Práctica Nº 2. Estructura de módulos (Instancias)

1. Datos de la práctica

Carrera	INGENIERÍA ELECTRÓNICA		
Semestre		Grupo	
Tipo de Práctica	☐ Laboratorio ☐ Simulación	Fecha	
Asignatura	Electrónica Digital I		
Unidad Temática			
Nº Alumnos por práctica	2	Nº Alumnos por reporte	2
Nombre del Profesor			
Nombre(s) de Alumno(s)	1. 2.		
Tiempo estimado		Vo. Bo. Profesor	
Comentarios			

2. Objetivo

Comprender la estructura de módulos en el lenguaje VHDL y la forma de conexión hacia otros módulos mediante un ejemplo de lógica combinacional.

3. Medios a utilizar

Por cada práctica y por cada puesto de laboratorio, los materiales a utilizar es:

Cantidad	Descripción
1	Computadora
1	Tarjeta de desarrollo Basys2 Digilent
1	Software Xilinx ISE® Webpackv14.7

4. Introducción

En VHDL un sistema digital está compuesto por la interconexión de un conjunto de módulos. Estos son unidades lógicas donde se puede especificar la descripción de un circuito digital, sea sencillo o complejo, de tal manera que puedan utilizarse para construir diseños de mayor complejidad, creando lo que se denomina un *diseño con jerarquía*. En esta práctica aprenderemos la estructura básica de un módulo y la forma en cómo se conecta a otros módulos.

5. Actividades previas

- Tabla de verdad del módulo ANDOR (Figura 1).

A	B	C	F

6. Desarrollo de la práctica

Crear un nuevo proyecto ISE®

En esta práctica vamos a crear un módulo llamado ANDOR con tres entradas (**A**, **B** y **C**) y una salida (**F**), después usaremos dos veces este mismo módulo en otro módulo superior llamado LOGIC.

Importante: Seguir las instrucciones del laboratorio 1 para crear un nuevo proyecto en ISE®.

1. Cerrar cualquier proyecto que esté abierto y crear un proyecto nuevo llamado ANDOR. El módulo debe tener 3 entradas (**A, B, C**) y una salida (**F**).

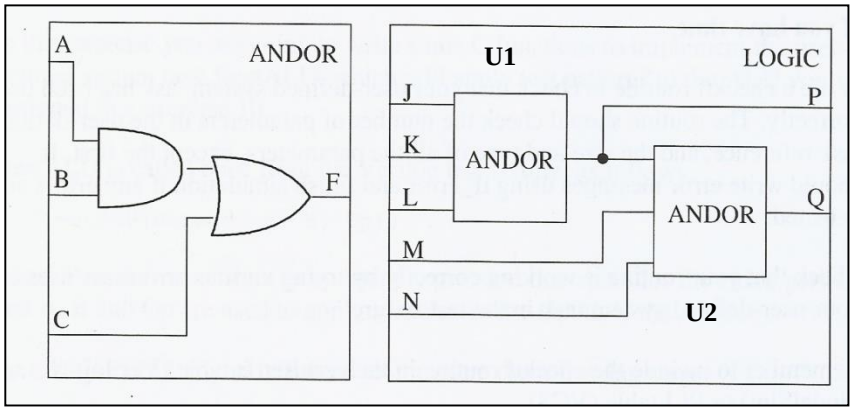
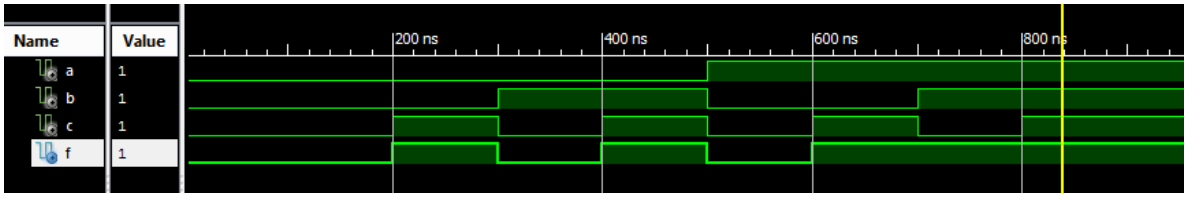


Figura 1. Diagrama de los módulos ANDOR y LOGIC.

2. Contenido del archivo ANDOR mostrado a continuación:

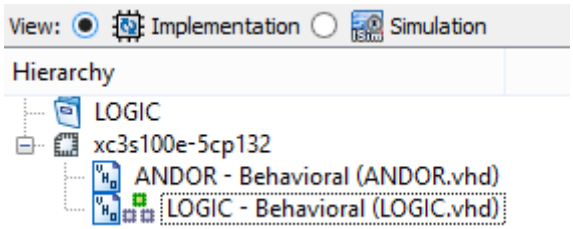
```
entity ANDOR is
port(
     A    :    in std_logic;
     B    :    in std_logic;
     C    :    in std_logic;
     F    :    out std_logic
);
end ANDOR;
architecture Behavioral of ANDOR is
begin
F <= (A and B) or C;
end Behavioral;
```

3. Realizar la simulación adecuada para comprobar el resultado del archivo ANDOR. Para probarlo tienen que generar las posibles combinaciones para las 3 entradas.
4. Guardar los cambios y correr el programa. El resultado de la Simulación se muestra a continuación en la figura.



Creación de Instancias en VHDL

5. Una vez creado el módulo ANDOR, este será utilizado para un archivo top que se encontrara en un nuevo proyecto, el cual instanciará dos veces ese modulo para formar uno mayor.
6. Realizar un nuevo proyecto llamado LOGIC. El módulo debe tener 5 entradas (**J, K, L, M, N**) y dos salidas (**P, Q**).
7. Agregar el código ANDOR.vhd al proyecto LOGIC, es decir quedaran dos archivos .vhd al proyecto a como se ve en la imagen.



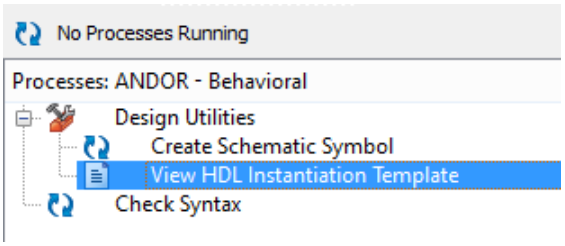
8. El archivo LOGIC para este proyecto servirá de archivo top dentro de la jerarquía de archivos, la función de este será instanciar (llamar) a los demás archivos. Para hacer esto, miraremos la estructura del código de LOGIC.

```
entity LOGIC is
    Port ( J : in  STD_LOGIC;
           K : in  STD_LOGIC;
           L : in  STD_LOGIC;
           M : in  STD_LOGIC;
           N : in  STD_LOGIC;
           P : out  STD_LOGIC;
           Q : out  STD_LOGIC);
end LOGIC;
architecture Behavioral of LOGIC is


begin


end Behavioral;
```

Acá se agregan los componentes

Acá se instancian los archivos que se utilizan en el proyecto

9. Para poder instanciar el código ANDOR se utiliza una opción del ISE que generara el código referente a ese archivo. Se selecciona el módulo ANDOR, luego en el desplegable Design Utilities se ejecutara la opción View HDL Instantiation Template a como se muestra en la imagen.



10. El código generado se muestra a continuación y se deberá insertar en el código de LOGIC en sus respectivas áreas: Componentes e Instancias.

```
    COMPONENT ANDOR
    PORT(
         A : IN std_logic;
         B : IN std_logic;
         C : IN std_logic;
         F : OUT std_logic
         );
    END COMPONENT;

    Inst_ANDOR: ANDOR PORT MAP(
         A => ,
         B => ,
         C => ,
         F =>
    );
```

11. El código anterior se insertará en el archivo LOGIC quedando de la siguiente manera:

```
entity LOGIC is
    Port ( J : in  STD_LOGIC;
           K : in  STD_LOGIC;
           L : in  STD_LOGIC;
           M : in  STD_LOGIC;
           N : in  STD_LOGIC;
           P : out  STD_LOGIC;
           Q : out  STD_LOGIC);
end LOGIC;

architecture Behavioral of LOGIC is
     COMPONENT ANDOR
     PORT(
          A : IN std_logic;
          B : IN std_logic;
          C : IN std_logic;
          F : OUT std_logic
          );
     END COMPONENT;
     signal F1 : std_logic;
begin

     U1: ANDOR PORT MAP(
          A => J,
          B => K,
          C => L,
          F => F1
     );
     U2: ANDOR PORT MAP(
          A => F1,
          B => M,
          C => N,
          F => Q
     );

     P <= F1;
end Behavioral;
```

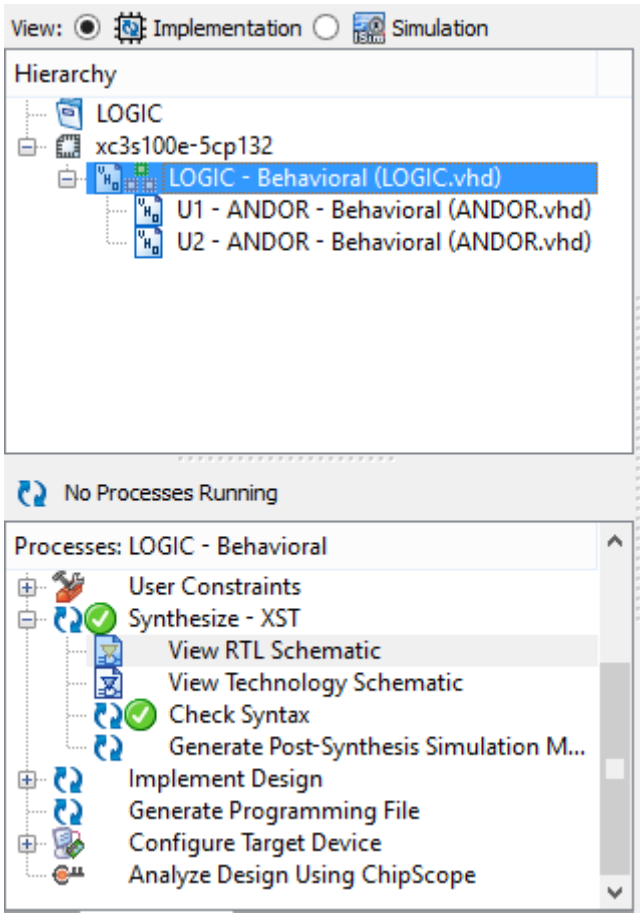
Componente del Archivo ANDOR

Instancias del Archivo ANDOR para generar el bloque final

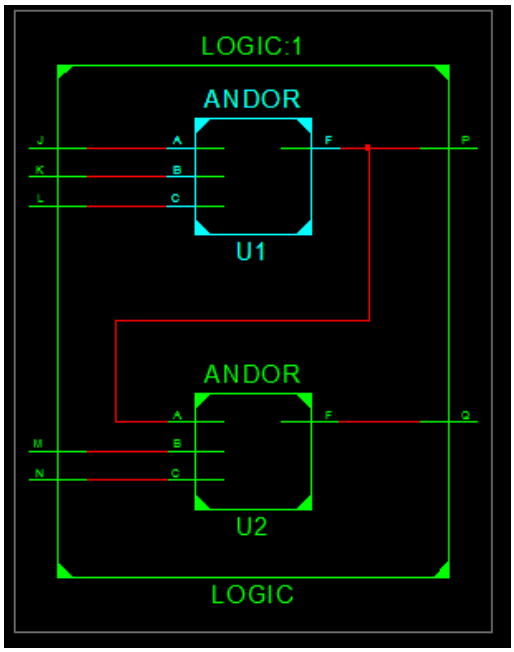
Explicación: Para poder generar el Bloque Final llamado LOGIC mostrado en la Figura 1, se tuvo que mandar a llamar el archivo ANDOR mediante un componente en el cual se expresan las entradas y salidas de ese archivo. La parte de Instancias es solamente la cantidad de veces que se repite el bloque ANDOR, en este caso seria 2 veces uniendo las salidas con sus respectivas entradas.

En el caso de que una salida sea utilizada para múltiples conexiones tanto interna como externa, es recomendable crear una señal para que pueda manejar los estados de otra manera.

12. Una vez finalizado el código, comprobaremos el diagrama de bloque final mediante la herramienta de Xilinx. Seleccionado el archivo top llamado LOGIC escoger la opción View RTL Schematic del desplegable de Synthesize-XST y darle clic a como se muestra en la figura.



13. Le aparecerá una ventana para seleccionar los archivos, escoja la opción de archivo top para poder ejecutarlo. Después de cierto tiempo aparece el bloque final con todas sus conexiones a como se aprecia en la figura.



Simulación de un circuito digital

14. Agregar un archivo de simulación .vhd al proyecto LOGIC para poder simular el comportamiento del circuito. Recuerde que es un circuito combinacional, así que tendrá que comentar aquellas líneas de código que involucren señales de tiempo (reloj)

Implementar el diseño en FPGA

15. Añadir un archivo .ucf al módulo principal siguiendo los pasos del laboratorio 1.
16. En el editor de texto, escribir las siguientes asignaciones:

```
net "J" loc = "G3" ;
net "K" loc = "B4" ;
net "L" loc = "K3" ;
net "M" loc = "L3" ;
net "N" loc = "P11";
net "P" loc = "P6" ;
net "Q" loc = "P7" ;
```

17. Comprobar si la sintaxis es correcta.
18. Comprobar el funcionamiento del circuito en la tarjeta de FPGA.

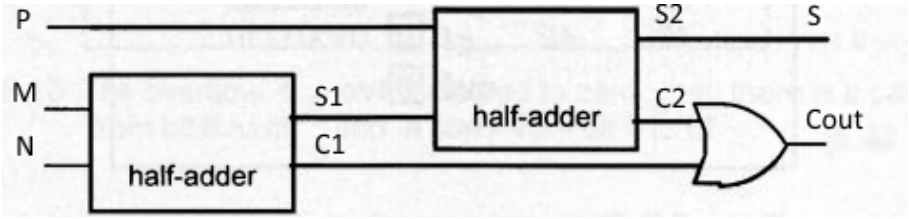
7. Actividades propuestas

Práctica

1. Realizar un proyecto que implemente un sistema combinacional que sea capaz de sumar dos bits y presente en su salida un bit de acarreo. Este sistema se le conoce como Semi Sumador (Half Adder) y tendrá el siguiente comportamiento según su tabla de verdad.

Entradas		Salidas	
A	B	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

2. Crear la simulación adecuada para poder implementarlo después en la tarjeta FPGA.
3. Crear nuevo proyecto que implemente el diagrama de bloques de un Sumador Completo (Full Adder) según la siguiente figura:



Actividades de aprendizaje

1 Indague acerca del lenguaje VHDL. Conteste.

¿Cuáles son las partes de un código VHDL? Describa brevemente cada una de sus partes por ejemplo entidad, arquitectura, componentes, etc.

El reporte debe incluir:

- Tabla de verdad de las variables.
- Descripción de las ecuaciones con compuertas lógicas.
- Descripción de las ecuaciones en VHDL.
- Verificación del programa en ISE. Utilice las capturas de pantalla (tecla: PRTSC) para explicar el procedimiento.
- Realizar las simulaciones del ejercicio del laboratorio 2.