UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Arithmetic Logic Unit - Proiect

Structura Sistemelor de Calcul

Autor: Dîrnu Talida

Grupa: 30236

Profesor îndrumător: Șl. Dr. Ing. Florin Lișman

Facultatea de Automatica
si Calculatoare

CTI

2 Decembrie 2023

Cuprins

1 Descrierea proiectului

ALU este o componentă cheie a unei unități centrale de prelucrare (CPU) dintr-un computer. Este un bloc de circuit logic digital care efectuează operațiile aritmetice și logice asupra datelor.

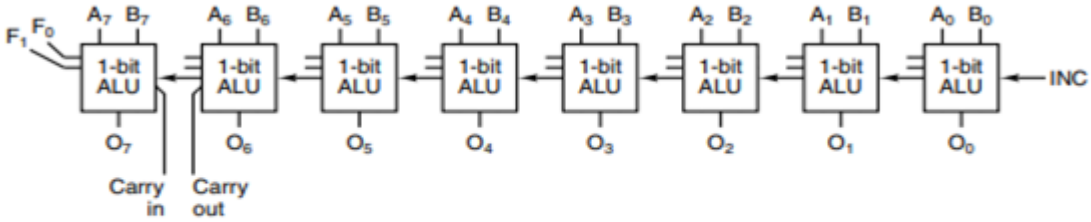


Figure 3-19. Eight 1-bit ALU slices connected to make an 8-bit ALU. The enables and invert signals are not shown for simplicity.

Ca și funcționalități ale acestui proiect, am ales să implementez operații pe numere pe 8 biți, primul bit fiind bitul de semn. Am implementat operațiile logice AND OR, XOR și NOT pe biți, iar ca operații aritmetice adunare, scădere, înmulțire și împărțire. Se va putea realiza selecția operației dorite folosind resursele plăcii, iar rezultatul operației va fi afișat în hexazecimal pe afișorul SSD.

2 Introducere. Descrierea implementării operațiilor

Având de implementat funcționalitățile prezentate anterior, am folosit ca și resurse în mare parte materialele de laborator și cele de curs, furnizate de profesul coordonator, astfel că am selectat pentru fiecare operație implementată câte o metodă, după preferințe. În urma implementării, unele secțiuni au fost modificate în funcție de rezultatele experimentale.

2.1 ADUNAREA

Pentru adunare am utilizat un **sumator pe 8 biți cu anticiparea transportului - Carry LookAhead Adder**, circuit care crește viteza operației de adunare prin reducerea timpului necesar pentru generarea semnalelor de transport. În cazul acestui sumator, intrarea de transport necesară pentru un etaj este generată în mod direct, utilizând semnale de la toate etajele precedente, în loc de a se aștepta propagarea lentă a transporturilor de la un etaj la altul.

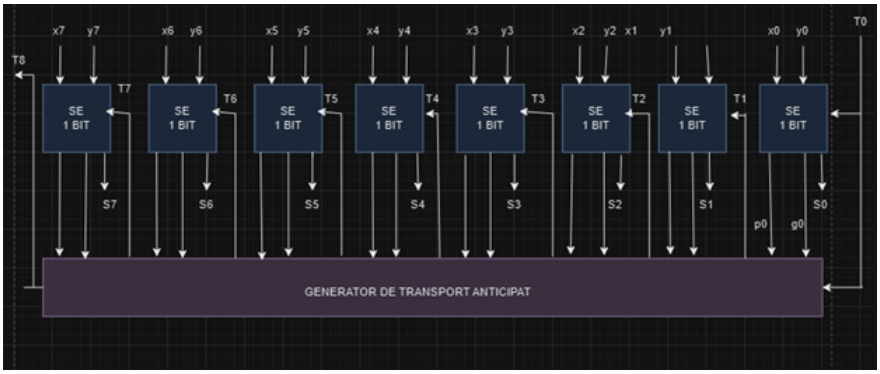


Figura 1: Schema bloc a sumatorului cu anticipare de transport pe 8 biți

Generatorul de transport anticipat convertește cele 8 seturi de semnale p, g în intrările de transport necesare pentru cele 8 sumatoare. Fiecare sumator de 1 bit produce și bitul corespunzător al sumei, în același mod ca și un sumator elementar.
Ecuații:

- T(i+1) = x(i) * y(i) + (x(i) + y(i)) * T(i)
- g = generarea transportului: g(i) = x(i) * y(i)
- p = propagarea intrării de transport la ieșirea de transport: p(i) = x(i) + y(i)

Componentele necesare identificate:

- **8 module de adunare pe un singur bit(SE)**: acestea efectuează adunarea a doi biți, producând suma și un transport. Fiecare modul poate aduna un bit din cei 8 biți ai unui număr
- **Un circuit generator de anticipare a transportului**: un circuit special care evaluează transportul dintre biți în timpul adunării și decide cum să propage acest transport la biții următori
- **Registre de stocare**: pentru a păstra rezultatul sumei și transportul pe măsură ce adunările sunt efectuate în serie, se utilizează registre pentru a stoca și actualiza aceste valori.

Pentru operația de adunare, în entitatea principală (ALU) a fost tratat cazul de overflow. Overflow la adunare are loc atunci când avem de a face cu următoarele cazuri, care indică un rezultat incorect matematic: pozitiv + pozitiv = negativ sau negativ + negativ = pozitiv. La nivel de cod, acest lucru se întâmplă atunci când biții 7 ai operanzilor sunt identici, dar bitul 7 al rezultatului este diferit.

2.2 SCĂDEREA

Scăderea se efectuează implicit tot prin sumatorul descris mai sus. Se folosește aceeași logică, dar în loc de al doilea operand normal, se va folosi complementul său față de 2 (C2 = not(B) + 1).

Similar cu operația de adunare, avem cazul de overflow și la scădere, în următoarele cazuri: pozitiv - negativ = negativ și negativ - pozitiv = pozitiv, adică biții 7 ai operanzilor diferă, iar bitul 7 al rezultatului diferă de bitul 7 al primului operand.

2.3 ÎNMULȚIREA

Pentru înmulțire, din metodele prezentate în cadrul laboratorului, am ales să folosesc **metoda Booth**.
În fiecare pas al operației de înmulțire prin metoda Booth, se testează doi biți alăturați ai înmulțitorului, Yi (bitul curent) și Yi-1 (bitul testat în pasul precedent, numit și bit de referință). Bitul de referință pentru bitul cel mai puțin semnificativ al înmulțitorului este Y-1 = 0. Testarea se efectuează începând cu bitul cel mai puțin semnificativ al înmulțitorului. În funcție de valoarea biților testați YiYi-1, se realizează operațiile indicate:

- **00** - deplasare produs parțial la dreapta
- **01** - adunare deînmulțit, deplasare produs parțial la dreapta
- **10** - scădere deînmulțit, deplasare produs parțial la dreapta
- **11** - deplasare produs parțial la dreapta

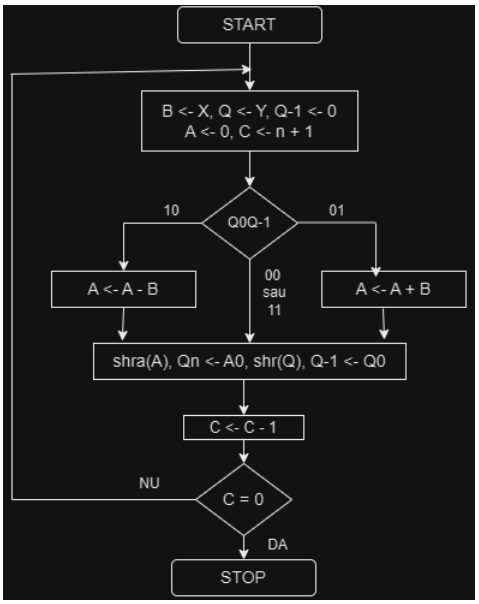


Figura 2: Metoda Booth - Organigramă

2.4 ÎMPĂRȚIREA

Pentru împărțire, am ales metoda de **împărțire cu refacerea restului parțial**.
Fiecare etapă a operației de împărțire începe cu o deplasare a restului parțial la stânga cu o poziție. Se efectuează apoi scăderea împărțitorului din restul parțial, obținându-se noul rest parțial. Dacă se obține un număr pozitiv, cifra corespunzătoare a câtului este 1. Dacă se obține un număr negativ, cifra corespunzătoare a câtului este 0, împărțitorul fiind adunat la restul parțial pentru a-l reface.
Pentru obținerea unui cât cu n cifre de mărime, aceste operații se repetă de n ori. Procesul de împărțire se poate opri dacă restul parțial devine 0.

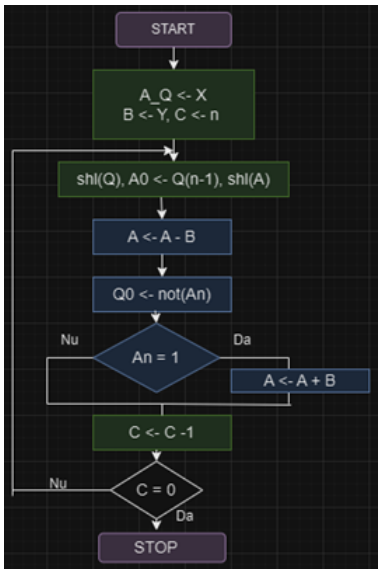


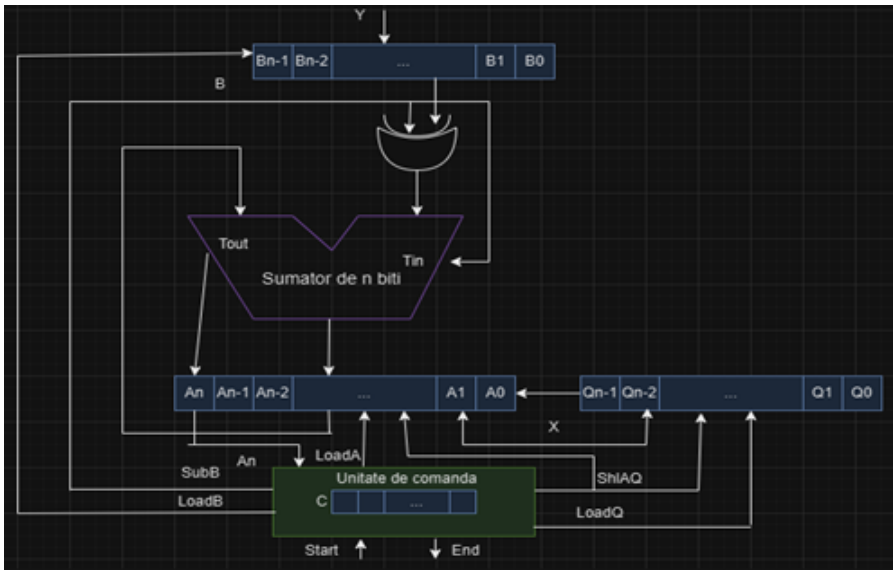
Figura 3: Împărțire cu refacerea restului parțial - Organigramă

În fiecare etapă a operației, se deplasează registrele A și Q la stânga cu o poziție (shl(A), shl(Q)), iar apoi se realizează o scădere a împărțitorului (registrul B) din restul parțial (registrul

A). Dacă rezultă un număr pozitiv în acumulator (An = 0), cifra câtului este 1, care se introduce în poziția Q0 a registrului Q. Dacă rezultatul este un număr negativ în acumulator (An = 1), cifra câtului care se introduce în poziția Q0 este 0. Deci, valoarea câtului care se obține în această etapă este: Q0 = not(An).
Componentele necesare identificate:

- **2 registre A și Q**: în aceste registre se încarcă deîmpărțitul X, partea mai semnificativă fiind încărcată în registrul A
- **Un registru B**: aici vom încărca împărțitorul Y
- **Un numărător C**: se inițializează cu numărul cifrelor de mărime ale împărțitorului
- **Un circuit de deplasare(Shifter)**: pentru deplasarea la stânga în fiecare etapă a registrelor A și Q
- **O unitate de adunare/scădere(Adder)**: pentru efectuarea scăderii împărțitorului din restul parțial

Deoarece metoda de împărțire utilizată este pentru numere fără semn, s-a efectuat împărțirea pentru reprezentarea în C2 în cazul numerelor negative, apoi câtul și restul au fost transformate după caz, conform unor reguli matematice simple: dacă doar unul dintre operanzi este negativ, câtul va fi luat în reprezentarea C2. Restul va avea semnul deîmpărțitului, astfel ca vom lua reprezentarea C2 în cazul în care bitul 7 al deîmpărțitului este 1.



3 Schema circuitului

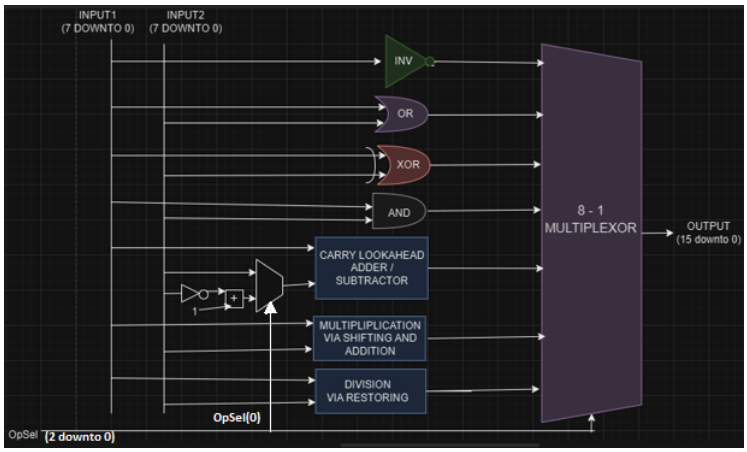


Figura 4: Schemă ALU simplificată

Entitatea ALU va avea ca și intrări două numere pe 8 biți (in1 și in2). Deoarece înmulțirea de numere binare a n biți are ca rezultat un produs cu o lungime de până la 2*n biți, rezultatul ALU (OUTPUT) va fi un număr binar reprezentat pe 16 biți (se vor insera biți de 0 sau 1 la început acolo unde este cazul - bitul de semn).

Pentru afișarea rezultatului, folosim afișorul Seven Segment Display asociat Basys3. Interfața folosește 7 leduri asociate fiecărei cifre; fiecare cifră este activată de un semnal de anod. Toate semnalele interfeței SSD (cele 7 semnale comune de catod și 4 semnale distincte de anod) sunt active pe 0. Semnalele de catod controlează ledurile care se aprind pe acele cifre care au semnalul de anod activ.

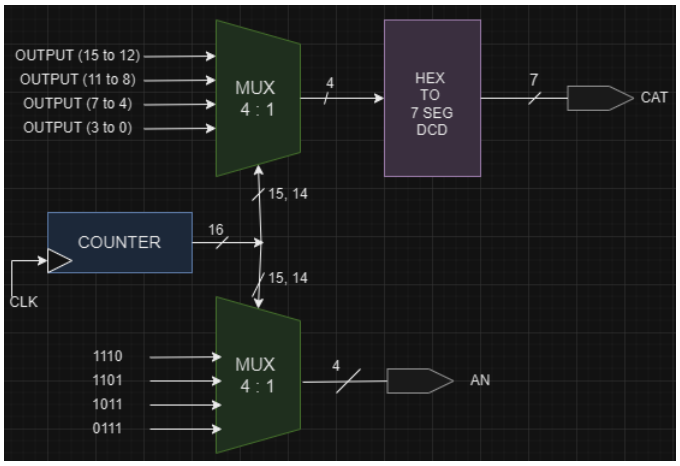


Figura 5: SSD

4 Manual de utilizare

Selecția operației va fi făcută de pe 3 switch-uri ale Basys3 (sw(2:0)) astfel:

- **000** - rezultatul operației logice INV (not) pe biți pentru primul operand
- **001** - rezultatul operației logice OR pe biți între cei doi operanzi
- **010** - rezultatul operației logice XOR pe biți între cei doi operanzi

- **011** - rezultatul operației logice AND pe biți între cei doi operanzi parțial
- **100** - rezultatul sumei dintre cei doi operanzi, cu inserarea bitului de semn pe primele 8 poziții
- **101** - rezultatul diferenței dintre cei doi operanzi, cu inserarea bitului de semn pe primele 8 poziții
- **110** - rezultatul înmulțirii celor doua numere (registrele A și Q concatenate)
- **111** - rezultatul împărțirii celor două numere (registrele quotient și remainder concatenate, transformate în funcție de semnele celor doi operanzi)

În entitatea ALU au fost definite 6 seturi de numere pentru testare; putem selecta pentru verificare unul din acestea prin 3 switch-uri (sw(15:13)).

Ledurile 15, respectiv 0 se vor aprinde în cazul în care flagul de overflow va fi activ în urma operației de adunare, respectiv scădere, marcând faptul că operația nu se poate efectua fără pierderi pe 8 biți și rezultatul nu va fi unul corect matematic.

5 Rezultate experimentale

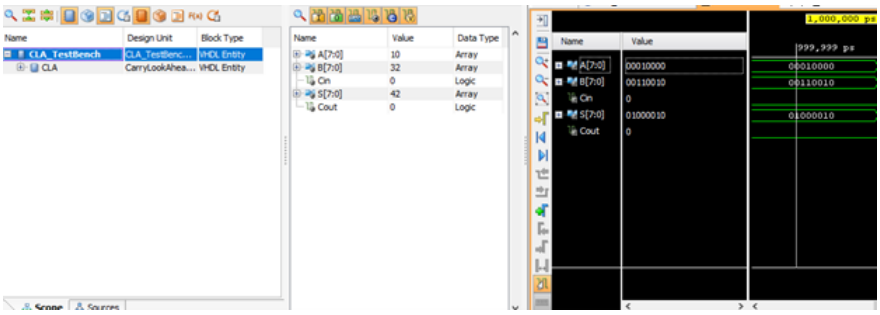


Figura 6: Adunare/Scădere - rezultatele simulării

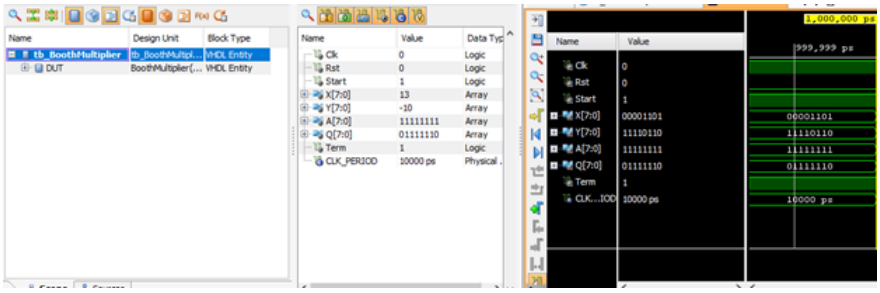


Figura 7: Înmulțire prin metoda Booth - rezultatele simulării

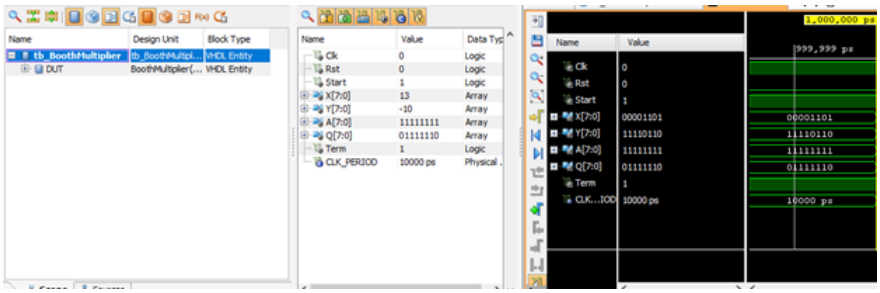


Figura 8: Împărțire cu refacerea restului parțial - rezultatele simulării

6 Bibliografie

1. “Structured Computer Organization”. Andrew S. Tanenbaum, p. 166 – 167.
2. “DE LA BIT LA PROCESOR. Introducere in Arhitectura Calculatoarelor”, F. Oniga, p. 34 – 36
3. Notite de curs, Structure Sistemelor de Calcul, G. Sebestyen, cursul 3
4. Notite de curs, Structura Sistemelor de Calcul, F. Lisman
5. “Arhitectura Calculatoarelor”, Baruch Zoltan Francisc, capitolul 6

7 Anexe

7.1 ANEXA 1 - Carry LookAhead Adder

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 23.11.2023 23:28:08
-- Design Name:
-- Module Name: CarryLookAheadAdder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CarryLookAheadAdder is
Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
       B : in STD_LOGIC_VECTOR (7 downto 0);
       Cin : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (7 downto 0);
       Cout : out STD_LOGIC
);
end CarryLookAheadAdder;

architecture Behavioral of CarryLookAheadAdder is

component PartialFullAdder
```

```
 Port ( input1 : in STD_LOGIC;
        input2 : in STD_LOGIC;
        Cin : in STD_LOGIC;
        Sum : out STD_LOGIC;
        P : out STD_LOGIC;
        G : out STD_LOGIC);
end component;

signal C: STD_LOGIC_VECTOR(7 downto 1);
signal P, G: STD_LOGIC_VECTOR(7 downto 0);

begin

    PFA0: PartialFullAdder port map(A(0), B(0), Cin, S(0), P(0), G(0));
    PFA1: PartialFullAdder port map(A(1), B(1), C(1), S(1), P(1), G(1));
    PFA2: PartialFullAdder port map(A(2), B(2), C(2), S(2), P(2), G(2));
    PFA3: PartialFullAdder port map(A(3), B(3), C(3), S(3), P(3), G(3));
    PFA4: PartialFullAdder port map(A(4), B(4), C(4), S(4), P(4), G(4));
    PFA5: PartialFullAdder port map(A(5), B(5), C(5), S(5), P(5), G(5));
    PFA6: PartialFullAdder port map(A(6), B(6), C(6), S(6), P(6), G(6));
    PFA7: PartialFullAdder port map(A(7), B(7), C(7), S(7), P(7), G(7));
    C(1) <= G(0) OR
        (P(0) AND Cin);
    C(2) <= G(1) OR
        (P(1) AND G(0)) OR
        (P(1) AND P(0) AND Cin);
    C(3) <= G(2) OR
        (P(2) AND G(1)) OR
        (P(2) AND P(1) AND G(0)) OR
        (P(2) AND P(1) AND P(0) AND Cin);

    C(4) <= G(3) OR
        (P(3) AND G(2)) OR
        (P(3) AND P(2) AND G(1)) OR
        (P(3) AND P(2) AND P(1) AND G(0)) OR
        (P(3) AND P(2) AND P(1) AND P(0) AND Cin);

    C(5) <= G(4) OR
        (P(4) AND G(3)) OR
        (P(4) AND P(3) AND G(2)) OR
        (P(4) AND P(3) AND P(2) AND G(1)) OR
        (P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR
        (P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND Cin);

    C(6) <= G(5) OR
        (P(5) AND G(4)) OR
        (P(5) AND P(4) AND G(3)) OR
        (P(5) AND P(4) AND P(3) AND G(2)) OR
```

```
        (P(5) AND P(4) AND P(3) AND P(2) AND G(1)) OR
        (P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR
        (P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND Cin);

    C(7) <= G(6) OR
        (P(6) AND G(5)) OR
        (P(6) AND P(5) AND G(4)) OR
        (P(6) AND P(5) AND P(4) AND G(3)) OR
        (P(6) AND P(5) AND P(4) AND P(3) AND G(2)) OR
        (P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND G(1)) OR
        (P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR
        (P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND Cin);

    Cout <= G(7) OR
        (P(7) AND G(6)) OR
        (P(7) AND P(6) AND G(5)) OR
        (P(7) AND P(6) AND P(5) AND G(4)) OR
        (P(7) AND P(6) AND P(5) AND P(4) AND G(3)) OR
        (P(7) AND P(6) AND P(5) AND P(4) AND P(3) AND G(2)) OR
        (P(7) AND P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND G(1)) OR
        (P(7) AND P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND G(0)) OR
        (P(7) AND P(6) AND P(5) AND P(4) AND P(3) AND P(2) AND P(1) AND P(0) AND Cin);

end Behavioral;

```

7.2 ANEXA 2 - Booth Multiplier

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 28.11.2023 14:43:03
-- Design Name:
-- Module Name: BoothMultiplier - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


```

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34
35  entity BoothMultiplier is
36      Port ( clk : in STD_LOGIC;
37             rst : in STD_LOGIC;
38             START : in STD_LOGIC;
39             X : in STD_LOGIC_VECTOR (7 downto 0);
40             Y : in STD_LOGIC_VECTOR (7 downto 0);
41             A : out STD_LOGIC_VECTOR (7 downto 0);
42             Q : out STD_LOGIC_VECTOR (7 downto 0);
43             STOP : out STD_LOGIC;
44             ProductReady : out STD_LOGIC);
45  end BoothMultiplier;
46
47  architecture Behavioral of BoothMultiplier is
48  signal QOQm1: std_logic_vector(1 downto 0);
49  signal LoadB: std_logic;
50  signal SubB: std_logic;
51  signal RstA: std_logic;
52  signal LoadA: std_logic;
53  signal ShrAQ: std_logic;
54  signal LoadQ: std_logic;
55  signal RstQm1: std_logic;
56  signal B: std_logic_vector(7 downto 0);
57  signal B_complemented: std_logic_vector(7 downto 0);
58  signal AdderResult: std_logic_vector(7 downto 0);
59  signal Cout: std_logic;
60  signal QQ: std_logic_vector(7 downto 0);
61  signal Q_final: std_logic_vector(7 downto 0);
62  signal Q_late: std_logic;
63  signal ProductReadyInternal: STD_LOGIC := '0';
64  signal Stop_internal: STD_LOGIC;
65  signal NewSetStarted: STD_LOGIC := '0';
66
67  component BoothUC
68      port(  clk : in STD_LOGIC;
69             rst : in STD_LOGIC;
```

```
70              START : in STD_LOGIC;
71              Q0Qm1 : in STD_LOGIC_VECTOR (1 downto 0);
72              LoadB : out STD_LOGIC;
73              SubB : out STD_LOGIC;
74              RstA : out STD_LOGIC;
75              LoadA : out STD_LOGIC;
76              ShrAQ : out STD_LOGIC;
77              LoadQ : out STD_LOGIC;
78              RstQm1 : out STD_LOGIC;
79              STOP : out STD_LOGIC);
80  end component;
81
82  component FDN
83      port(  Clk : in STD_LOGIC;
84             D : in STD_LOGIC_VECTOR (7 downto 0);
85             Rst : in STD_LOGIC;
86             CE : in STD_LOGIC;
87             Q : out STD_LOGIC_VECTOR (7 downto 0));
88  end component;
89
90  component TwosComplement
91      port( Enable : in STD_LOGIC;
92            Input : in STD_LOGIC_VECTOR (7 downto 0);
93            Output : out STD_LOGIC_VECTOR (7 downto 0));
94  end component;
95
96  component CarryLookAheadAdder
97      port(A: in STD_LOGIC_VECTOR (7 downto 0);
98           B : in STD_LOGIC_VECTOR (7 downto 0);
99           Cin : in STD_LOGIC;
100          S : out STD_LOGIC_VECTOR (7 downto 0);
101          Cout : out STD_LOGIC);
102 end component;
103
104 component RightShifter
105     port ( Clk : in STD_LOGIC;
106            D : in STD_LOGIC_VECTOR (7 downto 0);
107            SIGN : in STD_LOGIC;
108            Rst : in STD_LOGIC;
109            Load : in STD_LOGIC;
110            CE : in STD_LOGIC;
111            Q : out STD_LOGIC_VECTOR (7 downto 0));
112 end component;
113
114 component FD
115     port ( Clk : in STD_LOGIC;
116            Rst : in STD_LOGIC;
117            CE : in STD_LOGIC;
```

```
            D : in STD_LOGIC;
            Q : out STD_LOGIC);
end component;

begin
BOOTH_UC: BoothUC port map(clk, rst, START, QOQm1, LoadB, SubB, RstA,
        LoadA, ShrAQ, LoadQ, RstQm1, STOP_internal);
BOOTH_FDN: FDN port map(clk, X, rst, LoadB, B);
COMPLEMENT: TwosComplement port map(SubB, B, B_Complemented);
ADDER: CarryLookAheadAdder port map(QQ, B_complemented, SubB, AdderResult, Cout);
SHIFT_RIGHT_A: RightShifter port map(clk, AdderResult, QQ(7), RstA, LoadA, ShrAQ, QQ);
SHIFT_RIGHT_Q: RightShifter port map(clk, Y, QQ(0), rst, LoadQ, shrAQ, Q_final);
BOOTH_FD: FD port map(clk, RstQm1, shrAQ, Q_final(0), Q_late);

QOQm1(1) <= Q_final(0);
QOQm1(0) <= Q_late;
A <= QQ;
Q <= Q_final;

end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 28.11.2023 17:59:20
-- Design Name:
-- Module Name: BoothUC - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

```

```
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity BoothUC is
35      Port ( clk : in STD_LOGIC;
36             rst : in STD_LOGIC;
37             START : in STD_LOGIC;
38             Q0Qm1 : in STD_LOGIC_VECTOR (1 downto 0);
39             LoadB : out STD_LOGIC;
40             SubB : out STD_LOGIC;
41             RstA : out STD_LOGIC;
42             LoadA : out STD_LOGIC;
43             ShrAQ : out STD_LOGIC;
44             LoadQ : out STD_LOGIC;
45             RstQm1 : out STD_LOGIC;
46             STOP : out STD_LOGIC);
47  end BoothUC;
48
49  architecture Behavioral of BoothUC is
50   type state_type is (BEGIN_IT, INIT, VERIFY, ADD, SUBTRACT,
51                  SHIFT, DECREMENT_C, VERIFY_C, FINISH);
52   signal state: state_type;
53   signal C: natural := 0;
54  begin
55      process(state, clk, rst)
56      begin
57          if rst = '1' then
58              state <= BEGIN_IT;
59          elsif rising_edge(clk) then
60              case state is
61                  when BEGIN_IT => if START = '1' then
62                                       state <= INIT;
63                                   end if;
64                  when INIT => C <= 8;
65                               state <= VERIFY;
66                  when VERIFY => if Q0Qm1 = "01" then
67                                       state <= ADD;
68                                 elsif Q0Qm1 = "10" then
69                                        state <= SUBTRACT;
70                                 else
71                                        state <= SHIFT;
72                                 end if;
73                  when ADD => state <= SHIFT;
74                  when SUBTRACT => state <= SHIFT;
75                  when SHIFT => state <= DECREMENT_C;
76                  when DECREMENT_C => C <= C - 1;
```

```
                                    state <= VERIFY_C;
                when VERIFY_C => if C = 0 then
                                        state <= FINISH;
                                 else
                                        state <= VERIFY;
                                 end if;
                when FINISH => state <= FINISH;
            end case;
        end if;
end process;

process(state)
begin
    case state is
        when BEGIN_IT => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when INIT => LoadB <= '1'; SubB <= '0'; RstA <= '1'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '1'; RstQm1 <= '1'; STOP <= '0';
        when VERIFY => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when ADD => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '1';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when SUBTRACT => LoadB <= '0'; SubB <= '1'; RstA <= '0'; LoadA <= '1';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when SHIFT => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '1'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when DECREMENT_C => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when VERIFY_C => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '0';
        when FINISH => LoadB <= '0'; SubB <= '0'; RstA <= '0'; LoadA <= '0';
                ShrAQ <= '0'; LoadQ <= '0'; RstQm1 <= '0'; STOP <= '1';
    end case;
end process;

end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 28.11.2023 19:04:43
-- Design Name:
-- Module Name: FDN - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
```

```
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity FDN is
35      Port ( Clk : in STD_LOGIC;
36             D : in STD_LOGIC_VECTOR (7 downto 0);
37             Rst : in STD_LOGIC;
38             CE : in STD_LOGIC;
39             Q : out STD_LOGIC_VECTOR (7 downto 0));
40  end FDN;
41
42  architecture Behavioral of FDN is
43  signal QQ: std_logic_vector(7 downto 0) := "00000000";
44  begin
45
46  process(Clk, Rst, CE)
47  begin
48      if rising_edge(Clk) then
49          if (Rst = '1') then
50              QQ <= "00000000";
51          elsif (CE = '1') then
52              QQ <= D;
53          else
54              QQ <= QQ;
55          end if;
56       end if;
57  end process;
58  Q <= QQ;
59  end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 28.11.2023 19:18:54
-- Design Name:
-- Module Name: TwosComplement - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity TwosComplement is
    Port ( Enable : in STD_LOGIC;
           Input : in STD_LOGIC_VECTOR (7 downto 0);
           Output : out STD_LOGIC_VECTOR (7 downto 0));
end TwosComplement;

architecture Behavioral of TwosComplement is
begin

XOR_UNITS: for i in 0 to 7 generate
    Output(i) <= Enable XOR Input(i);
end generate;

end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 28.11.2023 19:37:31
-- Design Name:
-- Module Name: RightShifter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RightShifter is
    Port ( Clk : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR (7 downto 0);
           SIGN : in STD_LOGIC;
           Rst : in STD_LOGIC;
           Load : in STD_LOGIC;
           CE : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (7 downto 0));
end RightShifter;

architecture Behavioral of RightShifter is
    signal output: std_logic_vector(7 downto 0) := "00000000";
begin
    process(Clk, Rst, CE)
    begin
```

```
        if rising_edge(Clk) then
            if (Rst = '1') then
                output <= "00000000";
            else if (Load = '1') then
                output <= D;
            else if (CE = '1') then
                output <= SIGN & output(7 downto 1);
            else
                output <= output;
            end if;
            end if;
            end if;
        end if;
    end process;

Q <= output;
end Behavioral;
```

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

entity FD is
    Port ( Clk : in STD_LOGIC;
           Rst: in STD_LOGIC;
           CE : in STD_LOGIC;
           D : in STD_LOGIC;
           Q : out STD_LOGIC);
end FD;

architecture Behavioral of FD is
begin
 process(Clk)
   begin
   if(rising_edge(clk)) then
     if (Rst='1') then
       Q <= '0';
     elsif (CE='1') then
       Q <= D;
     end if;
   end if;
 end process;

end Behavioral;
```

7.3 ANEXA 3 - Restoring Division

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 30.11.2023 22:31:52
-- Design Name:
-- Module Name: RestoringDivision - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RestoringDivision is
    Port(
    rst : in STD_LOGIC;
    X : in STD_LOGIC_VECTOR(7 downto 0);
    Y : in STD_LOGIC_VECTOR(7 downto 0);
    A : out STD_LOGIC_VECTOR(7 downto 0);
    Q_out : out STD_LOGIC_VECTOR(7 downto 0);
    clk : in STD_LOGIC
    );
end RestoringDivision;

```

```
48  architecture Behavioral of RestoringDivision is
49      type state_type is (INIT, LEFT_SHIFT_PRIME, SUBTRACT_B,
50          LEFT_SHIFT, ADD_B, SPLIT_AQ, ASSIGN_Q0, SUBTRACT_C, VERIFY_C, FINISH);
51      signal state: state_type := INIT;
52      signal next_state: state_type := LEFT_SHIFT;
53      signal AQ: std_logic_vector(15 downto 0);
54      signal A_temp: std_logic_vector(7 downto 0);
55      signal Q_temp: std_logic_vector(7 downto 0);
56      signal C: natural := 0;
57      signal Cout_Sum: std_logic;
58      signal Cout_Diff: std_logic;
59      signal Sum: std_logic_vector(7 downto 0);
60      signal Diff: std_logic_vector(7 downto 0);
61      signal Y_Complemented: std_logic_vector(7 downto 0);
62      signal SubB: std_logic;
63
64      component CarryLookAheadAdder
65          port(A: in STD_LOGIC_VECTOR(7 downto 0);
66               B : in STD_LOGIC_VECTOR(7 downto 0);
67               Cin : in STD_LOGIC;
68               S : out STD_LOGIC_VECTOR(7 downto 0);
69               Cout : out STD_LOGIC
70      );
71      end component;
72
73      component TwosComplement
74          port( Enable : in STD_LOGIC;
75                Input : in STD_LOGIC_VECTOR (7 downto 0);
76                Output : out STD_LOGIC_VECTOR (7 downto 0));
77      end component;
78
79
80  begin
81  COMPLEMENT: TwosComplement port map('1', Y, Y_Complemented);
82  ADDER: CarryLookAheadAdder port map(
83   A => A_temp,
84   B => Y,
85   Cin => '0',
86   S => Sum,
87   Cout => Cout_Sum);
88  SUBTRACTOR: CarryLookAheadAdder port map(
89    A => A_temp,
90    B => Y_Complemented,
91    Cin => '1',
92    S => Diff,
93    Cout => Cout_Diff);
94
95
```

```
96  process(clk, rst, state)
97  begin
98      if (rst = '1') then
99          state <= INIT;
100     elsif rising_edge(clk) then
101         case state is
102             when INIT =>
103                 C <= 8;
104                 A_temp <= "00000000";
105                 AQ <= "00000000" & X;
106                 Q_temp <= X;
107                 state <= LEFT_SHIFT;
108
109             when LEFT_SHIFT =>
110                 AQ <= A_temp & Q_temp;
111                 state <= LEFT_SHIFT_PRIME;
112
113             when LEFT_SHIFT_PRIME =>
114                 AQ <= AQ(14 downto 0) & '0';
115                 state <= SPLIT_AQ;
116
117             when SPLIT_AQ =>
118                 A_temp <= AQ(15 downto 8);
119                 Q_temp <= AQ(7 downto 0);
120                 state <= SUBTRACT_B;
121
122             when SUBTRACT_B =>
123                 A_temp <= Diff;
124                 state <= ASSIGN_Q0;
125
126             when ASSIGN_Q0 =>
127                 if (A_temp(7) = '1') then
128                     Q_temp(0) <= '0';
129                     state <= ADD_B;
130                  else
131                     Q_temp(0) <= '1';
132                     state <= SUBTRACT_C;
133                  end if;
134
135             when ADD_B =>
136                 A_temp <= Sum;
137                 state <= SUBTRACT_C;
138
139             when SUBTRACT_C =>
140                 C <= C - 1;
141                 state <= VERIFY_C;
142
143             when VERIFY_C =>
```

```
144                 if C = 0 then
145                     state <= FINISH;
146                 else
147                     state <= LEFT_SHIFT;
148                 end if;
149 
150             when FINISH =>
151                 Q_out <= Q_temp;
152                 A <= A_temp;
153         end case;
154     end if;
155 end process;
156 
157 end Behavioral;
```