Diseños para resolver cada problema

Emanuel Chavarría Hernández
Fernando Andrés Fuchs Mora

CE 3201 - Taller de diseño digital
Instituto Tecnológico de Costa Rica
16 de agosto de 2025

1. Diseños

1.1. Pregunta 1

El sistema diseñado en esta sección tiene como objetivo recibir una entrada de 4 bits en binario, realizar una conversión a su correspondiente representación a código Gray y representar la salida mediante un display de 7 segmentos en formato hexadecimal. Como el diseño del sistema no necesita el uso de memoria se utilizó el modelado de comportamiento en SystemVerilog.

Fundamentación teórica

Para poder realizar el diseño del sistema es fundamental conocer los conceptos básicos que son involucrados. Se tiene que el código Gray es: ün sistema de numeración binario en el que dos números consecutivos difieren solamente en uno de sus dígitos."[1]. Este código fue diseñado para prevenir señales falsas o viciadas de los relés que se empleaban en las primeras computadoras. Sin embargo en la actualidad se utiliza para facilitar la corrección de errores en los sistemas de comunicaciones. [1] Para realizar una conversión de binario a Gray se tiene el siguiente procedimiento:

1. **Preservación del bit más significativo (MSB):** el primer bit del código Gray es igual al primer bit del número binario original.
2. **Generación de los bits restantes:** cada bit del código Gray se obtiene aplicando la operación lógica *XOR* entre el bit actual y el bit anterior del número binario. Matemáticamente:
$$G_i = B_i \oplus B_{i+1}$$
donde G_i representa el bit i del código Gray y B_i el bit i del código binario.
3. **Iteración:** se repite el proceso hasta completar la conversión de todos los bits.

Descripción del diseño

El diseño se compone de tres módulos principales:

1. **Módulo btg:** este módulo es el encargado de recibir la entrada de un número binario y convertirlo a su equivalente en código Gray mediante la operación lógica *XOR*.

2. **Módulo sevenseg:** este módulo es el encargado de recibir la entrada de un número binario con su respectiva conversión a código Gray y traducirlo a un valor hexadecimal al patrón de activación del 7 segmentos.

3. **Módulo top:** este módulo es el encargado de juntar los módulos descritos anteriormente para hacerlos funcionar como un todo. Recibe la entrada del número binario y su salida es la respectiva representación en hexadecimal por medio del 7 segmentos.

Para la creación del módulo btg se evaluaron dos alternativas que se detallan a continuación:

1. **Implementación mediante Ciclo for:** la primera alternativa consistió en analizar el algoritmo de conversión de un número binario a código Gray y se logró implementar un ciclo for el cual recorría todo el número binario y realizaba las operaciones correspondientes. Se realizaron las pruebas necesarias para comprobar el correcto funcionamiento de forma individual y de forma total. Se obtuvieron los resultados esperados de todas las pruebas realizadas sin embargo, esta no es una buena práctica para el tipo de diseño que se quiere implementar por lo tanto la propuesta fue rechazada.

2. **Uso de lógica combinacional:** la segunda alternativa consistió en el uso de la lógica combinacional mediante el bloque `always_comb` y realizar directamente las operaciones *XOR* asignándolas directamente al bus de salida. Se realizaron las pruebas necesarias para comprobar el correcto funcionamiento de forma individual y de forma total. Si bien es cierto esta propuesta hace lo mismo que la primera a nivel de resultado al usar la lógica combinacional se logran buenas prácticas de desarrollo para el resto del curso.

Para poder corroborar los resultados de ambas alternativas se implementó una tabla de verdad para la conversión del número binario a Gray para predecir la salida esperada. Se tiene la siguiente tabla:

Tabla 1. Tabla de verdad de código Binario a código Gray (4 bits).

Binario				**Gray**			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_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

1.2. Pregunta 2

Para implementar el restador de 4 bits se considero:

- ✓ **Propuesta A:** en base a las tablas de verdad se llego a una expresión por medio de min-términos (poco eficiente y mas uso de transistores).
- ✓ **Propuesta B (elegida):** Se busco simplificar la expresión obtenida para obtener una implementación mas compacta (utilizando compuertas XOR)

¿Por qué A? Es la mejor forma de implementarlo ya que es la mas eficiente en términos de cantidad de hardware y la verificación es más sencilla. **¿Por qué no B?** Funciona, pero usa más lógica de la necesaria.

Tabla de verdad unificada y min-términos. A partir del *restador completo* de 1 bit con entradas A, B, Bin y salidas $D, Bout$, se usa la tabla para obtener las expresiones en min-términos

A	B	Bin	D	$Bout$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Para D (filas con $D = 1$):

$$D \;=\; \overline{A\,B}\,Bin \;+\; \overline{A}\,B\,\overline{Bin} \;+\; A\,\overline{B\,Bin} \;+\; A\,B\,Bin$$

Para $Bout$ (filas con $Bout = 1$):

$$Bout \;=\; \overline{A\,B}\,Bin \;+\; \overline{A}\,B\,\overline{Bin} \;+\; \overline{A}\,B\,Bin \;+\; A\,B\,Bin$$

Después de simplificar las expresiones se obtiene:

$$Bout \;=\; \overline{A}(B + Bin) \;+\; (B \cdot Bin)$$

Partiendo de los min-términos de D:

$$D \;=\; \overline{A\,B}\,Bin \;+\; \overline{A}\,B\,\overline{Bin} \;+\; A\,\overline{B\,Bin} \;+\; A\,B\,Bin$$

donde se resalta que esta es exactamente la expansión canónica del **XOR**:

$$D \;=\; A \oplus B \oplus Bin$$

1.3. Pregunta 3

- ✓ **Propuesta A (elegida):** contador *síncrono* binario, parametrizable en N, con **reset asíncrono**. El incremento se hace con un bit que usa un "carry"/enable c_i entre bits.
- ✓ **Propuesta B:** arreglo de **T flip-flops** equivalentes (materia que no hemos visto ni en taller ni en Fundamentos)

¿Por qué A? Es directa para documentar, clara para verificar y se extiende a N bits con un patrón simple de c_i. **¿Por qué no solo B?** Es materia que como grupo de trabajo no estamos para nada familiarizados y se sale de nuestro alcance actual de conocimientos.

Tabla de verdad Se considera una celda con estado Q y un enable/carry C que intenta sumar 1 en ese bit. La tabla unificada (entradas $\rightarrow$ salidas) es:

Q	C	Q^+	C^+
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Min-términos.

$$Q^+ = \overline{Q}C \ + \ Q\overline{C} \qquad\qquad C^+ = QC$$

Forma simplificada (la que usamos).

$$Q^+ = Q \oplus C \qquad\qquad C^+ = Q \cdot C$$

Generalizando a N bits. Definimos $c_0 = en$ (un pulso por cada "paso"). Para $i = 0, \ldots, N-1$:

$$Q_i^+ = Q_i \oplus c_i, \qquad c_{i+1} = Q_i \cdot c_i.$$

Esto implementa el incremento $+1$ de forma síncrona, bit a bit.

Referencias

[1] Ángel Micelti, *Código Gray*, Accedido: 2025-08-15, 2023. dirección: https://angelmicelti.github.io/4ESO/EDI/cdigo_gray.html.