

Tema 2 – Multiplicator cu deplasare la stanga

Circuite VLSI

**Autor:**

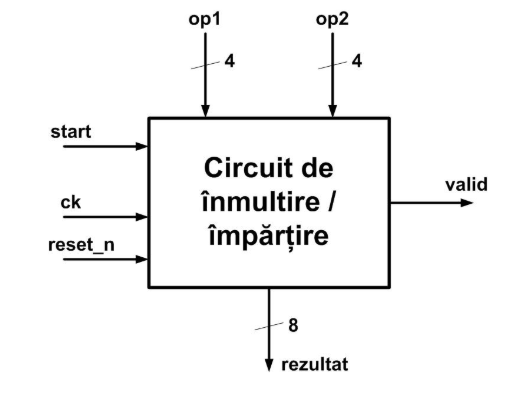
Dinu Ionuț Vlăduț

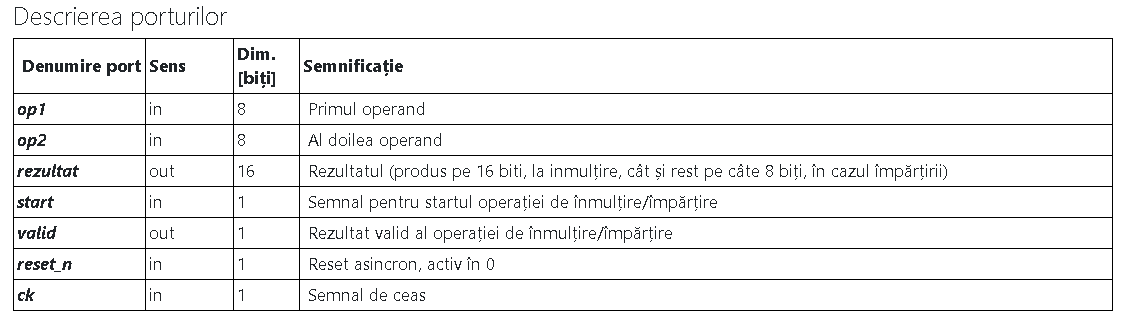
Calculatoare anul IV

Grupa 4LF781

[ionut.dinu@student.unitbv.ro](mailto:ionut.dinu@student.unitbv.ro)

**BRAȘOV, 2022**

****



### Constrangeri

Sistemul se va modela ca un sistem riguros sincron. Semnalul *ck* va fi primit de către toate registrele din sistem.

Se vor evita circuitele de divizare a frecvenței semnalului de tact.

Se vor folosi, eventual, circuite de generare a semnalelor de activare (*enable*).

Descrierea sistemului se va face în întregime în VHDL. În descrierea de nivel înalt se vor instanția două componente:

* cale de date
* cale de control

Algoritm de înmulțire cu deplasare dreapta

- resetează P (n+1 biți)

- încarcă deînmulțitul în A (n biți)

- încarcă înmulțitorul în B (n biți)

- repetă de n ori

- dacă LSB A = 1 atunci - P <= P + B

- deplasează cu o poziție dreapta A (MSB A = LSB P)

- deplasează cu o poziție stanga P (MSB P = 0)

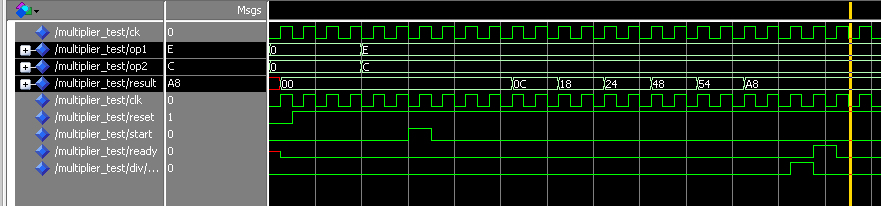
- P conține MSB produs, iar A conține LSB produs

Specificatii de proiectare

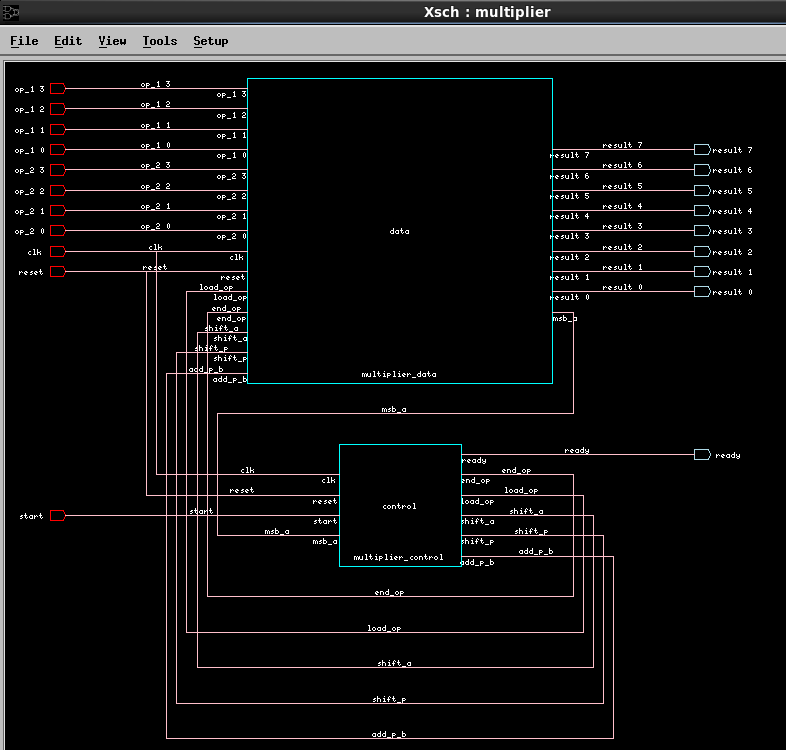
2. Algoritm de înmulțire cu deplasare stânga- resetează P (2\*n biți)  
- încarcă deînmulțitul in A (n biți)  
- încarcă înmulțitorul in B (n biți)  
- repetă de n ori  
- deplasează cu o poziție stânga P (LSB P = 0)  
- dacă MSB A = 1 atunci  
- P <= P + B  
- deplasează cu o poziție stânga A (LSB A = X)  
- P conține produs

233 x 195 = 45435  
233 = 1110 1001  
195 = 1100 0011  
Reg. P Reg. A  
----------- ---------  
0000 0000 0000 0000 1110 1001 - inițial  
0000 0000 0000 0000  
1100 0011  
-------------------  
0000 0000 1100 0011  
0000 0001 1000 0110 1101 001X - iterația 1  
0000 0001 1000 0110  
1100 0011  
-------------------  
0000 0010 0100 1001  
0000 0100 1001 0010 1010 01XX - iterația 2  
0000 0100 1001 0010  
1100 0011  
-------------------  
0000 0101 0101 0101  
0000 1010 1010 1010 0100 1XXX - iterația 3  
0001 0101 0101 0100 1001 XXXX - iterația 4  
0001 0101 0101 0100  
1100 0011  
-------------------  
0001 0110 0001 0111  
0010 1100 0010 1110 001X XXXX - iterația 5  
0101 1000 0101 1100 01XX XXXX - iterația 6  
1011 0000 1011 1000 1XXX XXXX - iterația 7  
1011 0000 1011 1000  
1100 0011  
-------------------  
1011 0001 0111 1011-------------------Produs = 11x4096 + 1x256 + 7x16 + 11 = 45435

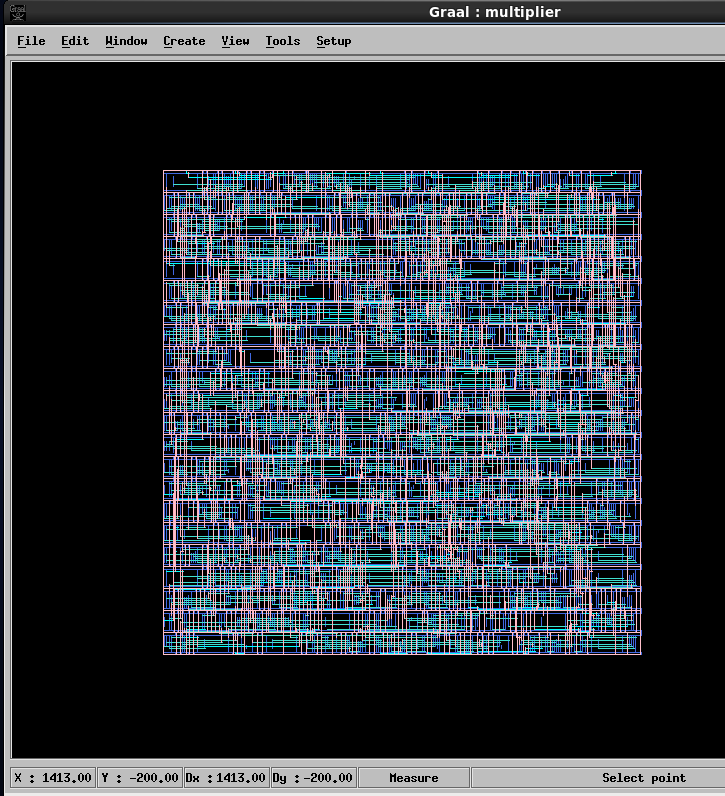
**Simularea circuitului in ModelSIM**



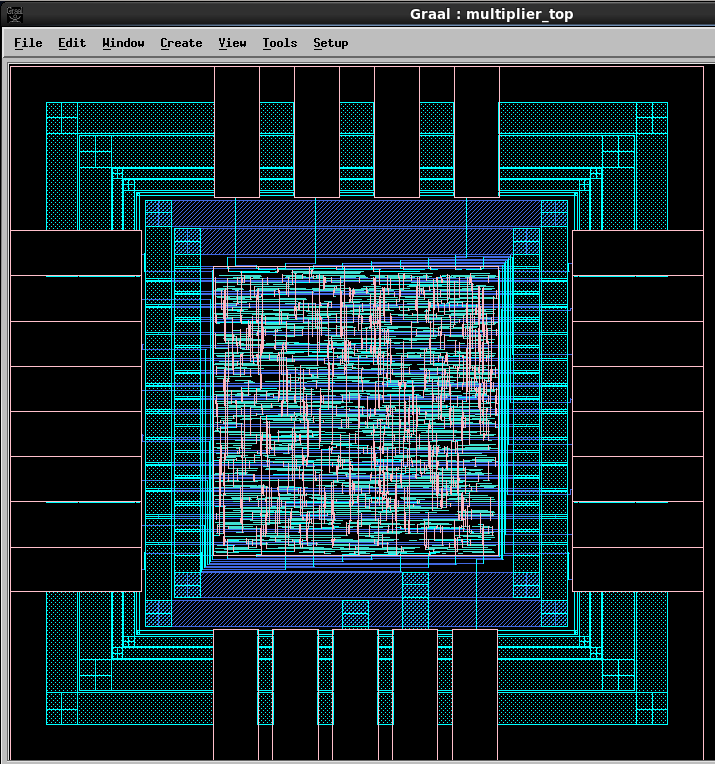
**Schema rezultata din Alliance utilizand comanda vasy**



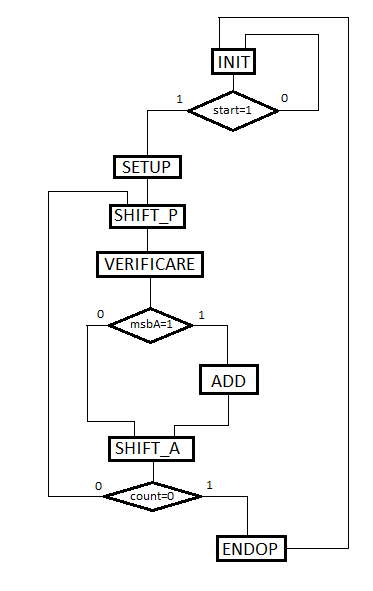
**Nucleu fara paduri**



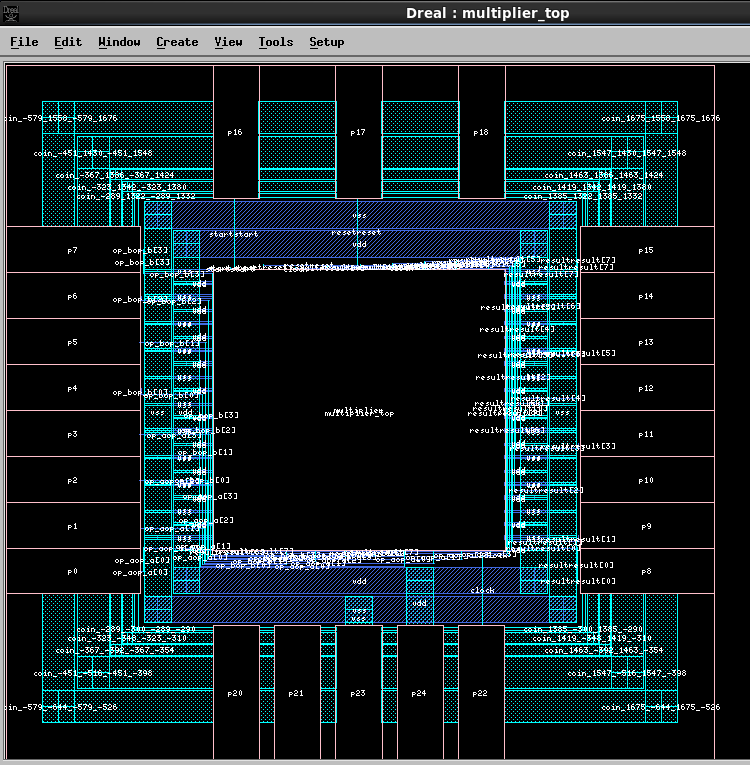
**Nucleu cu paduri**



**Schema bloc a functionarii**

****

**Schema finala a circuitului**



**Fisierele .vhd**

**Multiplier.vhd**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_arith.ALL;

USE IEEE.std\_logic\_unsigned.ALL;

entity multiplier is

    port ( op\_1 : in std\_logic\_vector (3 downto 0);

           op\_2 : in std\_logic\_vector (3 downto 0);

           result : out std\_logic\_vector (7 downto 0);

           clk : in std\_logic;

           reset : in std\_logic;

           start : in std\_logic;

           ready : out std\_logic

           );

end multiplier;

architecture multiplier\_arch of multiplier is

    signal load\_op : std\_logic;

    signal shift\_a : std\_logic;

    signal shift\_p : std\_logic;

    signal add\_p\_b : std\_logic;

    signal end\_op : std\_logic;

    signal msb\_a : std\_logic;

component multiplier\_control

    port ( clk : in std\_logic;

           reset : in std\_logic;

           start : in std\_logic;

           msb\_a : in std\_logic;

           ready : out std\_logic;

           end\_op : out std\_logic;

           load\_op : out std\_logic;

           shift\_a : out std\_logic;

           shift\_p : out std\_logic;

           add\_p\_b : out std\_logic );

end component;

component multiplier\_data

    port ( op\_1 : in std\_logic\_vector (3 downto 0);

           op\_2 : in std\_logic\_vector (3 downto 0);

           result : out std\_logic\_vector (7 downto 0);

           clk : in std\_logic;

           reset : in std\_logic;

           load\_op : in std\_logic;

           end\_op : in std\_logic;

           shift\_a : in std\_logic;

           shift\_p : in std\_logic;

           add\_p\_b : in std\_logic;

           msb\_a : out std\_logic );

end component;

begin

    DATA : multiplier\_data port map

    ( op\_1 => op\_1,

      op\_2 => op\_2,

      result => result,

      clk =>  clk,

      reset => reset,

      msb\_a => msb\_a,

      end\_op => end\_op,

      load\_op => load\_op,

      shift\_a => shift\_a,

      shift\_p => shift\_p,

      add\_p\_b => add\_p\_b );

    CONTROL : multiplier\_control port map

    ( clk =>  clk,

      reset => reset,

      start => start,

      msb\_a => msb\_a,

      ready => ready,

      end\_op => end\_op,

      load\_op => load\_op,

      shift\_a => shift\_a,

      shift\_p => shift\_p,

      add\_p\_b => add\_p\_b );

end multiplier\_arch;

**multiplier\_data.vhd**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_arith.ALL;

USE IEEE.std\_logic\_unsigned.ALL;

entity multiplier\_data is

    port ( op\_1 : in std\_logic\_vector (3 downto 0);

    op\_2 : in std\_logic\_vector (3 downto 0);

           result : out std\_logic\_vector (7 downto 0);

           clk : in std\_logic;

           reset : in std\_logic;

           load\_op : in std\_logic;

           end\_op : in std\_logic;

           shift\_a : in std\_logic;

           shift\_p : in std\_logic;

           add\_p\_b : in std\_logic;

           msb\_a : out std\_logic

           );

end multiplier\_data;

architecture multiplier\_data\_arch of multiplier\_data is

    signal a : std\_logic\_vector (3 downto 0);

    signal b : std\_logic\_vector (3 downto 0);

    signal p : std\_logic\_vector (7 downto 0);

    begin

        -- registru\_a se ocupa cu procesarea si updatarea valorii lui "a" unde este stocat primul operand

        registru\_a : process (clk) begin

           if(( not clk'stable ) and ( clk = '1' )) then

               if (reset = '0') then a <= "0000";

            end if;

                  if (load\_op = '1') then a <= op\_1;

              end if;

               msb\_a <= a(3);

                     if (shift\_a = '1') then

                        a <= a(2 downto 0) & '0';

               end if;

           end if;

        end process registru\_a;

      -- registru\_b se ocupa cu procesarea si updatarea valorii lui "b" unde este stocat al doilea operand

        registru\_b : process (clk) begin

           if(( not clk'STABLE ) and ( clk = '1' )) then

               if (reset = '0') then b <= "0000";

                  elsif (load\_op = '1') then b <= op\_2;

              end if;

           end if;

        end process registru\_b;

      -- registru\_p se ocupa cu procesarea si updatarea valorii lui "p" unde este stocat produsul celor 2 valori

        registru\_p : process (clk) begin

           if(( not clk'STABLE ) and ( clk = '1' )) then

               if (reset = '0') then p <= "00000000";

                  elsif (load\_op = '1') then p <= "00000000";

                     elsif (shift\_p = '1') then p <= p(6 downto 0) & '0';

                  elsif (add\_p\_b = '1') then p <= p + b;

                  end if;

           end if;

        end process registru\_p;

      -- adaug rezultatul din p la result

      result <= p;

    end multiplier\_data\_arch;

**multiplier\_control.vdh**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

entity multiplier\_control is

    port ( clk : in std\_logic;

           reset : in std\_logic;

           start : in std\_logic;

           ready : out std\_logic;

           msb\_a : in std\_logic;

           end\_op : out std\_logic;

           load\_op : out std\_logic;

           shift\_a : out std\_logic;

           shift\_p : out std\_logic;

           add\_p\_b : out std\_logic

           );

end multiplier\_control;

architecture multiplier\_control\_arch of multiplier\_control is

    type stare is (INIT, SETUP, VERIFICARE, SHIFTA, SHIFTP, ENDOP, ADD);

    signal curenta : stare;

    signal urmatoare: stare;

    signal valid : std\_logic;

    signal count : integer;

    begin

        CLC : process (reset, curenta, start, msb\_a, count) begin

           case curenta is

               -- starea INIT este starea initiala a FSM-ului, pentru a incepe operatiile reset si start trebuie sa fie 1, altfel se va duce in INIT

               when INIT =>

                  if (reset = '0') then urmatoare <= INIT;

                     elsif (start = '1') then urmatoare <= SETUP;

                        else urmatoare <= INIT;

                  end if;

               -- starea SETUP este starea dupa INIT in care se pregatesc operatorii de load si se incep operatiile

               when SETUP =>

                   if (reset = '0') then urmatoare <= INIT;

                      else urmatoare <= SHIFTP;

                   end if;

               -- starea SHIFTP este starea dupa SETUP in care se shifteaza la stanga operatorul p

               when SHIFTP =>

                   if (reset = '0') then urmatoare <= INIT;

                         else urmatoare <= VERIFICARE;

                   end if;

               -- starea VERIFICARE este starea dupa SHIFTP in care se verifica daca msb din a este 1

               when VERIFICARE =>

                   if (reset = '0') then urmatoare <= INIT;

                         elsif (msb\_a = '1') then urmatoare <= ADD;

                   else urmatoare <= SHIFTA;

                  end if;

               -- starea ADD este starea dupa VERIFICARE in care se aduna la p valoarea lui b

               when ADD =>

                  if (reset = '0') then urmatoare <= INIT;

                        else urmatoare <= SHIFTA;

                  end if;

               -- starea SHIFTA este starea dupa ADD in care se shifteaza valoarea lui a cu o pozitie la stanga

               -- si apoi se trece in SHIFTP daca inca au mai ramas operatii de facut, count >0 sau reset !=0

               when SHIFTA =>

                  if (reset = '0') then urmatoare <= INIT;

                     elsif (count = 0) then urmatoare <= ENDOP;

                        else urmatoare <= SHIFTP;

                 end if;

               -- starea ENDOP este starea finala din care se revine la initializarea sistemului in starea INIT

               when ENDOP =>

                   if (reset = '0') then urmatoare <= INIT;

                      else urmatoare <= INIT;

                   end if;

           end case;

           -- daca ne aflam in starea de SETUP, load\_op devine 1 si se incarca valorile operanzilor

           if (curenta = SETUP) then load\_op <= '1';

              else load\_op <= '0';

           end if;

           -- daca ne aflam in SHIFTP se activeaza shift\_p si incepe shiftarea registrului p la stanga

           if (curenta = SHIFTP)  then shift\_p <= '1';

              else shift\_p <= '0';

           end if;

          -- daca ne aflam in SHIFTA este similar cu SHIFTP doar ca e vorba de registrul a

           if (curenta = SHIFTA) then shift\_a <= '1';

              else shift\_a <= '0';

           end if;

           -- daca ne aflam in starea ADD, se activeaza semnalul care permite adaugarea valorii lui b la p

           if (curenta = ADD) then add\_p\_b <= '1';

              else add\_p\_b <= '0';

           end if;

           -- daca ne aflam in ENDOP se activeaza end\_op si se opreste sistemul

           if (curenta = ENDOP) then end\_op <= '1';

              else end\_op <= '0';

           end if;

       end process CLC;

       -- REG are grija sa mute sistemul dintr-o stare in alta

       REG : process (clk) begin

          if ((not clk'stable) and (clk = '1')) then

              curenta <= urmatoare;

          end if;

       end process REG;

       -- CONTOR are grija ca operatiile sa se execute de 4 ori (marima in biti a unui operand) pentru a nu aparea erori la rezultat

       CONTOR : process (clk) begin

           if(( not clk'STABLE ) and ( clk = '1' )) then

               if (reset = '0') then count <= 0;

                  elsif (curenta = INIT) then count <= 4;

                     elsif (curenta = SHIFTP) then count <= count - 1;

               end if;

           end if;

       end process CONTOR;

       VALIDARE : process (clk) begin

         if ((not clk'STABLE ) and (clk = '1')) then

             if (reset = '0') then valid <= '0';

                elsif (curenta = ENDOP) then valid <= '1';

                   elsif (curenta = INIT) then valid <='0';

             end if;

         end if;

       end process VALIDARE;

     ready <= valid;

end multiplier\_control\_arch;

**multiplier\_test.vhd**

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

entity multiplier\_test is

end multiplier\_test;

architecture multiplier\_test\_arch of multiplier\_test is

    component multiplier is

        port (   op\_1 : in std\_logic\_vector (3 downto 0);

                 op\_2 : in std\_logic\_vector (3 downto 0);

                 result : out std\_logic\_vector (7 downto 0);

                 clk : in std\_logic;

                 reset : in std\_logic;

                 start : in std\_logic;

                 ready : out std\_logic );

    end component;

    signal ck : std\_logic := '0';

    signal op\_1 : std\_logic\_vector (3 downto 0);

    signal op\_2 : std\_logic\_vector (3 downto 0);

    signal result : std\_logic\_vector (7 downto 0);

    signal clk : std\_logic;

    signal reset : std\_logic;

    signal start : std\_logic;

    signal ready : std\_logic;

    begin

        DIV : multiplier port map

         ( op\_1 => op\_1,

           op\_2 => op\_2,

           result => result,

           start => start,

           ready => ready,

           reset => reset,

           clk => clk );

        ck <= not ck after 5 ns;

        clk <= ck;

        op\_1  <= "0000", "1110" after 40 ns, "1100" after 220 ns;

        op\_2  <= "0000", "1100" after 40 ns, "1100" after 220 ns;

        start <= '0', '1' after 60 ns, '0' after 70 ns, '1' after 270 ns, '0' after 280 ns;

        reset <= '0', '1' after 10 ns, '0' after 215 ns, '1' after 225 ns;

end multiplier\_test\_arch;

**Fisier .pat**

**Multiplier.pat**

**in op\_1(3 DOWNTO 0);;;;;**

**in op\_2(3 DOWNTO 0);;;**

**in clk;;;**

**in reset;;;**

**in start;;;**

**in vdd;;**

**in vss;;;**

**out ready;;;**

**out result(7 DOWNTO 0);;**

**BEGIN**

**-- op\_1 op\_2 c r s vv r rezultat**

**-- l e t ds e**

**-- k s a ds a**

**-- e r d**

**-- t t y**

**pat\_0 : 1110 1100 1 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_1 : 1110 1100 0 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_2 : 1110 1100 1 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_3 : 1110 1100 0 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_4 : 1110 1100 1 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_5 : 1110 1100 0 0 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_6 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_7 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_8 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_9 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_10 : 1110 1100 1 1 1 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_11 : 1110 1100 0 1 1 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_13 : 1110 1100 1 1 1 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_14 : 1110 1100 0 1 1 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_15 : 1110 1100 1 1 1 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_16 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_17 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_18 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_19 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_20 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_21 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_22 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_23 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_24 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_25 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_26 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_27 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_29 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_30 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_31 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_32 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_33 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_34 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_35 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_36 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_37 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_38 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_39 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_40 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_41 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_42 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_43 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_44 : 1110 1100 1 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**pat\_45 : 1110 1100 0 1 0 10 ?\* ?\*\*\*\*\*\*\*\*;**

**end;**

**Fisier .c**

#include <genlib.h>

main()

{

int i;

   GENLIB\_DEF\_LOFIG("multiplier\_top");

   GENLIB\_LOCON("vdd",           IN,        "vdd"    );

   GENLIB\_LOCON("vss",           IN,        "vss"    );

   GENLIB\_LOCON("vdde",          IN,       "vdde"    );

   GENLIB\_LOCON("vsse",          IN,       "vsse"    );

   GENLIB\_LOCON("op\_1[3:0]",     IN,     "op\_1[3:0]" );

   GENLIB\_LOCON("op\_2[3:0]",     IN,     "op\_2[3:0]" );

   GENLIB\_LOCON("reset",         IN,      "reset"    );

   GENLIB\_LOCON("start",         IN,      "start"    );

   GENLIB\_LOCON("clk",           IN,          "clk"  );

   GENLIB\_LOCON("ready",         OUT,     "ready"    );

   GENLIB\_LOCON("result[7:0]",   OUT,   "result[7:0]");

   GENLIB\_LOINS ("pvsse\_sp", "p20", "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS ("pvdde\_sp", "p21", "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS ("pvddeck\_sp", "p22", "clock", "cki", "vdde", "vdd", "vsse", "vss",0);

   GENLIB\_LOINS ("pvssi\_sp", "p23", "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS ("pvddi\_sp", "p24", "cki", "vdde", "vdd", "vsse", "vss", 0);

   for (i = 0; i < 4; i++)

    GENLIB\_LOINS("pi\_sp",

    GENLIB\_NAME("p%d", i),

    GENLIB\_NAME("op\_1[%d]", i),

    GENLIB\_NAME("op\_1op\_1[%d]", i),

             "cki", "vdde", "vdd", "vsse", "vss", 0);

   for (i = 0; i < 4; i++)

    GENLIB\_LOINS("pi\_sp",

    GENLIB\_NAME("p%d", i + 4),

        GENLIB\_NAME("op\_2[%d]", i),

    GENLIB\_NAME("op\_2op\_2[%d]", i),

              "cki", "vdde", "vdd", "vsse", "vss", 0);

   for (i = 0; i < 8; i++)

    GENLIB\_LOINS("po\_sp",

      GENLIB\_NAME("p%d", i + 8),

          GENLIB\_NAME("resultresult[%d]", i),

      GENLIB\_NAME("result[%d]", i),

          "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS("pi\_sp", "p16",  "start", "startstart",  "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS("pi\_sp", "p17",  "reset", "resetreset",  "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS("pck\_sp", "p18",  "clk",  "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS("po\_sp", "p19",  "readyready", "ready",  "cki", "vdde", "vdd", "vsse", "vss", 0);

   GENLIB\_LOINS("multiplier", "multiplier",

        "op\_1op\_1[3:0]", "op\_2op\_2[3:0]",

        "resultresult[7:0]",

            "clock", "resetreset",

            "startstart",

            "readyready",

            "vdd", "vss", 0);

   GENLIB\_SAVE\_LOFIG();

   exit(0);

}

**Fisier .rin**

**#file: divider\_top.rin**

**width (vdd 100 vss 100)**

**west (p0 p1 p2 p3 p4 p5 p6 p7)**

**north (p16 p17 p18 p19)**

**east (p8 p9 p10 p11 p12 p13 p14 p15)**

**south (p20 p21 p23 p24 p22)**

**Fisier Makefile**

all:

    vasy -I vhd -a -p -o multiplier\_control multiplier\_control

    vasy -I vhd -a -p -o multiplier\_data multiplier\_data

    vasy -I vhd -a -p -o multiplier multiplier

    boog multiplier\_control multiplier\_control

    boog multiplier\_data multiplier\_data

    asimut -zd multiplier multiplier r1

    ocp -ring multiplier multiplier

    nero -V multiplier multiplier

    genlib -v multiplier\_top

    asimut -zd multiplier\_top mult r2

    ring multiplier\_top multiplier\_top

    druc multiplier\_top

    s2r -r multiplier\_top multiplier\_top