UNIVERSIDAD DE COLIMA

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

CURSO: IMPLEMENTACION DE CIRCUITOS DIGITALES

ACTIVIDAD 1: Introducción al Lenguaje VHDL

Profesor: Dr. Alberto Ochoa Brust

Esta actividad está planeada con un enfoque de aprendizaje virtual utilizando las TICs y las herramientas mencionadas en Classroom, en la cual lo más importante es el seguir la teoría y los ejemplos con el objetivo de aprender significativamente. Para ello utilizaras el ISE instalado en tu computadora para realizar los ejercicios planteados, y seguir paso a paso las instrucciones que se indican aquí. Presenta tus prácticas mediante la plataforma Classroom, sube en un solo documento (.PDF) los archivos fuente de tus programas y el reporte de la práctica, la cual se ponderara según al avance logrado al final de la sesión.

El VHDL fue desarrollado de forma muy parecida al lenguaje de programación ADA debido a que éste tiene una orientación hacia sistemas en tiempo real y al hardware en general, por lo que se le escogió como modelo para desarrollar el VHDL.

VHDL proviene de los acrónimos VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language); es un lenguaje de descripción y modelado, diseñado para describir la funcionalidad y la organización de sistemas de hardware digital.

Algunas ventajas del uso de VHDL para la descripción hardware son:

- VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción, hasta el nivel de definición estructural de puertas.
- Los diseños son independientes con respecto a la tecnología de donde se implementan (ASIC, Gate Array, FPGA, CPLD, etc.) y de la compañía que la fabrica (Intel, Xilinx, Altera, etc.).
- Promueve una depuración más fácil del diseño, sobre todo en diseños muy complejos.
- Facilita diseñar incrementalmente, es decir, diseñar, implementar y probar separadamente las partes del diseño.
- Permite el diseño concurrente o modular; varias personas pueden trabajar al mismo tiempo en las partes de un diseño.

Un diseño lógico descrito con VHDL se compone de dos módulos: la entidad y la arquitectura. En la entidad (ENTITY) se definen las entradas, salidas, aquí es donde se declaran los puertos del circuito, dirección (IN, OUT, INOUT) y el tipo de dato (INTEGER, BIT, STD_LOGIC, BOOLEAN, etc.), es decir, la manera en que se ve el circuito desde fuera; su sintaxis es como se muestra en la Figura 1.1.

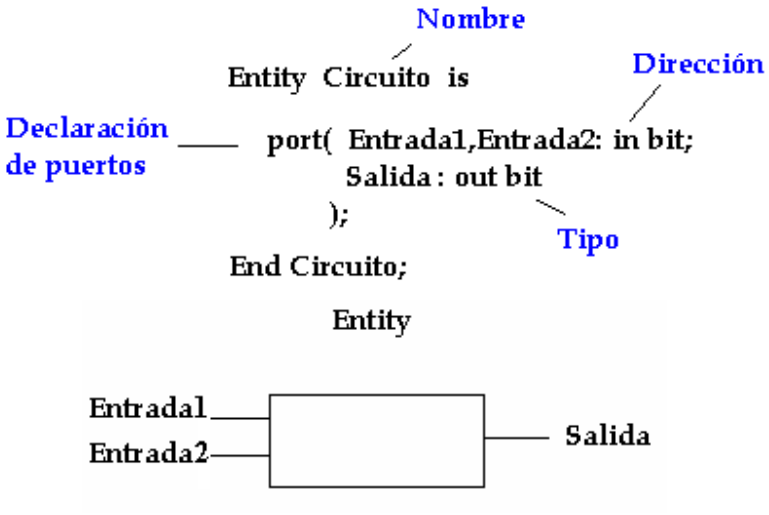

Figura 1.1: Representación gráfica de una entidad.

En la arquitectura (**ARCHITECTURE**) es donde se describe el comportamiento que tendrá el circuito, esto es, lo que hará con las señales de entrada para generar las señales de salida, aquí es donde se define el funcionamiento interno del circuito.

La declaración de la arquitectura debe constar de las siguientes partes como mínimo, aunque suelen ser más:

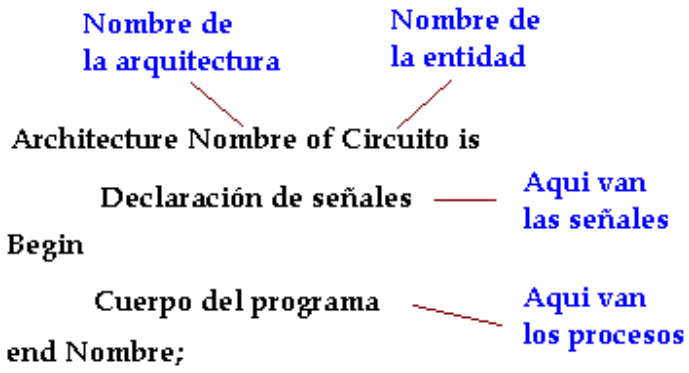

Figura 1.2. Cuerpo de la arquitectura.

EJEMPLO 1.1: Descripción de una compuerta or2_b1

El siguiente código en VHDL describe el comportamiento de una compuerta OR de 2 entradas, con una de ellas negadas: **(2 Puntos)**

```
-- Los comentarios inician con dos guiones
ENTITY or2_b1 IS
  PORT(entrada1: IN BIT; -- Puerto de entrada
       entrada2: IN BIT; -- Puerto de entrada
       salida: OUT BIT); -- Puerto de salida
END or2_b1;

ARCHITECTURE behavioral OF or2_b1 IS
  SIGNAL entrada_b: BIT; --Señal interna
BEGIN
  -- A la señal entrada_b se le asigna en negado del puerto entrada2
  entrada_b <= NOT entrada2;

  -- Al puerto salida se le asigna en la OR entre el puerto entrada2 y la señal entrada_b
  salida <= entrada1 OR entrada_b;
END behavioral;
```

La entidad *or2_b1*, que es el nombre dado al circuito, tiene 2 puertos de entrada (*entrada1* y *entrada2*) de tipo BIT y un puerto de salida (*salida*) también del tipo BIT. Los tipos de datos en VHDL, baste por el momento saber que el tipo BIT indica que el puerto o señal puede tomar los valores '0' ó '1'.

La entidad de un circuito es única, sin embargo, en VHDL puede haber varias arquitecturas relacionadas con la misma entidad; su utilidad se verá en el futuro. La arquitectura de este ejemplo se llama *behavioral* y está asociada con la entidad *or2_b1*, después de esta definición sigue la parte declarativa de la arquitectura, donde se declaran las señales internas (SIGNAL), contantes (CONSTANT) y tipos de datos definidos por el programador (TYPE). En este caso sólo tenemos una señal interna llamada *entrada_b* de tipo BIT, a la cual se tiene acceso, únicamente, en el interior del circuito. Luego sigue la sentencia BEGIN, que indica que las siguientes líneas de código deben ser interpretadas como instrucciones, las cuales deben terminar en punto y coma (;). La primera de las instrucciones es una asignación concurrente; a la señal *entrada_b* se le asigna concurrentemente en valor negado del puerto entrada2.

El operador de asignación concurrente “<=” es muy importante en el VHDL, puede jugar el papel de “alambre” o “wire”, análogamente con los diagramas con esquemáticos, en la interconexión de componentes; en otro contexto, el símbolo “<=” también puede interpretarse como un operador de comparación “menor o igual que”, pero esto se analizará en el futuro. Del lado

izquierdo del operador debe estar una señal interna o un puerto de salida y del lado derecho una operación válida entre señales internas o puertos de entrada. El hecho de que sea un operador concurrente implica que no tenga importancia el lugar donde sea es usado este operador, puede ser al principio, en medio o al final, pero después del BEGIN y antes del fin de la arquitectura. Esta es una de las características que causan más confusión cuando se empieza a programar en VHDL, ya que en un lenguaje convencional de programación, el lugar donde se declare una asignación de variable es trascendental para el funcionamiento del programa. Por ejemplo el siguiente código en lenguaje *C*:

```
entrada_b = ~ entrada2;
salida = entrada1 | entrada_b;
```

Produce resultados muy diferentes al código:

```
salida = entrada1 | entrada_b;
entrada_b = ~ entrada2;
```

Sin embargo, en VHDL el siguiente código hace la misma función que el del diseño or2_b1, a pesar que se cambien de lugar las instrucciones concurrentes:

```
salida <= entrada1 OR entrada_b;
entrada_b <= NOT entrada2;
```

La segunda instrucción también es una asignación concurrente; al puerto salida se le asigna la OR lógica entre el puerto entrada2 y la señal entrada_b. En la Figura 1.3 se puede ver una representación gráfica del circuito descrito por el código.

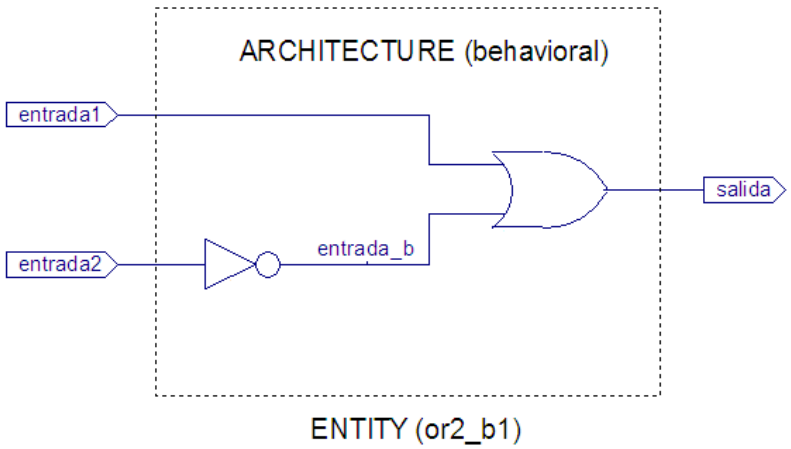

Figura 1.3: Representación en esquemáticos del diseño del ejemplo 1.1.

El operador NOT tiene la función lógica de negar el contenido de operando que esté después de él. Como es de esperarse el operador OR regresa dicha operación lógica de los operandos que estén entre él. En VHDL también están definidos los operadores lógicos: AND, NAND, NOR y XOR. Todos los operadores lógicos tienen la misma precedencia, por lo tanto muchas veces será necesario usar paréntesis para especificar correctamente la operación deseada. El funcionamiento es el habitual para este tipo de operadores. VHDL es muy delicado en cuanto los tipos de datos y los operadores permitidos, por ejemplo los operadores lógicos no están definidos para tipos de datos numéricos (INTEGER, REAL), por otro lado, los operadores aritméticos (+, –, *, /) no están permitidos para datos de tipo BIT.

EJEMPLO 1.2: Descripción de una compuerta and4_b3

El siguiente código en VHDL describe el comportamiento de una compuerta AND de 4 entradas, con 3 de ellas negadas: **(2 Puntos)**

```
ENTITY and4_b3 IS
  PORT(a: IN BIT; -- Puerto de entrada
       b: IN BIT; -- Puerto de entrada
       c: IN BIT; -- Puerto de entrada
       d: IN BIT; -- Puerto de entrada
       z: OUT BIT); -- Puerto de salida
END and4_b3;

ARCHITECTURE comportamiento OF and4_b3 IS
BEGIN
  z <= a AND (NOT b) AND (NOT c) AND (NOT d);
END comportamiento;
```

La entidad and4_b3, tiene 4 puertos de entrada (a, b, c, d) de tipo BIT y un puerto de salida (z) del mismo tipo. La arquitectura de este ejemplo se llama comportamiento y está asociada con la entidad and4_b3. En este caso no usamos ninguna señal interna, en una sola asignación concurrente realizamos todas las operaciones lógicas, usando paréntesis para especificar correctamente dichas operaciones. En la Figura 1.4 se aprecia una representación en diagramas esquemáticos del código.

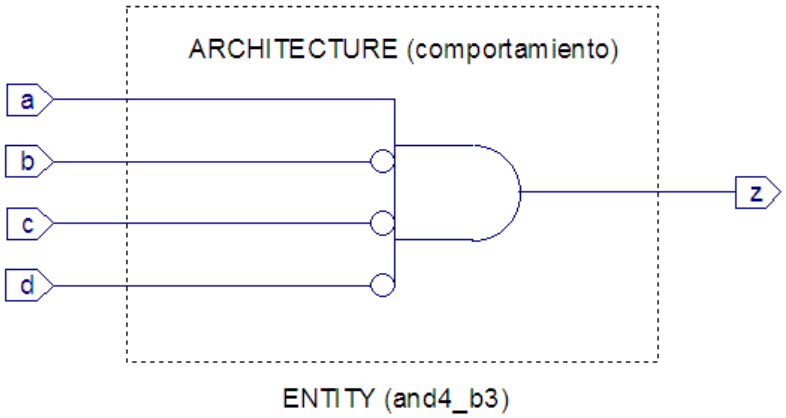

Figura 1.4: Representación en esquemáticos del diseño del Ejemplo 1.2.

En el futuro sustituiremos el tipo BIT por el tipo STD_LOGIC, debido a que éste es más adecuado para la simulación y síntesis de circuitos lógicos. Para poder usar este tipo de dato es necesario incluir la librería IEEE y hacer visible el paquete STD_LOGIC_1164, para ello se deben escribir este par de línea de código antes de definir la entidad:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

Las características del tipo STD_LOGIC, del estándar IEEE 1164-1993, se analizarán en futuras actividades.

PROBLEMA 1.1: Simulación de un diseño en VHDL. (2 Puntos)
El siguiente código en VHDL describe el comportamiento de un circuito dado:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY circuito IS
  PORT(entrada: IN STD_LOGIC; -- Puerto de entrada
       linea: IN STD_LOGIC; -- Puerto de entrada
       salida0: OUT STD_LOGIC; -- Puerto de salida
       salida1: OUT STD_LOGIC); -- Puerto de salida
END circuito;

ARCHITECTURE behavioral OF circuito IS
BEGIN
  salida0 <= entrada AND (NOT linea);
  salida1 <= entrada AND linea;
END behavioral;
```

Dibuje las señales de los puertos salida0 y salida1, en el "Test Bench" de la Figura 1.5.

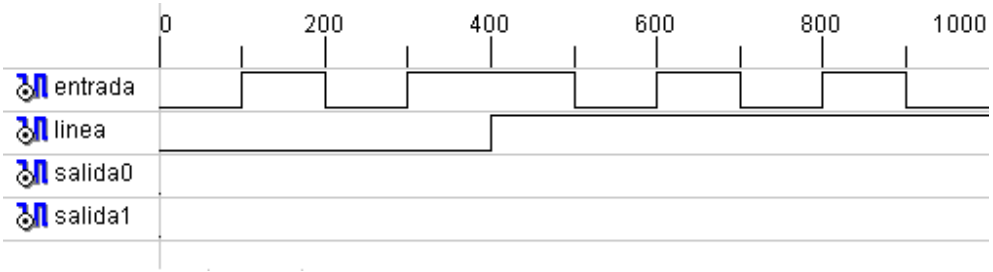

Figura 1.5: Cama de pruebas (Test Bench) para el problema 1.1.

PROBLEMA 1.2: (2 Puntos)
Diseñe un multiplexor de 2 a 1 en VHDL, como el hecho en las actividades pasadas, utilice el tipo STD_LOGIC en lugar del tipo BIT. Simule el diseño en el entorno ISE e impleméntelo en un CPLD.

PROBLEMA 1.3: (2 Puntos)
Diseñe un decodificador de 2 a 4 en VHDL, utilice el tipo STD_LOGIC en lugar del tipo BIT. Simule el diseño en el ISE e impleméntelo en el CoolRunner-II Starter o en una BASYS. Dicho diseño deberá tener una "Entity" con dos puertos de entrada (S0 y S1) y 4 se salida (Y0, Y1, Y2, Y3), que se relacionarán con la siguiente tabla de verdad:

Tabla I. Tabla de verdad de un decodificador de 2 a 4 líneas.

S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0