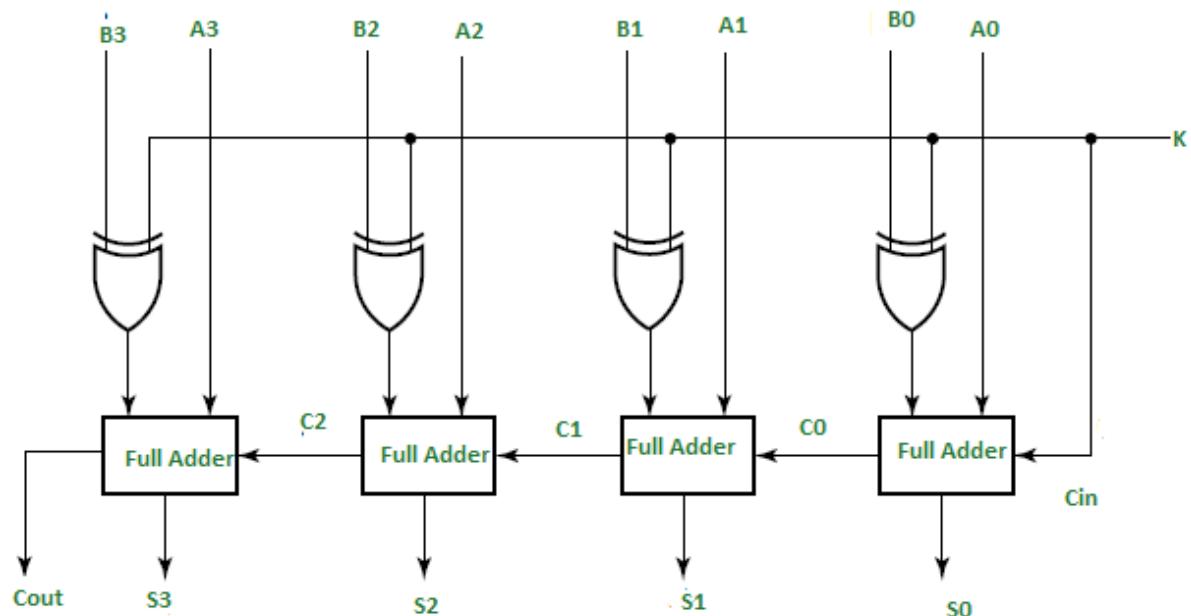


**Instituto Tecnológico y De Estudios
Superiores de Monterrey**
Laboratorio - Arquitectura de Computadoras
TE-2031.1

Lab 08 - Adders, Substractors and Multipliers



Entrega: 11 / 11 / 2022

Equipo 07:

Héctor Javier Pequeño Chairez A01246364

Gabriela Jazmín Álvarez Espinoza A00825719

Introducción

En la siguiente práctica se desarrollarán proyectos para ejecutar sumadores, acumuladores y una unidad de multiplicación. Para esto, deberán recordar conceptos como sumadores y el uso de los estatutos como “process()” el cual nos ayudará a detectar cambios en variables de interés para realizar la acumulación de valores. Además del uso de las simulaciones y el FPGA para confirmar que los diseños funcionen correctamente.

Desarrollo

Parte 1

Iniciaremos la definición de cada uno de los componentes que se encuentran en el siguiente diagrama.

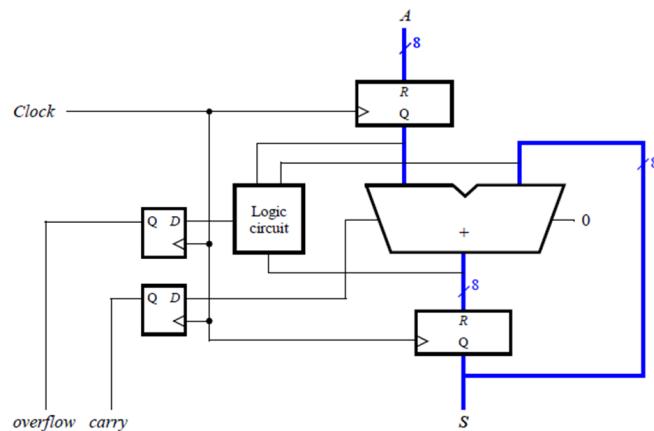


Diagrama a ejecutar.

Para esto iniciaremos definiendo el sumador de 8 bits. Pero para esto decidimos realizar un sumador de 1 bit con 1 bit, esto con el motivo de poder usar varios módulos que funcionen con un respectivo Carry In y Carry out, que al ser concatenados formarán nuestro sumador de 8 bits.

Sumador de 1 bit con 1 bit.

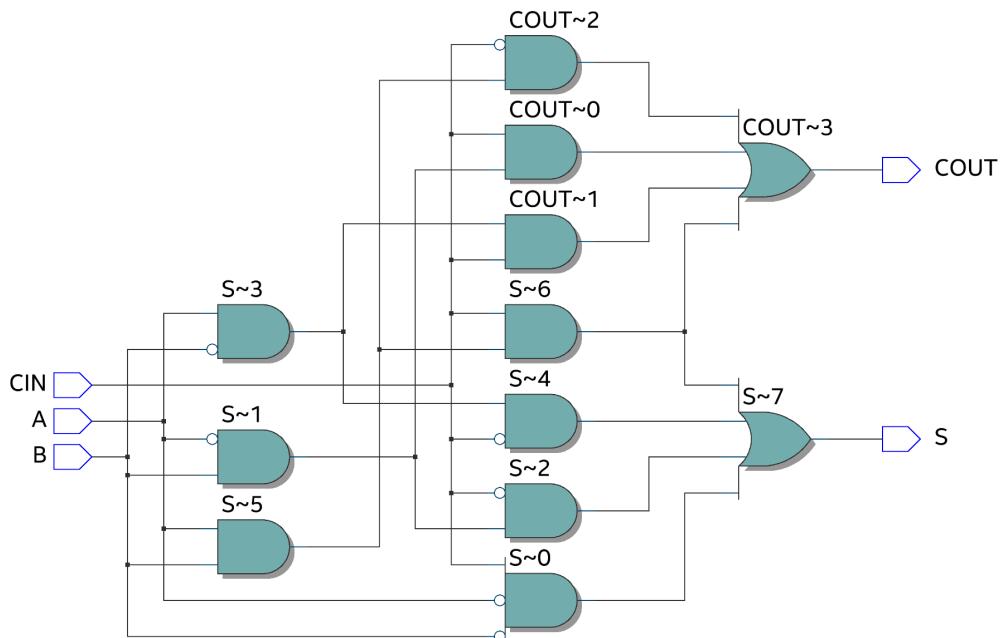


Diagrama generado.

Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

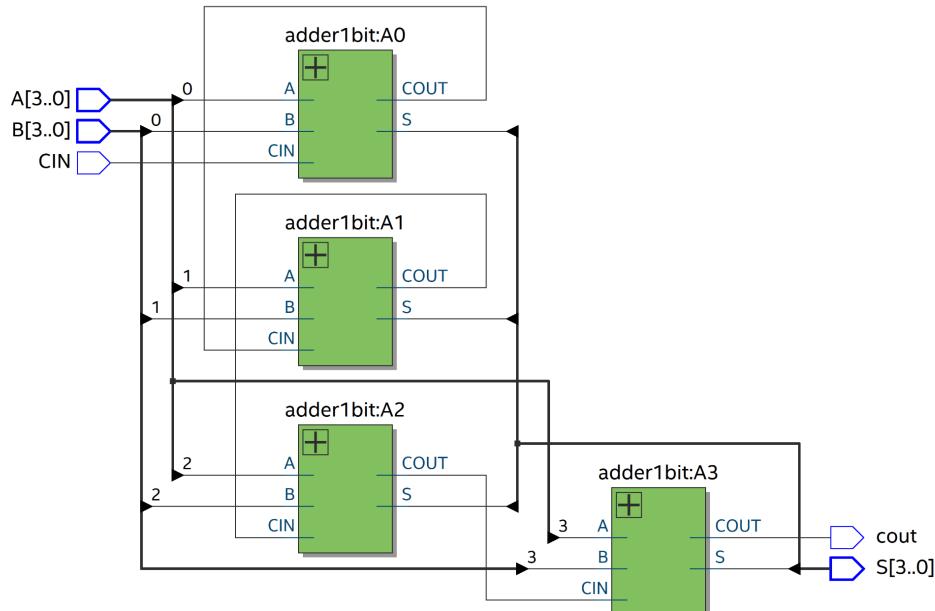
-- Sumador de 1 bit

entity adder1bit is
    port( A : in STD_LOGIC;          -- Input A de 1 bit
          B : in STD_LOGIC;          -- Input B de 1 bit
          CIN : in STD_LOGIC;        -- Carry de entrada
          S : out STD_LOGIC;         -- S representa la salida de la suma
          COUT: out STD_LOGIC);     -- Carry Out nos indica si existe el carry
end adder1bit;

--
architecture Behavioral of adder1bit is
begin
    -- Calcula la suma
    S <= ((not A) and (not B) and CIN) or
        ((not A) and B and (not CIN)) or
        (A and (not B) and (not CIN)) or
        (A and B and CIN);

    -- Calcula el carry en la suma
    COUT <= ((not A) and B and CIN) or
        (A and (not B) and CIN) or
        (A and B and (not CIN)) or
        (A and B and CIN);
end Behavioral;
```

Sumador de 4 bits



Circuito Generado.

Código:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

-- Adder de 4 bits, (4 bits con 4 bits)

entity adder4bits is
    port( A      : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada valor A 4 bits
          B      : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada valor B 4 bits
          CIN   : in STD_LOGIC;                          -- Entrada de CIN
          S     : out STD_LOGIC_VECTOR(3 downto 0); -- Salida valor A + B, 4 bits
          COUT: out STD_LOGIC );                      -- Salida Carry Out
end adder4bits;

-- Componente creado que suma 1 bit con 1 bit
architecture Behavioral of adder4bits is
    component adder1bit
        port( A      : in STD_LOGIC;           -- Input A de 1 bit
              B      : in STD_LOGIC;           -- Input B de 1 bit
              CIN   : in STD_LOGIC;          -- Carry de entrada
              S     : out STD_LOGIC;         -- S representa la salida de la suma
              COUT: out STD_LOGIC);        -- Carry Out nos indica si existe el carry
    end component;

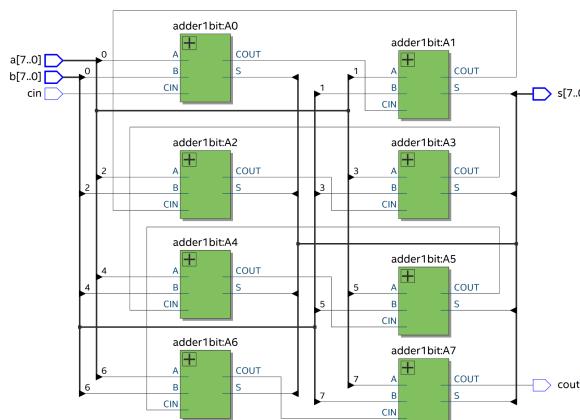
    signal c_aux1,c_aux2,c_aux3 : STD_LOGIC; -- Señales para transportar el carry
begin
    A0: adder1bit PORT MAP(A(0), B(0), CIN , S(0), c_aux1); -- Sumador 0
    A1: adder1bit PORT MAP(A(1), B(1), c_aux1, S(1), c_aux2); -- Sumador 1
    A2: adder1bit PORT MAP(A(2), B(2), c_aux2, S(2), c_aux3); -- Sumador 2
    A3: adder1bit PORT MAP(A(3), B(3), c_aux3, S(3), COUT); -- Sumador 3

end Behavioral;

```

Sumador 8 bits

En este caso pudimos haber utilizado 2 sumadores de 4 bits, pero decidimos expandir nuestro diseño del sumador de un bit 8 veces, esto nos deja el siguiente circuito con su respectivo código.



Sumador 8 bits, con salida de carry.

Código

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

-- Descripcion componente de un sumador de 8 bits
entity adder8bits is
    port( a    : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal A
          b    : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal B
          cin : in STD_LOGIC;                      -- Salida de la suma
          s   : out STD_LOGIC_VECTOR(7 downto 0); -- Salida del carry
          cout: out STD_LOGIC );
end adder8bits;
-- Fin descripcion del sumador

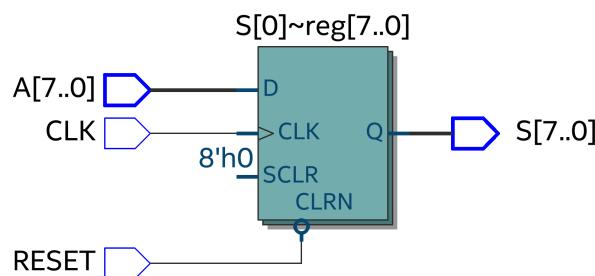
-- Inicia arquitectura
architecture behavioral of adder8bits is
    component adder1bit
        port( A    : in STD_LOGIC; -- Input A de 1 bit
              B    : in STD_LOGIC; -- Input B de 1 bit
              CIN : in STD_LOGIC; -- Carry de entrada
              S   : out STD_LOGIC; -- S representa la salida de la suma
              COUT: out STD_LOGIC); -- Carry Out nos indica si existe el carry
    end component;

    signal c_aux : STD_LOGIC_VECTOR(7 downto 1); -- Senal auxiliar para el carry
begin
    A0: adder1bit PORT MAP(a(0), b(0), cin    , s(0), c_aux(1)); -- Adder 0
    A1: adder1bit PORT MAP(a(1), b(1), c_aux(1), s(1), c_aux(2)); -- Adder 1
    A2: adder1bit PORT MAP(a(2), b(2), c_aux(2), s(2), c_aux(3)); -- Adder 2
    A3: adder1bit PORT MAP(a(3), b(3), c_aux(3), s(3), c_aux(4)); -- Adder 3
    A4: adder1bit PORT MAP(a(4), b(4), c_aux(4), s(4), c_aux(5)); -- Adder 4
    A5: adder1bit PORT MAP(a(5), b(5), c_aux(5), s(5), c_aux(6)); -- Adder 5
    A6: adder1bit PORT MAP(a(6), b(6), c_aux(6), s(6), c_aux(7)); -- Adder 6
    A7: adder1bit PORT MAP(a(7), b(7), c_aux(7), s(7), cout);      -- Adder 7

end behavioral;
```

Lógica

Este código nos ayudará a tener una lógica dentro de nuestro circuito, para tener en cuenta los valores de asignación y reset.



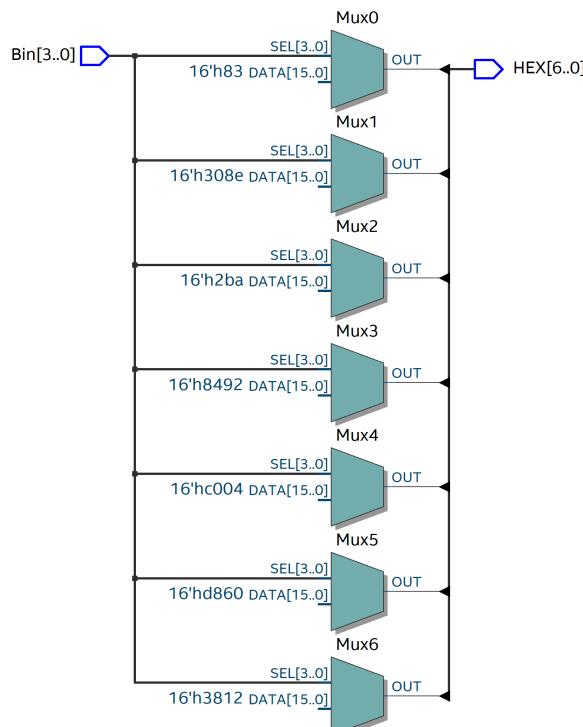
Circuito generado.

Código

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
-- Logical, se encarga de manejar la logica de nuestro
-- componente segun las entradas del RESET y el CLK
entity logical is
    port( A      : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada de los switches
          CLK     : in STD_LOGIC;                      -- Entrada senal de reloj
          RESET   : in STD_LOGIC;                      -- Entrada senal Reset
          S       : out STD_LOGIC_VECTOR(7 downto 0)); -- Salida de la acumulacion
end logical;
-- Fin componente logical
architecture behavioral of logical is
begin
    process(CLK, RESET)           -- Process que identifica un cambio en la senal
-- CLK y Reset, para verificar la secuencia de abajo
    begin
        if (RESET = '0') then    -- Si la senal RESET vale 0, reiniciamos el valor
            S <= "00000000";
        elsif (CLK'event AND CLK = '1') then -- Si se activa la senal del reloj
            S <= A;                  -- S toma el valor de A
        end if;
    end process;
end behavioral;
```

BCD a 7 segmentos

Antes de poder unificar todo el circuito, debemos crear una unidad de binario a decimal, para esta información desplegarla en un display de 7 segmentos.



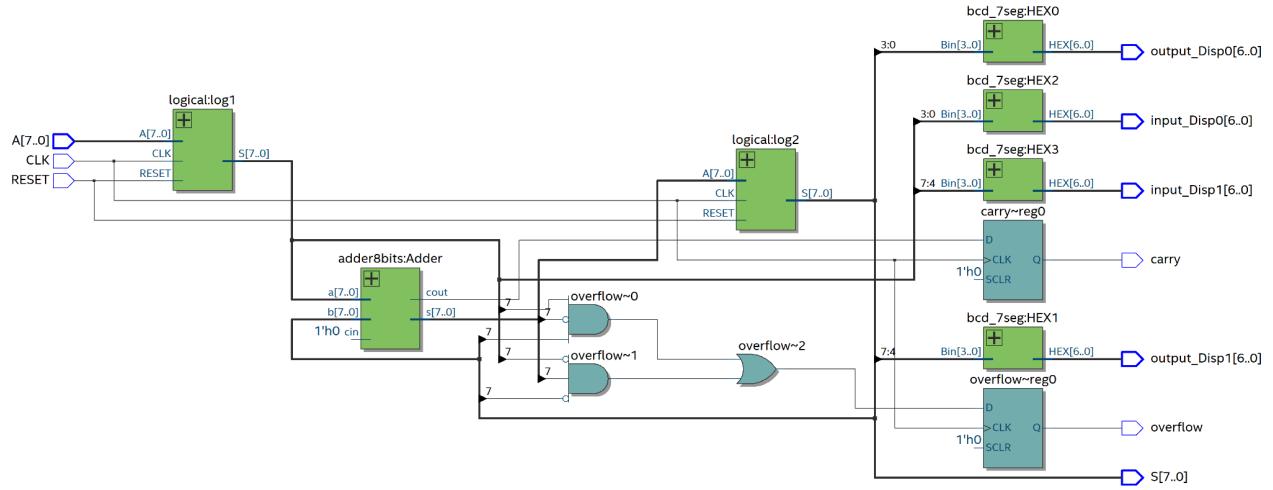
BCD a 7 segmentos.

Código

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
-- Entidad para construir un BCD a un display de 7 segmentos
entity bcd_7seg is
    Port ( Bin : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada del valor en binario
           HEX : out STD_LOGIC_VECTOR(6 downto 0)); -- Display de 7 segmentos
end bcd_7seg;
-- Fin de la entidad
-- Asignacion de valores por medio de acuaciones a cada segmento del display.
architecture Behavioral of bcd_7seg is
begin
    process(Bin)
    begin
        case Bin is
            when "0000" =>
                HEX <= "1000000"; -- 0
            when "0001" =>
                HEX <= "1111001"; -- 1
            when "0010" =>
                HEX <= "0100100"; -- 2
            when "0011" =>
                HEX <= "0110000"; -- 3
            when "0100" =>
                HEX <= "0011001"; -- 4
            when "0101" =>
                HEX <= "0010010"; -- 5
            when "0110" =>
                HEX <= "0000010"; -- 6
            when "0111" =>
                HEX <= "1111000"; -- 7
            when "1000" =>
                HEX <= "0000000"; -- 8
            when "1001" =>
                HEX <= "0010000"; -- 9
            when "1010" =>
                HEX <= "0001000"; -- A
            when "1011" =>
                HEX <= "0000011"; -- b
            when "1100" =>
                HEX <= "0100011"; -- c
            when "1101" =>
                HEX <= "0100001"; -- d
            when "1110" =>
                HEX <= "0000110"; -- E
            when "1111" =>
                HEX <= "0001110"; -- F
        end case;
    end process;
end Behavioral;
```

Acumulador de 8 bits

Ahora que tenemos todos los módulos definidos, debemos unificarlos en un solo módulo, el cual realizará la función del acumulador de 8 bits.



Circuito final diseñado.

Podemos observar como contamos con los 8 bits de entrada (A), además de la señal de reloj y la señal de RESET, estas señales son trabajadas por el sumador de 8 bits, y el control de lógica que ayuda a que funcione correctamente, por último en la parte de salidas tenemos los convertidores BCD conectados a los displays de 7 segmentos, con su debido overflow y señal de Carry.

Código

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
-- Inicio del acumulador de 8 bits
entity accumulator8bits is
    port( A          : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada de valores por SW
          CLK         : in STD_LOGIC;                      -- Senal de reloj
          RESET       : in STD_LOGIC;                      -- Senal de RESET
          S          : out STD_LOGIC_VECTOR(7 downto 0); -- Salida de LEDS con valor
          input_Displ : out STD_LOGIC_VECTOR(6 downto 0);
          input_Dispo : out STD_LOGIC_VECTOR(6 downto 0);
          output_Displ: out STD_LOGIC_VECTOR(6 downto 0);
          output_Dispo: out STD_LOGIC_VECTOR(6 downto 0);
          overflow     : out STD_LOGIC;                   -- Salida de OVERFLOW
          carry        : out STD_LOGIC);                  -- Salida de Carry
end accumulator8bits;
-- Fin de declaracion del acumulador de 8 bits

architecture behavioral of accumulator8bits is
-- Definicion de componentes
-- Adder de 8 bits
    component adder8bits
        port( a      : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal A
              b      : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal B
              cout  : out STD_LOGIC;                      -- Salida de suma
              cin   : in STD_LOGIC);                     -- Entrada de carry-in
    end component;
-- Registro para el carry
    type reg_type is record
        D : STD_LOGIC_VECTOR(7 downto 0);
        Q : STD_LOGIC_VECTOR(7 downto 0);
        SCLR : STD_LOGIC;
        CLK : STD_LOGIC;
    end record;
    signal carry_reg : reg_type;
-- Registro para el overflow
    type reg_overflow is record
        D : STD_LOGIC;
        Q : STD_LOGIC;
        SCLR : STD_LOGIC;
        CLK : STD_LOGIC;
    end record;
    signal overflow_reg : reg_overflow;
begin
    -- Se define el adder de 8 bits
    adder : adder8bits
        port map( a => a,
                  b => b,
                  cout => cout,
                  cin => cin);
    -- Se define el registro para el carry
    process(CLK)
    begin
        if (CLK'event and CLK = '1') then
            carry_reg.Q <- carry_reg.D;
        end if;
    end process;
    carry_reg.D <- cout;
    carry_reg.SCLR <- '1h0;
    carry_reg.CLK <- >CLK;
    -- Se define el registro para el overflow
    process(CLK)
    begin
        if (CLK'event and CLK = '1') then
            overflow_reg.Q <- overflow_reg.D;
        end if;
    end process;
    overflow_reg.D <- overflow;
    overflow_reg.SCLR <- '1h0;
    overflow_reg.CLK <- >CLK;
    -- Se definen las salidas
    S <- carry_reg.Q;
    output_Displ <- carry;
    output_Dispo <- overflow;
    input_Displ <- carry;
    input_Dispo <- overflow;
    -- Se definen las salidas de display
    disp0 : bcd_7seg:HEX0
        port map( Bin => bin3_0,
                  HEX => hex3_0);
    disp2 : bcd_7seg:HEX2
        port map( Bin => bin3_0,
                  HEX => hex3_0);
    disp3 : bcd_7seg:HEX3
        port map( Bin => bin7_4,
                  HEX => hex7_4);
    disp1 : bcd_7seg:HEX1
        port map( Bin => bin7_4,
                  HEX => hex7_4);

```

```

        b    : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal B
        cin : in STD_LOGIC;                      -- Salida de la suma
        s    : out STD_LOGIC_VECTOR(7 downto 0);-- Salida del carry
        cout: out STD_LOGIC );
    end component;
-- Logica
component logical
port( A      : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada de los switches
      CLK    : in STD_LOGIC;                      -- Entrada senal de reloj
      RESET  : in STD_LOGIC;                      -- Entrada senal Reset
      S      : out STD_LOGIC_VECTOR(7 downto 0));-- Salida de la acumulacion
end component;
-- bcd a 7 segmentos
component bcd_7seg is
    Port ( Bin : in STD_LOGIC_VECTOR(3 downto 0); -- Entrada del valor en binario
           HEX : out STD_LOGIC_VECTOR(6 downto 0)); -- Display de 7 segmentos
end component;

signal sA, sA2, sS : STD_LOGIC_VECTOR(7 downto 0); -- senales auxiliares para la
suma
signal sCarry, sOverflow : STD_LOGIC;                  -- senales para el carry y el
overflow
begin
-- Definición de entradas y salidas de módulos
    log1  : logical PORT MAP(A, CLK, RESET, sA ); -- sA toma el valor de A
    log2  : logical PORT MAP(sS, CLK, RESET, sA2); -- sA2 toma el valor de sS (suma
actual)
    Adder : adder8bits PORT MAP(sA, sA2, '0', sS, sCarry); -- Se ejecuta la suma entre
sA y sA2, guardandola en sS
    HEX3  : bcd_7seg PORT MAP(sA( 7 downto 4), input_Displ ); -- Desplpegamos en
displays
    HEX2  : bcd_7seg PORT MAP(sA( 3 downto 0), input_Dispo ); -- Desplpegamos en
displays
    HEX1  : bcd_7seg PORT MAP(sA2(7 downto 4), output_Displ); -- Desplpegamos en
displays
    HEX0  : bcd_7seg PORT MAP(sA2(3 downto 0), output_Dispo); -- Desplpegamos en
displays

-- Actuar en señales del CLK
process(CLK, RESET)
begin
    if CLK'event AND CLK='1' then
        -- Lógica combinacional para detectar Overflow
        overflow <= (sA(7) AND sA2(7) AND NOT sS(7)) OR
                    (NOT sA(7) AND NOT sA2(7) AND sS(7));
        carry <= sCarry; -- Asignamos carry
    end if;
end process;

S <= sA2; -- Asignamos valor final
end behavioral;

```

Pin Planner

Pin Planner - C:/Users/sr18/Documents/8vo Semestre/Laboratorio/Practicas/LabNewProjectWizard - Lab8_P1

File Edit View Processing Tools Window Help

Search altera.com

Top View - Wire Bond
MAX 10 - 10M50DAF484C7G

Groups

- Node Name Direction
- input_6...0 Output
- input_5...0 Output
- input_4...0 Output
- input_3...0 Output
- input_2...0 Output
- input_1...0 Output
- input_0...0 Output
- input_6...0 Output
- input_5...0 Output
- input_4...0 Output
- input_3...0 Output
- input_2...0 Output
- input_1...0 Output
- input_0...0 Output

Tasks

- Early Pin Planning
- Run I/O Assignm
- Export Pin Assign
- Pin Finder...
- Highlight Pins
- I/O Banks
- VREF Groups
- Edges
- Clock Pins

Pin Legend

Symbol	Pin Type
○	User I/O
●	User ass...
●	Fitter assi...
●	Unbonded...
●	Reserved ...
○	Other co...
E	DEV_OE
R	DEV_CLR
D	DIFF_n
P	DIFF_p
Q	DQ
S	DQS
S	DQSB
L	CLK_n
J	CLK_p
L	Other PLL
D	Other du...
T	TDI
K	TCK
M	TMS
Y	TDO
V	VREF
△	VCCP/VC...
A	VCCA
A	VCCINT

PIN_E17

Node Name	Direction	Location	I/O Bank	/REF Group	Iter Locat	I/O Standar	Reserved	rent Stren	Slew Rate	Differential P.	ct Preserva
A[7]	Input	PIN_A14	7	B7_N0	PIN_A14	2.5 V		12mA..ult)			
A[6]	Input	PIN_A13	7	B7_N0	PIN_A13	2.5 V		12mA..ult)			
A[5]	Input	PIN_B12	7	B7_N0	PIN_B12	2.5 V		12mA..ult)			
A[4]	Input	PIN_A12	7	B7_N0	PIN_A12	2.5 V		12mA..ult)			
A[3]	Input	PIN_C12	7	B7_N0	PIN_C12	2.5 V		12mA..ult)			
A[2]	Input	PIN_D12	7	B7_N0	PIN_D12	2.5 V		12mA..ult)			
A[1]	Input	PIN_C11	7	B7_N0	PIN_C11	2.5 V		12mA..ult)			
A[0]	Input	PIN_C10	7	B7_N0	PIN_C10	2.5 V		12mA..ult)			
carry	Output	PIN_A11	7	B7_N0	PIN_A11	2.5 V		12mA..ult)	2 (default)		
CLK	Input	PIN_A7	7	B7_N0	PIN_A7	2.5 V		12mA..ult)			
input_Dispo[6]	Output	PIN_B22	6	B6_N0	PIN_H12	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[5]	Output	PIN_C22	6	B6_N0	PIN_J12	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[4]	Output	PIN_B21	6	B6_N0	PIN_H13	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[3]	Output	PIN_A21	6	B6_N0	PIN_E11	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[2]	Output	PIN_B19	7	B7_N0	PIN_C8	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[1]	Output	PIN_A20	7	B7_N0	PIN_E10	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[0]	Output	PIN_B20	6	B6_N0	PIN_C7	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[16]	Output	PIN_E17	6	B6_N0	PIN_E13	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[15]	Output	PIN_D19	6	B6_N0	PIN_C9	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[14]	Output	PIN_C20	6	B6_N0	PIN_E12	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[3]	Output	PIN_C19	7	B7_N0	PIN_B14	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[2]	Output	PIN_E21	6	B6_N0	PIN_B7	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[1]	Output	PIN_E22	6	B6_N0	PIN_J11	2.5 V...ault)		12mA..ult)	2 (default)		
input_Dispo[10]	Output	PIN_F21	6	B6_N0	PIN_J10	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[6]	Output	PIN_C17	7	B7_N0	PIN_J13	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[5]	Output	PIN_D17	7	B7_N0	PIN_A18	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[4]	Output	PIN_E16	7	B7_N0	PIN_C14	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[3]	Output	PIN_C16	7	B7_N0	PIN_H14	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[2]	Output	PIN_C15	7	B7_N0	PIN_C16	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[1]	Output	PIN_E15	7	B7_N0	PIN_B15	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[0]	Output	PIN_C14	7	B7_N0	PIN_A15	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[6]	Output	PIN_B17	7	B7_N0	PIN_A19	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[15]	Output	PIN_A18	7	B7_N0	PIN_A16	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[14]	Output	PIN_A17	7	B7_N0	PIN_B16	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[13]	Output	PIN_B16	7	B7_N0	PIN_C15	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[12]	Output	PIN_E18	6	B6_N0	PIN_A20	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[11]	Output	PIN_D18	6	B6_N0	PIN_D15	2.5 V...ault)		12mA..ult)	2 (default)		
output_Dispo[10]	Output	PIN_C18	7	B7_N0	PIN_A17	2.5 V...ault)		12mA..ult)	2 (default)		
overflow	Output	PIN_B11	7	B7_N0	PIN_B11	2.5 V		12mA..ult)	2 (default)		
RESET	Input	PIN_B8	7	B7_N0	PIN_B8	2.5 V		12mA..ult)			
S[7]	Output	PIN_D14	7	B7_N0	PIN_D14	2.5 V		12mA..ult)	2 (default)		
S[6]	Output	PIN_E14	7	B7_N0	PIN_E14	2.5 V		12mA..ult)	2 (default)		
S[5]	Output	PIN_C13	7	B7_N0	PIN_C13	2.5 V		12mA..ult)	2 (default)		
S[4]	Output	PIN_D13	7	B7_N0	PIN_D13	2.5 V		12mA..ult)	2 (default)		
S[3]	Output	PIN_B10	7	B7_N0	PIN_B10	2.5 V		12mA..ult)	2 (default)		
S[2]	Output	PIN_A10	7	B7_N0	PIN_A10	2.5 V		12mA..ult)	2 (default)		
S[1]	Output	PIN_A9	7	B7_N0	PIN_A9	2.5 V		12mA..ult)	2 (default)		
S[0]	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA..ult)	2 (default)		

Asignación de pines.

Demostracion Parte 1:

<https://drive.google.com/file/d/1Sq0fmQabeRxvpHWeJr7V1In6Hxkdl4o/view?usp=sharing>

Simulación



Simulación obtenida del siguiente test bench,

Código

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity tb_accumulator8bits is
end tb_accumulator8bits;

architecture Behaviour of tb_accumulator8bits is
component accumulator8bits
    port( A           : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada A
          CLK         : in STD_LOGIC;                      -- Senal de reloj
          RESET       : in STD_LOGIC;                      -- Senal RESET
          S           : out STD_LOGIC_VECTOR(7 downto 0); -- Resultado
          input_Displ1: out STD_LOGIC_VECTOR(6 downto 0); -- Display 3
          input_Displ0: out STD_LOGIC_VECTOR(6 downto 0); -- Display 2
          output_Displ1: out STD_LOGIC_VECTOR(6 downto 0); -- Display 1
          output_Displ0: out STD_LOGIC_VECTOR(6 downto 0); -- Display 0
          overflow     : out STD_LOGIC;                   -- senal de overflow
          carry        : out STD_LOGIC);                  -- senal de carry
    end component;

    signal A           : STD_LOGIC_VECTOR(7 downto 0):= (others => '0'); -- Valor de entrada
    signal CLK         : STD_LOGIC := '0';                                -- Senal de reloj
    signal RESET       : STD_LOGIC := '1';                                -- Senal RESET
    signal S           : STD_LOGIC_VECTOR(7 downto 0); -- Resultado
    signal input_Displ1: STD_LOGIC_VECTOR(6 downto 0); -- Display 3
    signal input_Displ0: STD_LOGIC_VECTOR(6 downto 0); -- Display 2
    signal output_Displ1: STD_LOGIC_VECTOR(6 downto 0); -- Display 1
    signal output_Displ0: STD_LOGIC_VECTOR(6 downto 0); -- Display 0
    signal overflow     : STD_LOGIC;                                -- senal de overflow
    signal carry        : STD_LOGIC;                                -- senal de carry

begin
vectors: PROCESS
    Begin
        -- Suma 1
        CLK      <= '1';           -- Flanco alto de reloj
```

```

A          <= "00000001"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
-- Bajamos senal de reloj
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
-- Suma 1
CLK        <= '1';           -- Flanco alto de reloj
A          <= "00000001"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
-- RESET
--CLK      <= '0';           -- Flanco alto de reloj
--A       <= "00000000"; -- Anadimos valor de 1
RESET     <= '0';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
-- Suma 40
CLK        <= '1';           -- Flanco alto de reloj
A          <= "00100000"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
-- Suma 80
CLK        <= '1';           -- Flanco alto de reloj
A          <= "00100000"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
-- Suma C0
CLK        <= '1';           -- Flanco alto de reloj
A          <= "00100000"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
-- Suma 100
CLK        <= '1';           -- Flanco alto de reloj
A          <= "00100000"; -- Anadimos valor de 1
RESET     <= '1';           -- Reset en 0
wait for 20ns;
CLK        <= '0';           -- Flanco alto de reloj
wait for 10ns;
wait;
end PROCESS;

U1: accumulator8bits PORT MAP (A, CLK, RESET, S, input_Displ1, input_Dispo,
output_Displ1, output_Dispo, overflow, carry);
end;

```

Parte 2

Para la siguiente parte de la práctica, es necesario dotar con la capacidad de seleccionar entre suma y resta nuestro proyecto. Por lo que una de las entradas será denominada de la siguiente manera “add_sub”, cuando se encuentre en 1, restamos, cuando este en 0, sumaremos.

sub_Add

Este código nos permite generar un complemento a dos del input que sea asignado en caso de ser seleccionada la opción para generar esta resta, por lo que al generar una suma con un número negativo, podremos obtener este valor.

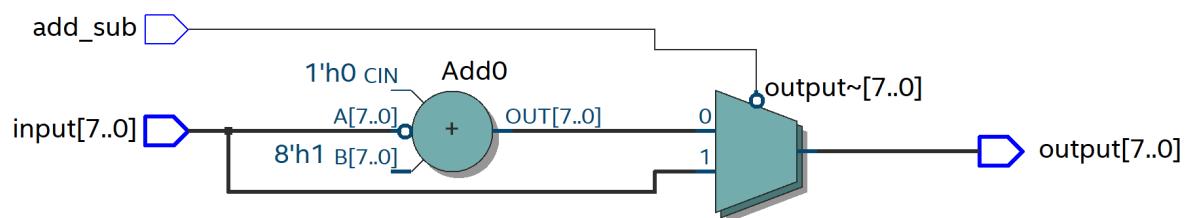


Diagrama generado para seleccionar entre obtener el complemento a 2 o mantener la entrada intacta.

Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
-- Entidad que permite realizar una inversion de la entrada
entity subAdd is
    port( input : in STD_LOGIC_VECTOR(7 downto 0); -- leemos la entrada
          add_sub : in STD_LOGIC; -- Entrada para elegir
          entre suma y resta
          output : out STD_LOGIC_VECTOR(7 downto 0));-- Salida que niega o mantiene la
          entrada
    end subAdd;
-- Fin de la entidad

architecture behavioral of subAdd is
begin
    process(add_sub) is -- Detectamos cambios en el pin de elección
    begin
        case add_sub is -- Segun el pin de elección
            when '0' => -- Si se encuentra en 0, mantenemos el input
                output <= input;-- Asignamos el input a la salida
            when others => -- Si se encuentra en 1, realizamos la resta
                output <= not input + 8'h1;
        end case;
    end process;
end behavioral;
```

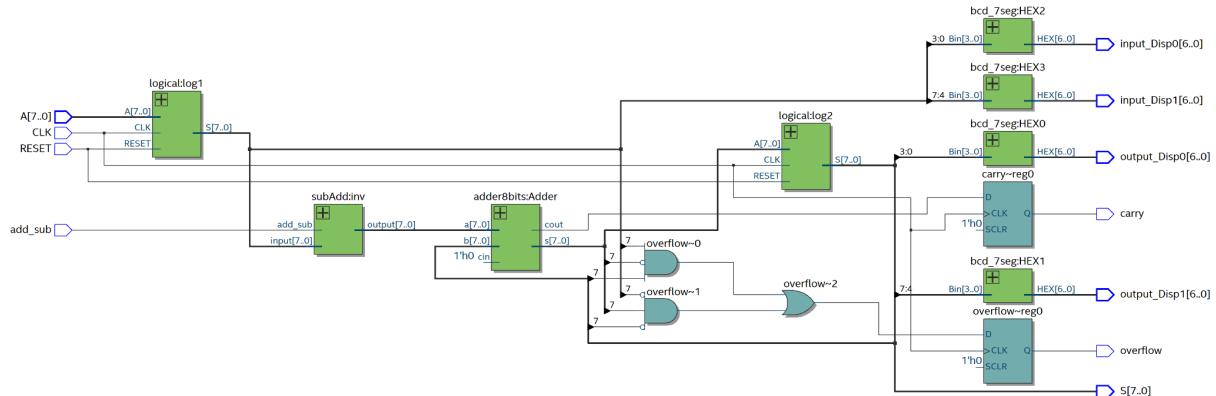
```

        when '1' => -- Si el resultado es 1 invertimos la netrada
                      output <= (NOT input) + 1; -- Obtenemos su complemento para al sumar
generar una resta
                                              -- y asignamos el output
      end case;
    end process;
end behavioral;

```

Acumulador Sumador Restador

El siguiente circuito es generado tomando en base el código de la parte 1, añadiendo el módulo de inversión.



Circuito generado.

Código:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity subAcum8bits is
  port( A           : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada A
        CLK         : in STD_LOGIC;                         -- Senal de reloj
        RESET       : in STD_LOGIC;                         -- Senal RESET
        add_sub     : in STD_LOGIC;                         -- Control de suma y
resta
        S           : out STD_LOGIC_VECTOR(7 downto 0); -- Resultado
        input_Displ : out STD_LOGIC_VECTOR(6 downto 0); -- Display 3
        input_Dispo : out STD_LOGIC_VECTOR(6 downto 0); -- Display 2
        output_Displ: out STD_LOGIC_VECTOR(6 downto 0); -- Display 1
        output_Dispo: out STD_LOGIC_VECTOR(6 downto 0); -- Display 0
        overflow     : out STD_LOGIC;                      -- senal de overflow
        carry        : out STD_LOGIC);                     -- senal de carry
end subAcum8bits;
-- Suma o resta 8 bits
architecture behavioral of subAcum8bits is
  -- Nos permite sumar 2 numeros de 8 bits
  component adder8bits
    port( a   : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal A
          b   : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada senal B
          cin : in STD_LOGIC;                   -- Salida de la suma
          S   : out STD_LOGIC_VECTOR(7 downto 0);-- Salida del carry
          cout: out STD_LOGIC );              -- Salida del cout
  end component;
  -- Variables auxiliares
  variable sTemp : STD_LOGIC_VECTOR(7 downto 0);
  begin
    -- Se declara el componente adder8bits
    adder8bits1 : adder8bits
      port map( a   => A,
                b   => B,
                cin => cin,
                S   => sTemp,
                cout=> cout );
    -- Se calcula el resultado
    S <= sTemp;
    -- Se calcula el cout
    cout <= cout;
    -- Se calcula el overflow
    overflow <= cout;
    -- Se calcula el carry
    carry <= cout;
  end;

```

```

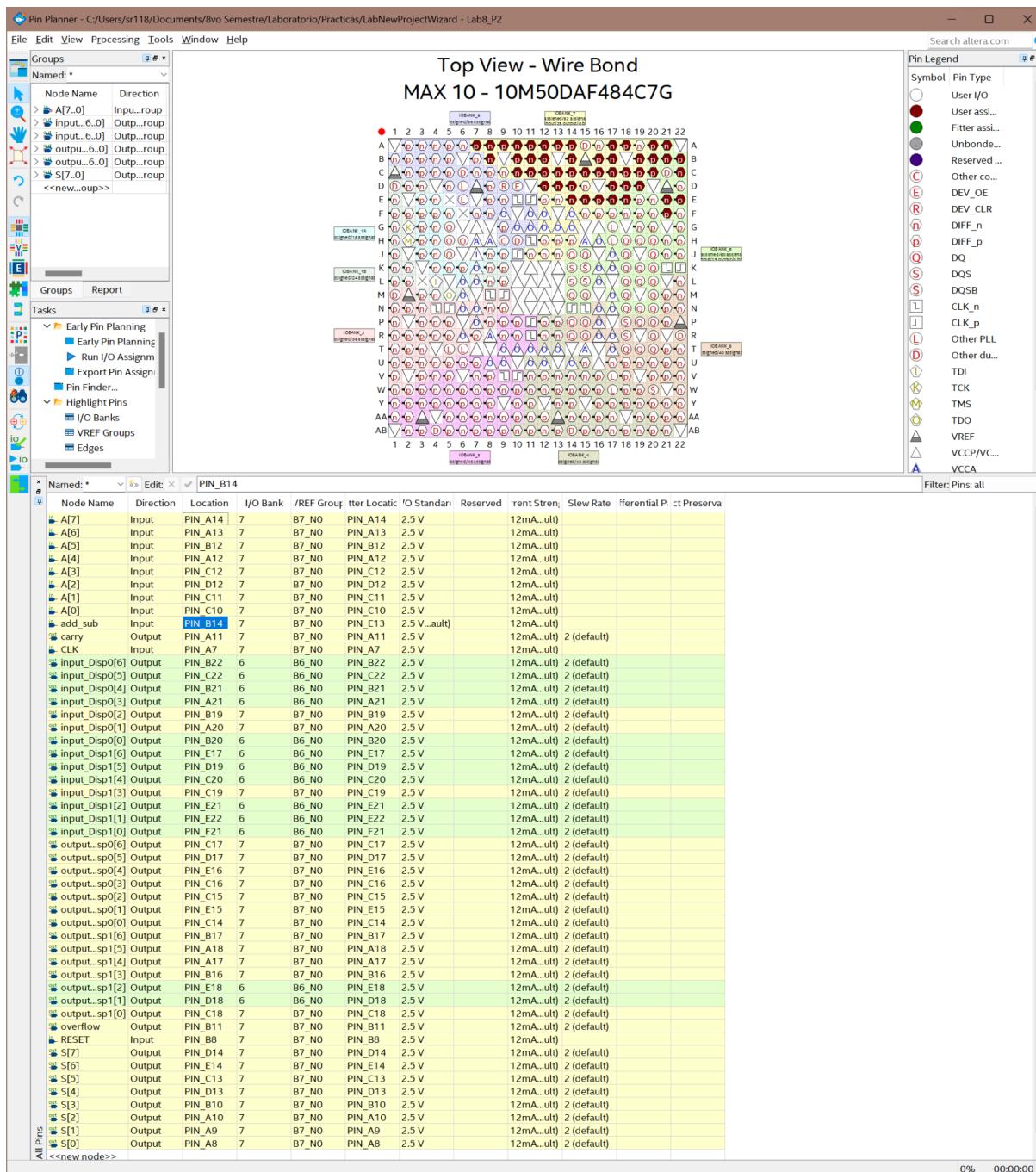
end component;
-- Nos permite realizar un control correcto con la señal de CLK y RESET
component logical
port( A      : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada de los switches
      CLK    : in STD_LOGIC;                      -- Entrada señal de reloj
      RESET : in STD_LOGIC;                      -- Entrada señal Reset
      S      : out STD_LOGIC_VECTOR(7 downto 0));-- Salida de la acumulacion
end component;
-- Despliega digitos en displays de 7 segmentos
component bcd_7seg is
    port ( bin : in STD_LOGIC_VECTOR (3 downto 0);
           HEX : out STD_LOGIC_VECTOR (6 downto 0));
end component;
-- Componente para realizar la inversion de la entrada en caso de ser necesitada
component subAdd is
    port( input   : in STD_LOGIC_VECTOR(7 downto 0); -- leemos la entrada
          add_sub : in STD_LOGIC;                      -- Entrada para elegir
entre suma y resta
          output  : out STD_LOGIC_VECTOR(7 downto 0));-- Salida que niega o mantiene la
entrada
    end component;
-- Señales auxiliares para el transporte del resultado, carry y overflows, ademas de la
señal de
-- inversion
    signal sA, sA2, sS : STD_LOGIC_VECTOR(7 downto 0);
    signal sCarry, sOverflow : STD_LOGIC;
    signal inverted : STD_LOGIC_VECTOR(7 downto 0);
begin
-- Definición de entradas y salidas de módulos
    log1 : logical PORT MAP(A, CLK, RESET, sA); -- sA toma el valor de A
    log2 : logical PORT MAP(sS, CLK, RESET, sA2); -- sA2 toma el valor de sS (suma
actual)
    Adder : adder8bits PORT MAP(inverted, sA2, '0', sS, sCarry); -- Se ejecuta la suma
entre sA y sA2, guardandola en sS
    HEX3  : bcd_7seg PORT MAP(sA( 7 downto 4), input_Displ1 ); -- Despliegamos en
displays
    HEX2  : bcd_7seg PORT MAP(sA( 3 downto 0), input_Displ0 ); -- Despliegamos en
displays
    HEX1  : bcd_7seg PORT MAP(sA2(7 downto 4), output_Displ1); -- Despliegamos en
displays
    HEX0  : bcd_7seg PORT MAP(sA2(3 downto 0), output_Displ0); -- Despliegamos en
displays
    inv   : subAdd PORT MAP(sA, add_sub, inverted);           -- Unidad de inversion

-- Actuar en señales del CLK
process(CLK, add_sub)
begin
    if CLK'event AND CLK='1' then
        -- Lógica combinacional para detectar Overflow
        overflow <= (sA(7) AND sA2(7) AND NOT sS(7)) OR
                    (NOT sA(7) AND NOT sA2(7) AND sS(7));
        carry <= sCarry; -- Asignamos carry
    end if;
end process;

```

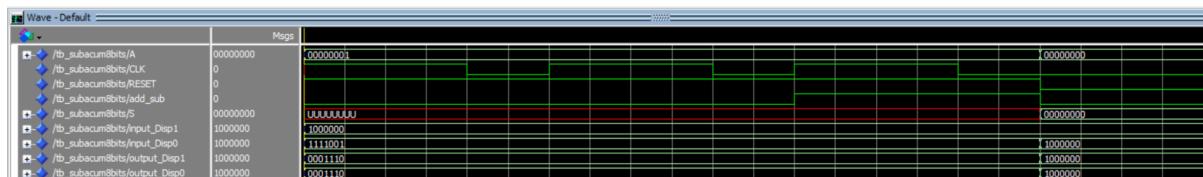
```
s <= sA2;  
end behavioral;
```

Pin Planner



Asignación de pines para la parte 2.

Simulación



En esta simulación se muestra el comportamiento de aumentar en 1 unidad la salida y después restarla al activar la señal de add_sub.

Código:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity tb_subAcum8bits is
end tb_subAcum8bits;

architecture Behaviour of tb_subAcum8bits is
    component subAcum8bits
        port( A           : in STD_LOGIC_VECTOR(7 downto 0); -- Entrada A
              CLK          : in STD_LOGIC;                      -- Senal de reloj
              RESET        : in STD_LOGIC;                      -- Senal RESET
              add_sub      : in STD_LOGIC;                      -- Control de suma y resta
              S            : out STD_LOGIC_VECTOR(7 downto 0); -- Resultado
              input_Displ1 : out STD_LOGIC_VECTOR(6 downto 0); -- Display 3
              input_Displ0 : out STD_LOGIC_VECTOR(6 downto 0); -- Display 2
              output_Displ1: out STD_LOGIC_VECTOR(6 downto 0); -- Display 1
              output_Displ0: out STD_LOGIC_VECTOR(6 downto 0); -- Display 0
              overflow     : out STD_LOGIC;                      -- senal de overflow
              carry        : out STD_LOGIC);                     -- senal de carry
    end component;

    signal A           : STD_LOGIC_VECTOR(7 downto 0):= (others => '0'); -- Valor de entrada
    signal CLK         : STD_LOGIC := '0';                                -- Senal de reloj
    signal RESET       : STD_LOGIC := '1';                                -- Senal RESET
    signal add_sub     : STD_LOGIC := '0';                                -- Control de suma y resta
    signal S            : STD_LOGIC_VECTOR(7 downto 0); -- Resultado
    signal input_Displ1 : STD_LOGIC_VECTOR(6 downto 0); -- Display 3
    signal input_Displ0 : STD_LOGIC_VECTOR(6 downto 0); -- Display 2
    signal output_Displ1: STD_LOGIC_VECTOR(6 downto 0); -- Display 1
    signal output_Displ0: STD_LOGIC_VECTOR(6 downto 0); -- Display 0
    signal overflow     : STD_LOGIC;                                     -- senal de overflow
    signal carry        : STD_LOGIC;                                     -- senal de carry

begin
vectors: PROCESS
    Begin
        -- Suma 1
        CLK      <= '1';          -- Flanco alto de reloj
        A       <= "00000001"; -- Anadimos valor de 1
        RESET   <= '1';          -- Reset en 0
        add_sub <= '0';          -- Mantenemos en suma
        wait for 20ns;
        -- Bajamos senal de reloj
        CLK      <= '0';          -- Flanco alto de reloj
        wait for 10ns;
        -- Suma 1
        CLK      <= '1';          -- Flanco alto de reloj

```

```

A           <= "00000001"; -- Anadimos valor de 1
RESET      <= '1';          -- Reset en 0
add_sub   <= '0';          -- Mantenemos en suma
wait for 20ns;
CLK         <= '0';          -- Flanco alto de reloj
wait for 10ns;
-- Resta 1
CLK         <= '1';          -- Flanco alto de reloj
A           <= "00000001"; -- Anadimos valor de 1
RESET      <= '1';          -- Reset en 0
add_sub   <= '1';          -- Mantenemos en suma
wait for 20ns;
CLK         <= '0';          -- Flanco alto de reloj
wait for 10ns;
-- RESET
CLK         <= '0';          -- Flanco alto de reloj
A           <= "00000000"; -- Anadimos valor de 1
RESET      <= '0';          -- Reset en 0
add_sub   <= '0';          -- Mantenemos en suma
wait for 20ns;
CLK         <= '0';          -- Flanco alto de reloj
wait;
end PROCESS;
U1: subAcum8bits PORT MAP (A, CLK, RESET, add_sub, S, input_Displ, input_Dispo,
output_Displ, output_Dispo, overflow, carry);
end;

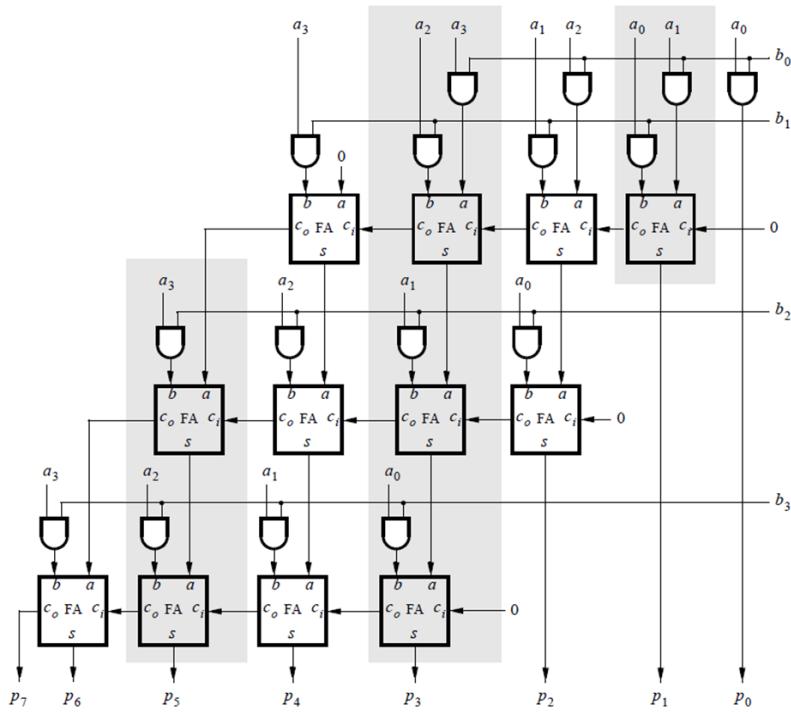
```

Demostración

<https://drive.google.com/file/d/1SzNEdcFzd8VELAVeEnvlSzLszKErWfJv/view?usp=sharing>

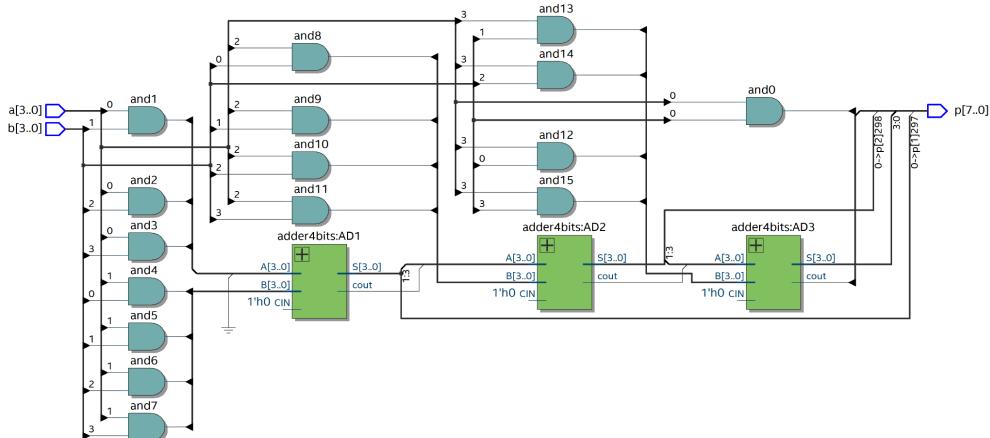
Parte 3

Por último, ejecutaremos un multiplicador. El diseño propuesto es el siguiente:



Diseño propuesto para la multiplicación.

Multiplicador 4 bits



Circuito generado por el siguiente código, contiene los 3 sumadores de 4 bits.

Código:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
-- Entidad que define los puertos del diagrama provisto en la practica
entity multi4bits is
    port( a : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada a, 4 bits de longitud
          b : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada b, 4 bits de longitud
          p : out STD_LOGIC_VECTOR(7 downto 0) -- P, resultado de la multiplicacion
        );
end multi4bits;
-- Fin de entidad
-- Importamos el sumador de 4 bits

```

```

architecture behavioral of multi4bits is
    component adder4bits
        port( a : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada al sumador a
              b : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada al sumador b
              cin : in STD_LOGIC;                      -- Carry de entrada
              s : out STD_LOGIC_VECTOR(3 downto 0);     -- Resultado de la suma
              cout: out STD_LOGIC );                  -- Carry de salida
    end component;
-- Fin sumador
-- Señales auxiliares

    signal l0, l1, l2, l3, add1_aux, add2_aux, add3_aux, aux_A, aux_B, aux_0, aux_1 : STD_LOGIC_VECTOR(3 downto 0);
    -- Señales auxiliares para las señales intermedias en el circuito
    signal col1, co2, co3 : STD_LOGIC;
    -- Señales auxiliares para las señales intermedias en el circuito

begin
    -- Instanciamos los componentes y los interconectamos como en el diagrama,
    10 <= (b(0) and a(3)) & (b(0) and a(2)) & (b(0) and a(1)) & (b(0) and a(0));
    11 <= (b(1) and a(3)) & (b(1) and a(2)) & (b(1) and a(1)) & (b(1) and a(0));
    12 <= (b(2) and a(3)) & (b(2) and a(2)) & (b(2) and a(1)) & (b(2) and a(0));
    13 <= (b(3) and a(3)) & (b(3) and a(2)) & (b(3) and a(1)) & (b(3) and a(0));

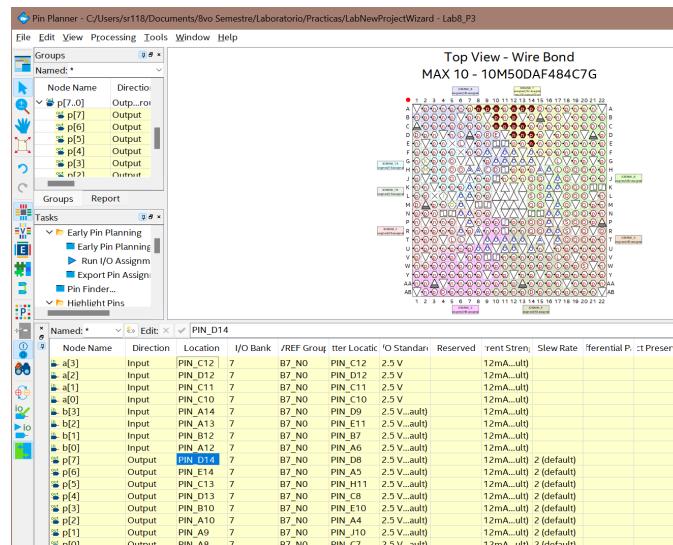
    AD0: adder4bits PORT MAP( '0' & l0(3 downto 1), l1, '0', add1_aux, col1);
    AD1: adder4bits PORT MAP( '0' & add1_aux(3 downto 1), l2, col1, add2_aux, co2);
    AD2: adder4bits PORT MAP( '0' & add2_aux(3 downto 1), l3, co2, add3_aux, co3);

    aux_A <= a;
    aux_B <= b;
    aux_0 <= add3_aux(0) & add2_aux(0) & add1_aux(0) & l0(0);
    aux_1 <= co3 & add3_aux(3) & add3_aux(2) & add3_aux(1);
    -- Asignamos el resultado a la variable de la multiplicacion
    p <= aux_1 & aux_0;

end behavioral;

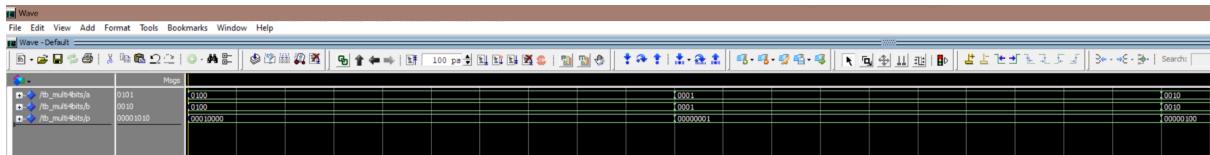
```

Pin planner



Asignación de pines, donde tenemos entrada a[sw(0-3)]y b[sw(4-7)], con una salida p(LEDs)

Simulación



Simulación ejecutada en ModelSim.

Código:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity tb_multi4bits is
end tb_multi4bits;

architecture Behaviour of tb_multi4bits is
component multi4bits
port( a : in STD_LOGIC_VECTOR(3 downto 0);      -- Entrada a,4 bits de longitud
      b : in STD_LOGIC_VECTOR(3 downto 0); -- Entrada b,4 bits de longitud
      p : out STD_LOGIC_VECTOR(7 downto 0) -- P, resultado de la multiplicacion
    );
end component;

signal a    : STD_LOGIC_VECTOR(3 downto 0):= (others => '0'); -- Valor de entrada a
signal b    : STD_LOGIC_VECTOR(3 downto 0):= (others => '0'); -- Valor de entrada b
signal p    : STD_LOGIC_VECTOR(7 downto 0); -- Valor de salida de la multiplicacion

begin
vectors: PROCESS
  Begin
    -- Iniciamos con la primera multiplicacion 4*4 = 16
    a <= "0100";
    b <= "0100";
    wait for 20ns;
    -- Iniciamos con la primera multiplicacion 1*1 = 1
    a <= "0001";
    b <= "0001";
    wait for 20ns;
    -- Iniciamos con la primera multiplicacion 2*2 = 4
    a <= "0010";
    b <= "0010";
    wait for 20ns;
    -- Iniciamos con la primera multiplicacion 5*2 = 10
    a <= "0101";
    b <= "0010";
    wait;
  end PROCESS;
  P1 : multi4bits port map(a, b, p);
end;
```

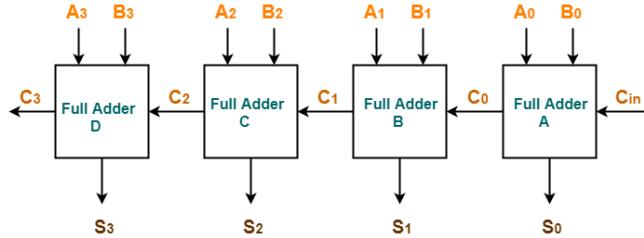
Demostración

<https://drive.google.com/file/d/1Sg-fZkGQyMm9y84ri-u9prext-kp4VL0/view?usp=sharing>

Explicacion conceptos:

- *Ripple-carry adder:*

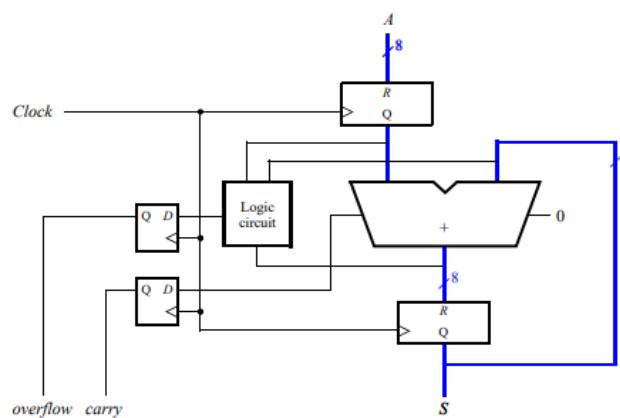
El ripple Carry Adder, es una cadena de sumadores que se encuentran conectados entre ellos, esto con el objetivo de poder transportar a través de ellos un Carry. Para esto contamos con 1 entrada que recibe el carry (en caso de que un adder anterior o de entrada ingrese dicho carry) y 1 salida para en caso de que la suma generará un carry, comunicar al siguiente adder. Cabe mencionar que los Adders cuentan con entradas para los números que se desean sumar y una salida donde se muestra el resultado de la suma.



Ejemplo de un: 4-bit Ripple Carry Adder.

- *Accumulator:*

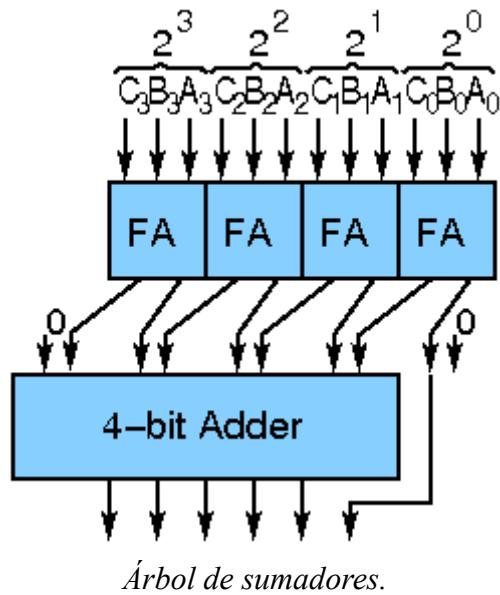
Un acumulador funciona de la siguiente manera: El valor de entrada se sumará con el nuevo valor de entrada. En este caso teníamos el input A, el cual era de 8 bits y la acumulación se realizaba cada que se generaba una señal de reloj en transición positiva, esto debe ser controlado por una señal de reloj, para poder dar tiempo suficiente a la entrada para cambiar de valor, ya que podría causar una suma de valores que no deseemos. Existe una unidad sumadora que cuenta con nuestros Adders conectados entre ellos y que reciben como entrada, tanto como el valor A y la salida de la suma, esto con el propósito de acumular el resultado. También contamos con salidas que indican carry y overflow en caso de que estos eventos sucedan.



Acumulador de 8 bits.

- *Tree-adder:*

Tree-adder, como su nombre lo menciona se refiere a la creación de sumadores ordenador en manera descendente, donde el resultado de una suma le llega a otro sumador, con un objetivo en específico. Este tipo de sumador se utilizó para realizar el circuito de la multiplicación, pero con la diferencia que teníamos una salida P_x , cada que salimos de una suma.



Problemas y soluciones

- **No se consideran todos los valores posibles:**

Para realizar la simulación en ModelSim, nos aparecían errores a la hora de querer graficar el comportamiento en la simulación, uno de estos errores es que en códigos como `bcd_7seg`, que es un código que nos permite realizar una conversión de binario a decimal, para desplegarlo en un display de 7 segmentos. Esto sucedía debido a que en los estatutos de “When” y “Case” hacía falta agregar en el último caso un “when others”, esto indicando que las demás combinaciones faltantes corresponden a un valor en concreto, esto solucionó el hecho de que no se graficaron las simulaciones.

Reflexiones Individuales

Hector Javier Pequeno Chairez: En esta práctica, recordamos conceptos importantes como el uso de `process()`, esto nos ayuda a detectar un cambio en las señales que le indiquemos, ejemplo de este fue el uso del `process(CLK)`, en donde necesitábamos seguir una lógica secuencial cada que se tenía un cambio en el flanco positivo de la señal del reloj. También, como es que debemos considerar todos los casos posibles si es que queremos utilizar la herramienta de ModelSim, ya que lanzaba mensajes de error, debido a que no todas las combinaciones posibles habían sido consideradas y por último, que los sumadores conectados en cascada y además en forma de árbol, nos pueden ayudar a realizar multiplicaciones, con la lógica que utilizamos cuando estamos más pequeños. En este caso realizamos los códigos de forma que creamos primero el módulo y luego lo instanciamos dentro de otro, para crear uno más complejo, considero que hubiera sido más fácil trabajarlos de manera de

“comportamiento” describiendo en 1 solo código la lógica de este, pero debido a que no sabía si tenía permitido realizarlo de esa manera, decidimos realizarlo de esta manera.

Gabriela Jazmín Álvarez Espinoza: Durante esta práctica el aplicar la parte de multiplicación creo que fue lo más complicado, sin embargo con ayuda de conceptos vistos en clase, así también la parte de sistemas digitales, nos ayudó a poder implementar y entender cómo los sumadores nos pueden ayudar a generar la operación. Algo para mí que de cierto modo fue retador, fue la parte de entender la lógica de almacenamiento de los árboles de sumadores, pues al inicio yo tenía la idea de implementación de guardar los valores en variables temporales y sumarlos, pero de esta manera no estaba funcionando. Por último, la parte de leer la señal clock fue uno de los mayores problemas, que con ayuda del método “process()” pudimos hacerlo funcionar correctamente.

Bibliografía

1. Global Static:

<https://www.aldec.com/en/support/resources/documentation/faq/1737>

2. VHDL constants:

https://peterfab.com/ref/vhdl/vhdl_renerta/source/vhd00022.htm#:~:text=Constant%20is%20an%20object%20whose,may%20also%20be%20deferred%20constants.

3. Manual DE 10-lite:

https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=1021&FID=a13a278281152b477e60203d34b1baa

4. Manual Practica:

<https://experiencia21.tec.mx/courses/286479/files/117929625?wrap=1>