Facultatea de Automatică și Calculatoare

ÎMPĂRȚIRE ZECIMALĂ

Indrumator: Lisman Dragos Florin
Student: Rus Ana-Maria Carina
Grupa: 30236
Data: 15.01.2024

Cuprins:

Rezumat

Proiectul are ca obiectiv întelegerea metodelor de împarțire zecimală și implementarea acestor metode într-un mod coerent. Obiectivele includ implementarea corecta a metodelor de împartire urmatoare: împartire cu refacere și împartire fara refacere, de asemenea unitățile lor de control ajută la familiarizarea cu FSM-uri.

Pentru a realiza proiectul, am folosit Vivado Xilinx, având codul sursa scris în limbajul VHDL.

Introducere

Contextul temei

Împărţirea zecimală este mai complexă decât împărţirea binară, deoarece cifrele câtului pot lua valori între 0 şi 9. Aceasta presupune efectuarea în fiecare etapă a unui număr variabil de adunări sau scăderi ale împărţitorului.
Metodele de împărţire binară pot fi aplicate şi pentru împărţirea zecimală, deoarece nu există deosebiri de principiu între acestea. Considerând că numerele sunt reprezentate în MS, se pot utiliza următoarele metode de împărţire zecimală:
• Metoda refacerii restului parţial;
• Metoda fără refacerea restului parţial;
• Metoda celor nouă multipli ai împărţitorului;
• Metoda înjumătăţirii împărţitorului;
• Metoda Gilman

Am ales să implementez metoda refacerii restului partial și metoda fără refacerea restului parțial. Acesta constă într-un multiplexor care alege catul, respectiv restul fiecarei metode în funcție de dorințele utilizatorului. Codul poate fi putin ineficient din cauza numarului de stări din unitățile de control.

Fundamentare teoretică

Metoda refacerii restului parţial

Metoda împărțirii cu refacerea restului parțial este o tehnică utilizată în procesul de împărțire, care implică realizarea repetată a scăderilor între divizor și restul parțial, până când obținem rezultatul dorit. În fiecare stagiu, se obţine o cifră a câtului pentru care restul parţial este încă pozitiv.
- Atunci când restul parţial devine negativ, se adună împărţitorul pentru a se reface restul parţial.
-Se foloseste un numărător inițializat cu 0 la începutul fiecarei etape, și incrementat la fiecare scădere a împărțitorului din restul parțial care ne ajuta la determiarea cifrei catului.
-Restul parțial se reface cand acesta devine negativ în urma scăderilor repetate
-Conţinutul decrementat al numărătorului reprezintă cifra câtului corespunzătoare etapei curente.
-Dacă numărătorul se incrementează numai atunci când restul parţial devine negativ, nu mai este necesară decrementarea acestuia. Înaintea începerii operaţiei, se testează dacă apare o depăşire şi dacă rezultatul operaţiei este zero, în mod similar cu împărţirea binară.
-Acest bloc conţine un numărător N care este decrementat în fiecare etapă a operaţiei şi un numărător N1 care se utilizează pentru numărarea operaţiilor de adunare sau de scădere efectuate pentru calculul unei cifre a câtului.

Considerăm $X = 684$, $Y = 28$. Execuția operației de împărțire este prezentată în Tabelul 3.3.

Tabelul 3.3. Execuția operației de împărțire 684:28 prin metoda refacerii restului parțial.

Pas	AZ	QZ	BZ	N	N_1	Operații
0	0 06	84	28	2	0	Inițializare
1	0 68 -	40	28	1	0	Deplasare stânga AZ_QZ
	28 0 40 -	40	28	1	1	Scădere BZ Incrementare N_1
	28 0 12 -	40	28	1	2	Scădere BZ Incrementare N_1
	28 9 84 +	40	28	1	3	Scădere BZ Incrementare N_1
	28 0 12	42	28	1	2	Adunare BZ Decrementare N_1, $QZ_0 = N_1$
2	1 24 -	20	28	0	0	Deplasare stânga AZ_QZ
	28 0 96 -	20	28	0	1	Scădere BZ Incrementare N_1
	28 0 68 -	20	28	0	2	Scădere BZ Incrementare N_1
	28 0 40 -	20	28	0	3	Scădere BZ Incrementare N_1
	28 0 12 -	20	28	0	4	Scădere BZ Incrementare N_1
	28 9 84 +	20	28	0	5	Scădere BZ Incrementare N_1
	28 0 12	24	28	0	4	Adunare BZ Decrementare N_1, $QZ_0 = N_1$

Câtul este 24, iar restul este 12.

Exemplu luat din laboratorul "Structura sistemelor de calcul – Dispozitive de înmulțire şi împărțire zecimală".

Metoda fără refacerea restului parţial

Metoda împărțirii fără refacerea restului parțial este o tehnică în care restul parțial nu este refăcut prin adunare cu divizorul după fiecare scădere. În schimb, procesul se bazează pe scăderi repetate ale divizorului din restul parțial până când restul devine mai mic decât divizorul. Algoritmul metodei fără refacerea restului parţial este următorul:

1. Se deplasează la stânga registrul combinat care conţine restul parţial şi deîmpărţitul. Se efectuează scăderi repetate ale împărţitorului din restul parţial, până când diferenţa devine negativă. Dacă s-au efectuat m1 scăderi, cifra câtului este m1-1.
2. Se deplasează la stânga registrul combinat care conţine restul parţial şi deîmpărţitul. Se efectuează adunări repetate ale împărţitorului la restul parţial, până când suma devine pozitivă. Dacă s-au efectuat m2 adunări, cifra câtului este 10-m2.
3. Se continuă cu etapele 1 şi 2, până când se obţin toate cifrele câtului.
4. Dacă după ultima etapă restul este negativ, se reface restul prin adunarea împărţitorului. Structura dispozitivului care implementează această metodă este similară cu cea a unui dispozitiv de împărţire zecimală care implementează metoda refacerii restului parţial.

Considerăm din nou $X = 684$, $Y = 28$. Execuția operației de împărțire este prezentată în Tabelul 3.4.

Tabelul 3.4. Execuția operației de împărțire 684:28 prin metoda fără refacerea restului parțial.

Pas	AZ	QZ	BZ	N	N_1	Operații
0	0 06	84	28	2	0	Inițializare
1	0 68 -	40	28	1	0	Deplasare stânga AZ_QZ
	28 0 40 -	40	28	1	1	Scădere BZ Incrementare N_1
	28 0 12 -	40	28	1	2	Scădere BZ Incrementare N_1
	28 9 84	42	28	1	3	Scădere BZ Incrementare N_1, $QZ_0 = N_1$-1
2	8 44 +	20	28	0	0	Deplasare stânga AZ_QZ
	28 8 72 +	20	28	0	1	Adunare BZ Incrementare N_1
	28 9 00 +	20	28	0	2	Adunare BZ Incrementare N_1
	28 9 28 +	20	28	0	3	Adunare BZ Incrementare N_1
	28 9 56 +	20	28	0	4	Adunare BZ Incrementare N_1
	28 9 84 +	20	28	0	5	Adunare BZ Incrementare N_1
	28 0 12	24	28	0	6	Adunare BZ Incrementare N_1, $QZ_0 = 10\text{-}N_1$

Câtul este 24, iar restul este 12.

Exemplu luat din laboratorul “Structura sistemelor de calcul – Dispozitive de înmulţire şi împărţire zecimală”.

Proiectare și implementare

Am ales sa introduc numerele dorite direct din cod, deoarece este o varianta mai usoara de înteles.

Sumator 4 biti:

Acest bloc implementează un sumator de 4 biți, care adună două numere binare de 4 biți (X și Y) și un bit de intrare de transport (Cin). Rezultatul adunării este furnizat în ieșirea S (STD_LOGIC_VECTOR de 4 biți), iar bitul de transport este furnizat în ieșirea Cout.
Sumatorul de 4 biți utilizează componente sumator1bit pentru fiecare bit al rezultatului, iar rezultatul final este obținut prin conectarea acestor componente.

Sumator Zecimal:

Acest bloc extinde funcționalitatea sumatorului4bit.
În acest bloc, un sumator4bit este utilizat pentru a aduna primii 4 biți ai numerelor X și Y, cu un bit de transport inițial (cin). Rezultatul adunării este stocat în sum și apoi utilizat pentru a aduna următorii 4 biți ai numerelor X și Y, generând astfel un rezultat final de 8 biți în sum1.
Bitul de transport final (cout) este furnizat în ieșirea Cout, iar rezultatul final al adunării este furnizat în ieșirea S.

Sumator 16 biti:

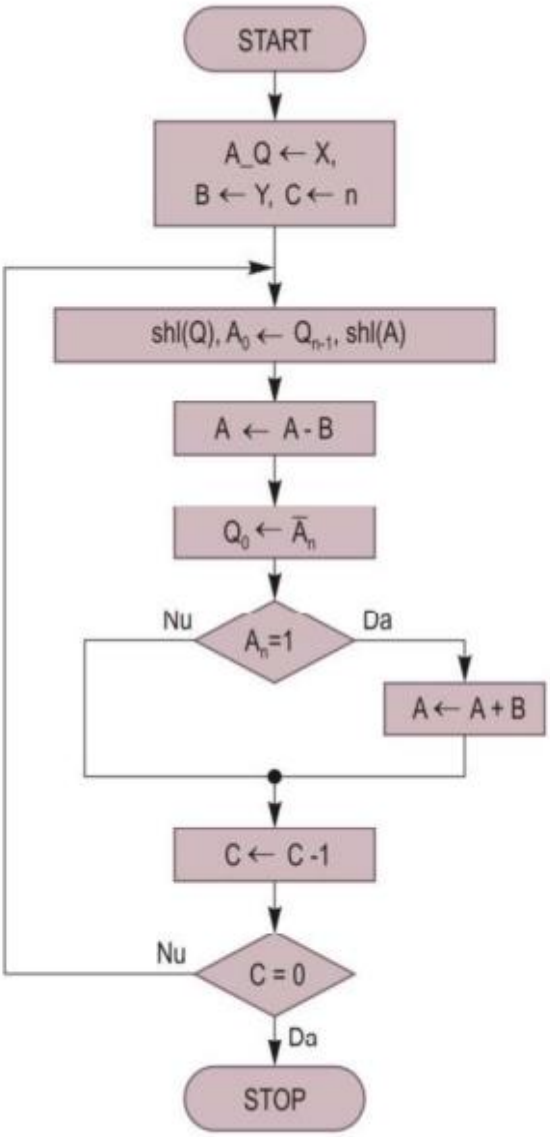
Acesta utilizează în total cinci instanțe ale sumatoruluiZecimal, conectându-le în serie pentru a aduna grupuri de 4 biți ai numerelor X și Y, împreună cu biții de transport corespunzători.
Rezultatul final al adunării este furnizat în ieșirea S (STD_LOGIC_VECTOR de 20 de biți), iar bitul de transport final este furnizat în ieșirea Cout.

Regiștrii

Pentru a putea păstra numerele împărțite a fost nevoie de niste regiștrii:
-a: pentru restul parțial și l-am declarat ca fiind registru de deplasare stânga
-q: pentru X registru de deplasare stânga
-b: pentru Y registru

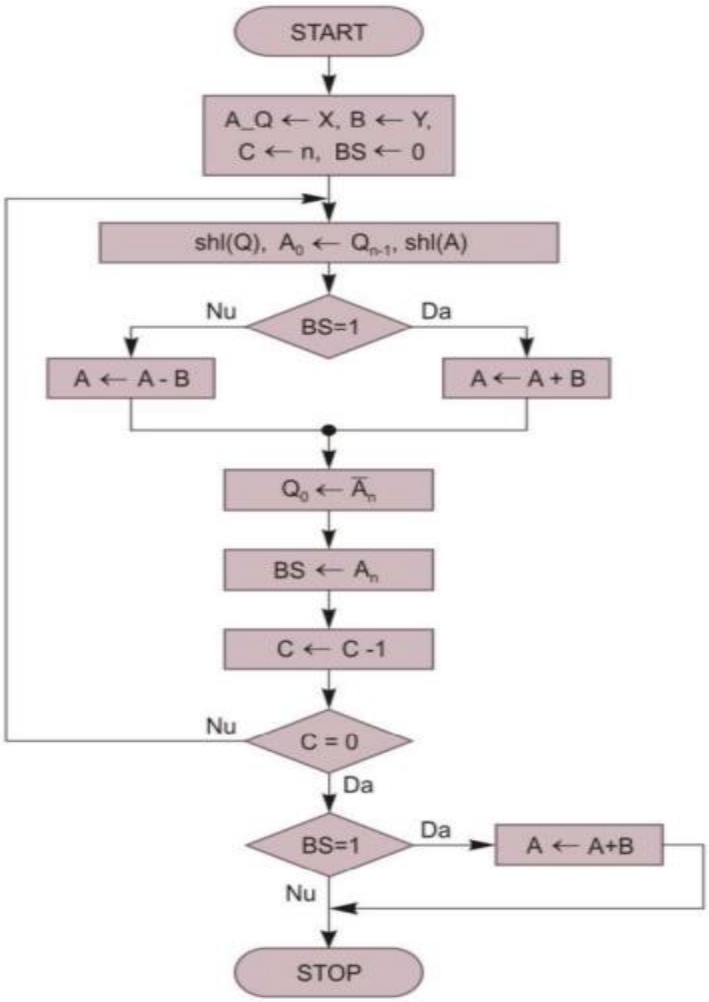
Unitatea de control cu refacerea restului

În unitatea de control am urmărit diagrama și am intrat printr-o serie de stări, fiecare avand funcționalitatea sa.
-idle: se face resetul împreună cu introducerea numărului de etape pe care îl are de facut programul
-initializare: inițializarea regiștrilor care păstrează împărțitorul și deîmpărțitul
-shiftarea lui A
-scaderea: scăderile repetate
-adunarea: adunarea în cazul în care restul e negativ
-scadsauadun: aici se decide dacă se face adunarea sau se continuă cu scăderile, în funcție de primul bit
-cifracat: se ocupa de decrementarea cifrei câtului
-sfarsit: se compară dacă am ajuns la sfârșit, dacă cifracat e 0


START
A_Q ← X,
B ← Y, C ← n
shl(Q), A0 ← Qn-1, shl(A)
A ← A - B
Q0 ← Ān
Nu
An=1
Da
A ← A + B
C ← C -1
Nu
C = 0
Da
STOP

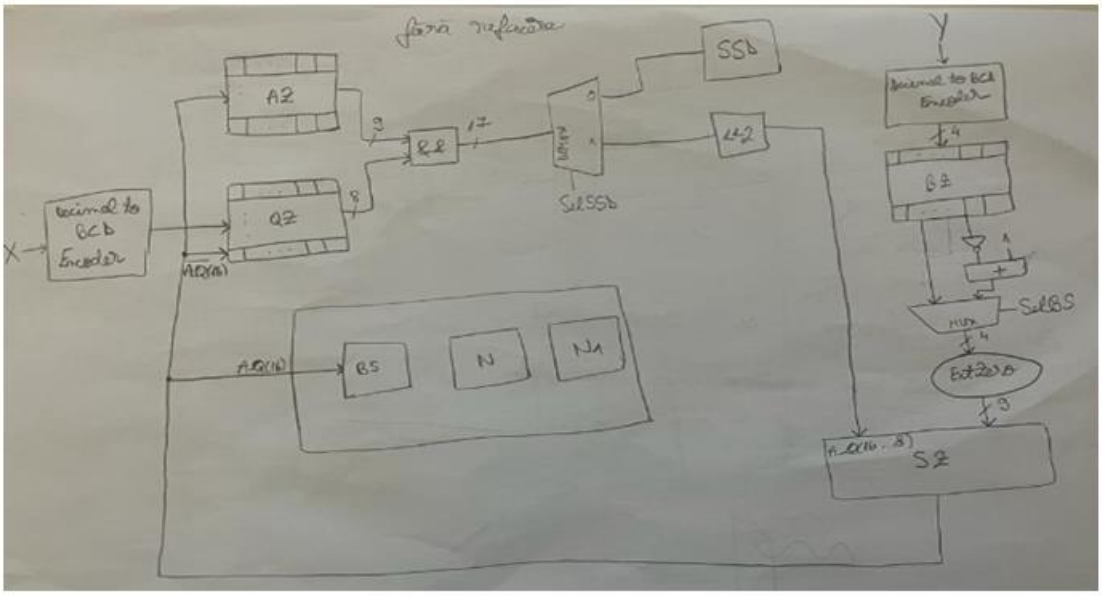
Unitatea de control fără refacerea restului

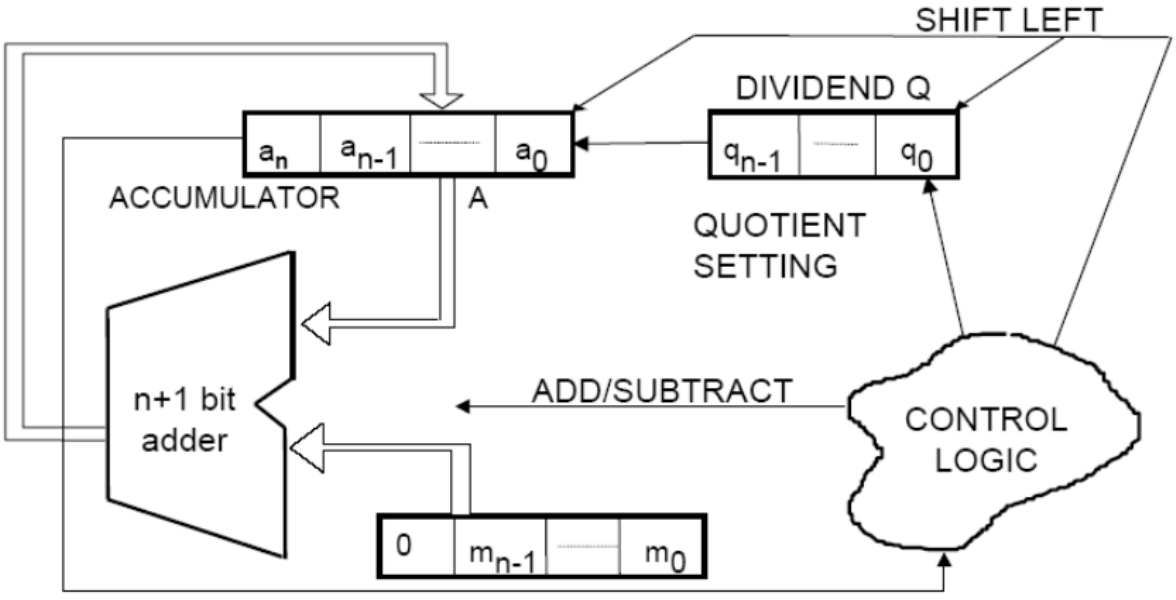
În unitatea de control am urmărit diagrama și am intrat printr-o serie de stări, fiecare având funcționalitatea sa.
-idle: se face resetul împreuna cu introducerea numărului de etape pe care îl are de făcut programul
-initializare: inițializarea regiștrilor care păstreaza împărțitorul și deîmpărțitul
-shiftarea lui A
-scaderea: scăderile repetate
-adunarea: adunarea în cazul în care restul e negativ
-verifbs: aici se decide daca se face adunarea sau se continua cu scaderile, în funcție de bistabil
-cifracat: se ocupa de decrementarea cifrei câtului
-sfarsit: se compară dacă am ajuns la sfarsit, dacă cifracat e 0



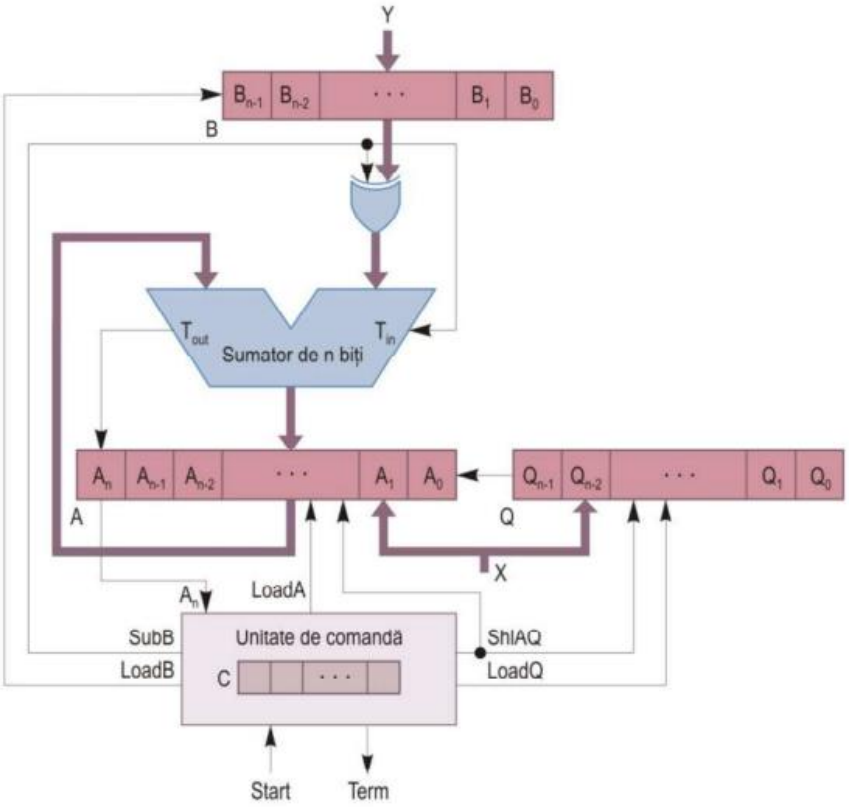
Împărțirea cu/fără refacerea restului parțial

Am legat componentele necesare fiecărei metode de împărțire: sumatoare și regiștrii împreuna cu unitățile de control aferente.
De asemenea bitii cei mai semnificativi a lui q sunt legați la intrarea sri pentru a pentru a se face deplasarea corecta, iar pentru q este cifra generata de unitatea de control. La sfarșit, a va fi restul parțial, iar q va fi câtul.





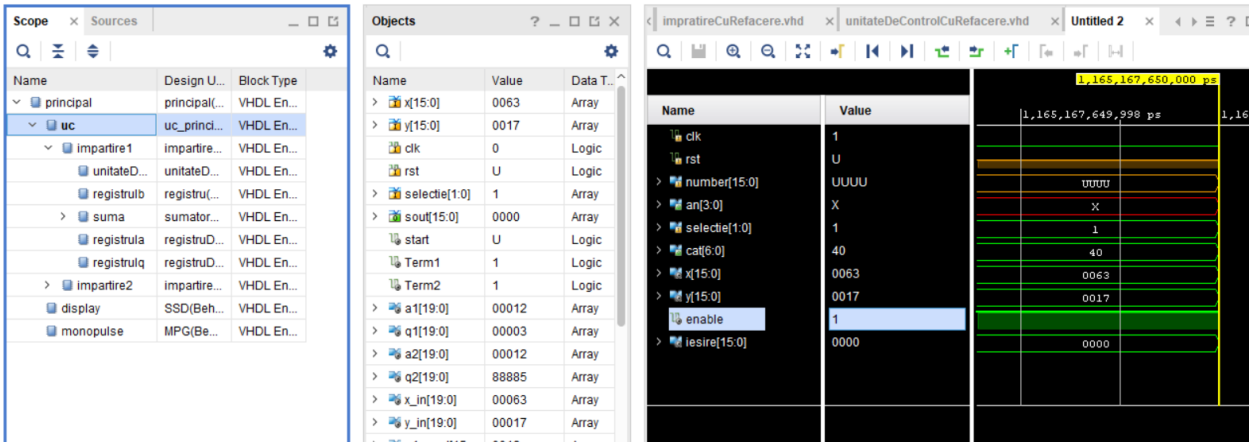
Împărțire fără refacere



Împărțire cu refacere

Rezultate

Proiectul nu este rezolvat în totalitate, având anumite greseli și nu am reușit implementarea acestuia pe plăcuță.



Pentru numerele X=63 și Y=17 câtul o sa fie 3, iar restul 12.

În urma simularii putem observa că câtul, respectiv restul celor doua metode sunt egale, iar valorile lor au rezultatul asteptat

a1=a2=12 (restul)

q1=q2=3 (câtul)

Concluzii

Rezultatul nu este în totalitate unul corect, neavând suficient exercițiu și înțelegere în ceea ce privește proiectele în limbajul VHDL, însa am incercat să mă apropiu cât de mult am putut de un rezultat acceptabil.

Ca dezvoltare ulterioara aș vrea să mentionez conectarea codului la placuta și introducerea numerelor de la plăcuță.

Bibliografie

Aplicatii-Proiectare-Digitala.pdf (utcluj.ro)

Dr. Baruch Zoltan Francisc - SSCAE - Cuprins (utcluj.ro)

https://ro.scribd.com/presentation/109619228/Decimal-Division-Implementation-Using-Vhdl

https://www.academia.edu/11042434/VHDL_Implementation_of_Non_Restoring_Division

https://fpgacoding.blogspot.com/p/blog-page_3.html

https://github.com/sergev/Guide-to-FPGA-Implementation-of-Arithmetic-Functions/blob/master/README.txt

Carte DE LA BIT LA PROCESOR. Introducere în arhitectura calculatoarelor, Florin Oniga

http://people.ee.duke.edu/~sorin/ece152/lectures/3.3-arith.pdf

https://media.digikey.com/pdf/Reference%20Design/Digi-Key%20DDS%20group/DKAN0003A_BCD_Division.pdf

https://media.digikey.com/pdf/Reference%20Design/Digi-Key%20DDS%20group/DKAN0003A_BCD_Division.pdf

https://media.digikey.com/pdf/Reference%20Design/Digi-Key%20DDS%20group/DKAN0003A_BCD_Division.pdf

Anexa

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

entity principal is
  Port (
   clk : in STD_LOGIC;
   rst : in std_logic;
   number : in STD_LOGIC_VECTOR (15 downto 0);
   an : out STD_LOGIC_VECTOR (3 downto 0);
   selectie: in std_logic_vector(1 downto 0);
   cat : out STD_LOGIC_VECTOR (6 downto 0)
  );
end principal;

architecture Behavioral of principal is

signal x: std_logic_vector(15 downto 0):=(others => '0');
signal y: std_logic_vector(15 downto 0):=(others => '0');
signal enable: std_logic;
signal iesire: std_logic_vector(15 downto 0);

component MPG is
    Port ( input : in STD_LOGIC;
           clk : in STD_LOGIC;
           enable : out STD_LOGIC
           );
end component;

component SSD is
    Port ( clk : in STD_LOGIC;
           number : in STD_LOGIC_VECTOR (15 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0)
```

```
        );
end component;

component uc_principal is
generic(n: natural);
  Port (
  x: in std_logic_vector(15 downto 0);
  y: in std_logic_vector(15 downto 0);
  clk: in std_logic;
  rst : in std_logic;
  selectie: in std_logic_vector(1 downto 0);
  sout: out STD_LOGIC_VECTOR(15 downto 0)
  );
end component;

begin

x(3 downto 0) <= x"3";
x(7 downto 4) <= x"6";
x(11 downto 8) <= x"0";
x(15 downto 12) <= x"0";

y(3 downto 0) <= x"7";
y(7 downto 4) <= x"1";
y(11 downto 8) <= x"0";
y(15 downto 12) <= x"0";

uc: uc_principal generic map (n => 20) port map (x,y,clk, rst, selectie,iesire);
display: SSD port map(clk, iesire, an, cat);
monopulse: MPG port map('1', clk, enable);

end Behavioral;
```

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

entity uc_principal is
generic(n: natural);
  Port (
x: in std_logic_vector(15 downto 0);
y: in std_logic_vector(15 downto 0);
clk: in std_logic;
rst : in std_logic;
selectie: in std_logic_vector(1 downto 0);
sout: out STD_LOGIC_VECTOR(15 downto 0)
  );
end uc_principal;

architecture Behavioral of uc_principal is

component impartireaCuRefacere is
generic(n : natural:= 20);
 Port (
 clk: in std_logic;
 rst : in STD_LOGIC;
 start : in STD_LOGIC;
 x : in STD_LOGIC_VECTOR(n-1 downto 0);
 y : in STD_LOGIC_VECTOR(n-1 downto 0);
 a : out STD_LOGIC_VECTOR(n-1 downto 0);
 q : out STD_LOGIC_VECTOR(n-1 downto 0);
 Term : out STD_LOGIC);
end component;

component impartireaFaraRefacere is
generic(n : natural:= 20);
 Port (
 clk: in std_logic;
 rst : in STD_LOGIC;
```

```
 start : in STD_LOGIC;
 x : in STD_LOGIC_VECTOR(n-1 downto 0);
 y : in STD_LOGIC_VECTOR(n-1 downto 0);
 a : out STD_LOGIC_VECTOR(n-1 downto 0);
 q : out STD_LOGIC_VECTOR(n-1 downto 0);
 Term : out STD_LOGIC);
end component;

signal start, Term1, Term2: std_logic;
signal a1, q1, a2, q2: std_logic_vector(19 downto 0);
signal x_in,y_in: std_logic_vector(19 downto 0):=(others => '0');
signal a1_sout, a2_sout, q1_sout, q2_sout, sel: std_logic_vector(15 downto
0):=(others => '0');

begin

x_in <= "0000" & x;
y_in <= "0000" & y;

impartire1: impartireaCuRefacere port map(clk => clk, rst => rst, start => '1', x =>
x_in, y => y_in,
a => a1, q => q1, Term => Term1);

impartire2: impartireaFaraRefacere port map(clk => clk, rst => rst, start => '1', x =>
x_in, y => y_in,
a => a2, q => q2, Term => Term2);

a1_sout <= a1(15 downto 0);
a2_sout <= a2(15 downto 0);

q1_sout <= q1(15 downto 0);
q2_sout <= q2(15 downto 0);

process(selectie)
begin
```

```
case selectie is
   when "00"=> sel<=a1_sout;
   when "01"=> sel<=q1_sout;
   when "10"=> sel<=a2_sout;
   when others=> sel<=q2_sout;
end case;
end process;
sout <= sel;
end Behavioral;
```

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

entity SSD is
   Port ( clk : in STD_LOGIC;
          number : in STD_LOGIC_VECTOR (15 downto 0);
          an : out STD_LOGIC_VECTOR (3 downto 0);
          cat : out STD_LOGIC_VECTOR (6 downto 0));
end SSD;

architecture Behavioral of SSD is

signal cnt: STD_LOGIC_VECTOR(15 downto 0);
signal dig:STD_LOGIC_VECTOR(3 downto 0);

begin

process(clk)
begin
   if rising_edge(clk) then
      cnt<=cnt+1;
   end if;
end process;

process( cnt(15 downto 14))
begin
   case cnt (15 downto 14) is
      when "00" => an <= "1110";
      when "01" => an <= "1101";
      when "10" => an <= "1011";
      when "11" => an <= "0111";
      when others => an <="1110";
   end case;
```

```
end process;


process( cnt(15 downto 14), number)
begin
  case cnt (15 downto 14) is
      when "00" => dig <= number(3 downto 0);
      when "01" => dig <= number(7 downto 4);
      when "10" => dig <= number(11 downto 8);
      when "11" => dig <= number(15 downto 12);
      when others => an <= number(3 downto 0);
   end case;
end process;



process(dig)
begin
   case dig is
      when "0001" => cat<= "1111001";  --1
      when "0010" => cat<= "0100100";  --2
      when "0011" => cat<= "0110000";  --3
      when "0100" => cat<= "0011001";  --4
      when "0101" => cat<= "0010010";  --5
      when "0110" => cat<= "0000010";  --6
      when "0111" => cat<= "1111000";  --7
      when "1000" => cat<= "0000000";  --8
      when "1001" => cat<= "0010000";  --9
      when "1010" => cat<= "0001000";  --A
      when "1011" => cat<= "0000011";  --b
      when "1100" => cat<= "1000110";  --C
      when "1101" => cat<= "0100001";  --d
      when "1110" => cat<= "0000110";  --E
      when "1111" => cat<= "0001110";  --F
      when others => cat<= "1000000";  --0
```

```
    end case;
end process;
end Behavioral;

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

entity MPG is
   Port ( input : in STD_LOGIC;
          clk : in STD_LOGIC;
          enable : out STD_LOGIC);
end MPG;

architecture Behavioral of MPG is

signal count_int : std_logic_vector(31 downto 0) :=x"00000000";
signal Q1 : std_logic;
signal Q2 : std_logic;
signal Q3 : std_logic;

begin

enable <= Q2 AND (not Q3);
process (clk)
  begin
   if (rising_edge(clk)) then
     count_int <= count_int + 1;
   end if;
end process;

process (clk)
   begin
      if clk'event and clk='1' then
```

```
        if count_int(15 downto 0) = "1111111111111111" then
           Q1 <= input;
        end if;
      end if;
end process;

process (clk)
begin
   if clk'event and clk='1' then
      Q2 <= Q1;
      Q3 <= Q2;
    end if;
end process;


end Behavioral;
```

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

entity unitateDeControlCuRefacere is
generic(n: natural);
Port(
   start: in std_logic;
   clk: in std_logic;
   rst: in std_logic;
   an: in std_logic;
   selmuxin: out std_logic;
   loadA,loadB,loadQ: out std_logic;
   shla,shlq: out std_logic;
   subb:out std_logic;
   term: out std_logic;
   nn: out std_logic_vector(3 downto 0)
   );
end unitateDeControlCuRefacere;

architecture Behavioral of unitateDeControlCuRefacere is
type tip_stare is
(idle,initializare,shiftareA,scadere,scadsauadun,adunare,cifracat,sfarsit,stop);
signal stare: tip_stare;
signal cn: natural:=5;
signal t: std_logic:='0';
signal n1 : std_logic_vector(3 downto 0):="0000";
begin

process(clk, rst, start, an)
begin
   if clk'event and clk = '1' then
      if rst = '1' then
         stare <= idle;
```

```
end if;

case stare is
   when idle =>
      t <= '0';
      loada <= '0';
      loadb <= '0';
      loadq <= '0';
      shla <= '0';
      shlq <= '0';
      subb <= '0';
      n1 <= (others => '0');
      nn <= "0000";
      cn <= 5;
      if start = '1' then
         stare <= initializare;
      else
         stare <= idle;
      end if;
   when initializare =>
       loada <= '1';
       loadb <= '1';
       loadq <= '1';
       selmuxin <= '1';
       shla <= '0';
       shlq <= '0';
       subb <= '0';
       n1<="0000";
       stare <= shiftareA;
   when shiftareA =>
       loada <= '0';
       loadb <= '0';
       loadq <= '0';
       shla <= '1';
       shlq <= '0';
```

```
    subb <= '0';
    selmuxin <= '0';
    n1<="0000";
    stare <= scadere;
when scadere =>
    loada <= '1';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '1';
    selmuxin <= '0';
    n1<= n1 + 1;
    stare <= scadsauadun;
when scadsauadun =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
    if(an ='1') then
      stare <= adunare;
    else
      stare <= scadere;
    end if;
when adunare =>
    loada <= '1';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    n1<= n1 - 1;
```

```
    stare <= cifracat;
when cifracat =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '1';
    subb <= '0';
    nn<=n1;
    selmuxin <= '0';
    cn <= cn - 1;
    stare <= sfarsit;
when sfarsit =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
    if cn = 0 then
      stare <= stop;
    else
      stare <= shiftareA;
    end if;
when stop =>
    t <= '1';
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
when others => stare <= idle;
```

```
      end case;

   end if;
end process;
term <= t;


end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity impartireaCuRefacere is
generic(n : natural:= 20);
 Port (
 clk: in std_logic;
 rst : in STD_LOGIC;
 start : in STD_LOGIC;
 x : in STD_LOGIC_VECTOR(n-1 downto 0);
 y : in STD_LOGIC_VECTOR(n-1 downto 0);
 a : out STD_LOGIC_VECTOR(n-1 downto 0);
 q : out STD_LOGIC_VECTOR(n-1 downto 0);
 Term : out STD_LOGIC);
end impartireaCuRefacere;

architecture Behavioral of impartireaCuRefacere is

signal loada,loadb,loadq,shla,shlq,subb,ovf,selmuxin: std_logic :='0';
signal an,outsuman: std_logic:='1';
signal outb,insum,outsum,outq: std_logic_vector(n-1 downto 0);
signal outa,ina: std_logic_vector(n downto 0);
signal nn: std_logic_vector(3 downto 0):=(others =>'0');

component unitateDeControlCuRefacere is
generic(n: natural);
```

```
Port(
   start: in std_logic;
   clk: in std_logic;
   rst: in std_logic;
   an: in std_logic;
   selmuxin: out std_logic;
   loadA,loadB,loadQ: out std_logic;
   shla,shlq: out std_logic;
   subb:out std_logic;
   term: out std_logic;
   nn: out std_logic_vector(3 downto 0)
   );
end component;

component registruDeplasareStanga is
GENERIC ( n : natural ) ;
 Port (
 d: in std_logic_vector(n-1 downto 0);
 sri: in std_logic_vector(3 downto 0);
 load: in std_logic;
 ce: in std_logic;
 clk: in std_logic;
 rst: in std_logic;
 q: out std_logic_vector(n-1 downto 0)
 );
end component;

component registru is
 GENERIC ( n : natural ) ;
 Port (
 d: in std_logic_vector(n-1 downto 0);
 ce: in std_logic;
 clk: in std_logic;
 rst: in std_logic;
 q: out std_logic_vector(n-1 downto 0)
```

```
 );
end component;

component sumator_16biti is
  Port (
    X : in STD_LOGIC_VECTOR (19 downto 0);
    Y : in STD_LOGIC_VECTOR (19 downto 0);
    Cin : in STD_LOGIC;
    S : out STD_LOGIC_VECTOR (19 downto 0);
    Cout : out STD_LOGIC
  );
end component;

begin

unitateDeControl: unitateDeControlCuRefacere generic map ( n => n) port map
(start=>start,clk=>clk,rst=>rst,an=>an,
selmuxin=>selmuxin,loada=>loada,loadb=>loadb,loadq=>loadq,shla=>shla,shlq=>s
hlq,subb=>subb,term=>term,nn=>nn);

registrulb: registru generic map(n => n) port
map(d=>y,ce=>loadb,clk=>clk,rst=>rst,q=>outb);

insum <= x"99999" - outB when subb = '1' else outB;

suma: sumator_16biti port map( x=>outa(n-1 downto
0),y=>insum,cin=>subb,s=>outsum, cout=>outsuman);

an<=outsum(n-1) or outsum(n-2) or outsum(n-3) or outsum(n-4) ;
ina <= '0' & x"00000" when selmuxin = '1' else an & outsum;

registrula: registruDeplasareStanga generic map(n => n+1) port map( d =>
ina,sri=> outq(n-1 downto n-4),load=>loada,
 ce=>shla,clk=>clk,rst=>rst,q=>outa);
```

```
registrulq: registruDeplasareStanga generic map(n => n) port map( d => x,sri=>
nn,load=>loadq,ce=>shlq,clk=>clk,
 rst=>rst,q=>outq);

q<=outq;
a<= outa(n-1 downto 0);

end Behavioral;


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

entity impartireaFaraRefacere is
generic(n : natural:= 20);
 Port (
 clk: in std_logic;
 rst : in STD_LOGIC;
 start : in STD_LOGIC;
 x : in STD_LOGIC_VECTOR(n-1 downto 0);
 y : in STD_LOGIC_VECTOR(n-1 downto 0);
 a : out STD_LOGIC_VECTOR(n-1 downto 0);
 q : out STD_LOGIC_VECTOR(n-1 downto 0);
 Term : out STD_LOGIC);
end impartireaFaraRefacere;

architecture Behavioral of impartireaFaraRefacere is

signal loada,loadb,loadq,shla,shlq,subb,ovf,selmuxin: std_logic :='0';
signal an,outsuman: std_logic:='1';
signal outb,insum,outsum,outq: std_logic_vector(n-1 downto 0);
signal outa,ina: std_logic_vector(n downto 0);
signal nn: std_logic_vector(3 downto 0):=(others =>'0');
```

```
component unitateDeControlFaraRefacere is
generic(n: natural);
Port(
   start: in std_logic;
   clk: in std_logic;
   rst: in std_logic;
   an: in std_logic;
   selmuxin: out std_logic;
   loadA,loadB,loadQ: out std_logic;
   shla,shlq: out std_logic;
   subb:out std_logic;
   term: out std_logic;
   nn: out std_logic_vector(3 downto 0)
   );
end component;

component registruDeplasareStanga is
GENERIC ( n : natural ) ;
 Port (
 d: in std_logic_vector(n-1 downto 0);
 sri: in std_logic_vector(3 downto 0);
 load: in std_logic;
 ce: in std_logic;
 clk: in std_logic;
 rst: in std_logic;
 q: out std_logic_vector(n-1 downto 0)
 );
end component;

component registru is
 GENERIC ( n : natural ) ;
 Port (
 d: in std_logic_vector(n-1 downto 0);
 ce: in std_logic;
 clk: in std_logic;
```

```
 rst: in std_logic;
 q: out std_logic_vector(n-1 downto 0)
 );
end component;

component sumator_16biti is
  Port (
    X : in STD_LOGIC_VECTOR (19 downto 0);
    Y : in STD_LOGIC_VECTOR (19 downto 0);
    Cin : in STD_LOGIC;
    S : out STD_LOGIC_VECTOR (19 downto 0);
    Cout : out STD_LOGIC
  );
end component;

begin

unitateDeControl: unitateDeControlFaraRefacere generic map ( n => n) port map
(start=>start,clk=>clk,rst=>rst,an=>an,
selmuxin=>selmuxin,loada=>loada,loadb=>loadb,loadq=>loadq,shla=>shla,shlq=>s
hlq,subb=>subb,term=>term,nn=>nn);

registrulb: registru generic map(n => n) port
map(d=>y,ce=>loadb,clk=>clk,rst=>rst,q=>outb);

insum <= x"99999" - outB when subb = '1' else outB;

suma: sumator_16biti port map( x=>outa(n-1 downto
0),y=>insum,cin=>subb,s=>outsum, cout=>outsuman);

an<=outa(n-1) or outa(n-2) or outa(n-3) or outa(n-4) ;
ina <= '0' & x"00000" when selmuxin = '1' else an & outsum;

registrula: registruDeplasareStanga generic map(n => n+1) port map( d =>
ina,sri=> outq(n-1 downto n-4),load=>loada,
```

```
 ce=>shla,clk=>clk,rst=>rst,q=>outa);

registrulq: registruDeplasareStanga generic map(n => n) port map( d => x,sri=>
nn,load=>loadq,ce=>shlq,clk=>clk,
rst=>rst,q=>outq);

q<=outq;
a<= outa(n-1 downto 0);

end Behavioral;


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

entity unitateDeControlFaraRefacere is
generic(n: natural);
 Port (
 start,clk,rst: in std_logic;
 an: in std_logic;
 selmuxin: out std_logic;
 loadA,loadB,loadQ: out std_logic;
 shla,shlq: out std_logic;
 subb:out std_logic;
 term: out std_logic;
 nn: out std_logic_vector(3 downto 0)
 );
end unitateDeControlFaraRefacere;

architecture Behavioral of unitateDeControlFaraRefacere is

type tip_stare is
(idle,initializare,shiftareA,decizie1,verifbs,scadere,adunare,cifracat,comparare,sto
p);
```

```
signal stare: tip_stare;
signal cn: natural:=5;
signal t: std_logic:='0';
signal n1 : std_logic_vector(3 downto 0):="0000";
signal bistabil : std_logic := '1';
begin

process(clk,rst,start,an)
begin
 if rising_edge(clk) then
   if rst = '1' then
      stare<=idle;
   end if;
   case stare is
      when idle =>
         t <= '0';
         loada <= '0';
         loadb <= '0';
         loadq <= '0';
         shla <= '0';
         shlq <= '0';
         subb <= '0';
         selmuxin <= '0';
         n1<=(others =>'0');
         nn<="0000";
         cn <= 4;
         if start = '1' then
            stare <= initializare;
         else
            stare <= idle;
         end if;
      when initializare =>
         loada <= '1';
         loadb <= '1';
         loadq <= '1';
```

```
    selmuxin <= '1';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    n1<="0000";
    bistabil<='0';
    stare <= shiftareA;
  when shiftareA =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '1';
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
    n1<="0000";
    stare <= decizie1;
   when decizie1 =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
    if an = bistabil then
       stare <= verifbs;
    else
       stare <= cifracat;
    end if;
   when verifbs =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
```

```
    shlq <= '0';
    subb <= '0';
    selmuxin <= '0';
    n1<= n1 + 1;
    if(an ='1') then
       stare <= adunare;
    else
       stare <= scadere;
    end if;
when scadere =>
    loada <= '1';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '1';
    selmuxin <= '0';
    stare <= decizie1;
when adunare =>
    loada <= '1';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '0';
    subb <= '0';
    stare <= decizie1;
when cifracat =>
    loada <= '0';
    loadb <= '0';
    loadq <= '0';
    shla <= '0';
    shlq <= '1';
    subb <= '0';
    selmuxin <= '0';
    stare <= comparare;
```

```
        if BS = '0' then
            nn<= n1-2;
        else
            nn <= "1010" - n1;
        end if;
      when comparare =>
        loada <= '0';
        loadb <= '0';
        loadq <= '0';
        shla <= '0';
        shlq <= '0';
        subb <= '0';
        selmuxin <= '0';
        cn <= cn - 1;
        bistabil <= an;
        if cn = 0 then
          stare <= stop;
        else
          stare <= shiftareA;
        end if;
      when stop =>
        t <= '1';
        loada <= '0';
        loadb <= '0';
        loadq <= '0';
        shla <= '0';
        shlq <= '0';
        subb <= '0';
        selmuxin <= '0';
      when others => stare <= idle;
   end case;
 end if;
end process;
term <= t;
end Behavioral;
```