Universidad Autónoma De Querétaro
Facultad de Ingeniería

Autonomous University of Querétaro

Faculty of Engineering.

Career: Automation Engineer.

Subject: Systems with reconfigurable logic.

Student: Martínez Murillo Omar Yarif.
Student: Diego Joel Zúñiga Fragoso.
Student: Daniela del Carmen Manríquez Navarro.
Student: Joselyn Gallegos Abreo.

Practice 4: SDLR

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Introduction

The concepts of half-adder and full-adder are explored, which allow for the addition of two binary numbers, including carry between different bit positions. The practice covers both the implementation of cascaded adders and the use of a carry look-ahead (CLA) technique to reduce the delay in multi-bit additions. For the subtractor, the foundation of addition was used, along with the two's complement to perform binary subtraction, which is essential for obtaining negative results in binary systems. The comparator is a circuit that compares two binary words and determines whether one is greater than, less than, or equal to the other. This practice includes the design of single-bit and multi-bit comparators, using cascading techniques to extend the size of the words. The multiplier circuit is also combinational and allows for the multiplication of two binary numbers. Various variables related to the multiplier are explored.

Methodology

Step 1: Comparator

- The process starts with the comparator. First, the configuration is set up in Aldec-Active.
- The entity is declared, followed by the architecture, where a half comparator is defined using the formulas discussed during the session.
- Then, the full comparator is declared within the architecture, along with its respective formulas.
- A test bench is developed in Aldec-Active to verify the comparison results.
- Once this is completed, the configuration is transferred to Quartus, where the input and output pins are assigned, and the program is loaded onto the FPGA. The program displays three results on the FPGA's displays.

Step 2: Adder

- Next, the adder is configured again in Aldec-Active, where the entity and architecture are defined.

- The architecture includes the adder formula presented in the session.
- A test bench is developed, and once verified, the configuration is transferred to Quartus. The input and output pins are assigned, and the program is loaded.
- The same procedure is repeated for the adder using the CLA method.

Step 3: Subtractor

- The subtractor is developed based on the adder, with the addition of the two's complement.
- This is then transferred to Quartus and programmed.

Step 4: Multiplier

- Lastly, the multiplier is developed by repeating the steps and using the corresponding formula.

Results

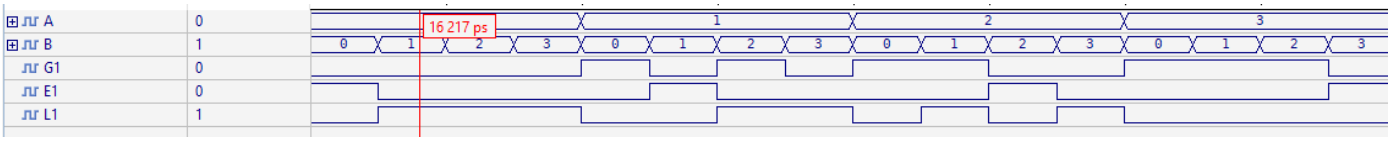
Base algorithm Comparator

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity compa is
4      port(
5      A: in std_logic_vector(1 downto 0);
6      B: in std_logic_vector(1 downto 0);
7
8      G1: out std_logic;
9      E1: out std_logic;
10     L1: out std_logic
11     );
12 end compa;
13
14 Architecture c1 of compa is
15 signal  G0: std_logic;
16 signal  E0: std_logic;
17 signal  L0: std_logic;
18 begin
19     -- medio comparador
20     G0<= ((A(0) AND (NOT B(0))));
21     E0<= (NOT(A(0) xor B(0)));
22     L0<= ((NOT A(0))AND B(0));
23     --Comparador completo
24     G1<= G0 or ((A(1) AND (NOT B(1))));
25     E1<= E0 and (NOT(A(1) xor B(1)));
26     L1<= L0 or ((NOT A(1))AND B(1));
27 end c1;
```

Test Bench

```
component Compa
    port(
    A: in  std_logic_vector(1 downto 0);
    B: in  std_logic_vector(1 downto 0);

    G1: out std_logic;
    E1: out std_logic;
    L1: out std_logic
    );
end component;

signal A,B:std_logic_vector(1 downto 0);  --Entradas
signal G1,E1,L1:std_logic; --Salidas

begin
    inicio: Compa port map (A=>A,B=>B,G1=>G1,E1=>E1,L1=>L1);
    Parasimu: process
    begin
    A<="00"; B<="00";  wait for 10ns;
    A<="00"; B<="01";  wait for 10ns;
    A<="00"; B<="10";  wait for 10ns;
    A<="00"; B<="11";  wait for 10ns;
    A<="01"; B<="00";  wait for 10ns;
    A<="01"; B<="01";  wait for 10ns;
    A<="01"; B<="10";  wait for 10ns;
    A<="01"; B<="11";  wait for 10ns;
    A<="10"; B<="00";  wait for 10ns;
    A<="10"; B<="01";  wait for 10ns;
```

Simulation



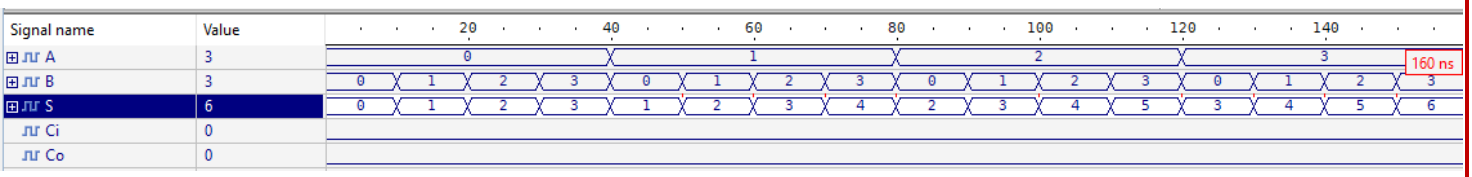
Base algorithm of Adder

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity sumador is
    generic(
        n: integer:=3

    );
    port(
        A, B: in std_logic_vector(n-1 downto 0);
        Ci: in std_logic;
        S: out std_logic_vector(n-1 downto 0);
        Co: out std_logic
    );
end sumador;

Architecture suma of sumador is
signal  carry: std_logic_vector(n downto 0);--por si se desborda
begin
    process (A,B, Ci, Carry)
    begin
        carry(0)<=Ci;
        for i in 0 to n-1 loop
            S(i)<= A(i) xor B(i) xor carry(i);
            carry(i+1)<= (A(i) and B(i)) or (A(i) and Carry(i)) or ( carry(i) and B(i));
        end loop;
        Co<=Carry(n);
    end process;
end suma;
```

Test Bench

```vhdl
    end component;

    signal A, B, S: std_logic_vector(2 downto 0);  -- Entradas
    signal Ci, Co: std_logic;  -- Salidas

begin
    inicio: sumador
        generic map (n => 3)  -- Se establece el genérico 'n' en 2
        port map (A => A, B => B, Ci => Ci, Co => Co, S => S);

    Parasimu: process
    begin
        Ci <= '0';  -- Inicializar 'Ci' al valor deseado
        A <= "000"; B <= "000"; wait for 10 ns;
        A <= "000"; B <= "001"; wait for 10 ns;
        A <= "000"; B <= "010"; wait for 10 ns;
        A <= "000"; B <= "011"; wait for 10 ns;
        A <= "001"; B <= "000"; wait for 10 ns;
        A <= "001"; B <= "001"; wait for 10 ns;
        A <= "001"; B <= "010"; wait for 10 ns;
        A <= "001"; B <= "011"; wait for 10 ns;
        A <= "010"; B <= "000"; wait for 10 ns;
        A <= "010"; B <= "001"; wait for 10 ns;
        A <= "010"; B <= "010"; wait for 10 ns;
        A <= "010"; B <= "011"; wait for 10 ns;
        A <= "011"; B <= "000"; wait for 10 ns;
```

Simulation



Base algorithm of Adder CLA

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity CLA_Sumador is
    generic(
        n : integer := 2  -- Cambia 'n' según el número de bits que desees sumar
    );
    port(
        A, B: in std_logic_vector(n-1 downto 0);
        Ci: in std_logic;
        S: out std_logic_vector(n-1 downto 0);
        Co: out std_logic
    );
end CLA_Sumador;

architecture Suma_CLA of CLA_Sumador is
    signal G, P: std_logic_vector(n-1 downto 0);  -- Señales de generación y propagación de acarreo
    signal C: std_logic_vector(n downto 0);       -- Señales de acarreo intermedio (C(0) es Ci)
begin
    -- Calcular señales de generación (G) y propagación (P)
    G <= A and B;  -- G(i) = A(i) AND B(i)
    P <= A or B;   -- P(i) = A(i) OR B(i)

    -- Acarreo inicial
    C(0) <= Ci;

    -- Calcular los acarreos utilizando las señales de generación y propagación
    C(1) <= G(0) or (P(0) and C(0));
    C(2) <= G(1) or (P(1) and C(1));
    C(3) <= G(2) or (P(2) and C(2));
    C(4) <= G(3) or (P(3) and C(3));

    -- Calcular la suma
    S <= A xor B xor C(n-1 downto 0);

    -- Acarreo de salida
    Co <= C(n);

end Suma_CLA;
```

Base algorithm of Subtractor

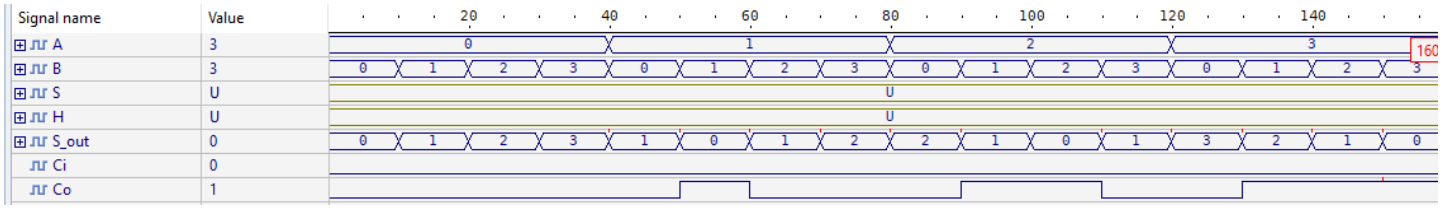
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity restador is
    generic(
        n: integer := 3  -- Tamaño del vector
    );
    port(
        A, B: in std_logic_vector(n-1 downto 0);
        Ci: in std_logic;
        S_out: out std_logic_vector(n-1 downto 0);
        --S: out std_logic_vector(n-1 downto 0);
        Co: out std_logic

    );
end restador;

architecture resta of restador is
    signal carry: std_logic_vector(n downto 0);  -- Señal de acarreo
    signal B_neg: std_logic_vector(n-1 downto 0);  -- Señal para almacenar el complemento de B
    signal S:  std_logic_vector(n-1 downto 0);
    signal H:  std_logic_vector(n-1 downto 0);
    --signal S_out:  std_logic_vector(n-1 downto 0);  -- Salida después de verificar S(0)
    --signal Ci: std_logic;
begin
    -- Proceso para realizar la resta
    process (A, B, Ci,carry,B_neg,H,S)
    begin
        -- Generar el complemento a 2 de B
        B_neg <= std_logic_vector(unsigned(not B) + 1);
        H<=B_neg;
        carry(0) <=Ci;

        -- Realizar la suma de A y el complemento a 2 de B (es equivalente a A - B)
        for i in 0 to n-1 loop
            S(i) <= A(i) xor B_neg(i) xor carry(i);
            carry(i+1) <= (A(i) and B_neg(i)) or (A(i) and carry(i)) or (B_neg(i) and carry(i));
        end loop;
        Co <= carry(n);
-- Si S(0) es 1, aplicar complemento a 2 a S
       if S(2) = '1' then
            S_out <= std_logic_vector(unsigned(not S) + 1);
        else
            S_out <= S;  -- Si no, simplemente pasa el valor de S
        end if;


    end process;
end resta;
```

Test Bench

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity REST_TB is
end REST_TB;

architecture TB of REST_TB is
    component restador
        generic(
            n: integer := 3  -- Tamaño del vector para A y B
        );
        port(
            A, B: in std_logic_vector(n-1 downto 0);
            Ci: in std_logic;
            --S: out std_logic_vector(n-1 downto 0);
            --H: out std_logic_vector(n-1 downto 0);
            S_out: out std_logic_vector(n-1 downto 0);
            Co: out std_logic


        );
    end component;

    signal A, B, S,H,S_out: std_logic_vector(2 downto 0);  -- Entradas
    signal Ci, Co: std_logic;  -- Señales para acarreo
```

```vhdl
begin
    -- Instancia del restador
    inicio: restador
        generic map (n => 3)
        port map (A => A, B => B, Ci => Ci, Co => Co,S_out=>S_out);

    -- Proceso de simulación para aplicar las señales de prueba
    Parasimu: process
    begin
         Ci <= '0';  -- Inicializar 'Ci' al valor deseado
        A <= "000"; B <= "000"; wait for 10 ns;
        A <= "000"; B <= "001"; wait for 10 ns;
        A <= "000"; B <= "010"; wait for 10 ns;
        A <= "000"; B <= "011"; wait for 10 ns;
        A <= "001"; B <= "000"; wait for 10 ns;
        A <= "001"; B <= "001"; wait for 10 ns;
        A <= "001"; B <= "010"; wait for 10 ns;
        A <= "001"; B <= "011"; wait for 10 ns;
        A <= "010"; B <= "000"; wait for 10 ns;
        A <= "010"; B <= "001"; wait for 10 ns;
        A <= "010"; B <= "010"; wait for 10 ns;
        A <= "010"; B <= "011"; wait for 10 ns;
        A <= "011"; B <= "000"; wait for 10 ns;
        A <= "011"; B <= "001"; wait for 10 ns;
        A <= "011"; B <= "010"; wait for 10 ns;
        A <= "011"; B <= "011"; wait for 10 ns;
```

Simulation



Codes in Quartus

Comparator

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Comparador is
    port(
    clock:in std_logic;
    A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1 downto 0);
    o_segmentos: out std_logic_vector(7 downto 0);      -- Salida p
    o_comunes: out std_logic_vector(3 downto 0)          -- Salida
    );
end Comparador;

architecture Multiplexado of Comparador is

     -- Las señales son variables que no tienen un pin asignado
    signal count        : integer range 0 to 10000;
    signal comun_index  : integer range 0 to 4;
```

```
    constant A_Letter   : std_logic_vector(7 downto 0) := "10001000";
    constant B_Letter   : std_logic_vector(7 downto 0) := "10000011";
    constant C_Letter   : std_logic_vector(7 downto 0) := "11000110";
    constant D_Letter   : std_logic_vector(7 downto 0) := "10100001";
    constant E_Letter   : std_logic_vector(7 downto 0) := "10000110";
    constant F_Letter   : std_logic_vector(7 downto 0) := "10001110";
    constant G_Letter   : std_logic_vector(7 downto 0) := "10000010";
    constant H_Letter   : std_logic_vector(7 downto 0) := "10001001";
    constant I_Letter   : std_logic_vector(7 downto 0) := "11111001";
    constant J_Letter   : std_logic_vector(7 downto 0) := "11100001";
    constant K_Letter   : std_logic_vector(7 downto 0) := "10000101";
    constant L_Letter   : std_logic_vector(7 downto 0) := "11000111";
    constant M_Letter   : std_logic_vector(7 downto 0) := "10110000";
    constant N_Letter   : std_logic_vector(7 downto 0) := "11001000";
    constant N2_Letter  : std_logic_vector(7 downto 0) := "10101010";
    constant O_Letter   : std_logic_vector(7 downto 0) := "11000000";
    constant P_Letter   : std_logic_vector(7 downto 0) := "10001100";
    constant Q_Letter   : std_logic_vector(7 downto 0) := "10011000";
    constant R_Letter   : std_logic_vector(7 downto 0) := "11001110";
    constant S_Letter   : std_logic_vector(7 downto 0) := "10010010";
    constant T_Letter   : std_logic_vector(7 downto 0) := "10000111";
    constant U_Letter   : std_logic_vector(7 downto 0) := "11000001";
    constant W_Letter   : std_logic_vector(7 downto 0) := "10000110";
    constant Z_Letter   : std_logic_vector(7 downto 0) := "10100100";
```

```
46      signal letter1       : std_logic_vector(7 downto 0):=I_Letter;
47      signal letter2       : std_logic_vector(7 downto 0):=G_Letter;
48      signal letter3       : std_logic_vector(7 downto 0):=U_Letter;
49      signal letter4       : std_logic_vector(7 downto 0):=A_Letter;
50      --Operaciones de señales
51      signal  G0: std_logic;
52      signal  E0: std_logic;
53      signal  L0: std_logic;
54      signal G1:  std_logic;
55      signal E1:  std_logic;
56      signal L1:  std_logic;
57  begin
58  -- Reloj para delay de multiplexado
59      delay_clock : process (clock)
60      begin
61          if rising_edge(clock) then
62              if count < 10000 then
63                  count <= count + 1;
64              else
65                  if comun_index < 4 then
66                      comun_index <= comun_index + 1;
67                  else
68                      comun_index <= 0;
```

```
71                  count <= 0;
72              end if;
73          end if;
74      end process;
75
76       -- Impresion de display individual
77      display_print : process (comun_index)
78      begin
79          case comun_index is
80              when 0 =>
81                  o_comunes   <= "0111";
82                  o_segmentos <= letter1;
83              when 1 =>
84                  o_comunes <= "1011";
85                  o_segmentos <= letter2;
86              when 2 =>
87                  o_comunes <= "1101";
88                  o_segmentos <= letter3;
89              when 3 =>
90                  o_comunes <= "1110";
91                  o_segmentos <= letter4;
92              when others =>
93                  o_comunes <= "1111";
94          end case;
95      end process;
```

```
97      process (A,B)
98      begin
99      -- medio comparador
100        G0<= ((A(0) AND (NOT B(0))));
101        E0<= (NOT(A(0) xor B(0)));
102        L0<= ((NOT A(0))AND B(0));
103        --Comparador completo
104        G1<= G0 or ((A(1) AND (NOT B(1))));
105        E1<= E0 and (NOT(A(1) xor B(1)));
106        L1<= L0 or ((NOT A(1))AND B(1));
107        ---resultado
108         if G1 = '1' then
109              letter1 <= S_Letter;
110              letter2 <= U_Letter;
111              letter3 <= P_Letter;
112              letter4 <= E_Letter;
113          elsif E1 = '1' then
114              letter1 <= I_Letter;
115              letter2 <= G_Letter;
116              letter3 <= U_Letter;
117              letter4 <= A_Letter;
118          elsif L1 = '1' then
119              letter1 <= I_Letter;
120              letter2 <= N_Letter;
121              letter3 <= F_Letter;
122              letter4 <= E_Letter;
```

Adder

```
1   library  IEEE;
2   use   IEEE.std_logic_1164.all;
3
4   entity sumador is
5      generic(
6         n: integer:=3
7
8      );
9      port(
10        A, B: in std_logic_vector(n-1 downto 0);
11        display     : out    std_logic_vector(7 downto 0);     -- S
12        common      : out    std_logic_vector(3 downto 0)      -- S
13
14     );
15  end sumador;
16
17  Architecture suma of sumador is
18  signal carry: std_logic_vector(n downto 0);--por si se desborda
19  signal Ci: std_logic := '0';  -- Inicializar `Ci` a '0'
20  signal S:  std_logic_vector(n-1 downto 0);
21  signal Co: std_logic;
```

```
22  begin
23
24      process (A,B, Ci, Carry)
25      begin
26         carry(0)<=Ci;
27         for i in 0 to n-1 loop
28            S(i)<= A(i) xor B(i) xor carry(i);
29            carry(i+1)<= (A(i) and B(i)) or (A(i) and Carry(i)) or ( carry(i) and B(i));
30         end loop;
31         Co<=carry(n);
32      end process;
33
34      common <= "1110";
35         with S select
36      display <="11000000"when"000",--0
37                "11111001"when"100",--1
38                "10100100"when"010",--2
39                "10110000"when"110",--3
40                "10011001"when"001",--4
41                "10010010"when"101",--5
42                "10000010"when"011",--6
43                "11111111"when others;
44  end suma;
```

Subtractor

```
 1  library  IEEE;
 2  use   IEEE.std_logic_1164.all;
 3
 4  entity Restador_1bit is
 5      port
 6      (
 7          A    : in std_logic;
 8          B    : in std_logic;
 9          Ci   : in std_logic;       -- Acarreo de entrada
10          Co   : out std_logic;      -- Acarreo de salida
11          S    : out std_logic    -- Resultado
12      );
13  end entity;
14
15  architecture Restador_1bit of Restador_1bit is
16  begin
17
18      S   <= A xor B xor Ci;
19      Co  <= ((not A) and B) or ((not A) and Ci) or (B and Ci);
20
21  end Restador_1bit;
```

```
1   library  IEEE;
2   use   IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity Restador_nbits is
6      generic (
7         n: integer := 2      -- Especificamos bits del sumado
8      );
9      port
10     (
11        -- Entrada y salida de bits
12        A, B     : in std_logic_vector(n-1 downto 0);       --
13          neg_LED     : out std_logic;
14        LEDS     : out std_logic_vector(n-1 downto 0);      --
15
16        -- Displays de 7 seg
17         clock       : in    std_logic;
18        display     : out    std_logic_vector(7 downto 0);
19        common      : out    std_logic_vector(3 downto 0)
20     );
21  end Restador_nbits;
```

```
23  Architecture Restador_nbits of Restador_nbits is
24
25     -- Para impresion de numeros en display
26     type num_array is array (0 to 9) of std_logic_vector(7 downto 0);
27      constant numbers : num_array :=
28      (
29          "11000000",  -- 0
30          "11111001",  -- 1
31          "10100100",  -- 2
32          "10110000",  -- 3
33          "10011001",  -- 4
34          "10010010",  -- 5
35          "10000010",  -- 6
36          "11111000",  -- 7
37          "10000000",  -- 8
38          "10011000"   -- 9
39      );
40
```

```
41      -- Componente con sumador de 1 bit completo
42      component Restador_1bit is
43          port
44          (
45              A  : in std_logic;
46              B  : in std_logic;
47              Ci    : in std_logic;       -- Acarreo de entrada
48              Co    : out std_logic;      -- Acarreo de salida
49              S  : out std_logic   -- Resultado
50          );
51      end component;
52
53      -- Señales para multiplexado de display
54       signal count        : integer range 0 to 10000;
55       signal comun_index  : integer range 0 to 4;
56
57       -- Señales para sumador de n bits
58      signal S : std_logic_vector(n-1 downto 0);       -- Salida de suma de n bits
59      signal C : std_logic;                        -- Acarreo de primer bit
60       signal neg  : std_logic;                             -- Booleano para saber si es n
61  begin
62
63      -- Generamos sumas completas de cada bit, y los ponemos en cascada con el acarreo d
64
65      Res0: Restador_1bit port map(A(0), B(0), '0', C, S(0));     -- Suma del bit 0
```

```
66      Res1: Restador_1bit port map(A(1), B(1), C, neg, S(1));
67
68       neg_LED <= not neg;
69
70       -- Reloj para delay de multiplexado
71       delay_clock : process (clock)
72       begin
73           if rising_edge(clock) then
74               if count < 10000 then
75                   count <= count + 1;
76               else
77                   if comun_index < 2 then
78                       comun_index <= comun_index + 1;
79                   else
80                       comun_index <= 0;
81                   end if;
82                   count <= 0;
83               end if;
84           end if;
85       end process;
86
87       -- Impresion de display individual
88       display_print : process (comun_index)
89       begin
```

```
89      begin
90          case comun_index is
91              when 1 =>
92                  if neg = '1' then              -- Si es negativo imprimi
93                      common <= "1101";
94                      display <= "10111111";
95                  end if;
96              when 2 =>
97                  common <= "1110";
98                  if neg = '1' then              -- Si es negativo aplicam
99                     display <= numbers(to_integer(unsigned(not S) + 1));
100                    LEDS <= not std_logic_vector(unsigned(not S) + 1);
101                else
102                    display <=   numbers(to_integer(unsigned(S)));
103                    LEDS <= not S;
104                end if;
105             when others =>
106                 common <= "1111";
107         end case;
108     end process;
109
110 end Restador_nbits;
111
```

Multiplier Unsigned

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.std_logic_arith.all;
4   use IEEE.std_logic_unsigned.all;
5
6   entity Mult_MUU is
7       generic
8       (
9           n: integer := 2
10      );
11      port
12      (
13          A, B    : in std_logic_vector(n-1 downto 0);
14          output  : out std_logic_vector(2*n-1 downto 0)
15      );
16  end Mult_MUU;
17
18  architecture Aritmetica of Mult_MUU is
19  begin
20      process(A, B)
21      variable MUU : unsigned(2*n-1 downto 0);
22      begin
23          MUU     :=  unsigned(A) * unsigned(B);
24          output  <=  not std_logic_vector(MUU); -- LEDS
25      end process;
26  end Aritmetica;
```

Multiplier Signed

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity Mult_MSS is
7      generic
8      (
9          n: integer := 2
10     );
11     port
12     (
13         A, B    : in std_logic_vector(n-1 downto 0);
14         output  : out std_logic_vector(2*n-1 downto 0)
15     );
16 end Mult_MSS;
17
18 architecture Aritmetica of Mult_MSS is
19 begin
20     process(A, B)
21     variable MSS :   signed(2*n-1 downto 0);
22     begin
23         MSS     :=  signed(A) * signed(B);
24         output  <=  not std_logic_vector(MSS); -- LEDS
25     end process;
26 end Aritmetica;
```

Conclusion

Each of these circuits was designed and simulated, demonstrating the importance of optimization techniques such as carry look-ahead (CLA) to improve performance in multi-bit operations.

The implementation on the FPGA allowed real-time observation of the behavior of these circuits, clearly displaying results on the device's displays. This demonstrates their practical applicability in reconfigurable hardware systems. Comparators can be used in control and classification systems, adders and subtractors in complex arithmetic operations, and multipliers in digital signal processing and advanced mathematical algorithms. FPGAs provide a flexible and efficient platform for these applications, allowing designs to be adjusted according to system needs.