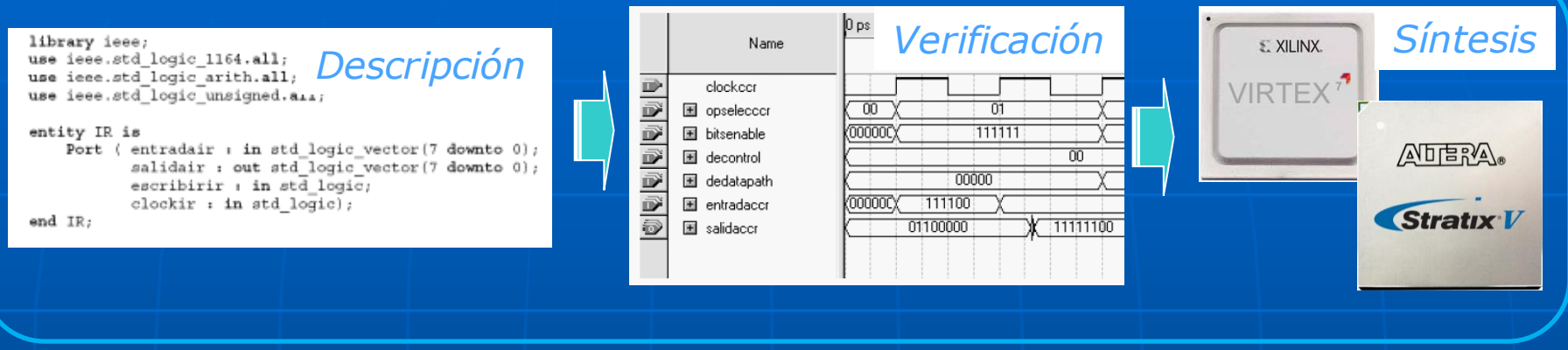



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity IR is
    Port ( entradair : in std_logic_vector(7 downto 0);
           salidair : out std_logic_vector(7 downto 0);
           escribirir : in std_logic;
           clockir : in std_logic);
end IR;
```

Introducción al diseño lógico con VHDL – Parte 2

SEÑALES VERSUS VARIABLES

SEÑALES

Las señales representan vínculos físicos entre procesos concurrentes.
Se pueden declaran en:
Packages: al describir componentes.
Entity: al describir los puertos de una entidad.
Architecture: al definir líneas internas.

Cuando una señal es del tipo INTEGER es importante definir su rango, sino el compilador asignará 32 bits por default.

En el caso de inferirse registros se generará gran cantidad de recursos innecesarios.

Las señales tienen vigencia en toda la descripción del circuito.
Sirven de nexo para comunicación entre componentes.
Se actualizan dentro de un 'Process' al final.

VARIABLES

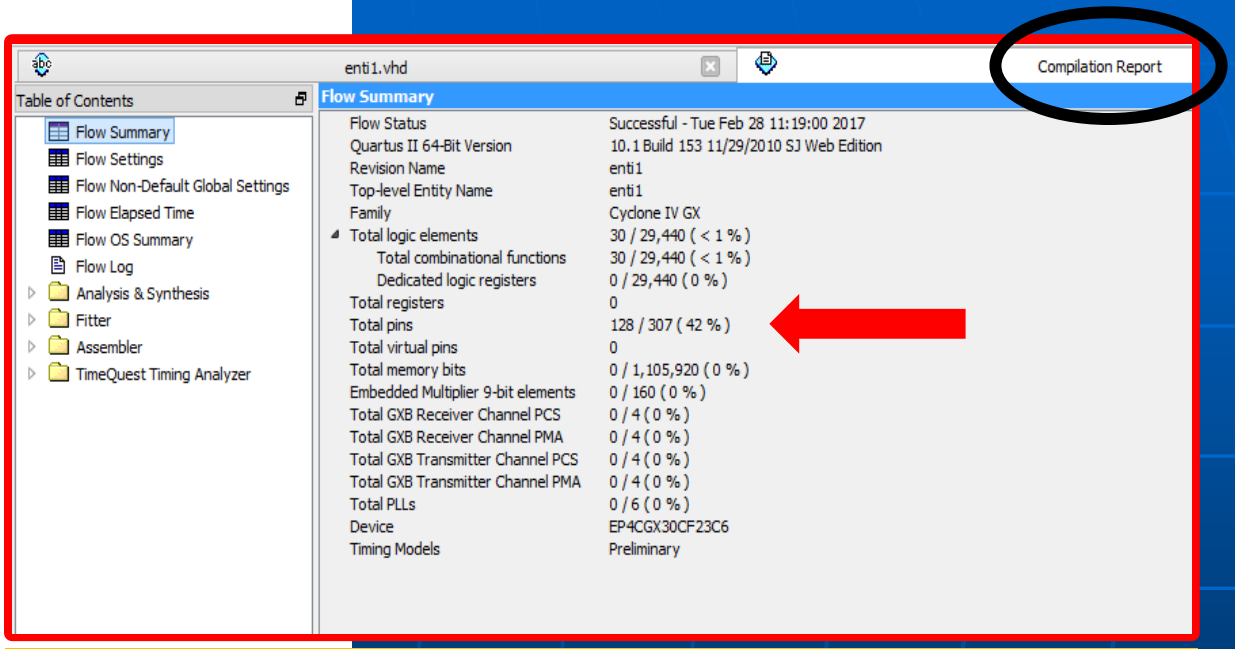
Las variables sólo tienen vigencia dentro del proceso donde están declaradas.
Proveen un mecanismo conveniente para almacenamiento local de información.
Se actualizan dentro de un 'Process' inmediatamente al invocarlas.
Utiles cuando se quiere escribir un código serializado
(Ver ejemplo de diseño de sumador ripple-carry con
FOR-LOOP ó FOR GENERATE)

SEÑALES

Cuando una señal es del tipo INTEGER es importante definir su rango, sino el compilador asignará 32 bits por default.

enti1.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity enti1 is
6
7      port
8      (
9         a1, b1   : in integer;
10        d1, e1   : out integer
11     );
12
13  end entity;
14
15  architecture maldef of enti1 is
16  SIGNAL a,b,c,d,e: integer := 0;
17  BEGIN
18   PROCESS (a,b,c)
19   BEGIN
20      c <= a;
21      d <= c + 12;
22      c <= b;
23      e <= c + 12;
24   END PROCESS;
25
26   a <= a1;
27   b <= b1;
28   d1 <= d;
29   e1 <= e;
30
31  END maldef;
32
```



Dos entradas (a1 y b1) y dos salidas (d1 y e1) especificadas como ENTEROS.
El compilador salvo alguna inicialización, las define como de 32 bits cada una (128pines /32 = 4 ports).

Una Solución es definir PORTS como arreglos de bits BIT_VECTOR ó STD_LOGIC_VECTOR).

PARA QUÉ SIRVEN LAS SEÑALES ?

Un ejemplo

```
entity and_nand is
      port (a, b : in bit;
                  z, zbar : out bit);
end;
architecture illegal of and_nand is
begin
          z <= a and b;
          zbar <= not z;
end;
```

Así no puede ser usado 'z', para leer su estado y modificar a zbar, porque 'z' es una salida.

```
entity and_nand is
      port (a, b : in bit;
                  z, zbar : out bit);
end;
architecture behaviour of and_nand is
      signal result : bit;
begin
      result <= a and b;
      z <= result;
      zbar <= not result;
end;
```

La solución es usar una 'señal' (result) para leer su estado y modificar zbar (y de paso a 'z' en este ejemplo).

CONVIENE SIEMPRE USAR SEÑALES Y AL FINAL DE «ARCHITECTURE» ASIGNARLAS A LOS PORTS DE SALIDA.

SEÑALES VERSUS VARIABLES

EJEMPLO

Definición de ´C´ como SEÑAL

enti1.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity enti1 is
6
7     port
8     (
9        a1, b1   : in integer;
10       d1, e1   : out integer
11    );
12
13 end entity;
14
15 architecture maldef of enti1 is
16    SIGNAL c, d, e : integer := 0;
17 BEGIN
18  PROCESS (a1,b1)
19  BEGIN
20     c <= a1;
21     d <= c + 12;
22     c <= b1;
23     e <= c + 12;
24  END PROCESS;
25
26  d1 <= d;
27  e1 <= e;
28
29 END maldef;
30
```

Defnición de ´C´ como VARIABLE

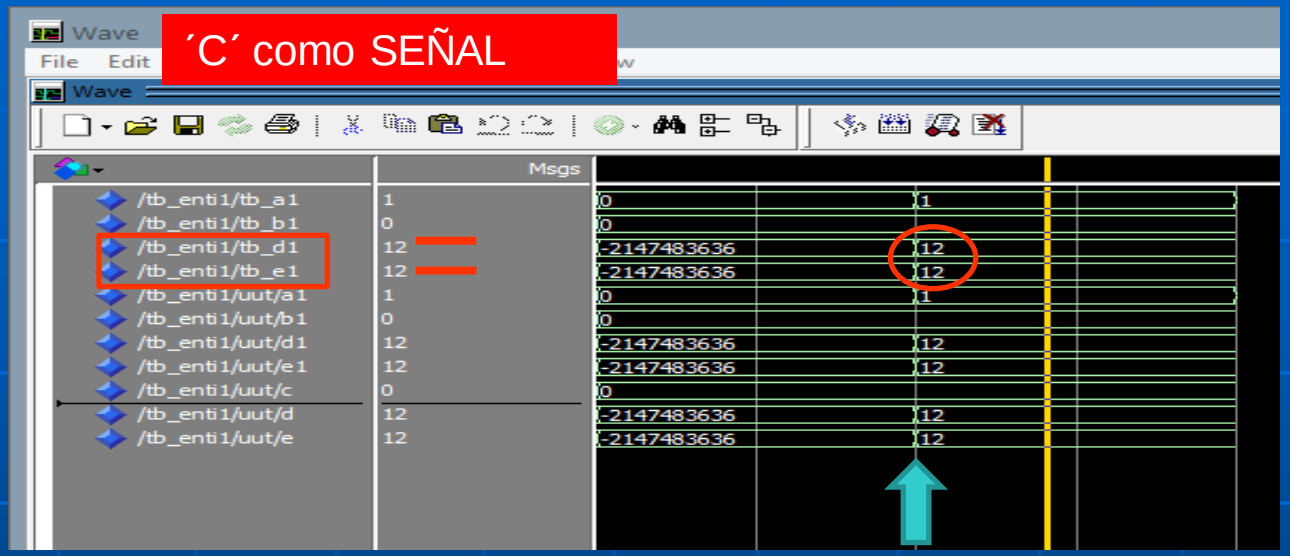
enti2.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity enti2 is
6
7     port
8     (
9        a1, b1   : in integer;
10       d1, e1   : out integer
11    );
12
13 end entity;
14
15 architecture biendef of enti2 is
16    SIGNAL d, e: integer := 0;
17 BEGIN
18  PROCESS (a1,b1)
19  variable c : integer := 0;
20  BEGIN
21     c := a1;
22     d <= c + 12;
23     c := b1;
24     e <= c + 12;
25
26  END PROCESS;
27
28  d1 <= d;
29  e1 <= e;
30
31 END biendef;
32
```

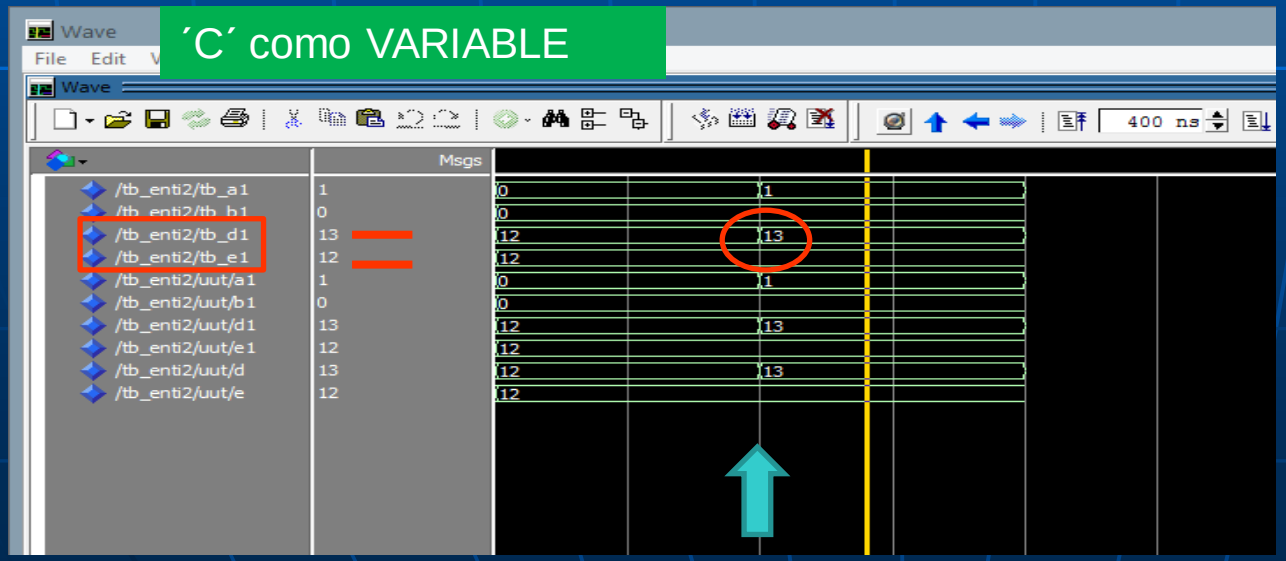
SEÑALES VERSUS VARIABLES

EJEMPLO (Continuación)

En t= 0 ns => a1 = 0 y b1 = 0.
En t= 200 ns => a1 = 1 y b1 = 0 (el cambio se muestra con la flecha ↑ .



d1 y e1 finalizan en 12.



e1 finaliza en 12
y
d1 finaliza en 13.

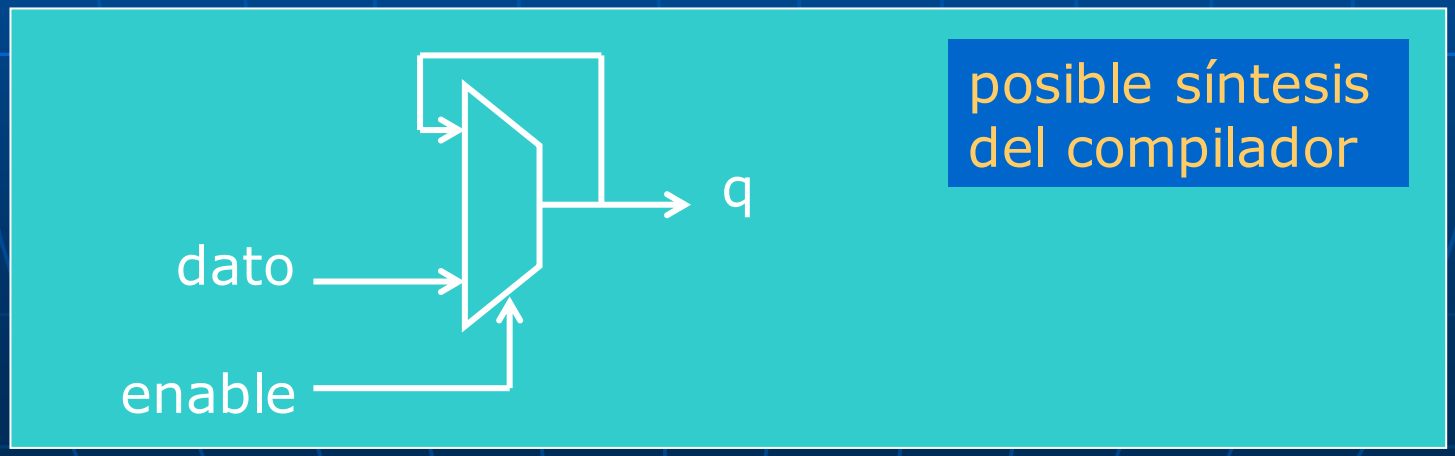
Diseño de Compuertas

Ejemplo: modelo de una compuerta AND

```
ENTITY and2 IS
        PORT (a:in bit; b:in bit; z:out bit);
END and2;

ARCHITECTURE funcional OF and2 IS
        BEGIN
            PROCESS (a,b)
                BEGIN
                        IF a=`1´and b=`1´ THEN z<=`1´;
                        ELSE z<=`0´;
            END PROCESS;
END funcional;
```

Diseño de Latches

```
ENTITY unlatch IS PORT ( enable, dato : IN std_logic;
        q : OUT std_logic);
END unlatch;

ARCHITECTURE tipo_latch1 OF unlatch IS BEGIN
  latch: PROCESS (enable, dato)
    BEGIN
        IF enable = `1´ THEN q <= dato; END IF;
  END PROCESS latch;
END tipo_latch1;
```



Diseño de Flip-Flops

Flip-Flop tipo "D"

```
LIBRARY ieee;
USE ieee.std_logc_1164.all;

ENTITY ff_tipod IS PORT ( d, reloj : IN std_logic;
            q : OUT std_logic);
END ff_tipod;

ARCHITECTURE tipo_reloj1 OF ff_tipod IS BEGIN
  PROCESS
    BEGIN
          WAIT UNTIL reloj = `1´; q <= d; --sensible a flanco ascendente
  END PROCESS;
END tipo_reloj1;
```

Otra forma de declarar la sensibilidad al flanco ascendente:

```
PROCESS (reloj) BEGIN
  IF reloj´EVENT AND reloj=`1´ THEN q <= d; END IF;
  END PROCESS;
```

Diseño de Flip-Flops

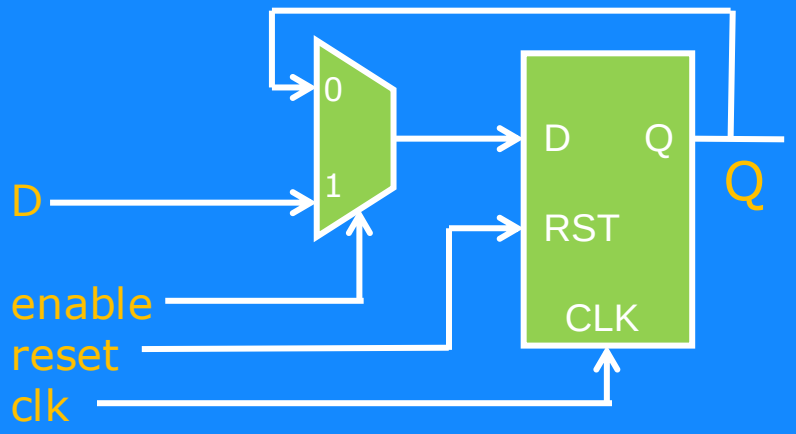
flip_flop_d1.vhd - Text Editor

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY flip_flop_d1 IS
    PORT
    (
        D, reset, clk, enable       : IN  std_logic;
        Q                           : OUT std_logic
    );
END flip_flop_d1;

ARCHITECTURE ffd1 OF flip_flop_d1 IS
BEGIN
    process(CLK, RESET, ENABLE)
        begin
            if RESET = '1' then
                Q <= '0';
            elsif rising_edge(CLK) then
            if Enable = '1' then
                Q <= D;
            end if;
            end if;
    end process;
END ffd1;
```

Flip-Flop tipo "D" con RESET asincrónico y habilitación de RELOJ

Aquí el RESET está fuera del proceso donde se evalúa que se hace cuando viene el flanco ascendente del reloj.
Como está antes de la sentencia de detección del flanco de clk el "reset" funciona como asincrónico.

Posible síntesis del compilador



Diseño de Flip-Flops

flip_flop_d2.vhd - Text Editor

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY flip_flop_d2 IS
    PORT
    (
        D, reset, clk, enable       : IN  std_logic;
        Q                           : OUT std_logic
    );
END flip_flop_d2;

ARCHITECTURE ffd2 OF flip_flop_d2 IS
BEGIN
    process(CLK, RESET, ENABLE)
        begin
            if rising_edge(CLK) then
                if RESET = '1' then
                    Q <= '0';
                elsif Enable = '1' then
                    Q <= D;
                end if;
            end if;
    end process;
END ffd2;
```

Flip-Flop tipo "D" con RESET sincrónico y habilitación de RELOJ

Ahora RESET está dentro del proceso y además se evalúa luego de detectar el flanco ascendente del reloj.
Por lo tanto el "reset" es sincrónico.

Diseño de Flip-Flops

Registro de 8 bits con RESET sincrónico

registro8bits.vhd - Text Editor

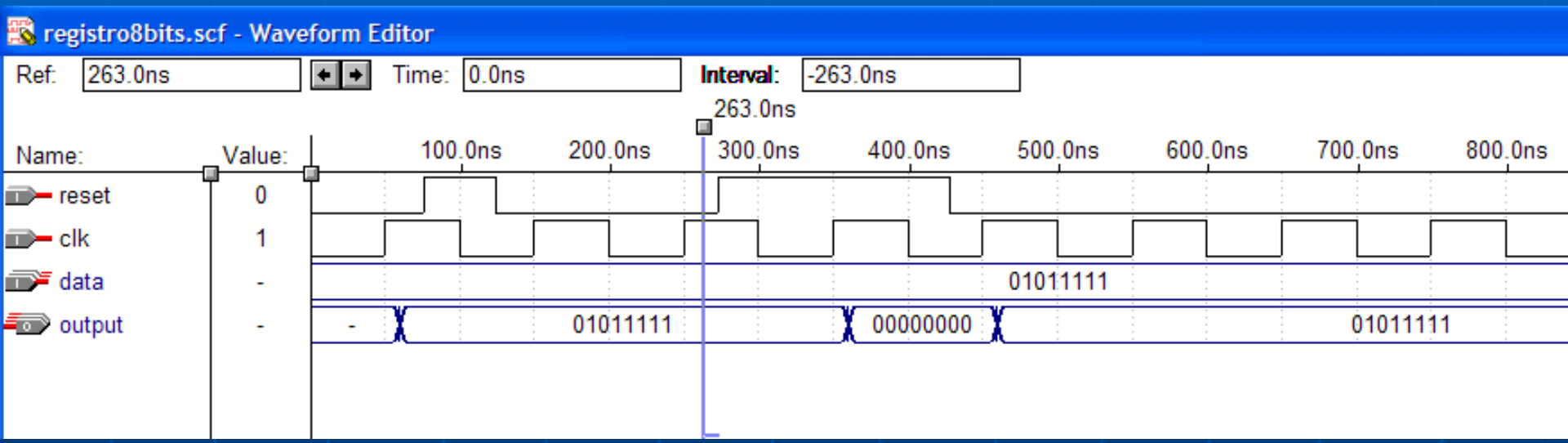
```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY registro8bits IS
    PORT
    (
        data         : IN  std_logic_vector(7 DOWNTO 0);
        reset, clk   : IN  std_logic;
        output           : OUT std_logic_vector(7 DOWNTO 0)
    );
END registro8bits;

ARCHITECTURE ffdx8 OF registro8bits IS
BEGIN
    process(CLK, RESET)
        BEGIN
            WAIT UNTIL (clk'EVENT AND clk ='1');
                IF RESET = '1' then
                     output <= "00000000";
                ELSE output <= data;
                END IF;
    END process;
END ffdx8;
```

El RESET está dentro del proceso donde se evalúa cuando viene el flanco ascendente del reloj

Diseño de Flip-Flops

Registro de 8 bits con RESET sincrónico

Simulación



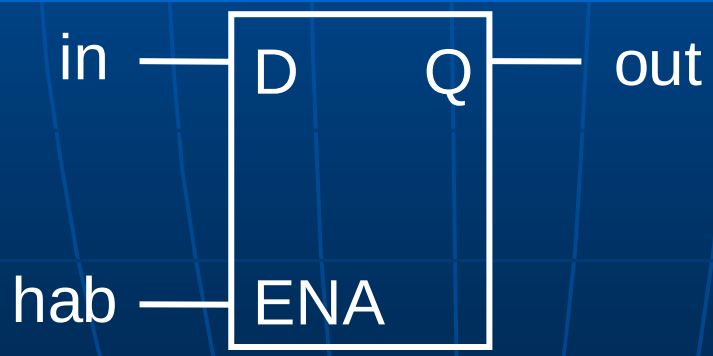
INFERENCIA DE ‘LATCH’ POR ESPECIFICACIÓN INCOMPLETA

ejemplo 1

```
signal in, out : std_logic_vector(3 downto 0);
signal hab : std_logic

process (hab, in)
    begin
      if hab = '1' then
       out <= in;
      end if;
end process;
```

No se definió que pasa cuando ‘hab’ es ‘0’ …!!!

El compilador, entonces, mantiene el último valor de la salida si pasa hab a ‘0’.



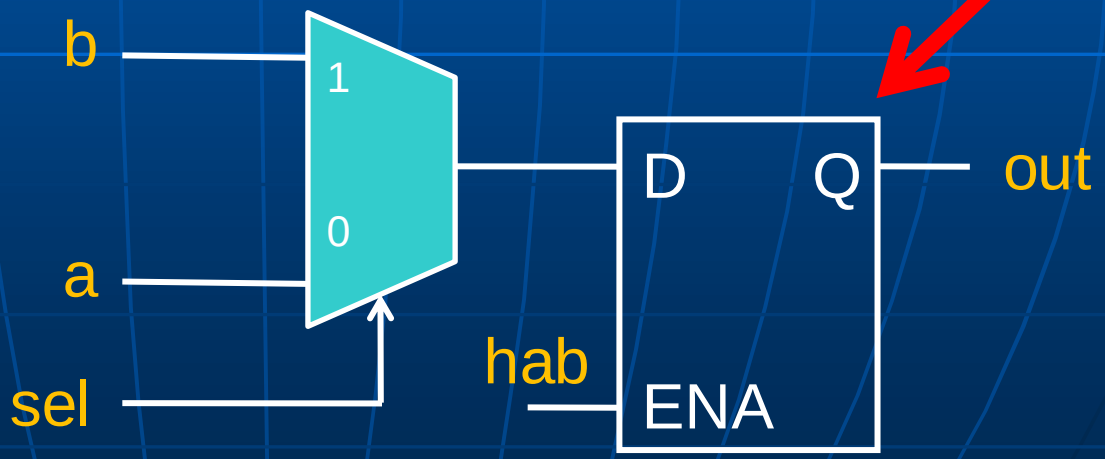
INFERENCIA DE 'LATCH' POR REQUERIMIENTO

ejemplo 2

```
process (hab, sel, a, b)
  begin
    if hab = '1' then
       if sel = '0' then
          z <= a;
       else
          z <= b;
       end if;
    end if;
end process;
```

En caso de requerir la habilidad de memorizar la salida del MUX, se omite exprofeso la opción de 'hab = 0' …

El compilador, entonces, mantiene el último valor de la salida si pasa hab a '0'.



Diseño de Contadores

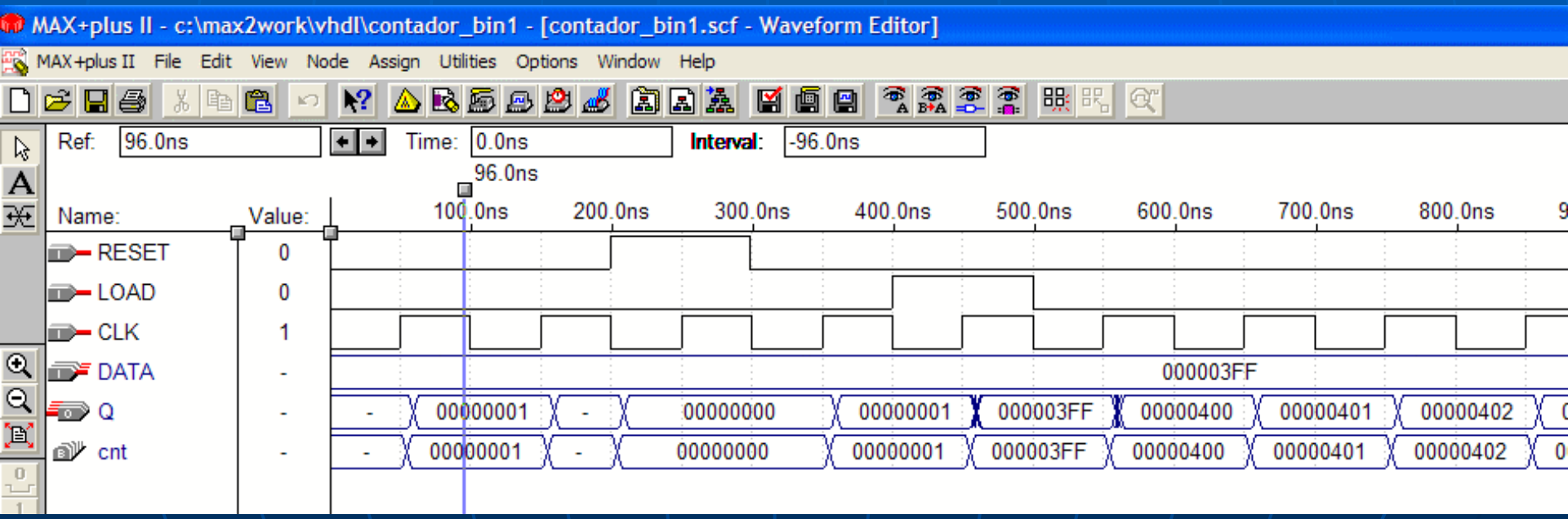
Contador binario progresivo de 32 bits con precarga y reset asincrónico

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity contador_bin1 is
generic ( WIDTH : integer := 32);
port (
  CLK, RESET, LOAD : in std_logic;
  DATA : in  unsigned(WIDTH-1 downto 0);
  Q    : out unsigned(WIDTH-1 downto 0));
end entity contador_bin1;

architecture contador_32 of contador_bin1 is
signal cnt : unsigned(WIDTH-1 downto 0);
begin
  process(RESET, CLK)
  begin
    if RESET = '1' then
      cnt <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD = '1' then
        cnt <= DATA;
      else
        cnt <= cnt + 1;
      end if;
    end if;
  end process;

  Q <= cnt;

end architecture contador_32;
```

Diseño de Contadores

Contador binario progresivo de 32 bits con precarga y reset asincrónico

Simulación

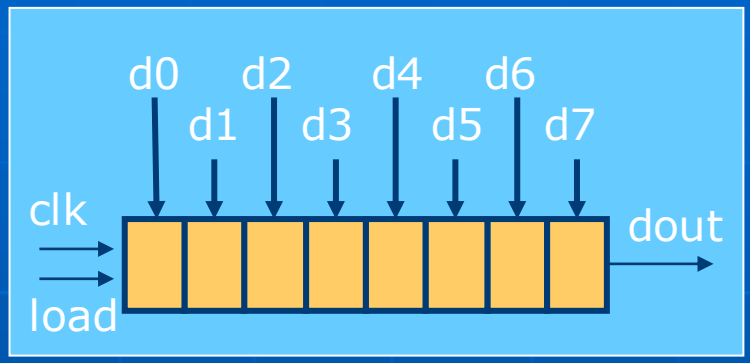


Diseño de Registro de Desplazamiento

RD paralelo-serie con carga sinc.

rd.vhd - Text Editor

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY rd IS PORT (
  d : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
  clk, load : IN STD_LOGIC;
  dout : OUT STD_LOGIC);
END rd;

ARCHITECTURE rdpara_serie OF rd IS
   SIGNAL reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
   BEGIN
   PROCESS (clk)
     BEGIN
       IF (clk'EVENT AND clk='1') THEN
         IF (load='1') THEN reg <= d;
            ELSE reg <= reg(6 DOWNTO 0) & '0';
         END IF;
       END IF;
   END PROCESS;
   dout <= reg(7);
END rdpara_serie;
```



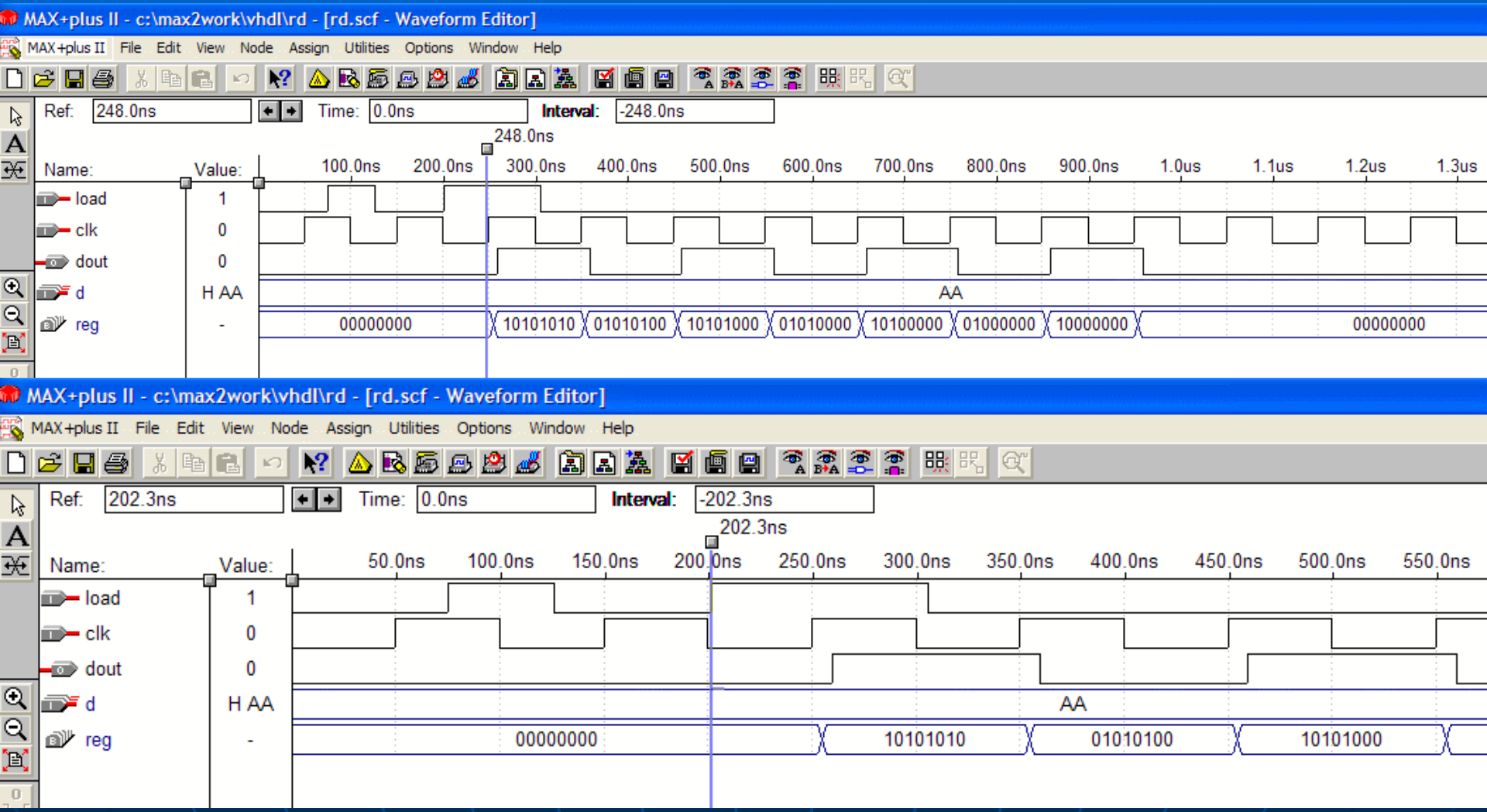
La carga del registro de desplazamiento es sincrónica ya que está dentro del proceso de control del flanco del reloj.

Esta operación con el operador de concatenación agrega en este caso un “0” en el bit LSB del RD.

Diseño de Registro de Desplazamiento

RD paralelo-serie con carga sinc.

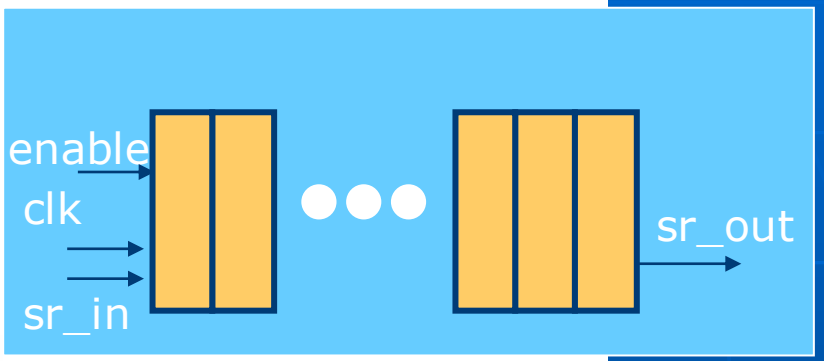
Simulación



Diseño de Registro de Desplazamiento

RD serie-serie.

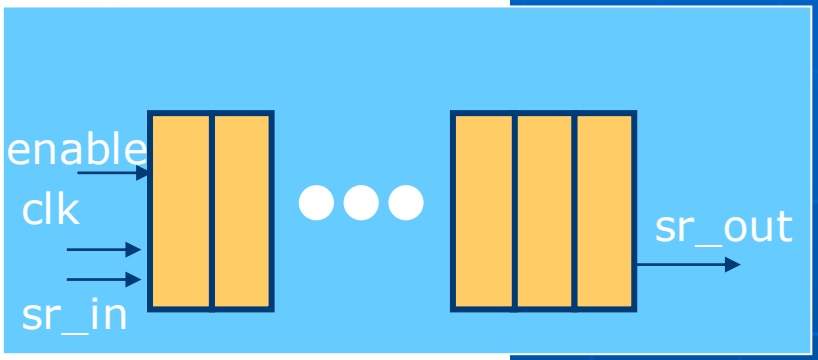
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity shift_1x64 is
5       port
6       (
7           sr_in : in std_logic;
8           enable   : in std_logic;
9           clk      : in std_logic;
10          sr_out   : out std_logic
11      );
12
13  end entity;
14
```



Diseño de Registro de Desplazamiento

RD serie-serie.

```
15  architecture rtl of shift_1x64 is
16
17      -- Build an array type for the shift register
18      type sr_length is array (63 downto 0) of std_logic;
19
20      -- Declare the shift register signal
21      signal sr: sr_length;
22
23  begin
24
25      process (clk)
26      begin
27          if (rising_edge(clk)) then
28              if (enable = '1') then
29
30                  -- Shift data by one stage; data from last stage is lost
31                  sr(63 downto 1) <= sr(62 downto 0);
32
33                  -- Load new data into the first stage
34                  sr(0) <= sr_in;
35
36              end if;
37          end if;
38      end process;
39
40      -- Capture the data from the last stage, before it is lost
41      sr_out <= sr(63);
42
```

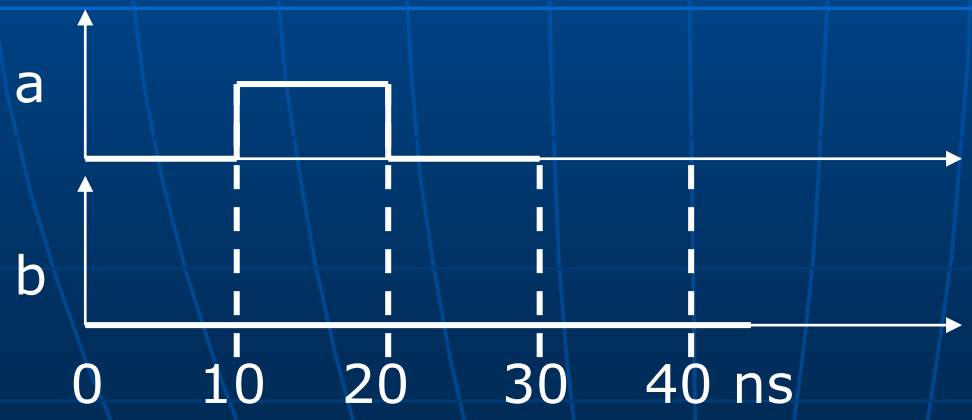


MODELADO POR COMPORTAMIENTO

Modelado por RETARDO INERCIAL (default en VHDL)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY buf IS
PORT ( a : IN std_logic;
       b : OUT std_logic);
END buf;

ARCHITECTURE buffer_ine OF buf IS
BEGIN
       b <= a AFTER 20 ns;
END buffer_ine;
```

delay = 20 ns

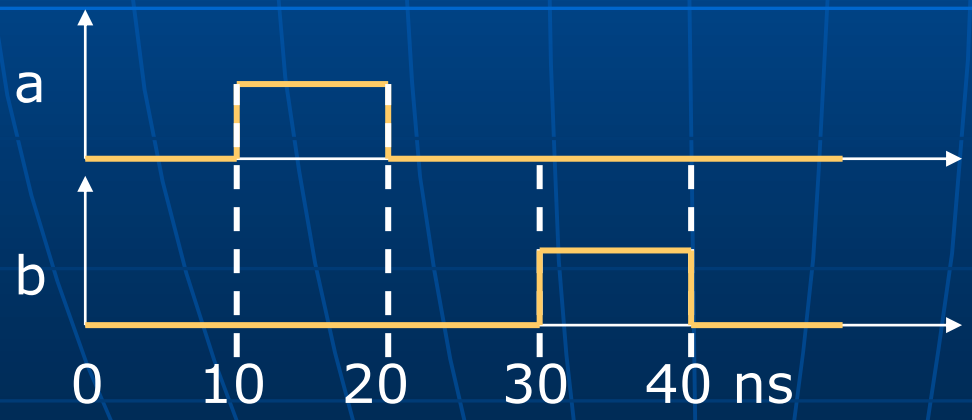


MODELADO POR COMPORTAMIENTO

Modelado por RETARDO de TRANSPORTE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY linea_ret IS
PORT ( a : IN std_logic;
       b : OUT std_logic);
END linea_ret;

ARCHITECTURE delay_line OF buf IS
BEGIN
       b <= a TRANSPORT AFTER 20 ns;
END delay_line;
```

delay = 20 ns



TEST BENCH EN VHDL

Con VHDL es posible modelar no sólo el hardware sino también realizar un "banco de pruebas" (test bench) para aplicar estímulos al diseño a fin de analizar los resultados ó comparar los mismos de dos simulaciones diferentes.

VHDL es entonces además de un lenguaje de descripción de hardware un lenguaje para la definición de estímulos a los modelos descriptos en el mismo entorno de diseño.

A diferencia de los simuladores propietarios de las empresas proveedoras de dispositivos lógicos programables, los test bench realizados en VHDL permiten no sólo describir los estímulos en forma textual sino que es posible también comparar los resultados obtenidos al ensayar un DUT (dispositivo bajo prueba) con otros ya previamente cargados y generar entonces reportes de errores en caso de disparidad.
Esto se denomina "simulación automática".

TEST BENCH EN VHDL

La simulación en VHDL permite mayor nivel de abstracción que con la síntesis.
Es posible por lo tanto modelizar el comportamiento de dispositivos sin que ellos sean luego sintetizados.
Por ejemplo en el caso de la simulación de un microprocesador donde se puede describir el comportamiento de una memoria aunque ésta no sea luego físicamente sintetizada.

Al ser VHDL un lenguaje portable es posible trabajar con simuladores de distintas empresas.

Ejemplo: El software ModelSim de Mentor Graphics.
(Existe una versión de trabajo para su uso con el Quartus II de Altera)

TEST BENCH EN VHDL

Diagrama de flujo de las opciones de simulación con el ModelSim-Altera

Es posible realizar simulaciones del tipo:

--Funcional RTL (Register Transfer Level)

--Post-síntesis

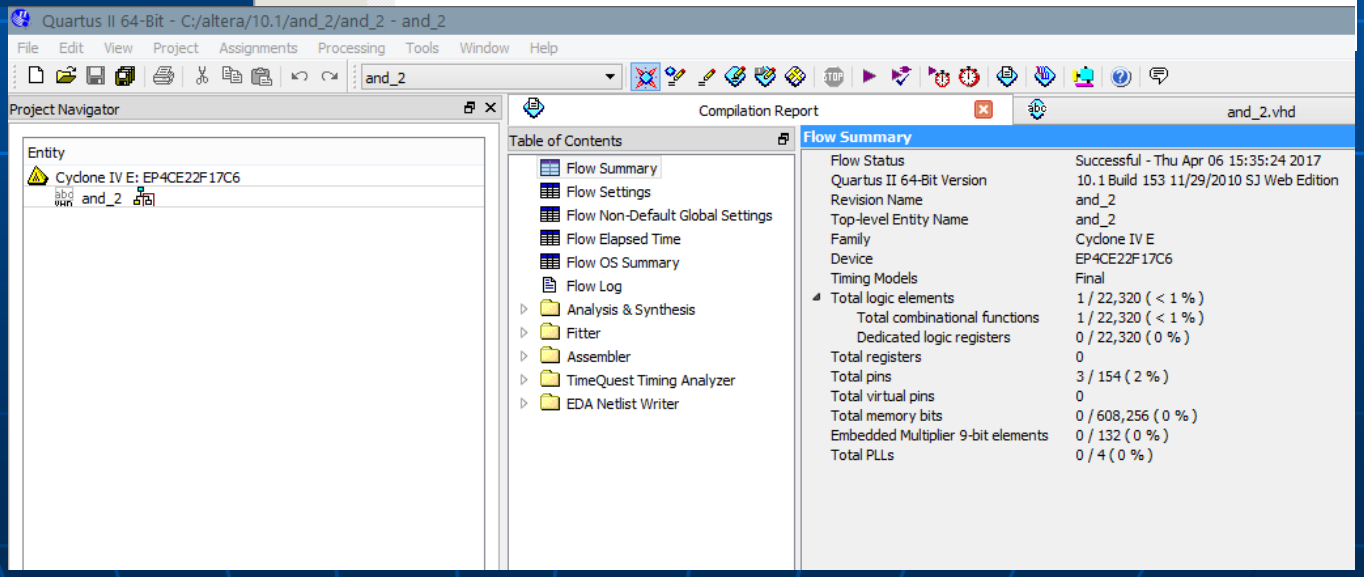
--Post-place&route (simulación temporal con información de timing precisa la cual debe ser provista por el fabricante del PLD ó ASIC)

TEST BENCH EN VHDL

Ejemplo de simulación sencilla con el software ModelSim de (Mentor Graphics)

Archivo VHDL que contiene el dispositivo a simular (and_2.vhd):
En este caso una compuerta AND de 2 entrads.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity and_2 is
5   Port ( entrada_a       : in std_logic;
6          entrada_b       : in std_logic;
7          salida_c        : out std_logic
8       );
9   end and_2;
10
11  architecture Comportamiento of and_2 is
12     begin
13        process(entrada_a, entrada_b)
14           begin
15              if (entrada_a = '1' and entrada_b = '1') then
16                 salida_c <= '1';
17              else
18                 salida_c <= '0';
19              end if;
20        end process;
21     end Comportamiento;
```



TEST BENCH EN VHDL

Ejemplo de simulación sencilla con ModelSim (continuación)

Archivo VHDL que contiene el test-bench de la "and" descripto anteriormente en otro archivo: (tb_and_2.vhd).

Sección para la generación de las señales de estímulo y acciones a tomar en caso de detectar inconsistencias en el funcionamiento.

En este caso se chequea si la salida está en '1' luego de 1ns y 21 ns al cambiar ambas entradas a '1'.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY tb_and_2 IS
5  END tb_and_2;
6
7  ARCHITECTURE behavior OF tb_and_2 IS
8
9      COMPONENT and_2 is
10     Port ( entrada_a    : in  std_logic;
              entrada_b    : in  std_logic;
              salida_c : out std_logic
            );
          end COMPONENT;

       signal entrada_a, entrada_b, salida_c : std_logic;

   BEGIN
19
20    uut: and_2 PORT MAP (entrada_a, entrada_b, salida_c);
21
22    estimulo_proc: process
23       begin
24          entrada_a <= '0';
25          entrada_b <= '0';
26          wait for 10 ns;
27          entrada_a <= '1';
28          entrada_b <= '1';
29          wait for 1 ns;
            assert (salida_c = '1')
31          report "Error en la salida. Deberia ser '1'"
32          severity ERROR;
33          wait for 10 ns;
34          assert (salida_c = '1')
35          report "Error en la salida. Deberia ser '1'"
36          severity ERROR;
37          wait for 10 ns;
38    end process;
39
40 END;
```

Instanciación de la entidad de la "and" ahora como un componente dentro de la entidad llamada "test_and_2"

TEST BENCH EN VHDL

Ejemplo de simulación con ModelSim