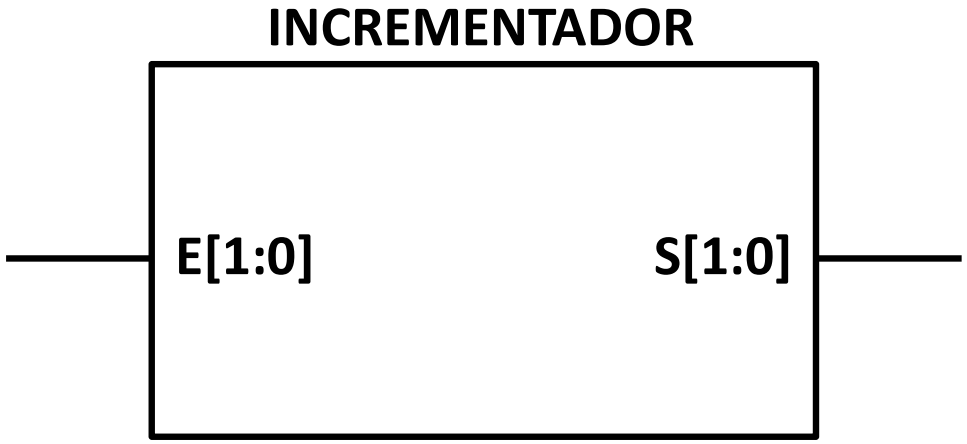
Diseño Digital I

Revisión del Modelado y Simulación VHDL de Sistemas Digitales

Introducción a los HDLs

La descripción del modelo lógico de un circuito digital consiste en la definición de su Interfaz (características de sus entradas y salidas) y de su Funcionamiento (relación funcional entre entradas y salidas).



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

Los Lenguajes para la Descripción de Hardware (HDLs, *Hardware Description Languages*) permiten definir el modelo lógico de los sistemas digitales empleando un lenguaje formal.

Introducción a los HDLs

Los HDLs tienen una sintaxis similar a los lenguajes de programación, pero una semántica diferente: sirven para construir modelos lógicos de sistemas digitales.

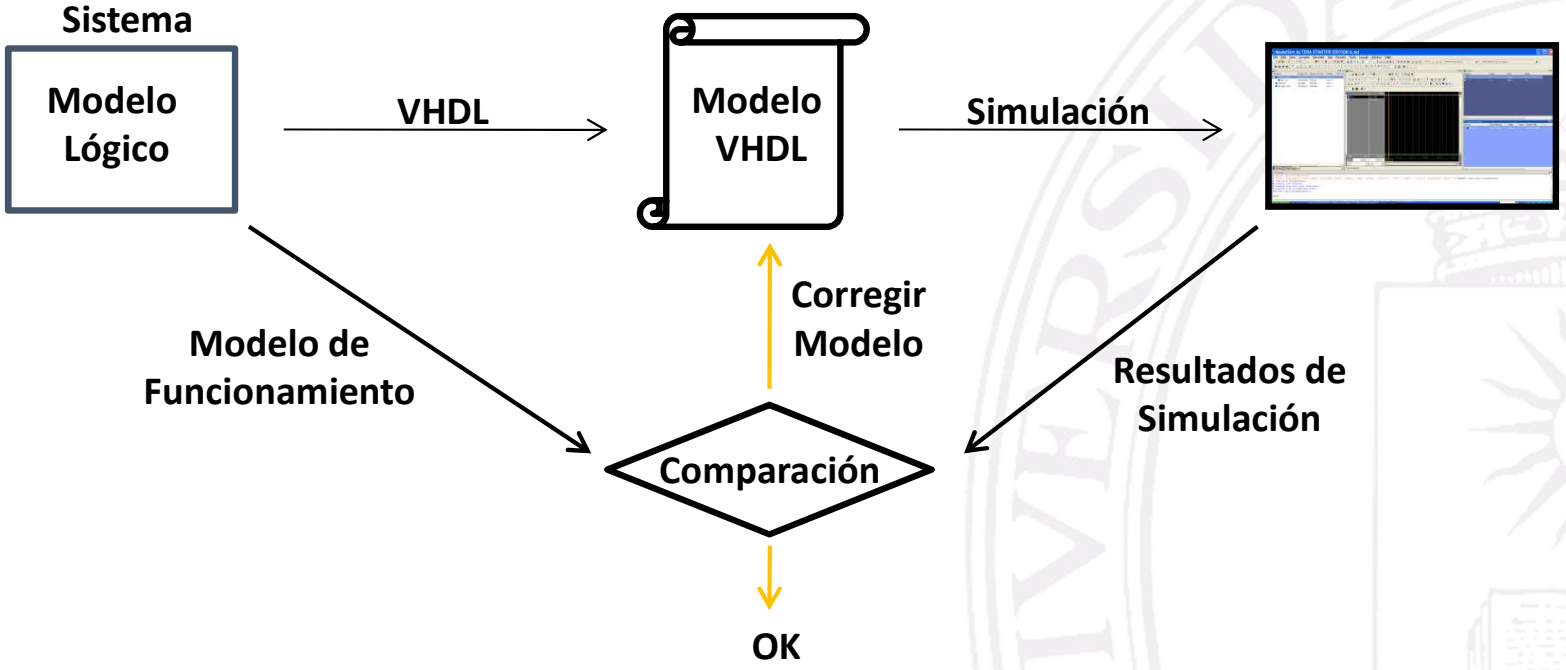
```
library ieee;
use ieee.std_logic_1164.all;

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
    if  E = "00" then
      S <= "01";

    elsif E = "01" then
      S <= "10";

    elsif E = "10" then
      S <= "11";

    else
      S <= "00";
    end if;
  end process;
end RTL;
```

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

Principios de modelado con VHDL

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

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

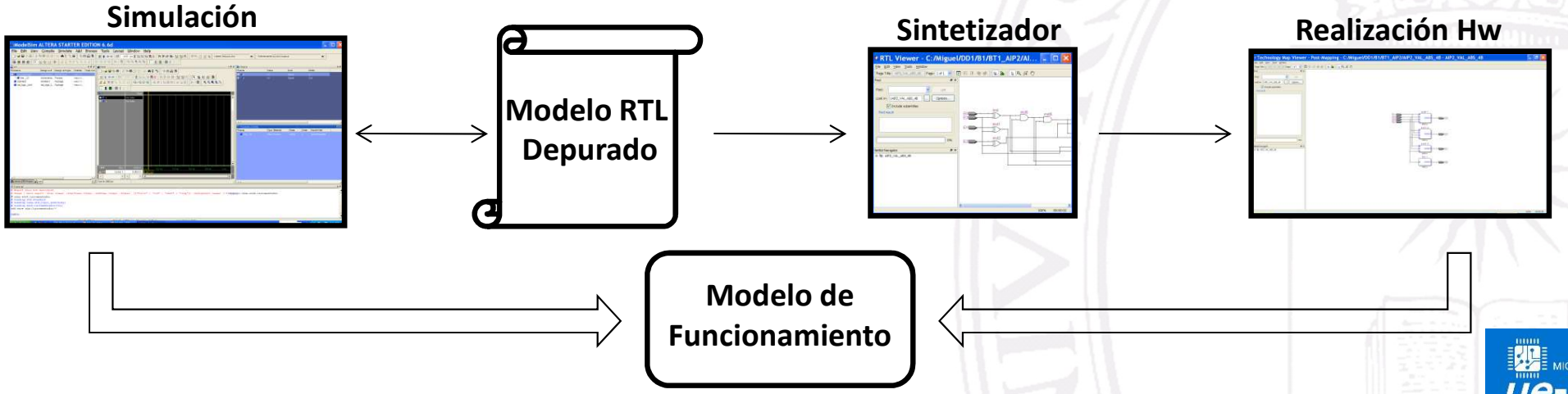


Principios de modelado con VHDL

• La principal ventaja del uso de modelos HDL es que pueden ser sintetizados automáticamente.

• Para que un modelo VHDL pueda ser sintetizado deben seguirse ciertas reglas en la codificación del modelo lógico. A los modelos que cumplen estas reglas se los denomina modelos RTL.

• Una herramienta de síntesis es capaz de generar, a partir de modelos RTL, circuitos que funcionan igual que los modelos en una simulación.



Modelado RTL con VHDL

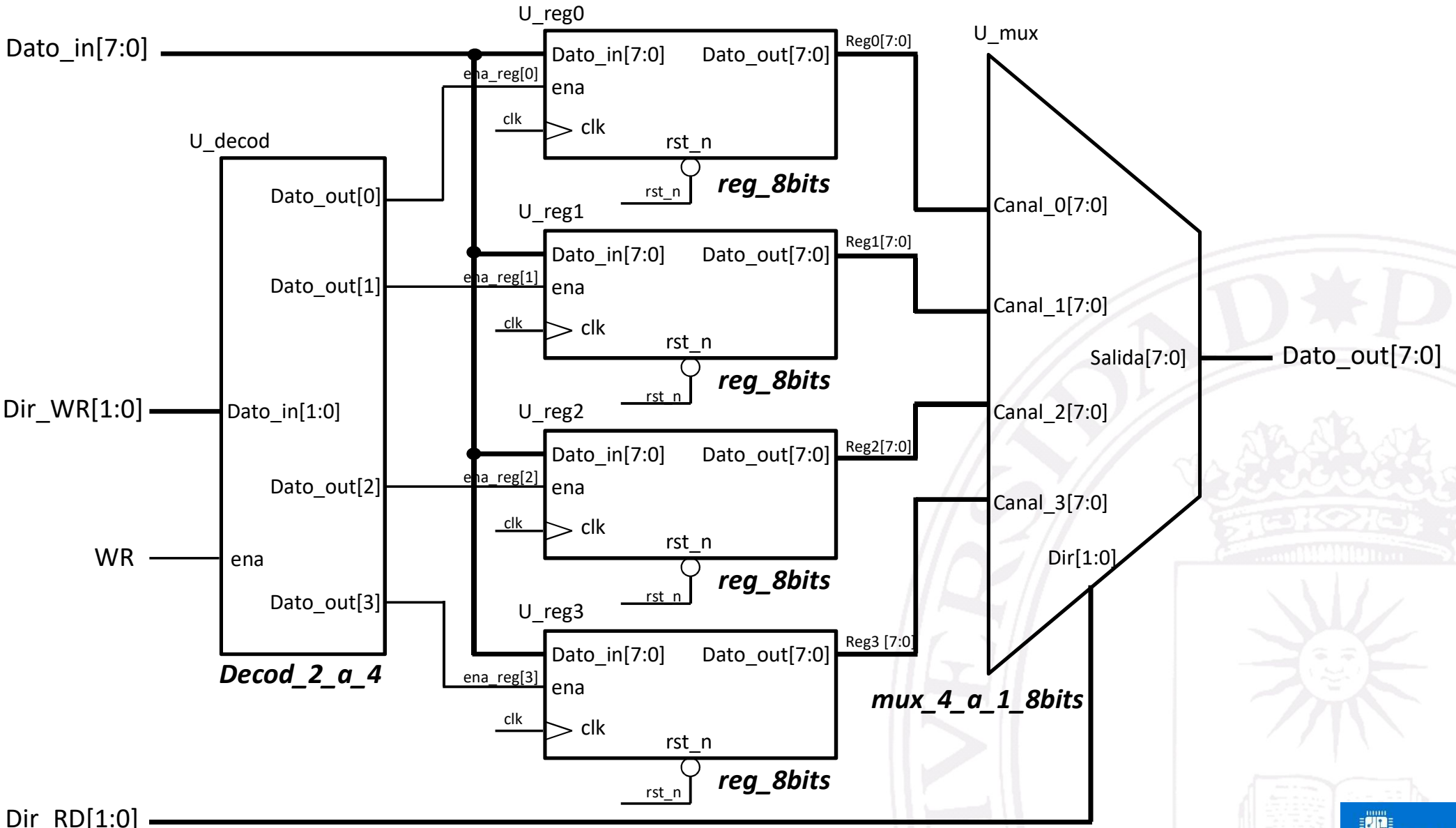
• En VHDL la interfaz de los sistemas digitales y su funcionamiento se describen separadamente, en dos unidades de código distintas: la interfaz en una **Declaración de Entidad** y el funcionamiento en un **Cuerpo de Arquitectura**.

```
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;
        DatoSel: BUFFER std_logic_vector(7 downto 0));
END ENTITY;

ARCHITECTURE rtl OF selector8bits IS
BEGIN
      PROCESS(DatoA, DatoB, Sel)
      BEGIN
         IF Sel = '0' THEN
           DatoSel <= Datoa;

        ELSE
          DatoSel <= DatoB;
        END IF;
      END PROCESS;
END rtl;
```

Modelado de la Interfaz

Modelado del Funcionamiento

Ejemplo: Banco de Registros



Modelado de Sistemas Combinacionales Simples (I)

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

entity decod_2_a_4 is
port(ena:      in     std_logic;
     Dato_in:  in     std_logic_vector(1 downto 0);
     Dato_out: buffer std_logic_vector(3 downto 0));
end entity;

architecture rtl of decod_2_a_4 is
begin
  process(ena, Dato_in)
  begin
    if ena = '1' then
      case Dato_in is
        when "00" =>
          Dato_out <= "0001";
        when "01" =>
          Dato_out <= "0010";
        when "10" =>
          Dato_out <= "0100";
        when "11" =>
          Dato_out <= "1000";
        when others =>
          Dato_out <= (others => 'X');
      end case;

    else
      Dato_out <= "0000";
    end if;
  end process;
end rtl;
```

CLÁUSULAS DE VISIBILIDAD Y DECLARACIÓN DE ENTIDAD

NOMBRE DEL MODELO Y NOMBRE, DIRECCIÓN Y TIPO DE DATOS DE LOS PUERTOS

DECODIFICADOR

CUERPO DE ARQUITECTURA CONTENIENDO UN PROCESO

PROCESO SENSIBLE A TODAS LAS ENTRADAS DEL SISTEMA COMBINACIONAL

ASIGNA VALOR A TODOS LOS BITS DE SALIDA PARA TODAS LAS COMBINACIONES DE ENTRADA

Modelado de Sistemas Combinacionales Simples (II)

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

entity mux_4_a_1_8bits is
port(Canal_0: in     std_logic_vector(7 downto 0);
     Canal_1: in     std_logic_vector(7 downto 0);
     Canal_2: in     std_logic_vector(7 downto 0);
     Canal_3: in     std_logic_vector(7 downto 0);
     Dir:     in     std_logic_vector(1 downto 0);
     Salida:  buffer std_logic_vector(7 downto 0));
end entity;

architecture rtl of mux_4_a_1_8bits is
begin

  Salida <= Canal_0 when Dir = 0 else
            Canal_1 when Dir = 1 else
            Canal_2 when Dir = 2 else
            Canal_3 when Dir = 3 else
            (others => 'X');
end rtl;
```

CLÁUSULAS DE VISIBILIDAD Y DECLARACIÓN DE ENTIDAD

NOMBRE DEL MODELO Y NOMBRE, DIRECCIÓN Y TIPO DE DATOS DE LOS PUERTOS

MULTIPLEXOR

CUERPO DE ARQUITECTURA CONTENIENDO UNA SENTENCIA CONCURRENTE DE ASIGNACIÓN CONDICIONAL

Modelado de Sistemas Secuenciales Simples

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

entity reg_8bits is
port(clk:      in     std_logic;
     rst_n:    in     std_logic;
     ena:      in     std_logic;
     Dato_in:  in     std_logic_vector(7 downto 0);
     Dato_out: buffer std_logic_vector(7 downto 0));
end entity;

architecture rtl of reg_8bits is
begin

  process(clk, rst_n)
  begin
    if rst_n = '0' then
      Dato_out <= (others => '0');

    elsif clk'event and clk = '1' then
      if ena = '1' then
        Dato_out <= Dato_in;
      end if;
    end if;
  end process;
end rtl;
```

CLÁUSULAS DE VISIBILIDAD Y DECLARACIÓN DE ENTIDAD

NOMBRE DEL MODELO Y NOMBRE, DIRECCIÓN Y TIPO DE DATOS DE LOS PUERTOS

REGISTRO

CUERPO DE ARQUITECTURA CONTENIENDO UN PROCESO

PROCESO SENSIBLE A LA ENTRADA DE RELOJ E INICIALIZACIÓN ASÍNCRONA

EVALÚA PRIORITARIAMENTE LA ACTIVIDAD DE LA ENTRADA ASÍNCRONA Y, DESPUÉS, LOS ESTADOS DE LAS ENTRADAS SÍNCRONAS EN LA OCURRENCIA DE UN FLANCO ACTIVO DE RELOJ

Modelado de Sistemas Complejos (I)

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

entity reg_file_4_8bits is
port(clk:       in     std_logic;
     rst_n:     in     std_logic;
     WR:        in     std_logic;
     Dir_WR:    in     std_logic_vector(1 downto 0);
     Dir_RD:    in     std_logic_vector(1 downto 0);
     Dato_in:   in     std_logic_vector(7 downto 0);
     Dato_out:  buffer std_logic_vector(7 downto 0));

end entity;

architecture estructural of reg_file_4_8bits is
  signal ena_reg:  std_logic_vector(3 downto 0);
  signal reg0:     std_logic_vector(7 downto 0);
  signal reg1:     std_logic_vector(7 downto 0);
  signal reg2:     std_logic_vector(7 downto 0);
  signal reg3:     std_logic_vector(7 downto 0);
begin

U_decod: entity Work.decod_2_a_4(rtl)
         port map(ena => WR,
                  Dato_in => Dir_WR,
                  Dato_out => ena_reg);

U_reg0: entity Work.reg_8bits(rtl)
        port map(clk => clk,
                 rst_n => rst_n,
                 ena => ena_reg(0),
                 Dato_in => Dato_in,
                 Dato_out => reg0);

 ------------------------
 ------------------------  etc.
```

CLÁUSULAS DE VISIBILIDAD Y DECLARACIÓN DE ENTIDAD

NOMBRE DEL MODELO Y NOMBRE, DIRECCIÓN Y TIPO DE DATOS DE LOS PUERTOS

BANCO DE REGISTROS
(INCOMPLETO)

CUERPO DE ARQUITECTURA CONTENIENDO SENTENCIAS DE EMPLAZAMIENTO DE MODELOS Y DECLARACIÓN DE SEÑALES

LAS SENTENCIA EMPLAZAN MODELOS DE SUBSISTEMAS (MÓDULOS) QUE INTERACTÚAN MEDIANTE LAS SEÑALES A LAS QUE ESTÁN CONECTADOS

Modelado de Sistemas Complejos (II)

BANCO DE REGISTROS (COMPLETO)

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

entity reg_file_4_8bits is
port(clk:       in     std_logic;
     rst_n:     in     std_logic;
     WR:        in     std_logic;
     Dir_WR:    in     std_logic_vector(1 downto 0);
     Dir_RD:    in     std_logic_vector(1 downto 0);
     Dato_in:   in     std_logic_vector(7 downto 0);
     Dato_out:  buffer std_logic_vector(7 downto 0));

end entity;

architecture estructural of reg_file_4_8bits is
  signal ena_reg:  std_logic_vector(3 downto 0);
  signal reg0:     std_logic_vector(7 downto 0);
  signal reg1:     std_logic_vector(7 downto 0);
  signal reg2:     std_logic_vector(7 downto 0);
  signal reg3:     std_logic_vector(7 downto 0);
begin

U_decod: entity Work.decod_2_a_4(rtl)
         port map(ena => WR,
                  Dato_in => Dir_WR,
                  Dato_out => ena_reg);

U_reg0: entity Work.reg_8bits(rtl)
        port map(clk => clk,
                 rst_n => rst_n,
                 ena => ena_reg(0),
                 Dato_in => Dato_in,
                 Dato_out => reg0);
```

```
U_reg1: entity Work.reg_8bits(rtl)
        port map(clk => clk,
                 rst_n => rst_n,
                 ena => ena_reg(1),
                 Dato_in => Dato_in,
                 Dato_out => reg1);

U_reg2: entity Work.reg_8bits(rtl)
        port map(clk => clk,
                 rst_n => rst_n,
                 ena => ena_reg(2),
                 Dato_in => Dato_in,
                 Dato_out => reg2);

U_reg3: entity Work.reg_8bits(rtl)
        port map(clk => clk,
                 rst_n => rst_n,
                 ena => ena_reg(3),
                 Dato_in => Dato_in,
                 Dato_out => reg3);

U_mux: entity Work.mux_4_a_1_8bits(rtl)
       port map(Canal_0 => reg0,
                Canal_1 => reg1,
                Canal_2 => reg2,
                Canal_3 => reg3,
                Dir => Dir_RD,
                Salida => Dato_out);

end estructural;
```

Modelado de Sistemas Complejos (III)

```
architecture rtl of reg_file_4_8bits is
  signal reg0:     std_logic_vector(7 downto 0);
  signal reg1:     std_logic_vector(7 downto 0);
  signal reg2:     std_logic_vector(7 downto 0);
  signal reg3:     std_logic_vector(7 downto 0);

begin
process(clk, rst_n)
begin
  if rst_n = '0' then
    reg0 <= (others => '0');
    reg1 <= (others => '0');
    reg2 <= (others => '0');
    reg3 <= (others => '0');

  elsif clk'event and clk = '1' then
    if WR = '1' then
      case Dir_WR is
        when "00" =>
          reg0 <= Dato_in;
        when "01" =>
          reg1 <= Dato_in;
        when "10" =>
          reg2 <= Dato_in;
        when "11" =>
          reg3 <= Dato_in;
        when others =>
          null;
      end case;
    end if;
  end if;
end process;

Dato_out <= reg0 when Dir_RD = 0 else
            reg1 when Dir_RD = 1 else
            reg2 when Dir_RD = 2 else
            reg3 when Dir_RD = 3 else
            (others => 'X');
end rtl;
```

BANCO DE REGISTROS
(ESTRUCTURA CON PROCESOS)

CUERPO DE ARQUITECTURA CONTENIENDO PROCESOS Y DECLARACIÓN DE SEÑALES

LOS PROCESOS INTERACTUAN POR MEDIO DE SEÑALES QUE EVALÚAN O A LAS QUE ASIGNAN VALOR

Test-Benches(I)

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_mux_4_a_1_8bits is
end entity;

architecture test of test_mux_4_a_1_8bits is
  signal Canal_0: std_logic_vector(7 downto 0);
  signal Canal_1: std_logic_vector(7 downto 0);
  signal Canal_2: std_logic_vector(7 downto 0);
  signal Canal_3: std_logic_vector(7 downto 0);
  signal Dir:     std_logic_vector(1 downto 0);
  signal Salida:  std_logic_vector(7 downto 0);

begin
dut: entity Work.mux_4_a_1_8bits(rtl)
     port map(Canal_0 => Canal_0,
              Canal_1 => Canal_1,
              Canal_2 => Canal_2,
              Canal_3 => Canal_3,
              Dir => Dir,
              Salida => Salida);
process
begin
  Canal_0 <= X"00";
  Canal_1 <= X"AA";
  Canal_2 <= X"55";
  Canal_3 <= X"FF";
  Dir <= "00";
  for i in 0 to 3 loop
    wait for 100 ns;
     Canal_0 <= not Canal_0;
     Canal_1 <= not Canal_1;
     Canal_2 <= not Canal_2;
     Canal_3 <= not Canal_3;
    wait for 100 ns;
     Dir <= Dir + 1;
  end loop;
end process;
end test;
```

DECLARACIÓN DE ENTIDAD SIN PUERTOS

DECLARACIÓN DE ESTÍMULOS

TEST DEL MULTIPLEXOR

EMPLAZAMIENTO DEL MODELO

PROCESO QUE MANEJA LOS ESTÍMULOS DE SIMULACIÓN

UTILIZA SENTENCIAS “WAIT FOR” PARA TEMPORIZAR LAS PRUEBAS DEL TEST

Test-Benches(II)

```
process
begin
  clk <= '0';
  wait for T_clk/2;
  clk <= '1';
  wait for T_clk/2;
end process;

process
begin
  rst_n <= '1';
  wait until clk'event and clk = '1';
  rst_n <= '0';
  wait until clk'event and clk = '1';
  WR <= '0';
  Dir_WR <= "00";
  Dir_RD <= "00";
  Dato_in <= X"55";
  wait until clk'event and clk = '1';
  rst_n <= '1';

  for i in 0 to 3 loop
    WR <= '1';
    wait until clk'event and clk = '1';
    WR <= '0';
    wait until clk'event and clk = '1';
    for j in 0 to 3 loop
      Dir_RD <= Dir_RD + 1;
      Dato_in <= Dato_in + X"11";
      wait until clk'event and clk = '1';
    end loop;
    Dir_WR <= Dir_WR + 1;
  end loop;
  wait;
end process;

end test;
```

TEST DEL BANCO DE REGISTROS
(FRAGMENTO)

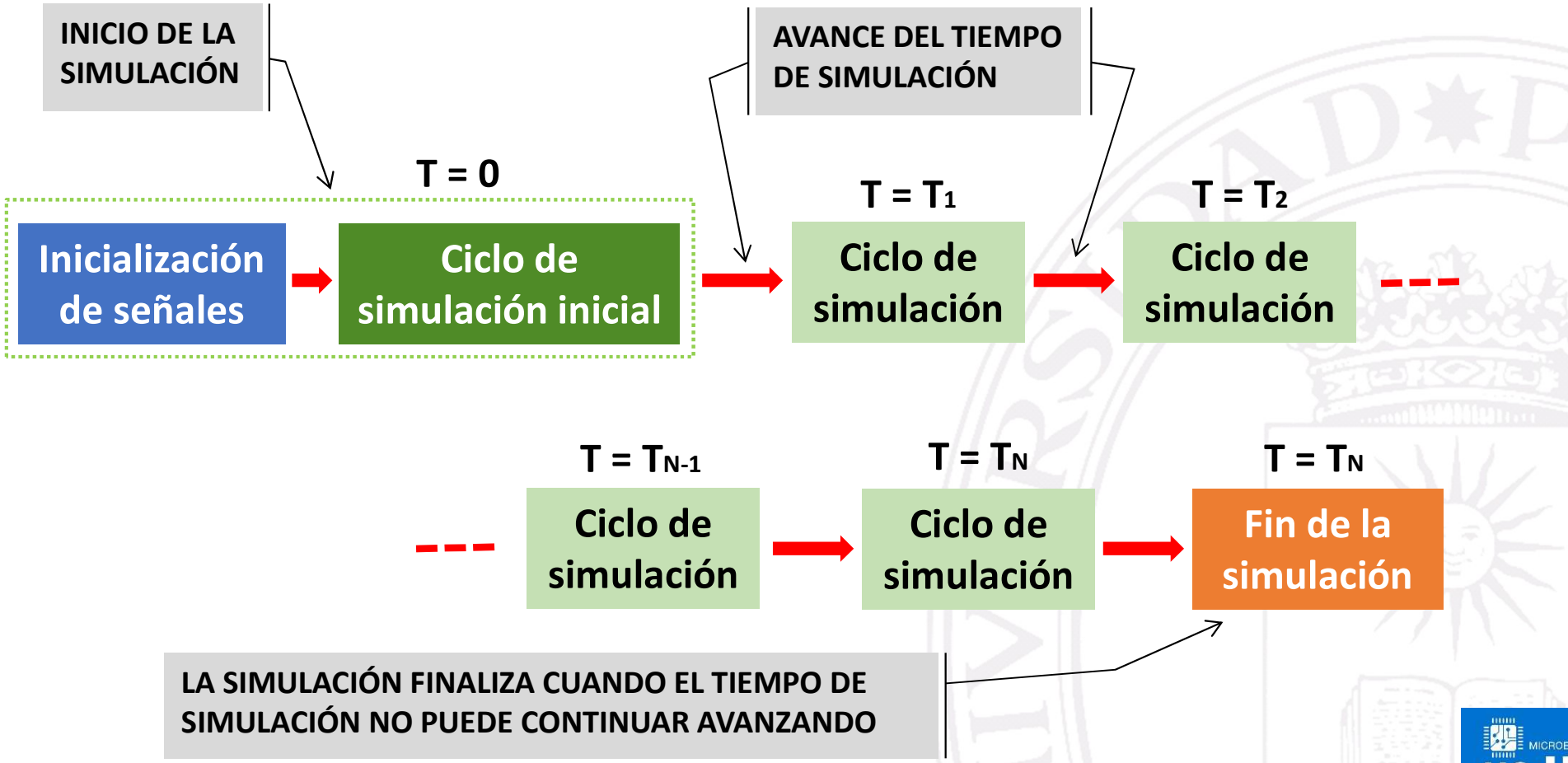
PROCESO PARA LA GENERACIÓN DEL RELOJ

PROCESO QUE MANEJA EL RESTO DE ESTÍMULOS DE SIMULACIÓN

UTILIZA SENTENCIAS "WAIT UNTIL" PARA TEMPORIZAR LAS PRUEBAS DEL TEST

Simulaciones VHDL (I)

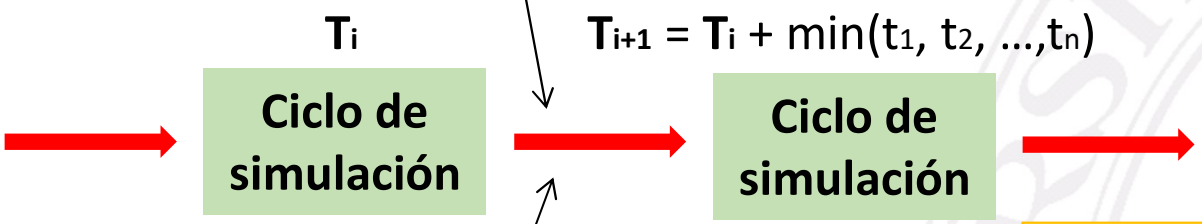
- Un simulador VHDL completa una simulación repitiendo la ejecución de **Ciclos de Simulación** en una secuencia finita de **instantes de Tiempo de Simulación (T)** que comienza en el instante **T = 0**



Simulaciones VHDL (II)

• Cuando termina la ejecución de un ciclo de simulación, el simulador avanza el tiempo sumando al actual (T), el menor tiempo que ha de transcurrir para que se reanude la ejecución de un proceso detenido por una sentencia WAIT FOR.

AL TERMINAR EL CICLO DE SIMULACIÓN EN T_i, HABRÁ *N* PROCESOS SUSPENDIDOS POR SENTENCIAS WAIT FOR **Y TIEMPOS t_1, t_2, …, t_n; EL SIMULADOR AVANZA EL TIEMPO DESDE T_i A T_i + min(t_1, t_2, …, t_n)**



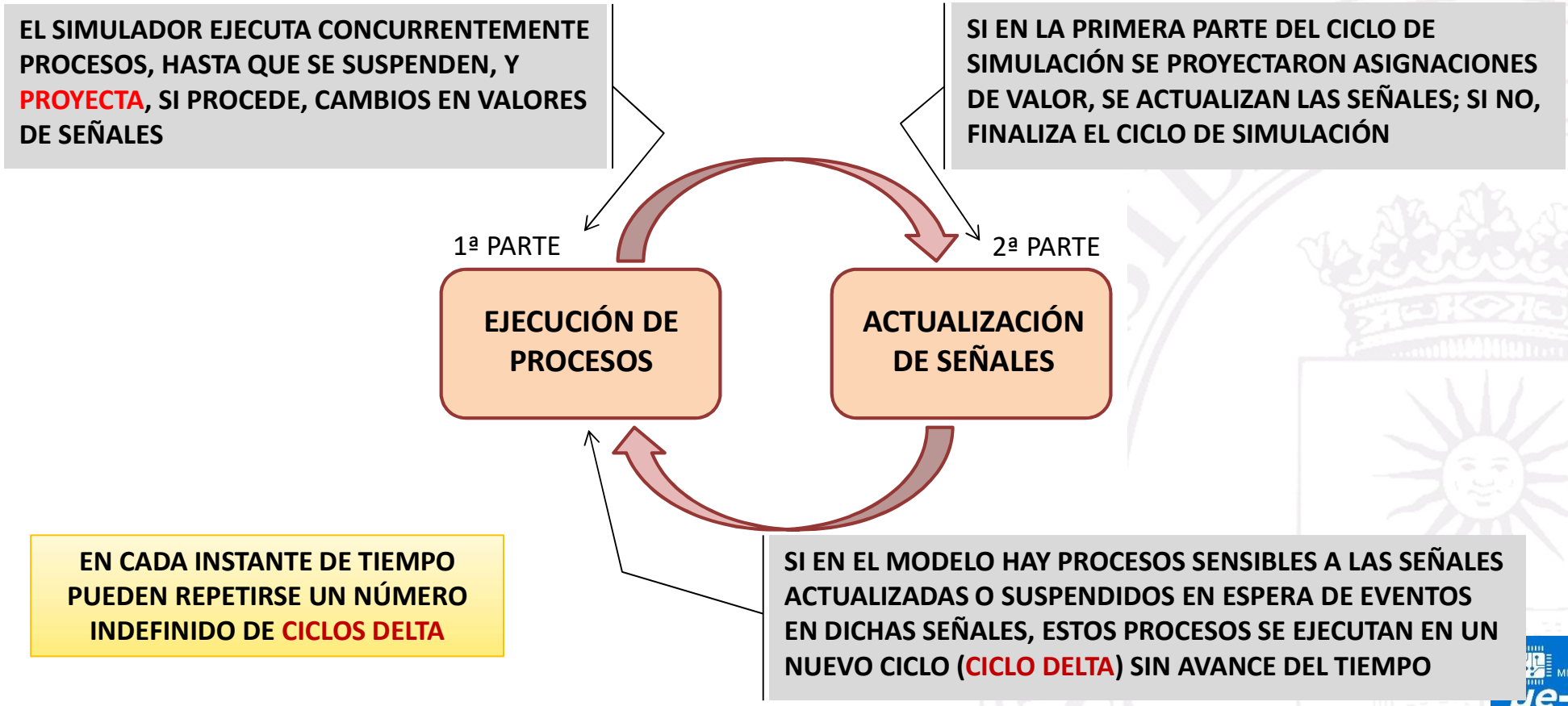
SI NO QUEDAN PROCESOS SUSPENDIDOS POR SENTENCIAS WAIT FOR**, EL SIMULADOR NO PUEDE AVANZAR EL TIEMPO Y FINALIZA LA SIMULACIÓN**

ESTE ES EL MECANISMO DE AVANCE DEL TIEMPO PARA MODELOS Y TEST-BENCHES EN LOS QUE NO SE EMPLEEN RETARDOS EN LA ASIGNACIÓN DE VALORES A SEÑALES

Ciclo de simulación VHDL

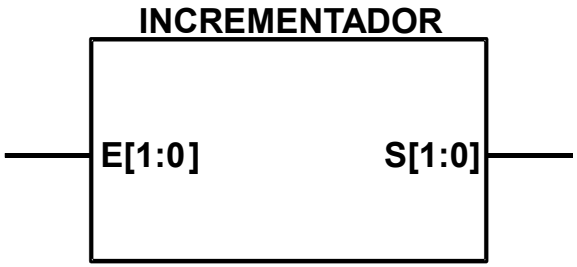
- Tiene dos partes: en la 1ª se **ejecutan procesos** y en la 2ª se **actualizan señales**.

- En la primera parte del ciclo de simulación inicial (T = 0) se ejecutan TODOS los procesos de la jerarquía test-bench; en el resto de instantes, únicamente los que esperaban reanudación



Ejemplo

- Modelo bajo prueba: Incrementador de 2 bits



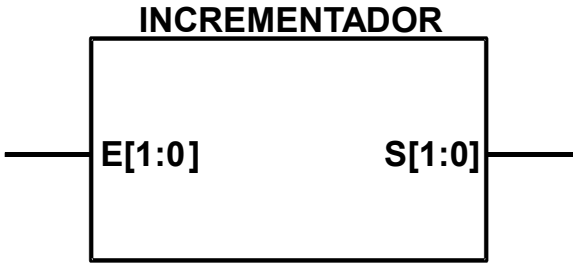
E1	E0	S1	S0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

```
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;
```

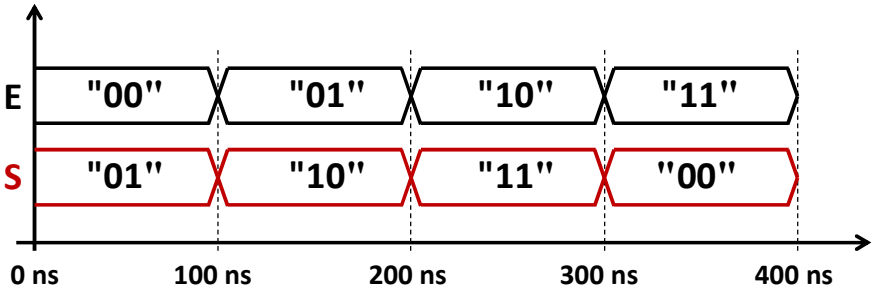
Ejemplo

- Test-Bench del incrementador

MODELO LÓGICO



ESTÍMULOS

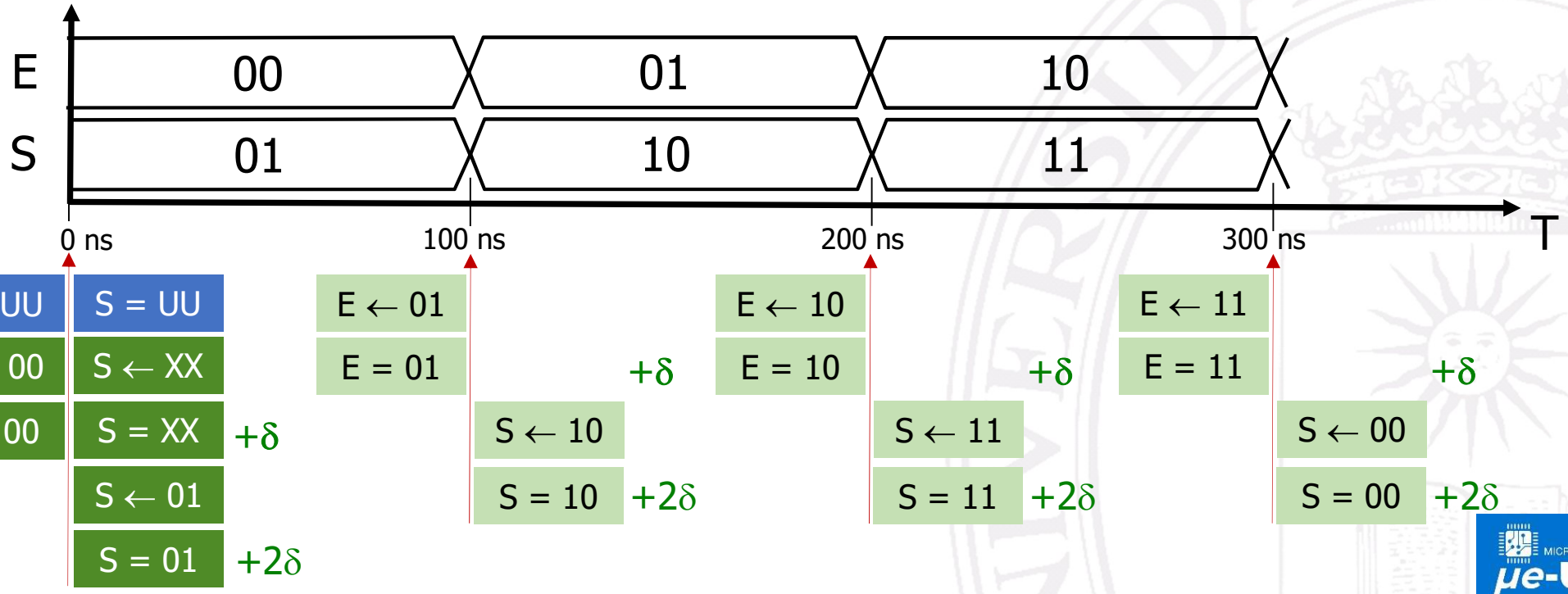



```
library ieee;
use ieee.std_logic_1164.all;

entity test_incrementador is
end entity;

architecture test of test_incrementador is
  -- Estímulos
  signal E: std_logic_vector(1 downto 0);
  signal s: std_logic_vector(1 downto 0);

begin

  -- Emplazamiento y conexión del modelo
  DUT: entity Work.INCREMENTADOR(rtl)
       port map(E => E,
                S => S);

  --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;
end test;
```

Ejemplo

```
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;
```

```
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;
```



- Más información: https://blogs.upm.es/ue-upm/

- Contacto: comunidad.microelectronica@upm.es