

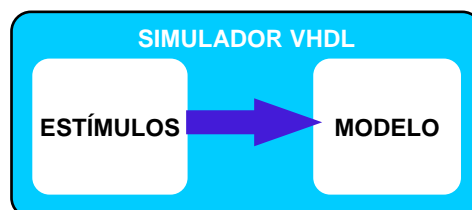
VHDL para simulación

VHDL para simulación

El VHDL fue creado para ser simulado (existe una forma *estándar* de simular las descripciones VHDL)

... además ...

se puede definir en VHDL tanto el circuito a simular como los estímulos a aplicar





VHDL para simulación

entity TbBiEstable is
end TbBiEstable;

declaración de la entidad

architecture SIMULA of TbBiEstable is
-- Declaración de componentes
component BiEstable
port (CLK, RESN, DATO : in bit;
Q : out bit);
end component;

declaración de componentes, señales y constantes

-- Declaración de señales y constantes
signal CLK, RESN, DATO, Q : bit;
constant CICLO : time := 100ns;

begin
-- Colocación y conexión de componentes
DFF : BiEstable
port map (CLK, RESN, DATO, Q);

colocación y conexión de componentes

-- generamos el reloj
RELOJ : process
begin
CLK <= '0';
wait for CICLO / 2;
CLK <= '1';
wait for CICLO / 2;
end process RELOJ;
-- generamos el test
TEST : process
begin
RESN <= '0';
DATO <= '1';
wait for CICLO * 2;
RESN <= '1';
wait for CICLO;
DATO <= '0';
wait for CICLO * 2;

Generación del reloj

asignación de valores al resto de las señales

assert false
report "FIN DE LA SIMULACION"
severity failure;
end process TEST;
end SIMULA;

control del fin de la simulación



Simulación: Bancos de prueba (Testbench)

- Hay que comprobar “que el circuito hace lo que debe” y “que no hace lo que no debe”
- Siempre se comprueban:
 - ✓ Inicializaciones
 - ✓ Cambios de modo de funcionamiento
 - ✓ Cobertura de código (100% aconsejable)
 - ✓ Interfaz con su entorno



Simulación: Cómo probar un modelo VHDL

Se pueden establecer varios niveles de verificación

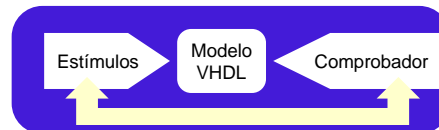
Estímulos + modelo VHDL
Inspección visual de los resultados



Estímulos + modelo VHDL + comprobador de resultados
Conocimiento de los resultados "a priori"



Estímulos dependientes de los resultados + modelo VHDL + comprobador de resultados



Simulación: Ejemplo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

-- Circuito síncrono;
-- Reset asíncrono activo por nivel bajo;
-- Cuando Load es 1 se carga con el ParallelData;
-- Cuando Shift es 1 desplaza de los bits menos significativos
-- a los mas significativos e introduce SerialData en el LSB

```
entity ShiftLoad is
  port ( clk : in std_logic;
        reset : in std_logic;
        Shift : in std_logic;
        Load : in std_logic;
        SerialData : in std_logic;
        ParallelData : in std_logic_vector(7 downto 0);
        DataOut : out std_logic_vector(7 downto 0));
end ShiftLoad;
```

```
architecture Behavioral of ShiftLoad is
  signal DatosInternos: std_logic_vector (7 downto 0);
begin
  process (clk, reset)
  begin
    if reset = '0' then
      DatosInternos <= (others => '0');
    elsif clk = '1' and clk'event then
      if Load = '1' then
        DatosInternos <= ParallelData;
      elsif Shift = '1' then
        for i in 7 downto 1 loop
          DatosInternos(i) <= DatosInternos (i-1);
        end loop;
        DatosInternos(0) <= SerialData;
      end if;
    end if;
  end process;
  DataOut <= DatosInternos;
end Behavioral;
```



Simulación: Ejemplo

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-- Declaración de la entidad
ENTITY shiftload_TBShiftLoad_vhd_tb IS
END shiftload_TBShiftLoad_vhd_tb;

-- Declaración de componentes
ARCHITECTURE behavior OF shiftload_TBShiftLoad_vhd_tb IS
COMPONENT shiftload
PORT( clk : IN std_logic;
      reset : IN std_logic;
      Shift : IN std_logic;
      Load : IN std_logic;
      SerialData : IN std_logic;
      ParallelData : IN std_logic_vector(7 downto 0);
      DataOut : OUT std_logic_vector(7 downto 0));
END COMPONENT;
SIGNAL clk : std_logic;
SIGNAL reset : std_logic;
SIGNAL Shift : std_logic;
SIGNAL Load : std_logic;
SIGNAL SerialData : std_logic;
SIGNAL ParallelData : std_logic_vector(7 downto 0);
SIGNAL DataOut : std_logic_vector(7 downto 0);
constant ciclo : time := 50 ns;

BEGIN
-- Colocación y conexión de componentes
uut: shiftload PORT MAP(
    clk => clk,
    reset => reset,
    Shift => Shift,
    Load => Load,
    SerialData => SerialData,
    ParallelData => ParallelData,
    DataOut => DataOut
);

-- proceso que genera el reloj
GenCLK: process
begin
    clk <= '0';
    wait for ciclo/2;
    clk <= '1';
    wait for ciclo;
end process GenCLK;

-- *** Test Bench - User Defined Section ***
...
```



Simulación: Ejemplo

```
-- *** Test Bench - User Defined Section ***
tb : PROCESS
BEGIN
--Iniciación
    reset <= '0';
    Shift <= '0';
    Load <= '0';
    SerialData <= '0';
    ParallelData <= (others => '0');
    wait for ciclo;
    assert (DataOut = "00000000")
        report "Error en la prueba del reset"
        severity error;
    wait for ciclo;
-- Desplazamiento
    reset <= '1';
    Shift <= '1';
    SerialData <= '1';
    wait for ciclo * 8;
-- Control del fin de la simulación
    assert (DataOut = "11111111")
        report "Error en la prueba de desplazamiento"
        severity error;
    wait for ciclo;

-- Carga
    ParallelData <= "01010101";
    Shift <= '0';
    Load <= '1';
    wait for ciclo;
    assert (DataOut = "01010101")
        report "Error en la prueba de carga"
        severity error;
    wait for ciclo;
-- Desplazamiento
    Shift <= '1';
    SerialData <= '0';
    wait for ciclo * 8;
    assert (DataOut = "00000000")
        report "Error en la segunda prueba de desplazamiento"
        severity error;
    wait for ciclo;
    Load <= '1';
    ParallelData <= (others => '1');
    wait for ciclo;
    assert (DataOut = "00000000" or DataOut = "11111111")
        report "Error al cargar con Shift igual a 1"
        severity error;
    wait for ciclo;
    reset <= '0';
    wait for ciclo;
    assert false
        report "Fin de la simulación"
        severity error;
END PROCESS tb;
-- *** End Test Bench - User Defined Section ***

END;
```



Recomendaciones de estilo

La descripción, además de óptima para síntesis y simulación, debe ser **legible**:

- ✓ Indentar correctamente el código
- ✓ Escribir un único comando por línea
- ✓ Comentar el significado de señales y variables, así como la funcionalidad del código
- ✓ Poner cabeceras en los ficheros (fecha, dependencias, autores, etc)
- ✓ Elegir nombres adecuados para señales, variables, entidades, procesos, paquetes, funciones, bibliotecas, etc.
- ✓ Existen guías de estilo para VHDL de dominio público: Guía PRENDA, European Space Agency: VHDL Modeling Guidelines.



Recomendaciones: modelado para síntesis

- ✓ Usar **SEÑALES** siempre que vayan a tener visibilidad fuera del proceso en el que modifica su valor
- ✓ Usar **VARIABLES** siempre que no vayan a tener visibilidad fuera del proceso en el que se declaran y se usan
- ✓ Para que una variable no genere un elemento de memoria, es imprescindible que se le asigne un valor cada vez que se ejecuta el proceso.
- ✓ Nunca se debe depender de los valores por defecto de las señales y variables.
- ✓ Nunca se deben inicializar señales y variables en la declaración
- ✓ Todas las señales de "entrada" o que se leen en el proceso deben estar en la lista de sensibilidad
- ✓ Todas las señales y variables asignadas toman un valor en cada camino de control
- ✓ Para permitir la optimización lógica se puede utilizar el valor '1'
- ✓ Es recomendable hacer la lógica secuencial con la construcción **if clk'event and clk='1'** (ó 0), no pudiendo aparecer ninguna otra condición en dicha construcción.
- ✓ Sólo debe aparecer un reloj en cada proceso secuencial. Si se tienen varios relojes, se deben poner en procesos separados.
- ✓ La construcción **if clk'event and clk='1'** (ó '0') no debe estar incluida en otra construcción (por ejemplo en un loop)
- ✓ No introducir código que genere lógica combinacional en procesos secuenciales a no ser que se utilicen variables



Recomendaciones: modelado para síntesis

- ✓ Conviene separar la parte secuencial de la combinacional en dos procesos distintos:
 - El proceso secuencial modela los registros
 - El proceso combinacional modela la asignación del estado siguiente
 - Las asignaciones de los valores de las salidas se pueden hacer en el proceso combinacional o en unas sentencias concurrentes
- ✓ Los **estados** se representarán con un **tipo específico** para cada máquina
- ✓ Las entradas y salidas de los bloques serán de tipo **std_logic** ó **std_ulogic**. Para asegurar que no se hacen conexiones indeseadas, se pueden usar tipos **std_ulogic** (no resueltos)
- ✓ En bloques de menor jerarquía se pueden utilizar tipos no estándar
- ✓ Las réplicas (instances) de componentes deben tener el mismo nombre que las entidades a las que corresponden.
- ✓ Las señales de entrada que no se vayan a usar se deben poner a un valor fijo (por ejemplo '1', para la optimización).
- ✓ Las señales de salida que no se vayan a usar se dejarán desconectadas.
- ✓ El sintetizador puede hacer una optimización más eficiente cuando el código con una misma funcionalidad está agrupado en funciones o procedimientos
- ✓ No se pueden usar llamadas recursivas
- ✓ No se deben usar dentro de subprogramas señales o variables que no hayan sido pasadas como parámetros en la llamada