Reporte de Proyecto de Laboratorio 2: Núcleo SPI Maestro

Mauricio Caamaño, Marco Espinoza, Tomás González

***Abstract*—El presente reporte detalla el proceso de ejecución para el desarrollo de una interfaz SPI maestro implementada en HDL para una FPGA de Altera. El diseño se elaboró siguiendo el ciclo de desarrollo recomendado para prototipado en FPGAs. Se demuestra la implementación del sistema, la cual involucró la implementación de una máquina de estados finita, memorias FIFO mediante el uso del Core IP de Altera y otros bloques funcionales. Las simulaciones, realizadas en ModelSim de Altera, muestran el comportamiento esperado del sistema bajo diferentes escenarios de prueba tanto de la transmisión como de la recepción de datos por la interfaz SPI.**

***Index Terms*—SPI, FIFO, FSM.**

I. Introducción

El sistema a desarrollar en este proyecto trata de una interfaz SPI maestro implementada con memorias FIFO tanto para la transmisión como la recepción de datos, el cual se puede integrar como parte de un sistema más grande que requiera comunicarse con otros dispositivos a través del protocolo SPI.

En este reporte se describe el proceso llevado a cabo para la elaboración del sistema siguiendo el proceso de ciclo de desarrollo para prototipado de FPGAs. Este proceso involucra la ejecución de las etapas de especificación, diseño de arquitectura, implementación y verificación, las cuales son descritas en cada una de las secciones posteriores.

II. Especificación de Requerimientos

Como procedimiento inicial para la elaboración de la interfaz SPI, se definió un documento de especificaciones que contiene una descripción de los requerimientos funcionales de las operaciones de transmisión y recepción. Además, el documento contiene restricciones de diseño, así como una sección dedicada al plan de pruebas a ejecutar para verificar la funcionalidad del sistema. La creación de este documento fue de gran ayuda para delimitar las capacidades del sistema y evitar la omisión de funcionalidades, así como conocer de antemano las pruebas suficientes para verificar el diseño, sin caer en casos redundantes o innecesarios. El apéndice A contiene el documento de especificaciones realizado al inicio del ciclo de desarrollo para el proyecto.

III. Arquitectura del Diseño

El diseño del núcleo SPI maestro parte del cumplimiento de las especificaciones, que contiene requerimientos de implementación para algunos de los bloques funcionales, como las memorias FIFO. El apéndice B contiene el diagrama de bloques del diseño completo, elaborado antes de comenzar con la implementación en HDL del sistema. El sistema es controlado mediante una máquina de estados finita, que controla tanto las operaciones de recepción como transmisión. Las memorias tipo FIFO para cada operación permiten que el sistema pueda recibir una serie de datos a transmitir, aún cuando la FSM se encuentra ocupada ya sea recibiendo o transmitiendo por SPI. Esto permite cumplir con las especificaciones del diseño. A continuación se describe la funcionalidad de cada uno de los bloques del diseño. Además, se realizó el ejercicio de estimar los recursos utilizados por cada bloque funcional.

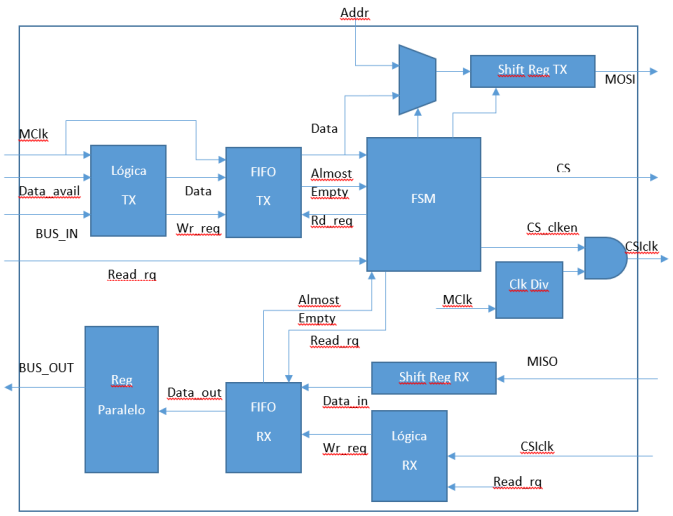

Figure 1. Diagrama de bloques del SPI

A. *Máquina de estados finita*

Esta máquina de estados se encarga de controlar toda la lógica del módulo SPI. Es el árbitro que le indica al resto de los módulos qué hacer. Se comunica con los FIFOs para indicarles en qué momento leer datos y le indica a los registros de corrimiento en qué momento moverse. Para poder guardar todos los estados y las variables internas, se estimó que la máquina de estados consuma alrededor de 10 a 20 registros. Tiene que haber un registro de unos 4 bits que guarde los estados. Además un registro que guarde el selector de mux que envía datos o dirección hacia el registro de corrimiento. Otras señales que tiene que utilizar registros son las señales que se comunican con los FIFOs para indicar la soliicitud de datos. Siendo este módulo el que toma las desiciones, se estima que sea el que más lógica combinacional consuma, utilizando cerca de 40 celdas lógicas combinacionales.

B. FIFO de transmisión

Este módulo es un FIFO de 15x8 bits que almacena los datos que entran por `BUS_IN` y los almacena hasta tener 15 bytes, en ese momento se le indica a la máquina de estados que el FIFO está listo y que se puede empezar a transmitir. Se van a necesitar registros para los datos de `full`, `almost_full`, `empty`, `usedw` y `q`. Se estimó que se necesiten unos 10 a 15 registros. Si se implementa con una memoria MLAB, debería utilizar sólo un bloque, ya que es lo suficientmente pequeño como para caber en uno.

C. FIFO de recepción

También es un FIFO de 15x8 bits y al igual que el de transmisión, se espera a que se llenen 15 bytes y luego se empieza a vaciar para entregarle los datos al registro paralelo. Al igual que el FIFO de transmisión, se estimaron de 10 a 15 registros. También debería caber en un módulo MLAB.

D. Divisor de reloj

Este módulo toma un reloj de 15 MHz y genera un reloj de 3,125 MHz. Este es un módulo sencillo que necesita un contador y un registro para almacenar el resultado. El contador necesita ser de 4 bits, por lo que en total se estimaron 5 bits.

E. Registro de corrimiento de transmisión

Es un registro de corrimiento de 8 bits que con un load carga un dato completo de 8 bits y en modo normal hace shift de esos datos hacia una salida de 1 bit. Al ser un registro con 8 bits, se estimó que se utilicen 8 registros.

F. Registro de corrimiento de recepción

Es un registro de corrimiento que toma una entrada de 1 bit y en cada ciclo de reloj va metiendo ese bit en el campo menos significativo de la salida y va corriendo hacia la izquierda los bits de la salida. La salida es de 8 bits. Al igual que el registro de corrimiento de transmisión, se estimó que se utilicen 8 registros.

G. Lógica previa de transmisión

Toma las señales de entrada y las convierte para que FIFO de transmisión las pueda entender de manera correcta. Esta lógica recibe dos señales y las envía al FIFO por lo que solo necesita 2 registros.

H. Registro de salida paralelo

Es un módulo que toma la salida de 8 bits del FIFO de recepción y va acomodando los datos en un gran registro de 8x15 bits, al final `BUS_OUT` es un bus de 120 bits que va a contener todos los 15 bytes de la transmisión Al convertir una señal de 8 bits e irla poniendo en un bus de 120 bits, se estimó que se necesitan 120 registros para guardar esos bits.

IV. Implementación en HDL

Esta sección describe el proceso de implementación en RTL de los principales módulos del diseño: Las memorias FIFO de transmisión y recepción, los registros de corrimiento, la máquina de estados finita y el bloque de registros de salida en paralelo. El sistema además incluye otra lógica de control adicional, descrita en la sección anterior de arquitectura de diseño.

A. Memorias FIFO

Para la implementación de las memorias tipo FIFO se utilizó el Core IP de Altera de este commponente, el cual provee un ayudante o "Wizard" para personalizar el tamaño, tipo de FIFO y señales de control. La figura 2 muestra las señales elegidas, que comprenden los buses de entrada y salida de datos en paralelo `data` y `q`, respectivamente; señales de solicitud de escritura y lectura `wrreq` y `rdreq`, señal para indicar el llenado del FIFO a 15 bytes `almost_full`, así como `empty` para indicar estado vacío del FIFO. Tanto la lectura como escritura a través del componente, se sincronizan mediante la señal de `clock`. Para iniciar el sistema, se envía la señal de reset a través de la señal de `sclr`

En el caso del FIFO de RX, se conecta además la señal `usedw` al registro de salida paralelo para poder controlar la cantidad de bytes transferidos al registro.

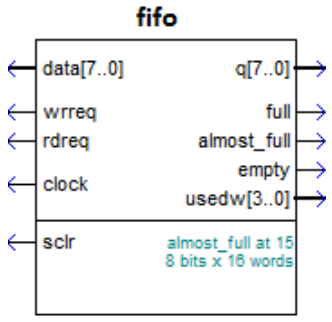

Figure 2. Memoria FIFO implementada del Core IP de Altera

B. Registros de corrimiento

Para la implementación de la operación de corrimiento, se utilizaron los operadores << y >>, los cuales resultan en la interpretación de registros de corrimiento por parte de la herramienta de análisis. Ambos registros de corrimiento son controlados por la señal de `load`, proveniente de la máquina de estados, para indicar el ingreso de un dato y comenzar con el corrimiento. Correspondiente al diseño, la operación de corrimiento se ejecuta a la frecuencia de la señal `SPI_CLK`. La figura 3 muestra el bloque del registro de corrimiento implementado para la transmisión.

C. Máquina de Estados Finita

Tal como se menciona en la sección de diseño de arquitectura, la máquina de estados que controla las operaciones de recepción y trasmisión SPI, se implementó con diez estados

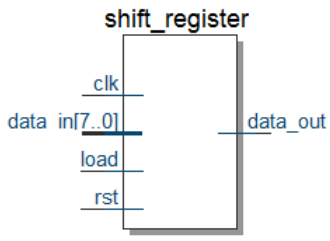

Figure 3. Registro de corrimiento de transmisión

mas un estado `idle`. La elaboración de este módulo se inició con ayuda de la herramienta de generación de máquinas de estado finitas de Quartus II. Debido a la complejidad de este módulo, se prefirió desarrollar el código RTL de manera manual para los diez estados, así como el comportamiento de las señales de salida.

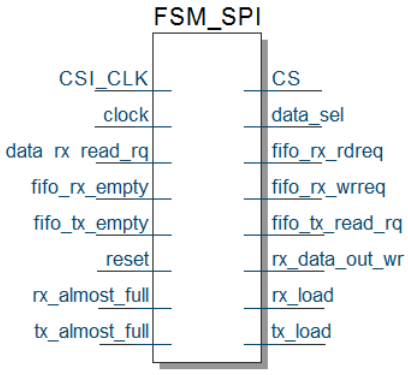

Figure 4. Máquina de Estados Finita

D. Registro de salida paralelo

Este módulo se implementó utilizando una estructura de control con *case* que selecciona entre los 120 bits del registro de salida `BUS_OUT`, mediante la señal `usedw` proveniente del FIFO de recepción. Esta señal indica la cantidad de registros llenados en la FIFO, lo cual es conveniente para determinar la dirección correspondiente en el registro de salida.

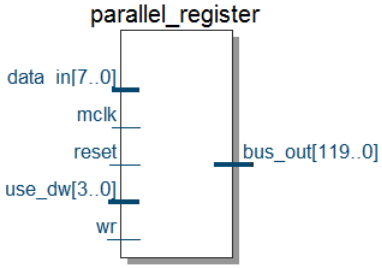

Figure 5. Registro de salida paralelo

V. Resultados

A. Consumo de recursos

La tabla I muestra los resultados de utilización de celdas lógicas combinacionales y de registros, así como la estimación

Table I
Consumo de celdas lógicas por módulo funcional

Módulo	LC Combinational		LC Registers	
	Est.	Impl.	Est.	Impl.
FSM_SPI	40	49	20	11
FIFO TX		26	15	19
FIFO RX		28	15	15
Parallel Reg		16	120	120
Pre_TX_module		1	2	2
Shift_Reg		8	8	9
Shift_Reg_RX		1	8	8
clk_div_mod		5	1	5

realizada al comienzo de la ejecución del diseño. La máquina de estados finita es la que domina en el uso de recursos combinacionales.

B. Simulaciones

1) Recepción de datos: Para demostrar la operación de recepción de datos SPI, se ejecutó la prueba diseñada en el plan de pruebas del documento de especificaciones del apéndice A, la cual describe el proceso para comenzar una recepción de datos. Este proceso inicia con la activación de la señal `Read_RQ`. La figura 6 muestra el comportamiento esperado, donde inmediatamente después de la presencia de este pulso, se envía la dirección a traves de `SPI_MOSI`, seguido por la recepción de 15 bytes por `SPI_MISO`. La transmisión se observa en sincronía con `SPI_clk` y además, `SPI_CS` se mantiene en bajo durante este período de tiempo. Todas las señales, a excepción de las entradas `SPI_MISO` y `Read_RQ`, son controladas por la lógica de control, gobernada por la máquina de estados finita. Inmediatamente después de

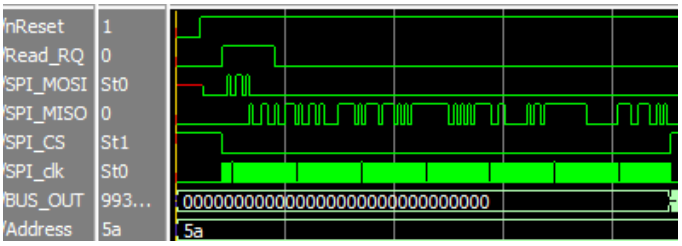

Figure 6. Operación de recepción por SPI

recibir los 15 bytes, se comienzan a transferir desde el FIFO de recepción, hasta el registro de 120 bits con salida `BUS_OUT`, tal como se muestra en la figura 7. Se comprobó que el valor final de este registro de salida corresponde a los 15 bytes enviados por el testbench a través de `SPI_MISO`.

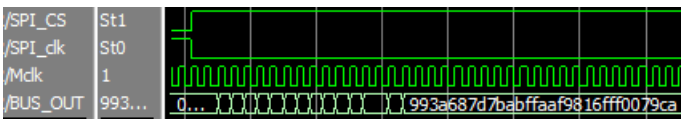

Figure 7. Transferencia de datos recibidos hacia registro de salida paralelo

2) Transmisión de datos: Una vez que la entrada `Data_Available` cambia de 1 a 0, se sabe que el proceso de transmisión de datos a dado inicio. Para poder verificar la funcionalidad del proceso de transmisión, se diseñó un testbench tal que mientras se está en el proceso de recepción de datos, el FIFO para la transmisión inicia el proceso de recepción de datos a ser transmitidos. En la figura siguiente se puede observar cuando esto sucede:

Como se puede apreciar, la señal `SPI_MISO` está recibiendo

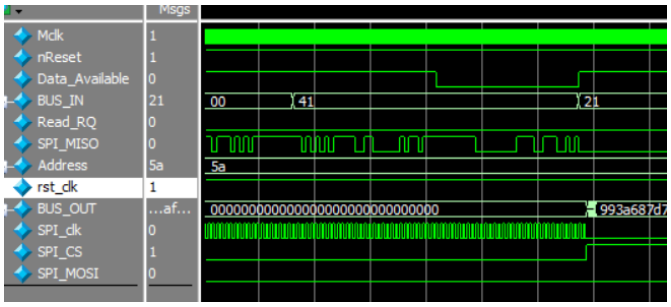

Figure 8. Inicio de transmisión de datos mientras se está en el proceso de recepción

datos, y en ese momento la señal de `Data_Available` pase a 0 para indicar que el byte 0x41 ha llegado por medio de la entrada `BUS_IN`. Una vez que el dato 0x41 es recibido, se puede observar como llega un nuevo dato, el cual es el 0x21. Una vez iniciada la recepción de los datos a transmitir, se van a recibir un total de 15 bytes, los cuáles se irán acumulando en el FIFO de transmisión de datos. Cuando el FIFO obtiene estos 15 bytes, la bandera `almost_full` se enciende para indicar a la máquina de estados que estamos listos para enviar el dato completo. Sin embargo, inmediatamente después otro dato ya viene en camino para ser transmitido, y por esta razón el dato a transmitir debe enviarse lo más rapido posible para así poder recibir el siguiente dato y que el mismo sea transmitido sin perder ningún byte.

En la siguiente figura se puede apreciar donde el primer dato a transmitir es enviado, y en paralelo el FIFO está recibiendo los bytes del siguiente dato a transmitir:

Como se puede observar, mientras el `SPI_MOSI` esta

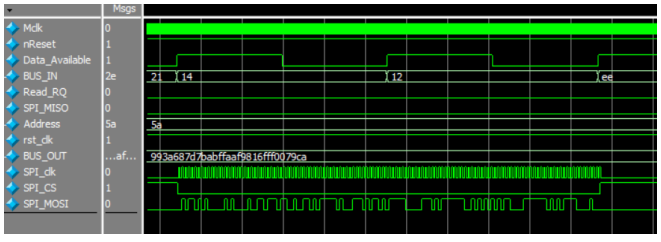

Figure 9. Transmisión de datos una vez recibidos los 15 bytes

haciendo la transmision de los 15 bytes recibidos, en el `BUS_IN` se siguen recibiendo bytes que estan asociados al siguiente dato a transmitir. Además, en la figura que se muestra a continuación, se puede ver que el primer byte enviado corresponde a la dirección 0x5a, y el siguiente corresponde al primer byte que fue recibido en el FIFO TX, el cual como se observó en la figura es 0x41, y seguidamente se observa que el siguiente byte corresponde al valor 0x21. Además en la figura 11 se muestra los 15 bytes enviados.

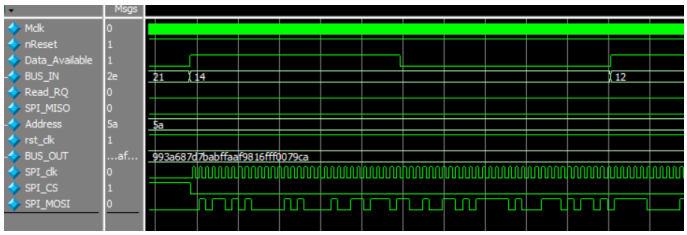

Figure 10. Transmisión del address y primeros 3 bytes

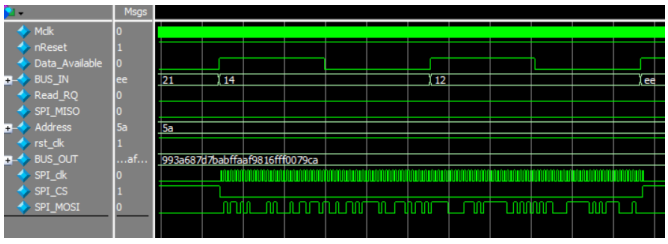

Figure 11. Transmisión completa del primer dato

Finalmente, los siguientes 15 bytes del segundo dato son recibidos en el FIFo TX, y transmitidos a través del `SPI_MOSI`, de manera que la siguiente figura muestra como los 2 datos son enviados. De esta manera, se verificó que el diseño elaborado es capaz de recibir datos para ser transmitidos mientras se están recibiendo datos, y además se enviaron 2 datos seguidos para demostrar que el FIFO TX es capaz de recibir un byte mientras los otros 15 bytes están siendo enviados.

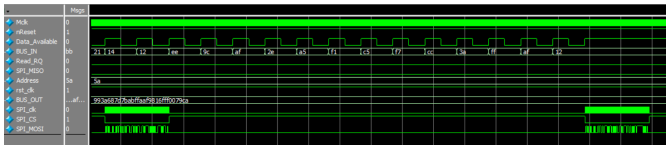

Figure 12. Transmisión completa del primer y segundo dato

VI. Cambios para implementación en ASIC

En caso de que este mismo código tuviera que implementarse en un ASIC se tendrían que hacer ciertos cambios para que sea compatible. Uno de los cambios es el manejo del reloj, en el caso de la FPGA se asume que el pin de `MClk` es de 25 MHz y además se tiene un divisor de frecuencia que tiene como salida un reloj `SPI_clk` de 3,125 MHz. Para el caso de este proyecto el reloj se generó en el testbench. En un ASIC se tiene que conectar el reloj a algún PLL y lógica de reloj que exista en alguna parte del chip. El divisor de reloj para el `SPI_clk` tendría que ser deshabilitado para el código ASIC y se usa como un puerto de entrada de reloj. El divisor estaría en

alguna lógica de reloj aparte del código del SPI Master. Los IP cores que se utilizaron en el diseño en FPGA probablemente no se tengan en el diseño del ASIC. Podría ser que haya que desarrollar módulos de FIFO manualmente o que se use otro tipo de IPs. Para esto se tendrían defines en el código que, dependiendo del tipo de compilación, cambie entre los IP cores de la FPGA y los módulos del ASIC. Muchas veces en diseño de ASICs se quiere total control sobre el diseño y la lógica dentro de los módulos muchas veces se programa a un nivel todavía más bajo. En vez de programar a un nivel de comportamiento, muchas veces se tienen macros que definen qué es un registro y se instancian los registros directamente en el código RTL en vez de utilizar reg. Si se crearan los módulos de esta manera, entonces se tendrían defines para cambiar entre el código por comportamiento para la FPGA y el código del ASIC con registros instanciados manualmente.

VII. Conclusiones

Se logró contruir un bloque SPI maestro que cumpliera el comportamiento descrito en los requisitos. La arquitectura propuesta inicialmente se implementó de manera bastante fiel, sólo hubo unos pocos cambios con respecto a algunas pocas señales, pero los bloques se mantuvieron iguales. El único bloque que no se utilizó fue el de lógica de recepción, pues las señales de entrada a la etapa de transmisión, se pudieron utilizar íntegras a como entran. La estimación de recursos con respecto a la cantidad de registros y bloques de memoria estuvo bastante acertada, la mayoría de los módulos cumplieron con la cantidad esperada de recursos.