CIRCUITOS DIGITALES II
PROYECTO 1 Y 2: DESARROLLO DE COMPUERTAS EN VHDL

Presentado por:
Hernando Bonilla Ramírez

Presentado a:
Ing. Carlos Hernán Tobar

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE TELECOMUNICACIONES

RESUMEN

Se propuso el desarrollo de diferentes compuertas lógicas y componentes electrónicos para llevar a cabo estos dos proyectos.

Para el proyecto 1 tenemos los siguientes componentes a llevar a cabo:

- Compuerta And
- Compuerta And16-bits.
- Compuerta Nand.
- Compuerta Not.
- Compuerta Not16-bits.
- Compuerta Or.
- Compuerta Or8way (8 entradas).
- Compuerta Or16-bits.
- Compuerta Xor.
- Demultiplexor.
- Demultiplexor4way (4 entradas).
- Demultiplexor8way.
- Multiplexor.
- Multiplexor16-bits.
- Multiplexor4way (4 entradas).
- Multiplexor4way16-bits.
- Multiplexor8way (8 entradas).
- Multiplexor8way16-bits.

Para el proyecto 2 tenemos a desarrollar los siguientes componentes

- Add16-bits.
- ALU (Unidad Aritmética Lógica).
- FullAdder.
- HalfAdder.
- Inc16.

EJEMPLO

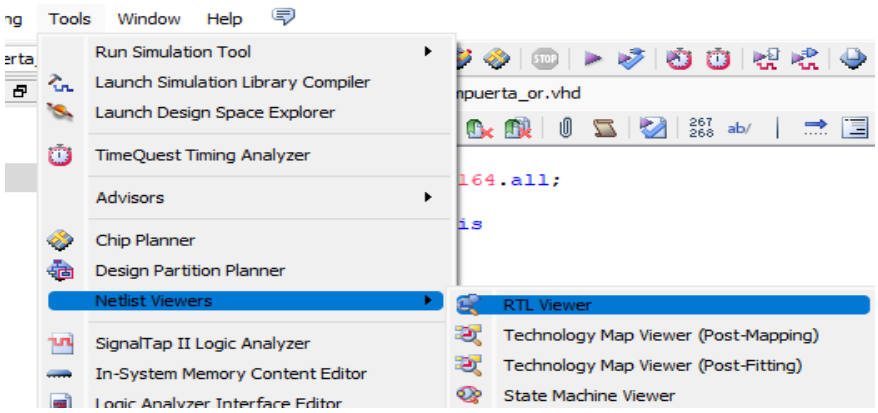
Como primer paso para resolver cualquiera de estos componentes, es declarar las librerías que se usaran en el código, seguido a eso, en la entidad del código se escribe el nombre que tendrá el archivo y se declaran los puertos, separando entre las entradas y salidas que tendrá la compuerta o componente, ya sea que la entrada o salida sea en forma de vector o de carácter binario, ya con eso se da por terminada la identidad y pasamos a la arquitectura de la compuerta o componente, en el cual nuevamente colocamos el nombre que se le puso al archivo anteriormente; en la arquitectura se declaran las señales que tendrá nuestro código, es decir, una variable que guarda un dato después de una operación, sin que se declare como salida dicho resultado, de igual forma que en la entidad, se debe declarar de que tipo son dichas señales, si son en forma de vector o de forma binaria.

Ahí mismo en la arquitectura, se declara el Begin, que es en el cual realizaremos las diferentes operaciones que tendrá nuestro código, de este modo dándole fin a la arquitectura y al código.

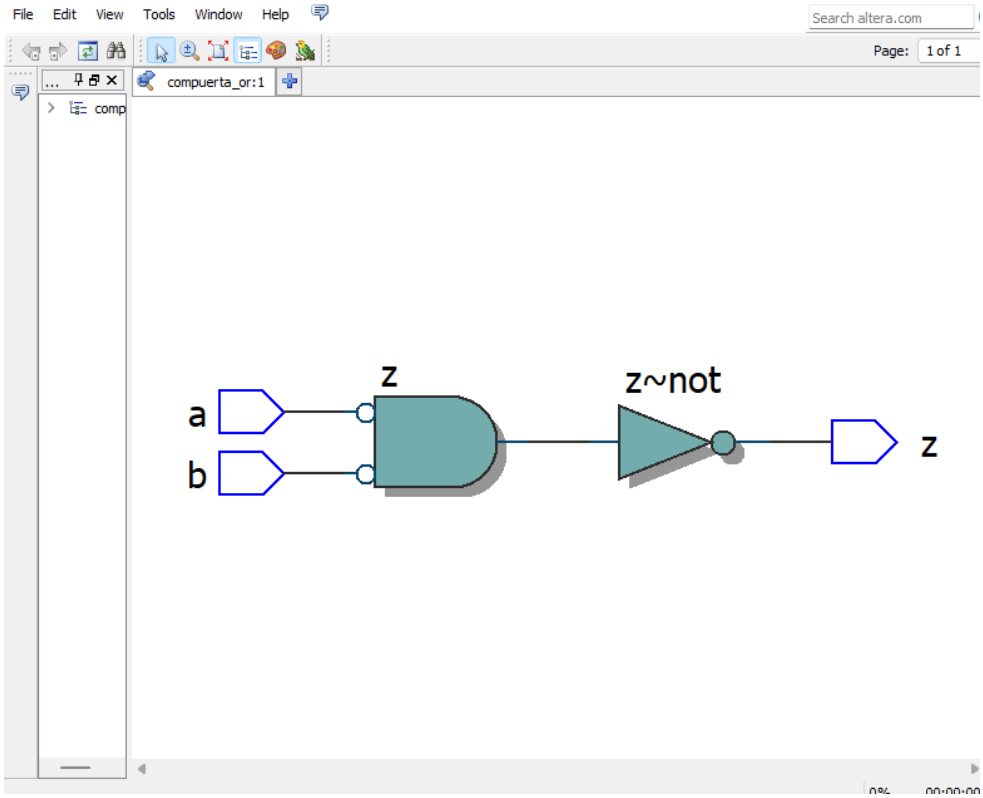
Este es el procedimiento que se llevó a cabo en todos los códigos, a continuación se pondrá el ejemplo de uno de ellos.

```
library ieee;

use ieee.std_logic_1164.all;


entity compuerta_or is

port (

a : in std_logic;

b : in std_logic;

z : out std_logic

);

end entity;

architecture arch of compuerta_or is

        signal x : std_logic := '0';

        signal y : std_logic := '0';


begin

x <= not a;

y <= not b;

z <= x nand y;

end architecture;
```

Por medio de Quartus, también se puede observar cual es el diagrama resultante del circuito, esto si el programa corre el código, es decir, si todo está correcto, y se puede ver de la siguiente manera



En este codigo, el cual es la compuerta Or, obtuvimos el siguiente diagrama



Por otro lado, al ya tener el codigo anterior hecho, lo que sigue es hacer el testbench, o banco de pruebas, es el codigo que nos permite ver que funcione correctamente.

Como primer paso, nuevamente se declaran las librerías, en la entidad declaramos el nombre del codigo, en este caso nada más es agregarle "_test" al final del nombre que ya tenía en el otro codigo, seguido a eso, en la arquitectura se declara el componente, el cual es el anterior codigo que se usó, en el cual, en el puerto se nombran las entradas y salidas que se usarán, y damos fin al componente.

Se declaran las señales, en este caso serían las que se usaron en el anterior codigo, y las de entradas y salidas, seguidos nuevamente de "_test", así mismo declarando que tipo de señal es, y a las que son entradas o salidas, se inicializan en '0'.

En el apartado dut1, escribimos el nombre del anterior codigo, para en el mapeo de puertos, declarar las entradas y salidas que teníamos antes, a su fase de prueba. De este modo, empezamos con el proceso, que consiste en colocar los valores que tendrán las entradas, y colocar el valor que se espera en la salida, y que dado el caso que no se cumpla, presentar el error, ese proceso se hace para todas las posibles combinaciones que tienen las entradas y el valor esperado en la salida. A continuación se adjuntara el codigo testbench que se uso en este caso como ejemplo.

```
library    IEEE;
use IEEE.std_logic_1164.all;
-- Entity (empty)
entity compuerta_or_test is
end entity;
-- Architecture (test process)
architecture arch_test of compuerta_or_test is
-- Component declaration
        component compuerta_or
                port(
                        a       :       in      std_logic;
                        b       :       in      std_logic;
                        z       :       out     std_logic
                );
        end component;
-- Signal declaration
        signal a_test       : std_logic := '0';
        signal b_test       : std_logic := '0';
        signal z_test  : std_logic := '0';
        signal x_test  : std_logic;
        signal y_test       : std_logic;
        begin
-- DUT instantiation
```

```
dut1        : compuerta_or
              port map (
                        a => a_test,
                        b => b_test,
                        z => z_test
              );
-- Stimulus generation
Stimulus : process
begin
  report "Inicio de la prueba de la compuerta or"
              severity note;
              a_test <= '0';
              b_test <= '0';
              wait for 1 ns;
              assert z_test = '0'
               report "Falla para x = 0"
               severity failure;
              a_test <= '0';
              b_test <= '1';
              wait for 1 ns;
              assert z_test = '1'
               report "Falla para x = 0"
               severity failure;
              a_test <= '1';
              b_test <= '0';
              wait for 1 ns;
              assert z_test = '1'
               report "Falla para x = 0"
               severity failure;
              a_test <= '1';
              b_test <= '1';
              wait for 10 ns;
```

```
                assert z_test = '1'
                 report "Falla para x = 0"
                 severity failure;
                report "Test successful"
                severity note;
                wait;
        end process;
end architecture;
```

Lo ultimo que se hizo, fue corroborar que todo funcionara bien mediante la aplicación de modelsim que ya viene con el programa de Quartus, que fue donde se llevó a cabo el desarrollo de los dos proyectos, y de esta manera concluir con las compuertas lógicas y componentes que se crearon.