Diseño de un generador de transacciones MDIO

Chacón Mora, Daniel Mauricio

14 de junio de 2022

Resumen

El circuito descrito a continuación tiene la función de generar transacciones MDIO de acuerdo con las especificaciones estipuladas en la cláusula 22 del estándar IEEE 802.3. Para su correcto funcionamiento, este diseño viene con diferentes pruebas las cuales corrieron sin problema. La naturaleza del problema a resolver hace que la comprensión jerárquica de los diferentes módulos sea dificil de visualizar al igual que su correcta implementación.
Palabras claves: *Síntesis, carga en serie, carga en paralelo.*

1. Descripción arquitectónica

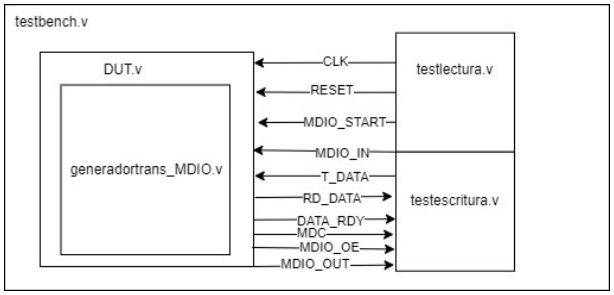

Figura 1: Estructura jerárquica

El convertidor está constituido por un módulo de banco de pruebas(testbench.v), dos módulos probadores que se encargan de realizar la prueba de lectura(testlectura.v) y escritura(testescritura.v) y un módulo *DUT* que instancia el módulo de generador de transacciones MDIO.

Para reconocer de qué operación se trata, se utilizaron declaraciones condicionales encargadas de verificar los valores presentes en T_DATA[29:28] con el objetivo de realizar la distinción entre *lectura* y *escritura* sin necesidad de hacer uso de una máquina de estados.

2. Plan de pruebas

2.1. Prueba 1: Transacción de lectura

Para realizar la prueba se establece que:

- RESET = 1
- MDIO_START = 1
- Se define para T_DATA un valor hexadecimal de 0x61115F1F.
- Se utiliza función de verilog para generar números random $urandom% que serán los cargados en la salida serial RD_DATA debido a que el valor de T_DATA únicamente define que operación se realizará.
- Se establece RESET=0 y todas las salidas se ponen en 0 en el siguiente flanco positivo.

2.1.1. Resultado

La salida RD_DATA obtiene los 16 bits adecuados luego de que la salida DATA_RDY se pone en alto luego de completar la recepción de los 32 bits de la palabra serial.

La señal MDIO_OE se mantiene en alto durante los primeros 16 bits y luego se pone en bajo, de esta manera se confirma que se realizó exitosamente la transacción de lectura.

2.2. Prueba 2: Transacción de escritura

Para realizar la prueba se establece que:

- Se inicia con MDIO_START = 0 y RESET = 1
- Se define para T_DATA un valor hexadecimal de 0x51115F1F.
- Se utiliza función de verilog para generar números random $urandom % que serán los cargados en la salida serial RD_DATA debido a que el valor de T_DATA únicamente define que operación se realizará.
- Se establece RESET=0 y todas las salidas se convierten en 0 en el flanco positivo siguiente.

2.2.1. Resultado

La salida MDIO_OUT transmite los 32 bits contenidos en la entrada T_DATA. Las salidas seriales DATA_RDY y RD_DATA se mantiene en bajo debido a que la transacción correspondiente a la prueba es de escritura, por lo tanto no hay ingreso de valores seriales y dichas señales mencionadas nunca logran ponerse en alto.

Se observa que la señal MDIO_OE se mantiene en alto durante los 32 ciclos de transacción y se pone en bajo una vez que se termina la escritura, de esta manera se confirma que la transacción se realiza de forma correcta.

2.3. Prueba 3: Sintetización

Se deben correr los siguientes comandos:

- read_verilog generadortrans_MDIO.v
- proc;opt;techmap;opt;
- dfflibmap -liberty cmos_cells.lib
- abc -liberty cmos_cells.libwrite
- write_verilog noatrr generadortrans_MDIO_s

  Se verifica que se genere una lista de compuertas utilizadas.

2.3.1. Resultado

La síntesis, por motivos que se desconocen, no se realiza de forma correcta. La cantidad de compuertas utilizadas es anormalmente baja y el archivo sintetizado carece de sentido.

3. Instrucciones de simulación

Dentro de los archivos se encuentra un *makefile* con los comandos necesarios para generar el archivo que se utilizará en el visualizador de ondas.

4. Ejemplo de resultados

4.1. Prueba 1: Transacción de lectura

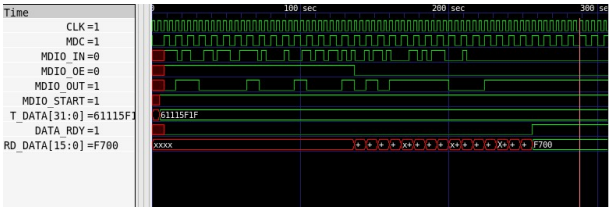

Figura 2: Ondas de transacción de lectura

La señal RD_DATA inicia indefinida debido a que en ese momento aún no se ha alcanzado la señal en alto de DATA_RDY que indica la transmisión completa de la palabra serial. Una vez DATA_RDY se pone en alto, se carga en RD_DATA los valores correspondientes.

4.2. Prueba 2: Transacción de escritura

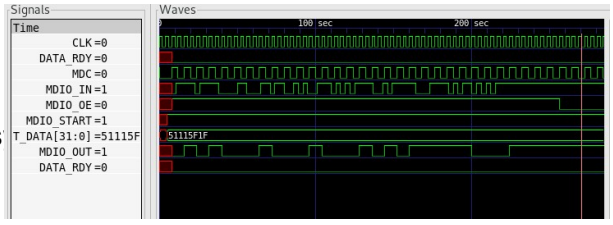

Figura 3: Ondas de transacción de escritura

Debido que la transacción es de escritura, la señal serial DATA_RDY nunca llega a ponerse en alto.

4.3. Prueba 3: Sintetización

```
7.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:        3
ABC RESULTS:               NOR cells:        3
ABC RESULTS:               NOT cells:        4
ABC RESULTS:        internal signals:        9
ABC RESULTS:           input signals:        6
ABC RESULTS:          output signals:        3
Removing temp directory.
```

Figura 4: Cantidad de compuertas utilizadas

Se observa como al sintetizar utilizando *yosys* se obtiene una cantidad muy baja de compuertas utilizadas.

```
generadortrans_MDIO_synth.v
generadortrans_MDIO.v
```

Figura 5: Error al generar descripción estructural

Al utilizar el comando write_verilog se genera el archivo sintetizado correspondiente, sin embargo es sospechoso debido a que el análisis visto en clases demostró que de ser posible la síntesis en esta práctica en particular, no tenía sentido.

5. Dificultades

Para esta tarea la parte más dificil de todas fue la inicial, el definir bien la jerarquía de módulos y su funcionamiento entre ellos, además de hallar la manera de identificar los bits correspondientes a lectura o escritura sin necesidad de utilizar una máquina de estados.

6. Conclusiones y recomendaciones

En conclusión, se aprendió el manejo de diferentes módulos especializados en una tarea para luego utilizarlo en conjunto con otros para el trabajo de forma paralela, incluyendo un módulo con la definición de estados.

Como recomendación, sería bueno que antes de asignar tareas tan especializadas se intente hacer algo similar antes para estar seguros de que partes como la síntesis sí sea posible de realizar debido a que en clases con el profesor no fue posible encontrar una solución a ese problema entre todos.