

Reporte Actividad 02: Compuertas lógicas

Estrada Rivera Gustavo de Jesús 220746114 | 23/08/2025

INTRODUCCIÓN

El presente reporte se centra en el **diseño y simulación de circuitos digitales básicos**. Para ello, se utilizó un lenguaje de descripción de hardware (HDL) para programar un dispositivo lógico reconfigurable, conocido como FPGA.

Es en este proceso que nos permitirá traducir conceptos teóricos de la electrónica digital, como las operaciones booleanas representadas en tablas de verdad, a una implementación física tangible y funcional. A través de la simulación, podremos verificar el comportamiento de nuestro diseño antes de implementarlo en hardware, asegurando que las salidas correspondan a las entradas esperadas según la teoría.

¿Qué es una FPGA?

Un **FPGA** (Field-Programmable Gate Array) o "Arreglo de Compuertas Programable en Campo" es un circuito integrado diseñado para ser configurado por un diseñador o un cliente después de su fabricación. A diferencia de los microprocesadores que ejecutan software, un FPGA no tiene una arquitectura fija; en su lugar, está compuesto por una matriz de bloques lógicos configurables (CLBs) y un sistema de interconexiones programables.

Relación con el lenguaje VERILOG:

La configuración de un FPGA no se realiza dibujando esquemáticos a mano, sino describiendo el hardware deseado mediante un **Lenguaje de Descripción de Hardware (HDL)**. Aquí es donde entra **Verilog**.

Verilog es uno de los lenguajes más populares para describir circuitos electrónicos digitales. A diferencia de lenguajes de programación convencionales como C++ o Python, que se ejecutan secuencialmente, en Verilog en su lugar, se usa para *describir* la estructura y el comportamiento de un circuito, incluyendo cómo fluyen los datos y cómo están conectados los componentes.

En palabras más simples, Verilog nos ayuda a describir hardware, y en general podemos resumirlo así:

- **Verilog:** Es el lenguaje que usas para **describir** el diseño de tu circuito digital.
- **FPGA:** Es el **chip físico** reconfigurable que implementa ese diseño.

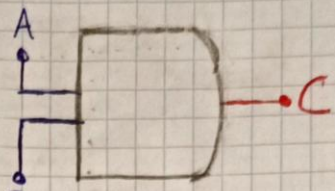
Compuertas Lógicas:

A continuación, se presentarán las siete compuertas lógicas junto a su simbología, sus tablas de verdad y su sintaxis en el lenguaje Verilog.

Compuerta AND

La salida es "1" (VERDADERO) sí y solo si **todas** sus entradas son "1".

Compuerta AND		
Entradas		Salida
A	B	C
0	0	0
1	0	0
0	1	0
1	1	1




AND
 $A * B = C$
Sintaxis en Verilog:
 $C = A \& B$

Compuerta NAND

Es la negación de una compuerta AND. La salida es "0" sí y solo si **todas** sus entradas son "1".

Compuerta NAND		
Entradas		Salida
A	B	C
0	0	1
1	0	1
0	1	1
1	1	0



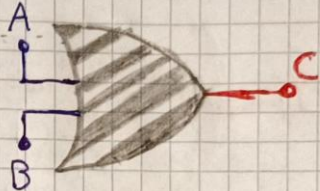
NAND
 $A * B = C$
Sintaxis en Verilog:
 $C = \sim(A \& B)$

Compuerta OR

La salida es "1" (VERDADERO) si **al menos una** de sus entradas es "1".

Compuerta OR

Entradas		Salida
A	B	C
0	0	0
1	0	1
0	1	1
1	1	1



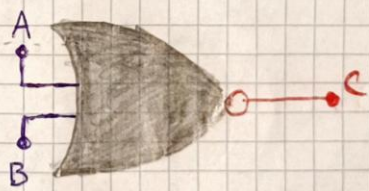
OR
 $A + B = C$
Syntax en Verilog
 $C = A | B$

Compuerta NOR

Es la negación de una compuerta OR. La salida es "1" sí y solo si todas sus entradas son "0".

Compuerta NOR

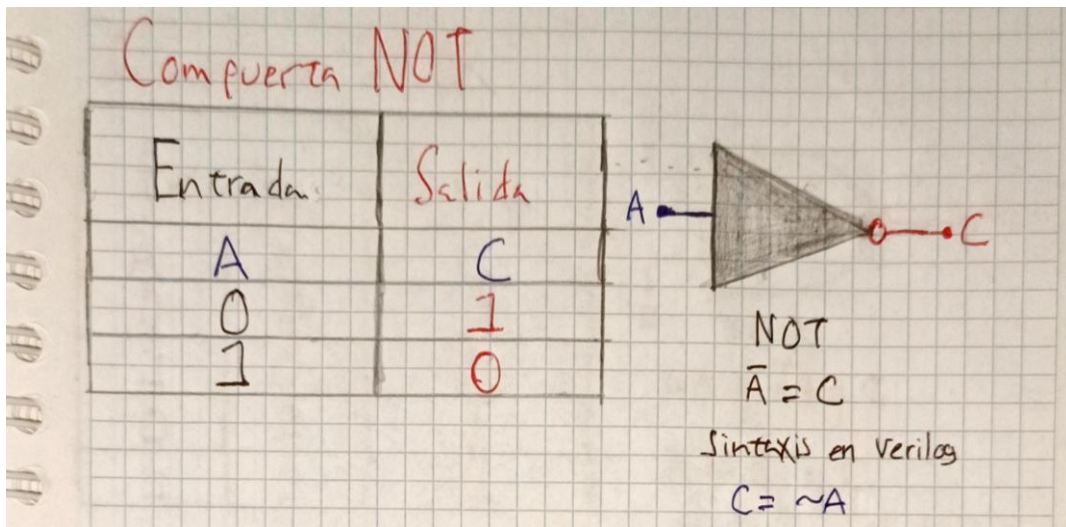
Entradas		Salida
A	B	C
0	0	1
1	0	0
0	1	0
1	1	0



NOR
 $A + B = C$
Syntax en Verilog
 $C = \sim(A | B)$

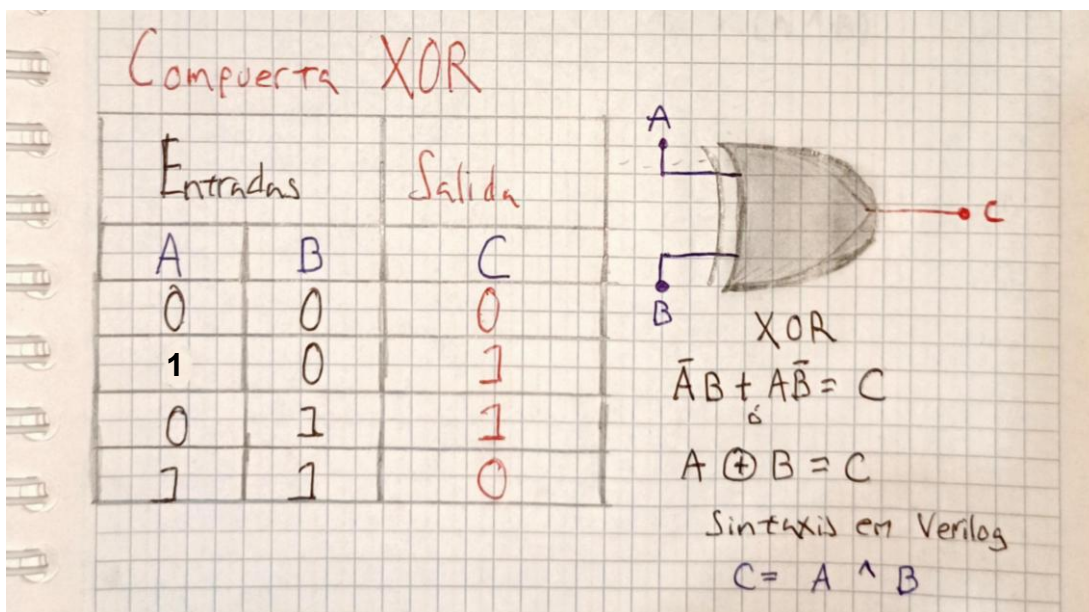
Compuerta NOT

La salida es el estado lógico opuesto a la entrada.



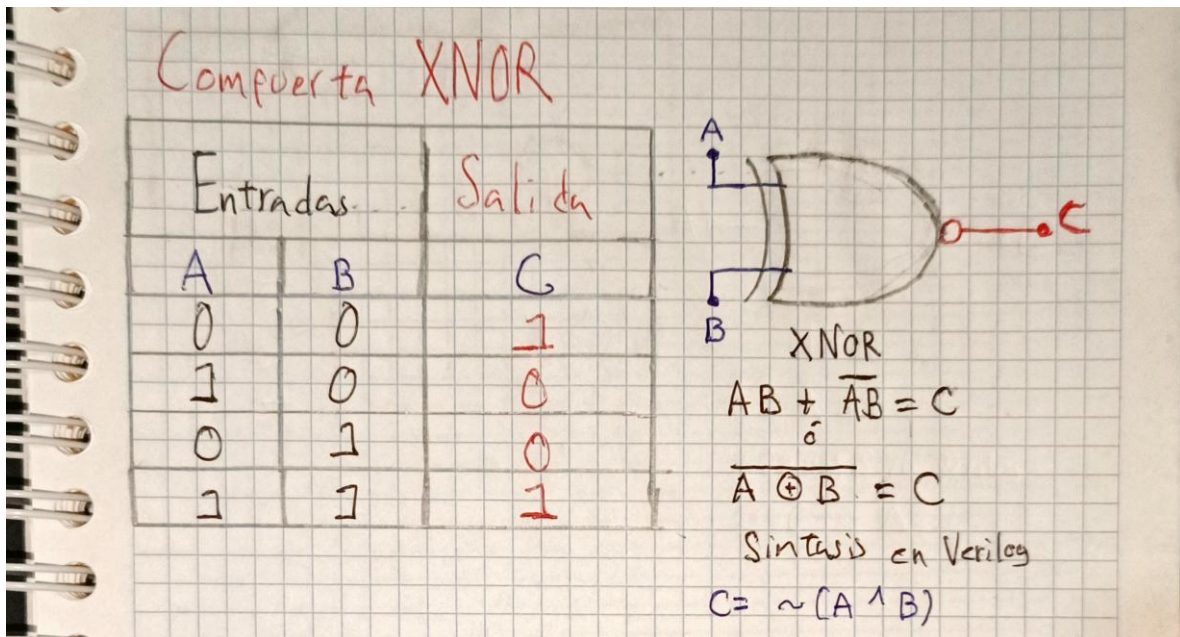
Compuerta XOR (OR Exclusiva)

La salida es "1" si las entradas son diferentes.



Compuerta XNOR (OR Exclusiva Negada)

Es la negación de una compuerta XOR. La salida es "1" si las entradas son **iguales**.



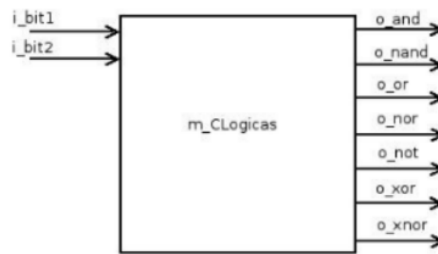
OBJETIVOS

Como objetivos podemos describir los siguientes:

- Comprender la relación entre FPGA y el lenguaje de descripción de hardware "Verilog" e implementarlo en trabajos futuros.
- Analizar las compuertas lógicas mediante sus tablas de verdad y su sintaxis en código para su implementación.
- Desarrollar un módulo único en lenguaje Verilog que implemente las siete compuertas lógicas solicitadas para su posterior simulación.
- Simular el funcionamiento del módulo y verificar que los resultados obtenidos coincidan con los esperados en las tablas de verdad.
- Relacionar la teoría de álgebra booleana con la práctica de diseño digital en FPGA.

DESARROLLO

Para realizar esta práctica se realizó una investigación previa del como funciona Verilog, las compuertas lógicas y el cómo se podrían implementar en este lenguaje. Para un ejemplo ilustrativo se diseño el modulo solicitado el cual consta de dos **entradas** (A y B) y siete salidas (Siete salidas para cada compuerta lógica: **AND**, **NAND**, **OR**, **NOR**, **NOT**, **XOR**, **XNOR**):



Una vez obtenido el diseño del módulo, a continuación, se pasará a explicar el desarrollo del código y su simulación:

Código:

Antes de empezar con el módulo de las compuertas, agregamos lo siguiente:

```
`timescale 1ns/1ns
```

Esto nos servirá para ahorrarnos un paso durante la simulación, ya que nos definirá la escala de tiempo de la simulación.

Ahora sí, empecemos con nuestro modulo. Para comenzar nombramos nuestro modulo para la práctica, en este caso la llamamos como “compuertas_logicas” de la siguiente manera:

```
module compuertas_logicas (
```

Una vez bautizado nuestro modulo, abrimos paréntesis para declara nuestras entradas y salidas (Input’s y Output’s) como se muestra ahora:

```
module compuertas_logicas (
    input A,
    input B,
    output AND,
    output NAND,
    output OR,
    output NOR,
    output NOT,
    output XOR,
    output XNOR
);
```

Definimos nuestras dos entradas (A y B) y nuestras salidas (Una por cada compuerta lógica), para evitar errores y darle algo más de “elegancia”, separamos cada una con comas y saltos de línea.

Una vez implementadas las entradas y salidas, cerramos paréntesis para ahora asignar las instrucciones del como operaran cada compuerta, es aquí que entra la

palabra clave **assign** en donde daremos la lógica combinacional para cada compuerta como se muestra a continuación:

```
assign AND    = A & B;
assign NAND   = ~(A & B);
assign OR     = A | B;
assign NOR    = ~(A | B);
assign NOT    = ~A;
assign XOR    = A ^ B;
assign XNOR   = ~(A ^ B);
```

Cada compuerta tiene su operación, y esa operación se representa con un símbolo en especial o como se le llama “la operación a nivel de bits” (AND: &, OR: |, XOR: ^, NOT: ~).

Una vez asignadas las instrucciones para cada una de las compuertas lógicas, agregamos lo siguiente para finalizar nuestra definición del modulo:

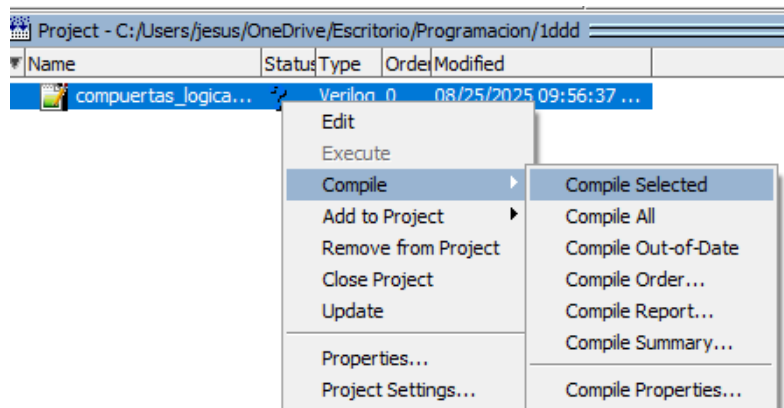
```
endmodule
```

Con esto nos da como resultado un código que describe de manera concisa, paralela y eficiente un circuito que ejecuta las siete operaciones lógicas fundamentales de forma simultánea.

Ahora veremos los resultados de la simulación mediante el software **Modelsim** en su versión más reciente.

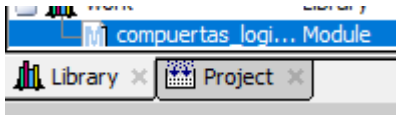
Simulación:

Una vez terminado nuestro modulo, pasaremos a compilar nuestro archivo .v desde Modelsim así:

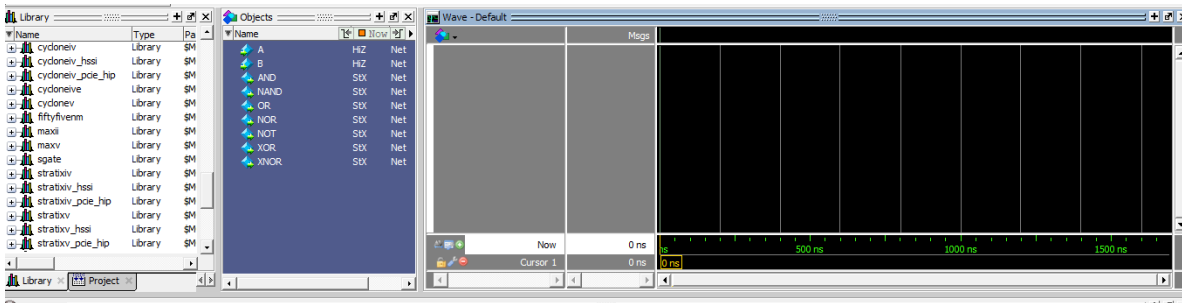


Compilamos para evitar errores, ahora una vez que aparezca una **palomita verde** en la pantalla nos indicara que el archivo se compilo correctamente.

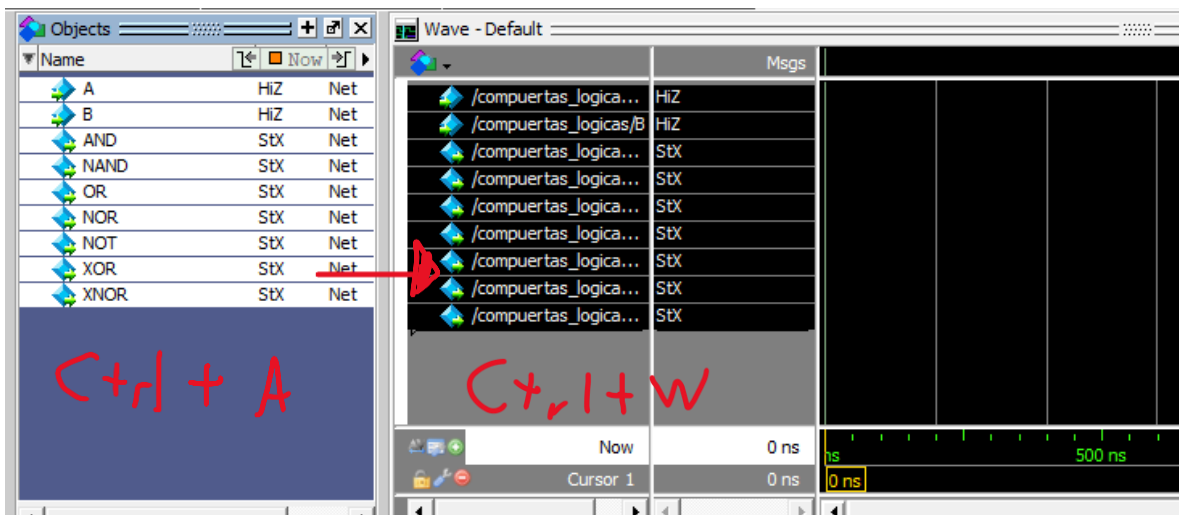
A continuación, para empezar la simulación, iremos al apartado de “library” para buscar la librería “work”, ahí daremos doble clic en el modulo que le dimos previamente el nombre de “compuertas_logicas”:



Al darle doble clic, se nos mostrara la siguiente pantalla:

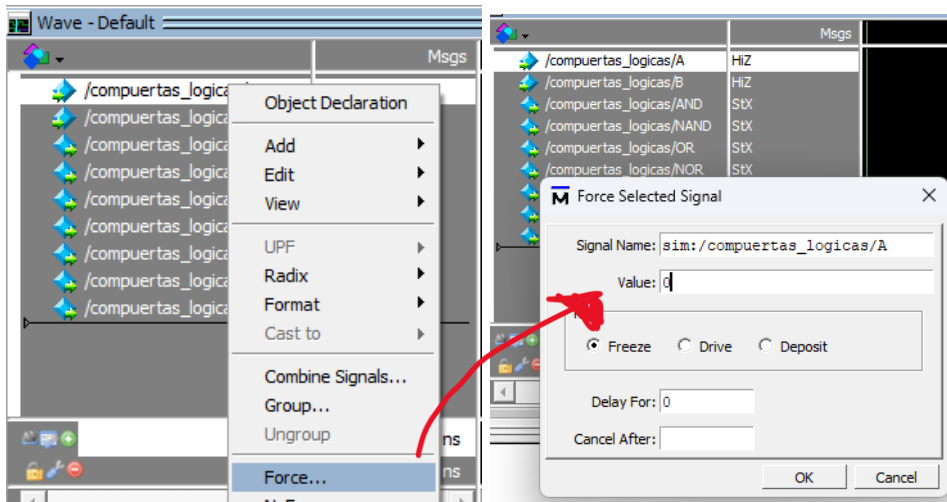


En el apartado de objetos, se nos muestra todo lo que definimos en nuestro modulo (entradas y salidas). Para poder empezar la simulación tendremos que mover esos “objetos” a la ventana de la derecha llamada Wave, para eso tendremos que presionar la combinación Ctrl+A para seleccionar todos los objetos y Ctrl+W para pasarlos de manera automática a la ventana Wave:

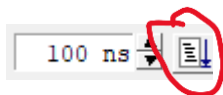


Una vez que nuestros objetos estén en la ventana Wave, podremos por fin empezar con la simulación. Para ello, tendemos que poner los valores correspondientes en las entradas, es aquí que entra los valores de las tablas de verdad de las compuertas, para poderle asignar un valor a las entradas daremos clic derecho e

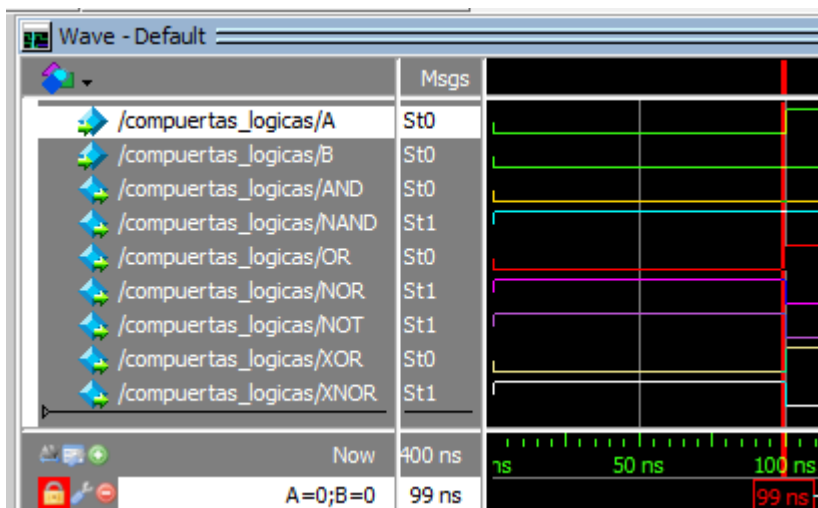
iremos a la pestaña “Force” en donde podremos asignar los valores requeridos para la simulación:



Una vez puesto los primeros valores de la tabla de verdad (0,0) antes de simular verificamos si la escala de tiempo sea la correcta, por preferancia del profesor, se nos pide que la simulación dure 100 nanosegundos (ns), es por eso que agregamos la directriz *timescale 1ns/1ns* en el codigo para que se ponga la escala de tiempo de manera automatica ya que por defecto y omitiendo el timesclae, la duracion de la simulacion seria de 100 picosegundos (ps), así ahorrandonos un paso en corregir los tiempos. Ahora una vez visto ese detalle, podemos comenzar la simulacion, para ello justo al lado de la marca de tiempo damos clic en run para iniciar la simulación:

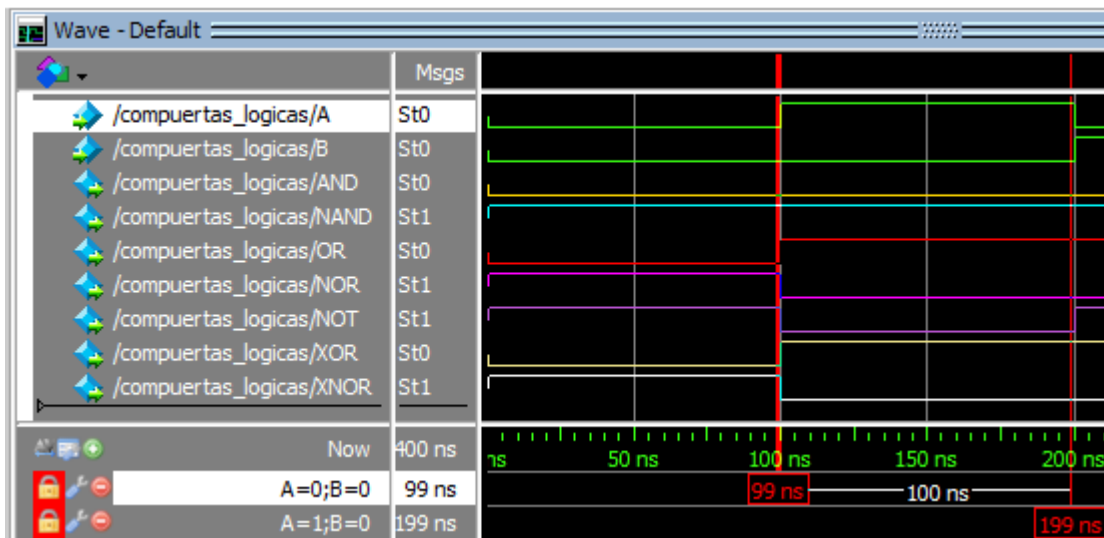


Una vez dado clic, la primera simulacion se habra realizado de manera correcta si aparece en la ventana Wave lo siguiente:



Con esto se dio la primera simulación de las primeras entradas de la tabla de verdad (A= 0, B=0), para conocer los resultados de las siguientes entradas, repetimos el mismo proceso:

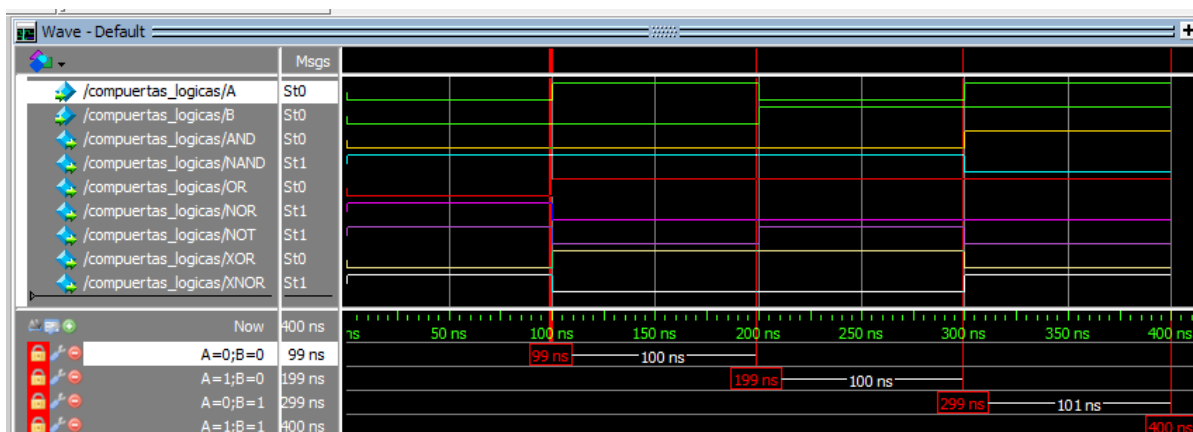
Simulación 2 (A= 1, B=0):



Simulación 3 (A= 0, B=1):



Simulación 4 (A= 1, B=1):



Una vez terminado las simulaciones tendríamos como resultado una pantalla similar a la anterior imagen.

Comprobando con las tablas de verdad de cada una de las compuertas lógicas, vemos que se verificó que las salidas coinciden exactamente con los valores de la misma, y que nuestro modulo se desarrolló con éxito :D.

CONCLUSIONES

Puedo concluir que esta practica me pareció interesante ya que me proporciono conceptos de hardware que no conocía o que no le daba su debida importancia y que me orilló a investigar mas a fondo sus características e implementaciones. También me sirvió para refrescar el tema de las compuertas lógicas y sus tablas de verdad, tema que la verdad tenia olvidado por completo, así que esta práctica me sirvió para darle una nueva repasada y ahora con más detalle.

REFERENCIAS

Wikipedia contributors. (2025, August 25). *Field-programmable gate array*. Wikipedia. https://en.wikipedia.org/wiki/Field-programmable_gate_array

¿Qué es una FPGA? | Supermicro. (n.d.). <https://www.supermicro.com/es/glossary/fpga>

What is FPGA Programming? (n.d.). FPGA4student.com. <https://www.fpga4student.com/2017/08/what-is-fpga-programming.html>

Arm Ltd. (n.d.). *What is FPGA?* Arm | the Architecture for the Digital World. <https://www.arm.com/glossary/fpga>

Introduction to FPGA Part 3 - Getting Started with Verilog. (n.d.). DigiKey. <https://www.digikey.com.mx/en/maker/projects/introduction-to-fpga-part-3-getting-started-with-verilog/9d9dbff29a4b45728521b2664bbd1df4>

<https://medium.com/%40roshanmaharana1510/introduction-to-fpga-with-verilog-ab6f02cbda34>

Compuertas lógicas – Sistemas Digitales. (n.d.). <https://virtual.cuautitlan.unam.mx/intar/sistdig/compuertas-logicas/>

Compuertas lógicas. (n.d.). <https://cienciayt.com/electronica/sistemas-digitales/compuertas-logicas/>

Logicbus. (2024, November 30). *Compuertas lógicas. Blog Logicbus*. <https://www.logicbus.com.mx/blog/compuertas-logicas/>

Logic Gates Verilog Code - Circuit fever. (n.d.). <https://circuitfever.com/logic-gates-verilog-code>