Universidad de Costa Rica | Facultad de Ingeniería | Escuela de Ingeniería Eléctrica

Máquinas de estado finitas: módulos generate, for loops, lógica secuencial

Roberto Sánchez Cárdenas - B77059

San José, 17 de octubre de 2020
Circuitos Digitales II

Índice

1. Introducción

Esta tarea tiene como fin aprender un poco sobre el procedimiento que se realiza para la verificación de módulos diseñados en Verilog. Para ello se brinda una descripción estructural de un contador Gray, y se dice de antemano que hay un error. El error debe ser buscado y explicado.

Para ello se corren múltiples pruebas que permiten validar el funcionamiento.

2. Contador Gray

Un contador Gray es un circuito que permite contar de manera ascendente o descendente cambiando única mete un bit respecto a la salida previa. Este tipo de contadores son muy utilizados en sistemas híbridos analógico-digital. Son especialmente útiles en el caso de sistemas digitales en el que se reciben pulsos a muy altas tasas, dado que únicamente debe variar un

dato, la probabilidad de error se disminuye significativamente, además de que permite guardar los datos en caso de que hallan interrupciones debido a la implementación con flip-flops. [1]

Otro uso común es para codificar los punteros hacia buffers de datos tipo FIFO (first in, first out), es decir, para manejar datos en colas. Se usa principalmente cuando los puertos de lectura y escritura funcionan a frecuencias de reloj distintas, esto con el fin de reducir la probabilidad de captar un dato en un estado transitorio. [2]

3. Descripción conductual

Para validar el comportamiento de la descripción estructural que fue proporcionada, se realizó la descripción conductual de un contador Gray. Para ello se utilizó un algoritmo basado en desplazamientos del código en codificación binaria normal.

Para hallar el código Gray, se toma el código binario y se desplaza una posición a la izquierda. El código desplazado deja un espacio en blanco que es llenado con un 0. Se muestra un ejemplo para el número 14 decimal a continuación.

$$\begin{array}{rcl} 01110 & = & 14'd \\ + \quad 00111 & & \\ \hline 01001 & & \end{array} \tag{1}$$

El módulo diseñado tiene un funcionamiento adecuado, se muestran los primeros 20 datos en la siguiente figura.

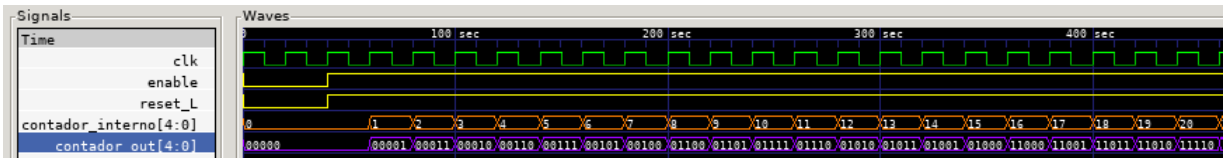

Figura 1: Contador Gray conductual

4. Verificación

Se diseñó un verificador automático que muestre en donde no concuerdan las señales. Para validar funcionamiento se hicieron múltiples pruebas variando las entradas de enable y reset por periodos largos y cortos. No se hallaron errores al variar estos datos.

Se procedió a correr la simulación por una cantidad de ciclos prolongada y en este caso sí se observan errores en ciertos punto. El módulo estructural posee un contador interno que cuenta cuantas veces ha contado hasta 31 decimal y se reinicia cuando llega a 3 decimal. En la siguiente figura se presenta el punto en donde hay un error.

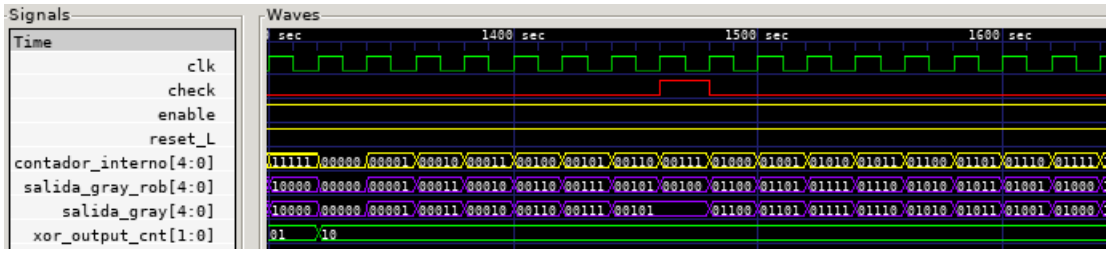

Figura 2: Punto de error en módulo estructural

El módulo estructural presenta una discrepancia cuando el contador xor_output_cnt = 01 y el dato que se desea convertir a binario es 00111. El mismo patrón se repite siempre en este punto. No se detectaron más errores.

4.1. Descripción estructural

Dado que no se tiene mucha información sobre la lógica detrás de esta descripción, se tuvo que investigar y seguir las señales por donde se van moviendo en el módulo. Un primer indicio que se tuvo fue la señal xor_output_cnt = 01, por lo que se comenzó a buscar por ahí. El código conflictivo es **00111**.

Inicialmente el módulo toma lo siguiente

$$003 = NOT(xor\_output\_cnt[0]) = 1 \tag{2}$$
$$004 = NOT(xor\_output\_cnt[1]) = 0 \tag{3}$$

Más adelante toma

$$010 = NAND(count[1], count[2]) = 0 \tag{4}$$
$$014 = NAND(count[0], count[1]) = 0 \tag{5}$$
$$038 = NOR(count[0], count[1]) = 0 \tag{6}$$
$$039 = NOR(count[3], count[4]) = 1 \tag{7}$$
$$040 = NAND(003, xor\_output\_cnt[1]) = 0 \tag{8}$$
$$041 = NOR(010, 040) = 1 \tag{9}$$
$$042 = NAND(039, 041) = 0 \tag{10}$$
$$043 = NOT(042) = 1 \tag{11}$$
$$044 = NOR(014, 043) = 0 \tag{12}$$
$$\tag{13}$$

Finalmente

$$salida\_gray[0] = NOR[038, 044] = 1$$

Esta salida es errónea dado que debería ser 0.

Si se observan con atención las compuertas y cables, se ve que en el cable 038 (ecuación 6) se tiene la respuesta correcta, sin embargo, este dato se compara con otros cable que traen combinaciones conflictivas. Por algún motivo, se realiza una NAND entre $count[1]$ y $count[2]$ y otra entre $count[3]$ y $count[4]$. Luego esto se compara con datos que dependen del estado de xor_output_cnt. Un ejemplo de caso donde no hay conflicto es con: $xor\_output\_cnt = 01$ y $counter = 01111$

Inicialmente el módulo toma lo siguiente

$$003 = NOT(xor\_output\_cnt[0]) = 1 \tag{14}$$
$$004 = NOT(xor\_output\_cnt[1]) = 0 \tag{15}$$

Más adelante toma

$$010 = NAND(count[1], count[2]) = 0 \tag{16}$$
$$014 = NAND(count[0], count[1]) = 0 \tag{17}$$
$$038 = NOR(count[0], count[1]) = 0 \tag{18}$$
$$039 = NOR(count[3], count[4]) = 0 \tag{19}$$
$$040 = NAND(003, xor\_output\_cnt[1]) = 0 \tag{20}$$
$$041 = NOR(010, 040) = 1 \tag{21}$$
$$042 = NAND(039, 041) = 1 \tag{22}$$
$$043 = NOT(042) = 0 \tag{23}$$
$$044 = NOR(014, 043) = 1 \tag{24}$$
$$\tag{25}$$

Finalmente

$$salida\_gray[0] = NOR[038, 044] = 0$$

En el caso anterior, con solo cambiar el bit en count[3] y dejar lo demás igual, el contador funciona de manera correcta. Estas comparaciones que se realizan están de más y se podrían arreglar si se eliminaran.

5. Recomendaciones

- Para solucionar el único error presente en el contador, se podría eliminar la comparación que se realiza respecto al contador de xor. Si se considera necesario, buscar una alternativa que no haga conflicto.
- Cuando no se ven indicios de error en pocos ciclos de reloj, es conveniente dejarlo por muchos ciclos, puesto que no sabemos si más adelante algo genera errores.
- Si no se logra observar el error en las señales, será necesario ir a buscar el comportamiento dentro de la estructura. Para ello se puede ver donde se carga la salida conflictiva e ir hacia atrás en los cables.
- Un verificador automático simplifica el hallar errores, por lo que se recomienda siempre implementar uno desde el inicio.

6. Tiempo de desarrollo

- Código: 2 horas
- Busqueda de error: 1 horas
- makefile: 5 minutos
- Obtención de resultados y análisis: 2 horas
- Reporte: 1 hora

7. Correr simulación

Para correr la simulación basta con usar el comando make en la dirección en donde se tiene los archivos.

Referencias

[1] M. Cohn and S. Even. A gray code counter. *IEEE Transactions on Computers*, C-18(7):662–664, 1969.

[2] Shweta Gupta and Meenakshi Munjal. Advanced level cyclic gray codes with application meenakshi. 4, 04 2014.