Módulo SPI Master

Trabajo Práctico Final

Microarquitecturas y Softcores

Carrera de Especialización en Sistemas Embebidos – FIUBA

Lucas Sebastián Kirschner

(kirschnerlucas1@gmail.com)

9/12/2025

Índice de contenido

1. Introducción

El presente trabajo tiene como objetivo el diseño, implementación y validación de un módulo SPI Master desarrollado como IP Core en VHDL, integrado dentro del entorno de procesamiento heterogéneo del dispositivo Xilinx Zynq-7000, compuesto por la Processing System (PS) basada en un microprocesador ARM Cortex-A9 y la Programmable Logic (PL) basada en tecnología FPGA. Este desarrollo se enmarca en el Trabajo Práctico Final de la asignatura Microarquitecturas y Softcores, perteneciente a la Carrera de Especialización en Sistemas Embebidos, y persigue la integración completa de un periférico personalizado en una arquitectura SoC mediante un bus estándar AXI4-Lite, complementado con una aplicación en lenguaje C ejecutada en la PS.

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 implementado mantiene la estructura funcional previamente desarrollada en la asignatura Circuitos Lógicos Programables, conservando sus capacidades internas, control automático de Chip Select (CS), soporte para los cuatro modos estándar de operación (CPOL/CPHA), configuración del ancho de palabra (8 o 16 bits), selección del orden de bits (MSB-first o LSB-first) y ajuste de frecuencia de SCK.

En la Figura 1 se presenta el diagrama general del sistema implementado. A la izquierda se observa el microcontrolador Cortex-A9, que accede al módulo SPI a través del bus AXI4-Lite, mediante el cual escribe los datos a transmitir y lee los datos recibidos. Este bus también proporciona las señales de reloj y reinicio del dominio AXI (s_axi_aclk y s_axi_aresetn), asegurando la correcta sincronización entre el procesador y la lógica programable.

Las señales internas del módulo, como tx_data_i, tx_dv_i, tx_rdy_o, rx_data_o y rx_dv, representan la lógica abstracta de transmisión y recepción utilizada por el núcleo SPI, pero ahora son expuestas al procesador mediante registros AXI mapeados en memoria, permitiendo su manipulación directa desde software.

A la derecha del módulo se encuentran las líneas físicas del bus SPI: el reloj (spi_clk_o), el dato de salida (spi_mosi_o), el dato de entrada (spi_miso_i) y la selección del esclavo

(spi_cs_o). Estas señales son capturadas en tiempo real mediante un núcleo Integrated Logic Analyzer (ILA), que permite verificar su comportamiento temporal dentro de la FPGA sin necesidad de instrumentación externa.

Este esquema resume la interacción completa entre software y hardware: el procesador configura y opera el núcleo SPI mediante AXI, mientras que el ILA facilita la inspección detallada de las señales generadas por la lógica implementada. Esto permite validar tanto la integración PS–PL como el funcionamiento correcto del módulo SPI en condiciones reales de ejecución.

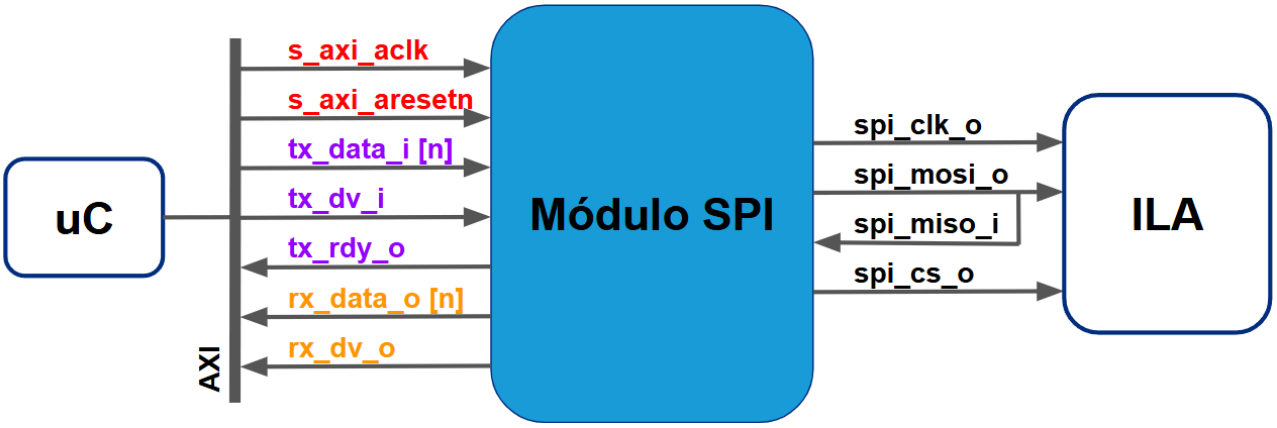

Figura 1. Diagrama lógico del sistema implementado.

2. Descripción del diseño y arquitectura

La Figura 2 presenta el diagrama en bloques del sistema implementado en el entorno Vivado. El diseño integra el procesador ARM Cortex-A9 del Zynq-7000 con el módulo SPI Master desarrollado en VHDL, permitiendo que el software ejecutado en la PS interactúe directamente con la lógica personalizada ubicada en la PL.

El procesador accede al módulo SPI a través de una interfaz AXI4-Lite, expuesta mediante el interconector AXI del diseño. Esta conexión permite que el núcleo sea controlado desde software mediante operaciones de lectura y escritura sobre registros mapeados en memoria, asignados automáticamente por el sistema. De esta manera, el PS puede iniciar transmisiones, cargar datos de salida, consultar el estado del periférico y leer los datos recibidos, manteniendo un esquema de comunicación simple y determinista.

El módulo SPI Master ocupa el rol central dentro del sistema, ya que implementa la lógica de transmisión síncrona y genera las señales físicas del protocolo: reloj (spi_clk_o), datos de salida (spi_mosi_o), datos de entrada (spi_miso_i) y chip select (spi_cs_o). Estas señales son dirigidas, en esta instancia, hacia un núcleo Integrated Logic Analyzer (ILA). El ILA permite capturar en tiempo real el comportamiento interno del módulo durante la operación, sin necesidad de disponer de hardware externo. Esto habilita la observación

directa de la actividad del bus SPI, así como la verificación de la temporización y secuencia de los bits transmitidos y recibidos.

El Block Design incluye además bloques auxiliares tales como el generador de reloj y el gestor de señales de reinicio, cuya función es únicamente proveer las condiciones de operación necesarias para la lógica PL. Estos bloques no forman parte del foco del presente informe, pero permiten garantizar que el módulo SPI y el ILA operen bajo un dominio de reloj estable y correctamente sincronizado.

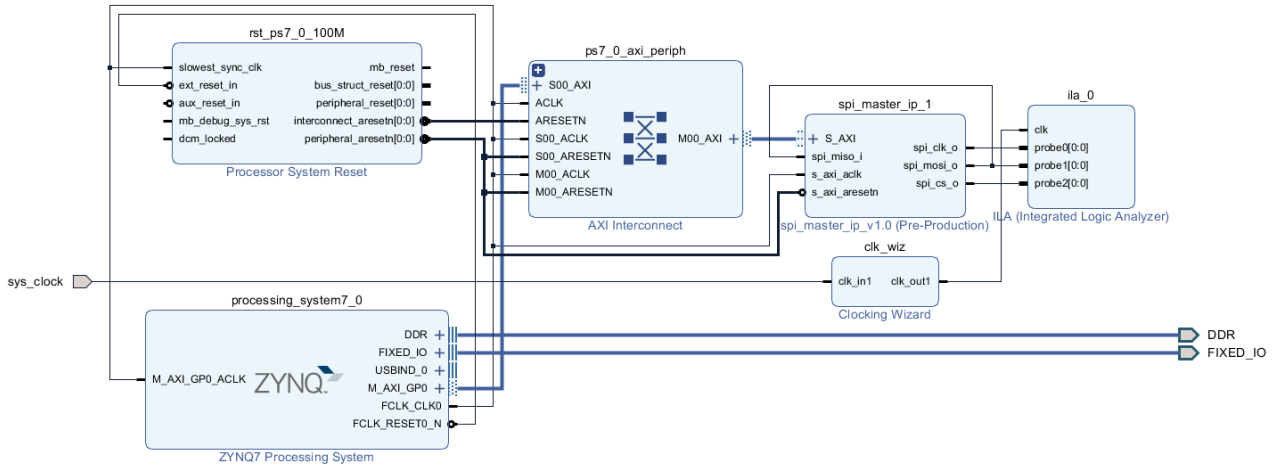

Figura 2. Diagrama lógico/estructural del sistema.

2.1. Descripción del IP SPI_MASTER

El núcleo desarrollado se entrega como un IP Core parametrizable denominado spi_master_ip_v1_0, empaquetado siguiendo el flujo estándar de Vivado para periféricos AXI4-Lite. El IP combina dos bloques principales:

- La entidad superior spi_master_ip_v1_0, que expone la interfaz AXI y los puertos físicos del bus SPI.
- El bloque interno spi_master_ip_v1_0_S_AXI, que implementa la lógica de esclavo AXI, el espacio de registros mapeados en memoria y la instanciación del núcleo funcional spi_top.

2.1.1. Parametrización y puertos externos

La entidad spi_master_ip_v1_0 define un conjunto de genéricos que permiten adaptar el comportamiento del módulo a distintos requerimientos de aplicación:

- DATA_SIZE (8 o 16 bits): longitud de la palabra SPI.
- MODE (0–3): modo de operación CPOL/CPHA.
- FIRST_BIT (MSB-first o LSB-first): orden de transmisión de bits.
- CLOCK_RATE_HZ: frecuencia del reloj de sistema asociado a la PL.

- SCK_TARGET_HZ: frecuencia objetivo del reloj de salida spi_clk_o.

Estos parámetros son configurables desde la pestaña Customization GUI del asistente de empaquetado de IP, lo que facilita su reutilización en distintos diseños sin modificar el código fuente.

En cuanto a los puertos, el IP expone:

- La interfaz AXI4-Lite esclavo (s_axi_aclk, s_axi_aresetn, señales de dirección, datos y control), utilizada por el procesador para acceder al periférico.
- Las líneas físicas del bus SPI: reloj (spi_clk_o), dato de salida (spi_mosi_o), dato de entrada (spi_miso_i) y selección de esclavo (spi_cs_o).

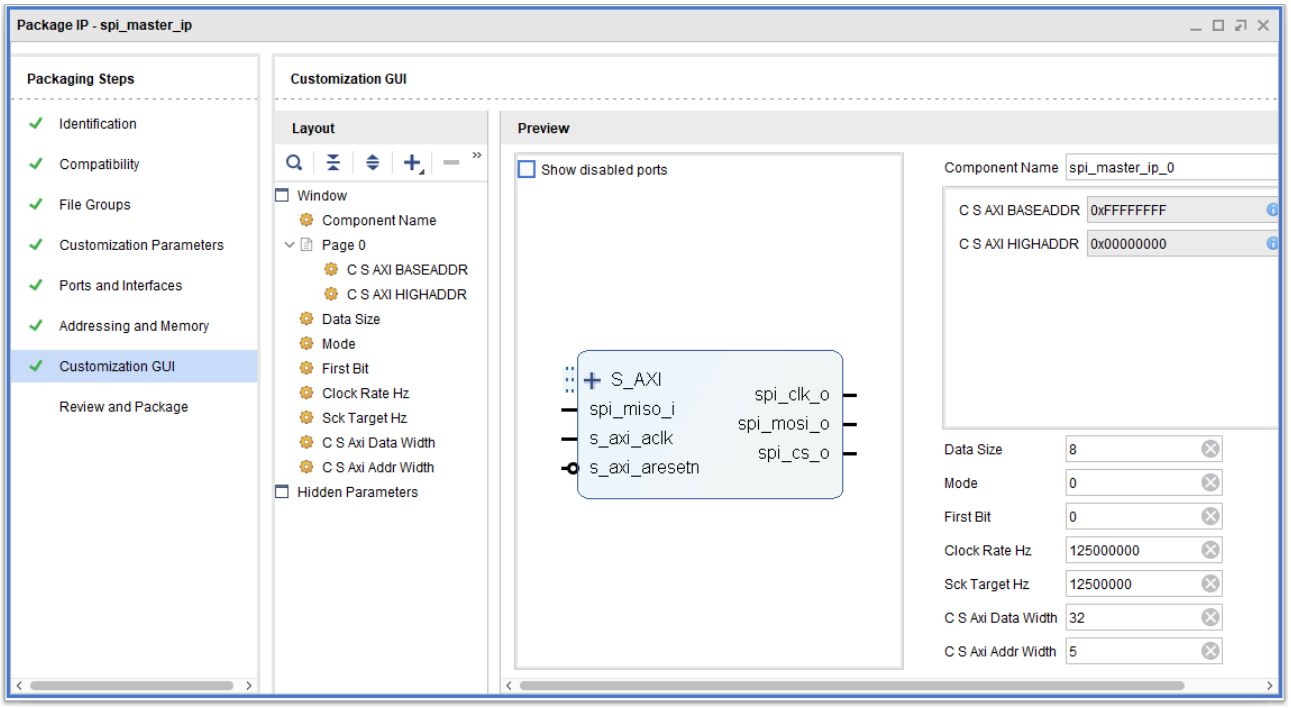

Figura 3. Parametrización y puertos externos del IP.

2.1.2. Jerarquía interna y núcleo funcional

Dentro de spi_master_ip_v1_0 se instancia el bloque spi_master_ip_v1_0_S_AXI, que concentra toda la lógica específica de AXI y la integración con el núcleo de comunicación serie. A su vez, este bloque contiene la instancia del módulo spi_top, desarrollado previamente en la asignatura Circuitos Lógicos Programables.

El módulo spi_top mantiene la misma arquitectura jerárquica que en el trabajo original:

- clk_gen: generación de spi_clk_o a partir del reloj de sistema y gestión de temporización según MODE.
- mosi_transfer: carga de la palabra de transmisión y envío secuencial de bits por

spi_mosi_o.

- miso_capture: muestreo de spi_miso_i y reconstrucción del dato recibido.
- cs_ctrl: activación y desactivación automática de spi_cs_o durante cada trama.

La entidad spi_top se conecta a spi_master_ip_v1_0_S_AXI mediante señales de alto nivel:

- Señales de control: clk_i (asociado a S_AXI_ACLK) y rst_i (derivado de S_AXI_ARESETN).
- Interfaz de transmisión MOSI: tx_data_i, tx_dv_i, tx_rdy_o.
- Interfaz de recepción MISO: rx_data_o, rx_dv_o.
- Interfaz SPI física: spi_clk_o, spi_mosi_o, spi_miso_i, spi_cs_o.

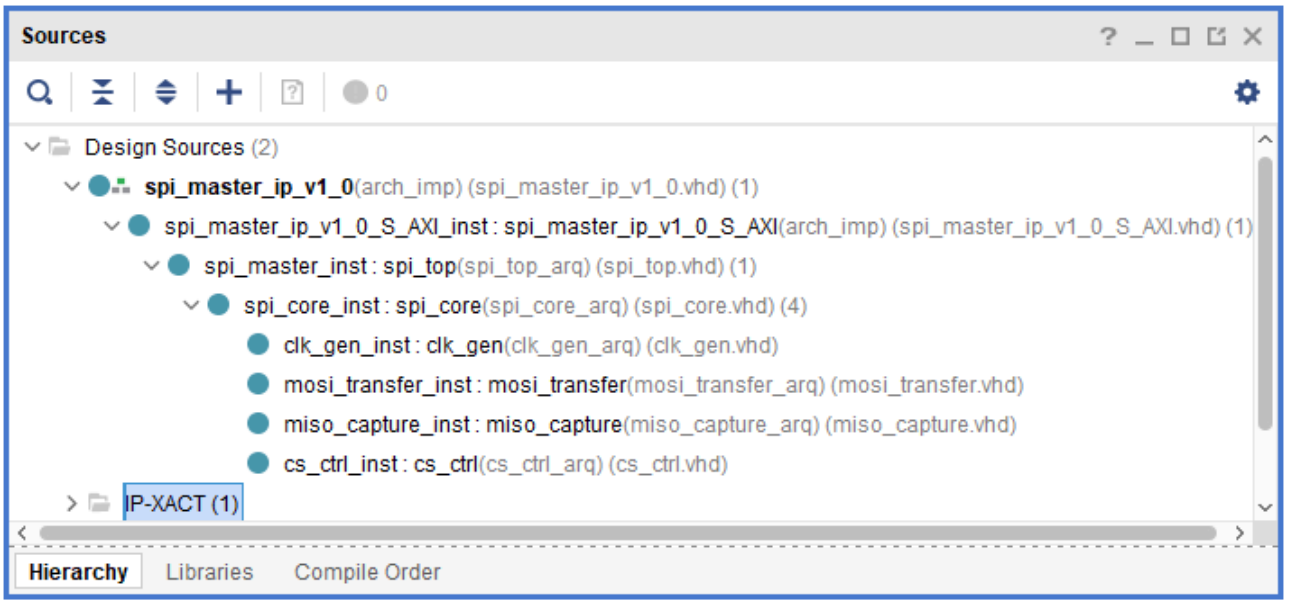

Figura 4. Jerarquía interna del IP.

2.1.3. Espacio de registros AXI y mapeo de señales

La lógica AXI del bloque spi_master_ip_v1_0_S_AXI se basa en el ejemplo de esclavo AXI4-Lite provisto por Vivado, extendido para adecuarse a las necesidades del módulo SPI. Se definen cinco registros esclavos de 32 bits (slv_reg0 — slv_reg4), accesibles desde el procesador mediante direcciones consecutivas:

- slv_reg0: registro de datos de transmisión. El procesador escribe en sus bits menos significativos la palabra a enviar por SPI (tx_data_i).
- slv_reg1: registro de control de envío. El bit 0 se utiliza como pulso de “dato válido” (tx_dv_i), indicando al núcleo SPI que debe iniciar una nueva transferencia con el valor cargado en slv_reg0.
- slv_reg2, slv_reg3 y slv_reg4: registros asociados al estado del núcleo y a los

datos recibidos.

Durante la integración se detectó un problema típico de señales con múltiples drivers (multi-driven nets). En el ejemplo original de Vivado, los registros esclavos pueden ser escritos por la lógica AXI y, a la vez, ser leídos sin restricciones. Sin embargo, en este diseño se requería que ciertas señales provinieron exclusivamente del núcleo spi_top (por ejemplo, tx_rdy_o, rx_data_o y rx_dv_o), pero al mismo tiempo estuvieran disponibles para lectura en el espacio de direcciones AXI.

Si se conectaran directamente estas salidas del núcleo a los registros slv_reg2, slv_reg3 y slv_reg4, coexistirían dos fuentes de conducción sobre las mismas señales: la lógica de escritura AXI y la lógica interna del módulo SPI, generando advertencias de síntesis y un comportamiento potencialmente indeterminado.

Para evitarlo, se adoptó la siguiente estrategia:

- Los registros slv_reg0 y slv_reg1 permanecen exclusivamente bajo control de la lógica AXI y sólo son escritos por el procesador.
- Para los registros asociados a salidas del SPI se introducen señales auxiliares: slv_reg2_aux, slv_reg3_aux y slv_reg4_aux.
- El núcleo spi_top escribe sus salidas en estas señales auxiliares:
    - tx_rdy_o → slv_reg2_aux(0)
    - rx_data_o → slv_reg3_aux(DATA_SIZE-1 downto 0)
    - rx_dv_o → slv_reg4_aux(0)
- El multiplexor de lectura AXI utiliza estas versiones auxiliares (*_aux) para construir el dato axi_rdata devuelto al procesador.

De esta forma, cada señal tiene un único driver: los registros slv_reg0 a slv_reg4 son exclusivamente escritos por AXI, mientras que las señales auxiliares son exclusivamente escritas por el núcleo SPI y sólo se utilizan en el camino de lectura. Esta separación elimina los conflictos de conducción, mantiene un comportamiento determinista y simplifica el análisis de síntesis e implementación.

```
440
441     -- Add user logic here
442     spi_master_inst: spi_top
443     generic map(
444       DATA_SIZE     => DATA_SIZE,        -- largo de la trama
445       MODE          => MODE,             -- modo SPI
446       FIRST_BIT     => FIRST_BIT,        -- orden de transferencia
447       CLOCK_RATE_HZ => CLOCK_RATE_HZ,    -- clk_i => 125 MHz
448       SCK_TARGET_HZ => SCK_TARGET_HZ    -- spi_clk_o => 12,5 Mbps
449     )
450
451     port map(
452       -- Senales de control
453       rst_i     =>  S_AXI_ARESETN,                        -- reset activo en bajo
454       clk_i     =>  S_AXI_ACLK,                           -- reloj del sistema
455
456       -- Senales MOSI
457       tx_data_i => slv_reg0(DATA_SIZE-1 downto 0),        -- dato de entrada a transmitir por spi_mosi_o
458       tx_dv_i   => slv_reg1(0),                           -- pulso: indica dato valido en tx_data_i
459       tx_rdy_o  => slv_reg2_aux(0),                       -- listo para aceptar un nuevo dato
460
461       -- Senales MISO
462       rx_data_o => slv_reg3_aux(DATA_SIZE-1 downto 0),    -- dato de salida recibido por spi_miso_i
463       rx_dv_o   => slv_reg4_aux(0),                       -- pulso: dato de entrada listo en rx_data_o
464
465       -- Interfaz SPI
466       spi_clk_o   =>  spi_clk_o,                          -- linea SCK
467       spi_mosi_o  =>  spi_mosi_o,                         -- linea MOSI (Master Out Slave In)
468       spi_miso_i  =>  spi_miso_i,                         -- linea MISO (Master In Slave Out)
469       spi_cs_o    =>  spi_cs_o                            -- linea CS (Chip Select)
470     );
471
472     -- User logic ends
```

Figura 5. Mapeo de genéricos y puertos en spi_master_ip_v1_0_S_AXI.vhd.

3. Software de prueba del sistema

Para validar el funcionamiento del IP spi_master_ip_v1_0 se desarrolló una aplicación bare-metal en C sobre el SDK de Xilinx, ejecutada en el procesador ARM Cortex-A9 del Zynq. El objetivo principal del software es ejercer el periférico SPI mediante accesos AXI4-Lite, generando tramas de transmisión periódicas y verificando la recepción de datos a través de un lazo de retorno (loopback.

La aplicación utiliza los encabezados generados automáticamente por Vivado (xparameters.h y spi_master_ip.h), que proporcionan la dirección base del periférico en el espacio AXI (SPI_BASEADDR) y las macros de acceso SPI_MASTER_IP_mWriteReg y SPI_MASTER_IP_mReadReg. A partir de estos elementos se definen constantes simbólicas para los offsets de los registros esclavos (REG0_TX_DATA a REG4_RX_DV), manteniendo una correspondencia directa con el mapa de registros descripto en la sección de hardware.

El programa principal (main) implementa un bucle infinito con la siguiente secuencia de operación:

1. Llama a spi_wait_tx_ready(), que realiza una espera ocupada sobre el bit 0 de slv_reg2. La función retorna únicamente cuando el núcleo SPI indica estar listo para aceptar una nueva trama (tx_rdy_o = '1').
2. Invoca spi_start_transfer(tx_byte), que escribe el byte a transmitir en slv_reg0 y genera un pulso de un ciclo sobre slv_reg1 (bit 0), actuando como señal de “dato válido” (tx_dv_i). Esto dispara internamente una transacción completa sobre el bus SPI.
3. Llama a spi_wait_and_read_rx(), que bloquea la ejecución hasta que el bit 0 de slv_reg4 indica la disponibilidad de un dato recibido (rx_dv_o = '1'). Una vez cumplida la condición, la función lee el valor de slv_reg3 y lo devuelve como byte recibido.
4. Finalmente, se imprime por UART el par de valores tx_byte / rx_byte mediante xil_printf, se introduce una pausa de 5 segundos (sleep(5)) y se incrementa el byte de prueba para la siguiente iteración.

Las tres funciones auxiliares encapsulan los detalles de acceso a los registros AXI, ofreciendo una interfaz de más alto nivel:

- spi_wait_tx_ready() se encarga de sincronizar el software con el estado interno del IP, evitando escribir una nueva trama mientras el núcleo se encuentra ocupado.
- spi_start_transfer() abstrae la secuencia correcta de escritura del dato y generación del pulso de tx_dv.
- spi_wait_and_read_rx() agrupa la espera por rx_dv y la lectura coherente del dato de recepción.

Este esquema de prueba, aunque sencillo, resulta suficiente para verificar la integridad extremo a extremo de la cadena de comunicación: desde el acceso AXI del procesador, pasando por el mapa de registros del IP, la lógica del núcleo spi_top y, finalmente, la observación de los resultados por UART. Además, la estructura modular del código permite su ampliación futura para incorporar mecanismos de verificación adicionales, como comparación automática de patrones, manejo de errores o pruebas con distintas configuraciones de modo SPI y tamaño de palabra.

4. Implementación

4.1. Plataforma y entorno de ejecución

La implementación y validación del diseño se llevaron a cabo en el servidor remoto provisto por la cátedra, equipado con una placa Xilinx Zynq-7000, la cual integra en un único dispositivo un procesador ARM Cortex-A9 (Processing System, PS) y una matriz de lógica programable (Programmable Logic, PL). El desarrollo se realizó utilizando el

entorno Vivado Design Suite, que permitió ejecutar las etapas de síntesis, implementación y generación del archivo de configuración del sistema.

El diseño final incorporó el núcleo spi_master_ip_v1_0 dentro de un Block Design que incluye, además, la interfaz AXI del Zynq, los bloques de gestión de reloj y reset generados automáticamente por Vivado y un núcleo Integrated Logic Analyzer (ILA) para inspección de señales internas. El uso del ILA resultó fundamental para verificar en tiempo real el funcionamiento de las señales spi_clk_o, spi_mosi_o y spi_cs_o, sin recurrir a instrumentación externa.

A su vez, la depuración del software ejecutado en el Cortex-A9 se realizó mediante la consola UART del sistema, accedida desde la herramienta minicom en el servidor remoto. Esto permitió observar el intercambio de datos entre el procesador y el IP mediante las funciones xil_printf, facilitando la verificación.

4.2. Ejecución de pruebas y verificación del sistema

La validación del sistema consistió en ejecutar una serie de pruebas que permitieron comprobar, de manera conjunta, el funcionamiento del IP SPI, la comunicación AXI y la aplicación en C desarrollada en el SDK. Las pruebas abarcaron dos niveles de observación:

4.2.1. Prueba funcional vía software (UART / minicom)

El software transmitió de forma continua un byte creciente mediante el registro slv_reg0, activando la transferencia con un pulso sobre slv_reg1.

Dado que en el Block Design se conectó la señal spi_mosi_o en la PL a la entrada spi_miso_i (loopback interno), se esperaba que el valor recibido coincidiera siempre con el valor transmitido.

La consola UART mostró resultados correctos en todas las transacciones, como se observa en la Figura 6. Cada línea presenta el par de datos enviado y recibido, confirmando la coherencia del sistema.

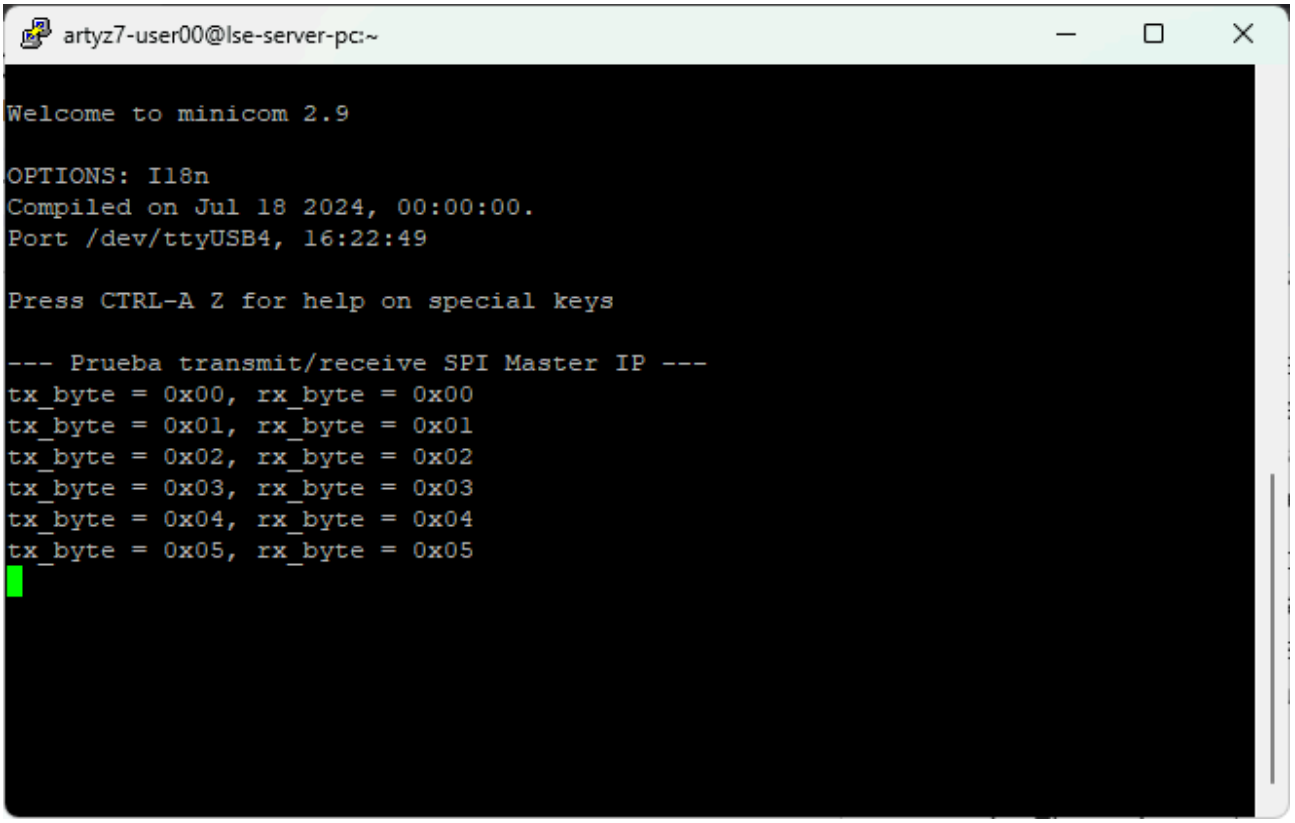

Figura 6. Par de datos transmitido/recibido visualizado a través de UART / minicom.

4.2.2. Verificación temporal mediante ILA

Se configuró el núcleo ILA para capturar las señales spi_clk_o, spi_mosi_o y spi_cs_o. La captura obtenida (Figura 7) permitió observar:

1. La correcta generación del tren de reloj spi_clk_o con la frecuencia indicada por el parámetro SCK_TARGET_HZ.
2. La activación de la línea spi_cs_o en nivel bajo únicamente durante la duración de la trama.
3. La transmisión serial de bits sobre spi_mosi_o en correspondencia con el dato enviado por software.

Estas observaciones verifican la implementación del modo SPI configurado (MODE 0: CPOL=0, CPHA=0) y la temporización esperada.

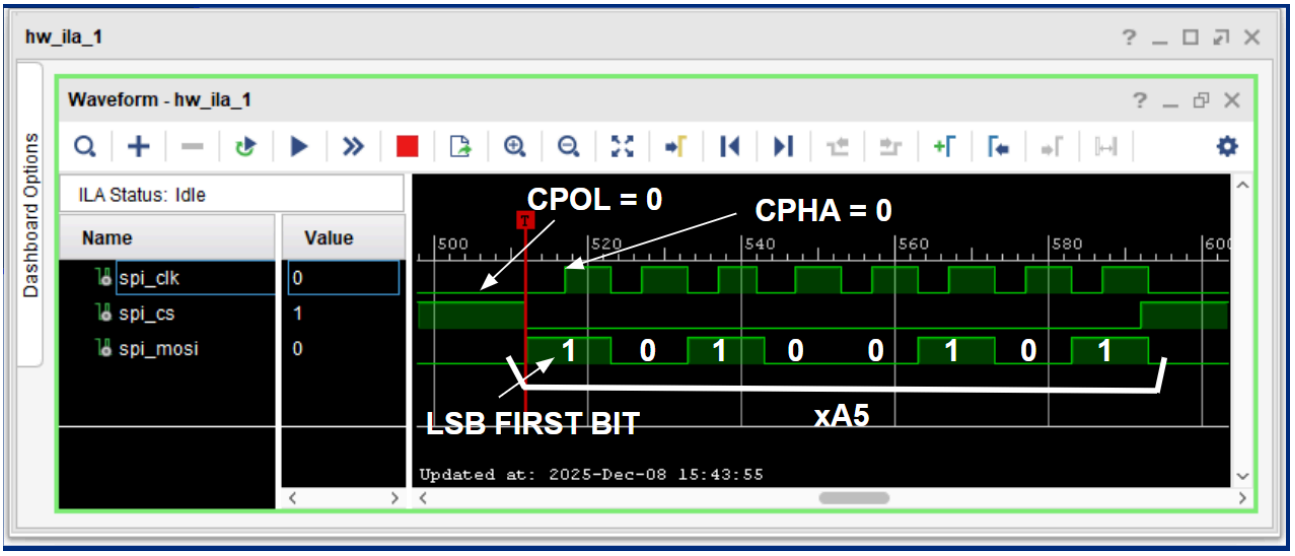

Figura 7. Verificación de parámetros del dato transmitido por medio del bloque ILA.

5. Utilización de recursos

La Figura 8 presenta la utilización de recursos del dispositivo luego de la etapa de post-implementation, correspondiente al diseño final que integra el núcleo spi_master_ip_v1_0, la infraestructura AXI necesaria para su operación, el Clocking Wizard y el núcleo ILA utilizado para la inspección de señales internas. Estas métricas permiten evaluar el impacto del IP y de los bloques complementarios sobre la FPGA, y constituyen un indicador relevante de su escalabilidad dentro de sistemas más complejos.

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

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1596	17600	9.07
LUTRAM	157	6000	2.62
FF	2456	35200	6.98
BRAM	0.50	60	0.83
IO	1	100	1.00
BUFG	4	32	12.50
MMCM	1	2	50.00

Figura 8. Utilización de recursos Post-Implementation.

La ocupación de LUTs y FFs permanece dentro de valores muy bajos (9.07 % y 6.98 % respectivamente), lo que indica que el diseño del núcleo SPI, junto con su lógica de

interfaz AXI4-Lite, representa una fracción menor del total disponible en el dispositivo. Esto permite afirmar que el periférico es liviano y fácilmente integrable en sistemas que requieran múltiples módulos adicionales o procesamiento digital complementario.

El consumo de LUTRAM y BRAM es prácticamente despreciable, reflejando que el diseño no utiliza buffers de almacenamiento significativos ni memoria embebida, y que toda la lógica secuencial se implementa mediante flip-flops convencionales.

En cuanto a los recursos de infraestructura del FPGA, el uso de BUFG es del 12.5 %, asociado a la distribución global del reloj proveniente del Zynq y a las señales de reloj adicionales empleadas por el Clocking Wizard. Este valor es razonable dentro de un diseño que incorpora dominios de reloj diferenciados para el bus AXI y para la lógica interna del SPI.

Por otro lado, la utilización de un MMCM (50 %) corresponde directamente al Clocking Wizard empleado para generar la frecuencia objetivo del reloj spi_clk_o. Dado que este recurso se utiliza exclusivamente para adaptar el reloj del sistema a los requerimientos del núcleo SPI, su ocupación es consistente con lo esperado y no representa una limitación para futuros incrementos en la complejidad del diseño.

Finalmente, el uso de recursos IO (1 %) es mínimo, ya que solamente se exportan las señales del bus SPI hacia el núcleo ILA para su inspección interna, sin emplear pines físicos de la FPGA.

6. Conclusiones

El trabajo presentado cumplió de manera integral con los objetivos establecidos para el Trabajo Práctico Final de la asignatura Microarquitecturas y Softcores, desarrollando un periférico SPI Master parametrizable en VHDL, integrándolo dentro de un sistema basado en el Zynq-7000 y validando su funcionamiento mediante una aplicación de software ejecutada en el procesador ARM Cortex-A9.

Desde el punto de vista del diseño digital, la adaptación del núcleo SPI previamente implementado en la materia Circuitos Lógicos Programables permitió reutilizar una arquitectura probada, incorporándole ahora una interfaz AXI4-Lite completamente funcional. Esto requirió un mapeo cuidadoso del espacio de registros, la separación de señales auxiliares (slv_regX_aux) para evitar condiciones de múltiples drivers, y la integración jerárquica del módulo dentro del flujo estándar de empaquetado de IP de Vivado. Como resultado, el periférico puede ser accedido de manera transparente desde software mediante operaciones de lectura y escritura sobre direcciones mapeadas en memoria, cumpliendo así con el paradigma PS–PL característico de los sistemas Zynq.

La verificación funcional realizada mediante la aplicación bare-metal en C permitió verificar el periférico en condiciones reales de operación. El software implementado demostró la correcta interacción entre el procesador y el IP, gestionando la sincronización de transmisión (tx_rdy), el inicio de cada trama (tx_dv), la recepción de datos (rx_dv) y la lectura del byte recibido. La utilización de la consola UART —accedida a través de minicom— permitió corroborar que el lazo de retorno implementado en el Block Design devolvía exactamente el valor transmitido, validando así la cadena completa de comunicaciones.

En paralelo, la inspección temporal mediante el núcleo ILA brindó evidencia de la correcta generación del reloj SPI, la secuencia de bits transmitidos por MOSI y la activación adecuada de la línea CS. La concordancia entre el comportamiento observado en el hardware y el esperado según el modo configurado (CPOL = 0, CPHA = 0) constituyó una verificación sólida de la temporización y de la lógica interna del núcleo.

El análisis de utilización de recursos confirmó que el IP desarrollado presenta una ocupación reducida de LUTs, FFs y BRAM, lo que lo convierte en un periférico liviano y fácilmente integrable en sistemas de mayor complejidad. Incluso considerando la presencia del Clocking Wizard y del ILA, el diseño mantiene un uso moderado de recursos globales, mostrando un balance adecuado entre funcionalidad, instrumentación y eficiencia.

En conjunto, el trabajo permitió consolidar conocimientos clave en diseño de hardware digital, empaquetado de IP, integración AXI, depuración basada en ILA y desarrollo de software embebido sobre el Zynq, sentando una base sólida para futuros desarrollos dentro del proyecto final de la carrera y en aplicaciones reales donde la interacción entre lógica programable y software juega un papel fundamental.