**Facultatea de Automatica și Calculatoare**

**Calculatoare și Tehnologia Informației**



**Proiectarea sistemelor numerice**

**Proiect:**

**MICROCONTROLLER PE 8 BIȚI**

**Profesor îndrumator: Realizat de:**

**Diana Pop Câmpean Casiana Ștefana**

**Grupa: 30218 Luca-Dan Adrian**

**CUPRINS**

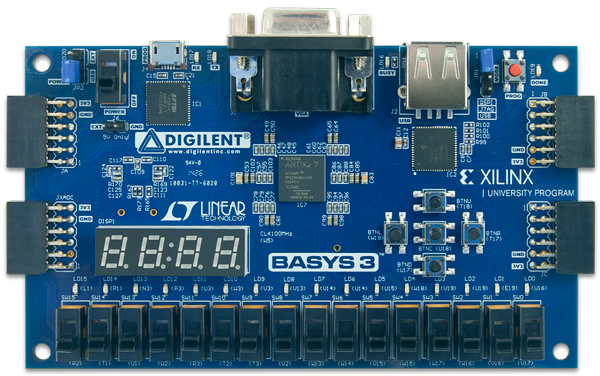
1. **Specificația proiectului**
2. **Schema bloc**
3. **Evidențierea unitații de comanda și de execuție**
4. **Etapele de priectare**
5. **Componentele**
6. **Semnificația notaților efectuate in proiect și a interfaței cu exteriorul(I/O)**
7. **Justificarea soluției alese**
8. **Instrucțiuni de utilizare și întreținere**
9. **Posibilități de dezvoltare ulterioare**

**1.Specificația proiectului**

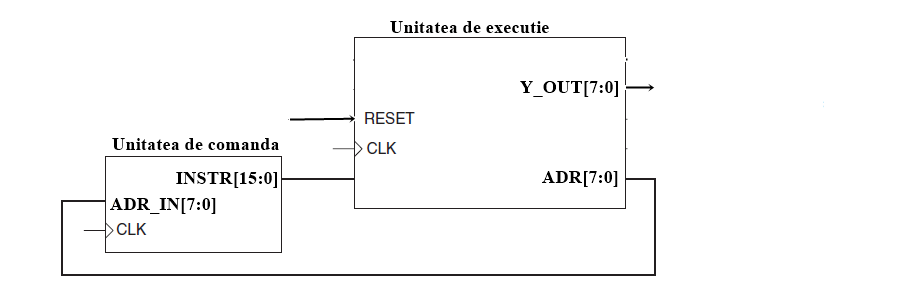
**Cerința proiectului:**

Să se realizeze un microcontroller pe 8 biți conform documentației XAPP213.pdf. Să se execute un program care să folosească toate intrucțiunile implementate, inclusiv întreruperi. Proiectul va fi realizat de 2 studenți.

Proiectul este implemetat pe plăcuța Basys 3, unde vom putea observa rezultatele finale pe afișor.

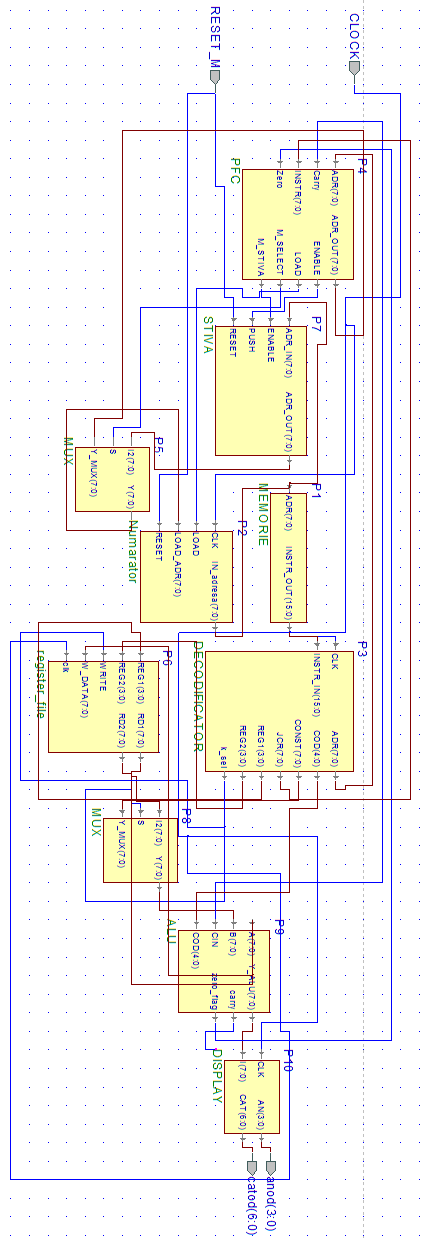
****

**2. Schema bloc**



**Dezvoltarea schemei bloc:**

**Dezvoltarea**



**3.Evidențierea unității de comandă și de execuție**

Microcontrolerul efectueaza operații specificate prin intrucțiuni pe 16 biți cu numere pe 8 biți stocate în memorie cu ajutorul blocului de regiștrii. El este format din două parți: unitatea de comandă și unitatea de execuție.

Unitatea de comandă este formată din memoria ROM, decodificatorul, Program Counter, Program Flow Control și Program Counter Stack-ul. Memoria ROM este cea care conține comanda pe 16 biți, aceasta urmând să fie tradusă de decodificator și transmisă mai departe către unitatea aritmetico-logică.

Unitatea de comandă conține până la 256 de instrucțiuni, acestea fiind executate secvențial cu ajutorul Program Counter-ului, care numără și parcurge fiecare intrucțiune din memorie.

Unitatea de execuție este alcătuită din blocul de registrii, Unitatea Aritmetico-Logică, un multiplexor si un afișor. În Unitatea aritmetco-logică vor intra două numere în funcție de intrucțiunea transmisă de decodificator, fie conținutul a doi regiștrii, fie o constantă și conținutul unui registru. Unitatea este cea care face operațiile, rezultatul fiind transmis înapoi într-un registru, urmând sa fie transmise pe afișor.

**4. Etapele de proiectare**

S-a parcurs documentația XAPP213. A fost analizată cu mare atenție arhitectura oferită acolo, după care ne-am decis asupra componentelor și asupra funcțiilor pe care acestea le îndeplinesc. Am studiat instrucțiunile din memorie și modurile de decodificare a acestora, urmând să scriem cele mai semnificative componente (UAL, memoria, blocul de regiștrii, stiva, decodificatorul de instrucțiuni ș.a).

După realizarea componentelor necesare, le-am unit într-un program principal, unde au fost declarate semnalele necesare.

S-au introdus date în memorie și s-au verificat în simulatorul Active HDL. După ce acestea au funcționat am introdus programul în Vivado unde am realizat fișierul XDC pentru a face legătura cu plăcuța Basys 3. Odată finalizată sinteza și implementarea programului urmate de generarea cu succes a bitstream-ului, am verificat dacă rezultatele din simulare coincid cu rezultatele afișate pe plăcuță.

**5.Componentele**

* Bloc de regiștrii
* Memoria ROM
* Program Counter
* Program Flow Control
* Stivă
* Unitate Aritmetico-Logică
* Afișor
* Decodificator
* Zero&Carry Flags
* Decodificator pentru afisor
* Divizor de frecvență
* Inversul divizorului de frecvență
* Multiplexor

**Blocul de registrii**

Blocul de regiștrii reprezintă spațiul de stocare central al microcontrollerului. Majoritatea operațiilor implică folosirea sau modificarea datelor stocate în blocul de regiștrii. Această componentă conține:

5 intrări: CLK, impulsul de tact.

REG1,REG2, care conțin adresele a doi dintre regiștrii, intrări care se transmit de la decodificator.

WRITE, este un semnal care dacă este acivat putem scrie în registru.

W\_DATA este o intrare pe 8 biți care va fii numărul scris în registru.

2 ieșiri:RD1, RD2, numere pe 8 biți, sunt datele care se află la adresele registrilor de la intrări, care vor fi transmiși către UAL.

**Memoria ROM**

Memoria ROM(Read only memory) este tipul de memorie care poate fi numai citită. Ea conține până la 256 de instrucțiuni de câte 16 biți, fiecare intrucțiune fiind stocată la o adresă de 8 biți.

În memorie sunt instrucțiunile ce trebuie citite și executate secvențial de către microcontroler. Aceste instrucțiuni sunt transmise mai departe către decodificator pentru a putea fi decodificate.

Memoria este formata din:

1 intrare: ADR, adresele pe 8 biți unde sunt stocate instrucțiunile.

1 ieșire: INSTR\_OUT, pe 16 biți, aceasta fiind transmisă la decodificator.

**Program Counter**

Program Counter-ul este un numărător pe 8 biți care accesează instrucțiunile din memoria ROM. Acesta se modifică în momentul în care avem o instucțiune de tip JUMP/CALL/RETURN , luând o anumită valoare impusă în memorie.

Program Counter-ul este format din:

4 intrări : CLK ( impulsul de tact) , RESET(resetează numărătorul), LOAD (intrarea pe 1 bit care verifică dacă există JUMP în instrucțiune), LOAD\_ADR(intrarea pe 8 biți care vine din Program Flow Control dacă există JUMP sau CALL, sau din stivă, dacă există RETURN).

O ieșire : IN\_adresa (pe 8 biți, care indică adresa pe care o apelăm in memorie).

**Stiva**

Stiva funcționează asemenea unei memorii, care ne ajută să revenim la o instrucțiune apelată anterior. Aceasta este compusă pe 15 nivele și în momentul în care există o instrucțiune de CALL/RETURN , aceasta este memorată pe ultimul nivel al acesteia. În cazul în care avem mai multe instrucțiuni de același tip, prima adresa memorată coboară un nivel, iar pe ultimul se reține ultima adresă. Stiva funcționează după principiul (Last In First Out).

Stiva este format din:

4 intrări : PUSH(intrare pe 1 bit care realizează incărcarea în momentul activarii), ENABLE(intrare pe 1 bit vine de la Program Flow dacă există instrucțiune de tip CALL/RETURN), RESET(intrarea ce realizează resetarea memoriei), ADR\_IN(adresa pe 8 biți venită de la Program Flow, adică adresa care se apelează).

O ieșire : ADR\_OUT(adresă pe 8 biți care intră în numărător in cazul în care există instrucțiune de tip RETURN).

**Program Flow Control**

Această componentă ne ajută la implementarea instrucțiunilor JUMP, CALL și RETURN. Ea conține 4 intrări:

-INSTR, pe 8 biți, este transmisă de decodifcator, ea este analizată in process și va determina tipul operației(JUMP, CALL sau RETURN) și dacă este condiționată sau nu.

-ADR, este tot pe 8 biți, fiind transmisă de decodificator, va fii utilizită în cazul în care vom dori să realizăm un CALL, aceasta fiind memorată în stivă.

-ZERO și CARRY, sunt semnale care vor fi necesare în cazul în care dorim să efectuăm o instrucțiune condiționată.

Componenta mai conține 4 ieșiri:

-ADR\_OUT, este pe 8 biți și va prelua datele de intrare de la ADR, urmând să fie transmise stivei în cazul operației CALL și RETURN.

-ENABLE va lua valoarea 0 dacă am ales JUMP și va lua 1 dacă este vorba de CALL și RETURN.

-LOAD va lua valoarea 1 dacă este o instrucțiune necondiționată, și 0 dacă este o instrucțiune necondiționată.

-M\_STIVA va fii transmis către stivă, va lua valoarea 0 dacă vom face RETURN, și 1 dacă facem CALL.

**Unitatea Aritmetico-Logică**

În Unitatea Aritmetico-Logică se realizează operațiile care trebuie să le efectuăm și anume : Load, And, Or, Xor, adunare cu și fără carry, scădere cu și fără carry, dar si grupul de shiftare stânga-dreapta, respectiv rotație stânga-dreapta.

Unitatea Aritmetico-Logică este formată din :

4 Intrări : CIN(intrare pe 1 bit pentru CARRY IN), COD(intrare pe 4 biți venită de la decodificator, care îi spune unității ce operație să efectueze), A, B(intrări pe 8 biți venite din blocul de regiștrii).

3 ieșiri : Y\_ALU (ieșire pe 8 biți care oferă rezultatul dintre operanzii A și B), carry(ieșire pe un bit care comunică momentul in care există carry out), zero\_flag(ieșire pe un bit care ne comunică dacă în urma operației realizate intre A și B rezultatul final este nul).

**Divizor de frecvență**

Divizorul de frecvență realizează împărțirea semnalului de tact, acesta ajutându-ne să observăm rezultatele o dată la aproximativ o secundă.

Divizorul de frecvență este format din :

O intrare pe un bit clk\_div (acesta este clock-ul normal care intră),

O ieșire pe un bit clk\_out (acesta este clock-ul divizat).

**Inversul divizorului de frecvență**

Din cauza faptului că pe afișor nu este posibil să scriem mai mult de o cifră simultan, am folosit acest “inversor” pentru a mări frecvența astfel încât să observăm datele corecte de pe afișor. Prin mărirea frecvenței, ochiului uman îi este imposibil sa percepă faptul ca cele 4 ecrane se aprind și se sting la o viteză foarte mare.

Acesta este compus din:

O intrare pe un bit: clk\_div (semnalul de tact normal)

O ieșire pe un bit si: clk\_out (semnalul de tact la o frecvență mult mai mare).

**Multiplexorul**

Multiplexorul 2:1 este folosit de două ori în programul principal:

O data pentru a selecta între cine se va realiza operația în Unitatea Aritmetico-Logică, alegând între o constantă(pe 8 biți) sau un număr(tot pe 8 biți) reținut într-un registru aflat în blocul de regiștrii.

A doua oară este utilizat în cazul în care dorim să efectuăm una din instrucțiunile JUMP, CALL sau RETURN. Pentru intrarea componentei Program Counter, noi vom alege între ieșirea de la componenta Program Flow Control când avem JUMP sau CALL, și între ieșirea de la componenta stivei cand avem RETURN.

Multiplexorul 2 la 1 este format din :

3 Intrări : Y\_MUX(intrare pe 8 biți venită de la un anumit registru), I2(intrare pe 8 biți venită de la decodificator, acesta fiind o constantă), S( intrare pe un bit care realizează selecția dintre celelalte două intrări pe 8 biți, această intrare vine de la decodificator si este 0 în cazul în care avem un registru si 1 dacă avem o constantă);

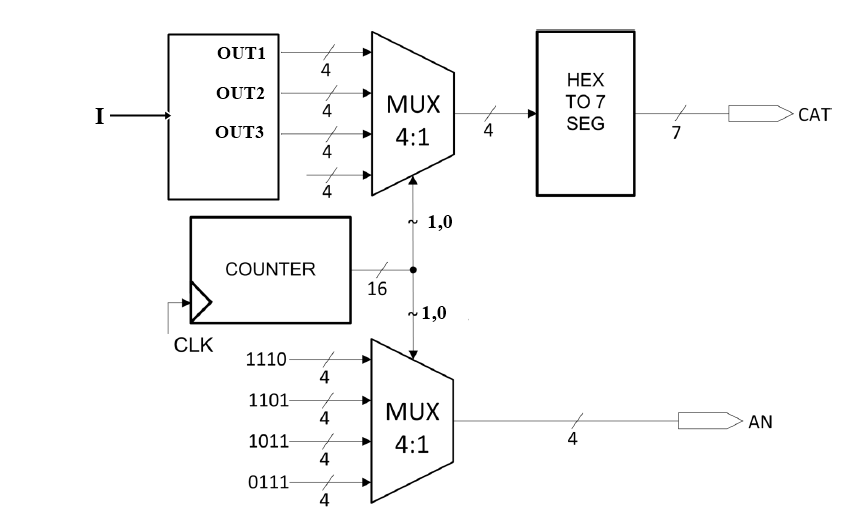
O ieșire : Y(ieșire pe 8 biți care va intra în Unitatea Aritmetico-Logică, acesta fiind una dintre intrările Y\_MUX sau I2).

**Afișorul**

Placa Basys vine echipată cu un afișor pe 7 segmente. Această interfață folosește șapte leduri pentru fiecare cifrăș fiecare cifră este activată de un semnal de anod. Toate semnalele sunt active pe 0. Semnalele de catod controlează ledurile care se aprind de pe acele cifre care au semnalul de anod activ.

Pentru a afișa patru cifre diferite este necesară implementarea unui circuit care trimite cifrele de pe semnalele de catod ale afișorului. Perioada maximă de reîmprospătare este astfel calculată încât ochiul uman să nu perceapă aprinderea succesivă a fiecărei cifre de pe afișor. Se realizează astfel o afișare ciclică a cifrelor(doar o cifră se afișează la un moment dat).

În figură se prezintă implementarea circuitului.



Avem 2 intrări: CLK, impulsul de tact și I, care este pe 8 biți, aceasta este transmisă de către UAL și va intra în componenta CIFRE. Deoarece numărul I este pe 8 biți, el este un număr zecimal pe 3 cifre, iar această componentă descompune cifrele, ieșirile sale fiind OUT1,OUT2 și OUT3, numerele pe 4 biți care corespund cu cifrele numărului, ieșiri care vor fi intrările afișorului.

2 ieșiri: semnalele de anod AN si de catod CAT, toate active pe zero.

**Decodificatorul**

Decoficatorul este o componentă esențială în proiect. El decodifică informația primită din memorie. Instrucțiunile sunt de mai multe tipuri: aritmetice, logice, de shiftare, jump, call și return. Decodificatorul analizează la fiecare instrucțiune anumite grupuri de biți, și in funcție de acestea se va realiza un anumit tip de operație, și se va decide dacă operația va fii între 2 regiștrii sau între un registru și o constantă.

Conține:

2 intrări:

CLK, impulsul de tact

INSTR\_IN, instrucțiunea pe 16 biți care vine din memorie și care urmează să fie decodificată

7 ieșiri:

CONST, aceasta va lua valoarea constantei, un numar pe 8 biți care va urma să fie introdus in UAL pentru a face o operație.

JCR și ADR, acestea va lua biții [15:12], respectiv [7:0] ai intrării INSTR\_IN, care vor fi transmiși mai departe la Program Flow în cazu în care avem o instrucțiune de tipul CALL, RETURN sau JUMP.

REG1,REG2 vor lua valorile adreselor pe 4 biți ale celor 2 registrii din instrucțiune.

COD va prelua codul pe 5 biți care va fii transmis către UAL pentru a determina tipul operației

k\_sel va lua valoarea 1 dacă operația se va realiza între 2 regiștrii sau 0 dacă operația este între un registru si o constantă.

Biții intrării INSTR\_IN(15 downto 12) reprezintă tipul operației și între cine se va realiza operația:

**“1100”** reprezintă faptul ca intruncțiunea conține o operație aritmetica sau logică, executată între 2 regiștrii. INSTR\_IN(11 downto 8) conține adresa primului registru, și INSTR\_IN(7 downto 4) conține adresa celui de-al 2-lea registru, iar INSTR\_IN(3 downto 0) conține tipul operației aritmetice sau logice astfel:

“0000” LOAD sX,sY

“0001” AND sX,sY

“0010” OR sX,sY

“0011” XOR sX,sY

“0100” ADD sX,sY

“0101” ADDCY sX,sY

“0110” SUB sX,sY

“0111” SUBCY sX,sY

Dacă INSTR\_IN(15 downto 12) se află în intervalul [0,7] atunci instrucțiunea conține o operație aritmetică sau logică, realizată între un registru si o constantă.

“0000” LOAD sX, kk

“0001” AND sX, kk

“0010” OR sX, kk

“0011” XOR sX, kk

“0100” ADD sX, kk

“0101” ADDCY sX, kk

“0110” SUB sX, kk

“0111” SUBCY sX, kk

Dacă INSTR\_IN(15 downto 12) este “1101” atunci va urma o operație de shifare, iar in funcție de INSTR\_IN(7 downto 0) aflam daca este shifare stânga sau dreapta, și INSTR\_IN(2 downto 0) ne va spune tipul shiftarii:

Dacă INSTR\_IN(7 downto 5) este "00001" atunci urmeaza Shift Right, iar instructiunea in funcție de INSTR\_IN(2 downto 0) va fii astfel:

“110” SR0 sX

“111” SR1 sX

“010” SRX sX

“000” SRA sX

“100” RR sX

Dacă INSTR\_IN(7 downto 5) este "00000" atunci urmeaza Shift Right, iar instructiunea in funcție de INSTR\_IN(2 downto 0) va fii astfel:

“110” SL0 sX

“111” SL1 sX

“100” SLX sX

“000” SLA sX

“011” RL sX

Dacă INSTR\_IN(15 downto 13) este “100” atunci urmează o instrucțiune de tipul jump, call sau return. Bitul 12 determina dacă instrucțiunea este condițională(1) sau necondițională(0). Dacă aceasta este condițională, atunci analizăm și biții 11 și 10 pentru a verifica condiția astfel:

“00” if Zero

“01” if NOT Zero

“10” if Carry

“11” if NOT Carry.

**6. Semnificația notaților efectuate in proiect și a interfaței cu exteriorul(I/O)**

În programul principal care unește componentele există două intrări, ambele pe un bit și anume RESET\_M ( acesta resetează adresa, si afișand 000 pe afișor), CLOCK (semnalul de tact care este divizat), respectiv două ieșiri care se numesc anod( iesire pe 4 biți ) și catod ( ieșire pe 7 biți) ambele fiind folosite pentru a scrie pe afișor rezultatul final.

**7. Justificarea soluției alese**

În urma parcurgerii documentaței proiectului, am ales această soluție deoarece am considerat că este este cea mai simplă de realizat pentru nivelul nostru. Am decis să implementăm fiecare componentă separat, utilizând arhitectura microcontrollerului pe 8 biți.

Fiecare componentă folosită respectă îndeaproape informațiile precizate în documentația proiectului:

-pentru a memora datele folosim blocul de regiștrii, care conține 16 regiștrii a cate 8 biți

-memoria este alcătuită din 256 de instrucțiuni de 16 biți

-Multiplexorul 2:1, care decide între cine se realizează operațiile din UAL

-UAL, care realizează toate instrucțiunile aritmetico-logice prezentate în documentație

-Program Counterul, care numără fiecare instrucțiune

-Program Flow Control, cu ajotorul căreia realizăm și transmitem intrucțiunile de tip JUMP, CALL și RETURN.

-Decodificatorul, care decodifică tipul de instrucțiune pe 16 biți, după regulile date in documentație

**8. Instrucțiuni de utilizare și întreținere**

Se introduc instrucțiunile pe 16 biți în memorie pe baza codificării exacte a fiecarei comenzi și în funcție de acestea vom putea observa rezultatele finale pe afișorul plăcuței Basys 3.

Un exemplu de listă de instrucțiuni care se poate introduce în memorie este următoarea:

signal MEM:tip\_mem :=

(

0=>"0000001000000001", --LOAD S2 KK (1)

1=>"1000001100000011", --CALL TO 3 (0)

2=>"1000100100001010", --JUMP TO 10 (0)

3=>"0001011001001101", --AND S6 S4 (47&77) (13)

4=>"0010011001001101",--OR S6 S4 (47 | 77) (111)

5=>"0011011001001101",--XOR S6 S4 (47^77)(98)

6=>"0100110001100100",--ADD S12 KK(100 +=100 =200)

7=>"0101100000110111", --ADDCY S8 KK(167 +55+1 =223 )

8=>"0110100100101000",--SUB S9 KK (90 -40 =50)

9=>"1000000010000000",--RETURN(ADR 2) (0)

10=>"0111001100010111",--SUBCY S3 KK(78 –23-1 =54)

11=>"1001101100001101",--CALL CONDITIONAT 13(0)

12=>"1001100100010110",--JUMP CONDITIONAL 22 (0)

13=>"1100000100100000",--LOAD S1 S2 (24)

14=>"1100110010100001",--AND S12 S10 (100& 213) (68)

15=>"1100110010100010",--OR S12 S10 (100 | 213) (245)

16=>"1100110010100011",--XOR S12 S10 (100^ 213) (177)

17=>"1100010001110100",--ADD s4 s7(138+75 =213)

18=>"1100110011100101", --ADDCY s10 s14 (100 +114 =215)

19=>"1100101111010110",--SUB s11 s13 (43 -33=10)

20=>"1100010100100111",--SUBCY s5 s2 (164 -24 =139)

21=>"1001100010000000", --RETURN(ADR 12) (0)

22=>"1101001100001110",--SR0 s3 (39)

23=>"1101011000001111",--SR1 s6 (151)

24=>"1101100100001010",--SRX s9 (45)

25=>"1101110000001000",--SRA s12 (178)

26=>"1101111100001100",--RR s15 (32)

27=>"1001110100000111",--JUMP INVALID

28=>"1101001100000110",--SL0 s3 (156)

29=>"1101011000000111",--SL1 s6 (95)

30=>"1101100100000100",--SLX s9 (180)

31=>"1101110000000000",--SLA s12 (201)

32=>"1101111100000011",--RL s15 (128)

others=>"0000000000000000");

Întrucât s-a folosit divizorul de frecvență pentru clock nu trebuie să apăsăm niciun buton, deoarece fiecare rezultat va fi afișat secvențial timp de o secundă.

Singurul contact fizic pe care îl vom avea cu plăcuța va fii în momentul în care dorim să resetăm și să începem de la prima adresă, folosind switch-ul R2.

Pentru a implementa proiectul pe plăcuța Basys 3, trebuie introdus proiectul în programul Vivado, unde se realizează urmatorul fișier XDC:

set\_property PACKAGE\_PIN W5 [get\_ports CLOCK]

set\_property IOSTANDARD LVCMOS33 [get\_ports CLOCK]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports CLOCK]

set\_property PACKAGE\_PIN R2 [get\_ports {RESET\_M}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {RESET\_M}]

set\_property PACKAGE\_PIN W7 [get\_ports {catod[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[0]}]

set\_property PACKAGE\_PIN W6 [get\_ports {catod[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[1]}]

set\_property PACKAGE\_PIN U8 [get\_ports {catod[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[2]}]

set\_property PACKAGE\_PIN V8 [get\_ports {catod[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[3]}]

set\_property PACKAGE\_PIN U5 [get\_ports {catod[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[4]}]

set\_property PACKAGE\_PIN V5 [get\_ports {catod[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[5]}]

set\_property PACKAGE\_PIN U7 [get\_ports {catod[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {catod[6]}]

set\_property PACKAGE\_PIN V7 [get\_ports dp]

set\_property IOSTANDARD LVCMOS33 [get\_ports dp]

set\_property PACKAGE\_PIN U2 [get\_ports {anod[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[0]}]

set\_property PACKAGE\_PIN U4 [get\_ports {anod[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {anod[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[2]}]

set\_property PACKAGE\_PIN W4 [get\_ports {anod[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {anod[3]}]

**9. Posibilități de dezvoltare ulterioare**

O primă posibilitate de dezvoltare ulterioară ar fi realizarea intrucțiunilor cuprinse în:

Input/Output Group

Aceste instrucțiuni permit scrierea manuală de către utilizator direct în oricare dintre regiștrii sau permit transmiterea datelor din regiștrii direct către un port extern, cu ajutorul decodificatorului.

INPUT sX,pp

INPUT sX, (sY)

OUTPUT sX,pp

OUTPUT sX, (sY)

Interrupt Group

Aceste instrucțiuni sunt asemănătoare cu CALL și RETURN, doar că o instrucțiune de întrerupere forțează microcontrollerul să meargă către ultima adresă din memorie “CALL FF”, iar cu ajutorul lui RETRUNI se revine la ultima adresă din stivă.

RETURNI ENABLE

RETURNI DISABLE

ENABLE INTERRUPT

DISABLE INTERRUPT

O altă posibilitate de dezvoltare ulterioară ar fi mărirea memoriei pentru a permite realizarea mai multor instrucțiuni. De la un număr de 256 de instrucțiuni putem ajunge la 1024, memoria având adresele pe 10 biți în loc de 8. De asemenea, pentru mărirea programului, putem să implementăm un bloc de regiștrii care să conțină 32 de regiștrii.