



**FACULTATEA: Automatică si Calculatoare**  
**SPECIALIZAREA: Calculatoare si Tehnologia Informației**  
**DISCIPLINA: Proiectarea sistemelor numerice**  
**PROIECT: Transmisie de date**

**Îndrumător Laborator:**

Opincariu Laurențiu

**Realizatori:**

Bakk Cosmin-Robert

Bâlc Horia-Ovidiu

# Cuprins

Specificația proiectului.....	3
Schema bloc.....	5
Proiectare și implementare.....	6
Lista de componente utilizate.....	18
Semnificația notațiilor I/O și a semnalelor interne.....	19
Justificarea soluției alese.....	25
Utilizare și rezultate.....	26
Posibilități de dezvoltare ulterioară.....	32

# 1. Specificația proiectului

## Cerința:

Să se proiecteze un sistem de comunicație care realizează transmiterea serială a datelor. Datele sunt transmise folosind pachete de date. Sistemul va implementa un generator de pachete de date și un detector care va verifica pachetul.

## Transmisie de date

Pentru a trimite informația în **pachete de date**, un sistem de comunicație utilizează o **linie serială** de date.

Fiecare pachet de date este format din: Segment de Start, Segment de Date și Sumă de Control. Împreună cu pachetele de date este transmis și un semnal de tact (clock) pentru sincronizare, activ pe frontul crescător. Formatul unui pachet de date este:

**Segment de Start** - 7 biți, împărțiți în 2 zone: un *Bit de Start*, întotdeauna cu valoarea 0 logic și *Cod de Start*, format din 6 biți.

**Segment de Date** - 16 biți grupați în 4 cuvinte de 4 biți, fiecare cuvânt poate lua valori de la 0 la 15.

**Suma de Control** - 4 biți care reprezintă rezultatul operației de SAU-EXCLUSIV între cele 4 cuvinte de 4 biți din Segmentul de Date.

Pentru acest sistem de comunicație se va realiza un prototip format din 2 blocuri logice, un **Detector** și un **Generator**.

Detectorul va funcționa doar la un anumit Cod de Start. El preia pachetul de date, calculează pentru Segmentul de Date suma de control și compară rezultatul obținut cu cel primit în Suma de Control.

Detectorul generează următoarele semnale de ieșire:

**Semnalizare Start SS** - ia valoarea 1 logic în momentul în care se detectează Bitul de Start al Segmentului de Start și rămâne în 1 logic, dacă s-a detectat Codul de Start corect, până la terminarea primirii pachetului de date; ia valoarea 0 logic imediat ce se constată că nu a sosit un Cod de Start corect.

**Semnalizare Mesaj SM** - ia valoarea 1 logic la începutul Segmentului de Date și rămâne în 1 logic până se termină primirea pachetului de date.

**Semnalizare control SC** - este setat inițial (la începutul unui pachet de date) la 0 logic; ia valoarea 1 logic doar dacă există identitate între suma de control calculată și cea primită și rămâne în 1 logic până la sosirea unui nou pachet de date.

**Final** - este setat inițial la 0 logic; ia valoarea 1 logic la terminarea primirii pachetului de date.

Semnalul de intrare **Reset** va inițializa Detectorul la un punct de pornire cunoscut și va pune toate ieșirile la valoarea 0 logic.

Generatorul trebuie să demonstreze că sistemul de recepție funcționează corect. El trimite Detectorului 2 pachete de date diferite. Funcționarea lui este coordonată de semnalul de control **Mod**, pe 2 biți.

Dacă Mod = 00 se trimite un Segment de Start corect, primul Segment de Date și Suma de Control.

Dacă Mod = 01 se trimite un Segment de Start corect, al doilea Segment de Date și Suma de Control.

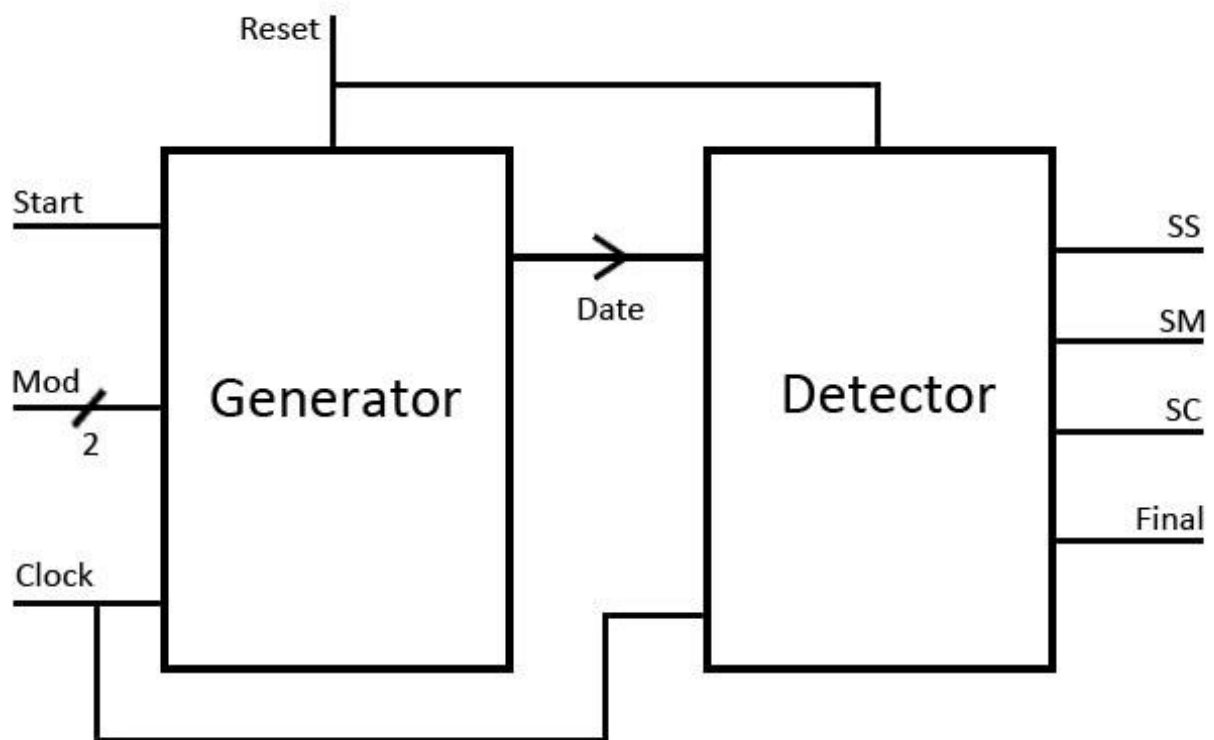
Dacă Mod = 10 se trimite un Segment de Start incorect, primul Segment de Date și Suma de Control.

Dacă Mod = 11 se trimite un Segment de Start corect, al doilea Segment de Date și o Sumă de Control greșită.

Semnalul de intrare **Reset** va inițializa și Generatorul.

Un semnal de intrare numit **Start** va fi folosit pentru începerea funcționării prototipului.

## 2. Schema bloc



### Intrări:

**Start (switch):** intrare pe 1 bit care e folosită pentru începerea funcționării prototipului.

**Mod (switch-uri):** intrare pe 2 biți care indică modul de generare al pachetului de date.

**Reset (switch):** intrare pe 1 bit care comandă resetarea aparatului.

**Clock (clock):** semnal de tact.

### Ieșiri:

SS (led): ieșire pe 1 bit care ia valoarea 1 logic la primirea unui Cod de Start corect.

SM(led): ieșire pe 1 bit care ia valoarea 1 logic la începerea primirii Segmenului de Date.

SC(led): ieșire pe 1 bit care ia valoarea 1 logic la detectarea Sumei de Control corecte.

Final(led): ieșire pe 1 bit care ia valoarea 1 logic la terminarea primirii pachetului de date.

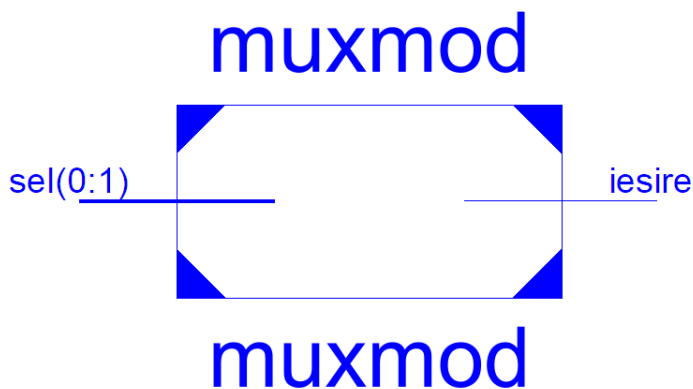
## 3. Proiectare și implementare

Am folosit libraria IEEE si pachetul 1164 inainte de fiecare entitate.

### Componente:

#### Mux mod:

Această componentă verifică dacă modul introdus este "10", caz în care codul de start trebuie să fie generat gresit. Ieșirea este '1' doar în cazul în care modul este "10".



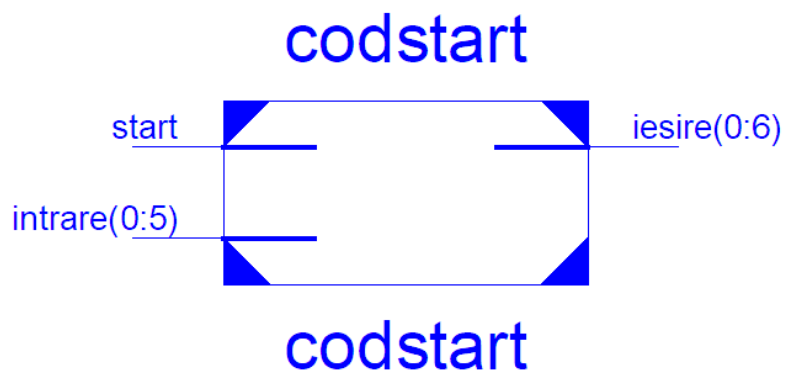
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity muxmod is
    port(sel:in std_logic_vector(0 to 1);
          iesire:out std_logic);
end muxmod;

architecture arh of muxmod is
begin
    process(sel)
    begin
        if (sel="10") then
            iesire<='1';
        else iesire<='0';
        end if;
    end process;
end arh;
```

### Cod Start:

Aceasta componenta formeaza Segmentul de Start, punand pe Bitul de Start valoarea '0', iar pe urmatoarele pozitii Codul de Start, generat corect sau gresit in functie de intrarea primita din componenta Mux mod.



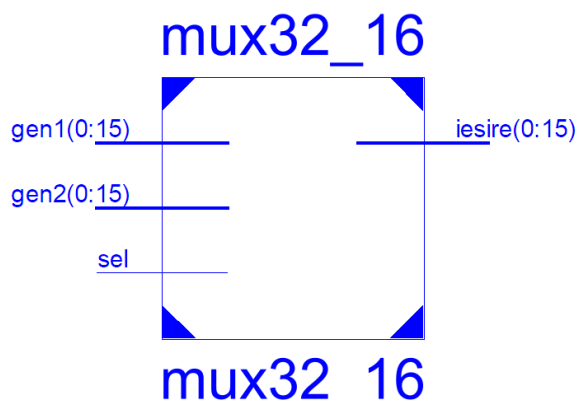
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity codstart is
    port(start:in std_logic;
          intrare:in std_logic_vector(0 to 5);
          iesire:out std_logic_vector(0 to 6));
end codstart;

architecture arh of codstart is
begin
    iesire(0)<='0';
    iesire(1 to 6)<=intrare xor start&start&start&start&start&start;
end arh;
```

### Mux 32\_16:

Aceasta componenta alege care dintre cele doua Segmente de Date sa fie transmise, in functie de al doilea bit al modului, iar iesirea este Segmentul de Date ales.



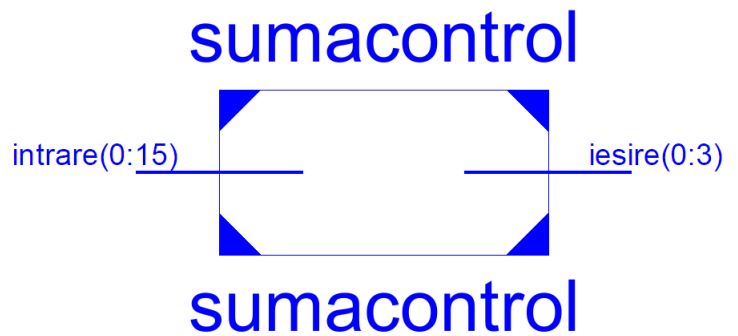
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux32_16 is
    port(gen1:in std_logic_vector(0 to 15);
          gen2:in std_logic_vector(0 to 15);
          sel:in std_logic;
          iesire:out std_logic_vector(0 to 15));
end mux32_16;

architecture mux of mux32_16 is
begin
    process(sel,gen1,gen2)
    begin
        if (sel='0') then
            iesire<=gen1;
        else iesire<=gen2;
        end if;
    end process;
end mux;
```

## Suma de Control

Aceasta componenta primește ca intrare Segmentul de Date și calculează Suma de Control, aceasta fiind rezultatul operației de SAU-EXCLUSIV dintre cele 4 cuvinte de 4 biți din Segmentul de Date.

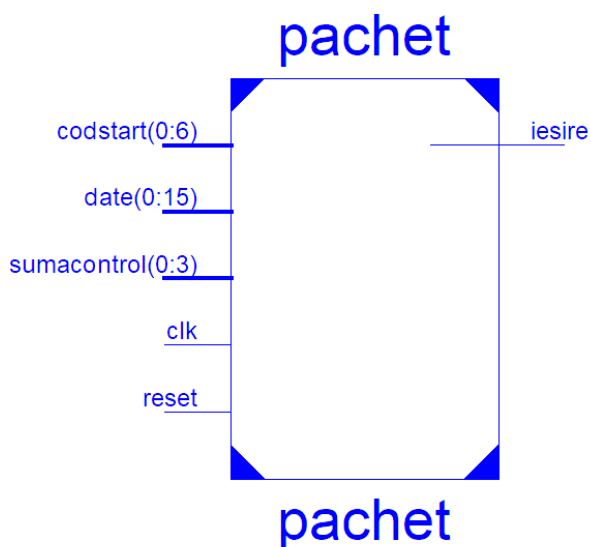


```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sumacontrol is
    port(intrare:in std_logic_vector(0 to 15);
          iesire:out std_logic_vector(0 to 3));
end sumacontrol;

architecture arh of sumacontrol is
begin
    process(intrare) is
    begin
        iesire(0)<=intrare(0) xor intrare(4) xor intrare(8) xor intrare(12);
        iesire(1)<=intrare(1) xor intrare(5) xor intrare(9) xor intrare(13);
        iesire(2)<=intrare(2) xor intrare(6) xor intrare(10) xor intrare(14);
        iesire(3)<=intrare(3) xor intrare(7) xor intrare(11) xor intrare(15);
    end process;
end arh;
```

## Pachet de date



Această componentă primește ca intrări componentele pachetului de date, respectiv Segmentul de Start, Segmentul de Date și Suma de Control. Pe lângă acestea, intrarea reset are rolul de a aduce variabila i (integer) la valoarea inițială (0). Pentru sincronizare se folosește intrarea clock. Ieșirea este de 1 bit, iar la fiecare tact ia valoarea elementului corespunzător în funcție de contor (i). Astfel, ieșirea va ajunge să primească pe rând toți biții pachetului de date.



```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity pachet is
    port(reset:in std_logic:='1';
          clk:in std_logic;
          codstart:in std_logic_vector(0 to 6);
          date:in std_logic_vector(0 to 15);
          sumacontrol:in std_logic_vector(0 to 3);
          iesire:out std_logic);
end pachet;

architecture arh of pachet is

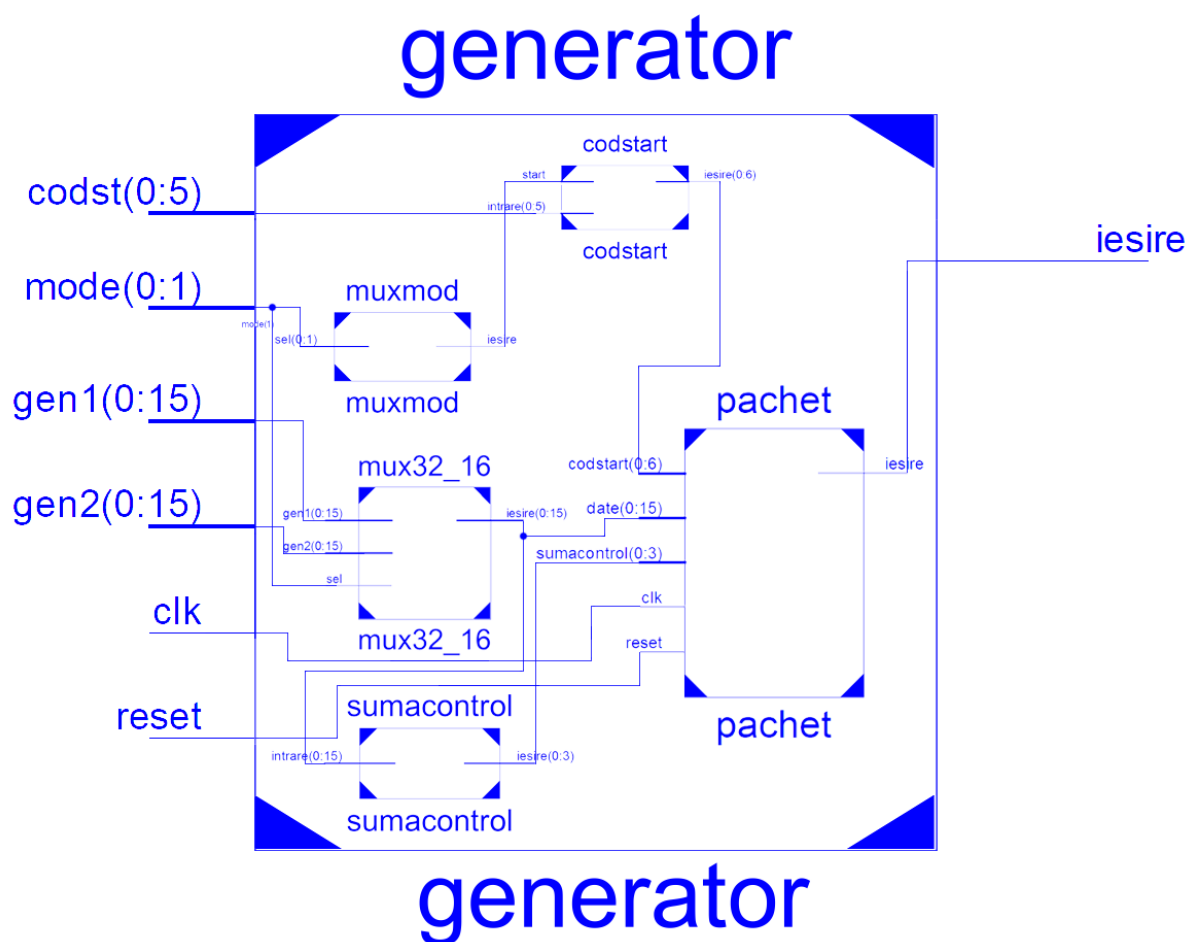
begin
    process (clk,reset)
        variable i:integer:=0;
    begin
        if (reset='1') then
            i:=0;
        elsif (clk'event and clk='1') then
            if (i=0) then
                iesire<=codstart(0);
            end if;
            if (i=1) then
                iesire<=codstart(1);
            end if;
            if (i=2) then
                iesire<=codstart(2);
            end if;
            if (i=3) then
                iesire<=codstart(3);
            end if;
            if (i=4) then
                iesire<=codstart(4);
            end if;
            if (i=5) then
                iesire<=codstart(5);
            end if;
            if (i=6) then
                iesire<=codstart(6);
            end if;
            if (i=7) then
                iesire<=date(0);
            end if;
            if (i=8) then
                iesire<=date(1);
            end if;
            if (i=9) then
                iesire<=date(2);
            end if;
            if (i=10) then
                iesire<=date(3);
            end if;
            if (i=11) then
                iesire<=date(4);
            end if;
            if (i=12) then
                iesire<=date(5);
            end if;
            if (i=13) then
                iesire<=date(6);
            end if;
            if (i=14) then
                iesire<=date(7);
            end if;
            if (i=15) then
                iesire<=date(8);
            end if;
            if (i=16) then
                iesire<=date(9);
            end if;
            if (i=17) then
                iesire<=date(10);
            end if;
            if (i=18) then
                iesire<=date(11);
            end if;
            if (i=19) then
                iesire<=date(12);
            end if;
            if (i=20) then
                iesire<=date(13);
            end if;
            if (i=21) then
                iesire<=date(14);
            end if;
            if (i=22) then
                iesire<=date(15);
            end if;
            if (i=23) then
                iesire<=sumacontrol(0);
            end if;
            if (i=24) then
                iesire<=sumacontrol(1);
            end if;
            if (i=25) then
                iesire<=sumacontrol(2);
            end if;
            if (i=26) then
                iesire<=sumacontrol(3);
            end if;
            i:=i+1;
        end if;
    end process;
end arh;

```

## Generator

Această componentă înglobează componentele prezentate anterior, și anume Mux mod, Cod Start, Mux 32\_16, Suma de Control si Pachet de date.

Componenta generează Segmentul de Start si Segmentul de Date în funcție de modul trimis, după care se calculează Suma de Control. Pachetul de date primește Segmentul de Start, Segmentul de Date si Suma de Control, pe care le stochează într-un vector care e trimis bit cu bit prin ieșire, la fiecare tact.



```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity generator is
    port(codst:in std_logic_vector(0 to 5):="010101";
          gen1:in std_logic_vector(0 to 15):="1011100011010101";
          gen2:in std_logic_vector(0 to 15):="1100101011000101";
          reset:in std_logic:='1';
          mode:in std_logic_vector(0 to 1);
          clk:in std_logic;
          iesire:out std_logic);
end generator;

architecture a of generator is
    component mux32_16 is
        port(gen1:in std_logic_vector(0 to 15);
              gen2:in std_logic_vector(0 to 15);
              sel:in std_logic;
              iesire:out std_logic_vector(0 to 15));
    end component mux32_16;

    component muxmod is
        port(sel:in std_logic_vector(0 to 1);
              iesire:out std_logic);
    end component muxmod;

    component sumacontrol is
        port(intrare:in std_logic_vector(0 to 15); iesire:out std_logic_vector(0 to 3));
    end component sumacontrol;

    component pachet is
        port(reset:in std_logic:='1';
              clk:in std_logic;
              codstart:in std_logic_vector(0 to 6);
              date:in std_logic_vector(0 to 15);
              sumacontrol:in std_logic_vector(0 to 3);
              iesire:out std_logic);
    end component pachet;

    component codstart is
        port (start:in std_logic;
              intrare:in std_logic_vector(0 to 5);
              iesire:out std_logic_vector(0 to 6));
    end component codstart;

    signal s1:std_logic;
    signal s2:std_logic_vector(0 to 6);
    signal s3:std_logic_vector(0 to 15);
    signal s4:std_logic_vector(0 to 3);
    signal s5:std_logic;
    signal s6:std_logic_vector(0 to 3);

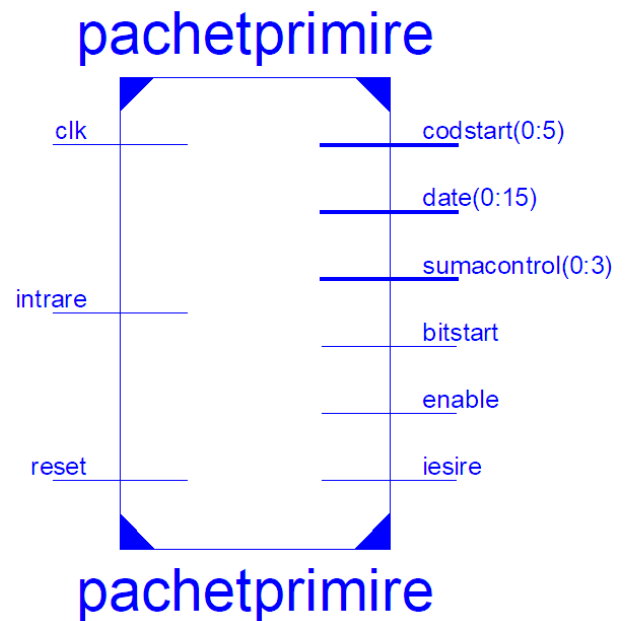
begin
    c1:muxmod port map(sel=>mode,iesire=>s1);
    c2:codstart port map(start=>s1,intrare=>codst,iesire=>s2);
    c3:mux32_16 port map(gen1=>gen1,gen2=>gen2,sel=>mode(1),iesire=>s3);
    c4:sumacontrol port map(intrare=>s3,iesire=>s4);
    s5<=mode(0) and mode(1);
    s6<=s4 xor s5&s5&s5&s5;

    c5:pachet port map(reset=>reset,clk=>clk,codstart=>s2,date=>s3,sumacontrol=>s6,iesire=>iesire);
end a;

```

## Pachet primire

Această componentă primește ca intrare un bit la fiecare tact și în funcție de un contor îl memorează în bitstart sau pe una din pozițiile vectorilor codstart, date, sumacontrol. Pentru a marca începerea primirii Segmentului de Date se folosește ieșirea "iesire", care ia valoarea 1 în momentul încărcării primului bit din vectorul date. Ieșirea "enable" ia valoarea 1 logic la finalul primirii întregului pachet de date. Intrarea Reset aduce contorul, ieșirile și Segmentul de Start la o stare inițială.



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

entity pachetprimire is
    port(reset:in std_logic:= '1';
          clk:in std_logic;
          bitstart:out std_logic:= '0';
          codstart:out std_logic_vector(0 to 5):="010101";
          date:out std_logic_vector(0 to 15);
          sumacontrol:out std_logic_vector(0 to 3);
          enable:out std_logic:= '0';
          intrare:in std_logic;
          iesire:out std_logic:= '0');
end pachetprimire;

architecture arh of pachetprimire is
begin
    process (clk,reset)
        variable i:integer:=0;
    begin
        if (reset='1') then
            i:=0;
            iesire<='0';
            enable<='0';
            bitstart<='0';
            codstart<="010101";
```

```

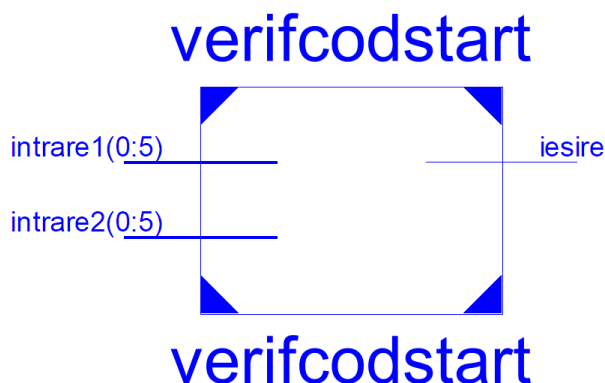
elsif (clk'event and clk='1') then
    if (i=1) then
        bitstart<=intrare;
    end if;
    if (i=2) then
        codstart(0)<=intrare;
    end if;
    if (i=3) then
        codstart(1)<=intrare;
    end if;
    if (i=4) then
        codstart(2)<=intrare;
    end if;
    if (i=5) then
        codstart(3)<=intrare;
    end if;
    if (i=6) then
        codstart(4)<=intrare;
    end if;
    if (i=7) then
        codstart(5)<=intrare;
    end if;
    if(i=8) then
        date(0)<=intrare;
        iesire<='1';
    end if;
    if(i=9) then
        date(1)<=intrare;
    end if;
    if(i=10) then
        date(2)<=intrare;
    end if;
    if(i=11) then
        date(3)<=intrare;
    end if;
    if(i=12) then
        date(4)<=intrare;
    end if;
    if(i=13) then
        date(5)<=intrare;
    end if;
    if(i=14) then
        date(6)<=intrare;
    end if;
    if(i=15) then
        date(7)<=intrare;
    end if;

    if(i=16) then
        date(8)<=intrare;
    end if;
    if(i=17) then
        date(9)<=intrare;
    end if;
    if(i=18) then
        date(10)<=intrare;
    end if;
    if(i=19) then
        date(11)<=intrare;
    end if;
    if(i=20) then
        date(12)<=intrare;
    end if;
    if(i=21) then
        date(13)<=intrare;
    end if;
    if(i=22) then
        date(14)<=intrare;
    end if;
    if(i=23) then
        date(15)<=intrare;
    end if;
    if(i=24) then
        sumacontrol(0)<=intrare;
    end if;
    if(i=25) then
        sumacontrol(1)<=intrare;
    end if;
    if(i=26) then
        sumacontrol(2)<=intrare;
    end if;
    if(i=27) then
        sumacontrol(3)<=intrare;
    end if;
    if(i=28) then
        enable<='1';
    end if;
    i:=i+1;
end if;
end process;
end arh;

```

## Verificare Cod Start

Această componentă primește ca intrări 2 vectori de câte 6 biți și are o ieșire, setată la 1 logic dacă cei 2 vectori primiți sunt egali sau la 0 logic în caz contrar.



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

entity verifcodstart is
    port(intrare1:in std_logic_vector(0 to 5);
          intrare2:in std_logic_vector(0 to 5);
          iesire:out std_logic:='1');
end verifcodstart;

architecture arh of verifcodstart is
begin
    process (intrare1,intrare2)
    begin
        if (intrare1=intrare2) then
            iesire<='1';
        else
            iesire<='0';
        end if;
    end process;
end arh;
```

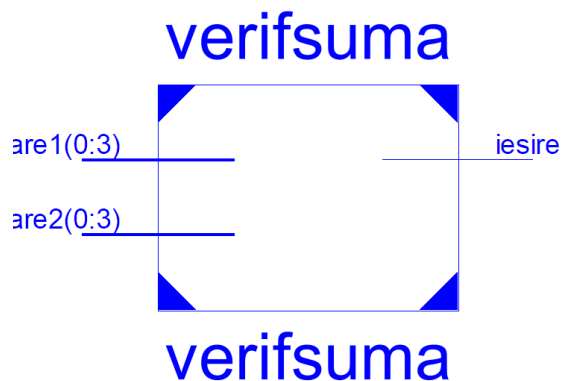
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity verifsuma is
    port(intrare1:in std_logic_vector(0 to 3);
          intrare2:in std_logic_vector(0 to 3);
          iesire:out std_logic:='1');
end verifsuma;

architecture arh of verifsuma is
begin
    process (intrare1,intrare2)
    begin
        if (intrare1=intrare2) then
            iesire<='1';
        else
            iesire<='0';
        end if;
    end process;
end arh;
```

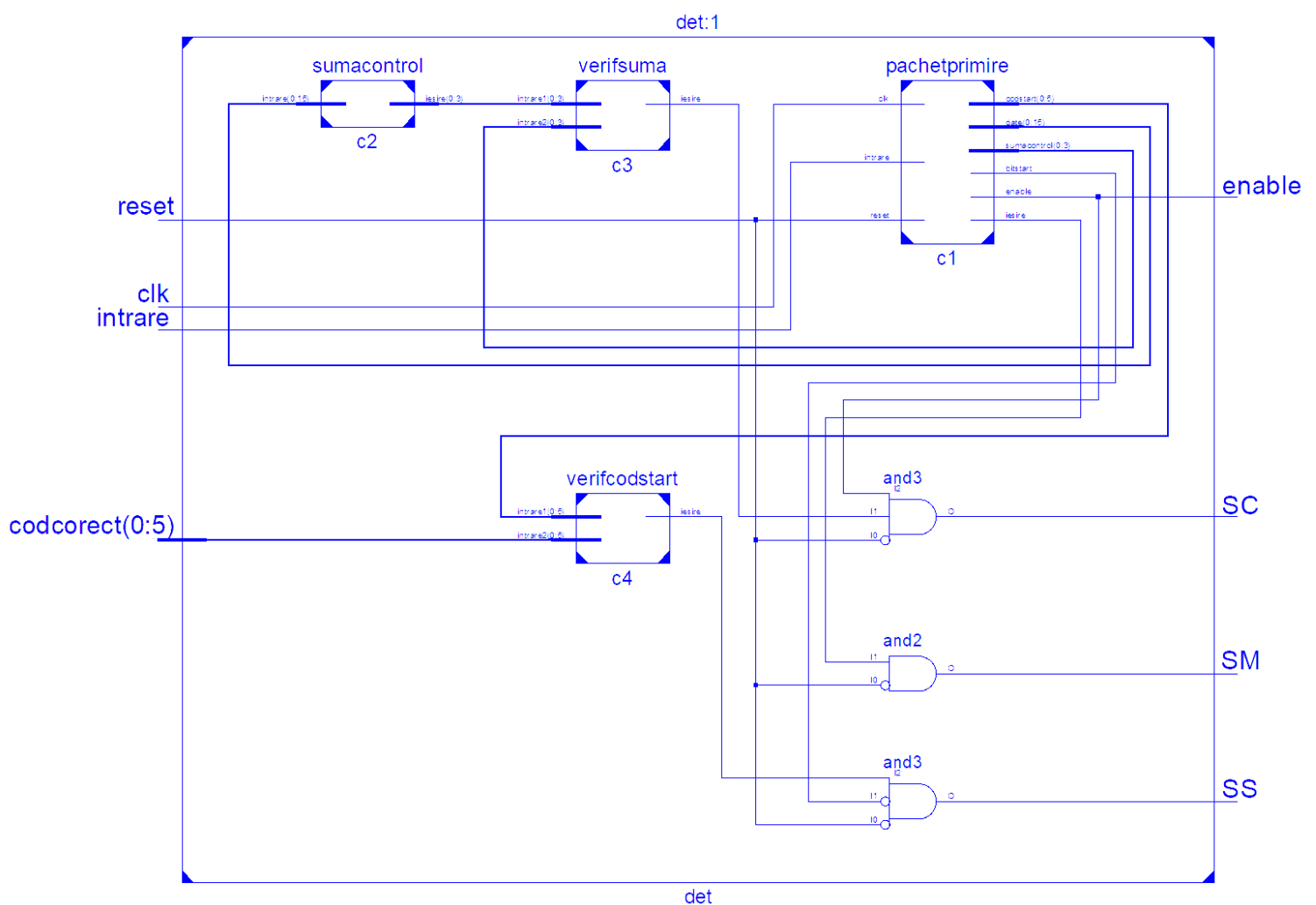
## Verificare Suma de Control

Această componentă primește ca intrări 2 vectori de câte 4 biți și are o ieșire, setată la 1 logic dacă cei 2 vectori primiți sunt egali sau la 0 logic în caz contrar.



## Detector

Această componentă înglobează componentele prezentate anterior, și anume Pachet de primire, Verificare Cod Start, Suma de Control si Verificare Suma de Control. Ieșirile SS, SM si SC sunt setate inițial la 0 logic. Componenta primește bit cu bit pachetul de date, verificând dacă Codul de Start trimis este corect. La începutul primirii pachetului de date, ieșirea SS ia valoarea 1 logic. În momentul în care s-a detectat un Cod de Start greșit, ieșirea SS ia valoarea 0 logic, altfel rămâne 1. La începerea primirii Segmentului de Date, ieșirea SM ia valoarea 1 logic. După primirea pachetului, ieșirea “final” ia valoarea 1 logic și se calculează Suma de Control. Această sumă calculată, împreună cu Suma de Control primită sunt trimise componentei Verificare Suma de Control. În cazul în care cele două sume sunt egale, ieșirea SC ia valoarea 1 logic, în caz contrar SC rămâne 0.



```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity det is
    port (intrare, clk: in std_logic;
          reset: in std_logic := '1';
          codcorect: in std_logic_vector(0 to 5) := "010101";
          SS, SM, SC: out std_logic := '0';
          enable: out std_logic);
end det;

architecture a of det is
    component pachetprimire is
        port (reset: in std_logic := '1';
              clk: in std_logic;
              bitstart: out std_logic := '0';
              codstart: out std_logic_vector(0 to 5) := "010101";
              date: out std_logic_vector(0 to 15);
              sumacontrol: out std_logic_vector(0 to 3);
              enable: out std_logic := '0';
              intrare: in std_logic;
              iesire: out std_logic := '0');
    end component pachetprimire;

    component verifsuma is
        port (intrarel: in std_logic_vector(0 to 3);
              intrare2: in std_logic_vector(0 to 3);
              iesire: out std_logic := '1');
    end component verifsuma;

    component verifcodstart is
        port (intrarel: in std_logic_vector(0 to 5);
              intrare2: in std_logic_vector(0 to 5);
              iesire: out std_logic := '1');
    end component verifcodstart;

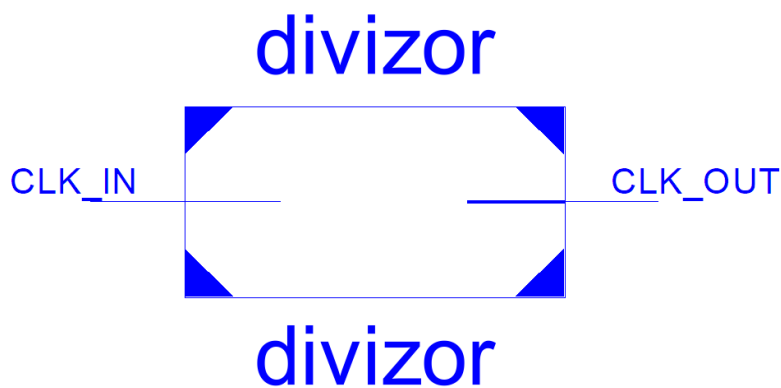
    component sumacontrol is
        port (intrare: in std_logic_vector(0 to 15);
              iesire: out std_logic_vector(0 to 3));
    end component sumacontrol;

    signal s1: std_logic := '0';
    signal s4, s6, en: std_logic;
    signal s2: std_logic_vector(0 to 15);
    signal s3: std_logic_vector(0 to 3);
    signal s5: std_logic_vector(0 to 3);
    signal scod: std_logic_vector(0 to 5);
    signal s7: std_logic := '1';
begin
    c1: pachetprimire port map (reset => reset, clk => clk, bitstart => s1,
                                codstart => scod, date => s2, sumacontrol => s3, enable => en, intrare => intrare, iesire => s4);
    c2: sumacontrol port map (intrare => s2, iesire => s5);
    c3: verifsuma port map (intrarel => s5, intrare2 => s3, iesire => s6);
    c4: verifcodstart port map (intrarel => scod, intrare2 => codcorect, iesire => s7);
    SS <= not(reset) and not(s1) and s7;
    SM <= not(reset) and s4;
    SC <= not(reset) and s6 and en;
    enable <= en;
end a;

```



## Divizor de frecvență



Această componentă primește ca intrare un semnal de tact de o anumită frecvență și printr-un contor, componenta va trimite pe ieșire un clock de o frecvență mai mică. De exemplu, un semnal de tact cu frecvența 100 MHz va scoate pe ieșire un semnal de tact cu frecvența de 1.5Hz (1 tact la 0.335 secunde).

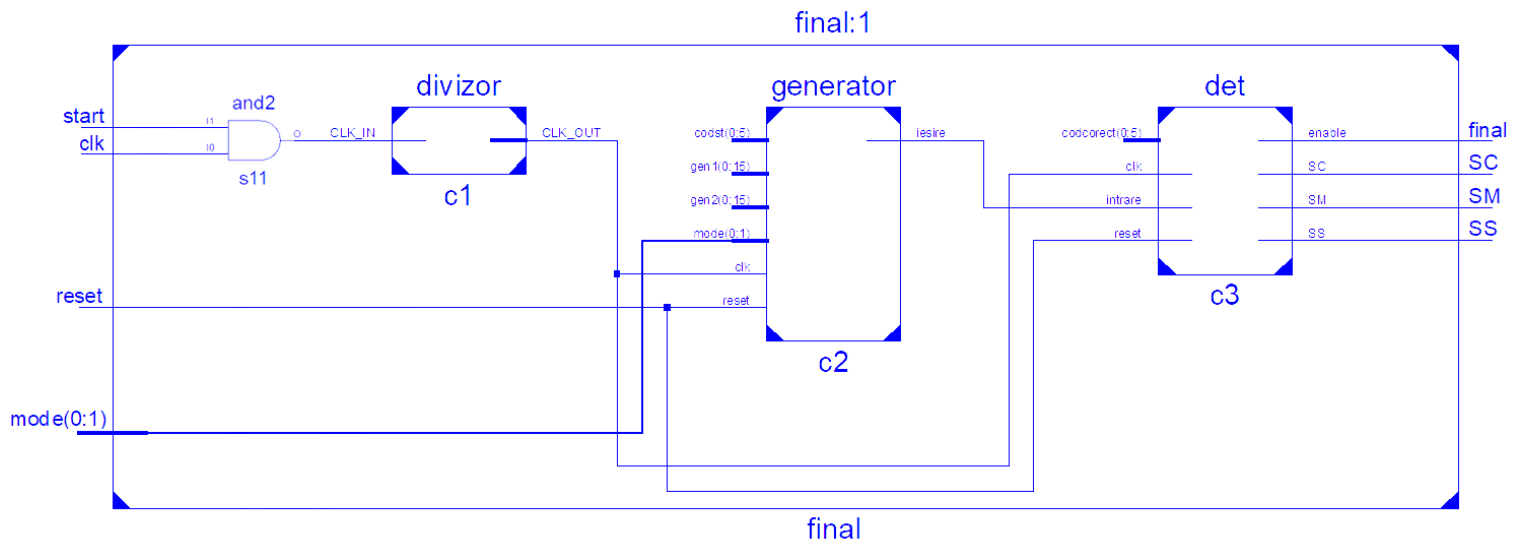
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity divizor is
    port(CLK_IN:in std_logic;
          CLK_OUT:out std_logic);
end divizor;

architecture arh of divizor is
begin
    process (CLK_IN)
        variable var_CLK:std_logic_vector(0 to 24):=(others=>'0');
    begin
        if (CLK_IN'event and CLK_IN='1') then
            var_CLK:=var_CLK+1;
        end if;
        CLK_OUT<=var_CLK(0);
    end process;
end arh;
```

## Final

Această componentă înglobează componentele Generator, Detector și Divizor de frecvență. Intrările sunt start(1 bit), mod(2 biți), reset(1 bit) și clock(1 bit), iar ieșirile sunt SS, SM, SC și "final". Dacă intrarea start e 1 logic, atunci fiecare semnal de tact intră în Divizorul de frecvență și va rezulta un semnal de tact care va intra în Generator și în Detector. Resetul este intrare atât pentru Generator, cât și pentru Detector. Modul intră în Generator, iar acesta va determina modul de funcționare al Generatorului. Ieșirea dată de Generator va intra în Detector, iar după primirea datelor, ieșirile vor lua valorile corespunzătoare, în funcție de modul ales.



```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity final is
    port(start,clk:in std_logic;
          reset:in std_logic:='1';
          mode:in std_logic_vector(0 to 1);
          SS,SM,SC,final:out std_logic);
end final;

architecture a of final is

    component det is
        port(intrare,clk:in std_logic;
              reset:in std_logic:='1';
              codcorect:in std_logic_vector(0 to 5):="010101";
              SS,SM,SC:out std_logic:='0';
              enable:out std_logic);
    end component det;

    component generator is
        port(codst:in std_logic_vector(0 to 5):="010101";
              gen1:in std_logic_vector(0 to 15):="1011100011010101";
              gen2:in std_logic_vector(0 to 15):="1100101011000101";
              reset:in std_logic:='1';
              mode:in std_logic_vector(0 to 1);
              clk:in std_logic;
              iesire:out std_logic);
    end component generator;

    component divizor is
        port(CLK_IN:in std_logic;
              CLK_OUT:out std_logic);
    end component divizor;

    signal s1,s2,sclk,fin:std_logic;
    signal modul:std_logic_vector(0 to 1);
begin
    s1<=clk and start;
    modul<=mode;
    c1:divizor port map(CLK_IN=>s1,CLK_OUT=>sclk);
    c2:generator port map(reset=>reset,mode=>modul,clk=>sclk,iesire=>s2);
    c3:det port map(intrare=>s2,clk=>sclk,reset=>reset,SS=>SS,SM=>SM,SC=>SC,enable=>fin);
    final<=fin;
end a;

```

## 4. Lista de componente utilizate

Componentele care au fost utilizate sunt:

- Mux mod
- Cod Start
- Mux 32\_16
- Suma de Control
- Pachet de date
- Generator
- Pachet primire
- Verificare Cod Start
- Verificare Suma de Control
- Detector
- Divizor de frecvență
- Final

## 5. Semnificația notațiilor I/O și a semnalelor interne

### **Mux mod**

#### Intrări

- sel (vector de 2 biți) - selecție

#### Ieșiri

- iesire (1 bit) – determinată de selecție (1 când selecția este “10”)

#### Semnale interne: -

### **Cod Start**

#### Intrări

- start (1 bit) – intrare
- intrare (vector de 6 biți) – codul de start inițial

#### Ieșiri

- iesire(vector de 7 biți) – primul bit este mereu 0, iar următorii 6 sunt generați în funcție de intrarea start

#### Semnale interne: -

### **Mux 32\_16**

#### Intrări

- gen1, gen2 (vectori de 16 biți) – segmentele de date
- sel (1 bit) - selecție

#### Ieșiri

- iesire (vector de 16 biți) – ales în funcție de selecție

#### Semnale interne: -

## **Suma de Control**

### Intrări

- intrare (vector de 16 biți) – Segmentul de date (4 cuvinte de 4 biți)

### Ieșiri

- ieșire (vector de 4 biți) – rezultatul operației SAU-EXCLUSIV între cele 4 cuvinte de 4 biți

### Semnale interne: -

## **Pachet de date**

### Intrări

- reset (1 bit) – aduce vectorul vect in starea inițială (tren de 'U')
- clk (1 bit) – semnal de tact
- codstart (7 biți) – Segmentul de Start generat
- date (16 biți) – Segmentul de Date ales
- sumacontrol (4 biți) – Suma de Control calculată

### Ieșiri

- ieșire (1 bit) – câte un bit al pachetului de date

### Semnale interne

- (nu e semnal) variabila i (integer) - contor

## **Generator**

### Intrări

- codst (vector de 6 biți) – codul de start inițial – "010101"
- gen1 (vector de 16 biți) – primul segment de date - "1011100011010101"
- gen2 (vector de 16 biți) – al doilea segment de date - "1100101011000101"
- reset (1 bit) – comandă reset-ul componentei Pachet de date
- mode (vector de 2 biți) – modul ales, trimis ca intrare sel a componentei Mux mod, iar al 2-lea bit este trimis și ca intrare sel a componentei Mux 32\_1 (mode(1))
- clk (1 bit) – semnal de tact

### Ieșiri

- ieșire (1 bit) – bitul care se trimite

### Semnale interne

- s1 (1 bit) – primește ieșirea ieșire a componentei Mux mod și este trimis ca intrare în componenta Cod Start (în intrarea start)
- s2 (vector de 7 biți) – primește ieșirea ieșire a componentei Cod Start și este trimis ca intrare în componenta Pachet de date (în intrarea codstart)
- s3 (vector de 16 biți) – primește ieșirea ieșire a componente Mux 32\_16 și este trimis ca intrare în componenta Pachet de date (în intrarea date)
- s4 (vector de 4 biți) – primește ieșirea ieșire a componentei Suma de Control și este folosit pentru calculul semnalului s6
- s5 (1 bit) – primește rezultatul operației ȘI dintre cei 2 biți ai intrării mode
- s6 (vector de 4 biți) – primește rezultatul operației SAU-EXCLUSIV dintre semnalul s4 și vectorul "s5&s5&s5&s5" (dacă s5 este 0, atunci s6 va lua valoarea Sumei de Control, altfel va fi Suma de Control negată) și este transmis ca intrare în componenta Pachet de date (în intrarea sumacontrol)

### **Pachet primire**

#### Intrări

- reset (1 bit) – aduce ieșirile bitstart, ieșire, enable și contorul i la starea inițială (0) și ieșirea codstart este inițializată la "010101"
- clk (1 bit) – semnal de tact
- intrare (1 bit) – bitul folosit pentru formarea ieșirilor

#### Ieșiri

- bitstart (1 bit) – ia valoarea intrării intrare în funcție de contorul i
- codstart (vector de 6 biți) – fiecare bit al său ia valoarea intrării intrare în funcție de contorul i
- date (vector de 16 biți) - fiecare bit al său ia valoarea intrării intrare în funcție de contorul i
- sumacontrol (vector de 4 biți) - fiecare bit al său ia valoarea intrării intrare în funcție de contorul i

- enable (1 bit) – ia valoarea 1 logic în momentul citirii intrării intrare când contorul i este 28 (finalul primirii pachetului de date)
- ieșire (1 bit) - ia valoarea 1 logic in momentul citirii intrării intrare cand contorul i este 8 (începerea primirii Segmentului de Date)

#### Semnale interne

- (nu e semnal) variabila i (integer) – contor, crește cu 1 la fiecare tact și ajuta la scrierea corectă a pachetului de date

#### **Verificare Cod Start**

##### Intrări

- intrare1 (vector de 6 biți) - Cod de Start trimis
- intrare2 (vector de 6 biți) - Cod de Start corect

##### Ieșiri

- iesire (1 bit) – ia valoare 1 logic dacă cele 2 intrări sunt egale, altfel 0

#### Semnale interne: -

#### **Verificare Suma de Control**

##### Intrări

- intrare1 (vector de 4 biți) – Suma de Control trimisă
- intrare2 (vector de 4 biți) – Suma de Control calculată

##### Ieșiri

- iesire (1 bit) – ia valoarea 2 logic dacă cele 2 intrări sunt egale, altfel 0

#### Semnale interne: -

#### **Detector**

##### Intrări

- intrare (1 bit) – intrarea primită de la generator
- reset (1 bit) – comandă reset-ul componentei Pachet primire și aduce ieșirile SS, SM si SC în starea lor inițială (0 logic)
- clk (1 bit) – semnal de tact
- codcorect (vector de 6 biți) – Codul de Start corect – “010101” și este trimis ca intrare în componenta Verificare Cod de Start (în intrarea intrare2)

## Ieșiri

- SS (1 bit) – este setat inițial la 1 logic, devine 0 în momentul detectării unui Cod de Start incorect
- SM (1 bit) – este setat inițial la 0 logic, devine 1 în momentul începerii primirii Segmentului de Date
- SC (1 bit) – este setat inițial la 0 logic, devine 1 la finalul primirii pachetului de date dacă Suma de Control a fost transmisă corect
- enable (1 bit) – este setat inițial la 0 logic, devine 1 la finalul primirii pachetului de date

## Semnale interne

- s1 (1 bit) – primește ieșirea bitstart a componentei Pachet primire și este folosit pentru calculul ieșirii SS
- s2 (vector de 16 biți) – primește ieșirea date a componentei Pachet primire și este trimis ca intrare în componenta Suma de Control (în intrarea intrare)
- s3 (vector de 4 biți) – primește ieșirea sumacontrol a componentei Pachet primire și este trimis ca intrare în componenta Verificare Suma de Control (în intrarea intrare2)
- s4 (1 bit) – primește ieșirea iesire a componentei Pachet primire și este folosit pentru calculul ieșirii SM
- s5 (vector de 4 biți) – primește ieșirea iesire a componentei Suma de Control și este trimis ca intrare în componenta Verificare Suma de Control (în intrarea intrare1)
- s6 (1 bit) – primește ieșirea iesire a componentei Verificare Suma de Control și este folosit pentru calculul ieșirii SC
- s7 (1 bit) – primește ieșirea iesire a componentei Verificare Cod Start și este folosit pentru calculul ieșirii SS
- en (1 bit) – primește ieșirea enable a componentei Pachet primire și este folosit pentru calculul ieșirii SS și dă valoarea ieșirii enable
- scod (vector de 6 biți) – primește ieșirea codstart a componentei Pachet primire și este trimis ca intrare în componenta Verificare Cod Start (în intrarea intrare1)



## **Divizor de frecvență**

### Intrări

- CLK\_IN (1 bit) – semnal de tact de intrare

### Ieșiri

- CLK\_OUT (1 bit) – semnal de tact de ieșire

### Semnale interne

- (nu e semnal) variabila var\_CLK (vector de 25 de biți) – contorizează numărul de semnale de tact de intrare și valoarea primului său bit este luată de către ieșirea CLK\_OUT

## **Final**

### Intrări

- start (1 bit) – este folosit pentru a determina funcționarea semnalului de tact
- clk (1 bit) – semnal de tact
- reset (1 bit) – comandă reset-ul componentelor Generator și Detector
- mode (vector de 2 biți) – este modul folosit pentru transmisia de date

### Ieșiri

- SS (1 bit) – este setat inițial la 1 logic, devine 0 în momentul detectării unui Cod de Start incorect
- SM (1 bit) – este setat inițial la 0 logic, devine 1 în momentul începerii primirii Segmentului de Date
- SC (1 bit) – este setat inițial la 0 logic, devine 1 la finalul primirii pachetului de date dacă Suma de Control a fost transmisă corect
- final (1 bit) - este setat inițial la 0 logic, devine 1 la finalul primirii pachetului de date

### Semnale interne

- s1 (1 bit) – este rezultatul operației ȘI dintre semnalul de tact clk și intrarea start și este transmis ca intrare în componenta Divizor de frecvență (în intrarea CLK\_IN)

- sclk (1 bit) – este ieșirea CLK\_OUT a componentei Divizor de frecvență și este transmis ca intrare în componentele Generator și Detector
- s2 (1 bit) – este ieșirea ieșire a componentei Generator și este transmis ca intrare în componenta Detector (în intrarea intrare)
- fin (1 bit) – este ieșirea enable a componentei Detector și dă valoarea ieșirii final
- modul (vector de 2 biți) – ia valoarea intrării mod și este transmis ca intrare în componenta Generator (în intrarea mode)

## 6. Justificarea soluției alese

În vederea realizării proiectului am utilizat o metodă mai ușoară în opinia noastră, și anume împărțirea proiectului pe componente. Fiecare componentă utilizată are câte o funcție diferită în vederea funcționării transmisiei de date.

Componentele folosite sunt legate în modulul principal printr-o descriere structurală, această descriere fiind una foarte utilă în cazul în care mai dorim să folosim componentele utilizate.

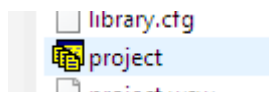
Transmisia de date serială este utilă deoarece, comparativ cu transmisia de date paralelă, pentru a transmite un pachet de date este necesară existența unei singure căi de transmisie. Cu toate că transferul paralel este mai rapid, majoritatea transmisiilor de date între calculatoarele moderne și periferice au loc în mod serial pentru a reduce costurile.



# 7. Utilizare si rezultate

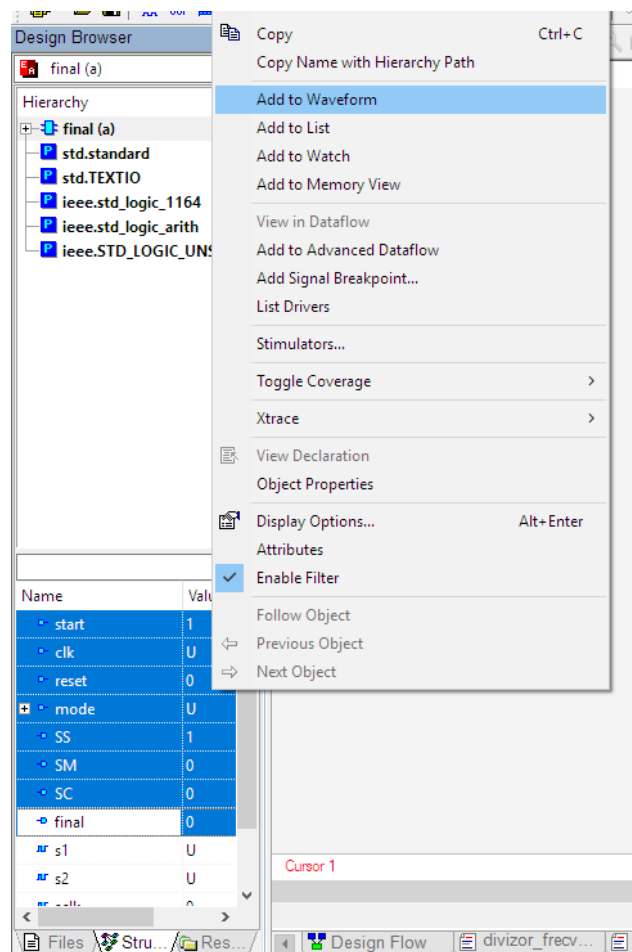
## 1. Active-HDL

În Active-HDL proiectul se folosește astfel:

1. Se deschide programul Active-HDL.
2. Din meniu se selectează File>Open și se deschide fișierul cu extensia .aws.




3. Se compilează toate fișierele folosind butonul “Compile All”. 
4. Pentru simulare se dă click pe “New Waveform”. 

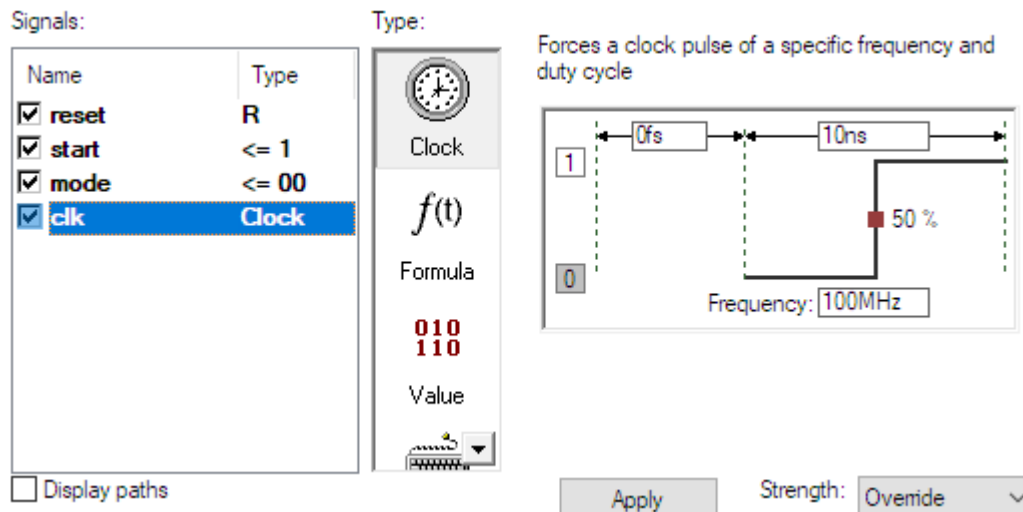


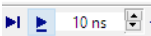
5. Din tab-ul Structure se selectează toate intrările și ieșirile pe care vrem să le urmărim. Se dă click dreapta pe ele și se selectează “Add to Waveform”.

6. Se selectează intrările și se dă click dreapta pe Stimulators.

 Stimulators...

7. Pentru fiecare semnal se selectează tipul: Clock pentru clk, Value sau Hotkey pentru reset și mode. Pentru funcționare, intrarea start trebuie să fie 1.



8. La început reset-ului trebuie să i se atribuie valoarea 1. Se alege un timp de simulare și se dă click pe “Run For”,  după care reset-ului i se atribuie valoarea 0. Se continuă simularea și se testează cele 4 moduri. La fiecare schimbare a modului trebuie să aibă loc și atribuirea valorii 1 pentru reset. În momentul în care ieșirea “final” se face 1, atunci pachetul de date a fost complet trimis. Se observă ieșirile SS, SM, SC. La finalul fiecărei simulări, acestea trebuie să fie:

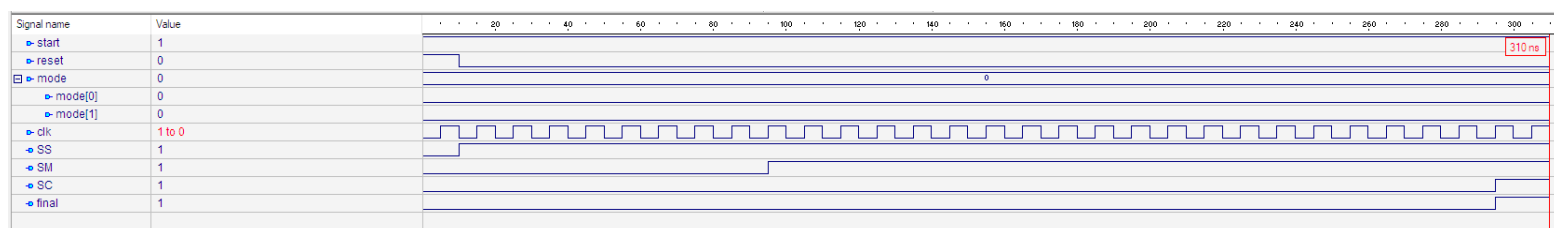
SS-1 SM-1 SC-1 (pentru mod 00)

SS-1 SM-1 SC-1 (pentru mod 01)

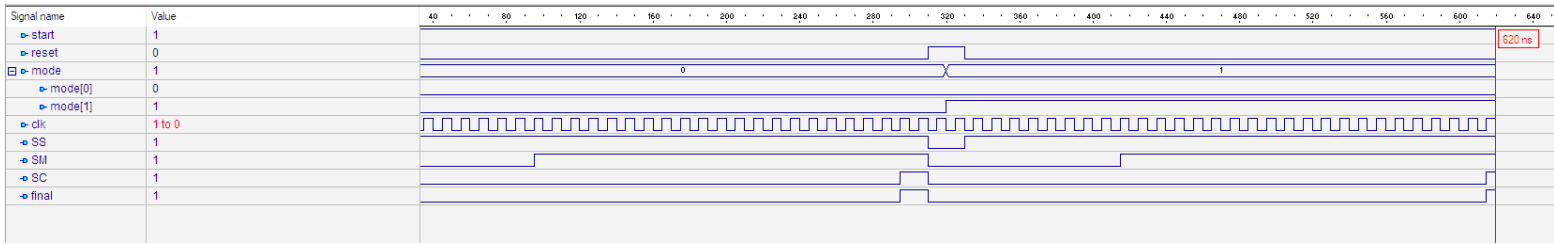
SS-0 SM-1 SC-1 (pentru mod 10)

SS-1 SM-1 SC-0 (pentru mod 11)

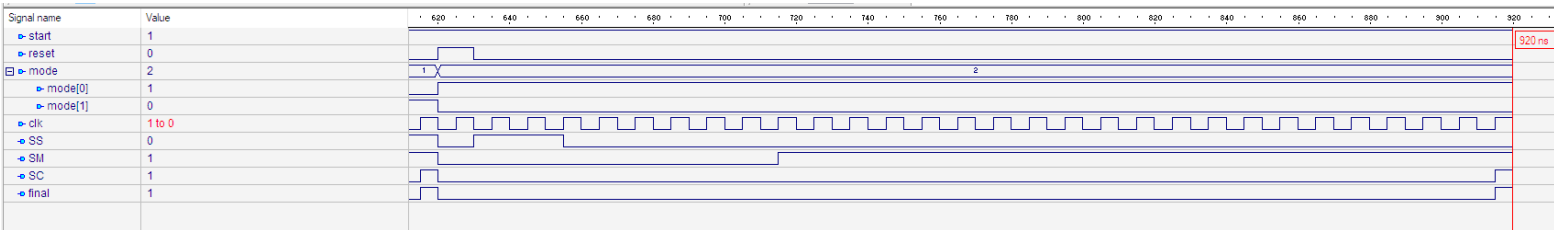
- mod 00 (echivalent cu mod 01):



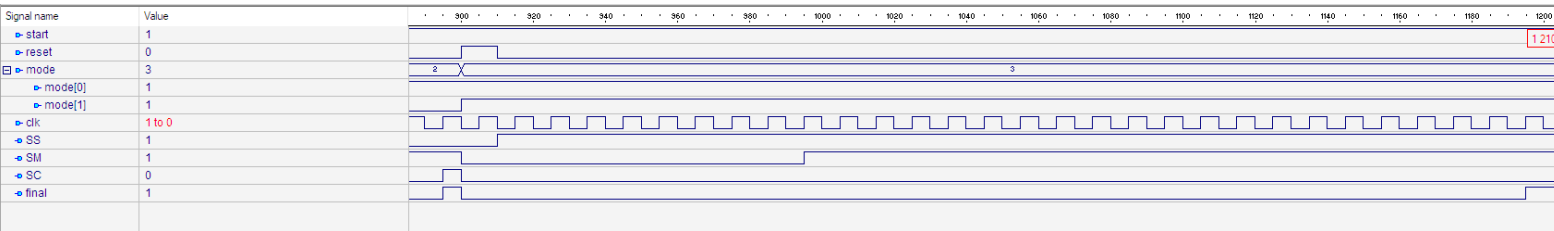
- mod 00 și mod 01:



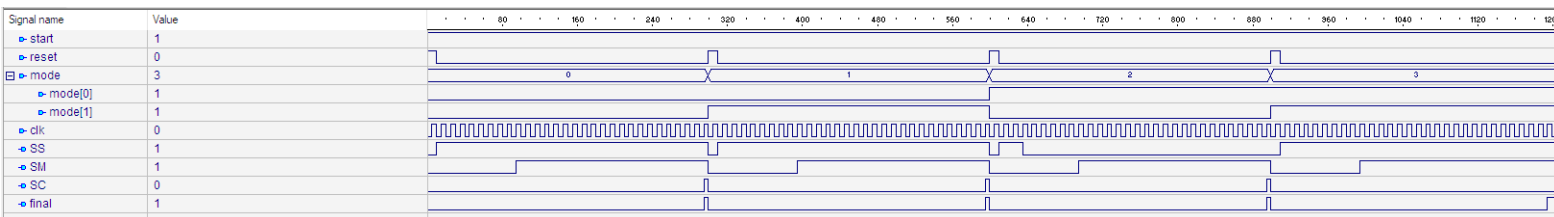
- mod 10:



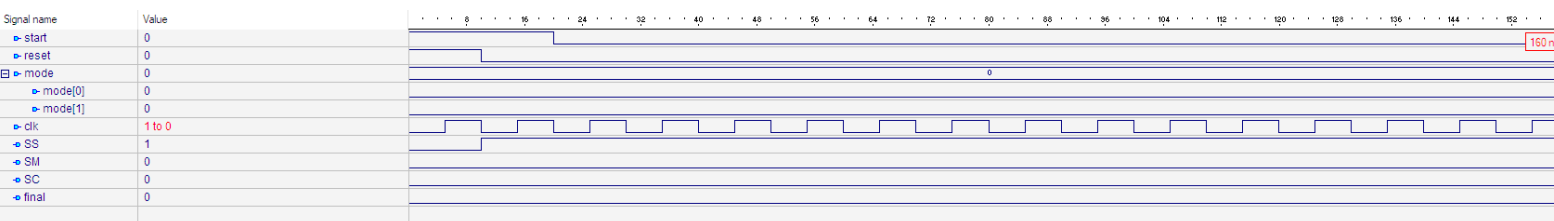
- mod 11:



Toate modurile:

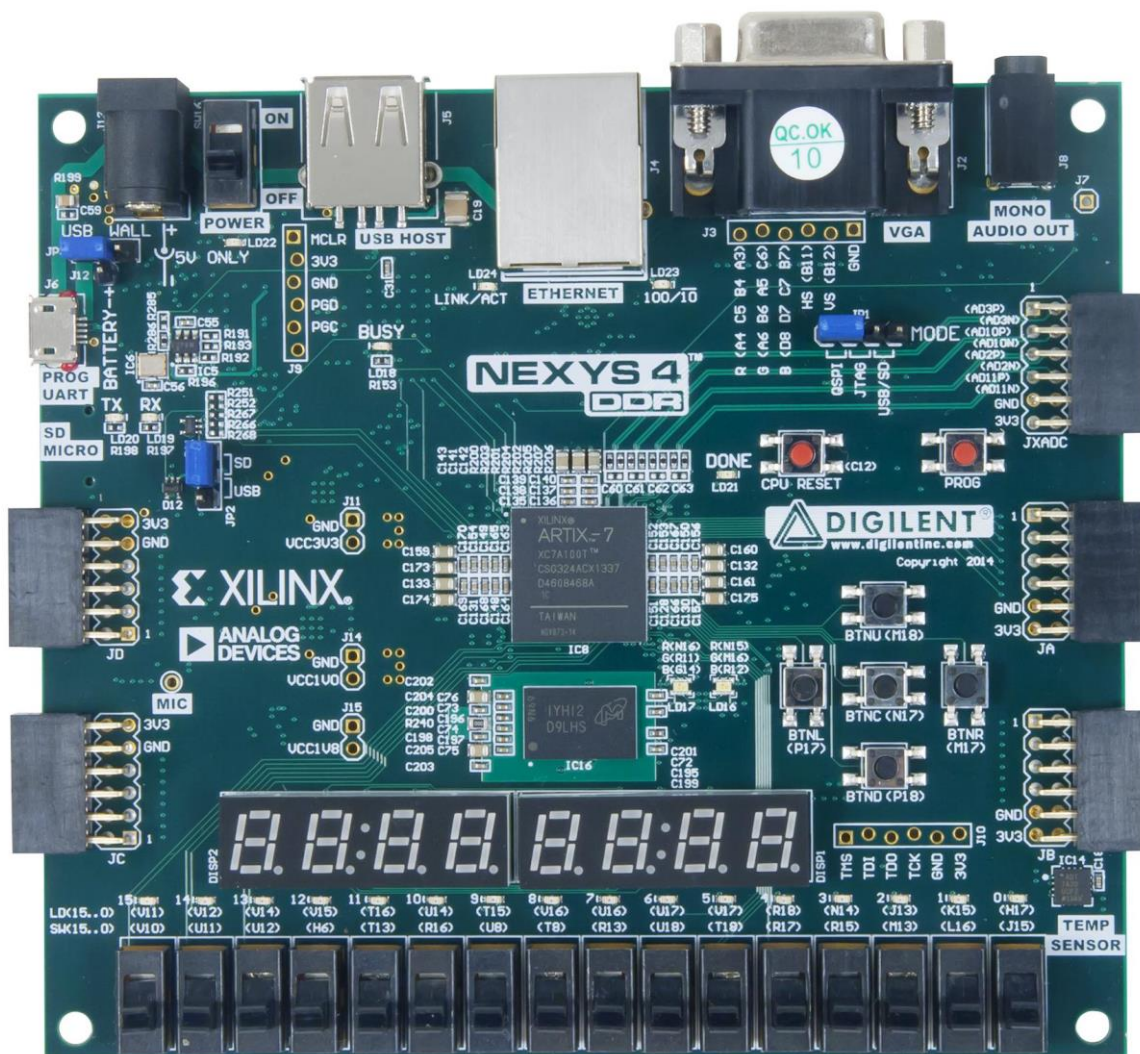


Dacă intrarea start este 0, atunci transmitia de date este in stare de “pauză” (clock-ul nu funcționează):



## 2. Xilinx ISE Design Suite si implementarea pe placa FPGA

Pentru a urmări proiectul pe o placa FPGA, vom folosi placa Nexys 4.

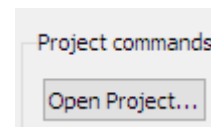


Printre componentele plăcii Nexys 4 se află 8 afișoare 7 segmente, 16 comutatoare cu 2 stări, 5 comutătoare de tip push-button și 16 leduri.

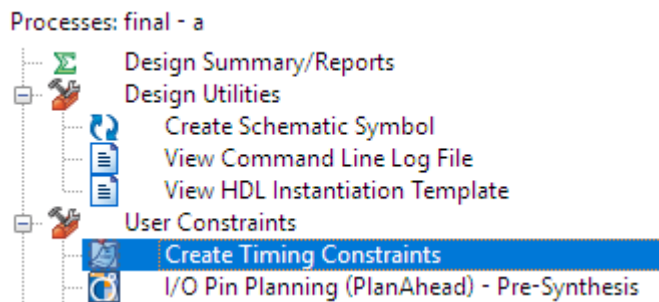
Placa are un oscilator intern cu o frecvență de 100 MHz, acesta fiind clock-ul intern al plăcii. Semnalul de tact de la oscilatori se conectează la unul dintre pinii de tact global. Astfel, utilizatorul are control asupra acestui semnal și se poate realiza divizarea de frecvență.

Pentru a testa proiectul pe placă, vom folosi programul Xilinx ISE Design Suite.  
În ISE Design Suite proiectul se folosește astfel:

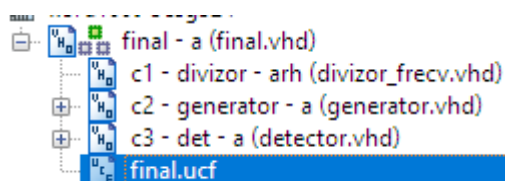
1. Se deschide programul ISE Design Suite.
2. Se dă click pe Open Project si se alege proiectul.



3. Se dă dublu click pe Create Timing Constraints.

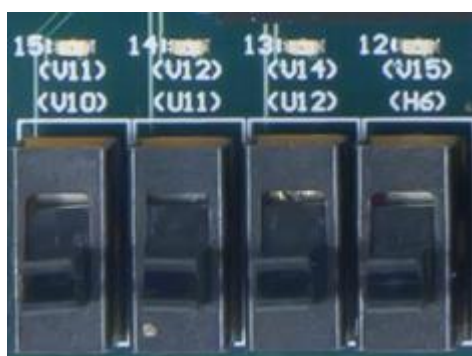


4. Un fișier .ucf va fi generat si adăugat proiectului.



5. În fișierul .ucf trebuie să se facă legăturile cu placa: intrările vor fi conectate la comutatoarele cu 2 stări sau la comutătoarele de tip push-button, iar ieșirile vor fi conectate la afișoarele 7 segmente sau la leduri.

Vom folosi următoarele componente:



Pentru intrări: comutatoarele cu 2 stări V10, U11, U12 și H6.

Pentru ieșiri: ledurile V11, V12, V14 și V15.

Clock-ul intern al plăcii are pinul E3.

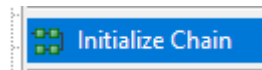
În fișierul .ucf vom scrie:

```
1 NET "start" LOC = V10 | IOSTANDARD=LVCMOS33;  
2 NET "mode(0)" LOC = U11 | IOSTANDARD=LVCMOS33;  
3 NET "mode(1)" LOC = U12 | IOSTANDARD=LVCMOS33;  
4 NET "reset" LOC = H6 | IOSTANDARD=LVCMOS33;  
5 NET "clk" LOC = E3 | IOSTANDARD=LVCMOS33;  
6 NET "SS" LOC = V11 | IOSTANDARD=LVCMOS33;  
7 NET "SM" LOC = V12 | IOSTANDARD=LVCMOS33;  
8 NET "SC" LOC = V14 | IOSTANDARD=LVCMOS33;  
9 NET "final" LOC = V15 | IOSTANDARD=LVCMOS33;
```

6. Se dă dublu click pe “Generate Programming File”.

7. Se dă dublu click pe “Configure Target Device” si se creează un proiect iMPACT.

8. Se selectează “Initialize Chain”.



9. Se dă click pe “Browse” si apoi se caută fișierul cu extensia .bit.

10. Se dă click pe “Program”, iar acum proiectul poate fi testat pe placa FPGA.



## 8. Posibilități de dezvoltare ulterioară

- Pentru o verificare mai strictă a datelor se pot realiza mai multe operații de verificare în cadrul detectorului.
- Pentru testarea funcționalității corecte a detectorului, pot fi introduse alte moduri.
- Pentru o afișare mai vizibilă a primirii de date, fiecare bit primit poate fi reprezentat pe un afișor.
- Datele ar putea fi păstrate într-o memorie și scrise într-un fișier text, împreună cu un anumit cod de eroare în cazul datelor detectate ca fiind incorecte. Datele ar putea fi accesate ulterior și decodificate.