IE0523 | Circuitos Digitales 2 Mario A. Castresana Avendaño

A41267

Tarea 2 | Contador binario sincrónico

1. Resumen

Para la presente tarea, se diseñaron dos módulos contadores con 4 modos de operación cada uno. Inicialmente, se parte del diseño de un contador simple de 4 bits capaz de operar bajo las siguientes modalidades:

1. Contador ascendente.
2. Contador descendente.
3. Contador descendente de 3 en 3.
4. Precargar un valor específico.

En la segunda parte de este proyecto se trabajó con un módulo contador de 32 bits hecho con 8 módulos individuales de 4 bits. En cada uno de los casos se implementó un solo módulo probador con todas las pruebas juntas para los dos contadores. Todo esto con ayuda del plan de pruebas propuesto en la tarea. Dentro de todas las pruebas lo más importante fue comprobar que el acarreo (Ripple-Carry Out) funcionaba correctamente en todos los contadores, para cerciorarse que cada uno se activara a su debido tiempo en el contador de 32 bits.

Cantidad de horas invertidas

Etapa	Tarea	Nombre del dueño	Tiempo Estimado	**Tiempo Dedicado**	Porcentaje de Avance	Fecha de Entrega	Comentarios
Información	ectura de especificacione	Mario Castresana	30 min	30min	100%	15/08/2016	Se logró entender la totalidad del problema a solucionar
	Lectura de información relacionada a sintaxis y uso correcto de ciertas cosas de Verilog	Mario Castresana	1h	1h	100%	15/08/2016	se aclararon ciertas dudas de forma
Programación	programación del primer prototipo	Mario Castresana	1h	45min	100%	15/08/2016	se logró terminar el contador de 4 bits
	programación del módulo de 32 bits	Mario Castresana	1h	2h:38min	100%	16/08/2016	problemas con los valores iniciales indefinidos y manejo del CLK en cada contador
	definición del módulo probador y plan de pruebas	Mario Castresana	2h	2h	100%	18/08/2016	definicion completa del testbench
	debugging	Mario Castresana	2h	5h	100%	19/08/2016	el contador de 32 bits dio más problemas de lo esperado
Reporte	confección del reporte	Mario Castresana	1 hora	1h	100%	12/08/2016	Error de actualización de Windows disparó en 100% el uso del disco provocando que la computadora se volviera muy lenta, tomó 2 horas encontrar la causa y repararla.
	Presentación	Mario Castresana	20min	15min	100%	12/08/2016	terminado a tiempo

Fig. 1 distribución del tiempo para este proyecto en particular.

Inconvenientes

Cabe mencionar que para este proyecto se tuvo un problema con la máquina virtual que se usa para compilar y ejecutar simulaciones en Verilog. El problema consistió en que el disco duro virtual se llenó a tal extremo que no fue posible copiar los archivos .v fuera de la máquina virtual, dejándolos atrapados dentro de la máquina virtual. Debido al problema de espacio, Linux no era capaz de iniciar correctamente dentro de VirtualBox. Esto obligó a tener que reiniciar el proyecto basado en el último respaldo hecho del mismo.

Código conductual

Los archivos .v incluidos con este reporte contienen la descripción conductual en Verilog de ambos contadores, tanto el de 4 bits como el de 32 bits. Cabe mencionar que se siguieron las convenciones establecidas para los nombres de cada entrada y salida:

Salidas

Q: salida del contador de 4 bits.

RCO: *Ripple-Carry Out* o acarreo de salida.

Entradas

CLK: clock.

ENB: Enable activo en alto.

MODO: modalidad de operación del contador según indique el usuario (00 contar ascendente, 01 conteo descendente, 10 conteo descendente de 3 en 3 y 11 es una precarga de un valor determinado)

D: entrada para valor deseado.

Notará, que algunas veces se antepone al nombre de un puerto, una letra minúscula ya sea **w**, que indica *wire*, **r**, indica *register*, **o**, para *output* y finalmente, **i**, para *input*. Esto con el objetivo de identificar como se está usando una variable en determinado módulo.

2. Descripción Arquitectónica

La primera sección de la tarea se compone de un contador de 4 bits que sigue el diseño de la siguiente figura:

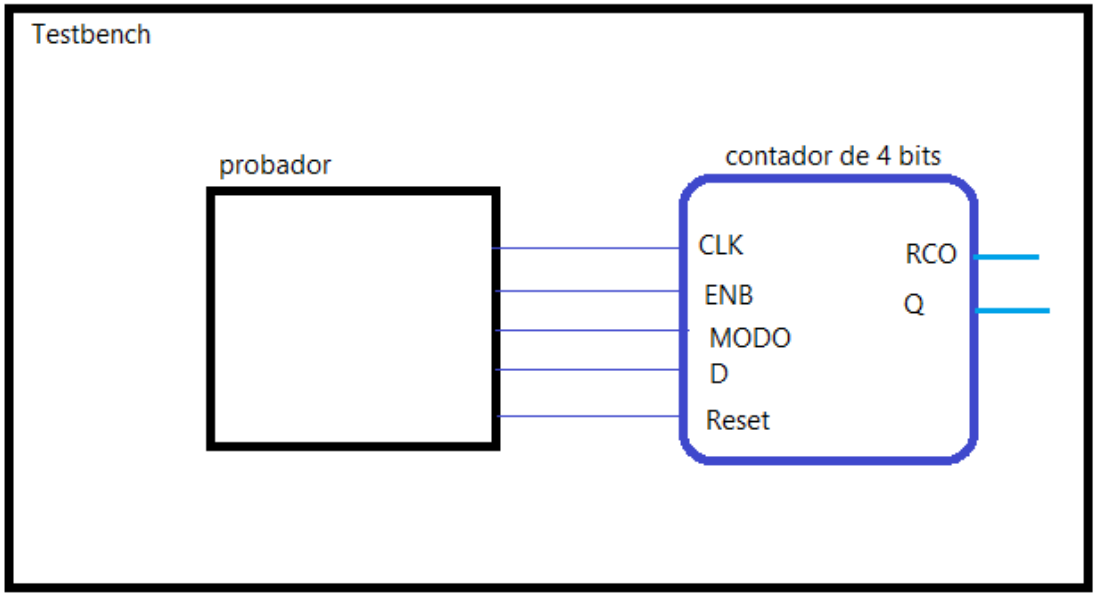

Fig. 2: arquitectura del código propuesto. Note que existe un módulo de estimulación de señales, que genera todas las pruebas de interés.

Tal y como se muestra en la figura de arriba, se tomó en cuenta las especificaciones que se adjuntaron en la tarea 2, para codificar el diseño. Se cuenta con un banco de pruebas o *testbench*, un módulo probador que genere todas las pruebas de interés para nuestro DUT y un módulo contador de 4 bits.

Descripción del módulo contador

Aunque la tarea pide explícitamente un contador de 4 bits, se decidió aprovechar las facilidades de parametrización que ofrece Icarus Verilog para generar un contador de N bits de acuerdo a las necesidades del programador. He aquí un extracto del código fuente:

```
module contador  # (parameter N = 4)
 (
output reg [N-1:0] oQ=0,      // salida Q del contador, valor inicial 0
input wire         iENB,
input wire [1:0]   iMODO,         // MODO de operación
input wire         iCLK,          // CLK input
input wire         iReset,         // reset input
input wire [N-1:0] iD,         // valor deseado de precarga
output reg         oRCO);
```

por default se utiliza un parámetro de 4 para indicar que es un contador de 4 bits.

En el caso del contador de 32 bits, este se compone de módulos más pequeños de 4 bits cada uno. Todo esto se instancia dentro de un módulo testbench que contiene un módulo probador y el contador. El módulo probador incluye todas las pruebas hechas.

La forma de conectar los contadores de 4 bits, se basa en la idea de que el RCO de uno se conecte al clock del siguiente para así habilitarlo y que use el flanco generado para contar sólo cuando se ocupa.

Cabe mencionar, que solo el contador de los cuatro bits menos significativos tiene la opción de ponerlo en modo 10 (descendente de 3 en 3), ya que solo éste debe descontar de 3 en 3, mientras los otros más significativos deben descontar de 1 en 1. Esto le evitará tener números inconsistentes al poner erróneamente todos los contadores en modo 10.

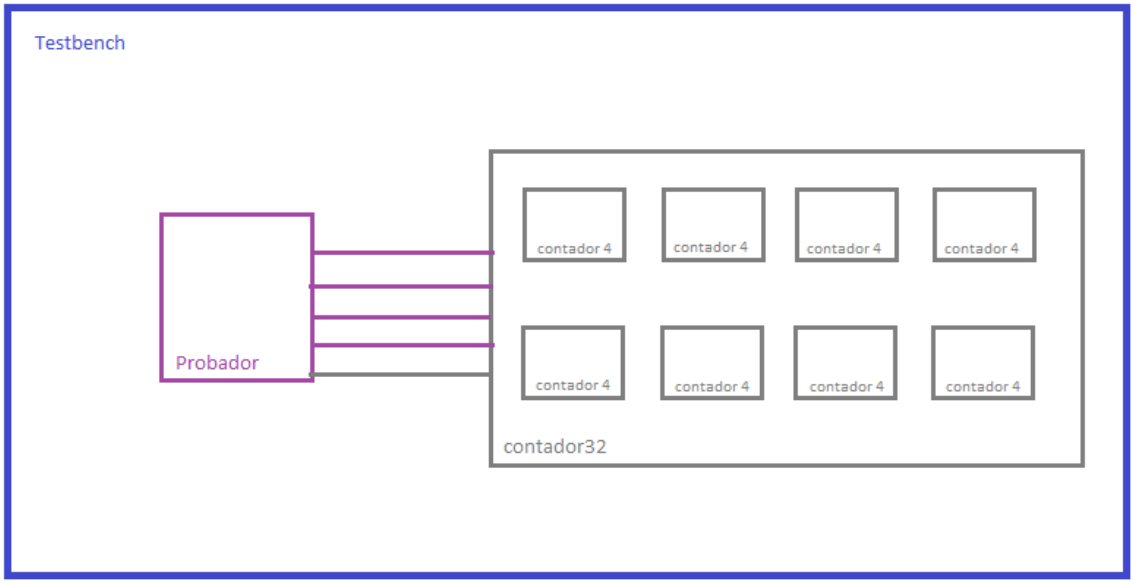

Fig. 3: diseño para el Contador de 32 bits y su respectivo plan de pruebas. Cada contador de 4 bits tiene su salida Q de 4 bits. Con ocho contadores instanciados dentro del contador32 tendremos un contador de 32 bits.

Como nota importante, se debe mencionar que el diseño propuesto en el presente proyecto incluye una entrada de Reset. Aunque no se pide explícitamente en las especificaciones, si se consideró importante ya que sin esa entrada los valores iniciales del contador nunca están definidos. Para evitar esta situación se incluye una señal de Reset en cada contador y al inicio de las pruebas se le manda un pulso de Reset a cada contador.

3. Plan de pruebas

Para el contador de 4 bits y el de 32, se montaron en un solo módulo probador las siguientes pruebas:

1. Contar hacia arriba

2. Contar hacia abajo
3. Contar de 3 en 3
4. Carga en paralelo

En la figura 4 se puede apreciar las pruebas 1 y 2.

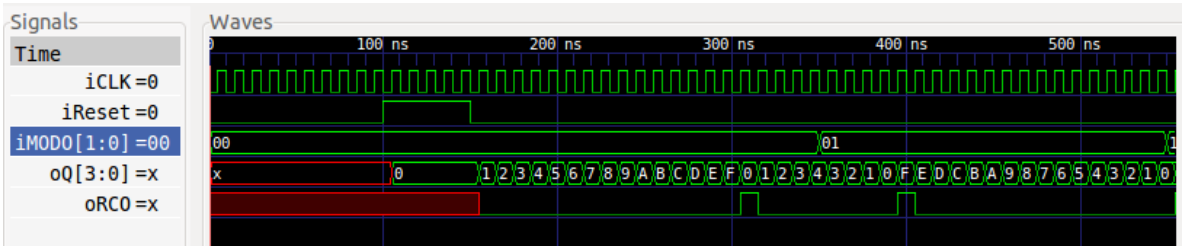

Fig. 4: en esta sección del test se puede apreciar la cuenta ascendente y descendente. Note que el carry RCO se produce justo después de sumar 1 al valor 0xF, lo cual es correcto. Note la ventaja de agregar un pulso de Reset para eliminar los valores indefinidos X.

Para las pruebas 3 y 4, se puede apreciar en la figura 5 que el carry en el caso de la cuenta ascendente de 3 en 3 se genera en el momento que pasamos a un número menor o igual que cero, lo cual resulta conveniente para el contador de 32 bits que se construirá luego.

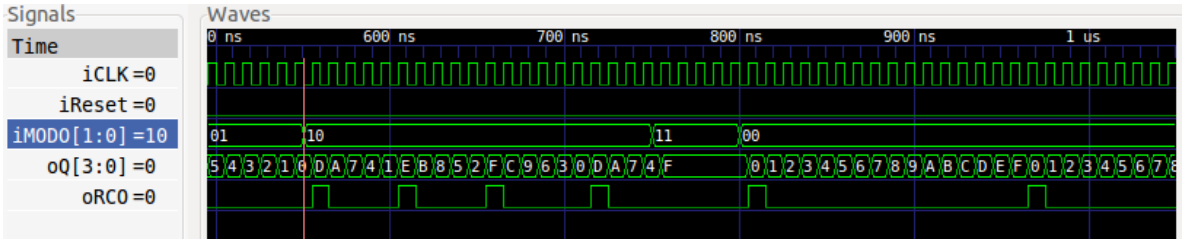

Fig. 5: pruebas para contar de forma descendente de 3 en 3 (iMODO=10). Se puede apreciar que el RCO se pone a 1 cuando resto y obtengo un valor igual o menor que 0 (es decir, cuando el contador da la vuelta). Al activar el modo de precarga iMODO=11 y precargando 0xF, se aprecia que el contador cuenta correctamente a partir de ese valor frontera.

Problemas encontrados

1. Valores indefinidos al principio de la simulaciones provocan un comportamiento errático en el contador: aunque esto se arregla fácilmente precargando un valor definido al principio de la simulación, esto no es una solución válida para el caso del contador de 32 bits, ya que la precarga funciona bien en el primer contador, pero en los otros no surte efecto al no tener estímulo del CLK hasta recibir el RCO de contador anterior.
2. Se agregó una parte de lógica combinacional para definir el modo del contador y otra secuencial para definir el comportamiento del contador, ya que los modos presentan problemas de timing al distribuirse a todos los contadores dentro del contador de 32 bits. Esto quiere decir que el contador empezaba a contar

erróneamente si se cambiaba el modo en los valores frontera, donde era necesario generar un carry para activar el contador adyacente.

3. Se agregó el Reset y valores iniciales para evitar comportamiento impredecible al inicio de las simulaciones o en el caso particular que exista un valor indefinido al inicio, que no sea por más de un ciclo de reloj.

4. Instrucciones de simulación

Las simluaciones se corrieron de manera simple usando los comandos

Para el contador de 32

$ **iverilog** -o prueba32 contador.v contador32.v probador.v **testbench.v**

$ **vvp** prueba32 > salida

(esto permite ver los valores del contador generados en un archivo llamado salida en el presente directorio)

$ **gtkwave** testContador.vcd

Para el contador básico

$ **iverilog** -o prueba contador.v probador.v **testContador.v**

$ **vvp** prueba &

$ **gtkwave** testContador.vcd

5. Resultados

Para esta sección, se muestran los resultados del plan de pruebas propuesto en la tarea 2.

1. Contar ascendente

La prueba fue completada con éxito, demostrando que los contadores efectivamente son capaces de generar los RCO a tiempo para activar al siguiente a su debido tiempo. (Revisar archivo adjunto salida, ya que la salida completa es muy larga)

Condición de frontera (precargando valor 0x0000FFFF)

Valor binario	decimal	MODO
00000000000000000000000000000000	**0**	**X**
00000000000000000111111111111xxxx	X	11
00000000000000000111111111111111	**65535**	**11**
00000000000000001000000000000000	**65536**	**00**

En el archivo incluido salida se puede apreciar bien la salida completa del programa

2. Contar descendente

Valor binario	decimal	MODO
00000000000000001000000100101011	**65835**	**01**
00000000000000001000000100101010	**65834**	**01**
00000000000000001000000100101001	**65833**	**01**
00000000000000001000000100101000	**65832**	**01**

3. Contar descende de 3 en 3

Valor binario	decimal	MODO
00000000000000001000000001100011	**65635**	**10**
00000000000000001000000001100000	**65632**	**10**
00000000000000001000000001011101	**65629**	**10**
00000000000000001000000001011010	**65626**	**10**

6. Conclusiones

- El proyecto fue una muy buena oportunidad de aprender a cómo usar la modularidad y la estrategia "divide y vencerás" para resolver un problema. Sin embargo, para el caso del contador de 32 bits, resulta mucho más sencillo utilizar las ventajas de parametrización que ofrece Verilog a la hora de hacer módulos. Así pues se puede hacer un contador de 32 bits cambiando el parámetro N (mostrado en contador.v) por 32 e instanciar contador como

  contador #(32) Cont0

  en vez de hacerlo como el resultado de juntar 8 contadores de 4 bits. Esto hubiera ahorrado mucho tiempo y nos permite explorar otras facilidades del lenguaje.

- El timing resultó ser un reto interesante en este diseño, ya que si no se toma en cuenta, los resultados de hecho son desastrosos.
- Para la próxima tarea sería interesante incluir una pequeña introducción a cómo hacer un pequeño módulo checker para comprobar automáticamente una pequeña matriz de pruebas.