ETSIST-UPM
Dpto. de Ingeniería Telemática y Electrónica

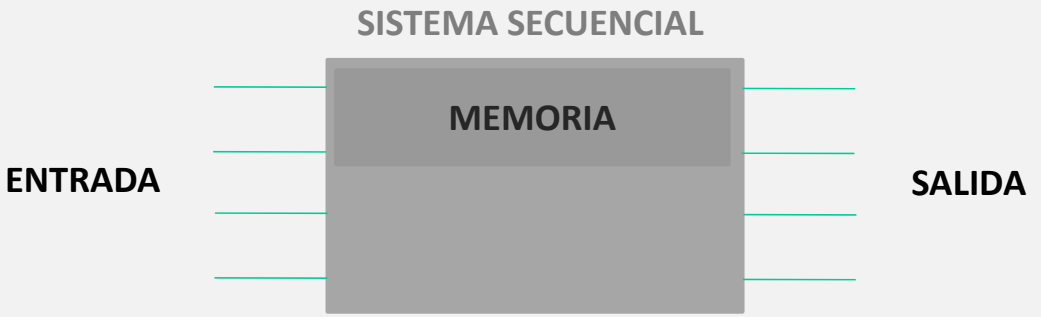
Diseño Digital 1
Bloque temático 1

BT1.P3
Modelado de Circuitos Secuenciales Síncronos Simples con VHDL

Principios de Funcionamiento de los Sistemas Secuenciales (I)

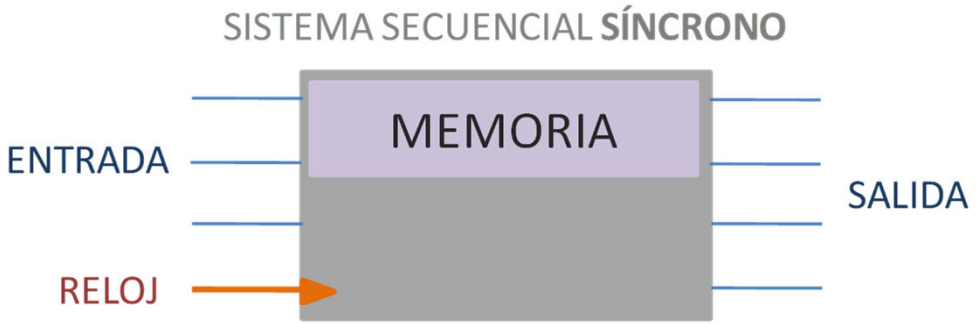
La respuesta de un sistema secuencial depende de la entrada y del estado de la memoria del sistema.

SALIDA = F (ENTRADA, MEMORIA)

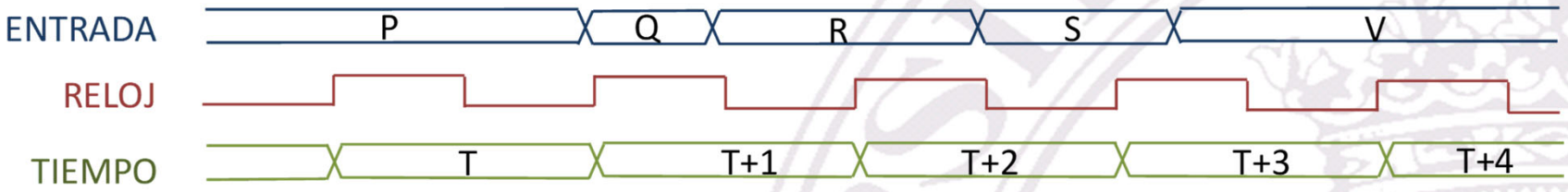


LA SALIDA PARA UNA MISMA COMBINACIÓN DE ENTRADA PUEDE SER DISTINTA EN DIFERENTES INSTANTES DE TIEMPO

Principios de Funcionamiento de los Sistemas Secuenciales (II)



LOS SISTEMAS EN QUE LA SUCESIÓN DE INSTANTES DE TIEMPO VIENE DEFINIDA POR UNA SEÑAL DE TIEMPO (**SEÑAL DE RELOJ**), SE DENOMINAN **SISTEMAS SECUENCIALES SÍNCRONOS**



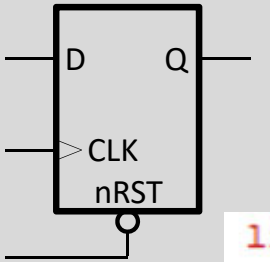
LOS SISTEMAS **SECUENCIALES SÍNCRONOS** MÁS **SIMPLES** SON AQUELLOS EN LOS QUE **LA SALIDA ES LA MEMORIA DE ESTADO**:

- **FLIP-FLOPS**
- **REGISTROS**
- **CONTADORES**

Modelado VHDL de la Interfaz de los Sistemas Secuenciales Síncronos Simples

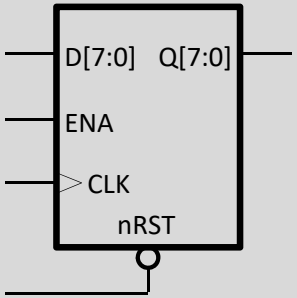
El modelo VHDL de **la interfaz** de un Sistema Secuencial Síncrono se materializa en una Declaración de Entidad, aplicando las reglas que ya conoce para el modelado de la interfaz de Sistemas Combinacionales.

FLIP-FLOP D




```
library ieee;
use ieee.std_logic_1164.all;

entity flip_flop_D is
port(
     CLK:  in std_logic;
     nRST: in std_logic;
     D:    in std_logic;
     Q:    buffer std_logic
     );
end entity;
```

REGISTRO 8 BITS




```
library ieee;
use ieee.std_logic_1164.all;

entity registro_8bits is
port(
     CLK:  in std_logic;
     nRST: in std_logic;
     ENA:  in std_logic;
     D:    in std_logic_vector(7 downto 0);
     Q:    buffer std_logic_vector(7 downto 0)
     );
end entity;
```

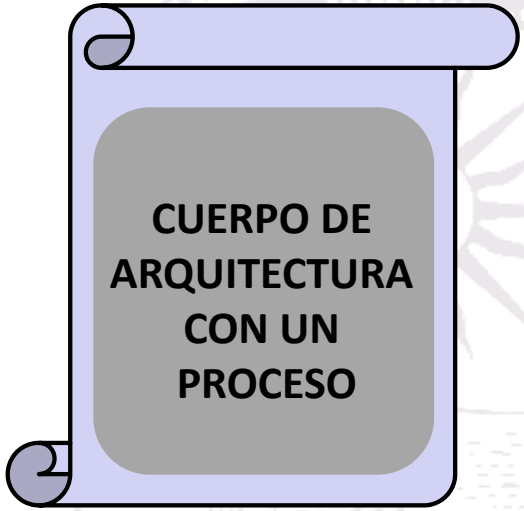
Modelado VHDL del funcionamiento de los Sistemas Secuenciales Síncronos Simples

En los circuitos secuenciales síncronos en los que la salida es el estado de memoria del circuito (**flip-flops, registros y contadores**), el **modelado del funcionamiento** puede realizarse de una manera muy sencilla y compacta: **en un proceso VHDL y con un esquema de codificación muy simple**.

ARQUITECTURAS DE MOORE SIN LÓGICA COMBINACIONAL DE SALIDA

- **FLIP-FLOPS**
- **REGISTROS**
- **CONTADORES**

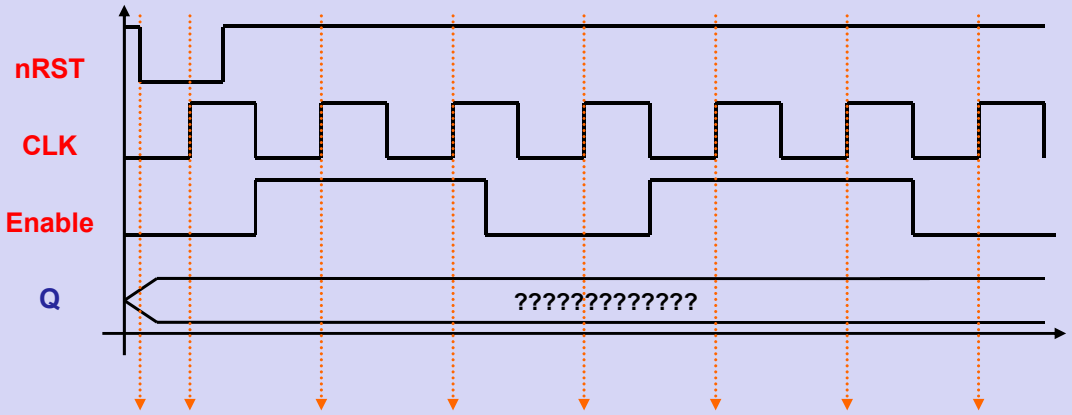
MODELO DE FUNCIONAMIENTO VHDL



Procesos para el modelado de Sistemas Secuenciales Síncronos Simples

- El proceso tiene que ser sensible **EXCLUSIVAMENTE** a la señal de reloj del circuito y a la entrada de inicialización asíncrona (reset asíncrono).
- **NO DEBE** ser sensible a las entradas síncronas que pudiera tener el sistema.

En un sistema secuencial síncrono **la salida sólo puede cambiar** cuando se activa la **entrada asíncrona de inicialización** o cuando ocurre **un flanco activo de reloj**.



Un **proceso** VHDL sólo **se ejecuta** cuando hay **eventos** (cambios de valor) en las **señales** a las que es **sensible**.

```
process(CLK, nRST)
begin
 ........



end process;
```

Procesos para el modelado de Sistemas Secuenciales Síncronos Simples

- El proceso debe asignar valor a la salida del sistema secuencial dando **prioridad al funcionamiento asíncrono** sobre **el síncrono**.

- Dentro del proceso que modela el circuito secuencial síncrono hay **una sentencia IF** que **prioriza la evaluación de la actividad de la entrada de inicialización asíncrona** sobre la evaluación del funcionamiento síncrono.

```
process(CLK, nRST)
begin
  if condición nRST activo then
    Inicialización asíncrona

  elsif ocurrencia del flanco activo de reloj then
    Especificacion del funcionamiento Síncrono

  end if;

end process;
```

Cada vez que se ejecuta el proceso, **si la entrada de inicialización asíncrona está activa**, se asigna a la salida el valor de inicialización del sistema.

SI la entrada de **inicialización asíncrona no está activa** y ha ocurrido un **flanco activo de reloj**, se asigna valor a la salida atendiendo al **funcionamiento síncrono** del sistema.

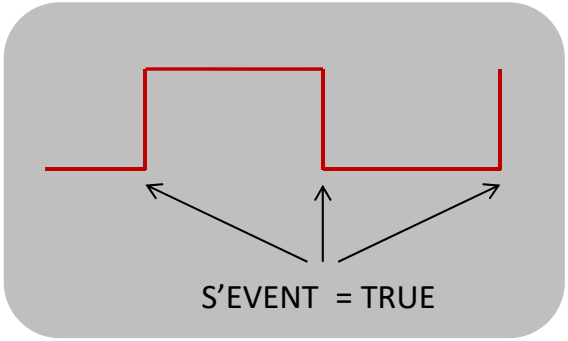
Procesos para el modelado de Sistemas Secuenciales Síncronos Simples

- Las entradas de **inicialización** asíncrona suelen ser **activas a nivel bajo**.

```
process(CLK, nRST)
begin
  if nRST = '0' then
    Inicialización asíncrona
```

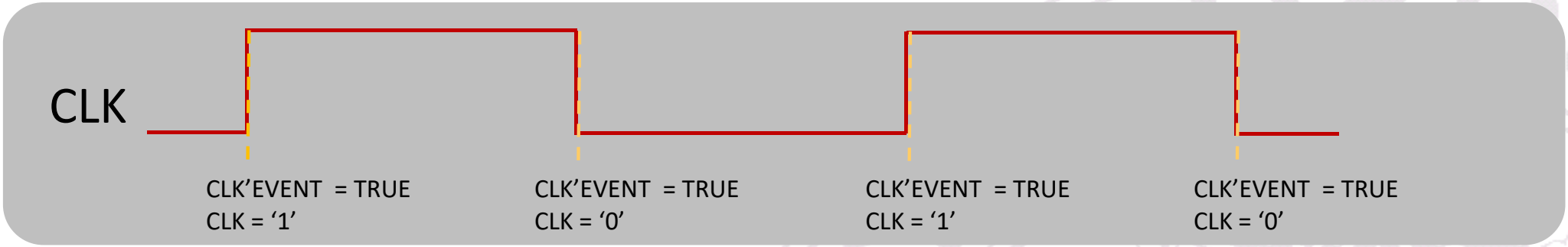
- El atributo VHDL **EVENT** de una señal, **S**, se escribe **S'EVENT** y es un valor booleano (puede tomar el valor **TRUE** o **FALSE**).

- **S'EVENT** toma el valor **TRUE**, durante una simulación, en los **instantes del tiempo simulación en que hay eventos** (cambios de valor) de la señal **S**.



Procesos para el modelado de Sistemas Secuenciales Síncronos Simples

- Para codificar la condición de **ocurrencia de un flanco activo de reloj** hay que emplear el atributo **EVENT**.



FLANCO DE SUBIDA

```
CLK'EVENT and CLK = '1'
```

FLANCO DE BAJADA

```
CLK'EVENT and CLK = '0'
```

```
process(CLK, nRST)
begin
  if nRST = '0' then
    Inicialización asíncrona

  elsif CLK'EVENT and CLK = '1' then
    Especificacion del funcionamiento Síncrono

  end if;

end process;
```

Reglas para la realización de procesos que modelen Sistemas Secuenciales Síncronos Simples

- El proceso tiene que ser sensible **EXCLUSIVAMENTE** a la señal de reloj del circuito y a la entrada de inicialización asíncrona (Reset asíncrono).
- Dentro del proceso que modela el circuito secuencial síncrono debe haber **una sentencia IF** que **priorice la evaluación de la actividad de la entrada de inicialización asíncrona** sobre la evaluación del funcionamiento síncrono.
- Las expresiones de control de condiciones de la sentencia **IF** (suponiendo que la entrada asíncrona es activa a nivel bajo y el sistema es activo por flanco de subida) son:
  - **nRST = '0'**
  - **CLK'EVENT and CLK = '1'**
- En la rama del IF correspondiente a la condición de funcionamiento síncrono se especifica el valor que toman las salidas en función de las entradas síncronas de control del sistema.

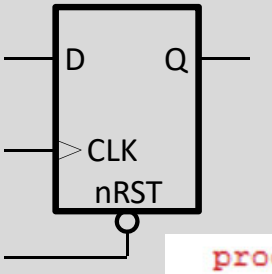
Los procesos que se construyen siguiendo estas reglas pueden **modelar** cualquier sistema secuencial síncrono con **arquitectura de Moore** en el que **la salida del sistema sea la memoria de estado**.

```
process(CLK, nRST)
begin
  if nRST = '0' then
    Inicialización asíncrona

  elsif CLK'EVENT and CLK = '1' then
    Especificacion del funcionamiento Síncrono

  end if;

end process;
```

Procesos para el modelado de Sistemas Secuenciales Síncronos Simples

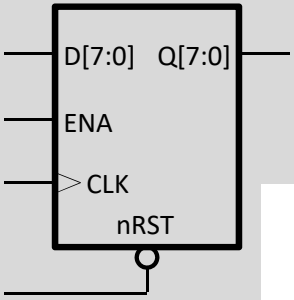
```
process(CLK, nRST)
begin
  if nRST = '0' then
    Inicialización asíncrona

  elsif CLK'EVENT and CLK = '1' then
    Especificacion del funcionamiento Síncrono

  end if;

end process;
```

FLIP-FLOP D




```
process(CLK, nRST)
begin
  if nRST = '0' then
    Q <= '0';

  elsif CLK'EVENT and CLK = '1' then
    Q <= D;

  end if;
end process;
```

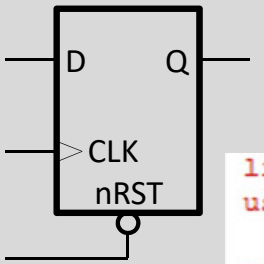
REGISTRO 8 BITS




```
process(CLK, nRST)
begin
  if nRST = '0' then
    Q <= (others => '0');

  elsif CLK'EVENT and CLK = '1' then
    if ENA = '1' then
      Q <= D;

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

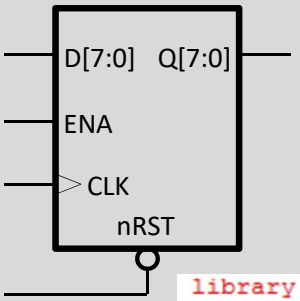
Ejemplos de modelado de Sistemas Secuenciales Síncronos Simples

FLIP-FLOP D



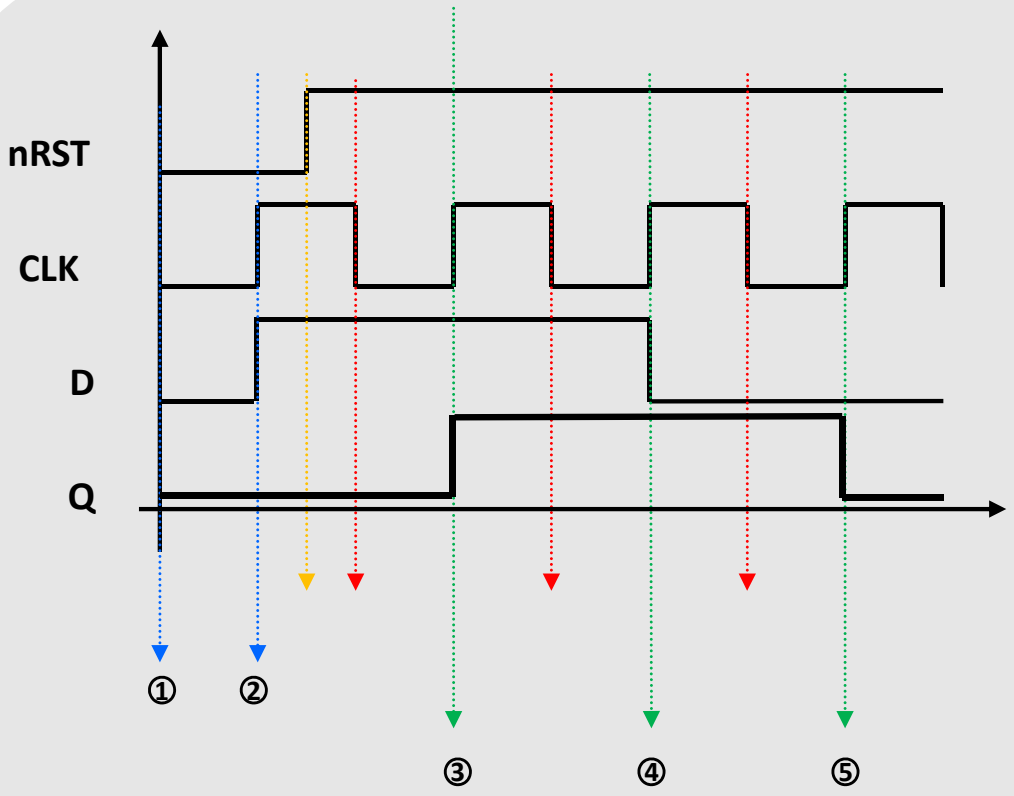

```
library ieee;
use ieee.std_logic_1164.all;

entity flip_flop_D is
port(
     CLK:  in std_logic;
     nRST: in std_logic;
     D:    in std_logic;
     Q:    buffer std_logic
     );
end entity;

architecture rtl of flip_flop_D is
begin
  process(CLK, nRST)
  begin
    if nRST = '0' then
      Q <= '0';

    elsif CLK'EVENT and CLK = '1' then
      Q <= D;

    end if;
  end process;
end rtl;
```

REGISTRO 8 BITS




```
library ieee;
use ieee.std_logic_1164.all;

entity registro_8bits is
port(
     CLK:  in std_logic;
     nRST: in std_logic;
     ENA:  in std_logic;
     D:    in std_logic_vector(7 downto 0);
     Q:    buffer std_logic_vector(7 downto 0)
     );
end entity;

architecture rtl of registro_8bits is
begin
  process(CLK, nRST)
  begin
    if nRST = '0' then
      Q <= (others => '0');

    elsif CLK'EVENT and CLK = '1' then
      if ENA = '1' then
        Q <= D;

      end if;
    end if;
  end process;
end rtl;
```

Simulación del modelo de un Flip-Flop



El proceso se ejecuta cada vez que hay un evento en las señales **nRST** o **CLK**

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity flip_flop_D is
5   port(
6        CLK:  in std_logic;
7        nRST: in std_logic;
8        D:    in std_logic;
9        Q:    buffer std_logic
10       );
11  end entity;
12
13  architecture rtl of flip_flop_D is
14  begin
15    process(CLK, nRST)
16    begin
17      if nRST = '0' then
18        Q <= '0';
19
20      elsif CLK'EVENT and CLK = '1' then
21        Q <= D;
22
23      end if;
24    end process;
25  end rtl;
```

- Eventos con nRST = ‘0’ — **1,2**
- Eventos con CLK’EVENT and CLK = ‘1’ — **3,4,5**
- Eventos con nRST = ‘1’
- Eventos con CLK’EVENT and CLK = ‘0’