

## **2.3. VHDL: Diseño de sistemas secuenciales**

## OBJETIVOS DEL TEMA:

- ❑ Repaso de circuitos secuenciales
- ❑ Descripción VHDL de circuitos secuenciales básicos:
  - a. Flip-Flop tipo D
  - b. Registros
  - c. Contadores
- ❑ Testbenches para circuitos secuenciales

### **2.3.1. Repaso de Lógica secuencial**

## 2.3.1. Repaso de Lógica secuencial

Hasta ahora solo hemos diseñado sistemas combinacionales.

“El valor de la salida en un instante  $t$ , solo depende del valor de las entradas en ese instante  $t$ ”

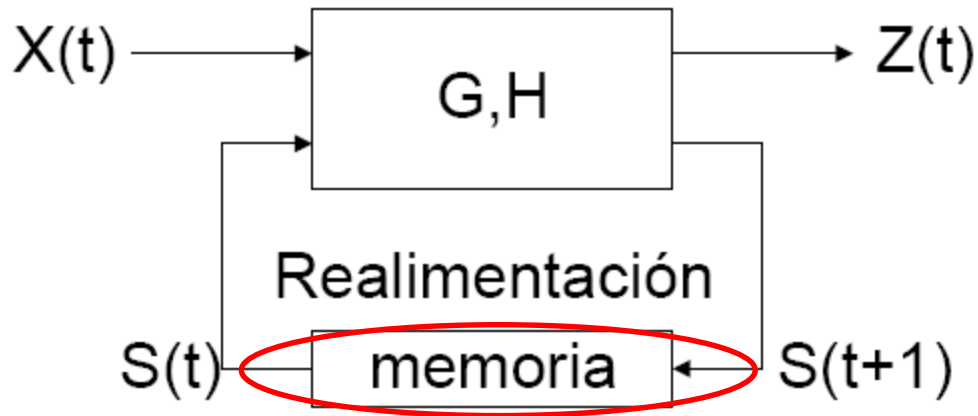
### ❑ Sistema secuencial

“La salida depende de los valores de las entradas en ese instante  $t$  y también de los valores que tuvieron en los instantes anteriores  $t-1$ .”

Son circuitos que “recuerdan” o tienen memoria de las situaciones por las que ha pasado el sistema.

A esas situaciones se les denominan estados.

## 2.3.1. Repaso de Lógica secuencial



$X(t)$ : entrada actual

$Z(t)$ : salida actual

$S(t)$ : estado actual

$S(t+1)$ : estado próximo

❑ Forma de operar de un sistema secuencial

Dado un estado  $S(t)$  y una entrada  $X(t)$  el sistema produce una salida  $Z(t)$  y el estado siguiente  $S(t+1)$ .

### **2.3.2. Tipos de Sistemas secuenciales**

## 2.3.2. Tipos de Sistemas secuenciales

### ❑ Asíncronos

- Pueden cambiar de estado en cualquier instante de tiempo en función de cualquier cambio en las señales de entrada.
- Son más frecuentes en la vida real.
- No suelen implementarse en FPGAs.

### ❑ Síncronos

- Sólo pueden cambiar de estado en determinados instantes de tiempo, es decir, están “sincronizados” con una señal que marca dichos instantes y que se conoce como **señal de reloj (Clk)**.
- El sistema sólo “hace caso de las entradas” en los instantes de sincronismo.
- Son más fáciles de diseñar y más controlables.

### **2.3.3. Descripción en VHDL de sistemas secuenciales**



## 2.3.3. Sistemas secuenciales en VHDL

### a) Combinacionales:

- **Asignación concurrentes de señales (Recomendado)**
  - ✓ Simple o incondicional (  $\leq$  )
  - ✓ Condicional (when-else)
  - ✓ Selectiva (with –select- when)
- **Procesos combinacionales (PROCESS)**
  - ✓ Lista de sensibilidad (todas las entradas al proceso)
    - Case
    - If-then
  - ✓ Definir “Else” para evitar latches

### b) Secuenciales (Siempre dentro de un proceso)

- **Procesos secuenciales con una lista de sensibilidad**
  - ✓ Sentencias síncronas

## 2.3.3. Sistemas secuenciales en VHDL

### Plantilla VHDL

```
Ejemplo: process (CLK, RESET)
begin
    if RESET = '1' then
        -- Asignaciones Asíncronas (Solo RESET)
        -- .....
        -- .....
    elsif rising_edge (CLK) then
        -- Asignaciones Síncronas
        Q <= D;
        -- .....
    end if;
end process;
```

- Todas las entradas asíncronas deben sincronizarse
- Fuera de la zona de asignaciones síncronas NO debe escribirse nada

## 2.3.3. Sistemas secuenciales en VHDL

### Consideraciones de codificación

- Tipos de sincronismo (flancos de reloj)
  - Flanco positivo → *rising\_edge* (*clock\_name*)
  - Flanco negativo → *falling\_edge* (*clock\_name*)
- La sentencia de reloj no debe incluir más condiciones

```
if clk'event and clk='1' then
    if J='1' then
        . . . . .
```

```
if clk'event and clk='1' and J='1' then
    . . . .
```

**No!**

## 2.3.3. Sistemas secuenciales en VHDL

### Consideraciones de codificación

- La sentencia de reloj debe de terminar con un *end if* y no debe de haber un *else* o *elsif* a continuación.

```
Biest_proc: Process (Reset, Clk)
begin
  if Reset = '0' then
    Q <= '0';
  elsif Clk'event and Clk='1' then
    Q <= Dato;
  else
    Q <= P;
  end if;
end process;
```

**No!**

- En la lista de sensibilidad del proceso SOLO debe aparecer la señal de reloj. (Excepto RESET asíncrono)

## **2.3.4. Repaso de Biestables**

## 2.3.4. Repaso: Biestables

- ❑ Dispositivo capaz de almacenar un bit (H ó L).
- ❑ Almacena un valor estable (H ó L) hasta que se produzca un cambio en las entradas.
- ❑ **Tabla de excitación**: refleja los valores de entrada que hacen evolucionar a la salida del estado actual al siguiente

## 2.3.4. Repaso: Biestables

### □ Clasificación del biestable por su sincronismo

- Asíncrono (**latch**)
- Síncrono:
  - Por nivel
  - Por flanco (**flip-flop**)

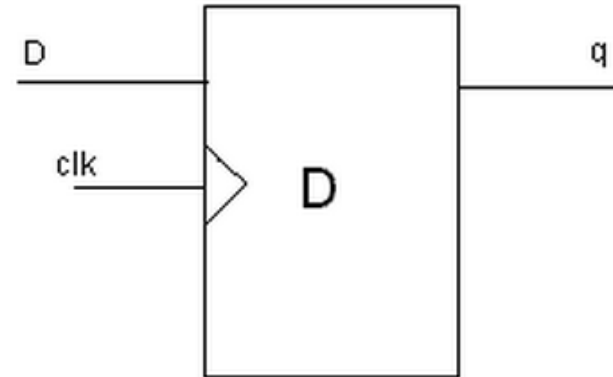
### □ Clasificación según las entradas de datos (FF síncronos):

- S-R: entradas de puesta a 1 (S, set) y puesta a 0 (R, reset)
- J-K: entradas de puesta a 1 (J, set) y puesta a 0 (K, reset)
- **D: entrada de datos (D)**
- T: entrada de inversión o basculamiento (toggle)

## 2.3.4. Repaso: Flip-Flop D

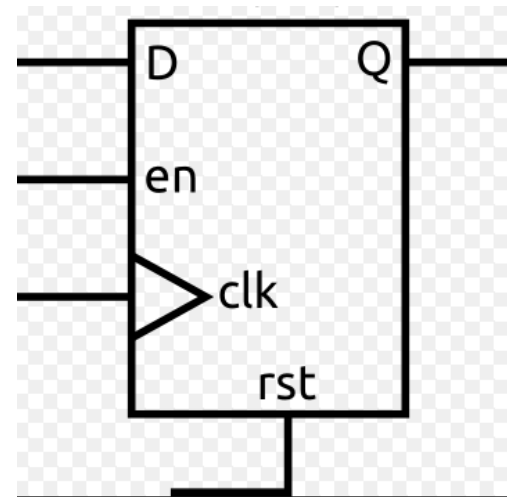
Atendiendo a las entradas:

### 1. Flip-Flop D básico



### 2. Flip-Flop D con señales de control (RESET, SET, ENABLE):

- RESET asíncrono
- ENABLE síncrono

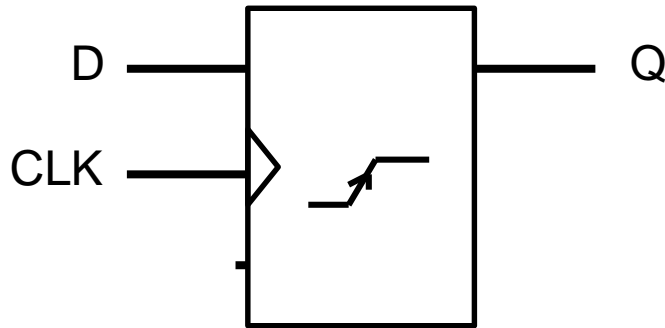




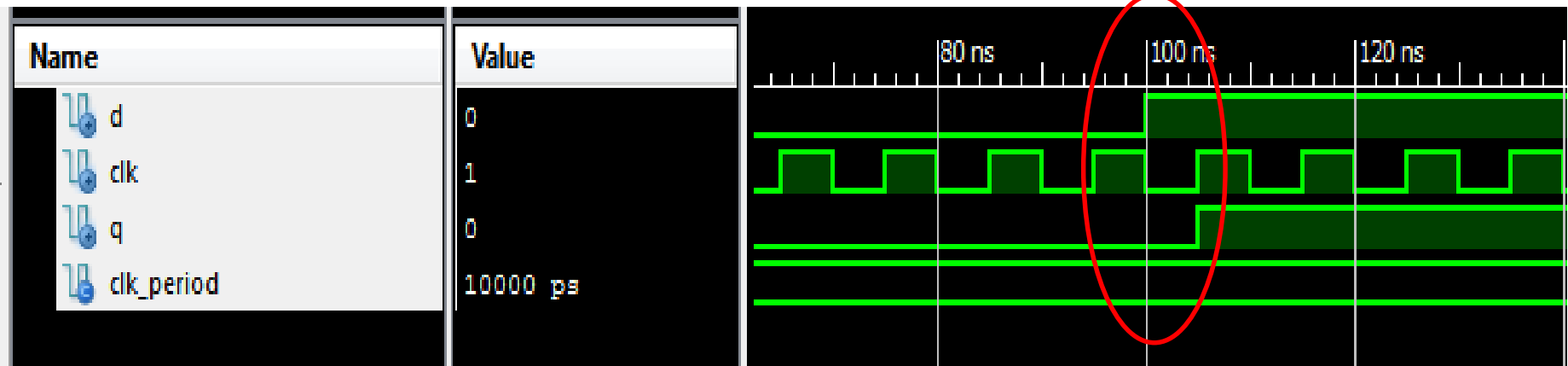
### **2.3.5. Descripción VHDL de FF tipo D**

## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D Básico



CLK	D	Q(t+1)
↑	0	0
↑	1	1
Resto	X	Q(t)



## 2.3.5. VHDL: Flip-Flop D

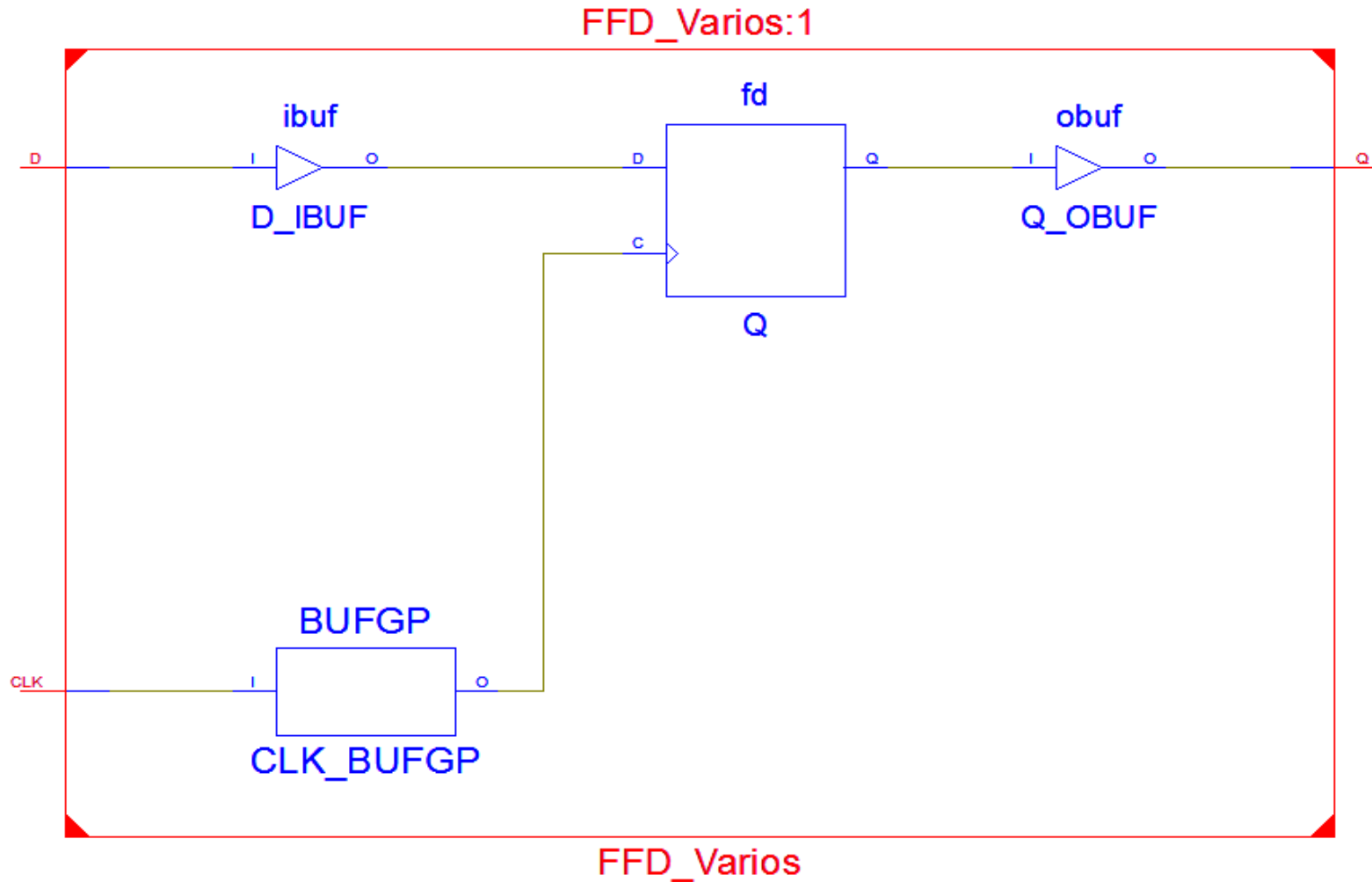
### Flip-Flop D Básico

```
Entity FFD_Basic is
    Port ( D : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Q : out STD_LOGIC ) ;
end FFD_Basic ;

architecture Behavioral of FFD_Basic is
begin
    process (CLK)
    begin
        if rising_edge (CLK) then
            Q<=D; // Sentencia síncrona
        end if;
    end process;
end Behavioral;
```

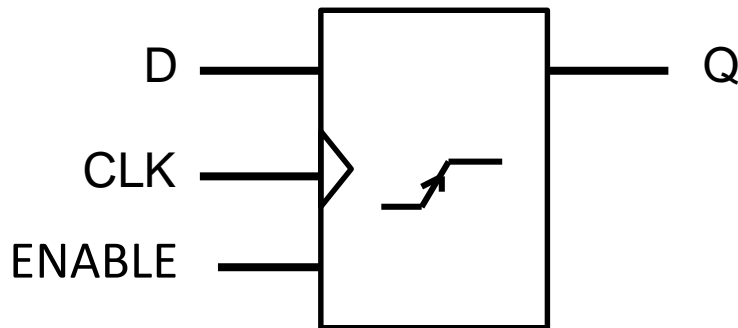
## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D Básico

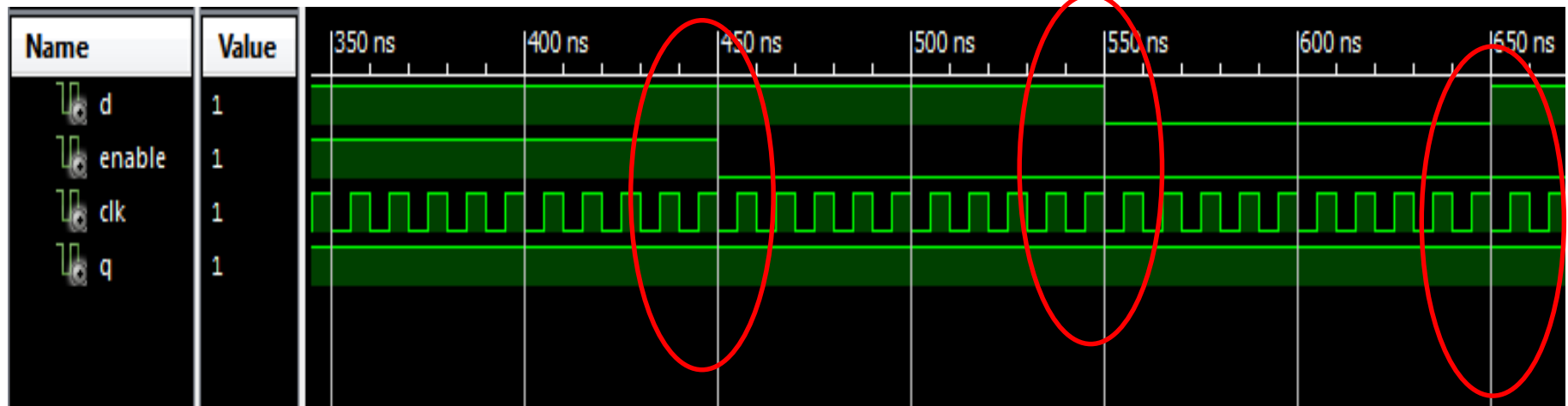


## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de habilitación (Enable síncrono)



CLK	ENABLE	D	Q(t+1)
↑	1	0	0
↑	1	1	1
↑	0	X	Q(t)



## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de habilitación (Enable síncrono)

```
Entity FFD_E is
    Port ( D : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          ENABLE : in  STD_LOGIC;
          Q : out  STD_LOGIC ) ;
end FFD_E ;
```

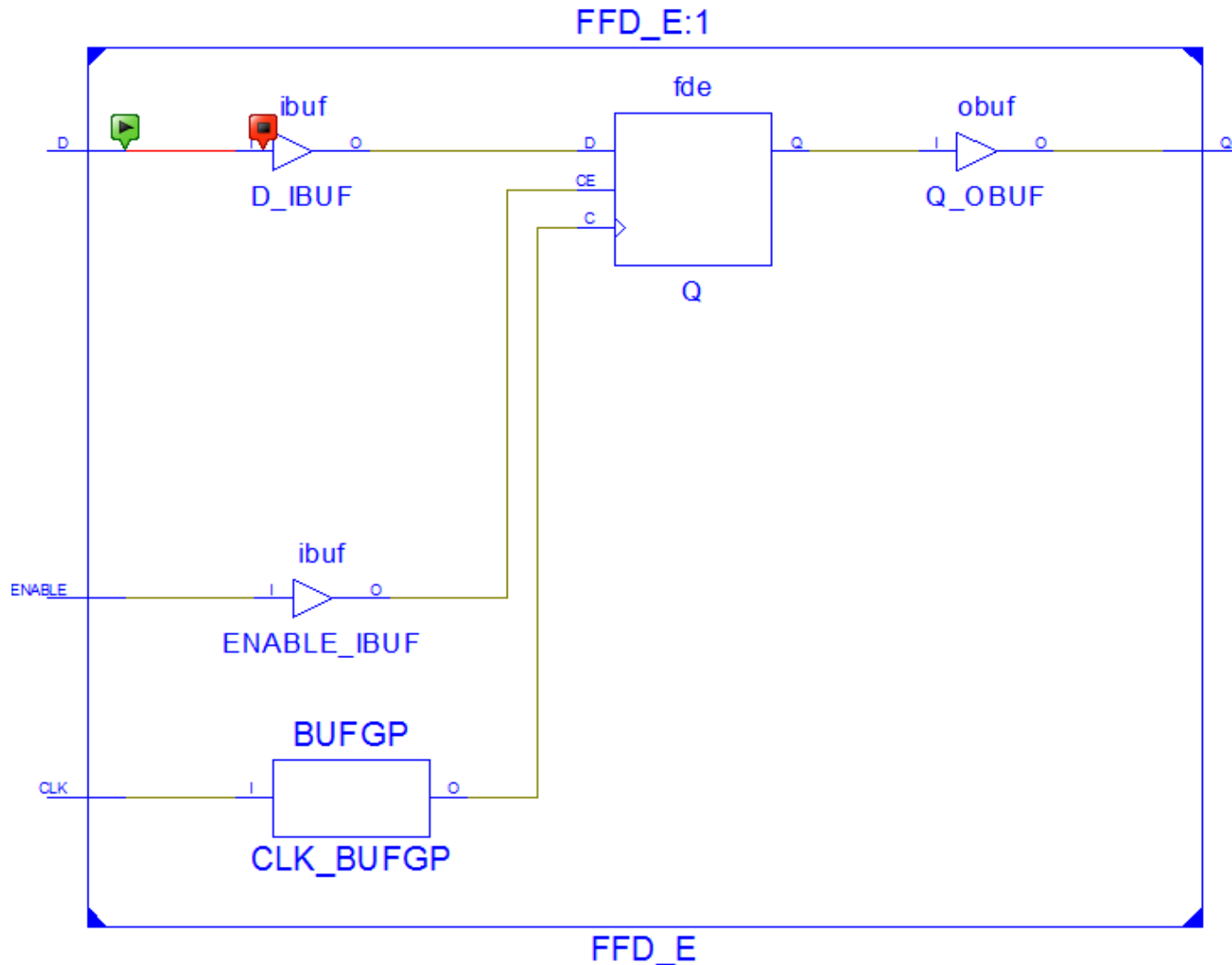
```
architecture Behavioral of FFD_E is
begin
    process (CLK)
    begin
        if rising_edge (CLK) then
            if (ENABLE='1') then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

Al mantener en la lista de sensibilidad solo la señal CLK, se consigue que la entrada ENABLE solo se analice cuando llegue el flanco activo de reloj (sincronizado)

Sentencias síncronas

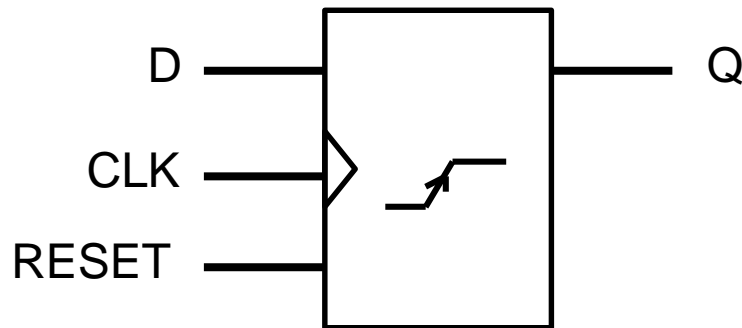
## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de habilitación (Enable síncrono)

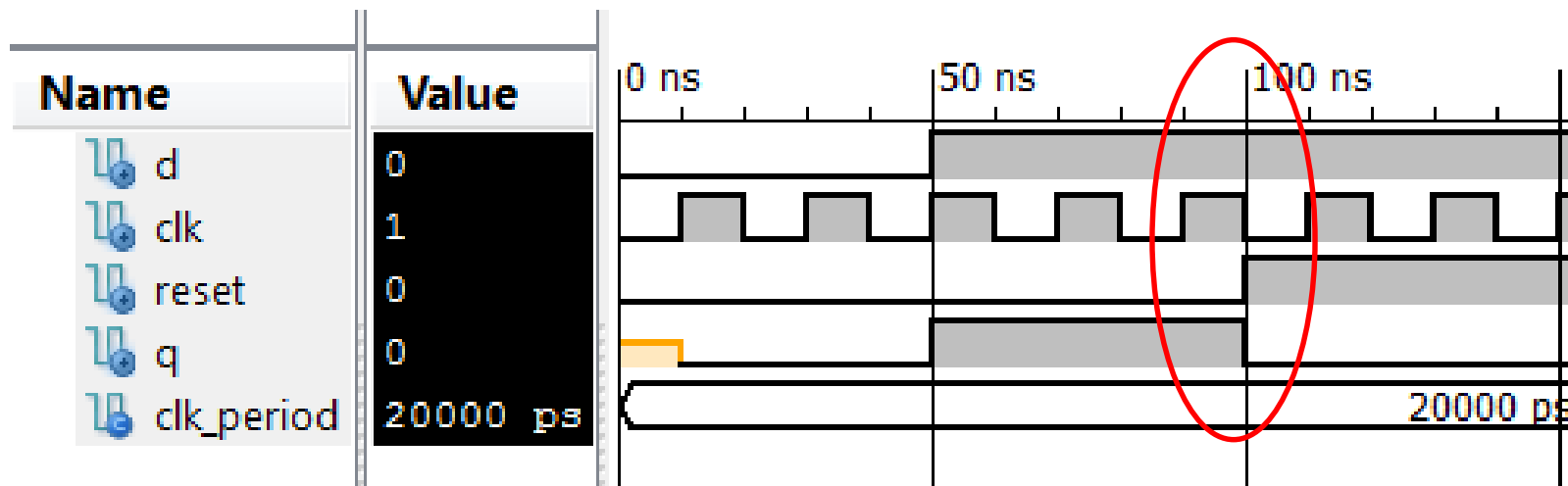


## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de puesta a cero (Reset asíncrono)



CLK	RESET	D	Q(t+1)
↑	0	0	0
↑	0	1	1
Resto	1	X	0





## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de puesta a cero (Reset asíncrono)

```
entity FFD_R is
    Port ( D : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          RESET: in  STD_LOGIC;
          Q : out  STD_LOGIC);
end FFD_R ;

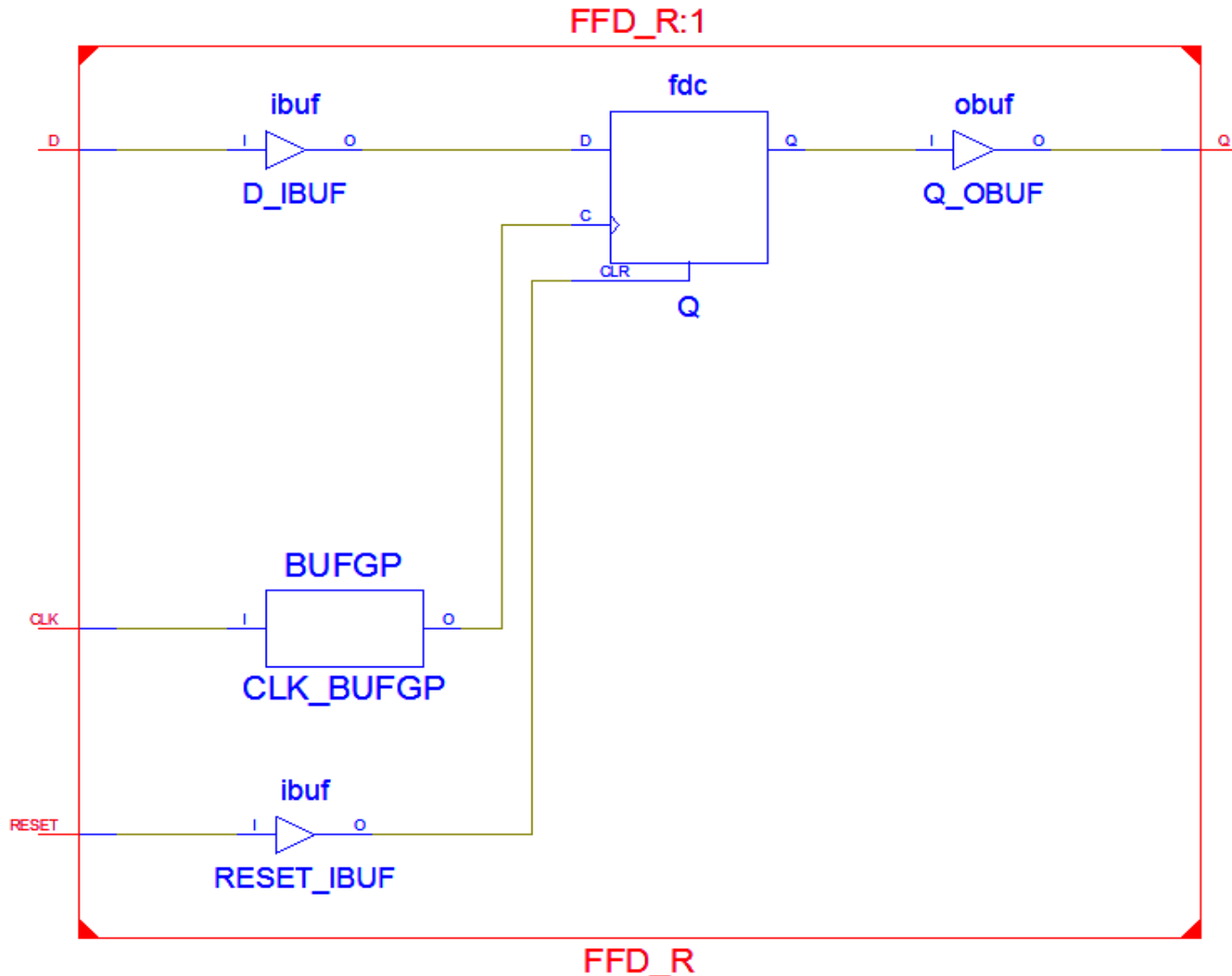
architecture Behavioral of FFD_R is
begin
    process (CLK, RESET)
    begin
        if RESET='1' then
            Q <= '0';
        elsif rising_edge (CLK) then
            Q <= D;
        end if;
    end process;
end Behavioral;
```

Se incluye la entrada RESET, por tanto un cambio en ella o en CLK inician el PROCESS

La construcción con IF establece la prioridad de RESET sobre CLK.

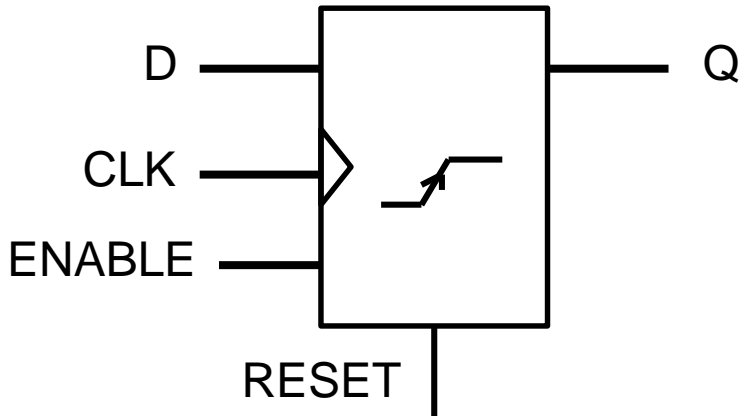
## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D con entrada de puesta a cero (Reset asíncrono)

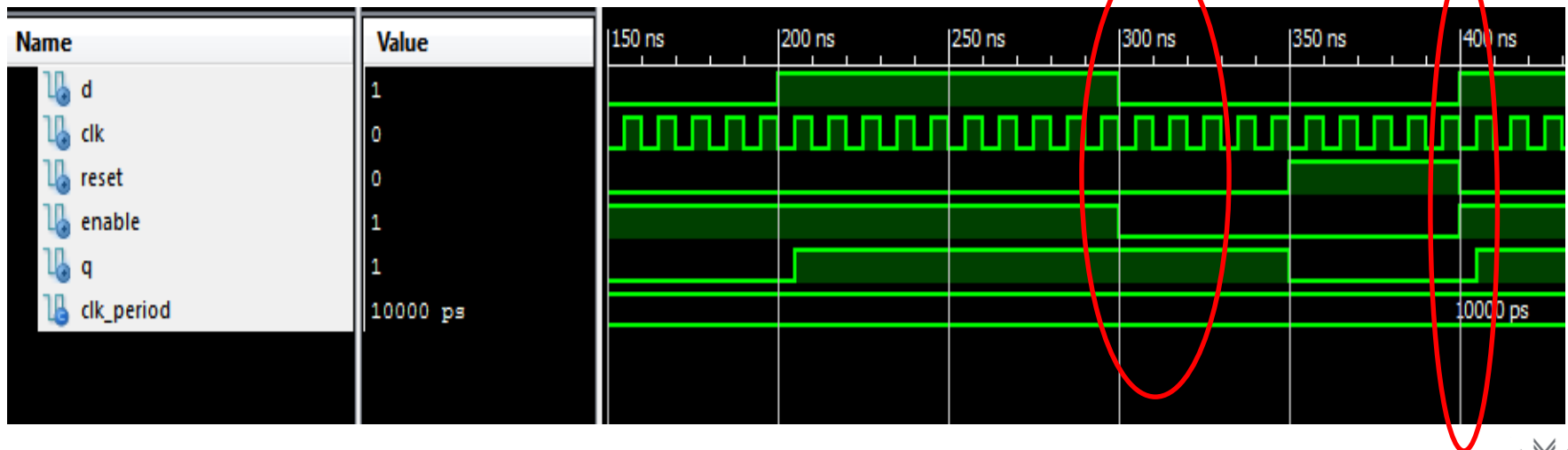


## 2.3.5. VHDL: Flip-Flop D

### Flip-Flop D completo (Reset asíncrono y Enable Síncrono)



CLK	RESET	ENABLE	D	Q(t+1)
↑	0	0	X	Q(t)
↑	0	1	0	0
↑	0	1	1	1
RESTO	1	X	X	0



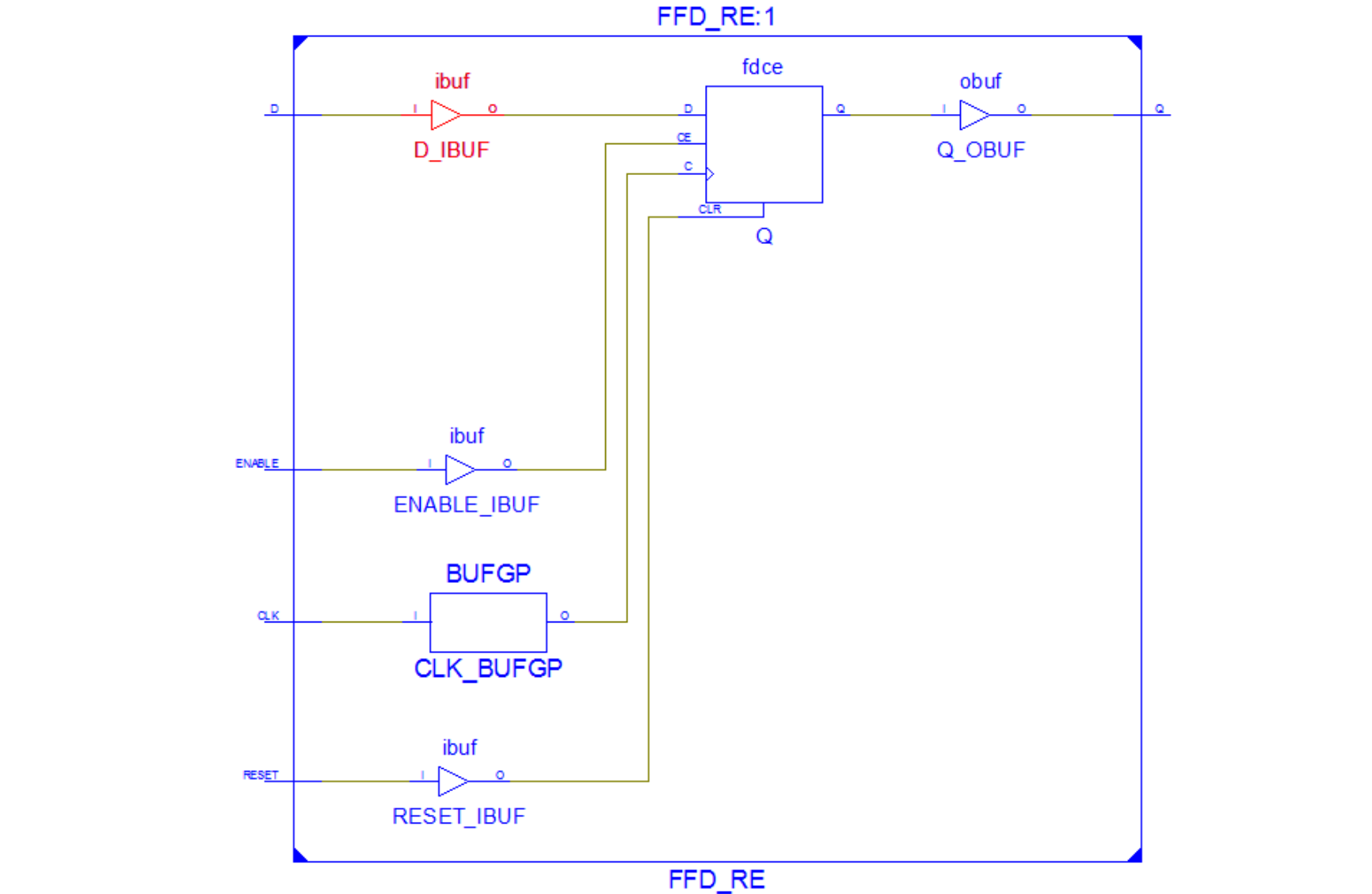
## 2.3.5. VHDL: Flip-Flop D

```
entity FFD is
  Port ( D : in  STD_LOGIC;
        CLK : in  STD_LOGIC;
        ENABLE : in  STD_LOGIC;
        RESET: in  STD_LOGIC;
        Q : out  STD_LOGIC);
end FFD;
architecture Behavioral of FFD is
begin
  process (CLK, RESET)
  begin
    if RESET='1' then
      Q<='0';
    elsif rising_edge(CLK) then
      { if ENABLE='1' then
          Q<=D;
        end if;
    end if;
  end process;
end Behavioral;
```

Solo CLK y RESET aparecen en la lista de sensibilidad.

ENABLE se establece como síncrono y permite la actualización ó no de Q.

### 2.3.5. VHDL: Flip-Flop D



### **2.3.6. Sincronización de entradas. Ejemplo: Detector de flancos**

## 2.3.6. Sincronización de entradas

**Proyecto15.** Diseño de circuito secuencial que detecte una transición de “0” a “1” en la entrada (Push) . Cuando se detecte el flanco positivo la salida(Pulse) tomará el valor “1”.

Nombre del Proyecto: **Proyecto15**

Nombre del fichero y de la entidad VHDL: **RisingEdge**

- Entradas: **Reset, Push, Clk**

- Salidas: **Pulse**

	Push en t-1	Push en t	Salida Pulse
	0	0	0
Flanco positivo	0	1	1
	1	0	0
	1	1	0

## 2.3.6. Sincronización de entradas

### Detector de flancos positivos (Fallos al no sincronizar la entrada Push)

```
-- SEQUENTIAL CIRCUIT
```

```
-- Stores Previous Push (FFD)
```

```
StorePrevPush:process (CLK,Reset)
```

```
begin
```

```
  if Reset='1' then
```

```
    PreviousPush<='0';
```

```
  elsif rising_edge(CLK) then
```

```
    PreviousPush<=Push;
```

```
  end if;
```

```
end process;
```

Almacena la entrada **Push** para actuar después como PreviousPush. Es un bit que puedo guardar en un FFD.

```
-- COMBINATIONAL CIRCUIT
```

```
Pulse<='1' when PreviousPush='0' and Push='1'
```

```
  else '0';
```

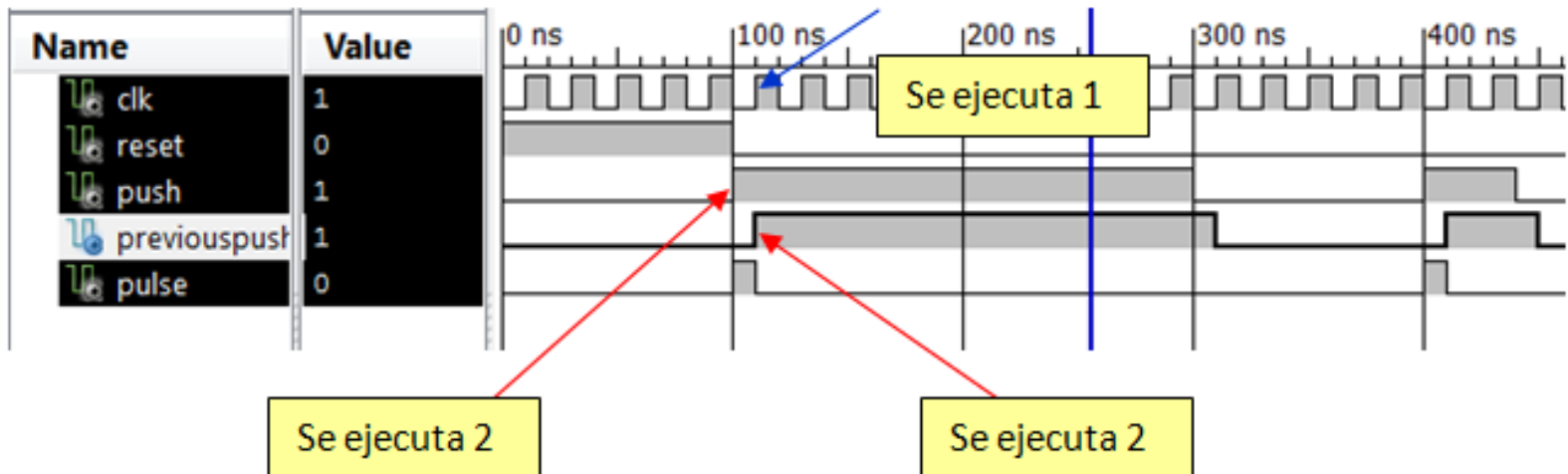
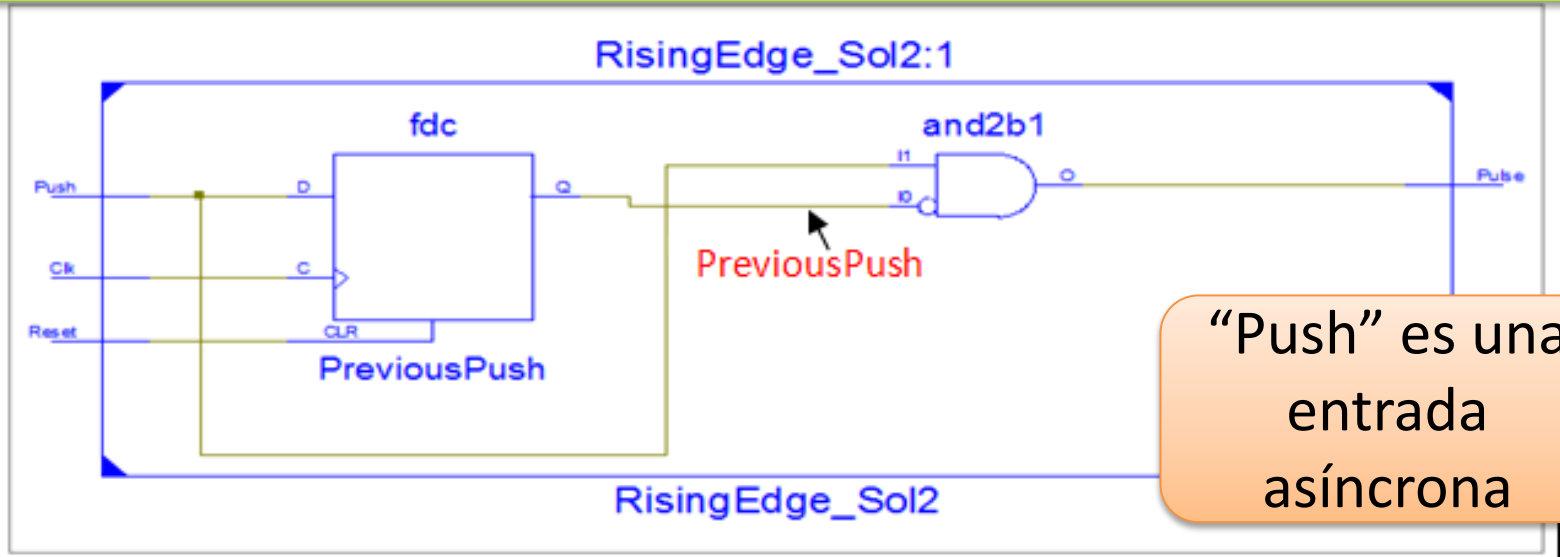
```
end Behavioral;
```

La sentencia de asignación a **Pulse** se ejecuta cuando haya un cambio en las entradas presentes en la asignación: **PreviousPush** y **Push**



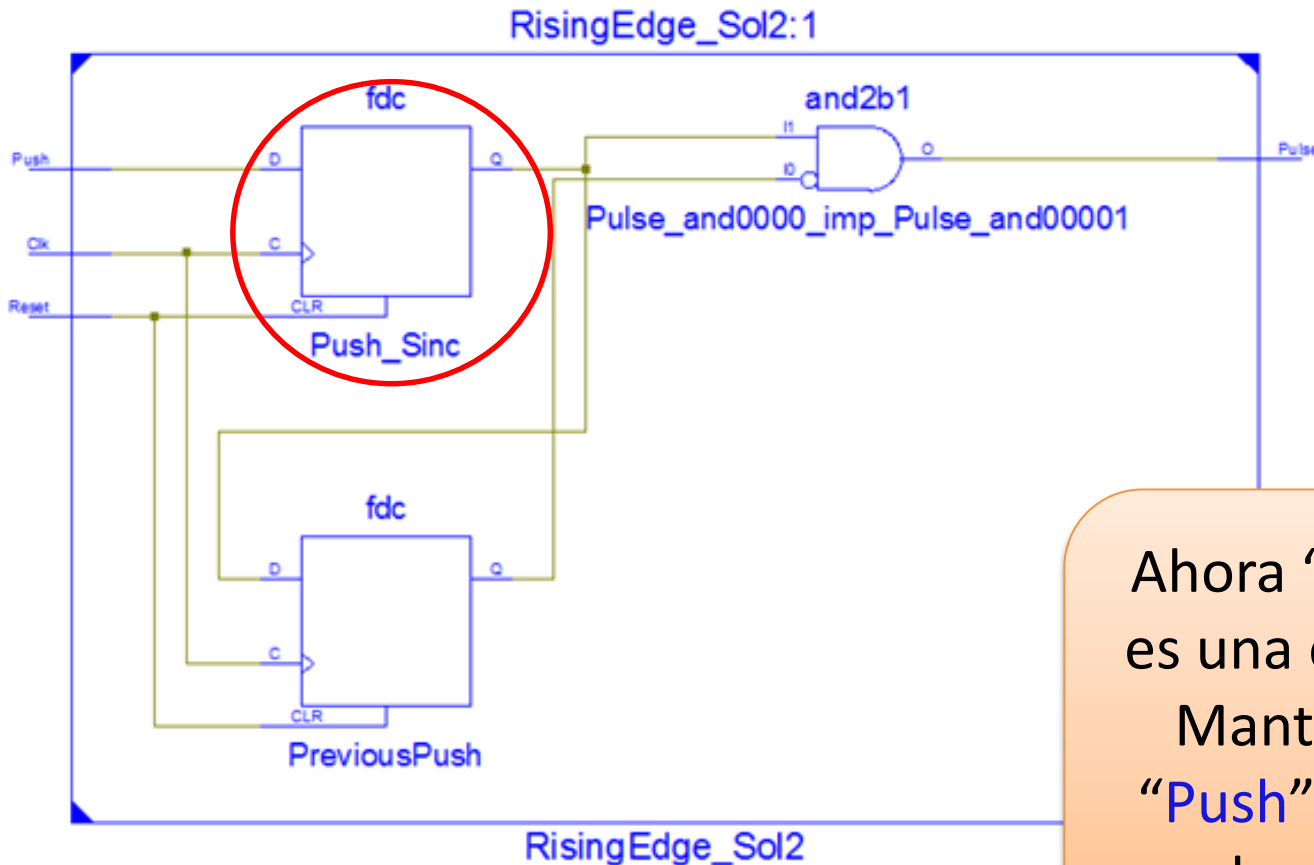
## 2.3.6. Sincronización de entradas

### Detector de flancos positivos (Fallos al no sincronizar la entrada Push)



### 2.3.6. Sincronización de entradas

### Detector de flancos positivos (Entrada registrada)



Ahora “RegisteredPush”  
es una entrada síncrona.  
Mantiene el valor de  
“Push” durante un ciclo  
de reloj al menos.

## 2.3.6. Sincronización de entradas

### Detector de flancos positivos (Entrada registrada)

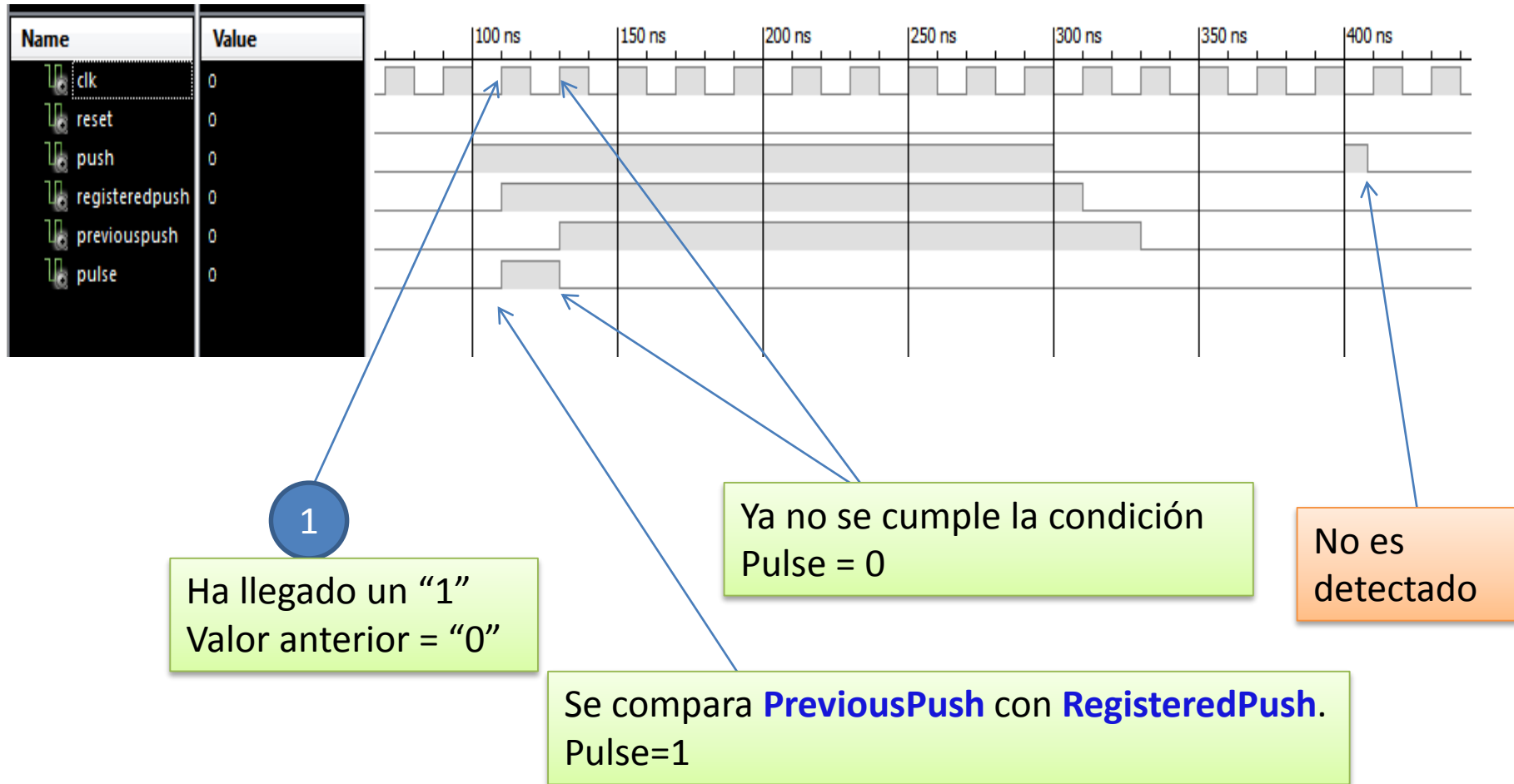
```
architecture Behavioral of RisingEdge is
    signal RegisteredPush: std_logic; -- To avoid async. input
    signal PreviousPush: std_logic; -- To store "Push"
Begin
    -----
    SEQUENTIAL CIRCUIT
    -----

    -- Synchronizes the input "Push"
    SincPush:process (CLK,Reset)
    begin
        if Reset='1' then
            RegisteredPush<='0';
        elsif rising_edge(CLK) then
            RegisteredPush<= Push;
        end if;
    end process;

    ...
```

1

## 2.3.6. Sincronización de entradas

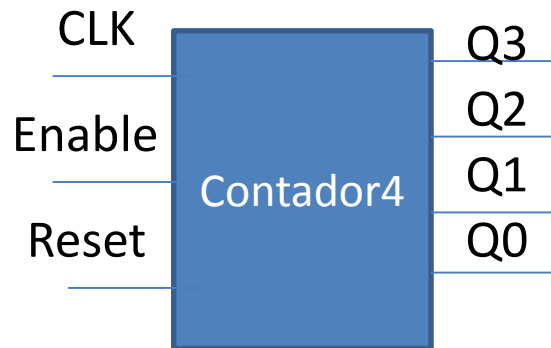


## **2.3.7. Contadores**

## 2.3.7. Contadores

### Contadores binarios

Circuito secuencial que genera una secuencia ordenada de salidas que se repite en el tiempo. Cuentan flancos de reloj.



### Tipos de contadores según el sincronismo

- **Síncronos**: todos los biestables que lo componen comparten la misma señal de reloj
- **Asíncronos**: no todos los biestables comparten la misma señal de reloj.

## 2.3.7. Contadores

### Contadores binarios síncronos

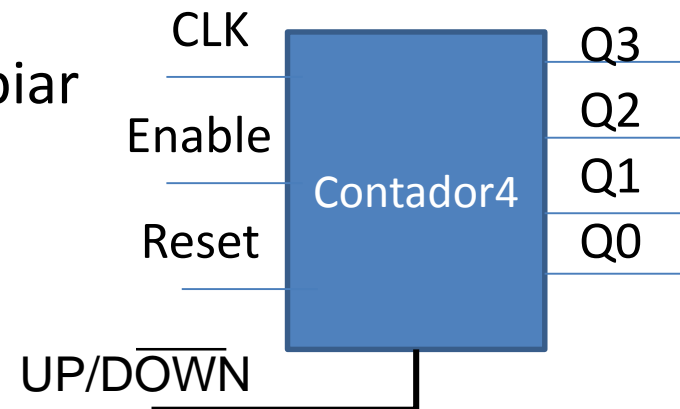
#### Tipos de Contadores según la cuenta (1/3)

- **Ascendente** → cuenta completa y creciente (De 0 a  $2^n - 1$ )
- **Descendente** → cuenta es completa y decreciente (De  $2^n - 1$  a 0)
- **Reversible**: la cuenta puede ser ascendente o descendente en función de una entrada de control.

**UP** → 0,1,2,3,0,1,2,3.....

**DOWN** → 3,2,1,0,3,2,1,0.....

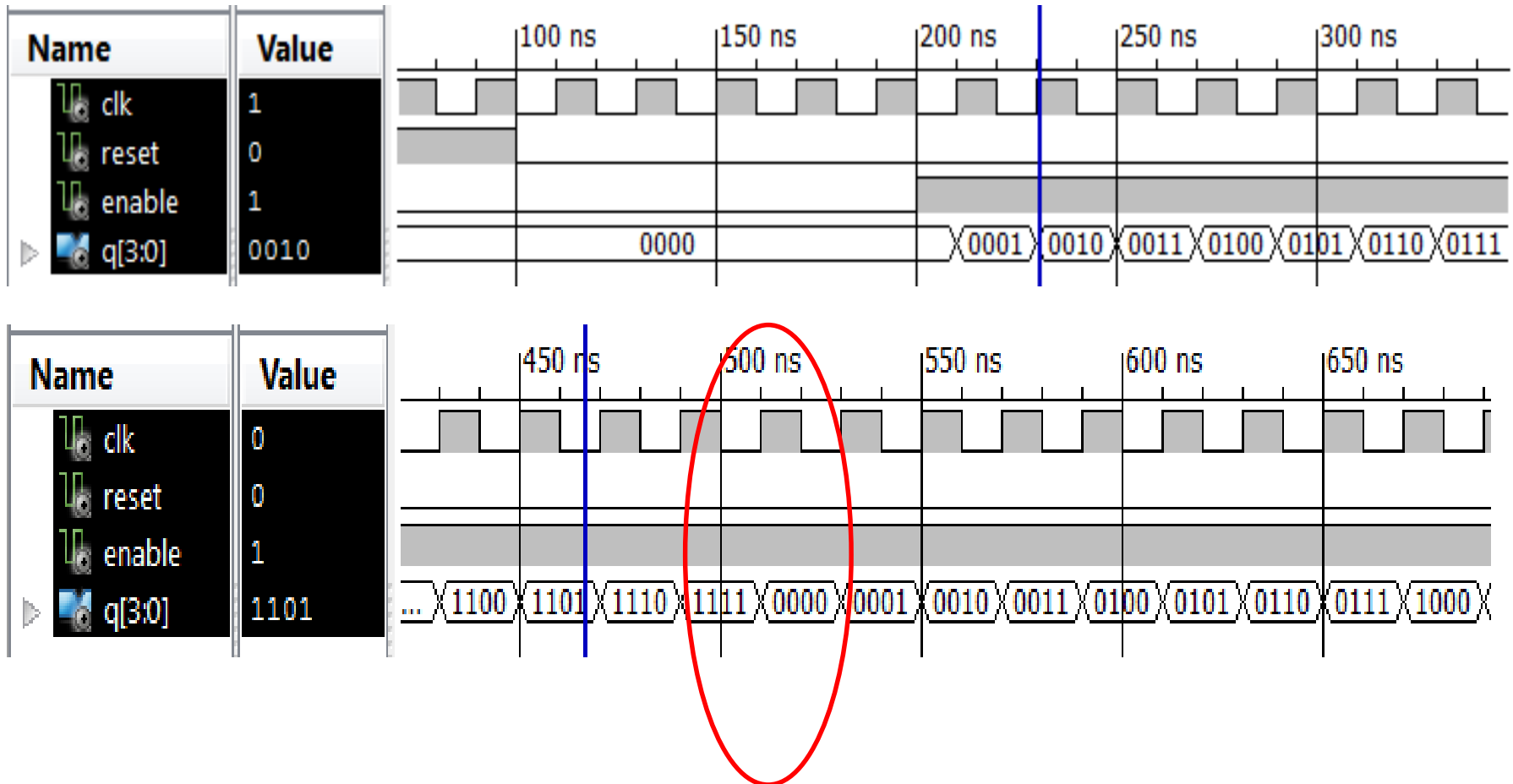
En cualquier momento es posible cambiar el sentido de la cuenta (UP/DOWN).



## 2.3.7. Contadores

### Contadores binarios síncronos

#### Síncrono ascendente

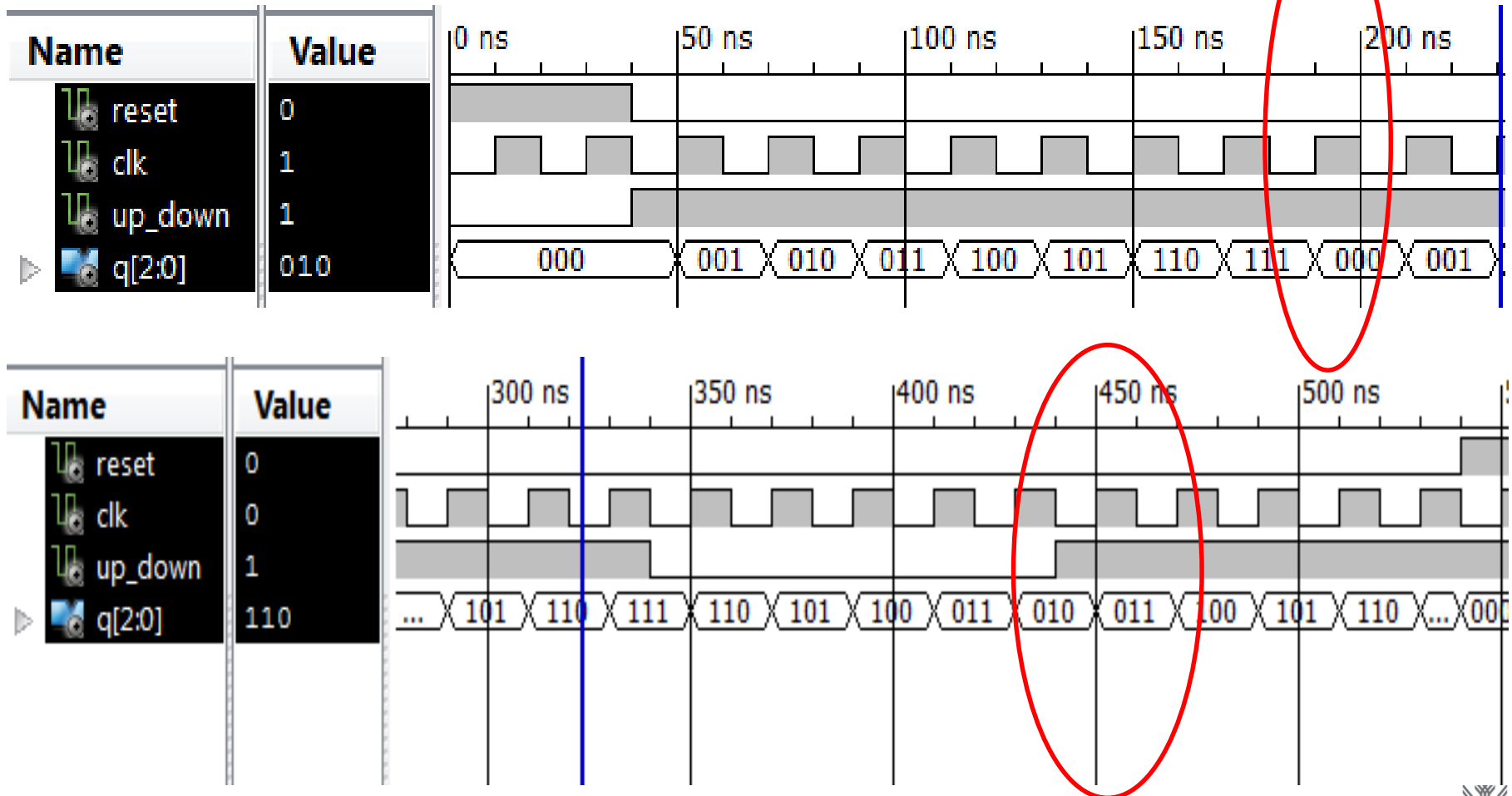




## 2.3.7. Contadores

### Contadores binarios síncronos

#### Contador síncrono reversible

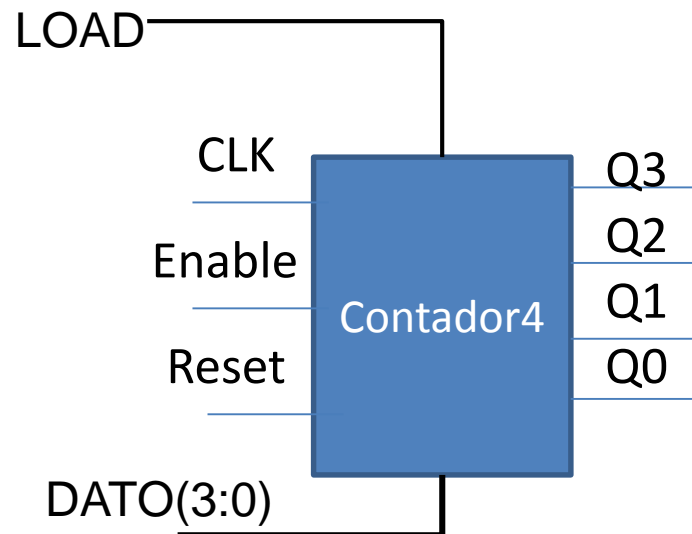


## 2.3.7. Contadores

### Contadores binarios síncronos

#### Tipos de Contadores según la cuenta

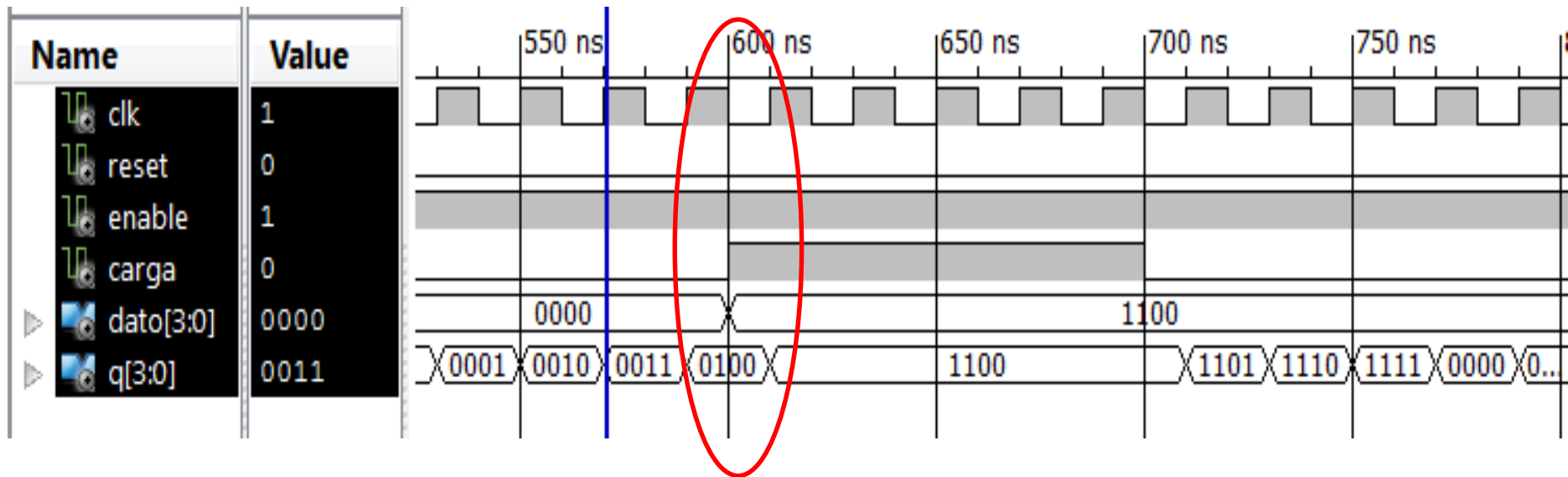
➤ **Con Precarga:** habilita la carga de un dato de n bits en el contador para contar a partir de él (**carga síncrona**). Incluye una entrada LOAD para activar la carga y la entrada del DATO.



## 2.3.7. Contadores

### Contadores binarios síncronos

#### Contador con precarga



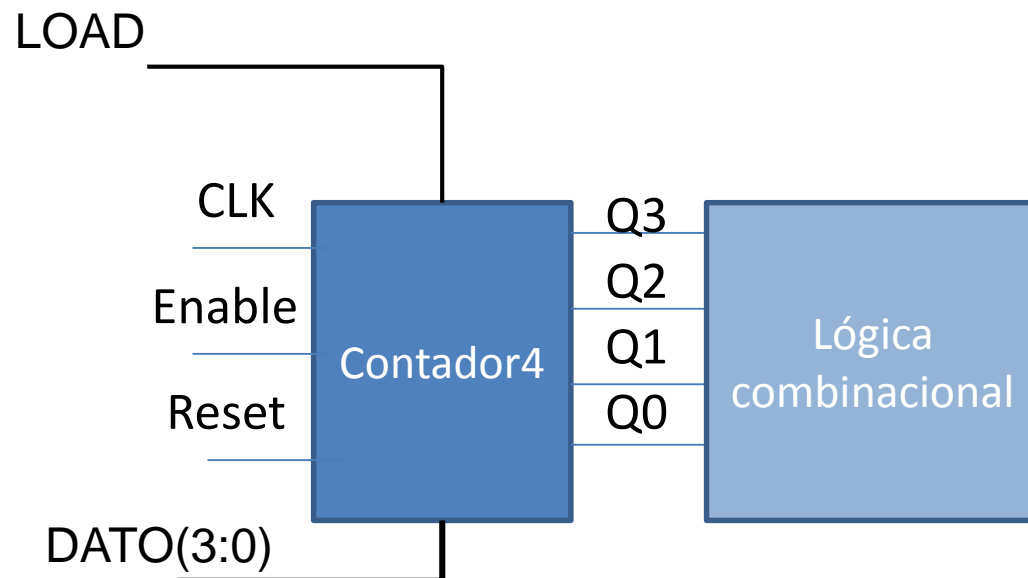
## 2.3.7. Contadores

### Contadores binarios síncronos

#### Tipos de Contadores según la cuenta(3/3)

➤ **De Módulo  $2^n - 1$ :** no realiza la secuencia completa de cuenta, esto es no va de 0 a  $2^n - 1$  ó viceversa. Requiere de lógica combinacional adicional.

P.e.: La secuencia puede ser, en el caso de  $n=4$ , solo de 0 a 9, ó de 10 a 15

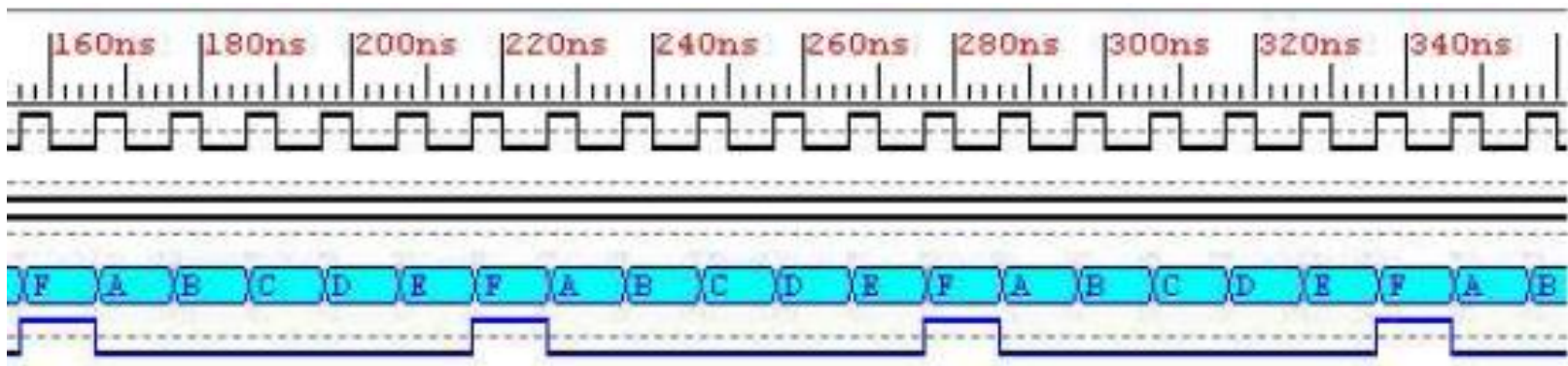


## 2.3.7. Contadores: Divisor de frecuencia

### Contadores binarios síncronos (Aplicaciones)

Permite crear una frecuencia de reloj menor que la de referencia (P.e: de 10ns \* 6 = 60 ns)

- No es necesario usar sus salidas Q. No deben usarse como nueva señal de reloj
- Se usa una única salida que avisa del fin de la cuenta establecida.



## **2.3.8. Descripción VHDL de Contadores**

## 2.3.8. Descripción VHDL de contadores

```
entity Counter_4 is
  Port ( CLK : in  STD_LOGIC;
        Reset : in  STD_LOGIC;
        Enable : in  STD_LOGIC;
        Q : out  STD_LOGIC_VECTOR (3 downto 0));
end Counter_4 ;

architecture Behavioral of Counter_4 is
  signal CUENTA: unsigned (3 downto 0); -- Señal donde se acumula
                                         --la cuenta

begin
  process (CLK, Reset)
  begin
    if Reset = '1' then -- reset asíncrono
      CUENTA <= "0000";
    elsif rising_edge(CLK) then
      if Enable='1' then
        CUENTA <= CUENTA + 1;
      end if;
    end if;
  end process;

  Q <= std_logic_vector(CUENTA); -- asignación al puerto de salida el
                                -- valor de la señal CUENTA. Conversión.

end Behavioral;
```

leee.numeric\_std

## **2.3.9. Divisor de frecuencia**



## 2.3.9. Divisor de frecuencia

### Concepto:

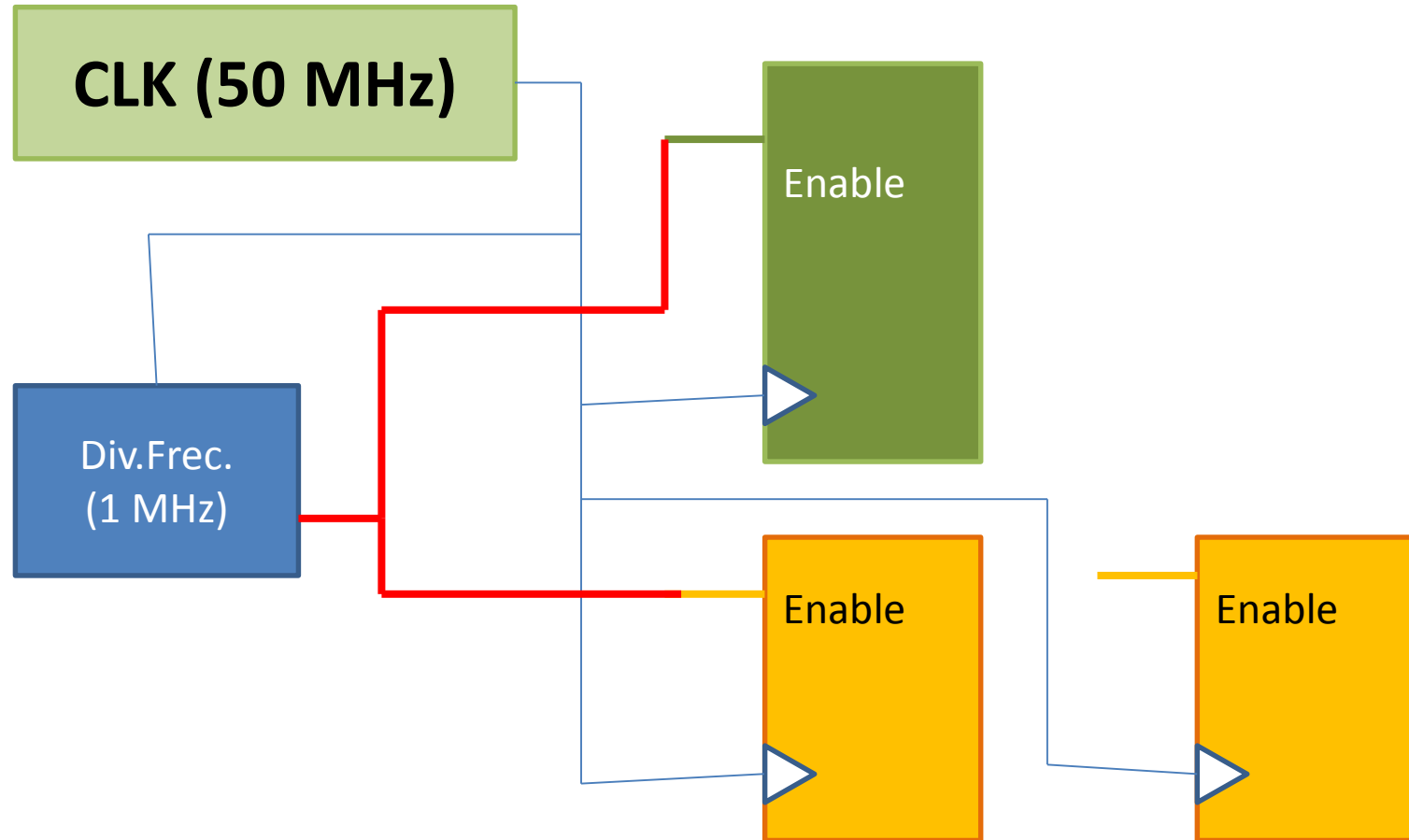
“Permite generar una señal de reloj de frecuencia inferior a la ofrecida por el CLK de sistema”.

### Aplicaciones:

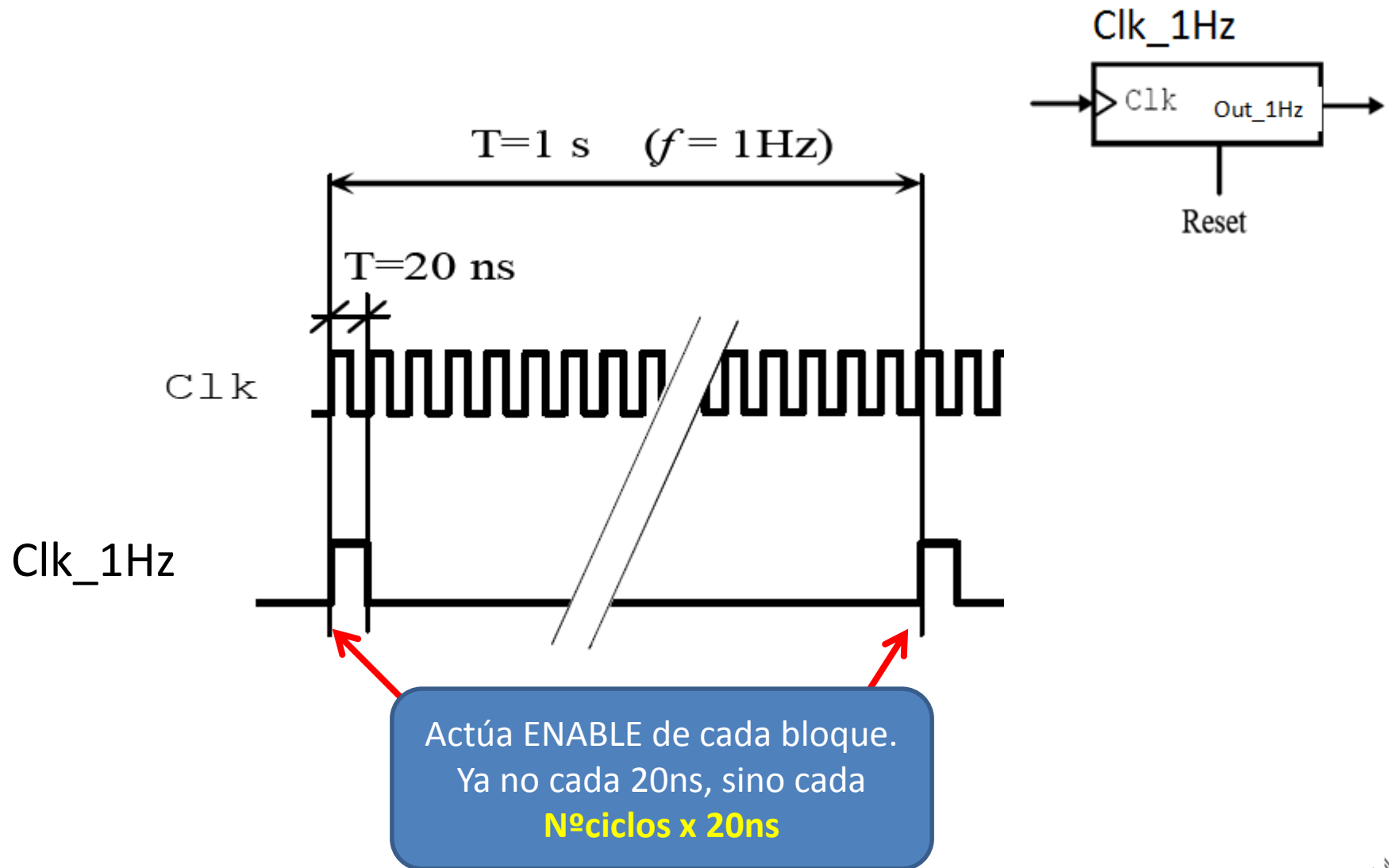
- 1.- Controlar la entrada Enable de los elementos secuenciales de un sistema que deban funcionar/actualizarse a un “ritmo” inferior al reloj del sistema.
- 2.- Crear un CLK\_de sistema de inferior frecuencia

## 2.3.9. Divisor de frecuencia

### Aplicación 1: Control de Enable



## 2.3.9. Divisor de frecuencia



## 2.3.9. Descripción VHDL de un divisor de frecuencia

```
architecture Behavioral of Clk_1Hz is
    constant EndCount: integer = 50000000;           -- Número de ciclos de CLK a contar.
    signal Count: integer range 0 to EndCount;        -- Subrango para contar de 0 a 50 mill.

begin
    process (CLK, RESET)
    begin
        -----
        -- RESET pone a 0 el contador y la señal de salida
        -----

        if RESET='1' then
            Count <= 0;
            Out_1Hz <='0';
            -----

            -- Mantiene a 0 la salida hasta que transcurren 5000000 ciclos
            -- entonces asigna '1' a la salida durante un ciclo de CLK
            -----

        elsif rising_edge(CLK) then
            if Count = EndCount - 1 then
                Count <= 0;
                Out_1Hz <='1';
                -- Si ya ha contado 50.000.000 ciclos de CLK
                -- Pone el contador a 0
                -- Manda un '1' durante un ciclo de CLK

            else
                Count <= Count + 1;
                -- Si no ha llegado al final de la cuenta
                -- incrementa la cuenta de ciclos

                Out_1Hz <='0';
                -- Mantiene la salida a '0', no hay pulso.

            end if;
        end if;
    end process;
end Behavioral;
```

Tema 2.2. (T.35)

### Tipos de datos: INTEGER (Predefinido VHDL)

❑ Está incluido en el lenguaje VHDL, en la librería Standard (Invocado por defecto en cualquier módulo VHDL).

❑ Enteros positivos y negativos si no se define un subrango.

$[-(2^{31}-1), +(2^{31}-1)] = [-2.147.483.647, +2.147.483.647]$

❑ Para usarlo en síntesis es recomendable definir un subrango.

**signal Entero: integer range -2 to 12;**

❑ Se recomienda usar solo para constantes

## Tipos de objetos en VHDL: **constantes**

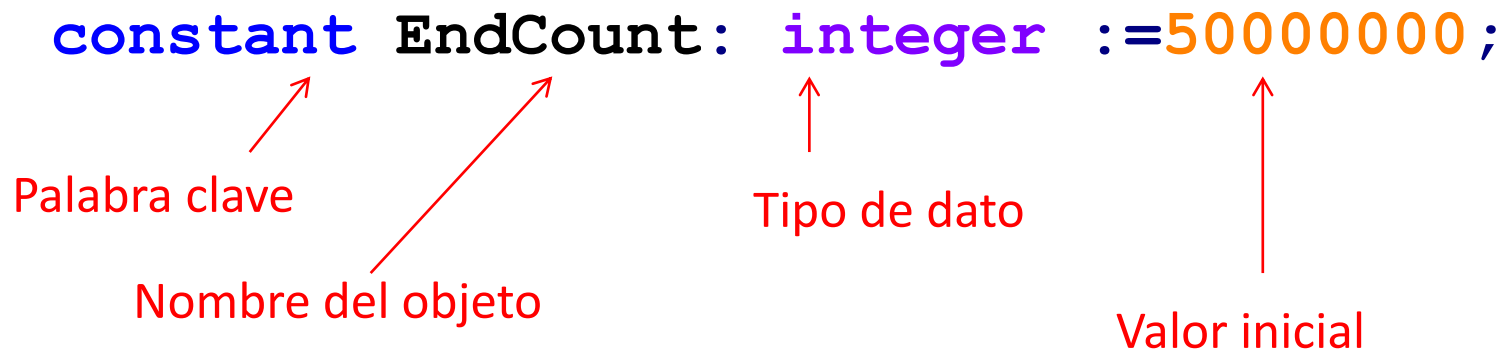
### Tipos de objetos de datos en VHDL para síntesis

#### Constantes

- ❑ Se pueden declarar en cualquier ámbito (arquitectura)
- ❑ Similar a una constante “software”
- ❑ Su uso es interesante cuando se prevé la reutilización del módulo
- ❑ Hacen más claro el código

**constant** EndCount: **integer** := **50000000** ;

Palabra clave      Nombre del objeto      Tipo de dato      Valor inicial



The diagram illustrates the syntax of a VHDL constant declaration. It shows the code 'constant EndCount: integer := 50000000 ;' with four red arrows pointing from labels below to specific parts of the code: 'Palabra clave' points to 'constant', 'Nombre del objeto' points to 'EndCount', 'Tipo de dato' points to 'integer', and 'Valor inicial' points to '50000000'.

## 2.3.9. Parpadeo de un LED. Versión Behavioral (Dos procesos)

```
architecture Behavioral of BlinkLED_Behavioral is
  constant EndCount: integer := 50000000; -- Ciclos a contar.
  signal Count: integer range 0 to EndCount;

  signal out_1Hz: std_logic; -- Salida del Counter Process

  signal CLK_1Hz: std_logic; -- Salida del FFTProcess

begin
  -----
  -- COUNTER PROCESS
  -----
  Counter: process (CLK, RESET)
  begin
    if RESET='1' then
      Count <= 0;
      Out_1Hz <='0';
    elsif rising_edge(CLK) then
      if Count = EndCount-1 then -- Si ya ha contado 50.000.000
                                -- ciclos de CLK
        Count <= 0; -- Pone el contador a 0
        Out_1Hz <='1'; -- Manda un '1' durante un ciclo de CLK
      else
        Count <= Count+1; - Si no ha llegado al final de la cuenta
                          - incrementa la cuenta de ciclos.
        Out_1Hz <='0'; -- Mantiene la salida a '0', no hay pulso.
      end if;
    end if;
  end process;
```

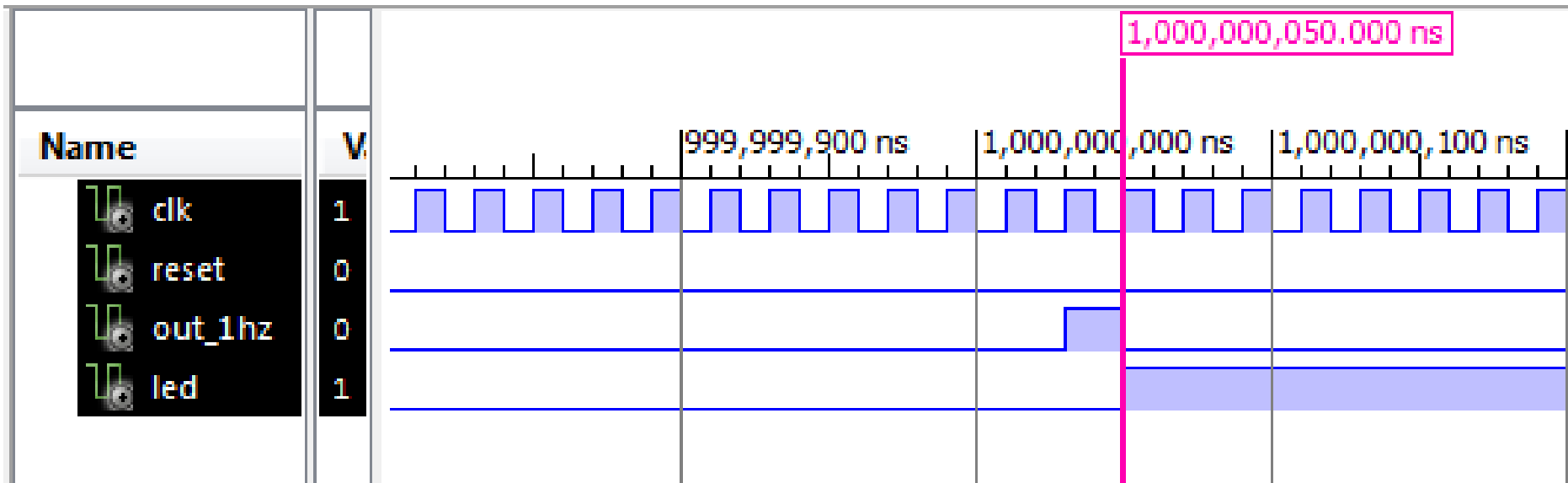
```
-----
-- FFT PROCESS
-----
FFT: process(CLK, RESET)
begin
  if RESET='1' then
    CLK_1Hz <='0';
  elsif rising_edge(CLK) then
    if Out_1Hz='1' then
      -- Sería como el Enable del FFT
      CLK_1Hz <= not CLK_1Hz;
    end if;
  end if;
end process;

LED <= CLK_1Hz; -- Asignación de valor a la salida

end Behavioral;
```

Proyecto21. Módulo  
BlinkLED\_Behavioral

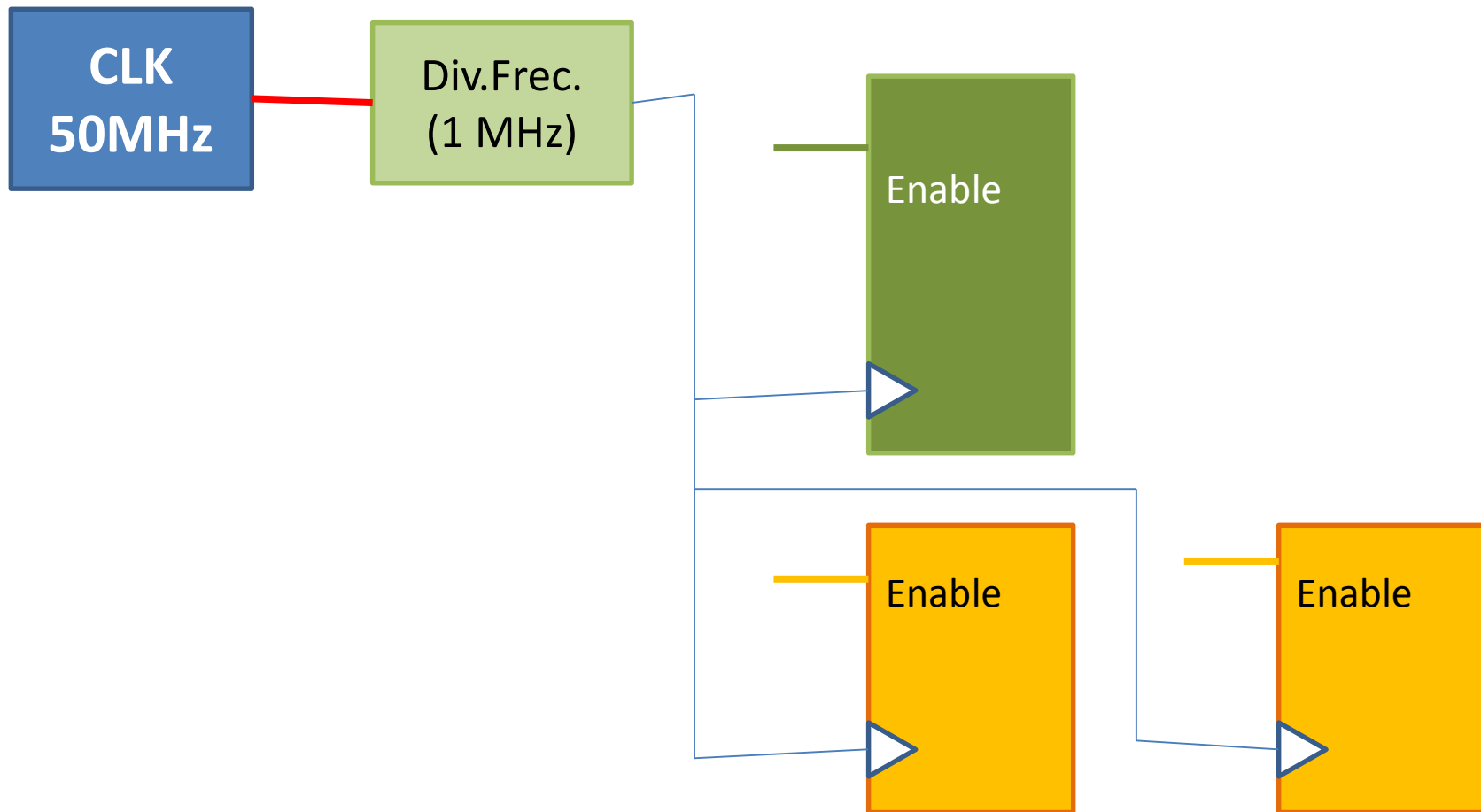
## 2.3.9. Parpadeo de un LED. Versión Behavioral (Dos procesos)





## 2.3.9. Divisor de frecuencia

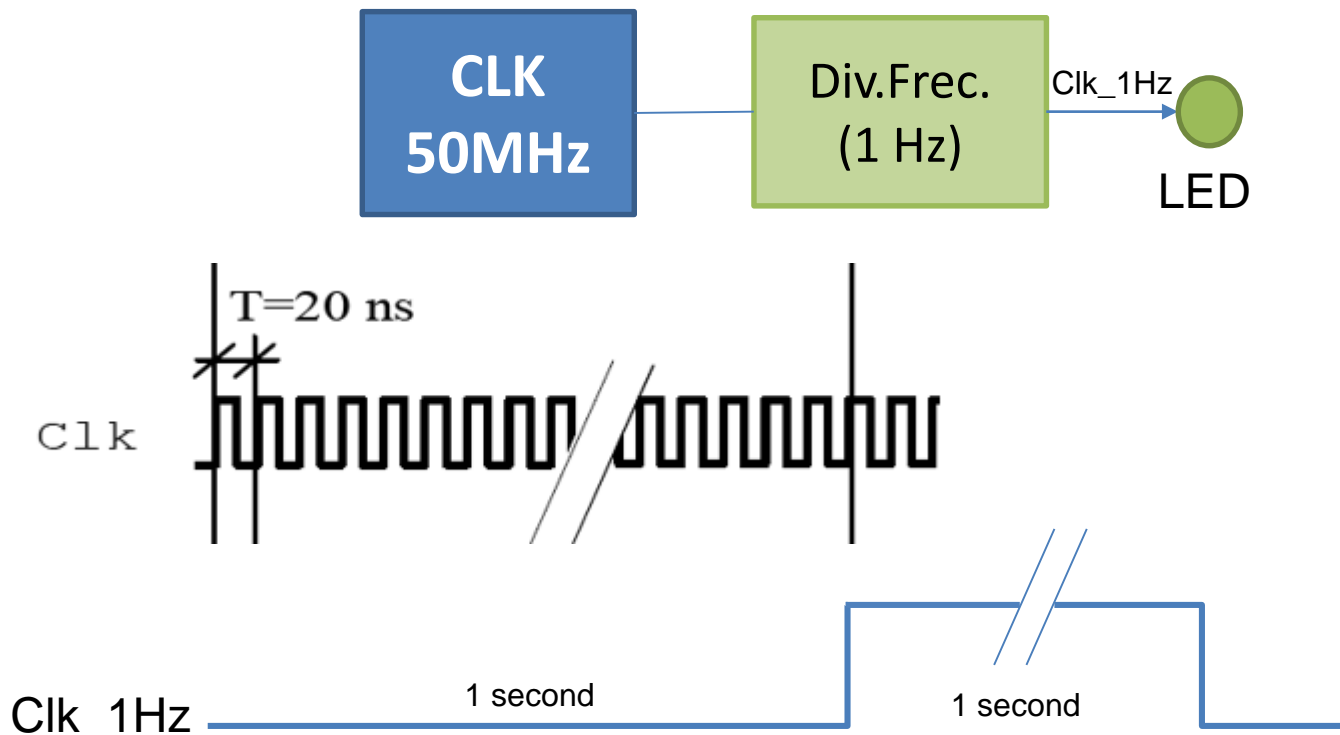
### Aplicación 2: Reloj de menor frecuencia



## 2.3.9. Divisor de frecuencia

### Aplicación 2: Reloj de menor frecuencia para conectar al CLK sistema

**Proyecto21:** Hacer parpadear un LED. Añadiremos a este proyecto un tercer módulo denominado **BlinkLED\_1Process**.



# BIBLIOGRAFIA

- **Free range VHDL.** Bryan Mealy, Fabrizio Tappero. (Creative Commons). <http://www.freerangefactory.org> (Mayo 2013)
- **Diseño de circuitos digitales con VHDL (URJC)**