Módulo SPI Master

Trabajo Práctico Final

Circuitos Lógicos Programables

Carrera de Especialización en Sistemas Embebidos – FIUBA

Lucas Sebastián Kirschner

(kirschnerlucas1@gmail.com)

14/10/2025

Índice de contenido

1. Introducción

El presente trabajo tiene como objetivo el desarrollo, simulación e implementación de un módulo SPI Master en lenguaje VHDL, destinado a su ejecución sobre una plataforma FPGA. El proyecto forma parte del trabajo práctico final de la asignatura Circuitos Lógicos Programables, perteneciente a la Carrera de Especialización en Sistemas Embebidos.

La elección del módulo SPI (Serial Peripheral Interface) responde a su amplia utilización en sistemas embebidos para la comunicación síncrona entre dispositivos digitales, tales como microcontroladores, conversores analógico-digitales, memorias y periféricos de propósito general. En particular, dentro del proyecto final de la especialización, se desarrolla una placa industrial que incorpora controladores de entradas y salidas y se comunica mediante este protocolo con un dispositivo maestro. Si bien en dicho proyecto no se emplea una FPGA, la implementación a nivel de lógica programable permite profundizar en los aspectos internos de temporización, control y sincronización del protocolo, favoreciendo una comprensión más completa de su funcionamiento.

El módulo desarrollado implementa un maestro SPI con control automático de Chip Select (CS), soporte para los cuatro modos estándar de operación (CPOL/CPHA), ancho de palabra configurable en 8 o 16 bits, selección del orden de transmisión de bits (MSB-first o LSB-first), y la posibilidad de ajustar la frecuencia del reloj de comunicación (SCLK) en función de la frecuencia del reloj base del sistema.

En la Figura 1 se observa el diagrama general del módulo implementado. A la izquierda se disponen los puertos de control, correspondientes a las señales de reloj, reinicio, transmisión y recepción de datos. Estas señales son manejadas por un microcontrolador ubicado en un nivel jerárquico superior, el cual determina los datos a transmitir y procesa los datos recibidos desde el esclavo SPI. A la derecha se encuentran los puertos propios del bus SPI, compuestos por las líneas de reloj (spi_clk_o), datos de salida (spi_mosi_o), datos de entrada (spi_miso_i) y selección de esclavo (spi_cs_o).

El desarrollo completo fue realizado en VHDL, utilizando el entorno Vivado para las etapas de síntesis, implementación y depuración mediante herramientas integradas como VIO (Virtual Input/Output) e ILA (Integrated Logic Analyzer). La simulación temporal permitió validar la correcta generación de las señales de reloj, sincronización y transferencia de datos, mientras que la implementación sobre el kit de desarrollo confirmó el correcto funcionamiento del diseño en hardware.

Figura 1. Diagrama en bloques del sistema.

2. Descripción del diseño y arquitectura

El diseño se organiza jerárquicamente en dos niveles principales (Figura 2):

1. spi_top (capa de integración/encapsulamiento).
2. spi_core (núcleo funcional), que agrupa cuatro submódulos especializados: clk_gen, mosi_transfer, miso_capture y cs_ctrl.

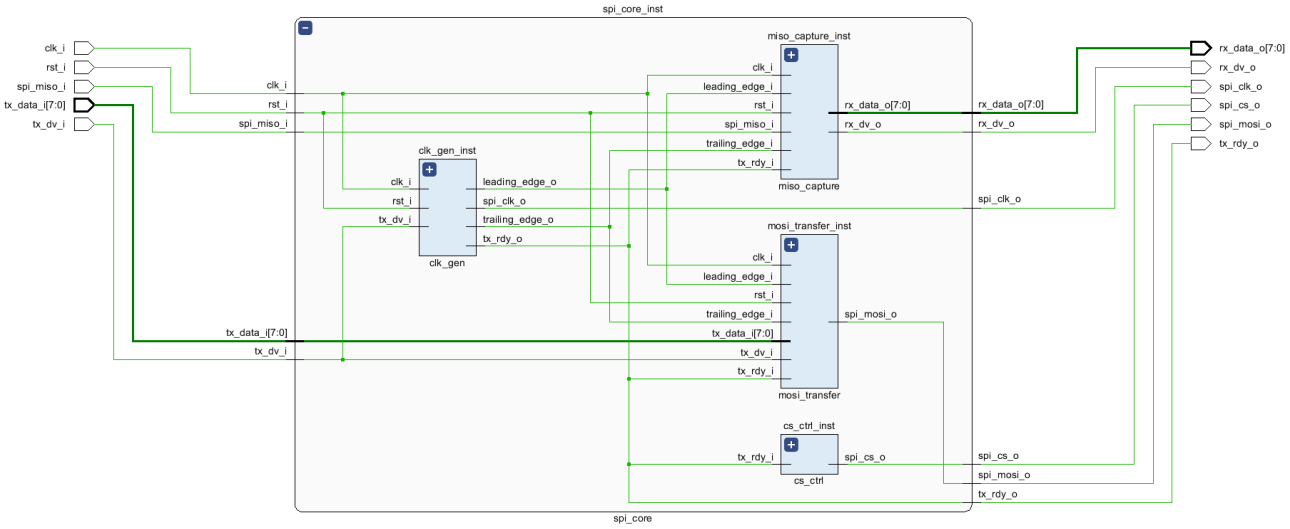

Figura 2. Diagrama lógico/estructural del sistema.

A continuación, se detalla la funcionalidad de cada bloque.

2.1. spi_top (capa de integración)

- Expone la interfaz externa del módulo (puertos de control y señales SPI).
- Define la parametrización del diseño (tamaño de palabra 8/16 bits, modo SPI 0–3,

orden de bits MSB/LSB, frecuencia base del sistema y objetivo de SCK).
- Instancia y conecta el núcleo spi_core, manteniendo al usuario abstraído de la lógica interna.

2.2. spi_core (núcleo funcional)

- Orquesta la secuencia completa de transmisión/recepción y el tiempo de cada etapa.
- Interconecta los submódulos dedicados a reloj, envío, captura y control de chip select, asegurando consistencia de temporización para cualquier combinación de CPOL/CPHA, tamaño de palabra y orden de bits.

2.2.1. clk_gen (generación y temporización de SCK)

- Genera el reloj SPI (spi_clk_o) a partir de clk_i, cumpliendo la frecuencia objetivo configurada.
- Implementa la polaridad (CPOL) y produce los flancos de referencia (leading/trailing) usados por los módulos de mosi_transfer y miso_capture según la fase (CPHA).
- Controla el ciclo de vida de una transferencia contabilizando flancos hasta completar la palabra (8/16 bits).
- Gestiona el handshake de inicio/ocupado/listo mediante tx_dv_i y tx_rdy_o, garantizando que el núcleo solo acepte un nuevo dato cuando ha finalizado la transacción en curso.

2.2.2. mosi_transfer (emisión por MOSI)

- Toma una muestra del dato de entrada cuando se solicita transmisión y lo congela internamente para toda la duración del frame.
- Presenta cada bit en spi_mosi_o en el instante exacto indicado por los flancos de clk_gen, respetando el orden de bits configurado (MSB-first o LSB-first).
- Se asegura de que el primer bit esté disponible conforme al modo CPHA.

2.2.3. miso_capture (captura por MISO)

- Muestrea la línea spi_miso_i en los flancos adecuados de SCK, definidos por CPHA, y reconstruye el vector recibido bit a bit en el orden configurado.
- Al completar la palabra, entrega el dato en paralelo por rx_data_o y emite un pulso de dato válido (rx_dv_o) para notificar al nivel superior.
- Su operación se sincroniza con clk_gen para mantener alineación temporal con la transmisión.

2.2.4. cs_ctrl (control automático de CS)

- Controla spi_cs_o de forma automática: activa (nivel bajo) durante toda la transferencia y desactiva al finalizar.

- Se coordina con el estado de ocupación/listo del núcleo para evitar particionar tramas y mantener la integridad de cada palabra.

3. Simulación temporal del sistema

3.1. Objetivo y banco de pruebas

El objetivo de la simulación es validar temporal y funcionalmente la transmisión y recepción de una palabra SPI en el modo 0 (CPOL=0, CPHA=0), con orden MSB-first y longitud de palabra de 8 bits, verificando además la gestión automática de CS, la sincronización de MOSI/MISO respecto de SCK y la señalización de handshake (tx_rdy_o, rx_dv_o).

El testbench instancia el bloque spi_top y provee los estímulos mínimos:

- Frecuencia de reloj del sistema (clk_i): 125 MHz → Período = 8 ns.
- Frecuencia objetivo de SCK (spi_clk_o): 12,5 MHz → Período = 80 ns.
- Parámetros de operación: MODE_0 (CPOL=0, CPHA=0), FIRSTBIT_MSB, DATA_SIZE=8.
- Reset: activo en bajo durante los primeros 20 ns; luego desactivado.
- Transmisión: al detectar tx_rdy_o='1', el testbench coloca tx_data_i = x"A5" y emite un pulso de tx_dv_i de 1 ciclo de clk_i.
- Respuesta del “esclavo”: genera por spi_miso_i el byte x"55", actualizando el bit en cada flanco descendente de spi_clk_o mientras spi_cs_o está activo. Este comportamiento es coherente con CPHA=0 (el maestro muestra en flanco ascendente y el esclavo actualiza en flanco descendente).

3.2. Simulación del sistema

La Figura 3 muestra la traza temporal de la transacción completa:

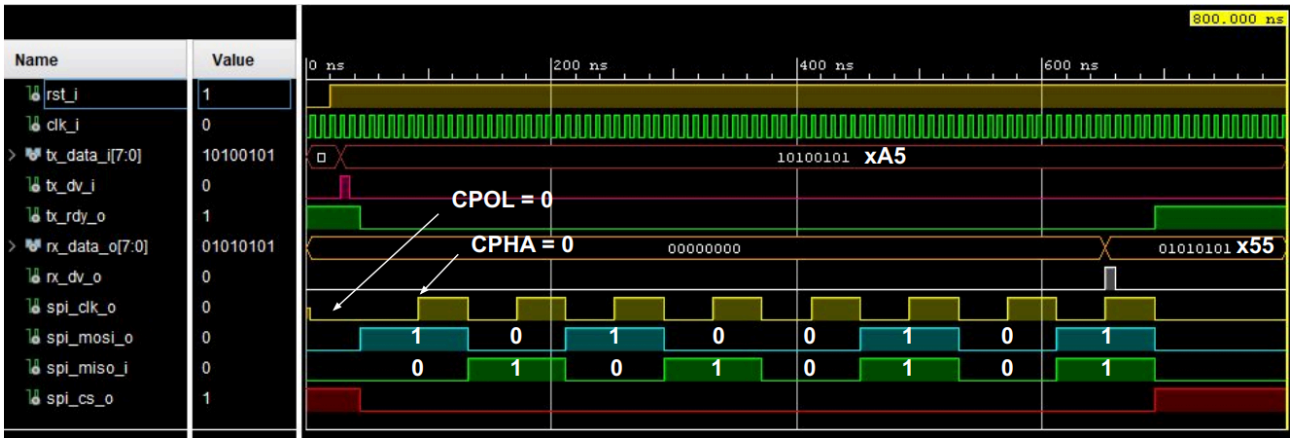

Figura 3. Simulación de comunicación SPI.

3.2.1. Estado inicial y reset

1. spi_clk_o en nivel bajo (CPOL=0) y spi_cs_o en alto (inactivo).
2. Tras 20 ns, se libera el reset y el núcleo queda listo (tx_rdy_o='1').

3.2.2. Inicio de transmisión

1. El testbench presenta tx_data_i = xA5 (1010_0101) y pulsa tx_dv_i.
2. El módulo toma el dato, pone en bajo tx_rdy_o (ocupado) y activa spi_cs_o en bajo para iniciar el frame.
3. Con CPHA=0, el primer bit (MSB) de MOSI se coloca antes del primer flanco ascendente de SCK.

3.2.3. Ráfaga de SCK y alineación de datos

1. Se genera una ráfaga de 8 periodos de spi_clk_o a 12,5 MHz (≈ 80 ns cada uno).
2. MOSI (spi_mosi_o): presenta los bits de xA5 en MSB-first, con estabilidad antes de cada flanco ascendente (instante de muestreo del esclavo para CPHA=0).
3. MISO (spi_miso_i): el modelo de esclavo actualiza x55 en cada flanco descendente, de modo que el maestro lo muestra en el ascendente siguiente (cumpliendo CPHA=0).
4. La sincronización que se observa en la figura (bit válido en MOSI al ascendente y actualización de MISO al descendente) es consistente con la matriz CPOL/CPHA.

3.2.4. Fin de palabra y notificaciones

Al completarse los 16 flancos (8 ciclos) programados por clk_gen, el núcleo:

1. Desactiva la ráfaga de SCK y retorna a idle bajo.
2. Desactiva CS (alto).
3. Asserta rx_dv_o durante un ciclo para indicar dato recibido válido.
4. Carga rx_data_o = x55 y reaserta tx_rdy_o='1', quedando listo para una nueva transacción

3.3. Análisis de resultados

1. Temporización: la figura muestra una relación 10:1 entre clk_i (8 ns) y spi_clk_o (80 ns), coherente con el objetivo de 12,5 MHz para SCK.
2. Modo 0 (CPOL=0, CPHA=0):
3. SCK idle en bajo y primer flanco ascendente como flanco de muestreo.
4. MOSI estable antes del ascendente; MISO actualizado en el descendente por el esclavo.
5. Integridad del frame: spi_cs_o permanece bajo durante toda la palabra y retorna alto al finalizar, sin fragmentación de trama.
6. Señales de control: tx_rdy_o se mantiene bajo durante la transferencia (ocupado) y vuelve a alto al finalizar; rx_dv_o genera un pulso indicando dato válido en

rx_data_o.

7. Datos: coincidencia entre MOSI = xA5 y MISO = x55, con rx_data_o = x55 al cierre del frame.

4. Implementación

4.1. Plataforma y entorno de ejecución

La implementación del diseño se llevó a cabo en el servidor remoto de la cátedra, que dispone de una placa FPGA Xilinx con soporte para el entorno Vivado Design Suite.

El proceso consistió en la síntesis, implementación y generación del bitstream del módulo spi_top_vio_ila, el cual integra el bloque funcional spi_top junto con los núcleos IP de depuración VIO (Virtual Input/Output) e ILA (Integrated Logic Analyzer).

El uso de estas herramientas de instrumentación permite interactuar con el diseño en tiempo real sin necesidad de hardware externo, posibilitando tanto la inyección de estímulos como la observación directa de señales internas desde el propio entorno de Vivado.

4.2. Estructura de la implementación

La Figura 4 muestra el esquema funcional del diseño implementado dentro del FPGA.

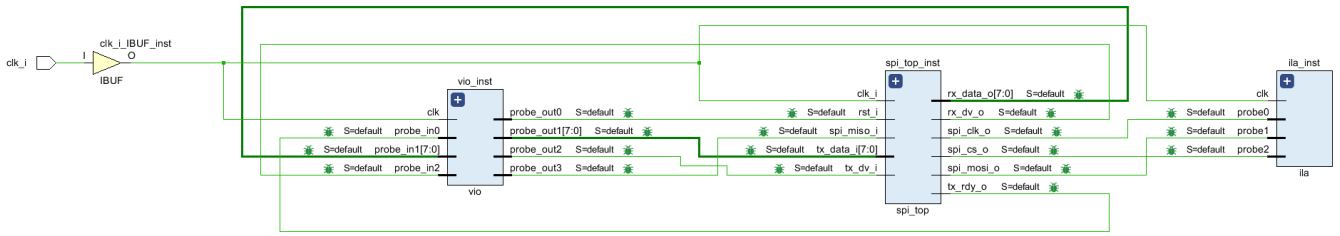

Figura 4. Esquema implementado en la FPGA.

Como se observa en la Figura 4, el archivo spi_top_vio_ila.vhd encapsula tres componentes principales, los cuales se describen a continuación.

4.2.1. Módulo spi_top

1. Contiene la lógica del maestro SPI descrita en VHDL.
2. Gestiona la comunicación completa: generación del reloj (spi_clk_o), transmisión (spi_mosi_o), recepción (spi_miso_i) y control automático del chip select (spi_cs_o).

4.2.2. Módulo VIO (Virtual Input/Output)

1. Permite simular entradas y monitorear salidas del diseño desde el entorno Vivado, a través de una interfaz gráfica.

2. Se utilizó para generar estímulos de prueba, tales como:
    a. rst_i (reset del sistema).
    b. tx_data_i (dato a transmitir, configurado como vector de 8 bits).
    c. tx_dv_i (pulso de dato válido).
    d. spi_miso_i (dato de entrada simulado para la línea MISO).
3. También se configuró para visualizar señales de salida relevantes:
    a. tx_rdy_o (estado del transmisor).
    b. rx_data_o (dato recibido).
    c. rx_dv_o (indicación de dato válido recibido).

4.2.3. Módulo ILA (Integrated Logic Analyzer)

1. Se empleó para capturar en tiempo real las señales de alta velocidad asociadas al bus SPI.
2. En particular, se conectaron las líneas:
    a. spi_clk_o (reloj SPI).
    b. spi_mosi_o (datos enviados).
    c. spi_cs_o (selección del esclavo).
3. Estas señales fueron muestreadas internamente y visualizadas mediante la herramienta de análisis de señales integrada en Vivado.

4.3. Ejecución y verificación en hardware

Una vez descargado el bitstream en la FPGA, se procedió a ejecutar pruebas en tiempo real desde Vivado.

A través del panel VIO, se configuró el valor de transmisión tx_data_i = x"A5" y se activó el pulso tx_dv_i. El módulo spi_top generó automáticamente el tren de pulsos correspondiente al reloj SPI, envió los bits por MOSI y controló la línea CS según la temporización establecida.

La Figura 5 muestra la captura obtenida con el ILA durante la transmisión del byte xA5 (b10100101), donde puede observarse la correcta correspondencia entre el reloj (spi_clk_o), los datos transmitidos (spi_mosi_o) y la señal de selección de esclavo (spi_cs_o).

En esta captura se aprecia que:

1. spi_mosi_o presenta los bits del dato xA5 en orden MSB-first, actualizándose en los flancos adecuados según el modo SPI configurado (CPOL=0, CPHA=0).
2. spi_cs_o se mantiene activo (bajo) durante toda la transferencia y retorna a alto una vez finalizada la palabra, indicando la finalización del frame SPI.

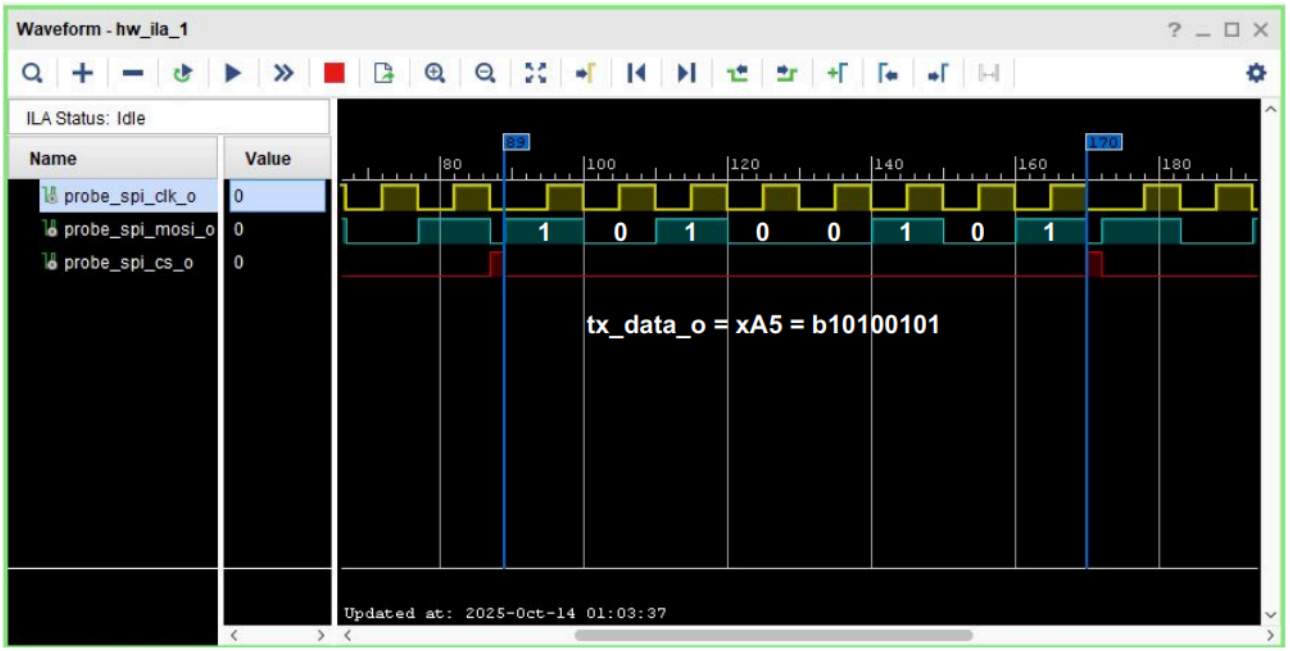

Figura 5. Señales de salidas observadas en módulo ILA.

5. Utilización de recursos

La Figura 6 presenta la tabla de utilización de recursos obtenidos luego de la implementación completa del diseño en la FPGA. Se incluyen los valores de Look-Up Tables (LUTs), Flip-Flops (FFs), puertos de entrada/salida (IO) y buffers globales de reloj (BUFG), comparando el número de elementos utilizados respecto del total disponible en el dispositivo.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	32	17600	0.18
FF	47	35200	0.13
IO	25	100	25.00
BUFG	1	32	3.13

Figura 6. Utilización de recursos Post-Implementation.

Los resultados muestran que el diseño presenta una baja demanda lógica, utilizando menos del 0.2 % de los recursos combinacionales (LUT) y secuenciales (FF) disponibles

en el dispositivo.

La utilización del 25 % de los puertos IO responde a la cantidad de señales expuestas externamente —correspondientes a las líneas del bus SPI y las señales de control utilizadas por los núcleos VIO e ILA—, mientras que la utilización de un único BUFG corresponde al buffer global de reloj empleado para distribuir la señal clk_i dentro del FPGA, garantizando baja latencia y uniformidad en la distribución del reloj.

6. Conclusiones

El desarrollo presentado cumplió satisfactoriamente con los objetivos propuestos para el trabajo práctico final, que consistían en diseñar, simular e implementar un módulo SPI Master en lenguaje VHDL, verificando su funcionamiento tanto en simulación como en hardware real mediante herramientas de instrumentación integradas en la FPGA.

La simulación funcional confirmó que el módulo cumple plenamente con la especificación establecida para los parámetros de operación seleccionados —DATA_SIZE = 8 bits, MSB-first y MODE 0 (CPOL=0, CPHA=0)—. Los resultados obtenidos demostraron el correcto cumplimiento de la temporalidad del protocolo SPI y la coherencia entre las señales CS, SCK, MOSI y MISO. Asimismo, se verificó el ajuste de frecuencia de SCK a partir del reloj de sistema clk_i, la entrega puntual de rx_dv_o, y la secuencia de ocupado/listo controlada mediante tx_rdy_o, garantizando una operación determinística y completamente encapsulada hacia el nivel superior.

La implementación en hardware permitió confirmar el comportamiento esperado del módulo bajo condiciones reales de ejecución, observándose en el Integrated Logic Analyzer (ILA) una correspondencia exacta entre los tiempos de conmutación y sincronización medidos en la FPGA y los obtenidos previamente en simulación.

El uso combinado de los núcleos VIO e ILA resultó fundamental para la validación del diseño, ya que posibilitó la interacción directa con las señales internas del módulo y la verificación en tiempo real de su funcionamiento, sin necesidad de instrumentación externa. Esta metodología facilitó el proceso de depuración y ajuste, optimizando el ciclo de desarrollo sobre la plataforma remota.

En cuanto a la utilización de recursos, los resultados de síntesis e implementación evidencian una baja ocupación de área lógica, con un uso inferior al 0,2 % de LUTs y FFs disponibles en el dispositivo. Este comportamiento confirma que el módulo SPI Master es eficiente y escalable, pudiendo integrarse sin inconvenientes dentro de sistemas digitales más complejos o emplearse en múltiples instancias paralelas.

En síntesis, el trabajo permitió aplicar de manera integral los conocimientos adquiridos sobre diseño digital, descripción en VHDL, simulación temporal e instrumentación de

hardware, consolidando la comprensión del protocolo SPI desde una perspectiva de implementación física en FPGA, y cumpliendo con todos los requerimientos planteados para el trabajo final de la asignatura Circuitos Lógicos Programables.