Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL 3313 Taller de Diseño Digital
II Semestre 2024

TEC | Tecnológico de Costa Rica

Instructivo de Laboratorio

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

Profesor

Kaleb Alfaro Bonilla, M.Sc.

Basado en instructivos con autoría de

Dr.-Ing. Pablo Alvarado Moya
Jeferson González Gómez, M.Sc.
Roberto Molina Robles, M.Sc.
Dr.-Ing. Jorge Castro-Godínez
Dr.-Ing. Pablo Mendoza Ponce

1. Introducción

Este laboratorio introduce el diseño de circuitos digitales por medio de lenguajes de descripción de hardware (HDLs, *Hardware Description Languages*). Un HDL es una forma de describir la especificación, comportamiento o estructura de un módulo de hardware, por medio de 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 el HDL llamado *SystemVerilog* en simulación y *Verilog* en descripción de hardware. Para más información sobre *SystemVerilog*, 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 empleando *SystemVerilog*/*Verilog* y su uso para el proceso de diseño, implementación y validación de circuitos digitales básicos.

- 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, así como evidencias de la documentación en el mismo repositorio.
- Toda documentación elaborada y respuestas al cuestionario previo debe estar localizable y accesible desde el reposotorio del laboratorio.

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. Las simulaciones finales de cada problema deben ser presentadas a nivel de **post-síntesis**.

2. Cuestionario previo

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

1. Investigue las características de las familias TTL bajo las series 74$*xx$ en particular las diferencias entre las variantes $* \in \{$L, LS y HC$\}$.
2. Investigue las características de la familia CMOS 4000.
3. Investigue qué cuidados deben tenerse al manipular las tecnologías CMOS.
4. Investigue el significado de los parámetros V_{IL}, V_{IH}, V_{OL}, V_{OH}, I_{IK}, I_{OK}
5. Investigue qué son los tiempos de propagación t_{PD}, t_{PLH} y t_{PHL} y los tiempos de transición t_t, t_r y t_f.
6. Investigue qué significa el término *fan-out* y cuáles valores típicos se encuentran en las familias TTL y CMOS.
7. Para cada una de las variantes TTL y CMOS especifique en una tabla:
   a) rango de tensión eléctrica de alimentación V_{CC} o V_{DD}, V_{SS}
   b) rango de tensiones de entrada y salida
   c) tiempos de propagación y transición
8. Revise la hoja de datos de los circuitos integrados 74$*$00,74$*$02,74$*$04,74$*$14, 4001, 4011, 4069 y 40106. Resuma para qué sirve cada uno.

9. Revise la estructura básica, a nivel de transistores, de una compuerta NAND en circuitos integrados CMOS.

10. Investigue sobre el concepto y el uso de los de circuitos *pull-up* y *pull-down* en electrónica digital.

11. Investigue qué es un circuito disparador Schmitt (*Schmitt trigger*). Revise las características técnicas del circuito 74∗14.

12. Investigue en que consiste la modulación de ancho de pulso (PWM).

13. Investigue qué es el efecto de rebote y típicos circuitos anti-rebote (*debouncing circuits*)

14. Explique qué es el modelado de comportamiento y de estructura en diseño digital. Brinde un ejemplo de cada uno.

15. Explique el proceso de síntesis lógica en el diseño de circuitos digitales, tanto para el desarrollo de un ASIC como para una FPGA.

16. Investigue sobre la tecnología de FPGAs. Describa el funcionamiento de la lógica programable en general, así como los componentes básicos de una.

17. Investigue sobre los proyectos de YOSYS y nextpnr ¿Cuáles son sus funciones como herramientas?

3. Procedimiento

A continuación se presentan 5 ejercicios prácticos, los cuales debe resolver de manera completa utilizando circuitos integrados discretos y/o descripción de hardware con *Verilog*. Documente en su repositorio todos las pasos de diseño que le llevan a la solución del problema.

Con respecto a las simulaciones (*testbench*), estas deben ser ejecutados a nivel de comportamiento (*behavioral*). Además, estas pruebas deben auto-comprobar los resultados. De preferencia describa sus simulaciones utilizando *SystemVerilog*.

3.1. Ejercicio 1. Circuitos digitales discretos

1. La Figura 1 muestra el diagrama de bloques de un codificador para matrices de 16 teclas. Este codificador permite extraer un valor de 4 bits en binario natural que representa la **posición** de la tecla presionada. En el caso de este laboratorio se diseñarán los bloques marcados en verde.

2. ¿Qué tipo de bloque combinacional se requiere para cada bloque a implementar? Justifique su respuesta en su diseño. Diseñe cada bloque usando álgebra booleana o mapas de Karnaugh. Minimice su circuito de tal manera que pueda implementarse con el **mínimo número de circuitos integrados**. Su diseño debe ser simplificado para usar **únicamente** compuertas NOT, NAND y/o NOR.

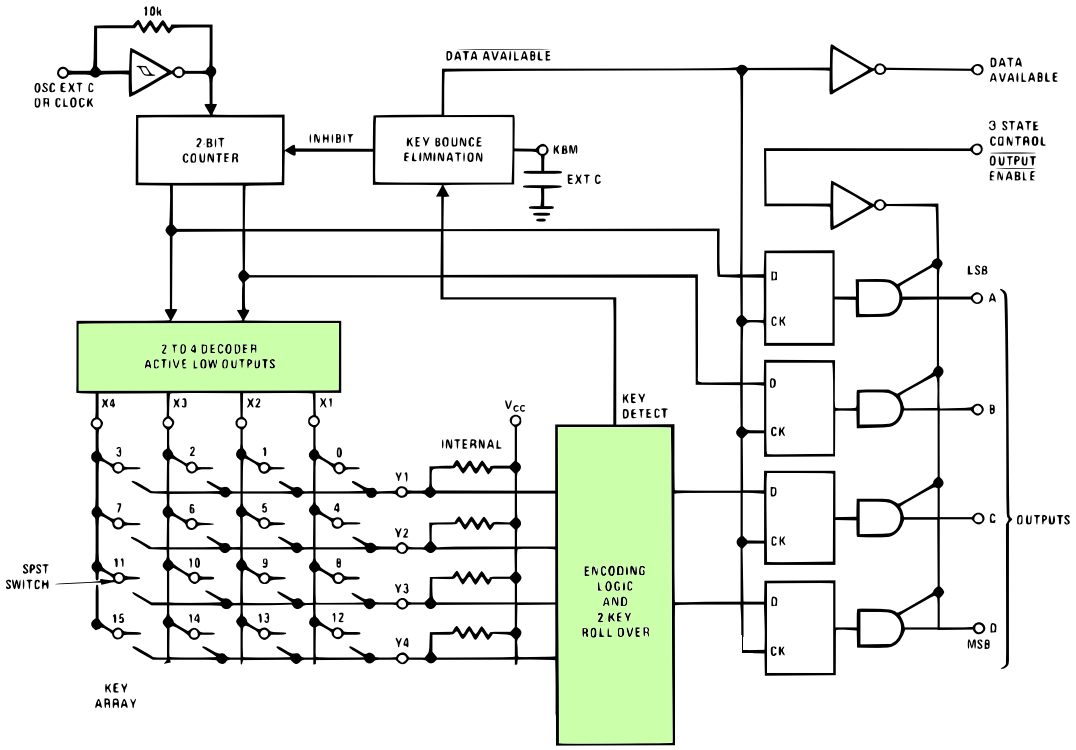

Figura 1: Diagrama de bloques para el codificador de matrices de 16 teclas MM74C922 (adaptado de su hoja de datos).

3. Implemente sus circuitos en una *protoboard* junto con un teclado alfanumérico. Para las conexiones debe basarse en el diagrama de la Figura 1. Para sus pruebas iniciales, haga uso de botones o un *dipswitch* en lugar del *contador de dos bits* y muestre sus salidas en un grupo de LEDs.
4. Tome en cuenta que su sistema debe funcionar con una tensión de alimentación de 3.3V.
5. Las resistencias de *pull-up* deben ser colocadas, o de otro modo el circuito no va a funcionar.
6. El panel de botones se puede adquirir o tomar prestado de bodega. Este se encuentra en el kit principiante de la raspberry pi.

3.2. Ejercicio 2. Switches, botones y LEDs

1. Diseñe un módulo que reciba como entradas los 4 *interruptores* hacia los pines de la FPGA que se utilizará en este laboratorio.
2. Las salidas del módulo se mostrarán en los 4 LEDs disponibles en la tarjeta de desarrollo.
3. El bloque a desarrollar debe convertir el código ingresado por los interruptores por el correspondiente a su complemento a 2 hacia los LEDs.
4. Implemente un banco de pruebas (*testbench*) para validar el funcionamiento de su diseño.

5. Realice una validación de su diseño en la tarjeta con FPGA.

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

3.4. Ejercicio 4. Modulación de ancho de pulso

1. Diseñe un módulo que reciba como entrada un código de 4 bits por medio de interruptores.
2. El módulo debe utilizar dicha entrada como código de nivel para una modulación PWM que se enviará como salida a un pin de la FPGA. El ancho del pulso debe ser alrededor de 1ms.
3. Se colocará como carga un LED de la tarjeta.
4. Valide el funcionamiento de su diseño mediante un *testbench* exhaustivo.
5. Realice una validación de su diseño en la tarjeta con FPGA que se utilizará a lo largo del curso.

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

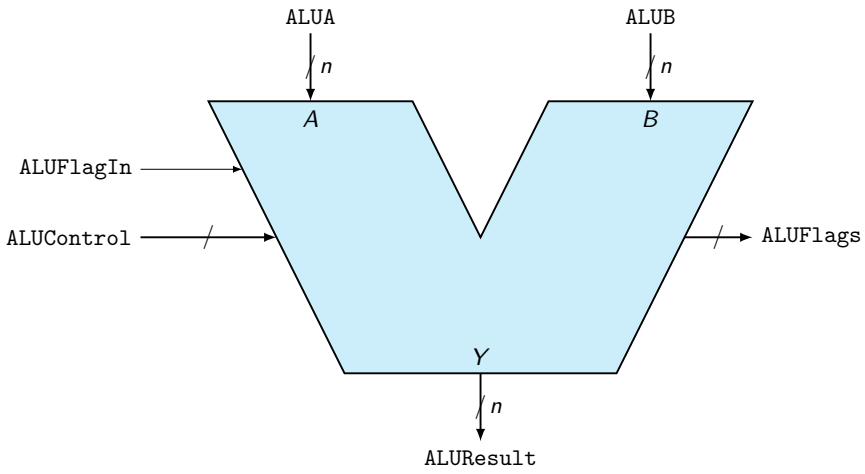

Figura 2: Diagrama de bloques de la ALU a diseñar.

- `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 *Verilog*. 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. Las pruebas deben ser elaboradas de manera aleatoria.

4. Evaluación

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

- **Inicio:** 26.07.2024
- **Cuestionario previo:** 03.08.2024
- **Fin:** 21.08.2024 (demostración funcional)
- **Entrega de documentación:** 21.08.2024

El presente laboratorio tiene un valor del **20 %** de la nota final del curso y se evaluará de la siguiente manera:

1. Cuestionario previo 10 %
2. Funcionalidad final 40 %
3. Documentación 30 %
4. Repositorio (control de versiones, división de tareas) 20 %

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.