INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

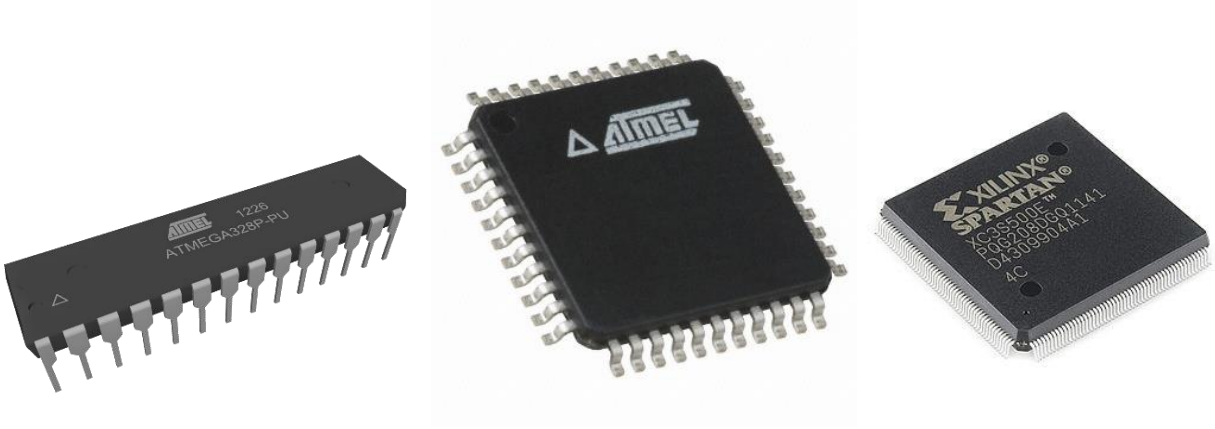
XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

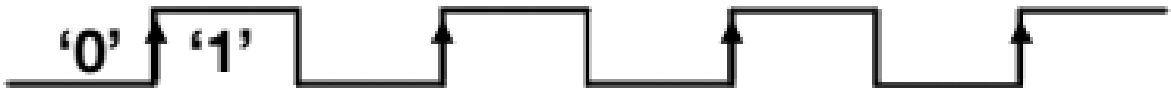
Verilog o VHDL: Señal de Reloj CLK

Contenido

Señal de Reloj o Clock (CLK)

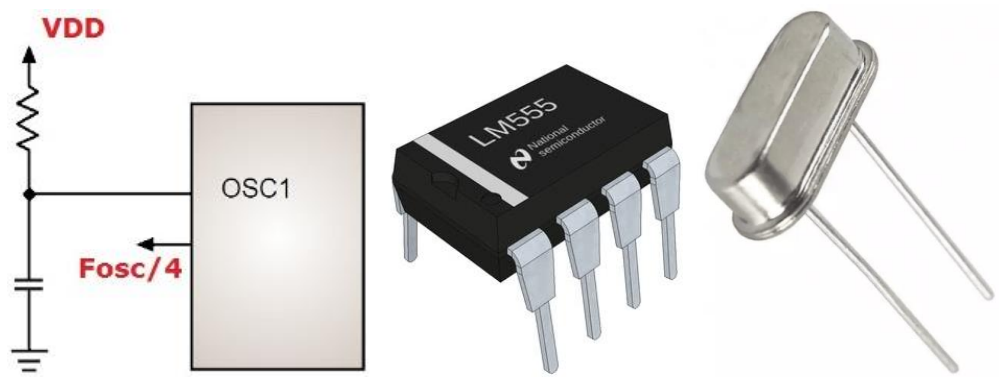
En el mundo de la electrónica digital, la señal CLK (también conocida como CRISTAL OSCILADOR o RELOJ) es simplemente un tren de escalones de unos y ceros lógicos con frecuencia constante, esta señal es de suma importancia porque le dice a todos los circuitos lógicos programables como los microcontroladores, microprocesadores (CPUs) o FPGAs cuantas veces por segundo deben mirar sus pines de entrada, ejecutar su código interno para poder procesarlas y dar una salida en sus pines, esto implica que la señal de reloj determina el rango del tiempo de muestreo en el que el circuito va a operar.





En la programación de micros y FPGAs también es de suma importancia la señal de reloj ya que ésta determinará la velocidad en la que se ejecutará cada línea de código, si no hubiera reloj en el micro o FPGA solo se ejecutaría la primera línea de código y ya, porque no habría una señal que le indique al circuito que debe brincar a la siguiente instrucción del código. El reloj por lo tanto dicta la velocidad de procesamiento, pero esto no implica que directamente podemos calcular las instrucciones por segundo que lee el dispositivo simplemente sacando la inversa de la frecuencia, ya que esto depende también de cada modelo de circuito integrado que estemos programando y de que el código no tenga bucles innecesarios o variables no usadas.

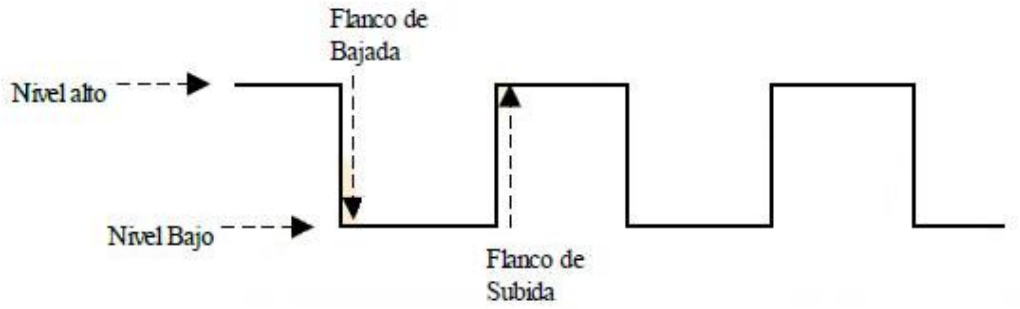
Los osciladores externos que puedo encontrar son variados, existen los osciladores RC que son simplemente un capacitor y una resistencia conectada a Vcc y tierra (este oscilador es solo para frecuencias menores a 1 MHz), se puede usar el circuito integrado LM555, transistores o se puede utilizar un circuito integrado llamado oscilador de cuarzo (este oscilador permite crear una señal de reloj con frecuencias de 1Mhz a 100MHz) que crea señales senoidales, triangulares o de diente de sierra para funcionar como reloj.



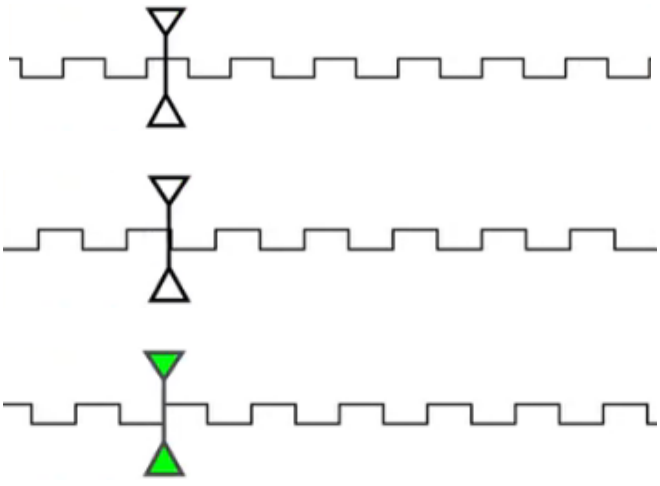
Oscilador RC, Circuito integrado LM555 y Oscilador de cuarzo

En los microprocesadores, microcontroladores o FPGAs el tiempo exacto en el que se hará el muestreo de las entradas ocurrirá en el flanco de subida de la señal o en el flanco de bajada, esto estará especificado para cada circuito programable, pero usualmente se hace en los flancos de subida.

El **flanco de subida** ocurre cuando CLK pasa del 0 al 1 lógico y el **flanco de bajada** ocurre cuando la señal de reloj pasa del 1 lógico (nivel alto) al 0 lógico (nivel bajo).



Saber esto es importante porque los circuitos integrados programables no me darán una respuesta instantánea en t=0, lo harán cuando el reloj se los dicte y esto pasará solo cuando exista un flanco de subida en la señal CLK, cuando la señal de reloj se encuentre en algún otro punto no pasará nada.



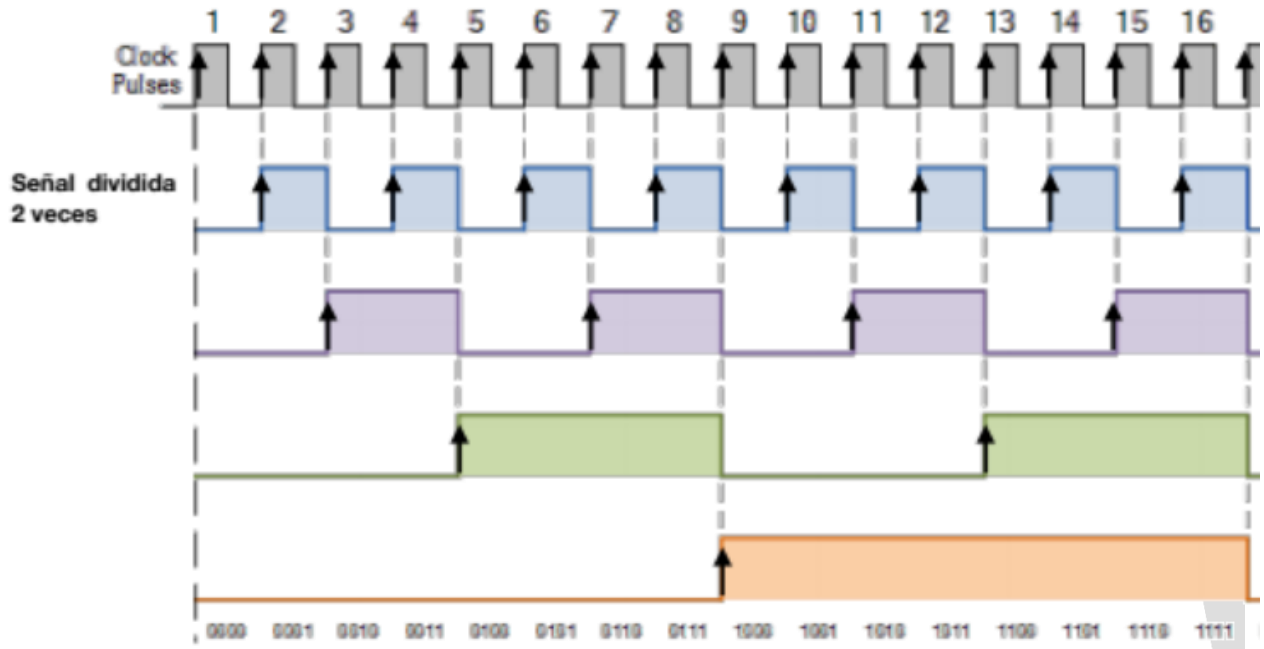
Estas señales de reloj tienen valores predeterminados en los FPGA que utilizamos. En la NEXYS 2 el reloj es de 50 MHz (20ns) y en la NEXYS 3 es de 100 MHz (10ns).

El tiempo se calcula con el inverso de la frecuencia porque está representado por el periodo de la señal de reloj.

$$T = \frac{1}{f} = \frac{1}{50e6} = 2e-8 = 0.00000002 = 20e-9 = 20[ns]$$

Divisor de Reloj en Verilog o VHDL

¿Qué pasa si quiero manejar una señal de reloj con una frecuencia menor a la que proporciona la tarjeta de desarrollo? (porque no puede ser mayor a la dictada por el dispositivo, solo puede ser menor). Para poder lograr esto tenemos que dividir el reloj, en la figura tenemos una señal de reloj en la cual por ejemplo se marcan los flancos de subida (también podemos trabajar con flanco de bajada es decir el cambio de '1' a '0').



Esto debe ser programado en la FPGA con lenguaje VHDL o Verilog para obtener una frecuencia menor a 50MHz en la NEXYS 2 o menor a 100MHz en la NEXYS 3.

En la NEXYS 2 la señal de reloj puede ser obtenida de un puerto llamado B8, esto se indica en el manual y aparece impreso sobre la misma placa de desarrollo.

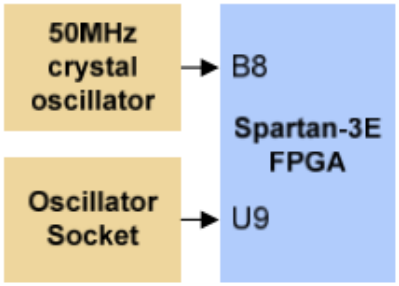
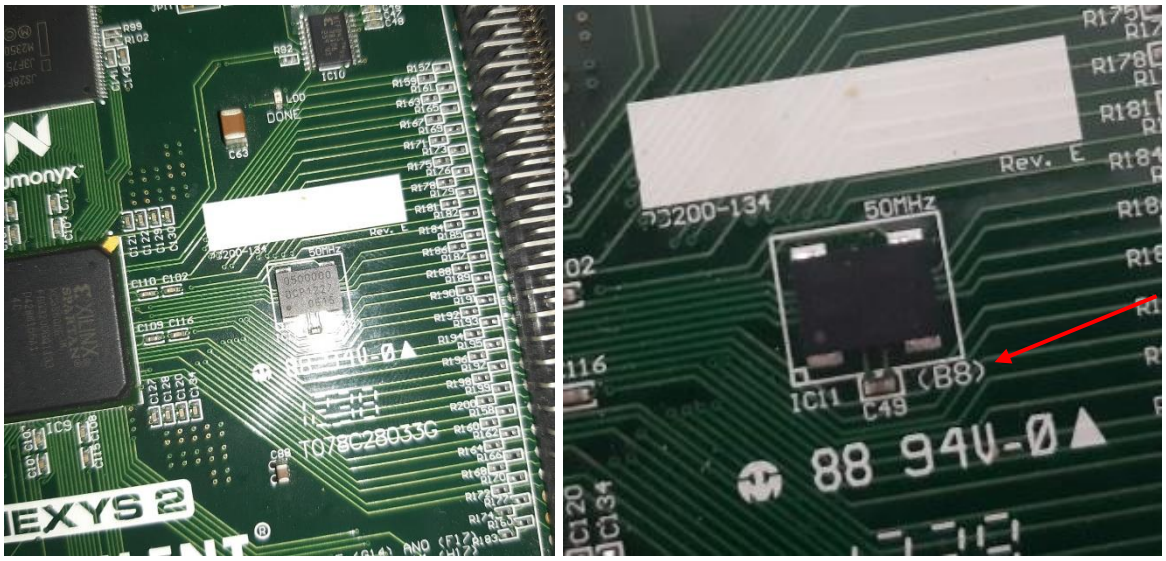


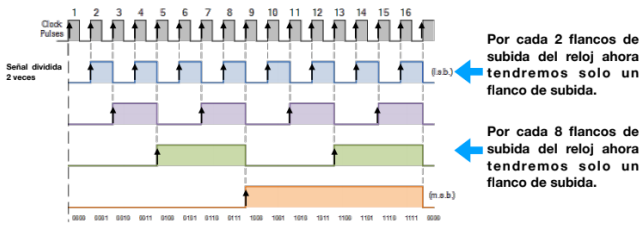
Figure 6: Nexys2 clocks

De este puerto sale la señal de 50MHz y debo hacer que entre a mi código de VHDL o Verilog por medio de una entrada para que pueda modificar su frecuencia.



La entrada por la que ingresará el reloj a nuestro código **debe ser de 1 bit** y solita realizará sus flancos de subida y bajada con una frecuencia de 50MHz sin que yo le tenga que hacer nada por medio de código, pero a veces esta frecuencia es demasiado alta y quiero que mi FPGA proporcione un reloj con una frecuencia más lenta, ya sea para que se ejecute más lentamente la acción programada en mis proyectos o para que cuente números en el ritmo que yo le diga, por eso es que puedo querer cambiar la frecuencia del reloj, esta señal **CLK de salida igual será de 1 bit**.

En la imagen mostrada se puede ver que el divisor de reloj literalmente divide la señal de reloj inicial en partes para obtener una nueva frecuencia menor a la inicial.



Esto dentro del código se hace creando un vector que tenga un tamaño i, osea i coordenadas, dependiendo de la coordenada que agarremos del vector creado, obtendremos una frecuencia diferente, para saber exactamente qué frecuencia se obtendrá se usa la siguiente fórmula:

$$fi = \frac{f}{2^{i+1}}$$

Donde fi es la frecuencia obtenida, i es la coordenada del vector que elegí y f es la frecuencia estándar de la tarjeta de desarrollo:

$$frecuencia\ estandard\ en\ la\ NEXYS\ 2 = 50MHz = 20[ns]$$

$$frecuencia\ estandard\ en\ la\ NEXYS\ 3 = 100MHz = 10[ns]$$

Por ejemplo, podemos ver que si elegimos la coordenada $i = 0$, obtenemos la mitad de la frecuencia que entró siguiendo la fórmula:

$$fi = \frac{f}{2^{i+1}} = \frac{50e6}{2^{0+1}} = 25e6 = 25MHz$$

Para no tener que estar haciendo este cálculo para cada posición del vector creado, existe una tabla con 25 posibilidades de las cuales puedo obtener varias frecuencias:

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

Esto implica que debo crear un vector con 25 posiciones ya sea en VHDL o Verilog y ver qué frecuencias tengo disponibles para elegir en la tabla. El vector de 25 coordenadas creado debe ser de tipo signal si estoy usando VHDL o de tipo reg si estoy usando Verilog, *se usa reg en Verilog porque este tipo de dato se usa para almacenar datos mientras que wire no puede hacer eso*.

La palabra reservada signal **existe solo en VHDL** y es usada para poder **guardar un valor dentro del código que solo pueda vivir durante la ejecución del programa**, sin tener la necesidad de usar una entrada o una salida, ya que estas dos deben estar vinculadas a algún elemento electrónico de la tarjeta de desarrollo, se le puede asignar un valor cualquiera usando el símbolo **:=** junto con el valor que quiero asignar. **Su equivalente en Verilog es el tipo de dato** reg que realiza la misma función.

1) VHDL: Declaración del tipo de dato signal.

```
signal divisorDeReloj: STD_LOGIC_VECTOR (24 downto 0);
--signal tipo vector de 25 bits con bit más significativo en la coordenada 24 y el menos
--en la coordenada 0
```

2) Verilog: Declaración del tipo de dato reg.

```
reg [24:0] divisorDeReloj;
--wire tipo vector de 25 bits con bit más significativo en la coordenada 24 y el menos
--en la coordenada 0
```

Usualmente cuando hago este tipo de programas que varían la frecuencia del reloj, declararemos también una entrada de 1 bit asignada a algún **push button** llamada **Reset**, esta sirve para reiniciar el conteo o el ciclo del reloj. Para realizar este reinicio dentro del código deberemos utilizar un condicional if.

Por lo tanto, lo que debo hacer para hacer el divisor de reloj es crear un vector (tipo signal si estoy usando VHDL o tipo reg si estoy usando Verilog) y meterlo en un condicional if que evaluará si el botón de Reset, está siendo presionado o no, si el botón está siendo presionado el programa llenará al vector de ceros en todos sus 25 bits y si no es el caso entonces el vector ingresará a un condicional else if que sumará un 1 al vector cada vez que la entrada que recibe el reloj proveniente de la NEXYS 2 sufra un flanco de subida, el condicional resultante en ambos lenguajes se muestra a continuación.

Divisor de Reloj en VHDL

1) Condicional if en VHDL:

Se debe crear el vector divisorDeReloj tipo signal para almacenar temporalmente la señal CLK que saldrá de nuestro código VHDL con la frecuencia que quiero en vez de usar directamente una salida out declarada en el código porque hay una parte dentro del condicional donde se debe leer el valor que tenía almacenado previamente divisorDeReloj para luego sumarle uno y asignarlo nuevamente al mismo

vector divisorDeReloj, esto no se puede hacer con los datos tipo out, ya que a las salidas solo se les puede asignar valores, no se les puede leer. Además, para poder realizar esta suma, se debe incluir en el código la librería:

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Previamente habíamos mencionado que el tiempo exacto en el que se hace el muestreo de las entradas ocurre en el flanco de subida de la señal de reloj CLK, esto se debe incluir dentro del código VHDL con la palabra reservada rising_edge( ), lo que hará esta instrucción es que cuando la entrada B8 (que está recibiendo la señal de reloj de 50MHz proveniente de la NEXYS 2) tenga un flanco de subida, el código VHDL lo reconocerá y ejecutará una acción, **esta instrucción se puede poner en cualquier parte del código VHDL**.

```
process (relojDeLaNEXYS2, Reset)
begin
        --El signo = en VHDL es un operador lógico de igualdad
        if (Reset = '1') then
                --El signo <= en VHDL es para asignar un valor
                divisorDeReloj <= "00000000000000000000000000";
        elsif rising_edge(relojDeLaNEXYS2) then
                --Para poder hacer esta operación en VHDL debo incluir la librería
                --use IEEE.STD_LOGIC_UNSIGNED.ALL; en el código
                divisorDeReloj <= divisorDeReloj + 1;
        end if;
end process;

--Fuera del process debo elegir una coordenada de mi signal divisorDeReloj para asignársela a la salida
--llamada salidaReloj, la coordenada elegida del vector divisorDeReloj será la que en la tabla se acerque
--más al valor de frecuencia que deseo obtener y la asignación simplemente se hace con el símbolo <=
salidaReloj <= divisorDeReloj[i];
```

Divisor de Reloj en Verilog

Realizar el divisor de reloj en Verilog es más complejo que en VHDL, se ve a continuación porqué:

2) Condicional if en Verilog:

Se debe crear el vector divisorDeReloj tipo reg para almacenar temporalmente la señal CLK que saldrá de nuestro código escrito en Verilog con la frecuencia que quiero en vez de usar directamente una salida output declarada en el código porque hay una parte dentro del condicional donde se debe leer el valor que tenía almacenado previamente divisorDeReloj para luego sumarle uno y asignarlo nuevamente al mismo vector divisorDeReloj, esto no se puede hacer con los datos tipo output, ya que a las salidas solo se les puede asignar valores, no se les puede leer. *Además, se usa en específico el tipo de dato reg porque*

este tipo de dato se usa para almacenar datos durante la ejecución de un bucle o condicional, mientras que wire *no puede hacer eso*. El uso de un tipo de dato reg también implica que al final cuando le quiera asignar su valor a una salida de mi programa, deberé usar la instrucción assign.

Previamente habíamos mencionado que el tiempo exacto en el que se hace el muestreo de las entradas ocurre en el flanco de subida de la señal de reloj CLK, esto se debe incluir dentro del código escrito en Verilog con la palabra reservada posedge( ), lo que hará esta instrucción es que cuando la entrada B8 (que está recibiendo la señal de reloj de 50MHz proveniente de la NEXYS 2) tenga un flanco de subida, el código VHDL lo reconocerá y ejecutará una acción. Esta instrucción es la que complica hacer el divisor de Reloj en Verilog porque **a fuerza se debe usar dentro del paréntesis de la instrucción** always@( ) **que sirve para poder usar condicionales o bucles y para su ejecución solo se deberá poner el nombre de la entrada, salida o wire a la que se aplicó dentro del paréntesis del condicional o bucle sin compararla con nada ni usar ningún operador lógico**. Esta última condición complica más entender como debo aplicar el condicional if, porque como posedge( ) hace que el condicional o bucle se ejecute por sí solo cuando exista un flanco de subida sin que yo específicamente se lo tenga que indicar dentro del condicional o bucle haciendo uso de operadores lógicos, debo hacer uso de un solo if para indicarle al programa que solamente cuando se ejecute el flanco de subida en el **Reset** se reinicie el conteo, pero para que al divisorDeReloj se le sume un uno, no debo de hacer ningún condicional, solito el always@(posedge( )) ejecuta esta instrucción, si yo la intento poner directamente el programa me da un error.

```
//El hecho de que tanto relojDeLaNEXYS2 como Reset estén usando el operador posedge() implica
//que los condicionales o bucles que usen esas entradas, salidas o wires, se ejecutarán por sí solos sin
//que yo específicamente deba indicar la operación lógica que se debe cumplir para que se ejecuten y
//además solo se ejecutarán cuando pasen del 0 lógico al 1 lógico, esto sirve para ambos casos porque
//el botón Reset al presionarlo pasará del 0 lógico al 1 lógico y el relojDeLaNEXYS2 igual.
always@ (posedge(relojDeLaNEXYS2), posedge(Reset))
begin
        if (Reset) then
                //El signo = en Verilog es para asignar un valor
                divisorDeReloj = 25'b00000000000000000000000000;
                //La instrucción 24'b000000000000000000000000 está asignando 25 números (')
                //binarios (b) con valor (00000000000000000000000000)
        else
                divisorDeReloj = divisorDeReloj + 1;
        end if;
end process;

//Fuera del always@ debo elegir una coordenada de mi reg divisorDeReloj para asignársela a la salida
//salidaReloj, la coordenada elegida del vector divisorDeReloj será la que en la tabla se acerque
//más al valor de frecuencia que deseo obtener y la asignación se hace con la instrucción assign y el
//símbolo =
assign salidaReloj <= divisorDeReloj[i];
```

Finalmente, para poder obtener una frecuencia en específico debo irme a la tabla del divisor de reloj y ver qué frecuencia de la tabla se acerca más a la que quiero, para entonces elegir esa coordenada del vector divisorDeReloj y asignársela a la salida de mi módulo.

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

$$fi = \frac{f}{2^{i+1}}$$

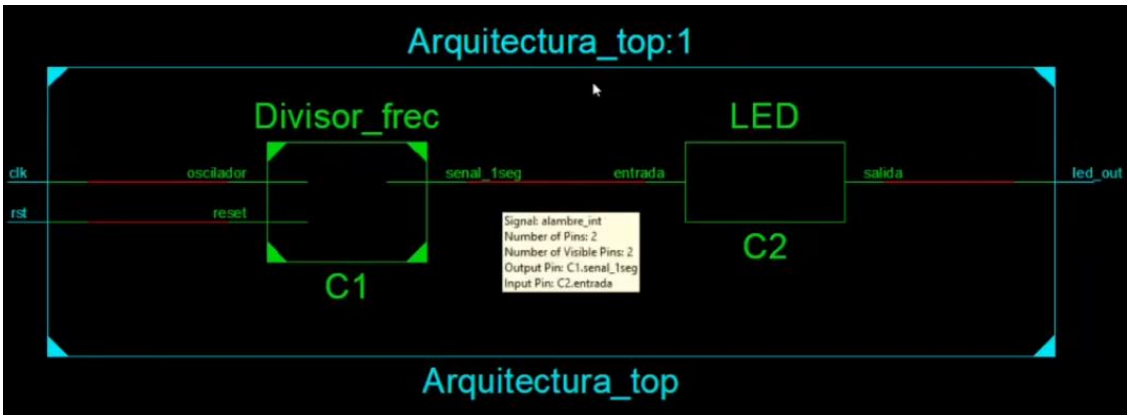
Aunque es más sencillo atenerme a usar solamente las frecuencias mostradas en la tabla, no me tengo que basar solo en ellas, si quiero por ejemplo una frecuencia menor puedo usar la fórmula para aumentar el tamaño del vector divisorDeReloj y obtener una frecuencia menor, por ejemplo:

$fi = \frac{f}{2^{25+1}} = \frac{50e6}{2^{25+1}} = 0.7450\ [Hz]$ y esto hará que el ciclo se repita cada $T = \frac{1}{f} = \frac{1}{0.7450} = 1.34\ [s]$

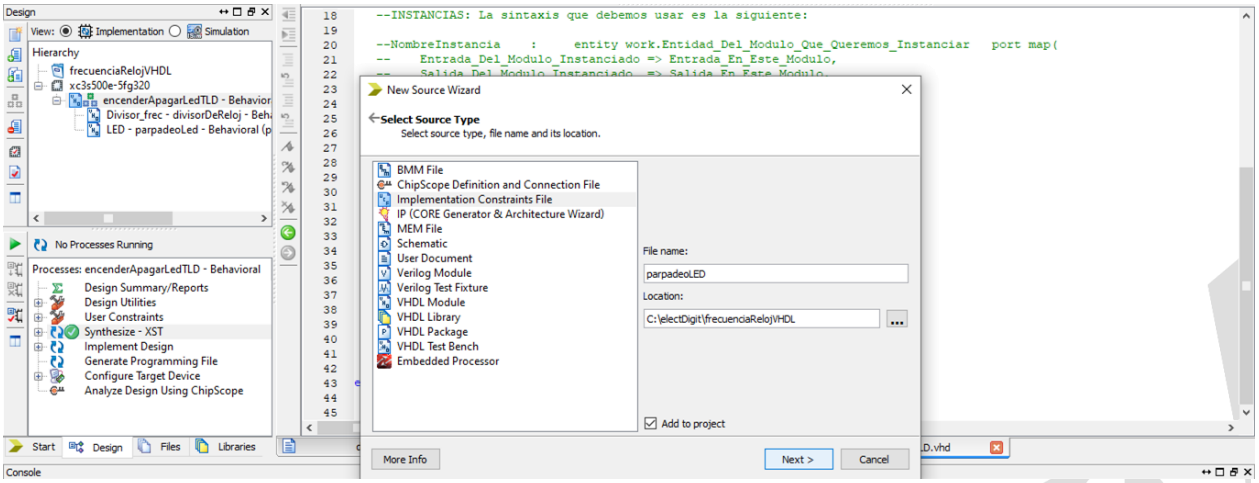
TLD: Hacer que Parpadee un Led

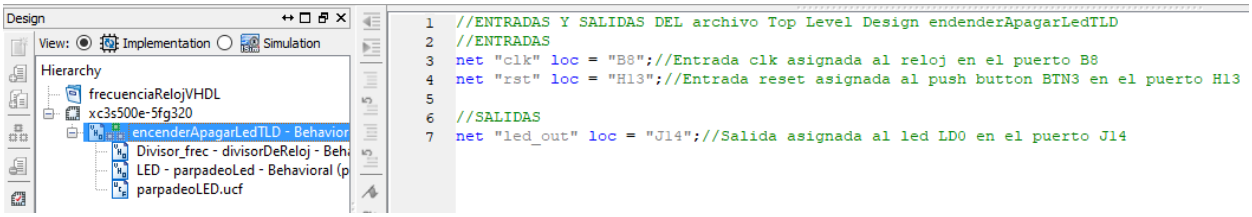
Este tipo de módulos donde modifico la frecuencia del reloj nunca pueden trabajar por sí solos, a fuerza debo unirlos con otros módulos para que puedan ser de utilidad, la acción más simple que un módulo divisor de reloj puede realizar es hacer que parpadee un led.

Para ello deberé crear un módulo aparte que prenda y apague un led y deberé unirlos siguiendo las instrucciones del siguiente diagrama de bloques:

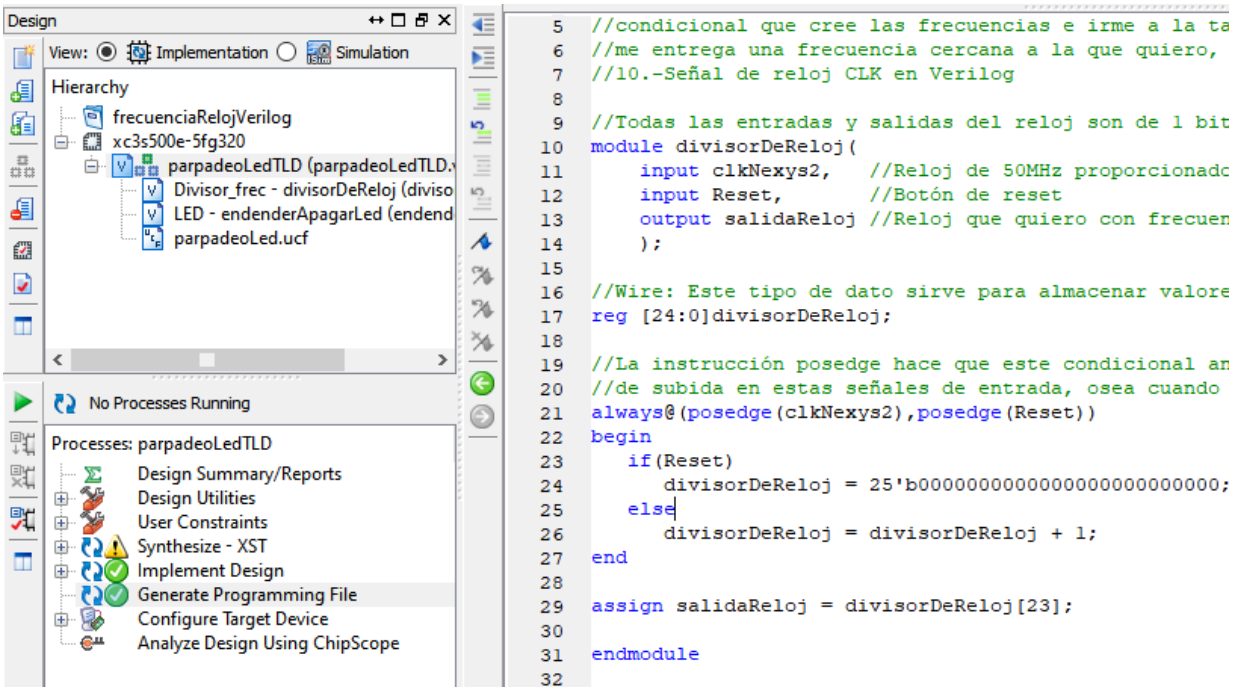


Al archivo TLD es al que le debo asignar un archivo UCF para indicar los puertos del reloj, del reset y del led que quiero que parpadee.





En específico cuando cree este tipo de divisor de reloj, al correrlo me aparecerá una advertencia de Synthesize, no hay que preocuparse por esto ya que solo ocurre porque solo estamos usando un solo bit del vector divisorDeReloj creado y esto le causa algo de conflicto al programa, pero aún funcionará correctamente.



Esta advertencia no aparecerá cuando use la última posición del vector divisorDeReloj por lo antes mencionado.

Código Verilog:

Divisor de Frecuencias Predeterminadas con la Tabla:

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

//En este caso quiero obtener una señal de reloj con frecuencia de 1Hz, osea que cada 1s se repita su ciclo.
//Para poder obtener una señal de reloj con esta frecuencia debo crear el vector divisorDeReloj de 24 bits, crear el
//condicional que establezca sus frecuencias e irme a la tabla para ver (después del condicional if) que coordenada
//del vector me entrega una frecuencia cercana a la que quiero, la tabla mencionada se encuentra en el documento
//8.1.-Senal de reloj CLK en Verilog o VHDL.

//Todas las entradas y salidas del reloj son de 1 bit
module divisorDeReloj(
    input clkNexys2,   //Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
    input Reset,       //Boton de reset.
    output salidaReloj //Reloj que quiero con frecuencia de 1Hz.
    );

          //reg: Este tipo de dato sirve para almacenar valores que solo sobrevivirán durante la ejecución del código y que
          //además se puedan usar dentro de un condicional o bucle.
```

```
        reg [24:0]divisorDeReloj;

        //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 de 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, también si uso un posedge, todas las entradas deben
        //ser activadas igual por un posedge.
        always@(posedge(clkNexys2),posedge(Reset))
        begin
                if(Reset)//Este condicional solo se ejecutara cuando exista un flanco de subida solo en el boton de reset.
                        divisorDeReloj = 25'b0000000000000000000000000;
                //No debo poner el caso cuando if(clkNexys2) porque eso ya lo está haciendo la instrucción always@(posedge()).
                else
                        divisorDeReloj = divisorDeReloj + 1;
        end

        //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
        assign salidaReloj = divisorDeReloj[16];
endmodule
```

Encender y Apagar un Led:

```
//MODULO QUE USA EL DIVISOR DE RELOJ PARA PRENDER Y APAGAR UN LED
module endenderApagarLed(
    input entrada,
    output reg salida//Las salidas usadas dentro de un condicional o bucle las debo declarar como reg.
    );

        //Condicional para encendido y apagado del led.
        always@(entrada)
        begin
                if(entrada==1'b1)
                begin
                        salida=1'b1; //Encender led.
                end
                else
                begin
                        salida=1'b0; //Apagar led.
                end
        end
endmodule
```

Módulo TLD:

```
//TLD: Parpadeo Led
module parpadeoLedTLD(
         //Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto
         //se ve en el documento 8.1.-Senal de reloj CLK en Verilog o VHDL.
         input clk,
         input rst,
         output led_out
         );

        //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 y sirve para poder usar algún valor internamente sin tener que estarlo
        //actualizando como los datos tipo reg.
        wire alambre_int;

        //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
        divisorDeReloj Divisor_frec(
                        .clkNexys2(clk),
                        .Reset(rst),
                        .salidaReloj(alambre_int)
                );

        //INSTANCIA DEL MODULO encenderApagarLed
        endenderApagarLed LED(
                        .entrada(alambre_int),
                        .salida(led_out)
                );
Endmodule
```

Código UTF:

```
//ENTRADAS Y SALIDAS DEL archivo Top Level Design endenderApagarLedTLD
//ENTRADAS:
net "clk" loc = "B8";//Entrada clk asignada al reloj en el puerto B8.
net "rst" loc = "H13";//Entrada reset asignada al push button BTN3 en el puerto H13.

//SALIDAS:
net "led_out" loc = "L15";//Salida asignada al puerto JA1.
```

Contadores Ascendentes y Descendentes

Posedge (Verilog) y Rising Edge (VHDL)

Las instrucciones always@(posedge( )) en Verilog y rising_edge( ) en VHDL utilizadas para crear el divisor de reloj en ambos lenguajes no solamente sirven para eso, el hecho de indicar en un código que cuando ocurra un flanco de subida del reloj, se ejecute una acción, se puede utilizar en una infinidad de aplicaciones, una de ellas es por ejemplo crear un contador ascendente o descendente que cuente en función del periodo de la señal de reloj, que es el inverso de su frecuencia.

A continuación, se llevarán a cabo los mismos ejercicios de contador ascendente y descendente con un botón de reset para reiniciar el conteo, realizados con los lenguajes VHDL y Verilog para que se noten las diferencias al usar ambos en una misma tarea.

Código Verilog:

Contador Ascendente:

```
//CONTADOR ASCENDENTE
module contadorAscendente(
    input clk,//Reloj de 50MHz de la NEXYS 2.
    input rst,//Botón de Reset.
    output [3:0] contador//Contador de 4 bits, desde 0 hasta 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] conteoAscendente = 4'b0000;//Se le da un valor inicial de 0 al vector
        //4'b0000 esta indicando que el valor es de cuatro (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 usar y además tiene su propio begin y end.
        always@(posedge(clk), posedge(rst))
        //La instrucción posedge hace que este condicional se ejecute solo cuando ocurra un flanco de subida en las entradas
        //clk o rst, osea cuando pasen de 0 lógico a 1 lógico, además la instrucción posedge() hará que el código se ejecute
        //solo, sin que yo directamente tenga que indicarlo con una operación lógica o un selector, si quiero que se ejecute
        //algo en específico cuando se dé el flanco de subida solo en una de las entradas del always, debo meter el nombre de
        //esa entrada en el paréntesis de un condicional o bucle.
        begin
                if(rst)//Este condicional se ejecutará cuando exista un flanco de subida solo en el boton de reset.
                        //Aquí puedo usar un digito hexadecimal que equivale a 4 bits binarios, esto se usa para
                        //escribir menos.
                        conteoAscendente = 1'h0;//Es un contador ascendente porque empieza a contar desde cero.
                        //1'h0 está indicando que el valor es un (1) numero (') hexadecimal (h) con valor (0), pero como
                        //un digito hexadecimal equivale a 4 bits binarios, es como si pusiera 0000.
                else
                //No debo poner el caso cuando if(clkNexys2) porque eso ya lo está haciendo la instrucción always@(posedge()).
                        conteoAscendente = conteoAscendente + 4'b0001;
                        //Aquí pude haber puesto 1'h1 en vez de poner 4'b0001, osea 0001.
                        //Es un contador ascendente porque cuenta uno a uno desde cero.
        end

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

Contador Descendente:

```
//CONTADOR DESCENDENTE
module contadorDescendente(
    input clkNexys2,//Reloj de 50MHz de la NEXYS 2.
    input reset,//Botón de Reset.
    output [3:0] contador//Contador de 4 bits, desde 15 hasta 0 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] conteoDescendente = 4'b1111;//Se le da un valor inicial de 15 al vector.
        //4'b1111 esta indicando que el valor es de cuatro (4) números (') binarios (b) con valor (1111).

        //always@() sirve para hacer operaciones matemáticas, condicionales o bucles, dentro de su paréntesis se deben poner
        //las entradas que usara y ademas tiene su propio begin y end.
        always@(posedge(clkNexys2), posedge(reset))
        //La instruccion posedge hace que este condicional se ejecute solo cuando ocurra un flanco de subida en las entradas
        //clk o rst, osea cuando pasen de 0 lógico a 1 lógico, ademas la instrucción posedge() hará que el código se ejecute
        //solo, sin que yo directamente tenga que indicarlo con una operación lógica o un selector, si quiero que se ejecute
        //algo en específico cuando se de el flanco de subida solo en una de las entradas del always, debo meter el nombre de
        //esa entrada en el paréntesis de un condicional o bucle.
        begin
                if(reset)//Este condicional se ejecutará cuando exista un flanco de subida solo en el boton de reset.
                        //Aquí puedo usar un digito hexadecimal que equivale a 4 bits binarios, esto se usa para
                        //escribir menos.
                        //Es un contador descendiente porque empieza a contar desde 15, osea F en hexadecimal.
                        conteoDescendente = 1'hF;
                        //1'hF está indicando que el valor es de un (1) numero (') hexadecimal (h) con valor (F), pero
                        //como un digito hexadecimal equivale a 4 bits binarios, es como si pusiera el 15 decimal,
                        //osea 1111.
                else
                //No debo poner el caso cuando if(clkNexys2) porque eso ya lo está haciendo la instrucción always@(posedge()).
                        conteoDescendente = conteoDescendente - 4'b0001;
                        //Aquí pude haber puesto 1'h1 en vez de poner 4'b0001, osea 0001.
                        //Es un contador descendiente porque cuenta uno a uno desde 15.
        end
        //En Verilog para poder asignar el valor de un reg a una salida debo usar la palabra reservada assign.
        assign contador = conteoDescendente;
endmodule
```

Código VHDL:

Contador Ascendente:

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

--Entidad: Declaro como entradas al reloj y al botón de reset y como salida al contador de 4 bits.
entity contadorAscendente is
    Port ( clkNexys2 : in  STD_LOGIC;--Reloj de 50MHz de la NEXYS 2.
           Reset : in  STD_LOGIC;--Botón de Reset.
           Contador : out  STD_LOGIC_VECTOR (3 downto 0)--Contador de 4 bits, desde 0 hasta 15 en binario.
                    );
end contadorAscendente;

--Arquitectura: Aquí se realiza el conteo usando una signal porque en ella leer los valores anteriores para
--sumarle uno y sobrescribirla para realizar el conteo ascendente.
architecture Behavioral of contadorAscendente is
--SIGNAL: Existe solo en VHDL y sirve para almacenar datos que solo sobrevivirán durante la ejecución del programa,
--no esta conectada a ningn puerto de la tarjeta de desarrollo y se le asignan valores con el símbolo :=
signal conteoAscendente : 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.
begin
        process(clkNexys2, Reset)--Process sirve para hacer operaciones matemáticas, condicionales o bucles.
        begin--Dentro del paréntesis de process() se deben poner las entradas que usar, 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 push button reset este presionado valdrá un 1 lógico porque hace corto circuito y
                        --reinicia el conteo de la signal conteoAscendente.
                        if(Reset='1') then
                        --Aquí se usan las comillas porque nos estamos refiriendo a un bit con valor 1 lógico.
                                --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 número
                                hexadecimal poniendo X"numero hexadecimal".
                                conteoAscendente <= "0000";--Es un contador ascendente porque empieza a contar desde cero.
                                --Aquí pude haber puesto X"0" en vez de poner "0000".
                        elsif(conteoAscendente >= "1000") then
                                conteoAscendente <= "0000";
                                --El conteo se puede hacer solo hasta un número, en este caso es hasta el 8.
```

```
                        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 leidas, solo se les puede asignar
                                --un valor.
                                conteoAscendente <= conteoAscendente + "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 módulo.
        Contador <= conteoAscendente;
end Behavioral;
```

Contador Descendente:

```
--CONTADOR DESCENDENTE
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Librerias que sirven solamente para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librera para poder realizar operaciones matemáticas sin considerar el signo.

--Entidad: Declaro como entradas al reloj y al botón de reset y como salida al contador de 4 bits.
entity contadorDescendente is
    Port ( CLK : in  STD_LOGIC;--Reloj de 50MHz de la NEXYS 2.
           Rst : in  STD_LOGIC;--Botón de Reset.
           Contador : out  STD_LOGIC_VECTOR (3 downto 0));--Contador de 4 bits, desde el 15 hasta el 0 en binario.
end contadorDescendente;

--Arquitectura: Aquí realizar el conteo usando una signal porque en ella leer los valores anteriores para
--restarle uno y sobrescribirla para realizar el conteo descendente.
architecture Behavioral of contadorDescendente 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 conteoDescendente : STD_LOGIC_VECTOR (3 downto 0) := "1111";--Se le da un valor inicial de 15 al vector.
--Se usa una signal porque las salidas no se pueden leer, solo escribir.
begin
        process(CLK, Rst)--Process sirve para hacer operaciones matemáticas, condicionales o bucles.
        begin--Dentro del paréntesis de process() se deben poner las entradas que usar, 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(CLK)) then
                        --Cuando el push button reset esta presionado valdrá un 1 lógico porque hace corto circuito y
                        --reinicia el conteo de la signal conteoAscendente.
                        if(Rst='1') then
                        --Aquí se usan las comillas porque nos estamos refiriendo a un bit con valor 1 lógico.
                                --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
                                --número hexadecimal poniendo X"numero hexadecimal".
                                conteoDescendente <= "1111";
                                --Es un contador descendente porque empieza a contar desde el 15.
                                --Aquí pude haber puesto X"F" que vale 15 en decimal en vez de poner "1111".
                        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 leidas, solo se les puede asignar
                                --un valor.
                                conteoDescendente <= conteoDescendente - "0001";
                                --Aquí pude haber puesto X"1" en vez de poner "0001".
                                --Es un contador ascendente porque cuenta uno a uno desde 15.
                        end if;
                end if;
        end process;

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