ETSIST-UPM
Dpto. de Ingeniería Telemática y Electrónica

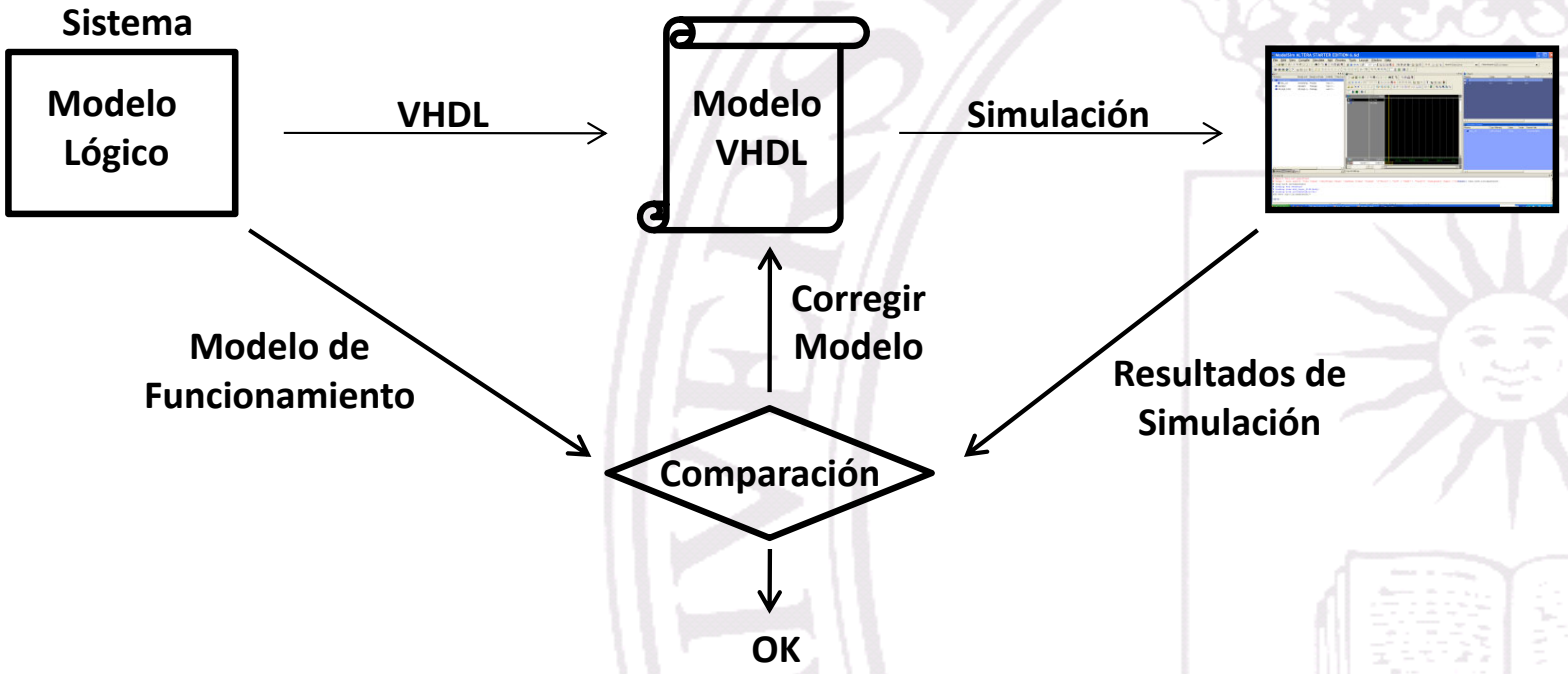
Diseño Digital 1
Bloque temático 1

BT1.P2
Test-Benches VHDL

Modelado basado en simulación

- El código de los modelos lógicos VHDL es ejecutable en un simulador VHDL: los modelos VHDL son simulables.

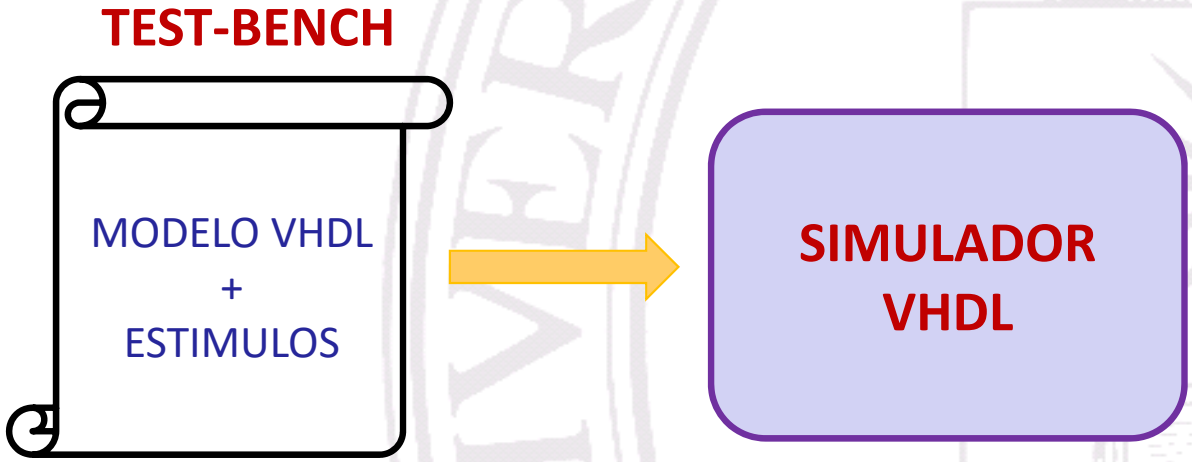
- El modelo VHDL de un sistema digital es correcto si al simularlo su comportamiento coincide con el del sistema que se pretende modelar.



Test-Bench VHDL

• Los simuladores VHDL ejecutan el código contenido en una Declaración de Entidad, sin puertos, y un Cuerpo de Arquitectura en el que se inserta el modelo que se quiere simular y donde se definen los estímulos con los que se desea probar el modelo.

• A esta pareja de unidades código, con estas características y finalidad, se las denomina TEST-BENCH (Banco de Pruebas).



Señales y sentencias de emplazamiento

• En un Test-Bench, el modelo del sistema que se quiere simular se inserta en un Cuerpo de Arquitectura utilizando una sentencia de emplazamiento.

• La sentencia de emplazamiento permite conectar el modelo a señales declaradas en el Cuerpo de Arquitectura.

• Las señales que se conectan al modelo deben declararse en el Cuerpo de Arquitectura.

• Sintaxis de las sentencias de declaración de señal:

```
signal nombre_de_señal: tipo de datos;
```

Ejemplos:

```
signal s_enable: std_logic;
signal s_Dato:   std_logic_vector(3 downto 0);
```

Test-Bench VHDL

• La sentencia de emplazamiento identifica el modelo que se inserta haciendo referencia a su Declaración de Entidad.

• Cada puerto de la declaración de entidad se conecta a una de las señales declaradas en el Test-Bench.

• Sintaxis de las sentencias de emplazamiento:

```
identificador: entity Work.nombre_de_entidad(nombre_de_arquitectura)
                port map
                        (lista de conexión);
```

La lista de conexión es una lista que indica qué señal está conectada a cada puerto:

```
puerto => señal,
```

Ejemplo: Emplazamiento y conexión de un Multiplexor

SELECTOR DE DATOS DE 8 BITS

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

entity Selector8bits is
port(DatoA: in std_logic_vector(7 downto 0);
     DatoB: in std_logic_vector(7 downto 0);
     Sel:   in std_logic;
     DatoSelect: buffer std_logic_vector(7 downto 0)
   );
end entity;
```

Nombre de arquitectura: **test**

El tipo de la señal tiene que coincidir con el del puerto al que se conecta.

Identificador de emplazamiento: **dut**

Test-Bench INCOMPLETO !!!!

Hay que definir los valores que toman los estímulos (s_DatoA, s_DatoB y s_Sel) durante la simulación.

TEST-BENCH DEL SELECTOR DE DATOS

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

entity test_Selector8bits is
end entity;

architecture test of test_Selector8bits is
  signal s_DatoA: std_logic_vector(7 downto 0);
  signal s_DatoB: std_logic_vector(7 downto 0);
  signal s_Sel:     std_logic;
  signal s_DatoSelect: std_logic_vector(7 downto 0);

begin

  dut: entity Work.Selector8bits(rtl)
       port map(DatoA => s_DatoA,
                DatoB => s_DatoB,
                Sel => s_Sel,
                DatoSelect => s_DatoSelect);

end test;
```

(1)

(2)

(3)

(1) Declaración de Entidad vacía
(2) Declaración de señales
(3) Sentencia de emplazamiento

Ejemplo: Emplazamiento y conexión de un Decodificador

DECODIFICADOR

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

entity Decodificador3a8 is
port(Enable: in std_logic;
     DatoIn: in std_logic_vector(2 downto 0);
     DatoOut: buffer std_logic_vector(7 downto 0)
    );
end entity;
```

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

entity test_Decodificador3a8 is
end entity;

architecture test of test_Decodificador3a8 is
  signal Enable:  std_logic;
  signal DatoIn:  std_logic_vector(2 downto 0);
  signal DatoOut: std_logic_vector(7 downto 0);

begin

  dut: entity Work.Decodificador3a8(rtl)
       port map(Enable => Enable,
                DatoIn => DatoIn,
                DatoOut => DatoOut);

end test;
```

(1)

(2)

(3)

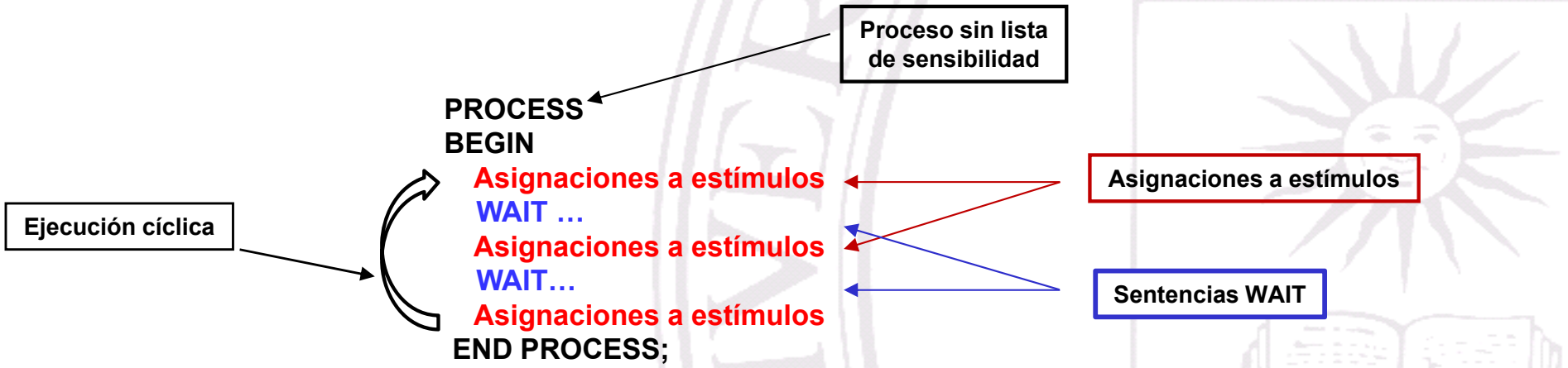
A cada señal suele dársele el mismo nombre que al puerto al que se conecta.

Las señales conectadas a entradas del modelo (Enable, DatoIn), son los estímulos con los que se dirige la simulación, las conectadas a las salidas (DatoOut), muestran la respuesta del modelo a dichos estímulos.

(1) Declaración de Entidad vacía
(2) Declaración de señales
(3) Sentencia de emplazamiento

Definición de estímulos en un Test-Bench

- Los estímulos que componen las pruebas que se realizan en una simulación se definen en procesos que contienen sentencias WAIT.
  - Son procesos que no tienen lista de sensibilidad.
  - Cada vez que se alcanza una sentencia WAIT el proceso se suspende.
  - El proceso se ejecuta cíclicamente: a la ejecución de la última sentencia del proceso le sucede la ejecución de la primera.



Definición de estímulos en un Test-Bench

- Cuando se ejecuta una sentencia WAIT el proceso se suspende.

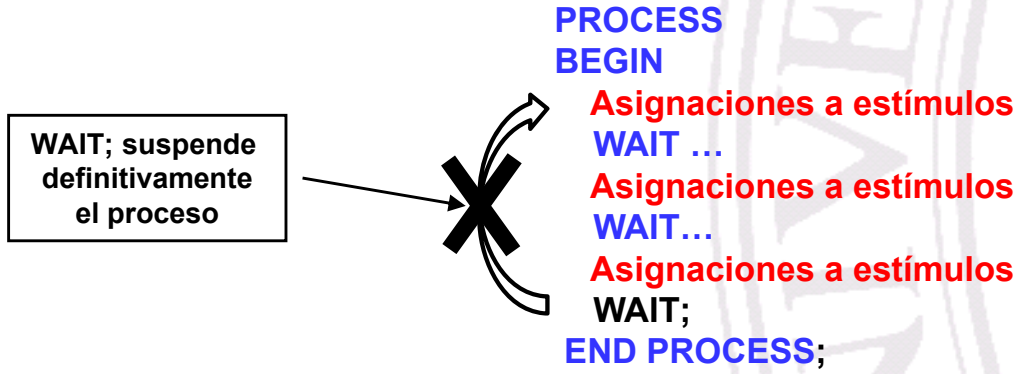
  WAIT FOR suspende el proceso durante un tiempo igual al especificado en la sentencia.

  ```
  WAIT FOR 50 ns; -- la ejecución del proceso se reanuda 50 ns después de
                  -- suspenderse.
  ```

  WAIT detiene definitivamente la ejecución del proceso.

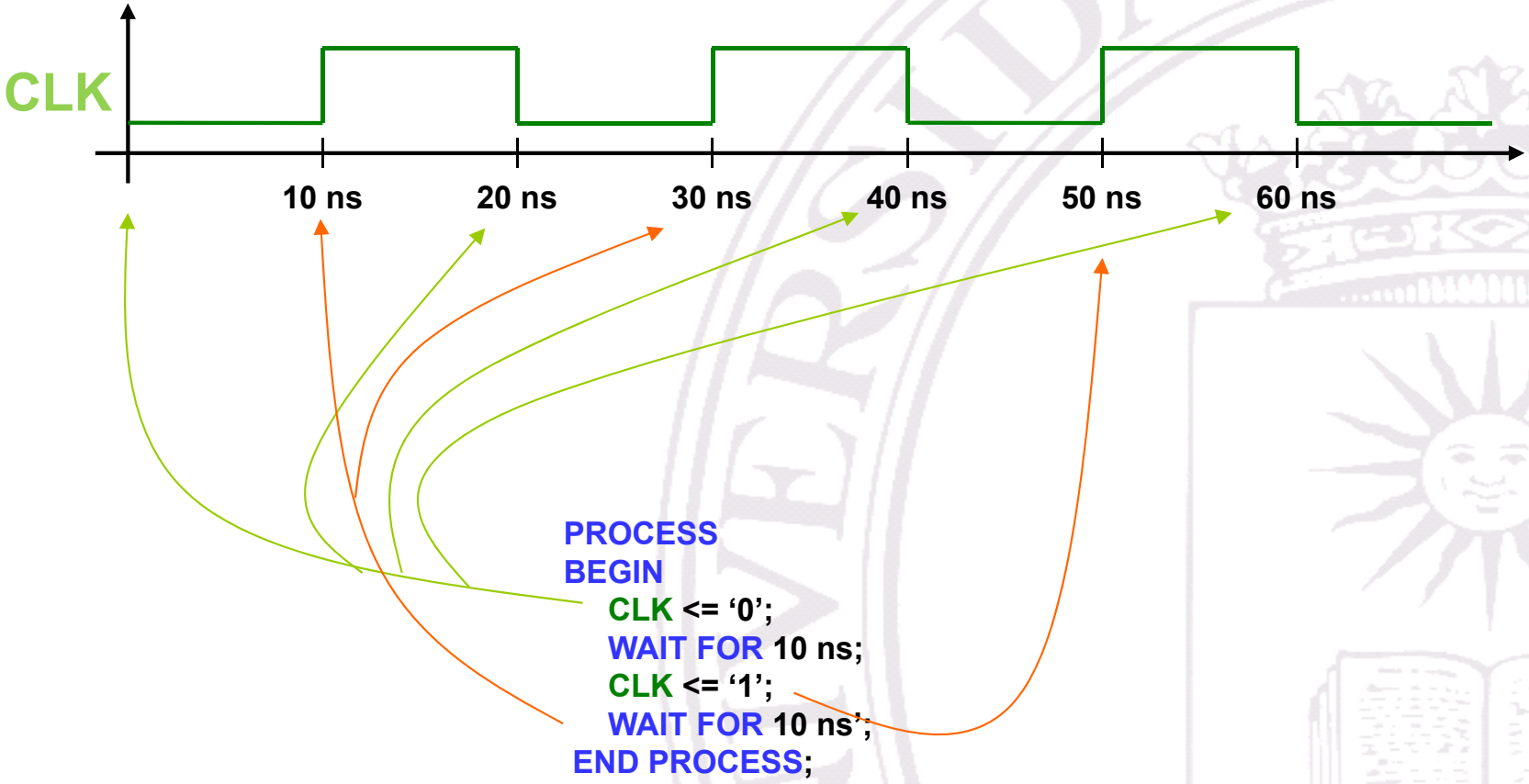
  ```
  WAIT; -- la ejecución del proceso no se vuelve a reanudar.
  ```

  WAIT UNTIL y WAIT ON se estudiarán más adelante.



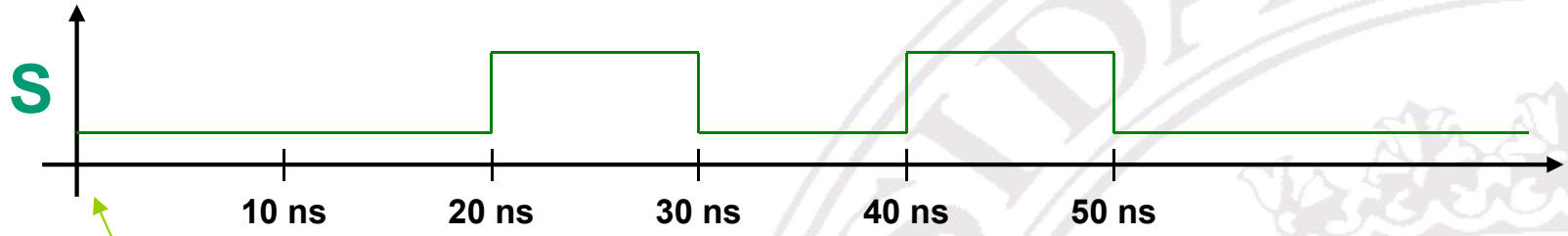
Ejecución cíclica de procesos

- Los procesos se empiezan a ejecutar en el tiempo de simulación T = 0.
- Después de ejecutar la última sentencia se continúa por la primera: este mecanismo se puede aprovechar para generar formas periódicas.



Procesos VHDL para el manejo de estímulos

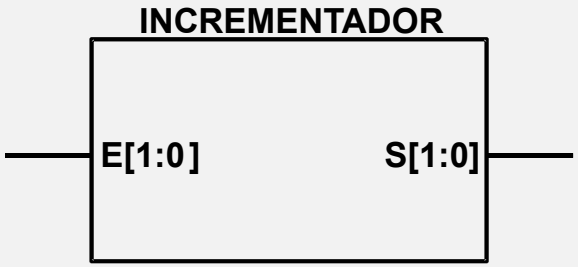
Cuando se ejecuta una sentencia WAIT FOR, la ejecución del proceso se detiene hasta que transcurre el tiempo de simulación indicado en la misma. El valor de tiempo se indica con un número entero y una escala (fs, ps, ns, us, ms, s, min, hr) que va desde los femtosegundos hasta las horas.




```
PROCESS
BEGIN
    S <= '0';
    WAIT FOR 20 ns;
    S <= '1';
    WAIT FOR 10 ns';
    S <= '0';
    WAIT FOR 10 ns;
    S <= '1';
    WAIT FOR 10 ns;
    S <= '0';
    WAIT;
END PROCESS;
```

Ejemplo

Modelo bajo prueba: Incrementador de 2 bits

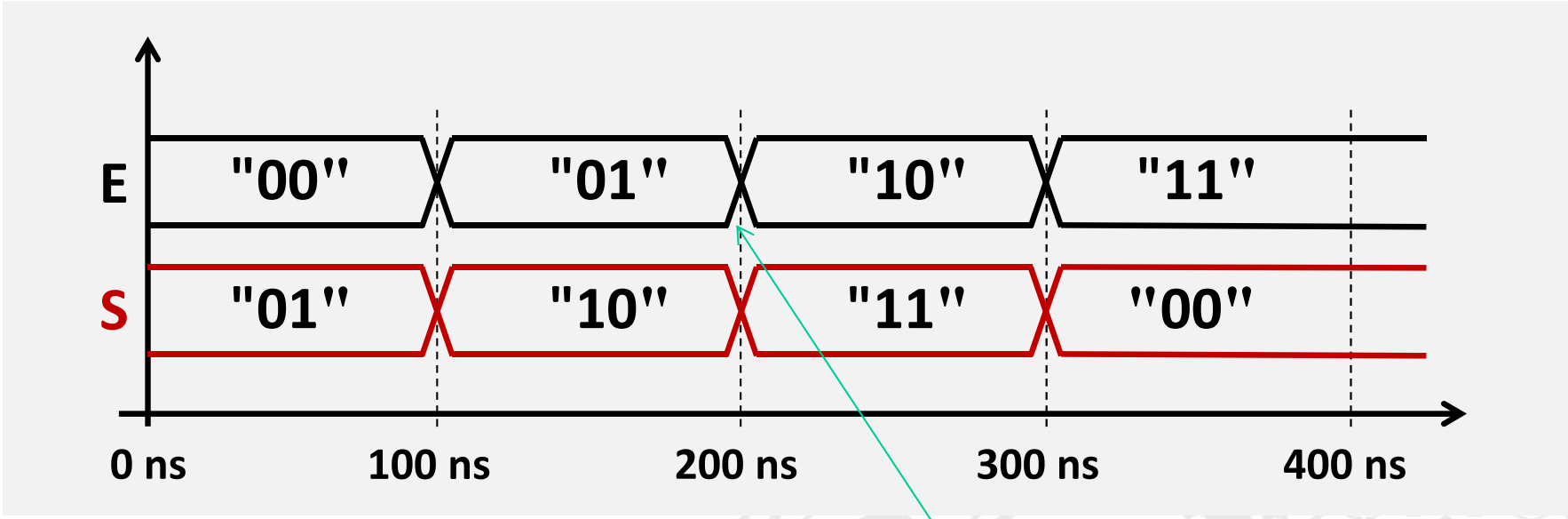


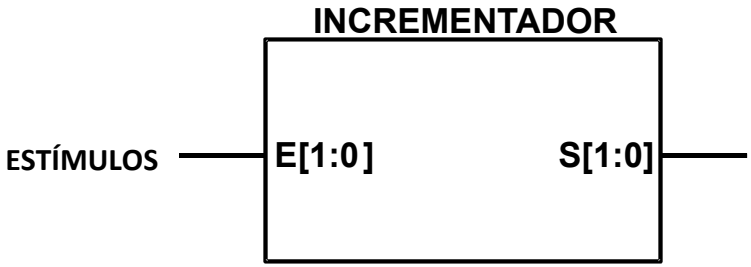
E1	E0	S1	S0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

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

entity INCREMENTADOR is
port(
     E: in std_logic_vector(1 downto 0);
     S: buffer std_logic_vector(1 downto 0)
     );
end entity;

architecture RTL of INCREMENTADOR is
begin
  process(E)
  begin
    case(E) is
      when "00" => S <= "01";
      when "01" => S <= "10";
      when "10" => S <= "11";
      when "11" => S <= "00";
      when others => S <= "XX";
    end case;
  end process;
end RTL;
```

Ejemplo

CODIFICACIÓN DE ESTÍMULOS

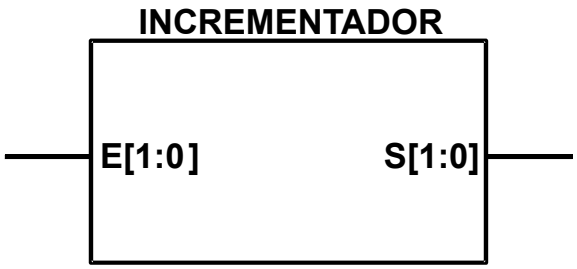




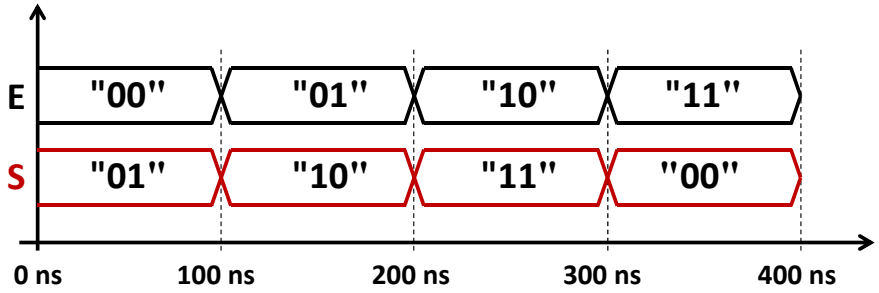

```
--Definición de estímulos
process
begin
  E <= "00";
  wait for 100 ns;
  E <= "01";
  wait for 100 ns;
  E <= "10";
  wait for 100 ns;
  E <= "11";
  wait;
end process;
```

Test-Bench del Incrementador

MODELO LÓGICO



ESTÍMULOS




```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity test_incrementador is
5   end entity;
6
7   architecture test of test_incrementador is
8     -- Estímulos
9     signal E: std_logic_vector(1 downto 0);
10    signal s: std_logic_vector(1 downto 0);
11
12  begin
13
14    -- Emplazamiento y conexión del modelo
15    DUT: entity Work.INCREMENTADOR(rtl)
16         port map(E => E,
17                  S => S);
18
19    --Definición de estímulos
20    process
21    begin
22      E <= "00";
23      wait for 100 ns;
24      E <= "01";
25      wait for 100 ns;
26      E <= "10";
27      wait for 100 ns;
28      E <= "11";
29      wait;
30    end process;
31  end test;
```

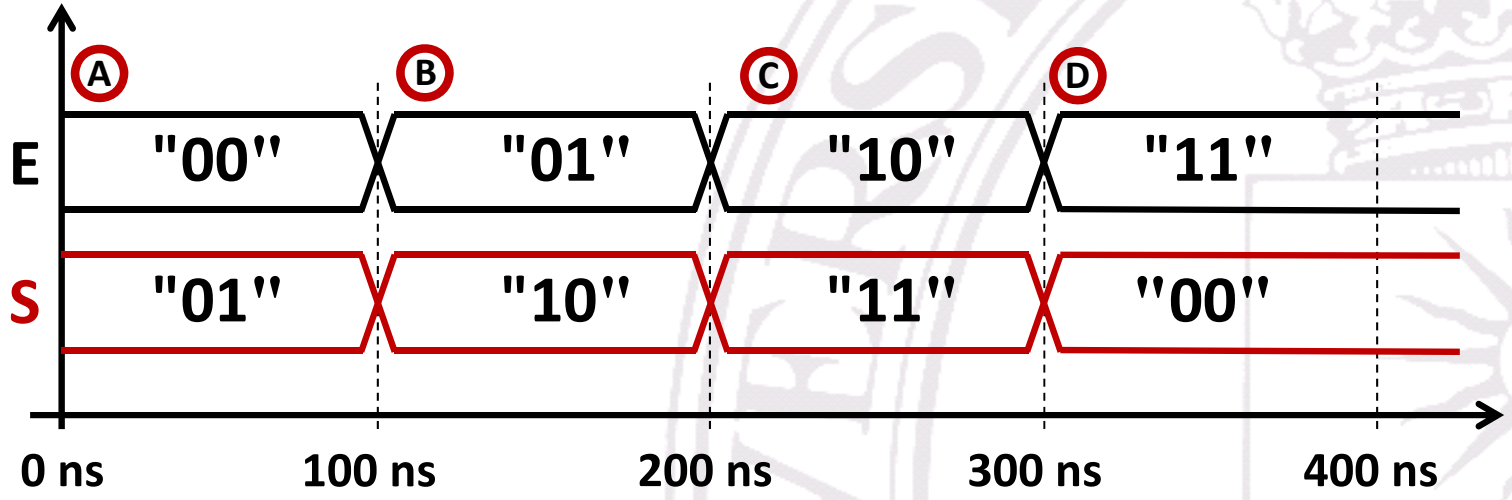
Test-Bench del Incrementador

```
20    process
21    begin
22      E <= "00";
23      wait for 100 ns;
24      E <= "01";
25      wait for 100 ns;
26      E <= "10";
27      wait for 100 ns;
28      E <= "11";
29      wait;
30    end process;
```

A → línea 22 · B → línea 24 · C → línea 26 · D → línea 28

```
13    process(E)
14    begin
15      case(E) is
16        when "00" => S <= "01";
17        when "01" => S <= "10";
18        when "10" => S <= "11";
19        when "11" => S <= "00";
20        when others => S <= "XX";
21      end case;
22    end process;
```

A → línea 16 · B → línea 17 · C → línea 18 · D → línea 19



RESULTADOS DE SIMULACIÓN CORRECTOS ⟹ MODELO VALIDADO