Universidad de Alcalá

Departamento de Electrónica

# Modelado de Sistemas Computacionales

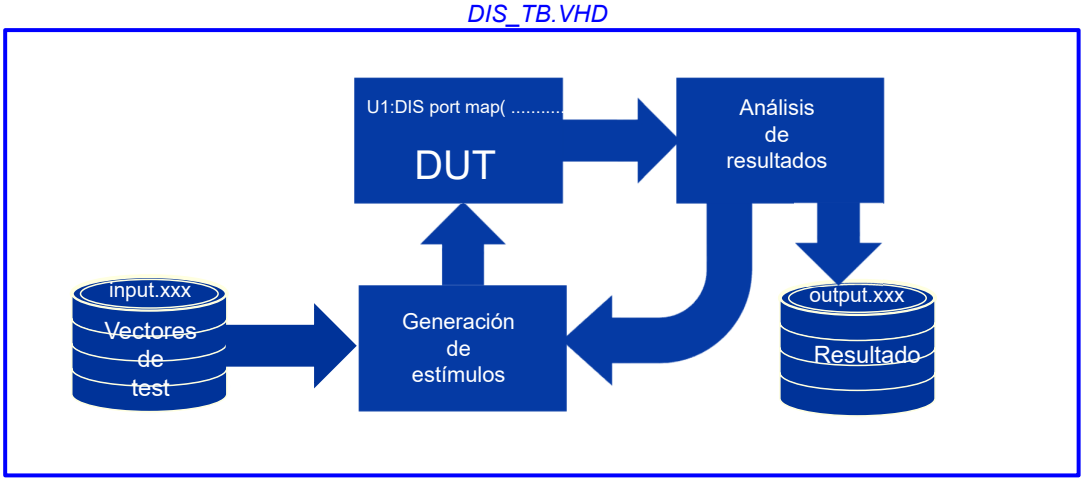
## Grado en Ingeniería de Computadores

# Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).



## Bancos de pruebas

- Un testbench o banco de prueba es un módulo VHDL en el que se instancia el componente a verificar y contiene código para:
  - Asignar los estímulos sobre el modelo a verificar.
  - Observar los resultados obtenidos e informar al usuario.




¡Los testbench no se sintetizan, son para simulación!

¡Se puede utilizar cualquier sentencia VHDL sin restricciones!

# Bancos de pruebas

El testbench es una entidad que no tienen puertos.

Se elige el estándar (87 ò 93) para el manejo de componentes

Si el estándar elegido es el del 87 se declara el componente. Si es el del 93 no

Se declaran, al menos, tantas señales como puertos tenga la entidad

Se instancia el componente según el estándar elegido.

Se asignan las señales antes declaradas a los puertos.

Se generan los estímulos a aplicar a las entradas.

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

entity decodificador_tb is
end decodificador_tb;

architecture sim of decodificador_tb is

  component decodificador
    port (
      A : in  std_logic_vector(2 downto 0);
      S : out std_logic_vector(7 downto 0));
  end component;

  signal A_i : std_logic_vector(2 downto 0);
  signal S_i : std_logic_vector(7 downto 0);

begin  -- sim

  DUT: decodificador
    port map (
        A => A_i,
        S => S_i);

process
 begin  -- process
  for i in 0 to 7 loop
    A_i <= std_logic_vector(to_unsigned(i, 3));
    wait for 10 ns;
  end loop;  -- i
 report "FIN SIMULACION" severity failure;
end process;

end sim;
```

Sólo para VHDL 87

Si hay genéricos se declaran, al menos, tantas constantes como genéricos tenga la entidad

⚠ No hay restricción en las sentencias a utilizar.

Un testbench no se sintetiza

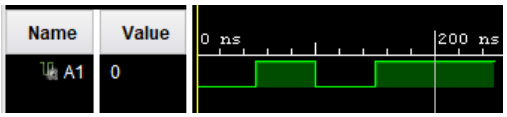


UAH
Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).

# Bancos de pruebas

- Generación de estímulos:
  - Asigna concurrente a señal con clausula ***after***.

```
A1<= '0','1' after 50 ns,'0' after 100 ns, '1' after 150 ns;
```




```
type time is range -2147483647 to 2147483647
        units
          fs;
                  ps = 1000 fs;
                  ns = 1000 ps;
                  us = 1000 ns;
                  ms = 1000 us;
                  sec = 1000 ms;
                  min = 60 sec;
                  hr = 60 min;
end units;
```

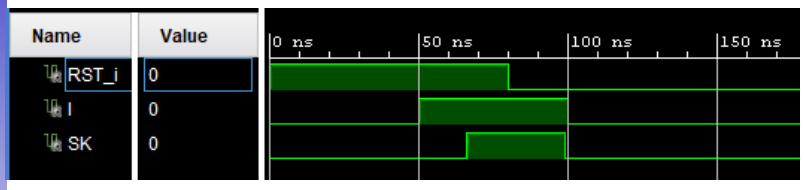
Tipo físico. No sintetizable

- Son poco flexibles.
- Difícil modelar la interrelación temporal entre señales.
- Por estos motivos sólo se suelen emplear en casos sencillos:
  - Señales periódicas ( p.e. reloj de un sistema secuencial).
  - Señales con pocos cambios (p.e. Señal de reset).

¡La cláusula ***after*** es ignorada por la herramientas de síntesis!

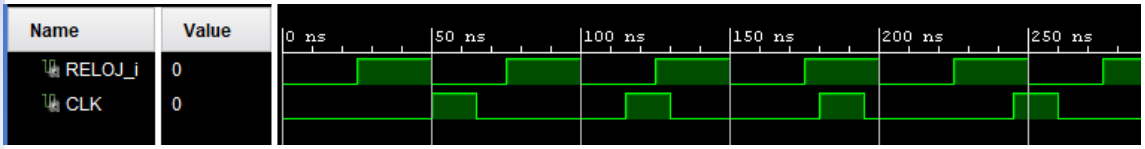
# Bancos de pruebas

## ❑ Asigna concurrente a señal con clausula *after*.




```
signal RST_i, I : std_logic;
signal SK       : std_logic:='0';
RST_i<='1', '0' after 80 ns;
I<='0', '1' after 50 ns, '0' after 100 ns;
SK<='1' after 66 ns, '0' after 99 ns;
```

Los tiempos deben ser consecutivos.




```
signal RELOJ_i,CLK : std_logic:='0';

RELOJ_i<= not RELOJ_i after 25 ns;

CLK<= '0'  after 15 ns when CLK='1' else '1' after 50 ns;
```

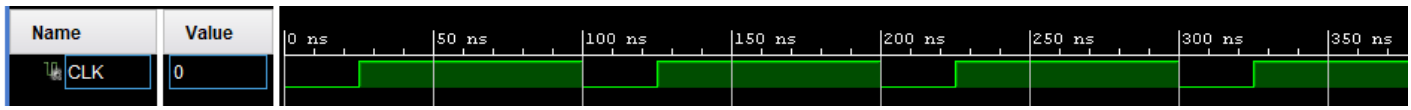


UAH
Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).

# Bancos de pruebas

## ❑ Generación de estímulos:

### ❑ Procesos con sentencias ***wait***.




```
signal CLK : std_logic;
```

```
process
begin  -- process
  clk<='0';
  wait for 25 ns;
  clk<='1';
  wait for 75 ns;
end process;
```

## Procesos con sentencias *wait*..

- Se dispone de un o o más procesos con el que se controla la evolución conjunta de las señales de forma secuencial.
  - Gran flexibilidad: empleo de bucles, subprogramas, etc.
  - Sentencias condicionales: Introducción de estímulos dependiendo de la evolución del circuito analizado.
  - Bucles: Pruebas exhaustivas.
- En su formato más sencillo se describe la secuencia de valores que toman las señales mediante pares:
  - Aplicación de estímulos: señal <= valor;
  - Espera hasta la siguiente conmutación: ***wait for*** <tiempo>;

```
process
  begin
...
      DATO_RX_OK_i <= '0';
      wait for CNT1;
...

end process;
```



UAH
Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).
Bancos de pruebas

## Procesos con sentencias *wait*..

- Son muy flexibles.
- Permiten generar estímulos muy complejos.
- Utilizando bucles se pueden hacer pruebas exhaustivas.
- Permiten, de una forma sencilla, variar los vectores de tests en función del resultado

```
process
begin
  -- CTRL a 1 mientras que el
  -- dispositivo no esté listo.
   while RDY /= '0' loop
      CTRL <= '1';
      wait until CLK='1';
   end loop;
   CTRL <= '0';
   wait for 10 ns;
   ...
end process;
```

```
process
 begin
  for i in 0 to 7 loop
    A_i <= std_logic_vector(to_unsigned(i, 3));
    wait for 10 ns;
  end loop;  -- i
 report "FIN SIMULACION" severity failure;
end process;
```

# Sentencia wait

- **Sentencia wait..**
  - Permite determinar el punto donde debe suspenderse la ejecución de un proceso y en qué condiciones debe reactivarse.
  - En un proceso pueden existir varias sentencias ***wait***.
  - Un proceso con sentencia ***wait*** no puede tener lista de sensibilidad

```
[etiqueta:] wait;
[etiqueta:] wait on señal_1 {señal_2, ...} ;
[etiqueta:] wait for expresión_temporal;
[etiqueta:] wait until condición_booleana;
```

Sólo esta es sintetizable



UAH
Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).

# Sentencia wait

- Modo ***wait on*** + lista de señales.

```
process
  begin
    cnt <= cnt +1;
    wait on clk;
end process;
```

El proceso se suspende hasta que se produce un evento en alguna de las señales de la lista.

- Modo ***wait until*** + expresión_booleana.

```
process
  begin
    cnt <= cnt +1;
    wait until clk='1';
end process;
```

El proceso se suspende hasta que se verifique la condición: pase de falso a cierto.

- Modo ***wait for***+ expresión_temporal.

```
process
  begin
    S2 <= '0';
    wait for 15 ns;
    S2 <= '1';
    wait for 25 ns;
 end process;
```

El proceso se suspende hasta que se verifique la condición: pase de falso a cierto.

- Modo ***wait***.

El proceso se suspende para siempre.

# Bancos de pruebas

```
entity BCD_7SEG is
  port (
    A       : in  std_logic_vector(3 downto 0);
    Display : out std_logic_vector(6 downto 0));
end BCD_7SEG;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity BCD_7SEG_tb is
end BCD_7SEG_tb;

architecture sim of BCD_7SEG_tb is
  signal A_i       : std_logic_vector(3 downto 0);
  signal Display_i : std_logic_vector(6 downto 0);
begin  -- sim

  DUT:entity work.BCD_7SEG
    port map (
        A       => A_i,
        Display => Display_i);

  process
  begin  -- process
    for j in 0 to 15 loop
      A_i<=std_logic_vector(to_unsigned(j,4));
      wait for 10 ns;
    end loop;  -- j
    assert false report "fin simulacion" severity FAILURE;
  end process;
end sim;
```

1. Los procesos son bucles infinitos: al llegar al final (***end process***) vuelven al principio (***begin***)

2. El proceso se suspender con una sentencia ***wait***

3. La sentencia ***wait*** fija las condiciones para despertar al proceso

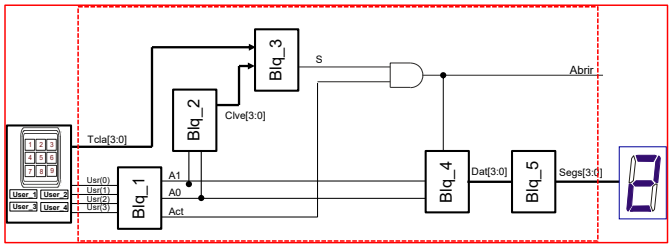


UAH
Tema 4: Simulación en VHDL, bancos de prueba (test-benchs).

# Ejemplo 1

Crear el código VHDL que modele un sistema digital que realiza el control de apertura de una puerta mediante dos datos: usuario y clave seleccionados con sendos teclados.

```
library ieee;
use ieee.std_logic_1164.all;
entity ejemplo_CC_tb is
end ejemplo_CC_tb;

architecture sim of ejemplo_CC_tb is
  signal Tcla_i  : std_logic_vector(3 downto 0);
  signal Usr_i   : std_logic_vector(3 downto 0);
  signal Segs_i  : std_logic_vector(6 downto 0);
  signal Abrir_i : std_logic;

begin  -- sim
  DUT : entity work.ejemplo_CC
    port map (
      Tcla  => Tcla_i,
      Usr   => Usr_i,
      Segs  => Segs_i,
      Abrir => Abrir_i);
  process
  begin  -- process
    Tcla_i <= x"7";
    Usr_i  <= "1111";
    wait for 50 ns;

    Tcla_i <= x"7";
    Usr_i  <= "1011";
    wait for 50 ns;

    Tcla_i <= x"7";
    Usr_i  <= "1110";
    wait for 50 ns;

    Tcla_i <= x"4";
    Usr_i  <= "1101";
    wait for 50 ns;
```



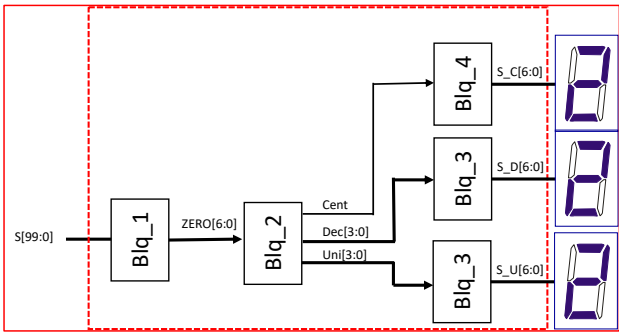

```
entity ejemplo_CC is
  port( Tcla            : in  std_logic_vector(3 downto 0);
        Usr             : in  std_logic_vector(3 downto 0);
        Segs            : out std_logic_vector(6 downto 0);
        Abrir           : out std_logic);
end;
```

```
    Tcla_i <= x"6";
    Usr_i  <= "1011";
    wait for 50 ns;

    Tcla_i <= x"5";
    Usr_i  <= "0111";
    wait for 50 ns;

    report "fin" severity failure;
  end process;

end sim;
```

# Ejemplo 2

Se pretende realizar un sistema que permite determinar el número de plazas libres de un aparcamiento y visualizarlas en 3 displays, siendo el número de plazas es 100.

```
ibrary ieee;
use ieee.std_logic_1164.all;
entity ejemplo_CC2_tb is
end ejemplo_CC2_tb;
architecture sim of ejemplo_CC2_tb is
  signal S_i   : std_logic_vector(99 downto 0);
  signal S_U_i : std_logic_vector(6 downto 0);
  signal S_D_i : std_logic_vector(6 downto 0);
  signal S_C_i : std_logic_vector(6 downto 0);
  signal ceros :integer:=0;
begin  -- sim

  DUT:entity work.ejemplo_CC2
    port map (
        S   => S_i,
        S_U => S_U_i,
        S_D => S_D_i,
        S_C => S_C_i);

  process (ceros)
  begin
    S_i<=(others=>'1');
    for j in 0 to ceros-1 loop
      S_i(j)<='0';
    end loop;  -- j
        -- wait for 1 ns;
  end process;
```




```
process
  begin  -- process
    ceros<=5;
    wait for 49 ns;
    ceros<=25;
    wait for 49 ns;
    ceros<=46;
    wait for 49 ns;
    ceros<=75;
    wait for 49 ns;
    ceros<=85;
    wait for 49 ns;
    ceros<=100;
    wait for 49 ns;
  report "fin" severity FAILURE;
  end process;
end sim;
```





# Bancos de pruebas

- Existes mecanismos para informar sobre el estado la simulación, e incluso detenerla.
  - Sentencia ***assert***: Comprueba condiciones y generar informes en función de su cumplimiento.
    - Se utiliza tanto en estructuras concurrentes como secuenciales.
    - Si la condición booleana no se cumple se envía la expresión del informe y dependiendo del nivel de severidad se puede interrumpir la simulación.
    - Sí no se usa ***report*** el mensaje es “Assertion violation”.

```
[etiqueta:] assert condición_booleana report[expresión informe] [severity_level];
```

```
type severity_level is (note, warning, error, failure);
```

- VHDL-93 ofrece una variación de ***assert*** sin condición,
  - Por defecto el valor de severidad es ***note***.

```
[etiqueta:] report[expresión informe] [severity_level];
```