Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL 3313 Taller de Diseño Digital
I Semestre 2025

TEC | Tecnológico de Costa Rica

Instructivo de Laboratorio

Lab. 1: Introducción al diseño digital con HDL y herramientas EDA de síntesis

Profesor

Dr.-Ing. Jorge Castro-Godínez
Dr.-Ing. Roberto Molina Robles

Basado en instructivos con autoría de

Dr.-Ing. Pablo Alvarado Moya
Dr.-Ing. Pablo Mendoza Ponce
Jeferson González Gómez, M.Sc.

1. Introducción

Este laboratorio introduce el diseño de circuitos digitales por medio de lenguajes de descripción de hardware (HDLs, *Hardware Description Languages*). Mediante un HDL se puede describir la especificación, comportamiento o estructura de un módulo de hardware, empleando una especificación programática. Pese a su similitud con un lenguaje de programación (tipos de datos, estructuras básicas, sintaxis, entre otros), los HDLs se emplean para **describir hardware**, por lo que para todo diseño se debe tener conocimiento completo de qué componente se está diseñando. Esto implica la realización previa de tablas de verdad, diagramas de estados, diagramas de bloques, etc., según sea necesario.

En este laboratorio, y durante el resto del curso, se trabajará con *SystemVerilog*. Para más información sobre el HDL, puede consultar las referencias [2, 3, 4, 1].

A lo largo de este laboratorio se desarrollarán varios ejercicios que le permitirán ganar más experiencia con descripción de hardware y su uso para el proceso de diseño, implementación y validación de circuitos digitales básicos.

Indicaciones generales:

- Lea y trate de comprender todo el trabajo solicitado antes de iniciarlo.
- En la documentación de los módulos a diseñar en este laboratorio, usted debe incluir las tablas de verdad y diagramas de bloque correspondientes a cada diseño desarrollado.
- Para la presentación funcional se le pedirá que muestre los circuitos propuestos, tanto a nivel de implementación como de simulación.
- Asegúrese de que le avance del laboratorio quede protocolado con suficientes *commits* en el repositorio.

Los sistemas desarrollados en este laboratorio deben ser completamente **sintetizables**. Tome en cuenta que dado que se está desarrollando lógica combinacional, los bloques desarrollados deben estar libres de elementos de memoria, especialmente *latches*. Compruebe, por medio de los mensajes y visualización del *netlist* de la herramienta, que no se generen dichos elementos como parte del proceso de síntesis. Las simulaciones finales de cada ejercicio deben ser presentadas a nivel de **post-síntesis**.

2. Cuestionario previo

Para el desarrollo de este laboratorio se deben responder las siguientes preguntas.

1. Explique qué es el modelado de comportamiento y de estructura en diseño digital. Brinde un ejemplo de cada uno empleando SystemVerilog.
2. Explique el proceso de síntesis lógica en el diseño de circuitos digitales, tanto para el desarrollo de un ASIC como para una FPGA.
3. Investigue sobre la tecnología de FPGAs. Describa el funcionamiento de la lógica programable en general, así como los componentes básicos de un chip de FPGAs.

3. Procedimiento

A continuación se presentan 5 ejercicios prácticos los cuales debe resolver de manera completa utilizando descripción de hardware. Documente en su bitácora todos las pasos de diseño que le llevan a la solución del problema. Con respecto a las simulaciones (*testbench*), estas deben ser ejecutadas tanto a nivel de comportamiento (*behavioral*) como en **post-síntesis**. Además, estas pruebas deben auto-comprobar (validar) los resultados.

3.1. Ejercicio 1. Switches, botones y LEDs

1. Diseñe un módulo que reciba como entradas las señales producidas por los 16 *interruptores* disponibles en la tarjeta con FPGA que se utilizará en el curso.
2. Las salidas del módulo se mostrarán en los 16 LEDs disponibles en la tarjeta de desarrollo.
3. El bloque a desarrollar debe subdividir los 16 *interruptores* en 4 grupos de 4. Lo mismo aplica para los LEDs. Cada grupo de 4 interruptores estará *gobernado* por un botón (*push button*) diferente en la tarjeta. De esta manera, al presionar un botón, el grupo de LEDs correspondiente se mantendrá apagado sin importar la combinación de interruptores del grupo asignado (los otros grupos no deben ser afectados).
4. Implemente un banco de pruebas (*testbench*) para validar el funcionamiento de su diseño. Realice su validación a nivel post-síntesis.
5. Realice una validación de su diseño en la tarjeta con FPGA.

3.2. Ejercicio 2. Multiplexor 4-to-1

1. Diseñe un multiplexor de cuatro entradas, parametrizado, para un ancho de datos (*bus width*) de entrada y salida variable.
2. Realice un banco de prueba (*testbench*) en el que se muestre de manera simple el funcionamiento del multiplexor. Muestre el resultado de la prueba para anchos de datos de 4, 8 y 16 bit, para los cuatro selecciones de entrada y para al menos un conjunto de datos de entrada de 50 datos en cada caso. Su *testbench* debe autovalidar las salidas esperadas.
3. Ejecute su prueba y asegúrese de que todos los errores y las advertencias producidas por la simulación hayan sido debidamente atendidas y corregidas.

3.3. Ejercicio 3. Decodificador para display de 7 segmentos

1. Diseñe un decodificador para un *display* de 7 segmentos. Este debe recibir un valor binario, representado con 4 bits, y mostrar su representación correspondiente en hexadecimal.
2. Valide el funcionamiento de su diseño mediante un *testbench* exhaustivo.
3. Modifique su diseño de manera que 4 grupos de 4 *switchs* provean un potencial valor a mostrar en el *display* de 7 segmentos, como se ilustra en la Figura 1. Estas entradas estarán multiplexadas al *display* de 7 segmentos (`sw 1-4`, `sw 5-8`, `sw 9-12`, y `sw 13-16`) y mediante dos de los botones (*push button*) disponibles (`btn 1-2`) realizará la selección de cuál de las 4 entradas se mostrará.
4. Implemente un *testbench* exhaustivo con el que valide el correcto funcionamiento de su diseño. Realice su validación a nivel post-síntesis.
5. Realice una validación de su diseño en la tarjeta con FPGA que se utilizará a lo largo del curso.

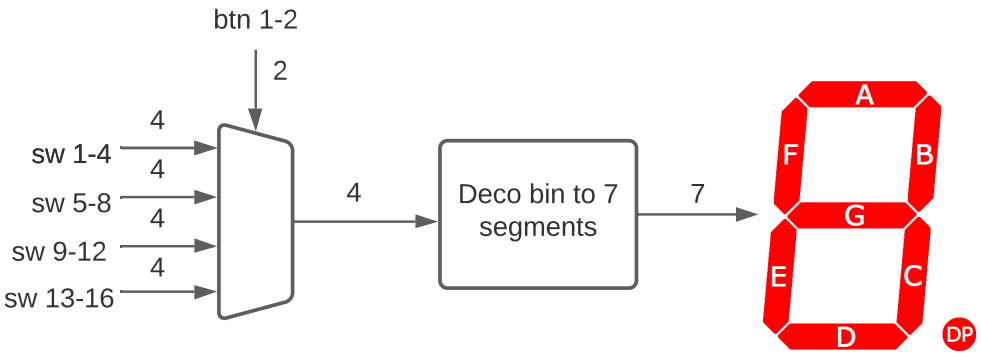

Figura 1: Diagrama de bloques para el decodificador de un *display* de 7 segmentos.

3.4. Ejercicio 4. Sumador y ruta crítica

1. Empleando un sumador completo de 1 bit, diseñe un *Ripple-Carry Adder* (RCA) con tamaño de palabra parametrizable (por defecto 8 bits).

2. Realice un banco de prueba (*testbench*) para el sumador del punto anterior. Evalué el funcionamiento del RCA para todos los posibles casos de datos de entrada. No olvide los casos en los que el sumador produce un *overflow*.

3. Ejecute su prueba y asegúrese de que todos los errores y las advertencias producidas por la simulación y síntesis hayan sido debidamente atendidas y corregidas.

4. Sintetice un RCA de 64-bits. Establezca como parte de sus *constraints* para la síntesis una frecuencia de reloj de 10 MHz. Corra una prueba en la que los datos de entrada al sumador cambien a una frecuencia de 100 M-samples/s. ¿Es el sumador capaz de producir resultados correctos?

5. Implemente un sumador que sea algo más eficiente que el RCA. Implemente un *Carry-Lookahead Adder* (CLA) de 16 bits y compárelo con un RCA de 16 bits en términos de área y retardo.

3.5. Ejercicio 5. Unidad aritmética lógica (ALU)

En este ejercicio deberá diseñar una ALU parametrizable de n bits, como se muestra en la Figura 2.

Dicha unidad deberá tomar dos números de n bits como entrada, así como un bus de control, denominado `ALUControl`, que debe permitir seleccionar cuál de las operaciones aplicar. Debe respetar los nombres indicados en la figura en la interfaz de su módulo. Las diferentes operaciones que debe realizar la ALU (incluyendo su código de operación en hexadecimal) son:

- `0h` - and,
- `1h` - or,
- `2h` - suma (en complemento a dos),
- `3h` - incrementar en uno el operando
- `4h` - decrementar en uno el operando

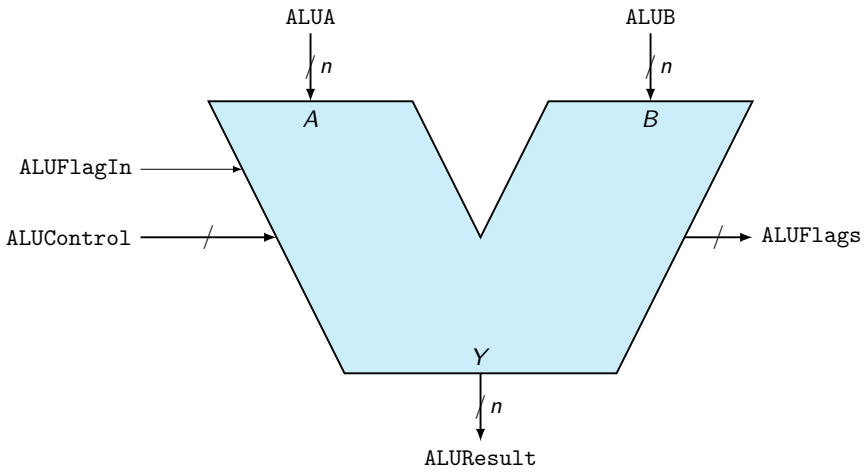

Figura 2: Diagrama de bloques de la ALU a diseñar.

- `5h` - not (sobre un operando),
- `6h` - resta (en complemento a dos),
- `7h` - xor,
- `8h` - corrimiento a la izquierda del operando A, y
- `9h` - corrimiento a la derecha del operando A.

La bandera de entrada `ALUFlagIn` representa el acarreo de entrada para la suma y resta, o el bit de entrada en los corrimientos hacia la izquierda o derecha. Su función es permitir conectar 2 ALU para proveer el doble del número de bits en estas operaciones. En las operaciones de negación, incremento y decremento, esta bandera selecciona cuál de los operandos utilizar (0 debe seleccionar al operando A y 1 al operando B). En el resto de operaciones lógicas, esta señal es ignorada.

Para las operaciones de corrimiento a la derecha o la izquierda, el operando a desplazar es siempre dado por A y cuántos bits a desplazar es siempre dado por B. La bandera `ALUFlagIn` indica con qué valor deben rellenarse los nuevos bits introducidos, y la bandera de salida C indica cual fue el "último" bit que salió de la ventana representada.

Por ejemplo, para una ALU de 8 bit, si $A = 1101\,0110_2$, y $B = 0000\,0011_2$, `ALUFlagIn` $= 1$, entonces, en un desplazamiento a la izquierda el resultado es $Y = 1011\,0111$, con $C = 0$, mientras que para un desplazamiento a la derecha el resultado es $Y = 1111\,1010$ con $C = 1$.

Adicionalmente la ALU deberá generar una bandera de resultado Cero (Z). Dicha bandera es uno si el resultado es cero independientemente de la operación lógica o aritmética seleccionada.

Con base en la descripción anterior:

1. Diseñe la ALU con un modelo utilizando *SystemVerilog*. Parta de los circuitos básicos (sumadores, restadores, de corrimiento, etc.) que considere necesarios. Muestre los diagramas de bloques, tablas de verdad y circuitos de cada módulo en el diseño.

2. Realice al menos un *testbench* de auto-chequeo, usando *SystemVerilog*, en el que se muestre de manera simple el funcionamiento **completo** de la ALU en 4 bits.

4. Evaluación

Las fechas oficiales de inicio y fin de este laboratorio son:

- **Inicio:** 19.02.2025
- **Cuestionario previo:** 26.02.2024
- **Planteamiento de la solución:** 28.02.2024
- **Fin:** 07.03.2024 (demostración funcional)
- **Entrega de documentación:** 07.03.2024

El presente laboratorio tiene un valor del **6 %** de la nota final del curso y se evaluará de la siguiente manera:

1.	Cuestionario previo	10 %
2.	Planteamiento del diseño	10 %
3.	Funcionalidad final	60 %
4.	Documentación	20 %
5.	Planeamiento	0 %
6.	Repositorio	0 %

Referencias

[1] Pong P. Chu. *FPGA Prototyping by SystemVerilog Examples*. Wiley, 2012.

[2] David Harris and Sarah Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, 2022.

[3] Stuart Sutherland, Simon Davidmann, and Peter Flake. *SystemVerilog for Design*. Springer, 2 edición, 2006.

[4] Donald Thomas. *Logic Design and Verification Using SystemVerilog*. CreateSpace, 2016.