Washington Christian University

Escuela Latinoamericana de Altos Estudios Empresariales

Convenio Interinstitucional

Carrera: Ingeniería en Tecnología de la Información

Módulo: Arquitectura del Computador

Unidad III

Informe – Laboratorio 2 con SystemC

Facilitador:

Prof. María Guerrero.

Presentado por:

Esnaider Monterrosa.

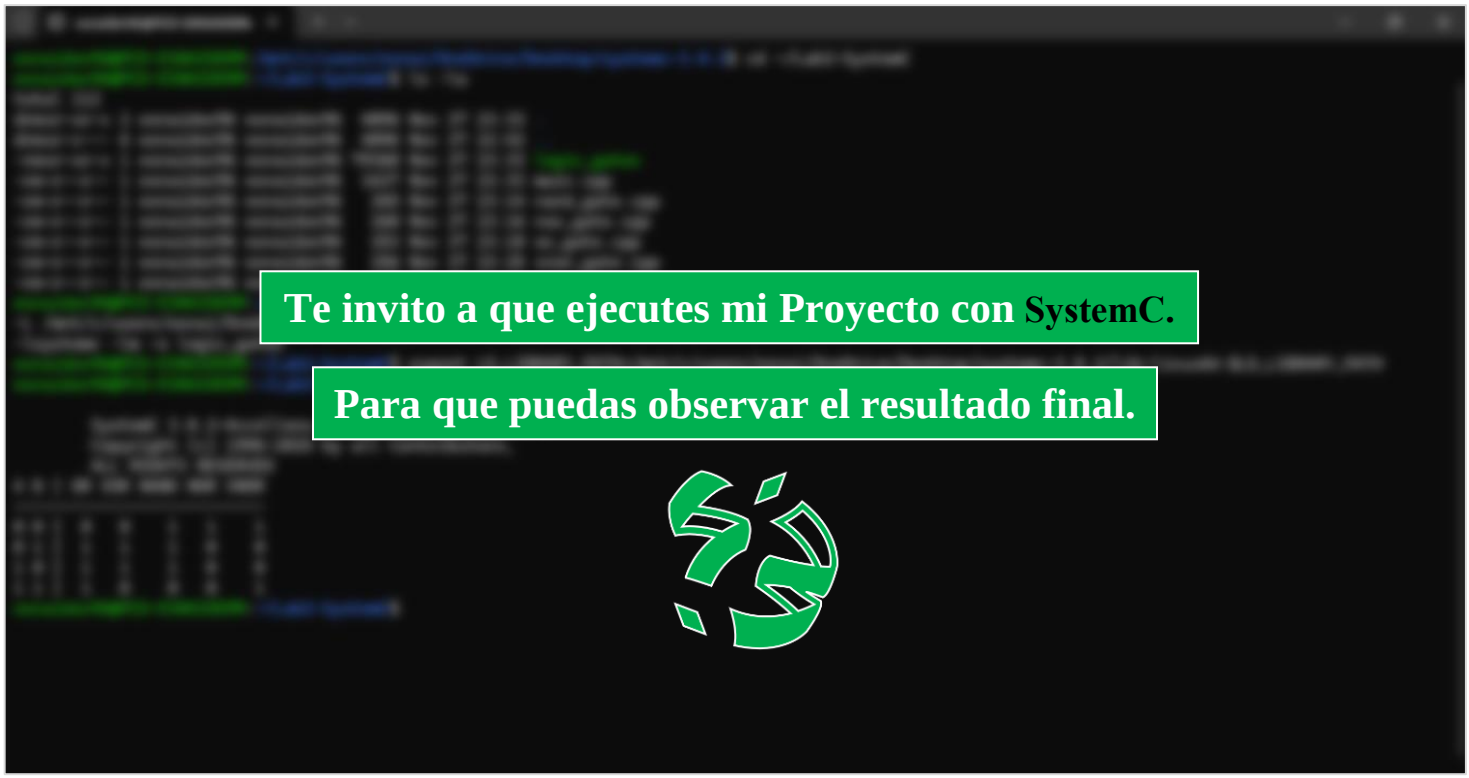
C.I. 25.417.024

Caracas, noviembre del 2025

INTRODUCCIÓN:

En el presente informe les voy a presentar cómo me he aplicado al **Laboratorio 2 de la Unidad III**, donde el objetivo principal fue meternos de lleno en **SystemC** para simular hardware de verdad. Aquí demuestro cómo logramos armar y poner a funcionar nuestras propias compuertas lógicas con **OR, XOR, NAND,** etc… usando **código C++** especializado, corriendo todo esto sobre un entorno Linux que no perdona errores.

Más allá de simplemente mostrar que el código compila y corre, en este trabajo explico con mis propias palabras la vuelta técnica detrás de la simulación. Analizo las diferencias claves entre los componentes del sistema como los **sc_signal** y **sc_in/out,** la importancia de manejar los tiempos con **.read()** y **.write()**, y por qué herramientas profesionales como **SystemC** son otras comparadas con simuladores visuales como **Logisim.** A continuación, detallo mis respuestas y los resultados de esta experiencia práctica.



INFORME – LABORATORIO 2 CON SYSTEMC

1) ¿QUÉ DIFERENCIA EXISTE ENTRE SC_SIGNAL Y SC_IN/SC_OUT?:

Yo desde mi opinión personal, lo entendí realizando el circuito en tiempo real. La diferencia es como comparar el enchufe con el cable:

- **Porque en el sc_in y sc_out:** yo lo comprendo que son como las compuertas o los terminales del módulo. Es por donde entra o sale la información de la entrada lógica. Es como decir por aquí recibo el voltaje o por aquí muestro el resultado. Son la parte fija de la caja, que es un módulo.
- **sc_signal:** es como el cable que usamos para conectar esas compuertas. En el **main.cpp** vi que las **sc_signal** no pertenecen a ninguna compuerta en específico, sino que son como los cables que tiramos en un **protoboard virtual** para unir la salida de una compuerta con la entrada de otra o con el banco de pruebas.

  Con esto quiero decir que los **sc_in/out** los veo y analizo que son como los bornes del aparato y la **sc_signal** es el cable pelao que usamos para hacer el empate entre ellos.

2) ¿POR QUÉ EN SYSTEMC LAS SEÑALES DEBEN LEERSE CON .READ() Y ESCRIBIRSE CON .WRITE()?:

Yo lo pienso porque **SystemC** no es **C++ normalito**, aquí estamos simulando hardware, con corriente y tiempos y no solo variables de software.

Porque no usamos el signo **igual** = directo porque en la vida real la electricidad no viaja instantáneamente.

- **Cuando usamos .write()**, le estamos diciendo al simulador, algo así en nuestro idioma humano: Hola mi pana, por favor te pido que agendes este cambio de valor para el próximo ciclo.
- **Y cuando usamos .read()**, nos aseguramos de leer el valor que tiene el cable ahora mismo y no el que va a tener en el futuro.

Porque si no lo hiciéramos así, se armaría un gran desorden y desastre con la afluencia, porque todos los módulos funcionan al mismo tiempo y podríamos leer datos viejos o malos. Pienso yo que es la forma de mantener el orden en la simulación.

3) ¿QUÉ OCURRIRÍA SI SE USA UNA COMPUERTA NAND EN CASCADA CON OTRA NOR?:

Yo entiendo que ponerlas en cascada significa que la **salida de la NAND** se conecta directo a una de las **entradas de la NOR.** Lo que pasaría es que crearíamos un circuito combinacional muchísimo más complejo, como lo he podido comprender.

Es por este motivo que la señal primero tiene que ser procesada por la **NAND**, en hacer su operación y esperar su pequeño retardo, y ese mismo resultado viaja para alimentar a la **NOR.** Es por eso que sencillamente, estamos combinando lógicas para crear una función nueva.

1) Si la **NAND, por ejemplo:** saca un 1, la **NOR** recibe ese 1 y reacciona apagándose, porque la **NOR** bota 0 si cualquiera de sus entradas es 1.
2) Por este procedimiento, técnicamente, se suman los retardos de trasmisión, la señal tarda el doble en estabilizarse porque tiene que cruzar dos alcabalas, que son las compuertas, en lugar de una sola.

4) ¿QUÉ VENTAJA OFRECE SYSTEMC FRENTE A UN LENGUAJE DE SIMULACIÓN VISUAL COMO LOGISIM?:

Desde lo que he podido aprender, conocer, y practicar pienso que **Logisim** es genial para aprender con dibujitos y cables de colores, en nuestros primeros pasos, pero pienso también que se queda corto cuando las cosas se ponen más serias.

Es por este motivo que en potencia y velocidad: **SystemC** es **C++** puro y fuerte. Eso compila y corre a la velocidad de la luz. **Logisim** he visto que es mucho más lento simulando.

En flexibilidad opino que: en **Logisim**, si hacemos un procesador gigante, la pantalla se llena de cables y al final no entenderíamos nada. Y en **SystemC**, como es de código, podemos describir sistemas inmensos, como, **por ejemplo:** el chip de un teléfono sin volvernos loco dibujando linecitas.

A nivel profesional: pienso que **SystemC** lo usan en las grandes industrias de verdad para diseñar hardware antes de fabricarlo. En cambio, **Logisim** es más como para uso académico, de aprendizaje, comparado con la robustez de **SystemC.**

En conclusión, analizo que **Logisim** es más para entender la lógica en nuestros primeros pasos; Y **SystemC** es para construir hasta una nave espacial si deseamos hacerlo y por supuesto si tenemos ese gran conocimiento tan amplio que necesitaríamos tener.

EJECUCIÓN DE MI PROYECTO:

```
esnaider96@PCD-ESNAIDERM:/mnt/c/users/esnai/OneDrive/Desktop/systemc-3.0.2$ cd ~/Lab2-SystemC
esnaider96@PCD-ESNAIDERM:~/Lab2-SystemC$ ls -la
total 112
drwxr-xr-x 2 esnaider96 esnaider96  4096 Nov 27 23:33 .
drwxr-x--- 6 esnaider96 esnaider96  4096 Nov 27 22:42 ..
-rwxr-xr-x 1 esnaider96 esnaider96 79360 Nov 27 23:33 logic_gates
-rw-r--r-- 1 esnaider96 esnaider96  1627 Nov 27 23:33 main.cpp
-rw-r--r-- 1 esnaider96 esnaider96   265 Nov 27 23:14 nand_gate.cpp
-rw-r--r-- 1 esnaider96 esnaider96   260 Nov 27 23:16 nor_gate.cpp
-rw-r--r-- 1 esnaider96 esnaider96   253 Nov 27 23:10 or_gate.cpp
-rw-r--r-- 1 esnaider96 esnaider96   256 Nov 27 23:25 xnor_gate.cpp
-rw-r--r-- 1 esnaider96 esnaider96   267 Nov 27 23:13 xor_gate.cpp
esnaider96@PCD-ESNAIDERM:~/Lab2-SystemC$ g++ main.cpp -I /mnt/c/users/esnai/OneDrive/Desktop/systemc-3.0.2/include \
-L /mnt/c/users/esnai/OneDrive/Desktop/systemc-3.0.2/lib-linux64 \
-lsystemc -lm -o logic_gates
esnaider96@PCD-ESNAIDERM:~/Lab2-SystemC$ export LD_LIBRARY_PATH=/mnt/c/users/esnai/OneDrive/Desktop/systemc-3.0.2/lib-linux64:$LD_LIBRARY_PATH
esnaider96@PCD-ESNAIDERM:~/Lab2-SystemC$ ./logic_gates

        SystemC 3.0.2-Accellera --- Nov 26 2025 23:42:51
        Copyright (c) 1996-2025 by all Contributors,
        ALL RIGHTS RESERVED
A B | OR XOR NAND NOR XNOR
-------------------------
0 0 |  0   0    1   1    1
0 1 |  1   1    1   0    0
1 0 |  1   1    1   0    0
1 1 |  1   0    0   0    1
esnaider96@PCD-ESNAIDERM:~/Lab2-SystemC$
```