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

Roberto Sánchez Cárdenas - B77059

San José, 16 de octubre de 2020
Circuitos Digitales II

Índice

1. Introducción

El presente trabajo presenta la descripción conductual y estructural de una máquina de estados finita. La descripción conductual se relizó en iVerilog y la estructural se obtuvo por medio del software Yosys. Dadas las optimizaciones utilizadas, se observan comportamientos interesantes que serán mencionados a continuación.

Se desea que la máquina de estados posea un comportamiento idéntico al siguiente y que la salida se como la que se muestra.

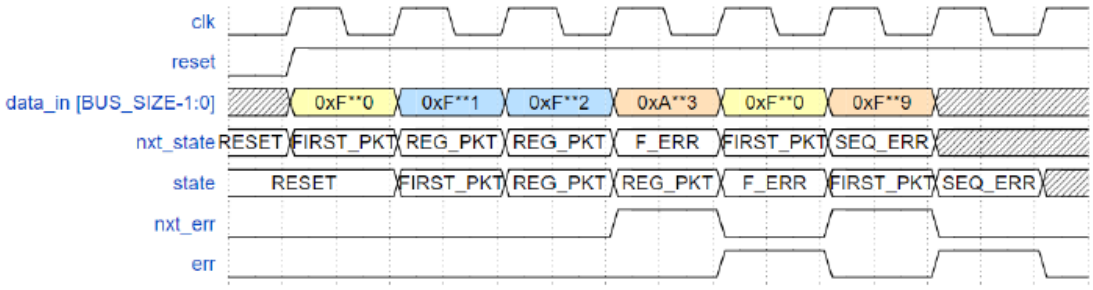

Figura 1: Comportamiento de la máquina de estados

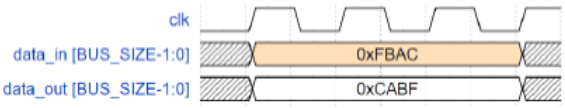

Figura 2: Salida de datos

Se puede observar que el estado actual de la máquina de estados, depende del estado en el que viene. Para ello se hizo una implementación de una sección de lógica combinacional, que calcula el próximo estado y una parte basada en flip flops que deja pasar estos estados cuando ya se tuvieron que haber calculado. La salida de datos es igual a la entrada pero en orden inverso. También se tiene una salida de control que es una compuerta or wise de case bit de cierta palabra, esto se puede implementar buscando si la palabra son solo 0's o si hay algún 1.

2. Resultados

En esta sección de muestran los resultados y una explicación de como se obtuvieron. Para el diseño de la máquina de estados se tiene una lógica que permite calcular el próximo estado basado en el estado en el que se encuentra. Esto se hizo por medio de cases basados en el estado actual. En ellos se revisan las condiciones que se deben cumplir. El módulo de cálculo de salidas y control se hizo con lógica de generates, y la lógica es muy simple. El módulo solo toma una entrada y la va a dar a una salida, sin embargo, implementar este módulo con generates y fors, se le indica que dados ciertos bits de entrada, estos se comiencen a acomodar en orden inverso en la salida. Se muestran los resultados en las siguiente figuras.

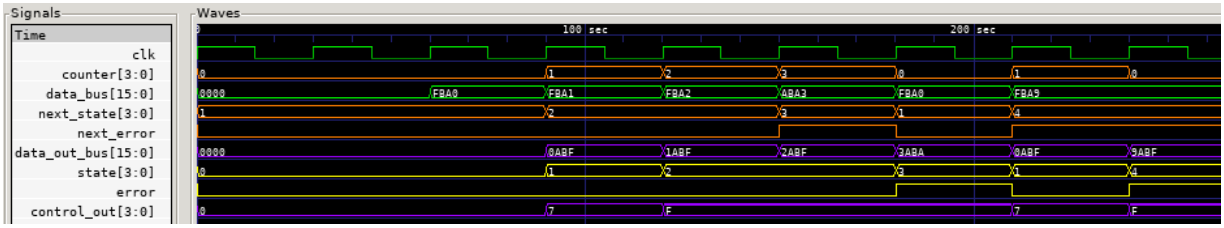

Figura 3: Respuesta de módulos conductuales

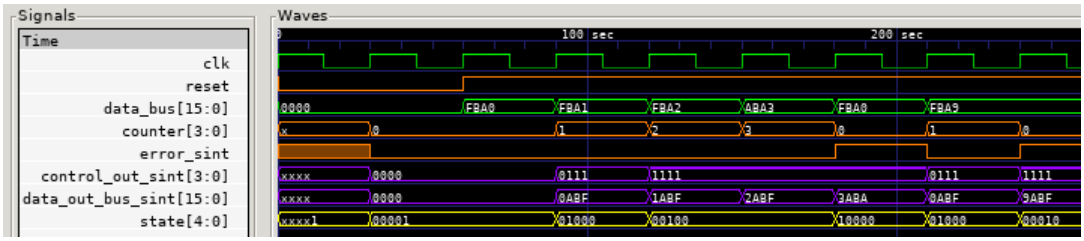

Figura 4: Respuesta de módulos sintetizados

Ambos módulos poseen un comportamiento como el esperado, el cual se muestra en las figuras 1 y 2. Hay una pequeña diferencia en el módulo estructura debido a las optimizaciones que hace Yosys en las máquinas de estado. Esta optimización tiene que ver con los estados, que en el caso del módulo estructurar se observa que la codificación es modificada para ser de tipo one-hot, la cual busca que únicamente haya un bit del código de estados en 1 y los demás en 0, esto se hace para reducir la lógica necesaria en la detección de los estados. Esta optimización se debe a la que se realiza al usar fsm; opt; en yosys.

Cabe destacar que la salida de datos no detecta cuando hay un error, esta siempre da la salida en orden inverso, por lo que si este módulo se quisiera conectar a otro dispositivo, se tendría que validad el error para ver cuáles datos son válidos y cuáles no.

3. Verificador

Para comprobar los resultados se diseñó un módulo de verificación en el que se compraran las señales del módulo estructural y conductual. Se observa que dado que los módulos tienen un funcionamiento idéntico, por lo tanto las salidas de checkeo se mantienen en bajo.

Figura 5: Checkeo de salidas

4. Conclusiones

- Los módulos generate permiten hacer lógica simple que puede ser complicada si se implementa pensando en un único módulo que haga todo el trabajo, dado que permiten decir específicamente qué parte de una entrada quiero ingresar y en donde la quiero sacar.
- La codificación one-hot simplifica la lógica utilizada a nivel estructural, pero si no se hace de esta manera en el conductal pueden existir diferencias al comparar.
- Los for loops simplifican en gran medida ciertas operaciones y son maneras de hacer operaciones completas en un solo ciclo de reloj.

5. Simulación

Para correr el código de esta simulación, basta con descomprimir el archivo, ir a la dirección en la que está el documento y correr el comando **$make**.

6. Tiempo de desarrollo

- Código: 3 horas
- Debuggeo: 3 horas
- makefile: 5 minutos
- Reporte: 1 hora