Facultatea de Automatică și Calculatoare

Disciplina: Proiectarea sistemelor numerice

Distribuitor de Coca Cola

Profesor Coordonator:

Diana Irena Pop

Student:

Giesswein Alexia

Grupa 30215

Anul I

Cuprins:

1.Specificația proiectului:

Să se proiecteze un automat distribuitor de Coca Cola. Preţul este de 1 leu. Se acceptă monede de 5, 10 şi 50 bani. Sistemul este prevăzut cu 5 fotocelule:

F0 – pentru moneda de 5 bani.

F1 – pentru moneda de 10 bani.

F2 – pentru moneda de 50 bani.

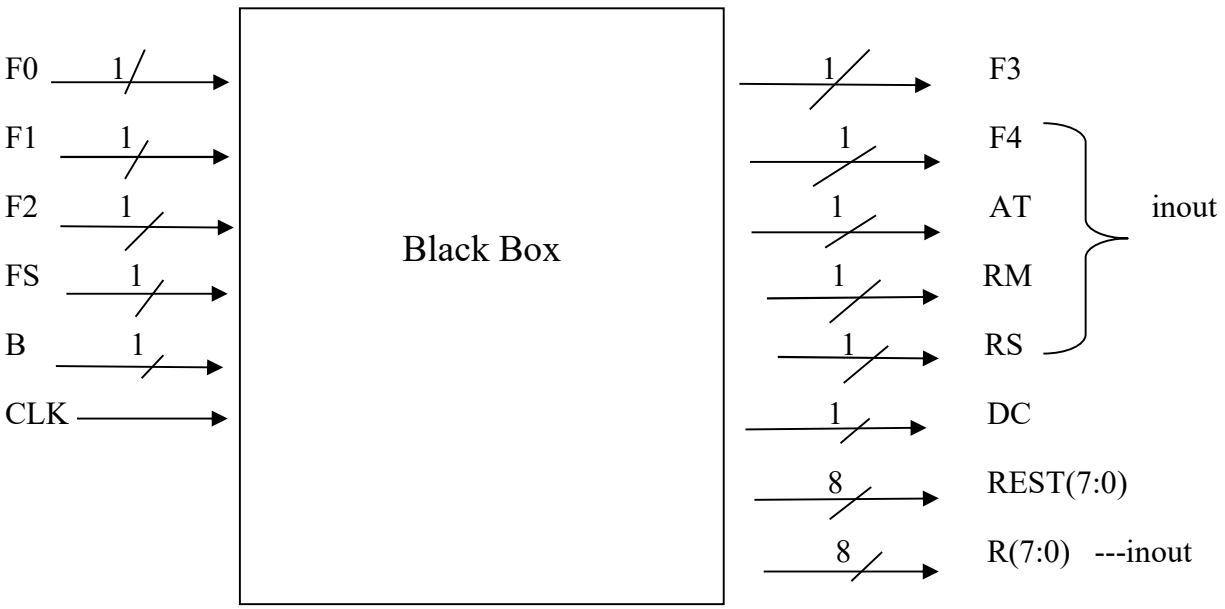
F3 – pentru respingere monedă (alta decât cele acceptate) sau corpuri străine.

F4 – pentru semnal de acceptare a monedei.

Dacă nu există Coca Cola atunci nu se acceptă nici un tip de monezi (FS). Se face verificarea pentru suma totală şi monezile sunt returnate dacă suma nu este completă (RM). Se eliberează rest, dacă este cazul.

Se generează semnale şi se semnalizează pentru acceptarea unei monezi (AM), a totalului (AT) şi pentru eliberarea de Coca Cola.

2. Schema bloc:



Intrările:

Denumire:	F0	F1	F2	FS	B	CLK
Descriere:	intrare pentru moneda de 5 bani	intrare pentru moneda de 10 bani	intrare pentru moneda de 50 bani	este sau nu Coca Cola	am terminat de introdus monedele	la fiecare semnal de ceas, se scrie în registru suma anterioara
Pe câți biți:	1 bit	1 bit	1 bit	1 bit	1 bit	

Ieșirile: inout

Denumire:	F3	F4	AT	RM	RS	REST	DC	R
Descriere:	respinge moneda	acceptă moneda	suma introdusă este 1 leu	suma introdusă este mai mică decât 1 leu --se returnează suma	suma introdusă este mai mare decât 1 leu --se dă rest	restul dat	se dă Cola	suma din registru
Pe câți biți:	1 bit	1 bit	1 bit	1 bit	1 bit	8 biți	1 bit	8 biți

2.1 Semnificația intrărilor și ieșirilor:

- Intrări:
  - F0 – este 1 dacă se introduce o monedă de 5 bani
    – este 0 dacă se introduce altceva
  - F1 – este 1 dacă se introduce o moneda de 10 bani
    – este 0 dacă se introduce altceva
  - F2 – este 1 dacă se introduce o moneda de 50 de bani
    – este 0 dacă se introduce altceva
  - FS – este 1 dacă este Coca Cola în automat
    – este 0 dacă nu este Coca Cola în automat
  - B – este un buton care are valoarea 1 când este apăsat, se apasă dacă s-au terminat de introdus monedele, iar suma introdusă este mai mică decât 1 leu
    – este 0 când nu este apăsat (dacă încă se introduc monede sau suma este mai mare sau egală cu 1 leu)
  - CLK – se scrie suma anterioara in registru la fiecare semnal de ceas
  - CON – o intrare constantă, are mereu valoarea 0
  - C – o intrare constantă, are mereu valoarea 100

- Ieșiri:
  - F3 – este 1 dacă se respinge moneda introdusă
    – este 0 dacă se acceptă moneda introdusă
  - DC – este 1 dacă suma totală este mai mare sau egală cu 1 leu (se dă Cola)
    – este 0 dacă suma este mai mică decât 1 leu (nu se dă Cola)
  - REST – restul care trebuie dat (0 dacă suma este mai mică sau egală cu 1 leu, sau diferența obținută în scăzător dacă suma este mai mare decât 1 leu)
  - I5 – o ieșire din al doilea comparator (este 1 dacă suma din registru este egală cu 0, altfel este 1)
  - I6 – o ieșire din al doilea comparator (este 1 dacă suma din registru este mai mică decât 0, altfel este 1)
- Inout:
  - F4 – este 1 dacă moneda este acceptată, este 0 dacă moneda este respinsă
    – este folosit ca intrare pentru a calcula biții inout-ului M(moneda introdusă)
  - AT – este 1 dacă suma este exact 1 leu, este 0 altfel
    – este folosit ca intrare pentru a determina valoarea ieșirii DC (drop Cola)
  - RS – este 1 dacă suma este mai mare decât 1 leu, este 0 altfel => se dă rest
    – este folosit ca intrare pentru a determina valoarea restului
  - RM – este 1 dacă suma este mai mică decât 1 leu, este 0 altfel
    – este folosit pentru a seta și reseta biții din registru (dacă RM=1 => suma este mai mică decât 1 leu, în registru se scrie valoarea introdusă, RM=0 => suma este mai mare sau egală cu 1 leu, se resetează biții din registru)
  - M – este valoarea monezii introduse, care se introduce în sumator pentru a calcula suma finală (moneda introdusă + suma din registru)
  - Suma – este valoarea sumei care este obținută din sumator și transmisă mai departe pentru a fi memorată în registru
  - I1 și I2 – valoarea sumei introduse, respectiv I2=100 atunci când RS=1, intrări în scăzător pentru a obține valoarea restului
  - I3 – valoarea sumei din registru, iar atunci când se apasă butonul B=1, dacă I3 este mai mare decât 0, se resetează registrul si se returnează suma din registru
  - I4 – este 1 dacă I3 este mai mare decât 0 și resetează biții registrului
  - D0, D1,...,D7 – biții care se pun în registru, care alcătuiesc suma memorată
  - F – F=F1+F2+F3, folosit pentru a obține valoarea lui F4 și F3
  - L – L=F1+F2, folosit pentru a obține valoarea unui bit a lui M(moneda introdusă)

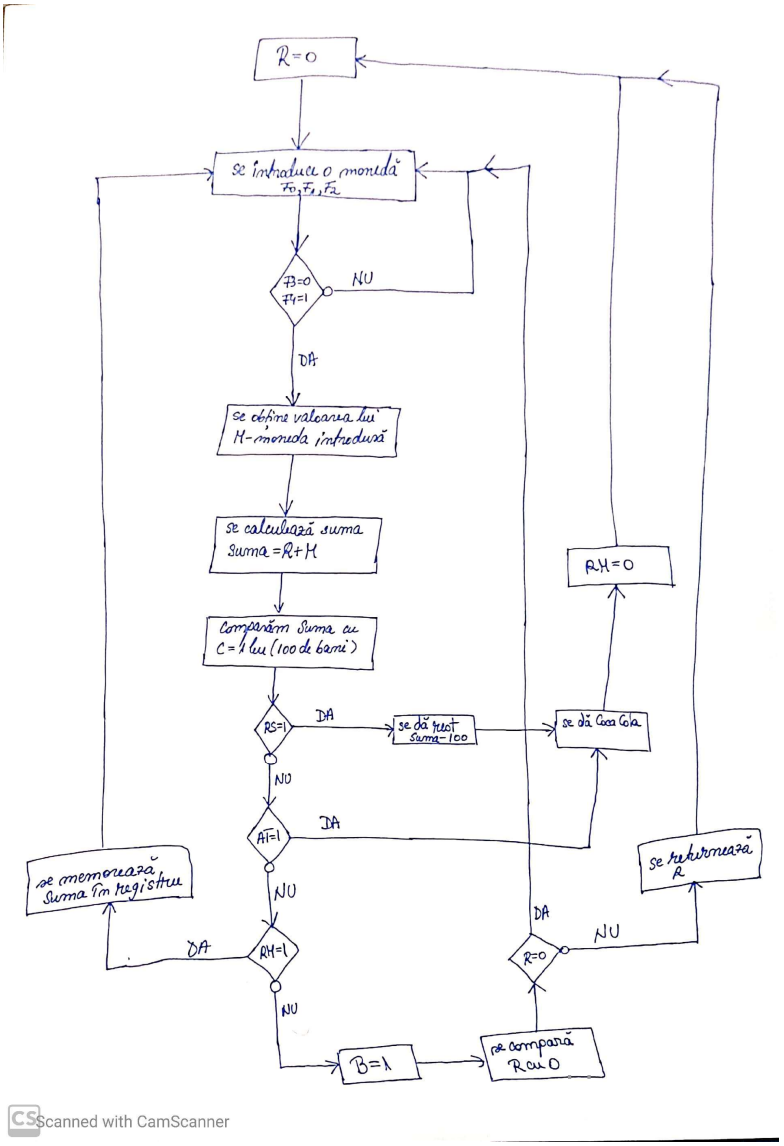
3. Unitatea de comandă și unitatea de execuție:

- Unitatea de comandă:
  - Generează semnale de comandă pentru întregul sistem de calcul.
  - Dirijează fluxul de date.
  - Reglează acțiunile sale în funcție de un semnal de ceas.

Unitatea de comandă este cea care controlează fiecare resursă din unitatea de execuție, oferindu-i semnalele de comandă corecte pentru toate resursele din aceasta, pentru ca algoritmul să fie executat corect. Pentru ca unitatea de comandă să transmită semnalele, aceasta trebuie să citească atât datele introduse, cât și starea în care se află resursele din unitatea de execuție.

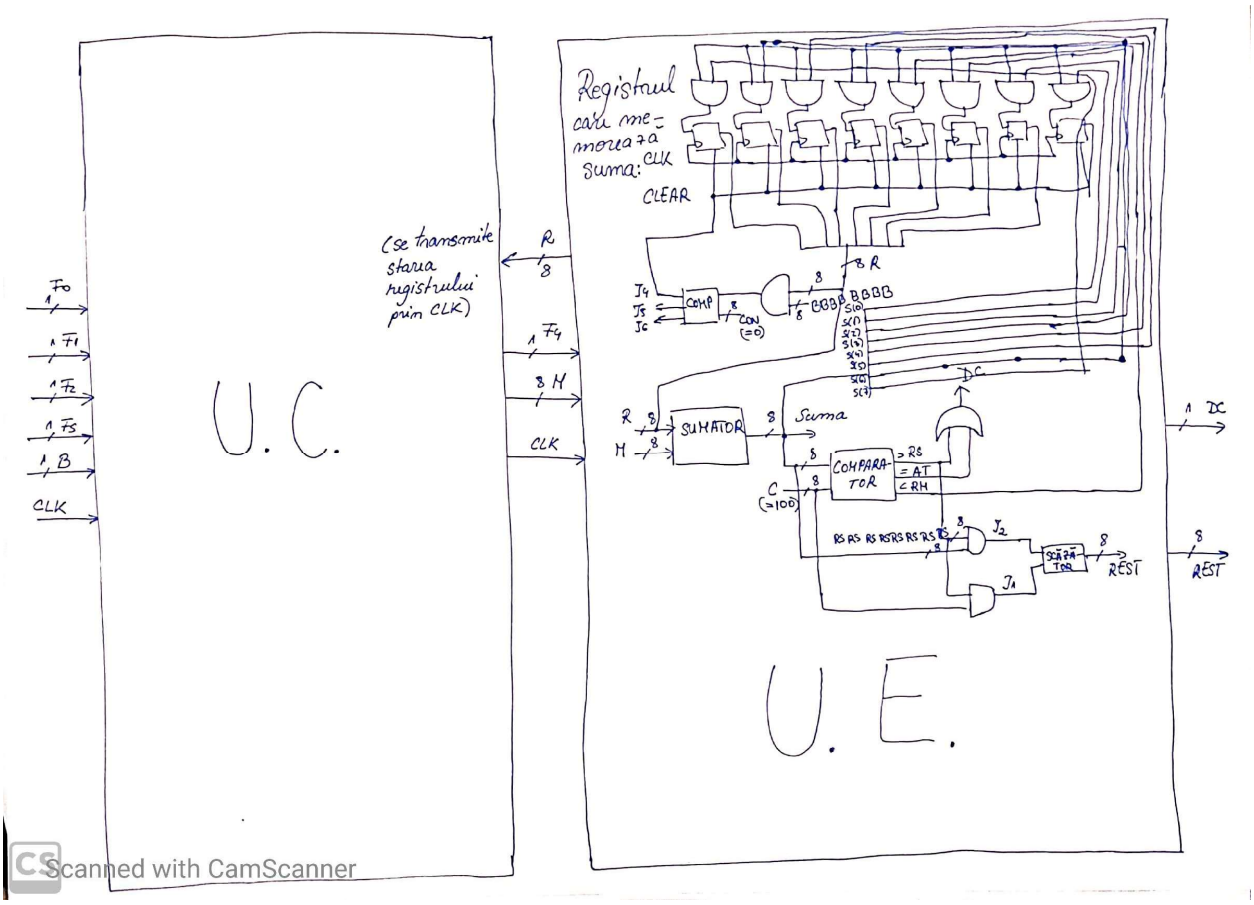
Pentru distribuitorul de Coca Cola, în unitatea de comandă se află cele 3 intrări pentru monezi (F0, F1, F2), intrarea FS pentru a transmite dacă mai este sau nu Coca Cola, intrarea B pentru a transmite terminarea introducerii monezilor (doar când suma este mai mică va fi nevoie) și semnalul de ceas.

Schemă inițială:



- Unitatea de execuție:
  - Execută instrucțiunile care îi sunt transmise de unitatea de comandă.
  - Realizează calcule aritmetice și logice cunoscute din matematică.
  - Conține toate resursele care execută comenzile transmise.

Pentru distribuitorul de Coca Cola, în unitatea de execuție se află toate resursele de care are nevoie pentru a îndeplini ce-i cere utilizatorul: registrul de memorare, realizate cu bistabile D (pentru a reține suma anterioară), un sumator (pentru a calcula suma finală=suma reținută în registru + moneda introdusă), un comparator (pentru a compara suma cu 1 leu—costul unei sticle de Cola), un scazător (pentru a obține restul care trebuie returnat) și înca un comparator (dacă suma introdusă este mai mică decât 1 leu și nu se mai introduc monede, se resetează bții registrului).



4. Proiectare:

4.1 Descrierea componentelor:

- Porți logice: Poarta ȘI, Poarta SAU, Poarta NOT – care ajută la obținerea unor componente sau alte intrări/ieșiri (ex. registru, inout-ul M, ieșirea F3, F4 etc.).
- Registrul de memorare este alcătuit din 8 bistabile D flip-flop, fiecare bistabil D reține un bit al sumei introduse anterior => în registru se memorează suma obținută până la momentul actual. La fiecare semnal de ceas, se introduce o monedă, se calculează suma si se reține în registru. În registru se află inițial valoarea 0, iar acesta se resetează când nu se mai introduc monede sau s-a dat un Coca Cola.
- Sumatorul calculează suma dintre moneda introdusă M si valoarea din registru R. Această sumă se scrie în registru la fiecare semnal de ceas.
- Un comparator care compară suma cu 1 leu (100 de bani) și decide dacă se dă Cola, dacă se dă rest sau dacă se continuă adunarea sumei.
- Un alt comparator care compară butonul B cu suma din registru, iar când B=1, dacă R>0 atunci se returnează R și se resetează registrul, R=0.
- Scăzătorul scade din sumă (dacă RS=1, s-a dat o sumă mai mare decât 1 leu) 1 leu(100 de bani) și calculează restul care trebuie dat.

4.2 Schema logică + explicații

- Explicații:
  Se introduce o monedă. Dacă FS=1 (este coca cola) și moneda este introdusă corect (F0+F1+F2=1) atunci F3=0 și F4=1 (se acceptă moneda), altfel F3=1 și F4=0 (se respinge moneda).

  Avem un sumator cu 2 intrări pe 8 biți în care calculăm suma introdusă.

Cele 2 intrări sunt: suma anterioară salvată în registru (inițial în registru va fi valoarea 0) R și moneda introdusă M.

Cu M am notat moneda introdusă.
- F0 5: 0000 0101
- F1 10: 0000 1010 => intrarea 0 0 F2 F2 F1 F0 (F1+F0) F0 =M
- F2 50: 0011 0010

Fiecare bit a lui M este obținut printr-o poartă ȘI între F4 și intrarea descrisă mai sus.

- dacă F4=0 (se respinge moneda) atunci se adauga 0 la sumă
- dacă F4=1 (se acceptă moneda) atunci se adauga moneda introdusă

Se salvează suma rezultată, în registrul de memorare. La fiecare bistabil D intrarea este obținută printr-o poartă ȘI între biții sumei (fiecare bistabil memorează un bit al sumei) și RM (care este 1 cât timp suma e mai mică decât 1 leu, iar când suma e mai mare RM=0 =>resetează biții din registru).

Se pune suma într-un comparator și se compara cu 1 leu(100 de bani).

Dupa ce se compara suma

--dacă ea este exactă AT=1 =>DC=1-se eliberează cola

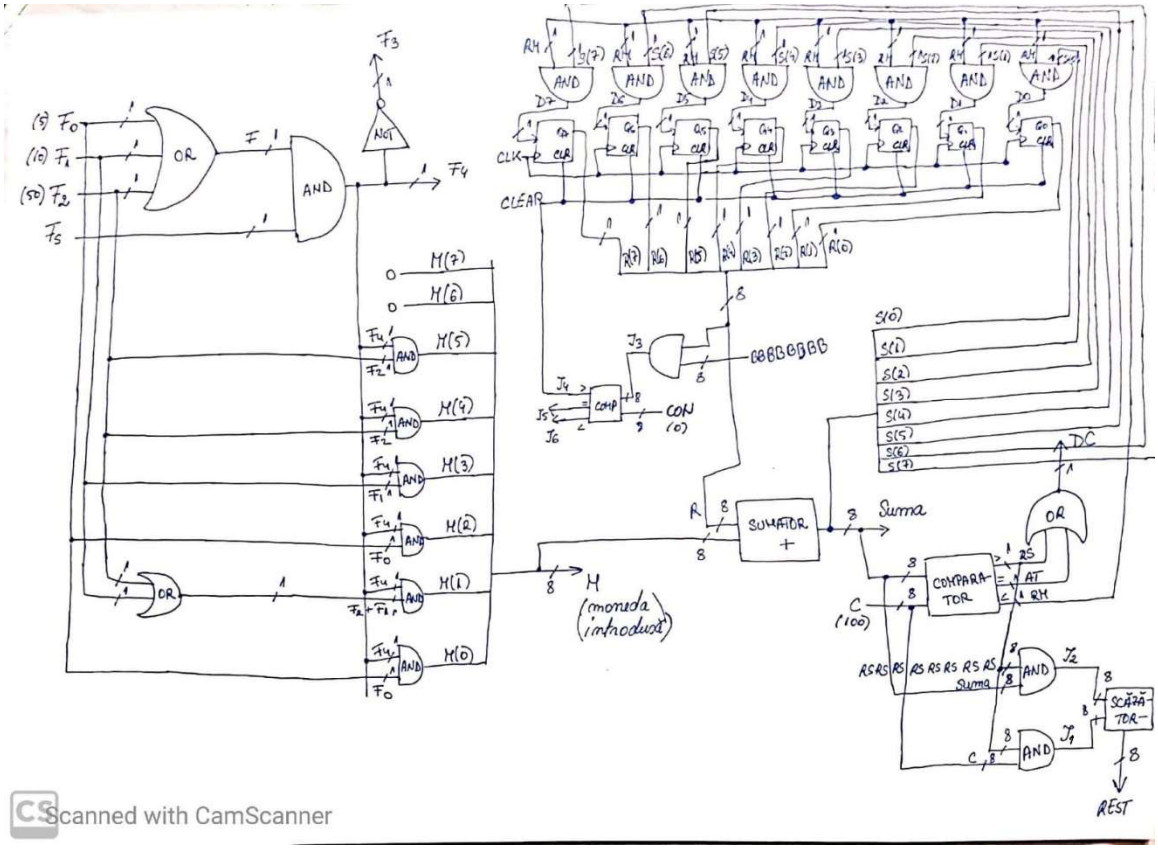
--dacă este mai mică decât 1leu RM=1 => se continuă memorarea sumei

--dacă este mai mare decât 1leu RS=1 =>DC=1-se eliberează cola și se calculează restul care trebuie dat

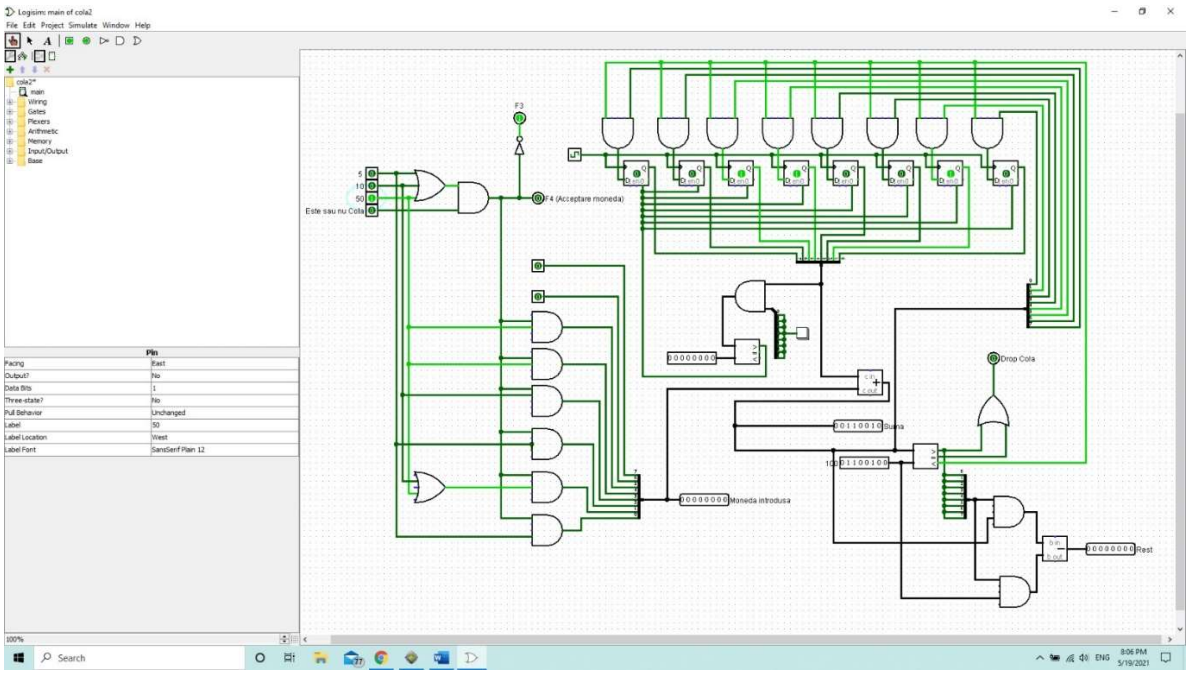
Punem într-un scăzător o poartă ȘI între sumă și RS (când RS=1 se pune în scăzător suma, altfel se pune 0) din care scădem o poartă ȘI între 100 de bani și RS (când RS=1 se scade 100, altfel 0) pentru a afla cât rest trebuie dat.

Mai avem o poartă ȘI între suma din registru și intrarea B (când B=1, s-au terminat de introdus monedele, iar suma este mai mică decât 1 leu, altfel este 0). Dacă suma este mai mare decât 0 (se pune într-un comparator) înseamnă că trebuie returnată valoarea din registru și se resetează registrul.

- Schema logică:



Schema logică în logisim:



4.3 Etapele de proiectare:

- Prima dată am identificat intrările și ieșirile, am alcătuit schema prezentată la Unitatea de comandă și m-am gândit la componentele care m-ar ajuta să o alcătuiesc.
- După ce am identificat componentele, am scris schema logică prezentată mai sus, inițial niște schițe pe hârtie, apoi am simulat-o si testat-o în logisim.
- După ce am verificat schema cu multe cazuri, am început să scriu codul în active-hdl. Întâi am scris codul pentru fiecare componentă în parte, după care le-am pus împreună.
- La final, am simulat cât mai multe cazuri pentru a verifica corectitudinea codului. A fost destul de dificil, abia după un timp mi-am dat seama că trebuie să am grijă cum introduc datele în simularea în vhdl, deoarece o monedă trebuie introdusă doar când CLK=0, iar când CLK=1 nu trebuie introdus nimic.

4.4 Codul în vhdl:

- Poarta SAU3:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity POARTA_SAU3 is
    port(A,B,CIN: in std_logic;
    COUT: out std_logic);
end entity POARTA_SAU3;
architecture flux of POARTA_SAU3 is
begin
    COUT <= A or B or CIN;
    end architecture flux;
```

- Poarta SAU2:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity POARTA_SAU is
    port(A,B: in std_logic;
    COUT: out std_logic);
end entity POARTA_SAU;
architecture flux of POARTA_SAU is
begin
    COUT <= A or B;
    end architecture flux;
```

- Poarta ȘI:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity POARTA_SI is
    port(A,B: in std_logic;
    Y: out std_logic);
end entity POARTA_SI;
architecture flux of POARTA_SI is
begin
    Y<= A and B;
end architecture flux;
```

- Poarta ȘI8:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity POARTA_SI8 is
    port(A,B: in std_logic_vector(7 downto 0);
    Y: out std_logic_vector(7 downto 0));
end entity POARTA_SI8;
architecture flux of POARTA_SI8 is
begin
    Y<= A and B;
end architecture flux;
```

- Poarta NOT:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity POARTA_NOT is
    port(A: in std_logic;
    Y: out std_logic);
end entity POARTA_NOT;
architecture flux of POARTA_NOT is
begin
    Y<= not A;
end architecture flux;
```

- Sumator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity SUMATOR is
    port(A,B: in unsigned(7 downto 0):="00000000";
    S:out unsigned(7 downto 0));
end entity SUMATOR;
architecture comp of SUMATOR is
begin
    process(A,B)
    begin
        S <= A+B;
    end process;
    end architecture comp;
```

- Comparator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity COMPARATOR is
    port(A,B: in unsigned(7 downto 0);
    F1,F2,F3:out std_logic);
end entity COMPARATOR;
architecture comp of COMPARATOR is
begin
    process(A,B)
    begin
            if A>B then F1<='1'; F2<='0';
F3<='0';
            elsif A=B then F1<='0'; F2<='1';
F3<='0';
            elsif A<B then F1<='0'; F2<='0';
F3<='1';
            end if;
            end process;
    end architecture comp;
```

- Scăzător:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity SCAZATOR is
    port(A,B: in unsigned(7 downto 0);
    S:out unsigned(7 downto 0));
end entity SCAZATOR;
architecture comp of SCAZATOR is
begin
    process(A,B)
    begin

            S <= A-B;
    end process;
    end architecture comp;
```

- Bistabil D:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity BISTABIL_D is
    port ( CLEAR, CLK : in std_logic;
           D: in std_logic;
           Q : out std_logic:='0');
end BISTABIL_D;
architecture comp of BISTABIL_D is
begin
    process(CLK, D, CLEAR)
    begin
      if CLEAR ='1' then Q <='0';
      elsif CLK'event and CLK='1' then Q <= D;


      end if;
    end process;
end architecture;
```

- Proiect:

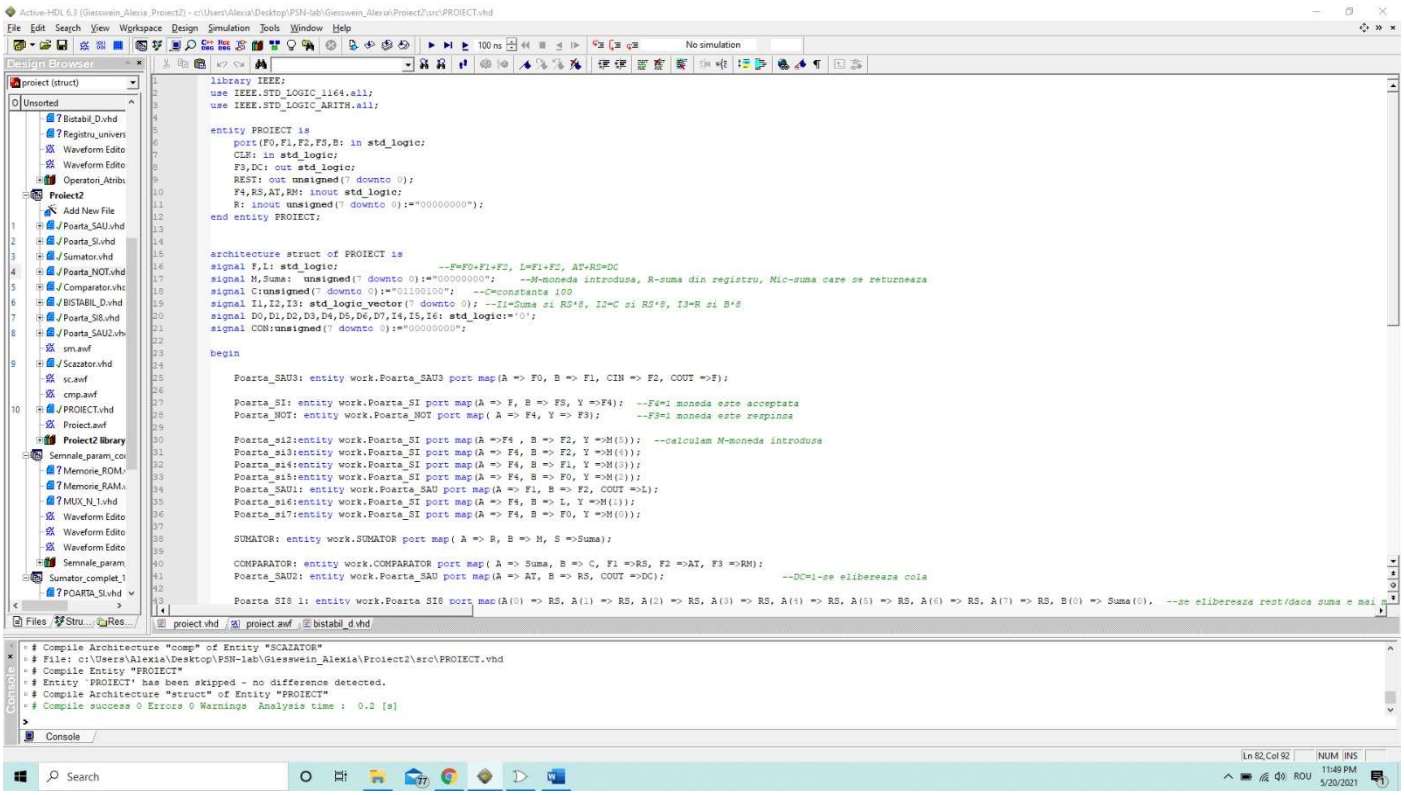
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;


entity PROIECT is
    port(F0,F1,F2,FS,B: in std_logic;
    CLK: in std_logic;
    F3,DC: out std_logic;
    REST: out unsigned(7 downto 0);
    F4,RS,AT,RM: inout std_logic;
    R: inout unsigned(7 downto 0):="00000000");
end entity PROIECT;



architecture struct of PROIECT is
signal F,L: std_logic;                                  --F=F0+F1+F2, L=F1+F2, AT+RS=DC
signal M,Suma:  unsigned(7 downto 0):="00000000";     --M-moneda introdusa, R-suma din registru, Mic-suma care se returneaza
signal C:unsigned(7 downto 0):="01100100";      --C=constanta 100
signal I1,I2,I3: std_logic_vector(7 downto 0); --I1=Suma si RS*8, I2=C si RS*8, I3=R si B*8
signal D0,D1,D2,D3,D4,D5,D6,D7,I4,I5,I6: std_logic:='0';
signal CON:unsigned(7 downto 0):="00000000";
```

```
    begin

        Poarta_SAU3: entity work.Poarta_SAU3 port map(A => F0, B => F1, CIN => F2, COUT =>F);

        Poarta_SI: entity work.Poarta_SI port map(A => F, B => FS, Y =>F4);       --F4=1 moneda este
acceptata

        Poarta_NOT: entity work.Poarta_NOT port map( A => F4, Y => F3);                  --F3=1 moneda
este respinsa

        Poarta_si2:entity work.Poarta_SI port map(A =>F4 , B => F2, Y =>M(5));--calculam M-moneda
introdusa

        Poarta_si3:entity work.Poarta_SI port map(A => F4, B => F2, Y =>M(4));

        Poarta_si4:entity work.Poarta_SI port map(A => F4, B => F1, Y =>M(3));

        Poarta_si5:entity work.Poarta_SI port map(A => F4, B => F0, Y =>M(2));

        Poarta_SAU1: entity work.Poarta_SAU port map(A => F1, B => F2, COUT =>L);

        Poarta_si6:entity work.Poarta_SI port map(A => F4, B => L, Y =>M(1));

        Poarta_si7:entity work.Poarta_SI port map(A => F4, B => F0, Y =>M(0));

        SUMATOR: entity work.SUMATOR port map( A => R, B => M, S =>Suma);

        COMPARATOR: entity work.COMPARATOR port map( A => Suma, B => C, F1 =>RS, F2
=>AT, F3 =>RM);

        Poarta_SAU2: entity work.Poarta_SAU port map(A => AT, B => RS, COUT =>DC);
                        --DC=1-se elibereaza cola

        Poarta_SI8_1: entity work.Poarta_SI8 port map(A(0) => RS, A(1) => RS, A(2) => RS, A(3) => RS,
A(4) => RS, A(5) => RS, A(6) => RS, A(7) => RS, B(0) => Suma(0),       --se elibereaza rest(daca suma e
mai mare)

                B(1) => Suma(1), B(2) => Suma(2), B(3) => Suma(3), B(4) => Suma(4), B(5) => Suma(5),
B(6) => Suma(6), B(7) => Suma(7), Y =>I1);

        Poarta_SI8_2: entity work.Poarta_SI8 port map(A(0) => RS, A(1) => RS, A(2) => RS, A(3) => RS,
A(4) => RS, A(5) => RS, A(6) => RS, A(7) => RS, B(0) => C(0),

                B(1) => C(1), B(2) => C(2), B(3) => C(3), B(4) => C(4), B(5) => C(5), B(6) => C(6), B(7)
=> C(7), Y =>I2);

        SCAZATOR: entity work.SCAZATOR port map( A(0) => I1(0),  A(1) => I1(1), A(2) => I1(2),
A(3) => I1(3), A(4) => I1(4), A(5) => I1(5), A(6) => I1(6), A(7) => I1(7),

                B(0)=> I2(0), B(1) => I2(1), B(2) => I2(2), B(3) => I2(3), B(4) => I2(4), B(5) => I2(5),
B(6) => I2(6), B(7) => I2(7), S =>REST);
```

```
Poarta_SI_0: entity work.Poarta_SI port map(A => Suma(0), B => RM, Y =>D0);
        --punem suma anterioara in registru

        BISTABIL_D_0: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D0, Q
=>R(0));

        Poarta_SI_1: entity work.Poarta_SI port map(A => Suma(1), B => RM, Y =>D1);

        BISTABIL_D_1: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D1, Q
=>R(1));

        Poarta_SI_2: entity work.Poarta_SI port map(A => Suma(2), B => RM, Y =>D2);

        BISTABIL_D_2: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D2, Q
=>R(2));

        Poarta_SI_3: entity work.Poarta_SI port map(A => Suma(3), B => RM, Y =>D3);

        BISTABIL_D_3: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D3, Q
=>R(3));

        Poarta_SI_4: entity work.Poarta_SI port map(A => Suma(4), B => RM, Y =>D4);

        BISTABIL_D_4: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D4, Q
=>R(4));

        Poarta_SI_5: entity work.Poarta_SI port map(A => Suma(5), B => RM, Y =>D5);

        BISTABIL_D_5: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D5, Q
=>R(5));

        Poarta_SI_6: entity work.Poarta_SI port map(A => Suma(6), B => RM, Y =>D6);

        BISTABIL_D_6: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D6, Q
=>R(6));

        Poarta_SI_7: entity work.Poarta_SI port map(A => Suma(7), B => RM, Y =>D7);

        BISTABIL_D_7: entity work.BISTABIL_D port map(CLEAR => I4, CLK => CLK, D=>D7, Q
=>R(7));

        Poarta_SI8_3: entity work.Poarta_SI8 port map(A(0) => B, A(1) => B, A(2) => B, A(3) => B, A(4) =>
B, A(5) => B, A(6) => B, A(7) => B, B(0) => R(0), --daca suma e mai mica decat 1 leu, se returneaza I3=R

                B(1) => R(1), B(2) => R(2), B(3) => R(3), B(4) => R(4), B(5) => R(5), B(6) => R(6), B(7) =>
R(7), Y =>I3);

        COMPARATOR2: entity work.COMPARATOR port map( A(0) => I3(0), A(1) => I3(1), A(2) => I3(2),
A(3) => I3(3), A(4) => I3(4), A(5) => I3(5), A(6) => I3(6), A(7) => I3(7), B => CON, F1 =>I4, F2 =>I5, F3
=>I6);

end architecture;
```

5. Instrucțiuni de utilizare:

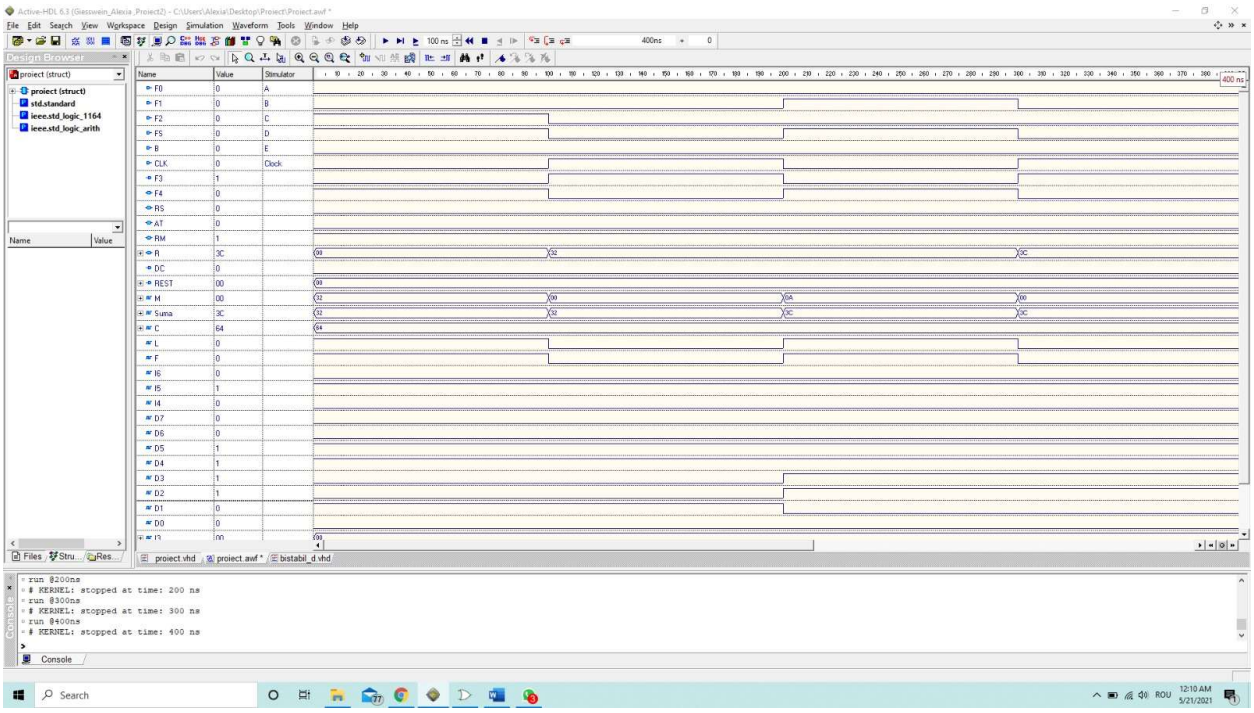
Utilizatorul deschide programul Active HDL și alege proiectul. În partea stângă se vor găsi codurile pentru fiecare componentă.



Pentru a inițializa simularea, întâi trebuie compilat proiectul de la “compile all”. Dacă nu este nicio eroare, trebuie mers la “Simulation” -> “Iniatialize simulation” pentru a inițializa simularea. Apoi se dă click pe “waveform” pentru a începe simularea. În waveform trebuie adăugate semnalele ->”Add signals”. La semnalele de intrare, trebuie click dreapta-> “Simulators”pentru a simula circuitul.

La intrările F0,F1,F2,FS și B putem pune “hotkey”și la CLK alegem “clock”.

Apoi putem schimba valorile intrărilor pentru a obține rezulatul dorit. Atenție însă, o valoare (o monedă) se introduce doar atunci când clockul se află pe 0 (CLK=0), iar când clockul este 1 (CLK=1) se introduce 0 (pentru aceasta setăm la clock frecvența de 5MHz). După ce am stabilit cât introducem, se apasă run pentru a începe simularea.



Simulare in vhdl

6. Justificarea soluției alese:

Am ales această soluție, deoarece, după ce am analizat toate celelalte posibilități, aceasta mi s-a părut cea mai eficientă și cea mai simplu de implementat.

La început, nu prea știam cum aș putea calcula suma, deoarece intrările erau pe 1 bit, iar suma pe 8 biți. Inițial m-am gândit să fac un numărător care să numere de câte ori se introduce o monedă, apoi să înmulțesc cu valoarea monedei, însă părea prea complicat. În final am găsit o modalitate ușoară, dar și eficientă de a obține suma, deoarece nu a trebuit sa folosesc multe componente, doar porți logice.

De asemenea a fost mai complicat să scriu suma în registru si să-l resetez, însă am reușit să scriu suma cu ajutorul semnalului de ceas și l-am resetat cu ajutorul ieșirii RM.

În final am folosit doar 4 componente și niște porți logice, care au făcut codul să fie mai ușor de implementat, de verificat, de urmărit erorile, de înțeles.

În opinia mea, soluția aleasă de mine este una eficientă și ușor de înțeles pentru toată lumea, de aceea am ales s-o implementez.