Electrónica Digital I

Código 2547510
Departamento de Ingeniería Electrónica y Telecomunicaciones
Facultad de ingeniería
Semestre 2017-1
Testbenchs I

Test Benchs

Un " ***testsbench"*** es un programa o modelo escrito en HDL con e propósito de verificar el correcto funcionamiento, mediante simulación, del sistema digital que está siendo diseñado.

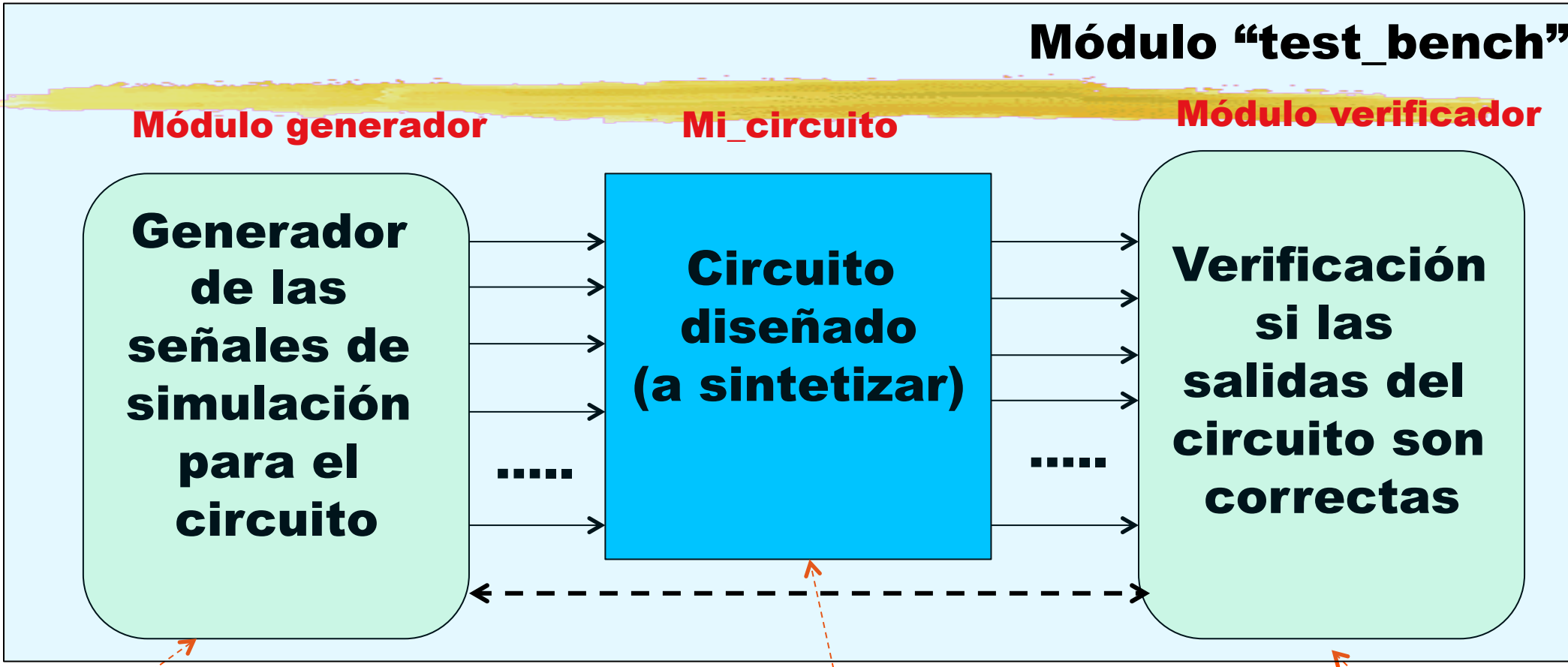
El lenguaje VHDL tiene muchos recursos (generalmente no sintetizables) para el modelado (simulación) como:

.

- Lectura, almacenamiento, formateo de datos.
- Comparación.
- Escritura de datos de simulación (estímulos de entrada y resultados de salida)

Estos recursos facilitan el diseño de los "test_bench".

Test Bench (estructura general..)



uede definirse un omponente que genere as señales para verificar el ircuito, o se pueden sar instruccione del VHDL ara generar las señales entro de la arquitectura

Circuito diseñado que se requiere verificar declarado como componente,

Componente encargado de verificar si las salidas son correctas. La verificación también se Puede hacer dentro de La arquitectura con instrucciones del VHDL

Componentes de un test bench

- Una entidad vacía (sin puertos) la cual define el "*test bench*"
- Señales internas para conectarlas a las entradas del circuito bajo test (para inyectar los estímulos de entrada) y monitorear las salidas.
- Declaración del componente bajo test.
- Instanciación del componente en la arquitectura.
- Generación de estímulos (puede hacerse en otro componente)
  - instrucciones para crear estímulos y bloques de procedimiento, monitoreo de las salidas y comparación.
- Instrucciones de *"autotesting"* para reportar valores, errores y advertencias ("*warnings"*).

Test Bench (ejemplo de la estructura general)

```
----------------------------------------------
-- Module Name: circuito_test_bench
----------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use STD.textio.all;
use IEEE.std_logic_textio.all;
library UNISIM;
use UNISIM.VComponents.all;

Entity circuito_test_bench Is
end circuito_test_bench;

Architecture behavior of circuito_test_bench Is
        Component mi_circuito
        port (
                puertos de mi_circuito…
                );
        End Component;

        Component generador
        port (
                puertos de generador…
                );
        End Component;

        Component verificador
        port (
                puertos del verificror…
                );
        End Component;
        Signal  ….;
        Signal  …
        ……
        procedures….
begin
        Cir1:   mi_circuito PORT MAP (
                        conexión de los puertos
                        …
                );

        cir2:   generador PORT MAP (
                        conexión de los puertos
                        ….
                );
         cir3:  verificador PORT MAP (
                        conexión de los puertos
                        ….
                );
end behavior;
```

Note que el test bench no tiene puertos de entrada-salida

Se declaran los tres Componetes previamente diseñados

Declaración de señales Para conectar los componentes

Se instancian los tres Componetes previamente diseñados

Procedimientos (procedures)

Los procedimientos proveen la capacidad de ejecutar "porciones de código" desde diferentes lugares del modelo. Son útiles tanto para modelar el circuito a sintetizar como para El diseño de los "test bench".

Los procedimientos podrían:

.

- Contener instrucciones para control de *timing*.
- Llamar o estar contenidos en otros procedimientos o funciones.

Procedimientos (procedures)

Un procedimiento se define (en la definición del módulo) como:

procedure *identifier* [input/output port declarations] **is**
    [variable declarations]
begin
procedure statements
.....
end *identifier*

Un procedimientos puede tener cero, uno o más argumentos. Los valores son pasados a y desde el procedimiento a través de los argumentos. Los argumento pueden ser de entrada y salida o mixtos.

Procedimientos (procedures)

Ejemplo:

```
procedure HAS_PROCEDURE (
      DIN : in STD_LOGIC_VECTOR(7 downto 0);
      DOUT : out STD_LOGIC_VECTOR(7 downto 0)) is
       variable k : integer := 0;
       variable count : integer := 0;
  begin
        for k in 0 to 7  loop
              count <= count + 1;
              DOUT(7 – count) <= DIN(7);
       end loop;
 end HAS_PROCEDURE;
```

Funciones

- En las funciones, a diferencia de los procedimientos, se retorna un valor simple cuando son ejecutadas

- Las funciones pueden implementar únicamente comportamiento combinacional. Se calcula un valor del retorno partir del valor presente de los argumentos de entrada.

- Cada línea de código se ejecuta secuencialmente. Las funciones son útiles si se cumple las siguientes condiciones:

  - No tienen atrasos, no hay construcciones con control de eventos, *timing* o atrasos.
  - Deben de tener al menos un argumento de entrada.
  - No tienen argumentos de salida o entrada-salida.
  - No hay asignaciones bloqueantes.

Funciones

Una función se define de la siguiente manera:

function ***identifier*** **[input port declarations]**
return type is [variable declarations]
begin

function statements

end ***identifier;***

Para llamar la función debe se en una instrucción de asignación, dentro d
un proceso, utilizando el nombre de la función. Ejemplo:

.....
Func_Out <= ***identifier*** **(input);**

Funciones

Ejemplo:

```
function HAS_FUNCTION (
    DIN : STD_LOGIC_VECTOR(7 downto 0);)
    return STD_LOGIC_VECTOR is
    variable k : integer := 0;
    variable count : integer := 0;
    variable reverse_bits : STD_LOGIC_VECTOR(7 downto 0) :=
"00000000";
   begin
      for k in 0 to 7 loop
           reverse_bits(7 – count) := DIN(count);
           count := count + 1;
       end loop;
     return reverse_bits;
    end HAS_TASK;
```

Modelando los atrasos

Existen dos técnicas para modelar los atrasos (el tiempo)

- **Atraso inercial**
- **Atraso por transporte.**

Atraso inercial:

- **En inglés *Inertial Delay Model* (es el modelo por defecto usado)**
- **Se utiliza para modelar el atraso de conmutación (tiempo de conmutación del circuito)**
- **Si un pulso de entrada es más pequeño que el tiempo de atraso la salida del circuito no cambia.**

Modelando los atrasos

Existen dos técnicas para modelar los atrasos (el tiempo)

- **Atraso inercial**
- **Atraso por transporte.**

Atraso inercial:

- **En inglés *Inertial Delay Model* (es el modelo por defecto usado)**
- **Se utiliza para modelar el atraso de conmutación (tiempo de conmutación del circuito)**
- **Si un pulso de entrada es más pequeño que el tiempo de atraso la salida del circuito no cambia.**

Modelando atraso inercial

```
process(a,b)
      begin
          z<= a and b; --  funcion and sin atraso

          -- modelo de atrasos con after, el mismo tphl y tplh
          z<= a and b AFTER 10 ns;

         --con diferente atraso

          If a = '1'  AND  b = '1' THEN
               z <= '1' AFTER 9 ns;     --  tplh
          THEN
               z <= '0' AFTER 10 ns;    -- tphl
          END IF;
END PROCESS;
```

Modelando atraso inercial

```
Modelado de atraso con transport
            If a = '1'  AND  b = '1' THEN
                 z <= '1' TRANSPORT 9 ns;     --  tplh
            THEN
                 z <= '0' TRANSPORT 10 ns;    -- tphl
            END IF;
```

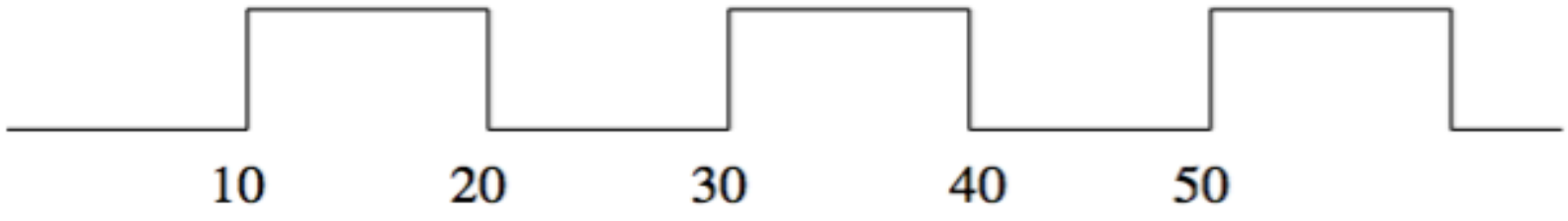
Asignación de múltiples atrasos a una señal

```
a <='0', '1' AFTER 8 ns, '0' AFTER 13 ns, '1' AFTER 50 ns;
```

Generación de señales de reloj

```
ARCHITECTURE testbench OF example IS
      SIGNAL clk : std_logic := '0';     -- initialize signals

BEGIN
      clk <= NOT clk AFTER 10 ns;
      -- creates periodic waveform on signal clk
      -- with a period of 20ns
```



Test Bench (ejemplo de la estructura general)

```
use STD.textio.all;
use IEEE.std_logic_textio.all;
Entity tutorial_tb Is
end tutorial_tb;

Architecture behavior of tutorial_tb Is
    Component tutorial
    port (
        swt : in STD_LOGIC_VECTOR(7 downto 0);
        led : out STD_LOGIC_VECTOR(7 downto 0)
        );
    End Component;

    Signal switch : STD_LOGIC_VECTOR(7 downto 0) := X"00";
    Signal led_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
    Signal led_exp_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
    Signal count_int_2 : STD_LOGIC_VECTOR(7 downto 0) := X"00";

    procedure expected_led (
        swt_in : in std_logic_vector(7 downto 0);
        led_expected : out std_logic_vector(7 downto 0)
    ) is

    Variable led_expected_int : std_logic_vector(7 downto 0) := "00000000";

    begin
        led_expected_int(0) := not(swt_in(0));
        led_expected_int(1) := swt_in(1) and not(swt_in(2));
        led_expected_int(2) := swt_in(2) and swt_in(3);
        led_expected_int(3) := led_expected_int(1) or led_expected_int(3);
        led_expected_int(7 downto 4) := swt_in(7 downto 4);

        led_expected := led_expected_int;
    end expected_led;
```

Test Bench (ejemplo.... continuación)

```
egin
    uut:  tutorial PORT MAP (
            swt => switch,
            led => led_out
          );

    process
        variable s : line;
        variable i : integer := 0;
        variable count : integer := 0;
        variable proc_out : STD_LOGIC_VECTOR(7 downto 0);

    begin
        for i in 0 to 127 loop
          count := count + 1;

          wait for 50 ns; switch <= count_int_2;

          wait for 10 ns; expected_led (switch, proc_out);
          led_exp_out <= proc_out;

          -- If the outputs match, then announce it to the simulator console.
          if (led_out = proc_out) then
              write (s, string'("LED output MATCHED at ")); write (s, count ); write (s, string'(". Expected: ")); write (s,
              writeline (output, s);
          else
             write (s, string'("LED output mis-matched at ")); write (s, count); write (s, string'(". Expected: ")); write (s,
             writeline (output, s);
          end if;

          -- Increment the switch value counters.
          count_int_2 <= count_int_2 + 2;
        end loop;

    end process;
end behavior;
```