TEC | Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

CE 3201 — Taller de Diseño Digital

Documento de diseño

Autores:

Fabricio González Cerdas

Jian Zheng Wu

Profesor:

Luis Barboza Artavia

1. Propuestas de diseño — Problema 1

En esta sección se presentan dos métodos para implementar la conversión de un número binario de 4 bits a su equivalente en código Gray. Ambas propuestas cumplen la misma función, pero difieren en la forma de expresar las ecuaciones lógicas y en la cantidad de compuertas utilizadas.

1.1. Método XOR

Este método utiliza compuertas XOR para definir cada bit de la salida. A partir de la definición estándar de conversión Binario → Gray, las ecuaciones son:

$$G_3 = B_3,$$
$$G_2 = B_3 \oplus B_2,$$
$$G_1 = B_2 \oplus B_1,$$
$$G_0 = B_1 \oplus B_0.$$

A continuación se presentan las tablas de verdad individuales para cada ecuación.

$G_3 = B_3$

B_3	G_3
0	0
1	1

$G_2 = B_3 \oplus B_2$

B_3	B_2	G_2
0	0	0
0	1	1
1	0	1
1	1	0

$G_1 = B_2 \oplus B_1$

B_2	B_1	G_1
0	0	0
0	1	1
1	0	1
1	1	0

$G_0 = B_1 \oplus B_0$

B_1	B_0	G_0
0	0	0
0	1	1
1	0	1
1	1	0

1.2. Método SOP

En este método cada XOR se reemplaza por su forma equivalente de Suma de Productos (SOP), utilizando únicamente AND, OR y NOT:

$$A \oplus B = A\overline{B} + \overline{A}B$$

Aplicando a cada bit:

$$G_3 = B_3, \qquad G_2 = B_3\overline{B_2} + \overline{B_3}B_2, \qquad G_1 = B_2\overline{B_1} + \overline{B_2}B_1, \qquad G_0 = B_1\overline{B_0} + \overline{B_1}B_0.$$

$G_3 = B_3$

B_3	G_3
0	0
1	1

$G_2 = B_3\overline{B_2} + \overline{B_3}B_2$

B_3	B_2	$\overline{B_3}$	$\overline{B_2}$	$B_3\overline{B_2}$	$\overline{B_3}B_2$	$\mathbf{G_2}$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

$G_1 = B_2\overline{B_1} + \overline{B_2}B_1$

B_2	B_1	$\overline{B_2}$	$\overline{B_1}$	$B_2\overline{B_1}$	$\overline{B_2}B_1$	G_1
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

$G_0 = B_1\overline{B_0} + \overline{B_1}B_0$

B_1	B_0	$\overline{B_1}$	$\overline{B_0}$	$B_1\overline{B_0}$	$\overline{B_1}B_0$	G_0
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

1.3. Selección de método

Ambos métodos generan la misma salida para todas las combinaciones de entrada. Sin embargo, para la implementación final se utilizará el método XOR, ya que permite una descripción más compacta y directa en el código y facilita su escalabilidad a un mayor número de bits. Además, se observa que en el método SOP las tablas de verdad incluyen más columnas intermedias y pasos lógicos, lo que evidencia un diseño más extenso y menos eficiente en comparación con la simplicidad del método XOR.

2. Propuestas de diseño — Problema 2

En este problema se tiene como objetivo el diseño e implementación de un restador completo de 4 bits, empleando el modelo estructural en VHDL. El desarrollo parte de la construcción y validación previa de un restador completo de 1 bit, el cual servirá como módulo base para la composición del sistema de mayor complejidad.

2.1. Método 1

En esta primera propuesta, se parte de la tabla de verdad de un restador completo de 1 bit la cual es:

A	B	C_{in}	Y	C_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

De la cual se infieren las ecuaciones para las dos salidas que brinda la tabla (Y y C_{out}):

$$Y = (A \oplus B) \oplus C_{in} \tag{1}$$

$$C_{out} = \overline{A}B + \overline{B}C_{in} + \overline{A}C_{in} \tag{2}$$

Para finalmente crear el circuito esquemático:

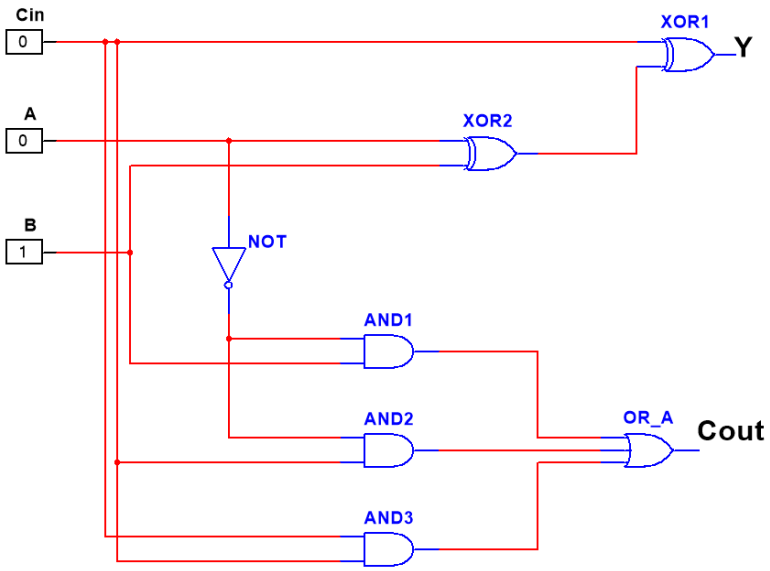

Figura 1: Circuito esquemático realizado en multisim

Donde la idea seria crear 4 "*instancias*" del restador completo de 1 bit, y que cada una manipule 2 bits (1 bit de A y otro de B), y tomando en cuenta los acarreos.

2.2. Método 2

Para la segunda propuesta, se pensó en implementar a partir de dos "$halfsubtractors$" y una compuerta OR adicional donde las salidas están dadas por:

$$d = A \oplus B \tag{3}$$

$$b = \overline{A}B \tag{4}$$

Creando la tabla de verdad:

A	B	$\overline{A}$	d	b
0	0	1	0	0
0	1	1	1	1
1	0	0	1	0
1	1	0	0	0

Finalmente el esquemático uniendo los dos "$half subtractors$" tendría una forma tal que así:

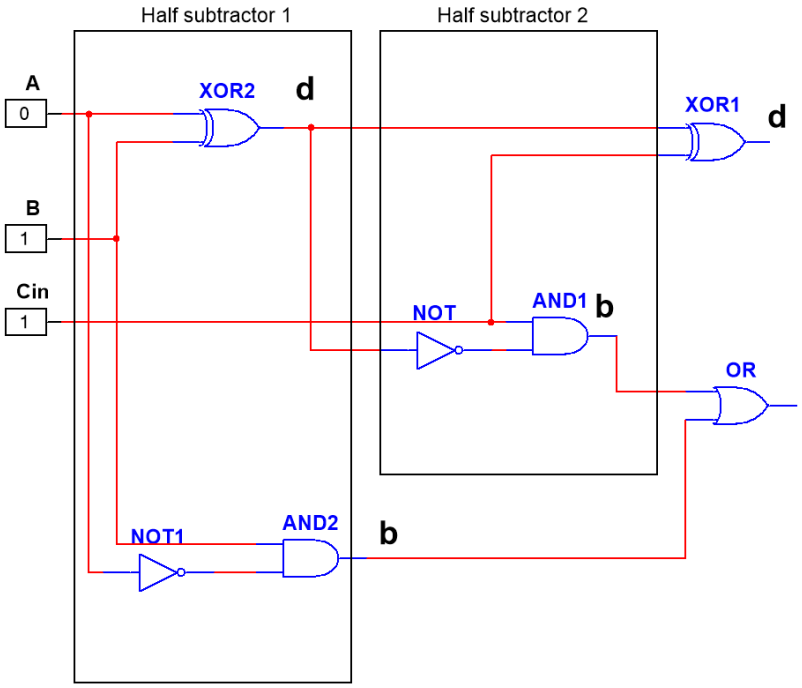

Figura 2: Circuito esquemático realizado en multisim

Donde observamos que los "$half subtractors$" se conectan en cascada, para así formar un restador completo de 1 bit, para al implementar el restador de 4 bits en código se repita lo que se mencionó anteriormente con la propuesta 1, de crear 4 "$instancias$"

2.3. Selección de método

A pesar de que ambas opciones son atractivas, como grupo se llegó a la conclusión que la opción 1 del restador completo de 1 bit resulta más conveniente que la opción 2 porque presenta un diseño directo a partir de compuertas lógicas, sin recurrir a la construcción modular mediante "$half subtractors$". Al trabajar de forma directa, el circuito evita conexiones adicionales y pasos intermedios, lo que simplifica la estructura general. Esto se traduce en un menor retardo de propagación, ya que las señales no necesitan atravesar dos etapas como ocurre en la 2, donde el resultado depende de la salida de un "$half subtractor$" antes de pasar al siguiente, optimizando así el uso de recursos como el consumo de electricidad en la implementación en la FPGA.

3. Propuestas de diseño — Problema 3

En este problema se requiere implementar un contador digital configurable que permita establecer y modificar sus valores de decenas y unidades de forma controlada. Para su desarrollo, se presentan dos enfoques distintos. El primero, denominado **Método Flip-Flops**, utiliza lógica secuencial con flip-flops tipo D para almacenar y actualizar el estado del contador. El segundo, llamado **Método BCD en cascada**, divide la lógica en dos módulos independientes para las decenas y las unidades, comunicados mediante señales de acarreo.

3.1. Método Flip-Flops

Este método se basa en el uso de flip-flops tipo D que almacenan el estado del contador y lo actualizan únicamente en el flanco positivo de la señal de reloj (`clk`). El contador es parametrizable en el número de bits N, permitiendo su escalabilidad para diferentes rangos de conteo. Además, incorpora un `reset` asíncrono para reiniciar su valor y una señal de incremento (`INC`) que determina cuándo debe avanzar.

Para ejemplificar el funcionamiento, se presenta el caso de un contador de 2 bits (Q_1, Q_0), cuyas ecuaciones son:

$Q_0^+ = Q_0 \oplus INC$

Q_0	INC	Q_0^+
0	0	0
0	1	1
1	0	1
1	1	0

$Q_1^+ = Q_1 \oplus (Q_0 \cdot INC)$

Q_1	Q_0	INC	$Q_0 \cdot INC$	Q_1^+
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Tabla de verdad combinada

La combinación de ambas ecuaciones produce la siguiente tabla de verdad del contador de 2 bits:

Q_1	Q_0	INC	$Q_0 \cdot INC$	Q_1^+	Q_0^+
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	0	0

Ecuación general para N bits

El comportamiento del método se generaliza con la ecuación:

$$Q_i^+ = Q_i \oplus \left(INC \cdot \prod_{j=0}^{i-1} Q_j \right)$$

donde cada bit Q_i cambia de estado únicamente si todos los bits menos significativos están en 1 y se recibe un pulso de incremento.

3.2. Método BCD en cascada

Este método propone implementar el contador mediante una estructura de conteo BCD (Binary Coded Decimal) en cascada, separando las **decenas** y las **unidades** en dos módulos independientes, cada uno con su propia lógica de incremento y reinicio.

En este enfoque, cada módulo recibe como entrada una señal de incremento (`INC`) y produce como salida su nuevo valor y una señal de acarreo (`carry`) que indica que se ha alcanzado el valor máximo permitido en ese dígito.

Funcionamiento básico:

- El módulo de unidades incrementa su valor en cada pulso de `INC`.
- Cuando las unidades alcanzan el valor 9 y reciben un nuevo pulso, se reinician a 0 y generan un `carry` que se envía al módulo de decenas.
- El módulo de decenas funciona de forma similar, pero su valor máximo es 6, lo que permite contar hasta 63 en total.

Ecuaciones lógicas: Si U representa el valor en unidades y D el valor en decenas:

$$U^+ = \begin{cases} 0, & \text{si } U = 9 \text{ y } INC = 1, \\ U + 1, & \text{si } INC = 1, \\ U, & \text{si } INC = 0. \end{cases}$$

$$D^+ = \begin{cases} 0, & \text{si } D = 6 \text{ y } carry_U = 1, \\ D + 1, & \text{si } carry_U = 1, \\ D, & \text{si } carry_U = 0. \end{cases}$$

Donde:

$$carry_U = (U = 9) \cdot INC$$

$$carry_D = (D = 6) \cdot carry_U$$

Tabla de verdad simplificada: A modo de ejemplo, para las unidades, se tiene:

U	INC	U^+
0–8	0	U
0–8	1	$U+1$
9	0	9
9	1	0

Para las decenas, considerando el acarreo:

D	$carry_U$	D^+
0–5	0	D
0–5	1	$D+1$
6	0	6
6	1	0

3.3. Selección de método

Tras evaluar las opciones de implementación, se optó por el método basado en flip-flops tipo D debido a su simplicidad lógica, menor cantidad de componentes adicionales y facilidad de escalabilidad para diferentes tamaños de contador. Este enfoque permite actualizar los bits únicamente en el flanco positivo de la señal de reloj y genera el avance de cada bit de forma implícita mediante la condición de que todos los bits menos significativos estén en 1, lo que actúa como un acarreo interno. Esta característica elimina la necesidad de manejar señales de acarreo explícitas como en el método BCD en cascada, simplificando el diseño y reduciendo la complejidad de interconexiones.