UNIVERSIDAD DE
COSTA RICA

Proyecto II
Circuitos Digitales II

Andrés Alvarado, B30316
Alejandro Rojas Cordero, B36049
Fernando Gonzáles, B43023

Escuela de Ingeniería Eléctrica
Universidad de Costa Rica
San José, Costa Rica

I-2018

Índice

abstract

En este proyecto se diseñará la capa de transacción de PCIe, a partir de 3 grandes módulos, el Round Robin, FSM y el FIFO

1. Objetivos

1.1. Objetivo General

- Implementar la capa de transacción PCIe adaptada a partir de los 3 módulos ya mencionados.

1.2. Objetivos Específicos

- Realizar una investigación sobre QoS, Arbitraje en sistemas digitales, priority flow control.
- Verificar el correcto funcinamiento de cada módulo por separado para poder unirlos.
- Sintetizar el diseño en Yosys y comprobar su correcto funcionamiento.

2. Bitácora

Etapa	Dueño	T. Estimado	T. Dedicado	Avance	Entrega	Comentarios
RoundRobin	Fernando	4	7	100 %	20/06	
FSM	Alejandro	4	6	100 %	25/06	
FIFO	Andrés			100 %		
Interconexión	Fernando y Andrés	4	10	100 %	30/06	
Pruebas	Andrés y Alejandro	3	5	100 %	30/06	
Síntesis	Fernando	1	3	100 %	30/06	Formato .dot
Documentación	Todos	1	3	100 %	30/06	

3. Investigación

3.1. QoS

La calidad de servicio o Quality of Service (QoS), es el rendimiento promedio de una red de computadoras, visto por los usuarios de la red. Este mide la calidad de los servicios que son considerados en varios aspectos del servicio de red, tales como tasas de errores, ancho de banda, rendimiento, retraso en la transmisión, disponibilidad, entre otros. Esto implica que la calidad de servicio sea muy importante para el transporte de tráfico con requerimientos especiales, ya que muchas aplicaiones hoy en día requieren de procesamiento de audio o video en tiempo real.

3.2. Arbitraje en sistemas digitales

Cuando se debe tomar una decisión respecto al orden de 2 señales asíncronas, se requiere el arbitraje para determinar cuál señal tiene prioridad sobre la otra, este se usa cuando las señales son solicitadas para una fuente compartida, como un bus de datos o una unidad funcional compartida.

3.3. Priority Flow Control

En una ruta de red que normalmente consta de múltiples saltos entre el origen y el destino, la falta de retroalimentación entre transmisores y receptores en cada salto es una de las principales causas de falta de fiabilidad. Los transmisores pueden enviar paquetes más rápido de lo que los receptores aceptan paquetes, y cuando los receptores se quedan sin espacio en el búfer disponible para absorber los flujos entrantes, se ven obligados a abandonar silenciosamente todo el tráfico que excede su capacidad. Esta semántica funciona bien en la Capa 2, siempre y cuando los protocolos de capa superior manejen la lógica de detección de caída y retransmisión.

Para las aplicaciones que no pueden generar confiabilidad en las capas superiores, la adición de funciones de control de flujo en la Capa 2 puede ofrecer una solución. El control de flujo permite la retroalimentación de un receptor a su emisor para comunicar la disponibilidad del búfer.

En el sistema que utiliza control de flujo basado en crédito, el receptor no necesita proporcionar grandes búfers para el control de flujo, porque el emisor sabe cuánto espacio queda en los búfers del receptor. Especialmente, se puede ahorrar espacio en el buffer de reserva para la demora del cable.

Si no se garantiza ningún crédito (espacio) en el lado del receptor, el remitente no enviará sus datos porque sabe que los datos se perderán.

4. Resultados

Como se dijo anteriormente, nuestro proyecto está dividido en tres grandes bloques y uno pequeño, de los cuáles veremos sus resultados a contunuación, para luego explicar el funcionamiento de los cuatro trabajando al mismo tiempo.

4.1. Instrucciones de utilización de la simulación

- Para simular la FSM hacer "make.en la carpeta FSM.
- Para el resto de simulaciones hacer "makez luego "make gtk".

4.2. Round Robin

Para esta sección del circuito, se utilizan dos partes principales. La primera es un ciclo en donde recibe un stream de bits para separarlos en un array temporal de dos en dos. Esto le permite luego al Round Robin tener una tabla de orden de prioridades. La segunda parte es similar a una máquina de estados en donde comienza a hacer peticiones a los FIFOs mientras revisa si tienen datos válidos o si no existe una pausa en el proceso.

Al encontrar una petición válida levanta un bit de read para que la etapa siguiente sepa que es una petición exitosa y que puede leer el bit sin ningún problema. Además, posee un bit de valid para permitir el funcionamiento del módulo. En caso de recibir una petición de pausa, este bit pasa a ser 0 y se detienen las peticiones hasta recibir la señal de continuación.

4.3. Instrucciones de utilización de la simulación

- Para simular el round robin hacer "make all.en la carpeta Round Robin.
- Para sintetizar hacer "make sintesis".

Las pruebas realizadas son:

1. Prueba de ciclo for: se pasa un stream de bits y se imprime el arreglo temporal para verificar que los datos se dividieron de manera correcta. Esta prueba fue exitosa.
2. Prueba de bit de read: Se revisa que el bit de read se encienda cuando comienza la transmisión de una petición válida. Esta prueba fue exitosa.
3. Prueba de empty: Se utilizan valores de 1's en empty para evitar que se hagan peticiones al FIFO que se encuentre lleno. Esta prueba fue exitosa.

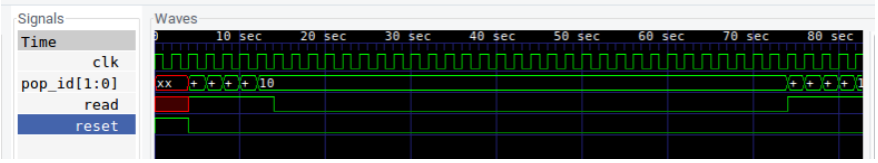

Figura 1: Round Robin simulación GtkWave

Como se observa en la figura 1, los cambios en la selección se realizan exitosa y rápidamente, en cada ciclo de reloj. El bit de read se levanta cuando existen peticiones exitosas y baja cuando la petición no se realiza por encontrarse con un FIFO vacío en el momento.

4.4. FSM

Para el diseño de esta máquina de estados, se utilizaron 6 estados:

- Estado Reset
- Idle
- Active
- Pause
- Continue
- Error

La máquina empieza en un estado de reset, el cuál pasa al Idle cuando esta señal está en bajo, este estado permite que se den todos los procesos de lectura y escritura, pero de estos estados el más importante es el estado Active, ya que en este se van a realizar todos los procesos para la implementación de la capa de transacción, este va a mandar las señales a los 3 estados siguientes, que son más como señales de transición entre los procesos, la idea de los estados de Pause y Continue es dar un aviso de que esta pasando con nuestros datos, pero inmediantamente devolverse al estado Active, por último el estado de Error, se da cuando hay un .ºverflow"de datos en el FIFO.
En la siguiente imagen podemos apreciar como se da la transición de datos entre los estados de nuestra máquina FSM, donde tenemos un Data-Fifo, el cuál va a mandar el dato en las señales pause, continue y error-full, si se da el cambio a uno de estos estados.

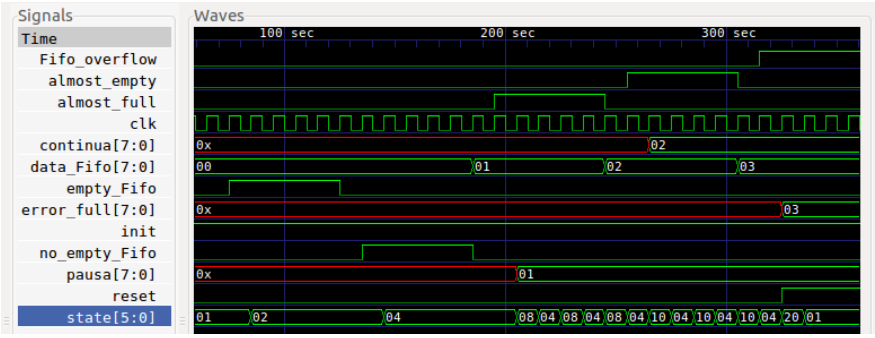

Figura 2: FSM simulación GtkWave

4.5. FIFO

Para este modulo se utilizó una maquina de estados para saber en que posicion de memoria se tiene que escribir.
Tiene 2 señales de entrada WRITE y READ que son utilizados para indicar si se dará una lectura y/o escritura. Cuando hay solo lectura se lee la primera posicion de lectura la cual va de 0 a 7, y posteriormente mueve todos los datos una posicion hacia abajo, es decir la 15:8 pasa a ser 7:0 para que se vaya limpiando y siempre se lea el primer dato que se ingresó. Si hay una escritura se fijará en que posicion esta y escribe en esa. Por si ejemplo esta en la posicion 4 escribira en los bits 31:24.
Tambien tiene un input llamado BIG el cual si esta encendido significa que el FIFO es de 16 bytes y sino es de 8 bytes.

El FIFO devuelve como output la informacion guardada y ademas 4 señales de información, Empty (si el FIFO esta vacio), Full (si el FIFO esta lleno), almostEmpty(Se esta quedando vacio) y almostFull (Se esta quedando lleno) y 2 señales de error, errStackOverflow (Trato de escribir cuando el FIFO estaba lleno) y errNoData (Se trata de hacer una lectura donde no hay datos).

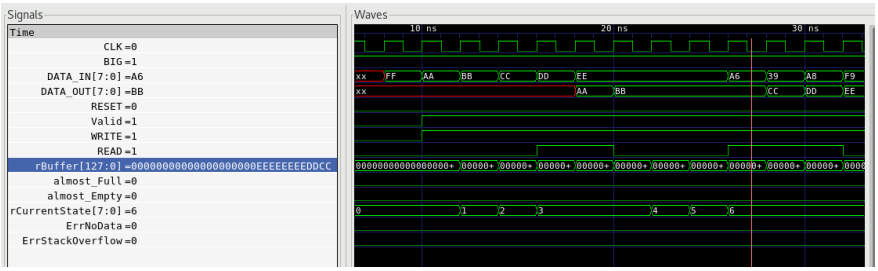

Figura 3: FIFO simulación GtkWave

Como se puede ver en la figura 3 La información que se le ingresa mediante data_in es la misma y en el mismo orden que sale en data_out.

4.6. Memory

Es un modulo para guardar información. Este se implementa con una matriz de 8 secciones de 8 bits. Para guardar informacion se tiene 2 punteros. Uno de escritura y el otro de lectura. Ambos punteros tienen 2 tipos de datos, indexR y posicionR. El index lleva que posicion tiene de la matrix y la posicion cuantas veces se ha utilizado.

Como en el FIFO tiene 2 inputs READ y WRITE que se tienen que activar para la lectura y escritura e igual las 2 señales de error, errOverflow y errNoDAta. Para saber que no hay información y tirar el error de errOverflow es cuando posicionR + 8 ¿posicionW. Y errNoData es cuando posicionR ¿posicionW.

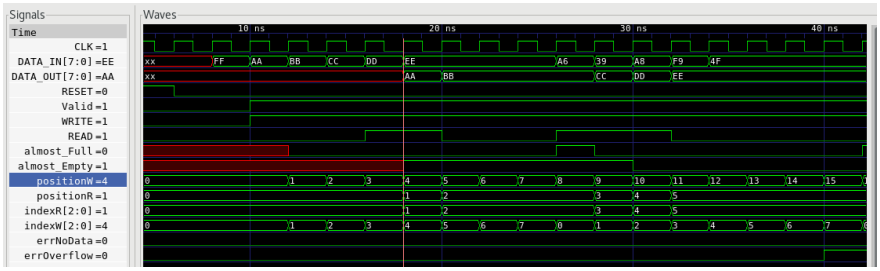

Figura 4: Memory simulación GtkWave

Como se puede ver en la figura 4 al igual que en el FIFO se ve como los datos en data_out son los mismos que entran en el data_in. Ademas se puede ver como hay un errOverflow ya que la posicionW ¿8 + posicionR

4.7. Módulo principal

Como se observa en la figura **??**, las señales pasan de las salidas de los FIFOs superiores a la entrada y luego salida del FIFO inferior siguiendo el patrón de pesos que el Round Robin maneja.

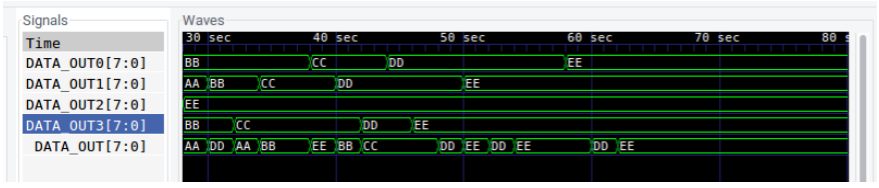

Figura 5: Módulo completo simulación GtkWave

Por lo tanto se puede afirmar que el módulo funciona correctamente.

5. Conclusiones

Del modelo diseñado se puede concluir que es un módulo que funciona correctamente si hablamos de módulos separados. Cada módulo por sí solo cumple sus funciones especificadas del diseño y las pruebas son exitosas. Caso contrario, el módulo en conjunto no pasó algunas de las pruebas por lo cual significa que aún se debe mejorar y revisar a profundidad.

Subdividir el diseño en pequeños módulos funcionales permitió solucionar problemas que tal vez en el modelo general no se hubieran encontrado con facilidad. Así mismo se pudo avanzar de manera más rápida dividiendo tareas y observando cómo las mismas se interconectaban y dependían de las etapas contiguas para realizar un diseño ideal que no presentara problemas de comunicación a futuro.

La etapa de síntesis se vio beneficiada por la subdivisión de los módulos, ya que permitió revisar errores de síntesis que pudieron haber afectado el módulo principal y hubiese sido casi imposible de encontrar el error.

Referencias

[1] Diccionario Español de Ingeniería. *Real Academia de Ingeniería de España*. 1er Edición. 2014.

[2] Digital Systems Engineering. *William J Dally John W Poulton*. Cambridge University Press. 1998.

[3] Priority Flow Control: Build Reliable Layer 2 Infrastructure. *Cisco*. Extraído el 29 de Junio de 2017 de: https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white-paper-c11-542809.html.