INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Verilog: Always, Bucles y Condicionales

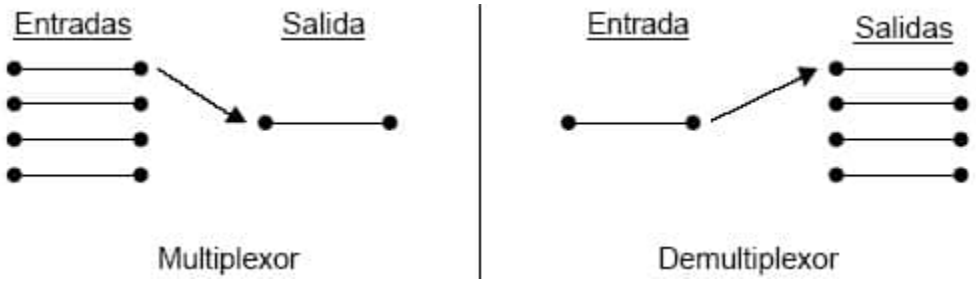
Contenido

Circuitos Combinacionales

Los circuitos combinacionales se caracterizan porque entran señales a un circuito, se procesa su información (osea que se combinan las señales) y finalmente se generan sus salidas.

Existen muchos tipos de circuitos combinacionales, estos de forma interna están conformados por una combinación de compuertas lógicas (AND, NAND, OR, NOR, XOR, XNOR o NOT), algunos de ellos son los siguientes:

- **Multiplexores (MUX):** Son circuitos combinacionales que se utilizan como selectores analógicos, ya que por medio de una perilla simbólica puede seleccionar una de sus múltiples entradas y conectarla con la única salida del circuito dependiendo de la condición que cumpla la entrada en función del selector.
- **Demultiplexores (DEMUX):** Circuito combinacional que realiza la operación inversa del MUX, es decir, toma una sola entrada y la distribuye a cualquiera de las salidas según el código de un selector, que dicta una condición para que se elija alguna de las salidas.



Este concepto en forma de circuitos digitales es el que realmente se lleva a cabo dentro del FPGA, pero en el código VHDL y Verilog se ocasiona a través de **condicionales** o **bucles**.

Always@ en Verilog

La instrucción always@ es usada siempre que quiera utilizar un **condicional** o **bucle**, cuenta con un **paréntesis llamado sensitibity list donde debo poner las entradas que voy a evaluar en el condicional o bucle** y tiene su propio begin y end. La instrucción también se usa para cuando quiera ejecutar operaciones matemáticas como una suma, resta, multiplicación o división de números binarios.

La sintaxis de always@ es la siguiente:

```
always@ (dentro de su paréntesis debo poner las entradas que vaya a usar)
// en Verilog Las entradas que vaya a usar dentro del condicional o bucle deben ir
//separadas entre ellas por una coma o la palabra reservada "or"
begin
        //Bucle o condicional que quiero usar
end
```

Operadores Relacionales

Para usar **condicionales** o **bucles** se suelen usar **operadores relacionales** para indicar las condiciones a evaluar:

- **Igual que: ==**
- **Diferente que: !=**
- **Mayor que: >**
- **Mayor o igual que: >=**
- **Menor que: <**
- **Menor o igual que: <=**

Wire y Reg en Verilog

Las palabras reservadas wire o reg existen solo en Verilog y son usadas para poder usar un valor dentro del código que solo pueda vivir durante la ejecución del programa, sin tener la necesidad de usar una entrada o una salida, ya que estas dos últimas deben estar vinculadas forzosamente a algún elemento electrónico de la tarjeta de desarrollo. Se declaran de la misma forma en la que se declara una entrada o salida, ya sea en forma de vector o de 1 bit:

```
wire nombreWire1;          --wire de 1 bit;
wire [n:0] nombreWire12; --wire tipo vector de n+1 bits

reg nombreReg1;            --reg de 1 bit;
reg [n:0] nombreReg2;      --reg tipo vector de n+1 bits
```

Si quiero asignar un valor a un wire debo usar la palabra reservada assign junto con el signo de igual (=), así como se hace la asignación de cualquier valor en Verilog. Para asignar valores a reg solo el signo igual (=).

Diferencias entre wire y reg

Su mayor diferencia es que wire sirve más que nada como una variable que no puede utilizarse en bucles o condicionales y la instrucción reg si puede servir para eso.

- wire**:** No puede usarse dentro de un always@ y debe utilizarse con la instrucción assign para asignarle un valor, además este se utiliza en diseños TLD (que se explicará luego).

```
wire [1:0] nombreWire1; --wire tipo vector de 2 bits
wire nombreWire2; --wire de 1 bit;

--Aquí puedo usar wire
assign nombreWire1 = 2'b01; --número binario 01 de 2 bits
```

```
assign nombreWire2 = entrada1 & entrada2; --compuerta AND

always@ (dentro de su paréntesis debo poner las entradas que vaya a usar)
begin
        --Aquí no puedo usar wire
end
```

- reg**:** Se puede anexar a una salida output pero no a una entrada input, no puede usar la palabra reservada assign y además se puede utilizar dentro de un always@ para crear condicionales o bucles.

```
output reg salida;
reg [1:0] nombreReg1; --reg tipo vector de 2 bits
reg nombreReg2; --reg de 1 bit;

--Aquí puedo usar reg
nombreReg1 = 2'b01; --número binario 01 de 2 bits

always@ (entrada1 or entrada2)
begin
        --Aquí también puedo usar reg
        nombreReg2 = entrada1 & entrada2; --compuerta AND
end
```

Asignación de Valores a Salidas en Verilog

En Verilog las salidas son declaradas con la palabra reservada output, pero siempre que vaya a usar una salida en un condicional o bucle, debo agregar la palabra reservada reg después de output, lo que hará este tipo de salida output reg **llamada register** es almacenar valores temporalmente, solo a ellas puedo asignar valores y a fuerza deben estar vinculadas a algún elemento electrónico de la tarjeta de desarrollo NEXYS 2, ya sea un **switch**, **botón**, **led** o un **display de 7 segmentos**.

Las salidas se declaran dentro de la declaración del módulo Verilog de la siguiente manera:

```
module nombreModulo (
    input       entrada1,       //Entrada de 1 bit
    //En Verilog siempre que vaya a usar una salida en algún condicional, lo debo declarar
    //como reg, lo que van a hacer estas salidas es almacenar valores temporalmente
    output reg salida1,         //Salida de 1 bit
    output reg salida2 //La última entrada o salida declarada no lleva una coma al final
    );
```

Si en Verilog quiero asignar un valor a una salida, ya sea el resultado de una operación lógica, el valor de una entrada a una salida, un número binario o un bit constante, debo usar el signo de igual (=), seguido del valor que quiero asignar, donde para indicarlo se sigue la nomenclatura explicada a continuación:

1. Indicando el número de bits.
2. Poniendo una apostrofe para indicar que es un número (‘).
3. Eligiendo el sistema numérico que se está usando, ya sea:
    a. binario (b)
    b. decimal (d)
    c. octal (o)
    d. hexadecimal (h)
4. Señalando el valor exacto del vector (número binario) o del bit.

- Ejemplo: Salida1 = **2'b00**;
    - La instrucción está asignando a la Salida1 dos (**2**) números (') binarios (**b**) con valor cero cero (**00**).

Condicionales en Verilog

Los **condicionales** son estructuras que analizan una **operación relacional** para ejecutar cierto código si el resultado de dicha operación lógica es cierto, sino simplemente lo ignora. Existen 3 tipos:

1.- Condicional if

Condicional if: Se usa cuando quiero analizar **una sola condición**, al final puede tener o no la instrucción else para indicar lo que pasa si no se cumple su condición, si hay más de una asignación de valor a una salida, cada if debe contar con su propio begin y end.

```
module nombreModulo (
    input       entrada1,       //Entrada de 1 bit
    //En Verilog siempre que vaya a usar una salida en algún condicional, lo debo declarar
    //como reg, lo que van a hacer estas salidas es almacenar valores temporalmente
    output reg salida1,         //Salida de 1 bit
    output reg salida2 //La última entrada o salida declarada no lleva una coma al final
    );
//Fuera del paréntesis del módulo es donde se declaran los condicionales o bucles
always@ (entrada1) //always@ tiene su propio begin y end
// en Verilog las entradas que vaya a usar dentro del condicional o bucle deben ir separadas entre ellas
//por una coma o la palabra reservada “or”.
        begin //begin perteneciente al always@ ( )
                //Condicional if con una sola asignación
                if (entrada1 == 1'b0) salida1 = 1'b0;
```

```
        //Condicional if con más de una sola asignación: Cuando esto pasa el condicional //tiene su
        propio begin y end
        if (entrada1 == 1'b1)
                begin
                        salida1 = 1'b0;
                        salida2 = 1'b1;
                end
    end //end perteneciente al always@ ( )
endmodule
```

2.- Condicional else if

Condicional else if: Se usa cuando quiero analizar **más de una sola condición relacionada con la otra**.

```
//Módulo: Dentro de éste va todo el código escrito en Verilog
module nombreModulo (
    //Dentro del paréntesis del módulo se declaran las entradas/salidas
    input       [3:0] entrada1, //Entrada tipo vector de 4 bits
    input       [2:0] entrada2, //Entrada tipo vector de 3 bits
    //En Verilog siempre que vaya a usar una salida en algún condicional, lo debo declarar
    //como reg, lo que van a hacer estas salidas es almacenar valores temporalmente
    output reg [1:0] salida1,    //Salida tipo vector de 2 bits
    output reg salida2 //La última entrada o salida declarada no lleva una coma al final
    );
//Fuera del paréntesis del módulo se indica lo que hará el programa con las entradas/salidas
//declaradas, aquí es donde se declaran las condicionales o bucles
always@ (entrada1 or entrada2) //always@ tiene su propio begin y end
// en Verilog Las entradas que vaya a usar dentro del condicional o bucle deben ir separadas entre ellas
//por una coma o la palabra reservada "or"
        begin //begin perteneciente al always@ ( )
                //Condicional else if con una sola asignación cada una
                if (entrada1 == 4'b0000) salida1 = 2'b01;
                else if (entrada1 == 4'b0001) salida1 = 2'b10;
                else if (entrada1 == 4'b0010) salida1 = 2'b11;
                else salida1 = 2'b00;
                 //La instrucción else se ejecuta cuando ninguna de las condiciones anteriores se cumple

                //Condicional if con más de una sola asignación: Cuando esto pasa el condicional
                //tiene su propio begin y end
                if (entrada2 == 3'b100)
                        begin
                                salida1 = 2'b10;
                                salida2 = 1'b0;
                        end
                else if (entrada2 == 3'b010)
                        begin
                                salida1 = 2'b01;
```

```
                    salida2 = 1'b1;
                end
            else
            //La instrucción else se ejecuta cuando ninguna de las condiciones anteriores se
            //cumple
                begin
                    salida1 = 2'b11;
                    salida2 = 1'b0;
                end
        end //end perteneciente al always@ ( )
endmodule
```

3.- Condicional case

Condicional case: Se usa cuando quiero elegir **más de una sola salida preprogramada que puede elegirse a través del valor que adopta una o varias entradas**, este número preprogramado de salidas se enumera y determina por medio de un valor llamado selector.

```
//El tamaño del número binario del selector se calcula de la siguiente manera:
//#salidas = 2 ^(#bits del selector)
//8 = 2 ^(3)
//Por lo tanto, si tengo 8 bits en mi vector de salida, el # de bits del selector debe ser de 3, osea [2:0]
//El mismo número de bits en el selector funciona también para cuando tengo 7, 6 o 5 bits en el vector de salida
//Este cálculo es para que pueda abarcar todas las salidas posibles en función al número de bits de mi vector.
module nombreModulo (
    input       [2:0] entradaSelector,      // Selector de 3 bits
    //En Verilog siempre que vaya a usar una salida en algún condicional, lo debo declarar
    //como reg, lo que van a hacer estas salidas es almacenar valores temporalmente
    output reg [4:0] salida,                //Salida tipo vector de 5 bits
    //La salida será determinada por el selector
    );
//Fuera del paréntesis del módulo se indica lo que hará el programa con las entradas/salidas
//declaradas, aquí es donde se declaran las condicionales o bucles
always@ (entradaSelector) //always@ tiene su propio begin y end
// en Verilog Las entradas que vaya a usar dentro del condicional o bucle deben ir separadas entre ellas
//por una coma o la palabra reservada "or"
        begin //begin perteneciente al always@ ( )
            //Condicional case que elije una de varias salidas a través de un selector, que puede o no ser
            //una entrada del programa, entre paréntesis se indica cuál es la variable que enumera sus
            //salidas.
            case (entradaSelector)
                valor_ entradaSelector1: salida = valorCaso1;
                valor_ entradaSelector2: salida = valorCaso2;
                valor_ entradaSelector3: salida = valorCaso3;
                …
                valor_ entradaSelector_n: salida = valorCaso_n;
                //La instrucción default se ejecuta cuando ninguno de los valores predeterminados
```

```
                    //por el condicional case es adoptado por la entrada selector.
                    default: salida = valor_de_caso_no_incluido_en_el_case;
                endcase //end perteneciente al case
        end //end perteneciente al always@ ( )
endmodule
```

Bucles en Verilog

Los **bucles** son estructuras que analizan una operación lógica para ejecutar cierto código de manera repetitiva, esto porque algunos programas tienen la necesidad de que un mismo código se ejecute varias veces, el código se ejecutará un número definido de veces si usamos el bucle for o se ejecutará un número indefinido de veces si usamos el bucle while:

1.- Bucle for

Bucle for: En Verilog para poder usar un bucle for debo declarar un tipo de dato llamado integer que simplemente es un número entero que me servirá para poder indicar las veces que se debe realizar el bucle.

```
//Módulo: Dentro de éste va todo el código escrito en Verilog
module nombreModulo (
    //Dentro del paréntesis del módulo se declaran las entradas/salidas
    input  [2:0] entrada, //Entrada tipo vector de 3 bits
    output [1:0] salida,   //Salida tipo vector de 2 bits
    );
//Fuera del paréntesis del módulo se indica lo que hará el programa con las entradas/salidas
//declaradas, aquí es donde se declaran las variables tipo integer y reg para usarse en el bucle
reg [9:0] variableRegDelBucle; //Variable del bucle for
//Esta variable numérica le indicará al bucle for cuantas veces se tiene que ejecutar, solamente se me obliga a
//declararla, su valor será asignado más adelante, dentro de los bucles del programa.
integer ciclos;

always@ (entrada) //always@ tiene su propio begin y end
        begin
                //Bucle for
                for (ciclos=0; ciclos<=2; ciclos=ciclos+1) begin
                        //La declaración del bucle for en Verilog es igual a la de cualquier otro idioma de alto
                        //nivel, indicando la variable que usará, de donde empieza a contar, de cuánto en
                        //cuanto va a contar y donde acaba su conteo.
                        variableRegDelBucle[ciclos] = 0;
                        //Este for lo que va a hacer es rellenar el vector variable variableRegDelBucle de ceros
                        //en sus posiciones de 0 a 9.
                end //end perteneciente al bucle for
        end //end perteneciente al always@ ( )
endmodule
```

2.- Bucle while

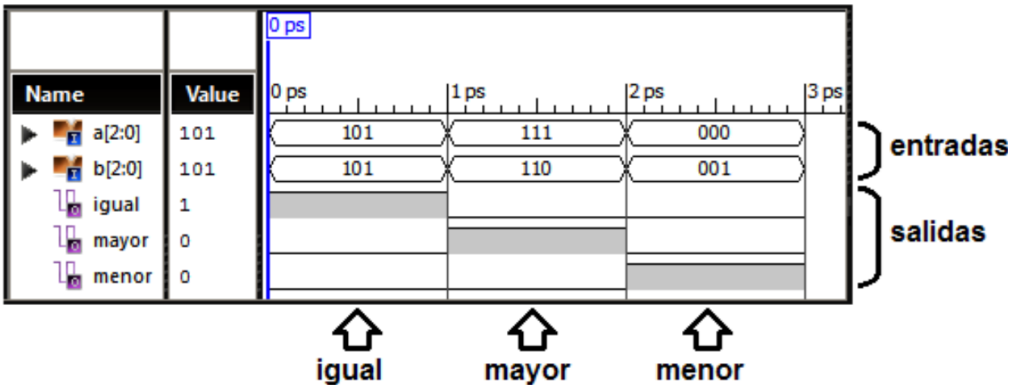
Bucle while: Este bucle es de tipo indefinido, se utiliza cuando se quiere que el programa se ponga a ejecutar una tarea de forma repetitiva sin saber si esta tendrá final o no, esto no es recomendable usarse en lenguajes VHDL ya que los bucles while tienen una naturaleza de ejecución indefinida, y la síntesis necesita conocer el número de iteraciones exacto para generar un circuito lógico equivalente.

Ejercicios de Condicionales y Bucles

A continuación, se llevarán a cabo los mismos ejercicios resueltos con los lenguajes VHDL y Verilog para que se noten las diferencias al usar ambos en una misma tarea.

1.-Comparador de 2 Números Binarios de 3 Bits - Condicional if

Se creará un programa en VHDL y Verilog que compare el valor de dos números binarios de 3 bits llamados A y B, indicando si son iguales, si A > B o si A < B, para ello se utilizará un condicional if.



Código Verilog:

```
//COMPARADOR: PARA HACER UN COMPARADOR DEBO HACER USO DE CONDICIONALES IF.
//De esta manera se ponen comentarios en Verilog.
//Verilog es un lenguaje case sensitive que distingue entre mayúsculas y minúsculas.

//Dentro de module va todo el Código de Verilog.
module comparador(
//Dentro del paréntesis del módulo se declaran las entradas y salidas que usa mi programa.
          //Las entradas se declaran con la palabra reservada input.
          //Cuando la entrada o salida se declara con corchetes es un vector de varios bits,
          //los vectores son números binarios completos que cuentan con un # determinado de bits.
    input [2:0] A,
          //El tamaño de este vector es de 2,1,0 osea de 3 bits, su bit más significativo está en la coordenada 2 y el menos
          //significativo en la coordenada 0.
    input [0:2] B,
          //El tamaño de este vector es de 0,1,2 osea de 3 bits, su bit más significativo está en la coordenada 0 y el menos
          //significativo en la coordenada 2.
          //Las salidas se declaran con la palabra reservada output.
          //En Verilog todas las salidas que vaya a usar en el condicional if debo declararlas como reg, osea Register
          //los tipos de salidas reg lo que van a hacer es almacenar valores temporalmente.
    output reg igual,  //Las salidas se declaran con la palabra reservada output.
    output reg menor,  //Cuando la entrada o salida se declara sin corchetes es de un solo bit.
    output reg mayor
    );

//CONDICIONAL IF
always@(A,B)//always se usa para poder usar los condicionales y tiene su propio begin y end.
          //always@(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
          //las entradas se pueden separar entre s por or o por comas.
          begin
```

```
        igual = 0;
        menor = 0;
        mayor = 0;
        //Es recomendable siempre inicializar mis salidas con algún valor.
                if(A == B)
                        begin
                                igual = 1; //La primera condición siempre va acompañada de un if.
                        end//Condicional igual que
                if(A > B)
                        begin
                                menor = 1;
                        end//Condicional mayor que
                if(A < B)
                        begin
                                mayor = 1;
                        end//Condicional menor que
        end//Al final del always@ se pone la instrucción end
endmodule//Al final del módulo se pone la instrucción endmodule
```

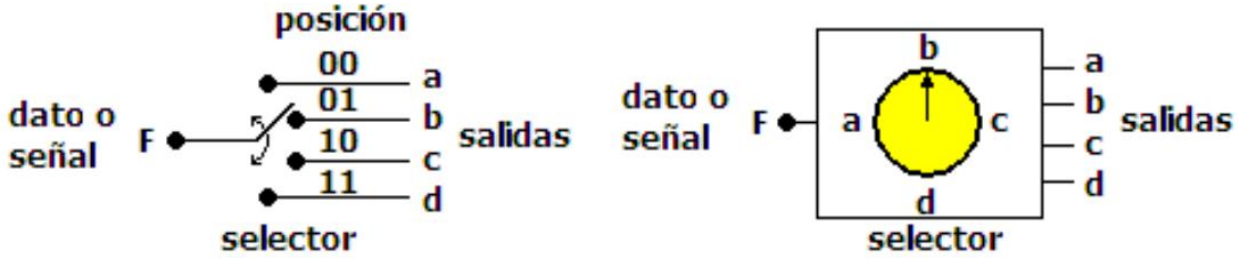
Código VHDL:

```
--Los comentarios se ponen con dos guiones seguidos
--El lenguaje VHDL no distingue entre mayúsculas y minúsculas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity comparador is
    Port ( a : in  STD_LOGIC_VECTOR (4 downto 0);
           b : in  STD_LOGIC_VECTOR (4 downto 0);
                --Entradas en forma de vector, osea número binario de 5 bits, su bit más significativo
                --esta en la coordenada 4 y el menos significativo en la coordenada 0.
           igual : out  STD_LOGIC;
           menor : out  STD_LOGIC;
           mayor : out  STD_LOGIC);
                        --Las salidas son de un bit
end comparador;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y
--end nombreArquitectura;
architecture nombreArquitectura of comparador is
begin
        --CONDICIONAL IF
        --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
        process (A, B)
                --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional).
                --las diferentes entradas se separan entre si por comas.
                begin
                        --En VHDL cuando se asigna el valor a una variable de un bit se usan comillas simples ''
                        --y cuando se quiere asignar valor a un vector se usan comillas dobles "".
                        igual <= '0';
                        mayor <= '0';
                        menor <= '0';
                        --Es recomendable siempre inicializar mis salidas con algún valor, las entradas no pueden
                        --ser inicializadas.

                        --En VHDL el contenido del condicional if se pone entre las instrucciones then y end if y
                        --su condición se pone después de la palabra reservada if sin la necesidad de estar entre
                        --paréntesis.
                        if (A = B)
                                then
                                        igual <= '1';
                                end if;--Al final de los end if debo poner punto y coma ;
                        --Condicional igual que
                        if (A > B)
                                then
                                        mayor <= '1';
                                end if;
                        --Condicional mayor que
                        if (A < B)
                                then
                                        menor <= '1';
                                end if;
                        --Condicional menor que
                end process;--Al final de end process debo poner punto y coma ;
end nombreArquitectura;--Al final del end de la arquitectura debo poner punto y coma ;
```

2.-Demultiplexores (DEMUX) 1X5 y 1X4 - Condicionales Case

Se creará un demultiplexor de 1 entrada y 2 salidas (2X1), **para ello se utilizará un bucle definido** for **del lenguaje de programación Verilog y VHDL.**

La aplicación del DEMUX es distribuir datos a través de pocas entradas. En conjunto con otros componentes se puede utilizar en sistemas de vigilancia, sistemas de seguridad, sistemas síncronos de transmisión de datos, controlador de un motor a pasos, sistemas de ahorro de energía, etc.



Código Verilog:

```
//DEMUX DE 1 ENTRADA Y 2 SALIDAS
//BUCLE FOR CON EL LENGUAJE VHDL: En VHDL para interactuar con un bucle for se necesita crear un
//tipo de dato llamado variable, este puede ser de 1 bit, un vector (número binario) o hasta puede
//ser un valor numérico entero.

//Dentro de module va todo el código de Verilog
module demuxBucleFor(
    input entrada, //Entrada de 1 bit.
    output salidaBit, //Salida de 1 bit.
    output [4:0] salidaNumBinario //Salida tipo vector de 5 bits.
    );

          //Antes del always pero dentro del módulo es donde se declaran las variables de tipo
          //reg e integer que interactúan con el bucle for.
          reg [5:0] varBucleFor;
          integer i;

          //BUCLE FOR: Bucle definido que se ejecuta varias veces.
          //always@ se usa para poder usar condicionales o bucles y tiene su propio begin y end
          always@(entrada)
          begin
                    //El bucle for se ejecutará varias veces para rellenar los bits de un vector de tipo
                    //reg con un 1 lógico, se indica el punto de partida, el límite y su avance.
                    for(i = 0; i <= 5; i = i + 1) begin
                              //Ahora accederemos a las 6 posiciones de la variable varBucleFor.

                              varBucleFor[i] = 1'b1;
                    end

                    //Reemplazo el número 111111 del vector varBucleFor por el número 010111, cambie solo sus
                    //coordenadas 5, 4 y 3.

                    varBucleFor[5:3] = 3'b010;
          end
          //A las salidas les asigno valores usando la palabra reservada assign y lo puedo hacer en
          //cualquier lugar del código
          //varBucleFor = 010111
          assign salidaNumBinario = varBucleFor[4:0];      //salidaNumBinario = 10111
          assign salidaBit = varBucleFor[5];               //salidaBit = 0
endmodule
```

Código VHDL:

```
--DEMUX DE 1 ENTRADA Y 2 SALIDAS
--BUCLE FOR CON EL LENGUAJE VHDL: En VHDL para interactuar con un bucle for se necesita crear un
--tipo de dato llamado variable, este puede ser de 1 bit, un vector (número binario) o hasta puede
--ser un valor numérico entero.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity demuxBucleFor is
    Port ( entrada : in  STD_LOGIC; --Entrada de 1 bit.
                              salidaBit : out  STD_LOGIC; --Salida de 1 bit.
```

```
                              --Salida tipo vector de 5 bits.
          salidaNumBinario : out  STD_LOGIC_VECTOR (4 downto 0));
end demuxBucleFor;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end
--nombreArquitectura;
architecture Behavioral of demuxBucleFor is
begin--Dentro de la arquitectura es donde se declaran los condicionales y bucles.
          --BUCLE FOR: Bucle definido que se ejecuta varias veces.

          --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
          process(entrada) --La entrada no se utilizará para centrarnos más en el funcionamiento del for.
          --Dentro del process pero antes de su begin es donde se declaran las variables de tipo
          --variable que interactúan con el bucle for.
          variable varBucleFor: STD_LOGIC_VECTOR (5 downto 0); --Variable tipo vector de 6 bits.

                    begin
                    --El bucle for se ejecutara varias veces para rellenar los bits de un vector de tipo
                    --variable con un 1 logico, parte de un numero inicial indicado antes de la palabra reservada
                    --"in" y llega hasta el numero indicado después de "to", tocando ese limite.
                              for i in 0 to 5 loop --Bucle que se ejecuta 6 veces.
                                        --Ahora accederemos a las 6 posiciones de la variable varBucleFor, además a las
                                        --variables se les asigna valores con el símbolo := en VHDL.
                                        varBucleFor(i) := '1';
                              end loop;

                              --Reemplazo el numero 111111 del vector varBucleFor por el numero 010111, cambie solo sus
                              --coordenadas 5, 4 y 3.

                              varBucleFor(5 downto 3):= "010";

                              --A las salidas les asigno valores usando el símbolo <=

                              --A las variables les asigno valores usando el símbolo :=
                              --varBucleFor = 010111
                              salidaNumBinario <= varBucleFor(4 downto 0);  --salidaNumBinario = 10111
                              salidaBit <= varBucleFor(5);                  --salidaBit = 0
                    end process;
end Behavioral;
```

3.-Demultiplexores (DEMUX) 1X5 y 1X4 - Condicionales Case

Se creará un demultiplexor de 1 entrada y 5 salidas (1X5), que utiliza un selector de 3 bits para elegir qué valor asignar a su salida, **para ello se utilizará un condicional** case **del lenguaje de programación Verilog.**

Se creará un demultiplexor de 1 entrada y 2 salidas (1X2), que utiliza un selector de 1 bit para elegir qué valor asignar a su salida, **para ello se utilizará un condicional** case **del lenguaje de programación VHDL.**

Código Verilog:

```
//DEMULTIPLEXOR DE 5 SALIDAS USANDO EL CONDICIONAL CASE
//Los Demultiplexores reciben una señal de entrada (el selector) y pueden dar varias señales digitales de salida
//diferentes en forma de vector, el vector es un número binario con un numero de bits constante, cada bit funciona como
//variable y puede adoptar cualquiera de los valores digitales, ya sea 1 lógico, 0 lógico o Z alta impedancia.
//Del vector puedo extraer cada bit individualmente y mandarlos por medio de un BUS al elemento electrónico que yo
//quiera.
//BUS es un conjunto de cables, en cada cable se transmiten individualmente los valores que vaya adoptando cada bit
//de la señal digital de salida (osea el numero binario) y se podrá elegir cualquiera de las señales de salida
//disponibles por medio del selector.

//El número de salidas disponibles que tengo van en función del tamaño del selector, si selector es de pocos bits
//puedo elegir pocas posibles salidas y viceversa.

//El Demultiplexor puede recibir dos señales de entrada en vez de solo el selector, la segunda entrada se llama
//enable y sirve para encender o apagar el demux, ya que, si el enable está en cero, todos los bits de la salida
//estaran en cero.

//El tamaño del número binario del selector se calcula de la siguiente manera:
//#salidas = 2 ^(#bits del selector)
//8 = 2 ^(3)
//Por lo tanto, si tengo 8 bits en mi vector de salida, el # de bits del selector debe ser de 3, osea [2:0]
//Este mismo número de bits en el selector funciona también para cuando tengo 7, 6 o 5 bits en el vector de salida
//Este cálculo es para que pueda abarcar todas las salidas posibles en función al número de bits en mi vector.

//Dentro de module va todo el Código de Verilog.
module demuxCase(
    input [2:0] selector,
    input enable,
          //En Verilog todas las salidas que se vaya a usar en los condicionales debo declararlas como reg, osea Register
          //los tipos de salidas reg lo que van a hacer es almacenar valores temporalmente.
    output reg [4:0] salida
          //La salida será determinada por el selector.
    );
```

```
        //CONDICIONAL CASE, es parecido al condicional switch case.
        always@(selector, enable)//always se usa para poder usar los condicionales, tiene su propio begin y end.
        //always@(entradas que vaya a usar dentro del case) las entradas se pueden separar entre sí por or o por comas.
        begin
                case(selector)//CASE se usa para evaluar los diferentes valores de la variable que tenga en su paréntesis
                        //El selector deber adoptar diferentes números binarios que abarquen todas las posibles salidas del
                        //demux, dependiendo del número de bits que tenga, es el número de posibilidades que puede aportar.
                        //Las llaves {} se usan para concatenar variables con bits y se usa para darle uso al enable.
                        3'b001:salida={enable, 4'b0000};      //selector con valor decimal 1 = salida 10000 de 5 bits.
                        //La instrucción 3'b001 está diciendo que si el selector tiene tres (3) números (') binarios(b) con
                        //valor cero cero uno (001) el demultiplexor asignara a la salida un vector con valor 10000 o 00000
                        //dependiendo del valor de la entrada enable.
                        3'b010:salida={1'b0, enable, 3'b000}; //selector con valor decimal 2 = salida 01000 de 5 bits.
                        3'b011:salida={2'b00, enable, 2'b00}; //selector con valor decimal 3 = salida 00100 de 5 bits.
                        3'b100:salida={3'b000, enable, 1'b0}; //selector con valor decimal 4 = salida 00010 de 5 bits.
                        3'b101:salida={4'b0000, enable};      //selector con valor decimal 5 = salida 00001 de 5 bits.
                        //Me faltan los casos para el valor decimal 0, 6 y 7 del selector.
                        //Para mi ultima condicion (o ultimas condiciones en este caso) siempre debo usar la instrucción
                        //default perteneciente al condicional case.
                        default:salida=4'bzzzz;               //selector con valor decimal 0, 6 o 7 = salida ZZZZZ de 5 bits
                        //El valor Z significa alta impedancia y lo que va a hacer es asignar un valor de voltaje que se
                        //encuentre entre el 0 lógico y el 1 lógico, por lo que en la salida no habrá nada.
                endcase
        end
        //El deMux funciona porque en el ucf voy a tomar cada bit del vector de salida como si fuera una salida individual
        //y la voy a asignar a diferentes leds, al display de 7 segmentos, a un BUS que se dirige a un motor a pasos o
        //cualquier otro dispositivo electrónico.
endmodule
```

Código VHDL:

```
--DEMULTIPLEXOR DE 2 SALIDAS USANDO EL CONDICIONAL CASE
--Los Demultiplexores reciben una señal de entrada (el selector) y pueden dar varias señales digitales de salida
--diferentes en forma de vector, el vector es un numero binario con un numero de bits constante, cada bit funciona

--como variable y puede adoptar cualquiera de los valores digitales, ya sea 1 lógico, 0 lógico o Z alta impedancia.
--Del vector puedo extraer cada bit individualmente y mandarlos por medio de un BUS al elemento electrónico que yo
--quiera.
--BUS es un conjunto de cables, en cada cable se transmitirán individualmente los valores que vaya adoptando cada bit
--de la señal digital de salida (osea el numero binario) y se podrá elegir cualquiera de las señales de salida

--disponibles por medio del selector.
--El número de salidas disponibles que tengo van en función del tamaño del selector, si selector es de pocos bits
--puedo elegir pocas posibles salidas y viceversa.

--El Demultiplexor puede recibir dos señales de entrada en vez de solo el selector, la segunda entrada se llama
--enable y sirve para encender o apagar el demux, ya que, si el enable está en cero, todos los bits de la salida
--estarán en cero.
--El tamaño del número binario del selector se calcula de la siguiente manera:
--#salidas = 2 ^(#bits del selector)
--2 = 2 ^(1)
--Por lo tanto, si quiero tener 2 salidas, el selector debe ser de 1 bit.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity demuxCase is
    Port ( selector : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR (3 downto 0));
                        --El tamaño del selector y el tamaño de la salida no dependen uno del otro
end demuxCase;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end nombreArquitectura;
architecture nombreArquitectura of demuxCase is
begin
        --CONDICIONAL CASE, es parecido al condicional switch case
        --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
        process(selector, enable)
        --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional
        --las diferentes entradas se separan entre s por comas
        begin
                case (selector) is --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable
                        --El selector deber adoptar diferentes números binarios que abarquen todas las posibles salidas del
                        --demux, dependiendo del número de bits que tenga es el número de posibilidades que puede aportar
                        --El símbolo & se usa para concatenar variables con bits y se usa para darle uso al enable
                        --En VHDL cuando se asigna el valor a una variable de un bit se usan comillas simples '' y cuando se
                        --quiere asignar valor a un vector se usan comillas dobles ""
                        when '0' => salida <= "000" & enable;        --selector con valor decimal 0 = salida 0001 de 4 bits
                        --Para la última condición siempre debo usar la instrucción when others del condicional case
                        when others => salida <= enable & "000";     --selector con valor decimal 0 = salida ZZZZ de 4 bits
                        --El tamaño del selector y el tamaño de la salida no dependen uno del otro
                end case;--Al final del end case debo poner punto y coma ;
        end process;--Al final del end process debo poner punto y coma ;
end nombreArquitectura;--Al final del end de la arquitectura debo poner punto y coma ;

--El deMux funciona porque en el ucf voy a tomar cada bit del vector de salida como si fuera una salida individual
```

```
--y la voy a asignar a diferentes leds, al display de 7 segmentos, a un BUS que se dirige a un motor a pasos o
--cualquier otro dispositivo electrónico.
```

4.-Demultiplexores (DEMUX) 1X4 y 1X9 - Condicional else if

Repite el ejercicio 2 ahora ejecutando diferentes diseños de DEMUX (1X4 y 1X9), **pero para ello se utilizará un condicionales** if **y** else if **en vez de utilizar condicionales** case.

Código Verilog:

```
//DEMULTIPLEXOR DE 4 SALIDAS USANDO EL CONDICIONAL IF
//Los Demultiplexores reciben una señal de entrada (el selector) y pueden dar varias señales digitales de salida en
//forma de vector, el vector es un numero binario con un numero de bits constante, cada bit funciona como variable y
//puede adoptar cualquiera de los valores digitales ya sea 1 lógico, 0 lógico o Z alta impedancia.
//Del vector puedo extraer cada bit individualmente y mandarlos por medio de un BUS al elemento electrónico que yo
//quiera.
//BUS es un conjunto de cables, en cada cable se transmiten individualmente los valores que vaya adoptando cada bit
//de la salida digital y se podrá elegir cualquiera de las señales de salida disponibles por medio del selector.

//El tamaño del número binario del selector se calcula de la siguiente manera:
//#salidas = 2 ^(#bits del selector)
//4 = 2 ^(2)
//Por lo tanto, si tengo 4 bits en mi vector de salida, el # de bits del selector debe ser de 2, osea [1:0]
//Este mismo número de bits en el selector funciona también para cuando tengo 7, 6 o 5 bits en el vector de salida y
//este cálculo es para que pueda abarcar todas las salidas posibles en función al número de bits en mi vector

//Dentro de module va todo el código de Verilog
module demuxIf(
          //Dentro del paréntesis de module se declaran las entradas y salidas
    //Las entradas se declaran con la palabra reservada input y si no tienen corchetes son de un bit
          input [1:0] selector,
          //En Verilog todas las salidas que vaya a usar en los condicionales debo declararlas como reg, osea Register
          //los tipos de salidas reg lo que van a hacer es almacenar valores temporalmente
          //Las salidas se declaran con la palabra reservada output y si tienen corchetes son de tipo vector, varios bits
    output reg [3:0] salida
    );

          //CONDICIONAL IF
          always@(selector)//always se usa para poder usar los condicionales y tiene su propio begin y end
          //always@(entradas que vaya a usar dentro del if) las entradas se pueden separar entre sí por or o por comas
          begin
                    if(selector==2'b01) salida = 4'b0001;       //selector con valor decimal 1 = salida 0001 de 4 bits
                    //La instrucción 2'b00 está diciendo que si el selector tiene dos(2) números(') binarios(b) con valor cero
                    //cero (00) el demultiplexor asignar a la salida un vector con valor 0001
                    else if(selector==2'b10) salida = 4'b0010; //selector con valor decimal 2 = salida 0010 de 4 bits
                    else if(selector==2'b11) salida = 4'b0100;  //selector con valor decimal 3 = salida 0100 de 4 bits
                    //Los else if sirven para cuando tengo varias condiciones diferentes en el condicional if
                    else salida=4'b1000;                       //selector con valor decimal 0 = salida 1000 de 4 bits
                    //Para la última condición (o ultimas condiciones) siempre debo usar la instrucción else, perteneciente al
                    //condicional if
                    //O también se usa para cuando ninguna de las anteriores condiciones anteriores se cumpla
          end
          //El deMux funciona porque en el ucf voy a tomar cada bit del vector de salida como si fuera una salida individual
          //y la voy a asignar a diferentes leds, al display de 7 segmentos, a un BUS que se dirige a un motor a pasos o
          //cualquier otro dispositivo electrónico.
endmodule
```

Código VHDL:

```
--DEMULTIPLEXOR DE 9 SALIDAS USANDO EL CONDICIONAL IF
--Los Demultiplexores reciben una señal de entrada (el selector) y pueden dar varias señales digitales de salida
--diferentes en forma de vector, el vector es un numero binario con un numero de bits constante, cada bit funciona

--como variable y puede adoptar cualquiera de los valores digitales, ya sea 1 lógico, 0 lógico o Z alta impedancia.
--Del vector puedo extraer cada bit individualmente y mandarlos por medio de un BUS al elemento electrónico que yo
--quiera.
--BUS es un conjunto de cables, en cada cable se transmitirán individualmente los valores que vaya adoptando cada bit
--de la señal digital de salida (osea el numero binario) y se podrá elegir cualquiera de las señales de salida

--disponibles por medio del selector.
--El número de salidas disponibles que tengo van en función del tamaño del selector, si selector es de pocos bits
--puedo elegir pocas posibles salidas y viceversa.

--El Demultiplexor puede recibir dos señales de entrada en vez de solo el selector, la segunda entrada se llama
--enable y sirve para encender o apagar el demux, ya que, si el enable está en cero, todos los bits de la salida
--estarán en cero.

--El tamaño del número binario del selector se calcula de la siguiente manera:
--#salidas = 2 ^(#bits del selector)
--16 = 2 ^(4)
```

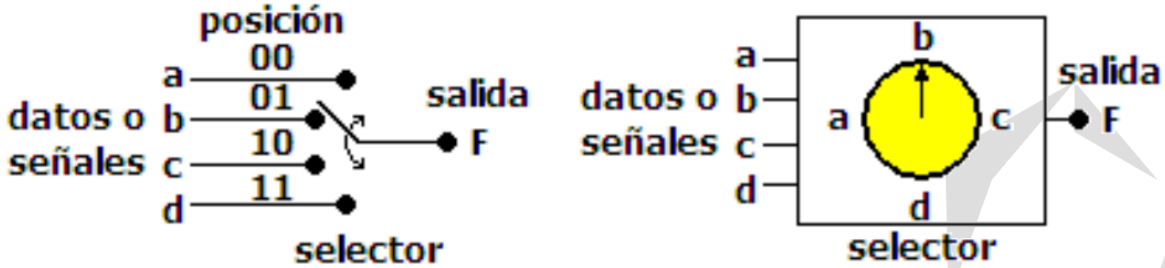
```vhdl
--Por lo tanto, si tengo 9 bits en mi vector de salida, el # de bits del selector debe ser de 4, osea [3:0]
--Este mismo número de bits en el selector funciona también para cuando tengo 10, 11, 12, 13, 14, 15 o 16 bits en el
--vector de salida. Este cálculo es para que pueda abarcar todas las salidas posibles en función al número de bits
--en mi vector
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity demuxIf is
    Port ( selector : in  STD_LOGIC_VECTOR (3 downto 0);
           enable : in  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR (8 downto 0));
                              --El tamaño del selector y el tamaño de la salida no dependen uno del otro
end demuxIf;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end nombreArquitectura;
architecture nombreArquitectura of demuxIf is
begin
          --CONDICIONAL IF
          --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
          process (selector, enable)
                    --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
                    --las diferentes entradas se separan entre sí por comas
                    begin
                              if (selector = "0001") then salida <= "00000000" & enable;
                                        --selector con valor decimal 1 = salida 000000001 de 9 bits
                                        elsif (selector = "0010") then salida <= "0000000" & enable & "0";
                                        --selector con valor decimal 2 = salida 000000010 de 9 bits
                                        elsif (selector = "0011") then salida <= "000000" & enable & "00";
                                        --selector con valor decimal 3 = salida 000000100 de 9 bits
                                        elsif (selector = "0100") then salida <= "00000" & enable & "000";
                                        --selector con valor decimal 4 = salida 000001000 de 9 bits
                                        elsif (selector = "0101") then salida <= "0000" & enable & "0000";
                                        --selector con valor decimal 5 = salida 000010000 de 9 bits
                                        elsif (selector = "0110") then salida <= "000" & enable & "00000";
                                        --selector con valor decimal 6 = salida 000100000 de 9 bits
                                        elsif (selector = "0111") then salida <= "00" & enable & "000000";
                                        --selector con valor decimal 7 = salida 001000000 de 9 bits
                                        elsif (selector = "1000") then salida <= "0" & enable & "0000000";
                                        --selector con valor decimal 8 = salida 010000000 de 9 bits
                                        elsif (selector = "1001") then salida <= enable & "00000000";
                                        --selector con valor decimal 9 = salida 100000000 de 9 bits
                                        --Para la última condición (o ultimas condiciones) siempre debo usar la instrucción else
                                        --perteneciente al condicional if
                                        else salida <= "ZZZZZZZZZ";
                                        --selector con valor decimal 0, 10, 11, 12, 13, 14, 15 o 16 = salida ZZZZZZZZZ de 9 bits
                              end if;
                              --Todos los elsif (que son los else if de VHDL) y el else deben ir dentro del if y end if;
                    end process;--Al final de end process debo poner punto y coma ;
end nombreArquitectura;--Al final del end de la arquitectura debo poner punto y coma ;
```

5.-Multiplexor (MUX) 9X1 y 6X1 - Condicional Case

Se creará un multiplexor de 6 entradas y 1 salida, que utiliza un selector de 4 bits para elegir qué valor asignar a su salida, **para ello se utilizará un condicional** case **del lenguaje de programación Verilog**.

Se creará un multiplexor de 9 entradas y 1 salida, que utiliza un selector de 4 bits para elegir qué valor asignar a su salida, **para ello se utilizará un condicional** case **del lenguaje de programación VHDL**.

La aplicación del MUX es compactar datos a través de pocas salidas. En conjunto con otros componentes se puede utilizar para cambiar los canales de un osciloscopio, sistemas de comunicación, vigilancia, conversión de conexión paralela a serial, generación de formas de onda y de funciones lógicas, enrutamiento de datos, etc.



Código Verilog:

```
//MULTIPLEXOR DE 6 ENTRADAS USANDO EL CONDICIONAL CASE
//Los multiplexores reciben varias señales de entrada (en este caso de un solo bit) y por medio de otra entrada
//llamada selector se elige una de ellas para que sea la salida, cambiando el selector se puede elegir otra
//señal para que sea la nueva salida

//El tamaño del número binario del selector se calcula de la siguiente manera:
//#entradas = 2 ^(#bits del selector)
//8 = 2 ^(3)
//Por lo tanto, si tengo 8, 7, 6 o 5 entradas, el # de bits del selector debe ser de 3, osea [2:0]

//Dentro de module va todo el código de Verilog
module muxCase(
    //Señales de entrada
          input A,
    input B,
    input C,
    input D,
    input E,
    input F,
    //El selector también es una señal de entrada, pero es de tipo vector
    input [2:0] selector,
    //En Verilog todas las salidas que vaya a usar en los condicionales debo declararlas como reg, osea Register
    //los tipos de salidas reg lo que van a hacer es almacenar valores temporalmente
    output reg salida
    //La salida ser determinada por el selector
    );

    //CONDICIONAL CASE, es parecido al condicional switch case
    always@(A or B or C or D or E or F)//always se usa para poder usar los condicionales, tiene su propio begin y end
          //always@(entradas que vaya a usar dentro del case) las entradas se pueden separar entre s por or o por comas
          begin
                    case(selector)//CASE se usa para evaluar los diferentes valores de la variable que tenga en su paréntesis
                              //El selector deber adoptar diferentes números que abarquen todas las posibles salidas del mux
                              //El numero binario del selector puede contar de uno a uno empezando en cero hasta #entradas-1
                              3'b000:salida=A;   //selector con valor decimal 0 = salida A
                              //La instrucción 3'b000 está diciendo que si el selector tiene tres(3) números(') binarios(b)
                              //con valor cero cero cero(000) el multiplexor asignar a la salida la entrada A.
                              3'b001:salida=B;   //selector con valor decimal 1 = salida B
                              3'b010:salida=E;   //selector con valor decimal 2 = salida C
                              3'b011:salida=D;   //selector con valor decimal 3 = salida D
                              3'b011:salida=E;   //selector con valor decimal 4 = salida E
                              3'b100:salida=F;   //selector con valor decimal 5 = salida F
                              //Me faltan los casos para el valor decimal 6 y 7 del selector
                              //Para la última condición (o últimas condiciones en este caso) siempre debo usar la instrucción
                              //default, perteneciente al condicional case
                              default:salida=1'bz;  //selector con valor decimal 6 o 7 = salida Z
                              //El valor Z significa alta impedancia y lo que va a hacer es asignar un valor de voltaje que se
                              //encuentre entre el 0 lógico y el 1 lógico, por lo que en la salida no habrá nada
                    endcase
          end
endmodule
```

Código VHDL:

```
--MULTIPLEXOR DE 9 ENTRADAS USANDO EL CONDICIONAL CASE
--Los multiplexores reciben varias señales de entrada (en este caso de 2 bits) y por medio de otra entrada
--llamada selector se elige una de ellas para que sea la salida, cambiando el selector se puede elegir otra
--señal para que sea la nueva salida

--El tamaño del número binario del selector se calcula de la siguiente manera:
--#entradas = 2 ^(#bits del selector)
--16 = 2 ^(4)
--Por lo tanto, si tengo 9, 10, 11, 12, 13, 14, 15 o 16 entradas, el # de bits del selector debe ser de 4, osea [3:0]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity muxCase is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           c : in  STD_LOGIC_VECTOR (1 downto 0);
           d : in  STD_LOGIC_VECTOR (1 downto 0);
           e : in  STD_LOGIC_VECTOR (1 downto 0)
           f : in  STD_LOGIC_VECTOR (1 downto 0);
           g : in  STD_LOGIC_VECTOR (1 downto 0);
           h : in  STD_LOGIC_VECTOR (1 downto 0);
           i : in  STD_LOGIC_VECTOR (1 downto 0);
                              --Senales de entrada
           selector : in  STD_LOGIC_VECTOR (3 downto 0);
                              --El selector también es una señal de entrada
           salida : out  STD_LOGIC_VECTOR (1 downto 0));
                              --La salida sera determinada por el selector
end muxCase;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end nombreArquitectura;
architecture nombreArquitectura of muxCase is
```

```
begin
        --CONDICIONAL CASE, es parecido al condicional switch case
        --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
        process(selector, a, b, c, d, e, f, g, h, i)
        --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
        --las diferentes entradas se separan entre si por comas.
        begin
                --En VHDL no se puede inicializar las entradas, por lo que estas deben ser asignadas a switches de la NEXYS 2
                --o a switches externos.
                case (selector) is --CASE se usa para evaluar los diferentes valores que pueda adoptar una variable
                        --El selector deber adoptar diferentes números binarios que abarquen todas las posibles salidas del
                        --demux, dependiendo del número de bits que tenga es el número de posibilidades que puede aportar
                        --El símbolo & se usa para concatenar variables con bits y se usa para darle uso al enable.
                        when "0001" => salida <= a;  --selector con valor decimal 1 = salida 0001 de 4 bits
                        when "0010" => salida <= b;  --selector con valor decimal 2 = salida 0001 de 4 bits
                        when "0011" => salida <= c;  --selector con valor decimal 3 = salida 0001 de 4 bits
                        when "0100" => salida <= d;  --selector con valor decimal 4 = salida 0001 de 4 bits
                        when "0101" => salida <= e;  --selector con valor decimal 5 = salida 0001 de 4 bits
                        when "0110" => salida <= f;  --selector con valor decimal 6 = salida 0001 de 4 bits
                        when "0111" => salida <= g;  --selector con valor decimal 7 = salida 0001 de 4 bits
                        when "1000" => salida <= h;  --selector con valor decimal 8 = salida 0001 de 4 bits
                        when "1001" => salida <= i;  --selector con valor decimal 9 = salida 0001 de 4 bits
                        --Para la última condición siempre debo usar la instrucción when others perteneciente al
                        --condicional case
                        --selector con valor decimal 10, 11, 12, 13, 14, 15 o 16 = salida ZZ de 2 bits
                        when others => salida <= "ZZ";
                        --El tamaño del selector y el tamaño de la salida no dependen uno del otro
                end case;--Al final del end case debo poner punto y coma ;
        end process;--Al final del end process debo poner punto y coma ;
end nombreArquitectura;--Al final del end de la arquitectura debo poner punto y coma ;
```

6.-Multiplexor (MUX) de 4 Entradas y 1 Salida (4X1) - Condicional else if

Repite el ejercicio 4 ahora ejecutando un diseño de MUX 4X1, **pero para ello se utilizará un condicionales if y else if en vez de utilizar condicionales case.**

Código Verilog:

```
//MULTIPLEXOR DE 4 ENTRADAS USANDO EL CONDICIONAL IF
//Los multiplexores reciben varias señales de entrada (en este caso de un solo bit) y por medio de otra entrada
//llamada selector se elige una de ellas para que sea la salida, cambiando el selector se puede elegir otra
//señal para que sea la nueva salida.

//El tamaño del número binario del selector se calcula de la siguiente manera:
//#entradas = 2 ^(#bits del selector)
//4 = 2 ^(2)
//Por lo tanto, si tengo 4 o 3 entradas, el # de bits del selector debe ser de 2, osea [1:0]

//Dentro de module va todo el código de Verilog
module muxIf(
    //Senales de entrada
         input A,
    input B,
    input C,
    input D,
          //El selector también es una señal de entrada, pero es de tipo vector
          input [1:0] selector,
          //En Verilog todas las salidas que vaya a usar en los condicionales debo declararlas como reg, osea Register
          //los tipos de salidas reg lo que van a hacer es almacenar valores temporalmente
    output reg salida
          //La salida ser determinada por el selector
    );

          //CONDICIONAL IF
          always@(A or B or C or D or selector)
          //always se usa para poder usar condicionales o bucles y tiene su propio begin y end
          //always@(entradas que vaya a usar dentro del if) las entradas se pueden separar entre s por or o por comas
          begin
                    if(selector==2'b00) salida=A;       //selector con valor decimal 0 = salida A
                    //La instrucción 2'b00 está diciendo que si el selector tiene dos(2) números(') binarios(b) con valor cero
                    //cero(00) el multiplexor asignar a la salida la entrada A.
                    else if(selector==2'b01) salida=B;  //selector con valor decimal 1 = salida B
                    else if(selector==2'b10) salida=C;  //selector con valor decimal 2 = salida C
                    //Los else if sirven para cuando tengo varias condiciones diferentes en el condicional if
                    else salida=D;                      //selector con valor decimal 3 = salida D
                    //Para la última condición (o ultimas condiciones) siempre debo usar la instrucción else
                    //O también se usa para cuando ninguna de las anteriores condiciones anteriores se cumpla.
          end
endmodule
```

Código VHDL:

```
--MULTIPLEXOR DE 4 ENTRADAS USANDO EL CONDICIONAL IF
--Los multiplexores reciben varias señales de entrada (en este caso de un solo bit) y por medio de otra entrada
--llamada selector se elige una de ellas para que sea la salida, cambiando el selector se puede elegir otra
--señal para que sea la nueva salida

--El tamaño del número binario del selector se calcula de la siguiente manera:
--#entradas = 2 ^(#bits del selector)
--4 = 2 ^(2)
--Por lo tanto, si tengo 4 o 3 entradas, el # de bits del selector debe ser de 2, osea [1:0]
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Las libreras IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL

--La ENTIDAD es donde declaro mis entradas y salidas
entity muxIf is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
                              --Señales de entrada
           selector : in  STD_LOGIC_VECTOR (1 downto 0);
                              --El selector también es una señal de entrada, pero es de tipo vector
           salida : out  STD_LOGIC);
                              --La salida será determinada por el selector
end muxIf;

--La ARQUITECTURA es donde declaro que harán mis entradas y salidas, tiene su propio begin y end nombreArquitectura;
architecture nombreArquitectura of muxIf is
begin
         --CONDICIONAL IF
         --process se usa para poder usar condicionales o bucles y tiene su propio begin y end process;
         process (a, b, c, d, selector)
                  --process(dentro de su paréntesis debo poner las entradas que vaya a usar en el condicional)
                  --las diferentes entradas se separan entre si por comas
                  begin
                           if (selector = "00") then salida <= a;       --selector con valor decimal 0 = salida a
                                    elsif (selector = "01") then salida <= b; --selector con valor decimal 1 = salida b
                                    elsif (selector = "10") then salida <= c; --selector con valor decimal 2 = salida c
                                    --Para la última condición (o últimas condiciones) siempre debo usar la instrucción else
                                    else salida <= d;                        --selector con valor decimal 3 = salida d
                           end if;
                           --Todos los elsif (que son los else if de VHDL) y el else deben ir dentro del if y end if;
                  end process;--Al final de end process debo poner punto y coma ;
end nombreArquitectura;--Al final del end de la arquitectura debo poner punto y coma ;
```

Operaciones Matemáticas Binarias en Verilog

La instrucción always@ no solamente sirve para crear **condicionales** o **bucles**, además se debe utilizar siempre que se quiera realizar las siguientes operaciones matemáticas sencillas con números binarios, ya que esta es una operación que se puede complicar cuando se realiza con números binarios, porque entran en juego otros conceptos como el bit de acarreo, que dice si la magnitud de la operación es positiva o negativa, entre otras cosas.

Las operaciones matemáticas sencillas con números binarios que se puede realizar con la instrucción always@ son:

- Suma binaria.
- Resta binaria.
- Multiplicación binaria.
    - La división binaria no se puede realizar de forma sencilla como las demás, para ello es necesario un proceso más complejo.

Una de las grandes ventajas de usar Verilog es que no se necesitan librerías adicionales para resolver operaciones matemáticas binarias básicas.

A continuación, se llevarán a cabo los mismos ejercicios matemáticos resueltos con los lenguajes VHDL y Verilog para que se noten las diferencias al usar ambos en una misma tarea.

Código Verilog:

```
//SUMA, RESTA, MULTIPLICACION Y DIVISION DE NUMEROS BINARIOS DE 4 BITS:
//En Verilog tengo la ventaja de que no tengo que usar libreras extra para poder hacer operaciones matemáticas.
module operacionMath(
    input [3:0] Num1,
    input [3:0] Num2,
          //Números binarios que se van a sumar, restar, multiplicar y dividir:
          //Todas los resultados de operaciones deben ser output reg porque se van a usar dentro de always@().
    output reg [4:0] Suma,
          //El resultado de la suma debe ser un bit mayor a los números sumados entre sí.
    output reg [3:0] Resta,
          //El resultado de la resta va a ser del mismo número de bits que los números restados entre sí.
    output reg [7:0] Multiplicacion
          //El resultado de la multiplicación va a tener el doble de bits que los números multiplicados entre sí.
    //output reg [3:0] Division
          //Verilog tiene problemas para hacer la división.
    );

//ALWAYS@ no solo se usa para crear condicionales o bucles, también puede servir para hacer operaciones matemáticas.
always@(Num1 or Num2)
          begin//always@() tiene su propio begin y end.
                    Suma = Num1+Num2;
                    Resta = Num1-Num2;
                    Multiplicacion = Num1*Num2;
                    //Division = Num1 / Num2; Verilog no puede sintetizar la división de forma sencilla.
                    //Todas las operaciones matemáticas binarias se pueden realizar de manera muy sencilla, exceptuando a la
                    //división, esta necesita un proceso más complejo.
          end
endmodule
```

Código VHDL:

```
--SUMA, RESTA, MULTIPLICACION Y DIVISION DE NUMEROS BINARIOS DE 4 BITS
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Estas libreras solo se declaran para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Esta librera nos permite hacer operaciones matemáticas con vectores o bits sin considerar su signo.
use IEEE.STD_LOGIC_ARITH.ALL;
--Esta librera nos permite hacer operaciones matemáticas usando process.

--Entidad: Aquí se declaran las entradas/salidas.
entity aritmetica is
    Port ( num1 : in  STD_LOGIC_VECTOR (3 downto 0);
           num2 : in  STD_LOGIC_VECTOR (3 downto 0);
           --Numeros binarios que se van a sumar, restar, multiplicar y dividir.
           suma : out  STD_LOGIC_VECTOR (4 downto 0);
           --El resultado de la suma debe ser un bit mayor a los números sumados entre sí.
           resta : out  STD_LOGIC_VECTOR (3 downto 0);
           --El resultado de la resta va a ser del mismo número de bits que los números restados entre sí.
           multiplicacion : out  STD_LOGIC_VECTOR (7 downto 0)
           --El resultado de la multiplicacion va a tener el doble de bits que los números multiplicados entre sí.
           --division : out  STD_LOGIC_VECTOR (3 downto 0)
         --VHDL tiene problemas con la librería IEEE.STD_LOGIC_ARITH.ALL; para hacer la división, por lo que
         --no se hizo.
         );
end aritmetica;

--Arquitectura: Aquí se declaran lo que harán las entradas/salidas, osea las operaciones matemáticas.
architecture operacionMatematica of aritmetica is
begin
          --PROCESS no solo se usa para crear condicionales o bucles, también junto con el uso de la biblioteca
          --IEEE.STD_LOGIC_ARITH.ALL; puede servir para hacer operaciones matemáticas con las entradas que le indique.
          process (num1, num2)
                    begin
                      --suma <= num1+num2; Si pongo así la suma me dará un error porque el vector suma es de 4 bits y los
                      --vectores num1 y num2 son de 2 bits, por lo que concateno ambos números con un bit 0.
                      suma <= ('0'&num1) + ('0'&num2);
                      --Encierro la concatenación en un paréntesis para que no se confunda el programa en la jerarquía de
                      --operaciones.
                      resta <= num1-num2;
                      multiplicacion <= num1*num2; --Con la multiplicación la librería no tiene problemas, es solo con la suma.
                      --division <= num1/num2; Así se haría la división si el programa me dejara hacerla directamente.
                      --Todas las operaciones matemáticas binarias se pueden realizar de manera muy sencilla, exceptuando a la
                      --división, esta necesita un proceso más complejo.
                    end process;
end operacionMatematica;
```