Proyecto 1: Juego de luces

Mario López Sáez 2 de marzo de 2025

ÍNDICE

Índice

1	Introducción 3		
2	Contador parametrizable 2.1 Parámetros	3 3 3 3	
3	Pruebas: Contador parametrizable 3.1 Parámetros	4 4 5 5	
4	Registro de desplazamiento parametrizable 4.1 Parámetros	7 7 7 7 7	
5	Pruebas: Registro parametrizable 5.1 Parámetros	7 7 7 7	
6	Juego de luces 6.1 Parámetros 6.2 Entradas 6.3 Salidas 6.4 Instancias 6.4.1 Contador 6.4.2 Registro 6.5 Notas sobre el código	8 8 8 8 8 8 8	
7	Pruebas: Juego de luces		

1 Introducción

El objetivo principal del proyecto es crear un juego de luces en el que la luz viaje del LEDG[0] al LEDG[7], y rebote de nuevo hacia el LEDG[1] (no cuento el 0 ya que es volver al primer estado ya definido).

Contamos con un reloj de 50 MHz, y será vital recordar el valor para poder controlar la velocidad del juego de luces.

2 Contador parametrizable

2.1 Parámetros

Parámetro: módulo (modulus). Número de cuentas (0 a modulus-1)

Parámetro local: n

Número de bits para contar hasta modulus-1.

n se puede calcular fácilmente mediante dos funciones

Sabemos que el mayor módulo posible con n bits es 2^n , por lo que $n = \log_2(\text{modulus})$, pero, n debe ser un natural, por lo que va a haber que aproximar nuestra función. Con un par de cálculos sencillos nos podemos dar cuenta de que hay que aproximar al alza.

Esto significa, que si n=3.14 habrá que guardar n=4. Esta función se llama ceil, y es el opuesto de floor, truncar (techo - suelo respectivamente en inglés)

Estos ejemplos en Python y en Verilog son lo mismo:

```
// Verilog
localparam n = $clog2(9);
```

2.2 Entradas

```
© № CLK — Señal de reloj a 50 MHz

© № UP — Entrada síncrona. A nivel alto el contador va hacia arriba y viceversa

© № ENABLE — Entrada síncrona a nivel alto

© № RSTn — Reset asíncrono a nivel bajo
```

2.3 Salidas

```
\bigcirc → \bigcirc COUNT [n-1:0] — Registro que guarda la cuenta \bigcirc → \bigcirc TC — Toggle counter - salida que se activa cuando la cuenta llega al final
```

2.4 Notas sobre el código

Always at ₩ O or ₩ RSTn

Bloque sencillo, con bloques condicionales.

Assign

Operador ternario:

```
assign OUTPUT = CONDICION ? VALOR_SI_TRUE : VALOR_SI_FALSE;
assign TC = ((UP && (COUNT == modulus-1)) || (~UP && ((COUNT == 0) && RSTn))) &&
(ENABLE) ? 1 : 0;
```

A la condición final le añadimos AND ENABLE y AND RSTn para que TC no se pueda quedar pillado. El efecto de AND RSTn es igualmente aplicable a la condición final, o como en mi código, a solo el caso en el que UP=0. Si UP=1, TC no se activaría nunca con RSTn =0, ya que con UP=1 miramos COUNT == modulus-1.

3 Pruebas: Contador parametrizable

Al estar utilizando el contador como divisor de frecuencia, comprobar que TC no se queda activo durante un largo periodo de tiempo es vital, ya que, si no, nuestro registro de desplazamiento funcionaría a la velocidad del reloj, y los LEDs irían muy rápido.

3.1 Parámetros

Escala de tiempo

1 ns - Unidad por defecto 100 ps - Precisión de la simulación

Parámetro local: T = 20 ns

Módulo: 20

3.2 Funcionamiento normal

Comprobaremos que el contador de manera natural funciona como debe.

Simulación

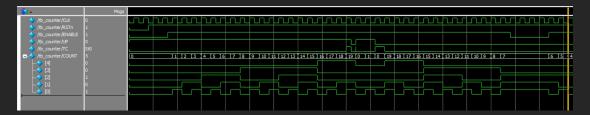


Figura 1: Simulación RTL. Funcionamiento normal del contador



Figura 2: Simulación Gate-Level.

3.3 Caso extremo 1: Con RSTn y UP = 0, ise queda TC pillado?

```
// RSTn con UP = 0 genera un TC prolongado?
UP = 0;
RSTn = 0;
#(T*2)
RSTn = 1;
#(T*4)
UP = 1;
#(T*2)
```

Simulación



Figura 3: Simulación RTL. Caso extremo 1.



Figura 4: Simulación Gate-Level. TC se activa un ciclo mínimo frente a la delta de RTL

3.4 Caso extremo 2: Apagamos ENABLE cuando TC = 1

```
// ENABLE = 0 cuando TC = 1 deja TC pillado en 1?
wait (duv.TC == 1);
ENABLE = 0;
#(T*4)
ENABLE = 1;
// Lo mismo pero con UP = 0
UP = 0;
#(T*8)
wait (duv.TC == 1);
ENABLE = 0;
#(T*4)
$display("Test finished");
$stop;
end
```

Simulación

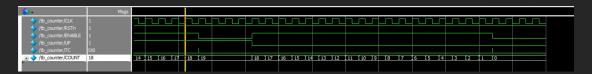


Figura 5: Simulación RTL. Caso extremo 2

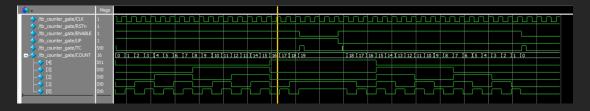


Figura 6: Simulación Gate-Level.

3.5 Vista general

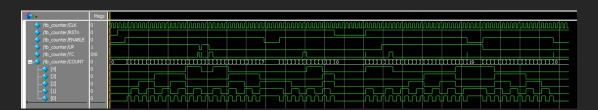


Figura 7: Simulación RTL. Vista general

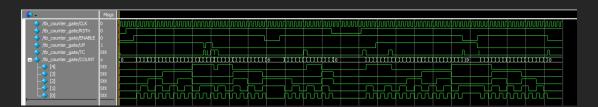


Figura 8: Simulación Gate-Level. Vista general

4 Registro de desplazamiento parametrizable

4.1 Parámetros

Parámetro: ancho

El tamaño de bits del registro.

4.2 Entradas

O ℷℷ CLK — Señal de reloj a 50 MHz

4.3 Salidas

 \bigcirc → \bigcirc OUT-P [ancho-1:0] — Salida en paralelo \bigcirc → \bigcirc SHIFT-OUT — Salida de serie

4.4 Notas sobre el código

Always at N O or N RSTn

Concatenación de señales. Ponemos a la izquierda la entrada SHIFT-IN y a la derecha OUT-P sin incluir OUT-P[0] para mantener el mismo tamaño.

```
OUT_P[ancho-1:0] <= {SHIFT_IN, OUT_P[ancho-1:1]};</pre>
```

Assign

Asignamos OUT-P[0] a SHIFT-OUT

5 Pruebas: Registro parametrizable

5.1 Parámetros

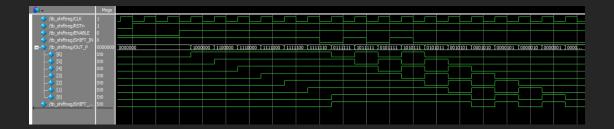
Escala de tiempo

1 ns - Unidad por defecto 100 ps - Precisión de la simulación

Parámetro local: T = 20 ns

Ancho: 7

5.2 Simulación RTL



5.3 Simulación Gate-Level

6 Juego de luces

6.1 Parámetros

Estado	Johnson	LEDG
1	0000000	00000001
2	1000000	00000010
3	1100000	00000100
4	1110000	00001000
5	1111000	00010000
6	1111100	00100000
7	1111110	01000000
8	1111111	10000000
9	0111111	01000000
10	0011111	00100000
11	0001111	00010000
12	0000111	00001000
13	0000011	00000100
14	0000001	00000010

Cuadro 1: $2 \cdot 7 = 14 \rightarrow 7 \ bits$

6.2 Entradas

O M CLK — Señal de reloj a 50 MHz

○ M ENABLE — Entrada síncrona a nivel alto

№ RSTn — Reset asíncrono a nivel bajo

6.3 Salidas

□ LEDG [7:0] — Señal de activación de los LEDs

6.4 Instancias

6.4.1 Contador

Módulo = $12.5\cdot 10^6$ (cuenta hasta $50\cdot 10^6$ en 1s, por lo que para obtener 0.25s, dividimos entre 4) UP = 1

6.4.2 Registro

ENABLE = TC

SHIFT-IN = !SHIFT-OUT

6.5 Notas sobre el código

Always at *

Asignamos un valor de led-pattern a todos los casos posibles de OUT-P

Assign

 ${\sf Directamente\ asignamos\ OUT-P=led-pattern}$

7 Pruebas: Juego de luces

En la prueba, he incluido un funcionamiento general, y el Caso extremo 2 del contador (apagar ENABLE cuando TC=1)

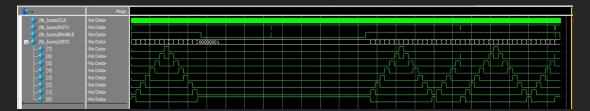


Figura 9: Simulación RTL.