Informe de laboratorio N°3
Sumador completo

Fernando Guiraud

Universidad tecnológica de Panamá
Microprocesadores
1EE141
fjguiraud@gmail.com

I. INTRODUCCIÓN

MODELO ESTRUCTURAL:
En el estilo estructural de modelado, una entidad se describe como un conjunto de componentes interconectados. Ejemplo: medio sumador. La declaración de entidad para halfadder especifica los puertos de interfaz para este cuerpo de arquitectura. El cuerpo de la arquitectura se compone de dos partes: la parte declarativa (antes de la palabra clave begin) y la parte de la declaración de sentencias (después de la palabra clave begin). Las declaraciones de dos componentes están presentes en la parte declarativa del cuerpo de la arquitectura. Estas declaraciones especifican la interfaz de los componentes que se utilizan en el cuerpo de la arquitectura. Los componentes declarados se instancian en la parte de la declaración del cuerpo de la arquitectura utilizando etiquetas de componentes para estas declaraciones de instanciación de componentes. Las señales en el mapa de puertos de un componente instanciado y las señales de puerto en la declaración del componente están asociadas por posición (llamada asociación posicional). Sin embargo, la representación estructural para el medio sumador no dice nada sobre su funcionalidad.

Se describirían modelos de entidad separados para los componentes XOR2 y AND2, cada uno con su propia declaración de entidad y cuerpo de arquitectura. Una declaración instanciada de componente es una declaración concurrente. Por lo tanto, el orden de estas declaraciones no es importante. El estilo estructural de modelado describe, solo una interconexión de componentes, sin implicar ningún comportamiento de los componentes mismos o de la entidad que representan colectivamente.

II. OBJETIVO

- Escribir un programa en VHDL para implementar el sumador completo utilizando dos medios sumadores y verificar la funcionalidad.

III. MATERIAL Y EQUIPO

- Tarjeta Elbert V2 – Spartan 3A FPGA Development Board

Figura 2: Tarjeta Spartan 3A FPGA

- Xilin ISE, 32-bit Project Navigator.

IV. DESARROLLO

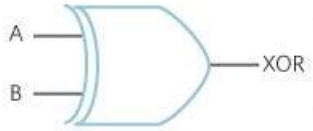
El primer paso para implementar el semisumador después de configurar la tarjeta Elbert V2, consiste en definir las entradas y salidas de este proyecto, tomando en cuenta que el ISE Project Navigator inicializa las librerías a utilizar por si solo:

```
entity sum is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           s : out  STD_LOGIC;
           co : out  STD_LOGIC);
end sum;
```

Se usan dos entradas y dos salidas, las entradas A y B corresponden a los valores booleanos que serán sumados, mientras que S corresponde al valor de la suma de A y B. Cin y co serán los bits de carry de entrada y salida respectivamente.

Para generar la operación de suma entre estas dos entradas, podemos emplear el uso de la compuerta XOR.

$$X = A \oplus B$$



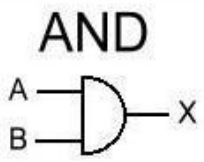
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Figura 3: Compuerta XOR y su tabla de la verdad

Como podemos ver en la figura 3, los valores de la tabla de la verdad corresponden a las salidas deseadas de los valores deseados del primer digito de la figura 1. La salida de esta compuerta será almacenada en la variable S.

Para generar el carry podemos emplear otra compuerta lógica que en este caso corresponde a la compuerta AND.

Tabla de verdad AND		
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1



Estas operaciones funcionan de igual forma que en el semisumador, la diferencia consiste en que para generar un sumador completo necesitamos dos semisumadores que contemplen el uso de un carry e entrada y un carry de salida, cin y co respectivamente.

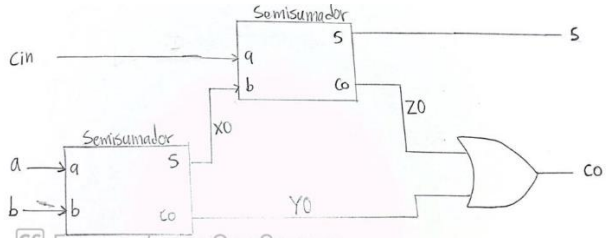
Para representar dos veces cada semisumador, creamos un component asociado con las variables del código sum principal:

```
Component semisum is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           s : out  STD_LOGIC;
           co : out  STD_LOGIC);
end Component;
```

Este component realiza la operación de un semisumador, la cual corresponde al siguiente código:

```
architecture Behavioral of semisum is

begin
s <= (a xor b);
co <= (a and b);
end Behavioral;
```

Después de tener el component definido, tenemos que mapear la función deseada, para esto seguimos el siguiente diagrama que ilustra el procedimiento necesario para crear el sumador completo.



Generamos las señales intermedias necesarias para mapear los componentes, y luego mapeamos las salidas correspondientes a la posición de la definición de las variables del componente.

```
Signal X0,Y0,Z0: STD_LOGIC;

begin

U1: semisum port map(a,b,cin,X0,Y0);
U2: semisum port map(cin,X0,cin,s,Z0);

co <= Y0 or Z0;

end Behavioral;
```

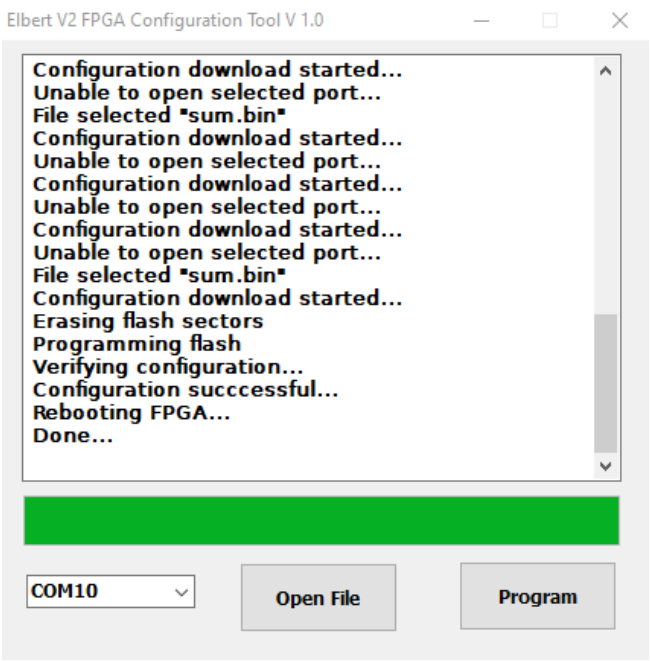
V. RESULTADOS

Para comprobar la funcionalidad de este código, procedemos a transferir el programa a la tarjeta, para esto tenemos que crear un archivo ucf que contenga la ruta de las entradas y salidas de la tarjeta que correspondan físicamente a el hardware deseado.

En este caso, representaremos las tres entradas como tres DP Switchs y cada salida con un led indicador.

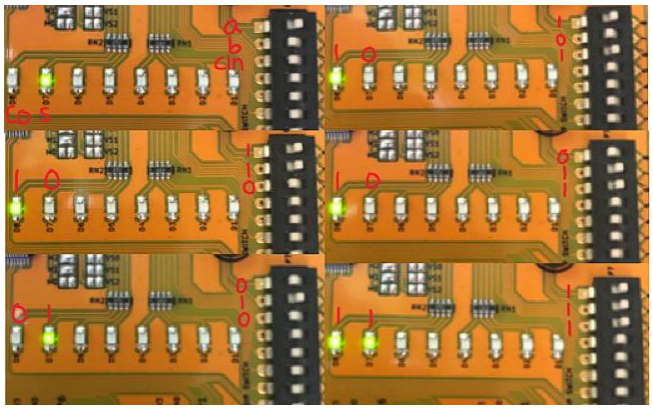
```
NET "co"            LOC = P46   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "s"             LOC = P47   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "a"       LOC = P70   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
NET "b"       LOC = P69   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;

NET "cin"       LOC = P68   | PULLUP  | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

Ahora procedemos a transferir el código a la tarjeta con el ejecutable proporcionado por el fabricante, en este caso Numato.



Escogemos el puerto correspondiente y enviamos el programa.

En la siguiente imagen podemos ver como las salidas corresponden a las salidas de en los leds, describiendo así las salidas.



VI. CONCLUSIÓN

Las operaciones de las compuertas lógicas pueden ayudarnos a simplificar los resultados de las entradas de manera significativa. Una compuerta lógica nos puede simplificar las cuatro posibles combinaciones de código necesario para representar de manera lógica las posibilidades de dos entradas y sus resultados dependiendo del operador.

Los components nos pueden ayudar a simplificar un código creando solamente una vez una estructura lógica y usándola repetidas veces.

El mapeo de las señales que se usan en cada component de manera encadenada contiene variables intermedias que deben ser guardadas temporalmente en señales declaradas y luego se transfieren a una señal de salida.

VII. REFERENCIAS

D. L. Perry, VHDL. New York: McGraw-Hill, 1991