UNIVERSIDAD NACIONAL DE LA PLATA

E1212 - TÉCNICAS DIGITALES

TRABAJO PRÁCTICO Nº 2

FACULTAD DE INGENIERÍA

Introducción al Lenguaje de Descripción de Hardware VHDL

Ejercicio 1

Desarrolle las siguientes cuestiones:

1. ¿Qué significa HDL y cuál es su propósito?
2. ¿Cuál es la diferencia clave entre un HDL y un lenguaje de programación de computadoras?
3. ¿Qué significa VHDL y qué usos tiene?
4. ¿En qué consiste la estructura básica de un archivo VHDL?
5. ¿Cuál es la función de la declaración ENTITY?
6. ¿Qué es la librería IEEE std_logic_1164? ¿Cuál es su importancia?
7. ¿Qué son los tipos *std_logic* y *std_ulogic*? ¿Y el *std_logic_vector*? ¿En qué casos conviene usar uno u otro? ¿Qué tipo de niveles son útiles para síntesis?
8. ¿Qué sección clave de un archivo VHDL define la operación de un circuito digital?
9. ¿Qué son las asignaciones concurrentes? ¿Y las secuenciales? ¿Qué las diferencia? Proponer ejemplos de uso.
10. ¿Qué es un *process*? ¿Cuál es la importancia de la lista de sensibilidad de un *process*?
11. ¿Cuál es la importancia en el orden en el que se escriben las sentencias concurrentes en VHDL?
12. ¿Cómo se introducen comentarios en VHDL?
13. ¿Qué son las señales (*signals*)? ¿Qué utilidad tienen?
14. ¿Qué es el operador de concatenación "&"? ¿Qué utilidad tiene?
15. ¿Qué función realiza la sentencia *(others => '0')*? ¿Cómo se utiliza?

Ejercicio 2

1. Codificar de forma concurrente en VHDL las siguientes funciones lógicas y simularlas con Modelsim. Analizar el esquema RTL generado por el Quartus II. ¿Es el que hubiese esperado?
2. Implementar algunas de ellas utilizando la estructura secuencial *process.* ¿Qué diferencias se pueden apreciar entre una forma de implementación y otra?

(a) $\mathrm{T} = \mathrm{HKL} + \mathrm{K}\bar{L}$

(b) $H = \mathrm{QWE} + \overline{\bar{Q} + W} + \overline{QE}$

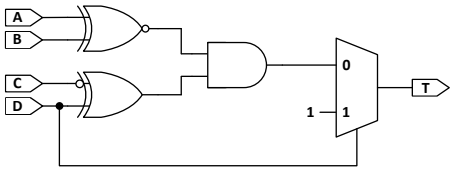
(c) $R = \overline{\overline{A + B} + \overline{\bar{C} + A}}$

(d) $O = \overline{\bar{A} + \overline{B \oplus \bar{C}}} + \bar{A} \oplus C$

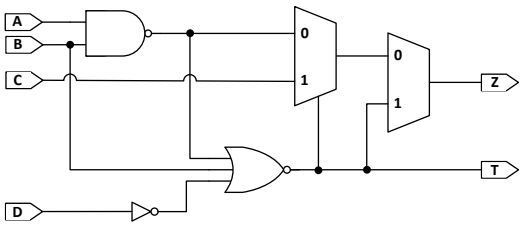
Ejercicio 3

1. Codificar en VHDL de forma concurrente los siguientes sistemas sin optimizarlos.
2. Repetir la implementación utilizando *process.*

3. En todos los casos analizar el esquema RTL generado por Quartus II. ¿Son los que hubiese esperado?



(a)



(b)

Ejercicio 4

Los siguientes códigos en VHDL describen el mismo sistema lógico pero de diferente manera.

1. Deducir la función lógica que codifican y realizar el diagrama esquemático a mano.
2. Generar el esquema RTL utilizando Quartus II. ¿Coincide lo que se observa con lo deducido en el inciso anterior? ¿A qué se debe lo que se observa? ¿Lo que surge de los esquemas puede representar algún problema? Verificar los mensajes de *warning* en el reporte de compilación de Quartus II al hacer el "Análisis y Síntesis". Investigar cómo evitar lo sucedido en cada caso.

(a)

```
…
dataOut <= dataInA when (sel0 = '0' and sel1 = '1') else dataInB when (sel0 = '1' and sel1 = '0');
…
```

(b)

```
…
pr0: process (sel0, sel1, dataInA, dataInB) is
begin
    if (sel0 = '0' and sel1 = '1') then
        dataOut     <= dataInA;
    elsif (sel0 = '1' and sel1 = '0') then
        dataOut     <= dataInB;
    end if;
end process;
…
```

Ejercicio 5

Dados los siguientes códigos VHDL donde todos los nombres que aparecen son puertos de entrada o de salida.

1. Realizar la tabla de verdad de todas las implementaciones.
2. Compilar en Quartus II cada uno de los casos y generar el esquema RTL. Con lo que se observa, ¿puede adelantarse si estos esquemas cumplen con las tablas de verdad del inciso anterior?
3. Cómo se explican y a qué se deben las diferencias que surgen entre:
   - 3.1 Las implementaciones (a) y (b).
   - 3.2 Las implementaciones (b) y (c).
   - 3.3 Las implementaciones (c) y (d).
4. ¿Qué ocurre si en la implementación (d) la línea etiquetada con el comentario "L0" se mueve a la línea etiquetada con el comentario "Ref1"? Generar el esquema RTL y justificar lo observado.

```
…
pr0: process (ctrl,rst,dataInA,dataInB) is
begin

   if rst = '1' then
         dataOut   <= '1';
   elsif ctrl(1) = '1' or ctrl(0) = '0' then
         dataOut   <= dataInB;
   elsif (ctrl(1) xor ctrl(0)) = '1' then
         dataOut   <= dataInA;
   else
         dataOut   <= '0';
   end if;

end process;
…
```

(a)

```
…
pr0: process (ctrl,rst,dataInA,dataInB) is
begin

   dataOut         <= '0';

   if (ctrl(1) xor ctrl(0)) = '1' then
         dataOut   <= dataInA;
   end if;

   if ctrl(1) = '1' or ctrl(0) = '0' then
         dataOut   <= dataInB;
   end if;

   if rst = '1' then
         dataOut   <= '1';
   end if;

end process;
…
```

(b)

```
…
pr0: process (ctrl,rst,dataInA,dataInB) is
begin

   dataOut         <= '0';

   if ctrl(1) = '1' or ctrl(0) = '0' then
         dataOut   <= dataInB;
   end if;

   if (ctrl(1) xor ctrl(0)) = '1' then
         dataOut   <= dataInA;
   end if;

   if rst = '1' then
         dataOut   <= '1';
   end if;

end process;
…
```

(c)

```
…
pr0: process (ctrl,rst,dataInA,dataInB) is
begin

   if ctrl(1) = '1' or ctrl(0) = '0' then
         dataOut   <= dataInB;
   end if;

   dataOut         <= '0'; -- L0

   if (ctrl(1) xor ctrl(0)) = '1' then
         dataOut   <= dataInA;
   end if;

   if rst = '1' then
         dataOut   <= '1';
   end if;
-- Ref1
end process;
…
```

(d)

Ejercicio 6

Dado el siguiente código VHDL:

1. Deducir qué función realiza el sistema.
2. Recodificarlo utilizando un *process*. ¿Puede lograrse alguna simplificación?
3. Verificar por simulación que ambas implementaciones presentan el mismo comportamiento.

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

entity TP2E6 is
  port (
         ctrlS     : IN      std_logic_vector(2 downto 0);
         ctrlD     : IN      std_logic;

         dataIn    : IN      std_logic_vector(7 downto 0);

         dataOut   : OUT     std_logic_vector(7 downto 0)
  );
end TP2E6;

architecture RTL of TP2E6 is

signal dataOutL : std_logic_vector(7 downto 0);
signal dataOutR : std_logic_vector(7 downto 0);

begin

dataOutL <= dataIn(7 downto 1) & '0'   when (ctrlD = '1' and ctrlS    = "001") else
            dataIn(7 downto 2) & "00"  when (ctrlD = '1' and ctrlS    = "010") else
            dataIn(7 downto 3) & "000" when (ctrlD = '1' and ctrlS    = "011") else
            dataIn                     when (ctrlD = '1' and ctrlS    = "000") else
            X"00"                      when (ctrlD = '1' and ctrlS(2) = '1'  ) else
```

```
            (others => '0');


with ctrlD & ctrlS select
   dataOutR <=  '0'   & dataIn(7 downto 1)    when '0' & "001",
                "00"  & dataIn(7 downto 2)    when '0' & "010",
                "000" & dataIn(7 downto 3)    when "0011",
                "00000000"                    when "0100" | "0101" | "0110" | "0111",
                (others => '0')               when others;

dataOut  <= dataOutL when ctrlD = '1' else dataOutR;

end RTL;
```

Ejercicio 7

Un decodificador BCD-7 Segmentos permite comandar un display 7-segmentos de forma sencilla simplemente escribiendo en formato binario el número que se pretende visualizar en el display. Un ejemplo comercial de este tipo de dispositivos es el CD4511BC.

1. Analizar la hoja de datos del circuito integrado propuesto.
2. Descartando las señales de control (LE, BI, LT) y teniendo en cuenta únicamente las entradas (A, B, C, D) y las correspondientes salidas de cada segmento, implementar en VHDL el decodificador utilizando la estructura concurrente más adecuada.
3. Repetir la implementación pero utilizando una estructura secuencial (*process*).
4. ¿Cómo debería recodificar la implementación en cualquiera de los casos si los segmentos fuesen activos en bajo? ¿Hay una forma eficiente de realizar esta operación?
5. Modificar ambas implementaciones para incluir una señal de control denominada "*segPol*" que permita seleccionar la polaridad de encendido de los segmentos de la siguiente forma: Si *segPol* es '0' los segmentos serán activos en bajo; caso contrario serán activos en alto.

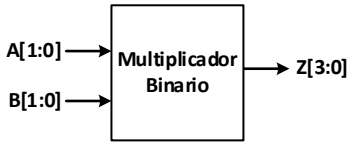
Ejercicio 8

En base al *barrel shifter* diseñado en la Práctica Nº 1:

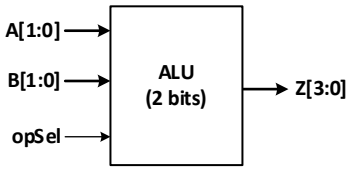
1. Implementarlo en VHDL utilizando una estructura concurrente.
2. Repetir para una estructura secuencial.

Ejercicio 9

1. En base al multiplicador binario sin signo de dos bits diseñado en la Práctica Nº 1:
   1.1 Implementarlo en VHDL utilizando las funciones lógicas obtenidas en la Práctica Nº 1.
   1.2 Proponer alternativamente una segunda implementación que lo haga directamente a partir de la tabla de verdad.



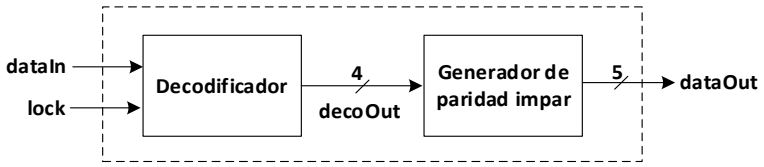
2. Extender la funcionalidad del diseño anterior incluyendo la posibilidad de sumar los operandos. El nuevo diseño deberá incluir una señal de control denominada "*opSel*" que permita seleccionar la operación a realizar de la siguiente forma: si *opSel* es '1' la salida Z deberá ser igual a la suma de los operandos y si es igual a '0' la salida Z deberá ser el producto de los mismos.



NOTA: Este tipo de módulos de procesamiento son los que se denomina comúnmente como Unidad Aritmético Lógica (ALU) y son de uso muy extendido en ámbito de diseño digital.

Ejercicio 10

En un dado sistema lógico debe implementarse un módulo que procese ciertos datos de interés para otra parte del sistema. El módulo está compuesto por un decodificador y un generador de paridad para aumentar la fiabilidad del sistema. El decodificador tiene la tabla de verdad que se muestra más abajo y, además, permite restringir el acceso a cierto rango de datos mediante el uso de la señal de control "lock". Implementar el módulo en VHDL respetando el diagrama en bloques siguiente y simular el sistema utilizando Modelsim.



Puerto	I/O	Descripción
dataIn	I	Dato de entrada
lock	I	Bloqueo del área restringida 0: El decodificador opera normalmente en todo su rango. 1: El decodificador impide la decodificación de las direcciones 5 a 7. En caso de querer acceder a dicho rango cuando lock=1 el decodificador debe devolver el valor de todos "0" (ceros) a la salida. Las direcciones de 0 a 4 no se ven afectadas por el valor de lock.
dataOut	O	Dato de salida

Tabla de verdad del decodificador	
dataIn (en decimal)	**decoOut (en binario)**
0	0111
1	0001
2	1000
3	0000
4	0011
5	X101
6	1010
7	11X1

X: Don't care

NOTA: Un bit de paridad impar debe ser "1" si la cantidad de unos en el resto de la palabra es par. Por ejemplo, si la palabra de datos es en binario "011" el bit de paridad impar que debe agregarse en la posición más significativa debe ser "1", de forma que la cantidad total de "1" sea impar. La palabra final quedará entonces "1011". Si, por ejemplo, la palabra en binario fuese "111" el bit de paridad deberá ser "0" porque la cantidad de unos ya es impar. La palabra quedará entonces "0111".