MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

AUTOMAT DE CUMPARAT BILETE DE TREN

STUDENT:

MOLDOVAN VLAD-MADALIN

COORDONATOR:

OPINCARIU LAURENTIU

Universitate Tehnica din Cluj-Napoca

Facultatea de automatica si calculatoare

Sectia: Calculatoare si tehnologia informatiei

An I , Seria B , Grupa 30217

Continut

4.Semnificatia notatiilor I/O si a semnalelor interne

5.Justificarea solutiei alese

6.Utilizare si rezultate

7.Posibilitati de dezvolatare ulterioara

1.Specificatie proiect

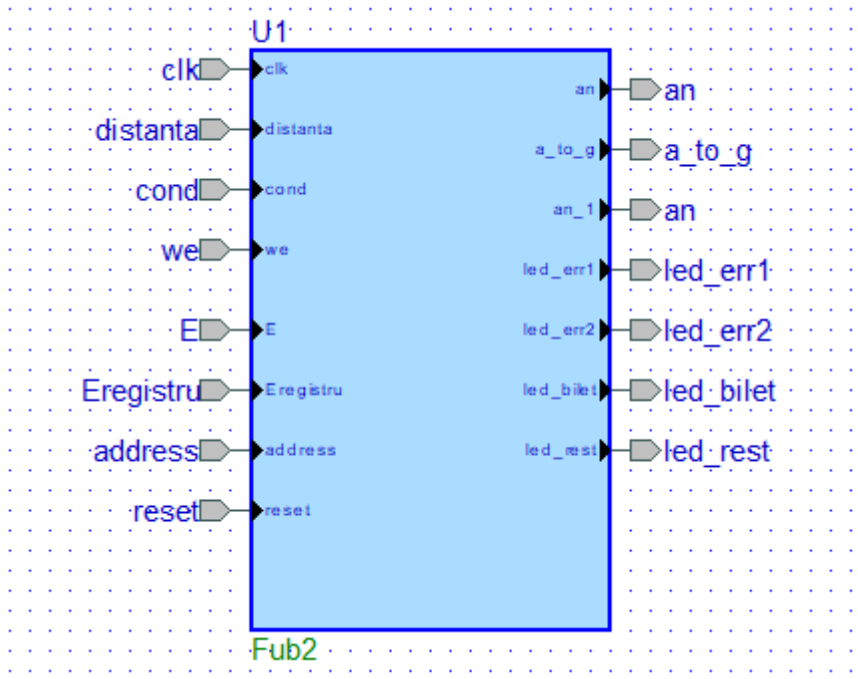
Enunt:

Să se proiecteze un automat pentru cumpărarea biletelor de tren. Cumpărătorul introduce distanţa până la destinaţie (în zeci de km). Costul biletului şi sumele introduse sunt afişate pe afişoare 7 segmente. Moneda utilizată este EURO. Preţul maxim pentru un bilet este de 100 Euro. Automatul primeşte suma necesară în hârtii sau monede şi eliberează biletele şi, eventual, restul. El dispune de o casă de bani care se încarcă la începutul funcţionării cu un număr de hârtii şi monede (toate posibilităţile între 1 euro şi 50 de euro). Lipsa de bilete, introducerea unei sume mai mici decât costul biletului sau imposibilitatea restituirii restului se semnalizează luminos. Se poate renunţa în orice moment la operaţie, cu restituirea sumei introduse, dacă este cazul.

Proiectul va fi realizat de 1 student.

2.Descriere schema bloc cu componente

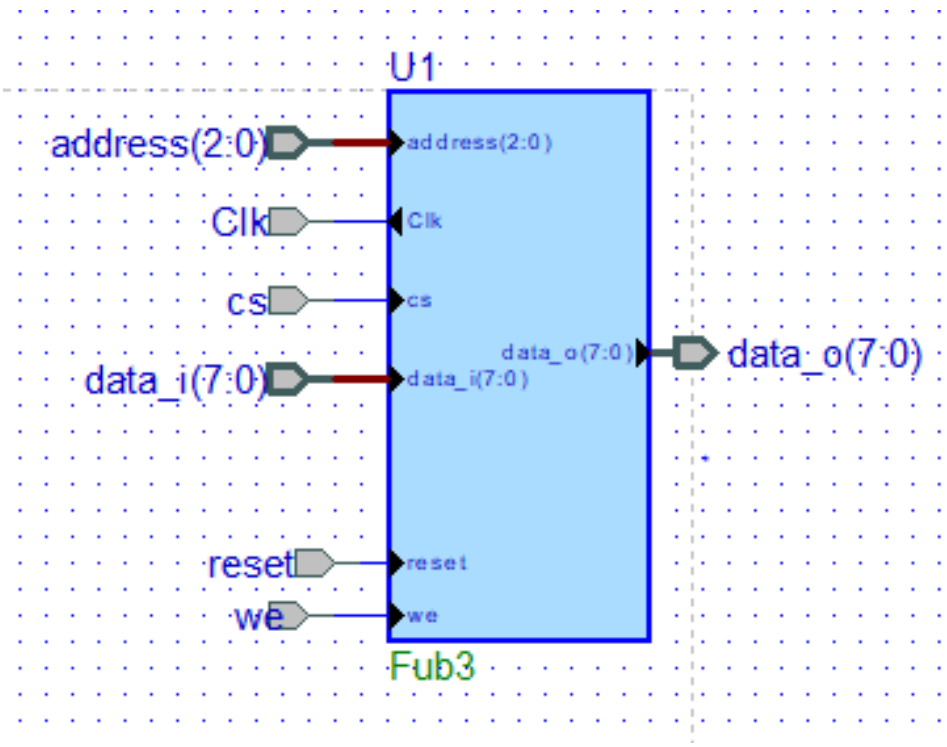
Cutia neagra:



Componente principale:

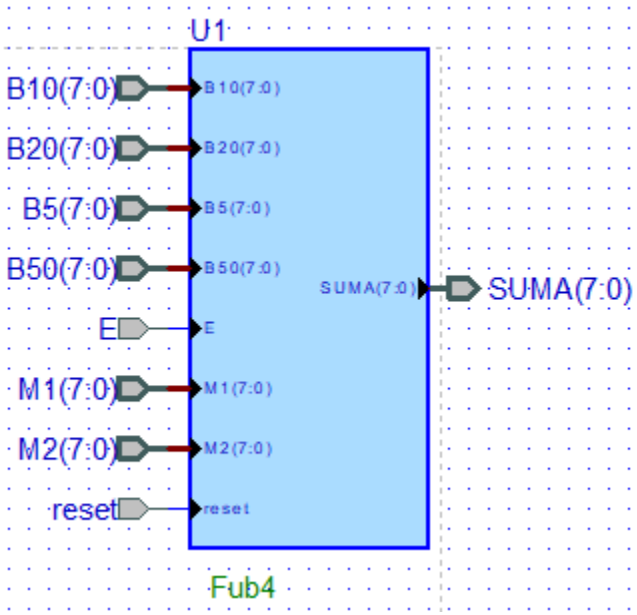
Memoria Ram

Aceasta componenta este folosita pentru a stoca numarul bancnotelor , monedelor si a biletelor de tren.



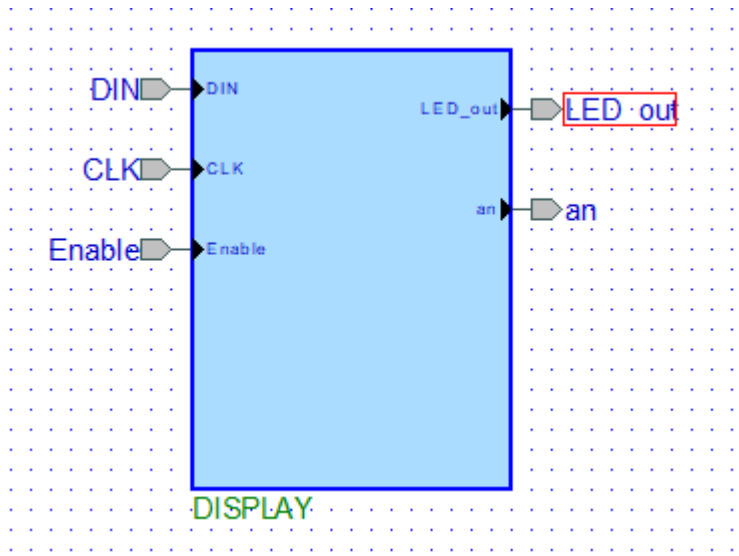
Suma Totala

Aceasta componenta este folosita pentru a insuma monedele si bancnotele care au fost introduse pentru a obtine suma finala introdusa.



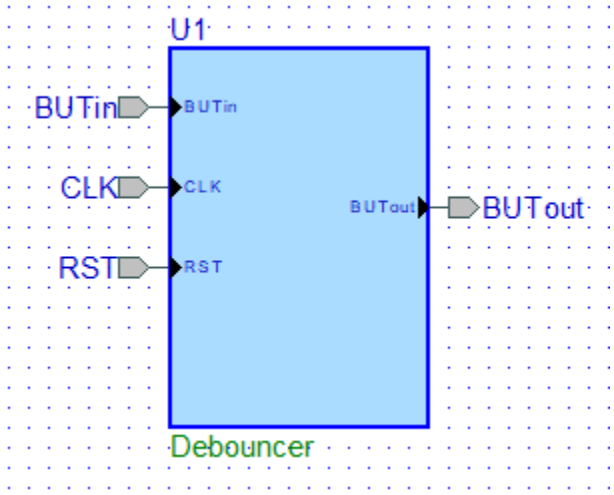
Display

Aceasta componenta este utilizata pentru a controla 7-segment display de pe placa FPGA.



Debouncer

Cu ajutorul acestei componente vom obtine semnale stabile pentru utilizarea butoanelor de pe placuta FPGA.



3.Lista de componente utilizate si implementare

Registru:

Aceasta componenta este folosita pentru a fi introdusa distanta de catre cumparator.Contine si un buton de reset pentru cazul in care a fost introdusa gresit distanta initial.

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.all;


entity reg is

        port (data_in: in std_logic_vector (7 downto 0);

        reset: in std_logic;

        enable: in std_logic;
```

```
        clk: in std_logic;
        data_out: out std_logic_vector (7 downto 0));
end reg;

architecture arh_reg of reg is
begin
process (clk, reset)
begin
        if clk='1' and clk'event then
                if enable='1' then data_out <= data_in;
                        end if;
                if reset='1' then
                 data_out<="00000000";
                end if;
        end if;
end process;
end arh_reg;
```

Comparator:

Este utilizat pentru a verifica daca suma introdusa este mai mare decat costul , caz afirmativ tranzactia va putea continua.

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

```
entity comparator is
        port(a: in std_logic_vector(7 downto 0);
        b: in std_logic_vector(7 downto 0);
        c:out std_logic_vector(1 downto 0));
end comparator;

architecture arh_comp of comparator is
begin
        process(a,b)
        begin

        if (a>b) then c<="01";
                elsif(a<b) then c<="10";
        elsif (a=b) then c<="11";
        end if;
        end process;
end arh_comp;
```

Scazator:

Este folosit pentru a realiza operatia "-" pe std_logic_vector.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_std.ALL;
```

```
entity scazator is
        port(A: in std_logic_vector (7 downto 0);
        B: in std_logic_vector (7 downto 0);
        rest: out std_logic_vector (7 downto 0));
end scazator;

architecture arh_scazator of scazator is
begin
        process(A,B)
        begin
                rest <= std_logic_vector(unsigned(A) - unsigned(B));
        end process;
end arh_scazator;
```

Sumator:

Este folosit pentru a realiza operatia “+” pe std_logic_vector.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_std.ALL;

entity sumator is
        port(A: in std_logic_vector (15 downto 0);
        B: in std_logic_vector (7 downto 0);
        total: out std_logic_vector (15 downto 0));
```

```
end sumator;


architecture arh_sumator of sumator is
begin
        process(A,B)
        begin
                total <= std_logic_vector(unsigned(A) + unsigned(B));
        end process;
end arh_sumator;
```

Memoria RAM:

In plus fata de cele mentionate mai sus , “address” va semnifica tipul valoarea monedei/bancnotei astfel: “111” => 50 euro , “110” => 20 euro , “101” => 10 euro , “100” => 5 euro , “011” => 2 euro , “010” => 1 euro , iar pentru combinatia “000” se vor retine numarul de bilete.”Data_i” va reprezenta numarul de bancnote/monede introduse pentru fiecare tip “address”.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;


entity memorie is
port (Clk : in std_logic;
                address : in std_logic_vector(2 downto 0);
                reset : in std_logic;
      we : in std_logic;
```

```
                cs : in std_logic;
        data_i : in std_logic_vector(7 downto 0);
        data_o : out std_logic_vector(7 downto 0)
    );
end memorie;

architecture arh_memorie of memorie is

type ram_t is array (7 downto 0) of std_logic_vector(7 downto 0);
signal ram : ram_t := ("00110111",
"00110111",
"00110111",
"00110111",
"00110111",
"00110111",
"00000000",
"00110111");

begin

process(Clk)
begin
   if Clk = '1' and Clk'EVENT  then
                if (CS = '1') then
```

```
if(we='1') then
   ram(conv_integer(address)) <= ram(conv_integer(address)) + data_i;
else
data_o <= ram(conv_integer(address));
   end if;
            end if;
            if (reset = '1') then
                  ram <= ("00110111",
                                    "00110111",
                                    "00110111",
                                    "00110111",
                                    "00110111",
                                    "00110111",
                                    "00000000",
                                    "00110111");
            end if;
      end if;
end process;
end arh_memorie;
```

Suma_totala:

In plus fata de cele mentionate mai sus , aceasta componenta are un buton de reset care poate fi utilizat pentru a reseta suma introdusa.

```
library  IEEE;
```

```
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_std.all;

entity Convertor_Suma is
	port(E: in std_logic;
	reset: in std_logic;
	M1: in std_logic_vector(7 downto 0);
	M2: in std_logic_vector(7 downto 0);
	B5: in std_logic_vector(7 downto 0);
	B10: in std_logic_vector(7 downto 0);
	B20: in std_logic_vector(7 downto 0);
	B50: in std_logic_vector(7 downto 0);
	SUMA: out std_logic_vector(7 downto 0));
end Convertor_Suma;

architecture Arch_Convertor_Suma of Convertor_Suma is
begin
	process(E,M1,M2,B5,B10,B20,B50)
	variable P0,P1,P2,P3,P4,P5,S: std_logic_vector(15 downto 0);
	begin
		if E = '1' then
		P0 := std_logic_vector(unsigned(M1) * "00000001");
		P1 := std_logic_vector(unsigned(M2) * "00000010");
		P2 := std_logic_vector(unsigned(B5) * "00000101");
```

```
                P3 := std_logic_vector(unsigned(B10) * "00001010");
                P4 := std_logic_vector(unsigned(B20) * "00010100");
                P5 := std_logic_vector(unsigned(B50) * "00110010");
                S :=  std_logic_vector(unsigned(P0) + unsigned(P1) + unsigned(P2) +
unsigned(P3) + unsigned(P4) + unsigned(P5));
                end if;
                if reset = '1' then
                        P0 := "0000000000000000";
                        P1 := "0000000000000000";
                        P2 := "0000000000000000";
                        P3 := "0000000000000000";
                        P4 := "0000000000000000";
                        P5 := "0000000000000000";
                        S :=  std_logic_vector(unsigned(P0) + unsigned(P1) + unsigned(P2) +
unsigned(P3) + unsigned(P4) + unsigned(P5));
                end if;
                SUMA <= S(7 downto 0);
        end process;
end Arch_Convertor_Suma;
```

Display:

Aceasta componenta opereaza displayul 7-segment a placutei FPGA prin intermediul LED_out si an .

```
library  IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.NUMERIC_std.all;

entity Display is
	port(DIN: in std_logic_vector(7 downto 0);
	CLK: in std_logic;
	ENABLE: in std_logic;
	LED_out: out std_logic_vector(6 downto 0);
	an: out std_logic_vector(3 downto 0)
	);
end Display;

architecture arh_display of Display is
component Binary_BCD is
	port(DIN: in std_logic_vector(7 downto 0);
		Dout: out std_logic_vector(11 downto 0));
end component;

component divizor is
	generic(N: NATURAL := 4);
	port(CLK: in std_logic;
	CE: in std_logic;
	RST: in std_logic;
	Q: out std_logic_vector(N-1 downto 0));
end component;
```

```
signal Numar: std_logic_vector(11 downto 0);
signal Countout: std_logic_vector(20 downto 0);
signal SEL: std_logic_vector(1 downto 0);
signal RST: std_logic := not ENABLE;
signal Digit: std_logic_vector(3 downto 0);
begin
        C1: Binary_BCD port map(DIN,Numar);
        C2: divizor generic map(N => 21) port map(CLK,'1',RST,Countout);
        SEL <= Countout(20 downto 19);
        process(SEL,ENABLE)
        begin
                if ENABLE = '1' then
                        case SEL is
                                when "00" => Digit <= Numar(3 downto 0); an <= "1110";
                                when "01" => Digit <= Numar(7 downto 4); an <= "1101";

                                when "10" => Digit <= Numar(11 downto 8); an <= "1011";
                                when others => an <= "1111";
                        end case;
                else
                        an <= "1111";
                end if;
        end process;
```

```
    process(Digit)
    begin
        case Digit is
          when "0000" => LED_out <= "0000001"; -- "0"
          when "0001" => LED_out <= "1001111"; -- "1"
          when "0010" => LED_out <= "0010010"; -- "2"
          when "0011" => LED_out <= "0000110"; -- "3"
          when "0100" => LED_out <= "1001100"; -- "4"
          when "0101" => LED_out <= "0100100"; -- "5"
          when "0110" => LED_out <= "0100000"; -- "6"
          when "0111" => LED_out <= "0001111"; -- "7"
          when "1000" => LED_out <= "0000000"; -- "8"
          when "1001" => LED_out <= "0000100"; -- "9"
          when others => LED_out <= "1111111";
        end case;
    end process;
end arh_display;
```

Binary to BCD:

Foloseste metoda “shift add 3”, ia valoarea BCD a unui numar de 8 biti i-l face compatibil pentru a fi afisat pe displayul 7-segment.Aceasta metoda este foarte usor de aplicat : folosim o variabila Z definite ca std_logic_vector 19 down to 0 (lungimea este data de numarul de biti a numarului in binar + 12 , numarul de biti folositi pentru a stoca un numar zecimal de 3 cifre in BCD) in care punem intrarea pe pozitia 10 downto 3 si la final numarul nostrum in BCD va fi pe pozitia 19 downto 8.

```
library  IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_std.all;

entity Binary_BCD is
        port(DIN: in std_logic_vector(7 downto 0);
                Dout: out std_logic_vector(11 downto 0));
end Binary_BCD;

architecture Arch_Binary_BCD of Binary_BCD is
begin
        process(DIN)
        variable Z:std_logic_vector(19 downto 0);
        begin
          for i in 0 to 17 loop
                Z(i) := '0';
          end loop;
          Z(10 downto 3) := DIN;
          for i in 0 to 4 loop
                if Z(11 downto 8) > "0100" then
                        Z(11 downto 8) := std_logic_vector(unsigned(Z(11 downto 8)) +
"0011");
                end if;
                if Z(15 downto 12) > "0100" then
```

```
                    Z(15 downto 12) := std_logic_vector(unsigned(Z(15 downto 12)) +
"0011");
                end if;
                Z(19 downto 1) := Z(18 downto 0);
            end loop;
            Dout <= Z(19 downto 8);
        end process;
end Arch_Binary_BCD;
```

Divizor de frecventa:

Este folosit pentru a stabiliza clock-ul , in special pentru diplayul 7-segment.

```
library  IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_std.all;

entity divizor is
        generic(N: NATURAL := 4);
        port(CLK: in std_logic;
        CE: in std_logic;
        RST: in std_logic;
        Q: out std_logic_vector(N-1 downto 0));
end divizor;

architecture arh_divizor of divizor is
```

```
begin
        process(CLK,RST,CE)
        variable count: std_logic_vector(N-1 downto 0) := (others => '1');
        begin
                if CLK='1' and CLK'EVENT then
                        if RST='1' then
                                count := (others => '0');
                        elsif CE = '1' then
                                count := std_logic_vector(unsigned(count)+"1");
                        end if;
                end if;

                Q <= count;
        end process;
end arh_divizor;
```

Debouncer:

Este un debouncer simplu folosit cu scopul de a stabiliza valoarea inputului BUTin si sa o transmita print BUTout.

```
library  IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_std.all;

entity Debouncer is
```

```
        port(BUTin: in std_logic;
                BUTout: out std_logic;
                RST: in std_logic;
                CLK: in std_logic);
end Debouncer;

architecture arh_debouncer of Debouncer is
component divizor is
        generic(N: NATURAL := 4);
        port(CLK: in std_logic;
        CE: in std_logic;
        RST: in std_logic;
        Q: out std_logic_vector(N-1 downto 0));
end component;
signal delay1,delay2,delay3: std_logic;
signal Q: std_logic_vector(21 downto 0);
signal BUT1,BUT2: std_logic;
begin
        L1: divizor generic map (N => 22) port map (CLK,'1',RST,Q);
        process(Q(21),RST,BUTin,BUT2)
        begin
                if RST = '1' or BUTin = '0' or BUT2 = '1' then
                        delay1 <= '0';
                        delay2 <= '0';
```

```
                delay3 <= '0';


            elsif Q(21) = '1' and Q(21)'event then
                delay1 <= BUTin;
                delay2 <= delay1;
                delay3 <= delay2;
            end if;
        end process;
        BUT1 <= delay1 and delay2 and delay3;
        process(CLK)
        begin
            if RST = '0' then
                if CLK = '1' and CLK'event then
                    BUT2 <= BUT1;
                end if;
            else
                BUT2 <= '0';
            end if;
        end process;
        BUTout <= BUT2;
end arh_debouncer;
```

MAIN:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity automat is
        port(clk: in std_logic;
        distanta: in std_logic_vector (7 downto 0);
        cond: in std_logic_vector(1 downto 0);
        we,E,Eregistru,reset: in std_logic;
        address: in std_logic_vector (2 downto 0);
        a_to_g: out std_logic_vector (6 downto 0);
        an: out std_logic_vector (3 downto 0);
        led_err1 : out std_logic;
        led_err2 : out std_logic;
        led_err3 : out std_logic;
        led_bilet : out std_logic;
        led_rest : out std_logic);
end automat;

architecture arh_automat of automat is

component reg is
        port (data_in: in std_logic_vector (7 downto 0);
        reset: in std_logic;
        enable: in std_logic;
```

```
        clk: in std_logic;
        data_out: out std_logic_vector (7 downto 0));
end component;

component comparator is
        port(a: in std_logic_vector(7 downto 0);
        b: in std_logic_vector(7 downto 0);
        c:out std_logic_vector(1 downto 0));
end component;

component scazator is
        port(A: in std_logic_vector(7 downto 0);
                B: in std_logic_vector(7 downto 0);
                rest : out std_logic_vector(7 downto 0));
end component;

component sumator is
        port(A: in std_logic_vector (15 downto 0);
        B: in std_logic_vector (7 downto 0);
        total: out std_logic_vector (15 downto 0));
end component;

component memorie is
port (Clk : in std_logic;
```

```
                address : in std_logic_vector(2 downto 0);
                reset : in std_logic;
        we : in std_logic;
                cs : in std_logic;
        data_i : in std_logic_vector(7 downto 0);
        data_o : out std_logic_vector(7 downto 0)
    );
end component;

component Convertor_Suma is
        port(E: in std_logic;
        reset :  in std_logic;
        M1: in std_logic_vector(7 downto 0);
        M2: in std_logic_vector(7 downto 0);
        B5: in std_logic_vector(7 downto 0);
        B10: in std_logic_vector(7 downto 0);
        B20: in std_logic_vector(7 downto 0);
        B50: in std_logic_vector(7 downto 0);
        SUMA: out std_logic_vector(7 downto 0));
end component;

component Display is
        port(DIN: in std_logic_vector(7 downto 0);
        CLK: in std_logic;
```

```
        ENABLE: in std_logic;
        LED_out: out std_logic_vector(6 downto 0);
        an: out std_logic_vector(3 downto 0));
end component;

component Debouncer is
        port(BUTin: in std_logic;
                BUTout: out std_logic;
                RST: in std_logic;
                CLK: in std_logic);
end component;

signal afisare: std_logic_vector(7 downto 0);

signal data_o: std_logic_vector (7 downto 0);
signal ok: std_logic_vector (1 downto 0);
signal ram : std_logic_vector (7 downto 0);
signal cost: std_logic_vector (7 downto 0);
signal rest: std_logic_vector (7 downto 0);

signal M1: std_logic_vector(7 downto 0) := "00000000";
signal M2: std_logic_vector(7 downto 0) := "00000000";
signal B5: std_logic_vector(7 downto 0) := "00000000";
signal B10: std_logic_vector(7 downto 0) := "00000000";
```

```
signal B20: std_logic_vector(7 downto 0) := "00000000";
signal B50: std_logic_vector(7 downto 0) := "00000000";
signal SUMA: std_logic_vector(7 downto 0);
signal total: std_logic_vector(15 downto 0);
signal dreset,RST: std_logic;
begin
        c0: Debouncer port map (reset,dreset,RST,clk);
        c1: reg port map (distanta,dreset,Eregistru,clk,cost);
        c2: memorie port map (clk,address,dreset,we,'1',distanta,data_o);
        c3: Convertor_Suma port map(E,dreset,M1,M2,B5,B10,B20,B50,SUMA);
        c4: comparator port map (SUMA,cost,ok);
        c5: scazator port map (SUMA,cost,rest);
        c6: sumator port map("0001000111010000",SUMA,total);
        c7: display port map(afisare,clk,'1',a_to_g,an);

        process(cond)
        begin
                case cond is
                        when "11" => afisare <= cost;
                        when "01" => afisare <= suma;
                        when "10" => afisare <= rest;
                        when others => afisare <= "00000000";
                end case;
        end process;
```

```
process(address)
begin
        case address is
                when "111" => B50 <= distanta;
                when "110" => B20 <= distanta;
                when "101" => B10 <= distanta;
                when "100" => B5  <= distanta;
                when "011" => M2 <= distanta;
                when "010" => M1 <= distanta;
                when others => NULL;
        end case;
end process;
process(ok,data_o)
begin
if(data_o > "00000000") then
        led_err1 <= '0';
else
        led_err1 <= '1';
end if;
if(ok="10") then
        led_err2<='1';
        led_rest<='0';
        led_bilet<='0';
```

```
        elsif (ok="01") then
                led_err2<='0';
                led_rest<='1';
                if(total - rest > "00000000") then
                        led_err3 <='0';
                else
                        led_err3 <='1';
                end if;
                led_bilet<='1';
        elsif   (ok="11") then
                led_err2<='0';
                led_rest<='0';
                if(SUMA > "00000000")  then
                        led_bilet<='1';
                else
                        led_bilet<='0';
                end if;
        end if;
        end process;
end arh_automat;
```

4.Semnificatia notatiilor

Registru:

data_in: distanta introdusa

reset: buton de reset

enable: enable pentru componenta

clk: semnalul de clock de pe FPGA

data_out: distanta memorata

Comparator:

a: suma introdusa

b: costul biletului

c: rezultatul comparatiei

Memorie_ram:

clk: semnalul de clock de pe FPGA

address: tipul bancnotei/monedei

reset: buton de reset

we: write enable

cs: chip select

data_i: numarul de bancnote/monede introduse

data_o: outputul memoriei

Suma_totala:

E: enable pentru componenta

Reset: buton de reset

M1: numarul de monede de 1 euro introduse

M2: numarul de monede de 2 euro introduse

B10: numarul de bancnote de 10 euro introduse

B20: numarul de bancnote de 20 euro introduse

B50: numarul de bancnote de 50 euro introduse

SUMA: suma totala introdusa.

Scazator:

A: suma introdusa

B: distanta introdusa

rest: rezultatul A-B

Sumator:

A: suma din casa de bani in stare initiala

B: suma introdusa

Total: rezultatul A+B

Display:

DIN: numarul care trebuie afisat

CLK: semnalul de clock de pe FPGA

ENABLE: enable pentru componenta

LED_out: controleaza semnalul catozilor

an: controleaza semnalul anozilor

Divizor_frecventa:

CLK: semnalul de clock de pe FPGA

CE: enable pentru semnalul de clokc

RST: buton de reset

Q: output

Binary to bcd:

DIN: numarul pe 8 biti in binar

Dout: numarul corespunzator in BCD pe 12 biti

Debouncer:

BUTin: semnalul initial

BUTout: semnalul stabilizat

RST: buton de reset pentru componenta

CLK: semnalul de clock de pe FPGA

MAIN:

clk: semnalul de clock de pe FPGA

distanta: distanta introdusa / numarul de bancnote de pe fiecare tip

cond: conditia pentru afisor : “11” => distanta introdusa” , “01” => suma introdusa , “10” => restul.

we: write enable (pentru memoria ram)

E: enable pentru component “suma_totala”

Eregistru: enable pentru registru

reset: buton de reset

address: tipul bancnotei introduse

a_to_g: controleaza semnalul catozilor

an: controleaza semnalul anozilor

led_err1: led pentru lipsa bilete

led_err2: led pentru cazul in care suma introdusa este mai mica decat costul biletuli

led_err3: led care semnaleaza imposibilitatea de a da rest

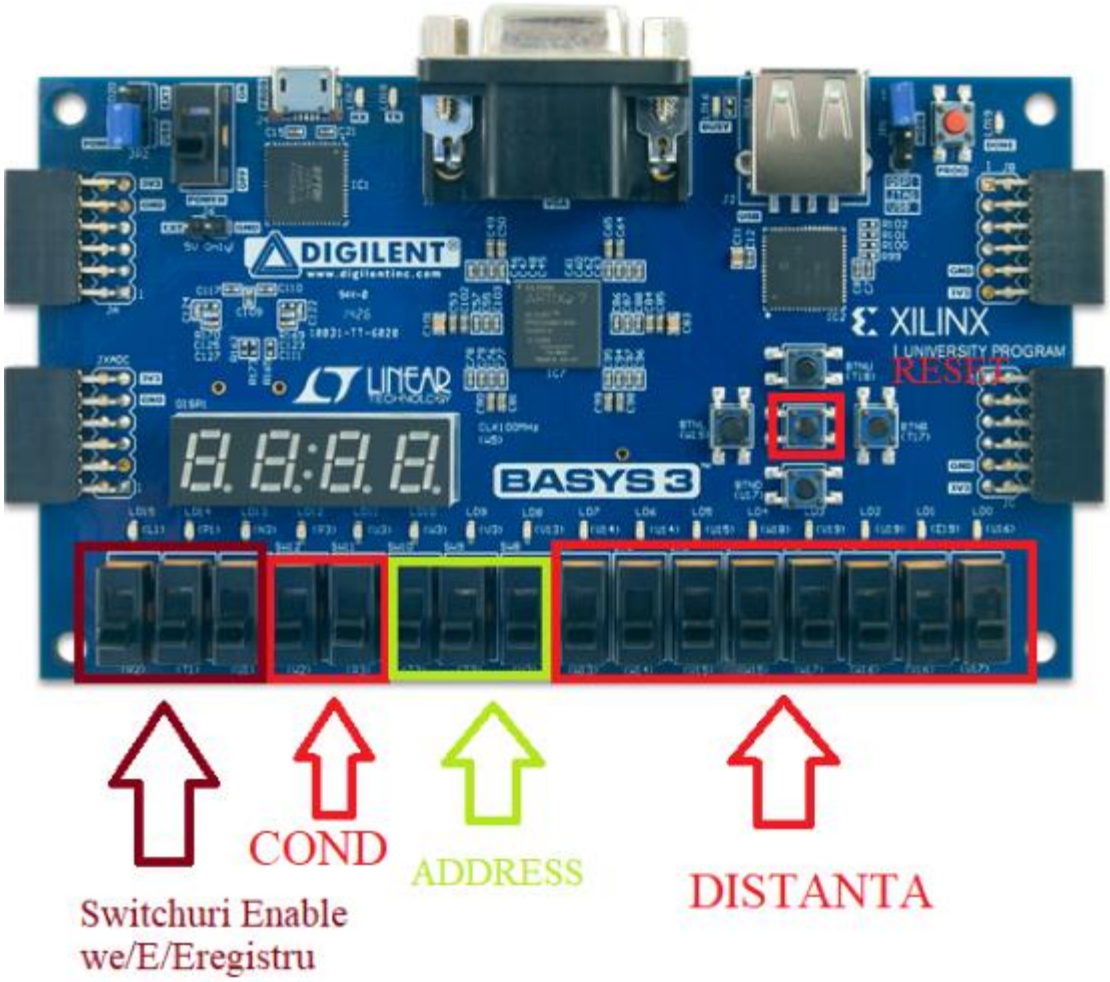
led_bilet: led pentru a semnala eliberarea biletului

led_rest: led pentru a semnala eliberarea restului

5.Justificarea solutiei alese

Proiectul consta in implementarea unui automat , iar varianta pe care am ales sa o implementez se bazeaza pe controlul din switchuri.O data ce am ales modul in care vreau sa implementez automatul , am incercat sa-l construiesc cat mai simplu si fara a ma complica prea mult.Aproape toate datele de intrare se afla pe switchuri (distanta,tipul bancnotei/monedei , numarul de bancnote/monede,controlul pentru afisor) pentru a face automatul cat mai usor de folosit pentru utilizator.

6.Utilizare si rezultate



Folosind maparea din imaginea de mai sus utilizarea automatului va fi mai clara.Prima data se va introduce distanta astfel: se va activa “Eregistru” dupa care se va introduce distanta prin switchurile “distanta” dupa care se va dezactiva “Eregistru”.Al doilea pas consta in introducerea bancnotelor sau monedelor prin activarea “E”,”we” si introducerea combinatiei corespunzatoare pe switchurile “address” (pentru valoare bancnotei / monedei pe care vrem sa o introducem) , iar pe switchurile “distanta” vom introduce numarul de bancnote / monde pe care dorim sa le introducem.Pentru a folosi afisorul 7 segment vom introduce pe switchurile “cond” combinatia corespunzatoare pentru ce dorim sa afisam(11=>costul biletului, 01=>suma

introdusa,10=>restul).In cazul in care s-a introdus ceva gresit , se poate folosi butonul de reset pentru a relua operatiile.

7.Posibilitati de dezvoltare ulterioara

Una dintre imbunatatirile care ar face acest automat mai functional este posibilitate de a lucra cu numere reale , nu doar intregi.Aceasta imbunatatire ar face automatul mult mai practic in zilele noastre .