

INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

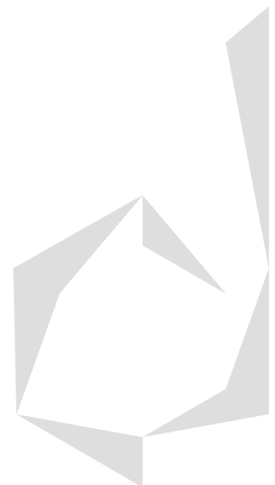
ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Simulador del ISE Xilinx

Contenido

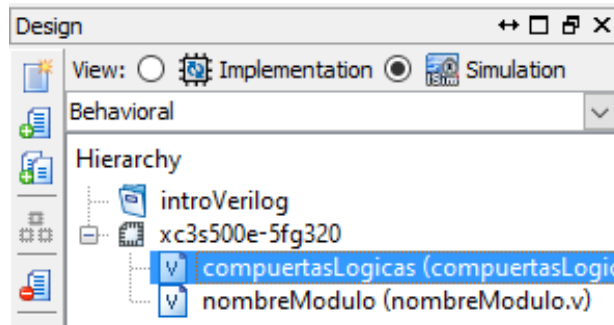
Simulador de ISE Xilinx	2
Código VHDL:	8
Código Verilog:	9
Simulación:	10
Conclusión:	12



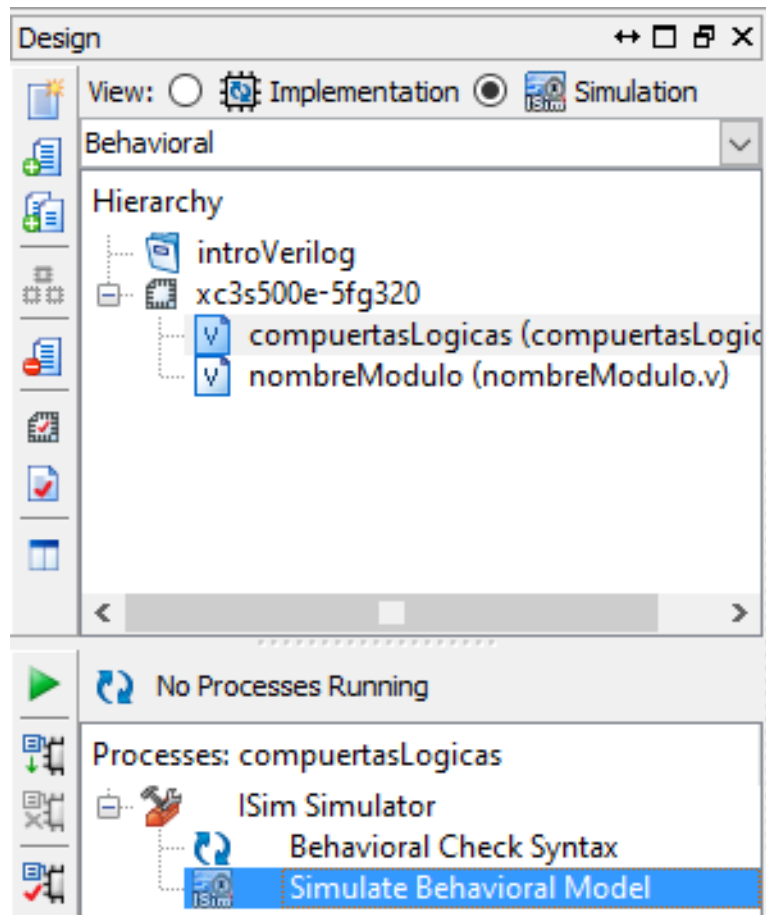
Simulador de ISE Xilinx

El simulador es una manera de comprobar que nuestro código cumple la función lógica que debería efectuar no importando si el código fue escrito en VHDL o Verilog, teniendo como ventaja la de proporcionar información del comportamiento del sistema antes de implementarlo a la tarjeta NEXYS.

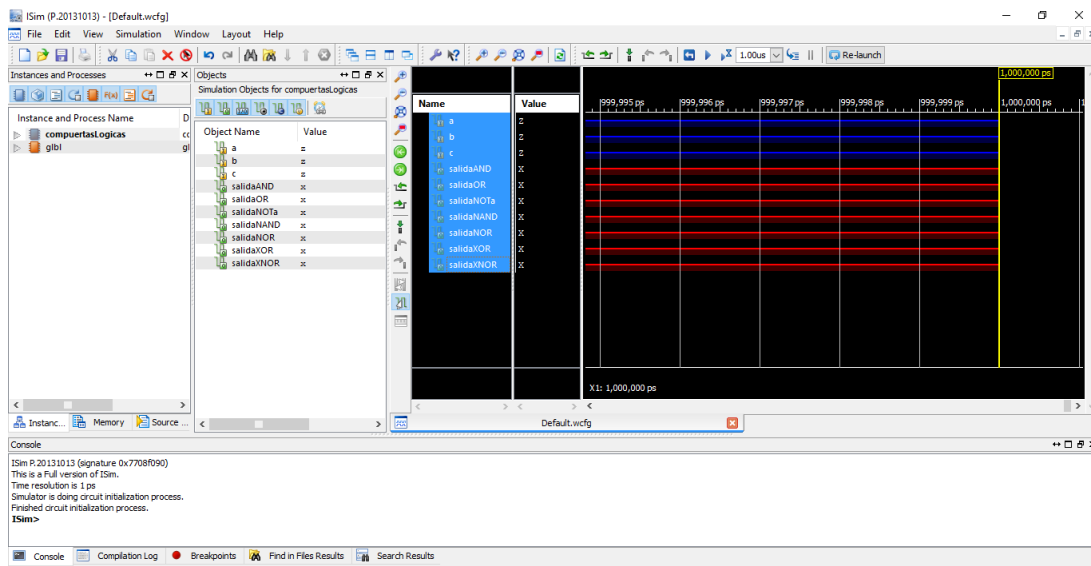
Podemos acceder al simulador si damos clic en el radio button que dice Simulation.



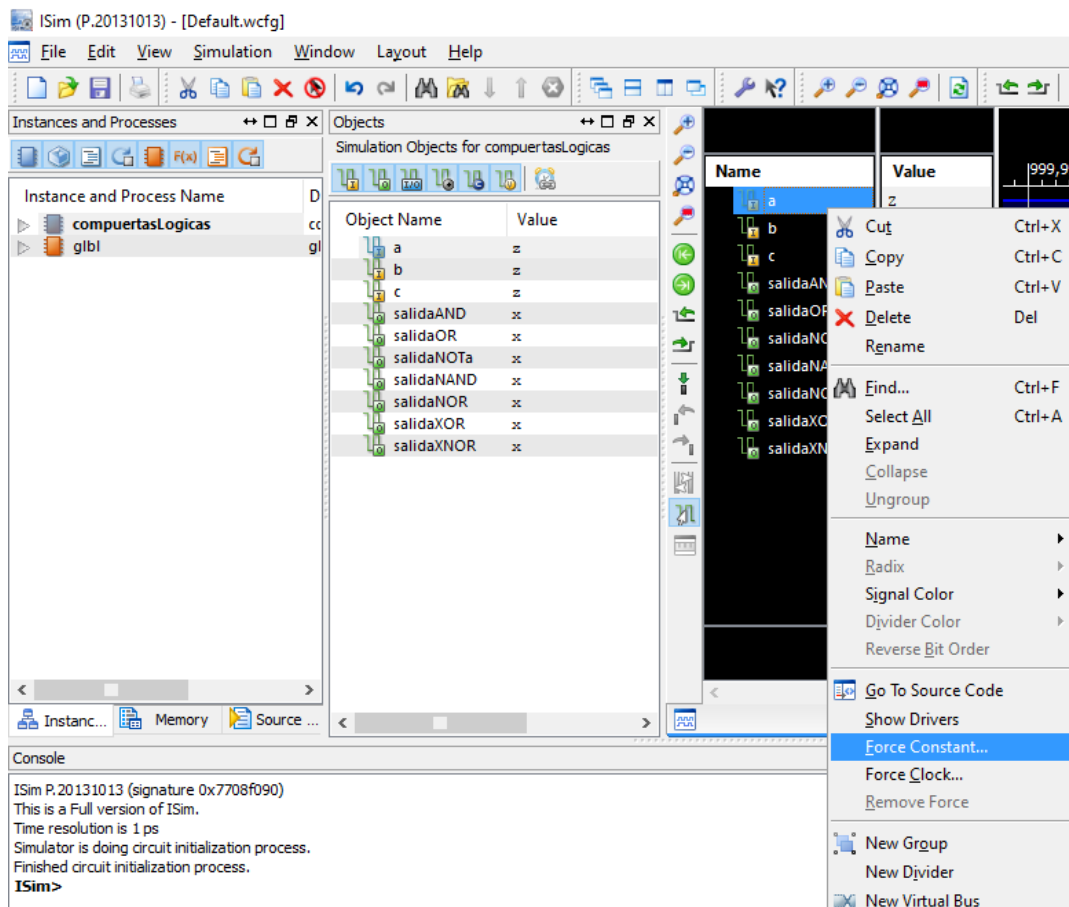
Después mientras tenemos seleccionado el módulo que queremos simular, debemos irnos a la parte inferior y dar doble clic en la subpestaña de ISim Simulator donde dice Simulate Behavioral Model.



Al hacer esto nos aparecerá una ventana emergente donde en un inicio no podremos ver nada.



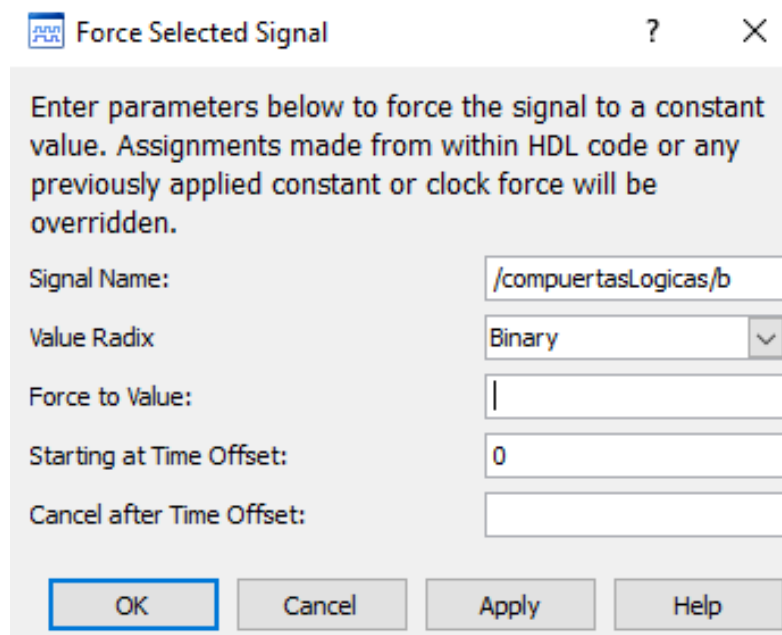
Para poder simular nuestro módulo deberemos dar clic derecho en las entradas del código (las entradas las muestra el simulador poniendo una pequeña I alado del nombre de la variable) y darle valores constantes por medio de la opción Force Constant.



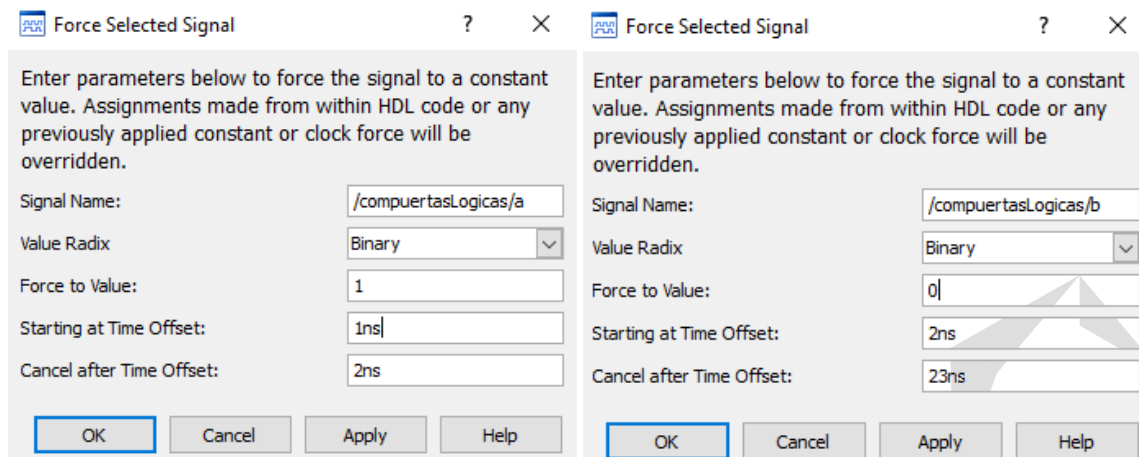
Si doy clic en la opción de **Force Constant** me saldrá una ventana donde puedo decirle a la entrada que adopte un **valor constante** a través de toda la simulación del programa:

- Si es una entrada de un bit puede adoptar un valor de 0 lógico, 1 lógico o Z alta impedancia.
- Si es por ejemplo un vector de 2 bits puede adoptar un valor de 00, 01, 10 o 11.

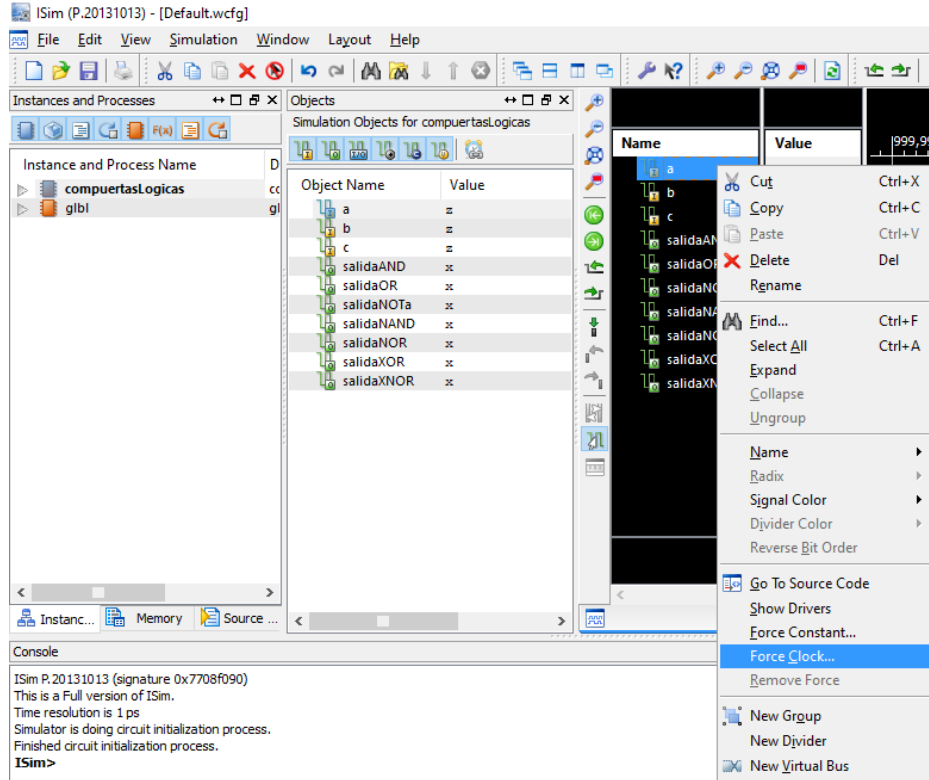
El **valor constante** que tomará la entrada lo asigno donde dice **Force to Value**, además puedo indicarle al programa **a partir de qué tiempo** en la simulación quiero que le dé ese valor en donde dice **Starting at Time Offset** o en qué tiempo quiero que **deje de tener ese valor constante** donde dice **Cancel after Time Offset**.



Un truco que puede servir para asignar varios valores sin tener que salirme de esta pestaña y dar clic derecho en cada elemento es cambiar el nombre de la variable directamente donde dice Signal Name y dar clic en el botón de Apply.



O puedo darle valores periódicos a tomar por medio de la opción Force Clock.



Donde dice Leading Edge Value debo poner el primer valor constante que adopta la variable y en donde dice Trailing Edge Value debo poner el segundo valor constante que adopta la variable a través de todo el ciclo, recordemos que los valores que puede adoptar la variable dependen de si es una entrada de un bit o si es un vector, finalmente en donde dice Period pondremos el periodo en el que se va a repetir la señal. Normalmente en las demás opciones no se pone nada.

Define Clock
?
X

Enter parameters below to force the signal to an alternating pattern (clock). Assignments made from within HDL code or any previously applied constant or clock force will be overridden

Signal Name:

Value Radix:

Leading Edge Value:

Trailing Edge Value:

Starting at Time Offset:

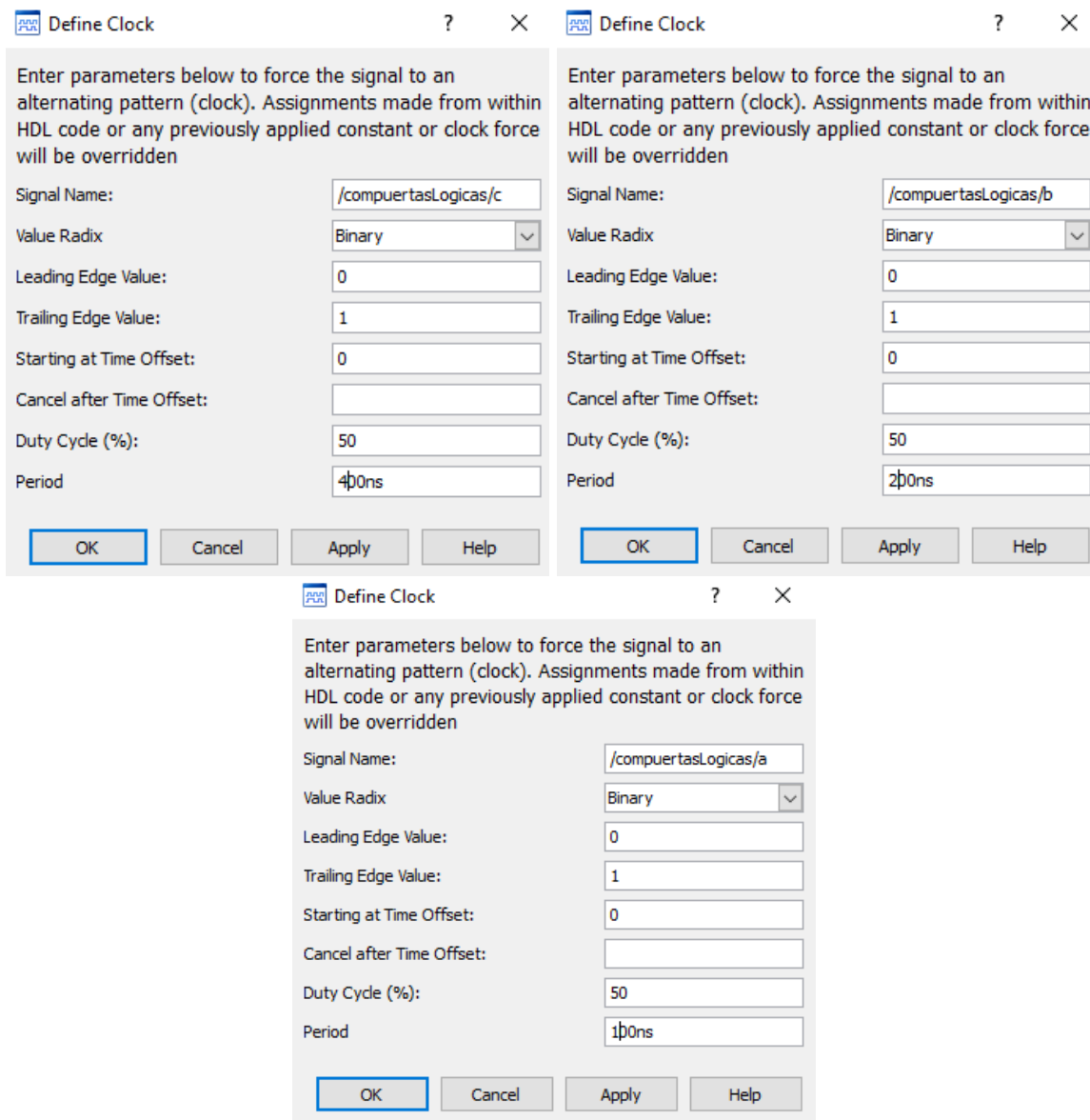
Cancel after Time Offset:

Duty Cycle (%):

Period:

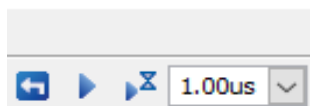
OK
Cancel
Apply
Help

Con esta opción de Force Clock lo que se debe de hacer para que las diferentes variables que tenga en mi código se acomoden como deben en la tabla de verdad para formar todas las combinaciones es que en la última entrada debo poner un número cualquiera en donde dice Period, pero cada vez que le asigne un periodo a la entrada de arriba, debo dividir este periodo entre 2, además todas tienen que tener el mismo Leading Edge Value y Trailing Edge Value.

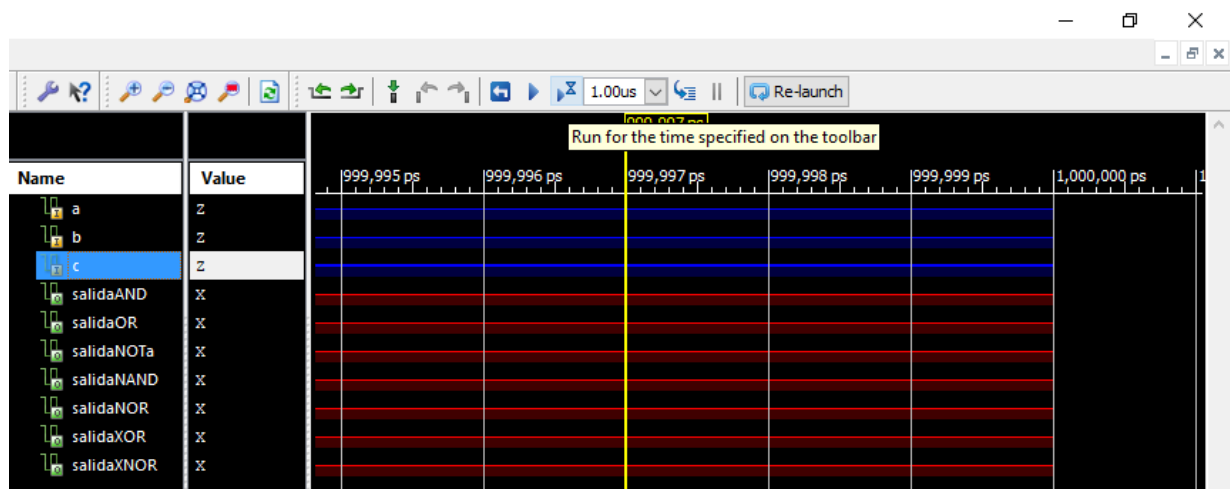


También me puede servir el truco para asignar varios valores sin tener que salirme de esta pestaña.

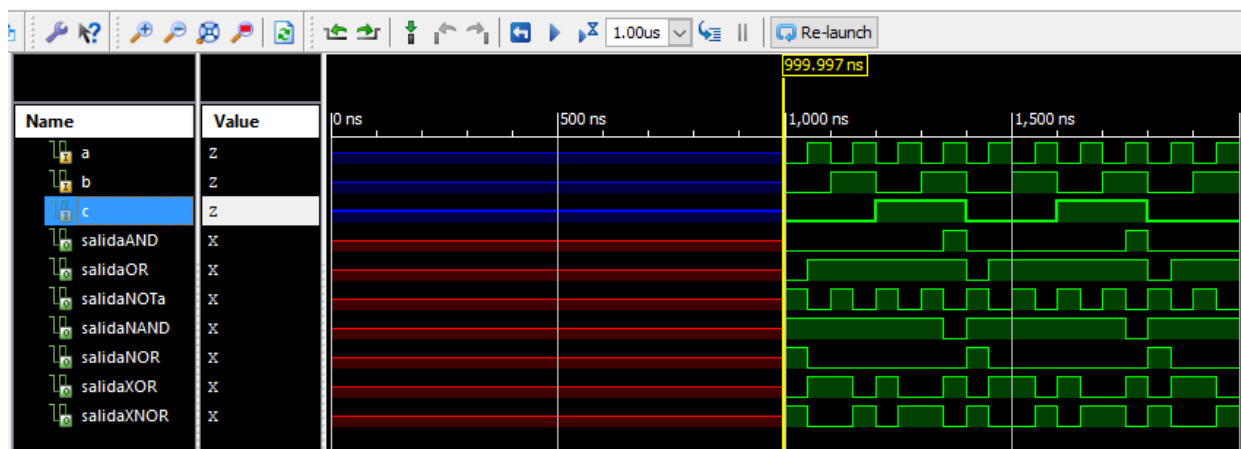
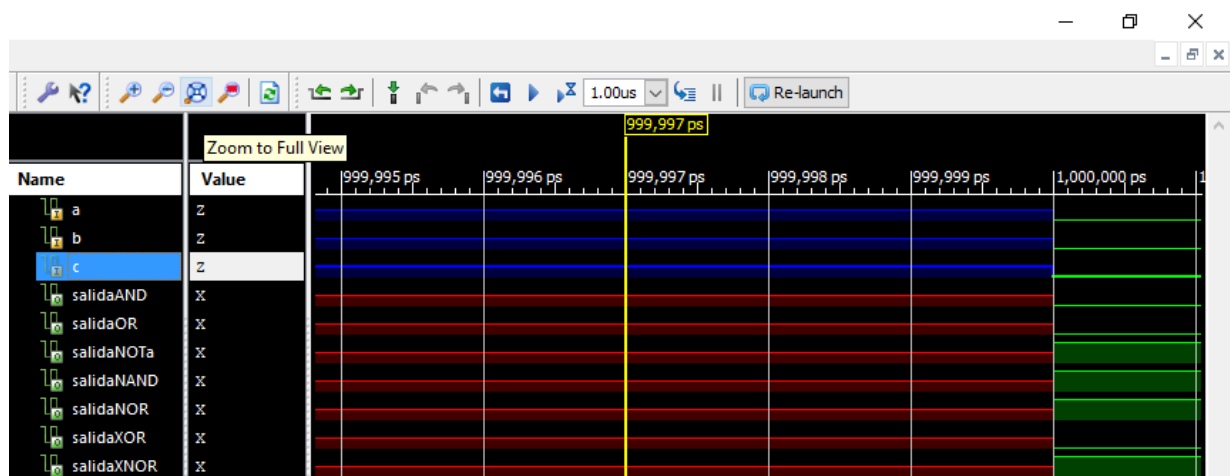
Debo tomar en cuenta el tiempo de simulación que se va a hacer para saber cuánto tiempo asignarle a cada periodo, este se ve hasta arriba de la ventana:



Ya que haya asignado estos valores de periodo debo dar clic en el botón azul que está alado del tiempo para que se efectúe la simulación.



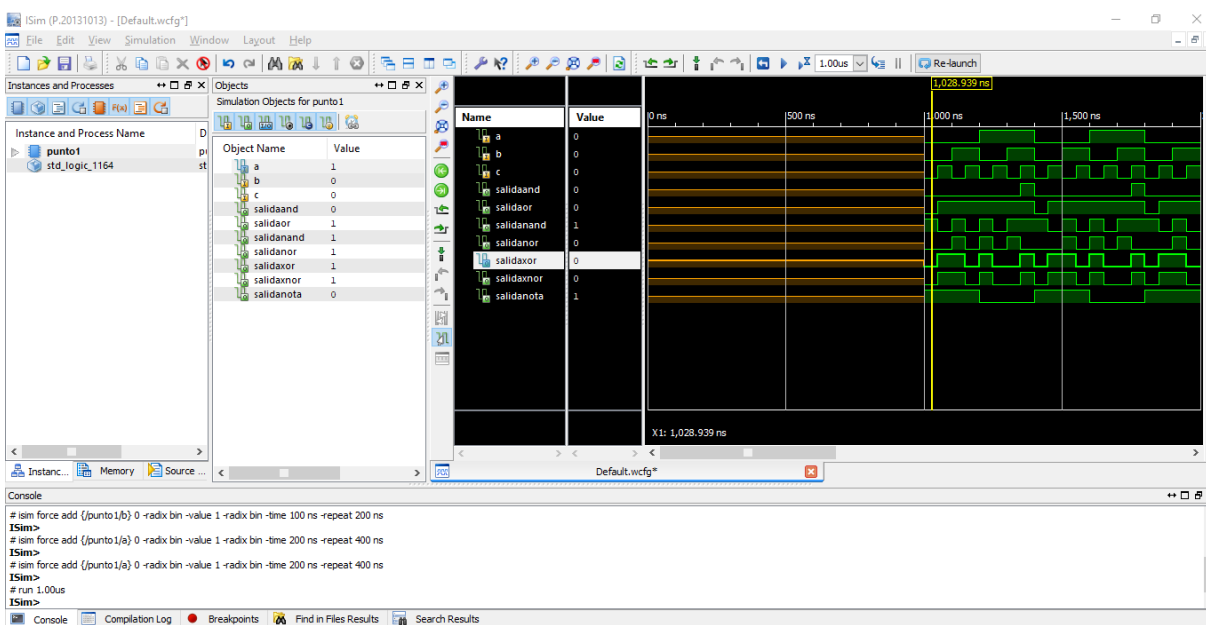
Luego debo dar clic en el botón que tiene una lupa para poder visualizar las señales de mi simulación.



Las señales verdes son todas mis entradas y salidas, a la derecha de cada una respectivamente.

La información simulada se puede visualizar en forma de tabla de verdad al poner nuestro cursor amarillo sobre un punto en específico de la combinación de señales de entrada (digitales) compuestas por 0 y 1.

Se puede utilizar para comprobar el funcionamiento de simples compuertas lógicas AND, OR, NAND, NOR, XOR, XNOR y NOT y resolver problemas lógicos simples o se puede efectuar para comprobar el resultado de la resolución de problemas lógicos más complejos, en donde se construye una tabla de verdad con un número de entradas que debe cumplir ciertas condiciones dadas por el problema, posteriormente se extrae una función lógica por medio del método de mapas de Karnaugh o por el método de Quine McCluskey y al simular el programa me podré dar cuenta si la función obtenida satisface todas las salidas propuestas en la tabla de verdad correctamente.



Al hacer la simulación no importa si el código está escrito con VHDL o Verilog esta se realiza de la misma manera y a diferencia de la implementación a la tarjeta por medio del archivo UCF, no importa tampoco si hay varios módulos dentro de una misma carpeta de proyecto, cada módulo se puede simular sin ningún problema.

Al realizar la simulación en el Simulador del programa ISE Xilinx obtengo lo siguiente:

Código VDHL:

```
--Declaración de librerías
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
--Entidad: Parte del código VHDL donde se declaran las entradas/salidas
entity punto1 is
  Port ( a : in STD_LOGIC;
```



```

    b : in STD_LOGIC;
    c : in STD_LOGIC;
    salidaAND : out STD_LOGIC;
    salidaOR : out STD_LOGIC;
    salidaNAND : out STD_LOGIC;
    salidaNOR : out STD_LOGIC;
    salidaXOR : out STD_LOGIC;
    salidaXNOR : out STD_LOGIC;
    salidaNOTa : out STD_LOGIC
        );
end punto1;

```

--Arquitectura: Parte del código VHDL donde se indican las operaciones lógicas que hará el programa con las entradas/salidas

```

architecture Compuertas of punto1 is
begin
    salidaAND <= a and b and c;
    salidaOR <= a or b or c;
    salidaNAND <= not (a and b and c);
    --El operador NAND con más de 2 variables se debe hacer negando la operación and entre las
    --3 o más variables
    salidaNOR <= not (a or b or c);
    --El operador NOR con más de 2 variables se debe hacer negando la operación or entre las 3
    --o más variables
    salidaXOR <= a xor b xor c;
    salidaXNOR <= not (a xor b xor c);
    --El operador XNOR con más de 2 variables se debe hacer negando la operación or entre las 3
    --o más variables
    salidaNOTa <= not a;
end Compuertas;

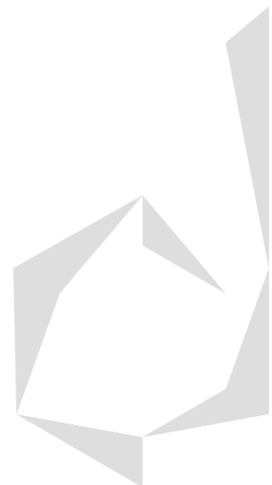
```

Código Verilog:

```

//1) Operaciones lógicas por medio de símbolos
//Módulo: Dentro de éste va todo el código escrito en Verilog
module nombreModulo (
    //Las entradas y salidas: van declaradas dentro del paréntesis
    input wire a,
    input wire b,
    output wire salidaAND,
    output wire salidaOR,
    output wire salidaNOTa,
    output wire salidaNAND,
    output wire salidaNOR,

```



```

output wire salidaXOR,
output wire salidaXNOR
);

```

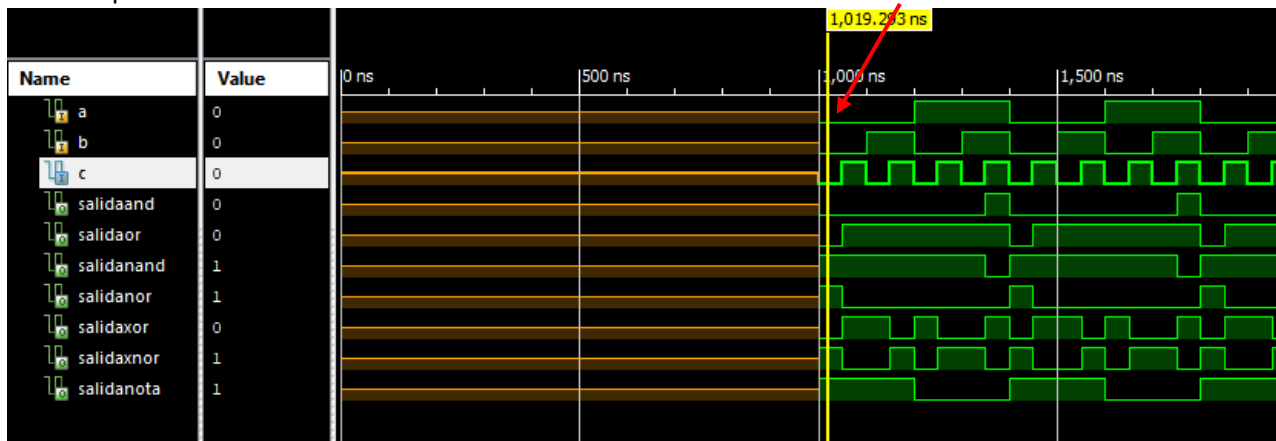
```

//Fuera del paréntesis del módulo se indica lo que hará el programa con las
//entradas/salidas declaradas, las operaciones más sencillas son las lógicas.
//Recuerdo que en Verilog para asignar un valor debo usar el signo igual =
assign salidaAND = a & b & c;
//La operación AND se hace con el símbolo & presionando "alt" + "6"
assign salidaOR = a | b | c;
//La operación OR se hace con el símbolo | presionando la tecla a la izquierda del
//número 1 en el teclado
assign salidaNOTa = ~(a);
//La operación NOT se hace con el símbolo ~ presionando "alt gr" + "+"
assign salidaNAND = ~(a & b & c);
//La operación NAND se hace negando la operación AND con el símbolo ~
//presionando "alt gr" + "+"
assign salidaNOR = ~(a | b | c);
//La operación NOR se hace negando la operación OR con el símbolo ~
//presionando "alt gr" + "+"
assign salidaXOR = ~(a ^ b ^ c);
//La operación XOR se hace con el símbolo ^ presionando "alt gr" + "{" + "space bar"
assign salidaXNOR = ~(a | b | c);
//La operación NOR se hace negando la operación OR con el símbolo ~
//presionando "alt gr" + "+"
endmodule

```

Simulación:

- 1) Cuando selecciono con el cursor (la línea vertical amarilla) que todas las entradas sean 0 lógico, lo que debería obtener en la salida es:

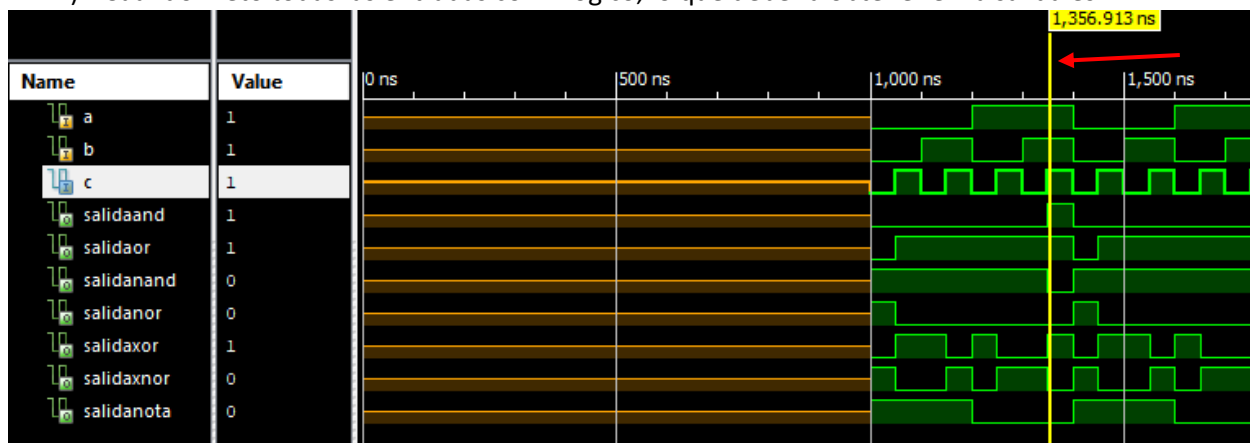


A	B	C	AND	OR	NAND	NOR	XOR	XNOR	NOT A
0	0	0	0	0	1	1	0	1	1
0	0	1	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0	1	1
1	0	0	0	1	1	0	1	0	0
1	0	1	0	1	1	0	0	1	0
1	1	0	0	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0	0

AND = 0, OR = 0, NAND = 1, NOR = 1, XOR = 0, XNOR = 0, NOT A = 1.

Y como puedo ver en la tabla de verdad eso es lo que obtengo, por lo tanto, el código está realizando su función correctamente.

2) Cuando meto todas las entradas con 1 lógico, lo que debería obtener en la salida es:

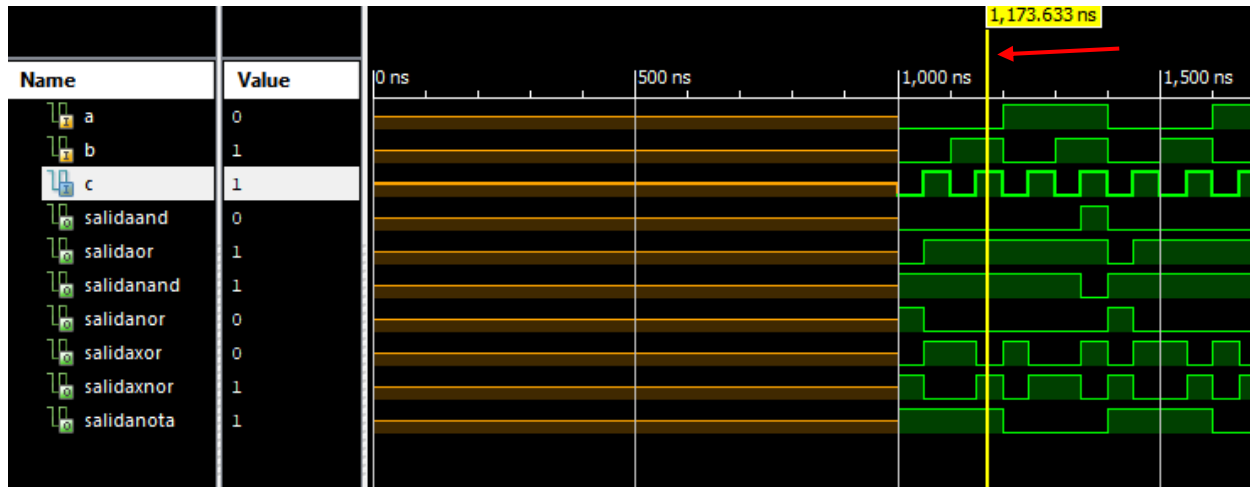


A	B	C	AND	OR	NAND	NOR	XOR	XNOR	NOT A
0	0	0	0	0	1	1	0	1	1
0	0	1	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0	1	1
1	0	0	0	1	1	0	1	0	0
1	0	1	0	1	1	0	0	1	0
1	1	0	0	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0	0

AND = 1, OR = 1, NAND = 0, NOR = 0, XOR = 1, XNOR = 0, NOT A = 0.

Y como puedo ver en la tabla de verdad eso es lo que obtengo, por lo tanto, el código está realizando su función correctamente.

- 3) Cuando selecciono con el cursor una entrada particular como a=0, b=1 y c=1, lo que debería obtener en la salida es:



A	B	C	AND	OR	NAND	NOR	XOR	XNOR	NOT A
0	0	0	0	0	1	1	0	1	1
0	0	1	0	1	1	0	1	0	1
0	1	0	0	1	1	0	1	0	1
0	1	1	0	1	1	0	0	1	1
1	0	0	0	1	1	0	1	0	0
1	0	1	0	1	1	0	0	1	0
1	1	0	0	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0	0

AND = 0, OR = 1, NAND = 1, NOR = 0, XOR = 0, XNOR = 1, NOT A = 1.

Y como puedo ver en la tabla de verdad de la simulación eso es lo que obtengo, por lo tanto, el código está realizando su función correctamente.

Conclusión:

Los circuitos lógicos sirven para controlar nuestros circuitos y que realicen ciertas funciones, esto para resolver problemas que parecen complejos de una forma muy sencilla, apoyándonos en las operaciones lógicas existentes AND, OR, NAND, NOR XOR, XNOR Y NOT. Cuando el problema no se puede resolver con las operaciones lógicas básicas directamente, dependiendo de nuestro número de variables de entrada A,B,C,D o E debemos recurrir a otros métodos ya que aumentará el tamaño de nuestra tabla de verdad, ya de igual manera crecerá el número de combinaciones que obtendremos en la tabla de verdad de nuestro problema lógico, relacionados por la siguiente fórmula:

$$\#combinaciones = 2^{\#variables}$$

Para estos casos podemos usar el mapa de Karnaugh para encontrar una función lógica que satisfaga todos los casos planteados, si el número de variables es de 4 o más variables A,B,C,D,E,F,G,...,N se debe

usar el método de Quine McCluskey para encontrar una función lógica que satisfaga la tabla de verdad propuesta para el problema.

Entonces los problemas lógicos se pueden complicar bastante y el uso del simulador nos puede decir si todas las salidas de nuestra tabla de verdad propuesta para el problema son satisfechas por la función que obtuvimos. Además nos ayuda a ver si tenemos problemas en el código como con las operaciones NAND, NOR y XNOR de 3 variables en donde nos percatamos que no daba el resultado correcto de la operación si poníamos directamente en el código los operadores nand, nor y xor, más bien teníamos que aplicar la operación con and, or y xor para después negarla, esto no lo hubiéramos podido visualizar sin la ayuda de nuestra simulación. Si este tipo de errores ocurren con operaciones sencillas, cuando hagamos operaciones más complejas visualizar el error puede ser más complicado.

