la el.
39                 if(i < 5 && bcd[3:0] > 4)
40                     bcd[3:0] = bcd[3:0] + 3;
41                 if(i < 5 && bcd[7:4] > 4)
42                     bcd[7:4] = bcd[7:4] + 3;
43             end
44         end
45
46     assign BCD0 = bcd[7:4];
47     assign BCD1 = bcd[3:0];
48
49 endmodule
```

Figura 3.5: Modul conversie ZCB

3.3 Operații ZCB

3.3.1 Adunarea în ZCB

Pentru codul ZCB se consideră codul ponderat 8-4-2-1. Adunarea în cod ZCB este identică cu adunarea binară. Suma pentru fiecare decadă trebuie corectată considerând transferul la decada următoare și rezultatele care au valori cuprinse între 10 și 15. În momentul adunării a două decade rezultă douăzeci de sume posibile. Dintre acestea, numai zece vor fi corecte, celelalte necesitând corecții. În tabelul următor sunt prezentate sumele a două cifre ZCB înainte de corectare și după corectare.

Suma incorectă					**Suma corectă**					
T_{pj}	S_{pj8}	S_{pj4}	S_{pj2}	S_{pj1}	T_j	S_{j8}	S_{j4}	S_{j2}	S_{j1}	
	0	0	0	0		0	0	0	0	= 0
	0	0	0	1		0	0	0	1	= 1
	0	0	1	0		0	0	1	0	= 2
	0	0	1	1		0	0	1	1	= 3
	0	1	0	0		0	1	0	0	= 4
	0	1	0	1		0	1	0	1	= 5
	0	1	1	0		0	1	1	0	= 6
	0	1	1	1		0	1	1	1	= 7
	1	0	0	0		1	0	0	0	= 8
	1	0	0	1		1	0	0	1	= 9
	1	0	1	0	1	0	0	0	0	= 10
	1	0	1	1	1	0	0	0	1	= 11
	1	1	0	0	1	0	0	1	0	= 12
	1	1	0	1	1	0	0	1	1	= 13
	1	1	1	0	1	0	1	0	0	= 14
	1	1	1	1	1	0	1	0	1	= 15
1	0	0	0	0	1	0	1	1	0	= 16
1	0	0	0	1	1	0	1	1	1	= 17
1	0	0	1	0	1	1	0	0	0	= 18
1	0	0	1	1	1	1	0	0	1	= 19

Tabela 3.5: Rezultatele adunării ZCB a două numere

Pentru o sumă egală sau mai mare decât 10, este necesar un transport spre decada următoare și o corecție ce constă în a scădea valoarea 10 din decada următoare. Scăderea valorii 10 este echivalentă cu adunarea complementului față de 2, care este 0110 sau 6 în zecimal. Din tabelul de mai sus se observă că transferul T_j în cazul sumei corectate este: $T_j = T_{pj} + t_j$ unde T_{pj} este transferul bitului S_{pj8} la stânga în cazul sumei necorectate, iar t_j este o funcție de S_{pj8}, S_{pj4}, S_{pj2}, S_{pj1}. Funcția t_j rezultă din diagrama Karnaugh din figura 3.6.

3.3.2 Scăderea în ZCB

Dacă se scade o cifră zecimală și eventual un împrumut dintr-o cifră zecimală, în codul 8-4-2-1, există 20 de diferențe corecte de la 9 la –10. În tabelul următor se prezintă diferențele incorecte și diferențele corecte ale unei operații de scădere.

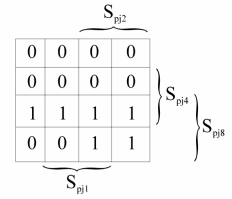

Figura 3.6: Diagraga Karnaugh pentru t_j

Diferență incorectă					**Diferență corectă**					
	1	0	0	1		1	0	0	1	= 9
	1	0	0	1		1	0	0	0	= 8
	0	1	1	1		0	1	1	1	= 7
	0	1	1	0		0	1	1	0	= 6
	0	1	0	1		0	1	0	1	= 5
	0	1	0	0		0	1	0	0	= 4
	0	0	1	1		0	0	1	1	= 3
	0	0	1	0		0	0	1	0	= 2
	0	0	0	1		0	0	0	1	= 1
	0	0	0	0		0	0	0	0	= 0
-1	1	1	1	1	-1	1	0	0	1	= -1
-1	1	1	1	0	-1	1	0	0	0	= -2
-1	1	1	0	1	-1	0	1	1	1	= -3
-1	1	1	0	0	-1	0	1	1	0	= -4
-1	1	0	1	1	-1	0	1	0	1	= -5
-1	1	0	1	0	-1	0	1	0	0	= -6
-1	1	0	0	1	-1	0	0	1	1	= -7
-1	1	0	0	0	-1	0	0	1	0	= -8
-1	0	1	1	1	-1	0	0	0	1	= -9
-1	0	1	1	0	-1	0	0	0	0	= -10

Tabela 3.6: Scăderea în ZCB

Dacă numărul împrumută o unitate (-1) el apare în reprezentarea prin complement față de 10. În continuare vom considera că operanzii negativi vor fi reprezentați în complement față de 9. În această reprezentare operația de scădere va fi mai ușor de realizat. Operația de scădere a doi operanzi ZCB constă în adunarea complementului față de 9 al scăzătorului la descăzut. Complementul față de 9 al unei cifre zecimale reprezentate în cod ZCB, se realizează conform diagramei prezentate în tabelul următor:

Din tabelul 3.7 rezultă ecuațiile logice pentru complementul față de 9:

$$\begin{aligned} c_8 &= d_8 * d_4 * d_2 \\ c_4 &= d_4 + d_2 \\ c_2 &= d_2 \\ c_1 &= d_1 \end{aligned} \tag{3.5}$$

Cifra ZCB				Complement față de 9			
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0

Tabela 3.7: Complementul față de 9 al unei cifre ZCB

3.4 Afișarea 7 segmente

Pentru a realiza afișarea 7 segmente se urmăresc detaliile din figura 3.7 și tabelul 3.8.

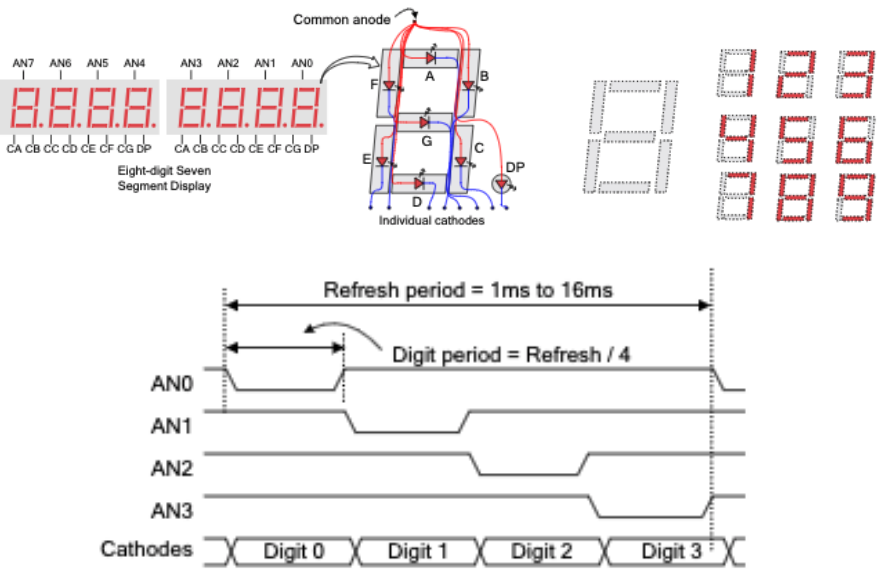

Figura 3.7: Afișaj 7 segmente

Astfel modulul de afișare 7 segmente în Verilog poate fi descris ca în figura 3.8.

Număr	A	B	C	D	E	F	G
0000	0	0	0	0	0	0	1
0001	1	0	0	1	1	1	1
0010	0	0	1	0	0	1	0
0011	0	0	0	0	1	1	0
0100	1	0	0	1	1	0	0
0101	0	1	0	0	1	0	0
0111	0	0	0	1	1	1	1
1000	0	0	0	0	0	0	0
1001	0	0	0	0	1	0	0
>=1010	1	1	1	1	1	1	1

Tabela 3.8: Codificare pe 7 segmente

C:/Users/Decebal/Desktop/lab3_new/lab3_new.srcs/sources_1/imports/temp1/afisare_7seg.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////


module afisare_7seg(
    input clk_out_seg, reset,
    input [3:0]BCD0, BCD1, BCD2, BCD3,
    output reg [7:0]An,
    output reg [6:0]Seg,
    output reg DP
    );

    reg [3:0]BCD;
    reg [1:0]S; // stari de afisare

always @(posedge clk_out_seg)
begin
    if (!reset)
        S=2'b00;

    An[7:4]=4'b1111;

    case (S)
        2'b00: {BCD, An[3:0]} = {BCD0,4'b0111};
        2'b01: {BCD, An[3:0]} = {BCD1,4'b1011};
        2'b10: {BCD, An[3:0]} = {BCD2,4'b1101};
        2'b11: {BCD, An[3:0]} = {BCD3,4'b1110};
    endcase

    S = S + 1;
    case (BCD)
        4'b0000 : Seg = 7'b0000001; //0 - a, b, c, d, e, f, g
        4'b0001 : Seg = 7'b1001111; //1
        4'b0010 : Seg = 7'b0010010; //2
        4'b0011 : Seg = 7'b0000110; //3
        4'b0100 : Seg = 7'b1101100; //4
        4'b0101 : Seg = 7'b0100100; //5
        4'b0110 : Seg = 7'b0100000; //6
        4'b0111 : Seg = 7'b0001111; //7
        4'b1000 : Seg = 7'b0000000; //8
        4'b1001 : Seg = 7'b0001100; //9
        4'b1010 : Seg = 7'b0001001; //A
        4'b1011 : Seg = 7'b1100000; //b
        4'b1100 : Seg = 7'b0110001; //C
        4'b1101 : Seg = 7'b1000010; //d
        4'b1110 : Seg = 7'b0110000; //E
        4'b1111 : Seg = 7'b0111000; //F
        default: Seg = 7'b1111111; //dispozitiv stins
    endcase

    if (S==2'b10)
        DP=0;
    else
        DP=1;
end
endmodule
```

Figura 3.8: Module afișaj 7 segmente

Cerință de lucru 1:

Proiectați restul elementelor conform schemei generale, realizați un modul top în Verilog și simulați în plăcuța disponibilă în cadrul laboratorului.

Cerință de lucru 2:

Se dorește proiectarea unui automat pentru băuturi răcoritoare. Pentru a simplifica problem, se consideră că se poate elibera un singur tip de băutură răcoritoare. Aceasta costă 3 RON. Automatul acceptă bancnote de 1, 5 , respectiv 10 RON și eliberează rest dacă s-a introdus o sumă mai mare decât prețul băuturii. Se presupune că există un mecanism pentru sortarea banilor și care emite trei semnale, câte unul pentru fiecare tip de bancnotă. Semnalele determină tranziția automatului dintr-o stare în alta.

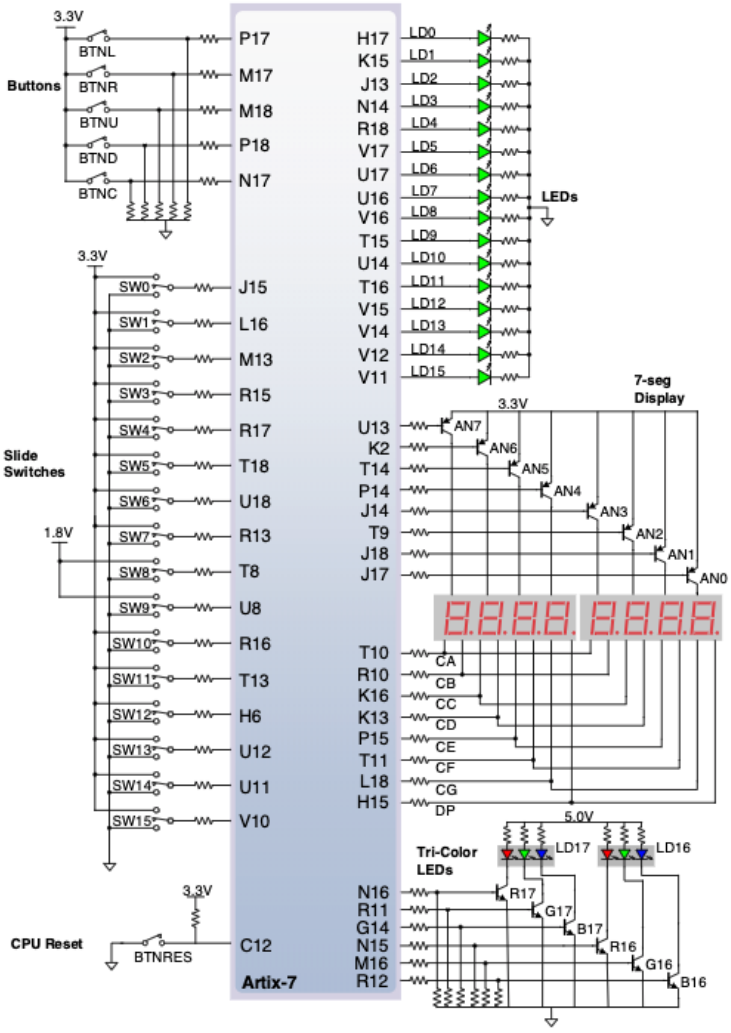

Figura 3.9: Maparea constrângerilor pentru placa Nexys