INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

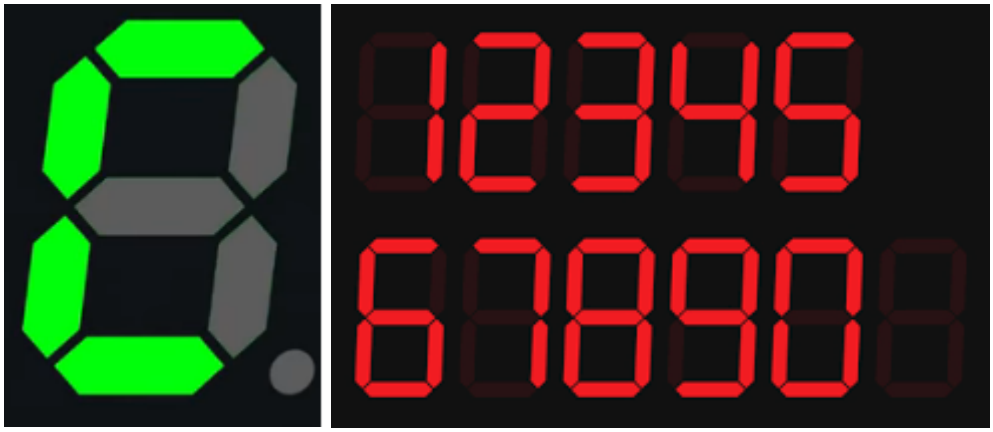
Display de 7 Segmentos: Mostrar Números Diferentes

Contenido

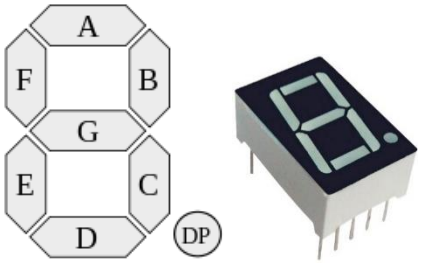
Módulo de 4 Displays de 7 Segmentos 2
TLD: Mostrar 4 Valores en los Displays 3
Diagrama TLD: 5
TLD: Conversión Binaria a Hexadecimal 8
Código Verilog: 8
Divisor de Reloj: 8
Contador/Selector: 8
MUX Decodificador BCD: 9
Módulo TLD: 10
Código UTF: 11
Código VHDL: 11
Divisor de Reloj: 11
Contador/Selector: 12
MUX Decodificador BCD: 13
Módulo TLD: 14
Código UTF: 15

Módulo de 4 Displays de 7 Segmentos

El nombre completo del display de 7 segmentos es decodificador BCD de 7 segmentos y simplemente son varios leds ordenados en una forma que puedan mostrar números del cero al nueve y un punto.



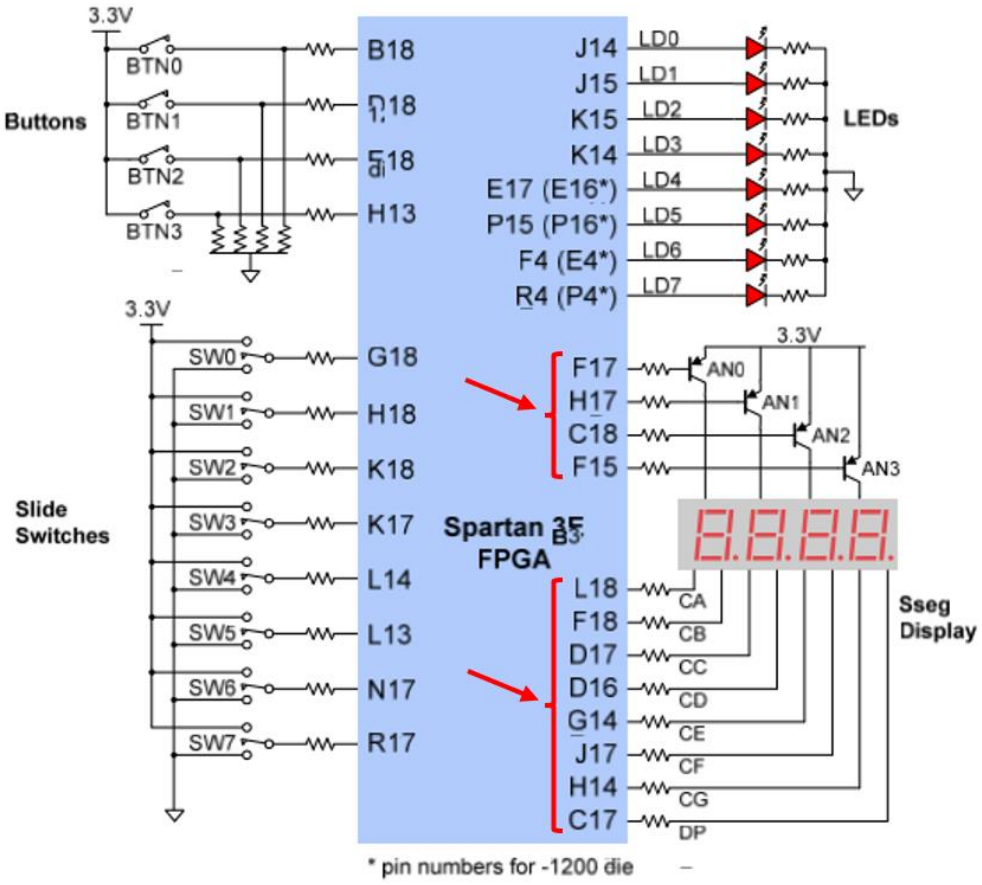
Cada uno de sus 7 segmentos tiene nombre y con segmento nos referimos a cada led que enciende cada parte del display.



Existen módulos que tienen 4 displays de 7 segmentos, pero al usar estos módulos nos encontramos con un problema, ya que no se puede mostrar dígitos diferentes en cada display a la vez, siempre se mostrará un número o letra igual en todos los displays, para resolver este problema lo que se puede hacer es prender solo uno de los 4 displays de 7 segmentos, pero aun así solo se está mostrando un dígito a la vez.

Para resolver este problema lo que se debe hacer es hacer un diseño TLD (Top Level Design) que haga uso de una señal de reloj, un contador, un multiplexor y un decodificador BCD.

En el manual de la NEXYS 2 se nos muestra que estos son los nombres y códigos que se deben incluir en el archivo ucf para que se envíen las salidas de nuestros programas a los displays de 7 segmentos y encendamos cada led A, B, C, D, E, F o G en todos los displays de 7 segmentos o que por medio de los 4 ánodos comunes encendamos individualmente cada uno de los 4 displays disponibles:



TLD: Mostrar 4 Valores en los Displays

La visualización de un solo dígito en un display de 7 segmentos resulta muy limitada. Sin embargo, existe una solución para mostrar múltiples dígitos al mismo tiempo. Esta solución implica utilizar una técnica llamada multiplexación (donde a través de una entrada y un selector, se elige cuál de las salidas disponibles en el código se mostrará).

Al poner la multiplexación en práctica, en lugar de mostrar un solo dígito a la vez, podemos aparentar que los 4 displays están encendidos, esto se logra prendiéndolos y apagándolos rápidamente de forma secuencial, mostrando un dígito diferente en cada uno. Aunque en realidad solo se están encendiendo y apagando de manera rápida, nuestros ojos retienen la imagen lo suficiente para percibir que los 4 displays están encendidos simultáneamente y mostrando dígitos diferentes.

En las televisiones sucede un efecto muy similar, donde cada unidad de la imagen, llamada píxel, se enciende individualmente. Sin embargo, la pantalla entera se refresca a una velocidad de 30 veces por

segundo. Esto permite que percibamos la imagen completa, aunque en realidad los píxeles se enciendan y apaguen uno a la vez.

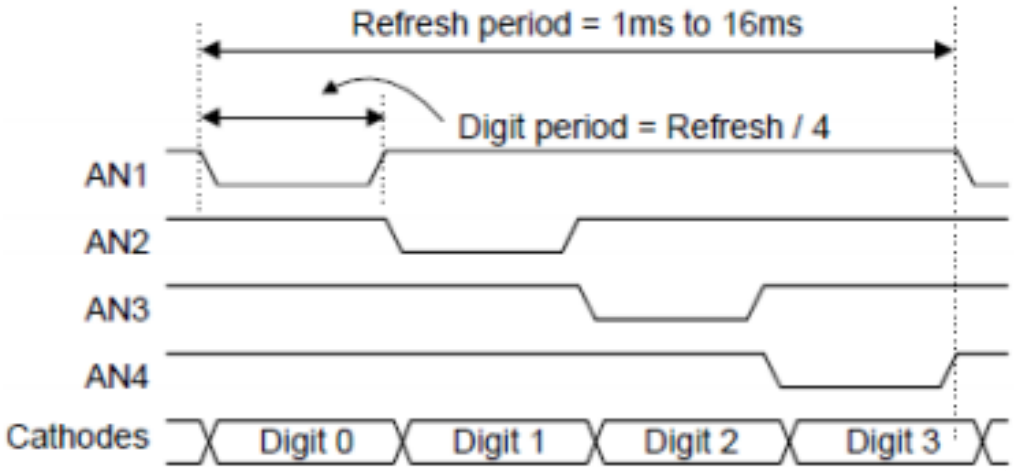
De esta manera, al utilizar la técnica de multiplexación en los módulos de 4 displays de 7 segmentos, podemos mostrar diferentes dígitos de forma simultánea, mejorando la capacidad de visualización y permitiendo la representación de múltiples números o datos al mismo tiempo.

Para poder mostrar 4 valores diferentes en los displays de 7 segmentos incluidos en la NEXYS 2, debemos codificar un diseño TLD con los siguientes módulos:

1. **Divisor de Reloj:** Primero debemos realizar un divisor de reloj para seleccionar una frecuencia de 60Hz a 1kHz, ya que los ojos humanos alcanzan a percibir frecuencias de 60Hz para abajo.
2. **Contador Selector:** Luego debemos crear un contador de 2 bits que me entregue 4 números binarios 00, 01, 10 y 11, esto para que en un módulo posterior estos puedan usarse como selector del multiplexor que elige cuál de los displays es encendido a la vez.
3. **Multiplexor:** Ya habiendo generado un selector, por medio de un MUX (Multiplexor) podemos ejecutar cualquier acción que queramos que se realice con los 4 dígitos de los displays de 7 segmentos.
    a. **Decodificador BCD**: En este caso lo que acciona el MUX es un decodificador BCD, con el cual podemos convertir cualquier número binario a su equivalente decimal, hexadecimal o el que sea y mostrar su resultado en los 4 displays de 7 segmentos.

Al inverso de la frecuencia de reloj que elijamos para que proporcione el divisor de reloj se le suele llamar **periodo de refresh** y dicta cuantas veces por segundo se correrá el programa para que muestre todos los dígitos en cada uno de los 4 displays de 7 segmentos.

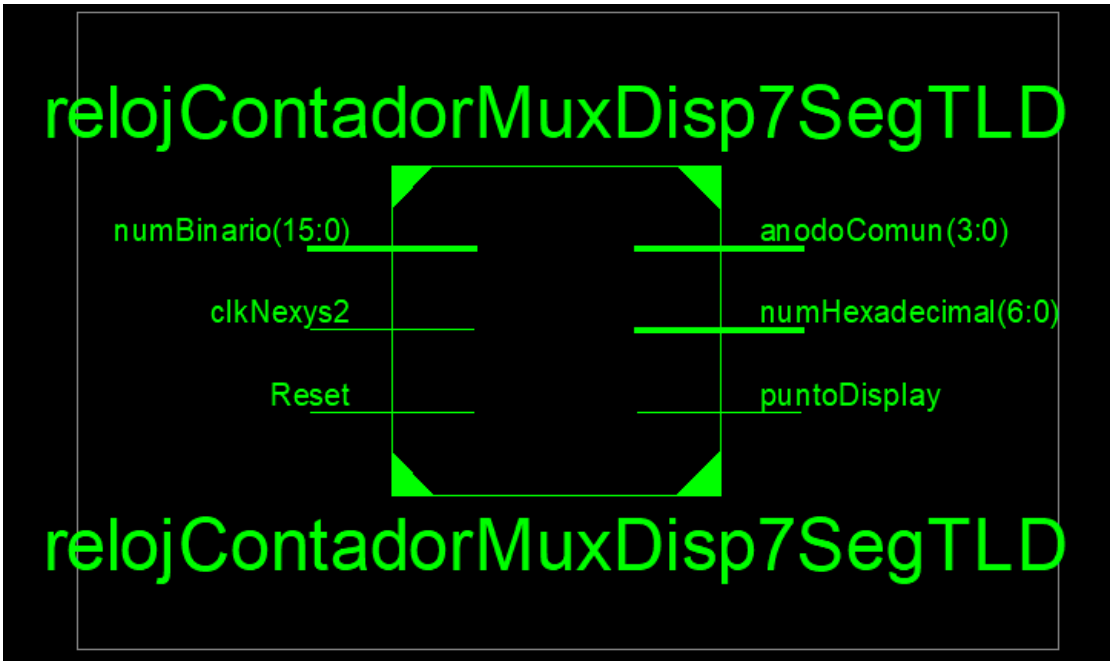
Cada ciclo de reloj, osea cada vez que se ejecute el programa en un **periodo de refresh**, se prenderán una vez los 4 displays de 7 segmentos.

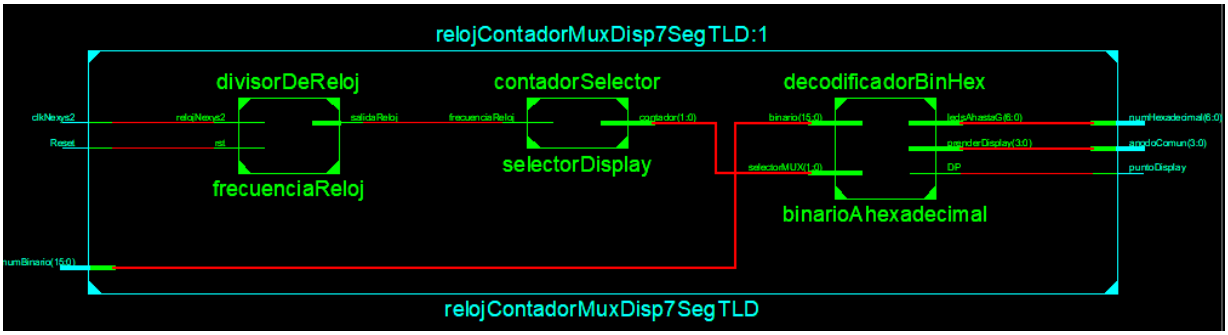


El decodificador lo que hará es traducir el código que sea al código BCD y finalmente al código que utiliza el display 7 Segmentos, este también se considera como código porque sus combinaciones de ceros y unos representan un número o letra mostrado en el display.

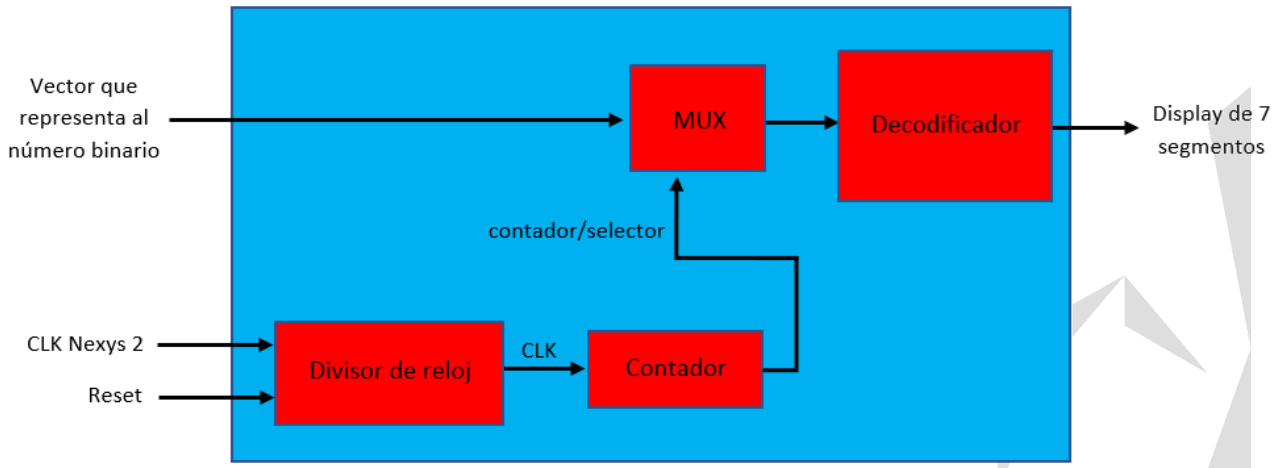
Diagrama TLD:

El diagrama de bloques RTL-Schematic que debemos codificar es el siguiente:





El diagrama de bloques global TLD (Diseño de Alto Nivel) puede tener combinado en un mismo bloque al Multiplexor y al decodificador o pueden estar separados de la siguiente manera:

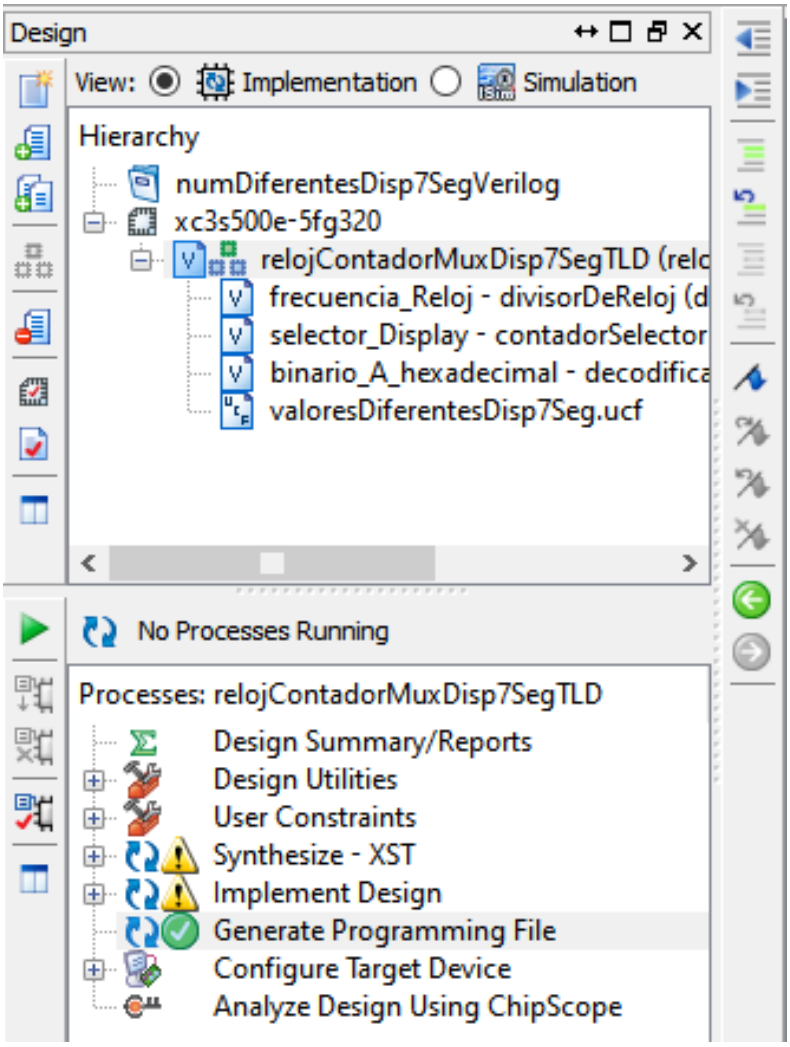


Las diferentes frecuencias del divisor de reloj que puedo elegir son las siguientes dependiendo de la coordenada del vector que elija, de la coordenada 19 para abajo son las frecuencias que debo elegir, ya que las demás si son percibidas por el ojo humano y se podrá ver como se encienden y apagan los displays:

q(i)	BASYS 2 y NEXYS 2		NEXYS 3 y NEXYS 4	
	Frecuencia (Hz)	Periodo (s)	Frecuencia (Hz)	Periodo (s)
i	50,000,000.00	0.00000002	100,000,000.00	0.00000001
0	25,000,000.00	0.00000004	50,000,000.00	0.00000002
1	12,500,000.00	0.00000008	25,000,000.00	0.00000004
2	6,250,000.00	0.00000016	12,500,000.00	0.00000008
3	3,125,000.00	0.00000032	6,250,000.00	0.00000016
4	1,562,500.00	0.00000064	3,125,000.00	0.00000032
5	781,250.00	0.00000128	1,562,500.00	0.00000064
6	390,625.00	0.00000256	781,250.00	0.00000128
7	195,312.50	0.00000512	390,625.00	0.00000256
8	97,656.25	0.00001024	195,312.50	0.00000512
9	48,828.13	0.00002048	97,656.25	0.00001024
10	24,414.06	0.00004096	48,828.13	0.00002048
11	12,207.03	0.00008192	24,414.06	0.00004096
12	6,103.52	0.00016384	12,207.03	0.00008192
13	3,051.76	0.00032768	6,103.52	0.00016384
14	1,525.88	0.00065536	3,051.76	0.00032768
15	762.94	0.00131072	1,525.88	0.00065536
16	381.47	0.00262144	762.94	0.00131072
17	190.73	0.00524288	381.47	0.00262144
18	95.37	0.01048576	190.73	0.00524288
19	47.68	0.02097152	95.37	0.01048576
20	23.84	0.04194304	47.68	0.02097152
21	11.92	0.08388608	23.84	0.04194304
22	5.96	0.16777216	11.92	0.08388608
23	2.98	0.33554432	5.96	0.16777216
24	1.49	0.67108864	2.98	0.33554432

Cuando creemos este módulo correctamente, nos aparecerá una advertencia al sintetizar el archivo bit que sube el módulo TLD a la NEXYS 2, pero no debemos preocuparnos por esto, ya que estos errores se crean debido a que el divisor de reloj ocupa solo un bit del vector que usa para crear el divisor de reloj

que da la frecuencia que quiero y porque en el decodificador que convierte de Binario a Hexadecimal se tiene un vector que siempre se queda con valor 1111 y éste nunca cambia.



Este es el Warning que indica que uno de los vectores del Decodificador se queda siempre con valor 1111:

```
Console
  Synthesizing Unit <decodificadorBinHex>.
      Related source file is "decodificadorBinHex.v".
WARNING:Xst:653 - Signal <anodoEncendido> is used but never assigned. This sourceless signal will be automatically connected to value 1111.
      Found 16x7-bit ROM for signal <ledsAhastaG>.
  .
      Summary:
          inferred   1 ROM(s).
          inferred   4 Multiplexer(s).
  Unit <decodificadorBinHex> synthesized.
```

Este es el Warning que indica que no estoy usando todas las coordenadas del vector divisorDeReloj:

```
Console
  *                      Low Level Synthesis                                 *
  ============================================================================
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_16> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_17> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_18> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_19> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_20> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_21> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_22> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
WARNING:Xst:2677 - Node <frecuencia_Reloj/divisorDeReloj_23> of sequential type is unconnected in block <relojContadorMuxDisp7SegTLD>.
```

TLD: Conversión Binaria a Hexadecimal

Código Verilog:

Divisor de Reloj:

```
//1.-DIVISOR DE RELOJ:
//Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.
module divisorDeReloj(
    input relojNexys2, //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
          input rst, //Botón de reset.
    output salidaReloj //Reloj que quiero con una frecuencia menor a 50MHz.
    );

          //REG: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo sirve
          //para almacenar y usar valores que sobrevivirán durante la ejecución del código y que además se deben usar dentro
          //de un condicional o bucle.
          reg [23:0]divisorDeReloj;
          //Este reg sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
          //dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.

          //POSEDGE: La instrucción posedge() solo puede tener una entrada o reg dentro de su paréntesis y a fuerza se debe
          //declarar en el paréntesis del always@(), ademas hace que los condicionales o bucles que estén dentro del always@()
          //se ejecuten por si solos cuando ocurra un flanco de subida en la entrada que tiene posedge() dentro de su
          //paréntesis, el flanco de subida ocurre cuando la entrada pasa de valer 0 lógico a valer 1 lógico y el hecho de
          //que la instrucción posedge() haga que el código se ejecute por sí solo significa que yo directamente no debo
          //indicarlo con una operacion logica en el paréntesis de los condicionales o bucles, si lo hago me dará error,
          //aunque si quiero que se ejecute una acción en específico cuando se dé el flanco de subida en solo una de las
          //entradas que usan posedge(), debo meter el nombre de esa entrada en el paréntesis del condicional o bucle.
          //Si uso posedge() en el paréntesis de un always@(), todas las entradas de ese always@() deben ser activadas igual
          //por un posedge().
          always@(posedge(relojNexys2), posedge(rst))
          begin
                    if(rst)
                              //En VHDL se puede declarar un numero hexadecimal para evitar poner muchos bits de un numero binario
                              //grande, pero en Verilog si hago esto el programa se confunde en el tipo de dato que está
                              //recibiendo y como consecuencia obtendremos un error.
                              divisorDeReloj = 24'b000000000000000000000000;
                              //En Verilog los valores se declaran poniendo el número de elementos que tiene, una comilla ', el
                              //sistema numérico con el que se está representado y finalmente el valor.
                    else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado Reset.
                              //No debo poner el caso cuando if(relojNexys2) porque eso ya lo está haciendo la instrucción
                              //always@(posedge(relojNexys2),...) por si sola.
                              divisorDeReloj = divisorDeReloj + 1;
          end

          //Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
          //en específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
          //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
          assign salidaReloj = divisorDeReloj[16];
          //En la tabla de frecuencias podemos ver que la coordenada 19 proporciona una frecuencia de 47.68Hz.
endmodule
```

Contador/Selector:

```
//2.-CONTADOR DE 2 BITS:
//En este código el contador es de 2 bits, esto implica que contar desde el cero hasta el 3 en forma binaria:
//00, 01, 10 y 11.
//Estos números binarios en el módulo siguiente representan al selector y el selector lo que hará es guardar en el
//reg digito (del siguiente modulo también) 4 números binarios que representen 1 digito hexadecimal, este barrido
//se debe hacer en orden y con la frecuencia dictada por el divisorDeReloj para prender individualmente cada display
//de 7 segmentos, se declara que el contador sea de 2 bits porque como en la NEXYS 2 hay 4 displays de 7 segmentos, lo
//que yo quiero es que durante un ciclo de reloj todos los displays sean encendidos una vez, mostrando cada digito del
//número hexadecimal que corresponda al número binario que está introduciendo por medio de switches, puedo hacer esto
//con 2 bits porque cuando el selector adopte los valores 00, 01, 10 y 11 prendera una vez cada digito en uno de los
//4 displays de 7 segmentos durante cada ciclo de reloj.

module contadorSelector(
    input frecuenciaReloj, //Este es el reloj proveniente del módulo divisorDeReloj
    output [1:0] contador
    );
          //REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o bucle, solo
          //sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta de desarrollo y se le
          //asignan valores con el símbolo =
          reg [1:0] conteoAscendente = 2'b00; //Se le da un valor inicial de 0 al vector.
          //2'b00 esta indicando que el valor es de dos (2) números (') binarios (b) con valor (00).

          //always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se deben
          //poner las entradas que usara y además tiene su propio begin y end.
          always@(posedge(frecuenciaReloj))
          //La instrucción posedge() hace que este condicional se ejecute solamente cuando ocurra un flanco de subida en la
          //entrada frecuenciaReloj, osea cuando pase de valer 0 lógico a valer 1 lógico, además la instrucción posedge()
          //hará que el código se ejecute por sí solo sin que yo directamente tenga que indicarlo con una operación lógica.
          begin
                    //Es un contador ascendente porque cuenta uno a uno desde cero.
```

```
                conteoAscendente = conteoAscendente + 2'b01;
        end

        //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
        assign contador = conteoAscendente;
endmodule
```

MUX Decodificador BCD:

```
//3.-DECODIFICADOR BINARIO A HEXADECIMAL CON MUX PARA MOSTRARLO EN LOS 4 DISPLAYS DE 7 SEGMENTOS A LA VEZ:
//Este código sirve para mostrar 4 números hexadecimales diferentes en los 4 displays de 7 segmentos, esto por sí solo
//no se puede lograr porque los displays de 7 segmentos solo pueden mostrar un solo número a la vez en todos los
//displays de la NEXYS 2, para lograrlo debemos usar un Multiplexor, un Divisor de Reloj y un Contador/Selector.
module decodificadorBinHex(
        input [15:0] binario,      //Entrada para meter un número binario por medio de switches.
    input [1:0] selectorMUX,       //Selector proveniente del módulo contador.
    output reg [3:0] prenderDisplay, //Vector de nodos comunes para prender cada display.
        //prenderDisplay es de tipo reg porque lo usar en un condicional if.
    output reg [6:0] ledsAhastaG,    //Vector con los leds A,B,C,D,E,F y G.
        //prenderDisplay es de tipo reg porque lo usar en un condicional case.
    output DP                      //Puntito en los displays.
    );


        //REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o bucle, solo
        //sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta de desarrollo y se le
        //asignan valores con el simbolo =
        reg [3:0] anodoEncendido = 4'b1111;
        reg [3:0] digito;


        //INICIALIZACION DE VALORES: Si quiero hacer esto fuera de cualquier always@() debo usar la instrucción assign.
        assign DP = 1'b1;//El punto de todos los displays siempre estar apagado.


        //MULTIPLEXOR: Existen muchas entradas y una salida.
        always@(selectorMUX, binario)
        begin : MUX //El nombre del always@() es MUX.
                case(selectorMUX)
                        //En el vector de entrada binario el bit más significativo se encuentra en la posición 15 y el menos
                        //significativo en la posición 0.
                        2'b00 : digito = binario [3:0];
                        //Cuando el selector valga 00, el reg de 4 bits llamado digito se llenará de los últimos 4 bits que
                        //haya en la entrada que recibe el numero binario, desde la posición 3 hasta la 0.
                        2'b01 : digito = binario [7:4];
                        //Cuando el selector valga 01, digito se llenará de los 4 penúltimos bits que haya en la entrada que
                        //recibe el numero binario, desde la posición 7 hasta la 4.
                        2'b10 : digito = binario [11:8];
                        //Cuando el selector valga 10, digito se llenará de los 4 segundos bits que haya en la entrada que
                        //recibe el numero binario, desde la posición 11 hasta la 8.
                 default : digito = binario [15:12]; //La última posibilidad de un case se pone como default.
                   //Cuando el selector valga 11, digito se llenará de los 4 primeros bits que haya en la entrada que recibe
                   //el numero binario, desde la posición 15 hasta la 12.
                endcase
        end


        //DECODIFICADOR: Transforma un código de pocos bits a uno de muchos bits, este en particular convierte de código
        //hexadecimal a código display 7 segmentos, los bits que mandamos al display de 7 segmentos se consideran como
        //código porque cada combinación de unos y ceros representa una letra o número.
        always@(digito)
        begin : HEXtoDISP7SEG //El nombre del always@() es HEXtoDISP7SEG.
                //NUMEROS HEXADECIMALES EN Verilog: 1 digito hexadecimal equivale a 4 bits en binario, esto nos podrá servir
                //para poner un numero binario grande sin tener la necesidad de poner un valor de muchos bits, como cuando
                //debo llenar un vector de puros ceros. Pero en Verilog esto no se puede aplicar porque el programa se
                //confunde con el tipo de dato que debe almacenar, por lo que crea una copia del dato y se hace un desastre,
                //por eso es mejor indicar los dígitos hexadecimales con su equivalente binario en el case.
                case(digito)
                        4'b0000 : ledsAhastaG = 7'b0000001;
                        //Pasa de 0 hexadecimal, osea 0000 a 0 en código display 7 segmentos.
                        4'b0001 : ledsAhastaG = 7'b1001111;
                        //Pasa de 1 hexadecimal, osea 0001 a 1 en código display 7 segmentos.
                        4'b0010 : ledsAhastaG = 7'b0010010;
                        //Pasa de 2 hexadecimal, osea 0010 a 2 en código display 7 segmentos.
                        4'b0011 : ledsAhastaG = 7'b0000110;
                        //Pasa de 3 hexadecimal, osea 0011 a 3 en código display 7 segmentos.
                        4'b0100 : ledsAhastaG = 7'b1001100;
                        //Pasa de 4 hexadecimal, osea 0100 a 4 en código display 7 segmentos.
                        4'b0101 : ledsAhastaG = 7'b0100100;
                        //Pasa de 5 hexadecimal, osea 0101 a 5 en código display 7 segmentos.
                        4'b0110 : ledsAhastaG = 7'b0100000;
                        //Pasa de 6 hexadecimal, osea 0110 a 6 en código display 7 segmentos.
                        4'b0111 : ledsAhastaG = 7'b0001101;
                        //Pasa de 7 hexadecimal, osea 0111 a 7 en código display 7 segmentos.
                        4'b1000 : ledsAhastaG = 7'b0000000;
                        //Pasa de 8 hexadecimal, osea 1000 a 8 en código display 7 segmentos.
                        4'b1001 : ledsAhastaG = 7'b0000100;
                        //Pasa de 9 hexadecimal, osea 1001 a 9 en código display 7 segmentos.
                        4'b1010 : ledsAhastaG = 7'b0001000;
                        //Pasa de A hexadecimal, osea 10, osea 1010 a A en código display 7 segmentos.
                        4'b1011 : ledsAhastaG = 7'b1100000;
                        //Pasa de B hexadecimal, osea 11, osea 1011 a b en código display 7 segmentos.
                        4'b1100 : ledsAhastaG = 7'b0110001;
```

```
                        //Pasa de C hexadecimal, osea 12, osea 1100 a C en código display 7 segmentos.
                        4'b1101 : ledsAhastaG = 7'b1000010;
                        //Pasa de D hexadecimal, osea 13, osea 1101 a d en código display 7 segmentos.
                        4'b1110 : ledsAhastaG = 7'b0110000;
                        //Pasa de E hexadecimal, osea 14, osea 1110 a E en código display 7 segmentos.
                        4'b1111 : ledsAhastaG = 7'b0111000;
                        //Pasa de F hexadecimal, osea 15, osea 1111 a F en código display 7 segmentos.
                endcase
        end


        //ENCENDER CADA DISPLAY DE 7 SEGMENTOS PARA APARENTAR QUE CADA UNO TIENE UN VALOR DIFERENTE:
        //Como no podemos tener desplegados 4 valores diferentes en los display de 7 segmentos, lo que se hace es que por
        //medio del selector proveniente del módulo contador con la frecuencia dictada por el divisorDeReloj, este código
        //prenda cada display individualmente tan rápido (osea con una frecuencia tan alta) que nuestros ojos no lo puedan
        //distinguir y veamos como si estuvieran prendidos al mismo tiempo con valores diferentes, cada display individual
        //se encenderá dependiendo del selector que haya entrado al programa, para luego asignará a la signal digito un valor
        //determinado dependiendo de los switches que estén activados en la entrada hexadecimal y por último prenda los
        //leds A,B,C,D,E,F o G correspondientes en todos los displays, este process lo que hará es que acceder a la salida
        //tipo vector llamada prenderDisplay y mandar un 0 lógico al nodo común del display que debe prender en ese
        //momento, para que después el selector cambie de valor y se ejecute nuevamente el código, pero ahora prendiendo
        //diferentes leds y otro display de 7 segmentos.
        always@(selectorMUX, anodoEncendido)
        begin : elegirDisplay //El nombre del always@() es elegirDisplay.
                //En un inicio todos los displays de 7 segmentos estarán apagados.
                prenderDisplay = 4'b1111;
                //Al usar el nombre de un vector y poner entre corchetes una de sus coordenadas, estoy accediendo solo a ese
                //bit del vector, en Verilog no se debe convertir manualmente lo que haya en el interior de su paréntesis a su
                //equivalente decimal, ya que el programa lo hace por sí solo.
                if(anodoEncendido[selectorMUX] == 1'b1) begin//== en Verilog sirve para ver si hay igualdad entre valores.
                        //Este condicional siempre será cierto porque el vector anodoEncendido esta inicializado con valor
                        //1111, por lo que cualquiera de sus coordenadas tiene un 1 lógico, lo importante de este
                        //condicional es que el programa escrito en Verilog convierte por sí solo al selector en su número
                        //decimal para acceder a una coordenada de su vector y la coordenada que corresponda la cambia a un
                        //0 lógico en el vector prenderDisplay, que prende cada display dependiendo de los bits que existan
                        //en el signal digito en ese momento, determinados por el
                        //valor del mismo selector.
                        prenderDisplay[selectorMUX] = 1'b0; //= en Verilog sirve para asignar un valor.
                end
        end
endmodule
```

Módulo TLD:

```
//4.-TLD (Top Level Design) relojContadorMuxDisp7SegVHDL:
//Este código sirve para unir los 3 módulos anteriores y poder realizar el decodificador binario a hexadecimal y
//mostrar en los 4 displays de 7 segmentos diferentes valores por medio del divisorDeReloj, el contadorSelector, el
//Multiplexor y el Decodificador Binario a Hexadecimal.
//Las entradas y salidas en este módulo son las que van a entrar y salir del diagrama de bloques global.
module relojContadorMuxDisp7SegTLD(
    input [15:0] numBinario,
    input Reset,
    input clkNexys2,
    output [6:0] numHexadecimal,
    output puntoDisplay,
    output [4:0] anodoComun
    );

        //WIRE: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
        //existe durante la ejecución del código, sirve para poder usar algún valor internamente cuando este no va a ser
        //utilizado dentro de un condicional o bucle y se le asignan valores con el símbolo =
        wire reloj;             //Reloj con frecuencia menor a 50MHz.
        wire [1:0] selector;    //Contador/selector para encender cada display de 7 segmentos.

        //INSTANCIAS:
        //Debo darle un nombre a cada instancia que cree, indicar el nombre del módulo que quiero instanciar y dentro de
        //un paréntesis asignarle a todas las entradas y salidas del módulo instanciado una entrada, salida o wire de este
        //modulo separadas por comas una de la otra, esto hará que lo que entre o salga del otro modulo, entre, salga o se
        //guarde en este. La sintaxis que debemos usar es la siguiente:

        //Nombre_Del_Modulo_Que_Queremos_Instanciar              NombreInstancia             (
        //                  .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),
        //                  .Nombre_De_La_Salida_Del_Modulo_Instanciado(Salida_En_Este_Modulo),

        //                  .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Salida_En_Este_Modulo),
        //                  .Nombre_De_La_Salida_Del_Modulo_Instanciado(Entrada_En_Este_Modulo),

        //                  .Nombre_De_La_Entrada_Del_Modulo_Instanciado(Signal_En_Este_Modulo),
        //                  .Nombre_De_La_Salida_Del_Modulo_Instanciado(Signal_En_Este_Modulo)
        //);

        //INSTANCIA DEL MODULO divisorDeReloj para obtener la frecuencia en la que quiero que se cree el contador/selector.
        divisorDeReloj frecuencia_Reloj(
                        .relojNexys2(clkNexys2),
                        //La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clkNexys2 de este módulo.
                        .rst(Reset),
                        //La entrada rst del divisorDeReloj se asigna a la entrada Reset de este módulo.
                        .salidaReloj(reloj)
                        //La salida salidaReloj del divisorDeReloj se asigna al wire reloj de este módulo.
                );
        //INSTANCIA DEL MODULO contadorSelector para obtener el contador/selector que prender cada display individualmente.
        contadorSelector selector_Display(
                        .frecuenciaReloj(reloj),
```

```
                        //A la entrada frecuenciaReloj del contadorSelector se le asigna el valor de la wire reloj de
                        //este módulo.
                        .contador(selector)
                        //La salida contador del contadorSelector se asigna al wire selector de este módulo.
                );
        //INSTANCIA DEL MODULO decodBinHex para cambiar de sistema numérico binario a hexadecimal, recibir el selector del
        //contador y prender cada uno de los 4 displays dependiendo del número binario que este recibiendo y con la
        //frecuencia del selector, prender y apagar los 4 displays tan rápido que para el ojo humano parezca que todos están
        //encendidos al mismo tiempo con valores diferentes.
        decodificadorBinHex binario_A_hexadecimal(
                        .binario(numBinario),
                        //La entrada binario de decodificadorBinHex se asigna a la entrada numBinario de este módulo.
                        .selectorMUX(selector),
                        //A la entrada selectorMUX de decodificadorBinHex se le asigna el valor de la wire selector de
                        //este módulo.
                        .prenderDisplay(anodoComun),
                        //La salida anodoComun de decodificadorBinHex se asigna a la salida anodoComun de este módulo.
                        .ledsAhastaG(numHexadecimal),
                        //La salida ledsAhastaG de decodificadorBinHex se asigna a la salida numHexadecimal de este módulo.
                        .DP(puntoDisplay)
                        //La salida DP de decodBinHex se asigna a la salida puntoDislpay de este módulo.
                );
endmodule
```

Código UTF:

```
//ENTRADAS Y SALIDAS DEL MODULO TDL:
//El diseño TLD sirve para hacer la conversión binario -> hexadecimal y mostrarlo como números diferentes en los
//4 displays de 7 segmentos de la placa NEXYS 2 como si estos tuvieran valores diferentes.

//ENTRADAS DEL MODULO decodificadorBinHex
//SWITCHES DE LA NEXYS 2
net "numBinario[0]" loc = "G18";//Bit 1 (menos significativo) del número binario conectado al switch SW0.
net "numBinario[1]" loc = "H18";//Bit 2 del número binario conectado al switch SW1.
net "numBinario[2]" loc = "K18";//Bit 3 del número binario conectado al switch SW2.
net "numBinario[3]" loc = "K17";//Bit 4 del número binario conectado al switch SW3.
net "numBinario[4]" loc = "L14";//Bit 5 del número binario conectado al switch SW4.
net "numBinario[5]" loc = "L13";//Bit 6 del número binario conectado al switch SW5.
net "numBinario[6]" loc = "N17";//Bit 7 del número binario conectado al switch SW6.
net "numBinario[7]" loc = "R17";//Bit 8 del número binario conectado al switch SW7.
//SWITCHES EXTERNOS CONECTADOS A LOS PUERTOS JA
net "numBinario[8]" loc = "L15";//Bit 9 del número binario conectado al puerto JA1 perteneciente a los puertos JA.
net "numBinario[9]" loc = "K12";//Bit 10 del número binario conectado al puerto JA2 perteneciente a los puertos JA.
net "numBinario[10]" loc = "L17";//Bit 11 del número binario conectado al puerto JA3 perteneciente a los puertos JA.
net "numBinario[11]" loc = "M15";//Bit 12 del número binario conectado al puerto JA4 perteneciente a los puertos JA.
//Los puertos JA5 y JA11 están conectados a tierra y JA6 y JA12 están conectados a Vcc = 3.3[V].
net "numBinario[12]" loc = "K13";//Bit 13 del número binario conectado al puerto JA7 perteneciente a los puertos JA.
net "numBinario[13]" loc = "L16";//Bit 14 del número binario conectado al puerto JA8 perteneciente a los puertos JA.
net "numBinario[14]" loc = "M14";//Bit 15 del número binario conectado al puerto JA9 perteneciente a los puertos JA.
net "numBinario[15]" loc = "M16";//Bit 16 del número binario conectado al puerto JA10 perteneciente a los puertos JA.


//ENTRADAS DEL MODULO divisorDeReloj
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "Reset" loc = "H13";//El botón de Reset está conectado al push button BTN3.


//SALIDAS DEL MODULO decodificadorBinHex
//Leds A,B,C,D,E,F y G del display de 7 segmentos.
net "numHexadecimal[6]" loc = "L18";//1er bit del vector conectado al puerto CA (led A) del display de 7 segmentos.
net "numHexadecimal[5]" loc = "F18";//2do bit del vector conectado al puerto CB (led B) del display de 7 segmentos.
net "numHexadecimal[4]" loc = "D17";//3er bit del vector conectado al puerto CC (led C) del display de 7 segmentos.
net "numHexadecimal[3]" loc = "D16";//4to bit del vector conectado al puerto CD (led D) del display de 7 segmentos.
net "numHexadecimal[2]" loc = "G14";//5to bit del vector conectado al puerto CE (led E) del display de 7 segmentos.
net "numHexadecimal[1]" loc = "J17";//6to bit del vector conectado al puerto CF (led F) del display de 7 segmentos.
net "numHexadecimal[0]" loc = "H14";//7mo bit del vector conectado al puerto CG (led G) del display de 7 segmentos.
//Punto del display de 7 segmentos.
net "puntoDisplay" LOC = "C17";//Conectado al puerto DP (punto) del display de 7 segmentos.


//SALIDAS DEL MODULO decodificadorBinHex
//Para que el orden se respete debo meter conectar el bit más significativo al nodo AN3 y el bit menos significativo
//al nodo AN0.
net "anodoComun[0]" loc = "F17";
//Bit menos significativo conectado al puerto AN0, 1er display de 7 segmentos de los 4 disponibles.
net "anodoComun[1]" loc = "H17";
//Conectado al puerto AN1, 2do display de 7 segmentos de los 4 disponibles.
net "anodoComun[2]" loc = "C18";
//Conectado al puerto AN2, 3er display de 7 segmentos de los 4 disponibles.
net "anodoComun[3]" loc = "F15";
//Bit más significativo conectado al puerto AN3, 4to display de 7 segmentos de los 4 disponibles.
```

Código VHDL:

Divisor de Reloj:

```
--1.-DIVISOR DE RELOJ:
--Este proceso sirve para dictarle al reloj en que frecuencia quiero que opere.
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.

entity divisorDeReloj is
    Port ( relojNexys2 : in  STD_LOGIC;   --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
           rst : in  STD_LOGIC;           --Botón de reset.
           salidaReloj : out  STD_LOGIC); --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí se hará el divisor de reloj haciendo uso de una signal y condicionales if.
architecture frecuenciaNueva of divisorDeReloj is
--SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
--existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
--arquitectura y antes de su begin, se le asignan valores con el símbolo :=
signal divisorDeReloj : STD_LOGIC_VECTOR (23 downto 0);
--Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas en la tabla del divisor de reloj
--dependiendo de la coordenada que elijamos tomar del vector para asignársela a la salida.
begin
        process(relojNexys2, rst)
        begin
                if(rst='1') then--Cuando el botón Reset sea presionado valdrá 1 lógico y el divisor de reloj se reiniciará.
                        --NUMEROS HEXADECIMALES EN VHDL: 1 digito hexadecimal equivale a 4 bits en binario, esto nos puede
                        --servir para poner un numero binario grande sin tener la necesidad de poner un valor de muchos
                        --bits, como cuando debo llenar un vector de puros ceros, declaro un numero hexadecimal poniendo
                        --X"numero hexadecimal"
                        divisorDeReloj <= X"000000";
                        --Poner X"000000" equivale a poner "00000000000000000000000".
                elsif(rising_edge(relojNexys2)) then
                        --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco
                        --de subida en la señal de reloj clkNexys2 proveniente de la NEXYS 2.
                        divisorDeReloj <= divisorDeReloj + 1;--Esto crea al divisor de reloj.
                end if;
        end process;

        --Debo asignar el contenido de una coordenada de la signal divisorDeReloj a salidaReloj para obtener una frecuencia
        --en especifico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
        salidaReloj <= divisorDeReloj(17);--En la tabla se ve que la coordenada 19 proporciona una frecuencia de 47.68Hz.

end frecuenciaNueva;
```

Contador/Selector:

```vhdl
--2.-CONTADOR DE 2 BITS:
--En este código el contador es de 2 bits, esto implica que contara desde cero hasta 3 en forma binaria:
--00, 01, 10 y 11.
--Estos números binarios en el módulo siguiente representaran al selector y el selector lo que hará es guardar en la
--signal digito (del siguiente modulo también) 4 números binarios que representen 1 digito hexadecimal, este barrido
--se debe hacer en orden y con la frecuencia dictada por el divisorDeReloj para prender individualmente cada display
--de 7 segmentos.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías que sirven solamente para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: En la entidad se declara que el contador sea de 2 bits porque como en la NEXYS 2 hay 4 displays de 7
--segmentos, lo que yo quiero es que durante un ciclo de reloj todos los displays sean encendidos una vez, mostrando
--cada digito del número hexadecimal que corresponda al número binario que se está introduciendo por medio de switches,
--puedo hacer esto con 2 bits porque cuando el selector adopte los valores 00, 01, 10 y 11 prendera una vez cada
--digito en uno de los 4 displays de 7 segmentos durante cada ciclo de reloj.
entity contadorSelector is
    Port ( frecuenciaReloj : in  STD_LOGIC;--Este es el reloj proveniente del módulo divisorDeReloj.
           contador : out  STD_LOGIC_VECTOR (1 downto 0));
end contadorSelector;

architecture contador2Bits of contadorSelector is
--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del programa, no
--está conectada a ningún puerto de la tarjeta de desarrollo y se le asignan valores con el símbolo :=
signal conteoAscendente : STD_LOGIC_VECTOR (1 downto 0) := "00";--Se le da un valor inicial de 0 al vector.
begin
        process(frecuenciaReloj)
        begin
                --La instruccion rising_edge indica que cada que ocurra un flanco de subida en el reloj, se le sumara un 1
                --binario al valor que tenia previamente el vector conteoAscendente, esto hará que se cree la secuencia
                --00, 01, 10 y 11 en el selector, específicamente en ese orden.
                if(rising_edge(frecuenciaReloj)) then
                        conteoAscendente <= conteoAscendente + "01";
                        --El conteo solito volverá a ser 00 cuando se supere el valor 11 en el vector conteoAscendente
                        --de 2 bits.
                end if;
        end process;

        --Se usa una signal en vez de hacer el contador directo con la salida contador porque a las salidas solo se
        --les puede asignar un valor, no se les puede leer.
        contador <= conteoAscendente;

end contador2Bits;
```

MUX Decodificador BCD:

```
--3.-DECODIFICADOR BINARIO A HEXADECIMAL CON MUX PARA MOSTRARLO EN LOS 4 DISPLAYS DE 7 SEGMENTOS A LA VEZ:
--Este código sirve para mostrar 4 números hexadecimales diferentes en los 4 displays de 7 segmentos, esto por sí solo
--no se puede lograr porque los displays de 7 segmentos solo pueden mostrar un solo número a la vez en todos los
--displays de la NEXYS 2, para lograrlo debemos usar un Multiplexor, un Divisor de Reloj y un Contador/Selector.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librerías declaradas para poder hacer operaciones matemáticas sin considerar el signo o para hacer conversiones de
--binario a integer (osea de binario a decimal).

--Entidad: Voy a declarar como entrada un vector que me permitirá introducir el numero binario por medio de switches
--y un selector proveniente del módulo contador que me servirá para poder mostrar un numero a la vez en solo uno de
--los 4 displays de 7 segmentos, esto se debe hacer así porque en los displays de 7 segmentos solo se puede mostrar
--un numero a la vez, no se pueden mostrar números diferentes en cada display. Las salidas declaradas son los nodos
--comunes que activan cada display con un 0 lógico y los leds A,B,C,D,E,F y G de cada display que también se activan
--con un 0 lógico para mostrar letras o números, se declara aparte el DP, osea el led que prende el puntito en todos
--los displays de 7 segmentos.

entity decodificadorBinHex is
    Port ( binario : in  STD_LOGIC_VECTOR (15 downto 0);--Entrada para meter un numero binario por medio de switches.
           selectorMUX : in  STD_LOGIC_VECTOR (1 downto 0);--Selector proveniente del módulo contador.
           prenderDisplay : out  STD_LOGIC_VECTOR (3 downto 0);--Vector de nodos comunes para prender cada display.
           ledsAhastaG : out  STD_LOGIC_VECTOR (6 downto 0);--Vector con los leds A,B,C,D,E,F y G.
           DP : out  STD_LOGIC);--Puntito en los displays.
end decodificadorBinHex;

--Arquitectura: Aquí se va a hacer un multiplexor que al recibir el selector del módulo contadorSelector, me permita
--mostrar un numero en solo uno de los displays de 7 segmentos.
architecture binarioAhexadecimal of decodificadorBinHex is
--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del programa, no
--está conectada a ningún puerto de la tarjeta de desarrollo y se le asignan valores con el símbolo :=
signal anodoEncendido : STD_LOGIC_VECTOR (3 downto 0) := "1111";--En un inicio ningún display esta encendido.
--Este vector me servir para prender cada uno de los displays, dependiendo de si el numero binario tiene solo
--unidades, decenas o centenas y así poder mostrar su equivalente correspondiente en hexadecimal.
signal digito : STD_LOGIC_VECTOR (3 downto 0);
--El vector digito me permitirá realizar la conversión Binario -> Hexadecimal.
begin
	--INICIALIZACION DE VALORES
	DP <= '1';--El punto siempre estará apagado.

	--MULTIPLEXOR: Existen muchas entradas y una salida.
	--LECTURA DE NUMEROS BINARIOS PARA INTRODUCURLOS EN SU DIGITO HEXADECIMAL CORRESPONDIENTE.
	MUX : process(selectorMUX, binario)
	begin
		case(selectorMUX) is
		--En el vector de entrada binario el bit mas significativo se encuentra en la posición 15 y el menos
		--significativo en la posición 0.
			when "00" => digito <= binario(3 downto 0);
			--Cuando el selector valga 00, la signal de 4 bits llamada digito se llenará de los últimos 4 bits
			--que haya en la entrada que recibe el numero binario, desde la posición 3 hasta la 0.
			when "01" => digito <= binario(7 downto 4);
			--Cuando el selector valga 01, digito se llenará de los 4 penúltimos bits que haya en la entrada que
			--recibe el numero binario, desde la posición 7 hasta la 4.
			when "10" => digito <= binario(11 downto 8);
			--Cuando el selector valga 10, digito se llenará de los 4 segundos bits que haya en la entrada que
			--recibe el numero binario, desde la posicion 11 hasta la 8.
			when others => digito <= binario(15 downto 12);
			--La última posibilidad de un case se pone como when others.
			--Cuando el selector valga 11, digito se llenará de los 4 primeros bits que haya en la entrada que
			--recibe el numero binario, desde la posición 15 hasta la 12.
		end case;
	end process MUX;

	--DECODIFICADOR: Transforma un código de pocos bits a uno de muchos bits, este en particular convierte de código
	--hexadecimal a código display 7 segmentos, los bits que mandamos al display de 7 segmentos se consideran como
	--código porque cada combinación de unos y ceros representa una letra o número.
	HEXtoDISP7SEG : process(digito)
	begin
		--NUMEROS HEXADECIMALES EN VHDL: 1 digito hexadecimal equivale a 4 bits en binario, esto nos puede servir para
		--poner un numero binario grande sin tener la necesidad de poner un valor de muchos bits en VHDL, como cuando
		--debo llenar un vector de puros ceros, declaro un número hexadecimal poniendo X"numero hexadecimal".
		case(digito) is
			when X"0" => ledsAhastaG <= "0000001"; --Pasa de X"0", osea 0000 a 0 en código display 7 segmentos.
			when X"1" => ledsAhastaG <= "1001111"; --Pasa de X"1", osea 0001 a 1 en código display 7 segmentos.
			when X"2" => ledsAhastaG <= "0010010"; --Pasa de X"2", osea 0010 a 2 en código display 7 segmentos.
			when X"3" => ledsAhastaG <= "0000110"; --Pasa de X"3", osea 0011 a 3 en código display 7 segmentos.
			when X"4" => ledsAhastaG <= "1001100"; --Pasa de X"4", osea 0100 a 4 en código display 7 segmentos.
			when X"5" => ledsAhastaG <= "0100100"; --Pasa de X"5", osea 0101 a 5 en código display 7 segmentos.
			when X"6" => ledsAhastaG <= "0100000"; --Pasa de X"6", osea 0110 a 6 en código display 7 segmentos.
			when X"7" => ledsAhastaG <= "0001101"; --Pasa de X"7", osea 0111 a 7 en código display 7 segmentos.
			when X"8" => ledsAhastaG <= "0000000"; --Pasa de X"8", osea 1000 a 8 en código display 7 segmentos.
			when X"9" => ledsAhastaG <= "0000100"; --Pasa de X"9", osea 1001 a 9 en código display 7 segmentos.
			when X"A" => ledsAhastaG <= "0001000"; --Pasa de X"A", osea 10, osea 1010 a A en código display.
			when X"B" => ledsAhastaG <= "1100000"; --Pasa de X"B", osea 11, osea 1011 a b en codigo display.
			when X"C" => ledsAhastaG <= "0110001"; --Pasa de X"C", osea 12, osea 1100 a C en codigo display.
			when X"D" => ledsAhastaG <= "1000010"; --Pasa de X"D", osea 13, osea 1101 a d en codigo display.
			when X"E" => ledsAhastaG <= "0110000"; --Pasa de X"E", osea 14, osea 1110 a E en codigo display.
			when others => ledsAhastaG <= "0111000";
			--Pasa de X"F", osea 15, osea 1111 a F en código display 7 segmentos.
```

```
                end case;
        end process HEXtoDISP7SEG;

        --ENCENDER CADA DISPLAY DE 7 SEGMENTOS PARA APARENTAR QUE CADA UNO TIENE UN VALOR DIFERENTE:
        --Como no podemos tener desplegados 4 valores diferentes en los display de 7 segmentos, lo que se hace es que por
        --medio del selector proveniente del módulo contador con la frecuencia dictada por el divisorDeReloj, este código
        --prenda cada display individualmente tan rapido (osea con una frecuencia tan alta) que nuestros ojos no lo puedan
        --distinguir y veamos como si estuvieran prendidos al mismo tiempo con valores diferentes, cada display individual
        --se encenderá dependiendo del selector que haya entrado al programa, para luego asignar a la signal digito un valor
        --determinado dependiendo de los switches que estén activados en la entrada hexadecimal y por ultimo prenda los
        --leds A,B,C,D,E,F o G correspondientes en todos los displays, este process lo que hará es que acceder a la salida
        --tipo vector llamada prenderDisplay y mandar un 0 lógico al nodo común del display que se debe prender en ese
        --momento, para que después el selector cambie de valor y se ejecute nuevamente el código pero ahora prendiendo
        --diferentes leds y otro display de 7 segmentos.
        elegirDisplay : process(selectorMUX, anodoEncendido)
        begin
                --En un inicio todos los displays de 7 segmentos estarán apagados.
                prenderDisplay <= "1111";
                --Al usar el nombre de un vector y poner entre paréntesis una de sus coordenadas, estoy accediendo solo a ese
                --bit del vector, además con la instrucción conv_integer() estoy convirtiendo lo que haya en el interior de su
                --paréntesis a su equivalente decimal.
                if(anodoEncendido(conv_integer(selectorMUX))= '1') then
                --Este condicional siempre será cierto porque el vector anodoEncendido esta inicializado con valor 1111, por
                --lo que cualquiera de sus coordenadas tiene un 1 lógico, lo importante de este condicional es que convierte
                --al selector en su número decimal y la coordenada que corresponda la cambia a un 0 lógico en el vector
                --prenderDisplay, que prende cada display dependiendo de los bits que existan en la signal digito en ese
                --momento, determinados por el valor del selector.
                        prenderDisplay(conv_integer(selectorMUX)) <= '0';
                end if;
        end process elegirDisplay;
end binarioAhexadecimal;
```

Módulo TLD:

```
--4.-TLD (Top Level Design) relojContadorMuxDisp7SegVHDL:
--Este código sirve para unir los 3 módulos anteriores y poder realizar el decodificador binario a hexadecimal y
--mostrar en los 4 displays de 7 segmentos diferentes valores por medio del divisorDeReloj, el contadorSelector, el
--Multiplexor y el Decodificador Binario a Hexadecimal.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerías que sirven solamente para poder usar el lenguaje VHDL.

--Entidad: Las entradas y salidas en este módulo son las que van a entrar y salir del diagrama de bloques global.
entity relojContadorMuxDisp7SegTLD is
    Port ( numBinario : in  STD_LOGIC_VECTOR (15 downto 0);
           Reset : in  STD_LOGIC;
           clkNexys2 : in  STD_LOGIC;
           numHexadecimal : out  STD_LOGIC_VECTOR (6 downto 0);
           puntoDisplay : out  STD_LOGIC;
           anodoComun : out  STD_LOGIC_VECTOR (3 downto 0));
end relojContadorMuxDisp7SegTLD;

--Arquitectura: Aquí se declaran las instancias de los demás módulos para poder unir entradas, salidas y signals.
architecture encenderDisplaysTLD of relojContadorMuxDisp7SegTLD is
--SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
--existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
--arquitectura y antes de su begin.
signal reloj : STD_LOGIC;--Reloj con frecuencia menor a 50MHz.
signal selector : STD_LOGIC_VECTOR (1 downto 0);--Contador/selector para encender cada display de 7 segmentos.
begin
        --INSTANCIAS:
        --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del módulo que quiero instanciar,
        --usar la palabra reservada port map(); y dentro de su paréntesis asignarle a todas las entradas y salidas del
        --modulo instanciado una entrada, salida o signal de este módulo separadas por comas una de la otra, esto hará que
        --lo que entre o salga del otro modulo, entre, salga o se guarde en este.
        --La sintaxis que debemos usar es la siguiente:

        --NombreInstancia            :                  entity work.Entidad_Del_Modulo_Que_Queremos_Instanciar     port map(
        --                  Entrada_Del_Modulo_Instanciado => Entrada_En_Este_Modulo,
        --                  Salida_Del_Modulo_Instanciado  => Salida_En_Este_Modulo,

        --                  Entrada_Del_Modulo_Instanciado => Salida_En_Este_Modulo,
        --                  Salida_Del_Modulo_Instanciado  => Entrada_En_Este_Modulo,

        --                  Entrada_Del_Modulo_Instanciado => Signal_En_Este_Modulo,
        --                  Salida_Del_Modulo_Instanciado  => Signal_En_Este_Modulo
        --);

        --INSTANCIA DEL MODULO divisorDeReloj para obtener la frecuencia en la que quiero que se cree el contador/selector.
        frecuenciaReloj : entity work.divisorDeReloj port map(
                          relojNexys2 => clkNexys2,
                          --La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clkNexys2 de este módulo.
                          rst => Reset,
                          --La entrada rst del divisorDeReloj se asigna a la entrada Reset de este módulo.
                          salidaReloj => reloj
                          --La salida salidaReloj del divisorDeReloj se asigna a la signal reloj de este módulo.
                );
        --INSTANCIA DEL MODULO contadorSelector para obtener el contador/selector que prendera cada display individualmente.
        selectorDisplay : entity work.contadorSelector port map(
                          frecuenciaReloj => reloj,
                          --A la entrada frecuenciaReloj del contadorSelector se le asigna el valor de la signal reloj de
                          --este módulo.
                          contador => selector
```

```
                        --La salida contador del contadorSelector se asigna a la signal selector de este módulo.
                );
        --INSTANCIA DEL MODULO decodBinHex para cambiar de sistema numérico binario a hexadecimal, recibir el selector del
        --contador y prender cada uno de los 4 displays dependiendo del número binario que este recibiendo y con la
        --frecuencia del selector, prender y apagar los 4 displays tan rápido que para el ojo humano parezca que todos están
        --encendidos al mismo tiempo con valores diferentes.
        binarioAhexadecimal : entity work.decodificadorBinHex port map(
                        binario => numBinario,
                        --La entrada binario de decodificadorBinHex se asigna a la entrada numBinario de este módulo.
                        selectorMUX => selector,
                        --A la entrada selectorMUX de decodificadorBinHex se le asigna el valor de la signal selector de
                        --este módulo.
                        prenderDisplay => anodoComun,
                        --La salida prenderDisplay de decodificadorBinHex se asigna a la salida anodoComun de este módulo.
                        ledsAhastaG => numHexadecimal,
                        --La salida ledsAhastaG de decodificadorBinHex se asigna a la salida numHexadecimal de este módulo.
                        DP => puntoDisplay
                        --La salida DP de decodBinHex se asigna a la salida puntoDislpay de este módulo.
                );
end encenderDisplaysTLD;
```

Código UTF:

```
//ENTRADAS Y SALIDAS DEL MODULO TDL:
//El diseño TLD sirve para hacer la conversión binario -> hexadecimal y mostrarlo como números diferentes en los
//4 displays de 7 segmentos de la placa NEXYS 2 como si estos tuvieran valores diferentes.

//ENTRADAS DEL MODULO decodificadorBinHex:
//SWITCHES DE LA NEXYS 2:
net "numBinario[0]" loc = "G18";//Bit 1 (menos significativo) del número binario conectado al switch SW0.
net "numBinario[1]" loc = "H18";//Bit 2 del número binario conectado al switch SW1.
net "numBinario[2]" loc = "K18";//Bit 3 del número binario conectado al switch SW2.
net "numBinario[3]" loc = "K17";//Bit 4 del número binario conectado al switch SW3.
net "numBinario[4]" loc = "L14";//Bit 5 del número binario conectado al switch SW4.
net "numBinario[5]" loc = "L13";//Bit 6 del número binario conectado al switch SW5.
net "numBinario[6]" loc = "N17";//Bit 7 del número binario conectado al switch SW6.
net "numBinario[7]" loc = "R17";//Bit 8 del número binario conectado al switch SW7.
//SWITCHES EXTERNOS CONECTADOS A LOS PUERTOS JA:
net "numBinario[8]" loc = "L15";//Bit 9 del número binario conectado al puerto JA1 perteneciente a los puertos JA.
net "numBinario[9]" loc = "K12";//Bit 10 del número binario conectado al puerto JA2 perteneciente a los puertos JA.
net "numBinario[10]" loc = "L17";//Bit 11 del número binario conectado al puerto JA3 perteneciente a los puertos JA.
net "numBinario[11]" loc = "M15";//Bit 12 del número binario conectado al puerto JA4 perteneciente a los puertos JA.
//Los puertos JA5 y JA11 están conectados a tierra y JA6 y JA12 están conectados a Vcc = 3.3[V].
net "numBinario[12]" loc = "K13";//Bit 13 del número binario conectado al puerto JA7 perteneciente a los puertos JA.
net "numBinario[13]" loc = "L16";//Bit 14 del número binario conectado al puerto JA8 perteneciente a los puertos JA.
net "numBinario[14]" loc = "M14";//Bit 15 del número binario conectado al puerto JA9 perteneciente a los puertos JA.
net "numBinario[15]" loc = "M16";//Bit 16 del número binario conectado al puerto JA10 perteneciente a los puertos JA.


//ENTRADAS DEL MODULO divisorDeReloj:
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "Reset" loc = "H13";//El botón de Reset está conectado al push button BTN3.


//SALIDAS DEL MODULO decodificadorBinHex:
//Leds A,B,C,D,E,F y G del display de 7 segmentos:
net "numHexadecimal[6]" loc = "L18";//1er bit del vector conectado al puerto CA (led A) del display de 7 segmentos.
net "numHexadecimal[5]" loc = "F18";//2do bit del vector conectado al puerto CB (led B) del display de 7 segmentos.
net "numHexadecimal[4]" loc = "D17";//3er bit del vector conectado al puerto CC (led C) del display de 7 segmentos.
net "numHexadecimal[3]" loc = "D16";//4to bit del vector conectado al puerto CD (led D) del display de 7 segmentos.
net "numHexadecimal[2]" loc = "G14";//5to bit del vector conectado al puerto CE (led E) del display de 7 segmentos.
net "numHexadecimal[1]" loc = "J17";//6to bit del vector conectado al puerto CF (led F) del display de 7 segmentos.
net "numHexadecimal[0]" loc = "H14";//7mo bit del vector conectado al puerto CG (led G) del display de 7 segmentos.
//Punto del display de 7 segmentos:
net "puntoDisplay" LOC = "C17";//Conectado al puerto DP (punto) del display de 7 segmentos.


//SALIDAS DEL MODULO decodificadorBinHex:
//Para que el orden se respete debo meter conectar el más significativo al nodo AN3 y el bit menos significativo al
//nodo AN0.
net "anodoComun[0]" loc = "F17";
//Bit menos significativo conectado al puerto AN0, 1er display de 7 segmentos de los 4 disponibles.
net "anodoComun[1]" loc = "H17";
//Conectado al puerto AN1, 2do display de 7 segmentos de los 4 disponibles.
net "anodoComun[2]" loc = "C18";
//Conectado al puerto AN2, 3er display de 7 segmentos de los 4 disponibles.
net "anodoComun[3]" loc = "F15";
//Bit más significativo conectado al puerto AN3, 4to display de 7 segmentos de los 4 disponibles.
```