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: Suma, Resta, Multiplicación y Decodificador BCD

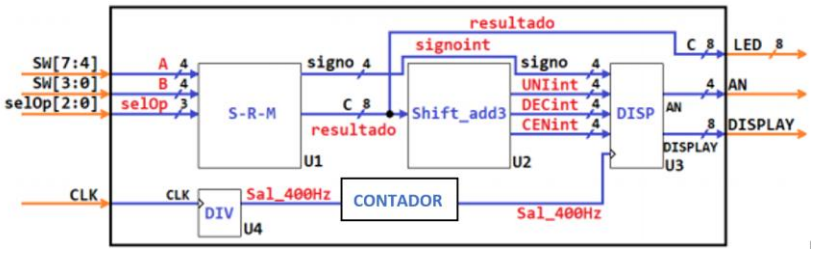
Contenido

TLD: Operaciones Matemáticas con 2 Números Binarios

El diagrama de bloques para poder realizar las operaciones: Suma, resta o multiplicación de 2 números binarios de 4 bits y mostrar el resultado en los 4 displays de 7 segmentos se muestra a continuación.

En el diagrama de bloques TLD (Top Level Design) se debe agregar:

- **DIV:** Un módulo divisor de reloj.
- **CONTADOR:** Un módulo contador/selector que elija cual de los displays de 7 segmentos se prende a la vez para mostrar dígitos distintos.
- **S-R-M:** Un módulo sumador, restador y multiplicador de 2 números binarios de 4 bits que se ingresarán al programa por medio de switches
- **Shift_add3**: Un módulo decodificador de números binarios a código BCD por medio del método Shift Add-3
- **DISP:** Un módulo que interprete el código BCD y reciba la señal de reloj con frecuencia modificada para poder mostrar 4 valores diferentes en los 4 displays de 7 segmentos incluidos en la NEXYS 2.



Divisor de Reloj: Módulo DIV

Del divisor de reloj se elegirá una frecuencia menor a 47.8 Hz que se encuentra en la coordenada 19 del vector que divide al reloj, para que esta frecuencia modificada le llegue al módulo **DISP**.

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

Decodificador Binario a BCD: Módulo Shift Add-3

El módulo describe un algoritmo para convertir números binarios a código BCD (usado para que los displays de 7 segmentos puedan interpretar números binarios).

En este caso vamos a convertir un número binario de máximo 8 bits, que podrá ir del 00000000 al 11111111, o del hexadecimal 00 al FF o del número decimal 0 al 255.

El método Shift Add-3 funciona siguiendo los siguientes pasos:

1. Recorre el número binario un bit a la izquierda.
2. Recorrer a la izquierda el número binario hasta que el valor en cualquiera de las columnas de las unidades, decenas o centenas del código BCD sea mayor a 4 en decimal, osea a 100 en binario
3. Se suma 3 decimal a ese valor en forma binaria, osea se le suma el 011.
4. Se pone abajo el resultado de la suma y se bajan los demás ceros y unos que llevábamos también.
5. Se repiten los pasos anteriores hasta que en la columna de binario no quede ningún cero o uno.

Operación	Centenas	Decenas	Unidades	Binario	
Hexadecimal				F	F
Inicio				1111	1111
Shift 1			1	1111	111
Shift 2			11	1111	11
Shift 3			111	1111	1
Add 3			1010	1111	1
Shift 4		1	0101	1111	
Add 3		1	1000	1111	
Shift 5		11	0001	111	
Shift 6		110	0011	11	
Add 3		1001	0011	11	
Shift 7	1	0010	0111	1	
Add 3	1	0010	1010		
Shift 8	10	0101	0101		
BCD	2	5	5		

De la tabla podemos observar que el resultado deberá ser almacenado en un vector de 10 bits y para poder recorrer todas las posiciones de la tabla debemos crear un vector de 18 bits.

Resultado BCD =	A11 A10 A9 A8	A7 A6 A5 A4	A3 A2 A1 A0		
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		
Posiciones =	S19 S18 S17 S16	S15 S14 S13 S12	S11 S10 S9 S8	S7 S6 S5 S4 S3 S2 S1 S0	

Los pasos descritos anteriormente son usados para resolver el método a mano, pero si queremos estandarizarlo para poder resolver el problema para un número binario de máximo 8 bits y pasarlo a código estos son los pasos que tenemos que seguir cuando se cree el código en los lenguajes VHDL y Verilog.

Primero que nada, en el código se deben crear dos vectores, uno de entrada y otro de salida o uno de entrada y 3 de salida, si es que se quiere separar el resultado de la conversión en unidades, decenas y centenas:

VHDL:

```
entity nombreEntidad is
   Port ( numBinario  : in STD_LOGIC_VECTOR (7 downto 0);
          --Vector de entrada de 8 bits para el número binario que quiero convertir
           salidaBCD : out  STD_LOGIC_VECTOR (11 downto 0)
          --Vector de salida de 12 bits para el código BCD de salida que será el equivalente del
          --número binario de entrada, este puede ser separado en centenas, decenas y unidades.
                              );
end nombreEntidad;
```

Además, se debe declarar un vector tipo variable que me permita recorrer todas las posiciones del método Shift Add-3 por medio de un bucle for para que pueda convertir un número binario de 8 bits:

```
variable posiciones: STD_LOGIC_VECTOR (17 downto 0);
--Variable tipo vector que puede almacenar todas las 18 posiciones posibles del método shift Add-3
```

Verilog:

```
module nombreModulo (
          input [7:0] numBinario,
         --Vector de entrada de 8 bits para el número binario que quiero convertir
          output [7:0] salidaBCD
         --Vector de salida de 12 bits para el código BCD de salida que será el equivalente del
         --número binario de entrada, este puede ser separado en centenas, decenas y unidades.
```

```
                    );
end nombreEntidad;
```

Además, se debe declarar un tipo de dato reg y otro integer que me permitan recorrer todas las posiciones del método Shift Add-3 por medio de un bucle for para que pueda convertir un número binario de 8 bits:

```
reg [17:0] posiciones;
integer ciclos1erBucleFor;
integer ciclos2doBucleFor;
--Variable tipo vector que puede almacenar todas las 18 posiciones posibles del método shift Add-3
```

Los pasos del algoritmo son iguales, no importando que lenguaje se elija:

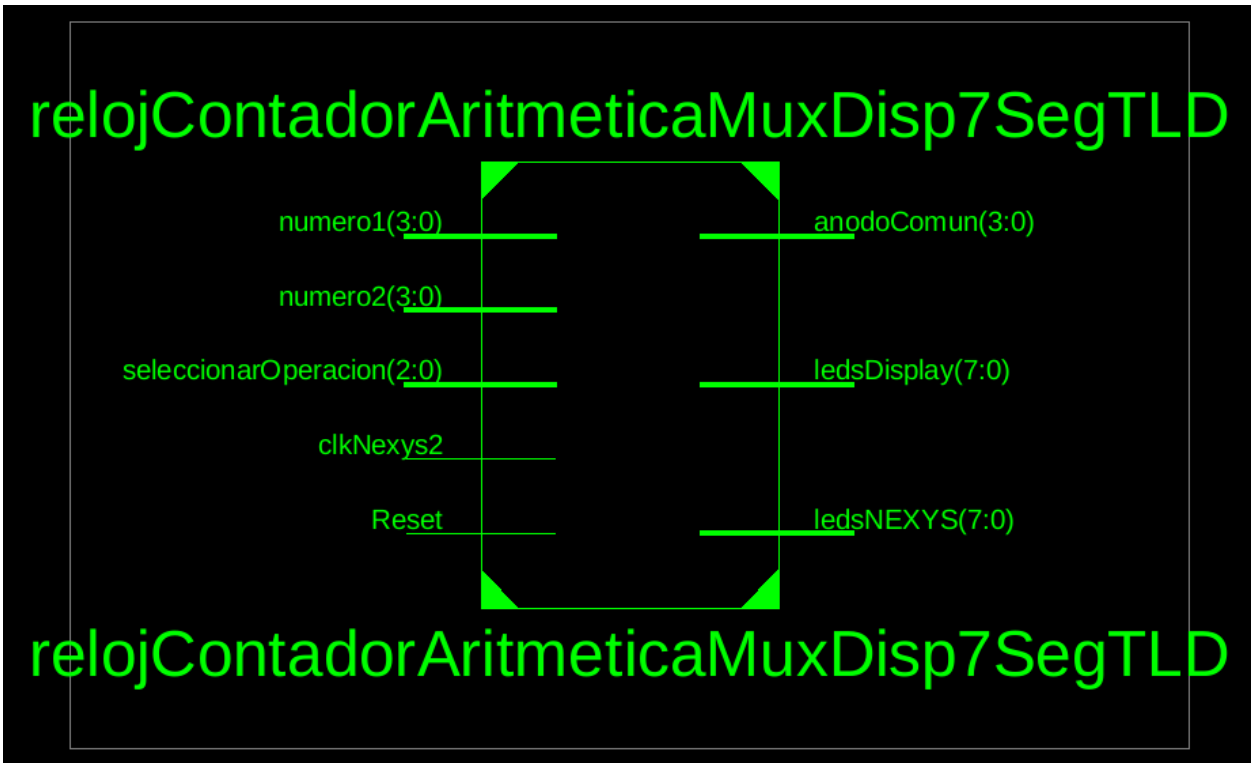
1. Recorre el numero binario a la izquierda un bit, esto máximo se puede repetir 3 veces, durante las operaciones **SHIFT1**, **SHIFT2** y **SHIFT3** como se ve en la tabla donde se resolvió el método. Si recorremos 3 posiciones el número original, el vector posiciones de 8 bits estará situado de la posición **S3** a la **S10** porque en ese punto solo hay 3 bits en la columna de las Unidades y 5 bits en la columna Binario.
2. Después del **SHIFT3** se empezará a analizar cada columna para ver si el número binario contenido en cada una es mayor a 4. Después de haber ocurrido 8 corrimientos (osea después del SHIFT8), el número a convertir ya estará abarcando las columnas de unidades, centenas y decenas, por lo que se debe analizar en estos corrimientos, osea **SHIFT4**, **SHIFT5**, **SHIFT6**, **SHIFT7** y **SHIFT8** si el número es mayor a 4 en cada columna para que si es así se le sumen 3 unidades.
    a. Cada columna la puedo analizar individualmente si analizo las coordenadas del vector posiciones que las abarcan:
        i. Columna de Unidades: Posiciones **S11**, **S10**, **S9** y **S8**.
        ii. Columna de Decenas: Posiciones **S15**, **S14**, **S13** y **S12**.
        iii. Columna de Centenas: Posiciones **S19**, **S18**, **S17** y **S16**.

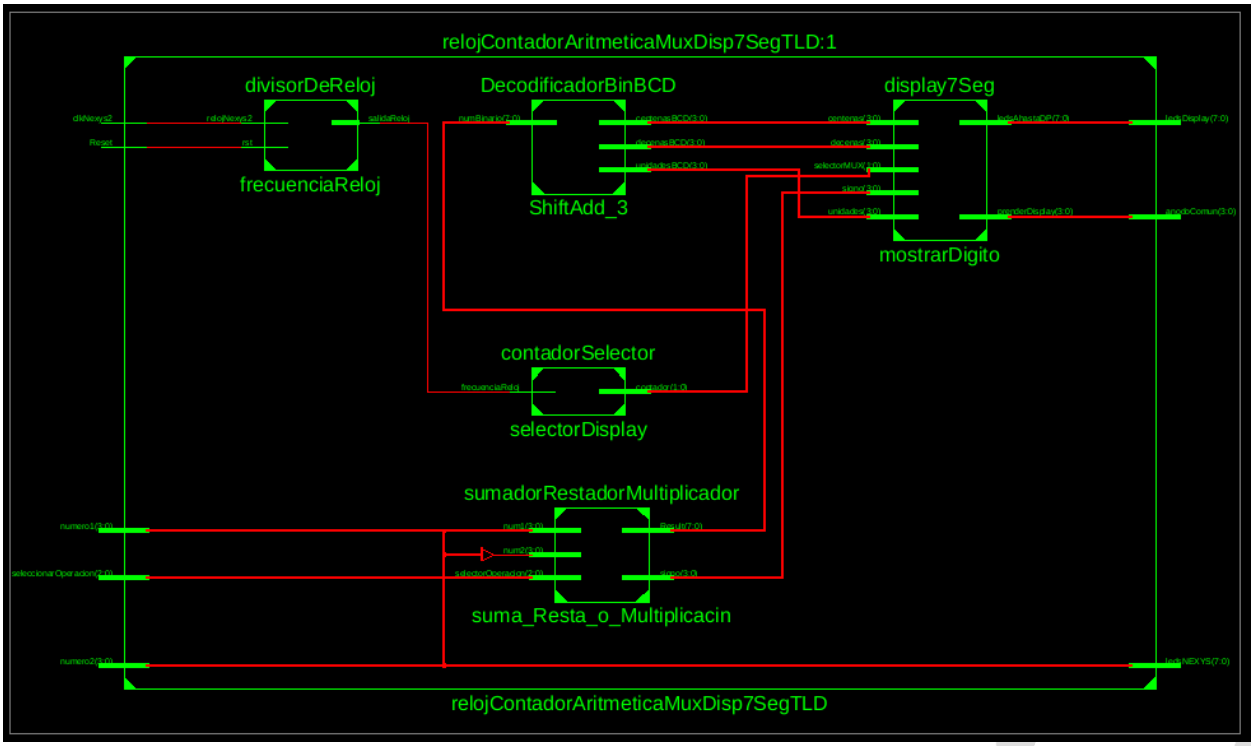
    Cada una de estas columnas debe ser analizada para ver si se le puede aplicar la operación **Add-3** con un condicional if, excepto por la columna de las Centenas, ya que nunca se llega a aplicar esta operación cuando queremos convertir un número binario de 8 bits.

3. Repetir el paso 1 hasta que no haya ningún bit en la columna de Binario, esto abarca las posiciones: **S19**, **S18**, **S17**, **S16**, **S15**, **S14**, **S13**, **S12**, **S11**, **S10**, **S9** y **S8**.

Código VHDL:

Diagrama TLD:





Divisor de Reloj (DIV):

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

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 "000000000000000000000000".
                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 específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
        salidaReloj <= divisorDeReloj(16);--En la tabla se ve que la coordenada 16 proporciona una frecuencia de 381.47Hz.
end frecuenciaNueva;
```

Contador/Selector (CONTADOR):

```
--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 representarán al selector y el selector lo que hará es guardar en la
--signal digito (del siguiente modulo también) 4 números binarios que representen 1 dígito 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 se busca es que, durante un ciclo de reloj, todos los displays sean encendidos una vez, mostrando
--cada digito del resultado de la operación matemática de los 2 números binario introducidos 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 instrucción rising_edge indica que cada vez que ocurra un flanco de subida en el reloj, se le sumara un 1
                --binario al valor que tenía 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;
```

Sumador Restador Multiplicador (S-M-R):

```
--3.-MULTIPLEXOR PARA SELECCIONAR QUE LA OPERACION SEA SUMA, RESTA O MULTIPLICACION:
--Este proceso sirve para elegir qué operación se hace con los dos números binarios entrantes num1 y num2.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Estas librerías solo se declaran para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Esta librería nos permite hacer operaciones matemáticas con vectores o bits sin considerar su signo.
use IEEE.STD_LOGIC_ARITH.ALL;
--Esta librería nos permite hacer operaciones matemáticas usando la instrucción process.

entity sumadorRestadorMultiplicador is
    Port ( num1 : in  STD_LOGIC_VECTOR (3 downto 0);
           num2 : in  STD_LOGIC_VECTOR (3 downto 0);
                  --Números binarios siempre positivos que se van a sumar, restar o multiplicar.
           selectorOperacion : in  STD_LOGIC_VECTOR (2 downto 0);
                  --Con este selector por medio del MUX indicaremos que operación se ejecutara.
           Result : out  STD_LOGIC_VECTOR (7 downto 0);
                  --El resultado de la multiplicación va a tener el doble de bits que los números multiplicados entre sí.
                  --El resultado de la suma debe ser un bit mayor a los números sumados entre sí.
                  --El resultado de la resta va a ser del mismo número de bits que los números restados entre sí.
                  --La multiplicación demanda más bits, por lo cual el vector que almacena el resultado será de ese tamaño
           signo : out  STD_LOGIC_VECTOR (3 downto 0));
                  --Este vector indicara el signo de la operación después en el display de 7 segmentos.
end sumadorRestadorMultiplicador;

--Arquitectura: Aquí se declara lo que harán las entradas/salidas, osea las operaciones matemáticas.
architecture seleccionarOperacion of sumadorRestadorMultiplicador 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, selectorOperacion)
        begin
                case(selectorOperacion) is
                        --SUMA:
                        when "001" =>
                        --Result <= num1+num2; Si pongo así la suma me dará un error porque el vector Result es de 8 bits y
                        --los vectores num1 y num2 son de 4 bits cada uno, por eso es que concateno ambos números con 4 bits
                        --0000, la concatenación se hace con el signo &.
                                Result <= ("0000" & num1) + ("0000" & num2);
                                --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".
                                signo <= X"E";
                                --Poner X"E" equivale a poner "1110" y esto indica un signo positivo +.

                        --RESTA:
                        when "010" =>
                        --Para la resta se debe hacer un condicional if ya que aquí es donde puede cambiar el signo de la
                        --operación, dependiendo de si num1 es mayor o igual a num2, si esto es así, la magnitud se saca
                        --solo haciendo la resta
                        --y el signo se queda como positivo.
                                if(num1 >= num2) then
                                        Result <= ("0000" & num1) - ("0000" & num2);
                                        signo <= X"E";
                                        --Poner X"E" equivale a poner "1110" y esto indica un signo positivo +.
                        --Si num1 es menor a num2, el signo se convertirá a negativo y la magnitud se debe obtener restando
                        --num2 menos num1.
                                else
                                        Result <= ("0000" & num2) - ("0000" & num1);
                                        signo <= X"F";
                                        --Poner X"F" equivale a poner "1111" y esto indica un signo negativo -.
                                end if;
                        --MULTIPLICACION:
                        when "100" =>
                        --Cuando haga la multiplicación no necesito concatenar ceros de más porque la operación por si sola
                        --llena el vector Result de 8 bits y el signo se quedará como positivo siempre.
                                Result <= num1 * num2;
                                signo <= X"E";
                                --Poner X"E" equivale a poner "1110" y esto indica un signo positivo +.
                        when others =>
                        --Cuando se seleccione otra opción se mostrarán puros ceros en los displays de 4 segmentos.
                                Result <= "00000000";
                                signo <= X"E";
                                --Poner X"E" equivale a poner "1110" y esto indica un signo positivo +.
                end case;
        end process;
end seleccionarOperacion;
```

Decodificador BCD (Shift_add3):

```
--4.-CONVERTIDOR BINARIO a BCD: MTODO SHIFT ADD-3
--Este método sirve para convertir números binarios a código BCD, el código BCD sirve para que las maquinas nos puedan
--ensenar números en un display de 7 segmentos, donde cada 4 bits representan un digito decimal.

library IEEE;
```

```vhdl
use IEEE.STD_LOGIC_1164.ALL;
--Estas libreras solo se declaran para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Esta librería nos permite hacer operaciones matemáticas con vectores o bits sin considerar su signo.
use IEEE.STD_LOGIC_ARITH.ALL;
--Esta librería nos permite hacer operaciones matemáticas usando process.

entity DecodificadorBinBCD is
    Port ( numBinario : in  STD_LOGIC_VECTOR (7 downto 0);
           centenasBCD : out  STD_LOGIC_VECTOR (3 downto 0);
           decenasBCD : out  STD_LOGIC_VECTOR (3 downto 0);
           unidadesBCD : out  STD_LOGIC_VECTOR (3 downto 0));
end DecodificadorBinBCD;

architecture SHIFTADD3 of DecodificadorBinBCD 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 codigoBCDcompleto : STD_LOGIC_VECTOR (11 downto 0);
--Esta signal sirve para almacenar el código BCD completo que incluye sus 4 bits de unidades, decenas y centenas.
begin
        --Dentro del process se va a poner el algoritmo para ejecutar el método Shift Add-3, para entender el procedimiento
        --debo meterme al documento 11.-Sum, Res, Mult, Decod BCD y 4 Valores Disp 7 Seg.
        process(numBinario)
        --Las variables se deben declarar dentro de process y deben estar antes de su begin, estas solo van a poder existir
        --y ser usadas dentro del process donde estén declaradas, su sintaxis es: variable nombreVariable: tipo_de_dato;
        variable posiciones : STD_LOGIC_VECTOR (19 downto 0);
        begin
                --LLENAR DE CEROS TODAS LAS POSICIONES POSIBLES (OSEA LAS COLUMNAS) DE LA TABLA SHIFT ADD-3
                for ejecucionBucle in 0 to 19 loop
                        --Este bucle se va a repetir 20 veces y lo que hará es limpiar todos los bits de la
                        --variable posiciones.
                        posiciones(ejecucionBucle) := '0'; --Con limpiar nos referimos a llenar de ceros el vector.
                end loop;

                --METER EL NUMERO BINARIO EN LA VARIABLE POSICIONES PARA QUE ENTRE A LA TABLA DEL METODO SHIFT ADD-3
                posiciones(7 downto 0) := numBinario(7 downto 0);
                --Con esta instrucción metemos el valor de la entrada llamada numBinario a las posiciones 0,1,2,3,4,

                --5,6 y 7 del vector posiciones, esto se hace para que este en la posición inicial de la tabla.

                --MÉTODO SHIFT ADD-3 A LAS COLUMNAS: UNIDADES, DECENAS Y CENTENAS
                for i in 0 to 7 loop --En la tabla se repite la operación hasta SHIFT8, por eso se ejecuta 8 veces.
                        --Este bucle se va a repetir 8 veces y lo que hará es ejecutar las operaciones SHIFT1, SHIFT2,
                        --SHIFT3, SHIFT4, SHIFT5, SHIFT6, SHIFT7 y SHIFT8 ya que estas son todas las veces que se puede
                        --recorrer el numero binario a la izquierda en la tabla del método SHIFT ADD-3 aplicado a un número
                        --binario de máximo 8 bits, durante estos recorrimientos se debe analizar cada columna de Unidades,
                        --Decenas y Centenas.

                        --COLUMNA DE UNIDADES: ADD-3
                        if posiciones(11 downto 8) > "100" then
                                --Cada columna la puedo analizar individualmente si analizo las coordenadas del vector
                                --posiciones que las abarcan, las posiciones 11, 10, 9 y 8 abarcan la columna de Unidades.
                                posiciones(11 downto 8) := posiciones(11 downto 8)+"11"; --Add3
                                --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                --osea 100 se le suma el numero decimal 3, osea 011.
                        end if;

                        --COLUMNA DE DECENAS: ADD-3
                        if posiciones(15 downto 12) > "100" then
                                --Las posiciones 15, 14, 13 y 12 abarcan la columna de Decenas.
                                posiciones(15 downto 12) := posiciones(15 downto 12)+"11";--Add3
                                --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                --osea 100 se le suma el numero decimal 3, osea 011.
                        end if;

                        --COLUMNA DE CENTENAS: ADD-3
                        if posiciones(19 downto 16) > "100" then
                                --Las posiciones 19, 18, 17 y 16 abarcan la columna de Centenas.
                                posiciones(19 downto 16) := posiciones(19 downto 16)+"11";--Add3
                                --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                --osea 100 se le suma el numero decimal 3, osea 011.
                        end if;

                        --SHIFT: Este pedazo de código aplicara los SHIFT1,2,3,4,5,6,7 y SHIFT8
                        posiciones(19 downto 1) := posiciones(18 downto 0);
                        --Esta operación del bucle es la operación SHIFT y es usada para mover un lugar a la izquierda todo
                        --el vector cuando ningún número en ninguna columna sea mayor a 4, osea 100.
                end loop;

        --GUARDO EL RESULTADO DEL METODO SHIFT ADD-3 EN LA SIGNAL.
        codigoBCDcompleto <= posiciones(19 downto 8);
        --Las posiciones 19 downto 8 son todas las que abarca el código BCD al terminar de ejecutarse el método
        --SHIFT ADD-3 porque ya no debe quedar ningún bit en la columna BINARIO.
        end process;

        --SEPARAR EL RESULTADO DEL MTODO SHIFT ADD-3, OSEA AL CODIGO BCD EN UNIDADES, DECENAS Y CENTENAS:
        --A las salidas les asigno valores usando el símbolo <= y lo puedo hacer en cualquier lugar del código.
        centenasBCD <= codigoBCDcompleto(11 downto 8);--4 bits del código BCD representan un dígito decimal, osea 1 centena.
        decenasBCD  <= codigoBCDcompleto(7 downto 4);--4 bits del código BCD representan un dígito decimal, osea 1 decena.
        unidadesBCD <= codigoBCDcompleto(3 downto 0);--4 bits del código BCD representan un dígito decimal, osea 1 unidad.
end SHIFTADD3;
```

Resultado Mostrado en los 4 Displays Distintos Simultáneamente (DISP):

```
--5.-MUX DISPLAY 7 SEGMENTOS:
--Este código sirve para mostrar 4 números decimales 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 3 vectores que me permitirán introducir el código BCD proveniente del modulo
--DecodificadorBinBCD para saber las unidades, decenas y centenas del número decimal creado por una suma, resta o
--multiplicacion previa de 2 números binarios iniciales, ingresados al programa por medio de switches, las
--operaciones son realizadas por el módulo sumadorRestadorMultiplicador que además manda a este módulo un vector
--que indica el signo del número decimal resultante de la operacion, además declaro como entrada 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,G y DP de cada display que también se activan con un 0
--lógico para mostrar letras o números.
entity display7Seg is
    Port ( signo : in  STD_LOGIC_VECTOR (3 downto 0);--Signo proveniente del módulo sumadorRestadorMultiplicador.
           unidades : in  STD_LOGIC_VECTOR (3 downto 0);--Código BCD que representa las unidades.
           decenas : in  STD_LOGIC_VECTOR (3 downto 0);--Código BCD que representa las decenas.
           centenas : in  STD_LOGIC_VECTOR (3 downto 0);--Código BCD que representa las centenas.
           selectorMUX : in  STD_LOGIC_VECTOR (1 downto 0);--Selector proveniente del módulo contadorSelector.
           prenderDisplay : out  STD_LOGIC_VECTOR (3 downto 0);--Vector de nodos comunes para prender cada display.
           ledsAhastaDP : out  STD_LOGIC_VECTOR (7 downto 0));--Vector con los leds A,B,C,D,E,F,G y DP.
end display7Seg;

--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 mostrar4Valores of display7Seg 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 digito : STD_LOGIC_VECTOR (3 downto 0);
--El vector digito me permitirá realizar la conversión Binario -> Hexadecimal.
begin
        --MULTIPLEXOR: Existen muchas entradas y una salida.
        --ENCENDIDO DE CADA DISPLAY PARA MOSTRAR UNIDADES, DECENAS Y CENTENAS:
        digitoDisplay : process(selectorMUX, unidades, decenas, centenas, digito, signo)
        begin
                case(selectorMUX) is
                --En el vector de entrada binario el bit más significativo se encuentra en la posición 15 y el menos
                --significativo en la posición 0.
                        when "00" =>--UNIDADES
                                prenderDisplay <= "1110";
                                --Cuando el selector valga 00, la salida de 4 bits llamada prenderDisplay encenderá solo
                                --el 4to display mandando un 0 lógico a su nodo común.
                                digito <= unidades;
                                --Y a la signal digito se le asignara el valor de la entrada unidades.

                        when "01" =>--DECENAS
                                prenderDisplay <= "1101";
                                --Cuando el selector valga 01, la salida de 4 bits llamada prenderDisplay encenderá solo
                                --el 3er display mandando un 0 lógico a su nodo común.
                                digito <= decenas;
                                --Y a la signal digito se le asignara el valor de la entrada decenas.

                        when "10" => --CENTENAS
                                prenderDisplay <= "1011";
                                --Cuando el selector valga 10, la salida de 4 bits llamada prenderDisplay encenderá solo
                                --el 2do display mandando un 0 lógico a su nodo común.
                                digito <= centenas;
                                --Y a la signal digito se le asignara el valor de la entrada centenas.

                        when others =>--SIGNO
                                prenderDisplay <= "0111";
                                --Cuando el selector valga 11, la salida de 4 bits llamada prenderDisplay endendera solo
                                --el 1er display mandando un 0 lógico a su nodo común.
                                digito <= signo;
                                --Y a la signal digito se le asignara el valor de la entrada signo.
                end case;
        end process digitoDisplay;


        --DECODIFICADOR: Transforma un código de pocos bits a uno de muchos bits, este en particular convierte de código
        --BCD 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.
        BCDtoDISP7SEG : 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 numero hexadecimal poniendo X"numero hexadecimal".
                case(digito) is
                        when X"0" => ledsAhastaDP <= "00000011"; --Pasa de X"0", osea 0000 a 0 en código display.
                        when X"1" => ledsAhastaDP <= "10011111"; --Pasa de X"1", osea 0001 a 1 en código display.
                        when X"2" => ledsAhastaDP <= "00100101"; --Pasa de X"2", osea 0010 a 2 en código display.
                        when X"3" => ledsAhastaDP <= "00001101"; --Pasa de X"3", osea 0011 a 3 en código display.
```

```
                        when X"4" => ledsAhastaDP <= "10011001"; --Pasa de X"4", osea 0100 a 4 en código display.
                        when X"5" => ledsAhastaDP <= "01001001"; --Pasa de X"5", osea 0101 a 5 en código display.
                        when X"6" => ledsAhastaDP <= "01000001"; --Pasa de X"6", osea 0110 a 6 en código display.
                        when X"7" => ledsAhastaDP <= "00011111"; --Pasa de X"7", osea 0111 a 7 en código display.
                        when X"8" => ledsAhastaDP <= "00000001"; --Pasa de X"8", osea 1000 a 8 en código display.
                        when X"9" => ledsAhastaDP <= "00001001"; --Pasa de X"9", osea 1001 a 9 en código display.
                        when X"F" => ledsAhastaDP <= "11111101"; --Pasa de X"F", osea 1111 a signo negativo.
                        --Esto fue elegido de manera random, pudo ser cualquier otro valor el que fuera el signo negativo y
                        --para el signo positivo solo haremos que no aparezca nada con la instrucción when others.
                    when others => ledsAhastaDP <= "11111111";
                    --Cualquier valor que no hayamos indicado no mostrara nada en el display.
                end case;
        end process BCDtoDISP7SEG;
end mostrar4Valores;
```

Módulo TLD:

```
--6.-TLD (Top Level Design) relojContadorAritmeticaMuxDisp7SegVHDL:
--Este código sirve para unir los 3 módulos anteriores y poder realizar la suma, resta o multiplicación de 2 números
--binarios, para ello primero se realiza la operación matemática, luego se aplica el decodificador binario a código
--BCD y finalmente su resultado se muestra en los 4 displays de 7 segmentos, mostrando así diferentes valores por
--medio del divisorDeReloj, el contadorSelector, el Multiplexor y el Decodificador Binario a BCD.

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 relojContadorAritmeticaMuxDisp7SegTLD is
    Port ( numero1 : in  STD_LOGIC_VECTOR (3 downto 0);
           numero2 : in  STD_LOGIC_VECTOR (3 downto 0);
           seleccionarOperacion : in  STD_LOGIC_VECTOR (2 downto 0);
           clkNexys2 : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           ledsNEXYS : out  STD_LOGIC_VECTOR (7 downto 0);
           ledsDisplay : out  STD_LOGIC_VECTOR (7 downto 0);
           anodoComun : out  STD_LOGIC_VECTOR (3 downto 0));
end relojContadorAritmeticaMuxDisp7SegTLD;

--Arquitectura: Aquí se declaran las instancias de los demás módulos para poder unir entradas, salidas y signals.
architecture Behavioral of relojContadorAritmeticaMuxDisp7SegTLD 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.
signal resultado : STD_LOGIC_VECTOR (7 downto 0);--Almacena el resultado de la operación matemática y enciende leds.
signal SIGNO : STD_LOGIC_VECTOR (3 downto 0);--Almacena el vector que denota el signo de la operación matemática.
signal CENTENAS : STD_LOGIC_VECTOR (3 downto 0);--Almacena los 4 bits del código BCD que denotan las centenas.
signal DECENAS : STD_LOGIC_VECTOR (3 downto 0);--Almacena los 4 bits del código BCD que denotan las decenas.
signal UNIDADES : STD_LOGIC_VECTOR (3 downto 0);--Almacena los 4 bits del código BCD que denotan las unidades.
begin
        --INICIALIZACION DE VALORES
        ledsNEXYS <= numero1 & numero2;--El punto siempre estará apagado.

        --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 sumadorRestadorMultiplicador para la suma, resta o multiplicación de números binarios.
        suma_Resta_o_Multiplicacin : entity work.sumadorRestadorMultiplicador port map(
                          num1 => numero1,
```

```
                        --A la entrada num1 del sumadorRestadorMultiplicador se le asigna el valor de la entrada numero1 de
                        --este módulo.
                        num2 => numero2,
                        --A la entrada num2 del sumadorRestadorMultiplicador se le asigna el valor de la entrada numero2 de
                        --este módulo.
                        selectorOperacion => seleccionarOperacion,
                        --A la entrada selectorOperacion del sumadorRestadorMultiplicador se le asigna el valor de la
                        --entrada seleccionarOperacion de este módulo.
                        Result => resultado,
                        --La salida Result del selectorOperacion se asigna a la signal resultado de este módulo.
                        signo => SIGNO
                        --La salida signo del selectorOperacion se asigna a la signal SIGNO de este módulo.
                );
        --INSTANCIA DEL MODULO DecodificadorBinBCD para cambiar de sistema numérico binario a código BCD por medio del
        --método Shift Add-3.
        ShiftAdd_3 : entity work.DecodificadorBinBCD port map(
                        numBinario => resultado,
                        --A la entrada numBinario de DecodificadorBinBCD se le asigna el valor de la signal resultado de
                        --este módulo.
                        centenasBCD => CENTENAS,
                        --La salida centenasBCD de DecodificadorBinBCD se asigna a la signal centena de este módulo.
                        decenasBCD => DECENAS,
                        --La salida decenasBCD de DecodificadorBinBCD se asigna a la signal decena de este módulo.
                        unidadesBCD => UNIDADES
                        --La salida unidadesBCD de DecodificadorBinBCD se asigna a la signal unidad de este módulo.
                );
        --INSTANCIA DEL MODULO display7Seg para recibir el selector del contador y prender cada uno de los 4 displays
        --dependiendo del código BCD que este recibiendo, y con la frecuencia del selector, prender y apagar los 4 displays
        --tan rápido que al ojo humano parezca que todos están encendidos al mismo tiempo con valores diferentes.
        mostrarDigito : entity work.display7Seg port map(
                        signo => SIGNO,
                        --A la entrada signo de display7Seg se le asigna el valor de la signal SIGNO de este modulo
                        unidades => UNIDADES,
                        --A la entrada unidades de display7Seg se le asigna el valor de la signal UNIDADES de este modulo
                        decenas => DECENAS,
                        --A la entrada decenas de display7Seg se le asigna el valor de la signal DECENAS de este modulo
                        centenas => CENTENAS,
                        --A la entrada centenas de display7Seg se le asigna el valor de la signal CENTENAS de este módulo.
                        selectorMUX => selector,
                        --A la entrada selectorMUX de display7Seg se le asigna el valor de la signal selector de
                        --este módulo.
                        prenderDisplay => anodoComun,
                        --La salida prenderDisplay de display7Seg se asigna a la salida anodoComun de este módulo.
                        ledsAhastaDP => ledsDisplay
                        --La salida ledsAhastaG de display7Seg se asigna a la salida numHexadecimal de este módulo.
                );
end Behavioral;
```

Código UCF:

```
//ENTRADAS Y SALIDAS DEL MDULO TDL:
//El diseño TLD sirve para hacer la conversión binario -> BCD y mostrarlo como números diferentes en los
//4 displays de 7 segmentos de la placa NEXYS 2 como si estos tuvieran valores diferentes después de haberles
//aplicado una suma, resta o multiplicación a dos números binarios, las entradas y salidas usadas aquí deben
//ser las del módulo TLD.


//ENTRADAS DEL MODULO sumadorRestadorMultiplicador
//SWITCHES DE LA NEXYS 2
//NUMERO BINARIO 1
net "numero1[0]" loc = "L14";//Bit 1 (menos significativo) del número binario 1 conectado al switch SW4.
net "numero1[1]" loc = "L13";//Bit 2 del número binario 1 conectado al switch SW5.
net "numero1[2]" loc = "N17";//Bit 3 del número binario 1 conectado al switch SW6.
net "numero1[3]" loc = "R17";//Bit 4 (más significativo) del número binario 1 conectado al switch SW7.
//NUMERO BINARIO 2
net "numero2[0]" loc = "G18";//Bit 1 (menos significativo) del número binario 2 conectado al switch SW0.
net "numero2[1]" loc = "H18";//Bit 2 del número binario 2 conectado al switch SW1.
net "numero2[2]" loc = "K18";//Bit 3 del número binario 2 conectado al switch SW2.
net "numero2[3]" loc = "K17";//Bit 4 (más significativo) del número binario 2 conectado al switch SW3.

//PUSH BUTTON DEL SELECTOR PARA ELEGIR LA OPERACION SUMA, RESTA O MULTIPLICACION
net "seleccionarOperacion[0]" loc = "H13";//Operación Suma conectada al push button BTN3.
net "seleccionarOperacion[1]" loc = "E18";//Operación Resta conectada al push button BTN2.
net "seleccionarOperacion[2]" loc = "D18";//Operación Multiplicación conectada al push button BTN1.


//ENTRADAS DEL MODULO divisorDeReloj
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "Reset" loc = "B18";//El botón de Reset está conectado al push button BTN0.


//SALIDAS DEL MODULO TLD: relojContadorAritmeticaMuxDisp7SegTLD para encender leds indicativos
//Leds A,B,C,D,E,F y G del display de 7 segmentos
net "ledsNEXYS[7]" loc = "R4"; //1er bit (más significativo) de la signal resultado conectada al LED LD7.
net "ledsNEXYS[6]" loc = "F4"; //2do bit de la signal resultado conectada al LED LD6.
net "ledsNEXYS[5]" loc = "P15";//3er bit de la signal resultado conectada al LED LD5.
net "ledsNEXYS[4]" loc = "E17";//4to bit de la signal resultado conectada al LED LD4.
net "ledsNEXYS[3]" loc = "K14";//5to bit de la signal resultado conectada al LED LD3.
net "ledsNEXYS[2]" loc = "K15";//6to bit de la signal resultado conectada al LED LD2.
net "ledsNEXYS[1]" loc = "J15";//7mo bit de la signal resultado conectada al LED LD1.
net "ledsNEXYS[0]" loc = "J14";//8vo bit (menos significativo) de la signal resultado conectada al LED LD0.
```

```
//SALIDAS DEL MODULO decodificadorBinHex
//Leds A,B,C,D,E,F y G del display de 7 segmentos
net "ledsDisplay[7]" loc = "L18";//1er bit del vector conectado al puerto CA (led A) del display de 7 segmentos.
net "ledsDisplay[6]" loc = "F18";//2do bit del vector conectado al puerto CB (led B) del display de 7 segmentos.
net "ledsDisplay[5]" loc = "D17";//3er bit del vector conectado al puerto CC (led C) del display de 7 segmentos.
net "ledsDisplay[4]" loc = "D16";//4to bit del vector conectado al puerto CD (led D) del display de 7 segmentos.
net "ledsDisplay[3]" loc = "G14";//5to bit del vector conectado al puerto CE (led E) del display de 7 segmentos.
net "ledsDisplay[2]" loc = "J17";//6to bit del vector conectado al puerto CF (led F) del display de 7 segmentos.
net "ledsDisplay[1]" loc = "H14";//7mo bit del vector conectado al puerto CG (led G) del display de 7 segmentos.
net "ledsDisplay[0]" loc = "C17";//8vo bit del vector conectado al puerto DP, osea el punto del display de 7 segmentos.


//SALIDAS DEL MODULO decodificadorBinHex
//Para que el orden que puse en el código se respete debo 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.
```

TLD: Funciones Matemáticas con 1 Número Binario

Se repetirá el mismo concepto del ejercicio anterior, pero ahora se aplicará una de 2 funciones matemáticas a un número binario introducido al programa a través de switches, las funciones matemáticas son las siguientes:

- Función 1:

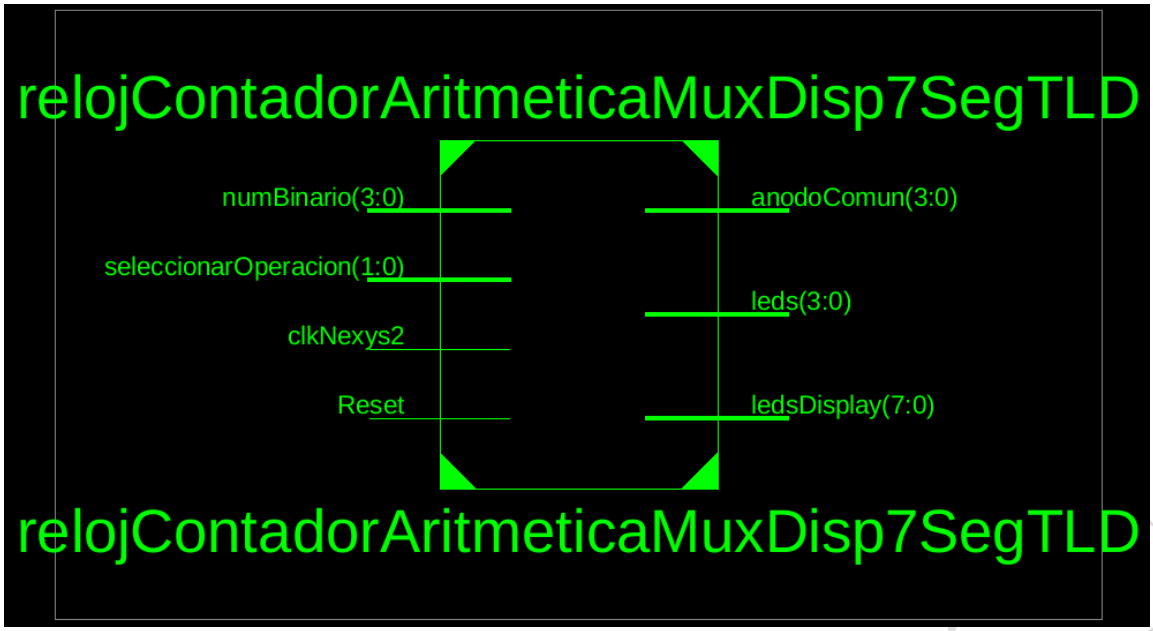
$$Y(x) = 2 * x + 2$$

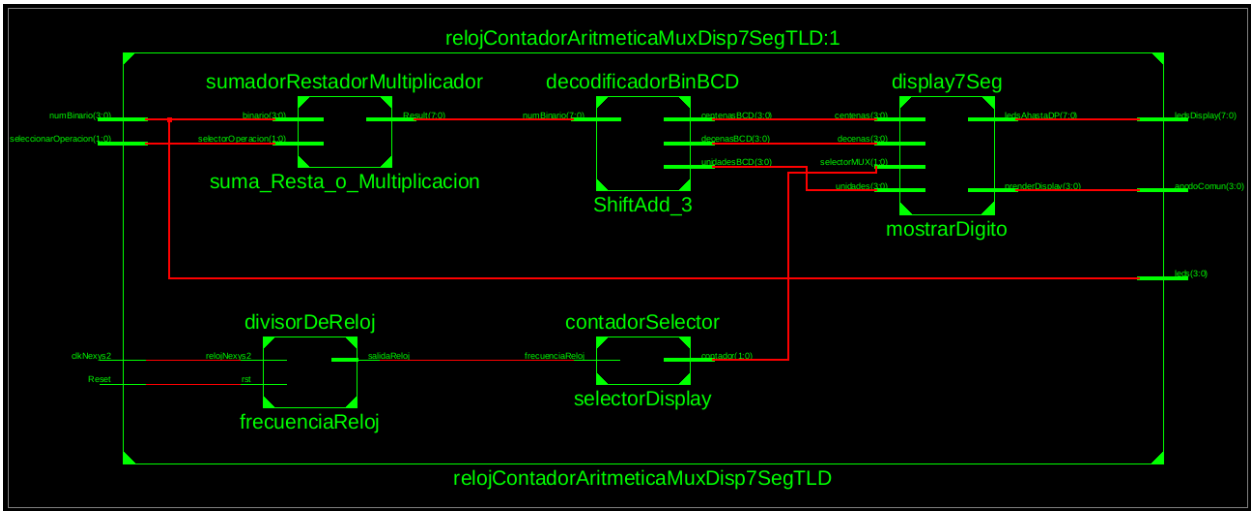
- Función 2:

$$Y(x) = x^2 = x * x$$

Código Verilog:

Diagrama TLD:





Divisor de Reloj (DIV):

```
//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@(), además 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 si solo, significa que yo directamente no debo
        //indicarlo con una operación lógica 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 a un número 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;
                else //Esto se ejecutará cuando no se cumpla la condición anterior, osea cuando no sea presionado el 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 (CONTADOR):

```
//2.-CONTADOR DE 2 BITS:
//En este código el contador es de 2 bits, esto implica que contara desde el cero hasta el 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 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 dígito 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 está 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 instruccion 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
```

Función Matemática (S-M-R):

```
//3.-MULTIPLEXOR PARA SELECCIONAR UNA FUNCION que sume o multiplique el numero binario entrante:
//Este proceso sirve para elegir que función se aplica al número binario entrante, además para hacer
//operaciones matemáticas básicas, se pueden crear funciones con los circuitos lógicos.
module funcionMatematica(
    input [3:0] binario,//Número binario entrante al que le voy a aplicar 2 posibles funciones.
    input [1:0] selectorOperacion,//Este selector estará conectado a push buttons para elegir la función que quiera.
    output reg [7:0] Result//Aquí se guardará el resultado para mandarse al decodificador, que es el módulo siguiente.
    );

          //ELEGIR FUNCIONES
          always@(binario or selectorOperacion)
          //always@() sirve para hacer condicionales o bucles y dentro de su paréntesis se deben indicar las entradas que van a
          //usar los condicionales o bucles que estén dentro del always.
          begin
                    case(selectorOperacion)
                              2'b01 : Result = 2*binario+2;
                              //Cuando el selector valga 10, se realizará la función Y(x)=2*x+2.
                              2'b10 : Result = binario*binario;
                              //Cuando el selector valga 10, se realizará la función Y(x)=x*x, osea x al cuadrado.
                              default : Result = 8'b00000000;
                              //Si quiero que se ejecute algo cuando no se cumpla ninguna de las dos condiciones anteriores uso
                              //default, si ninguna condición anterior se cumple, el vector Result se llenar de ceros.
                    endcase
          end
endmodule
```

Decodificador BCD (Shift_add3):

```
//4.-DECODIFICADOR BINARIO a BCD: MTODO SHIFT ADD-3
//Este método sirve para convertir números binarios a código BCD, el código BCD sirve para que las maquinas nos puedan
//ensenar números a través de displays de 7 segmentos, donde cada 4 bits representan un digito decimal.

module decodificadorBinBCD(
    input [7:0] numBinario,//Resultado del módulo anterior que le realizo una función matemática a un número binario.
    output [3:0] centenasBCD,
    output [3:0] decenasBCD,
    output [3:0] unidadesBCD

           //El código BCD describe cada digito decimal con 4 bits que indican sus unidades, decenas y centenas.
    );

          //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 [11:0] codigoBCDcompleto;
          reg [19:0] posiciones;

          integer ciclos;
          integer i;

          //Dentro de always@() se va a poner el algoritmo para ejecutar el método Shift Add-3, para entender el procedimiento
          //debo meterme al documento 11.-Sum, Res, Mult, Decod BCD y 4 valores Disp 7 Seg.
          always@(numBinario)
          begin
                    //LLENAR DE CEROS TODAS LAS POSICIONES POSIBLES (OSEA LAS COLUMNAS) DE LA TABLA SHIFT ADD-3
                    for(ciclos=0; ciclos<=19; ciclos=ciclos+1) begin
                              //Este bucle se va a repetir 20 veces y lo que hará es limpiar todos los bits de la variable
                              //posiciones.

                              posiciones[ciclos] = 1'b0;//Con limpiar nos referimos a llenar de ceros el vector.
                    end


                    //METER EL NUMERO BINARIO EN LA VARIABLE POSICIONES PARA QUE ENTRE A LA TABLA DEL MTODO SHIFT ADD-3
```

```
                posiciones[7:0] = numBinario[7:0];
                //Con esta instrucción metemos el valor de la entrada llamada binario a las posiciones 0,1,2,3,4,
                //5,6 y 7 del vector posiciones, esto se hace para que este en la posición inicial de la tabla.


                //METODO SHIFT ADD-3 A LAS COLUMNAS: UNIDADES, DECENAS Y CENTENAS
                for(i=0; i<=7; i=i+1) begin
                        //Este bucle se va a repetir 8 veces y lo que hará es ejecutar las operaciones SHIFT1, SHIFT2,
                        //SHIFT3, SHIFT4, SHIFT5, SHIFT6, SHIFT7 y SHIFT8, ya que estas son todas las veces que se puede
                        //recorrer el numero binario a la izquierda en la tabla del método SHIFT ADD-3, aplicado a un número
                        //binario de máximo 8 bits, durante estos recorrimientos se debe analizar cada columna de Unidades,
                        //Decenas y Centenas.

                        //COLUMNA DE UNIDADES: ADD-3
                        if(posiciones[11:8]>4'b0100) begin
                                //Cada columna la puedo analizar individualmente si analizo las coordenadas del vector
                                //posiciones que las abarcan, las posiciones 11, 10, 9 y 8 abarcan la columna de Unidades.
                                posiciones[11:8] = posiciones[11:8] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el número decimal 3, osea 011.
                        end

                        //COLUMNA DE DECENAS: ADD-3
                        if(posiciones[15:12]>4'b0100) begin
                                //Las posiciones 15, 14, 13 y 12 abarcan la columna de Decenas.
                                posiciones[15:12] = posiciones[15:12] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el numero decimal 3, osea 011.
                        end

                        //COLUMNA DE CENTENAS: ADD-3
                        if(posiciones[19:16]>4'b0100) begin
                                //Las posiciones 19, 18, 17 y 16 abarcan la columna de Centenas.
                                posiciones[19:16] = posiciones[19:16] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el numero decimal 3, osea 011.
                        end

                        //SHIFT: Este pedazo de código aplicara los SHIFT1,2,3,4,5,6,7 y SHIFT8.
                        posiciones[19:1] = posiciones[18:0];
                        //Esta operación del bucle es la operación SHIFT y es usada para mover un lugar a la izquierda todo
                        //el vector cuando ningún número en ninguna columna sea mayor a 4, osea 100.
                end

                //GUARDO EL RESULTADO DEL METODO SHIFT ADD-3 EN EL REG
                codigoBCDcompleto = posiciones[19:8];
                //Las posiciones 19 downto 8 son todas las que abarca el código BCD al terminar de ejecutarse el método
                //SHIFT ADD-3 porque ya no debe quedar ningún bit en la columna BINARIO.
        end


        //SEPARAR EL RESULTADO DEL MTODO SHIFT ADD-3, OSEA AL CODIGO BCD EN UNIDADES, DECENAS Y CENTENAS
        //A las salidas les asigno valores usando la palabra reservada assign y lo puedo hacer en cualquier lugar del código.
        assign centenasBCD = codigoBCDcompleto[11:8];//4 bits del código BCD representan un digito decimal, osea 1 centena
        assign decenasBCD  = codigoBCDcompleto[7:4];//4 bits del código BCD representan un digito decimal, osea 1 decena
        assign unidadesBCD = codigoBCDcompleto[3:0];//4 bits del código BCD representan un digito decimal, osea 1 unidad
endmodule
```

Resultado en 4 Displays Distintos Simultáneamente (DISP):

```
//5.-MUX DISPLAY 7 SEGMENTOS:
//Este código sirve para mostrar 4 números decimales 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 display7Seg(
    input [3:0] unidades, //Código BCD que representa las unidades.
    input [3:0] decenas, //Código BCD que representa las decenas.
    input [3:0] centenas, //Código BCD que representa las centenas.
    input [1:0] selectorMUX, //Selector proveniente del módulo contadorSelector.
    output reg [3:0] prenderDisplay, //Vector de nodos comunes para prender cada display.
    output reg [7:0] ledsAhastaDP //Vector con los leds A,B,C,D,E,F,G y DP.
    );

        //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 [3:0] digito;

        //MULTIPLEXOR: Existen muchas entradas y una salida.
        //ENDENDIDO DE CADA DISPLAY PARA MOSTRAR UNIDADES, DECENAS Y CENTENAS.
        always@(selectorMUX or unidades or decenas or centenas or digito)
        begin : digitoDisplay //El nombre del always@() es digitoDisplay.
                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 : begin //UNIDADES
                        //Cuando el selector valga 00, la salida de 4 bits llamada prenderDisplay encenderá solo el 3er
                        //display, mandando un 0 lógico a su nodo común.
                                prenderDisplay = 4'b1101;
                                digito = unidades;
                        end
```

```
                        2'b01 : begin //DECENAS
                        //Cuando el selector valga 01, la salida de 4 bits llamada prenderDisplay encenderá solo el 2do
                        //display, mandando un 0 lógico a su nodo común.
                                prenderDisplay = 4'b1011;
                                digito = decenas;
                        end

                        2'b10 : begin //CENTENAS
                        //Cuando el selector valga 01, la salida de 4 bits llamada prenderDisplay encenderá solo el 1er
                        //display, mandando un 0 lógico a su nodo común.
                                prenderDisplay = 4'b0111;
                                digito = centenas;
                        end
                        default : begin //La última posibilidad de un case se pone como default
                        //Cuando el selector valga 11 no se prendera ningún display.
                                prenderDisplay = 4'b1111;
                                digito = 4'b1111;
                    end
                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 : ledsAhastaDP = 8'b00000011;
                        //Pasa de 0 hexadecimal, osea 0000 a 0 en código display 7 segmentos.
                        4'b0001 : ledsAhastaDP = 8'b10011111;
                        //Pasa de 1 hexadecimal, osea 0001 a 1 en código display 7 segmentos.
                        4'b0010 : ledsAhastaDP = 8'b00100101;
                        //Pasa de 2 hexadecimal, osea 0010 a 2 en código display 7 segmentos.
                        4'b0011 : ledsAhastaDP = 8'b00001101;
                        //Pasa de 3 hexadecimal, osea 0011 a 3 en código display 7 segmentos.
                        4'b0100 : ledsAhastaDP = 8'b10011001;
                        //Pasa de 4 hexadecimal, osea 0100 a 4 en código display 7 segmentos.
                        4'b0101 : ledsAhastaDP = 8'b01001001;
                        //Pasa de 5 hexadecimal, osea 0101 a 5 en código display 7 segmentos.
                        4'b0110 : ledsAhastaDP = 8'b01000001;
                        //Pasa de 6 hexadecimal, osea 0110 a 6 en código display 7 segmentos.
                        4'b0111 : ledsAhastaDP = 8'b00011011;
                        //Pasa de 7 hexadecimal, osea 0111 a 7 en código display 7 segmentos.
                        4'b1000 : ledsAhastaDP = 8'b00000001;
                        //Pasa de 8 hexadecimal, osea 1000 a 8 en código display 7 segmentos.
                        4'b1001 : ledsAhastaDP = 8'b00001001;
                        //Pasa de 9 hexadecimal, osea 1001 a 9 en código display 7 segmentos.
                        default : ledsAhastaDP = 8'b11111111;
                        //Cuando digito valga algún número binario que no sea una de mis condiciones, no se prendera ningún
                        //display.
                endcase
        end
endmodule
```

Módulo TLD:

```
//6.-TLD (Top Level Design) relojContadorAritmeticaMuxDisp7SegVHDL:
//Este código sirve para unir los 3 módulos anteriores y poder realizar la suma, resta o multiplicación de números
//binarios, para ello primero se realizan operaciones matemáticas precargadas que se aplican al único número binario
//que ingresa al programa, luego se aplica el decodificador binario a código BCD y finalmente su resultado se muestra
//en los 4 displays de 7 segmentos, mostrando así diferentes valores por medio del divisorDeReloj, el contadorSelector,
//el Multiplexor y el Decodificador Binario a BCD.
module relojContadorAritmeticaMuxDisp7SegTLD(
    input [3:0] numBinario,
    input [1:0] seleccionarOperacion,
    input clkNexys2,
    input Reset,
    output [3:0] leds,
    output [7:0] ledsDisplay,
    output [3:0] anodoComun
    );

         assign leds = numBinario;
        //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.
        wire [7:0] resultado;//Almacena el resultado de la operación matemática y encender sus leds en el display.
        wire [3:0] CENTENAS;//Almacena los 4 bits del código BCD que denotan las centenas.
        wire [3:0] DECENAS;//Almacena los 4 bits del código BCD que denotan las decenas.
        wire [3:0] UNIDADES;//Almacena los 4 bits del código BCD que denotan las unidades.

        //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 frecuenciaReloj(
                          .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 prender cada display individualmente.
        contadorSelector selectorDisplay(
                          .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 sumadorRestadorMultiplicador para aplicar las funciones matemáticas al número binario.
        funcionMatematica seleccionarFuncion(
                          .binario(numBinario),
                          //A la entrada binario del sumadorRestadorMultiplicador se le asigna el valor de la entrada
                          //numBinario de este módulo.
                          .selectorOperacion(seleccionarOperacion),
                          //A la entrada selectorOperacion del sumadorRestadorMultiplicador se le asigna el valor de la
                          //entrada seleccionarOperacion de este módulo.
                          .Result(resultado)
                          //La salida Result del selectorOperacion se asigna a la signal resultado de este módulo.
                );
        //INSTANCIA DEL MODULO DecodificadorBinBCD para cambiar de sistema numérico binario a código BCD por medio del
        //método Shift Add-3.
        decodificadorBinBCD ShiftAdd_3(
                          .numBinario(resultado),
                          //A la entrada numBinario de DecodificadorBinBCD se le asigna el valor de la signal resultado de
                          //este módulo.
                          .centenasBCD(CENTENAS),
                          //La salida centenasBCD de DecodificadorBinBCD se asigna a la signal centena de este módulo.
                          .decenasBCD(DECENAS),
                          //La salida decenasBCD de DecodificadorBinBCD se asigna a la signal decena de este módulo.
                          .unidadesBCD(UNIDADES)
                          //La salida unidadesBCD de DecodificadorBinBCD se asigna a la signal unidad de este módulo.
                );
        //INSTANCIA DEL MODULO display7Seg para recibir el selector del contador y prender cada uno de los 4 displays
        //dependiendo del código BCD que está recibiendo, y con la frecuencia del selector, prender y apagar los 4 displays
        //tan rápido que al ojo humano parezca que todos estén encendidos al mismo tiempo con valores diferentes.
        display7Seg mostrarDigito(
                          .unidades(UNIDADES),
                          //A la entrada unidades de display7Seg se le asigna el valor de la signal UNIDADES de este módulo.
                          .decenas(DECENAS),
                          //A la entrada decenas de display7Seg se le asigna el valor de la signal DECENAS de este módulo.
                          .centenas(CENTENAS),
                          //A la entrada centenas de display7Seg se le asigna el valor de la signal CENTENAS de este módulo.
                          .selectorMUX(selector),
                          //A la entrada selectorMUX de display7Seg se le asigna el valor de la signal selector de este
                          //módulo.
                          .prenderDisplay(anodoComun),
                          //La salida prenderDisplay de display7Seg se asigna a la salida anodoComun de este módulo.
                          .ledsAhastaDP(ledsDisplay)
                          //La salida ledsAhastaG de display7Seg se asigna a la salida numHexadecimal de este módulo.
                );
endmodule
```

Código UCF:

```
//ENTRADAS Y SALIDAS DEL MODULO TDL:
//El diseño TLD sirve para hacer la conversión binario -> BCD y mostrarlo como números diferentes en los
//4 displays de 7 segmentos de la placa NEXYS 2 como si estos tuvieran valores diferentes después de haberle
//aplicado una función matemática al número binario introducido con switches, las entradas y salidas usadas
//aquí deben ser las del módulo TLD.


//ENTRADAS DEL MODULO sumadorRestadorMultiplicador
//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 2 conectado al switch SW1.
net "numBinario[2]" loc = "K18";//Bit 3 del número binario 2 conectado al switch SW2.
net "numBinario[3]" loc = "K17";//Bit 4 (más significativo) del número binario conectado al switch SW3.

//PUSH BUTTON DEL SELECTOR PARA ELEGIR LA FUNCION MATEMATICA
net "seleccionarOperacion[0]" loc = "H13";//Función Y(x)=2*x+2 conectada al push button BTN3.
net "seleccionarOperacion[1]" loc = "E18";//Función Y(x)=x*x conectada al push button BTN2.
```

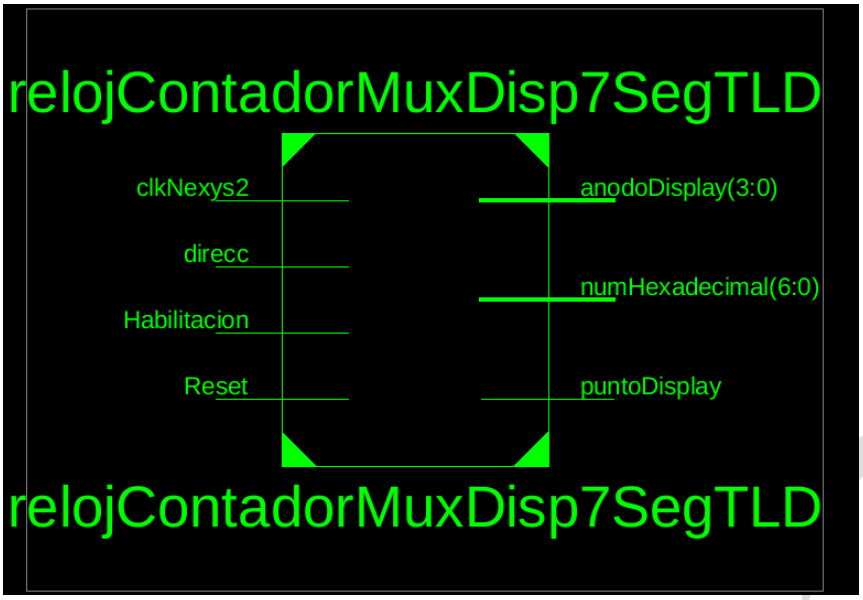
```
//ENTRADAS DEL MODULO divisorDeReloj
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.
net "Reset" loc = "B18";//El botón de Reset está conectado al push button BTN0.


//SALIDAS DEL MODULO TLD: relojContadorAritmeticaMuxDisp7SegTLD para encender leds indicativos.
//Leds A,B,C,D,E,F y G del display de 7 segmentos
net "leds[3]" loc = "K14";//1er bit (más significativo) de la signal resultado conectada al LED LD3.
net "leds[2]" loc = "K15";//2do bit de la signal resultado conectada al LED LD2.
net "leds[1]" loc = "J15";//3er bit de la signal resultado conectada al LED LD1.
net "leds[0]" loc = "J14";//4to bit (menos significativo) de la signal resultado conectada al LED LD0.


//SALIDAS DEL MODULO decodificadorBinHex
//Leds A,B,C,D,E,F y G del display de 7 segmentos
net "ledsDisplay[7]" loc = "L18";//1er bit del vector conectado al puerto CA (led A) del display de 7 segmentos.
net "ledsDisplay[6]" loc = "F18";//2do bit del vector conectado al puerto CB (led B) del display de 7 segmentos.
net "ledsDisplay[5]" loc = "D17";//3er bit del vector conectado al puerto CC (led C) del display de 7 segmentos.
net "ledsDisplay[4]" loc = "D16";//4to bit del vector conectado al puerto CD (led D) del display de 7 segmentos.
net "ledsDisplay[3]" loc = "G14";//5to bit del vector conectado al puerto CE (led E) del display de 7 segmentos.
net "ledsDisplay[2]" loc = "J17";//6to bit del vector conectado al puerto CF (led F) del display de 7 segmentos.
net "ledsDisplay[1]" loc = "H14";//7mo bit del vector conectado al puerto CG (led G) del display de 7 segmentos.
net "ledsDisplay[0]" loc = "C17";//8vo bit del vector conectado al puerto DP, osea el punto del display de 7 segmentos.


//SALIDAS DEL MODULO decodificadorBinHex
//Para que el orden que puse en el código se respete debo 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.
```

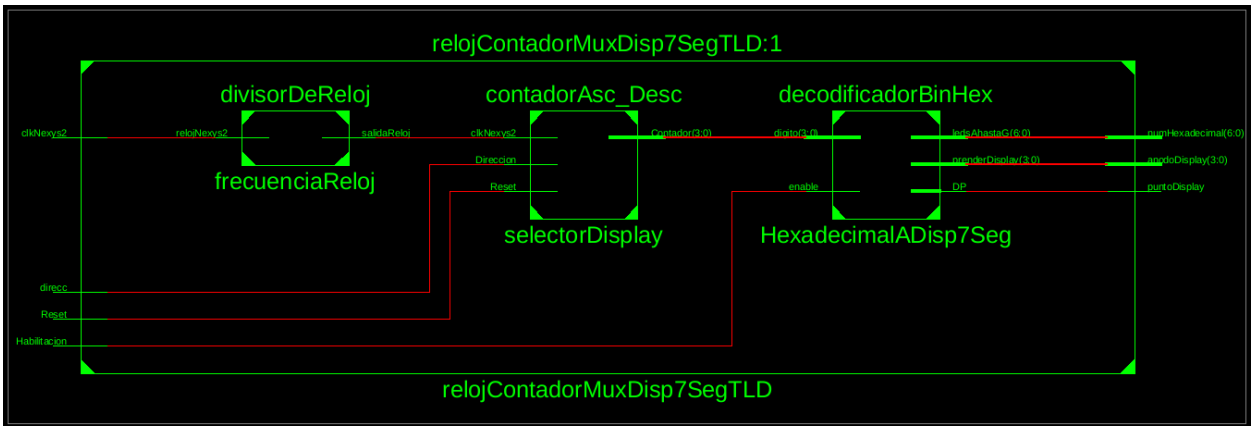
TLD: Contador Hexadecimal Ascendente y Descendente

Se repetirá el mismo concepto de los dos ejercicios anteriores, pero ahora se aplicará para crear un contador hexadecimal ascendente y descendente usando los dos lenguajes de programación Verilog y VHDL:

Código Verilog:

Diagrama TLD:





Divisor de Reloj (DIV):

```
//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.
    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 [24:0]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@(), además 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 operación lógica 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))
        begin
                //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[24];
        //En la tabla de frecuencias podemos ver que la coordenada 24 proporciona una frecuencia de 1.49Hz.
endmodule
```

Contador Ascendente/Descendente y Selector de los 4 Displays (DISP):

```
//2.-CONTADOR DE 4 BITS:
//En este código el contador es de 4 bits, esto implica que contara desde cero hasta 15 en forma binaria.

module contadorAsc_Desc(
    input clkNexys2, //Este es el reloj de 50MHz proveniente del módulo divisorDeReloj.
          input Reset, //Botón de Reset.
          input Direccion, //Botón que fija si el contador es ascendente o descendente.
    output [3:0] Contador //Contador de 4 bits, desde cero hasta el 15 en binario.
    );

        //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 [3:0] conteoAscendenteDescendente = 4'b0000; //Se le da un valor inicial de 0 al vector.
        //4'b0000 está indicando que el valor es de dos (4) números (') binarios (b) con valor (0000).

        //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(clkNexys2))
        //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
                if (Reset == 1'b0) begin
```

```
                    //Si el botón reset está en 0 lógico, el vector conteoAscendenteDescendente puede adoptar dos posibles valores
                              if (Direccion == 1'b0) begin
                              //Si el switch de dirección no está accionado, osea que vale cero, la dirección será ascendente y al
                              //dar clic al reset se empezará el conteo desde 0000.
                                        conteoAscendenteDescendente = 4'b0000;
                              end else begin
                              //Si el switch de dirección está accionado, osea que vale 1 lógico, la dirección será descendente y
                              //al dar clic al reset se empezará el conteo desde F hexadecimal, osea 1111.
                                        conteoAscendenteDescendente = 4'b1111;
                              end
                    end else if (Direccion == 1'b1) begin
                    //Si damos clic al botón dirección, al register conteoAscendenteDescendente se le restara un 1 a lo que lleve
                    //almacenado hasta ese punto.
                              conteoAscendenteDescendente = conteoAscendenteDescendente - 4'b0001;
                    end else begin
                    //Si no damos clic al botón dirección, al register conteoAscendenteDescendente se le sumara un 1 a lo que
                    //lleve almacenado hasta ese punto.
                              conteoAscendenteDescendente = conteoAscendenteDescendente + 4'b0001;
                    end
          end

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

Decodificador Binario a Hexadecimal:

```
//3.-DECODIFICADOR BINARIO A HEXADECIMAL PARA MOSTRARLO EN EL DISPLAY DE 7 SEGMENTOS:

module decodificadorBinHex(
    input [3:0] digito,              //Número binario de 4 bits proveniente del módulo contador.
    input enable,                    //Switch de enable para prender o apagar el led donde se muestra el conteo.
    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 usara en un condicional case.
    output DP                        //Puntito en los displays.
    );

          //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 estará apagado.
          //Los leds del display de 7 segmentos se prenden con 0 lógico y se apagan con 1 lógico.

          //PRENDER O APAGAR EL DISPLAY DE 7 SEGMENTOS CON LA ENTRADA ENABLE
          always@(enable)
          begin : encenderDisplay //El nombre del always@() es encenderDisplay.
                    if(enable == 1'b1) begin//== en Verilog sirve para ver si hay igualdad entre valores.
                              prenderDisplay = 4'b1110; //= en Verilog sirve para asignar un valor.
                              //4'b0000 está indicando que el valor es de cuatro (4) números (') binarios (b) con valor (1110),
                              //esto prenderá solo un display de los 4 disponibles en la tarjeta de desarrollo NEXYS 2.
                    end
                    else begin
                              prenderDisplay = 4'b1111;
                              //Con esto apago todos los displays de 7 segmentos, si el switch de enable no está activo.
                    end
          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
endmodule
```

Módulo TLD:

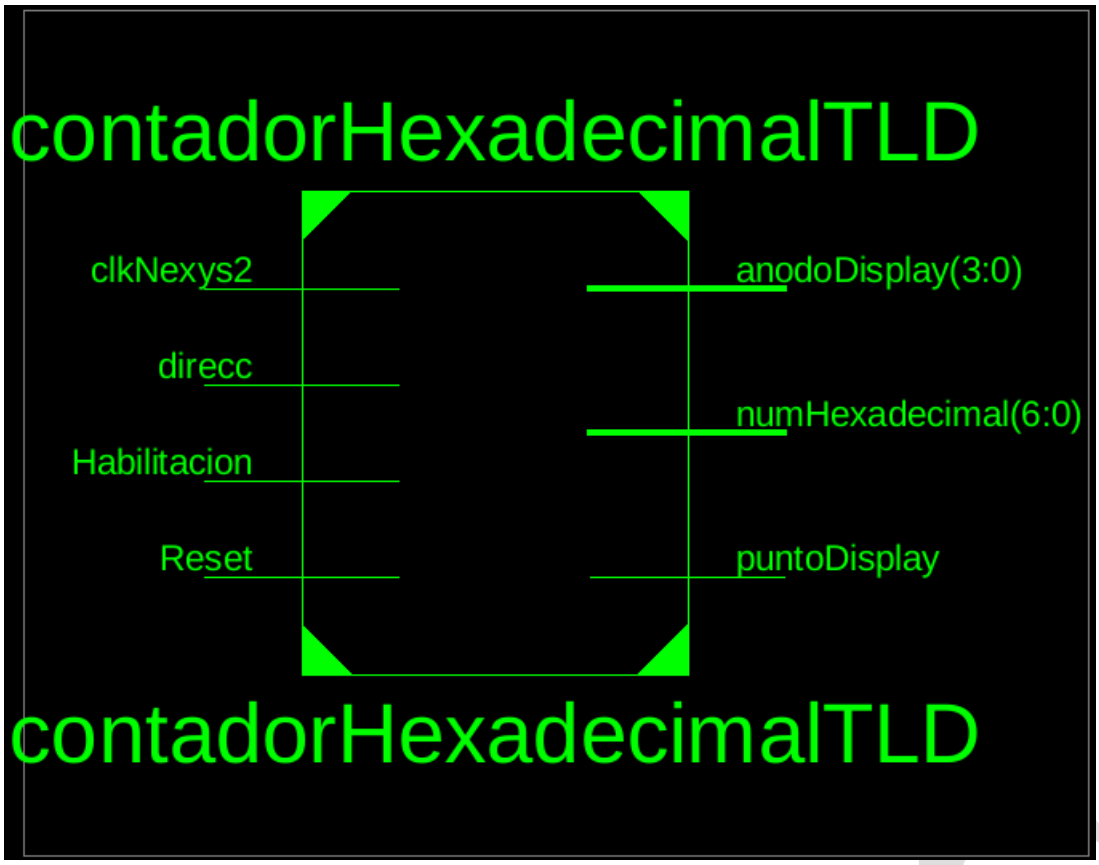
```
//4.-TLD (Top Level Design) relojContadorMuxDisp7SegVHDL:
//Este código sirve para unir los 3 módulos anteriores y poder realizar el decodificador binario a hexadecimal y
//mostrar en los 4 displays de 7 segmentos diferentes valores por medio del divisorDeReloj, el contador Ascendente
//Descendente 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 Habilitacion,

          input direcc,
    input Reset,
    input clkNexys2,
    output [3:0] anodoDisplay,
    output puntoDisplay,
    output [6:0] numHexadecimal
    );

        //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 [3:0] selector;    //Contador/Selector para encender cada uno de los 4 displays 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 frecuenciaReloj(
                        .relojNexys2(clkNexys2),
                        //La entrada relojNexys2 del divisorDeReloj se asigna a la entrada clkNexys2 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 prendera cada display individualmente.
        contadorAsc_Desc selectorDisplay(
                        .clkNexys2(reloj),
                        //A la entrada clkNexys2 del contadorAsc_Desc se le asigna el valor de la wire reloj de este módulo.
                        .Reset(Reset),
                        //La entrada Reset del contadorAsc_Desc se asigna a la entrada Reset de este módulo.
                        .Direccion(direcc),
                        //La entrada Direccion del contadorAsc_Desc se asigna a la entrada Reset de este módulo.
                        .Contador(selector)
                        //La salida contador del contadorAsc_Desc se asigna a la wire selector de este módulo.
                );
        //INSTANCIA DEL MODULO decodBinHex para cambiar de sistema numérico binario a hexadecimal, recibirá el selector del
        //contador y prendera el display dependiendo del número binario que este recibiendo y con la frecuencia del reloj,
        //prendera y apagara los leds que esten en el display.
        decodificadorBinHex HexadecimalADisp7Seg(
                        .digito(selector),
                        //A la entrada digito de decodificadorBinHex se le asigna el valor del wire selector de este módulo.
                        .enable(Habilitacion),
                        //La entrada enable de decodificadorBinHex se asigna a la entrada Habilitacion de este módulo.
                        .prenderDisplay(anodoDisplay),
                        //La salida prenderDisplay de decodificadorBinHex se asigna a la salida anodoDisplay 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
```

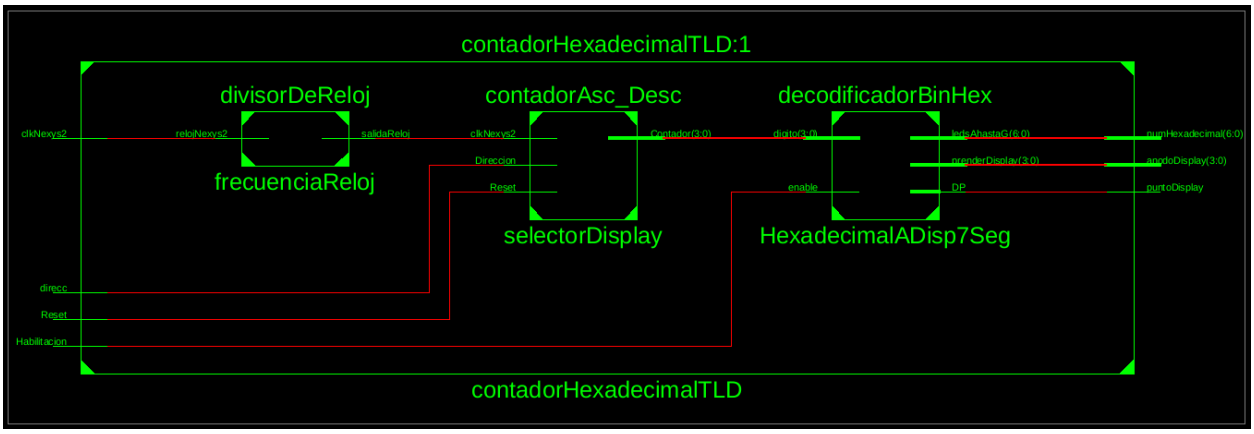
Código UCF:

```
//ENTRADAS DEL MODULO decodificadorBinHex
net "Habilitacion" loc = "G18";//El botón de habilitación o enable está conectado al switch SWO.


//ENTRADAS DEL MODULO Contador
net "direcc" loc = "H18";//El botón de dirección está conectado al switch SW1.
net "Reset" loc = "K18" ;//El Reset está conectado al switch SW2 y reiniciara los valores cuando este en 0 lógico.


//ENTRADAS DEL MODULO divisorDeReloj
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.


//SALIDAS DEL MODULO decodificadorBinHex
//nodo del display de 7 segmentos
net "anodoDisplay[3]" loc = "F17" ;//El nodo AN3 es el que prende el 4to display o 1er display de izquierda a derecha.
net "anodoDisplay[2]" loc = "H17" ;//El nodo AN2 es el que prende el 3er display.
net "anodoDisplay[1]" loc = "C18" ;//El nodo AN1 es el que prende el 2do display.
net "anodoDisplay[0]" loc = "F15" ;//El nodo AN0 es el que prende el 1er display de derecha a izquierda.

//Punto del display de 7 segmentos
net "puntoDisplay" LOC = "C17";//Conectado al puerto DP (punto) del display de 7 segmentos.

//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.
```

Código VHDL:

Diagrama TLD:





Divisor de Reloj (DIV):

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

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.
           salidaReloj : out  STD_LOGIC); --Reloj que quiero con una frecuencia menor a 50MHz.
end divisorDeReloj;

--Arquitectura: Aquí 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 (24 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)
        begin

                if(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 específico, cada coordenada del vector corresponde a una frecuencia en la tabla del divisor de reloj.
        salidaReloj <= divisorDeReloj(24);--En la tabla se ve que la coordenada 24 proporciona una frecuencia de 1.49 Hz.
end frecuenciaNueva;
```

Contador Ascendente/Descendente y Selector de los 4 Displays (DISP):

```
--2.-CONTADOR ASCENDENTE Y DESCENDENTE
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: Declaro como entradas al reloj, al botón reset, dirección, enable y como salida al contador de 4 bits.
entity contadorAsc_Desc is
    Port ( clkNexys2 : in  STD_LOGIC;--Reloj de 50MHz de la NEXYS 2.
           Reset : in  STD_LOGIC;--Botón de Reset.
           Direccion : in  STD_LOGIC;--Botón que indica si el contador es ascendente o descendente.
           Contador : out  STD_LOGIC_VECTOR (3 downto 0)--Contador de 4 bits, desde cero hasta 15 en binario.
                        );
end contadorAsc_Desc;

--Arquitectura: Aquí se realiza el conteo usando una signal, porque en ella se leerán los valores anteriores, para
--sumarle uno y sobrescribirla y así realizar el conteo ascendente.
architecture Behavioral of contadorAsc_Desc is
--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del programa, no
--esta conectada a ningún puerto de la tarjeta de desarrollo y se le asignan valores con el símbolo :=
signal conteoAscendenteDescendente : STD_LOGIC_VECTOR (3 downto 0) := "0000";--Se le da un valor inicial de 0 al vector.
--Se usa una signal porque las salidas no se pueden leer, solo escribir, osea asignarles valores.
begin
        process(clkNexys2, Reset, Direccion)
```

```
        --Process sirve para hacer operaciones matematicas, condicionales o bucles.
        begin--Dentro del paréntesis de process() se deben poner las entradas que usara, tiene su propio begin y end.
                        --La instrucción rising_egde() hace que el condicional se ejecute solo cuando haya un flanco
                        --de subida en la entrada clkNexys2 que recibe el reloj de 50MHz de la NEXYS 2.
                        if(rising_edge(clkNexys2)) then
                                --Cuando el switch del reset no este presionado valdrá 0 lógico y reiniciará el conteo
                                --de la signal conteoAscendente.
                                if(Reset='0') then
                                --Aquí se usan las comillas porque nos estamos refiriendo a un bit con valor 1 lógico.
                                        if(Direccion='0')then
                                                --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".
                                                conteoAscendenteDescendente <= "0000";
                                                --Al dar clic al botón reset, empezara el conteo desde cero si la
                                                --dirección es ascendente.
                                                --Aquí pude haber puesto X"0" en vez de poner "0000".
                                        else
                                                conteoAscendenteDescendente <= "1111";
                                                --Al dar clic al botón reset se empezará desde 15 el conteo si la
                                                --dirección es descendente.
                                                --Aquí pude haber puesto X"F" en vez de poner "1111", después de haber
                                                --tomado el valor inicial declarado arriba de "0000".
                                        end if;
                                elsif(Direccion='1') then
                                        conteoAscendenteDescendente <= conteoAscendenteDescendente - X"1";
                                        --Si damos clic al botón dirección, a la signal conteo se le restar lo que lleve
                                        --almacenado.
                                        --Aquí puse X"1" en vez de poner "0001" por simple facilidad, ya que 4 bits
                                        --binarios equivalen a un número hexadecimal.
                                else
                                        --Por esta instrucción es que debo hacer uso de una signal en vez de usar
                                        --directamente la salida Contador, ya que las salidas no pueden ser leídas, solo
                                        --se les puede asignar un valor.
                                        conteoAscendenteDescendente <= conteoAscendenteDescendente + "0001";
                                        --Aquí pude haber puesto X"1" en vez de poner "0001".
                                        --Es un contador ascendente porque cuenta uno a uno desde cero.
                                end if;
                        end if;
        end process;

        --Fuera del process debo asignar el valor que haya en la signal a la salida de mi modulo
        Contador <= conteoAscendenteDescendente;
end Behavioral;
```

Decodificador Binario a Hexadecimal:

```
--3.-DECODIFICADOR BINARIO A HEXADECIMAL PARA MOSTRARLO EN EL DISPLAY DE 7 SEGMENTOS:
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 ( digito : in  STD_LOGIC_VECTOR (3 downto 0);--Numero binario de 4 bits proveniente del módulo contador.
           enable : in  STD_LOGIC;--Botón de habilitación.
           prenderDisplay : out  STD_LOGIC_VECTOR (3 downto 0);--Nodo común para prender un 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 mostrarEnDisplay of decodificadorBinHex is
begin
        --INICIALIZACION DE VALORES
        DP <= '1';--El punto siempre estará apagado.

        --PRENDER O APAGAR EL DISPLAY DE 7 SEGMENTOS CON LA ENTRADA ENABLE
        OnOffDisplay : process(enable)
        begin
        --El enable prende o apaga directamente al display de 7 segmentos del conteo, se puso de esta manera porque el
        --display se prende con un 0 y se apaga con un 1.
                if(enable = '1') then
                        prenderDisplay <= "1110";
                else
                        prenderDisplay <= "1111";
                end if;
        end process;
```

```
        --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)--A los procesos se les puede poner nombre
        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 código display.
                        when X"C" => ledsAhastaG <= "0110001"; --Pasa de X"C", osea 12, osea 1100 a C en código display.
                        when X"D" => ledsAhastaG <= "1000010"; --Pasa de X"D", osea 13, osea 1101 a d en código display.
                        when X"E" => ledsAhastaG <= "0110000"; --Pasa de X"E", osea 14, osea 1110 a E en código 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;
end mostrarEnDisplay;
```

Módulo TLD:

```
--4.-TLD (Top Level Design):
--Este código sirve para unir los 4 módulos anteriores y poder realizar el decodificador binario a hexadecimal y
--mostrar en 4 displays de 7 segmentos valores hexadecimales diferentes por medio del divisorDeReloj, el contador
--ascendente/descendente 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 contadorHexadecimalTLD is
    Port ( Habilitacion : in  STD_LOGIC;
           direcc : in  STD_LOGIC;
                  Reset : in  STD_LOGIC;
                  clkNexys2 : in  STD_LOGIC;
                  anodoDisplay : out STD_LOGIC_VECTOR (3 downto 0);
                  puntoDisplay : out  STD_LOGIC;
           numHexadecimal : out  STD_LOGIC_VECTOR (6 downto 0));
end contadorHexadecimalTLD;

--Arquitectura: Aquí se declaran las instancias de los demás módulos para poder unir entradas, salidas y signals.
architecture encenderDisplayTLD of contadorHexadecimalTLD 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 (3 downto 0);--Selector de los 4 Displays.

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.
                          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.contadorAsc_Desc port map(
                          clkNexys2 => reloj,
                          --A la entrada clkNexys2 del contadorAsc_Desc se le asigna el valor de la signal reloj de
                          --este módulo.
                          Reset => Reset,
                          --La entrada Reset del contadorAsc_Desc se asigna a la entrada Reset de este módulo.
                          Direccion  => direcc,
```

```
                        --La entrada Direccion del contadorAsc_Desc se asigna a la entrada Reset de este módulo.
                        Contador => selector
                        --La salida contador del contadorAsc_Desc 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 el display dependiendo del número binario que está recibiendo y con la frecuencia del reloj,
        --prender y apagar los leds que están en el display.
        HexadecimalADisp7Seg : entity work.decodificadorBinHex port map(
                        digito => selector,
                        --A la entrada digito de decodificadorBinHex se le asigna el valor de la signal selector de
                        --este módulo.
                        enable => Habilitacion,
                        --La entrada enable de decodificadorBinHex se asigna a la entrada Habilitacion de este módulo.
                        prenderDisplay => anodoDisplay,
                        --La salida prenderDisplay de decodificadorBinHex se asigna a la salida anodoDisplay 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 encenderDisplayTLD;
```

Código UCF:

```
//ENTRADAS DEL MODULO decodificadorBinHex
net "Habilitacion" loc = "G18";//El botón de habilitación o enable está conectado al switch SWO.

//ENTRADAS DEL MODULO Contador
net "direcc" loc = "H18";//El botón de dirección está conectado al switch SW1.
net "Reset" loc = "K18" ;//El Reset está conectado al switch SW2 y reiniciara los valores cuando este en 0 lógico.

//ENTRADAS DEL MODULO divisorDeReloj
net "clkNexys2" loc = "B8" ;//El reloj de 50MHz viene del puerto B8.

//SALIDAS DEL MODULO decodificadorBinHex
//nodo del display de 7 segmentos
net "anodoDisplay[3]" loc = "F17" ;//El nodo AN3 es el que prende el 4to display o 1er display de izquierda a derecha.
net "anodoDisplay[2]" loc = "H17" ;//El nodo AN2 es el que prende el 3er display.
net "anodoDisplay[1]" loc = "C18" ;//El nodo AN1 es el que prende el 2do display.
net "anodoDisplay[0]" loc = "F15" ;//El nodo AN0 es el que prende el 1er display de derecha a izquierda.

//Punto del display de 7 segmentos
net "puntoDisplay" LOC = "C17";//Conectado al puerto DP (punto) del display de 7 segmentos.

//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.
```