**Proiectarea Circuitelor Integrate**

**Proiect: Algoritm de împărțire fără restaurare (numere pozitive)**

**Profesor coordonator : Sef lucr.dr.ing.**[**Tulbure Traian**](http://etc.unitbv.ro/~tulbure/)

**Student: Gheorghe Stefan**

**Specializarea: Electronica Aplicata**

**Anul de studiu: IV**

Cuprins

[Specificatii de proiectare 3](#_Toc535363451)

[Justificarea proiectarii 4](#_Toc535363452)

[Descrierea in VHDL a circuitului 6](#_Toc535363453)

[Layout-ul final generat in Alliance 13](#_Toc535363454)

# Specificatii de proiectare

|  |  |  |  |
| --- | --- | --- | --- |
| **Denumire port** | **Sens** | **Dim. [biți]** | **Semnificație** |
| ***op1*** | in | 8 | Primul operand |
| ***op2*** | in | 8 | Al doilea operand |
| ***cat*** | out | 8 | Catul impartirii |
| ***rest*** | out | 8 | Restul impartirii |
| ***start*** | in | 1 | Semnal pentru startul operației de înmulțire/împărțire |
| ***valid*** | out | 1 | Rezultat valid al operației de înmulțire/împărțire |
| ***reset*** | in | 1 | Reset |
| ***ck*** | in | 1 | Semnal de ceas |

**Algoritmul de impartire:**

- resetează P (n+1 biți)

- încarcă deîmpărțitul în A (n biți)

- încarcă împărțitorul în B (n biți)

- repeta de n ori

- dacă P este negativ (MSB=1) atunci

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

- P <= P + B

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

- P <= P + (-B) - deplasează cu o poziție stânga A (LSB A = not MSB P)

- dacă P este negativ (MSB=1) atunci

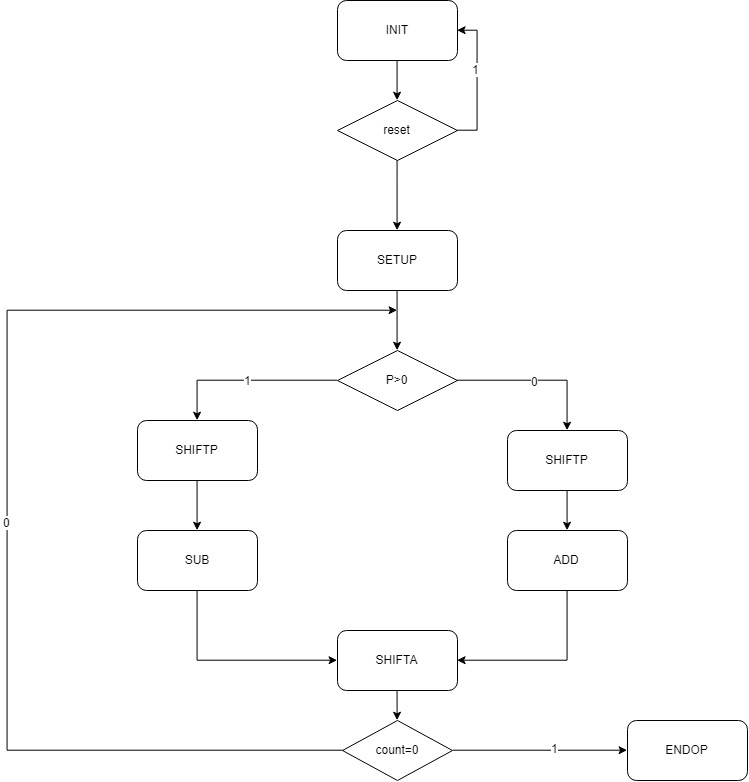
- P <= P + B

- P conține REST

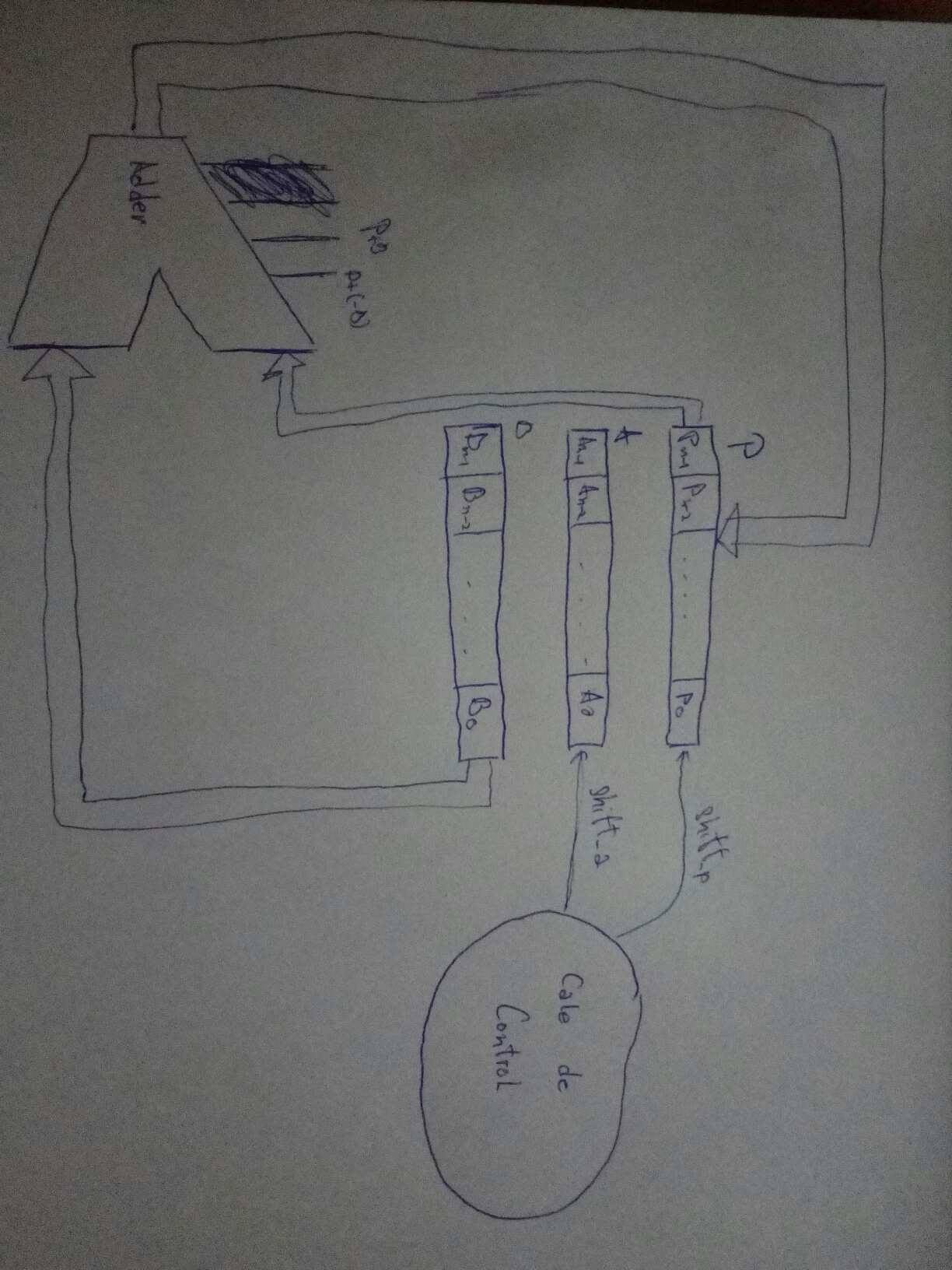
- A conține CÂT

# Justificarea proiectarii

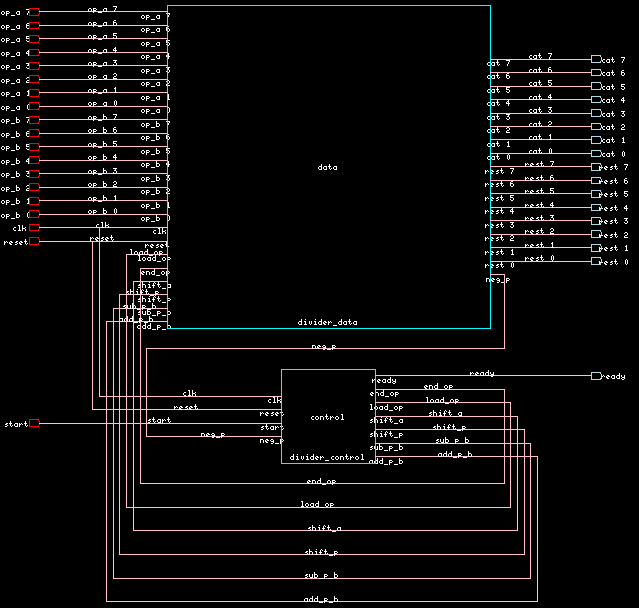
Calea de control:

****

Cale de date

****

Schema Bloc

****

Semnale interne

|  |  |  |
| --- | --- | --- |
| **Denumire** | **Dimensiune[biti]** | **Semnificatie** |
| load\_op | **1** | **Semnalul de incarcare a operanzilor** |
| end\_op | **1** | **Semnalul de finalizare a operatiei** |
| shift\_a | **1** | **Semnalul de deplasare a registrului a** |
| shift\_p | **1** | **Semnalul de deplasare a registrului p** |
| sub\_p\_b | **1** | **Semnalul de scadere a registrului b din registrul p** |
| add\_p\_b | **1** | **Semnalul de adunare a registrului b cu registrul p** |
| neg\_p | **1** | **Primul bit al registrului p** |

# Descrierea in VHDL a circuitului

Calea de control:

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_arith.ALL;

USE IEEE.std\_logic\_unsigned.ALL;

entity divider\_control is

port ( clk : in std\_logic;

reset : in std\_logic;

start : in std\_logic;

neg\_p : 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;

sub\_p\_b : out std\_logic;

add\_p\_b : out std\_logic

);

end divider\_control;

architecture divider\_control\_arch of divider\_control is

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

signal curenta : stare;

signal urmatoare: stare;

signal valid : std\_logic;

signal count : integer;

begin

CLC : process (reset, curenta, start, neg\_p, count) begin

case curenta is

when INIT =>

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

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

else urmatoare <= INIT;

end if;

when SETUP =>

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

else urmatoare <= SHIFTP;

end if;

when SHIFTP =>

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

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

else urmatoare <= SUB;

end if;

when SUB =>

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

else urmatoare <= SHIFTA;

end if;

when SHIFTA =>

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

elsif (neg\_p = '1') then urmatoare <= ADD;

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

else urmatoare <= SHIFTP;

end if;

when ADD =>

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

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

else urmatoare <= SHIFTP;

end if;

when ENDOP =>

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

else urmatoare <= INIT;

end if;

end case;

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

else load\_op <= '0';

end if;

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

else shift\_p <= '0';

end if;

if (curenta = SUB) then sub\_p\_b <= '1';

else sub\_p\_b <= '0';

end if;

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

else shift\_a <= '0';

end if;

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

else add\_p\_b <= '0';

end if;

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

else end\_op <= '0';

end if;

end process CLC;

REG : process (clk) begin

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

curenta <= urmatoare;

end if;

end process REG;

CONTOR : process (clk) begin

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

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

elsif (curenta = INIT) then count <= 8;

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 divider\_control\_arch;

Calea de date

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

USE IEEE.std\_logic\_arith.ALL;

USE IEEE.std\_logic\_unsigned.ALL;

entity divider\_data is

port ( op\_a : in std\_logic\_vector (7 downto 0);

op\_b : in std\_logic\_vector (7 downto 0);

result : out std\_logic\_vector (15 downto 0);

rest : 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;

sub\_p\_b : in std\_logic;

add\_p\_b : in std\_logic;

neg\_p : out std\_logic

);

end divider\_data;

architecture divider\_data\_arch of divider\_data is

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

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

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

signal r : std\_logic\_vector (15 downto 0);

begin

reg\_a : process (clk) begin

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

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

elsif (load\_op = '1') then a <= op\_a;

elsif (shift\_a = '1') then

a <= a(6 downto 0) & (not p(7));

end if;

end if;

end process reg\_a;

reg\_b : process (clk) begin

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

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

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

end if;

end if;

end process reg\_b;

reg\_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) & a(7);

elsif (sub\_p\_b = '1') then

p <= p - b;

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

end if;

end if;

end process reg\_p;

reg\_r : process (clk) begin

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

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

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

elsif (end\_op = '1') then

r(15 downto 8)<= a;

r(7 downto 0)<= p;

end if;

end if;

end process reg\_r;

neg\_p <= p(7);

result <= r;

end divider\_data\_arch;

Modulul top

LIBRARY IEEE;

USE IEEE.std\_logic\_1164.ALL;

entity divider is

port ( op\_a : in std\_logic\_vector (7 downto 0);

op\_b : in std\_logic\_vector (7 downto 0);

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

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

clk : in std\_logic;

reset : in std\_logic;

start : in std\_logic;

ready : out std\_logic

);

end divider;

architecture divider\_arch of divider is

signal load\_op : std\_logic;

signal shift\_a : std\_logic;

signal shift\_p : std\_logic;

signal sub\_p\_b : std\_logic;

signal add\_p\_b : std\_logic;

signal end\_op : std\_logic;

signal neg\_p : std\_logic;

component divider\_control

port ( clk : in std\_logic;

reset : in std\_logic;

start : in std\_logic;

neg\_p : 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;

sub\_p\_b : out std\_logic;

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

end component;

component divider\_data

port ( op\_a : in std\_logic\_vector (7 downto 0);

op\_b : in std\_logic\_vector (7 downto 0);

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

rest : 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;

sub\_p\_b : in std\_logic;

add\_p\_b : in std\_logic;

neg\_p : out std\_logic );

end component;

begin

DATA : divider\_data port map

( op\_a => op\_a,

op\_b => op\_b,

cat=>cat,

rest=>rest,

clk => clk,

reset => reset,

neg\_p => neg\_p,

end\_op => end\_op,

load\_op => load\_op,

shift\_a => shift\_a,

shift\_p => shift\_p,

sub\_p\_b => sub\_p\_b,

add\_p\_b => add\_p\_b );

CONTROL : divider\_control port map

( clk => clk,

reset => reset,

start => start,

neg\_p => neg\_p,

ready => ready,

end\_op => end\_op,

load\_op => load\_op,

shift\_a => shift\_a,

shift\_p => shift\_p,

sub\_p\_b => sub\_p\_b,

add\_p\_b => add\_p\_b );

end divider\_arch;

# Layout-ul final generat in Alliance

