Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Implementación de la etapa de arranque (*bootstrap*) de un microprocesador basado en RISC-V para el Sistema de Reconocimiento de Patrones Acústicos (SiRPA).

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Diego Alexander Salazar Sibaja

Cartago, Costa Rica
Septiembre, 2017

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Implementación de la etapa de arranque (bootstrap) de un microprocesador basado en RISC-V para el Sistema de Reconocimiento de Patrones Acústicos (SiRPA), realizado por el señor Diego Alexander Salazar Sibaja y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Ing. Roberto Molina Robles

Profesor lector

Ing. Daniel Kokhemper Granados

Profesor lector

Ing. Alfonso Chacón Rodriguez

Profesor asesor

Cartago, 30 de noviembre de 2017

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.
En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Diego Alexander Salazar Sibaja
Cartago, 23 de noviembre de 2017
Céd: 7-0238-0365

Resumen

En el siguiente trabajo, se realizó una unidad de arranque(*bootstrap*) para el microprocesador basado en RISC-V del Sistema de Reconocimiento de Patrones Acústicos (por sus siglas SiRPA). En el momento del desarrollo de este proyecto, la arquitectura del mapa de memoria para el microprocesador del proyecto SiRPA estaba siendo reestructurada, por lo que la unidad a implementar se realizó bajo el concepto de utilizar como banco de memoria una SRAM de puerto simple.
Este sistema se implementó en una FPGA, con el fin de realizar pruebas funcionales de comunicación con la fuente del programa, una tarjeta micro SD.
Además, la unidad se implemento al nivel de implementación lógica en la herramienta Synopsys, en la tecnología de 180 nm.
Finalmente, la unidad *bootstrap* queda funcional con una gran serie de tarjetas microSD tipo SDSC y SDHC, que siguen la convención de protocolo según *SD Association*.

Palabras clave: FPGA, microSD, SPI, SDSC, SDHC, SiRPA, RISC-V.

Abstract

In the following work, a unit of program instruction loading (hereinafter *bootstrap* unit) was carried on the RISC-V based microprocessor of the Acoustic Pattern Recognition System (SiRPA). At the time of the development of this project, the memory map architecture for the SiRPA project microprocessor was being restructured, so the unit to be implemented was made under the concept of using a single port SRAM as a memory bank.
The unit has the communication functions, in SPI protocol, with the program source, data bus controller and the instruction program writing in memory.
This system was implemented on an FPGA, in order to perform functional tests of communication with the source of the program, a micro SD card.
In addition, the unit was implemented at the logic implementation level in the Synopsys tool, in 180 nm technology.
Finally, the *bootstrap* unit is compatible with a large number of SDSC and SDHC microSD cards, which follow the protocol convention according to the SD Card Association.

Keywords: FPGA, microSD, SPI, SDSC, SDHC, SiRPA, RISC-V.

A mis queridos padres

Agradecimientos

El resultado de este trabajo no hubiese sido posible sin el apoyo de mis padres, los cuales han sido mi sustento en este camino tan largo.
Además, un agradecimiento muy grande a mi novia, Alejandra Perez, que sin sus palabras de apoyo ni su amor, el camino hubiese sido más difícil.
A todos los amigos y profesores que estuvieran apoyándome durante todo este tiempo, de quienes aprendí todo lo que sé.
Finalmente un agradecimiento especial al Dr. Ing. Alfonso Chacón y Ing. Reinaldo Castro por el apoyo brindado durante la culminación de la tesis.

Diego Alexander Salazar Sibaja
Cartago, 2 de diciembre de 2017

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Acrónimo

FPGA	*Field Programmable Gate Array* (Matriz de compuertas programables)
SPI	*Serial Peripheral Interface* (Interfaz serial de periféricos)
UVM	*Universal Verification Method* (Método de Verificación Universal)
HFV	*Hardware Functional Verification* (Verificación Funcional de Hardware)
SiRPA	Sistema de Reconocimiento de Patrones Acústicos
HDL	*Hardware Description Language* (Lenguaje de Descripción de Hardware)
ASIC	*Application-Specific Integrated Circuit* (Circuito Integrado de Aplicación Específica)
UUT	*Unity Under Test* (Unidad Bajo Prueba)
FF	*Flip Flop* (Unidad de memoria)

Capítulo 1

Introducción

1.1. Entorno del proyecto

En Costa Rica se maneja un sistema de áreas protegidas y parques nacionales que protege un extensión cercana a 13.000 km^2, es decir, al 25 % del territorio. Se protege gran cantidad de flora y fauna, además de algunos patrimonios históricos. Toda esta riqueza natural es equivalente a cerca de 10.000 especies de animales y plantas, aproximadamente un 5 % de las especies descritas a nivel mundial.[18]

Se debe considerar que estos recursos naturales protegidos traen una serie de beneficios importantes para Costa Rica. Uno de ellos es una amplia fuente de materias primas, propiciado por las condiciones que generan los diferentes ambientes naturales, y que permiten la extracción de recursos. En ese ámbito, el país tiene diferentes normativas y guías para realizar este tipo de actividades causando el menor impacto posible, como la Ley para la gestión integrada del recurso hídrico, en zonas estrictamente delimitadas.[7]

Otro de los beneficios es el turismo. La gran diversidad de flora y fauna se ha vuelto un estandarte del país, con el cual se atrae a gran parte del turismo que visita el territorio nacional. En el 2016, el ingreso económico por motivo de turismo fue cercano a los 3657 millones de dolares, superando la exportación de productos como el café y banano, lo cual representó 6.4 % del producto interno bruto del país. [9]

A pesar de la importancia de los sistemas de áreas protegidas y parques nacionales, existen grandes retos debido a la extensión del territorio de cada área protegida. Uno de ellos es la protección de las tierras alejadas o de difícil acceso contra la tala de árboles y la caza de animales. Por tanto, nace la necesidad de implementar un método de monitoreo, en el cual se pueda tener una respuesta para alguna eventual amenaza.

Una de las propuestas más consolidadas es la implementación de una red de sensores inalámbricos, de bajo consumo energético, con la posibilidad de transmitir la información hasta un punto central. Actualmente, no se cuentan con registros actualizados que indican la presencia en el mercado de algún dispositivo que cuente con las características descritas.

Paralelamente, en el Instituto Tecnológico de Costa Rica (TEC), en el Laboratorio de Diseño de Circuitos Integrados, en adelante referido como DCILab, se ha trabajado en el proyecto denominado “Sistema Electrónico integrado en chip (por sus siglas en inglés SoC)”, para el reconocimiento de patrones de audio de sonido de motosierras y disparos. Esto con la finalidad de crear una base de análisis para la propuesta de sensores y la

protección ambiental.

La implementación de esta red de sensores, bajo la propuesta del DCILab, trabaja desde la adquisición de las señales acústicas hasta la notificación ante un operario de las señales identificadas. En la figura 1.1, se observa un diagrama de un nodo de esta red de sensores que permite la identificaciones de patrones acústicos.

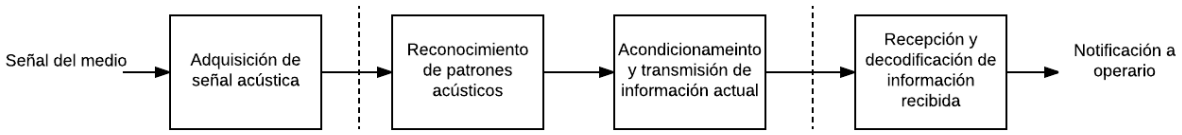

Figura 1.1: Diagrama de funcionamiento de un nodo de la red de sensores sobre la que se incorporaría el SiRPA para protección de ambientes naturales. [4].

El reconocimiento del patrón acústico obtenido es un proyecto desarrollado en el DCILab como "Sistema de Reconocimiento de Patrones Acústicos"(denominado SiRPA). Este ha pasado por diferentes etapas de desarrollo, que han incluido tanto las etapas de identificación como de clasificación de patrones acústicos (basada en modelos ocultos de Markov), esta última desarrollada sobre un microprocesador basado en instrucciones ISA RISC-V.[16]

Las pruebas hasta ahora del sistema se han hecho sobre FPGA y simulaciones de pruebas de síntesis física sobre tecnologías de CMOS.

1.2. Descripción del problema

Durante el desarrollo del SiRPA se optó por el diseño de un microprocesador de aplicación específica, debido a que los microprocesadores comerciales de propósito general no cumplen con las necesidades del proyecto. La implementación fue pensada alrededor de un procesador de aplicación específica (ASIP en adelante, por sus siglas en inglés: Application Specific Integrated Processor), lo suficientemente avanzado para ejecutar los algoritmos necesarios para reconocer patrones, pero de prestaciones de potencia contenidas en algunos miliwatts.

Como cualquier procesador, un ASIP requiere código de programa para ejecutar el algoritmo deseado. La primera versión del SiRPA, incluyó dicho código sobre ROM generado en las LUTs de la FPGA. Este método, sin embargo, es totalmente inviable cuando se desee integrar en silicio el SiRPA, pues no habría forma de modificar dicho código o ajustar variables del sistema. Por ello, se requiere de una etapa de *bootstrap*, que lea el código de una fuente del almacenamiento externo y lo guarde en la memoria RAM del sistema. Esto le otorgará programabilidad necesaria al SiRPA. A esta misma interfaz, además, servirá para ajustar variables y otros parámetros del SiRPA en tiempo de ejecución desde la misma.

1.3. Síntesis del problema

Ausencia de una etapa de *bootstrap* en el SiRPA.

1.4. Enfoque de la solución

Se específica que la comunicación con la fuente de información exterior se realice a través del protocolo de comunicación SPI. El protocolo de comunicación SPI, según SparkFun en [20], es un bus de interfaz comúnmente usado para enviar datos entre dos microprocesadores o periféricos pequeños como tarjetas SD. Este protocolo usa líneas de datos y sincronización por separado. El uso de esta interfaz entonces proverá de flexibilidad de programación al SiRPA.

En cuanto a la fuente externa con el programa de microprocesador, se propuso el uso de una tarjeta de memoria SD que cumpla con los requerimientos de la *SD Association*, con el fin de seguir el protocolo de comunicación estándar.

La implementación de la unidad será a través de un diseño tipo *Register Transfer Language* (RTL) basado en arquitectura secuencial. El lenguaje utilizado para realizar la descripción del diseño es Verilog.

La comprobación de funcionamiento de esta unidad es a través de distintas pruebas de comportamiento e implementación en una FPGA, para asegurar el cumplimiento de los estándares de las tarjetas SD y microSD.

Finalmente se plantea realizar la síntesis de la unidad con las herramienta EDA de *Synopsys* con bibliotecas de celdas estándar XFAB de 180 nm. El flujo de diseño en esta herramienta se muestra en la figura 1.2. En esta se muestra todo el flujo de diseño, desde la etapa *front end*(sintesís lógica y simulaciones por comportamiento y post síntesis) hasta la etapa *back end* (síntesis física, verificación de temporizado, potencia, impedancias parásitas, entre otros).

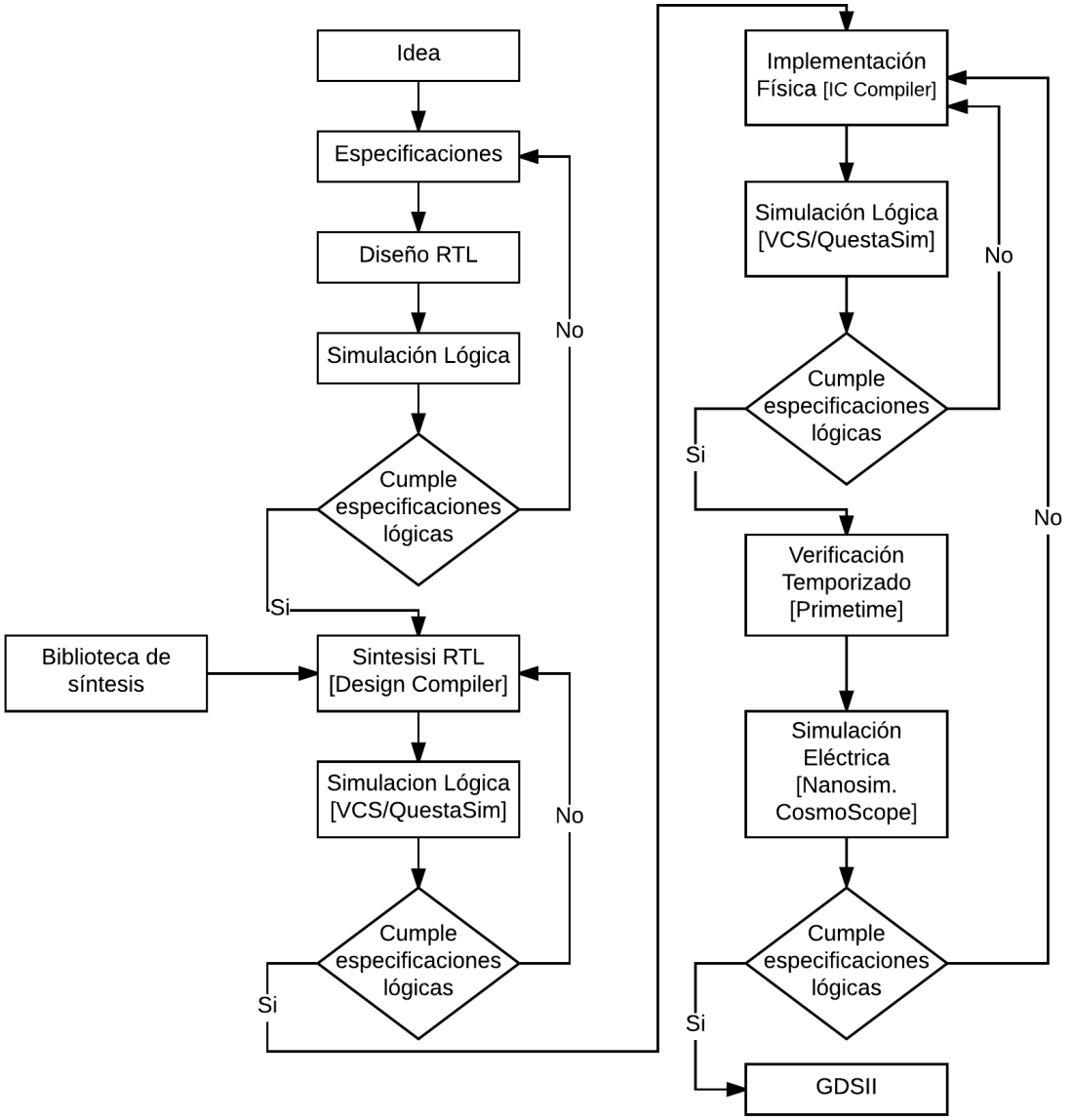

Figura 1.2: Flujo de diseño en las herramientas EDA de Synopsys.[17]

1.5. Trabajos anteriores

El SiRPA ha tenido diferentes desarrollos de hardware y software, que han contribuido al estado actual del mismo. Todas las soluciones desarrolladas para dicho proyecto se han hecho con el fin de lograr una optimización de recursos. Se han realizado diferentes tipos de soluciones, con el fin de mejorar puntos específicos, como los algoritmos de reconocimiento, implementaciones de hardware y mejoras del modelo. Algunos de los trabajos que han abarcado estos puntos son:

- Para el procesamiento de las señales se utilizaron dos herramientas importantes con el fin de obtener diferentes informaciones. Para extraer las características de las señales se utiliza un tipo especial de transformada de Fourier llamada Transformada de Ondícula y para el reconocimiento del patrón acústico se utilizó Modelos Ocultos de Markov. Lo anterior fue desarrollado por Sáenz en [21].

- Smith en [19] realiza una de las primeras aproximaciones de los Modelos Ocultos de Markov para la identificación de los sonidos de disparos y motosierras dentro de un ambiente de bosque. Además, se utilizan técnicas específicas para el filtrado digital de las señales obtenidas. Este trabajo continúa con los algoritmos trabajados por Saenz en [21].

- Salas en [15], realiza la primera implementación en FPGA del proyecto SiRPA. La implementación tendría diferentes etapas, todas con el fin de realizar el procesamiento de las señales, a partir de los algoritmos de extracción de características de las señales y los Modelos Ocultos de Markov para el reconocimiento del patrón.

- En el desarrollo del SiRPA se han realizado distintas implementaciones de hardware. Sequeira en [17] se realiza la implementación de hardware a través de un generador, escrito en C++, de código VHDL, con el fin de obtener una reducción energética. Siempre en [17], el objetivo de la implementación con una reducción de área y menor consumo energético. Alfaro en [2], se realizan las propuestas de diferentes módulos funcionales capaces de realizar funciones básicas para el procesamiento y reconocimiento de señales acústicas.

- Abrahams en [1] realiza una integración física por el método de "correcta construcción"(CBC), a través de la descripción del SiRPA en HDL. Esto con el fin de crear los archivos necesarios para la síntesis física del sistema por medio de la herramienta Synopsys.

- Montero en [4] realiza una implementación en el lenguaje de C/C++ del proyecto en un sistema embebido que ejecuta un sistema operativo GNU/Linux, con el fin de valorar si las constantes y valores de los algoritmos a implementar tienen validez. El procesamiento de las señales se realizó a través de diferentes algoritmos de análisis de discriminantes lineales, todos para modelos ocultos de Markov. Además, se tomó el modelo desarrollado en este proyecto como un punto de comparación con las implementaciones en FPGA o ASIC.

- Villegas en [24] realiza una simulación de los algoritmos trabajados con anterioridad en la herramienta LabVIEW de National Instruments, para comprobar la periodicidad y características presentes en sonidos de máquinas de combustión internas, como automóviles y motosierras.

- Salazar en [16] realizó la implementación de la unidad de clasificación del SiRPA basado en el algoritmo de Modelos Ocultos de Markov, a través del desarrollo de un microprocesador de aplicación específica. Este se implementó en una FPGA, pero la descripción de hardware se hizo en lenguaje de programación C. Además, se realizó una verificación funcional, comparando resultados contra un marco referencial.

1.6. Estado del arte

A continuación se enumeran algunos de los principales trabajos relacionados con la implementación de una unidad de *bootstrap*:

- Zhang en [25] realiza la implementación y diseño de un módulo *bootstrap* para la arquitecturas de las nuevas computadoras o SCPI-dBUS. En el mismo contemplan los protocolos de seguridad de los nuevos microprocesadores y los mapas de memoria de las RAM.

- Hartono en [8] realiza el diseño de una arquitectura para *bootstrap* y un *debugger* para un *Multiprocessor System On-Chip (MPSoC)*. En este caso la unidad diseñada era escalable, debido a la naturaleza del MPSoC, debido al incremento del número de núcleos del procesador, según especifican en dicho trabajo.

- En el trabajo de Lewandowski en [14], se realiza el desarrollo de un nuevo *bootstrap* para el microcontrolador de Atmel, de la familia AVR ATMega. Este microcontrolador cuenta con dos formas de programación, pero en dicho trabajo se pretende diseñar una nueva unidad de *bootstrap*, más pequeña y rápida. Con esto se presente dejar al programador con más espacio de memoria flash, reducir el tiempo de carga de programa o actualización del firmware del microcontrolador.

1.7. Objetivos

1.7.1. Objetivo general

- Implementar una unidad de carga de instrucciones de programa para un microprocesador basado en la arquitectura de conjunto de instrucciones RISC-V.

1.7.2. Objetivos específicos

- Implementar una versión inicial en lenguaje de descripción de hardware una unidad *bootstrap* para un microprocesador ISA RISC-V compatible.

  Indicador: *Verificación funcional RTL del código descrito a partir de pruebas testbench que comprueben el comportamiento y cumpla especificaciones de comunicación con tarjetas de memorias SD y memoria SRAM de puerto simple.*

- Comprobar en una implementación a nivel de FPGA la unidad de bootstrap.

  Indicador: *La unidad debe ser capaz de leer de la tarjeta MicroSD el código de bootstrap del microprocesador y trasladarlo a la zona de memoria indicada, sin ningún error.*

- Sintetizar lógicamente de manera correcta los módulos RTL de la unidad *bootstrap* sobre celdas estándar de una tecnología XFAB de 180nm..

  Indicador: *Verificación funcional RTL del código descrito a partir de simulación post síntesis lógica que comprueben el comportamiento y cumpla especificaciones de comunicación con tarjetas de memorias SD y memoria SRAM de puerto simple.*

Capítulo 2

Marco teórico

2.1. Unidad de *bootstrap*

La unidad de *bootstrap* o "arranque en frío" es la encargada de los procesos iniciales para un microprocesador. En ella se verifican los estados iniciales de diferentes componentes bajo control del microprocesador, como las memorias. [6]

Estas unidades entran en funcionamiento al iniciar el microprocesador o el sistema donde se encuentre el microprocesador. El microprocesador, en la mayoría de los sistemas, coloca la dirección de CPU a una ubicación de memoria de programa de instrucciones específica. En esta se encuentra un salto condicional hacia el inicio del programa de instrucciones, donde la condición evaluada es el final de las tareas del *bootstrap*. [6]

Las principales tareas del *bootstrap* son:

- Carga del programa de instrucciones hacia la memoria de programa de instrucciones.
- Comprobar el estado de la memoria.
- Realizar un diagnóstico de los periféricos del microprocesador.

2.2. Protocolo SPI

El protocolo de interfaz de periférico serial (SPI, del inglés *Serial Peripheral Interface*) es un protocolo de bus común, utilizado como interfaz de comunicación para diferentes dispositivos como sensores o tarjetas SD. Los dispositivos vinculados a este tipo de protocolo de comunicación desempeñan un rol entre maestro y esclavo. El maestro es el dispositivo que coordina la comunicación y el esclavo es un dispositivo que recibe o envía datos de un coordinador.

En este protocolo se encuentran cuatro líneas de comunicación, divididas de la siguiente manera: una línea de reloj `(SCK)`, habilitador de esclavo`(SS)`, datos desde esclavo a maestro `(MISO)` y datos de maestro a esclavo`(MOSI)`. [20] La línea de habilitador de esclavo o *Slave Select* es una señal distinta por cada esclavo, caso contrario a las demás señales, las cuales comparten el mismo bus para todos los esclavos.[13]

En este tipo de comunicación se puede distinguir dos principales topologías. La primera de estas es de un módulo SPI maestro conectado a una única unidad SPI esclavo. La segunda se compone de un módulo SPI maestro conectado con múltiples unidades SPI

esclavos. En la figura 2.1 se logra observar las dos principales topologías mencionadas para este tipo de comunicación.

La comunicación por medio del protocolo SPI se basa en una comunicación con un único

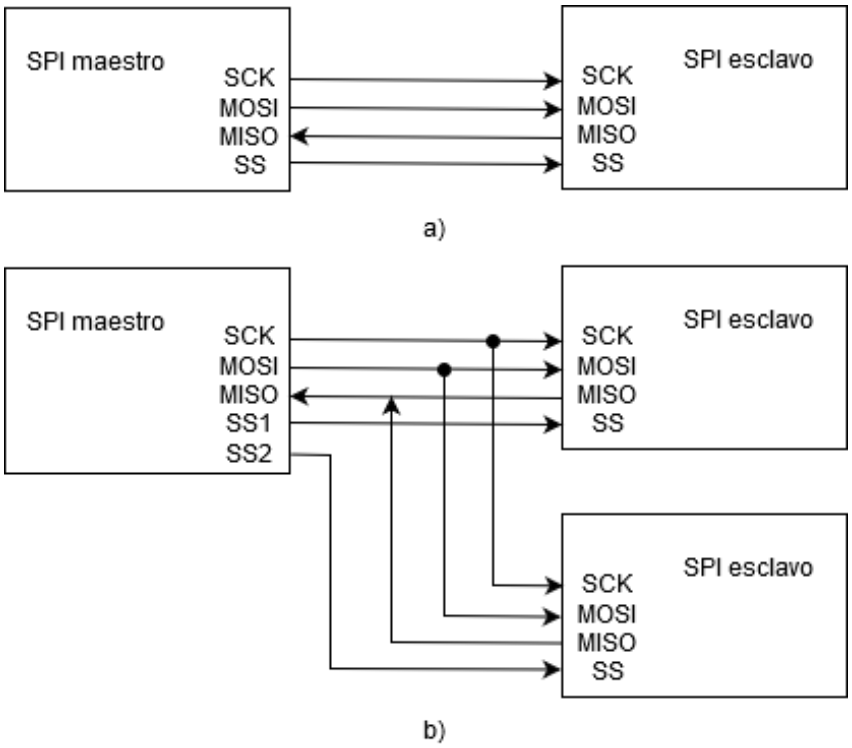

Figura 2.1: Topologías del protocolo SPI. En la figura 2.1.a se observa una topología con un solo módulo esclavo y en 2.1.b se observa una topología de dos módulos esclavos.

maestro. En otras palabras, este módulo central es el encargado de inicializar todos los módulos esclavos. Así mismo, el *host* coordina el envío y recepción de datos del o los esclavos, enviando una señal a través de la línea `SS`. Por convención, esta señal pasa a cero lógico cuando se selecciona un esclavo. Cuando esto ocurre, la señal de reloj es habilitada para ser enviada al esclavo, de caso contrario, se encuentra inhabilitada. El *host* envía información a través de la línea `MOSI`, mientras recibe los datos del esclavo a través de la línea de datos denominada `MISO`.

Existen cuatro modos de comunicación posibles en el protocolo SPI. Estos se definen a partir del flanco del reloj de sincronización donde se realiza el muestreo o lectura de los datos y el flanco donde se comienza la transmisión o escritura de datos. Estas señales son más fácilmente identificables como polaridad de reloj (CPOL) y fase de reloj (CPHA).

En la figura 2.2 se logra observar los distintos modos posibles en este protocolo, donde se muestra el flanco de muestreo (*Sampling edge*), el flanco de inicio de comunicación(*Toggling edge*), la polaridad de reloj y la fase de reloj, con respecto a la señal `SCK`, según el modo especificado. El *Mode 0* y el *Mode 3*, indicados en la figura, realiza la escritura de datos en el flanco negativo de reloj y la lectura en el siguiente flanco positivo,sin embargo se diferencian en que las señales de reloj se encuentran en cero lógico y uno lógico, respectivamente, mientras no se realiza envío de datos. Por el contrario, el *Mode 1* y el *Mode 2* realiza la escritura de datos en el flanco positivo de reloj y la lectura en el

siguiente flanco negativo,pero se diferencian en que las señales de reloj se encuentran en cero lógico y uno lógico, respectivamente, mientras no se realiza envío de datos.

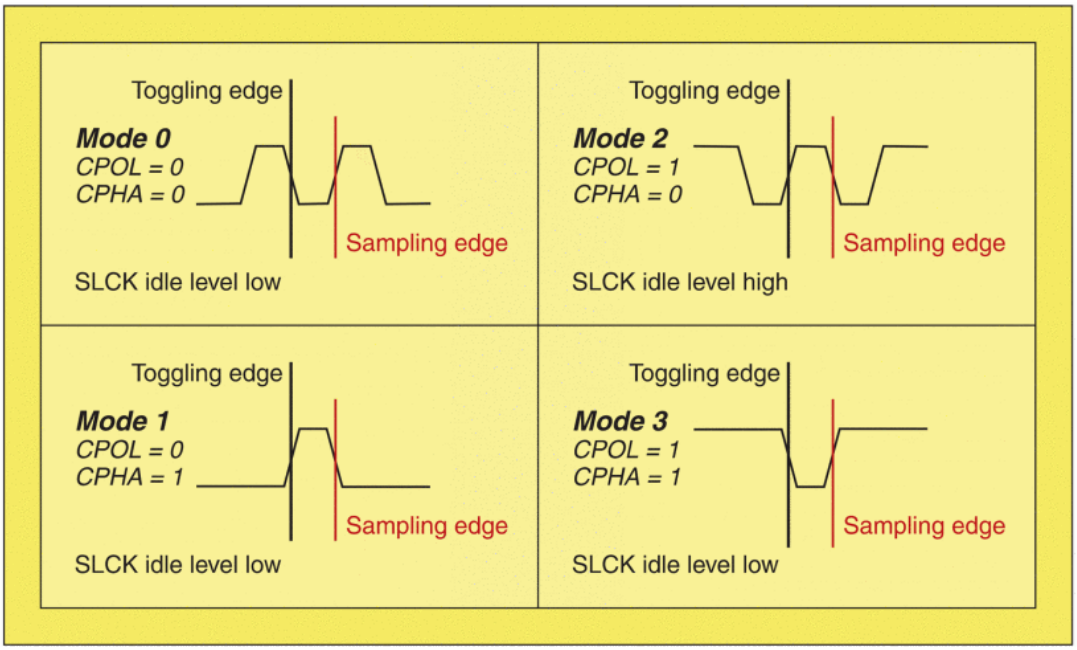

Figura 2.2: Modos de comunicación en el protocolo SPI, donde la linea de color negro indica el flanco de reloj de escritura de dato y la linea roja el flanco de reloj de lectura.[13]

2.3. Tarjetas de memoria *Secure Digital*

Las tarjetas de memoria *Secure Digital* (denominadas tarjetas de memoria SD o tarjetas SD) son dispositivos que permiten el almacenamiento de datos y capaces de usar en muchos dispositivos portátiles. *SD Association* es el ente encargado de establecer las características estándares para el mercado sobre las tarjetas SD.

Aunque hay variedad de tarjetas SD, todas tienen características y formato de comunicación común, lo que permite compatibilidad con los distintos dispositivos.

A continuación se detalla algunas de las principales características estándar definidas por *SD Association*, referentes a este tipo de memorias, en el documento de *Physical Layer Simplifield Specification Ver. 6.0*. [3] En el apéndice A se encuentra una tabla con la clasificación de las tarjetas según tres características: capacidad de memoria, rango de tensión de operación y modo de velocidad de bus.

2.3.1. Protocolos de comunicación

Las tarjetas de memoria SD cuentas con tres protocolos de comunicación. El primero es el protocolo de bus SD, el cual cuenta con 6 buses de flujo de bits y basado en comandos de ejecución. El segundo protocolo es el SPI, en el cual se trabajan con menos líneas de datos, aunque mantiene un lineamiento de comando y flujos de datos. El tercer protocolo es el *Ultra High Speed Phase* o UHS bus.

Para efectos de este proyecto se hace énfasis en el protocolo SPI de comunicación de la tarjeta de memoria SD.

Sin importar el tipo de protocolo, los comunicación es coordinada por un controlador tarjeta de memoria SD o *host*. Existen controladores capaces de establecer comunicación con varias tarjetas, para esto deben enviar dos tipos de comandos: comandos punto-a-punto o direccionados y comandos generales. Los comandos de punto-a-punto son comandos enviados a tarjetas específicas; por el contrario, los comandos generales son comandos enviados a todas las tarjetas SD controladas.

La tarjeta de memoria SD debe ser inicializada cuando entra en operación, con el fin de identificar el tipo de protocolo de comunicación a ejecutar e inicializar valores internos de la tarjeta.

2.3.2. Dimensiones y asignaciones de contactos tarjetas SD

Existen tres tipos de tarjetas SD según su tamaño, los cuales son las tarjetas SD, tarjetas miniSD y las tarjetas microSD. Las tarjetas SD tienen las siguientes dimensiones estándar de fabricación: 24 x 32 x 2,1 mm o 24 x 32 x 1,4 mm. Las tarjetas miniSD tienen dimensiones de 20 x 21.5 x 1,4 mm. Las tarjetas micro SD tienen dimensiones de 15 x 11 x 1 mm.

En todos los tipos de tarjetas, menos las UHS, existen los mismos puertos de comunicación. En total son 9 pines o puertos. En la figura 2.3 se muestra la ubicación de los mismos en la tarjeta de memoria SD.

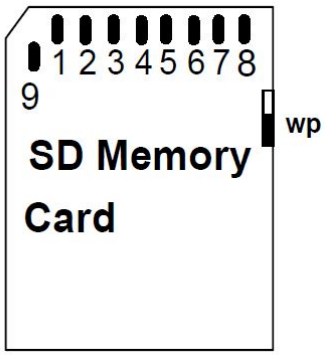

Figura 2.3: Formato típico de una tarjeta de memoria SD: forma e interfaz (vista superior).[3]

En la tabla 2.1 se define la función de cada contacto, según el protocolo de comunicación y respecto a la enumeración indicada de la figura 2.3.

En las tarjetas que ejecutan el protocolo UHS cambia la cantidad de contactos metálicos o pines de la misma. En la figura 2.4 se encuentra una vista frontal de una tarjeta habilitada para este tipo de protocolo.

Tabla 2.1: Definición de funciones de los contactos de la tarjeta de memoria SD según protocolo de comunicación. [3]

Pin #	Modo SD Bus			Modo SPI Bus		
	Nombre	Tipo	Descripción	Nombre	Tipo	Descripción
1	CD/DAT3	I/O/PP	Tarjeta detectada/ Linea de datos(bit 3)	CS	I	*Chip Select*
2	CMD	I/O/PP	Comando/ Respuesta	DI	I	Dato entrante
3	VSS1	S	Alimentación negativa de tensión	VSS	S	Alimentación negativa de tensión
4	VDD	S	Alimentación positiva de tensión	VDD	S	Alimentación positiva de tensión
5	CLK	I	Reloj	SCK	I	Reloj
6	VSS2	S	Alimentación negativa de tensión	VSS2	S	Alimentación negativa de tensió
7	DAT0	I/O/PP	Linea de datos(bit 0)	DO	O/PP	Datos salida
8	DAT1	I/O/PP	Linea de datos(bit 1)	RSV	–	–
9	DAT2	I/O/PP	Linea de datos(bit 2)	RSV	–	–

Leyenda tabla 2.1

S: Fuente de alimentación I: Señal de entrada O: Señal de salida
PP: Señal tipo I/O

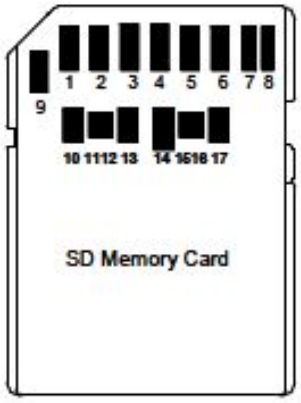

Figura 2.4: Tarjeta de memoria SD UHS-II: forma e interfaz (vista superior). [3]

2.4. SRAM de puerto sencillo

La memoria estática de acceso aleatorio estático, o por sus siglas en inglés SRAM, es un tipo de memoria construida a partir de semiconductores, que utiliza circuitos biestables para el almacenamiento de bits. [22]

Las SRAM al ser memorias estáticas no requieren que la información sea renovada constantemente, caso contrario a la DRAM (RAM dinámica).

Existen dos tipos de SRAM: asíncrona y síncrona. La SRAM síncrona es más rápida que su similar asíncrona.[22]

2.5. FIFO

FIFO es el acrónimo para las siglas en inglés de *First Input First Output* o en español "Primero entra, primero sale". Este tipo de unidad es útil como almacenamiento intermedio entre distintas unidades, cuyas velocidades de procesamiento son distintas. [12]

Existen otros tipos de unidades de almacenamiento intermedias, como las LIFO, cuyas siglas vienen del inglés *Last Input First Output*, o en español "Último entra, primero sale".

Una FIFO puede ser implementada en hardware o software, teniendo cada implementación sus ventajas y desventajas. Una implementación en software trae gran flexibilidad de cambio, sin embargo, una implementación en hardware provee mayor velocidad de procesamiento.

No hay un único tipo de unidades FIFO, sino que existen diferentes diseños de las unidades. Por lo general, tienen buses de entrada de datos y salida de datos, así como banderas que indican si la FIFO se encuentra llena o vacía. También, suelen encontrarse diseños con señales de control sobre la habilitación de escritura o lectura. [12]

Capítulo 3

Diseño de unidad *bootstrap*

La unidad de *bootstrap* diseñada tiene como función realizar la carga de instrucciones de programa para el microprocesador de una tarjeta de memoria SD, por medio del protocolo SPI y guardarlo en una memoria SRAM de puerto simple.

La tarjeta de memoria SD utilizada es de tipo SDHC, debido a que son las más fáciles de conseguir en el mercado actual, además de su compatibilidad con la mayoría de dispositivos y gran capacidad de memoria (mayores a 2GB).

La unidad esta diseñada para que una vez realizada su función principal permita que el microprocesador ejecute operaciones sobre la memoria SRAM o la tarjeta SD.

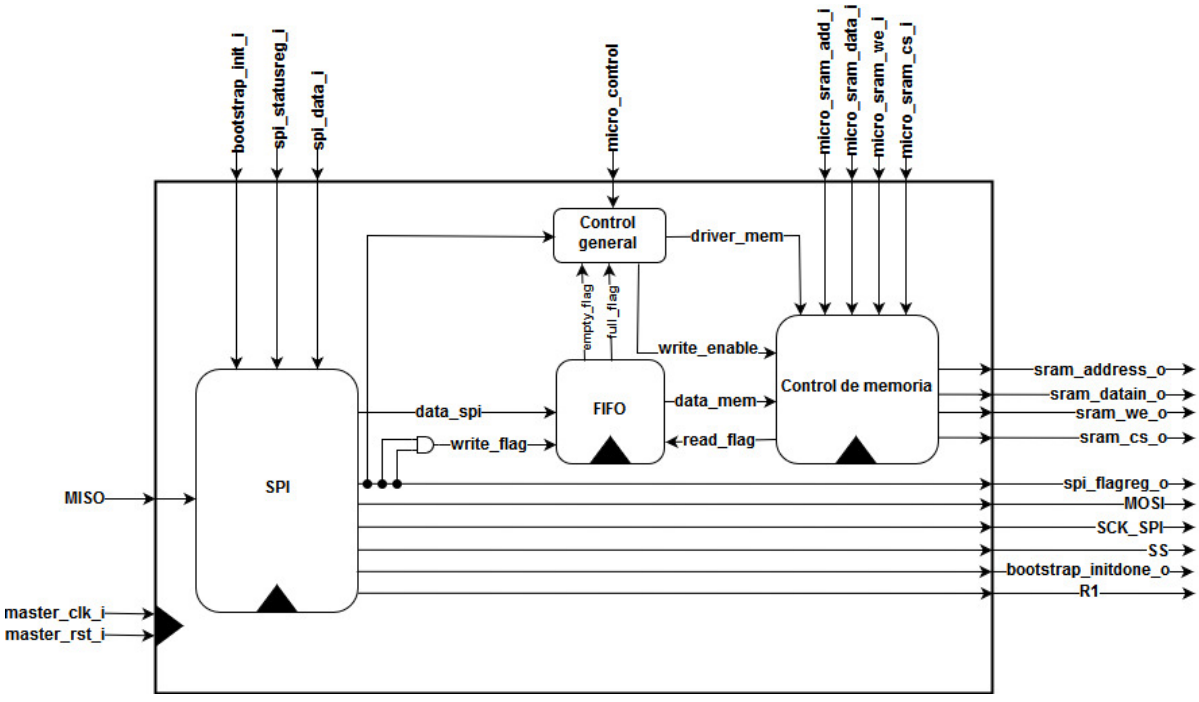

Figura 3.1: Diagrama RTL superior de la unidad de *bootstrap* diseñada.

En la figura 3.1 se distinguen tres bloques principales: `SPI` (interfaz SPI con tarjeta de memoria SD), `FIFO` y el `Control de memoria` (interfaz de control de memoria). Cada bloque será descrito en las siguientes secciones.

El diseño se concibió para implementar las dos principales funciones de la unidad, que son el acceso de información en una tarjeta SD y la escritura/lectura de una memoria SRAM de puerto sencillo, lo cual se logra a través de los bloques SPI y Control de memoria, respectivamente. El módulo `FIFO` permite que estos bloques trabajen independientemente.

El control general es una sección de la unidad *bootstrap* que se encarga de verificar el estado de la `FIFO` y controlar el inicio y modo de operación en el módulo `Control de memoria`.

En la tabla 3.1 se observa la descripción de las entradas, salidas y señales internas del *bootstrap*.

Tabla 3.1: Señales de la unidad *bootstrap*.

Señal	Ancho de bus (bits)	Tipo de señal	Módulo	Señal en módulo destino
master_clk_i	1	Entrada	Todos	-
master_rst_i	1	Entrada	Todos	-
MISO	1	Entrada	SPI	MISO
bootstrap_init_i	1	Entrada	SPI	spi_init_i
spi_statusreg_i	8	Entrada	SPI	spi_status_reg
spi_data_i	48	Entrada	SPI	spi_data_i
micro_control	1	Entrada	-	-
micro_sram_add_i	13	Entrada	Control de mem.	micro_sram_add_i
micro_sram_data_i	32	Entrada	Control de mem.	micro_sram_data_i
micro_sram_we_i	1	Entrada	Control de mem.	micro_sram_we_i
micro_sram_cs_i	1	Entrada	Control de mem.	micro_sram_cs_i
sram_address_o	13	Salida	Control de mem.	sram_address_o
sram_datrain_o	32	Salida	Control de mem.	sram_datain_o
sram_we_o	1	Salida	Control de mem.	sram_we_o
sram_cs_o	1	Salida	Control de mem.	sram_cs_o
spi_flagreg_o	3	Salida	SPI	spi_flagreg_o
MOSI	1	Salida	SPI	MOSI
SS	1	Salida	SPI	SS
SCK_SPI	1	Salida	SPI	SCK_SPI
. R1	8	Salida	SPI	R1
bootstrap_initdone_o	1	Salida	SPI	spi_initdone_o
data_spi	32	Interna	SPI /FIFO	spi_data_o fifo_datawrite_i
write_flag	1	Interna	SPI/FIFO	spi_flag_reg fifo_writeflag_i
data_mem	32	Interna	FIFO/Control de mem.	fifo_readdata_o fifo_datain_i
read_flag	1	Interna	FIFO/Control de mem.	fifo_readflag_i read_fifo_o
driver_mem	1	Interna	CG/Control de mem.	micro_control
write_enable	2	Interna	CG/Control de mem.	write_enable
empty_flag	1	Interna	CG/FIFO	fifo_emptyflag_o
full_flag	1	Interna	CG/FIFO	fifo_fullflag_o

3.1. Interfaz SPI con tarjeta de memoria SD

El interfaz SPI es la sección del *bootstrap* encargada del control de la tarjeta de memoria SD y la ejecución del algoritmo de arranque. Para su implementación se tendrá que tomar en cuenta las especificaciones físicas de las tarjetas, ya que estas serán limitantes del diseño. En esta sección se presenta un resumen de las principales características de las tarjetas, el diseño del interfaz y una verificación funcional por medio de pruebas de comportamiento.

3.1.1. Características de la tarjeta de memoria SD

Para la determinación de las características de implementación del interfaz SPI del *bootstrap* se deben considerar algunas características físicas de las tarjetas de memoria SD.

Entre las características a considerar están los registros de información de la tarjeta, el control de reloj y frecuencias aceptadas, la tensión de alimentación de la tarjeta, el código de redundancia cíclica (por sus siglas en inglés CRC), comandos, respuestas y otras especificaciones del protocolo SPI en las tarjetas de memoria SD.

3.1.1.1. Registros de la tarjeta de memoria SD

En la tarjeta SD o microSD existe una serie de registros con información referente a estados de error, identificación física de la tarjeta, estado específico de los datos, estado de la tarjeta SD, entre otros. Todos ellos son accesibles a través de comandos específicos (refiérase a la sección 3.1.1.6) . Cada registro contiene una extensión variable, dependiente de la información que suministran o contengan.[3]

En la tabla 3.2, se muestra un resumen de los registros de la tarjeta de memoria SD, con el nombre asociado al registro, tamaño y descripción de registro.

Tabla 3.2: Descripción de registros de la tarjeta de memoria SD. [3]

Nombre	Tamaño	Descripción
OCR	32 bits	Este registro contiene datos sobre información de operación, en cuanto al tensión de operación, estado de alimentación de la tarjeta y otros datos. Sus valores son accesibles por medio del comando 8 o CMD8.
CID	128 bits	Este registro contiene información de la tarjeta que le permite ser identificada y es un valor único por tarjeta. Esta identificación se compone por código del fabricante, nombre de producto, serial del producto y fecha de manufacturación.
CSD	128 bits	Este registro provee información relacionada con el acceso al contenido de la tarjeta SD. La estructura del mismo varía si la tarjeta es de capacidad estándar, alta o extendida. Los valores de este registro son modificables con el comando 27 o CMD27.
RCA	16 bits	El registro RCA es un registro editable capaz de guardar una dirección temporal que el *host* le otorga a la tarjeta para su reconocimiento. Es accesible a través del comando 7 o CMD7. No habilitado en protocolo SPI.
DSR	16 bits	Registro para manejo de *drivers*, configura los controladores de salida de la tarjeta.
SCR	64 bits	El Registro de Configuración de Tarjeta SD o SCR es un registro que provee información de las características especiales de la tarjeta SD. Contiene la especificación de la estructura del SCR y versión específica de la tarjeta SD.

3.1.1.2. Control de reloj

La señal de sincronización de la tarjeta de memoria SD es controlada por el *host*. El manejo de esta señal permite a la tarjeta entrar en un modo de ahorro de energía o de flujo de datos.

La memoria se encuentra en modo de ahorro de energía cuando la señal de reloj esta en cero lógico. En este modo se asegura que no se ejecuten procesos internos.

En caso contrario, para entrar en modo de flujo de datos, la señal de reloj debe estar habilitada. Cuando esto sucede se ejecuta los procesos internos y se es capaz de procesar los datos recibidos.

En el caso que se encuentre en modo de flujo de datos y no se envía algún comando, la señal de comandos debe permanecer en uno lógico.[3]

3.1.1.3. Tiempos de bus en tarjetas de memoria SD

De acuerdo con *SD Association* en [3], la tarjeta de memoria SD debe tener una frecuencia de reloj de 25MHz, para una velocidad estándar. Si se utiliza una configuración de alta velocidad, la frecuencia puede ser mayor a 50 MHz. Para cada frecuencia existen ciertas limitaciones en los buses de datos, para asegurar la integridad de los datos.

El tiempo de *set up* es el tiempo mínimo que debe mantenerse un dato antes del cambio de flanco de un reloj con el fin de asegurar su debido procesamiento. En el caso del tiempo de *hold* es el tiempo mínimo que debe mantenerse un dato después del cambio de flanco de reloj para asegurar su debido procesamiento.

En las figuras 3.2 y 3.3 se observan los diagramas de tiempos de los buses de datos a velocidad estándar de tarjeta SD, en las líneas `MOSI` (*Card Input*) y `MISO` (*Card Output*), respectivamente. De las figuras se puede observar que la transmisión de datos a velocidad estándar puede tener un valor de frecuencia de reloj máximo de 50 MHz. El tiempo de *set up* de los datos de entrada a la tarjeta es mínimo 5 ns, al igual que el tiempo de *hold*.

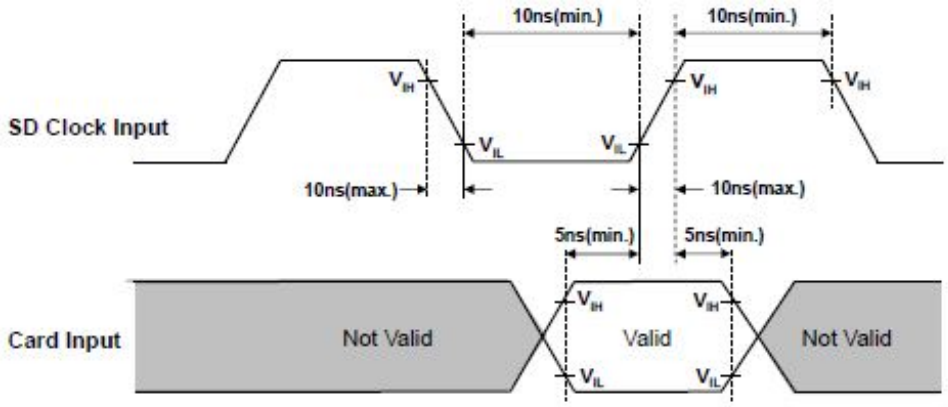

Figura 3.2: Diagrama de tiempos para bus de entrada de datos, a velocidad estándar de tarjeta SD(25MHz). [3]

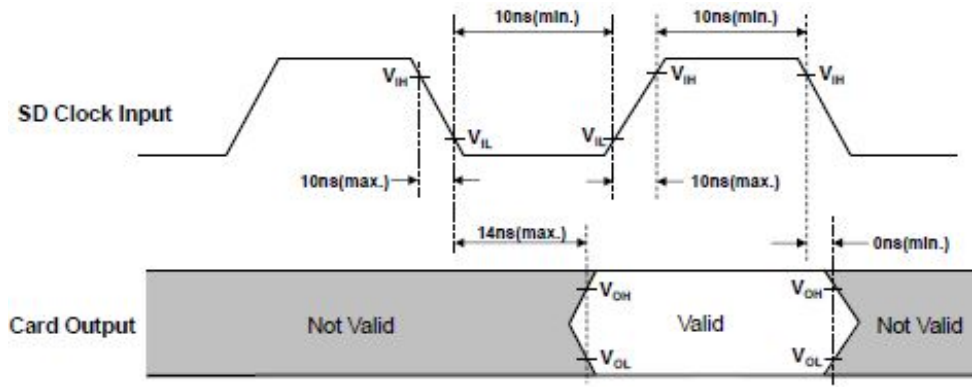

Figura 3.3: Diagrama de tiempos para bus de salida de datos, a velocidad estándar de tarjeta SD(25MHz). [3]

3.1.1.4. Alimentación tarjeta SD

El esquema de alimentación de una tarjeta de memoria SD es manejada por el *host*. La alimentación de la tarjeta se suministra por medio de pines o contactos específicos en la tarjeta, los cuales se especificaron en la sección 2.3.2.

En la figura 3.4, se muestra el diagrama de inicio de alimentación de la tarjeta SD.

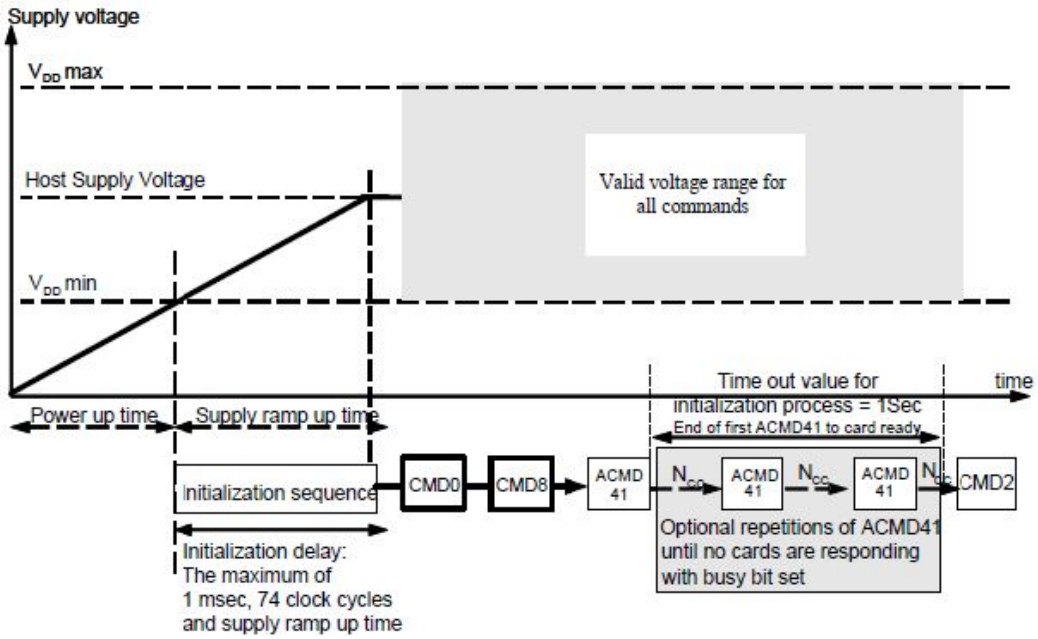

Figura 3.4: Diagrama de inicialización de tensión de alimentación de tarjeta SD. [3]

Cuando la tarjeta de memoria detecta la tensión de alimentación, requiere un tiempo mínimo para alcanzar una tensión de umbral mínima o VDDmin. A este período se le denomina *"Power up time"*. Posteriormente, procede una etapa denominada *"Supply ramp up time"*, que es donde se realiza una secuencia interna de inicialización, en la cual la tensión aumenta desde VDDmin hasta el valor de operación en un tiempo máximo de

1 ms y se envía 74 ciclos de reloj antes de enviar cualquier comando.[3]

Después de este proceso, la tarjeta SD es capaz de recibir comandos. Primero se debe realizar una inicialización de la tarjeta, lo cual comienza con el comando 0 o `CMD0`, pero esto se explicará detalladamente en la sección 3.1.1.7.

3.1.1.5. *Cyclic Redundancy Code(CRC)*

El *Cyclic Redundancy Code* o por sus siglas CRC es un bloque de datos para la protección de comandos, respuestas y transferencias de datos de la tarjeta de memoria SD. Cada comando tiene un código CRC específico, al igual que cada bloque de transferencia de datos.

El código CRC puede tener diferente ancho de palabra. En el caso de las tarjetas SD estos se limitan a 7 bits y 16 bits. El código CRC7 tiene ancho de 7 bits y es usado para todos los comandos,la mayoría de respuesta de la tarjeta y la respuesta de los registros `CID` y `CSD`. El procesamiento o generación se da a través del polinomio de la ecuación 3.1.

$$\begin{gathered} Polinomio\ generador = G(x) = x^7 + x^3 + 1 \\ M(x) = (1^{er}\ bit) * x^n + (2^{ndo}\ bit) * x^{n-1} + ... + (último\ bit) * x^0 \\ CRC[6:0] = Resultado\ final[M(x) * x^7 / G(x)] \end{gathered} \tag{3.1}$$

El primer bit va a ser el bit más significativo de la palabra, ya sea comando, respuesta, `CID` o `CSD`. El grado del polinomio $M(x)$ es el número de bits del cual se generará el código CRC. En el caso de los comandos y respuestas n tendrá valor de 39. En el caso de la respuesta de los registros `CID` o `CSD`, el grado del polinomio es 119.

En la figura 3.5 se puede observar un generador de código CRC7, implementado a nivel de hardware. Este se compone de un arreglo de 7 *flip flops*, donde la salida de cada uno es un bit del código CRC7. El bit menos significativo es la salida del primer FF, de izquierda a derecha. El bit más significativo es la salida del último FF, de izquierda a derecha. Los datos de entrada es la palabra a generarle el código CRC en forma serial, empezando por el bit más significativo. Para generar el código CRC7 se debe multiplicar (operación lógica AND) el bit de entrada de la palabra a generarle el código y el bit más significativo del generador. Este resultado será el bit menos significativo del generador. Además, este resultado debe sumarse (operación lógica OR) con el bit 2 del generador, para dar como resultado el bit 3 del mismo. Todos los bits son desplazados hacia la derecha con cada ciclo de reloj. La operación finaliza al haber procesado toda la palabra a la cual se desea generar el código CRC7.

El código CRC16, en forma similar al CRC7, es un código de longitud de 16 bits que busca la protección de un bloque de datos en modo de transferencia en las líneas de datos, para el caso del protocolo SD bus. De igual forma, este código se puede generar a través de un polinomio y es el que se muestra en la ecuación 3.2.

$$\begin{gathered} Polinomio\ generador = G(x) = x^{16} + x^{12} + x^5 + 1 \\ M(x) = (1^{er}\ bit) * x^n + (2^{ndo}\ bit) * x^{n-1} + ... + (último\ bit) * x^0 \\ CRC[6:0] = Resultado\ final[M(x) * x^{16} / G(x)] \end{gathered} \tag{3.2}$$

De igual manera, el primer bit es el más significativo. El grado del polinomio depende del tamaño de bloque a transmitir. En el caso máximo, el valor de n debe ser igual o menor a

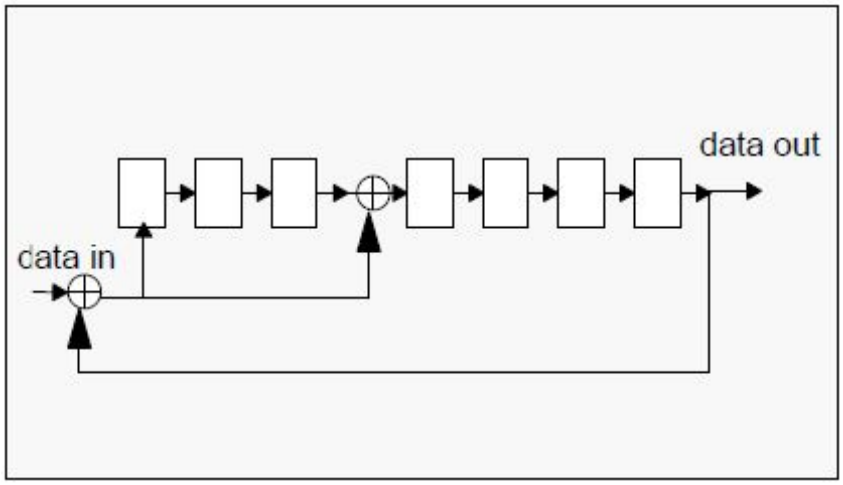

Figura 3.5: Generador de código CRC7. Cada recuadro es un *flip flop* y el circulo es una operación aritmética. El código CRC7 se obtiene de la salida de cada FF una vez procesada la palabra. [3]

16383. Así mismo, se generará un código CRC16 por cada línea de transmisión de datos.

En la figura 3.6 se puede observar un generador de código CRC16, implementado a nivel de hardware. Similar a la implementación de hardware para la obtención del CRC7, se compone de un arreglo de *flip flops*, en este caso de 16. La salida de cada uno es un bit del código CRC16. El bit menos significativo es la salida del primer FF, de izquierda a derecha. El bit más significativo es la salida del último FF, de izquierda a derecha. Los datos de entrada es la palabra a generarle el código CRC en forma serial, empezando por el bit más significativo. Para generar el código CRC7 se debe multiplicar (operación lógica AND) el bit de entrada de la palabra a generarle el código y el bit más significativo del generador. Este resultado será el bit menos significativo del generador. Además, este resultado debe sumarse (operación lógica OR) con el bit 4 y el bit 11 del generador, para dar como resultado el bit 5 y bit 12 del mismo, respectivamente. Todos los bits son desplazados hacia la derecha con cada ciclo de reloj. La operación finaliza al haber procesado toda la palabra a la cual se desea generar el código CRC16.

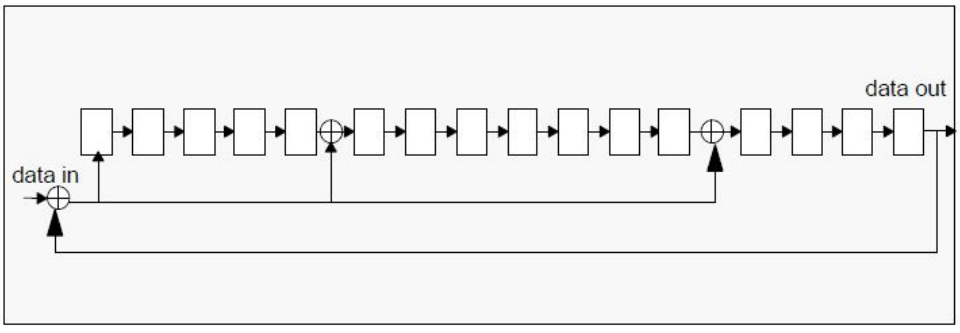

Figura 3.6: Generador de código CRC16. Cada recuadro es un *flip flop* y el circulo es una operación aritmética. El código CRC16 se obtiene de la salida de cada FF una vez procesada la palabra. [3]

3.1.1.6. Comandos y respuestas de control para tarjetas de memoria SD

La tarjeta de memoria SD tiene la característica de comunicación por diferentes protocolos; sin embargo, para todos los protocolos ejecutables por la tarjeta, existe una serie de comandos que permite diferentes operaciones en la misma. Algunos comandos se utilizan para acceder o modificar valores de los registros de la tarjeta SD, para inicializar la tarjeta SD o para escritura o lectura de memoria en la tarjeta SD.

Todos los comandos tienen un ancho de palabra de 48 bits y una estructura determinada. En la tabla 3.3 se muestra el formato de los comandos. Este incluye un primer bit de inicio con un cero lógico, seguido de un bit con valor de uno lógico que indica el inicio de la transmisión. Los siguientes seis bits son el indicador de comando, que es el número binario del comando a enviar. Posteriormente, se encuentran una serie de 32 bits conocidos como el argumento, que pueden variar según los comandos. Finalmente, los últimos ocho bits se componen por el código CRC7 y un bit de finalización. [3]

Tabla 3.3: Formato de comandos para tarjeta de memoria SD. [3]

Descripción	Bit de inicio	Bit de transmisión	Indicador de comando	Argumento	CRC7	Bit final
Posición del bit	47	46	[45:40]	[39:8]	[7:1]	0
Ancho	1	1	6	32	7	1
Valor	’0’	’1’	x	x	x	’1’

En el anexo B se encuentra la lista de comandos que soporta una tarjeta SD. Además de los comandos, existen una serie de respuestas por parte de la tarjeta SD que indican errores o estados específicos de la comunicación.

Existen distintos tipos de respuestas. En el protocolo SPI todos los datos se envían de forma serial y con el bit más significativo de primero. Estas respuestas indican errores de estado por comandos ilegales o problemas con el código CRC, entre otros. A continuación se explican las diferentes respuestas de la tarjeta SD.

- Formato R1

  Este es una respuesta que se envía en la mayoría de los comandos. Este tiene un tamaño de un byte y siempre el bit más significativo tiene un valor de cero lógico. Los restantes bits son indicaciones de error. Los errores son señalados con uno lógico. En algunos casos la respuesta R1 puede ir acompañada por algunos bits en cero lógico que indican que la tarjeta se encuentra ocupada. La estructura de la respuesta R1 se observa en la figura 3.7.

  - El bit de `In idle state` indica que la tarjeta esta en proceso de inicialización.
  - El bit de `Erase reset` indica que se ejecutó un comando en que se borró un bloque de datos.
  - El bit de `Illegal command` indica que fue detectado un comando ilegal.
  - El bit de `Communication CRC error` indica error en el código CRC recibido.

- El bit `Erase sequence error` indica error en una secuencia de borrar datos.
- El bit `Address error` indica error en cuanto a la dirección utilizada para algún comando. Esto debido a que no concuerda con el ancho de bloque de escritura.
- El bit `Parameter error` indica que existe error en el argumento del comando enviado.

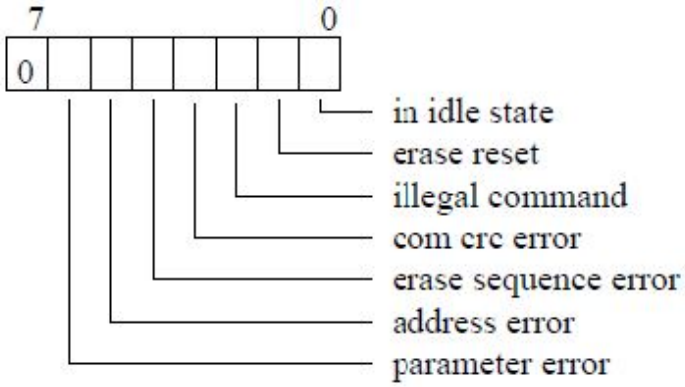

Figura 3.7: Formato de la respuesta R1. Este formato identifican errores de los comandos enviados a la tarjeta de memoria SD. [3]

- Formato R2

  Este formato de respuesta es el que se produce al responder al comando *SEND_STATUS*, y tiene dos bytes de longitud. El formato se puede observar en la figura 3.8. El primer byte tiene la misma configuración que una respuesta `R1`. En el segundo byte se encuentran las banderas que se explicarán a continuación.

  - El bit de `Erase param` indica que se envió una sección, sector o bloque inválido para borrar.
  - El bit de `Write protect violation` que un comando intento escribir en un bloque protegido contra escritura.
  - El bit de `Card ECC failed` indica que el sector al que se quiere acceder esta corrupto.
  - El bit de `CC error` indica que un controlador interno de la tarjeta falló.
  - EL bit de `Error` indica un error general o desconocido durante alguna operación.
  - El bit de `Write protect erase skip | lock/unlock command failed` indica un fallo de operación, cuando se borra un sector protegido contra escritura u ocurre un error durante el bloqueo o desbloqueo de la tarjeta SD.
  - El bit de `Card is locked` indica cuando la tarjeta SD se bloquea o protege contra escritura por parte del usuario. .

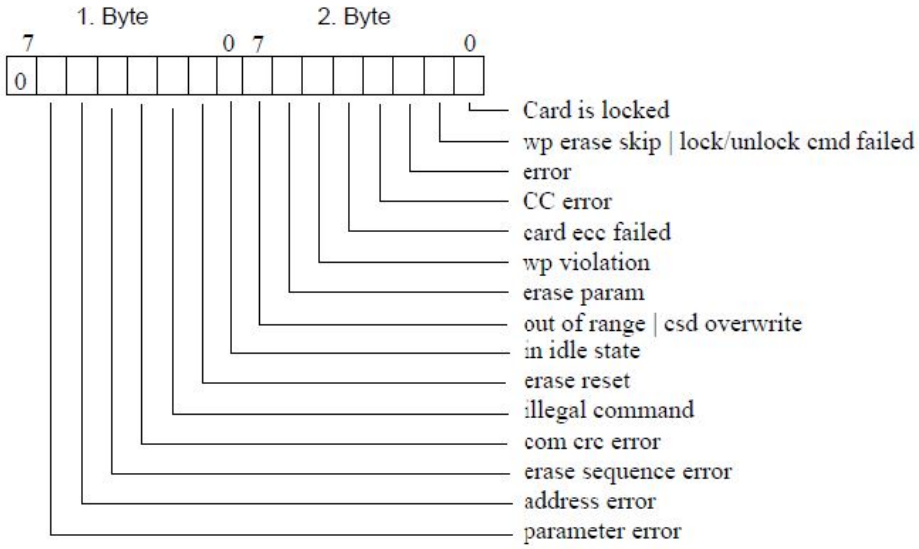

Figura 3.8: Formato de la respuesta R2. Este formato es una extensión de identificación de errores de comandos, en especial los comandos relacionados con la escritura, lectura o borrado de sectores de memoria. [3]

- Formato R3

  Este tipo de respuesta se genera cuando la tarjeta SD recibe el comando *READ_OCR* y tiene un ancho de palabra de 5 bytes. Se compone, desde el bit más significativo, por una respuesta tipo `R1` en el primer byte y el contenido del registro OCR en los últimos cuatro bytes. En la figura 3.9 se observa el formato de esta respuesta.

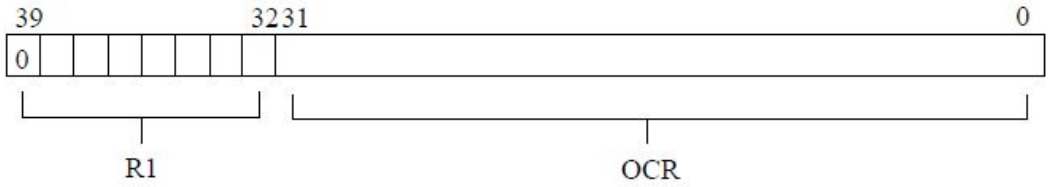

Figura 3.9: Formato de la respuesta R3. [3]

- Formato R7

  Esta respuesta se produce cuando la tarjeta SD recibe el comando *SEND_IF_COND*. La respuesta tiene un ancho de palabra de 4 bytes. La estructura de esta respuesta se muestra en la figura 3.10. El primer byte es una respuesta tipo `R1`. En los siguientes tres bytes se encuentra la versión de comando (`command version`), una serie de bits reservados (`reserved bits`) y el rango de tensión aceptado(`voltage accepted`). En el último byte se encuentra el patrón de verificación (`check pattern`).

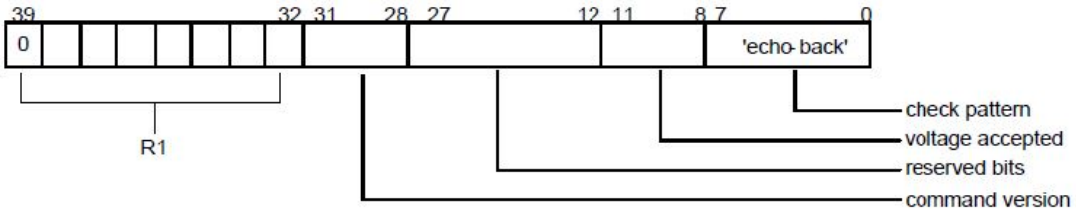

Figura 3.10: Formato de la respuesta R7. Esta es la respuesta de la tarjeta al comando *SEND_IF_COND*. [3]

- Ficha de respuesta de datos (`data response token`)

  Este tipo de respuesta es enviada al *host* al final de un bloque de datos en una operación de escritura en la tarjeta. Es de un byte de ancho de palabra y tiene el formato visto en la figura 3.11.

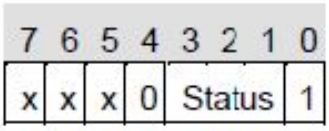

Figura 3.11: Formato de una respuesta tipo `data response token`. [3]

Los tres bits llamados estado o *status* pueden tener tres distintos valores, los cuales tienen un significado específico sobre la aceptación o rechazo de los datos para la escritura. En la tabla 3.4 se encuentran el valor esperado de estos bits y su correspondiente significado.

Tabla 3.4: Valores y significado de los bits *status* de la respuesta tipo *data response token*

Valor en binario	Significado
010	Datos aceptados
101	Datos rechazados debido a CRC.
110	Datos rechazados debido a error de escritura

- Ficha de arranque de bloque (`Start Block Tokens`) y ficha de parar transmisión (`Stop Tran Token`).

  Los comandos de lectura y escritura de datos tienen asociados transferencias de datos. Cada bloque de datos enviados y transmitidos tiene un byte de cabecera, que indica si es un dato de un comando de lectura o escritura de bloque único, lectura o escritura de bloques múltiples o final de dato para un bloque en escritura múltiple.

  Para lectura de bloque único, escritura de bloque único o lectura de múltiples

bloques se utiliza el byte de cabecera 8'b11111110.

Para escritura de múltiple bloques, el primer byte del bloque debe ser la cabecera 8'b11111100. Al finalizar el bloque de datos se debe enviar el byte 8'b11111101. Esta última cabecera solo se utiliza para separar cada bloque en la escritura de bloques múltiples, no detiene la escritura.

- Ficha de error de datos (`Data Error Token`)

  Cuando la operación de lectura de datos falla y la tarjeta no pueda enviar datos, responderá con un mensaje de un byte indicando el tipo de fallo. En la figura 3.12 se observa la estructura de este byte.

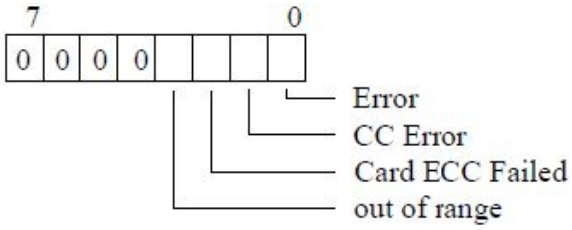

Figura 3.12: Formato de *Data Error Token*. [3]

3.1.1.7. Inicialización en las tarjetas de memorias SD

El protocolo SPI en las tarjetas de memoria SD consiste en un protocolo secundario de comunicación, aunque igualmente válido. La comunicación SPI se caracteriza por ser *full duplex*, pero en las tarjetas de memoria SD la comunicación es *half duplex*, ya que la tarjeta espera primero un comando para liberar una respuesta. Si el *host* y la tarjeta SD se envían datos al mismo tiempo la comunicación no será exitosa.

El protocolo SPI en la tarjeta de memoria SD se caracteriza por el flujo de datos en bytes o múltiplos de bytes. Todos los comandos o flujos de datos son enviados en bytes, alineados con respecto a la señal de selección de circuito y la señal de reloj. Para todos los comandos existe una respuesta, que indica si existe algún error de lectura de comando.

Este tipo de protocolo de comunicación para la memoria se selecciona a la hora de inicializar la tarjeta. Se deben enviar una serie de comandos para preparar la tarjeta para la escritura y lectura de datos. En la figura 3.13 se observa el algoritmo necesario para inicializar la tarjeta SD.

Como se observa de la figura 3.13, se describe la etapa de inicialización de la tarjeta SD. Este ocurre después la inicialización de la tensión de alimentación. Posterior a esto se envía el `CMD0` con la señal de `CS` en cero lógico (esta señal debe permanecer en bajo por el resto del proceso). Después de este comando se envía el `CMD8`, el cual es obligatorio y específica a la tarjeta el rango de tensión a trabajar. El `CMD58` es opcional en esa etapa y se envía cuando se necesita identificar tarjetas que no trabajan con el rango de tensión designado por el *host*. Los comandos `CMD55` y `ACMD41` se envía con el fin de inicializar la tarjeta. La tarjeta responderá con `R1` si el bit de `'idle state'` esta en uno lógico (lo que significa que la tarjeta se esta inicializando). Cuando el bit de `'idle state'` sea un cero lógico, la tarjeta habrá finalizado la inicialización y es posible seguir con el siguiente

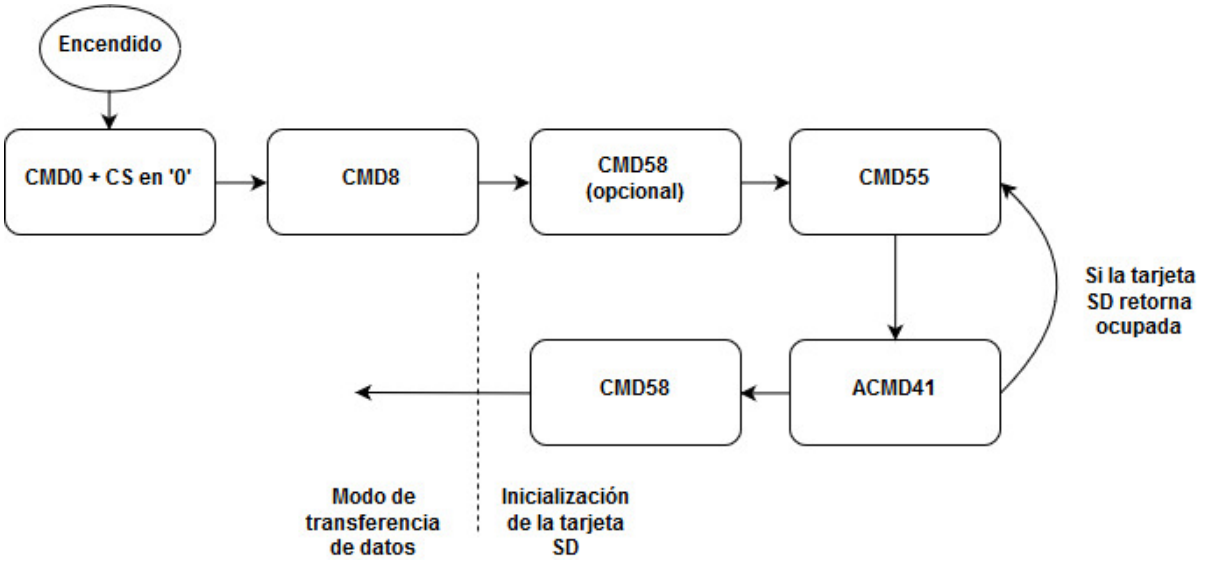

Figura 3.13: Algoritmo de inicialización de tarjeta SD en modo protocolo SPI. [3]

comando. Para finalizar la inicialización de la tarjeta, se envía el comando `CMD58`.

Ahora la tarjeta se encuentra en modo de transferencia de datos. En este modo se permite la lectura o escritura de bloques de memoria, que en el caso de las tarjetas tipo SDHC y SDXC tiene un tamaño de 512 bytes. En otro casos estos son variables y se especifican con el comando `CMD16` o *SET_BLOCK_LEN*.

En la escritura de datos, el protocolo SPI permite la escritura de un bloque único o múltiples bloques simultáneamente. Para ello es necesario el uso de los comandos `CMD24` o `CMD25`. Cuando el comando es enviado, en el caso de escritura de bloque único la tarjeta responderá con R1 sin errores, aceptando el comando. Posteriormente se debe enviar el bloque de datos, teniendo como cabecera un byte con el `Start Block Token`. Después de enviar los datos, se recibe el byte de `Data Response Token` y un estado de ocupado. En la figura 3.14 se observa un diagrama de tiempo con la comunicación de escritura antes descrita.

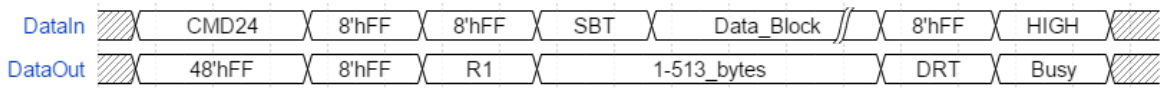

Figura 3.14: Diagrama de tiempo de escritura de bloque único en tarjeta SD en modo protocolo SPI. [3]

En la lectura de datos para este protocolo se permite la lectura de un único bloque o de múltiples bloques. Para realizar estas operaciones se deben utilizar los comandos `CMD17` y `CMD18`, respectivamente. Cuando el comando es enviado, la tarjeta responderá al *host* con `R1`. Posterior, la tarjeta SD enviará el bloque de datos de lectura y código CRC relacionado a los datos enviados, de un tamaño de 16 bits o dos bytes. En la figura 3.15 se observa un diagrama de tiempo con la comunicación de lectura antes descrita.

Figura 3.15: Diagrama de tiempo de lectura de bloque único en tarjeta SD en modo protocolo SPI. [3]

3.1.2. Diseño de interfaz SPI con tarjeta de memoria SD

Para el diseño del interfaz SPI se definen especificaciones básicas, con base a las características físicas de las tarjetas de memoria SD, definidas en la sección 3.1.1. En la tabla 3.5 se resumen las especificaciones básicas de diseño.

Tabla 3.5: Especificaciones de diseño de la comunicación SPI con tarjeta microSD

Característica	Especificación
Frecuencia de SPI	25 MHz
Tamaño de palabra de instrucción	48 bits
Tamaño de palabra de respuesta	8bits, 16 bits, 32 bits y 512 bytes
Modo de operación SPI soportados	Modo 0
Flanco de captura de datos	Flanco negativo de reloj
Algoritmo de inicialización	Sí
Código CRC de seguridad	Obligatorio en comandos de inicialización

En la figura 3.16 se observa el algoritmo de inicialización del interfaz SPI. Este tiene como objetivo inicializar la tarjeta SD, deshabilitar la verificación del código CRC y hacer lectura de bloques específicos de memoria. Estas operaciones conforman la función de *bootstrap* y una vez finalizadas, la unidad habrá realizado su función de arranque.

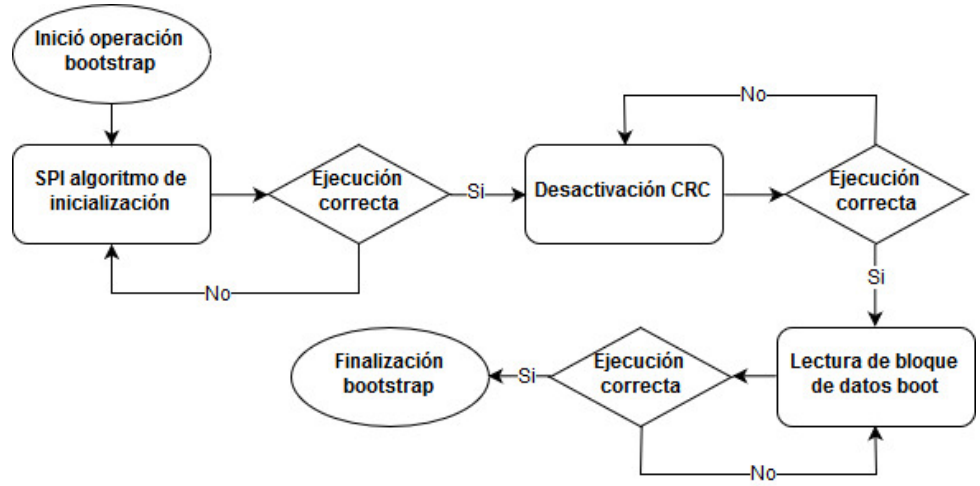

Figura 3.16: Algoritmo de inicialización *bootstrap*, interfaz SPI con tarjeta de memoria SD.

El primer paso es realizar la inicialización de la tarjeta de memoria SD, según el algoritmo descrito en la figura 3.13. Luego se deshabilita en la tarjeta SD la identificación del código CRC. Esto se realiza por dos motivos: el primero de ellos es evitar el aumento de ciclos de reloj para el envío de los comandos, ya que son necesarios 40 ciclos de reloj para obtener dicho código; el segundo es que a través de la respuestas `R1` y `Data Error Token` se puede verificar si la información recibida es correcta. Finalmente, se realiza la lectura de los bloques de memoria de la tarjeta SD.

En la figura 3.17 se logra observar un diagrama general de entradas y salidas del interfaz SPI del *bootstrap* con la tarjeta de memoria SD.

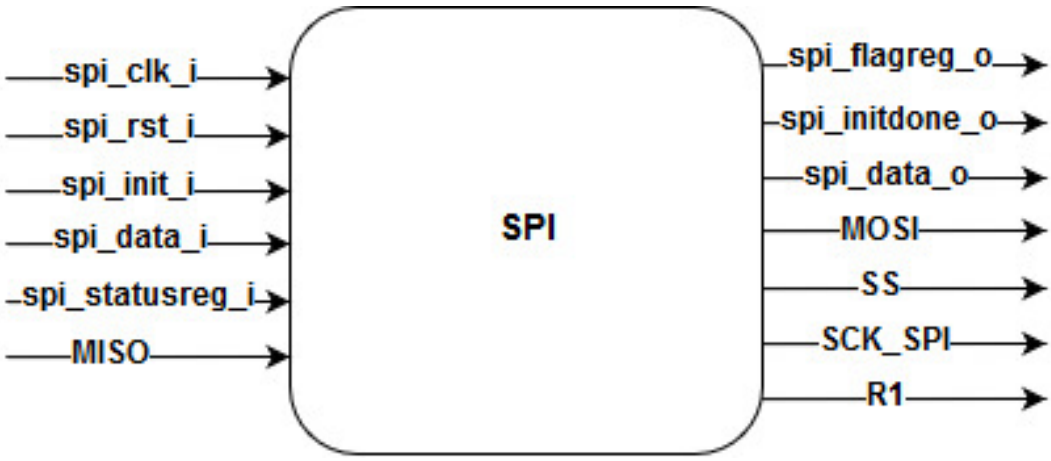

Figura 3.17: Diagrama general de entradas y salidas para el interfaz SPI del *bootstrap* con la tarjeta de memoria SD.

En la tabla 3.6 se caracterizan las señales del interfaz SPI, según el ancho de palabra, tipo de señal y descripción de la señal.

Tabla 3.6: Señales de entrada y salida del interfaz SPI del *bootstrap* con la tarjeta de memoria SD

Señal	Puerto	Tamaño (bits)	Descripción
spi_clk_i	Entrada	1	Señal de reloj
spi_rst_i	Entrada	1	Señal de reset
spi_data_i	Entrada	48	Señal de datos
spi_statusreg_i	Entrada	7	Registro de señales de control
spi_init_i	Entrada	1	Señal de ejecución algoritmo *bootstrap*
MISO	Entrada	1	Señal de datos SPI
spi_flagreg_o	Salida	3	Registro con banderas de salida
spi_initdone_o	Salida	1	Señal finalización de ejecución algoritmo *bootstrap*
spi_data_o	Salida	32	Señal de datos
R1	Salida	8	Registro de código de errores
MOSI	Salida	1	Señal de datos SPI
SCK_SPI	Salida	1	Señal de reloj SPI
SS	Salida	1	Señal de *chip select*

Existen dos señales de control: spi_init_i y spi_statusreg_i. Las señal spi_init_i indica el inicio del algoritmo del bootstrap mostrado en la figura 3.16. La señal spi_statusreg_i es un vector de configuración y estado de 7 bits, donde cada bit controla una característica específica del interfaz SPI. En la tabla 3.7 se describe el formato de este vector.

Tabla 3.7: Formato de la señal spi_statusreg_i del interfaz SPI.

bit 6	HABILITACIÓN INTERFAZ: Habilita el módulo para su ejecución.
bit 5:3	DIVISOR DE RELOJ: Señal que controla la escala de división de frecuencia para SCK_SPI.
bit 2	microSD LECTURA: SPI en modo lectura de tarjeta microSD.
bit 1	microSD ESCRITURA: SPI en modo escritura de tarjeta microSD.
bit 0	OPERACIÓN: Indica el inicio de la operación de envío/recepción de datos.

La señales spi_flagreg_o y spi_initdone_o son banderas, que indica la finalización o comienzo de ciertos procesos. La señal spi_initdone_o indica la finalización del algoritmo del *bootstrap*. La señal spi_flagreg_o está compuesta por banderas que indican el inicio o finalización de actividades de las operaciones. En la tabla 3.8 se muestra el formato y descripción de la señal spi_flagreg_o.

Tabla 3.8: Formato de la señal `spi_flagreg_o` del interfaz SPI.

bit 2	DATOS DE ESCRITURA: En modo de microSD escritura activada, este bit habilita la llegada de datos a enviar
bit 1	PALABRA TERMINADA: Indica la recepción/envío completo de palabra de 32 bits
bit 0	OPERACIÓN TERMINADA: Indica finalización de la operación de envío/recepción

En la figura 3.18 se muestra el diagrama completo del interfaz SPI del *bootstrap* y en la tabla 3.9 se describen las señales internas del interfaz SPI.

Tabla 3.9: Señales internas del interfaz SPI del *bootstrap* con la tarjeta de memoria SD

Señal	Tamaño (bits)	Descripción
`spi_data`	48	Señal de datos
`spi_statusreg`	8	Registro de señales de control
`spi_flagreg`	3	Registro con banderas de salida
`R1`	8	Registro de código de errores

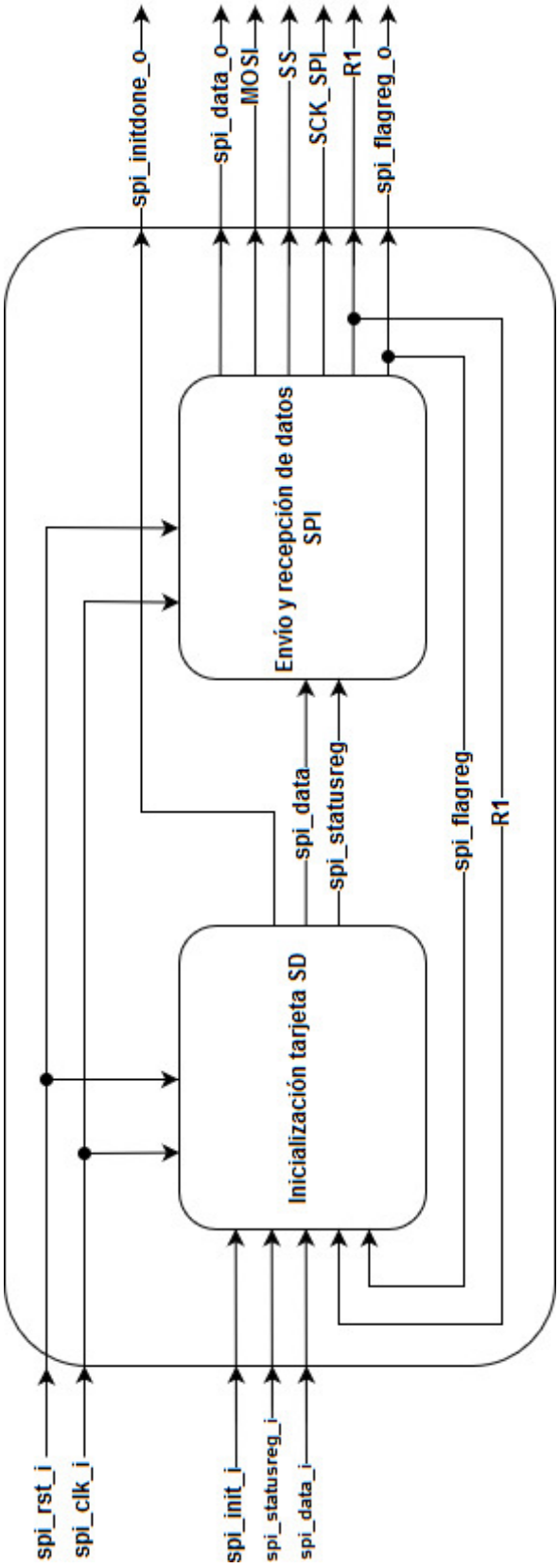

Figura 3.18: Diagrama de bloques de las unidades que componen el interfaz SPI del *bootstrap* con la tarjeta de memoria SD.

De la figura 3.18 se observan dos bloques principales.

- *Inicialización tarjeta SD*: Este bloque es una máquina de estados encargada del cumplimiento del algoritmo SPI de *bootstrap*, enviando los datos y las señales de control necesarias. El módulo se habilita en este modo con la señal `spi_init_i`, de lo contrario los datos que se envían y la señal que controla el envío son `spi_statusreg_i` y `spi_data_i`, respectivamente. La señal `spi_initdone_i` se activa al finalizar el algoritmo.

  En la figura 3.19 se observa el diagrama de estados de control de la máquina de estados del bloque. En el diagrama se ven los valores de salida de las señales internas.

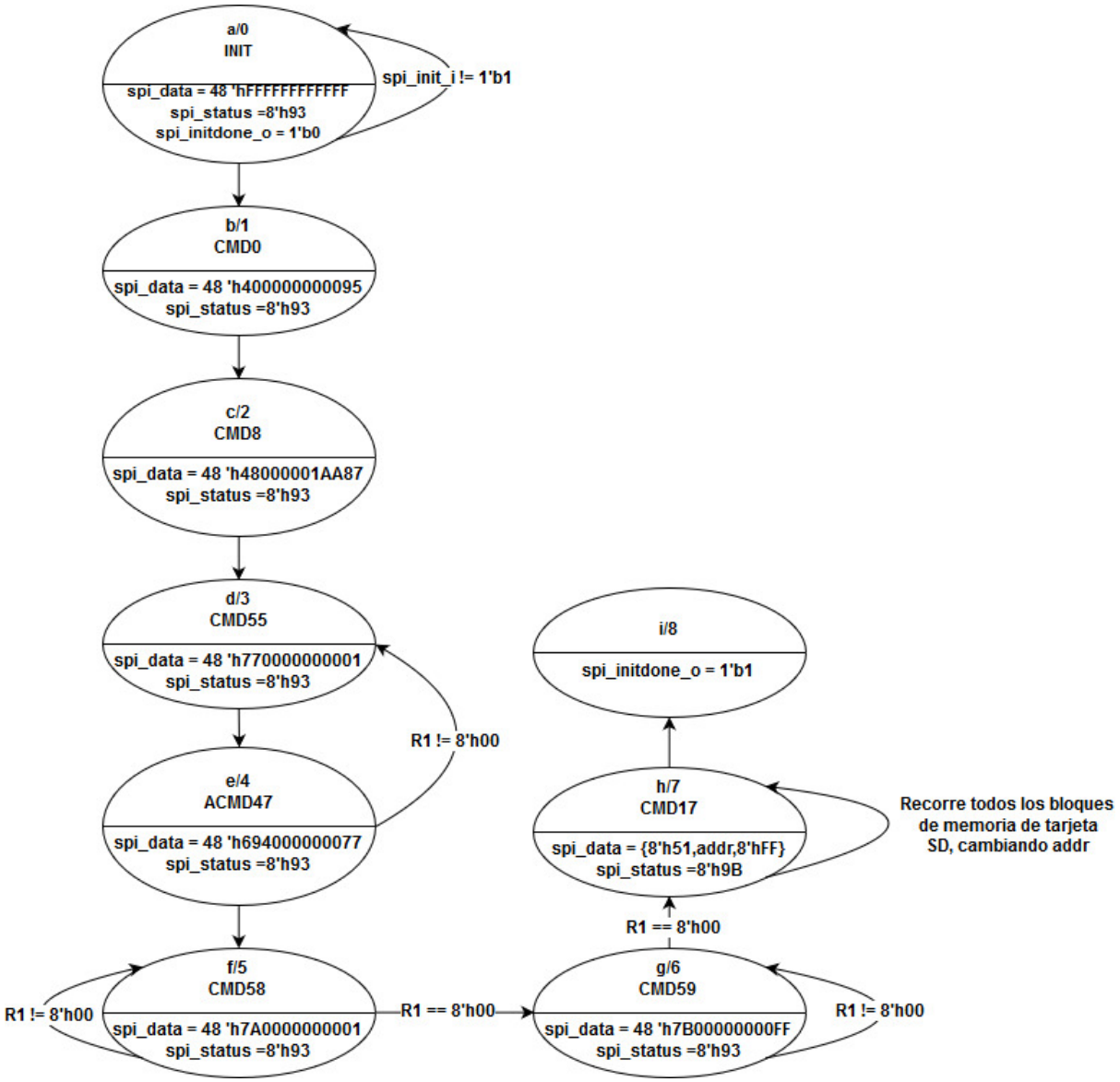

Figura 3.19: Diagrama de estados de control para la inicialización de la tarjeta SD y operación de arranque.

- *Envío y recepción de datos SPI*: Este módulo se encarga del envió y la recepción de los datos a la tarjeta de memoria SD. Tiene la capacidad de esperar respuestas de diferentes tamaños, según el comando enviado. En la recepción o envío de datos lo realiza en palabras con ancho de 32 bits, 128 palabras por operación.

Se realizo la implementación del interfaz de esa manera con el fin de tener un bloque que ejecute de manera autónoma la inicialización y ejecución del algoritmo diseñado para el *bootstrap* a partir de la habilitación del mismo, pero una vez finalizado permita al microprocesador ejecutar operaciones sobre la tarjeta SD.

3.1.3. Verificación del módulo interfaz SPI con tarjeta de memoria SD

Debido a la importancia de los tiempos de bus y el formato de los comandos, se implementó en HDL Verilog una prueba para el módulo de interfaz SPI.

Se realizan diferentes pruebas (*testbench*) de comportamiento a nivel RTL para verificar los características anteriormente descritas.

3.1.3.1. Prueba de tiempos de bus para módulo interfaz SPI con tarjeta de memoria SD

En este *testbench* se verifica el cumplimiento, bajo simulación, de las especificaciones de tiempo para los buses de datos en el protocolo de comunicación SPI.

En la figura 3.20 se observa la simulación obtenida de ModelSim Altera de comportamiento de bus para la señal `MOSI`.

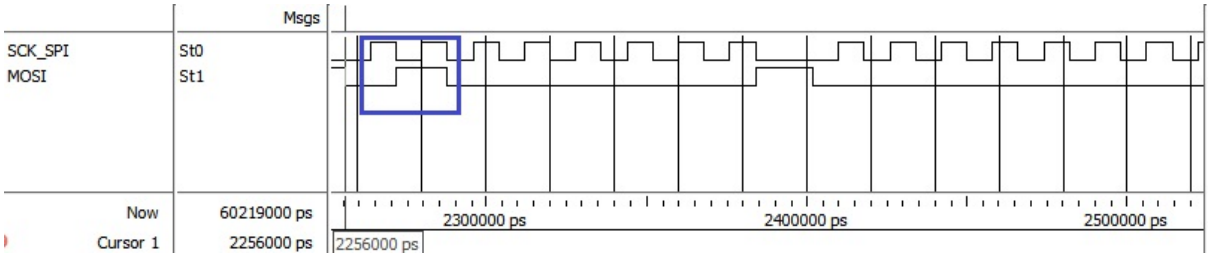

Figura 3.20: Prueba de verificación tipo *testbench* del comportamiento de bus de la señal `MOSI` (Data Input en la tarjeta de memoria SD) para módulo interfaz SPI con tarjeta de memoria SD.

De la figura 3.20 se observa que el dato de salida de la señal MOSI, encerrado en el recuadro, es estable antes del flanco positivo de la señal de reloj y generado en el flanco negativo del ciclo de reloj anterior. Dada esta condición se asegura que el dato cumpla con las especificaciones de bus para las tarjetas de memoria SD, siempre que se trabaje con las frecuencias establecidas para operación estándar u operación de alta velocidad.

En la figura 3.21 se observa la simulación obtenida de ModelSim Altera de comportamiento de bus para la señal `MISO`. En el mismo se observa, en aumento, el cambio de dato en el bus de dato `MISO` y como el mismo es capturado. Dada esta condición se asegura la

captura de datos provenientes de la tarjeta de memoria SD, siempre que se trabaje con las frecuencias establecidas para operación estándar u operación de alta velocidad.

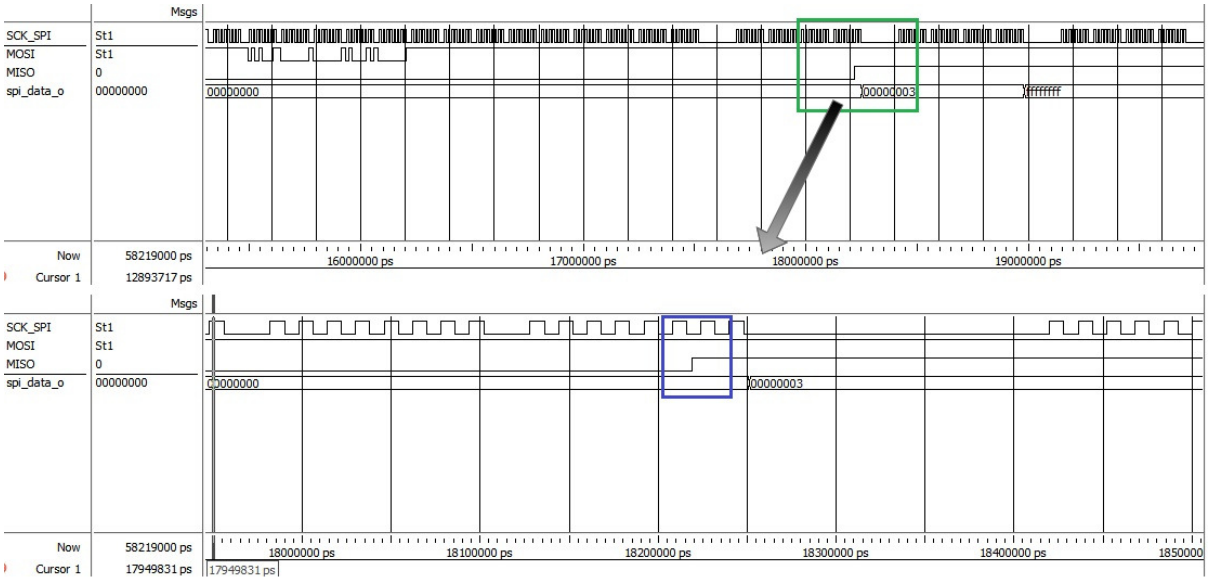

Figura 3.21: Prueba de verificación tipo *testbench* del comportamiento de bus de la señal `MISO` (Data Output en la tarjeta de memoria SD) para módulo interfaz SPI con tarjeta de memoria SD.

3.1.3.2. Prueba de formato de comando para módulo interfaz SPI con tarjeta de memoria SD

En esta prueba se verifica que los comandos sean enviados de manera correcta: el primer bit en transmisión sea el más significativo, cumplan con los tiempos de bus y se envíe el dato correcto.

En la figura 3.22 se observa la simulación del envío del comando 0 o `CMD0`. En dicha figura se observa como el comando se envía en bytes, con un pequeño interludio, donde la señal de reloj esta en cero lógico. Además cumple con las especificaciones mencionadas anteriormente.

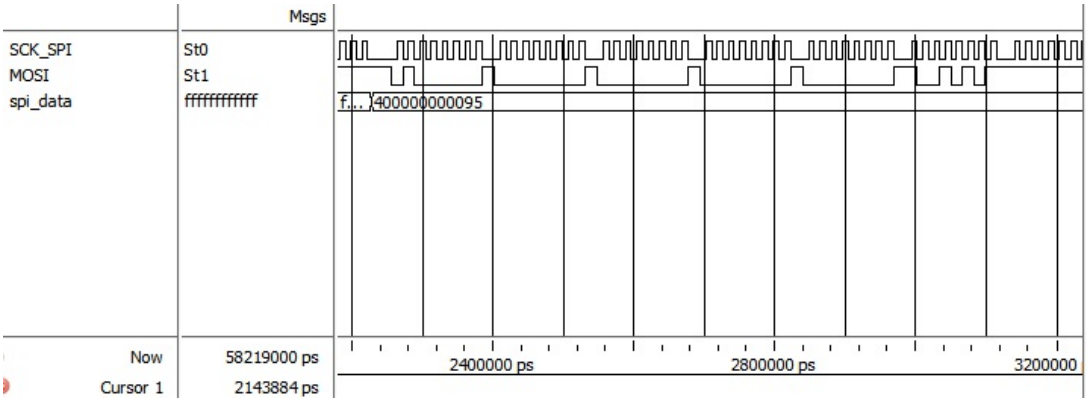

Figura 3.22: Prueba de verificación tipo *testbench* del formato de comando para módulo interfaz SPI con tarjeta de memoria SD.

3.2. Diseño de módulo FIFO

La FIFO implementada para la unidad de *bootstrap* es una FIFO síncrona circular. En la figura 3.23 se observa un diagrama de funcionamiento de los punteros de lectura y escritura, los cuales pueden poseer el mismo valor de dirección o distinto. [12]

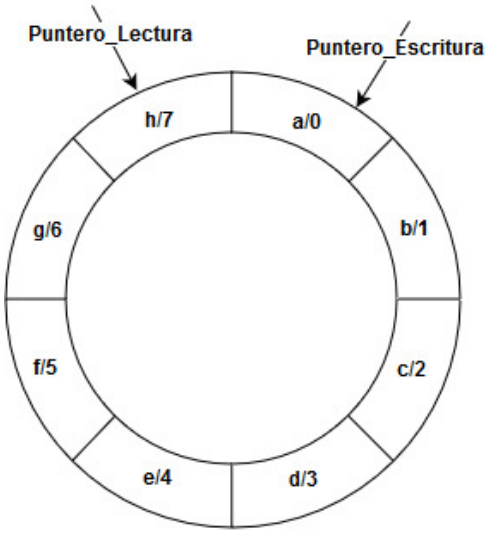

Figura 3.23: Estructura de memoria y punteros en FIFO síncrona circular.

En la figura 3.24 se observa el diseño RTL y el diagrama de entradas y salidas del módulo FIFO.

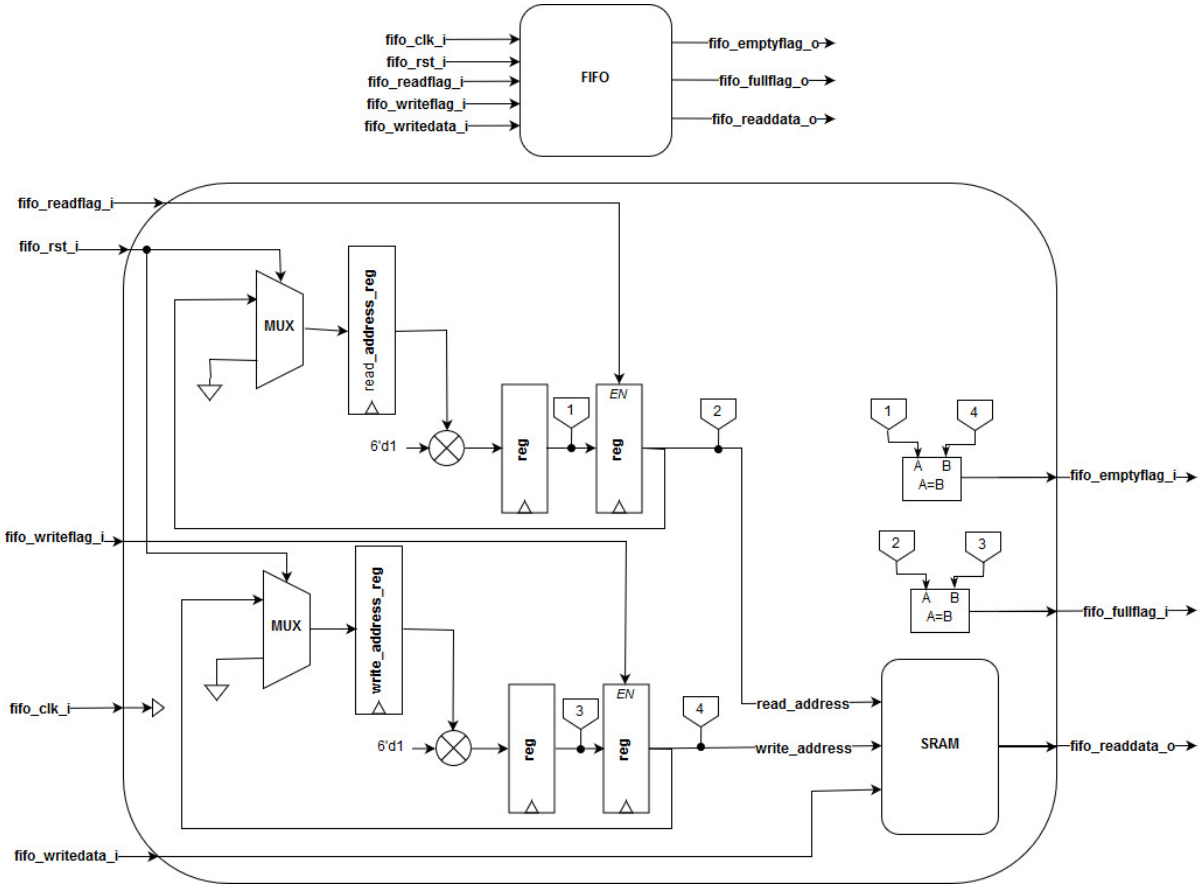

Figura 3.24: Diseño RTL y el diagrama de entradas y salidas del módulo FIFO.

En la tabla 3.10 se encuentra la descripción de las señales del módulo FIFO.

Tabla 3.10: Señales de entrada y salida del módulo FIFO.

Señal	Puerto	Tamaño (bits)	Descripción
`fifo_clk_i`	Entrada	1	Señal de reloj.
`fifo_rst_i`	Entrada	1	Señal de reset.
`fifo_writedata_i`	Entrada	32	Señal de datos.
`fifo_writeflag_i`	Entrada	1	Bandera para escritura de datos.
`fifo_readflag_i`	Entrada	1	Bandera para lectura de datos.
`fifo_readdata_o`	Salida	32	Señal de datos de salida.
`fifo_emptyflag_o`	Salida	1	Bandera de memoria FIFO vacía.
`fifo_fullflag_o`	Salida	1	Bandera de memoria FIFO llena.

El módulo FIFO tiene la señales de control `fifo_writeflag_i` y `fifo_readflag_i`, que controlan cuando se realiza una escritura o lectura de datos de la memoria interna de la FIFO. El tamaño de la memoria interna de la FIFO es de 256 bytes.

La operación de lectura hace uso de dos ciclos de reloj, de igual manera para la escritura. Es necesario el uso de dos ciclos de reloj por la memoria interna del módulo. Esto se debe al diseño secuencial del módulo.

Las operaciones de lectura y escritura son independientes entre sí, debido a la naturaleza de los punteros. Para tener control de no leer espacios de memoria vacíos o escribir cuando la memoria de la FIFO este llena se tiene las banderas de `fifo_emptyflag_i` y `fifo_fullflag_i`, que indican cuando en la FIFO no hay datos o cuando esta llena, respectivamente.

En la figura 3.25 se puede observar el diagrama de tiempo para la lectura de datos.

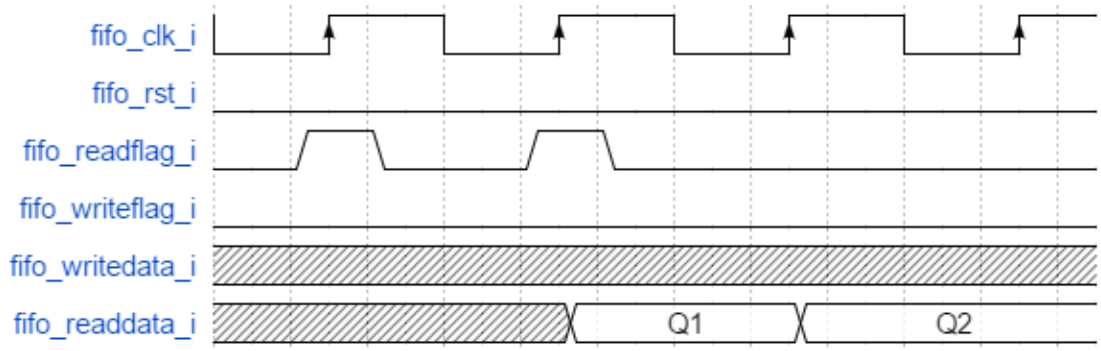

Figura 3.25: Diagrama de tiempo de lectura de datos de FIFO. La lectura del dato se realiza con la identificación de la señal `fifo_readflag_i` en alto en el flanco positivo del reloj y el dato está disponible para lectura en el siguiente flanco positivo del reloj. En este caso se muestra la lectura continua de dos datos de la FIFO.

3.3. Interfaz de control de memoria

El bloque de control de memoria o interfaz de control de memoria es la unidad encargada de la escritura, lectura y multiplexación de señales del *bootstrap* y el microprocesador con

la memoria SRAM de puerto sencillo.

Para el diseño del módulo se debe contemplar las características de la memoria SRAM y del módulo FIFO, ya que uno de ellos es la fuente de los datos y el otro el destino. En esta sección se analiza las características de la memoria SRAM como limitantes de diseño y se especifica las características de diseño del interfaz.

3.3.1. Características de la memoria SRAM de puerto simple

El diagrama de entradas y salidas de la memoria SRAM de puerto sencillo de destino para el programa de instrucciones se muestra en la figura 3.26.

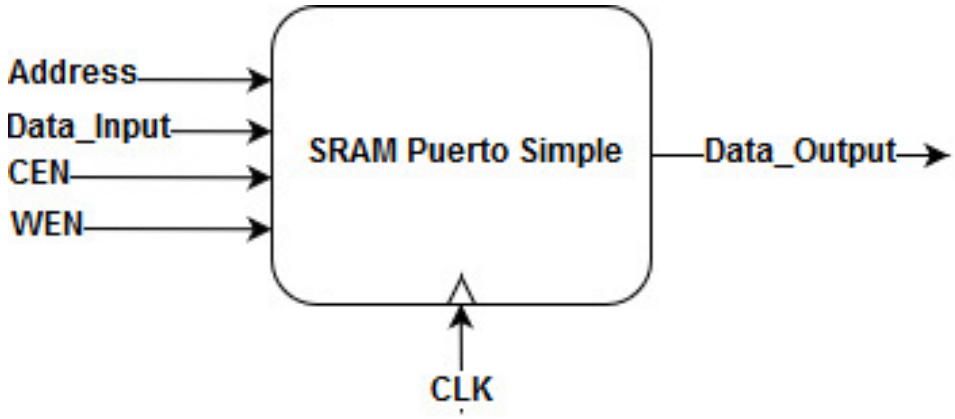

Figura 3.26: Diagrama de entradas y salidas de SRAM de puerto simple para el proyecto SiRPA.

Las señales de la memoria SRAM mostrada en la figura 3.26 se detallan en la tabla 3.11. Estas provienen de la definición de señales de datos de la memoria que se utilizará para el microprocesador de destino de esta unidad.

Tabla 3.11: Señales de entrada y salida de SRAM de puerto simple

Señal	Puerto	Tamaño (bits)	Descripción
`Address`	Entrada	14	Señal de dirección de memoria.
`Data_Input`	Entrada	32	Señal de datos de entrada.
`CEN`	Entrada	1	Señal de habilitación de memoria.
`WEN`	Entrada	1	Señal de habilitación de escritura.
`CLK`	Entrada	1	Señal de reloj.
`Data_Output`	Salida	32	Señal de datos de salida.

Para poder acceder a las operaciones de escritura y lectura en la memoria SRAM, se debe hacer uso de las señales de control `CEN` y `WEN`. Ambas señales activan la escritura o

lectura cuando tienen un valor de cero lógico. En la figura 3.27 se observa un diagrama de tiempo para la lectura de datos.

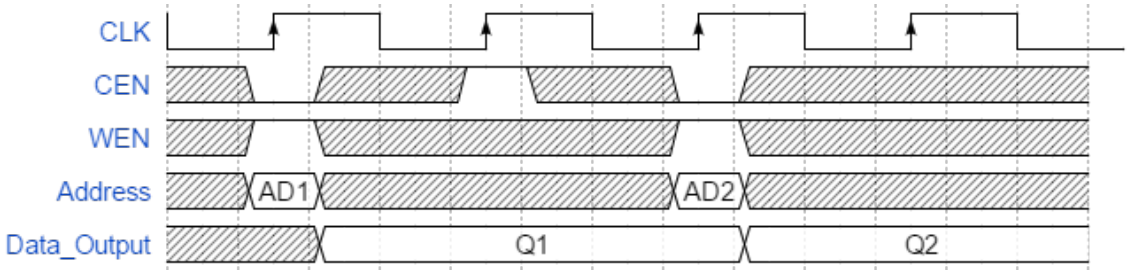

Figura 3.27: Diagrama de tiempo para la lectura de datos en SRAM de puerto sencillo.

En la figura 3.28 se observa un diagrama de tiempo para la escritura de datos en la SRAM de puerto simple.

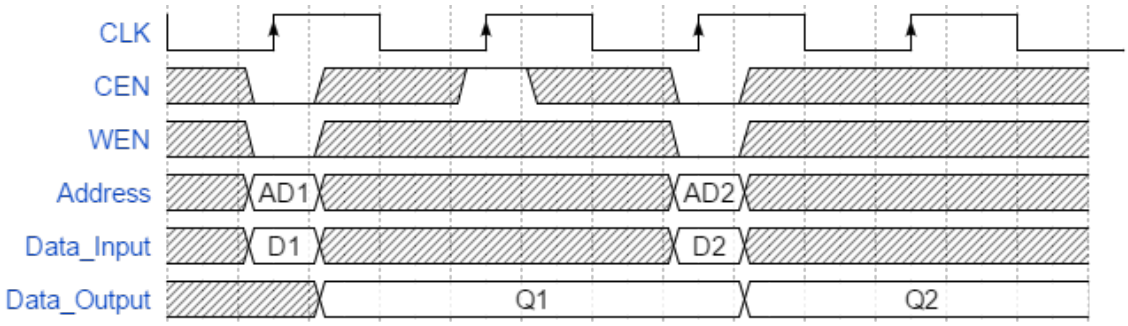

Figura 3.28: Diagrama de tiempo para la escritura de datos en SRAM de puerto sencillo.

3.3.2. Diseño de interfaz de control de memoria

El interfaz de control de memoria tiene como función la lectura de datos desde la FIFO y la escritura de datos en la memoria SRAM o salida de datos, según se especifique. Además, esta unidad debe permitir el acceso a la memoria SRAM al microprocesador.

En la figura 3.29 se muestra un diagrama de señales de entrada y salida del interfaz de control de memoria.

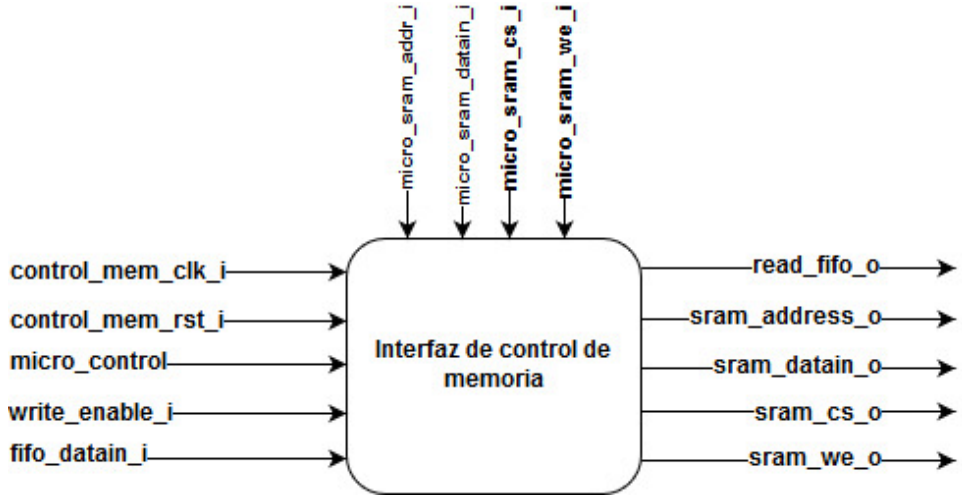

Figura 3.29: Diagrama de entradas y salidas del interfaz de control de memoria.

En la tabla 3.12 se describan las señales de entrada y salida del interfaz controlador de memoria SRAM de puerto simple..

Tabla 3.12: Señales de entrada y salida del interfaz controlador de memoria SRAM de puerto simple.

Señal	Puerto	Tamaño (bits)	Descripción
control_mem_clk_i	Entrada	1	Señal de reloj.
control_mem_rst_i	Entrada	1	Señal de reset.
micro_control	Entrada	1	Señal multiplexación de señales.
write_enable_i	Entrada	2	Señal que habilita modo de escritura.
fifo_datain_i	Entrada	32	Señal de datos.
micro_sram_addr_i	Entrada	13	Señal de datos.
micro_sram_datain_i	Entrada	32	Señal de datos.
micro_sram_cs_i	Entrada	1	Señal de control de memoria.
micro_sram_we_i	Entrada	1	Señal de control de memoria.
read_fifo_i	Salida	1	Señal de control de FIFO.
sram_addr_i	Entrada	13	Señal de datos.
sram_datain_i	Entrada	32	Señal de datos.
sram_cs_i	Entrada	1	Señal de control de memoria.
sram_we_i	Entrada	1	Señal de control de memoria.

Las señales `micro_sram_addr_i`, `micro_sram_datain_i`, `micro_sram_cs_i` y `micro_sram_we_i` son señales de control, provenientes del microprocesador, para control la memoria SRAM.

La multiplexación se realiza con la señal `micro_control`, que controlará si la memoria SRAM será controlada por la unidad *bootstrap* o por el microprocesador.

Todo dato obtenido por el interfaz SPI, será almacenado de forma intermedia en el módulo FIFO. El interfaz controlador de memoria es el único módulo capaz de ejecutar la lectura de datos del módulo FIFO.

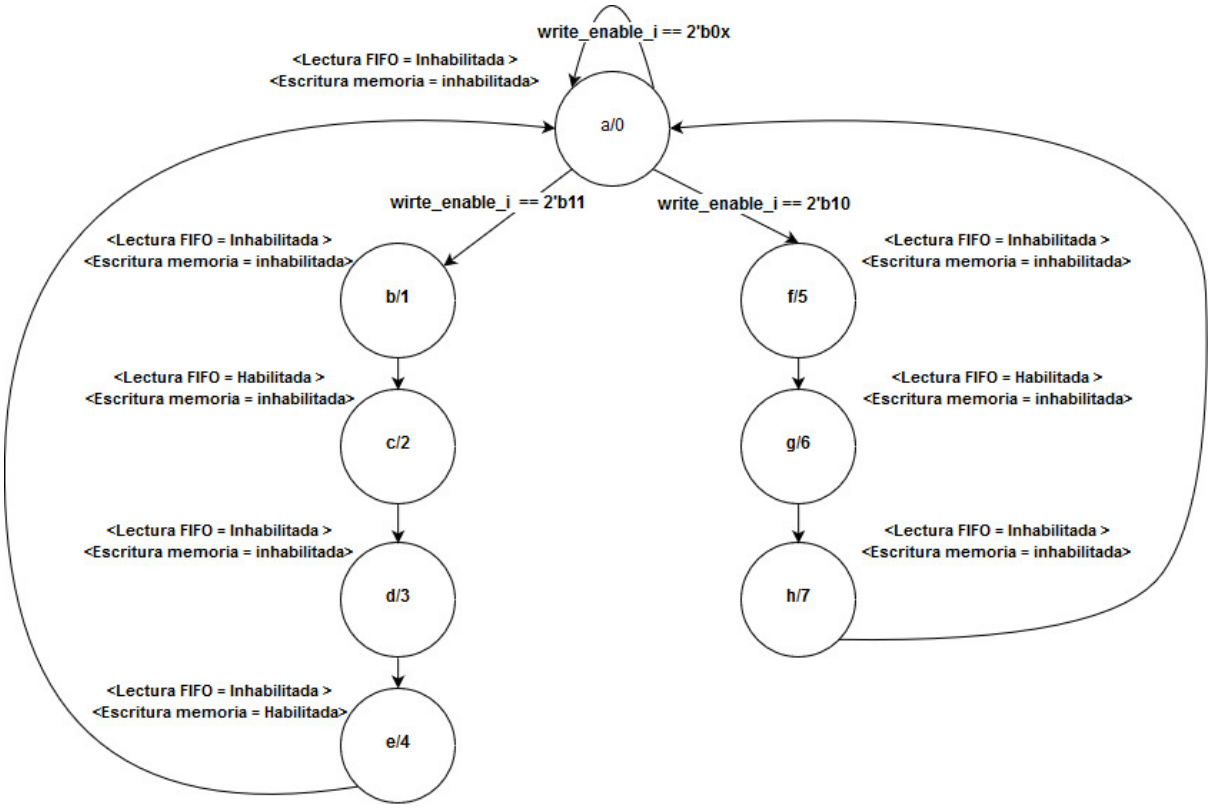

Figura 3.30: Diagrama de estados del controlador de procesamientos de datos y escritura en memoria SRAM.

En la figura 3.30 se observa el algoritmo de control de la FSM de obtención de datos de la FIFO y escritura en memoria SRAM. La señal `write_enable_i` habilita dos modos de salida de datos para la operación de lectura en el módulo interfaz SPI. Uno de ellos, habilita la lectura del dato en el módulo FIFO y la escritura en la memoria SRAM. Ambas operaciones se realizan en diferentes ciclos de reloj, debido a que para asegurar la integridad del dato desde el módulo FIFO es necesario dos ciclos de reloj, por su comportamiento secuencial, y para la escritura en la memoria SRAM son necesarios otros dos ciclos de reloj.

El segundo modo de operación solo incluye la lectura del dato desde el módulo FIFO y su salida por la señal `sram_datain_i`. No conlleva escritura en memoria.

En la figura 3.31 se observa el diseño RTL del controlador de memoria. Se observan dos

principales unidades de multiplexación. Una primera etapa de multiplexación se encarga de asignar los valores de las señales según el modo de operación habilitado por la señal `write_enable_i`. Una segunda etapa de multiplexación determina si la memoria es controlada por el interfaz de control de memoria o directamente por el microprocesador. Esta segunda etapa tiene como señal de selección a `micro_control`. La implementación de esta segunda etapa de multiplexación se realiza para que ya ejecutada la operación de carga de programa de instrucciones, el microprocesador tome el control de la SRAM.

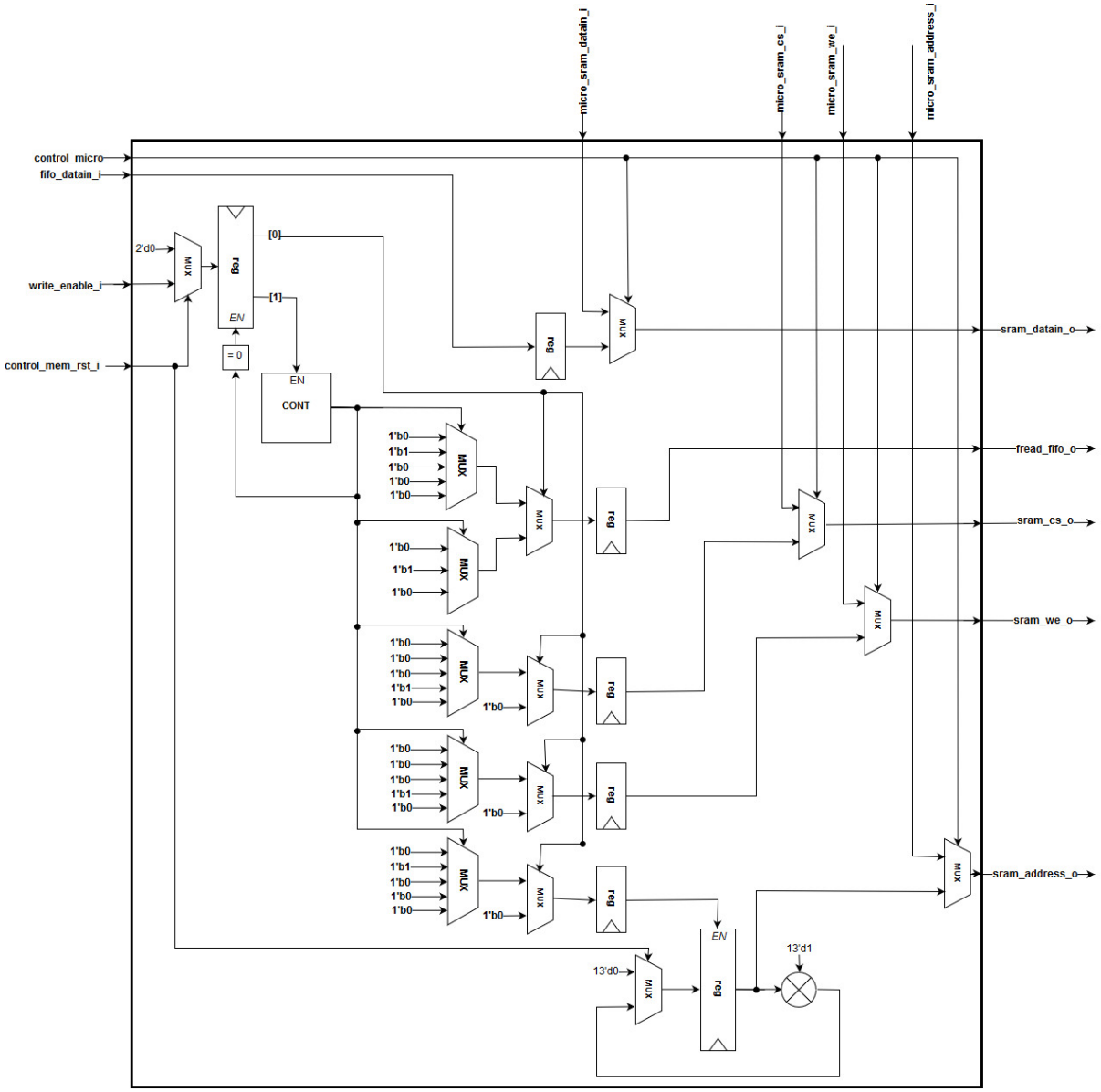

Figura 3.31: Diseño RTL del interfaz de control de memoria SRAM. Tiene dos modos de operación, controlados por la señal `write_enable_i`. Este diseño tiene una doble etapa de multiplexación, la primera identifica el modo de operación y la segunda el controlador de las señales de salida hacia la memoria SRAM.

Capítulo 4

Comprobación de de la unidad *bootstrap* sobre una FPGA

En este capítulo se detalla el desarrollo de la prueba funcional de la unidad de *bootstrap* en un dispositivo programable FPGA. Se utilizó la tarjeta de desarrollo de Nexys 4 de Digilent Inc. que posee una FPGA Artyx 7 (ver más datos de la misma en [11]). Para esto se desarrolla un módulo en lenguaje Verilog para la verificación de la unidad, en el que se emula el control de la unidad y la memoria de destino de datos.

4.1. Descripción del módulo para la verificación

La unidad de *bootstrap* se ha detallado en el capítulo 3 y en esta sección se verá como parte de la unidad bajo prueba (UUT). El módulo de prueba a implementar en la FPGA esta compuesta por el controlador de la unidad *bootstrap*, la instanciación de una SRAM y la instanciación de la unidad *bootstrap*.

En la figura 4.1 se observa el módulo de prueba completo.

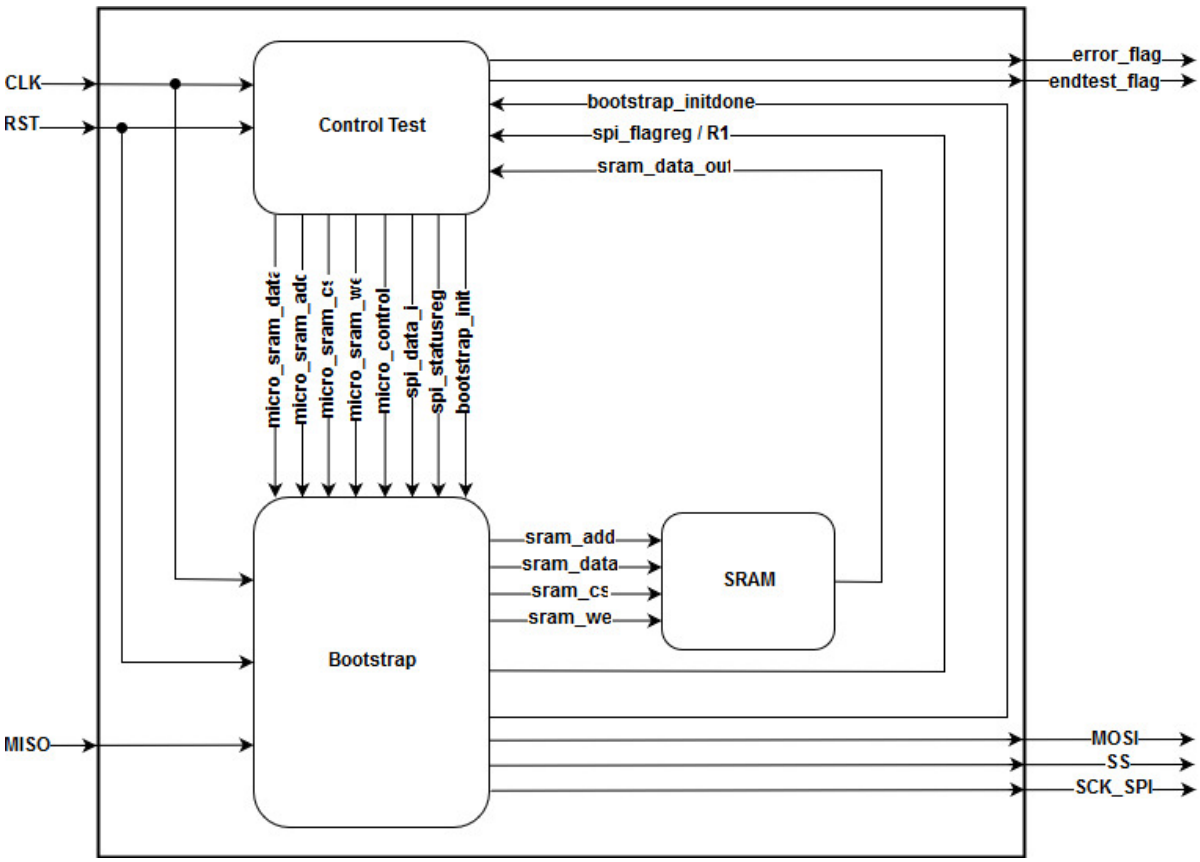

Figura 4.1: Módulo de prueba completa de unidad *bootstrap*. Esta se compone del control de la unidad *bootstrap*, la instanciación de una memoria SRAM y de la unidad *bootstrap*.

De la figura 4.1 se pueden observar tres módulos: `Bootstrap`, `SRAM` y `Control Test`. El primero es la instanciación unidad de *bootstrap* que se pondrá a prueba.

El bloque `SRAM` es un arreglo de registros multiplexados de 8192 direcciones y tamaño de palabra de 32bits, implementado a través de la descripción a nivel de comportamiento en Verilog de Vivado.

El módulo `Control Test` es una máquina de estados que controla un circuito diseñado para realizar la prueba de verificación de la unidad *bootstrap*. El diagrama de estados de la máquina de estados del módulo `Control Test` se observa en la figura 4.2.

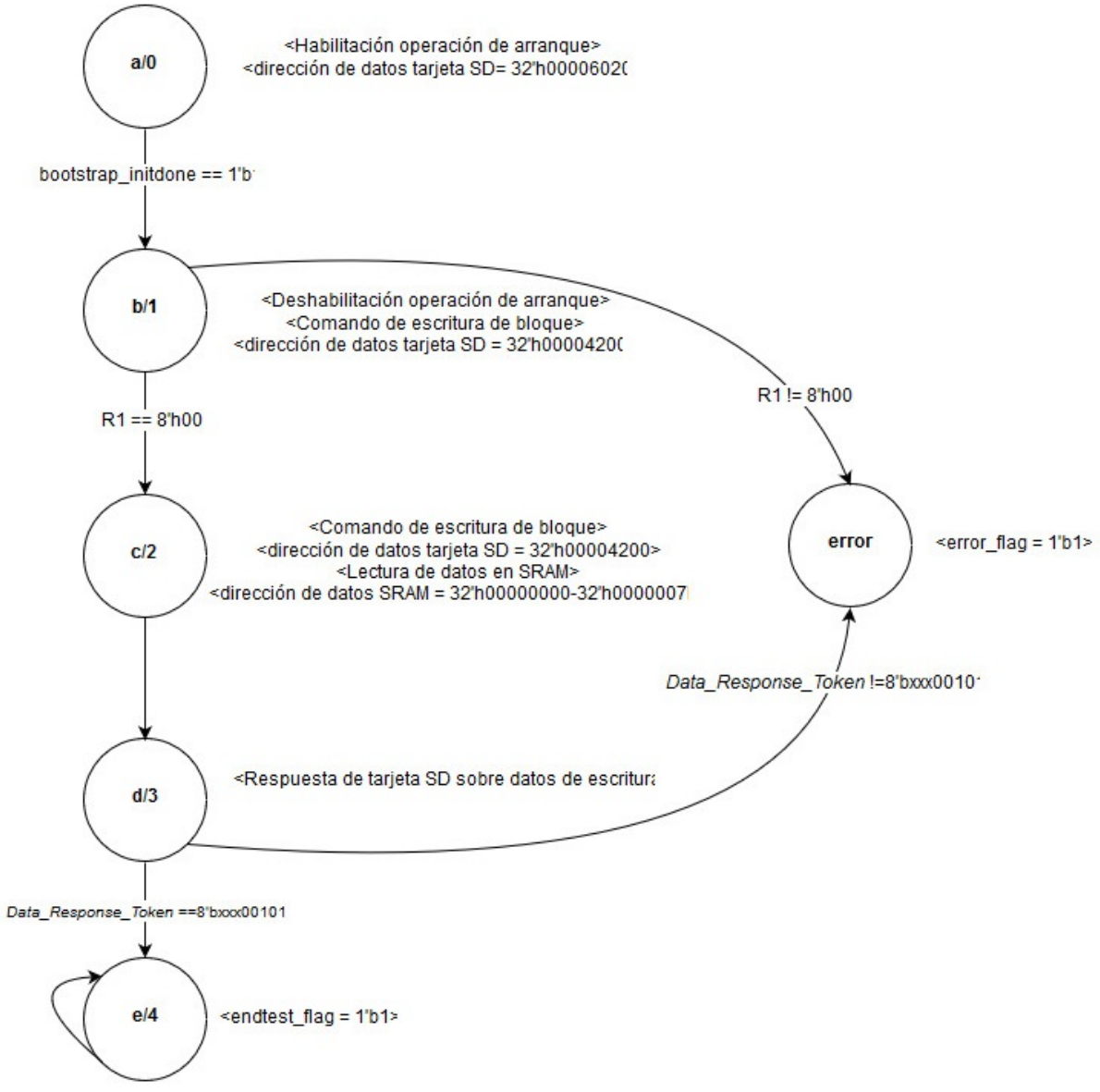

Figura 4.2: Diagrama de estados del módulo `Control Test`. Este representa la ejecución de la prueba para evaluar la unidad *bootstrap*.

Además, en la figura 4.3 se observa un diagrama con el flujo de datos y señales de control entre la tarjeta de memoria SD, la unidad *bootstrap* y la memoria estática SRAM. Según

el estado del módulo `Control Test` se observa un flujo distinto de datos. El *bootstrap* y la SRAM están implementados en la Nexys 4. Con base a la figura se observa que todos los datos deben ser procesados por la unidad diseñada, lo que permite evaluar el funcionamiento de la misma, permitiendo conocer si cumple con las especificaciones de diseño.

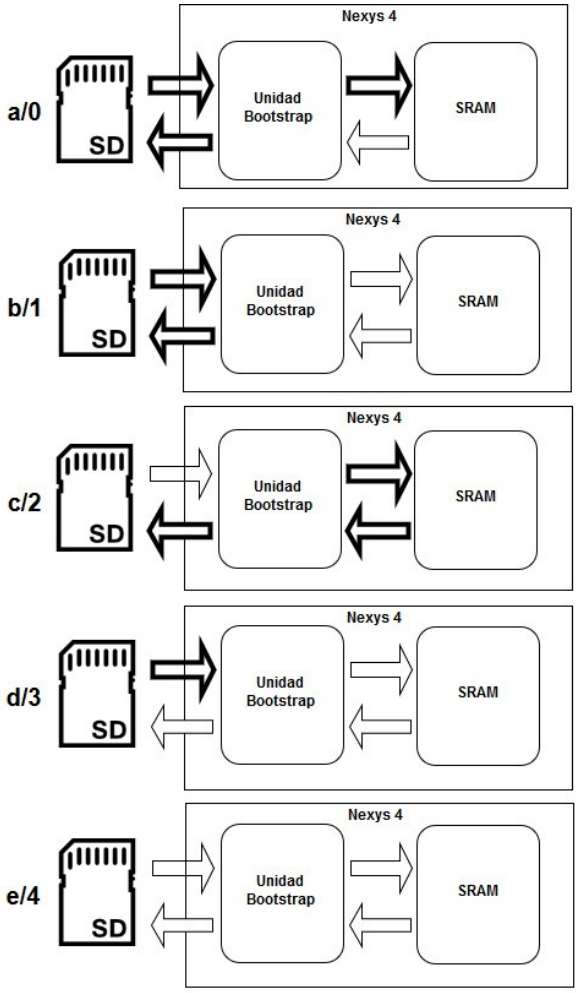

Figura 4.3: Diagrama de flujo de datos entre la tarjeta de memoria SD y la unidad *bootstrap* y memoria SRAM (estas últimas implementadas en la Nexys 4), según el estado de control de `Control Test`.

En la tabla 4.1 se describen las entradas y salidas del módulo para la verificación de la unidad *bootstrap*.

Tabla 4.1: Señales de entradas y salidas del módulo para la verificación de la unidad *bootstrap*

Señal	Puerto	Tamaño(bits)	Descripción
`CLK`	Entrada	1	Señal de reloj
`RST`	Entrada	1	Señal de reset.
`MISO`	Entrada	1	Señal SPI/microSD
`MOSI`	Salida	1	Señal SPI/microSD
`SS`	Salida	1	Señal SPI/microSD
`SCK_SPI`	Salida	1	Señal SPI/microSD
`error_flag`	Salida	1	Señal de error
`endtest_flag`	Salida	1	Señal de finalización

4.2. Simulaciones para el módulo de verificación

Definido el módulo de verificación a utilizar, se procedió a realizar una simulación de comportamiento de la misma.

En este se busca que cumpla con el flujo de prueba para el cual fue diseñado, pasando por inicialización de la tarjeta de memoria SD, operación de carga de instrucciones, lectura de memoria SRAM y escritura de bloque de datos en la tarjeta de memoria SD.

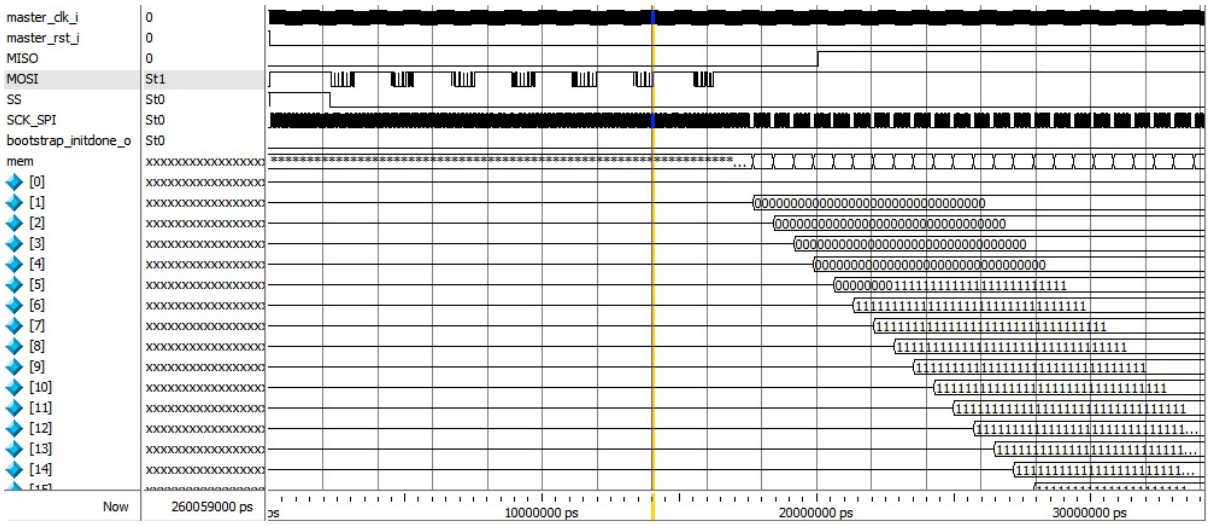

Figura 4.4: Simulación del ambiente de verificación completo de la unidad *bootstrap*, inicialización de tarjeta de memoria SD y carga de programa bloque de datos.

En la figura 4.4 se observa la parte de inicialización de la tarjeta de memoria SD, lectura de un bloque de datos y la escritura de datos en la memoria de la SRAM en la simulación del módulo de verificación completa de la unidad *bootstrap* en ModelSim Altera.

Figura 4.5: Simulación del ambiente de verificación completo de la unidad *bootstrap*, escritura de datos en tarjeta de memoria SD.

En la figura 4.5 se observa la parte de escritura de datos en tarjeta de memoria SD de la simulación del módulo de verificación completa de la unidad *bootstrap* en Vivado.

Esta simulación permite corroborar una correcta sintaxis en el diseño a nivel de transferencia de registros y de comportamiento referente al algoritmo que se desea que cumpla. Sin embargo, este tipo de simulación no refleja el comportamiento real de la unidad y del módulo de verificación en la FPGA. Es necesario realizar simulaciones que tomen en consideración especificaciones de síntesis, ruteo, posicionamiento y retardos de tiempos. En la figura 4.6 se observa el comando de inicialización `ACMD41` en la simulación post-implementación de tiempo, en el cual se consideran las características antes mencionadas. Se observa de la figura como se cumple con el modo cero de comunicación SPI, con el cual trabaja las tarjetas de memoria SD.

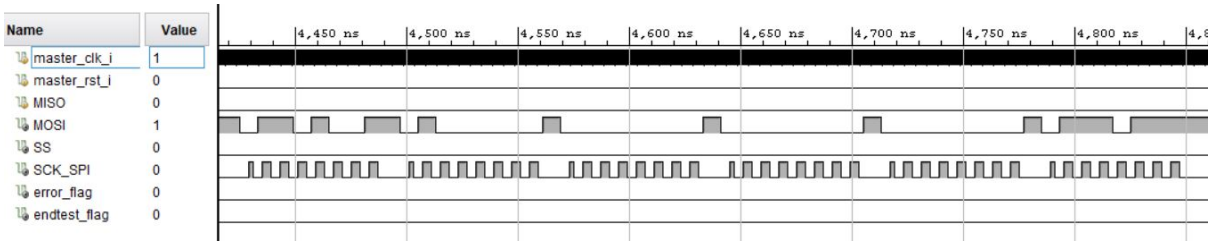

Figura 4.6: Simulación post-implementación de tiempo del ambiente de verificación completo de la unidad *bootstrap*, visualización comando `ACMD41`.

En la figura 4.7 se observa todos los comandos de inicialización de la tarjeta de memoria SD en la simulación post-implementación de tiempo en la herramienta Vivado. Si se compara con respecto a la figura 4.5, se observa que guarda un comportamiento similar y sabiendo que la generación de los comandos son iguales, se puede decir que ambos flujos de datos son iguales.

Figura 4.7: Simulación post-implementación de tiempo del ambiente de verificación completo de la unidad *bootstrap*, visualización comandos de inicialización.

4.3. Resultados sobre una placa de desarrollo Nexys 4

En la sección 4.2 se verificó por simulación el funcionamiento de la unidad *bootstrap*, donde los tiempos de los buses concuerdan con las especificaciones físicas y logrando una escritura de memoria sin errores. A partir de esos resultados se concluye que la unidad es capaz de establecer comunicación con la tarjeta de memoria y realizar las operaciones de lectura/escritura del módulo de memoria.

Para la comprobación de la unidad mediante la implementación en la Nexys 4 se realiza una prueba que requiere la ejecución de todas las funciones de la unidad *bootstrap*. Esta consta de realizar la inicialización de la tarjeta de memoria SD y lectura de un bloque de datos. Posteriormente, se deberá realizar la escritura de este bloque en el módulo SRAM. Una vez finalizado, se escribirá en un bloque de memoria específico la información leída en el primer paso. Se considera que la unidad ejecuta sus funciones correctamente si se realiza una copia del bloque de datos original.

El reloj maestro con el que trabaja el módulo de verificación será el reloj de la Nexys 4, que es de 100 MHz. El protocolo SPI con la tarjeta de memoria SD tendrá un reloj de sincronización de velocidad estándar, es decir, 25 MHz.

En primer instancia se realiza una verificación de la señal generada por la FPGA y la obtenida por simulación. En la figura 4.8 se muestra el comando `ACMD41` enviado como parte del proceso de inicialización de la tarjeta de memoria SD. Comparada con la figura 4.6 se puede determinar que las señal generada por la Nexys 4 cumplen los criterios de diseño, ya que ambas señales son iguales.

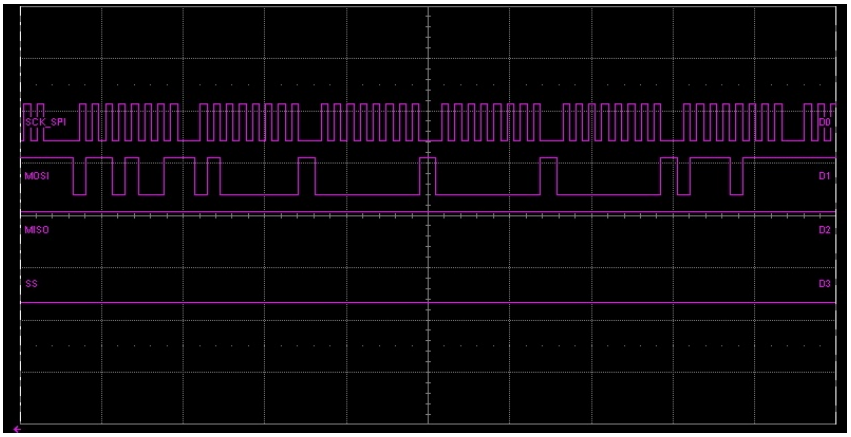

Figura 4.8: Comando `ACMD41` de visualización de tarjeta de memoria SD generado por la FPGA en la placa Nexys 4. Tomado del osciloscopio LeCroy WaveSurfer 64Xs-A.

Además se realiza la verificación de los tiempos de transición del reloj SPI de cero lógico a uno lógico (flanco positivo) y de uno lógico a cero lógico(flanco negativo). En la figura 4.9 se observa la medición de la señal en el osciloscopio LeCroy WaveSurfer 64Xs-A. De esta se obtiene que la frecuencia a la que oscila la señal de reloj es de 25,144 MHz, el tiempo de transición en el flanco positivo es de 7,148 ns y el tiempo de transición en el flanco negativo es de 7,002 ns. También se realiza la medición del tiempo entre el dato de la línea `MOSI` y el flanco positivo del reloj. Este valor es de 22.4 ns. Con base a lo anterior se puede decir que la unidad implementada en la FPGA cumple con los tiempos de bus de la tarjeta de memoria SD.

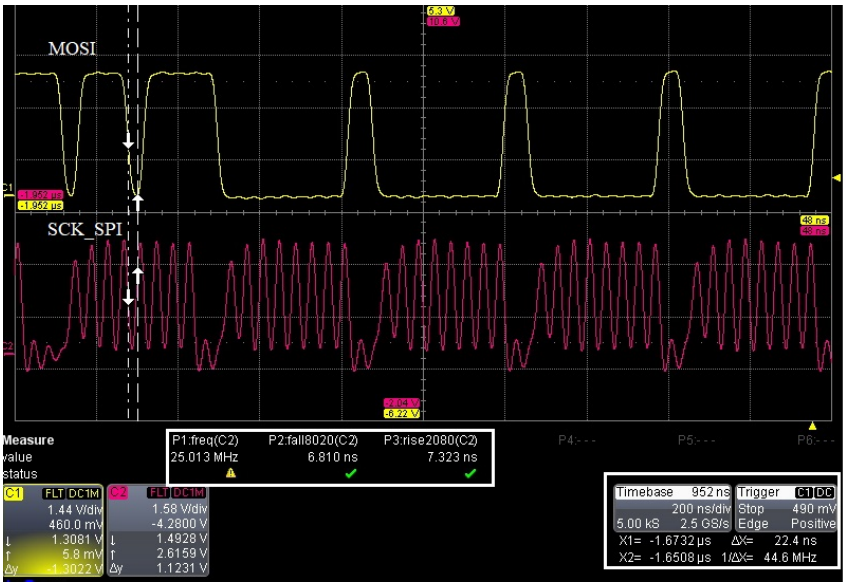

Figura 4.9: Verificación de los tiempos de transición, frecuencia y *set up time* del reloj SPI y la línea `MOSI`. Encerrados en un recuadro se encuentra las mediciones obtenidas por el osciloscopio LeCroy WaveSurfer 64Xs-A.

La segunda verificación es la ejecución de la prueba ya descrita en la figura 4.2. En la figura 4.10 se observa el flujo de prueba completo para la verificación, que incluye la verificación de los bloques de memoria en la tarjeta de memoria SD y la ejecución de la prueba implementada en el módulo `Control Test`.

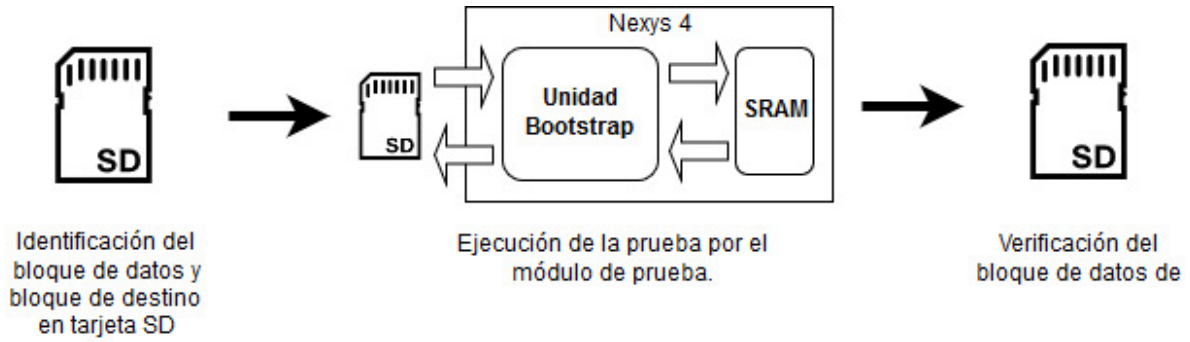

Figura 4.10: Flujo de prueba de verificación de la unidad *bootstrap* implementada en la placa Nexys 4.

La tarjeta utilizada para la prueba es una microSD, tipo SDHC y 4GB de almacenamiento. Dentro de la misma se almacena un archivo de extensión *.txt* con un contenido de 81 bytes. En la figura 4.11, se observa el bloque de datos observado por el programa WinHex, donde se detalla la dirección física (32'h00006020) donde se encuentra el archivo y su contenido.

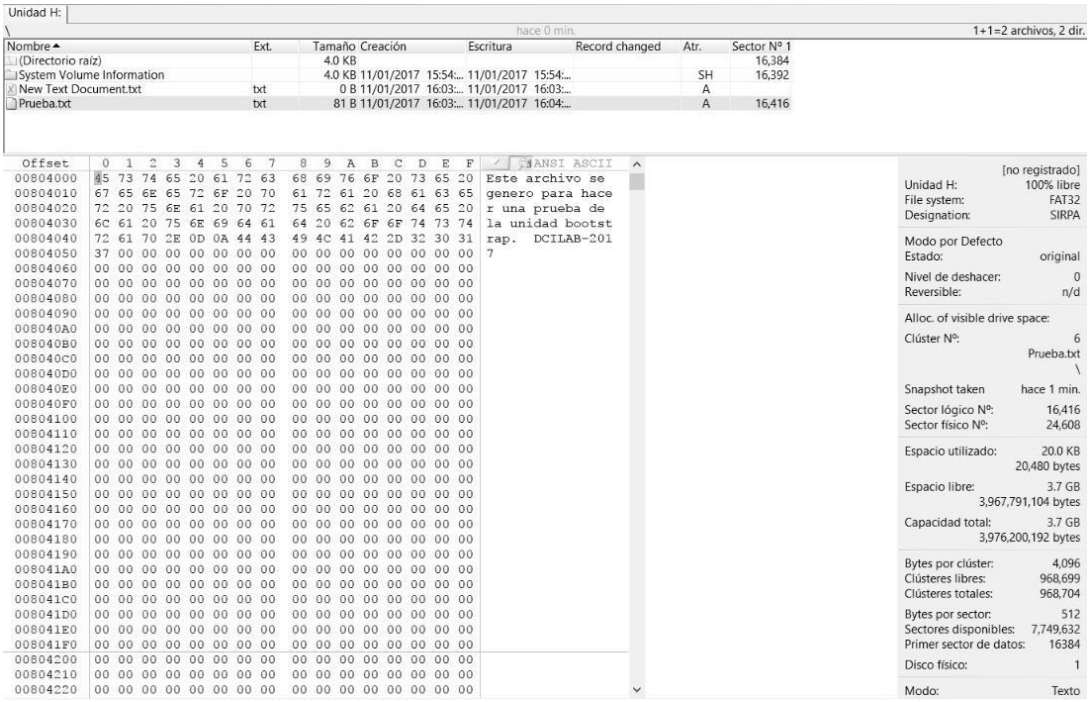

Figura 4.11: Archivo de prueba en el mapa físico de memoria de tarjeta microSD tipo SDHC. Tomado de WinHex.

En la figura 4.12 se observa el bloque de memoria que se destina para escribir la copia

de datos antes de realizar la prueba. La ubicación se escogió bajo el criterio de que fuera un espacio de escritura disponible en la tarjeta. La ubicación determinada para escritura fue 32'h00004200.

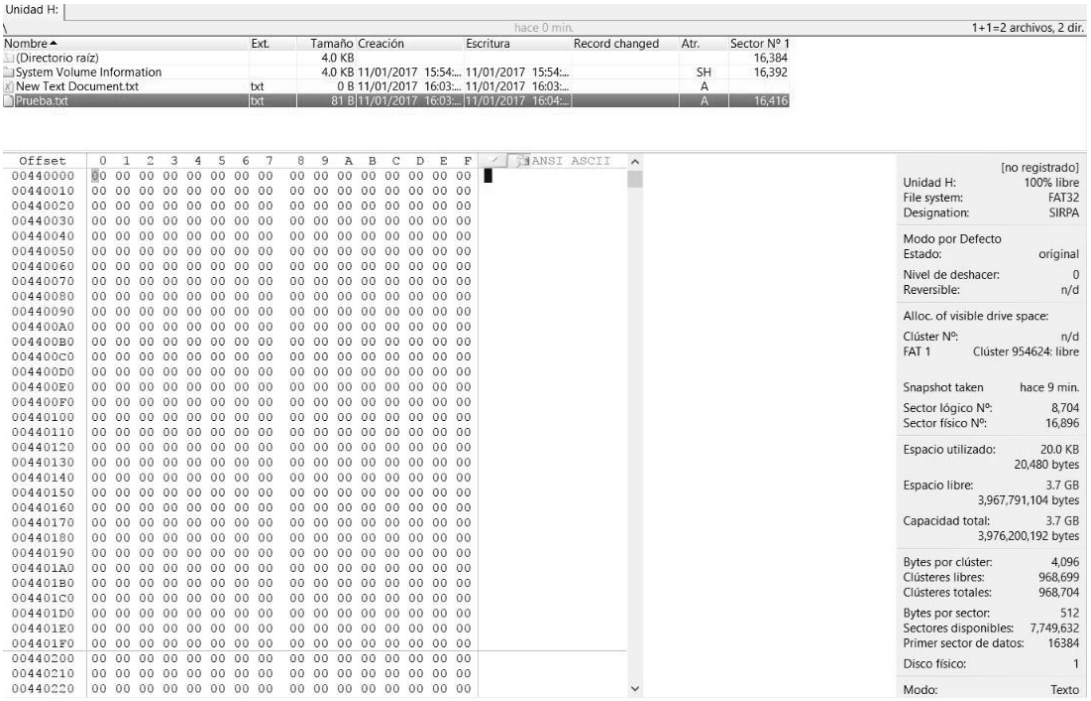

Figura 4.12: Bloque de escritura para prueba de la unidad *bootstrap* en el mapa físico de memoria de tarjeta microSD tipo SDHC antes de la prueba. Tomado de WinHex.

Una vez implementado el módulo de verificación de la unidad en la Nexys 4, se realiza la prueba. Finalizada la misma, se realiza la verificación de escritura en el bloque de datos de la tarjeta microSD, en el espacio de memoria ya especificado. En la figura 4.13 se observa el contenido en el bloque de datos.

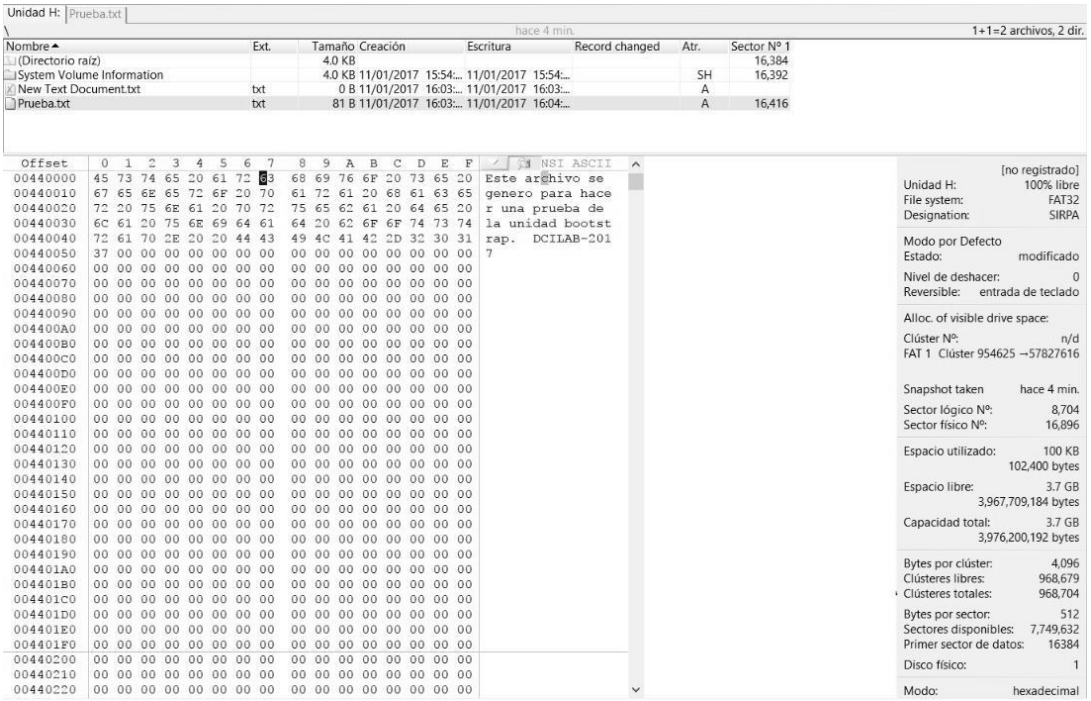

Figura 4.13: Bloque de escritura para prueba de la unidad *bootstrap* en el mapa físico de memoria de tarjeta microSD tipo SDHC después de la prueba. Se puede observar, comparado con la figura 4.12 que el bloque no se encuentra vacío, tiene escrito los datos del bloque 32'h00006020, observados en la figura 4.11. Tomado de WinHex.

A partir de lo visto en las figuras 4.11, 4.12 y 4.13 se concluye que la unidad *bootstrap* realiza correctamenta la prueba establecida para su verificación, esto debido a que genera una copia de datos entre bloques de la tarjeta de memoria microSD. Para realizar la copia fue necesario de la ejecución de todas las funciones de la unidad: inicialización de tarjeta SD, carga de programa de instrucciones, escritura en SRAM de puerto simple, multiplexación de señales para control de memoria por el microprocesador (en este caso simulado por el módulo de verificación) y control de la unidad SPI para la escritura en tarjeta SD.

4.4. Reporte de recursos utilizados

En la tabla 4.2 se puede observar un resumen del reporte de los recursos utilizados por el módulo de verificación en la Nexys 4, generado por la herramienta Vivado.

Tabla 4.2: Recursos utilizados por módulo de verificación de unidad *bootstrap*

Recurso	Utilizado	Disponible	Porcentaje utilizado
LUT	424	63400	0,67 %
LUT RAM	76	19000	0,40 %
FF	239	126800	0,19 %
BRAM	8	135	5,93 %
IO	8	210	3,81 %

Este reporte incluye los recursos utilizados por el módulo de verificación de la unidad, donde se encuentra instanciada la unidad *bootstrap*. Por esta razón el uso de entradas y salidas solo significan el 3,81 % de los recursos disponibles, debido a que solo se hace uso de seis señales, aunque la unidad *bootstrap* tenga señales de salida de 32 bits.

La mayor utilización de recursos se encuentra en los recursos BRAM y IO, con 5,93 % y 3,81 %, respectivamente. Estos recursos se utilizan para las unidades memoria y puertos de salidas-entradas.

4.5. Reporte de tiempos

Conocer la viabilidad del diseño en términos de limitaciones de tiempo permite saber si este es capaz de alcanzar todas las restricciones establecidas por un usuario. La herramienta Vivado provee un reporte de tiempos donde se analiza las rutas con mayor retraso, con lo cual se puede analizar el slack.

El slack se define como la diferencia entre el tiempo actual y el tiempo deseado para una ruta de tiempos, lo cual define a que frecuencia puede o no operar el circuito. Se divide en dos tipos; el *setup slack* que es el tiempo de setup por el dato menos el tiempo de arribo por el dato y *hold slack* que es el tiempo de arribo del dato menos el tiempo de hold requerido por el dato.

Según los valores obtenidos se puede concluir diferentes características del diseño. Existen tres posibilidades de criterio:

- Ambos valores de slack son positivos, lo que indica que el diseño funciona a la frecuencia especificada y posee un margen disponible.
- Ambos valores de slack son cero, lo que indica que el diseño funciona a la frecuencia especificada, pero no posee margen disponible.
- Al menos un slack es negativo, lo que indica que el diseño no funciona eficientemente a la frecuencia especificada. En este caso se deben acelerar rutas problemáticas.

Para el diseño implementado se muestran los valores obtenidos por la herramienta de Vivado en la tabla 4.3. Como se deduce de los datos mostrados en la tabla, la unidad *bootstrap* funciona eficientemente a esta frecuencia.

Tabla 4.3: Resumen de reporte de tiempos post implementación del módulo de verificación de la unidad *bootstrap* generado por la herramienta de Vivado. CLK=100 MHz

Peor slack de rutas	Tiempo(ns)
Setup	0.719
Hold	0.070

4.6. Reporte de consumo de potencia

Es necesario conocer la viabilidad del diseño en cuanto a consumo energético, con el fin de saber si es capaz de cumplir con las restricciones del diseño. Al igual que con el reporte de tiempos, la herramienta Vivado permite obtener un reporte de consumo energético. En la tabla 4.4 se muestra un resumen de los datos obtenido del reporte.

Tabla 4.4: Resumen de reporte de consumo energético post implementación del módulo de verificación de la unidad *bootstrap* generado por la herramienta de Vivado.

Potencia	Consumo (mW)
Dinámica	17
Estática	97
Total	114

Los datos mostrados están generados a partir del funcionamiento del circuito a una frecuencia de 100 MHz. Si se aumenta la frecuencia de operación, el consumo de potencia dinámica también aumentará, debido a que se aumenta las transiciones en el circuito. Por el contrario, al disminuir la frecuencia se obtendrá menos consumo de potencia dinámica, debido que las transiciones ocurrirán con menor frecuencia.

Capítulo 5

Síntesis lógica de la unidad *bootstrap* usando las herramientas EDA de Synopsys

En este capítulo se describe la síntesis lógica RTL y simulación lógica post síntesis RTL de la unidad *bootstrap* utilizando las herramientas EDA de Synopsys. La síntesis lógica se realiza con la herramienta *Design Compiler*, la cual convierte el diseño escrito en lenguaje de descripción de hardware en un mapeo de lista de nodos a nivel de compuertas en una tecnología en específico, en este caso XFAB 180 nm. [23]

La simulación post síntesis lógica RTL se realiza con la herramienta *VCS*. Esta realiza la simulación lógica de la lista de nodos a nivel de compuertas generado por el *Design Compiler*.

5.1. Síntesis lógica RTL la unidad *bootstrap* en la herramienta Design Compiler

Para realizar la síntesis lógica RTL de la unidad se debe tener la implementación del diseño en lenguaje de descripción de hardware. La herramienta utiliza la biblioteca sintética de la tecnología (para este proyecto es XFAB 180nm), los archivos del diseño en HDL (en este caso Verilog) y un archivo con restricciones de diseño. [23]

La síntesis lógica es ejecutada a través de dos scripts. Las restricciones de diseño se encuentran en un script, donde se especifican características de temporizado del reloj (frecuencia de 100 MHz y transición de 0.5 ns), modelo de carga para el cableado (en este caso se obtienen de la fuente `D_CELLS_LL_LP5MOS_typ_1_80V_25C`), retardos máximos y mínimos de las señales de entrada del diseño (máximo de 7.4 ns y mínimo 4 ns), retardos máximos y mínimos de las señales de salida (máximo de 6 ns y mínimo 3 ns) y configuraciones de celdas de entradas y salidas. Todos los valores se definen con el criterio de obtener un modelo más realista de la unidad, al incorporar tiempos de retardo a las señales.

En otro script se encuentran los comandos para la ejecución de la síntesis. En la figura 5.1 se muestra el el diagrama de flujo del script para la síntesis lógica del diseño.

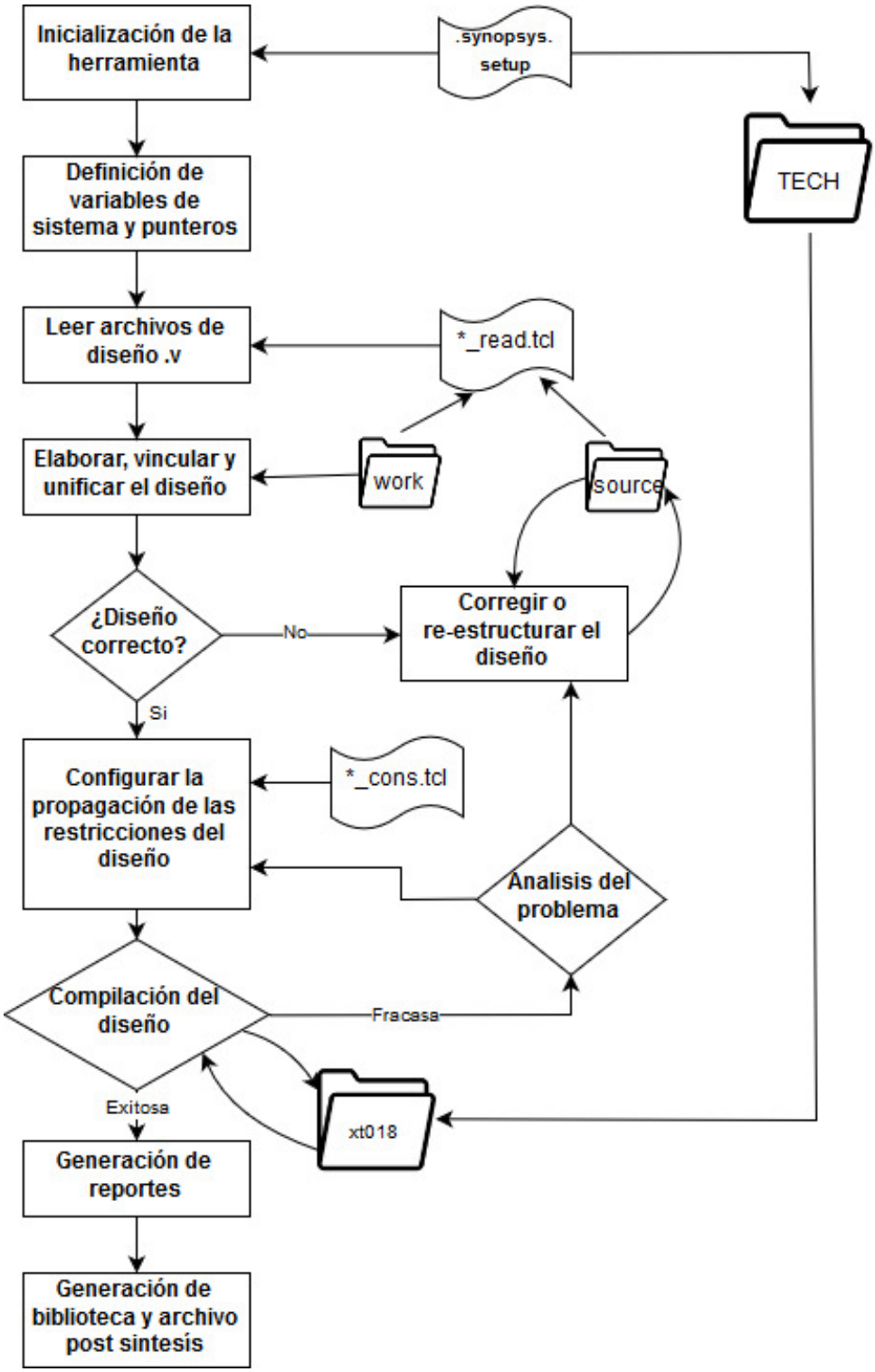

Figura 5.1: Diagrama de flujo de scripting que ejecuta la síntesis lógica RTL en *Design Compiler*. [5]

El script donde se encuentra los comandos de ejecución de la herramienta es el script principal, donde se realiza la definición de punteros y variables de sistemas hasta la generación de archivos post síntesis. Otro archivo de importancia es el script `.synopsys.setup`, donde se ejecutan la inicialización de parámetros, variables y bibliotecas de diseño. Este archivo es ejecutado al invocar *Design Compiler*. [5]

Una vez ejecutado el script se principal, se obtiene reportes de área utilizada, celdas utilizadas, puertos, potencia y tiempo. Además se puede obtener una visualización RTL de la unidad sintetizada. En la figura 5.2 se observa el diagrama RTL post síntesis lógica generado por *Design Compiler*.

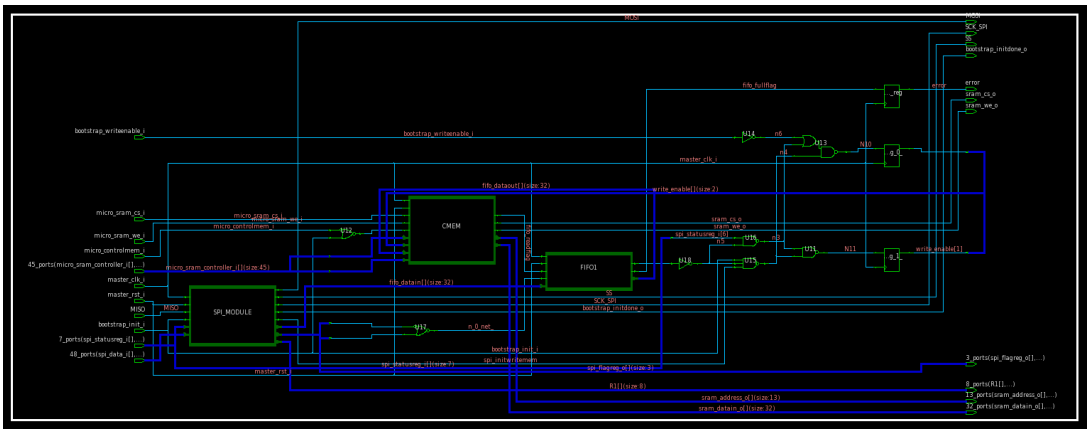

Figura 5.2: Diagrama RTL post síntesis lógica generado por *Design Compiler* para la unidad *bootstrap*.

El resumen de los reportes antes mencionados se observa en la tabla 5.1. Se observan algunos de los criterios de los reportes de tiempo, cantidad de celdas, área y potencia.

Tabla 5.1: Resumen de reportes generados por *Design Compiler* para la unidad *bootstrap*

Reporte	Característica	Valor
Tiempo	Periodo de reloj	10 ns
	Tiempo de ruta critica de slack	0.01 ns
	Peor violación de hold slack	0.00 ns
Cantidad de celdas	Cantidad de celdas	6601
	Cantidad de puertos total	709
	Cantidad de celdas inversores	288
Área	Área de celdas	190131.25 nm^2
	Área de diseño	201303.11 nm^2
Potencia	Potencia dinámica total	6.3227 mW

A partir de los datos del reporte de tiempo, se observa que el *set up* slack es un valor mayor a cero, por lo que la unidad sintetizada funciona a la frecuencia especificada, aunque el margen disponible no es muy grande. También, el *hold* slack es igual a cero, lo que indica que la unidad sintetizada funciona según las especificaciones establecidas, sin embargo no tiene margen disponible. Como se explico anteriormente, el slack se define como la diferencia entre el tiempo actual y el tiempo deseado para una ruta de tiempos.

5.2. Simulación post síntesis lógica RTL de la unidad *bootstrap* en la herramienta VCS

Se realizó un análisis a nivel de comportamiento de la unidad *bootstrap* sintetizada, con el fin de verificar su funcionamiento. Para realizar esto es necesario las primitivas de la biblioteca de síntesis de la tecnología, la lista de nodos a nivel de compuerta (archivo generado por la síntesis lógica) y el archivo Verilog de *testbench* para la unidad.

El *testbench* diseñado evalúa que se realice la operación de arranque de la unidad, al colocar en uno lógico la señal de `boostrap_init_i`, encargada de dar inicio a dicha operación. En la figura 5.3 se observa una sección de la simulación obtenida, donde se observa el `ACMD41`. Si se realiza una comparación con la figura 4.8 que contiene el mismo comando; puede observarse que los datos de salida son los mismos.

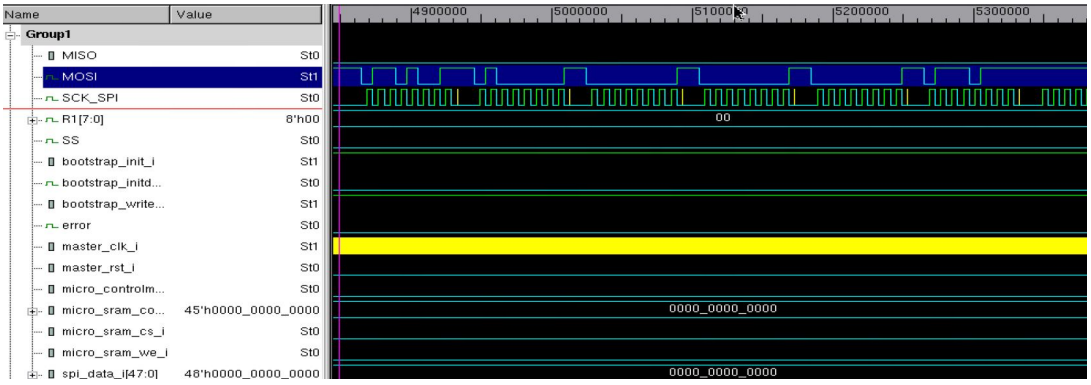

Figura 5.3: Comando `ACMD41` obtenido de la simulación post síntesis lógica RTL de la unidad *bootstrap*. Obtenida de la salida `MOSI`.

En la figura 5.4 se observa el flujo de comandos para inicialización de la tarjeta de memoria SD y las señales de control de la unidad. Se comparan las señales del SPI obtenidas en la simulación post síntesis lógica RTL con las señales SPI bajo simulación post implementación de tiempo de la Nexys 4 (figura 4.7), concluyendo que son iguales.

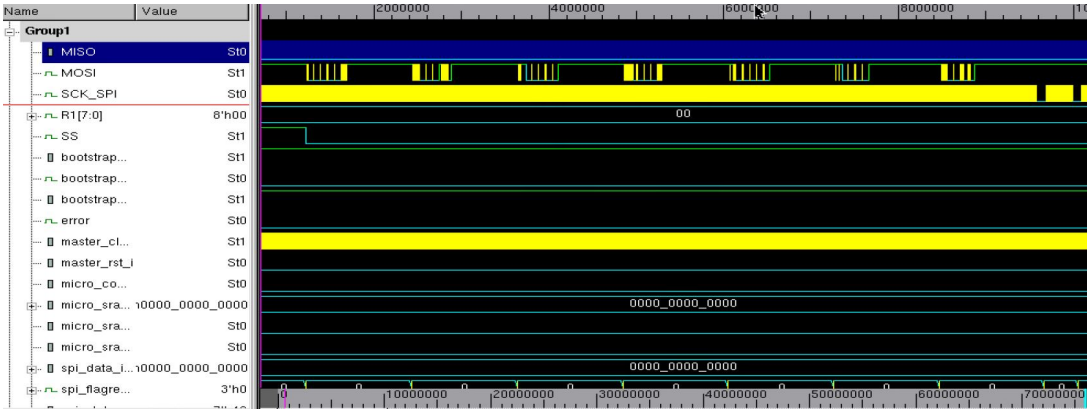

Figura 5.4: Simulación post síntesis lógica RTL de la unidad *bootstrap*. Se visualiza las señales de inicialización de la tarjeta SD, el reloj principal y las señales de control para la memoria SRAM provenientes del microprocesador.

Por lo anterior se puede deducir que la unidad sintetizada sería capaz de mantener las especificaciones de diseño planteadas para la unidad *bootstrap*, al cumplir con especificaciones de tiempos y un flujo de datos esperados para la señales que se observan en la figura 5.3 y 5.4. No obstante, para una aseveración completa de funcionamiento de esta unidad, debería generarse un marco de pruebas más intensivo. El tiempo asignado al proyecto no bastó para cumplir esta actividad, que deberá sin embargo complementarse para estar más seguros de la funcionalidad del diseño.

Capítulo 6

Conclusiones y recomendaciones

6.1. Conclusiones

- Las simulaciones RTL proveen de una verificación preliminar de un circuito digital, pero no son suficientes para garantizar su funcionamiento.
- Enviar únicamente los comandos de inicialización básicos para la tarjeta de memoria SD disminuye el tiempo de ejecución de la carga de instrucciones y disminuye el uso de recursos.
- La implementación de una etapa de generación y verificación del código CRC de seguridad duplica el tiempo de envío de comandos y duplica el tiempo de la lectura de datos.
- La verificación en FPGA es un paso importante para asegurar la funcionalidad de una unidad digital, como en este caso la unidad de *bootstrap* implementada, que pudo interfazarse correctamente a una tarjeta de memoria SD.
- Una prueba de simulación post síntesis de un circuito digital, permite verificar parcialmente la funcionalidad en términos de temporizado del mismo sobre una biblioteca CMOS particular. Esta simulación, no obstante, debe enmarcarse en una metodología adecuada para subir su fiabilidad, y debe llevarse al nivel de simulación post colocación y enrutamiento para ser exhaustiva.

6.2. Recomendaciones

- Se sugiere realizar una investigación sobre la estructura de memoria de las tarjetas SD, para poder determinar los bloques de memoria en los cuales se encontrará el programa de instrucciones. Además determinar el tipo de tarjeta a utilizar definitivamente, debido a que para el desarrollo de este proyecto se realizo entorno a una tarjeta tipo SDHC, lo cual limita el tamaño de bloque de datos.
- Se recomienda realizar un adecuado plan de piso del diseño, con énfasis en la etapa de implementación, con el fin de mejorar el cierre de temporización *(timing closure)* y el flujo de datos. Al respecto, se recomienda, para el diseño en FPGA, seguir el manual [10], mientras que para la síntesis en un circuito integrado, debe estudiarse más el flujo de herramientas de backend Synopsys.

- Es obligatoria la implementación de un ambiente de verificación funcional, donde el diseño sea puesto bajo pruebas que estresen más la unidad, con el fin de garantizar su fiabilidad.

Bibliografía

[1] Abrahams, L. *Implementación de una metodología para lograr la integración física "correcta por construcción (CBC)" del sistema de reconocimiento de patrones acústicos (SiRPA).* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2012.

[2] Alfaro, L. *Implementación en hardware del Sistema de Reconocimiento de Patrones Acústicos (SiRPA).* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2013.

[3] Association, T. C. S. C. *SD Specifications Part 1 Physical Layer Simplified Specification V 6.0.* SD Card Association, April,2017.

[4] C. Salazar-García, R. C.-G., and Chacón-Rodríguez, A. *"An embedded system for the detection of illegal hunting and logging".* presented at the Circuits Systems (LASCAS), 2017 IEEE 8th Latin American Symposium on, 2017, pp. 1–4., 2017.

[5] Castro, R. *Integración de un microprocesador de aplicación específica en un flujo de diseño de circuitos integrados digitales.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2017.

[6] de Educación y Cultura España., M. *Formación profesional a distancia: informática aplicada.* Ministerio de Educación y Cultura. Madrid, España., 1998.

[7] de la Nación en Desarrollo Humano Sostenible (Costa Rica), P. E. *Vigesimosegundo Informe Estado de la Nación en Desarrollo Humano Sosten.* PEN-CONARE.–22 edición. –San José C., 2016.

[8] Hartono, D., Lee, S., and Tang, C. A scalable bootloader and debugger design for an noc-based multi-processor soc.

[9] ICT. *Divisas por concepto de turismo.* Instituto Costarricense de Turismo. Recuperado el 2 de diciembre de 2017de http://www.ict.go.cr/es/estadisticas/cifras-economicas.html, 2017.

[10] Inc., D. *Vivado Design Suite Tutorial.* Digilent. Recuperado el 25 de octubre del 2017 de de japan.xilinx.com/support/documentation/sw_manuals_j/xilinx2014_2/ug938-vivado-design-analysis-closure-tutorial.pdf., Última revisión 06 de abril del 2014.

[11] Inc., D. *Nexys 4 FPGA Board Reference Manual.* Digilent. Recuperado el 25 de octubre del 2017 de de

https://reference.digilentinc.com/_media/nexys:nexys4:nexys4_rm.pdf., Última revisión 11 de abril del 2016.

[12] Instrument., T. *FIFO Architecture, Functions, and Applications.* Recuperado el 25 de octubre del 2017 de http://www.ti.com/lit/an/scaa042a/scaa042a.pdf, 1999.

[13] Leens, F. *An introduction to I2C and SPI protocols.* IEEE Instrumentation Measurement Magazine (Volume: 12, Issue: 1), February, 2009.

[14] Marcin Lewandowski, Tomasz Orczyk, P. P. Dedicated avr bootloader for performance improvement of prototyping process. Proceedings of the 24th International Conference "Mixed Design of Integrated Circuits and Systems",, 2017.

[15] Salas, E. *Reconocimiento en tiempo real de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA de modelos ocultos de Markov.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2010.

[16] Salazar-Garcia, C.; Alfaro-Hidalgo, L. C.-D. M. M.-A. J. C.-G. R. R. J. C.-R. A. A.-M. P. *"Digital integrated circuit implementation of an identification stage for the detection of illegal hunting and logging,".* in Circuits Systems (LASCAS), 2015 IEEE 6th Latin American Symposium on , pp.1-4, 24-27 Feb. 2015, 2015.

[17] Sequeira, M. *Módulo de reducción de dimensiones espectrales en un sistema de reconocimiento de patrones acústicos de motosierras y disparos por medio de una implementación en FPGA.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2011.

[18] SINAC. *Vida Silvestre.* Sistema Nacional de Áreas de Conservación Costa Rica. Recuperado el 2 de diciembre de 2017de http://www.sinac.go.cr/ES/visasilves/Paginas/default.aspx, 2017.

[19] Smith, L. *Reconocimiento digital en línea de patrones acústicos para la protección del ambiente por medio de HMM.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2008.

[20] Sparkfun. *Serial Peripheral Interface(SPI).* SparkFun. Recuperado el 27 de mayo del 2017 de https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi, 2017.

[21] Sáenz, M. *Reconocimiento de patrones acústicos para la protección del ambiente utilizando wavelets y Modelos Ocultos de Markov.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2006.

[22] Technology., I. D. *SRAMs.* Recuperado el 25 de octubre del 2017 de https://www.idt.com/products/memory-logic/srams, 2017.

[23] Valverde J., Porras D., C. R. S. J. L. G. *Tutorial para Flujo de Dise no de Circuitos Integrados Digitales con Tecnología IBM 8RF (CMOS 0.13m).* Instituto Tecnológico de Costa Rica, Escuela de Electrónica, DCILAB, 2016.

[24] Villegas, A. *Diseño de un detector de sonidos de motosierra con bajo consumo energético.* Tesis de Licenciatura, Escuela de Ingeniería en Electrónica, ITCR, 2013.

[25] Zhang, J., and Shao, F. A bootloader module is designed and implemented based on a new computer architecture. Second International Conference on Future Networks, 2010.

Apéndice A

Clasificación de las tarjetas de memoria SD

Tabla A.1: Tipos de tarjetas de memoria SD según capacidad de memoria, rango de tensión de operación y modo de velocidad de bus.

Característica	Tipo	Descripción
Capacidad de memoria	Memorias SD de capacidad estándar (SDSC)	Memorias menores o iguales a 2GB
	Memorias SD de alta capacidad (SDHC)	Memorias mayores a 2GB y menores o iguales a 32GB.
	Memorias SD de capacidad extendida (SDXC)	Memorias mayores a 32GB y hasta 2TB.
Rango de tensión de operación	Memorias SD de alta tensión (SDSC)	Tarjetas con un rango de operación de 27-3.6V
	UHS-II Tarjeta de memoria SD	Tarjetas con un rango de operación de dos tipos; VDD1 que comprende 2.7-3.6V y VDD2 que comprende el rango de 1.7-1.95V
Modo de velocidad de bus	Modo de velocidad por defecto	Señal de 3.3V y frecuencia de 25 MHz. Tasa de transmisión hasta 12.5 MB/s
	Modo de alta velocidad	Señal de 3.3V y frecuencia de 50 MHz. Tasa de transmisión hasta 25 MB/s
	SDR12	Tarjeta UHS-I con señal de 1.8V y frecuencia de 25 MHz.Tasa de transmisión hasta 12.5 MB/s
	SDR25	Tarjeta UHS-I con señal de 1.8V y frecuencia de 50 MHz. Tasa de transmisión hasta 25 MB/s

Característica	Tipo	Descripción
Modo de velocidad de bus	SDR50	Tarjeta UHS-I con señal de 1.8V y frecuencia de 100 MHz. Tasa de transmisión hasta 50 MB/s
	SDR104	Tarjeta UHS-I con señal de 1.8V y frecuencia de 208 MHz. Tasa de transmisión hasta 104 MB/s
	DDR50	Tarjeta UHS-I con señal de 1.8V y frecuencia de 50 MHz, muestreado en ambos flancos de reloj. Tasa de transmisión hasta 50 MB/s
	UHS156	Tarjeta UHS-II con rango de frecuencias de 26 MHz a 52 MHz y hasta 1.56 Gbps por línea.
	UHS624	Tarjeta UHS-II con rango de frecuencias de 26 MHz a 52 MHz y hasta 6.24 Gbps por línea

Apéndice B

Comandos SPI para la tarjeta de memoria SD

Tabla B.1: Comandos y argumentos para tarjeta de memoria SD. [3]

Indice de comando	Modo SPI	Argumento	Resp.	Abreviación	Descripción
CMD0	Si	[31:0]bits en '0'	R1	*GO_IDLE_STATE*	Reset a la tarjeta SD
CMD1	Si	[31]bit reservado [30] HCS [29:0]bits en '0'	R1	*SEND_OP_COND*	Envía al *host* información sobre el proceso de inicialización
CMD2	No	-	-	-	-
CMD3	No	-	-	-	-
CMD4	No	-	-	-	-
CMD5	No	-	-	-	-
CMD6	Si	[31]Mode 0:Verifica 1:Cambia a modo switch [30:0]bits en '0'	R1	*SWITCH_FUNC*	Cambia o verifica estado del modo switch.
CMD7	No	-	-	-	-
CMD8	Si	[30:0]bits en '0' [11:8]Voltaje alimentación [7:0]patrón de verificación	R7	*SEND_IF_COND*	Envía a la tarjeta el rango de voltaje de operación
CMD9	Si	[31:0]bits en '0'	R1	*SEND_CSD*	Tarjeta SD envía envía dato de reg. CSD
CMD10	Si	[31:0]bits en '0'	R1	*SEND_CID*	Tarjeta SD envía envía dato de reg. CID
CMD11	No	-	-	-	-

Indice de comando	Modo SPI	Argumento	Resp.	Abreviación	Descripción
CMD12	Si	[31:0]bits en '0'	R1	*STOP_ TRANSMISSION*	Forzar tarjeta a detener el envío de datos de lectura.
CMD13	Si	[31:0]bits en '0'	R1	*SEND_STATUS*	Tarjeta SD envía envía dato de reg. status
CMD14	No	-	-	-	-
CMD15	No	-	-	-	-
CMD16	Si	[31:0]tamaño de bloque	R1	*SET_ BLOCKLEN*	En caso de tarjetas SDSC este comando determina el ancho de bloque. En tarjetas SDHC y SDXC es un tamaño fijo de 512 bytes.
CMD17	Si	[31:0]dirección	R1	*READ_ SINGLE_ BLOCK*	Lectura de un bloque de datos.
CMD18	Si	[31:0]dirección	R1	*READ_ MULTIPLE_ BLOCK*	Lectura continua de bloques de datos.
CMD19	No	-	-	-	-
CMD20	No	-	-	-	-
CMD21	No	-	-	-	-
CMD23	No	-	-	-	-
CMD24	Si	[31:0]dirección	R1	*WRITE_ BLOCK*	Escritura de un bloque de datos.
CMD25	Si	[31:0]dirección	R1	*WRITE_ MULTIPLE_ BLOCK*	Escritura continua de bloques de datos.
CMD26	No	-	-	-	-
CMD28	Si	[31:0]dirección	R1	*SET_ WRITE_ PROTECTION*	Modifica la protección de escritura. No valido para tarjetas SDHC y SDXC.
CMD29	Si	[31:0]dirección	R1	*SET_ WRITE_ PROTECTION*	Deshabilita la protección de escritura. No valido para tarjetas SDHC y SDXC.

Índice de comando	Modo SPI	Argumento	Resp.	Abreviación	Descripción
CMD30	Si	[31:0]dirección de escritura	R1	*SET_ WRITE_ PROTECTION*	Habilita la protección de escritura en un bloque de datos.No valido para tarjetas SDHC y SDXC.
CMD31	No	-	-	-	-
CMD32	Si	[31:0]dirección	R1	*ERASE_WR_ BLK_START_ ADDR*	Dirección de inicio del bloque a para borrar.
CMD33	Si	[31:0]dirección	R1	*ERASE_WR_ BLK_END_ ADDR*	Dirección final del bloque a para borrar.
CMD34-37	No	-	-	-	-
CMD38	Si	[31:0]bits en ‘0’	R1	*ERASE*	Borra todo el contenido entre direcciones de inicio y final de CMD33 y CMD32.
CMD39	No	-	-	-	-
CMD40	No	-	-	-	-
CMD41	No	-	-	-	-
CMD42	Si	[31:0]bits en ‘0’	R1	*LOCK_ UNLOCK_*	Resetea las configuraciones de bloque de seguridad.
CMD43-49	No	-	-	-	-
CMD51	No	-	-	-	-
CMD50	No	-	-	-	-
CMD52-54	No	-	-	-	-
CMD55	Si	[31:0]bits en ‘0’	R1	*APP_CMD*	Define a la tarjeta que el siguiente comando es de aplicación especifica.
CMD56	Si	[31:1]bits en ‘0’ [0]RD/WR	R1	*GEN_CMD*	Permite la lectura o escritura de un bloque de datos de una aplicación especifica.
CMD57	No	-	-	-	-
CMD58	Si	[31:0]bits en ‘0’	R3	*READ_OCR*	Lectura del reg. OCR de la tarjeta SD.

Indice de comando	Modo SPI	Argumento	Resp.	Abreviación	Descripción
CMD59	Si	[31:1]bits en '0' [0]opción CRC	R1	*CRC_ON_OFF*	Habilita opción de código CRC. 1:Habilitado 0:Deshabilitado
CMD60-63	-	-	-	-	Reservado.
ACMD6	No	-	-	-	-
ACMD13	Si	[31:0]bits en '0'	R2	*SD_STATUS*	Se envía status de la tarjeta SD.
ACMD17	-	-	-	-	Reservado.
ACMD18	-	-	-	-	Reservado.
ACMD19-21	-	-	-	-	Reservado.
ACMD22	Si	[31:0]bits en '0'	R1	*SEND_NUM_WR_BLOCKS*	Envía el número de bloques a escribir.
ACMD23	Si	[31:23]bits en '0' [22:0]Núm. de bloques	R1	*SET_WR_BLK_ERASE_COUNT*	Envía el número de bloques para ser borrados después de de escritura.
ACMD24	-	-	-	-	Reservado.
ACMD25	-	-	-	-	Reservado.
ACMD26	-	-	-	-	Reservado.
ACMD38	-	-	-	-	Reservado.
ACMD39-40	-	-	-	-	Reservado.
ACMD41	Si	[31]bit en '0' [30]HCS [29:0]bit en '0'	R1	*SD_SEND_OP_COND*	Envía información sobre proceso de inicialización.
ACMD42	Si	[31:1]bit en '0' [0]set_cd	R1	*SET_CLR_CARD_DETECT*	Conecta(1) o desconecta(0) la resistencia de 50kOhm del contacto CS de la tarjeta SD.
ACMD43-49	-	-	-	-	Reservado.
ACMD51	Si	[31:0]bits en '0'	R1	*SEND_SCR*	Lectura del reg. SCR.