INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

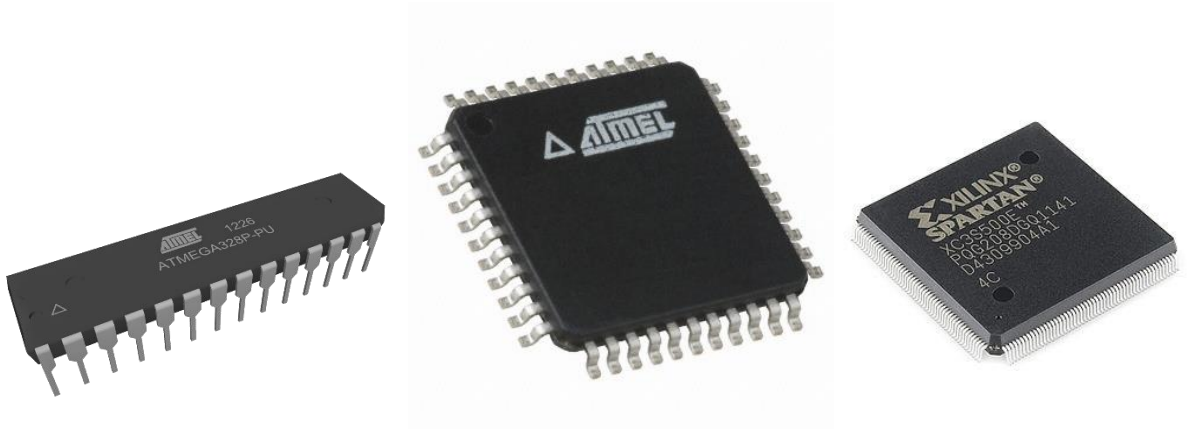
XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

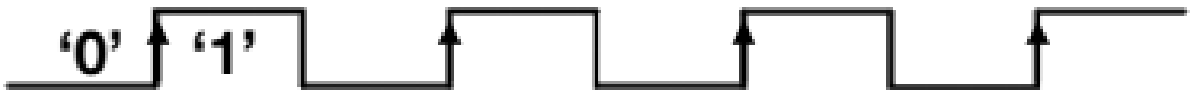
Solamente VHDL: Señal de Reloj CLK

Contenido

Señal de Reloj o Clock (CLK)

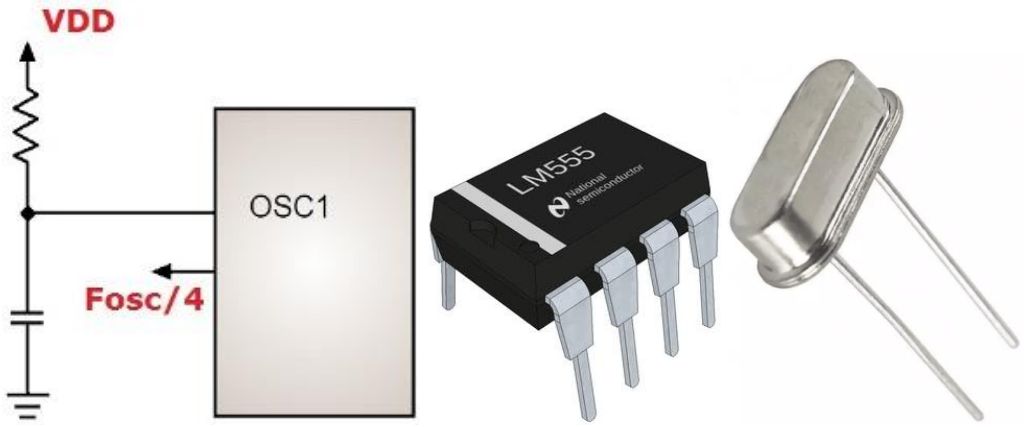
En el mundo de la electrónica digital, la señal CLK (también conocida como CRISTAL OSCILADOR o RELOJ) es simplemente un tren de escalones de unos y ceros lógicos con frecuencia constante, esta señal es de suma importancia porque le dice a todos los circuitos lógicos programables como los microcontroladores, 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.





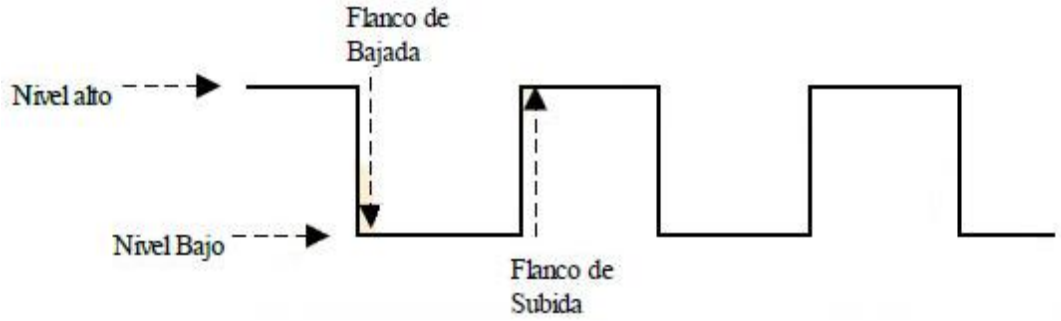
En la programación de micros y FPGAs también es de suma importancia la señal de reloj ya que esta determinará la velocidad en la que se ejecutará cada línea de código, si no hubiera reloj en el micro o FPGA solo se ejecutaría la primera línea de código y ya, porque no habría una señal que le indique 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, se puede usar 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.

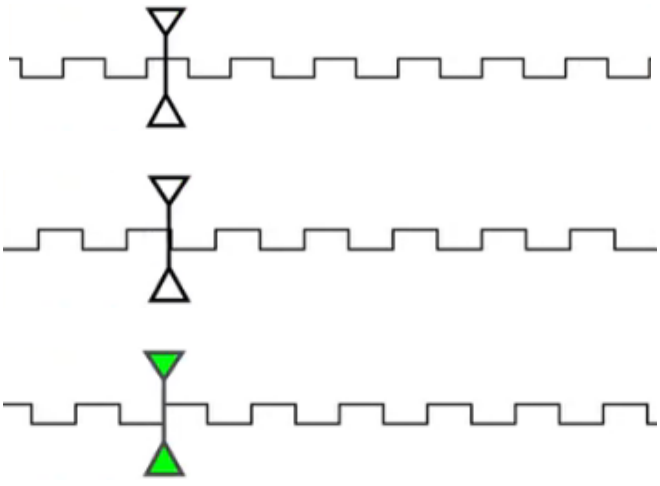


En los microprocesadores, microcontroladores o FPGAs el tiempo exacto en el que se hará el muestreo de las entradas ocurrirá en el flanco de subida de la señal o en el flanco de bajada, esto estará especificado para circuito integrado, pero usualmente se hace en los flancos de subida.

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 Solamente Aplicable en 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 crear un contador, esto creará una señal de reloj interno con un nivel alto muy pequeño y por medio de esta señal es que se crea la señal de reloj con la frecuencia que deseo.

Señal de reloj interno

Señal de reloj con la frecuencia que yo quiero

Esto puede ser programado en la FPGA (con lenguaje VHDL solamente) 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.

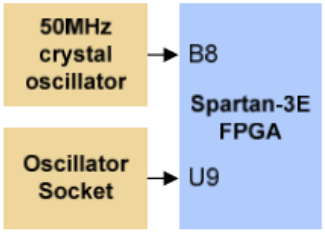
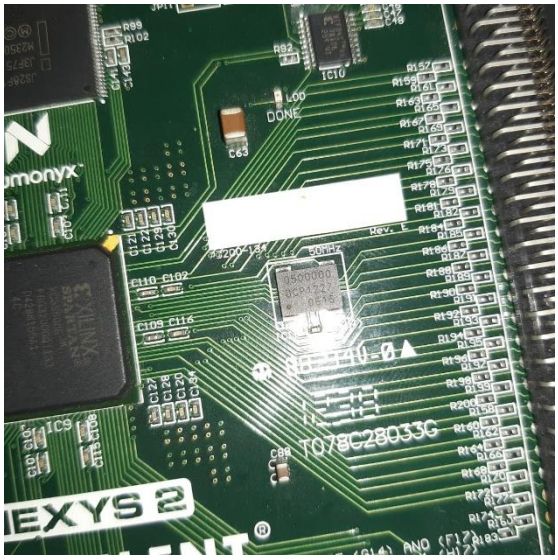


Figure 6: Nexys2 clocks

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 in 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** y el divisor de reloj en VHDL se realiza de la siguiente manera:

Primero que nada, debo crear un elemento llamado signal en el código para que me realice un conteo, este $Contador$ dependerá de la frecuencia final que quiero obtener y lo usaré para poder crear la señal de reloj interno que solo tendrá un pequeño nanosegundo de nivel alto, eventualmente la señal de reloj interno me servirá para poder marcarle el paso a la señal de reloj que deseo obtener.

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 les puede asignar un valor cualquiera usando el símbolo **:=** junto con el valor que quiero asignar.

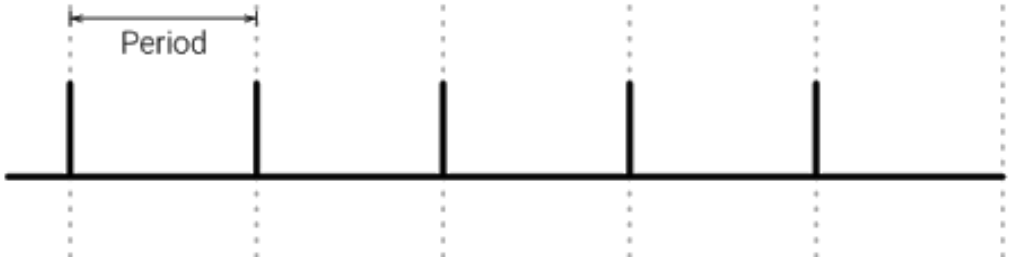
$$Conteo' = \frac{frecuencia\ estandard}{frecuencia\ deseada}$$

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

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

$$Contador = \frac{Conteo'}{2} - 1$$

El resultado de la variable $Contador$ creará la siguiente señal:



Usualmente cuando hago este tipo de programas que varían la frecuencia del reloj, declararemos 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.

Dentro del código VHDL deberé crear dos process diferentes, en el primero usaré esta variable $Contador$ para crear la señal de reloj interno, la señal de reloj interno igual deberá estar almacenada en un signal.

Por lo tanto, lo que debo hacer para poder introducir esta variable $Conteo\ real$ a mi código es declarar una signal e indicar que sea de tipo integer (osea tipo de dato decimal entero). La sintaxis para declarar una signal e indicar que es un dato tipo integer es la siguiente:

```
signal nombreSignal1: STD_LOGIC; --signal de 1 bit;
signal nombreSignal2: STD_LOGIC_VECTOR (2 downto 0); --signal tipo vector
signal nombreSignal3: integer range numInicial to numFinal := numInicial;
--signal tipo integer o entero con rango de valores predefinido
signal nombreSignal4: integer;
--signal tipo integer o entero sin rango de valores predefinido
```

El primer process deberá realizar un conteo desde cero hasta el resultado que haya obtenido de $Contador$ dependiendo de la frecuencia que quiera obtener.

Ejemplo:

1) Si por ejemplo quiero obtener un reloj que sea de 400Hz en mi NEXYS 2, la operación y el código que deberé introducir será el siguiente:

$$Conteo' = \frac{50MHz}{frecuencia\ deseada}$$

$$Conteo' = \frac{50MHz}{400}$$

$$contador = \frac{Conteo'}{2} = \frac{125000}{2} - 1 = 62499$$

Por lo tanto, en el código deberé meter la siguiente signal:

```
signal contador: integer range 0 to 62499 := 0;
 --Esta signal lo que hará es realizar el conteo desde cero hasta la variable conteo real
--para después meterla a un process y obtener la señal de reloj que quiero.
```

2) Si por ejemplo quiero obtener un reloj que sea de 1Hz en mi NEXYS 2, la operación y el código que deberé introducir será el siguiente:

$$Conteo' = \frac{50MHz}{frecuencia\ deseada}$$

$$Conteo' = \frac{50e6}{1}$$

$$contador = \frac{50e6}{1} - 1 = \frac{50000000}{2} - 1 = 24999999$$

Por lo tanto, en el código deberé meter la siguiente signal:

```
signal contador: integer range 0 to 24999999 := 0;
--Esta signal lo que hará es realizar el conteo desde cero hasta la variable conteo real
--para después meterla a un process y obtener la señal de reloj que quiero.
```

Se puede declarar la signal $Contador$ como un integer range para tener mayor certeza de que el valor máximo del conteo no se vea nunca rebasado y la frecuencia entregada sea la correcta, pero por facilidad es mejor implemente declarar la signal $Contador$ como un integer e introducir el valor de $Contador$ directamente en el paréntesis del condicional if.

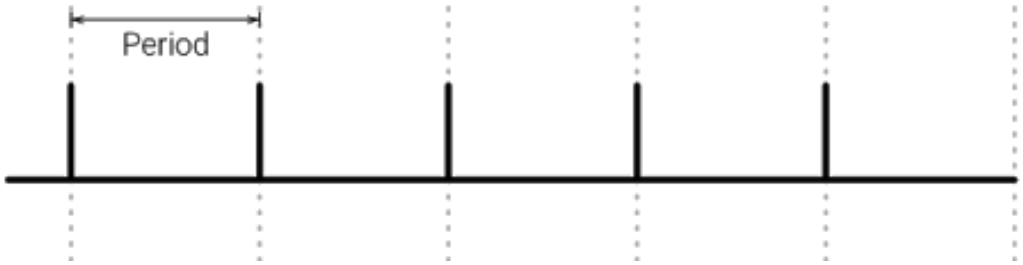
Aparte de la signal contador que representa el valor de $Conteo\ real$ deberé crear otra signal llamada **clk_interno** que sea de 1 bit y almacene la señal de reloj interno antes mencionada que solo mantiene su nivel alto por un nanosegundo.

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 con la palabra reservada rising_edge, lo que está indicando esta instrucción es que cuando la entrada B8 que está recibiendo la señal de reloj (de 50MHz de la NEXYS 2) tenga un flanco de subida, el código lo reconocerá y ejecutará una acción.

```
--Las entradas que vaya a usar en el condicional o bucle se ponen en su paréntesis (llamado sensitivity
--list) separadas por comas, estas deben ser el reloj y el botón de reset
process (relojDeLaNEXYS2, reset)
```

```
begin
        --Dentro del condicional if es donde se crea la señal de reloj interna de la siguiente manera:
        if (rising_edge(relojDeLaNEXYS2)) then
                --El signo = en VHDL es un operador lógico de igualdad
                --Se pone con comillas el valor porque representa 1 bit con valor de 0 lógico
                if (reset = '1') then
                        --Se pone sin comillas el valor porque representa un número decimal
                        contador <= 0; --El signo <= en VHDL es para asignar un valor
                --En vez de declarar la signal tipo integer con un rango de valores predefinido puedo
                --ponerlo solo directamente en la condición del if y así puedo editar la frecuencia de mi
                --reloj simplemente cambiando el valor numFinal en esta instrucción del código
                elsif (contador = numFinal) then
                        contador <= contador + 1;
                        clk_interno <= contador + 1;
                end if;
        end if;
end process;
```

Esta es la señal interna creada por el primer process:

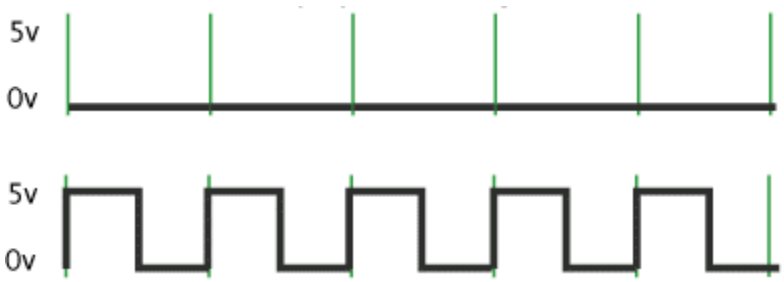


Ahora si se hará el segundo process que creará la señal de reloj, para ello se debe crear otra signal llamada **clk_salida** que almacenará temporalmente la señal CLK que saldrá de nuestro código VHDL con la frecuencia que quiero. Esto se debe hacer usando una signal en vez de usar directamente la salida out declarada en el código porque dentro del segundo process hay una parte en la que se debe leer el valor que tenía almacenado previamente **clk_salida** para luego negarlo y asignarlo nuevamente a la misma **clk_salida** y 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, si quisiera hacer esto directamente tendría que declarar la salida como inout, osea como entrada o salida de tipo bidireccional y esto puede traer más problemas.

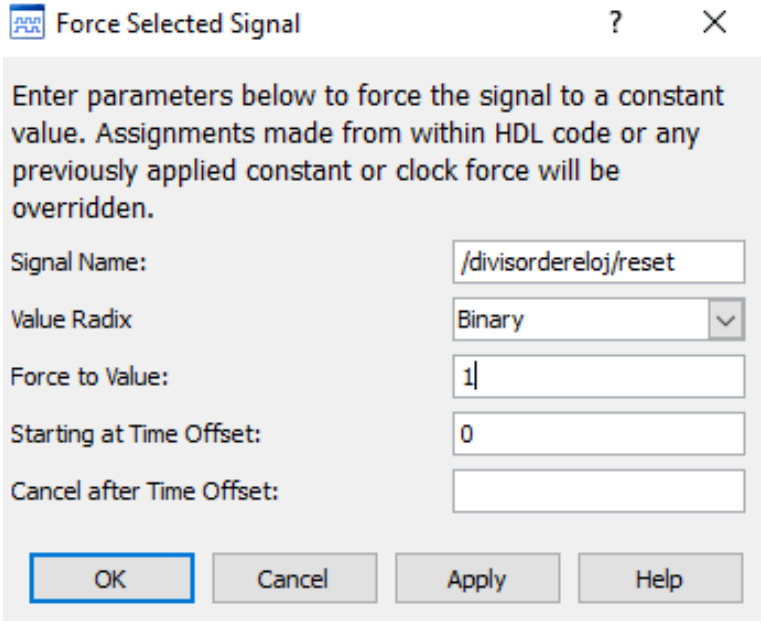
```
process (clk_interno, reset)
begin
        --El botón de reset vale 0 lógico cuando lo presiono
        --Se pone con comillas el valor porque representa 1 bit con valor de 0 lógico
        if (reset = '1') then
                --Se pone con comillas el valor porque representa un 0 lógico
                clk_salida <= '0'; --El signo <= en VHDL es para asignar un valor
        elsif rising_edge (clk_interno) then
                --Gracias a esta línea de código es que se debe declarar la segunda signal para
                --almacenar el valor de la señal de salida
```

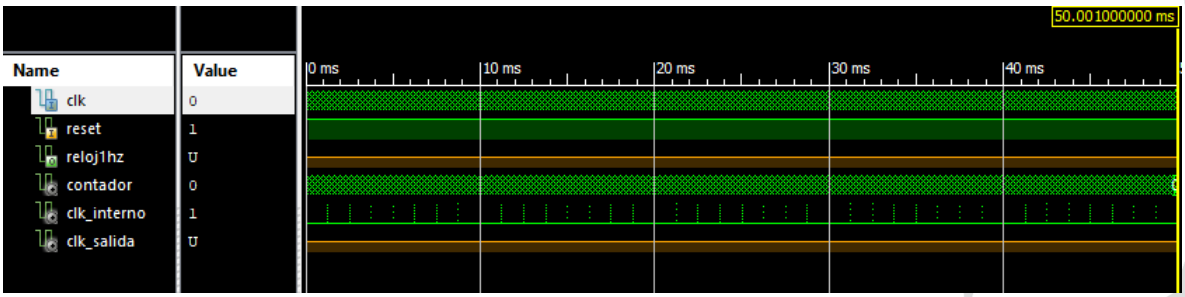
```
                clk_salida <= not (clk_salida);
        end if;
end process;

--Fuera del segundo process debo asignar lo que tiene almacenado la signal clk_salida a la salida del reloj
salidaReloj <= clk_salida;
```

Ya con esta salida obtengo la señal que yo quería haciendo uso de la señal de reloj interna:



Esto en el código funciona bien, pero al simularlo tengo algunos problemas ya que si declaro la entrada reset como 1 siempre no se ve la señal de reloj de salida:



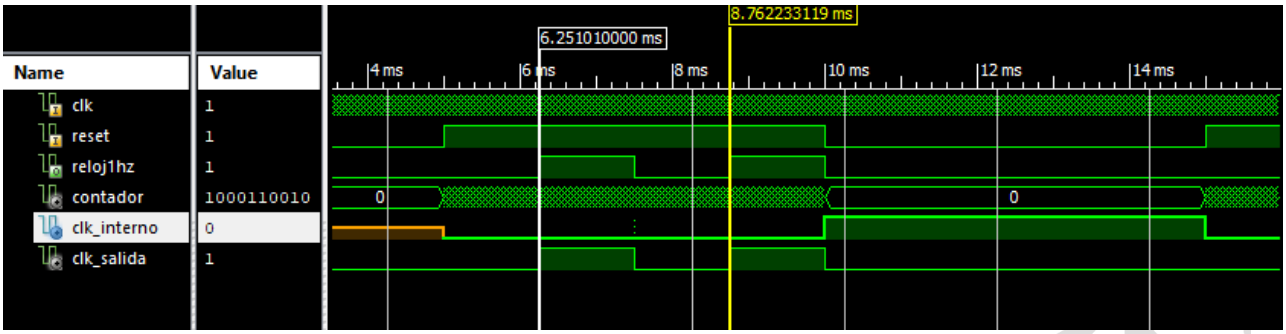


Pero si hago que el reset esté oscilando entre el 1 y el 0 lógico si se alcanza a ver, por ejemplo, este es el reloj de salida medido para una frecuencia de 400Hz, ya que el simulador no alcanza a simular más de 500ms y la señal de reloj de 1Hz requiere más de 1s para poder verla:

En esta el contador llega hasta 62499 y su periodo debe ser de 2.5ms, esto lo puedo medir añadiendo un cursor extra en la simulación en el botón que dice Add marker

```
29  --quiero obtener
30  architecture Behavioral of divisorDeReloj is
31
32  signal contador : integer range 0 to 62499 := 0;--El contador sirve para crear la señal de reloj interno
33  signal clk_interno : STD_LOGIC;--la señal de reloj interno solo su nivel alto por un nanosegundo
34  --se crea la señal de reloj que quiero usando la señal de reloj interno y se almacena en esta signal para
35  --posteriormente asignarsela a la salida, se hace uso de una signal y no se almacena directamente en la salida
36  --reloj1Hz porque posteriormente se deberá leer y asignar al mismo tiempo esta misma signal y eso no se puede hacer
37  --con las salidas
38  signal clk_salida : STD_LOGIC;
39
40  begin
41     --process() sirve para poder usar condicionales o bucles, puede tener su propio nombre y tiene su propio begin y end
42     --Este primer process sirve para crear la señal de reloj interno usando el contador y para reiniciar el conteo cuando
43     --presiono el botón de reset
44     relojInterno:process(CLK, reset)
45     begin
46        --La instrucción rising_edge hace que este condicional se ejecute cuando ocurra un flanco de subida en la
47        --señal de reloj CLK proveniente de la NEXYS 2
48        if rising_edge(CLK) then
49           --Cuando el push button reset esté presionado valdrá cero lógico porque hace corto circuito y reiniciará el
50           --conteo de la signal contador
51           --El signo = en VHDL sirve para ver si hay igualdad
52           if(reset='0') then--Como la entrada reset es de 1 bit, se le debe asignar el valor poniendo comillas
53              --El signo <= en VHDL sirve para asignar un valor
54              contador <= 0;--Como el contador es de tipo integer se le debe asignar el valor decimal sin poner comillas
55           elsif(contador=62499) then
56              contador <= 0;
```

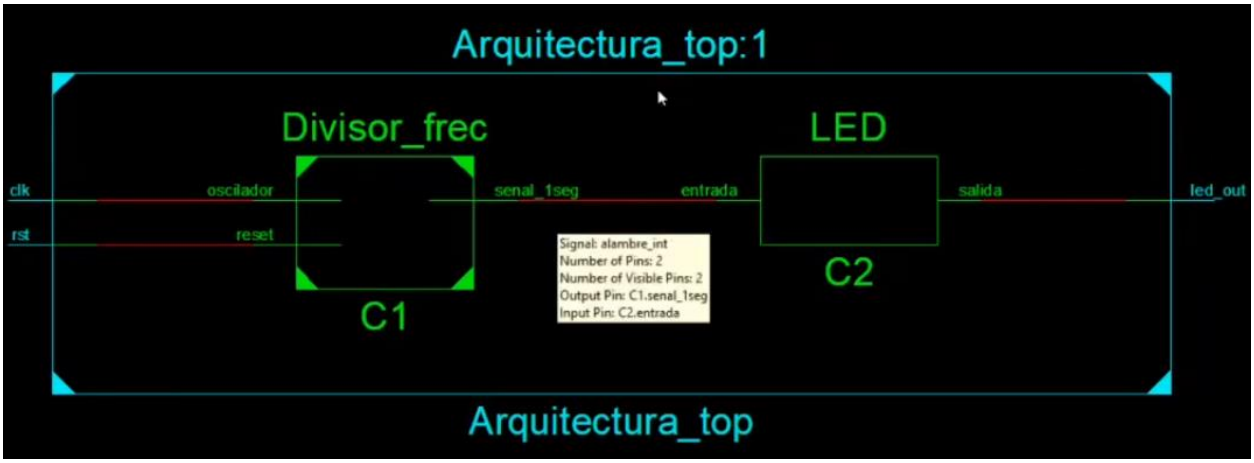
Si restamos 8.76 - 6.25 = 2.51, y este es el resultado esperado porque el tiempo del periodo T para la frecuencia de 400Hz es de 2.5ms.



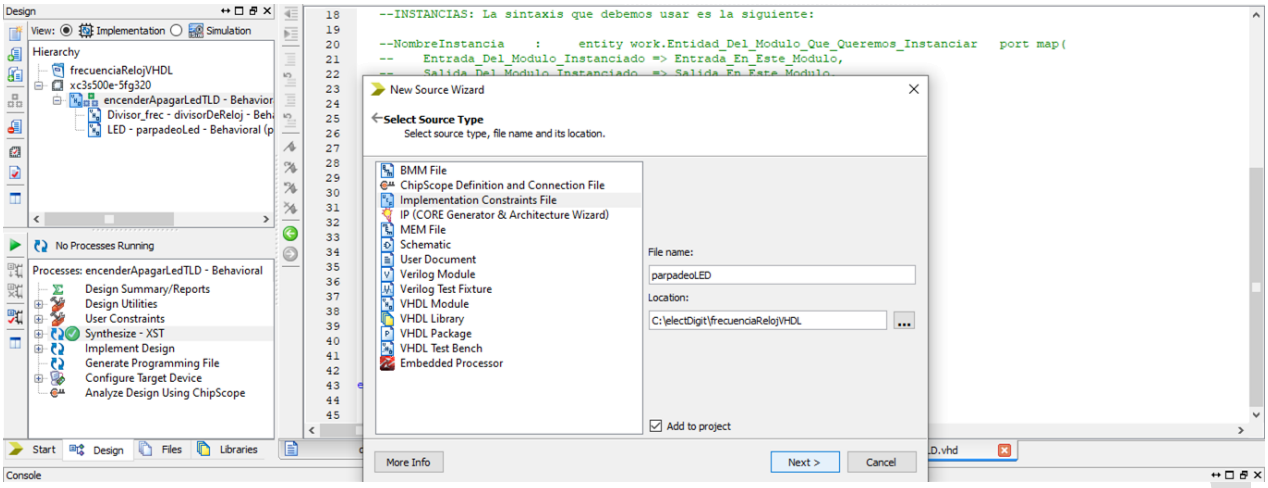
TLD: Hacer que Parpadee un Led

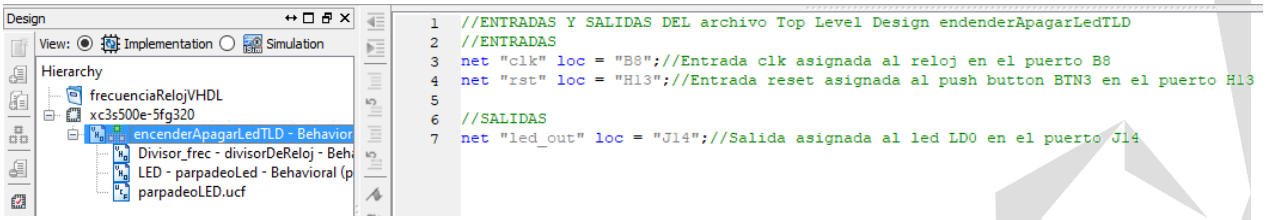
Este tipo de módulos donde modifico la frecuencia del reloj nunca pueden trabajar por sí solos, a fuerza debo unirlos con otros módulos para que puedan ser de utilidad, la acción más simple que un módulo divisor de reloj puede realizar es hacer que parpadee un led.

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



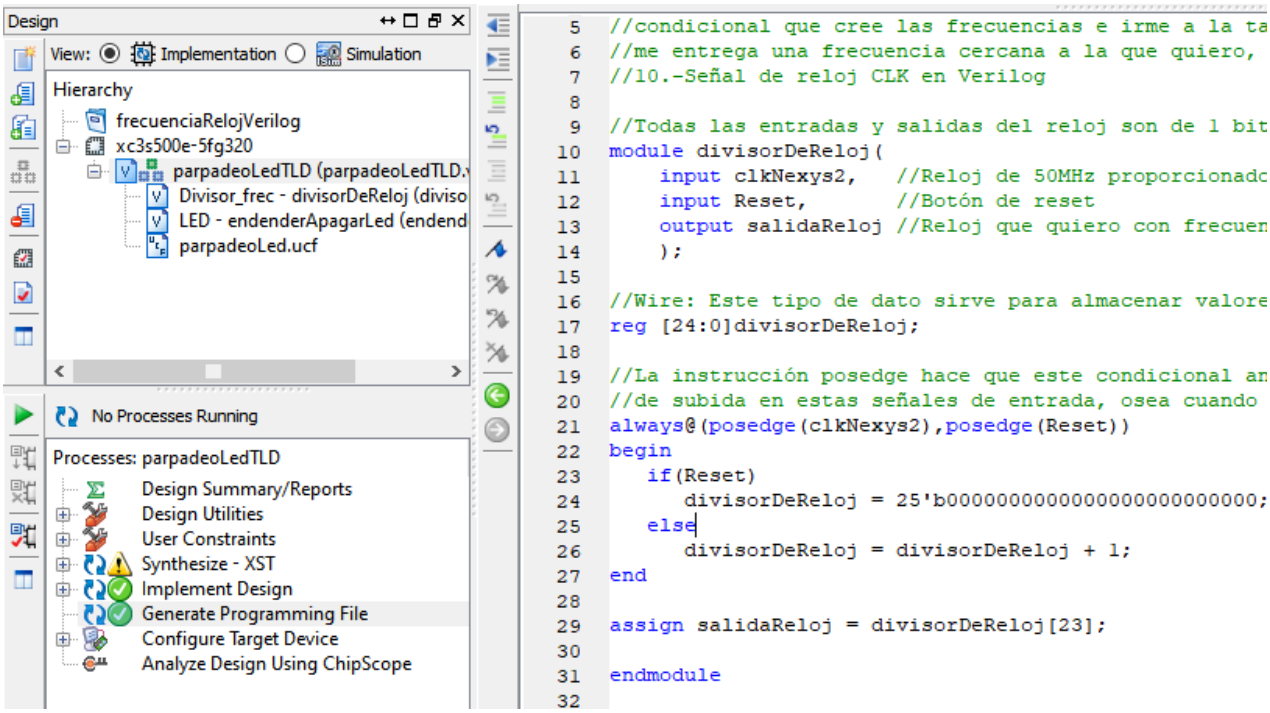
Al archivo TLD es al que le debo asignar un archivo UCF para indicar los puertos del reloj, del reset y del led que quiero que parpadee.





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 VHDL:

Divisor de Frecuencias Calculadas de Forma Precisa:

```
--DIVISOR DE RELOJ: Este proceso sirve para dictarle al reloj en qué 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, el cálculo
--que debo de hacer para obtener una señal CLK que tenga la frecuencia que yo quiero es el siguiente:
--Conteo'=50e6/frecuenciaDeseada
--Contador = (Conteo'/2)-1
--Este resultado solo sobrevivir dentro del programa y debe guardarse en una variable que vaya contando desde cero
--hasta el resultado que hayamos obtenido de Contador para crear una señal interna que después la puedo usar para
--obtener la verdadera señal de reloj que quiero de 1Hz

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Libreras para poder usar el lenguaje VHDL

--Entidad: Aquí se declaran las entradas y salidas, lo que declarar es la entrada del botón de reset, la entrada del
--reloj de 50MHz proporcionado por la NEXYS 2 y la salida de reloj con la frecuencia que quiero obtener.
entity divisorDeReloj is
    Port ( relojEntrada50MHz : in  STD_LOGIC;--Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8
           reset : in  STD_LOGIC;--Botón de reset
                                 relojSalida : out  STD_LOGIC);--Reloj que quiero con frecuencia de 1Hz
end divisorDeReloj;

--Arquitectura: Aquí voy a realizar el divisor de reloj con mis entradas y salidas, esto lo hará creando 3 signals y 2
--process, el 1ero me crear una señal interna y el segundo usar esa señal interna para darme la señal de reloj que
--quiero obtener.
architecture Behavioral of divisorDeReloj 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 contador : integer;--El contador sirve para crear la señal de reloj interno
signal i_clk : STD_LOGIC;
--la señal de reloj interno solo mantiene su nivel alto por un nanosegundo, pero sirve para después crear la señal de
--reloj con la frecuencia que yo quiero
signal clk_out : STD_LOGIC;
```

```
--Crear la señal de reloj con la frecuencia que quiero usando la señal de reloj interno, posteriormente se almacenará
--en la signal clk_out para que al final del código le asigne su valor a la salida llamada relojSalida, se hace uso de
--una signal y no se almacena directamente en la salida relojSalida porque posteriormente se deber leer y
--sobrescribir al mismo tiempo el valor de esta misma signal y eso no se puede hacer con las salidas

begin
--process() sirve para poder usar condicionales o bucles, puede tener su propio nombre y tiene su propio begin y end
          --Este primer process sirve para crear la señal de reloj interno usando el contador y para reiniciar el conteo cuando
          --presiono el botón de reset
          relojInterno : process(relojEntrada50MHz, reset)
                    begin
                    --La instrucción rising_edge hace que este condicional se ejecute solo cuando ocurra un flanco de subida en la
                    --señal de reloj CLK proveniente de la NEXYS 2 de 50MHz
                              if(rising_edge(relojEntrada50MHz)) then
                              --Cuando el push button reset este presionado valdrá uno lógico porque hace corto circuito y
                              --reiniciará el conteo de la signal contador.
                              --El signo = en VHDL sirve para ver si hay igualdad
                                        --IF ANIDADO
                                        --Como la entrada reset es de 1 bit, se le debe asignar el valor poniendo comillas.
                                        if(reset = '1') then
                                                  --El signo <= en VHDL sirve para asignar un valor
                                                  contador <= 0;
                                                  --Como el contador es de tipo integer se le debe asignar el valor decimal sin
                                                  --poner comillas.

                                        --Conteo para obtener la señal CLK de 1Hz, osea que cada 1s se repita su ciclo
                                        --Conteo'=50e6/1
                                        --Contador = (Conteo'/2)-1 = (25e6)-1 = 24999999
                                        --Este valor lo introduzco en el elsif para que el contador se reinicie cuando llegue a su
                                        --valor máximo.
                                        elsif(contador = 62499) then
                                                  contador <= 0;
                                                  --Por esta línea de código es que la señal interna tiene nivel alto solo en un
                                                  --pequeño nanosegundo, ya que vale 1 lógico solamente cuando el contador está en
                                                  --el valor máximo que calculamos.
                                                  i_clk <= '1';
                                        else
                                                  --Las signal se pueden leer y asignar en la misma línea sin problema.
                                                  contador <= contador +1;
                                                  --Por esta línea de código es que la señal interna está en nivel bajo en la
                                                  --mayora de su ciclo, ya que vale 0 lógico durante todo el conteo del contador
                                                  --excepto cuando está en su valor máximo.
                                                  i_clk <= '0';
                                        end if;
                              end if;
          end process relojInterno;

          --El segundo process sirve para crear la señal CLK que quiero usando el reloj interno que creamos anteriormente, además
          --también puede ser detenida por el mismo botón de reset.
          relojDeSalida : process(i_clk, reset)
                    begin
                              if (reset = '1') then--Cuando el botón de reset es presionado el reloj de salida vale 0 lógico
                                        clk_out <= '0';
                              --Cuando hay un flanco de subida en el reloj interno, al reloj de salida se le asigna el valor
                              --negado del valor que tena anteriormente para crear la señal de reloj que quiero.
                              elsif(rising_edge(i_clk)) then
                                        --Por esta línea de código es que tuvimos que crear una signal en vez de usar directo la
                                        --salida reloj1Hz, ya que se está leyendo y sobrescribiendo el valor en una misma línea de
                                        --código y las salidas no se pueden leer, solo se les puede asignar un valor.
                                        clk_out <= not clk_out;
                              end if;
          end process relojDeSalida;

--Fuera del segundo process asignaremos el valor de la signal clk_salida a la salida reloj1Hz
relojSalida <= clk_out;

end Behavioral;
```

Encender y Apagar un Led:

```
--MODULO QUE USA EL DIVISOR DE RELOJ PARA PRENDER Y APAGAR UN LED
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--En la entidad se declaran las entradas y salidas
entity encenderApagarLed is
    Port ( entrada : in  STD_LOGIC;
           salida : out  STD_LOGIC);
end encenderApagarLed;

--En la arquitectura se indican las acciones a ejecutar con las entradas y salidas
architecture Behavioral of encenderApagarLed is
begin
process(entrada)
          begin
                    if(entrada='1')then
                              salida <= '1';      --Encender led.
                    else
                              salida <= '0'; --Apagar led.
                    end if;
          end process;
end Behavioral;
```

Módulo TLD:

```
--TLD: Parpadeo Led
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Declaramos como entradas y salidas solo los pines que salgan del diagrama de bloques global en mi TLD, esto se ve
--en el documento 8.2.-Senal de reloj CLK Solamente en VHDL.
entity parpadeoLedTLD is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           led_out : out  STD_LOGIC);
end parpadeoLedTLD;

--Arquitectura: Aquí voy a declarar todas las instancias de mis demás módulos para poderlos usar y sus signal.
architecture Behavioral of parpadeoLedTLD is
--Signal: Es un tipo de dato que sirve para almacenar un valor que no salga del bloque global TLD.
signal alambre_int : STD_LOGIC;
--Las signal se declaran antes del begin de la arquitectura.
begin
--INSTANCIAS:
          --Debo darle un nombre a cada instancia que cree, indicar el nombre de la entidad del modulo 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
          Divisor_frec: entity work.divisorDeReloj port map(
                              relojEntrada50MHz => clk,
                              reset => rst,
                              relojSalida => alambre_int
                    );
          --INSTANCIA DEL MODULO encenderApagarLed

          LED: entity work.encenderApagarLed port map(
                              entrada => alambre_int,
                              salida => led_out
                    );
end Behavioral;
```

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 = "J14";//Salida asignada al led LD0 en el puerto J14.
```

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 leídas, 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;
```