Laboratorio 1: Taller de Diseño Digital

Gabriel Fernández Vargas y Emanuel Rojas Fernández

August 16, 2025

Problema 1: Conversión Binario a Gray con Visualización Hexadecimal

Diseñar un módulo que convierta un número binario de 4 bits en código Gray y muestre su equivalente hexadecimal en un display de 7 segmentos. Se usarán como entradas los switches SW0–SW3 y como salidas los segmentos a–g del display HEX0.

Propuesta 1: Decodificador Directo Combinacional (Solución Implementada)

Descripción

Esta propuesta utiliza un enfoque puramente combinacional:

1. Capturar los 4 bits de entrada desde los switches y formar un vector binario.
2. Convertir el número binario a código Gray usando operaciones XOR:

$$\begin{aligned} \text{gray}[3] &= \text{bin}[3] \\ \text{gray}[2] &= \text{bin}[3] \oplus \text{bin}[2] \\ \text{gray}[1] &= \text{bin}[2] \oplus \text{bin}[1] \\ \text{gray}[0] &= \text{bin}[1] \oplus \text{bin}[0] \end{aligned}$$

3. Decodificar el valor Gray a 7 segmentos mediante expresiones lógicas combinacionales, sin usar `case`, para mostrar su equivalente hexadecimal.

Tabla de verdad de la conversión binario a Gray

Ventajas y Desventajas

Ventajas:

- Muy eficiente en hardware, ya que es puramente combinacional.

SW3	SW2	SW1	SW0	Gray3	Gray2	Gray1	Gray0
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

Table 1: Tabla de verdad para la conversión de binario a Gray (4 bits).

- No requiere memoria ni secuencias de control.
- Fácil de probar con simulación o en FPGA real.

Desventajas:

- Las expresiones lógicas para el decodificador de 7 segmentos son largas y propensas a errores si se hacen manualmente.

Propuesta 2: Conversión mediante ROM/Tabla de Look-Up

Descripción

En esta propuesta se usa una memoria ROM de 16 palabras, donde cada palabra representa los 4 bits Gray y la activación de los segmentos a–g:

1. La combinación de switches SW3–SW0 se usa como dirección de la ROM.
2. Cada dirección contiene la palabra que activa los segmentos del display.
3. La ROM puede implementarse con un arreglo de registros:

   reg [6:0] rom [0:15];

4. La salida de la ROM se conecta directamente a los pines a–g del display.

SW3	SW2	SW1	SW0	Gray	Segmentos a-g
0	0	0	0	0000	7'b1111110
0	0	0	1	0001	7'b0110000
0	0	1	0	0011	7'b1101101
0	0	1	1	0010	7'b1111001
0	1	0	0	0110	7'b0110011
0	1	0	1	0111	7'b1011011
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1000	7'b1111111

Table 2: Tabla de look-up para implementación mediante ROM.

Tabla de look-up (ejemplo)

Ventajas y Desventajas

Ventajas:

- Fácil de implementar y mantener.
- Reduce el riesgo de errores en expresiones lógicas largas.
- Permite cambios rápidos modificando la tabla de la ROM.

Desventajas:

- Ocupa más recursos en la FPGA.
- No es tan eficiente en términos de velocidad como un decodificador combinacional.

Conclusión del Problema 1

Se presentan dos alternativas para la conversión de binario a Gray y su visualización en hexadecimal:

1. **Decodificador combinacional directo:** solución implementada y eficiente, adecuada para FPGAs con pocos recursos.
2. **Tabla ROM/Look-up:** alternativa más flexible y fácil de mantener, aunque con mayor uso de recursos.

Problema 2: Restador Completo de 4 Bits

Enunciado

1. Diseñar un restador completo de 4 bits con modelo de estructura en VHDL. Partir del diseño de un restador completo de 1 bit.

2. Realizar un *testbench* para el restador del punto anterior, en VHDL. Mostrar las pruebas para al menos 4 valores diferentes de operandos.

3. Implementar el restador completo en FPGA. Utilice los switches de la tarjeta para los datos de entrada, así como los displays de 7 segmentos para mostrar el resultado correcto (en hexadecimal).

Ejemplo de Restador de 1 Bit

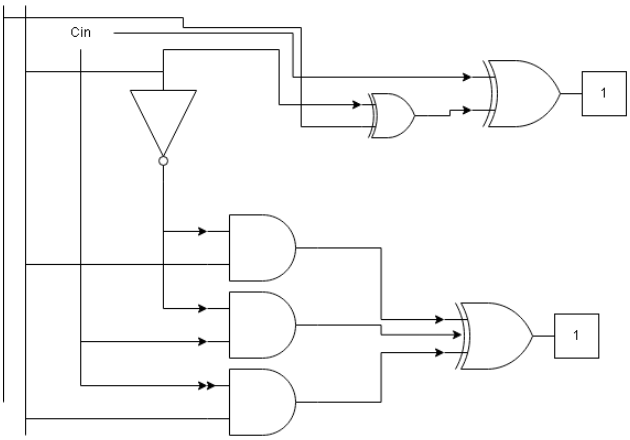

Figure 1: Diagrama lógico de un restador completo de 1 bit.

Propuesta 1: Restador con Borrow Encadenado

- Se utiliza el restador completo de 1 bit como bloque básico.
- Se encadenan 4 bloques para construir el restador de 4 bits.
- La salida de borrow de cada bloque se conecta a la entrada de borrow del siguiente.

Ventajas:

- Diseño modular y escalable.
- Fácil de implementar en FPGA.

Desventajas:

- Retardo acumulativo por propagación de borrow.

Propuesta 2: Restador mediante Complemento a 2

- Convierte el operando B a su complemento a 2.
- Suma A + (complemento a 2 de B).
- El resultado de la suma corresponde a A - B.

Ventajas:

- Reutiliza bloques de sumadores ya disponibles en FPGA.
- Mayor velocidad comparado con borrow en cascada.

Desventajas:

- Requiere hardware adicional para el complemento a 2.

Conclusión del Problema 2

Ambos enfoques son viables:

- El diseño con borrow encadenado es más simple y didáctico.
- El diseño con complemento a 2 es más rápido y eficiente en hardware moderno.

Problema 3: Contador Parametrizable

Enunciado

1. Diseñar un contador parametrizable de N bits con reset asincrónico.
2. Realizar un *testbench* de auto-chequeo para el contador. Mostrar el resultado para 2, 4 y 6 bits.
3. Implementar el contador de 6 bits en FPGA, con botón para aumentar el contador, un switch para reset, y visualización en display de 7 segmentos (decimal o hexadecimal).

Propuesta 1: Contador Ascendente Sencillo

- Incrementa en 1 cada vez que se activa el pulso de reloj.
- Tiene entrada de reset asincrónico para reiniciar el valor.

Propuesta 2: Contador Parametrizable con Valor Inicial

- Además del incremento, permite cargar un valor inicial mediante los switches.
- Esto lo hace más flexible para pruebas y aplicaciones reales.

Conclusión del Problema 3

- El contador ascendente sencillo es ideal para aplicaciones básicas.
- El contador parametrizable con valor inicial es más útil en escenarios prácticos.

References

[1] David M. Harris y Sarah L. Harris, *Digital Design and Computer Architecture*, 2da edición, Morgan Kaufmann Publishers, 2013.