BANCOS DE PRUEBA CON VHDL (TESTBENCHES)

Microarquitecturas y softcores

Laboratorio de Sistemas Embebidos

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Bancos de prueba con VHDL

¿Cómo realizar la prueba de un dispositivo descripto en VHDL?

Bancos de prueba con VHDL

- Lograr un Testbench de calidad es imprescindible para la verificación de los diseños
- No están regidos por las normas que se aplican en la síntesis de circuitos
- Se utiliza un generador de señales de prueba y se analiza la respuesta del circuito mediante simulación
- No eliminan por completo la necesidad de probar el circuito una vez sintetizado

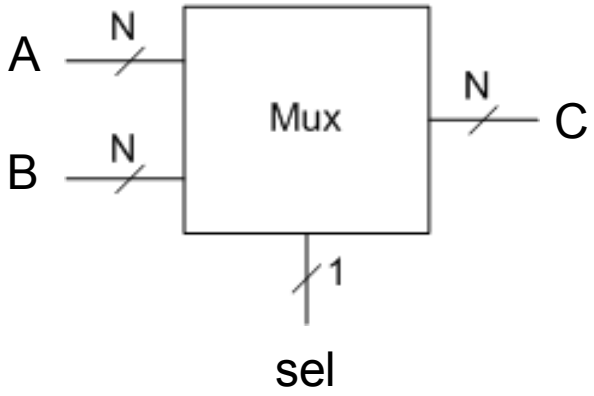
Bancos de prueba con VHDL

Formas de aplicar un estímulo

- Aplicación explícita del estímulo utilizando process
- Sentencia For-Loop
- Array
- Utilización de archivos

Aplicación explícita del estímulo utilizando process

Componente a probar (DUT)



Aplicación explícita del estímulo utilizando process

```
entity mux_tb is
end;

architecture beh of mx_tb is
  -- declaración del componente a probar
   signal mux_in0, mux_in1, mux_out: std_logic_vector(1 downto 0);
   signal sel: std_logic;
begin
   pp: mux port map(mux_in0, mux_in1, sel, mux_out);
   estimulo: process
   begin
     sel <= '0';
      mux_in0 <= "00"; mux_in1 <= "10"; wait for 30 ns;
      mux_in0 <= "01"; mux_in1 <= "11"; wait for 30 ns;

      sel <= '1';
      mux_in0 <= "00"; mux_in1 <= "10"; wait for 30 ns;
      mux_in0 <= "01"; mux_in1 <= "11"; wait for 30 ns;
      wait;
   end process;
end;
```

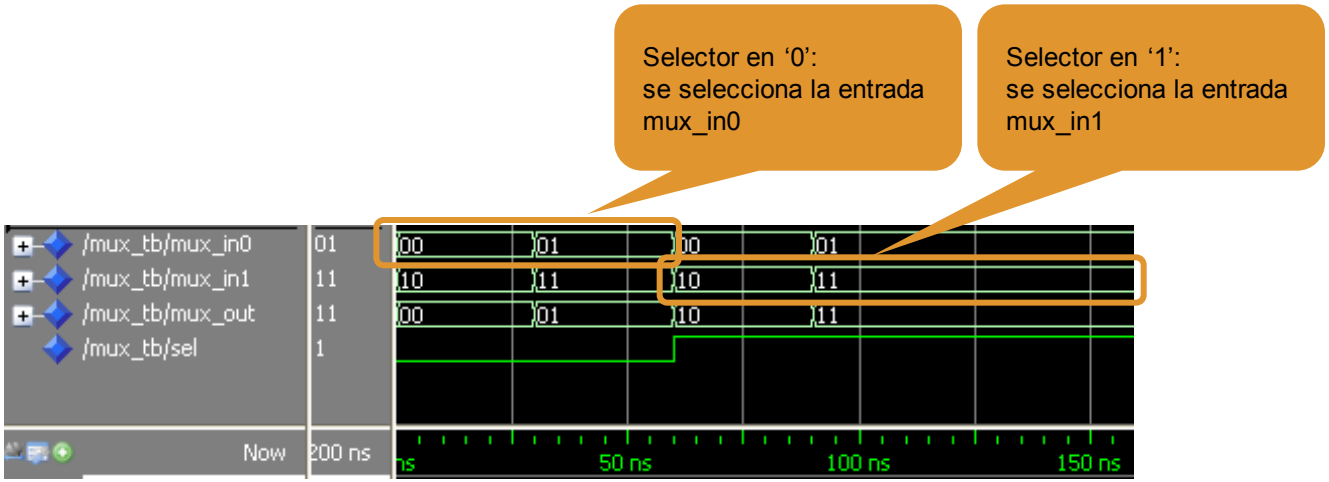
Se debe incluir la declaración del componente a probar

Declaración de las señales de prueba

Instanciación del componente a probar

Process utilizado para el armado de las señales de prueba

Aplicación explícita del estímulo utilizando process



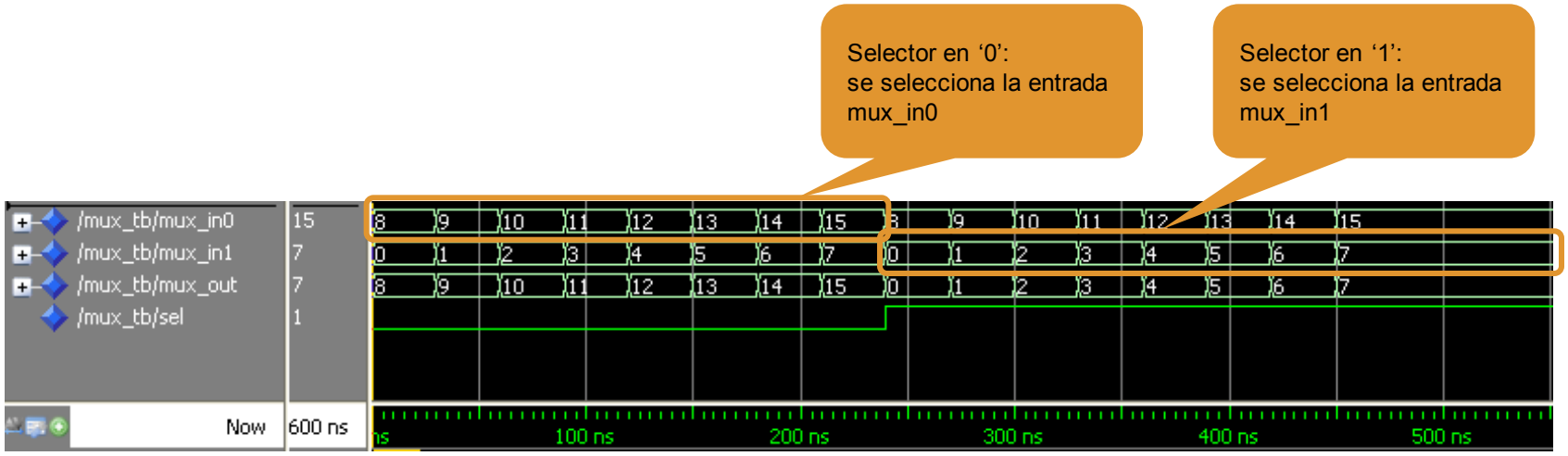
Aplicación explícita del estímulo utilizando For-Loop

```
entity mux_tb is
end;

architecture beh of mx_tb is
    -- declaración del componente a probar
    constant N_test: natural:= 4;
    signal mux_in0, mux_in1, mux_out: std_logic_vector(N_test-1 downto 0);
    signal sel: std_logic;
begin
    pp: mux generic map(N_test) port map(mux_in0, mux_in1, sel, mux_out);

    estimulo: process
    begin
      for i in 0 to 7 loop
        sel <= '0';
        mux_in1 <= std_logic_vector(to_unsigned(i, N_test));
        mux_in0 <= std_logic_vector(to_unsigned(i+8, N_test));
        wait for 30 ns;
      end loop;

      for i in 0 to 7 loop
        sel <= '1';
        mux_in1 <= std_logic_vector(to_unsigned(i, N_test));
        mux_in0 <= std_logic_vector(to_unsigned(i+8, N_test));
        wait for 30 ns;
      end loop;

      wait;
    end process;
end;
```

Se debe incluir la librería numeric_std

Bancos de prueba con VHDL

Aplicación explícita del estímulo utilizando For-Loop



Bancos de prueba con VHDL

Aplicación explícita del estímulo utilizando Arrays

```
entity mux_tb is
end;

architecture beh of mx_tb is
    -- declaración del componente a probar
    constant N_test: natural:= 4;
    signal mux_in0, mux_in1, mux_out: std_logic_vector(N_test-1 downto 0);
    signal sel: std_logic;
begin
    pp: mux generic map(N_test) port map(mux_in0, mux_in1, sel, mux_out);
    estimulo: process
      type vect_sim is array(0 to 7) of std_logic_vector(N_test-1 downto 0);
      variable vect_sim1: vect_sim:= ("0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111");
      variable vect_sim0: vect_sim:= ("1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111");
    begin
      for i in 0 to 7 loop
        sel <= '0';
        mux_in0 <= vect_sim0(i);
        mux_in1 <= vect_sim1(i);
        wait for 30 ns;
      end loop;
      for i in 0 to 7 loop
        sel <= '1';
        mux_in0 <= vect_sim0(i);
        mux_in1 <= vect_sim1(i);
        wait for 30 ns;
      end loop;
      wait;
    end process;
end;
```

Bancos de prueba con VHDL

Aplicación explícita del estímulo utilizando un archivo

- VHDL provee un tipo de datos llamado **file** que permite el manejo de archivos tanto de entrada como de salida
- Los archivos pueden abrirse de tres maneras: **read_mode** para lectura, **write_mode** para escritura y **append_mode** para agregar datos al final de un archivo existente
- Pueden declararse en la parte declarativa de una arquitectura, un proceso o un subprograma

Bancos de prueba con VHDL

Aplicación explícita del estímulo utilizando For-Loop

```
entity mux_tb is
end;

architecture beh of mx_tb is
    -- declaración del componente a probar
    -- declaración de constantes y señales IDEM ejemplo anterior
begin
    pp: mux generic map(N_test) port map(mux_in0, mux_in1, sel, mux_out);
    estimulo: process
      variable sel_aux: std_logic_vector(0 downto 0);
      file arch_stim: text open READ_MODE is "datos_simulacion.txt";
      variable linea: line; variable dato1, dato2, dato3: integer;
      variable ch: character:= ' ';
    begin
      while not (endfile(arch_stim)) loop
        readline(arch_stim, linea);
        read(linea, dato1); read(linea, ch); read(linea, dato2); read(linea, ch); read(linea, dato3);
        sel_aux := std_logic_vector(to_unsigned(dato1, 1));
        sel <= sel_aux(0);
        mux_in0 <= std_logic_vector(to_unsigned(dato2, N_test));
        mux_in1 <= std_logic_vector(to_unsigned(dato3, N_test));
        wait for 30 ns;
      end loop;
      wait;
    end process;
end;
```

Se debe incluir la librería std.textio

datos_simulacion.txt

```
0 0 8
0 1 9
0 2 10
0 3 11
0 4 12
0 5 13
0 6 14
0 7 15
1 0 8
1 1 9
1 2 10
1 3 11
1 4 12
1 5 13
1 6 14
1 7 15
```

Instrucciones Assert y Report

- Instrucciones que se utilizan para verificar una condición y emitir un mensaje durante la simulación
- Pueden usarse en cualquier parte de un process

```
assert condition
    [ report expression ] [ severity expression ];
```

```
[ label : ] report expression [ severity expression ] ;
```

Instrucciones Assert y Report: Ejemplos

```
assert MemoriaLibre >= MEM_LIMITE_MINIMO
report "Memoria baja, se sobrescribirán primeros valores"
severity note;
```

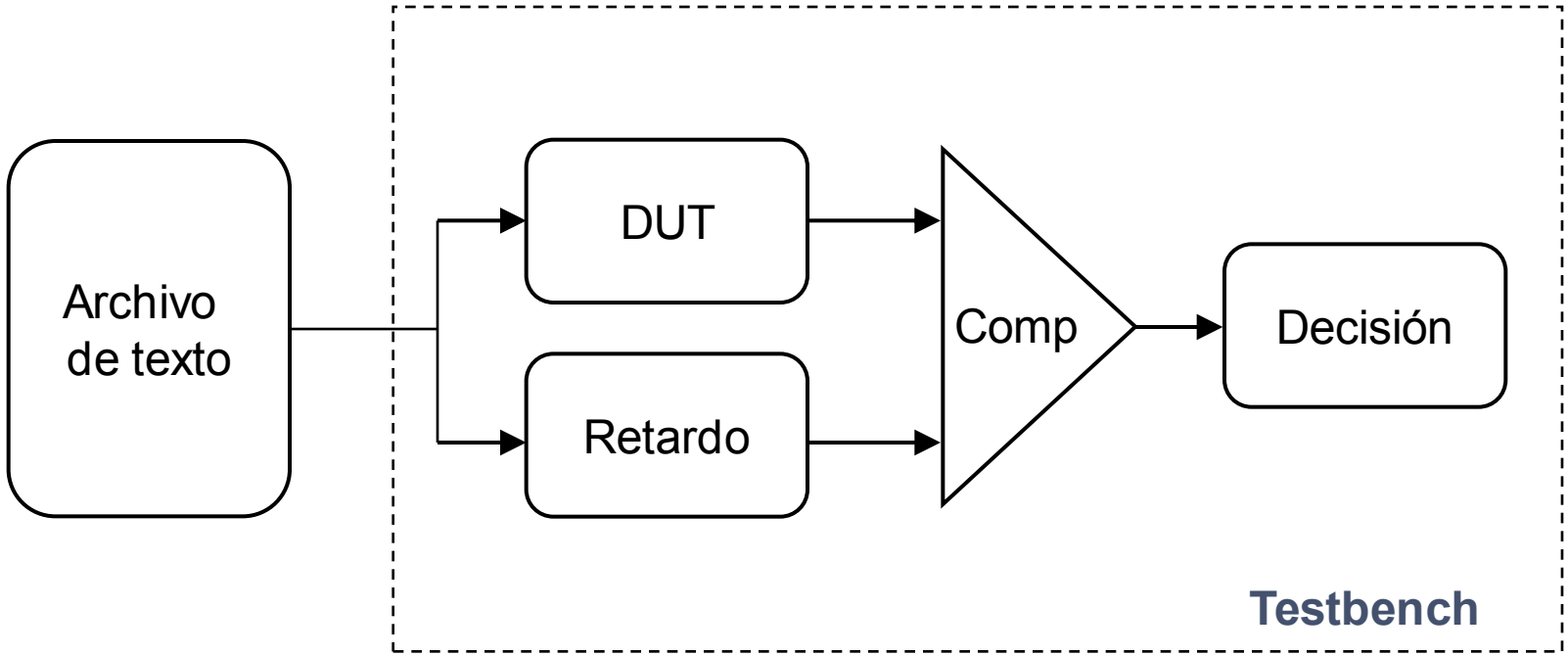
```
assert NumeroBytes /= 0
report "Se recibió paquete sin datos"
severity warning;
```

```
assert AnchoDePulso >= 2 ns
report "Pulso demasiado chico. No generará interrupción."
severity error;
```

```
report "Inicio de simulación."
```

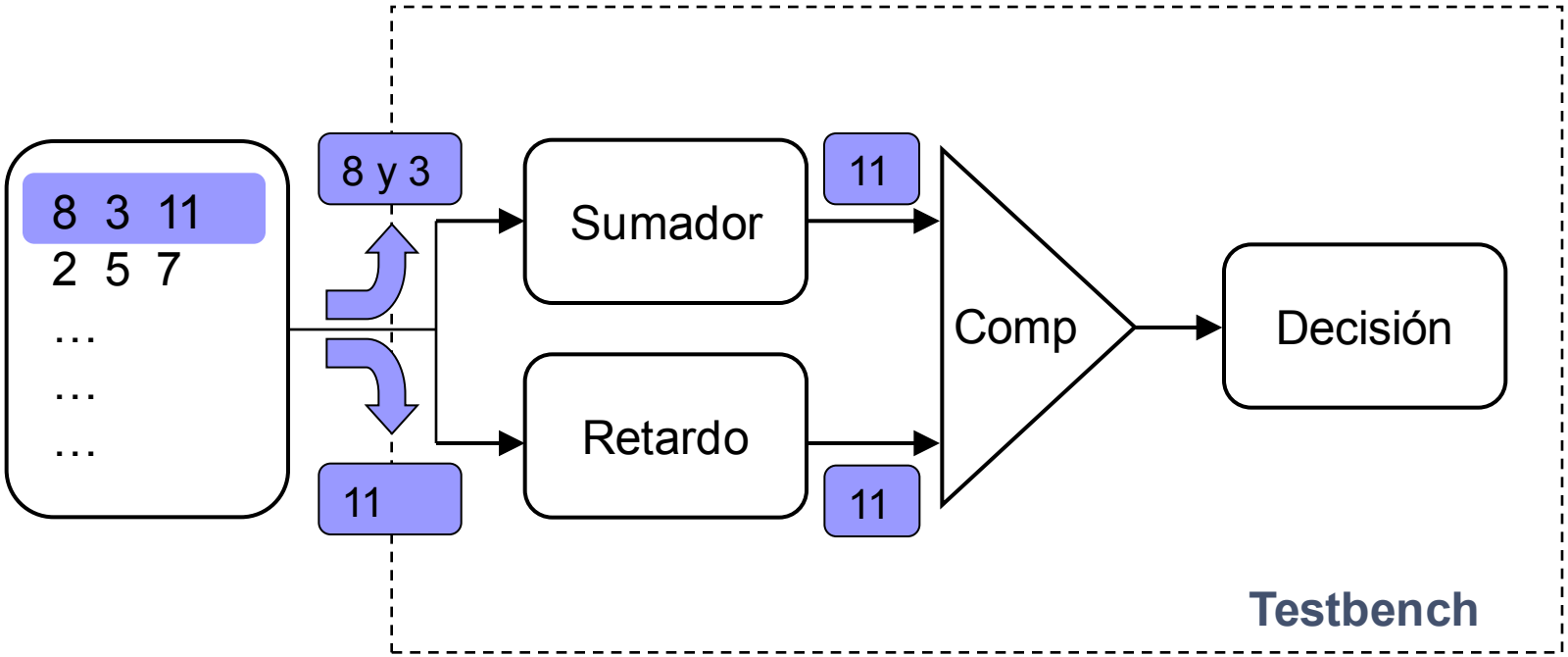
Bancos de prueba con VHDL

Testbench para el TP de Punto Flotante



Bancos de prueba con VHDL

Testbench para el TP de Punto Flotante



Bancos de prueba con VHDL

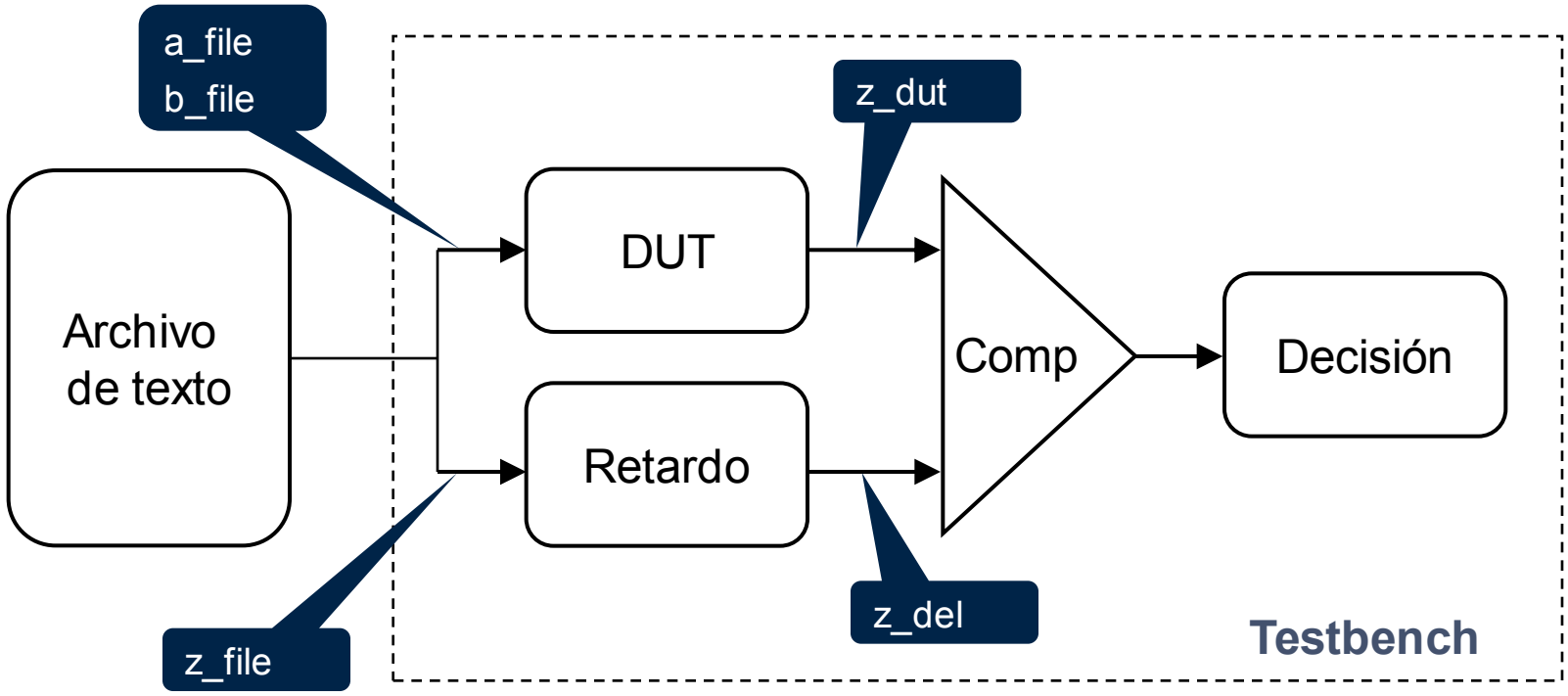
Testbench para el TP de Punto Flotante

- **Caso 1: el DUT no posee clock**

- **Caso 2: el DUT posee clock**
  En este caso el DUT necesita N ciclos de reloj para obtener un resultado válido a su salida por lo que se debe retardar el valor “resultado” obtenido del archivo de patrones de prueba.

Bancos de prueba con VHDL

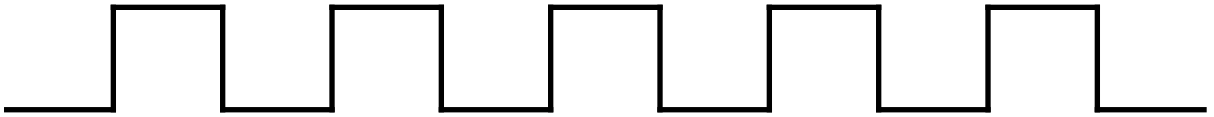
Testbench para el TP de Punto Flotante



Bancos de prueba con VHDL

Testbench para el TP de Punto Flotante

Diagrama temporal para un DUT con retardo de 2 ciclos de reloj



a_file	8	2	1		
b_file	3	5	2		
z_file	11	7	3		
z_dut			11	7	3

Bancos de prueba con VHDL

Testbench para el TP de Punto Flotante

- **Caso 1: el DUT no posee clock**

  En este caso las señales z_aux y z_del debe conectarse directamente con un cable

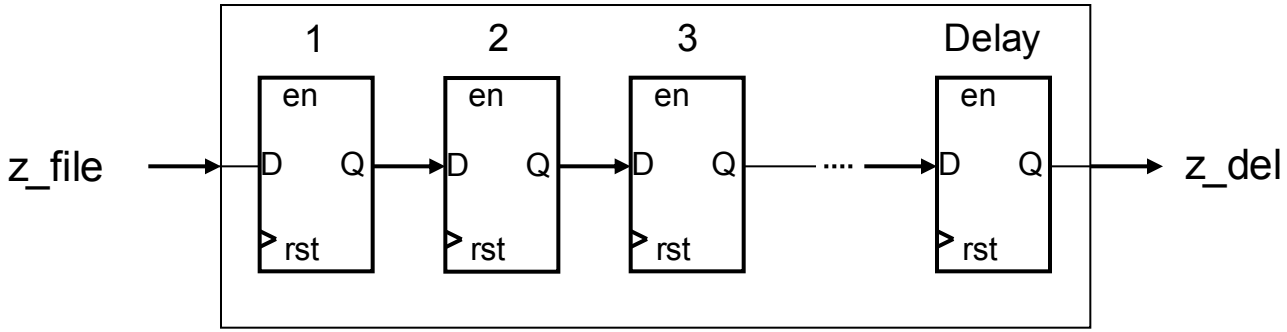
- **Caso 2: el DUT posee clock**

  En este caso se debe introducir un retardo en la señal z_file, de acuerdo a la cantidad de ciclos de reloj que necesite el DUT para procesar los datos de entrada.

Bancos de prueba con VHDL

Testbench para el TP de Punto Flotante

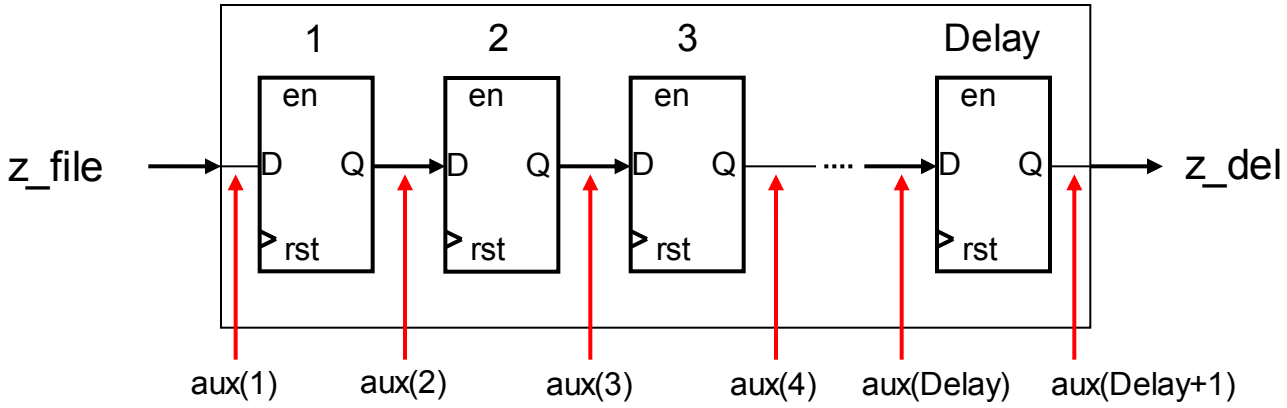
Como generar el retardo?



Ejercicio: Cómo se podría implementar utilizando la sentencia generate?

Testbench para el TP de Punto Flotante

Como generar el retardo?



Ayuda: utilizar una señal aux de DELAY+2 elementos

Testbench para el TP de Punto Flotante

Como generar el retardo?

```
aux(0) <= A;
gen_retardo: for i in 0 to DELAY generate
    sin_retardo: if i = 0 generate
        aux(1) <= aux(0);
    end generate;
    con_retardo: if i > 0 generate
        aa: ffd port map(clk => clk, rst => '0', D => aux(i), Q => aux(i+1));
    end generate;
end generate;
B <= aux(DELAY+1);
```

FIN