Introducción al Diseño con VHDL
Parte 1/3

Dr. Juan Carlos Herrera Lozada
jlozada@ipn.mx

Este documento muestra las características básicas que cumple el diseño digital en VHDL. En esta primera parte no se abarcan tópicos como la sobrecarga de operadores, llamadas a subrutinas o creación de paquetes por usuario; que se reservan para prácticas avanzadas. Los códigos aquí listados están dirigidos a la Síntesis en ISE WebPack y/o Foundation de Xilinx, aunque por tratarse de un estándar se pueden sintetizar (con algunas restricciones) en cualquier otro software de propietario.

Caso de Estudio 1: Ejecución Serie y Ejecución Concurrente

En términos arquitecturales, una ejecución (procesamiento) serial siempre es secuencial, debido a que debe respetar un orden para no perder la lógica descrita. En una ejecución concurrente el procesamiento se resuelve de forma independiente, no importando el orden en que se describe la lógica. En este sentido cabe aclarar que la ejecución serial es un caso particular de la ejecución concurrente. En VHDL la naturaleza de la ejecución de instrucciones es concurrente, por lo que con la palabra reservada process es posible generar una ejecución secuencial a partir de un modelo de concurrencia. Analícese el **código 1** que modela un multiplexor 4 a 1, sin process. El **código 2** es una variante del anterior con ejecución secuencial.

```
-- Código 1
-- Ejecución Concurrente
library ieee;
use ieee.std_logic_1164.all;
entity mux_conc is
 port (
 I0,I1,I2,I3: in std_logic_vector(7 downto 0);
 sel: in std_logic_vector(1 downto 0);
 Y: out std_logic_vector(7 downto 0)
      );
end mux_conc;

architecture mux_arch of mux_conc is
begin
 WITH sel SELECT
   Y <= I0 when "00",
        I1 when "01",
        I2 when "10",
        I3 when others;
end mux_arch;
```

```
--Código 2
-- Ejecución Secuencial (Serie)
library ieee;
use ieee.std_logic_1164.all;
entity mux_seq is
 port (
 I0,I1,I2,I3:in std_logic_vector(7 downto 0);
 sel : in std_logic_vector(1 downto 0);
 Y : out std_logic_vector(7 downto 0)
       );
end mux_seq;

   architecture mux_arch of mux_seq is
   begin
    process (sel, I0, I1, I2, I3)
      begin
        CASE sel  IS
         when "00" =>  Y <= I0;
         when "01" =>  Y <= I1;
         when "10" =>  Y <= I2;
         when others => Y <= I3;
      end case;
    end process;
   end mux_arch;
```

Lo más recomendable es utilizar los procesos, ya que permiten modularizar el diseño utilizando menos cantidad de hardware.

Caso de Estudio 2: Descripción de Lógica Combinacional y Lógica Secuencial

VHDL no incluye alguna palabra reservada que le indique al sintetizador qué tipo de lógica se implementará; por lo mismo, la única forma de diferenciarlos es precisamente a través de las asignaciones.

Recordemos que la diferencia básica entre un circuito de lógica combinacional y uno de lógica secuencial radica en el "elemento de memoria", entonces en una ejecución concurrente es fácil observar que si la señal que está siendo asignada no interviene en el enunciado de asignación, se tiene una descripción combinacional como se muestra a continuación. En otras palabras, no existe un lazo de retroalimentación.

```
data <= b when m = '1' Else c;     -- Combinacional
data <= b when m = '1' Else data;  -- Secuencial
```

Para una ejecución serial, cuando la lista sensitiva del proceso incluye todas las señales implicadas en las asignaciones, y además todas las variables y señales que intervienen están asignadas y contemplan todos los casos en las condiciones, se sintetizará un diseño combinacional. El **código 3** modela un diseño combinacional; nótese las asignaciones y que todos los casos en la condición `if` (multiplexor) están cubiertos. Para el caso de lógica secuencial equivalente (**código 4**), el condicional `if` implementa un latch que espera por b para almacenar el valor de c y asignárselo a d.

```
-- Código 3
-- Lógica Combinacional
library ieee;
use ieee.std_logic_1164.all;
entity comb is
  port (
 a, b, c: in std_logic;
 e: out std_logic
       );
end comb;

architecture comb_arch of comb is
signal d: std_logic;
begin
 process (a, b, c, d)
   begin
     if b = '1' then d <= c;
     else d <='0';
     end if;
     e <= d and a;
   end process;
end comb_arch;
```

```
-- Código 4
-- Lógica Secuencial
library ieee;
use ieee.std_logic_1164.all;
entity seq is
  port (
 a, b, c: in std_logic;
 e: out std_logic
       );
end seq;

architecture seq_arch of seq is
signal d: std_logic;
begin
 process (a, b, c, d)
   begin
     if b = '1' then d <= c;
     end if;
     e <= d and a;
   end process;
end seq_arch;
```

Los diseños secuenciales pueden obtener sus derivaciones clásicas (sincronismo y asincronismo) a través de las variables en la lista sensitiva. Por ejemplo, al incluir una señal de reloj el diseño implica lógica secuencial síncrona, tal y como se verá en algunos ejemplos posteriores.

Caso de Estudio 3: Descripciones Intuitivas

Al tratarse de un lenguaje, es posible obtener soluciones variadas a un mismo problema. La intención única no es que funcione el diseño; sino que funcione óptimamente, con el mínimo de recursos dentro del dispositivo programable (GAL, CPLD, FPGA, etc.), y que adicione versatilidad (que sea fácilmente modificable para diseños posteriores). Por lo anterior, es imprescindible conocer la estructuración y proponer varias soluciones al mismo problema, evaluando todas las matices. A continuación, el **código 5** modela un comparador de dos números de 8 bits. El **código 6** es el equivalente al modelo síncrono.

```
-- Código 5
-- Comparador de 8 bits
library ieee;
use ieee.std_logic_1164.all;
entity comp_s is
port (
A, B: in STD_LOGIC_VECTOR(7 downto 0);
Menor, Igual, Mayor: out STD_LOGIC
       );
end comp_s;

architecture comp_s_arch of comp_s is
begin
process(A,B)
begin
Menor <= '0';
Igual  <= '0';
Mayor <= '0';
    if ( A < B ) then Menor <= '1';
    elsif (A = B) then Igual <= '1';
    else Mayor <= '1';
    end if;
end process;
end comp_s_arch;
```

```
-- Código 6
-- Comparador de 8 bits, diseño síncrono
library ieee;
use ieee.std_logic_1164.all;
entity comp is
port (
CLK, RESET: in STD_LOGIC;
A, B: in STD_LOGIC_VECTOR(7 downto 0);
Menor, Igual, Mayor: out STD_LOGIC
       );
end comp;

architecture comp_arch of comp is
begin
process(CLK,RESET)
begin
if (RESET = '1') then
Menor <= '0';
Igual  <= '0';
Mayor <= '0';
   ELSIF (CLK'event and CLK ='1') then
    if ( A < B ) then Menor <= '1';
    elsif (A = B) then Igual <= '1';
    else Mayor <= '1';
    end if;
end if;
end process;
end comp_arch;
```

Caso de Estudio 4: VHDL Para Síntesis

Los siguientes códigos están escritos para sintetizarse directamente, por lo cual no requieren cambios drásticos para adaptarse a alguna otra solución.

```
-- Código 7
-- Contador de 4 bits, ascendente/descendente, con carga, chip enable y reset.
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
ENTITY conta IS
  PORT(
         CLK, RESET, CE, LOAD, DIR: in STD_LOGIC;
         DIN: in STD_LOGIC_VECTOR (3 downto 0);
         COUNT: inout STD_LOGIC_VECTOR (3 downto 0));
  END conta;

ARCHITECTURE mi_cont OF conta IS
```

```
begin
process (CLK, RESET)
begin
   if RESET='1' then
      COUNT <= "0000";
   elsif CLK='1' and CLK'event then
      if LOAD='1' then
         COUNT <= DIN;
      else
         if CE='1' then
            if DIR='1' then
               COUNT <= COUNT + 1;
            else
               COUNT <= COUNT - 1;
            end if;
         end if;
      end if;
   end if;
end process;

END mi_cont;
```

```
-- Código 8
-- Registro de 8 bits, con reset asíncrono y load síncrona
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
  port (  clk,reset:  in std_logic;
          sload: in std_logic;
          din   : in std_logic_vector(7 downto 0);
          dout  : out std_logic_vector(7 downto 0)
   );
end reg8;

architecture arch_reg of reg8 is
  signal  reg_state: std_logic_vector(7 downto 0);
begin
  dout <= reg_state;
  process(clk)
     begin
     if (reset = '1') then
        -- reset asíncrono
        reg_state <= "00000000";
     elsif (clk'event and clk='1') then
       -- load síncrona
       if (sload = '1') then
          reg_state <= din;
       end if;
     end if;
  end process;
end arch_reg;
```

```
--Código 9
--Decodificador 3 a 8
library ieee;
use ieee.std_logic_1164.all;
entity dec3to8 is
port
(
 sel: in  std_logic_vector(2 downto 0);
 en: in  std_logic;
 y: out std_logic_vector(7 downto 0)
);
end dec3to8;

architecture arch_dec of dec3to8 is
begin
  process (sel,en)
  begin

    y <= "11111111";
    if (en = '1') then
```

```
       case sel is
        when "000" => y(0) <= '0';
         when "001" => y(1) <= '0';
         when "010" => y(2) <= '0';
         when "011" => y(3) <= '0';
         when "100" => y(4) <= '0';
         when "101" => y(5) <= '0';
         when "110" => y(6) <= '0';
         when "111" => y(7) <= '0';
        when others =>  y(7) <= '0';
       end case;
    end if;
  end process;
end arch_dec;
```

```
-- Código 10
-- Máquina de estados que implementa una cerradura electrónica
-- La secuencia reconocida es : 0, 1, 2
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
ENTITY DET_SEC IS
  PORT(
         CLK, RESET, X: in STD_LOGIC;
         Z: out STD_LOGIC);
END DET_SEC;

ARCHITECTURE DETECTA OF DET_SEC IS
type state_type is (CERO, UNO, DOS, TRES);
signal estado, estado_siguiente: state_type;
begin
registra:process (CLK, RESET)
begin
   if RESET='1' then
      estado  <= CERO;
   elsif CLK='1' and CLK'event then
      estado <= estado_siguiente;
      --end if;
   end if;
end process registra;

fun_estado_siguiente: process (X, estado)
begin
case estado is
when CERO => if X='1' then
estado_siguiente <= UNO;
else
estado_siguiente <= CERO;
end if;

when UNO => if X='1' then
estado_siguiente <= DOS;
else
estado_siguiente <= CERO;
end if;

when DOS => if X='1' then
estado_siguiente <= TRES;
else
estado_siguiente <= CERO;
end if;

when TRES => if X='1' then
estado_siguiente <= CERO;
else
estado_siguiente <= CERO;
end if;
end case;
end process fun_estado_siguiente;

salida: process (X, estado)
begin
case estado is
```

```
when CERO => Z <= '0';
when UNO => Z <= '0';
when DOS => Z <= '0';
when TRES => Z <= '1';
when others => null;
end case;
end process salida;
end detecta;
```

```
-- Código 11.
-- ALU de 4 bits, con corrección al acarreo de salida generado en la adición.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity alu4 is
    port (
        a, b: in STD_LOGIC_VECTOR (3 downto 0);
        SEL: in STD_LOGIC_VECTOR (2 downto 0);
        dato: out STD_LOGIC_VECTOR (7 downto 0)
    );
end alu4;

architecture alu4_arch of alu4 is
signal d: STD_LOGIC_VECTOR (7 downto 0);
signal carry_in: STD_LOGIC;
signal an, bn: STD_LOGIC_VECTOR (4 downto 0);
begin
process (a, b, SEL)
begin
d <= "00000000";
carry_in <= '0';
an(4) <= carry_in;
bn(4) <= carry_in;
an(3 downto 0) <= a;
bn(3 downto 0) <= b;
case SEL is
               when "000" => d(4 downto 0) <= an+bn;
               when "001" => d(3 downto 0) <= a-b;
               when "010" => d <= a*b;
               when "011" => d(3 downto 0) <= a or b;
               when "100" => d(3 downto 0) <= a and b;
               when "101" => d(3 downto 0) <= a xor b;
               when "110" => d(3 downto 0) <= not a;
               when "111" => d(3 downto 0) <= a;
               when others => NULL;
       end case;
dato <= d;
end process;
end alu4_arch;
```