INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA ELECTRÓNICA
EL3313 TALLER DE DISEÑO DIGITAL
PROF. M.SC. KALEB ALFARO BADILLA
GRUPO 20
I SEMESTRE 2024

Laboratorio 1: Introducción al diseño digital con HDLs y herramientas EDA de síntesis

---

DOCUMENTACIÓN

Realizado por:

Anthony Artavia Salazar
Yailyn Priscilla Campos Zamora
Samuel Montenegro Gómez

29/02/2024

Índice

1. EJERCICIO 1

En respuesta al inciso 2, se hizo un análisis del número de entradas y salidas para cada bloque, llegando a la conclusión de que el primer bloque verde, que posee 2 entradas y 4 salidas, cumple el requisito para ser un decodificador, esto porque para que un bloque sea decodificador debe tener m entradas y 2^m salidas, lo cual se aprecia en la figura 1.1. Además, siguiendo el mismo análisis, el segundo bloque es un codificador dado que en su definición requiere n entradas y m salidas.

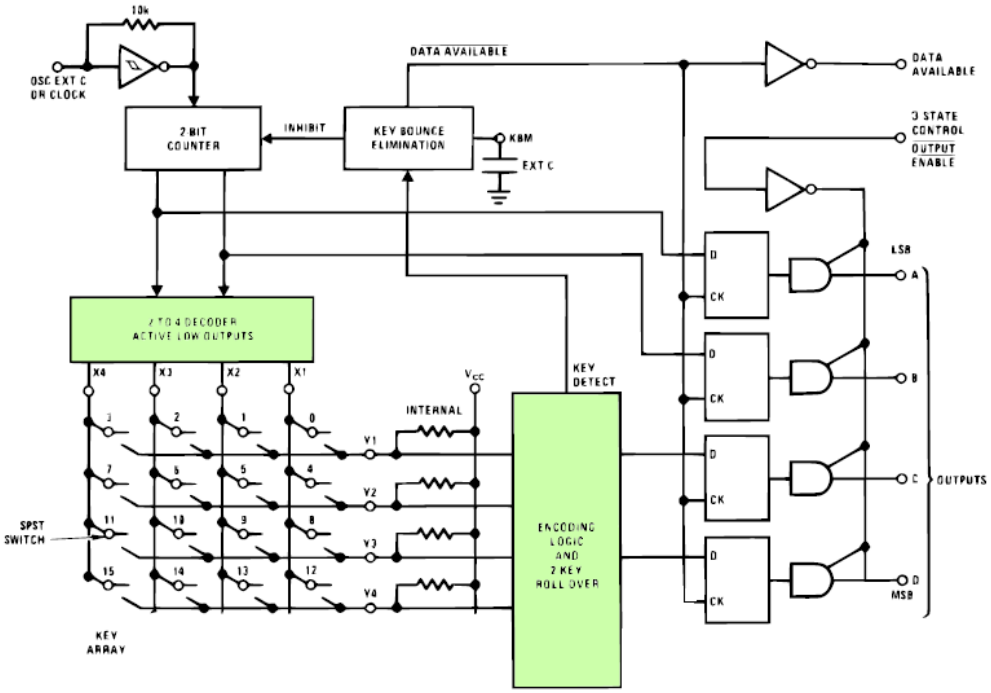

Figura 1.1: Diagrama de bloques para el codificador de matrices de 16 teclas MM74C922 (adaptado de su hoja de datos.

Siguiendo con la solución del inciso 2, se procedió a realizar tablas de verdad de cada bloque con el fin de obtener el respectivo mapa de Karnaugh y así encontrar las ecuaciones lo más simplificadas posibles. A continuación se adjuntan las tablas de verdad.

Tabla de verdad del decodificador

A	B	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Tabla de verdad del codificador

A	B	C	D	Y1	Y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Figura 1.2: Tablas de verdad para cada bloque.

Conociendo las tablas de verdad, por consiguiente ya es posible iniciar con el proceso para el desarrollo de las tablas de Karnaugh. Al haber más de una salida, es necesario hacer Karnaugh por cada salida, la resolución se obtuvo de la siguiente manera:

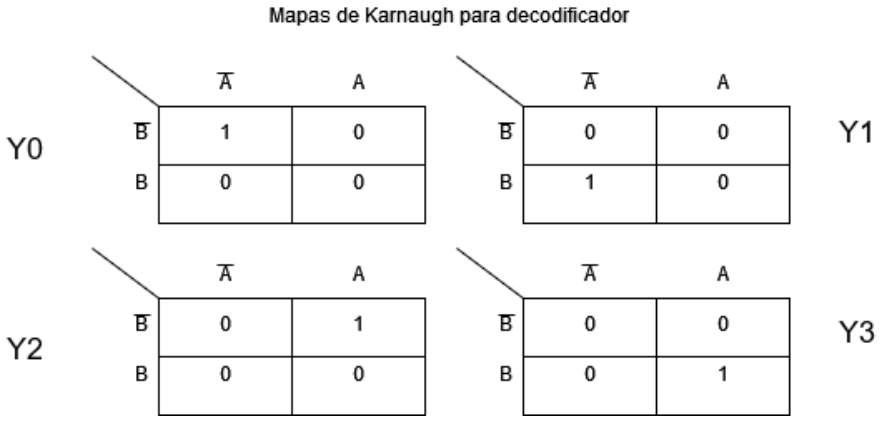

Figura 1.3: Mapa de Karnaugh para decodificador.

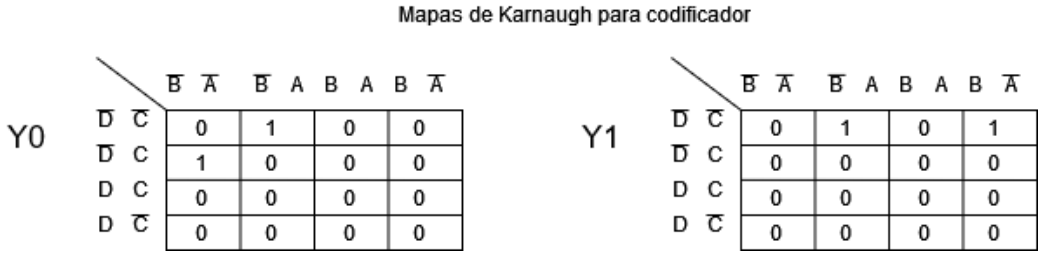

Figura 1.4: Mapa de Karnaugh para codificador.

Observando los mapas de Karnaugh que se recolectaron, finalmente se procede a documentar las ecuaciones obtenidas para posteriormente ensamblar un circuito con solo compuertas $NOR, NAND$ y NOT. Las ecuaciones de salida para el decodificador son:

- $$Y_0 = \overline{AB}$$
- $$Y_1 = \overline{A}B$$
- $$Y_2 = A\overline{B}$$
- $$Y_3 = AB$$

Posteriormente, las ecuaciones de salida para el codificador son:

- $$Y_0 = \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D}$$
- $$Y_1 = \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D}$$

Finalmente, el circuito integrado con solo compuertas $NOR, NAND$ y NOT fue diseñado de la siguiente manera

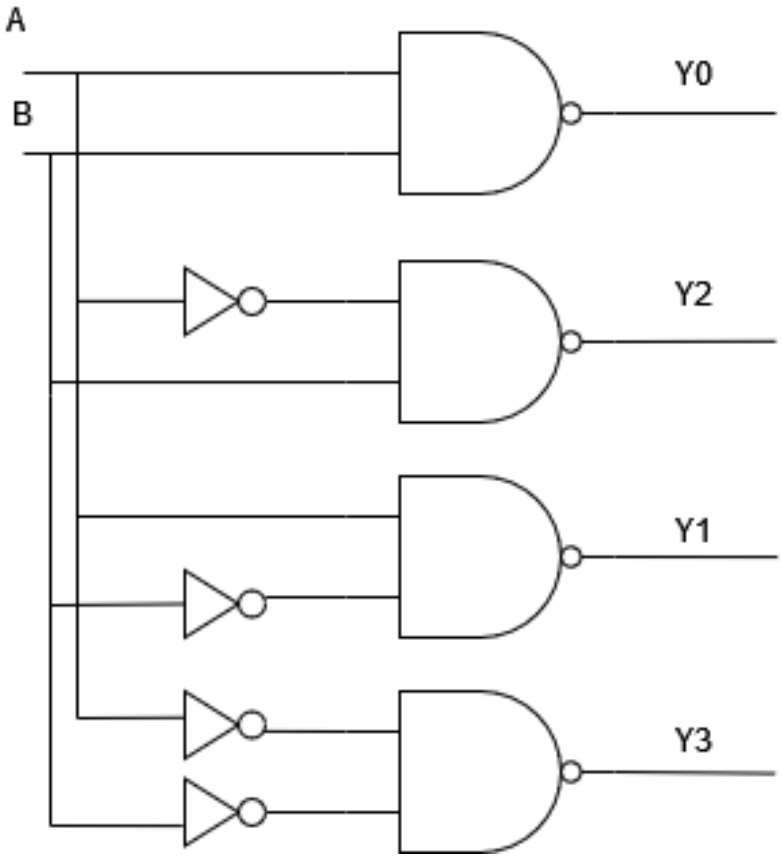

Figura 1.5: Circuito integrado para decodificador.

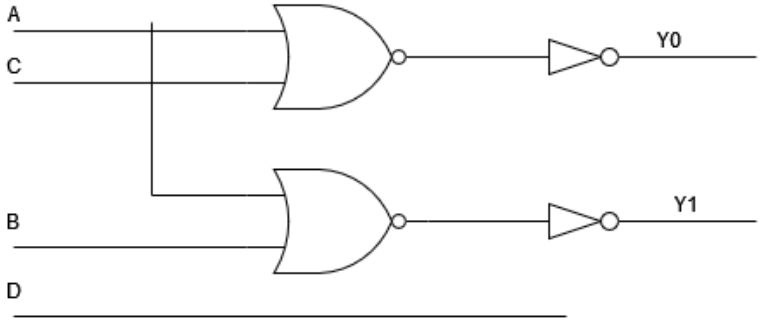

Figura 1.6: Circuito integrado para codificador.

Respecto al inciso 3, esta sección fue la de mayor causante de dificultades dado que no se tenía conocimiento previo sobre como implementar un teclado alfúmerico en una protoboard con la integración que requería el inciso, por ende, finalmente no se pudo obtener alguna información relevante en cuanto a la solución de este inconveniente. Factores como el tiempo, el poco conocimeiento, la falta de un teclado de ese tipo pudieron haber contribuido con estas dificultades, sin embargo, finalmente no se logró algún avance en la solución de esto.

2. EJERCICIO 2

Para este diseño lo primero que se hizo fue buscar los puertos de la FPGA en este documento: Nexys4 DDR™ FPGA Board Reference Manual. Habiéndolo estudiado, se procedió con la idea del diseño del módulo donde se realizaría lo solicitado; dicha idea quedó plasmada en el diagrama de bloques de la siguiente manera:

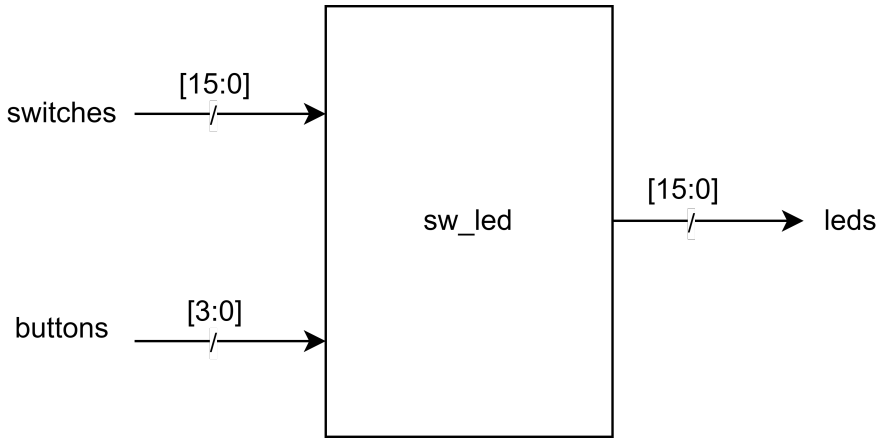

Figura 2.1: Diagrama de bloques para ejercicio 2.

La idea es que los subsets de los leds, que encienden con un 1 lógico, se mantengan apagados ó en 0 lógico mientras se mantenga presionado el button que los gobierna. Para la agrupación de los switches y su conexión con leds, la idea fue utilizar condicionales ternarios para cumplir la siguiente tabla de verdad, para cada led:

Switch	Button	Led
0	0	0
0	1	0
1	0	1
1	1	0

Con respecto a las complicaciones encontradas, solamente fue para el testbench porque no se sabía cómo generar número aleatorios, pero buscando en diversas fuentes de Google se encontró el comando urandom().

3. EJERCICIO 3

Para iniciar se añade la tabla de verdad para un multiplexor 4:1 [1]:

S_1	S_0	A	B	C	D	Out
0	0	0	x	x	x	0
0	0	1	x	x	x	1
0	1	x	0	x	x	0
0	1	x	1	x	x	1
1	0	x	x	0	x	0
1	0	x	x	1	x	1
1	1	x	x	x	0	0
1	1	x	x	x	1	1

Figura 3.1: Tabla de verdad de un multiplexor 4:1.

Ya mostrada la tabla de verdad, se procede con el diagrama de bloques de un multiplexor 4:1.

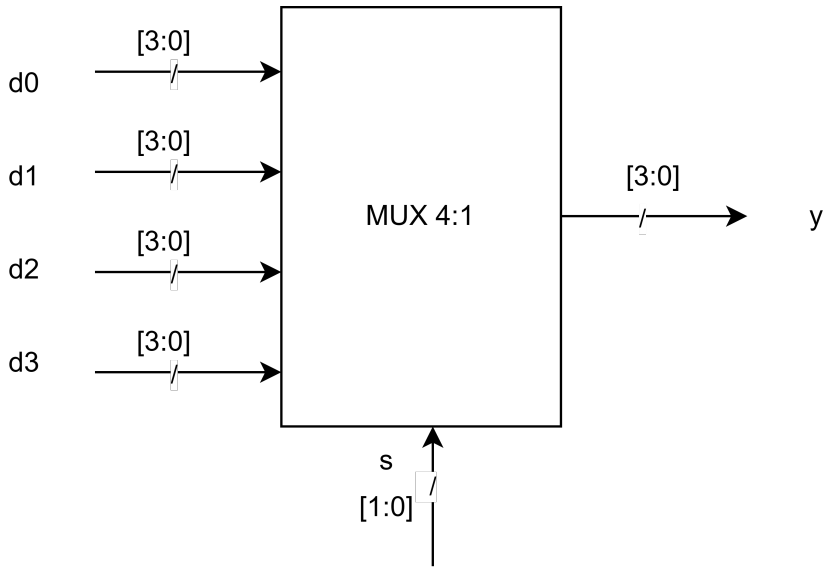

Figura 3.2: Diagrama de bloques de un multiplexor.

Con las figuras 3.1 y 3.2 fue posible realizar el multiplexor solicitado. Para el presente problema no se tuvieron dificultades, debido a que para el problema 2 ya se había solucionado el problema para solucionar números aleatorios con System Verilog gracias a la instrucción urandom().

4. EJERCICIO 4

Lo primero que se hizo fue diseñar el decodificador para el display de siete segmentos, la tabla de verdad es la siguiente:

Bits del número binario				Segmentos del display						
b_3	b_2	b_1	b_0	s_a	s_b	s_c	s_d	s_e	s_f	s_g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	1	1	1	1	1	1	0
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Figura 4.1: Tabla de verdad para decodificador de 7 segmentos.

Luego, usando productos de sumas se encontraron las ecuaciones booleanas para cada salida de la tabla, es decir, para cada segmento del display:

$$s_a = (b_3 + b_2 + b_1 + !b_0) \cdot (b_3 + !b_2 + b_1 + b_0)) \tag{4.1}$$

$$s_b = (b_3 + !b_2 + b_1 + !b_0) \cdot (b_3 + !b_2 + !b_1 + b_0) \cdot (!b_3 + !b_2 + b_1 + b_0) \cdot (!b_3 + !b_2 + !b_1 + b_0) \cdot (!b_3 + !b_2 + !b_1 + !b_0) \tag{4.2}$$

$$s_c = (b_3 + b_2 + !b_1 + b_0) \cdot (!b_3 + !b_2 + b_1 + b_0) \cdot (!b_3 + !b_2 + !b_1 + b_0) \cdot (!b_3 + !b_2 + !b_1 + !b_0) \tag{4.3}$$

$$s_d = (b_3|b_2|b_1|!b_0)\&(b_3|!b_2|b_1|b_0)\&(b_3|!b_2|!b_1|!b_0)\&(!b_3|b_2|b_1|!b_0)\&(!b_3|b_2|!b_1|b_0)\&(!b_3|!b_2|!b_1|!b_0) \tag{4.4}$$

$$s_e = (b_3|b_2|b_1|!b_0)\&(b_3|b_2|!b_1|!b_0)\&(b_3|!b_2|b_1|b_0)\&(b_3|!b_2|b_1|!b_0)\&(b_3|!b_2|!b_1|!b_0)\&(!b_3|b_2|b_1|!b_0) \tag{4.5}$$

$$s_f = (b_3|b_2|b_1|!b_0)\&(b_3|b_2|!b_1|b_0)\&(b_3|b_2|!b_1|!b_0)\&(b_3|!b_2|!b_1|!b_0) \tag{4.6}$$

$$s_g = (b_3|b_2|b_1|b_0)\&(b_3|b_2|b_1|!b_0)\&(b_3|!b_2|!b_1|!b_0)\&(!b_3|!b_2|b_1|b_0)\&(!b_3|!b_2|b_1|!b_0) \tag{4.7}$$

Dada la longitud y complejidad que representan estas ecuaciones, no fueron simplificadas con Karnaugh, por lo que se procedió entonces a ingresarlas así en el módulo del decodificador.

El testbench se hizo para cada posible dígito hexadecimal posible, y funciona correctamente. Además, para armar el decodificador completo se usó el multiplexor diseñado en el ejercicio 3, asimismo se diseñó un codificador para los switches, de modo que permitan generar ese número de 4 bits o dígito hexadecimal, el codificador funciona con concatenación de bits.

Posteriormente se crea un top module para decodificador completo, en este se interconectó los módulos del codificador de switches, el multiplexor 4:1 y decodificador para el display de siete segmentos. A continuación los diagramas de bloques respectivos.

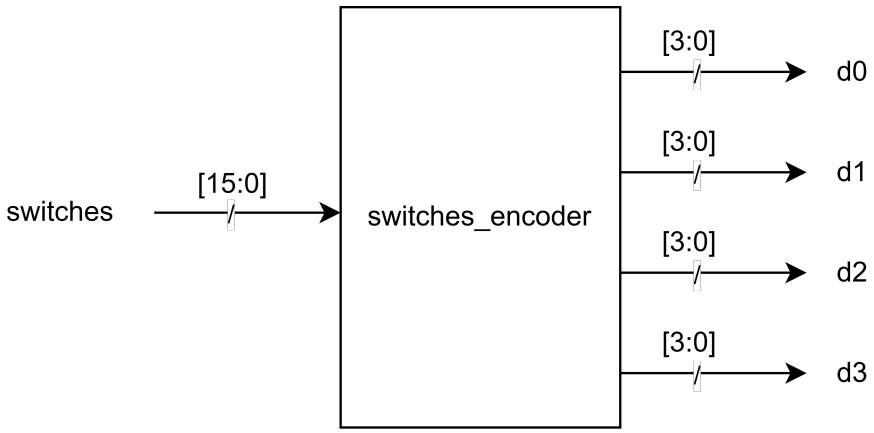

Figura 4.2: Diagrama de bloques para codificador.

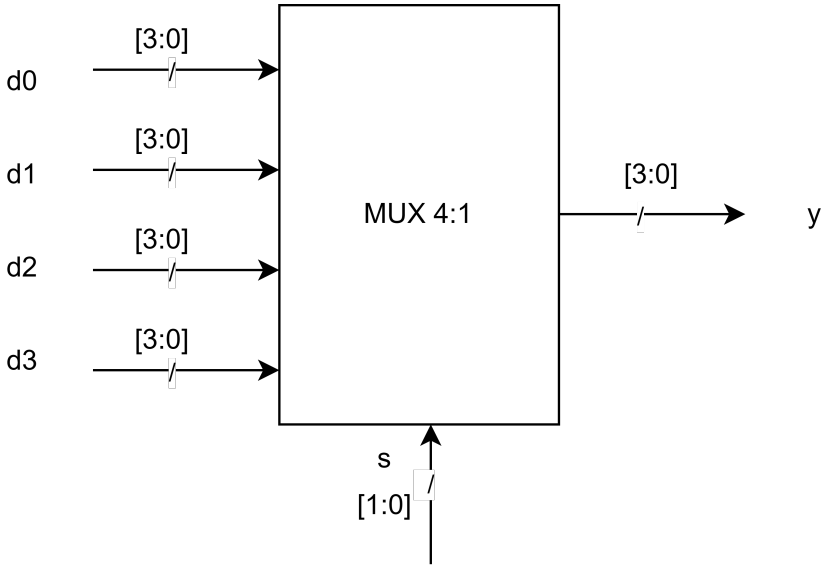

Figura 4.3: Diagrama de bloques del multiplexor.

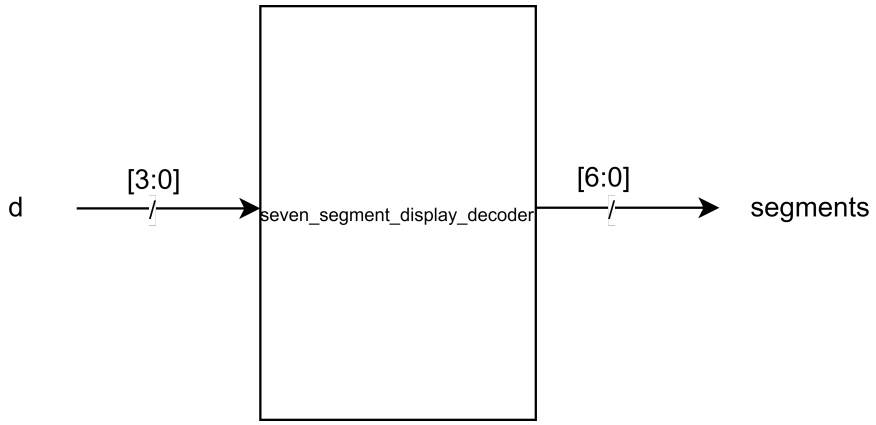

Figura 4.4: Diagrama de bloques del decodificador para display de siete segmentos.

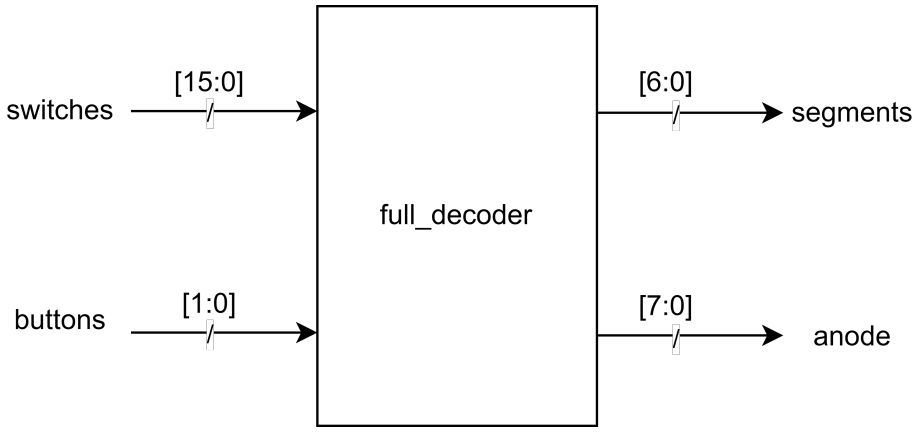

Figura 4.5: Diagrama del top module.

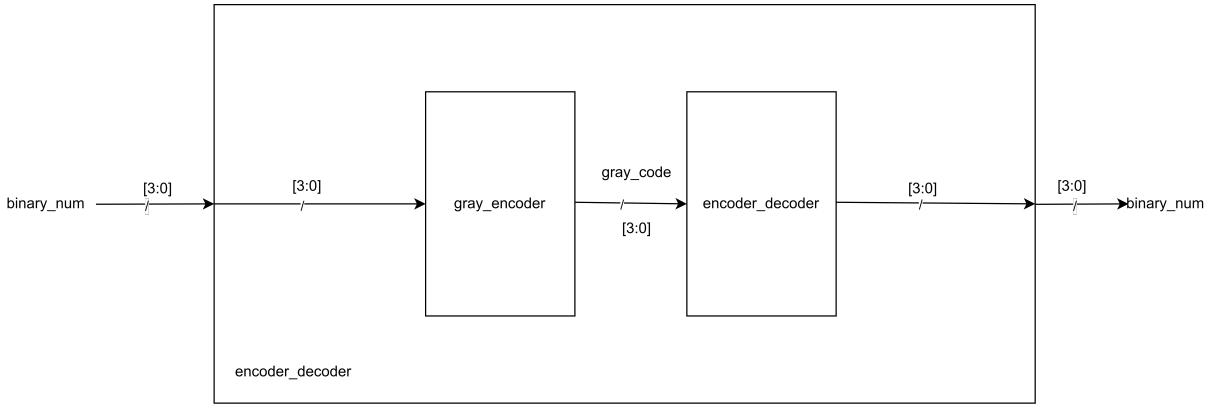

Figura 4.6: Interconexión de módulos.

Al momento de simular, el testbench muestra un comportamiento esperado para el decodificador, además

el diseño se sintetiza y genera el bitstream pero en la FPGA no se comporta como se espera. Por ende, en la sección de dificultades: primero se hizo un diseño como todo en uno que no incluía el codificador de switches, no servía y es que estaba haciendo varias asignaciones al mismo tiempo que no tenía sentido porque unas dependían de otras, pero se asignaban al mismo tiempo. Luego, para el segundo intentó se ocurrió empezar de cero, usar los módulos por aparte e instanciarlos en un top module para el decodificador completo, aquí se diseño el codificador para los switches, al probarlo había problemas por nombres de switches repetidos debido al copy/paste, también porque en algunas señales no se indicó el número de bits que eran. Luego de corregir esos errores, se hizo el testbench y muestra el comportamiento esperado, se sintetiza y genera el bitstream pero no se comporta como debería en la FPGA. No se pudo solucionar el comportamiento inesperado en la FPGA pues no se logró determinar qué pasa, si están mal las conexiones físicas o cual es el error. Otro problema fue que al inicio estaban según se había analizado encendiendo los segmentos con 1 pero encienden con 0. Cuando se observó se procedió a invertir las ecuaciones del decodificador.

5. EJERCICIO 5

La representación en código Gray de los números binarios de 4 bits es la siguiente, donde dos números adyacentes en una secuencia deben diferir en un único bit.

Bits del número binario				Bits del código Gray			
b3	b2	b1	b0	b'3	b'2	b'1	b'0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Figura 5.1: Representación de bits en binario y en código de gray.

La fórmula para pasar de binario a Gray y de Gray a binario fue obtenida de [2]. De aquí se extrae que “para convertir un número binario (en Base 2) a código Gray, simplemente se le aplica una operación XOR con el mismo número desplazado un bit a la derecha, sin tener en cuenta el acarreo.” Con base en el texto se hizo el codificador. También de esta fuente se obtuvo el algoritmo para el decodificador de Gray, es decir, pasar de Gray a binario. Luego simplemente se interconectaron estos módulos en un top module que se llama encoder-decoder. A continuación los respectivos diagramas de bloques

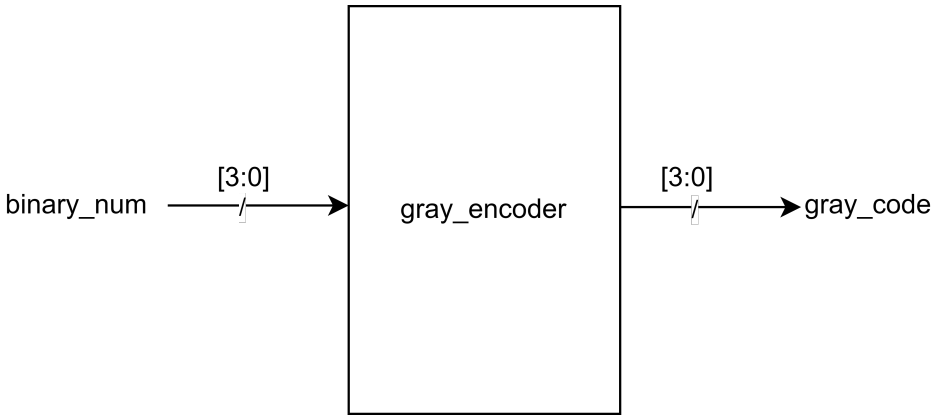

Figura 5.2: Diagrama del encoder.

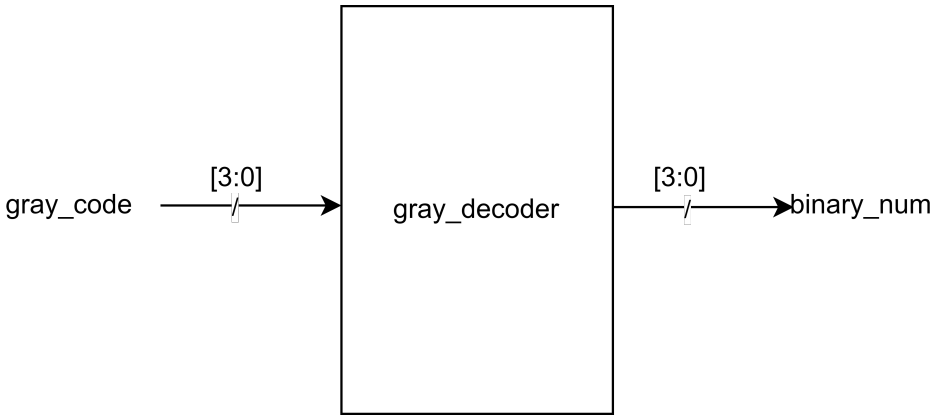

Figura 5.3: Diagrama del decoder.

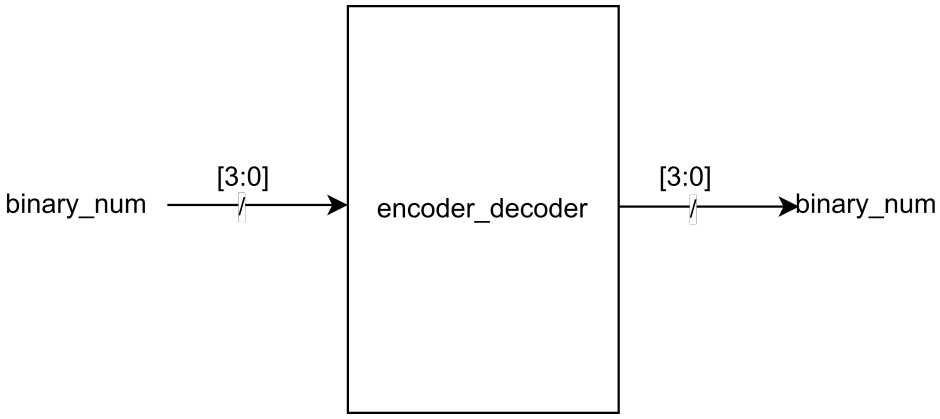

Figura 5.4: Diagrama del top module.

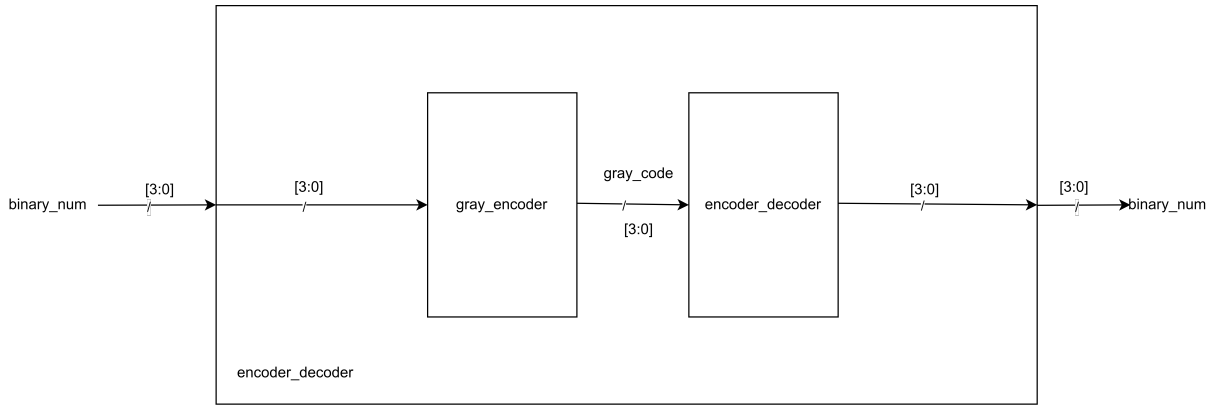

Figura 5.5: Diagrama de interconexión de módulos.

Para este ejercicio no hubieron casi dificultades, sino únicamente en Git hub se presentaron inconvenientes, los cuales fueron resueltos con ayuda del profesor.

6. EJERCICIO 6

Para implementar una ALU en System Verilog primero se inició por buscar información sobre una ALU, en relación a su definición, tipo de lógica (combinacional), entre otros que ayudaran sobre como se implementaría el diseño interno del mismo, esta información fue recolectada de [3]. El diagrama de bloque de nivel 1 brindado por el enunciado del ejercicio fue un gran apoyo para la identificación de entradas,salidas, señales de control. Posteriormente se procedió a realizar un diseño de la posible implementación con lápiz y papel para tener la seguridad de que la abstracción del funcionamiento fue el correcto. Haciendo esto, el primer diseño tenía falencias en el control de las ALUFlagIn y por ende fue una gran dificultad en el momento de programar. Habiéndose entendido las falencias, después de varios intentos se obtuvo el diagrama de bloques que representaría el diseño final de la ALU, a continuación se adjuntan dichos diagramas:

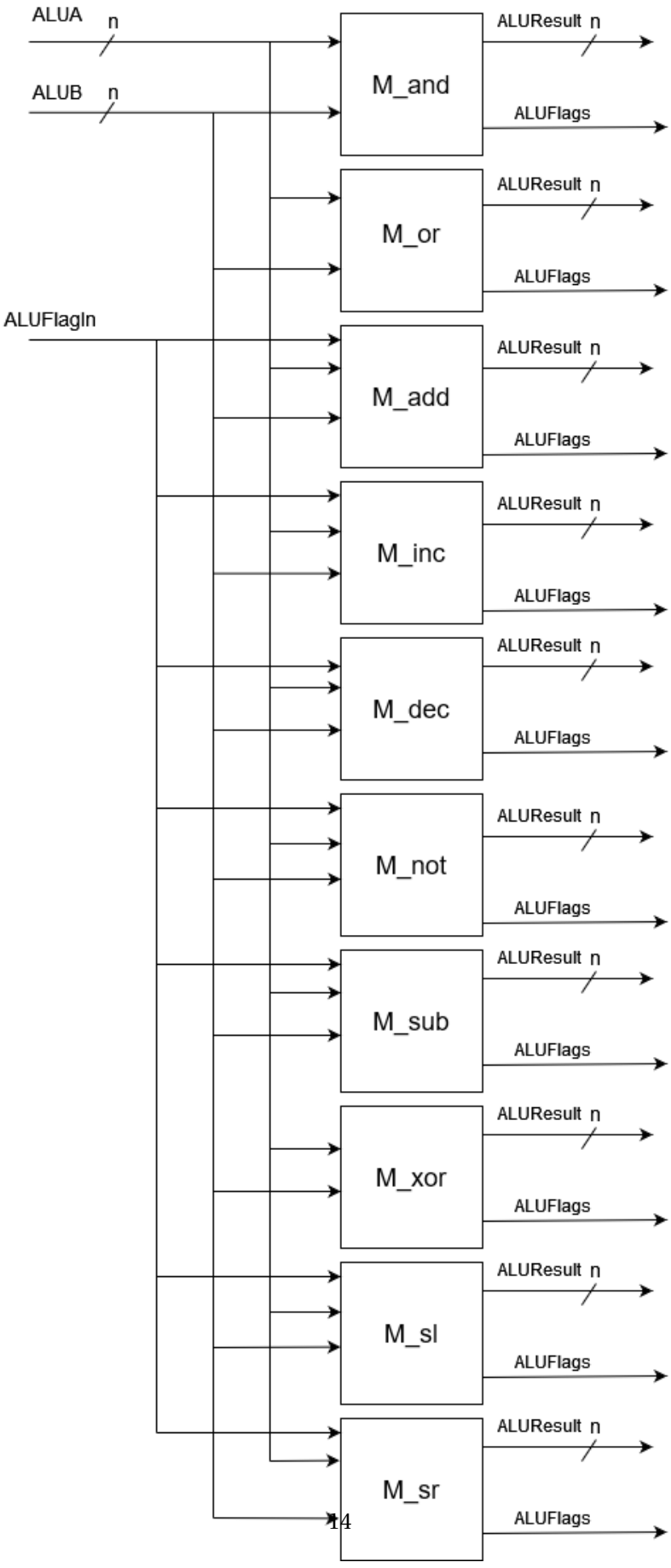

Figura 6.1: Diagrama de módulos para operación de ALU.

Dentro de cada módulo la implementación de las operaciones fue relativamente sencillo, ya que varios de ellas era solo utilizar el operando y ya System Verilog lo ejecutaba, sin embargo, también se presentaron dificultades, por ejemplo con el módulo de shift left y shift right, donde los operandos de System Verilog no eran útiles por la especificaciones del enunciado. El diagrama de bloques final de la ALU se hizo por medio de 3 multiplexores, uno donde se almacena el resultado final de la operación, un segundo multiplexor para controlar la ALUFlagZ y el tercero con la señal de control ALUFlagC, el significado de estas banderas fue especificado en el enunciado del ejercicio. Estos muxes fueron diseñados como:

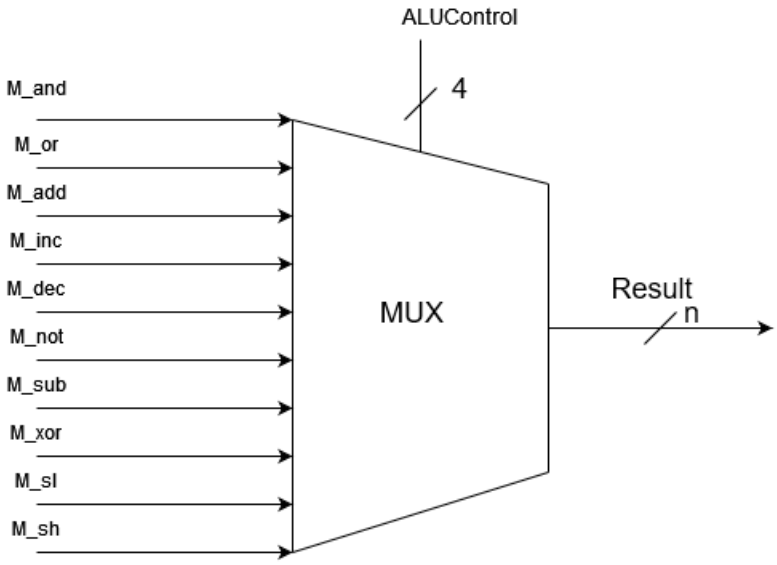

Figura 6.2: Diagrama de módulos para operación de ALUResult.

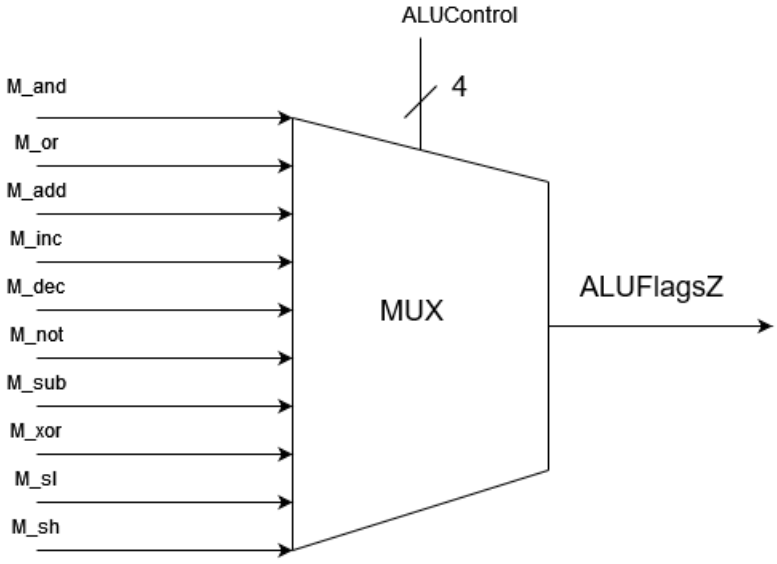

Figura 6.3: Diagrama de módulos para operación de ALUFlagsZ.

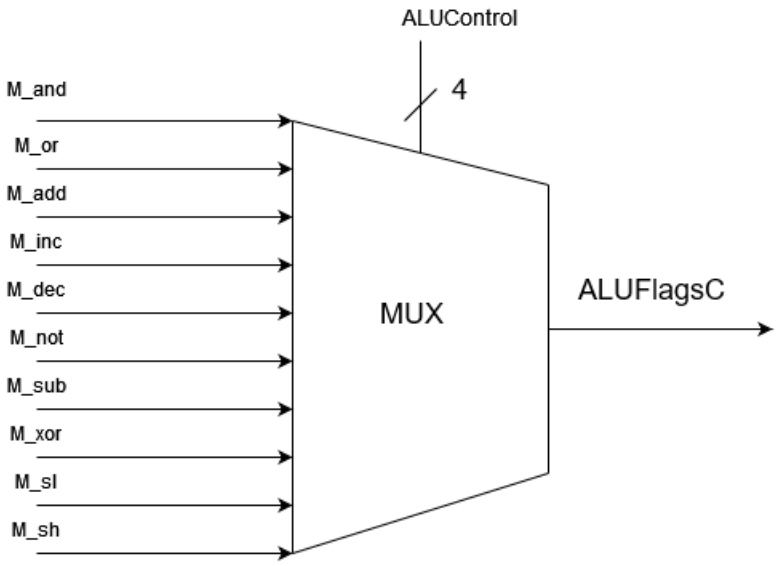

Figura 6.4: Diagrama de módulos para operación de ALU.

REFERENCIAS

[1] H. Fooladvand, K. Abbasian and H. Baghban, *High-Performance 4 × 1 Multiplexer based on Single-Walled Carbon Nanotube Field Effect Transistor with CMOS-like Pass-Transistor Logic*, Revista Ingeniería, 2020.

[2] Código Gray [Online]. Available: `https://es.wikipedia.org/wiki/C%C3%B3digo_Gray#Base_2_a_Gray`. [Accesed: 24-Feb-2024].

[3] S. Harris and D. Harris, *Digital design and computer architecture: RISC-V Edition.* Oxford, England: Morgan Kaufmann, 2021.