

## CIRCUITOS COMBINACIONALES

Hernández Castellanos César Uriel

Centro de Investigación en Computación  
Instituto Politécnico Nacional, México  
*uuriel12009u@gmail.com*

**Resumen:** En el presente informe se diseña e implementan tres circuitos combinacionales en los que podemos encontrar un sumador con propagación de acarreo, decodificador hexadecimal y un convertidor de número binario a decimal.

**Palabras Clave:** Circuitos combinacionales, Diseño digital, VHDL, Verilog.

### 1. Introducción.

#### 1.1. Electrónica digital

El gran desarrollo experimentado por la electrónica en los últimos años ha propiciado que la mayoría de los equipos actuales funcionen con sistemas digitales. Un sistema digital se caracteriza por utilizar señales discretas, es decir, señales que toman un número finito de valores en cierto intervalo de tiempo [2].

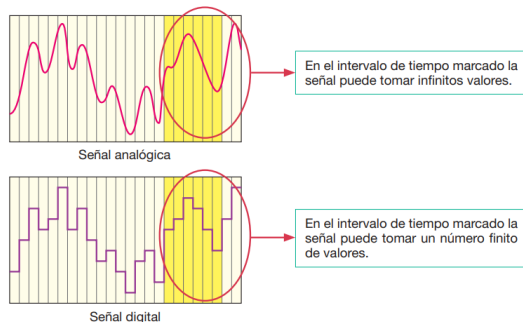


Figura 1: Comparativa gráfica de dos señales

En la figura 1 es posible apreciar que la señal inferior corresponde a la digitalización de la señal analógica, y contiene información suficiente para poder reconstruir la señal digital[2].

Todas las telecomunicaciones modernas (Internet, telefonía, móvil, etc) están basadas en el uso de este tipo de sistemas[2].

Son múltiples las razones que han favorecido el uso extensivo de los sistemas digitales, entre ellas: Mayor fiabilidad, disposición de soporte matemático, dominio de las tecnologías de fabricación y una amplia distribución comercial[2].

#### 1.2. Circuito combinacional

De manera genérica podemos decir que los circuitos combinacionales son aquellos en los que una serie de variables  $X_i$ , definen una serie de funciones  $Z_j$ . Las principales restricciones que caracterizan el comportamiento funcional [1] son las siguientes:

1. Cada combinación de valores de  $X_i$  define un sólo valor de  $Z_j$
2. Los valores de  $Z_j$  sólo pueden cambiar cuando cambian los valores de  $X_i$

La Figura 1 muestra un bloque combinacional genérico.



Figura 2: Circuito combinacional

#### 1.3. Sumador con propagación de acarreo

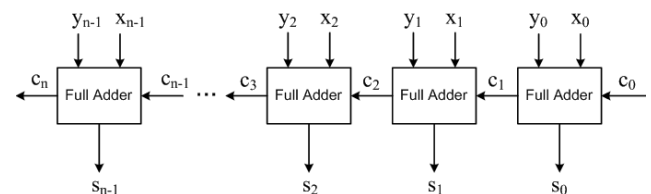


Figura 3: Sumador con propagación de acarreo

Un sumador con propagación de acarreo, también denominado sumador de acarreo serie, es aquel constituido por varios sumadores completos en los cuales el acarreo de salida se conecta a la entrada de acarreo del sumador siguiente [1].

## 1.4. Decodificador

Un decodificador o descodificador es un circuito combinacional, cuya función es inversa a la del codificador, es decir, convierte un código binario de entrada de N bits de entrada y M líneas de salida (N puede ser cualquier entero y M es un entero menor o igual a  $2^n$ ) [1]

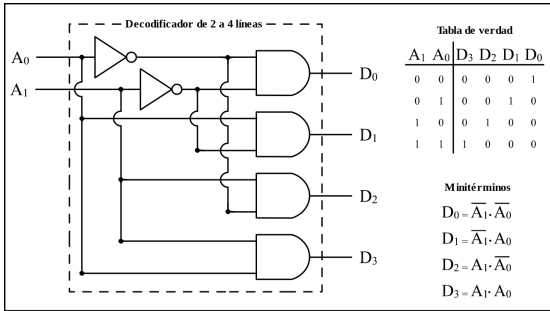


Figura 4: Decodificador de 2 a 4 bits

## 1.5. Buffer triestado

En electrónica digital, la lógica triestado permite puertos de salida con valor 0, 1 o alta impedancia.

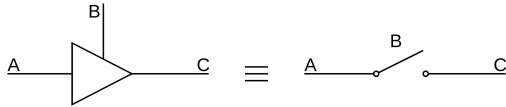


Figura 5: Buffer triestado visto como un interruptor

## 1.6. Multiplexor

Los multiplexores son circuitos combinacionales con varias entradas y una única salida de datos. Están dotados de entradas de control capaces de seleccionar una, y solo una, de las entradas de datos para permitir su transmisión desde la entrada seleccionada hacia dicha salida [1].

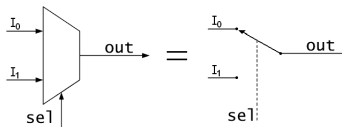


Figura 6: Esquema de un multiplexor

## 1.7. Comparador

Un comparador es un circuito electrónico, ya sea analógico o digital, capaz de comparar dos señales de entrada y variar la salida en función de cuál es mayor [1].

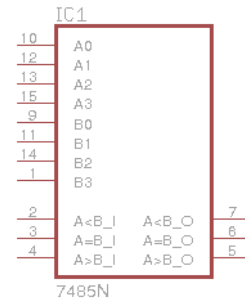


Figura 7: Comparador de 4 bits

## 2. Descripción del problema.

Se proponen dar solución a las problemáticas que se listan a continuación.

1. Implementar un sumador con propagación de acarreo generalizado.
2. Implementar un circuito capaz de convertir un número binario de cuatro bits a un número decimal que se muestre en un par de displays.
3. Diseñar un decodificador hexadecimal.

La implementación de los circuitos descritos anteriormente serán llevados en la placa de desarrollo siguiente:

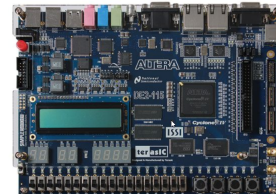


Figura 8: Altera DE2-115

Los circuitos deberán ser entregados tanto en simulación de cada uno de los circuitos como en la placa de desarrollo.

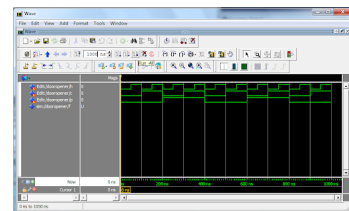


Figura 9: Entorno de simulación

### 3. Descripción de la solución.

Se propone implementar un circuito que englobe a los tres puntos descritos en la sección anterior.

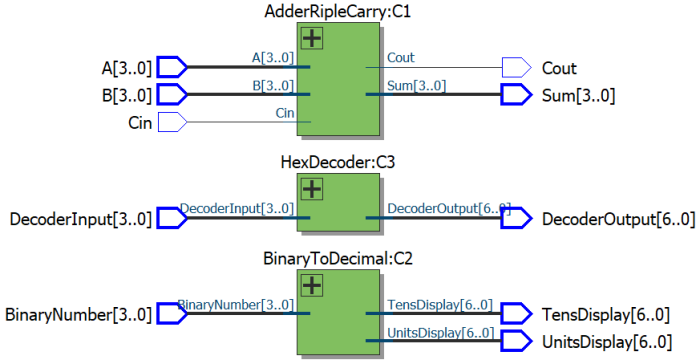


Figura 10: Diseño final

#### 3.1. Sumador con propagación de acarreo generalizado.

La implementación del sumador con propagación de acarreo se llevó a cabo a partir de un sumador completo para que después fuera llamado N veces en nuestra entidad principal, esto por medio de un ciclo y una constante que nos indique el número de bits del sumador.

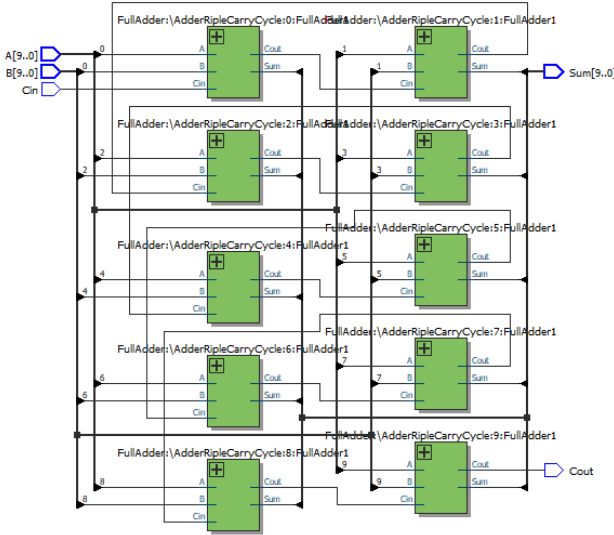


Figura 11: Sumador generalizado

En la figura anterior es posible apreciar un sumador de 10 bits el cual contiene dos entradas de 10 bits, una salida del mismo tamaño y un acarreo de salida.

Finalmente como se mencionó el sumador de la figura se basa en un sumador completo que se encuentra descrito por las siguientes ecuaciones.

$$Si = A \oplus (B \oplus Cin)$$

$$Cout = (B \wedge Cin) \oplus (A \wedge Cin) \oplus (A \wedge B)$$

Las ecuaciones anteriores pueden ser deducidas directamente de su table de verdad de un sumador completo.

A continuación mostramos el resultado de sintetizar las ecuaciones anteriores.

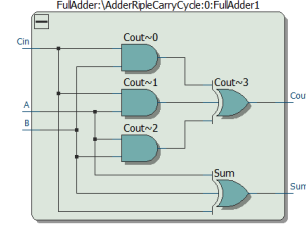


Figura 12: Sumador completo

#### 3.2. Convertidor de binario a decimal

Para éste punto se partió de un circuito previamente dado en donde se desconocían dos de los componentes del circuito.

Dichos componentes fueron inferidos después de llevar a cabo un análisis del funcionamiento del circuito, por lo que se llegó a la conclusión de que el circuito necesitaba un multiplexor que tuviera como señal de control la salida del comparador que indica si nos encontramos en el área de las unidades o decenas.

Por lo que al recibir un uno lógico éste multiplexor tendrá como salida el bus de datos que representa al cero en un display de siete segmentos y un uno en caso contrario.

También se desconocía un segundo circuito que tenía 3 salidas que iban conectados a los cuatro principales multiplexores, para éste circuito se infirió un buffer triestado que nos proporcionará un estado de alta impedancia en caso de encontrarnos en las unidades y en el caso contrario tendrá como salida los estados lógicos proporcionados por el número binario de entrada.

El circuito final quedó modelado como se muestra en la siguiente figura:

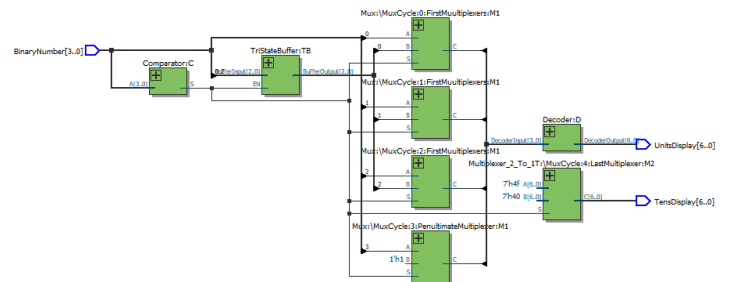


Figura 13: Convertidor de binario a decimal

### 3.3. Decodificador hexadecimal

El último punto trata de implementar un decodificador hexadecimal, para esto se hizo uso de constantes que almacenarán los códigos necesarios para desplegar cada uno de los símbolos del sistema hexadecimal.

El circuito resultante es el que se muestra a continuación:

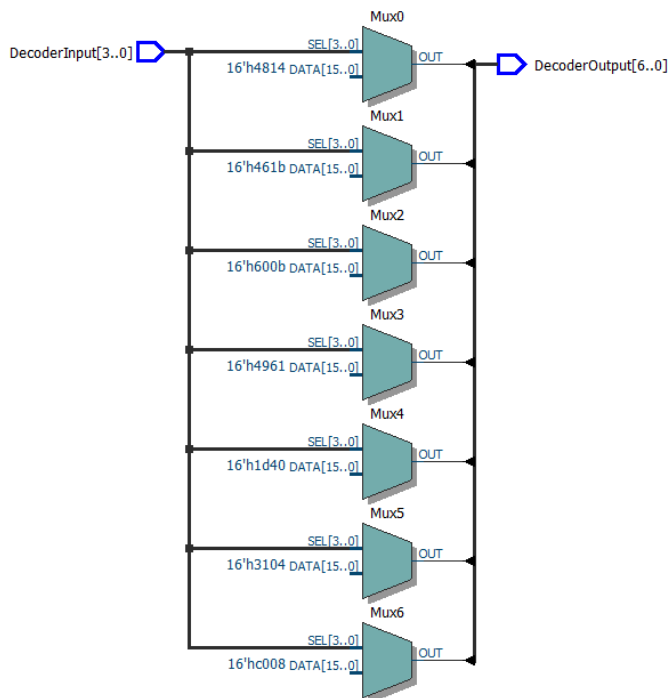


Figura 14: Decodificador hexadecimal

## 4. Implementación y Simulación

Para mayores detalles de la simulación e implementación se puede consultar en el anexo del mismo documento.

## 5. Conclusiones.

1. El conocimiento de los diferentes circuitos combinatoriales resulta de gran utilidad al tratar de dar solución a problemas cotidianos.
2. Los sumadores son muy importantes en los sistemas en los que se procesan datos numéricos.
3. Analizar un circuito combinatorial consiste en obtener la función de salida a partir de las entradas y las puertas a las que se encuentran conectadas.
4. Un decodificador nos auxilia para tener una visualización de los datos de una manera más entendible.
5. El diseño digital a nivel de componentes resulta ventajoso cuando se desea implementar circuitos más complejos.

## 6. Referencias.

- [1] Bosque Pérez, G. and Fernández Rodriguez, P. (2016). Principios de Diseño de Sistemas Digitales. 1st ed. Vitoria: Euskal, p.89.
- [2] Couch II, L. (2011). Sistemas de comunicacion digitales y analogicos. Pearson Educacion de Mexico S.A. de C.V.

## 7. Anexo

### 7.1. Adder Riple Carry

#### 7.1.1. Entidad principal

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY AdderRipleCarry IS
    GENERIC (
        AdderSize : INTEGER := 10
    );
    PORT (
        A,B : IN  STD_LOGIC_VECTOR(AdderSize - 1 DOWNT0 0);
        Cin : IN  STD_LOGIC;
        Cout: OUT STD_LOGIC;
        Sum : OUT STD_LOGIC_VECTOR(AdderSize - 1 DOWNT0 0)
    );
END AdderRipleCarry;

ARCHITECTURE AdderRipleCarryArchitecture OF AdderRipleCarry IS

    SIGNAL Carrys : STD_LOGIC_VECTOR(AdderSize DOWNT0 0);

    COMPONENT FullAdder IS
        PORT (
            A,B : IN  STD_LOGIC;
            Cin : IN  STD_LOGIC;
            Cout: OUT STD_LOGIC;
            Sum : OUT STD_LOGIC
        );
    END COMPONENT;

BEGIN
    Carrys(0)<=Cin;
    AdderRipleCarryCycle : FOR I IN 0 TO AdderSize-1 GENERATE
        FullAdder1 : FullAdder
            PORT MAP(
                A => A(I),
                B => B(I),
                Cin => Carrys(I),
                Cout => Carrys(I+1),
                Sum => Sum(I)
            );
    END GENERATE;
    Cout<=Carrys(AdderSize);

END AdderRipleCarryArchitecture;
```

### 7.1.2. Sumador completo

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY FullAdder IS

    PORT (
        A,B : IN  STD_LOGIC;
        Cin : IN  STD_LOGIC;
        Cout: OUT STD_LOGIC;
        Sum  : OUT STD_LOGIC
    );

END FullAdder;

ARCHITECTURE FullAdderArchitecture OF FullAdder IS BEGIN

Cout<= (B and Cin) xor (A and Cin) xor (A and B);
Sum  <= A xor(B xor Cin);

END FullAdderArchitecture;
```

7.1.3. Simulación de sumador cuando  $C_{in} = 1$ ,  $A = [0-15]$

B = 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

B = 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1															
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

B = 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2															
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

B = 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3															
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

B = 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4															
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

B = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5															
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

B = 6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6															
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

B = 7

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7															
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

B = 8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8															
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

B = 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9															
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

B = 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10															
11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

B = 11

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11															
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

B = 12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12															
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

B = 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13															
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

B = 14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14															
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B = 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15															
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

7.1.4. Simulación de sumador cuando  $C_{in} = 0$ ,  $A = [0-15]$

B = 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

B = 6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6															
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

B = 12

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
12															
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

B = 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

B = 7

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7															
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

B = 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
13															
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

B = 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2															
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

B = 8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8															
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

B = 14

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14															
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

B = 3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3															
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

B = 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
9															
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

B = 15

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15															
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

B = 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4															
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

B = 10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10															
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

B = 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5															
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

B = 11

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11															
11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26



## 7.2. Convertidor de binario a decimal

### 7.2.1. Entidad principal

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE WORK.MyPackage.ALL;

ENTITY BinaryToDecimal IS
    PORT (BinaryNumber          : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
          UnitsDisplay,TensDisplay : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END BinaryToDecimal;

ARCHITECTURE BinaryToDecimalArchitecture OF BinaryToDecimal IS

    SIGNAL MuxOutputs: STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL SymbolZero: STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000001";
    SIGNAL SymbolOne:  STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111001";
    SIGNAL TriStateBufferOutput : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL ComparatorOutput: STD_LOGIC;

BEGIN

    C : Comparator PORT MAP(A=>BinaryNumber,S=>ComparatorOutput);

    FOR I IN 0 TO 4 GENERATE

        IF(I < 4 and I /= 3) GENERATE
            M1 : Mux PORT MAP(A=>BinaryNumber(I),B=>TriStateBufferOutput(I),C=>MuxOutputs(I),
                               S=>ComparatorOutput);
        END GENERATE;

        IF(I = 3) GENERATE
            M1 : Mux PORT MAP(A=>BinaryNumber(I),B=>'1',C=>MuxOutputs(I),S=>ComparatorOutput);
        END GENERATE;

        IF (I >= 4) GENERATE
            M2 : Multiplexer_2_To_1T PORT MAP(A=>SymbolOne,B=>SymbolZero,C=>TensDisplay,
                                                S=>ComparatorOutput);
        END GENERATE;

    END GENERATE;

    D : Decoder PORT MAP(DecoderInput => MuxOutputs,DecoderOutput => UnitsDisplay);

    TB : TriStateBuffer PORT MAP(BufferInput=>BinaryNumber(2 DOWNTO 0),EN=>ComparatorOutput,
                                   BufferOutput=>TriStateBufferOutput);

END BinaryToDecimalArchitecture;
```

### 7.2.2. Comparador

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Comparator IS
    PORT (
        A    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        S    : OUT STD_LOGIC
    );

END Comparator;

ARCHITECTURE ComparatorArchitecture OF Comparator IS

BEGIN

    S <= '1' when A <= 9 else '0';

END ComparatorArchitecture;
```

### 7.2.3. Buffer triestado

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY TriStateBuffer IS
    PORT (
        BufferInput    : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
        EN             : IN  STD_LOGIC;
        BufferOutput    : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );

END TriStateBuffer;

ARCHITECTURE TriStateBufferArchitecture OF TriStateBuffer IS

BEGIN

    BufferOutput <= BufferInput when (NOT EN) ELSE (OTHERS => 'Z');

END TriStateBufferArchitecture;
```

#### 7.2.4. Decodificador

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Decoder IS
    PORT (
        DecoderInput : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        DecoderOutput: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END Decoder;

ARCHITECTURE DecoderArchitecture OF Decoder IS
    CONSTANT SymbolZero  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000001";
    CONSTANT SymbolOne   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111001";
    CONSTANT SymbolTwo   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0010010";
    CONSTANT SymbolThree : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000110";
    CONSTANT SymbolFour  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1001100";
    CONSTANT SymbolFive  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0100100";
    CONSTANT SymbolSix   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0100000";
    CONSTANT SymbolSeven : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0001110";
    CONSTANT SymbolEight : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000000";
    CONSTANT SymbolNine  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0001100";
    BEGIN
        DECODER: PROCESS(DecoderInput,DecoderOutput) BEGIN
            CASE DecoderInput IS
                WHEN "0000" => DecoderOutput <= SymbolZero;
                WHEN "0001" => DecoderOutput <= SymbolOne;
                WHEN "0010" => DecoderOutput <= SymbolTwo;
                WHEN "0011" => DecoderOutput <= SymbolThree;
                WHEN "0100" => DecoderOutput <= SymbolFour;
                WHEN "0101" => DecoderOutput <= SymbolFive;
                WHEN "0110" => DecoderOutput <= SymbolSix;
                WHEN "0111" => DecoderOutput <= SymbolSeven;
                WHEN "1000" => DecoderOutput <= SymbolEight;
                WHEN "1001" => DecoderOutput <= SymbolNine;
                WHEN "1010" => DecoderOutput <= SymbolZero;
                WHEN "1011" => DecoderOutput <= SymbolOne;
                WHEN "1100" => DecoderOutput <= SymbolTwo;
                WHEN "1101" => DecoderOutput <= SymbolThree;
                WHEN "1110" => DecoderOutput <= SymbolFour;
                WHEN "1111" => DecoderOutput <= SymbolFive;
                WHEN OTHERS => DecoderOutput <= (OTHERS => '1');
            END CASE;
        END PROCESS DECODER;
    END DecoderArchitecture;
END DecoderArchitecture;
```

### 7.2.5. Multiplexor 1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Mux IS
    PORT (
        A,B,S: IN  STD_LOGIC;
        C: OUT STD_LOGIC);
END Mux;

ARCHITECTURE MuxA OF Mux IS
BEGIN

    C <= A when S else B;

END MuxA;
```

### 7.2.6. Multiplexor 2

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Multiplexer_2_To_1T IS
    PORT (
        A,B,C: IN  STD_LOGIC_VECTOR(6 DOWNT0 0);
        S      : IN STD_LOGIC);
END Multiplexer_2_To_1T;

ARCHITECTURE MuxA OF Multiplexer_2_To_1T IS
BEGIN

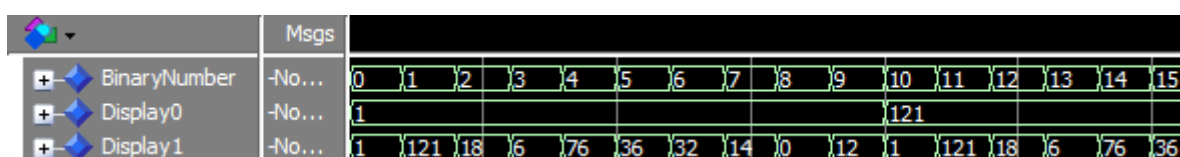
    C <= B when S else A;

END MuxA;
```

### 7.2.7. Simulación

Los números decimales mostrados en la simulación corresponden con los siguientes números en siete segmentos:

- (Num,Num7seg) = {(1,0),(121,1),(18,2),(6,3),(76,4),(36,5),(14,6),(0,8),(12,9)}



Msgs	
-No...	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-No...	1
-No...	1 121 18 6 76 36 32 14 0 12 1 121 18 6 76 36

Figura 15: Simulación final

### 7.3. Decodificador hexadecimal

#### 7.3.1. Entidad principal

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY HexDecoder IS
    PORT (
        DecoderInput : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        DecoderOutput: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
END HexDecoder;

ARCHITECTURE HexDecoderArchitecture OF HexDecoder IS

BEGIN
    DECODER: PROCESS(DecoderInput,DecoderOutput) BEGIN
        CASE DecoderInput IS
            WHEN "0000" => DecoderOutput <= SymbolZero;
            WHEN "0001" => DecoderOutput <= SymbolOne;
            WHEN "0010" => DecoderOutput <= SymbolTwo;
            WHEN "0011" => DecoderOutput <= SymbolThree;
            WHEN "0100" => DecoderOutput <= SymbolFour;
            WHEN "0101" => DecoderOutput <= SymbolFive;
            WHEN "0110" => DecoderOutput <= SymbolSix;
            WHEN "0111" => DecoderOutput <= SymbolSeven;
            WHEN "1000" => DecoderOutput <= SymbolEight;
            WHEN "1001" => DecoderOutput <= SymbolNine;
            WHEN "1010" => DecoderOutput <= SymbolA;
            WHEN "1011" => DecoderOutput <= SymbolB;
            WHEN "1100" => DecoderOutput <= SymbolC;
            WHEN "1101" => DecoderOutput <= SymbolD;
            WHEN "1110" => DecoderOutput <= SymbolE;
            WHEN "1111" => DecoderOutput <= SymbolF;
            WHEN OTHERS => DecoderOutput <= (OTHERS => '1');
        END CASE;
    END PROCESS DECODER;
END HexDecoderArchitecture;

```

#### 7.3.2. Simulación

Los números decimales mostrados en la simulación corresponden con los siguientes números en siete segmentos:

- (Num,Num7seg) = {(1,0),(121,1),(18,2),(6,3),(76,4),(36,5),(14,6),(0,8),(12,9),(8,A),(96,B),(49,C),  
(66,D),(48,E),(56,F)}

Msgs																	
/hexdecoder_vhd_t...	15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
/hexdecoder_vhd_t...	56	1	121	18	6	76	36	32	14	0	12	8	96	49	66	48	56

Figura 16: Simulación final