Universidad de Costa Rica

IE-523 - Circuitos Digitales II

Profesor: Jorge Soto

Tarea 8: Verificación de un contador gray de 5 bits

Estudiante:
Moisés Campos Zepeda

Carné:
B31400

I semestre - 2019

1. Contabilización del tiempo tomado en la tarea

- Buscar información del contador gray: se tardó un tiempo de 20 mín en investigar sobre el contador y buscar.
- Para el diseño del testbench 50 mín .
- En el diseño de las pruebas para el probador con un verificador automático (*checker.v*). temporal tardó 130 mín.
- En el diseño de la descripción conductual del contador gray se invirtió aproximadamente 150 mín.
- En la modificación del *makefile*, el *config.gtkw* y el *README.md* del git se tardó un tiempo de 70 mín.
- En la confección del presente reporte, se tomo un tiempo de 160 mín.
- En total se tardó 610 mín.

2. Contador gray

El contador gray implementa el código gray, el cual consiste en la ordenación del sistema de números binarios de modo que dos valores sucesivos difieran en un solo bit (dígito binario). El código binario reflejado fue diseñado originalmente para evitar salidas espurias de interruptores electromecánicos. Black (2019).

Los códigos gray se usan ampliamente para:

- los encoders mecánicos, ya que un ligero cambio de posición solo afecta a un bit. Usando un código binario típico, hasta n bits podrían cambiar, y las desalineaciones leves entre los elementos de lectura podrían causar lecturas extremadamente incorrectas.
- facilitar la corrección de errores en las comunicaciones digitales, como la televisión digital terrestre y algunos sistemas de televisión por cable.

Donde **s** la señal que selecciona la línea de salida.

3. Diseño del testbench

Para verificar la funcionalidad se utilizó el mismo banco de pruebas (*testbench.v*), del Proyecto 1. Por lo anterior, en este se trabajo en la reescritura de las variables generadas por AUTOINST, lo cual contempló la mayoría del tiempo de diseño. Este archivo además se encarga de llamar el modelo conductual y compararlo con el estructural a través del probador.

4. Probador con verificador automático

Este se encarga de probar las entradas: primero mandando un **reset_L** bajo. Luego activando esta señal y el **enable**, dejando correr los contadores por 3 ciclos de incremento, lo cual corresponde a 96 ciclos de la señal **clk**. Lo anterior considerando que cada 32 incrementos se reinician los registros internos **count** en el sintetizado y **counter** en el modelo conductual.

Listing 1: Pruebas realziadas en el módulo probador.v

```
initial begin
    //(...)
       // Prueba #2: Reset alto. Valido primer dato
        enable <= 1;
        reset_L <= 1;

        repeat(596) begin
        @(posedge clk);
        //count <= count + 1;
        end

        repeat(4) begin
        @(posedge clk);
        //count <= count + 1;
        end

        repeat(2) begin
        @(posedge clk);
        enable <= 0;
        end

        repeat(2) begin
        @(posedge clk);
        reset_L <= 0;
        end

        $finish;
   end
```

5. Descripción conductual diseñada

Esta se basa en el Código Gray previamente estudiado. La conversión de binario a este se puede modelar con el siguiente algoritmo, donde ^ es XOR o una operación o-exclusiva, y >> es un desplazamiento a la derecha de bits sin signo :

Listing 2: Algoritmo de binario sin signo a Código Gray en verilog.

```
module contador_gray_cond(...); 0
    //(...)
    always@(*)begin
    salida_gray = numero_bin ^ (numero_bin >> 1);
    end
endmodule
```

6. Ejemplos de los resultados

Antes de sintetizar es importante recordar los resultados originales que se tenían en la descripción conductual:

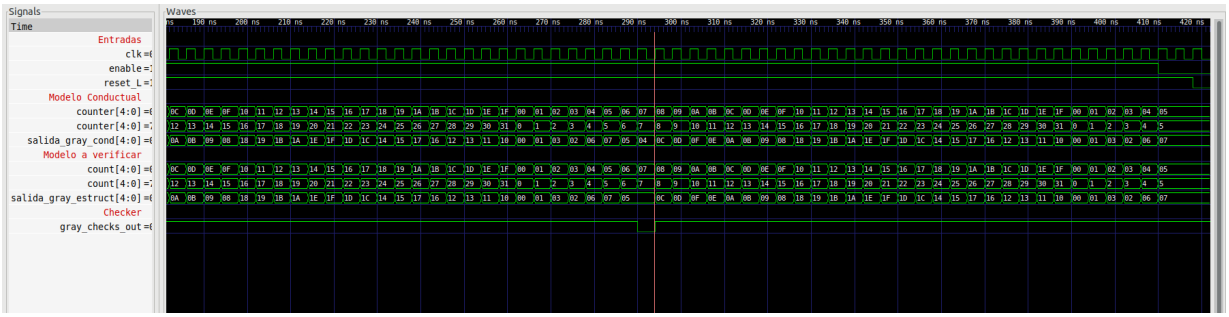

Figura 1: Primeros resultados obtenidos al correr $make all. Nótese la caída de la señal **gray_checks_out**, lo cual indica una diferencia entre el modelo conductual y el estructural sintetizado, posiblemente ahí radica el error.

Para los resultados se tomo el modelo conductual diseñado y se le comparo con el sintetizado, multiplexor de 4 bits sintetizado. En la gráfica mostrada en **GTKWave**, al acercarse la caída de la señal **gray_checks_out** se observa lo siguiente:

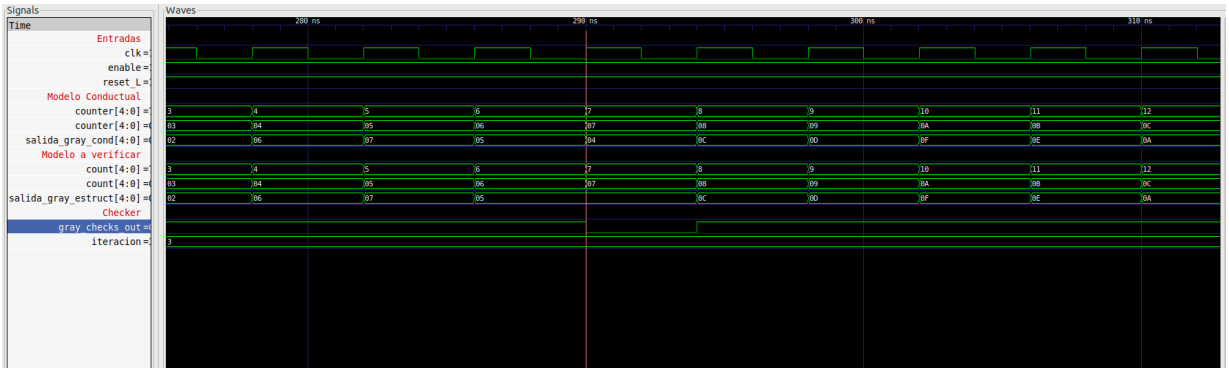

Figura 2: Acercamiento a la caída de la señal **gray_checks_out** a partir de los 290 *ns*.

Al aumentar las muestras de 96 en la Prueba 2 conforme a la figura 1 del archivo *probador.v* a 596 se observó que el error reiteraba, por lo cual se tomo la gráfica mostrada en la figura 3:

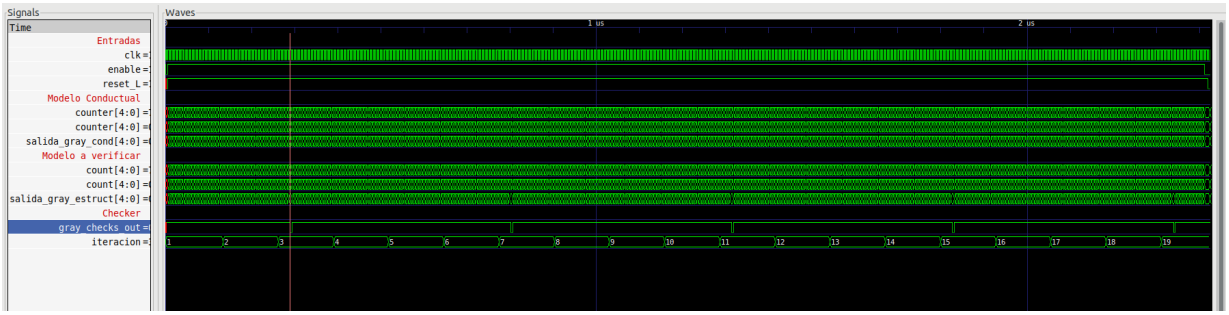

Figura 3: Prueba para revisar la reiteración del error de la caída de la señal **gray_checks_out**, con otra señal **iteración**.

7. Análisis y Conclusiones

De la figura 2 observa que el modelo sintetizado falla en el tercer ciclo del contador a los 290 ns, al quedarse en el valor equivalente a 6 (00101) cuando debería avanzar al valor 7 (00100). Nótese que en el siguiente flanco, la **salida_gray_estruct** recupera el siguiente valor de manera adecuada, por lo que observa que puede haber un error a la hora de llegar al valor de 6 en una tercera iteración del contador. Tal error puede resultar por alguna bandera (registro o señal) que se active adrede en tal iteración y número.

De la figura 3 se observa que el error aparece en las iteraciones 3, 7, 11, 15 y 18. De lo anterior se puede suponer que esa bandera puede tener un valor inicial el cual incrementa en cada iteración, y al pasar 3 iteraciones adicionales (2 en el caso inicial).

Referencias

Black, P. E. (2019). Gray code. *Dictionary of Algorithms and Data Structures*. Recuperado el 7 de junio de 2019 de https://xlinux.nist.gov/dads/HTML/graycode.html.

Anexos

El código del presente módulo, el módulo sintetizado, el probador, el checker, el makefile, el banco de pruebas, y el script en yosys que se ven a continuación se puede encontrar en: https://github.com/moisotico/Digitales-II/tree/master/Tarea8

Descripción conductual

Para poder verificar el desperfecto de la descricpión sintetizada, de diseña el siguiente módulo conductual del contador grey de 5 bits en Verilog.

Listing 3: Módulo contador_gray_cond.v

```
1  module contador_gray_cond(
2                  input                clk,
3                  input                enable,
4                  input                reset_L,
5                  output reg  [4:0]    salida_gray );
6
7      //Counter with an extra bit
8      reg [4:0] counter;
9
10     //SIZE: amount counted before it resets
11     parameter SIZE = 32;
12
13     always@(posedge clk)begin
14
15         //resets the main 5 {0 to 4} bits
16         if( !reset_L )begin
17                counter<= 'b0;
18         end else if (enable) begin
19            //toggle pendiente
20             if(counter < (SIZE -1) )begin
21                 counter <= counter + 1;
22             end else begin
23                 counter <= 'b0;
24             end
25         end
26     end
27
28     always@(*)begin
29         salida_gray = counter ^ (counter >> 1);
30     end
31 endmodule
```