Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Programa de Licenciatura en Ingeniería Electrónica

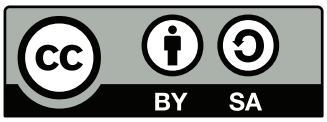
Implementación en FPGA de un sistema para el almacenamiento y transmisión de datos provenientes de una unidad de adquisición de señales de sensores cardiacos magnéticos

Informe de Trabajo Final de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Carlos Andrey Morales Zamora

Cartago, 13 de junio, 2024



Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado referencias, he indicado las fuentes mediante citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente trabajo.

Carlos Andrey Morales Zamora

Cartago, 13 de junio de 2024

Céd: 1-1686-0125

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Trabajo Final de Graduación
Acta de Aprobación

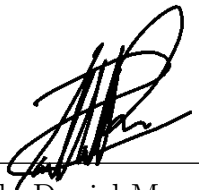
Defensa de Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura

El Tribunal Evaluador aprueba la defensa del trabajo final de graduación denominado *Implementación en FPGA de un sistema para el almacenamiento y transmisión de datos provenientes de una unidad de adquisición de señales de sensores cardiacos magnéticos*, realizado por el señor Carlos Andrey Morales Zamora 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

Dr. Alfonso Chacón Rodríguez
Profesor Lector

Dr. Juan José Montero Rodríguez
Profesor Lector



Dr. Pablo Daniel Mendoza Ponce
Profesor Asesor

Cartago, 13 de junio, 2024

Resumen

Este documento aborda la problemática sobre el manejo de grandes volúmenes de datos biomédicos, específicamente señales cardiacas, para su almacenamiento y transmisión prolongados. El objetivo principal fue diseñar e implementar un sistema en FPGA que permitiera el almacenamiento y transmisión en tiempo real de los datos biomédicos, manteniendo la integridad de los datos. Se comparó el rendimiento del sistema propuesto con dispositivos comerciales, destacando las ventajas del diseño personalizado.

De esta forma, para el desarrollo de este documento, se diseñaron dos sistemas independientes. El primer sistema gestiona el almacenamiento de los datos crudos con una tasa de escritura de 100 Mbps en una tarjeta microSD SDXC. El segundo sistema transmite a 10 Mbps los datos procesados mediante SPI hacia un convertidor serial a USB 2.0 lo que facilita la transferencia a un computador. Ambos diseños fueron implementados cargando los *bitstreams* de los sistemas en una FPGA.

Los resultados confirmaron que el sistema de almacenamiento alcanza una tasa de transferencia de 100 Mbps con pruebas exitosas que resultaron en 7.2 GB de datos crudos almacenados. El sistema de transmisión demostró una configuración adecuada del convertidor como controlador SPI y la capacidad de la unidad de hardware para transferir datos a 10 Mbps. Además, se verificó que la computadora captura y almacena los datos procesados correctamente en un archivo binario de 114 MB.

El sistema en FPGA desarrollado demuestra ser una solución robusta y eficiente para el desafío de transmitir y almacenar grandes cantidades de datos biomédicos. Puesto que con tasas de escritura de 100 Mbps y transmisión de 10 Mbps, el sistema no solo cumple con los objetivos de rendimiento, sino que también supera a los dispositivos existentes en el mercado. Este representa un avance significativo en la gestión de señales cardiacas, ya que garantiza la integridad de los datos en aplicaciones críticas. Los resultados respaldan la propuesta e incentivan a futuras investigaciones.

Palabras clave: FPGA, SystemVerilog, microSD, SPI, almacenamiento, transmisión.

Abstract

This paper addresses the problem of handling large volumes of biomedical data, specifically cardiac signals, for long-term storage and transmission. The main objective was to design and implement an FPGA system that would allow real-time storage and transmission of biomedical data while maintaining data integrity. The performance of the proposed system was compared with commercial devices, highlighting the advantages of the custom design.

For the development of this paper, two independent systems were designed: the first one manages the storage of the raw data, with a write rate of 100 Mbps on a microSD SDXC card, and the second system transmits at 10 Mbps the processed data via SPI to a serial to USB 2.0 converter, facilitating the transfer to a computer. Both designs were implemented by loading the system's bitstreams onto an FPGA.

The results confirmed that the storage system achieves a transfer rate of 100 Mbps with successful tests resulting in 7.2 GB of raw data stored. The transmission system demonstrated proper configuration of the converter as an SPI controller and the the hardware unit's ability to transfer data at 10 Mbps. In addition, the computer was verified to capture and store the processed data correctly in a 114 MB binary file.

The developed FPGA system proves to be a robust and efficient solution to the challenge of transmitting and storing large amounts of biomedical data. With write rates of 100 Mbps and transmission rates of 10 Mbps, the system not only meets performance targets, but also surpasses existing devices on the market. It represents a significant advance in cardiac signal management, ensuring data integrity in critical applications. The results support the proposal and encourage future research.

Keywords: FPGA, SystemVerilog, microSD, SPI, storage, transmission.

Índice general

Índice de figuras

Índice de tablas

Lista de símbolos y abreviaciones

Abreviaciones

ADC	Convertidor Analógico Digital
CAU	Universidad Christian-Albrechts de Kiel
CRC	Código de Redundancia Cíclica
DSP	Procesamiento Digital de Señales
FDX	Full Duplex
FIFO	Firt Input Firt Output
FPGA	Field Programmable Gate Array
FSM	Máquina de Estados Finita
FTDI	Future Technology Devices International
GPIO	Entrada/Salida de Propósito General
HDL	Lenguaje de Descripción de Hardware
HVL	Lenguaje de Verificación de Hardware
I2C	Inter-Integrated Circuit
IC	Circuito Integrado
JTAG	Joint Test Action Group
LSB	Bit menos significativo
MIU	Módulo Interfaz de Usuario
MSB	Bit más significativo
RTL	Register-Transfer Level
SD	Secure Digital
SES	Sistema de Empaquetado y Segmentación de Bits

SHM	Sistema Host para Múltiples Tarjetas microSD
SPI	Serial Peripheral Interface
UCC	Unidad Central de Comandos
UCD	Unidad Central de Datos
UCM	Unidad Central de Múltiples Tarjetas microSD
UCR	Unidad de Control de Reloj
UHS-I	Ultra-High Speed Phase I

Capítulo 1

Introducción

1.1. Magnetocardiografía

Los seres vivos tienen la capacidad de generar sus propios campos magnéticos que son conocidos como campos magnéticos biológicos [1]. Estos campos se estudian en dos áreas principales: la Magnetobiología que investiga los efectos de los campos magnéticos en los seres vivos; y, el Biomagnetismo que se dedica a investigar los campos magnéticos generados por los propios organismos [2].

Los campos magnéticos biológicos producidos por el cuerpo humano son extremadamente débiles, en escalas de nT y fT. Como referencia, el campo magnético de la Tierra es del orden de 50uT [3]. Estos campos tienen su origen en corrientes eléctricas que circulan en algunas células, por ello el biomagnetismo establece diversos campos magnéticos en el cuerpo humano según el órgano o parte del cuerpo asociado [1]:

- Magnetopneumograma: campos magnéticos asociados a partículas ferromagnéticas presentes en el pulmón.
- Magnetocardiograma: campos magnéticos producidos por la despolarización del corazón.
- Magnetocardiograma Fetal: campos magnéticos producidos por la despolarización del corazón del feto.
- Neuromagnetismo: campos magnéticos producidos por el cerebro.

Mediante la magnetocardiografía (MCG) es posible determinar el magnetocardiograma del cuerpo humano, lo que permite medir y mapear los campos magnéticos generados por la actividad eléctrica dentro del corazón [4]. Sin embargo, presenta el inconveniente para lograr mediciones precisas dada la baja intensidad de dichos campos y la presencia de otros campos magnéticos mucho más intensos, como lo pueden ser la Tierra o el ruido

ambiental [5]. En la figura 1.1 se muestran los diferentes campos magnéticos biológicos del ser humano y las principales fuentes de ruido magnético.

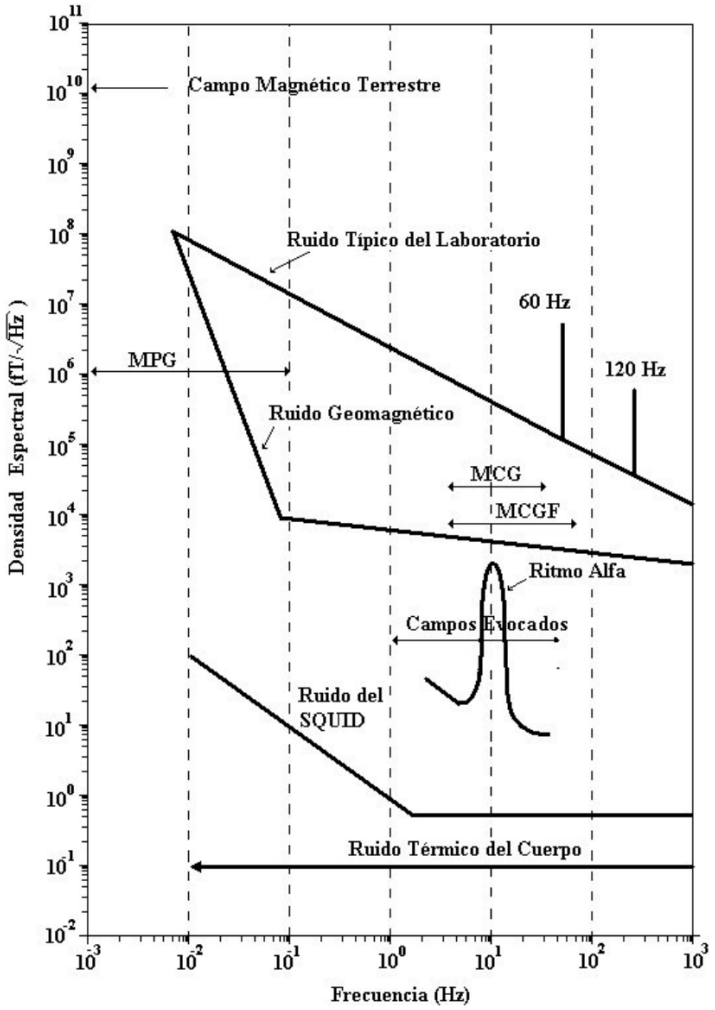


Figura 1.1: Densidad Espectral de los Campos Magnéticos del Ser Humano y Ruidos Magnéticos en Función de la Frecuencia [1]

Dado que los campos magnéticos medidos por el magnetocardiografía (MCG) se generan a partir de corrientes eléctricas que, a su vez, son producidas por la actividad eléctrica del corazón, permite al MCG obtener el ritmo y la frecuencia cardíaca. Por lo cual, en comparación con otras técnicas de diagnóstico cardíaco, como la electrocardiografía (ECG), el MCG presenta numerosos beneficios. Uno de ellos es una mejor resolución espacial, puesto que detalla con mayor precisión la actividad eléctrica del corazón, permitiendo detectar alteraciones en el ritmo cardiaco y anomalías estructurales. De esta manera, facilita el diagnóstico de enfermedades cardiovasculares, como arritmias, infartos de miocardio y cardiopatías congénitas [4] [6].

Otro aspecto de gran relevancia radica en su carácter no invasivo, es decir, no requiere intervención física en el cuerpo del paciente, pues a diferencia de otros métodos de diagnóstico como angiografía o ecocardiografía invasiva, la MCG no implica riesgos asociados a la introducción de catéteres o sondas en el cuerpo del paciente [6].

Sin embargo, la magnetocardiografía presenta una serie de dificultades para su implementación, por ejemplo, la interferencia de los campos magnéticos externos, como lo pueden ser equipos eléctricos y estructuras metálicas que podrían llegar a interferir con las señales de baja intensidad del cuerpo humano. Otra problemática se relaciona con el uso de magnetómetros altamente sensibles utilizados para medir el magnetocardiograma, ya que son altamente costosos y requieren de condiciones e instalaciones especializadas para su uso, lo que restringe su empleo en entornos médicos, entre otros[6].

Es por ello que países dedicados a la investigación han realizado un gran esfuerzo en desarrollar nuevas técnicas innovadoras para la detección del magnetocardiograma. Entre las iniciativas se destaca el grupo de investigación CRC 1261, el cual mediante el trabajo investigativo, desarrolló sensores biomagnéticos capaces de medir el magnetocardiograma con una implementación que permite la portabilidad y un uso generalizado. Este grupo está conformado por una alianza entre cuatro instituciones de gran prestigio y renombre en la investigación y desarrollo con respectivas sedes en Alemania. Las instituciones son la Universidad Christian-Albrechts de Kiel (CAU), la Universidad Hospital Schleswig-Holstein (UKSH), el Instituto Fraunhofer de Tecnología del Silicio (ISIT) y el Instituto Leibniz para la Pedagogía de las Ciencias Naturales y las Matemáticas (IPN) [7].

1.2. Antecedente

El presente documento contribuye al proyecto ‘Quantitative Evaluation for Magnetoelectric Sensor Systems in Biomagnetic Diagnostics’, liderado por el candidato doctoral MSc. Johan Solis Arbustini. Esta contribución promueve e impulsa la cooperación interdisciplinaria e interuniversitaria, en el marco del proyecto más amplio denominado ‘Demodulation System for Converse Magneto-Electric Sensor’. El MSc. Arbustini, actualmente afiliado al Departamento de Ingeniería Eléctrica e Ingeniería de la Información de la CAU, se enfoca en el desarrollo de sensores magnéticos aplicados a la biomedicina y la electromedicina, específicamente en la medición de parámetros cardíacos mediante sensores magnéticos para la caracterización de la actividad eléctrica del corazón.

1.3. Sistema para pruebas de un sensor magnético

El sistema propuesto por la CAU para la realización de pruebas de un sensor magnético se muestra en la figura 1.2. En el cual cuentan con un sensor con las siguientes características:

- Frecuencia de resonancia en torno a 500 KHz.
- Señal de salida con un comportamiento de señal modulada AM.
- Rango dinámico de 100 dB (profundidad/modulación AM a partir de 0.01 %).
- Ancho de banda de 12 kHz.

El sistema incluye un ADC de 12 bits por muestra con una velocidad de 8 Msps, lo que implica una transferencia neta de información de 96 Mbps. Además, cuenta con un DSP que produce una salida de 32 bits a una frecuencia de 50 kHz. Para las pruebas se requiere una medición continua de al menos 10 minutos, lo que implica un total de 4800 millones de muestras del ADC y 30 millones de datos procesados del DSP.

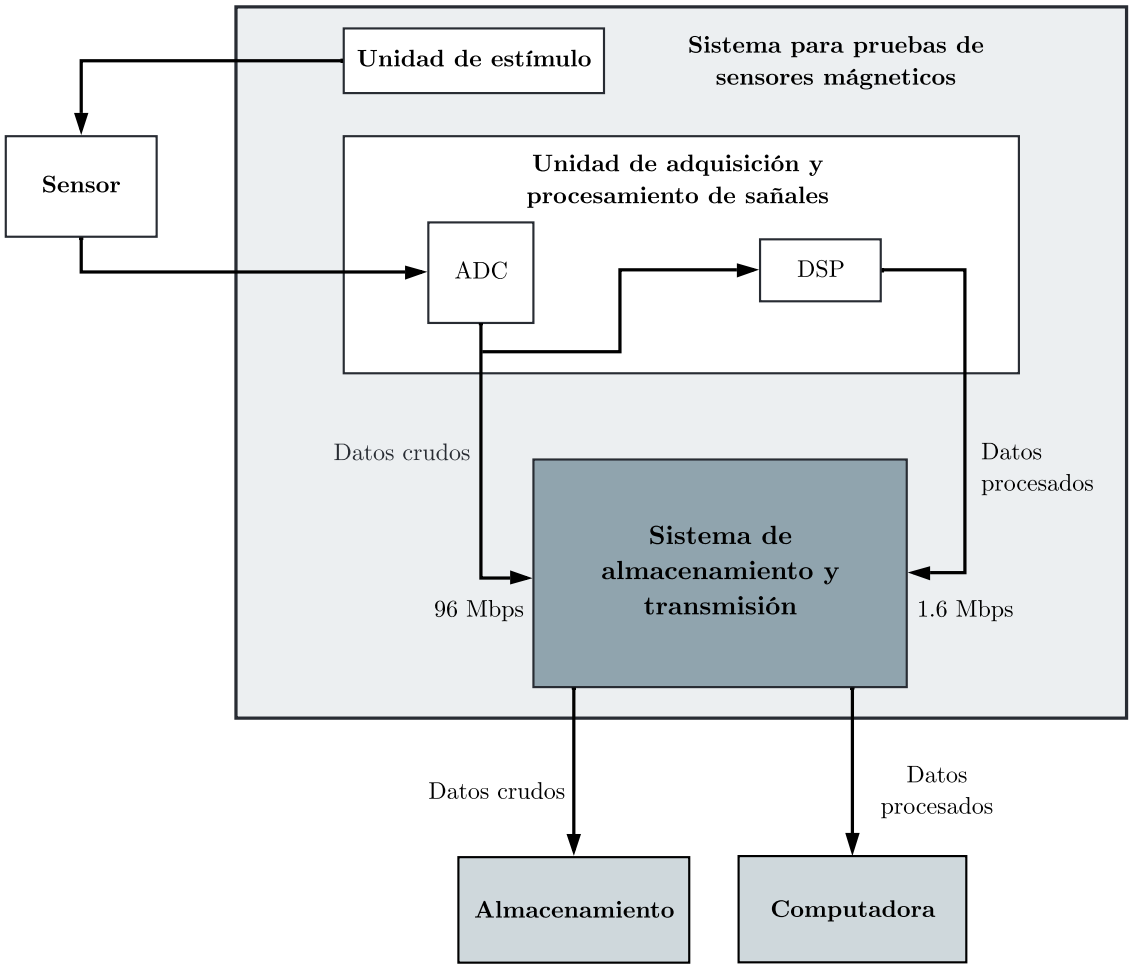


Figura 1.2: Sistema para Pruebas de un Sensor Magnético

En la actualidad, no existe un sistema que en tiempo real tenga la capacidad de almacenar datos crudos en una memoria no volátil, ni que pueda transmitir datos procesados a una computadora. Por lo tanto, este documento surge de la necesidad de implementar un sistema de almacenamiento y transmisión que cumpla con los requisitos mencionados anteriormente sin pérdida de información, ya que la integridad de los datos es especialmente crucial en sistemas biomédicos, donde su precisión es vital para el diagnóstico, seguimiento y tratamiento de pacientes.

1.4. Sistema de almacenamiento y transmisión

Para solventar el problema anterior, se plantea la implementación de un sistema de almacenamiento y transmisión en una FPGA mediante dos sistemas independientes como se muestra en la figura 1.3.

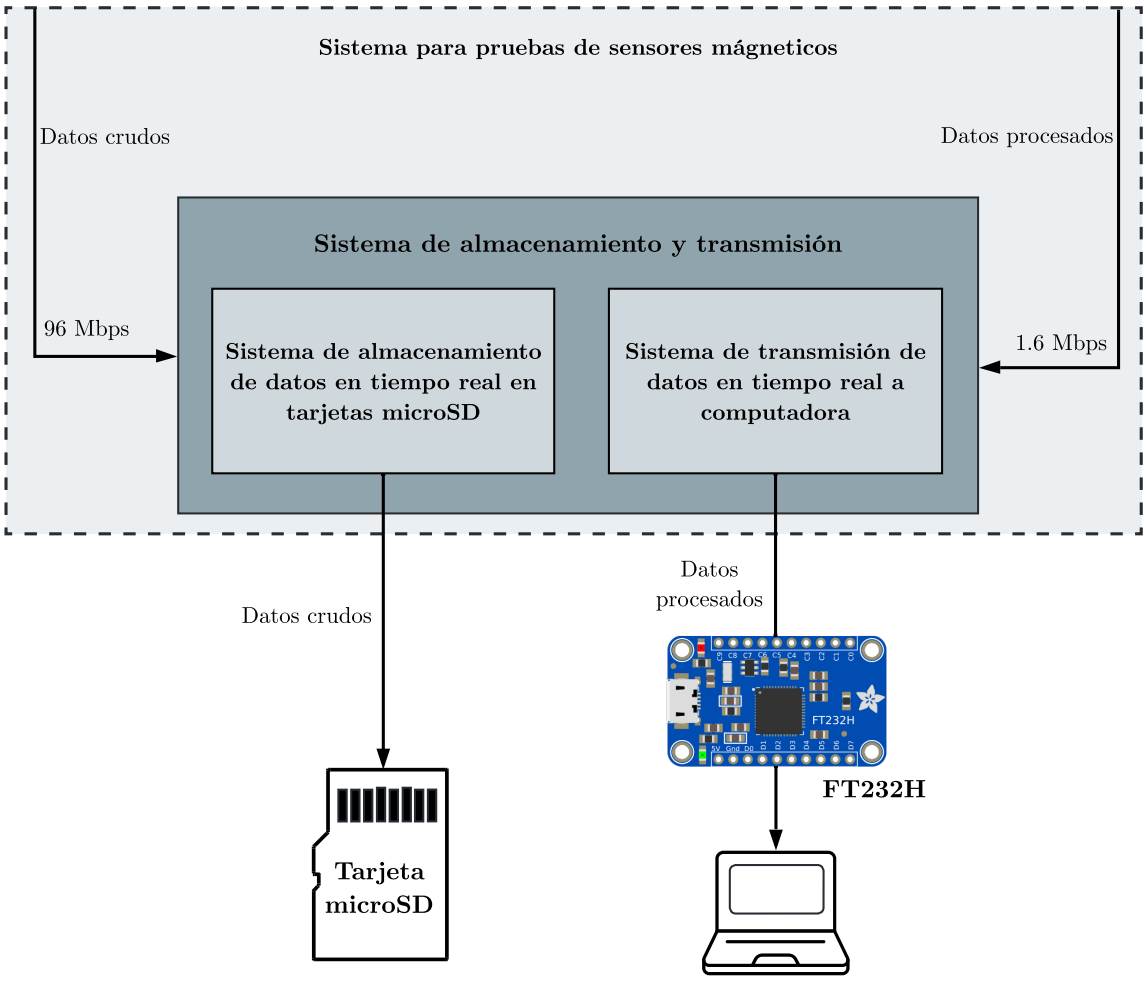


Figura 1.3: Sistema de Almacenamiento y Transmisión

Por una parte, el sistema de almacenamiento de datos en tiempo real en tarjetas microSD tiene como propósito almacenar los datos crudos provenientes del ADC en una tarjeta microSD mediante un subsistema host de múltiples tarjetas microSD compatible con el estándar UHS-I para tarjetas SDHC y SDXC a una tasa de transferencia de 100 Mbps mediante el protocolo SD 4-bits.

Por otra parte, el sistema de transferencia de datos en tiempo real a computadora tiene como propósito transmitir los datos procesados provenientes del DSP a un dispositivo FT232H a una tasa de transferencia de 10 Mbps por medio del protocolo SPI en el que posteriormente el dispositivo trasmitirá los datos procesados a un computador mediante el protocolo USB 2.0.

1.5. Objetivos y estructura del documento

El documento tiene como objetivo general diseñar e implementar un sistema de almacenamiento y transmisión en tiempo real mediante hardware, utilizando una FPGA para almacenar los datos crudos provenientes de un ADC con una tasa de transferencia de 96 Mbps a una tarjeta microSD así como transmitir los datos procesados de un DSP con una tasa de transferencia de 1.6 Mbps a una computadora. Adicionalmente, realiza una comparativa de rendimiento con el sistema propuesto y dispositivos de adquisición de datos del mercado.

Por otra parte, en cuanto a la estructura de este documento, este se divide en cinco capítulos. En el primer capítulo se presenta el trabajo realizado previo a la elaboración de este documento. En el segundo capítulo se abordan los fundamentos teóricos necesarios para explicar el sistema desarrollado. Los dos capítulos siguientes se centran en los dos subsistemas propuestos, y finalmente, se encuentran las conclusiones y recomendaciones.

Capítulo 2

Estado del arte

Durante la visita a la Universidad Christian-Albrechts de Kiel (CAU) en Alemania, se realizaron una serie de pruebas teóricas utilizando dispositivos de adquisición de datos de última generación. Los dispositivos y sus principales características se muestran en la tabla 2.1, los cuales cuentan con las mejores especificaciones disponibles en el mercado.

Tabla 2.1: Parámetros Utilizados

Dispositivo	Compañía	ADC				Tamaño del Búfer por canal
		Resolución (bits)	Tasa de muestreo (Msps)	Voltaje (Vpp)	Ancho de banda (Hz)	
STEMlab 125-14 N_LN [8]	RedPitaya	14	15.63	1	60M	16384 Zynq 7010 BRAM (128MB, 512 MB DDR3 RAM)
Analog Discovery 3 [9]	Digilent / NI	14	12.5	5	500k	16000
Logic Pro 8 [10]	Salaea	12	50	20	1M	RAM del computador conectado
PXI-4480 [11]	NI	24	2	1.5	9M	8221
HF2LI [12]	Zurich Instruments	14	10	5	40.3M	2048

La finalidad de las pruebas consistía en comprobar la viabilidad de los dispositivos como un sustituto a la unidad de adquisición y procesamientos de señales. Dado que los dispositivos de adquisición de datos cuentan con ADC internos y sistemas de transferencia o almacenamientos de datos para su posterior lectura o manejo.

Los datos obtenidos de mayor interés en el documento se muestran en la tabla 2.2 donde se visualizan los tamaños de los archivos generados por los diferentes dispositivos para una prueba con una frecuencia de resonancia de un tono de 500 kHz y un mensaje de un tono de 10.1 Hz con una profundidad de 0.01 % y una duración de un segundo. Cabe destacar que todos los datos son presentados en formato ASCII. El dispositivo HF2LI no aplica para la tabla dado que este dispositivo solo genera el espectro en frecuencia de los datos obtenidos.

Adicionalmente en la figura 2.1 se muestra la problemática presente en todos los dispositivos debido a las limitantes del buffer y como consecuencia se obtienen perturbaciones en la señal debido a la pérdida de datos.

Tabla 2.2: Tamaño de los Archivos Obtenidos (1 segundo)

Dispositivo	Tamaño del archivo (MB)	Tasa de muestreo (Msps)	Formato de archivo
STEMlab 125-14 N_LN	328.026	15.625	.txt
Analog Discovery 3	114.441	12.5	.mat
Logic Pro 8	261	12.5	.csv
PXI-4480	367	20	.mat

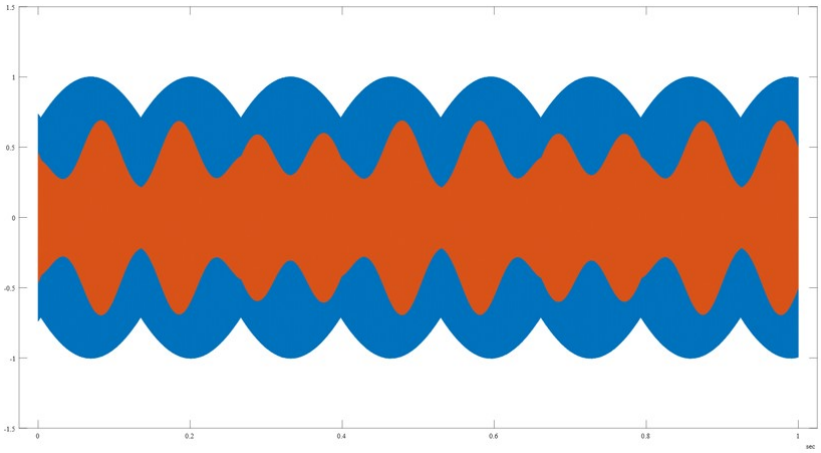


Figura 2.1: Señal Teórica Obtenida con Pérdida de Datos

Capítulo 3

Marco teórico

3.1. FPGA

Una FPGA (Field Programmable Gate Array) es un circuito integrado que se basa en una matriz de bloques de lógica configurables (CLBs) conectados a través de interconexiones programables. Las FPGA son programables en campo, es decir, pueden ser programadas y reprogramadas en cualquier momento para realizar cualquier función lógica [13].

3.1.1. Nexys4 DDR

El documento se ha elaborado utilizando la Nexys4 DDR que es una plataforma de desarrollo de circuitos digitales basada en una FPGA Artix-7 100T de Xilinx®. La Nexys4 DDR cuenta con una serie de puertos y periféricos entre los que se encuentran switches, LEDs, display 7-segmentos, puente USB-UART, puertos Pmod y conector de tarjeta microSD [14].

La FPGA Artix-7 cuenta con las siguientes características:

- 15850 secciones lógicos, cada uno con cuatro LUT de 6 entradas y 8 flip-flops.
- 4860 kbits de RAM de bloque rápido.
- Seis módulos de gestión de reloj, cada uno con un phase-locked loop (PLL).
- 240 secciones DSP.
- Velocidades de reloj interno superior a 450MHz.
- Convertidor analógico-digital en chip (XADC).

Los datos de configuración de la FPGA se almacenan en archivos denominados "bitstreams" que tienen la extensión de archivo .bit. El software Vivado de Xilinx puede crear bitstreams a partir de archivos fuentes como VHDL, Verilog o SystemVerilog.

3.1.2. SystemVerilog

SystemVerilog es una extensión de Verilog que combina elementos HDL (Hardware Description Lenguage) y HVL (Hardware Verification Lenguage) estandarizado como IEEE 1800-2012 [15]. SystemVerilog es el lenguaje empleado por este documento para generar los bitstream usados en la Nexys4 DDR.

Esta extensión es ampliamente utilizada para la creación de RTL (Register Transfer Level), el cual es una representación abstracta de un diseño de circuito digital. En el nivel RTL, los componentes del circuito se describen en términos de cómo los datos se transfieren y transforman desde las entradas hasta las salidas. Esta representación permite diseñar y optimizar circuitos a un nivel de abstracción más alto que el nivel de puerta, lo que facilita el manejo de diseños complejos [16].

3.1.3. ILA

El ILA (Integrated Logic Analyzer) es un núcleo IP (Intellectual Property) personalizable de Xilinx que se utiliza para monitorizar las señales internas de un diseño RTL. Este núcleo consume recursos de BRAM (Block Random Access Memory) en la FPGA, cuya cantidad depende de la configuración del ILA y en particular de la profundidad de los datos de muestra. El ILA cuenta con ecuaciones de disparo booleanas y disparos de transición de flancos, así como con múltiples puertos de prueba, lo que permite combinarlos en una única condición de disparo. El ancho de disparo, ancho de datos y profundidad de los datos pueden ser seleccionados por el usuario [17].

3.2. Protocolo SPI

El protocolo Serial Peripheral Interface (SPI) es una tecnología introducida por Motorola en 1979 [18]. El SPI es una interfaz síncrona basada en FDX, por lo que dispone de una línea encargada del proceso de sincronización y dos líneas de datos encargadas de recibir y transmitir respectivamente la información de manera simultánea [19].

3.2.1. Estructura

El SPI se estructura en torno a nodos. El nodo principal, comúnmente llamado controlador, se encarga de transmitir información a los subnodos también conocidos como periféricos, así como de controlar el proceso de sincronización. El periférico, por su parte, se encarga de recibir y enviar información al controlador. Esta jerarquía le permite al protocolo contar con múltiples periféricos controlados únicamente por un controlador [19].

3.2.2. Bus

Existen cuatro líneas lógicas encargadas de la comunicación entre controlador y periférico:

- **SDO** (Serial Data output): línea responsable de llevar los bits que provienen del controlador hacia el periférico.
- **SDI** (Serial Data input): línea responsable de llevar los bits que provienen del periférico hacia el controlador.
- **SCLK** (Serial Clock): línea proveniente del controlador, responsable de enviar la señal de reloj para sincronizar los nodos.
- **CS** (Chip Seect): línea encargada habilitar o deshabilitar un periférico.

La figura 3.1 ilustra una configuración de SPI que involucra un controlador y un único periférico. Sin embargo, esta configuración puede adaptarse para incluir a 'n' periférico, donde `n` representa cualquier número entero, cada uno con su propia señal CS.

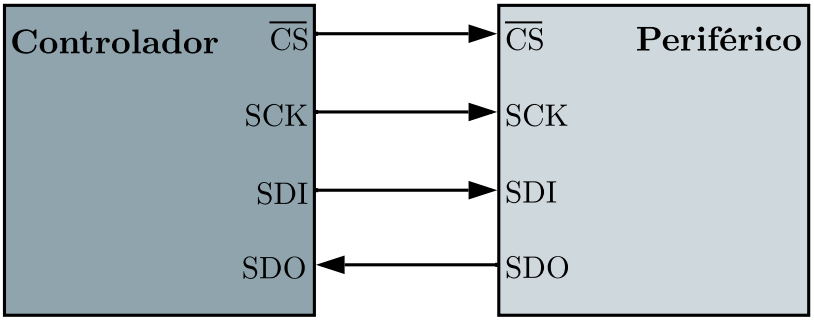


Figura 3.1: Diagrama de Bloque del Protocolo SPI

3.2.3. Transferencia de datos

El nodo o dispositivo controlado es el encargado de establecer los parámetros de transferencia de datos, entre los que se encuentra [18] [20]:

- Frecuencia de generación de reloj (señal `SCLK`): influye en la velocidad de transferencia de datos. El protocolo SPI presenta una tasa de transmisión máxima de 10 Mbps [21].
- Longitud de una palabra: define el tamaño del dato a transmitir. Comúnmente y para los propósitos de este documento, se asume que la longitud de una palabra es de 1 byte.

- Dirección de transmisión: define el primer bit de la palabra a transmitir, el bit más significativo (MSB) o el menos significativo (LSB).

De esta forma, se define el estado inactivo o de reposo del SPI cuando la señal `CS` se encuentra inactiva. Típicamente, la señal `CS` es activa en bajo, lo que implica que el controlador debe enviar un '0' lógico para habilitar la comunicación con el dispositivo periférico.

Para iniciar la comunicación SPI, el nodo controlador activa las señales `SCLK` y `CS`, el cual establece el inicio para la transmisión de datos. Durante las transacciones SPI los bits se desplazan serialmente hacia la línea `SDO` mientras que los datos entrantes se muestrean desde la línea `SDI`, todo esto sincronizado por la señal de reloj [19].

3.2.4. Sincronización

El flanco de la señal de reloj `SCLK` es el encargado de sincronizar el desplazamiento y el muestreo de los datos en el protocolo SPI. Por medio de los bits `CPOL` y `CPHA` en el nodo controlador es posible seleccionar el flanco ascendente o descendente del reloj para muestrear y/o desplazar los datos. La polaridad del reloj (`CPOL`), como su nombre lo indica, determina la polaridad del reloj durante el estado inactivo. Durante este estado la línea del reloj puede estar en bajo (`CPOL` = 0) o en alto (`CPOL` = 1). La fase del reloj (`CPHA`) define en cuál flanco de reloj se muestrea y/o desplazan los datos [22]. En la tabla 3.1 se definen los cuatro modos SPI disponibles dado los bit de `CPOL` y `CPHA`.

Tabla 3.1: Modos SPI

Modo SPI	CPOL	CPHA	Flanco de Reloj
0	0	0	Muestreo en flanco ascendente. Desplazamiento en flanco descendente.
1	0	1	Muestreo en flanco descendente. Desplazamiento en flanco ascendente.
2	1	0	Muestreo en flanco descendente. Desplazamiento en flanco ascendente.
3	1	1	Muestreo en flanco ascendente. Desplazamiento en flanco descendente.

En la figura 3.2 se observa el diagrama de tiempo del protocolo SPI en modo 0.

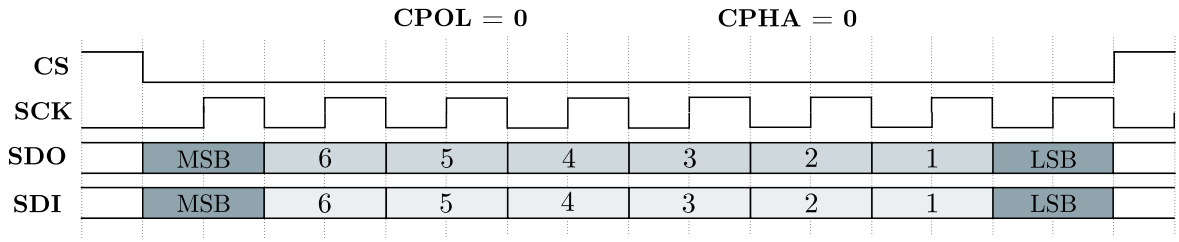


Figura 3.2: Diagrama de Tiempo del Protocolo SPI Modo 0

3.3. Convertidores USB a serial

Los convertidores USB a serial son componentes de hardware que posibilitan a un dispositivo con puerto USB comunicarse con dispositivos que utilizan protocolos de comunicación serial. Estos permiten una comunicación bidireccional procesando las señales USB en señales seriales y viceversa.

3.3.1. FT232H

El FT232H es un USB 2.0 de alta velocidad (480 Mb/s) a UART/FIFO, conforma la 6ª generación de IC fabricados por Future Technology Devices International (FTDI) [23]. Tiene la particularidad de integrar un "Multi-Protocol Synchronous Serial Engine" (MPSSE), lo que le permite al FT232H interactuar de manera eficiente con protocolos síncronos; entre los que se encuentran JTAG, I2C, SPI (Controlador) y bit-bang [23]. Adicionalmente cuenta con pines digitales que utiliza para leer y escribir, los cuales pueden ser utilizados libremente por el usuario. En la figura 3.3 se visualiza el esquemático del IC FT232H.

El MPSSE es completamente configurable y su programación se realiza mediante la transmisión de comandos a través del flujo de datos. Dichos comandos pueden enviarse de forma individual o en paquetes, donde este último es el más eficiente. MPSSE es capaz de alcanzar una velocidad de datos máxima sostenida de 30 Mbits/s [23].

3.3.2. Placa de conexión

Los ICs, como el FT232H, son dispositivos electrónicos miniaturizados que contienen múltiples componentes interconectados, lo cual puede presentar ciertas limitaciones en términos de accesibilidad y facilidad de uso para la mayoría de los usuarios.

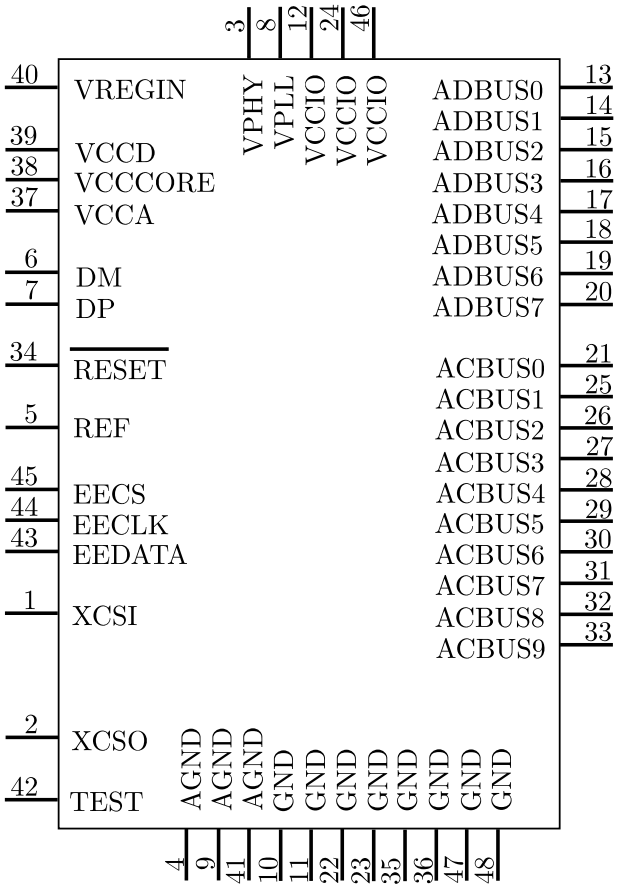


Figura 3.3: Esquemático del IC FT232H [23]

Para abordar estas limitaciones inherentes, Adafruit ha diseñado la placa de conexión Adafruit FT232H Breakout. Esta placa proporciona una interfaz física robusta y fácil de usar para el IC FT232H, lo anterior permite a los usuarios aprovechar sus funcionalidades sin tener que lidiar con las complejidades de trabajar directamente con el IC.

La placa Adafruit FT232H Breakout proporciona una interfaz física para los pines ADBUS y ACBUS del IC FT232H (ver figura 3.3), lo que permite su uso en pines GPIO. Los pines ADBUS (D0 a D7) en el chip FT232H se utilizan para UART serial y otros protocolos seriales (ver figura 3.4). Algunos de estos pines también pueden funcionar como GPIO. Los pines ACBUS (C0 a C9) en el chip FT232H se destinan únicamente como GPIO. Sin embargo, es importante tener en cuenta que los pines C8 y C9 no son controlados por software y su funcionalidad solo puede modificarse cambiando la EEPROM del chip [24].

3.3.3. SPI-Controlador

Este documento se enfoca en el uso del FT232H como un dispositivo controlador para protocolo SPI. Para ello, el IC se debe configurar mediante comandos de software y realizar

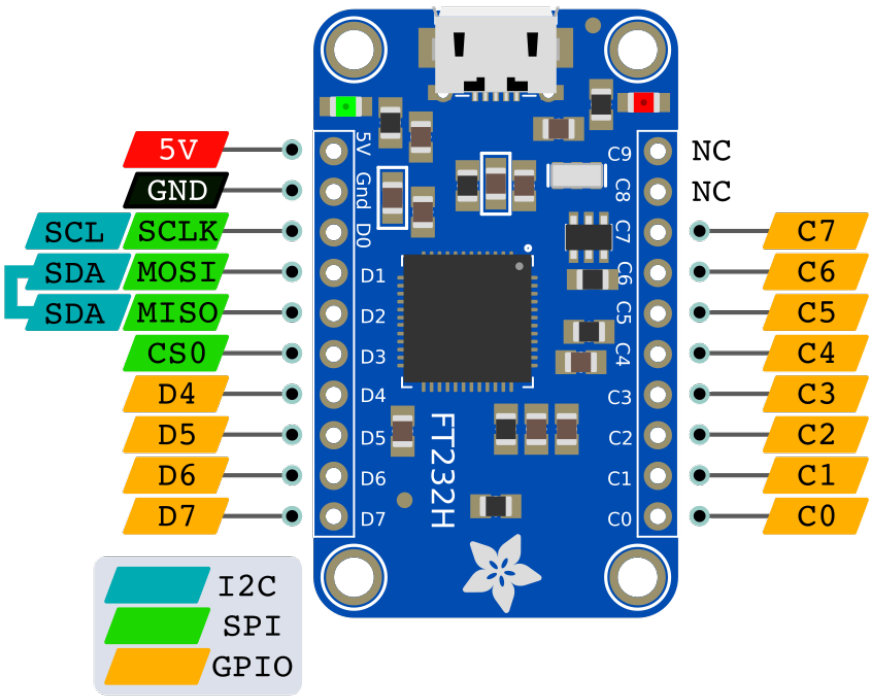


Figura 3.4: Pines de Entrada y Salida del FT232H Breakout [25]

las correctas conexiones de línea (ver figura 3.4).

Para la transferencia de datos el IC emplea dos búfer tipo FIFO de 1 kbyte cada uno para el envío y recepción de los datos en donde cada uno es controlado por el "USB Protocol Engine" y el bloque de control FIFO [23]. Los datos del PC host se almacenan en el búfer de transmisión (TX) de 1 kB para ser utilizados por el MPSSE. Por otro lado, los datos del MPSSE se almacenan en el búfer de recepción (RX) de 1 kB para ser enviados al PC host cuando se solicite.

Tanto la frecuencia de reloj `SCLK`, como el modo SPI se configuran mediante software, así como la asignación del número de periféricos.

3.3.4. Software

El FT232H es un dispositivo que puede ser controlado mediante software por medio del lenguaje de programación de alto nivel Python. Se hace uso de la biblioteca `PyFtdi`, lo que permite la interacción con dispositivos FTDI desde el espacio de usuario [26].

Para configurar el FT232H como un SPI-Controlador, se emplea la clase `SpiController`, lo que instancia un controlador SPI. Posteriormente se realizan las respectivas configuraciones para comunicarse con dispositivos SPI.

3.4. Tarjeta microSD

Las tarjetas SD (Secure Digital) son tarjetas de memoria no volátiles diseñadas únicamente para el almacenamiento de datos. Las tarjetas microSD son tarjetas SD con un encapsulado de menor tamaño. Son utilizadas comúnmente para aplicaciones multimedia en dispositivos como teléfonos celulares o cámaras, además son ampliamente utilizadas en sistemas digitales como microprocesadores, microcontroladores, chips DPS o FPGA. Entre sus ventajas se encuentran su movilidad, alta capacidad de almacenamiento, altas tasas de transferencia, bajo costo, bajo consumo de energía y alto grado de adaptación [27].

La información presentada en esta sección se basa en el análisis y la interpretación del documento "SD Specifications Part 1 Physical Layer Specification" [28].

3.4.1. Capacidad de memoria

Las tarjetas microSD se clasifican en 3 tipos en términos de capacidad de memoria:

- **SDSC** (Standard Capacity SD Memory Card) admiten una capacidad de hasta 2GB (2^{32}bytes).
- **SDHC** (High Capacity SD Memory Card) admiten una capacidad de más 2GB (2^{32}bytes) hasta 32GB (2^{35}bytes).
- **SDXC** (Extended Capacity SD Memory Card) admiten una capacidad de más 32GB (2^{35}bytes) hasta 2TB (2^{41}bytes).

3.4.2. Protocolo de comunicación

Las tarjetas microSD admiten dos protocolos de comunicación: SD y SPI. La tarjeta reconoce qué protocolo solicita el host durante el comando de reinicio. Para los propósitos de este documento, el protocolo SD fue el utilizado para la comunicación de las tarjetas SD.

El protocolo SD es una interfaz síncrona punto a punto basada en nodos. En general, presenta un nodo controlador (host) y múltiples nodos periféricos (tarjetas SD).

3.4.3. Líneas bidireccionales

El bus de un protocolo SD presenta la característica de contar con líneas bidireccionales, las cuales pueden funcionar como entradas o como salidas en un circuito digital. Esto se logra mediante el término "tri-state", el cual indica que una línea lógica puede existir en tres estados posibles:

- **Alto** (‘1’ lógico)
- **Bajo** (‘0’ lógico)
- **Alta impedancia** (Z)

El estado de alta impedancia es la principal diferencia entre una línea tri-state y una línea binaria. Cuando se encuentra en alta impedancia, la línea de señal se desconecta del circuito. Esta característica facilita que múltiples dispositivos compartan la misma línea de señal sin interferir entre sí. Para lograr esto, un solo dispositivo debe estar en un estado alto o bajo en un momento dado, mientras que todos los demás deben estar en estado de alta impedancia.

La tarjeta microSD se encuentra en modo de entrada cuando la línea esté en estado de alta impedancia y en modo de salida en cualquier otro estado.

3.4.4. Bus SD

El bus de un protocolo SD presenta las siguientes líneas lógicas:

- **CMD**: línea bidireccional encargada de transmitir los comandos y respuestas. Un ‘comando’ es un token que envía el host para iniciar una operación; una ‘respuesta’ es un token emitido por la tarjeta en respuesta a un comando.
- **DAT**: línea bidireccional encargada de transferir los datos de la tarjeta al host o viceversa.
- **CLK**: línea unilateral generada por el host, es la encargada de sincronizar los nodos mediante una señal de reloj.

Tanto en la línea CMD como en la línea DAT, los bits se transmiten de forma serial en donde el bit más significativo (MSB) se transmite primero y el menos significativo (LSB) se transmite de último.

Modo de operación SD

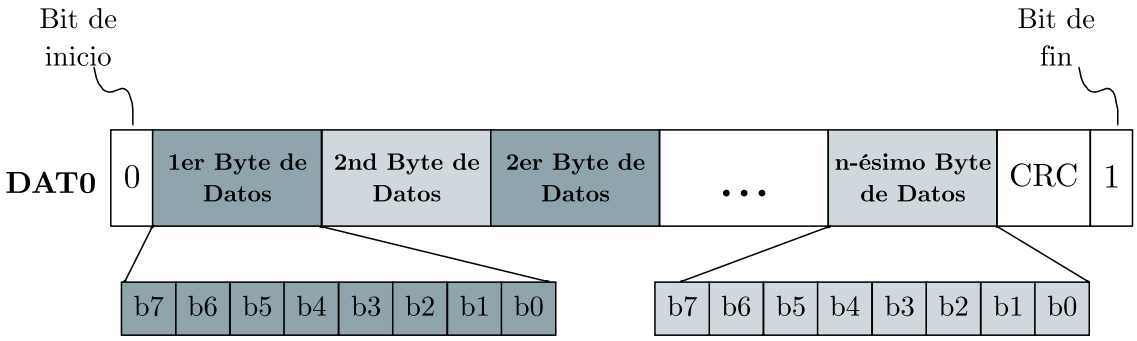
Las tarjetas SD presentan dos modos de operación, los cuales definen el ancho del bus DAT utilizado para la transmisión de los datos y el tipo de formato que tiene el bloque de datos:

- **Modo 1-bit**: transmite los datos en una única línea (`DAT0`).
- **Modo 4-bits**: transmite los datos en cuatro líneas en paralelo (`DAT3` a `DAT0`).

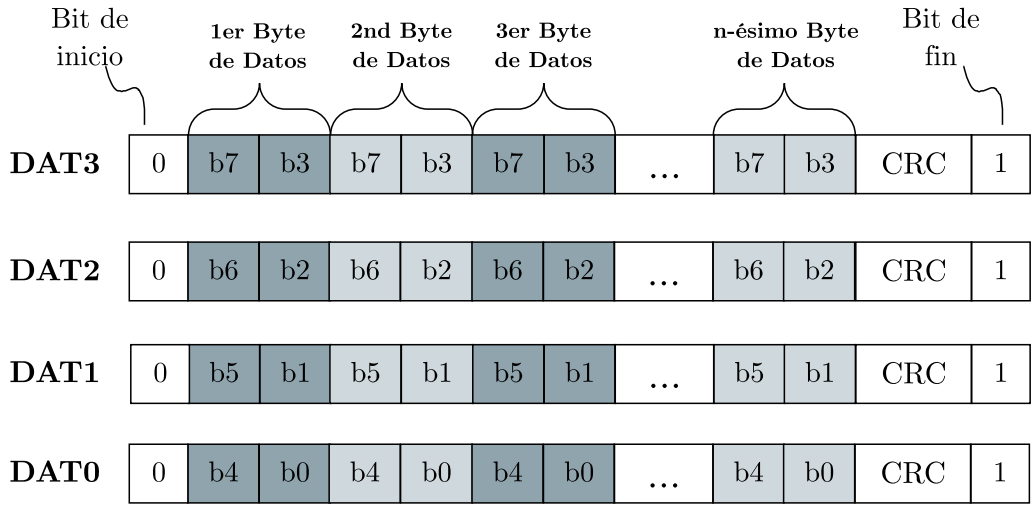
El modo 4-bits permite una mayor velocidad de transferencia de datos en comparación con el modo de 1-bit al transferir cuatro bits de datos de forma simultánea en lugar de uno. Por defecto la tarjeta microSD inicia en modo 1-bit y para cambiar el modo de operación SD se debe enviar el comando `SET_BUS_WIDTH` (ver sección 3.4.9).

Formato del bloque de datos

En la figura 3.5 se muestra el formato del bloque de datos según el modo de operación SD. El bit de inicio corresponde a un '0' lógico y el bit final a un '1' lógico. Cada dato del bloque, sin importar el modo de operación SD, es 1 byte (8 bits). Cada bloque de datos contiene un CRC (ver sección 3.4.11), posterior al último dato enviado.



(a) Modo 1-bit



(b) Modo 4-bit

Figura 3.5: Formato del Paquete de Datos

Topología

La figura 3.6 muestra la topología del bus del sistema de tarjetas de memoria SD. En la velocidad por defecto, los buses de todas las tarjetas SD están conectadas entre sí en el host. Sin embargo, en UHS-I (ver sección 3.4.8) el host habilita un único bus para comunicarse con una tarjeta microSD específica. En ambos casos, se puede acceder a una tarjeta mediante el comando `SELECT/DESELECT_CARD` (ver sección 3.4.9).

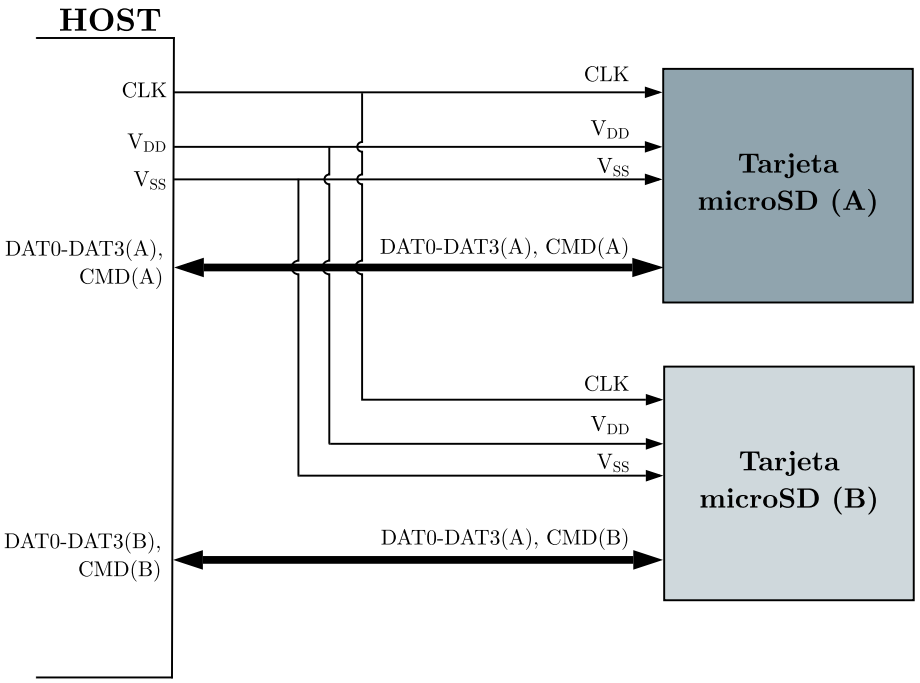
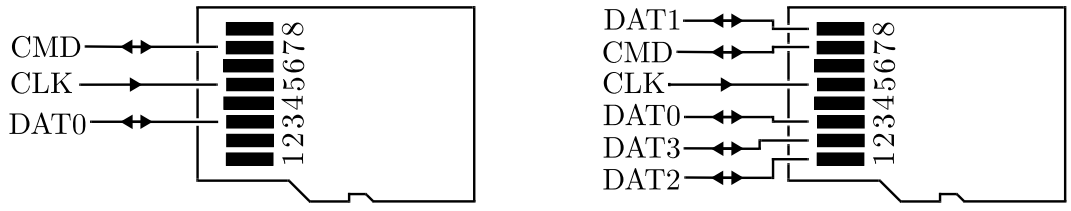


Figura 3.6: Topología del Bus SD

3.4.5. Pines

La figura 3.7 muestra la forma general y los contactos de interfaz de la tarjeta microSD. La tabla 3.2 define los contactos de la tarjeta para un protocolo SD.



(a) Modo 1-bit **(b)** Modo 4-bit

Figura 3.7: Forma e Interfaz de la Tarjeta microSD (vista inferior)

Tabla 3.2: Contactos de la tarjeta microSD para un protocolo SD

	Protocolo SD		
Pin	Nombre	Tipo	Descripción
1	CD/DAT3[1]	I/O/PP[2]	Tarjeta Detectada/Línea de datos (bit 3).
2	CMD	I/O/PP	Comando/Respuesta.
3	V_{SS1}	S	Fuente de alimentación a tierra.
4	V_{DD}	S	Fuente de alimentación.
5	SCK	I	Reloj.
6	V_{SS2}	S	Fuente de alimentación a tierra.
7	DAT0	I/O/PP	Línea de datos (bit 0).
8	DAT1[3]	I/O/PP	Línea de datos (bit 1).
9	DAT2	I/O/PP	Línea de datos (bit 2).

3.4.6. Control de reloj

La tarjeta microSD utiliza el flanco ascendente del reloj para muestrear los datos y el flanco descendente del reloj para el desplazamiento de los datos. El host puede ajustar la frecuencia de reloj dentro de los rangos definidos por el modo de velocidad establecido (ver sección 3.4.8) o deshabilitarlo.

Durante el estado de identificación de tarjeta (ver sección 3.4.14) el host debe operar con una frecuencia de reloj en el rango de (100-400) kHz. Una vez superado el estado, el host es libre de ajustar la frecuencia de reloj según sus necesidades.

Es obligatorio mantener el reloj habilitado para que la tarjeta emita datos o tokens de respuesta.

[1]Las líneas DAT extendidas (DAT1 a DAT3) son operativas hasta después del comando SET_BUS_WIDTH (ver sección 3.4.9). El host mantendrá sus líneas DAT1 a DAT3 en modo de entrada mientras no se utilicen (alta impedancia).

[2]S: fuente de alimentación; I: entrada; O: salida mediante controlador push-pull; PP: I/O mediante controladores push-pull

[3]Al encender la tarjeta, la línea tiene una resistencia pull-up de 50 kOhms habilitada en la tarjeta. La resistencia tiene dos funciones: detección de tarjeta y selección de protocolo de comunicación. Para la detección de la tarjeta, el host detecta la línea en alto. Para selección de tarjeta, el host puede llevar la línea en alto o dejar la línea en alto para seleccionar el modo SD. Si el host pone la línea en bajo, se seleccionará el modo SPI. La resistencia pull-up debe ser desconectada por el host durante la transferencia de datos mediante el comando SET_CLR_CARD_DETECT (ver sección 3.4.9).

3.4.7. Registros de información

Cada tarjeta microSD cuenta con un conjunto de registros de información que contienen detalles relevantes sobra la misma. En la tabla 3.3 se muestran los registros de interés para este documento.

Tabla 3.3: Registros de Información

Nombre	Ancho	Descripción
`CID`	128	▪ Número de identificación de la tarjeta. ▪ Número individual de la tarjeta para su identificación.
`OCR`	32	Registro de condiciones de operación.
`RCA`	16	▪ Dirección relativa de la tarjeta. ▪ Dirección del sistema local de una tarjeta, sugerida dinámicamente por la tarjeta y aprobada por el host durante la inicialización.

3.4.8. UHS-I

UHS-I (Ultra-High Speed Phase I) es un estándar de velocidad para tarjetas SD que permite velocidades máximas teóricas de transferencia de hasta 104MB/s en un bus de 4-bits con interfaz de controlador de un solo periférico [28].

Modos de velocidad de bus

En la tabla 3.4 se definen los modos de velocidad disponibles para UHS-I:

Tabla 3.4: Modos de Velocidad de Bus

Modo de velocidad del bus	Máxima velocidad del bus [MB/s]	Máxima frecuencia de reloj [MHz]	Voltaje [V]	Max. Corriente [mA/3.6V VDD]		
				SDSC	SDHC	SDXC
SDR104	104	208	1.8	-	800	800
SDR50	50	100	1.8	-	400	400
DDR50	50	50	1.8	-	400	400
SDR25	25	50	1.8	-	200	200
SDR12	12.5	25	1.8	-	100	100/150
High Speed	25	50	3.3	200	200	200
Default Speed	12.5	25	3.3	100	200	100/150

Velocidades de escritura

La tabla 3.5 presenta las velocidades de escritura mínimas de una tarjeta microSD categorizadas por el fabricante. Estas velocidades son inalterables y están determinadas por las especificaciones del dispositivo.

Tabla 3.5: Velocidades de Escritura Mínima [29]

Velocidad de escritura mínima [MB/s]	Categoría de velocidad[4]		
	Clase de Velocidad[5]	Grado de Velocidad UHS-I	Clase de Velocidad de Video[6]
			
90			V90
60			V60
30		U3	V30
10	C10	U1	V10
6	C6		V6
4	C4		
2	C2		

3.4.9. Comandos

Tipos de comandos

Existen cuatro tipos de comandos definidos para controlar la tarjeta microSD:

- **Comando broadcast sin respuesta** (`bc`): el host envía un token a cada tarjeta microSD conectado al host. La(s) tarjeta(s) no remitirá(n) un token de respuesta. Este comando solo aplica si cada línea `CMD` de los periféricos están conectadas entre sí. Si están separadas, cada periférico aceptará el comando por separado en su respectivo turno.
- **Comando broadcast con respuesta** (`bcr`): el host envía un token a una tarjeta microSD. La tarjeta acepta el comando y emite un token de respuesta. Este comando solo aplica si todas las líneas `CMD` están separadas, el comando es aceptado y respondido por cada tarjeta por separado.

[4] Cada rango de las categorías tienen soporte para manejar rangos inferiores a él.

[5] Un host UHS-I es compatible con esta categoría. Representa la velocidad de escritura mínima de una tarjeta microSD, una tarjeta SDXC conectada a un host sin UHS-I utilizará esta categoría de velocidad.

[6] La clase de velocidad de vídeo es una nueva tecnología incorporada en 2016, está diseñada para soportar velocidades de transferencia de archivos multimedia en 4k [29]. Esta categoría de velocidad es compatible con UHS-I.

- **Comando direccionado sin transferencia de datos en DAT (`ac`)**: el host envía un token a una tarjeta previamente seleccionada mediante el comando `SELECT_DESELECT_CARD` (ver más adelante). La tarjeta microSD acepta el comando y emite un token de respuesta.
- **Comando direccionado con transferencia de datos en DAT (`adtc`)**: el host envía un token a una tarjeta microSD previamente seleccionada mediante el comando `SELECT_DESELECT_CARD` (ver más adelante). La tarjeta micro SD acepta el comando y cambia su estado para la transmisión o recepción de datos (ver 3.4.15). Posteriormente emite un token de respuesta.

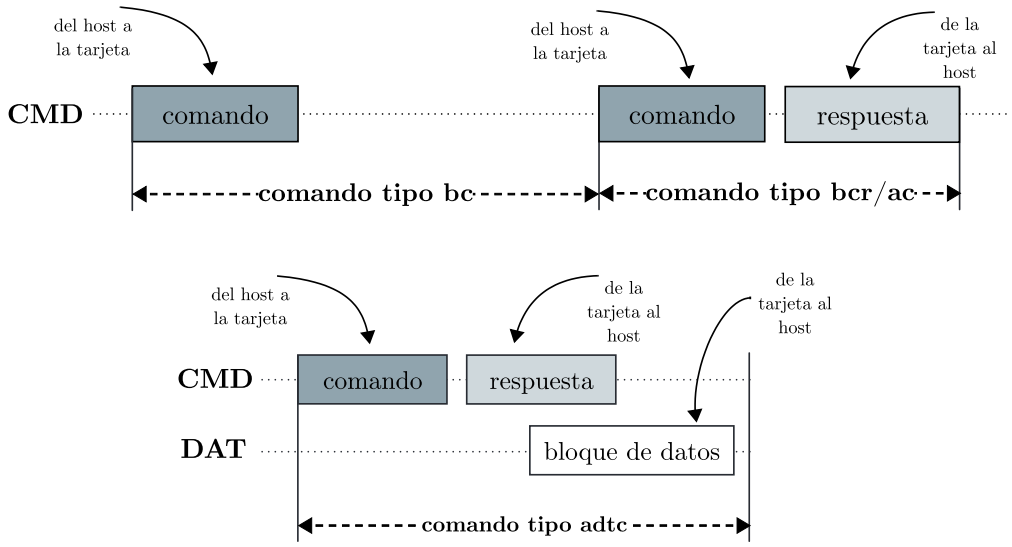


Figura 3.8: Tipos de Comandos

Formato del comando

Todos los comandos tienen una longitud de código fija de 48 bits. La siguiente tabla muestra el formato de un comando de una tarjeta microSD.

Tabla 3.6: Formato del Comando

Posición	[47]	[46]	[45:40]	[39:8]	[7:1]	[0]
Bits	1	1	6	32	7	1
Valor	'0'	'1'	x	x	x	'1'
Descrip.	Bit de inicio	Bit de transmisión[7]	Índice de comando[8]	Argumento[9]	CRC7[10]	Bit de fin

[7] Un '1' en el bit de transmisión se transmite un token del host a la tarjeta microSD (comando) y un '0' que se transmite un token de la tarjeta al host (respuesta).
[8] Representa el valor hexadecimal del número del comando.
[9] Este campo varía según el comando a utilizar.
[10] Cada código de comando está protegido por un CRC7 (ver sección 3.4.11).

Lista de comandos

La tabla 3.7 describe todos los comandos estándar para la tarjeta microSD de interés para este documento. La tarjeta microSD ignora los "stuff bits" (bits de relleno) y los bits reservados en un argumento. Los "stuff bits" son bits que la operación del comando no requiere, los bits reservados son bits utilizados por el token de respuesta y no del comando. Ambos campos en este documento se establecen en 0.

Tabla 3.7: Definición de Valores de `CRC status`

CMD	Tipo	Argumento	Resp.	Abreviatura	Descripción
CMD0	ac	[31:0] stuff bits	R1	GO_IDLE_STATE	Restablece todas las tarjetas al estado inactivo (ver sección 3.4.14).
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Solicita a cualquier tarjeta que envíe el número CID de la línea CMD.
CMD3	bcr	[31:0] stuff bits	R6	SEND_ RELATIVE_ADDR	Solicita a la tarjeta que publique un nuevo número RCA (ver sección 3.4.14).
CMD7	ac	[31:16] RCA [15:0] stuff bits	R1b	SELECT_ DESELECT_CARD	Alterna una tarjeta entre los estados de stand-by y transfer (ver 3.4.15).
CMD8	bcr	[31:12] bits reservados [11:8] VHS [7:0] check pattern	R7	SEND_IF_COND	Envía la condición de la interface del host, la cual incluye información sobre el voltaje de alimentación del host (VHS) y pregunta a la tarjeta microSD si la tarjeta admite el voltaje (ver sección 3.4.14).
CMD12	ac	[31:0] stuff bits	R1b	STOP_ TRANSMISSION	Fuerza a la tarjeta a detener la transmisión de datos (lectura o escritura).
CMD24	adtc	[31:0] dirección	R1	WRITE_BLOCK	Escribe un bloque de datos.

Tabla 3.7: Definición de Valores de CRC status

CMD25	adtc	[31:0] dirección	R1	WRITE MULTIPLE_BLOCK	Escribe continuamente bloques de datos hasta que se produce un STOP_TRANSMISSION (ver sección 3.4.15).
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indica a la tarjeta que el siguiente comando es un comando específico de la aplicación en lugar de un comando estándar.

La tabla 3.8 describe todos los comandos específicos de aplicación para la tarjeta microSD de interés para este documento. Los comandos ACMD proporcionan una extensión a los comandos estándares y pueden tener el mismo número de CMD. Para que la tarjeta reconozca el comando como un ACMD se debe enviar tras un comando APP_CMD.

Tabla 3.8: Lista de Comandos Específicos de Aplicación (ACMD)

CMD	Tipo	Argumento	Resp.	Abreviatura	Descripción
ACMD6	ac	[31:0] stuff bits [1:0] modo de operación	R1	SET_BUS_WIDTH	Define el modo de operación: "00" = modo 1-bit, "10" = modo 4 bits.
ACMD41	bcr	[31] bit reservado [30] HCS [29] bit reservado [28] XPC [27:25] bits reservados [24] S18R [23:8] OCR [7:0] bits reservados	R3	SD_SEND_OP_COND	Envía información de soporte de capacidad de host (HCS) y solicita a la tarjeta a la que accede que envíe su contenido de registro OCR en la respuesta en la línea CMD (ver 3.4.14).
ACMD42	ac	[31:1] stuff bits [0] set_cd	R1	SET_CLK_CARD_	Mediante set_cd = "1" conecta y set_cd = "0" desconecta la resistencia pull-up de 50 kOhm en CD/DAT3.

3.4.10. Respuestas

Todas las respuestas se envían a través de la línea de comandos `CMD` donde el bit de transmisión tiene un valor de '0' lógico. Estas presentan una longitud de 48 bits, a excepción de `R2`. Así mismo, todas las respuestas excepto `R3` están protegidas por un CRC7 (ver sección 3.4.11).

Respuesta R1

Representa el token de respuesta para la mayoría de comandos. La siguiente tabla muestra el formato de un tipo de respuesta `R1` de una tarjeta microSD.

Tabla 3.9: Formato de Respuesta R1

Posición	[47]	[46]	[45:40]	[39:8]	[7:1]	[0]
Bits	1	1	6	32	7	1
Valor	'0'	'0'	x	x	x	'1'
Descrip.	Bit de inicio	Bit de transmisión	Índice de comando[11]	Estado de la tarjeta[12]	CRC7	Bit de fin

Respuesta R1b

La respuesta `R1b` es un token de respuesta `R1` con la salvedad de que presenta una señal de ocupado opcional transmitida en la línea de datos. El host debe comprobar si la tarjeta microSD está ocupada. Una señal de ocupado es un '0' lógico enviado por la tarjeta en la línea `DAT0`, mientras esté activa la tarjeta rechaza cualquier información que envíe el host, ya sea un comando por la línea `CMD` o datos por las líneas `DAT`. El host debe esperar a que la señal de ocupado cambie a '1' lógico para continuar con el flujo normal de operación.

[11] La tarjeta envía el valor hexadecimal que representa el número del comando al que responde.
[12] La respuesta contiene el estado de la tarjeta microSD según el comando enviado.

Respuesta R2

La respuesta R2 tiene una longitud de 136 bits y contiene los bits [127:1] del contenido del registro CID ante el envío de un comando CMD2 por parte del host. La siguiente tabla muestra el formato de una respuesta R2.

Tabla 3.10: Formato de Respuesta R2

Posición	[135]	[134]	[133:128]	[127:8]	[7:1]	[0]
Bits	1	1	6	120	7	1
Valor	'0'	'0'	'111111'	x	x	'1'
Descrip.	Bit de inicio	Bit de transmisión	Reservado	CID	CRC7	Bit de fin

Respuesta R3

La respuesta R3 abarca el contenido del registro OCR como respuesta a ACMD41. Esta respuesta envía una cadena de '1' lógicos en lugar de CRC7. La siguiente tabla muestra el formato de una respuesta R3.

Tabla 3.11: Formato de Respuesta R3

Posición	[47]	[46]	[45:40]	[39:8]	[7:1]	[0]
Bits	1	1	6	32	7	1
Valor	'0'	'0'	'111111'	x	'1111111'	'1'
Descrip.	Bit de inicio	Bit de transmisión	Reservado	OCR	Reservado	Bit de fin

Respuesta R6

La respuesta R6 envía una nueva dirección dinámica para la tarjeta microSD ante un comando CMD3. El formato para una respuesta R6 se muestra en la siguiente tabla.

Tabla 3.12: Formato de Respuesta R6

Posición	[47]	[46]	[45:40]	[39:24]	[25:8]	[7:1]	[0]
Bits	1	1	6	16	16	7	1
Valor	'0'	'0'	x	x	x	x	Ì'
Descrip.	Bit de inicio	Bit de transmisión	Índice del comando	Nueva RCA publicada (argumento[31:16])	Estado de la tarjeta (argumento[15:0])	CRC7	Bit de fin

Respuesta R7

La respuesta R7 envía la información del voltaje admitida por la tarjeta como respuesta ante un comando CMD8. La siguiente tabla muestra el formato de una respuesta R7.

Tabla 3.13: Formato de Respuesta R7

Posición	[47]	[46]	[45:40]	[39:20]	[19:16]	[15:8]	[7:1]	[0]
Bits	1	1	6	18	4	8	7	1
Valor	'0'	'0'	x	'0x00000'	x	x	x	'1'
Descrip.	Bit de inicio	Bit de transmisión	Índice del comando	Reservado	Voltaje Aceptado (VHS)	Check pattern	CRC7	Bit de fin

La siguiente tabla muestra los valores y su respectiva definición del campo VHS.

Tabla 3.14: Definición de Valores de VHS

Valor	Definición
4'b0000	No definido
4'b0001	2.7-3.6V
4'b0010	Reservado para rango de baja tensión
4'b0100	Reservado
4'b1000	Reservado
Otros	No definido

3.4.11. CRC

El CRC está diseñado para proteger los comandos, las respuestas y la transferencia de datos de la tarjeta microSD contra errores de transmisión en el bus de la tarjeta microSD.

CRC7

Un CRC7 es código binario de 7 bits que se utiliza para proteger todos los comandos y todas las respuestas (excepto R3), se calcula mediante las siguientes ecuaciones:

$$G(x) = x^7 + x^3 + 1 \tag{3.1}$$

$$M(x) = \text{primer bit} \cdot x^n + \text{segundo bit} \cdot x^{n-1} + \ldots + \text{último bit} \cdot x^0 \tag{3.2}$$

$$\text{CRC[6:0]} = \text{Remanente}\left[\frac{M(x)\cdot x^7}{G(x)}\right] \tag{3.3}$$

El primer bit es el MSB del comando o respuesta. El grado `n` del polinomio es el número de bits protegidos por CRC disminuido en uno. El número de bit a proteger para un CRC7 es de 40 para todos los comandos y respuestas (`n` = 39), excepto la respuesta `R2`, la cual es de 136 (`n` = 135).

El host debe generar el CRC7 del comando y comprobar el CRC7 de la respuesta. Si el valor CRC7 enviado por el host no coincide con el generado en la tarjeta durante la comprobación, la tarjeta rechaza el comando y no envía ninguna respuesta. Si el valor CRC7 enviado por la tarjeta no coincide con el generado por el host durante la comprobación, el host determina cómo proceder. La figura 3.9 muestra un diagrama de bloques de un generador/comprobador CRC7.

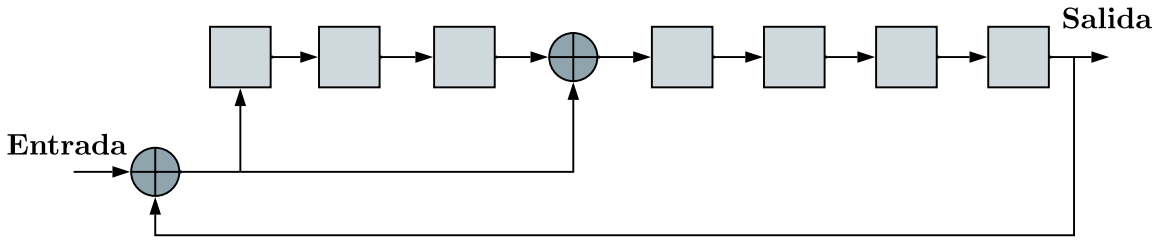


Figura 3.9: Diagrama de Bloques de un Generador/Comprobador CRC7

CRC16

El CRC16 es un código binario de 16 bits que se utiliza para la protección de transferencia de bloques de datos en la línea `DAT`, se calcula mediante las siguientes ecuaciones:

$$G(x) = x^{16} + x^{12} + x^5 + 1 \tag{3.4}$$

$$M(x) = \text{primer bit}\cdot x^n + \text{segundo bit}\cdot x^{n-1} + \ldots + \text{último bit}\cdot x^0 \tag{3.5}$$

$$\text{CRC[15:0]} = \text{Remanente}\left[\frac{M(x)\cdot x^{16}}{G(x)}\right] \tag{3.6}$$

El primer bit es el MSB del bloque. El grado `n` del polinomio es el número de bits protegidos por CRC disminuido en uno. El número de bit a proteger para un CRC16 es equivalente al tamaño del bloque de datos. Se utiliza el mismo método CRC6 para el modo 1-bit y 4-bit, con la salvedad que el modo 4-bits el CRC16 se realiza para cada línea de `DAT` por separado. El host debe generar un CRC16 de los datos enviados en cada línea `DAT` y

comprobar el CRC16 para cada línea `DAT` posterior a recibir un bloque de datos. La figura 3.10 muestra un diagrama de bloques de un generador/comprobador CRC16:

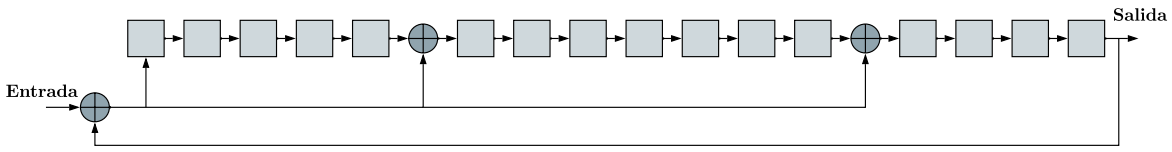


Figura 3.10: Diagrama de Bloques de un Generador/Comprobador CRC16

CRC status

Después de finalizar el envío de un bloque de datos, la tarjeta microSD responde con un `CRC status` en la línea `DAT0`. Este valor le indica al host el estado de los datos transferidos, las siguientes tablas muestran el formato de `CRC status` y la definición de cada valor respectivamente:

Tabla 3.15: Formato de `CRC status`

Posición	[4]	[3:1]	[0]
Bits	1	3	1
Valor	`0'	`x'	`1'
Descripción	Bit de inicio	CRC status	Bit de fin

Tabla 3.16: Valores de `CRC status`

Valor	**Definición**
3'b010	Sin error
3'b101	Error de transmisión
3'b111	Error de programación flash[13]
Otros	No definido

[13]La tarjeta no envía ningún `CRC status` al host y, por tanto, no hay bit de inicio.

3.4.12. Temporización

Las tarjetas microSD presentan diversos tiempos de operación según el tipo de comando y de respuesta, el host debe respetar estos tiempos. Con el propósito de facilitar la comprensión de los siguientes esquemas, la tabla 3.17 muestra las abreviaturas utilizadas.

Tabla 3.17: Símbolos del Diagrama de Tiempo

Símbolo	Descripción
S	Bit de inicio (‘0’ lógico).
T	Bit de transmisión (Host = ‘1’, Tarjeta = ‘0’).
P	Un ciclo de reloj con ‘1’ lógico.
E	Bit de fin (‘1’ lógico).
Z	Estado de alta impedancia (conducido débilmente a ‘1’).
*	Repetición.
CRC	Código de Redundancia Cíclica.
	Host activo.
	Tarjeta activa.

La tabla 3.18 define todos los valores de temporización de interés para este documento. Todos los valores están definidos para un ciclo de reloj.

Tabla 3.18: Valores de Temporización

Parámetro	Min.	Max.	Descripción
N_{CR}	2	64	Período entre un bit de fin de comando y un bit de inicio de respuesta (excepto `CMD2` y `ACMD41`).
N_{ID}	5	8	Período entre un bit de fin de comando y un bit de inicio de respuesta para `CMD2` y `ACMD41`.
N_{RC}	8	-	Período entre un bit de fin de respuesta y un bit de inicio del siguiente comando.
N_{CC}	8	-	Período entre un bit de fin de comando (`CMD0`) y un bit de inicio del siguiente comando.
N_{WR}	2	-	Período entre un bit de fin de respuesta y un bit de inicio de datos de escritura o un período entre un bit final de `CRC status` y un bit inicial de los siguientes datos de escritura.
N_{SB}	2	2	Período entre un bit final de `CMD12` y un bit inicial de ocupado durante una operación de bloque de escritura múltiple.

Temporización de comando-respuesta

La siguiente figura muestra el tiempo de respuesta de todos los comandos, con la excepción de `CMD2` y `ACMD41`.

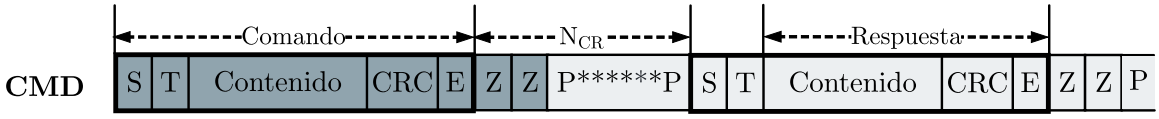


Figura 3.11: Tiempo de Respuesta de Comandos

Para el caso de `CMD2` y `ACMD41` el tiempo de respuesta se muestra en la siguiente figura.

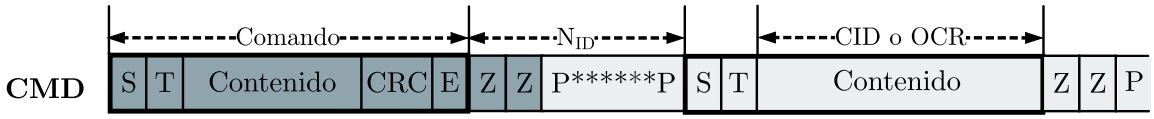


Figura 3.12: Tiempo de Respuesta de los Comandos CDM2 y ACMD41

Temporización de última respuesta-siguiente comando

Después de recibir la última respuesta de la tarjeta, el host puede iniciar la siguiente transmisión de comandos después de al menos N_{CR} ciclos de reloj, como se puede observar en la siguiente figura.

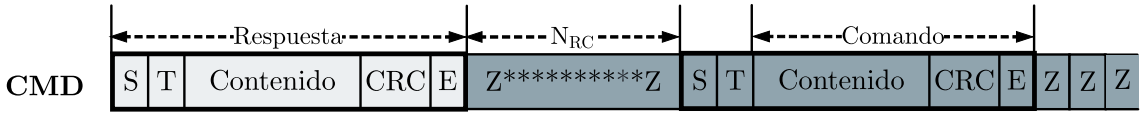


Figura 3.13: Tiempo entre Última respuesta y Siguiente Comando

Temporización posterior a un comando tipo bc

Para comandos sin respuesta, el host puede enviar el siguiente comando después de al menos N_{CC} ciclos de reloj, como se puede ver en la siguiente figura.

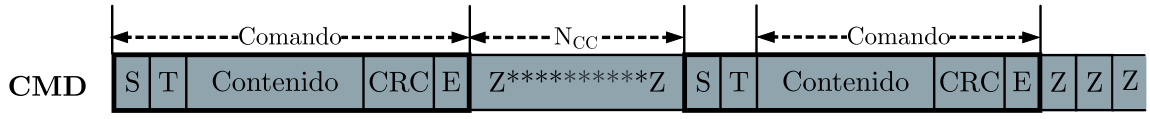


Figura 3.14: Tiempo Posterior a un Comando sin Respuesta

Temporización de escritura de bloques múltiples

Cada bloque de datos enviado por el host mediante el comando `CMD25` se escribe en la tarjeta microSD de no haber errores de transmisión (ver tabla 3.16). Una vez aceptados los datos, la tarjeta entra en estado de ocupado (ver siguiente figura), enviando un bit 'S' a través de la línea `DAT0`. El host no puede enviar otro bloque de datos hasta que la tarjeta SD envíe un bit 'E'.

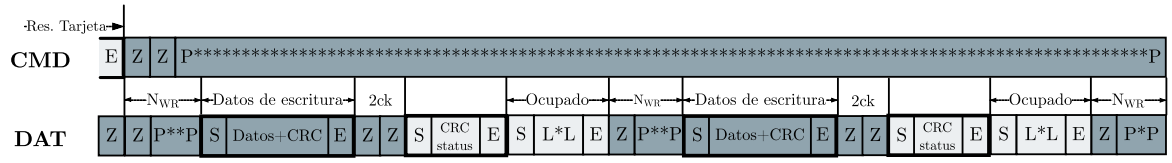


Figura 3.15: Tiempo de Escritura de Bloques Múltiples

Al detener la trasmisión de datos mediante el comando `CMD12`, la tarjeta entra en un estado de programación enviando un bit 'S' en la línea `DAT0` (ver siguiente figura). El estado finaliza la operación y se prepara para recibir otra operación. El host debe esperar que la línea `DAT0` envié un bit 'E'.

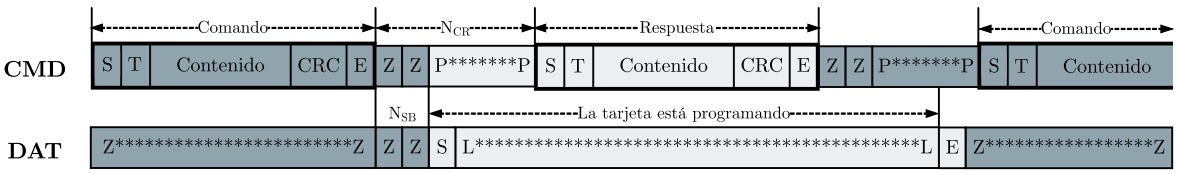


Figura 3.16: Tiempo Durante la Detención de Escritura

3.4.13. Estados de operación

El protocolo SD presenta tres estados de operación que aplican para el host y la tarjeta microSD. Cada estado de operación presenta estados de la tarjeta. La siguiente tabla contiene los estados de operación con su respectivos estados de tarjeta.

Tabla 3.19: Estados de Operación

Estado de Operación	Estado de la Tarjeta	Abreviatura
Estado Inactivo	Estado Inactivo	`ina`
Estado de Identificación de Tarjeta	Estado Idle	`idle`
	Estado Listo	`ready`
	Estado de Identificación	`ident`
Estado de Transferencia de Datos	Estado de Espera	`stby`
	Estado de Transferencia	`tran`
	Estado de Envío de Datos	`data`
	Estado de Recepción de Datos	`rcv`
	Estado de Programación	`prg`
	Estado Desconectado	`dis`

3.4.14. Estado de identificación de tarjeta

En el estado de identificación de tarjeta, el host detecta, restablece, valida, inicializa e identifica cada tarjeta microSD disponible. La figura 3.17 muestra el diagrama de flujo que describe los procesos realizados por el host en este estado. El host debe ejecutar este flujo de operación para cada tarjeta microSD.

Detección de tarjeta

Tras el encendido, el host debe comprobar la señal del pin `CD/DAT3` de cada ranura microSD. Una señal en alto indica la presencia de una tarjeta microSD. Las ranuras con la señal en bajo serán consideradas como tarjetas en estado `ina` por parte del host.

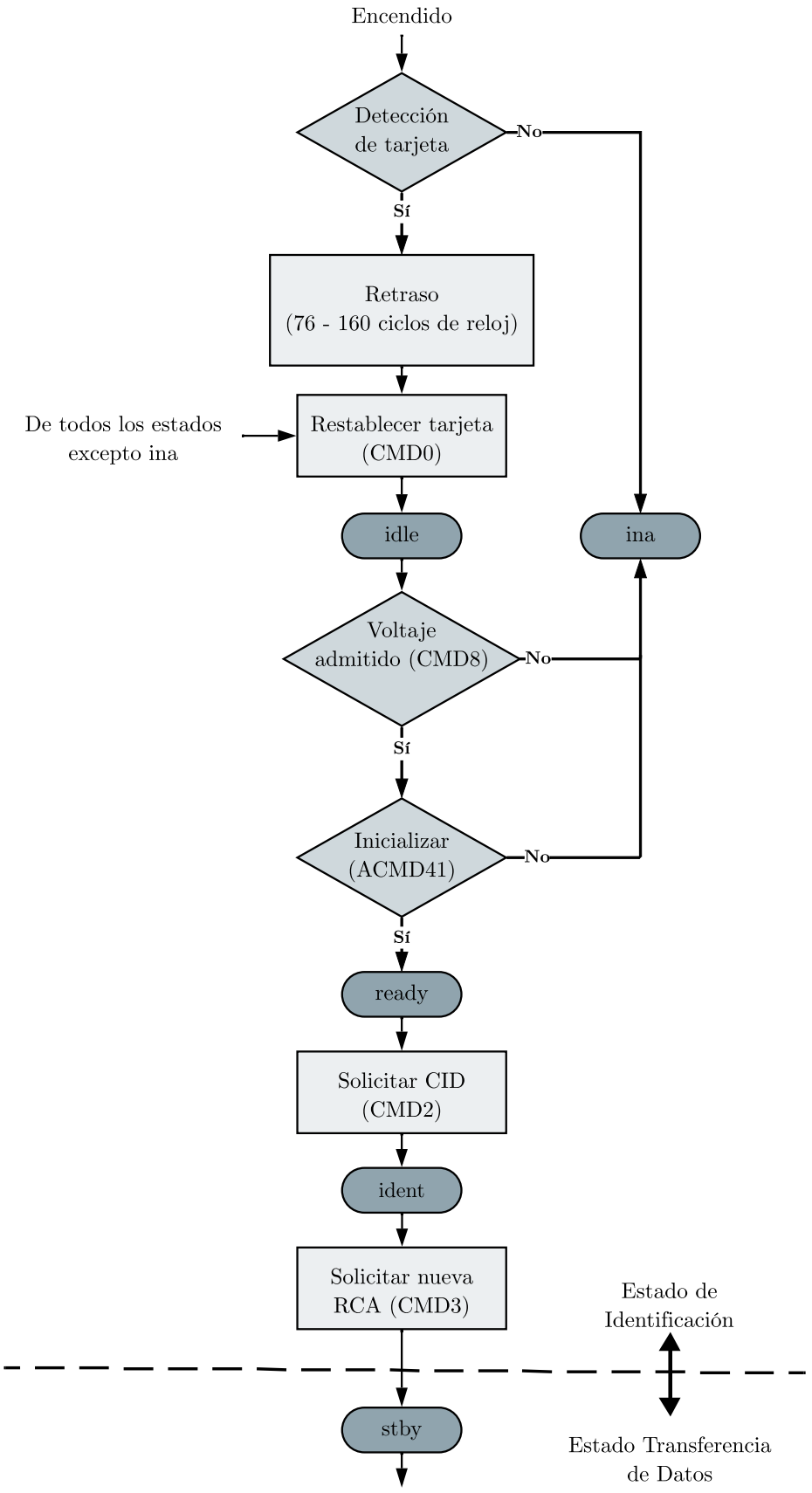


Figura 3.17: Diagrama de Flujo del Estado de Identificación de Tarjeta

Retraso de inicio

Es recomendable que después del encendido y antes de restablecer la tarjeta, el host espere alrededor de 76 - 160 ciclos de reloj antes de enviar `CMD0`. Esto con el fin de estabilizar el voltaje de alimentación (ver figura 3.18).

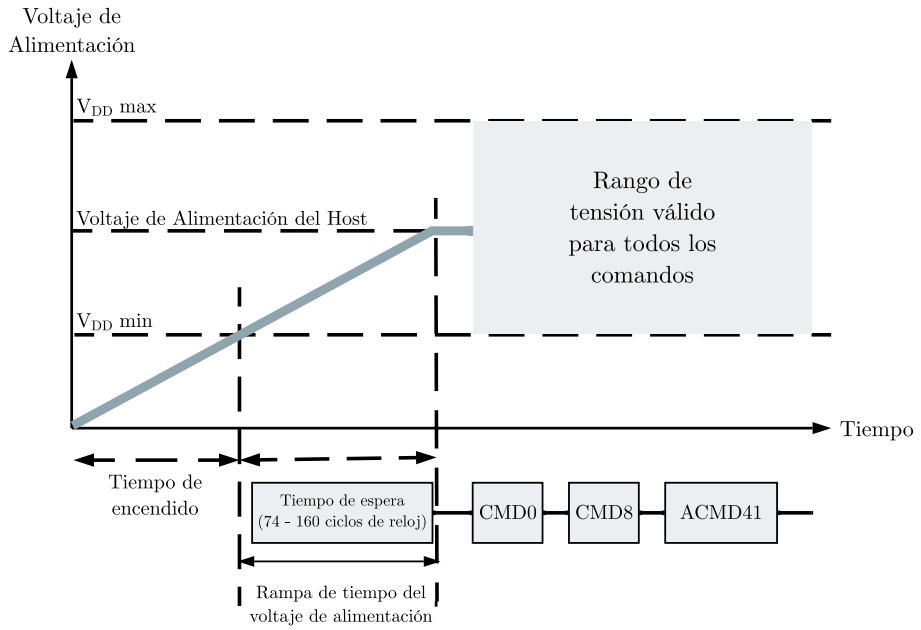


Figura 3.18: Tiempo Estabilización del Voltaje de Alimentación

Restablecer tarjeta (CMD0)

El comando `GO_IDLE_STATE` (`CMD0`) es el comando de reinicio de software y establece cada tarjeta en estado `idle` independientemente del estado actual de la tarjeta (excepto el estado `ina`). Las tarjetas activas tras el encendido se establecen en el estado `idle`, el host puede enviar `CMD0` para corroborar que la tarjeta se encuentre en dicho estado. Sin embargo, si la tarjeta ya se encuentra en estado `idle` no se ve afectada por este comando.

Después del encendido o de un comando `CMD0`, la línea `CMD` se establece en modo entrada, esperando el bit de inicio del siguiente comando. Las tarjetas se inician con una dirección de tarjeta relativa predeterminada (`RCA` = 16'h0000).

Voltaje admitido (CMD8)

Al inicio de la comunicación en el estado `idle`, el host debe comprobar cuál es la tensión admitida por la tarjeta. El host emite el comando `CMD0` con un voltaje específico, con la suposición de compatibilidad con la tarjeta. Para verificar la capacidad de la tarjeta de

funcionar con el voltaje de alimentación del host, se utiliza el comando SEND_IF_COND (CMD8).

El comando CMD8 obtiene el estado de funcionamiento de la interfaz de la tarjeta de memoria SD. La tarjeta comprueba la validez de las condiciones de funcionamiento analizando el argumento de CMD8 y el host comprueba la validez analizando la respuesta de CMD8. Los campos del argumento son los siguientes:

- **VHS**: voltaje de alimentación del host, se rige por la tabla 3.14.
- **Check pattern**: cualquier valor asignado por el host (se recomienda 8'hAA).

Si la tarjeta puede funcionar con el voltaje de alimentación del host, la respuesta se hace eco del valor 'VHS' y del 'check pattern' que se establecieron en el argumento del comando. Caso contrario, el host considera la tarjeta en estado ina.

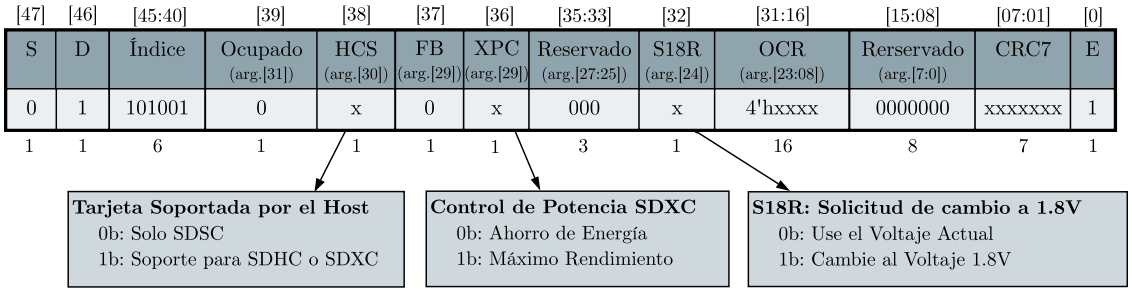
Solicitar OCR (ACMD41)

El comando SD_SEND_OP_COND (ACMD41) esta diseñado para brindar al host un mecanismo para identificar las tarjetas compatibles y rechazar aquellas que no lo son. El comando ACMD41 envía como argumento las condiciones de operación del host, la tarjeta inicia el proceso de inicialización utilizando los valores proporcionados. Una vez finalizado el host debe validar si las condiciones de operación de la tarjeta coinciden con las solicitadas. Al ser un comando de aplicación específica, ACM41 debe enviar posterior a un CMD55. El RCA del argumento en este caso es siempre el predeterminado.

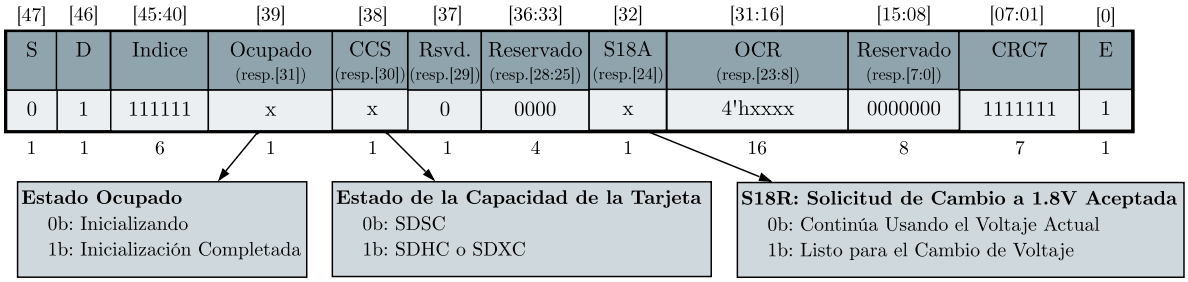
El comando ACMD41 presenta dos etapas, las cuales el host debe ejecutar en orden:

1. **Inquiry CMD41**: es el comando de consulta, la tarjeta reconoce el comando ACMD41 como Inquiry CMD41 si el campo [23:8] del argumento se establece en cero. Se utiliza para obtener el registro OCR de la tarjeta. Este comando no genera la inicialización e ignora los campos [31:24] del argumento.
2. **First ACMD41**: es el comando que genera la inicialización. La tarjeta reconoce el comando ACMD41 como First ACMD41 si el campo [23:8] del argumento se establece diferente de cero. Los campos [31:24] del argumento serán efectivos. Es importante que el campo [23:8] del argumento sea igual al registro OCR obtenido en el comando Inquiry CMD41.

Los valores del argumento y de la respuesta del comando `ACMD41` se muestran en la figura 3.19.



(a) Argumentos del Comando ACMD41



(b) Valores de Respuesta de ACMD41

Figura 3.19: Argumento y Respuesta de ACMD41

Una vez comenzada la inicialización, el host debe repetir el comando `First ACMD41` hasta que el bit de ocupado del argumento se establezca en 1 o haya transcurrido un tiempo máximo de 1 segundo. Los valores `CCD` y `S18A` de la respuesta son válidos cuando el bit de ocupado se establezca en '1'. Las tarjetas incompatibles que no logran inicializar se envían al estado `ina`. La figura 3.20 muestra el diagrama de flujo que el host debe seguir para inicializar la tarjeta. Completado exitosamente el proceso de inicialización la tarjeta entra al estado `ready`.

Tras alcanzar este punto, la tarjeta adopta el modo de velocidad predeterminada, es decir, "Default Speed". Si el host desea iniciar el proceso para cambiar el modo de velocidad, debe ejecutar el comando CMD11, seguido del comando CMD42 durante el estado de transferencia. No obstante, este procedimiento excede el alcance de este documento, que trata únicamente el caso de un host utilizando el modo de velocidad predeterminada.

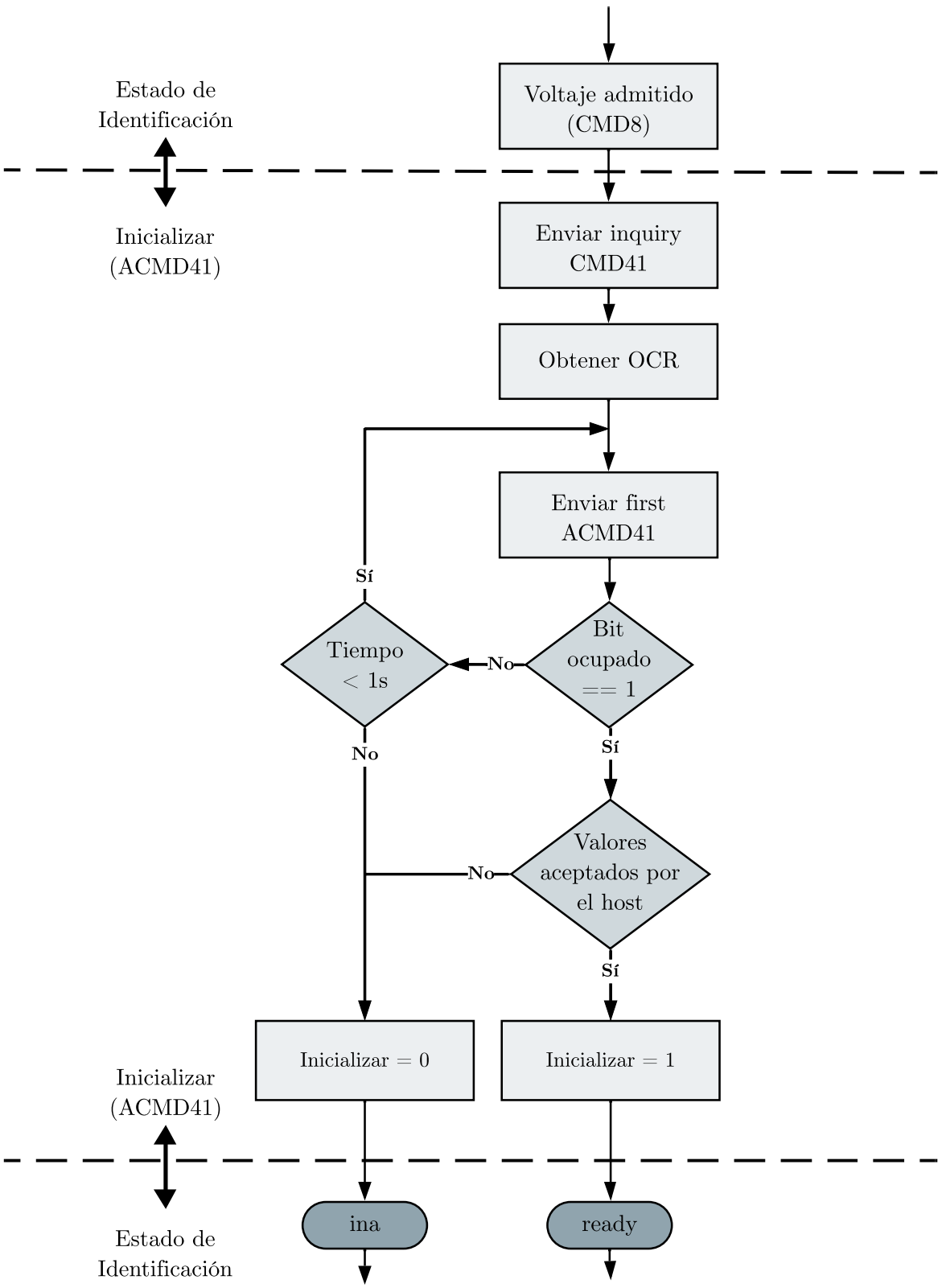


Figura 3.20: Diagrama de Flujo de `ACMD41`

Solicitar CID (CMD2)

En estado `ready` es necesario identificar la tarjeta para acceder a ella. El comando `ALL_SEND_CID` (`CMD2`) se utiliza para solicitar a la tarjeta el envío de su número de identificación de tarjeta (`CID`). Una vez enviado el `CID`, la tarjeta pasa al estado de `ident`.

Solicitar RCA (CMD3)

Un número de identificación consta de 128 bits, lo cual lo hace impráctico para acceder constantemente a una tarjeta microSD. El comando `SEND_RELATIVE_ADDR` (`CMD3`) es un mecanismo que le brinda al host la capacidad de solicitar a la tarjeta la generación y envío de una nueva dirección relativa (`RCA`). La `RCA` (16 bits) es más corta que un `CID` y es la utilizada para direccionar la tarjeta en modo de transferencia de datos futuros. Una vez la tarjeta envíe el `RCA`, pasa al estado de `stby`. El host puede solicitar una nueva `RCA` cuantas veces considere, una vez que la tarjeta se encuentre en estado `stby`.

3.4.15. Estado de transferencia de datos

El estado de transferencia de datos es el estado principal de la tarjeta microSD. En este estado, la tarjeta puede realizar todas las operaciones que el host requiera. La figura 3.21 muestra el diagrama de flujo que el host debe realizar para acceder a los diferentes estados de la tarjeta y completar una operación. Se hace la notación que el diagrama es una simplificación del estado de transferencia de datos para ilustrar las secciones de interés en este documento.

Seleccionar/Deseleccionar tarjeta (CMD7)

Todas las tarjetas a la espera de ser seleccionadas por el host o finalizar el estado de identificación se deben encontrar en estado `stby` a la espera del envío de alguna operación por parte del host. Si el host va a iniciar una operación en una determinada tarjeta, debe utilizar el comando `SELECT_DESELECT_CARD` (`CMD7`) para direccionar del estado `stby` al estado `tran`, adjuntando el `RCA` en el argumento del comando.

Siempre es importante establecer la tarjeta nuevamente al estado `stby`, ya sea porque el host no va a realizar más operaciones en el estado `tran` o porque se va a hacer un cambio de tarjeta. Para ello, el host debe volver a enviar `CMD7` en el estado `tran` con cualquier otro valor en `RCA` para deseleccionar la tarjeta y direccionarla al estado `stby`. Cuando se envía `CMD7` para deseleccionar, la tarjeta no envía una respuesta al host. Se recomienda que el host reserve el valor predeterminado de `RCA` (16'h0000) para deseleccionar las tarjetas.

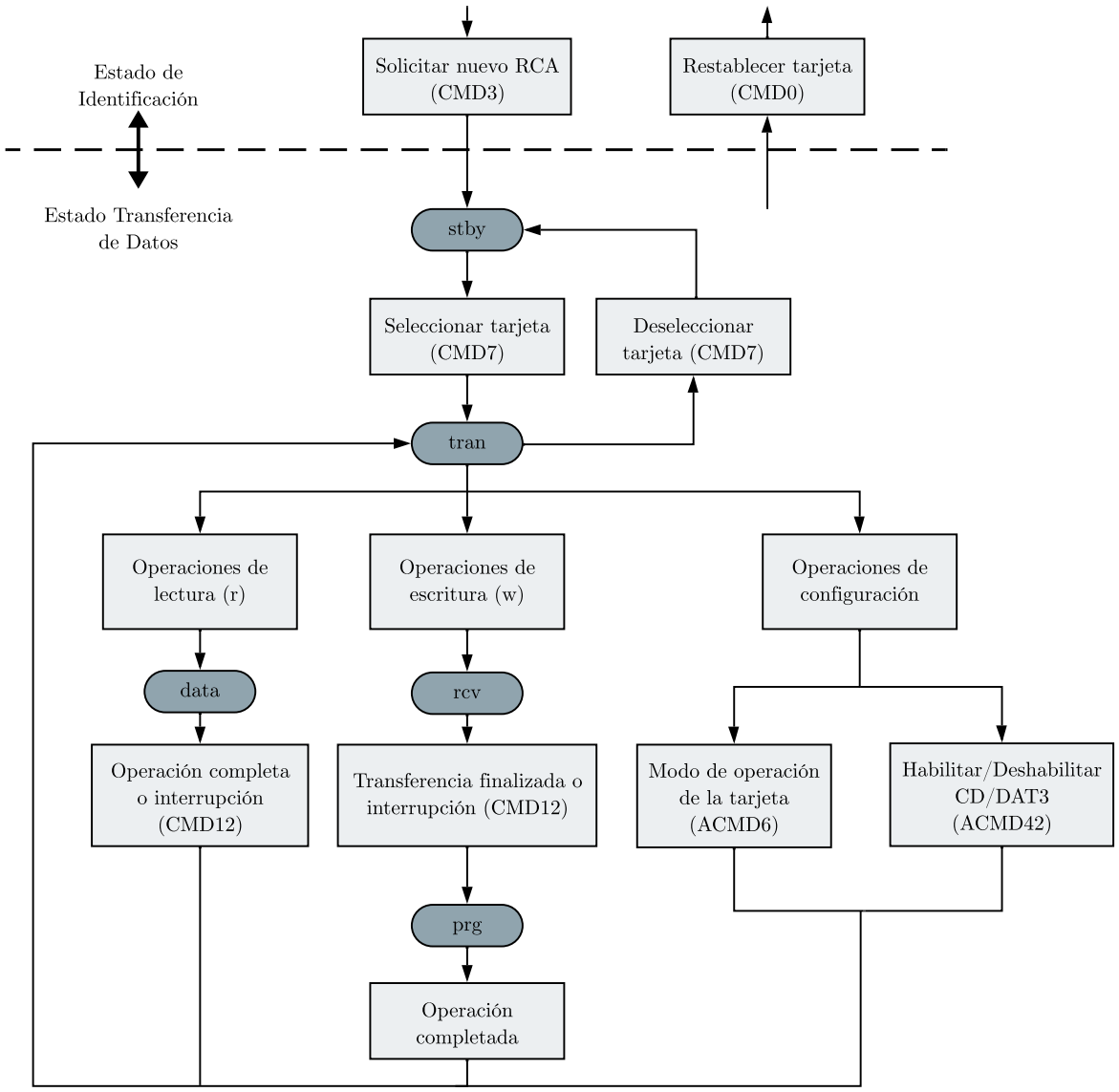


Figura 3.21: Diagrama de Flujo del Estado de Transferencia de Datos

Modo de operación SD de la tarjeta (ACMD6)

El estándar UHS-I solo admite el modo de operación SD de 4-bits, por defecto las tarjetas microSD inician en el modo 1-bit. El comando `SET_BUS_WIDHT` (`ACMD6`) esta diseñado para que el host pueda cambiar el modo de operación SD de la tarjeta seleccionada. Al ser un comando de aplicación específica se debe enviar posterior a un `CMD55`. Sin embargo, es importante resaltar que el comando `CMD55` contenga el `RCA` de la tarjeta seleccionada en el argumento, aunque ya se haya seleccionado previamente mediante el comando `CMD7`.

Habilitar/Deshabilitar resistencia en CD/DAT3 (ACMD42)

Al encenderse las tarjetas microSD, inician con un '1' lógico en el pin `CD/DAT3` mediante una resistencia pull-up. Para que el host pueda iniciar una transferencia de datos en modo 4-bits, debe previamente deshabilitar la resistencia pull-up de este pin mediante el comando `SET_CLR_CARD_DETECT` (`ACMD42`).

Operación de escritura (wr)

Existen dos comandos generales para iniciar una operación de escritura en una tarjeta microSD:

- **WRITE_BLOCK (CMD24)**: escribe un único bloque de datos en la dirección adjuntada en el argumento.
- **WRITE_MULTIPLE_BLOCK (CMD25)**: escribe continuamente bloques de datos iniciando en la dirección adjuntada en el argumento hasta que el host envíe el comando `STOP_TRANSMISSION` (`CMD12`).

En las tarjetas SDSC la longitud de bloque de datos se debe establecer mediante el comando `CMD16` (externo a este documento). En las tarjetas SDHC y SDXC la longitud del bloque tiene un valor fijo de 512 bytes. El host debe asegurarse de transmitir 128 bytes por cada línea `DAT`, con sus respectivos bits de inicio, parada y CRC16. La ubicación donde las tarjetas microSD almacenan un bloque de datos se denomina sector. Por ende, un sector tiene un tamaño de 512 bytes y la cantidad de sectores define el tamaño de almacenamiento de la tarjeta. La dirección en el argumento corresponde al número hexadecimal del sector a escribir.

El host envía el comando de escritura y, posterior a la respuesta, la tarjeta entra en el estado `rcv` en donde espera a recibir el bloque de datos por parte del host. Una vez finalizada una transacción, la tarjeta envía el respectivo `CRC status` y cambia al estado `prg` donde envía el bit de ocupado. Cuando la tarjeta esté lista, pasa al estado `tran`. En este punto, el host puede realizar otra operación. En el caso de estar realizando escritura multibloques, una vez en el estado `tran`, la tarjeta se traslada automáticamente al estado `rcv` a la espera del siguiente bloque de datos.

El host puede interrumpir el proceso de escritura mediante el comando `CMD12`. Es posible interrumpir el proceso en las diversas etapas de operación de escritura. Sin embargo, el documento se centra en la interrupción del proceso cuando la tarjeta está a la espera del siguiente bloque de datos. De esta manera, el host envía `CMD12` en lugar de enviar un bloque de datos. La tarjeta recibe y acepta el comando y cambia del estado `rcv` al estado `prg` sin haber recibido datos. En el estado `prg` la tarjeta concluye la operación respectiva y, una vez lista, se traslada al estado `tran` a la espera de una nueva operación.

Capítulo 4

Sistema de transmisión de datos en tiempo real a computadora

4.1. General

Este capítulo se enfoca en el desarrollo de un "Sistema de transmisión de datos en tiempo real a computador". El sistema que aquí se detalla tiene como propósito transmitir a una computadora, por un período de 10 minutos, los datos provenientes del módulo DSP de la unidad de adquisición y procesamiento de señales.

Los datos son transferidos mediante lotes por medio del protocolo SPI a un IC FT232H con una tasa de transferencia de 10 Mbps, de tal forma que se brinda un margen amplio entre la tasa de transmisión y la tasa de transferencia de datos del DSP. El IC se encarga de convertir los datos para su posterior transmisión a una computadora a través del protocolo USB 2.0.

El sistema está compuesto por dos unidades, la primera implementada en hardware en la FPGA y la segunda implementada en software en una computadora.

4.2. Diseño de la unidad de hardware

El diseño RTL del sistema consta de tres secciones como se muestra en la figura 4.1:

- **SPI FIFO**: almacena los datos provenientes del módulo DSP para su transmisión.
- **Detector de Flancos**: sincroniza el módulo SPI Periférico con el SPI Controlador detectando los flancos ascendentes del reloj `sck_i`.
- **SPI Periférico**: realiza las comunicaciones con el SPI Controlador y gestiona la transmisión de los datos.

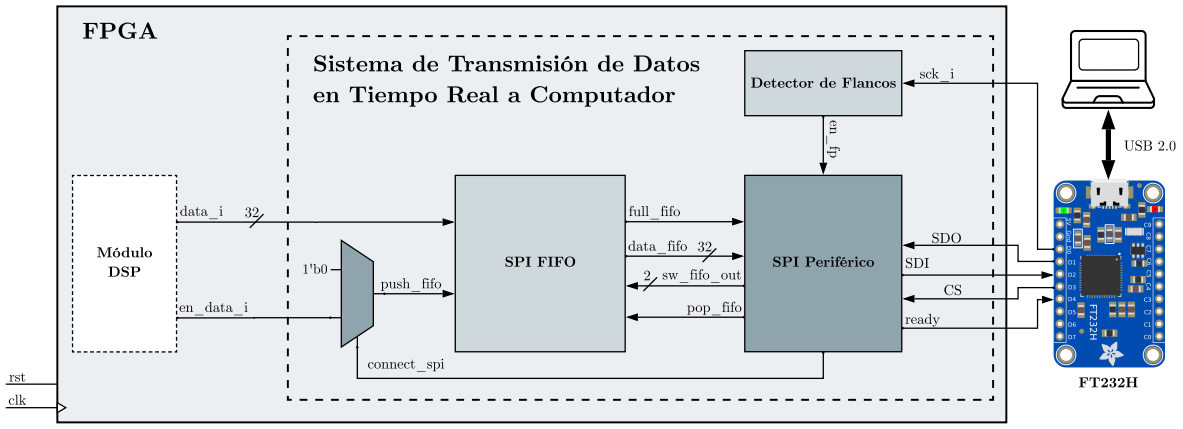


Figura 4.1: Diagrama de Bloques del Sistema de Trasmisión de Datos en Tiempo Real a Computador

4.2.1. SPI FIFO

El SPI FIFO almacena los datos de entrada del sistema provenientes de la unidad DSP. El módulo funciona internamente mediante 3 módulos FIFOs en paralelo, diseñados por medio del IP core "FIFO Generator" de Vivado, con un ancho de escritura de 32 bits y una profundidad de 2048. Por tanto, el SPI FIFO cuenta con un tamaño total de 32.768 kB. La elección de este tamaño radica principalmente en la cantidad de lotes y lo que este documento va a denominar "tiempo de ocupado", el cual es un tiempo en donde el IC FT232H no permite recibir datos por parte del periférico. Este tiempo es variable y no se encuentra documentado por la hoja de datos del FT232H. Por tanto, el tamaño del FIFO se determinó inicialmente mediante resultados experimentales para evitar la pérdida de datos durante este tiempo de inactividad. Estos resultados proporcionaron un punto de partida. A partir de ahí, el tamaño final del FIFO se definió en función de la cantidad de lotes a enviar. En la figura 4.3 se muestra el módulo SPI FIFO con sus respectivos componentes internos.

El módulo utiliza un registro para asignar un FIFO específico para el ingreso de los datos cuyo funcionamiento está basado en el diagrama de flujo de la figura 4.2, así mismo cuenta con la señal `sw_fifo_out`, controlada por el SPI Periférico, que asigna un FIFO específico para el egreso de los datos. El FIFO SPI opera mediante un sistema rotativo de FIFOs, en el que cada uno desempeña uno de los siguientes cuatro roles:

- **Captura**: el FIFO está en el proceso de capturar los datos entrantes. El Gestor de Entrada redirecciona las señales de entrada del módulo al FIFO de este estado. Una vez que se ha llenado, pasa al estado "Lleno".
- **Lleno**: el FIFO ha completado su proceso de llenado y está lista para transferir los datos. Una vez que el FIFO en el estado de "Transferencia" ha completado su

proceso de transferencia de datos, el FIFO de este estado transiciona al estado de "Transferencia".

- **Transferencia**: el FIFO está actualmente en el proceso de transferir los datos. El SPI Periférico redirecciona las señales de salida del módulo al FIFO de este estado. Una vez que se ha vaciado, pasa al estado "Espera".
- **Espera**: es una transición entre el estado de "Transferencia" y "Captura". El FIFO está a la espera de comenzar a capturar los datos entrantes. Una vez que el FIFO en el estado de "Captura" ha completado su proceso de captura de datos, este FIFO transiciona al estado de "Captura"

Al iniciar el sistema, todos los FIFOs están en el estado de "Espera", excepto un, el cual comienza en el estado de 'Captura'. A medida que el sistema avanza, cada FIFO transiciona secuencialmente a través de los cuatro roles: "Captura", "Lleno", "Transferencia" y "Espera".

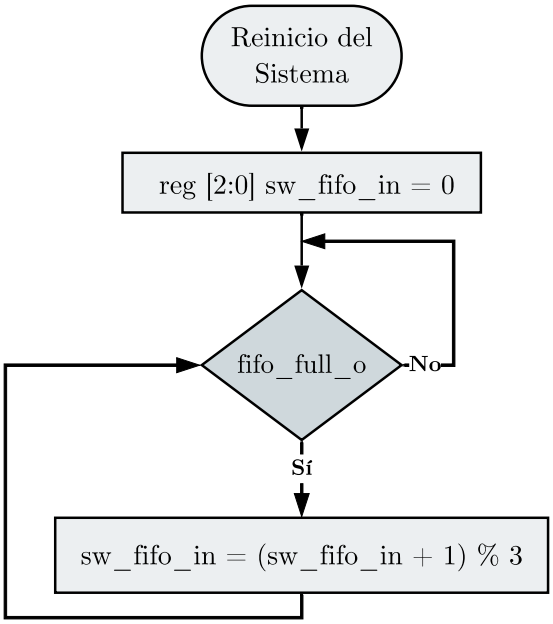


Figura 4.2: Gestor de Entrada

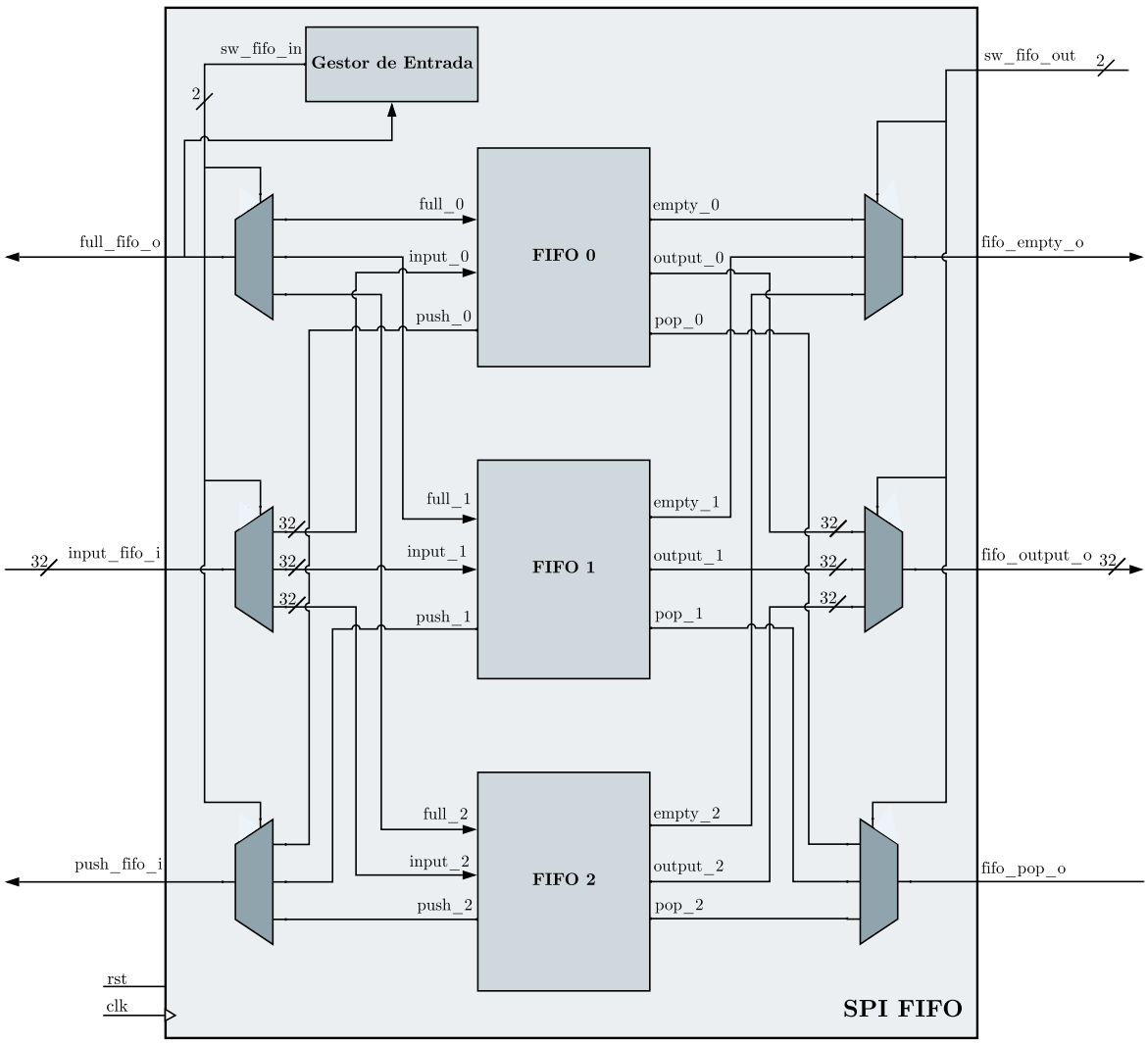


Figura 4.3: Diagrama de Bloques de SPI FIFO

4.2.2. Detector de Flancos

Dado que el FT232H funciona como un Controlador SPI y el sistema se basa en un Periférico SPI, es necesario poder sincronizar la señal de reloj proveniente del Controlador. El detector de flanco utiliza un registro para retener el valor anterior de `sck_i`. Mediante la operación `AND` del valor actual de `sck_i` y el valor del registro, se genera un pulso en la señal `en_fp` en el flanco positivo de `sck_i`. El SPI Periférico utiliza esa señal para preparar la información a transmitir e identificar el momento adecuado para capturar la información enviada por el FT232H.

4.2.3. SPI Periférico

Es el módulo principal del sistema que se comunica con el Controlador SPI (FT232H). Su funcionamiento está basado en el sistema de comando - respuesta. Por lo que lee, captura e interpreta la información proveniente de la señal `SDO` (comando) y transmite mediante un mux la respuesta ante el comando, los datos almacenados en el FIFO SPI o un valor en alto o bajo según se requiera. Para los casos de envío de comando o datos, utiliza la lógica Selección de Bit, la cual selecciona el bit correspondiente mediante la señal `index` para transmitir el comando o el dato de forma serial. En la figura 4.4 se muestra el diagrama de bloques asociado a este módulo.

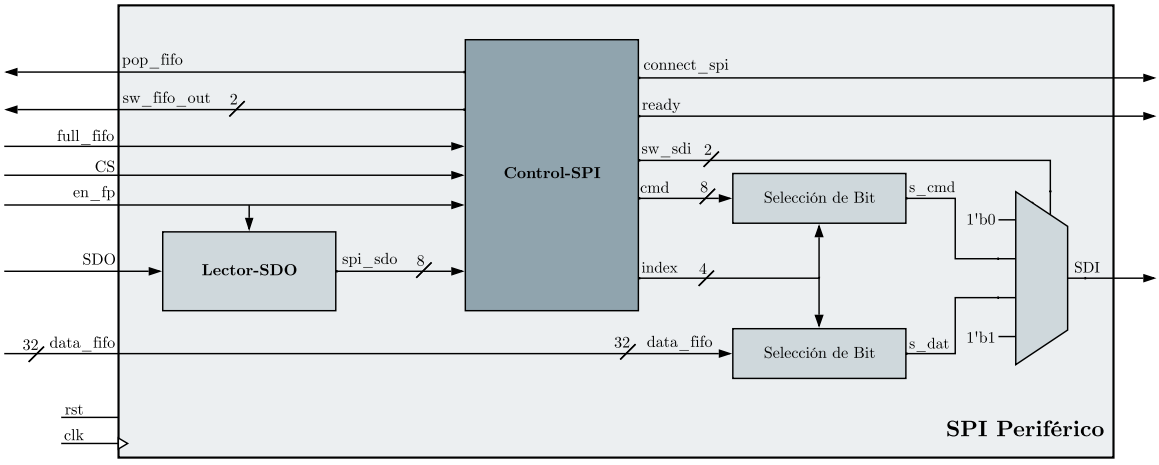


Figura 4.4: Diagrama de Bloques del SPI Periférico

Lector-SDO

El SPI Periférico utiliza este módulo para leer y capturar los datos provenientes de la línea `SDO`. El módulo mediante un registro de 1 byte realiza desplazamiento de 1 bit a la izquierda para recopilar serialmente el dato de 1 byte transmitido por el Controlador SPI. El proceso se activa cada vez que detecta un pulso en `en_fp`.

Control-SPI

Módulo encargado de interpretar los comandos provenientes del Controlador SPI, además controla qué información se va a transmitir de manera serial mediante `sw_sdi` e `index`. Crea la respuesta a transmitir y administra la forma en cómo se transmiten los datos. El Control-SPI fue diseñado mediante un FSM de Mealy. En la figura 4.5 se muestra el diagrama de estado que describe el funcionamiento del FSM del Control-SPI y en la tabla 4.1 se describen los estados que lo componen.

Tabla 4.1: Descripción de Estados del FSM de SPI Periférico

Estado	Descripción
ON_SPI	▪ Espera el inicio de transferencia de datos por parte del Controlador.
READ_SDO	▪ Lee y verifica que el dato transferido por el Controlador corresponde al comando de inicio (8'hDA). • Si no corresponde se traslada nuevamente a ON_SPI.
SPI_RESP	▪ Envía una respuesta con el valor de 8'h1E como confirmación de recibido del comando al Control SPI.
WAIT_FIFO	▪ Espera que el FIFO SPI se encuentre en su máxima capacidad.
SPI_SEND	▪ Envía serialmente el dato a transmitir. ▪ Completado el envío de un dato: • Terminó el lote, se traslada a COUNT_BATCH. • El FIFO actual está vacío, redirecciona al siguiente FIFO.
COUNT_ BATCH	▪ Si la línea SDO se encuentra en alto: • Se traslada a SPI_SEND si aún no ha transferido el total de paquetes. • Se traslada a WAIT_FIFO si ha transferido todos los paquetes. ▪ Si la línea SDO se encuentra en bajo, finaliza la operación y se traslada al estado inicial.

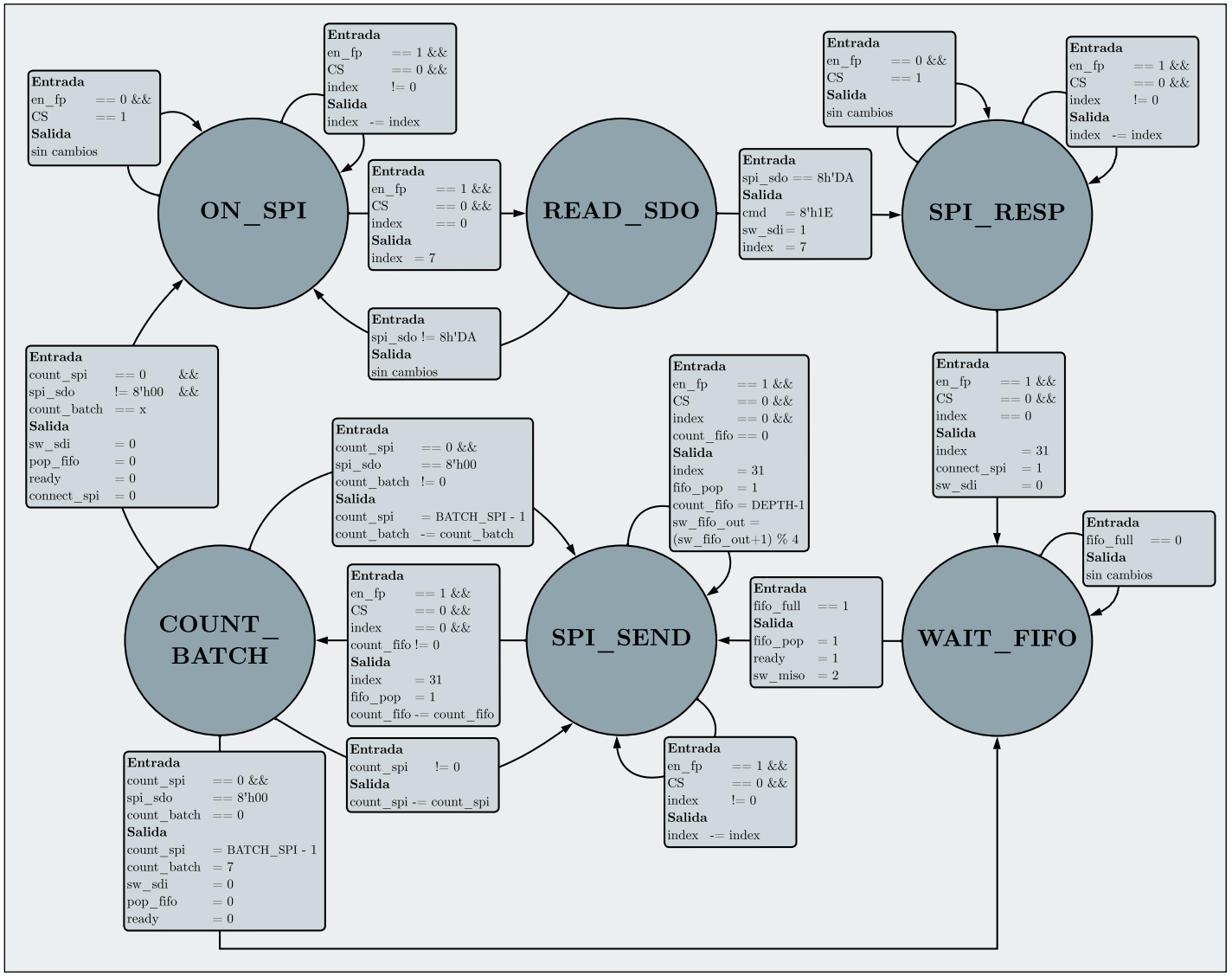


Figura 4.5: Diagrama de Estados del SPI Periférico

4.2.4. Funcionamiento general del sistema

El sistema inicia al recibir el comando de inicio específico por parte del Controlador SPI (FT232H). Este comando corresponde al valor de 8'hDA. Una vez recibido e identificado el comando, envía una respuesta con el valor de 8'h1E como confirmación de recibido al Controlador SPI.

El Controlador SPI está configurado para recibir 1 kB de datos (ver sección 3.3.3) por parte del Periférico. Por este motivo, el Control-SPI gestiona la transmisión de datos mediante lotes. Un lote equivale a 250 datos de 32 bits. Antes de transmitir los datos, el Control-SPI espera que el FIFO SPI cambie la señal de `full_fifo` a alto, lo que indica que un FIFO interno ha llegado a su capacidad máxima. Por tanto, el sistema puede iniciar a transmitir un total de 8 lotes con un sobrante de 192 B, es decir, 48 datos que no se transmitirán.

Mediante la señal `ready` conectada a un GPIO del FT232H, el sistema le indica al IC que puede dar inicio a la recepción de datos mediante el protocolo SPI. El motivo de enviar una señal a un GPIO y no aprovechar el FDX del protocolo SPI, enviando una respuesta de inicio de transferencia. Se debe principalmente a que el FT232H recibiría en su gran mayoría datos "basura", ya que cada dato recibido debe almacenarlo en el búfer interno RX. del IC. El vaciado de este búfer genera atrasos no documentados en su hoja de datos. Por esta razón, el inicio de transferencia de datos se transmite de forma externa al protocolo y se utiliza el protocolo de comunicación SPI para uso exclusivo de información relevante. El SPI Periférico envía constantemente los datos de forma serial y el Control-SPI supervisa la cantidad de lotes enviados. Completada la transmisión de un lote de datos, el FT232H deshabilita las señales `sck_i` y `CC` e inicia el "Estado de Ocupado". Cuando finaliza, lee nuevamente la señal `ready` y continúa con la transferencia de un nuevo lote de datos.

En el instante en que el Control-SPI detecta que se han trasferido los 8 lotes, espera nuevamente a que el FIFO SPI cambie la señal de `full_fifo` a alto, lo que indica al sistema el inicio de la trasferencia de los nuevos 8 lotes, con la salvedad de que los primeros datos a transferir corresponden a los 6 datos sobrantes que no se lograron transmitir en la primera iteración. Una vez enviados estos datos, el Control-SPI redirecciona la salida del FIFO SPI al FIFO interno siguiente y retoma el proceso normal de transferencia.

El proceso continúa con el flujo anteriormente descrito, hasta que durante la transferencia la línea proveniente del Controlador SPI cambia su estado a alto. El Lector-SDO detecta la línea en alto y el Control-SPI concluye el proceso de transferencia una vez finalizada la transacción del lote en curso.

4.3. Diseño de la unidad de software

El chip FT232H debe ser configurado y programado mediante el lenguaje de programación Python como se especifica en el Capítulo 3.3.4. El FT232H se configura como un controlador SPI con un reloj de 10 MHz y un único periférico, adicionalmente se asignó `D4` como entrada (GPI). En el Listado 4.1 se muestra la inicialización y configuración utilizada en este documento para el FT232H.

Listing 4.1: Código Python para la configuración del FT232H como Controlador SPI

```
# Crear una instancia de SpiController con un
# solo Chip Select (CS)
spi = SpiController(cs_count=1)

# Configurar el IC FTDI para SPI
spi.configure('ftdi://ftdi:232h:1/1')

# Obtener una interfaz SPI en el primer puerto
# con un divisor de reloj de 10 MHz
slave = spi.get_port(cs=0, freq=10E6, mode=0)

# Obtener un puerto GPIO para manejar un pin extra
gpio = spi.get_gpio()

# Establecer A*BUS4 (D4) como entrada (GPI)
gpio.set_direction(0x10, 0x00)
```

En la figura 4.6 se presenta el diagrama de flujo del software para la solicitud y captura de datos provenientes del hardware.

Una vez inicializado el driver y los punteros archivos, se procede a enviar un comando de inicio de datos hacia la FPGA, el software espera una respuesta desde el chip. Posteriormente, abre el archivo binario y realiza una transacción de un lote de 250 datos SPI (1 B) multiplicado por 4, ya que un dato del DSP (32 bits) equivale a 4 datos SPI. Por tanto, solicita 250 (lote) datos DSP. Finalizada la transferencia, escribe el lote en el archivo binario y verifica si el tiempo transcurrido ha excedido el tiempo establecido por el usuario. De no serlo, vuelve a solicitar un nuevo lote de datos y repite el proceso. Al haber superado o igualado el tiempo, solicita un nuevo lote de datos, pero envía al periférico una señal en alto. Terminada la transferencia, escribe el último lote en el archivo binario y cierra el archivo.

Posteriormente, se ejecutan dos funciones opcionales para la verificación de las pruebas. La primera función toma el archivo binario y lo convierte en texto para poder manipular los datos obtenidos. El segundo accede al archivo de texto y realiza una comprobación comparativa entre el valor previo y el valor actual, validando si el valor actual es consecutivo al valor previo, es decir, si es igual al valor previo incrementado en uno.

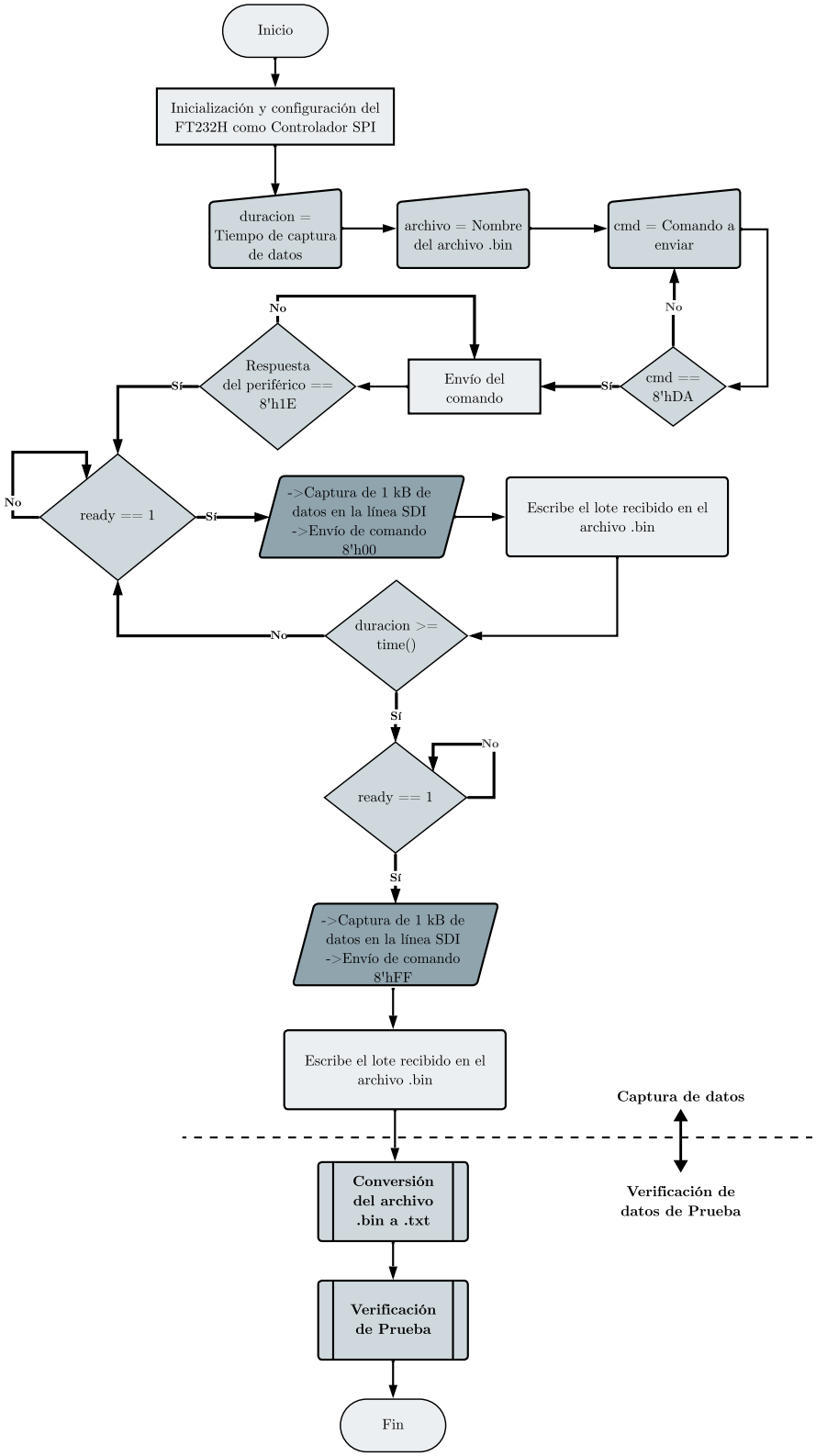


Figura 4.6: Diagrama de flujo del Proceso de Captura de Datos

4.4. Análisis de resultados

4.4.1. General

En esta sección se muestran los resultados obtenidos de la implementación del "Sistema de transmisión de datos en tiempo real a computadora". El sistema se implementó en una FPGA Artix-7 en la plataforma de desarrollo Nexys 4 DDR, cargando los bitstreams del sistema a partir del diseño RTL realizado en el lenguaje de descripción de hardware SytemVerilog en el software Vivado de Xilinx®.

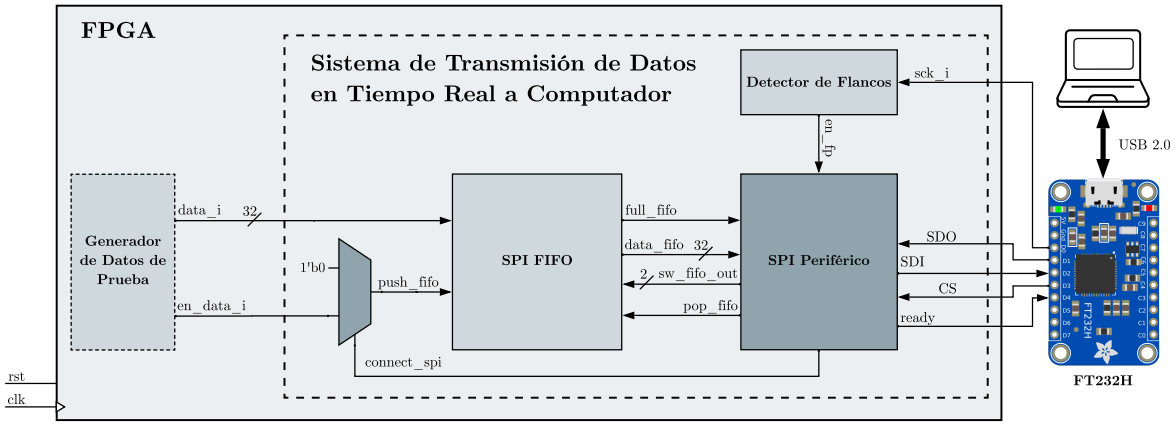


Figura 4.7: Diagrama de Bloques del Sistema de Transmisión de Datos en Tiempo Real a Computadora con Módulo de Pruebas

Las pruebas se realizaron mediante el uso de un módulo de prueba, el cual simula una unidad DSP con una salida de datos de 32 bits a una frecuencia de 50 kHz. El módulo de prueba se observa en la figura 4.7 y consiste en un contador de datos a una frecuencia de 50 kHz (el mismo módulo envía un pulso a la entrada `en_data_i` para indicar que hay un dato válido nuevo). Los datos son generados de manera secuencial descrita mediante la función 4.1, donde 'n' es un número entero que incrementa una unidad por cada pulso. Lo que permite verificar la integridad de los datos a lo largo del sistema.

$$f(n) = 10000000_{16} + n \tag{4.1}$$

Los resultados presentados se realizaron en la etapa de implementación mediante el uso del IP core "ILA" de Vivado. Por consiguiente, los resultados corresponden a una prueba física en la FPGA. Se debe mencionar que cada unidad de tiempo en los diagramas de tiempo obtenidos por el ILA representan un valor de 10 ns. El reloj del sistema se estableció en 100 MHz en el Archivo de Restricciones, como se muestra a continuación:

```
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0
    5} [get_ports clk_pi]
```

Se empleó un puerto PMOD, asignando 5 pines a las señales: `sck_i`, `SDO`, `SDI`, `CS` y `ready` para la conexión del FT232H. Adicionalmente, la tabla 4.2 detalla los recursos de la FPGA empleados, como se puede observar el diseño planteado no llega a ocupar una gran cantidad de área. En la tabla 4.3 se presentan los tiempos de mayor interés durante la implementación, como se puede observar al ser valores positivos es una indicación que a nivel de implementación se ejecuta de forma correcta. Por lo cual, no hay problemas de temporización. Asimismo, al contar con un 'Wort Negative Slack (WNS)', evidencia que se tiene un margen muy amplio de temporización con lo cual se muestra que es posible aumentar la velocidad máxima del reloj hasta máximo de 193.42 MHz. Adicionalmente, la figura 4.8 muestra la potencia consumida por el sistema, lo cual como se puede observar presenta un consumo de 0.101 W.

Tabla 4.2: Recursos de la FPGA utilizados

Recurso	Utilizado	Disponible	Utilizado (%)
LUT	339	63400	0.53
FF	273	126800	0.22
BRAM	6	135	4.4
IO	10	210	4.76
BUFG	1	32	3.13

Tabla 4.3: Tiempos de la Implementación del Sistema

Setup	Hold	Pulse Width
Wort Negative Slack (WNS): 4.830 ns	Worst Hold Slack (WHS): 0.119 ns	Worts Pulse Width Slack (WPWS): 4.500 ns

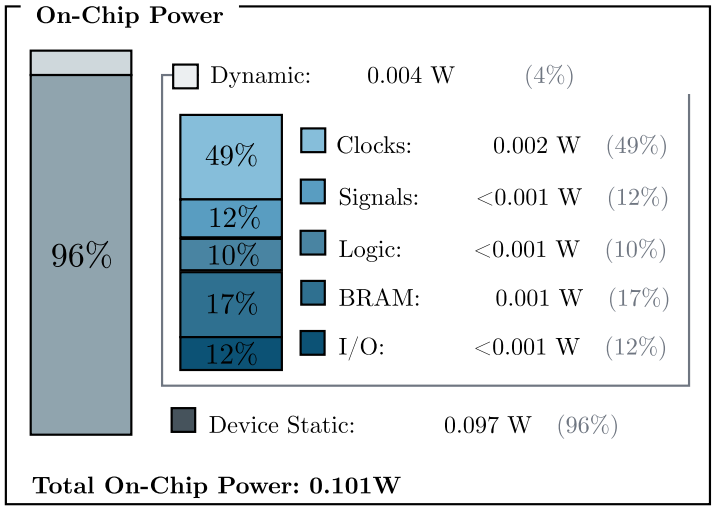


Figura 4.8: Total de Potencia Utilizada

4.4.2. Comando y Respuesta de Inicio del Sistema

La figura 4.9 muestra el diagrama de tiempo de una transmisión de envío del comando por parte del Controlador SPI y la respuesta ante el comando por parte del Periférico. El Controlador SPI habilitó una transferencia de 2 bytes, bajando la señal `CC` y habilitando el reloj `sck` a 10 MHz. El primer byte correspondió al envío del comando por parte del FT232H en donde el IC desplazó el bit del comando en el flanco descendente del reloj y el Periférico logró obtener el bit en cada flanco ascendente. El valor resultante del comando correspondió a 8'hDA, por lo que el Periférico reconoció correctamente el comando y envió el valor de 8'h1E como respuesta. Finalmente, al completarse la transacción de 2 bytes, el controlador deshabilitó en un período posterior las señales `CC` y `sck`. De esta forma, es posible establecer una conexión entre el Controlador y el Periférico realizando una acción específica tomando en consideración que el sistema no es el controlador, por tanto debe ajustarse al FT232H.

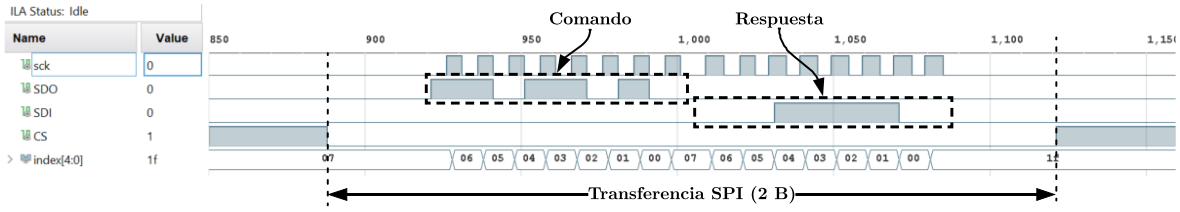


Figura 4.9: Diagrama de Tiempo Comando y Respuesta de Inicio del Sistema

4.4.3. Captura de Datos

En la figura 4.10 se observa cómo una vez que finalizó el proceso de inicio mediante el comando y respuesta, se dio inicio al proceso de captura de datos. El sistema habilitó la señal `connect_spi`, lo que indicó que se ha establecido una conexión con el Controlador. A la vez, habilitó la entrada del módulo DSP (o módulo de prueba), lo que permitió la entrada de datos. Se puede visualizar cómo el sistema capturó el dato y almacenó en el primer FIFO cada pulso de la señal `en_data`, que corresponde a la señal de listo proveniente del módulo DSP. Adicionalmente, se puede ver que los FIFOs restantes se encontraron en estado de "Espera".

En la figura 4.11 se muestra cuando el primer FIFO llegó a su capacidad máxima, en ese momento, el gestor de entrada redireccionó las señales de entrada del módulo FIFO SPI al segundo FIFO y continuó almacenando en dicho reservorio. Por otra parte, se observa cómo el primer FIFO generó únicamente un pulso en la señal de `full`, esto se dio porque inmediatamente el Control-SPI detectó el FIFO en estado de "Lleno", habilitó un pulso en la señal `pop` dejando el primer dato que se obtuvo a la salida del FIFO 0 o lo que es lo mismo, en la salida del FIFO SPI, listo para ser transmitido. A su vez, el Control-SPI habilitó la señal de `ready`, lo que le indicó al FT232H dar inicio a la transferencia de

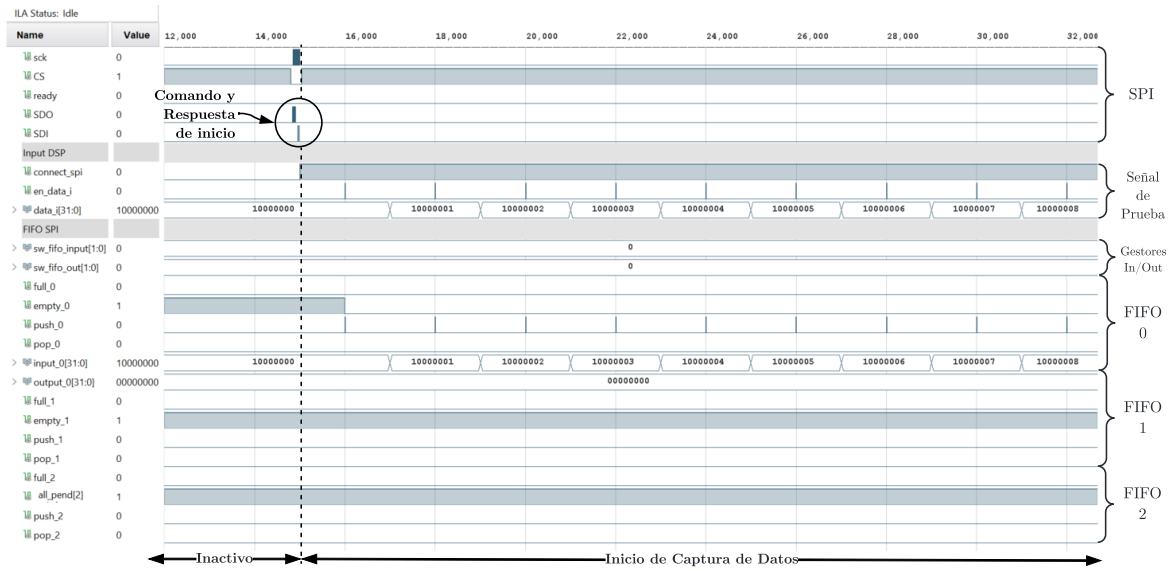


Figura 4.10: Diagrama de Tiempo de Inicio de Captura de Datos

datos. Sin embargo, y como se puede llegar a notar, el FT232H no inició inmediatamente la transferencia de datos. Eso se debió principalmente a que el FT232H se ejecuta con código de alto nivel, el cual presenta atrasos dado a la computadora; a la vez, se deben sumar atrasos debido al cableado del GPI.

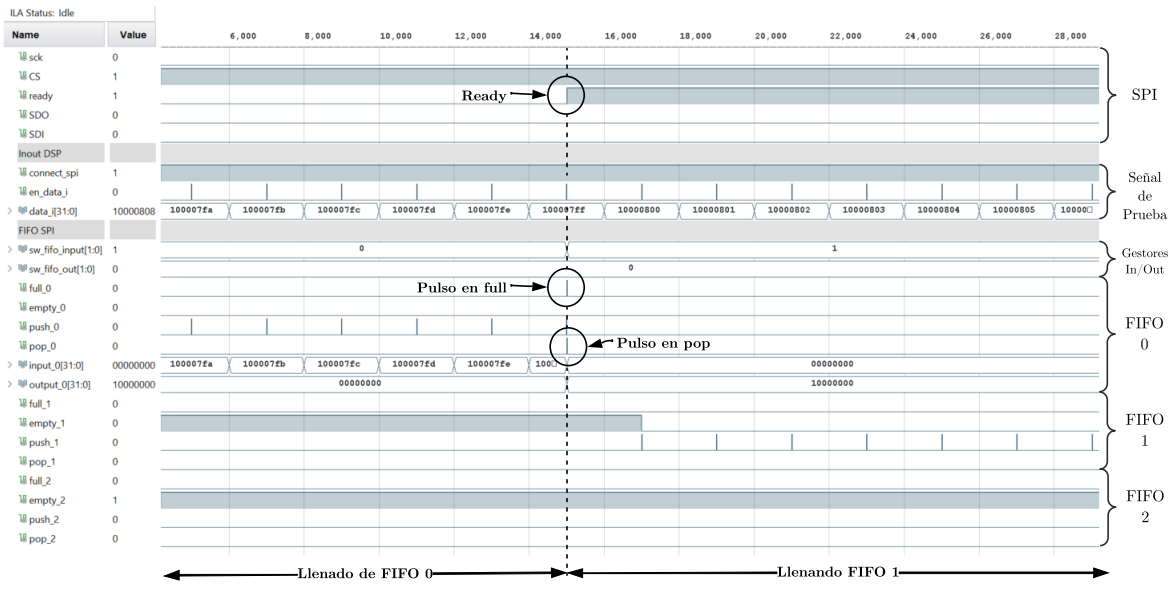


Figura 4.11: Diagrama de Tiempo de los Datos Listo para Transferir

4.4.4. Transferencia de Datos

Una vez que el IC reconoció la señal en alto de `ready`, el FT232H inició la transferencia de los 250 lotes contenidos en el primer FIFO mediante la habilitación de la señal `CC` y el reloj `sck` a 10 MHz, como se muestra en la figura 4.12. Además, se puede ver cómo FIFO 0 recibió constantemente la señal `pop`, enviando nuevos datos una vez que la transmisión del anterior finalizó.

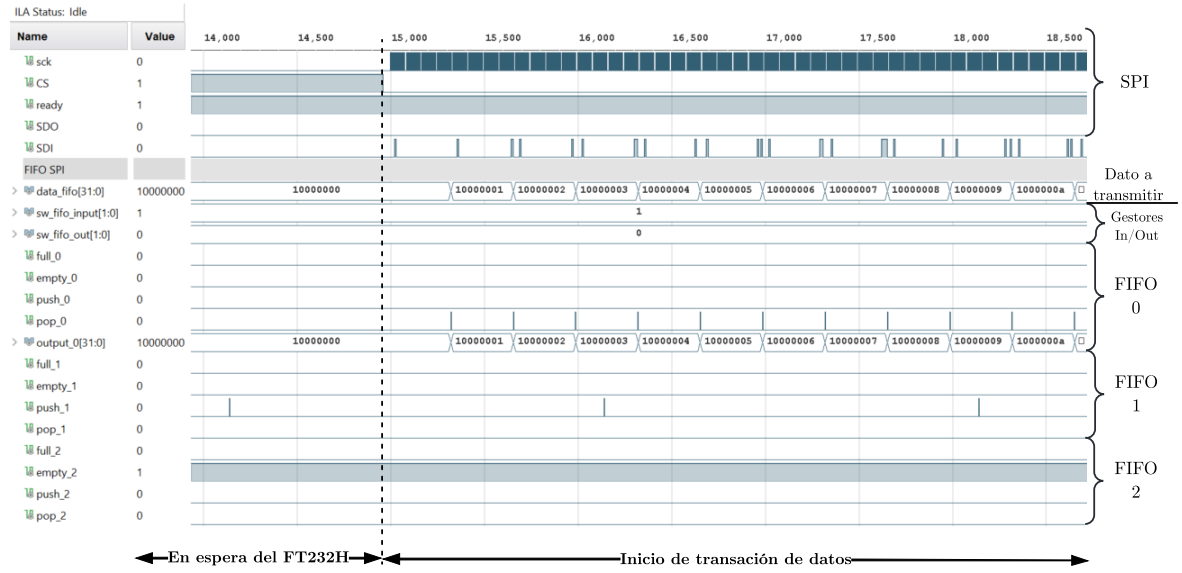


Figura 4.12: Diagrama de Tiempo de Inicio de Transferencia

En la figura 4.13 se muestra la finalización de la transmisión de un lote de datos por parte del sistema y el FT232H. Se puede ver cómo el IC deshabilitó las señales `CC` y `sck` y posteriormente entró al estado de ocupado. Además, se puede observar cómo el sistema se mantuvo a la espera de que el IC volviera a iniciar la transferencia para continuar con la transmisión del siguiente lote. Por otro lado, los datos del módulo DSP siguieron ingresando al sistema. El FIFO SPI se encargó de "absorber" los datos entrantes para no perder ningún dato durante el proceso de ocupado del IC.

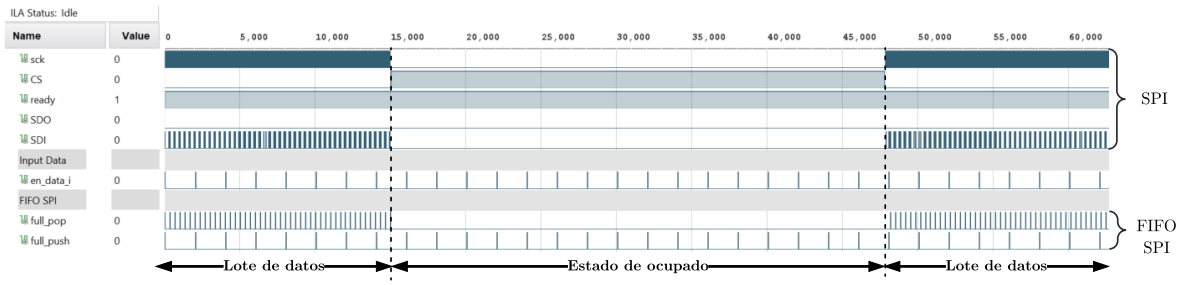


Figura 4.13: Diagrama de Tiempo de Fin de Envío de Lote

Sin embargo, durante las pruebas, se observó que el tiempo de ocupado variaba cada vez que el IC entraba al estado ocupado. En general, se mantuvo en rangos como el mostrado en la figura anterior. No obstante, presentó tiempos de ocupado, en donde alcanzó lapsos de hasta un máximo de 18 ms (ver apéndice B). Dado que los FIFOs tienen una capacidad limitada de almacenamiento, se establece una relación lineal entre la profundidad del FIFO y el tiempo máximo de ocupada, ya que la profundidad del FIFO aumenta proporcionalmente con el tiempo máximo para evitar la pérdida de datos debido a un 'overflow'. Esta relación se observa en la recta de la figura 4.14.

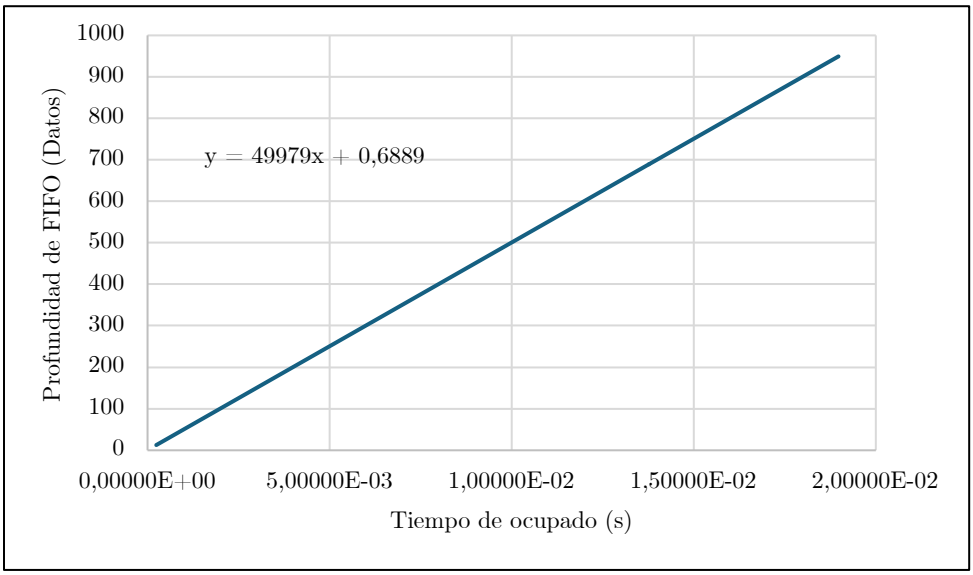


Figura 4.14: Profundidad del FIFO SPI vs Tiempo Ocupado

Adicionalmente, se deben considerar los datos que entran posterior al finalizar el estado de ocupado, ya que el FIFO SPI puede sufrir 'overflow' a causa de ese período. Esto se debe a que si la transferencia de datos no es lo suficientemente rápida comparada con el ingreso de datos, es posible que el FIFO SPI no logre vaciarse lo suficiente para cuando el IC entre nuevamente en estado de ocupado. Lo que causaría que el FIFO no tenga la capacidad necesaria de almacenamiento como para resguardar todos los datos durante ese tiempo inactivo. Cada dato del módulo DSP se transfiere al FT232H a una frecuencia de 312.5 kHz. Por lo cual se logran transferir 6.25 datos al FT232H por cada nuevo ingreso de datos del módulo DSP al sistema. Y dado que el estado de ocupado se da cada 250 datos, únicamente ingresan 40 nuevos datos durante una trasferencia de un lote.

Por tanto, tomando en cuenta la gráfica de la función utilizando el tiempo máximo obtenido en las pruebas (18 ms), más los 40 datos que ingresan durante la transferencia. Se determinó que un FIFO interno debía tener una profundidad de al menos 1024 datos. Considerando el hecho de que cada FIFO está en un estado diferente (Lleno, Transferencia y Captura). Partiendo de ese valor, se establece cada FIFO con una profundidad de 2048 datos (el doble del mínimo). Con el fin de transferir 8 lotes de datos por cada FIFO y, además, proporcionar un mayor margen para proteger los datos sobre posibles tiempos

de ocupado mayores a los obtenidos.

En la figura 4.15 se evidencia el proceso de cambio de las señales de salida del módulo FIFO SPI por medio del Control-SPI. Una vez que el FIFO 1 llegó a su máxima capacidad, el sistema transmitió los bits no incluidos en los lotes del FIFO anterior. Como se puede apreciar, una vez que terminó de enviar todos los datos del FIFO 0, el Control-SPI redireccionó inmediatamente al FIFO 1 continuando con los datos sin ningún tipo de interrupción. En paralelo, el sistema siguió recibiendo datos en el tercer FIFO (estado de "Llenado"). Con este sistema se logró transferir datos con el formato establecido por el FT232H, optimizando el tiempo de tal forma que únicamente el IC detenga el sistema durante los lapsos de estado ocupado. Además de transferir datos en lotes con un tamaño de 1 B, se pudo aprovechar al máximo el búfer RX del IC, optimizando al máximo la capacidad de transmisión de datos hacia el computador.

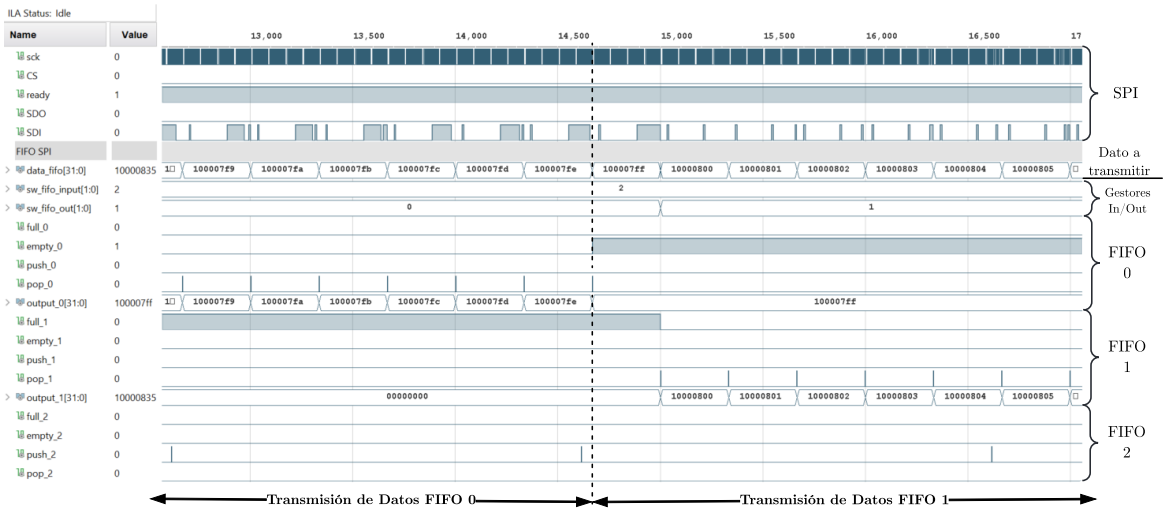


Figura 4.15: Diagrama de Tiempo de Cambio de Señales de Salida del FIFO SPI

La cantidad de datos residuales que quedan en un FIFO posterior a un envío de un lote de datos es variable; sin embargo, redireccionar los FIFOs internos por medio del Control-SPI y no por señales de llenado o vaciado propios de los FIFOs, permitió al sistema conocer cuántos datos residuales y cuántos datos del siguiente FIFO interno debe transferir para completar un lote de datos.

En la figura 4.16, se demuestra cómo el sistema continuó transfiriendo datos de forma continua durante un período de 10 minutos. Se observa que el dato de prueba siguió incrementando su valor sin perder la secuencia. Durante el envío del lote, el FT232H estableció que el tiempo de transferencia había finalizado, por lo que envió constantemente un '1' en la señal `SDO`. Una vez que se completó el lote, el sistema detectó correctamente la señal, finalizando la transferencia.

Una vez finalizada la transmisión de datos, el código de alto nivel cerró el archivo binario, almacenando todos los datos recuperados enviados por el sistema. Llegado a este punto

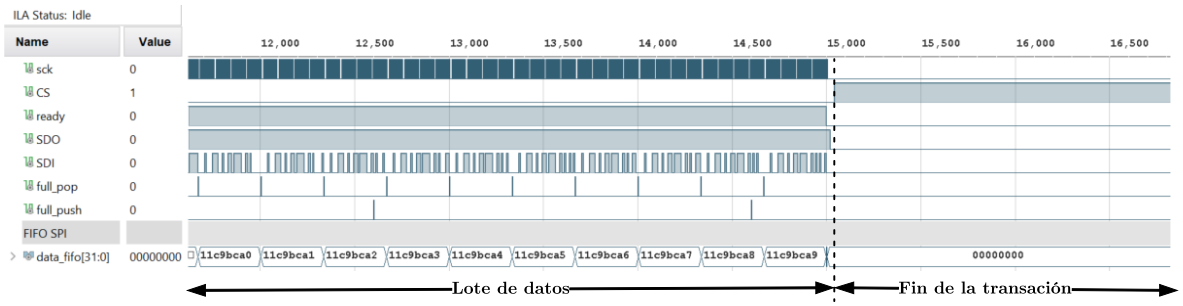


Figura 4.16: Diagrama de Tiempo de Fin de una Transacción con una Duración de 10 min

el sistema ha finalizado completamente. De forma opcional y para verificar los datos de prueba como se muestra en la figura 4.6, se convirtió el archivo binario en un archivo de texto para poder observar y manipular los datos obtenidos. En las tablas 4.4 y 4.5, se presentan los primeros 280 datos y los últimos 280 obtenidos durante la prueba con una duración de 10 minutos.

Finalmente, accede al archivo de texto y verifica la integridad de los datos. Como se puede observar en la figura 4.17 la función de verificación retornó que todos los datos se transfirieron correctamente y no hubo pérdidas de datos durante el proceso de transferencia. En el apéndice A se puede observar la interfaz de usuario en terminal de consola completa utilizada durante las pruebas.

```
============================================================

Verifying the hexadecimal counter in Captura_Datos_10min.txt...

 -> THE HEXADECIMAL COUNTER IS CORRECT!!!

------------------------------------------------------------
```

Figura 4.17: Comprobación en Alto Nivel de la Integridad de los Datos

4.4.5. Tamaño de Archivos

En la tabla 4.6 se observan los archivos obtenidos dado diferentes tiempos de captura de datos. Se puede notar que al tratarse de archivos binarios, el tamaño de las pruebas no excede los 150 MB, a pesar de haber recibido una gran cantidad de datos por un tiempo prolongado.

Tabla 4.6: Tamaño de Archivos Generados según el Tiempo de Captura

Duración (s)	Cantidad de Datos	Tamaño (B)	
		.bin	.txt
1	49 250	192 K	480 K
5	249 250	973 K	2.37 M
60	2 997 250	11.4 M	28.5 M
300	14 998 250	57.2 M	143 M
600	29 998 250	114 M	286 M

Tabla 4.4: Primeros 280 Datos de Pruebas Obtenidos

10000000	10000028	10000050	10000078	100000a0	100000c8	100000f0
10000001	10000029	10000051	10000079	100000a1	100000c9	100000f1
10000002	1000002a	10000052	1000007a	100000a2	100000ca	100000f2
10000003	1000002b	10000053	1000007b	100000a3	100000cb	100000f3
10000004	1000002c	10000054	1000007c	100000a4	100000cc	100000f4
10000005	1000002d	10000055	1000007d	100000a5	100000cd	100000f5
10000006	1000002e	10000056	1000007e	100000a6	100000ce	100000f6
10000007	1000002f	10000057	1000007f	100000a7	100000cf	100000f7
10000008	10000030	10000058	10000080	100000a8	100000d0	100000f8
10000009	10000031	10000059	10000081	100000a9	100000d1	100000f9
1000000a	10000032	1000005a	10000082	100000aa	100000d2	100000fa
1000000b	10000033	1000005b	10000083	100000ab	100000d3	100000fb
1000000c	10000034	1000005c	10000084	100000ac	100000d4	100000fc
1000000d	10000035	1000005d	10000085	100000ad	100000d5	100000fd
1000000e	10000036	1000005e	10000086	100000ae	100000d6	100000fe
1000000f	10000037	1000005f	10000087	100000af	100000d7	100000ff
10000010	10000038	10000060	10000088	100000b0	100000d8	10000100
10000011	10000039	10000061	10000089	100000b1	100000d9	10000101
10000012	1000003a	10000062	1000008a	100000b2	100000da	10000102
10000013	1000003b	10000063	1000008b	100000b3	100000db	10000103
10000014	1000003c	10000064	1000008c	100000b4	100000dc	10000104
10000015	1000003d	10000065	1000008d	100000b5	100000dd	10000105
10000016	1000003e	10000066	1000008e	100000b6	100000de	10000106
10000017	1000003f	10000067	1000008f	100000b7	100000df	10000107
10000018	10000040	10000068	10000090	100000b8	100000e0	10000108
10000019	10000041	10000069	10000091	100000b9	100000e1	10000109
1000001a	10000042	1000006a	10000092	100000ba	100000e2	1000010a
1000001b	10000043	1000006b	10000093	100000bb	100000e3	1000010b
1000001c	10000044	1000006c	10000094	100000bc	100000e4	1000010c
1000001d	10000045	1000006d	10000095	100000bd	100000e5	1000010d
1000001e	10000046	1000006e	10000096	100000be	100000e6	1000010e
1000001f	10000047	1000006f	10000097	100000bf	100000e7	1000010f
10000020	10000048	10000070	10000098	100000c0	100000e8	10000110
10000021	10000049	10000071	10000099	100000c1	100000e9	10000111
10000022	1000004a	10000072	1000009a	100000c2	100000ea	10000112
10000023	1000004b	10000073	1000009b	100000c3	100000eb	10000113
10000024	1000004c	10000074	1000009c	100000c4	100000ec	10000114
10000025	1000004d	10000075	1000009d	100000c5	100000ed	10000115
10000026	1000004e	10000076	1000009e	100000c6	100000ee	10000116
10000027	1000004f	10000077	1000009f	100000c7	100000ef	10000117

Tabla 4.5: Últimos 280 Datos de Pruebas Obtenidos

11c9bb92	11c9bbba	11c9bbe2	11c9bc0a	11c9bc32	11c9bc5a	11c9bc82
11c9bb93	11c9bbbb	11c9bbe3	11c9bc0b	11c9bc33	11c9bc5b	11c9bc83
11c9bb94	11c9bbbc	11c9bbe4	11c9bc0c	11c9bc34	11c9bc5c	11c9bc84
11c9bb95	11c9bbbd	11c9bbe5	11c9bc0d	11c9bc35	11c9bc5d	11c9bc85
11c9bb96	11c9bbbe	11c9bbe6	11c9bc0e	11c9bc36	11c9bc5e	11c9bc86
11c9bb97	11c9bbbf	11c9bbe7	11c9bc0f	11c9bc37	11c9bc5f	11c9bc87
11c9bb98	11c9bbc0	11c9bbe8	11c9bc10	11c9bc38	11c9bc60	11c9bc88
11c9bb99	11c9bbc1	11c9bbe9	11c9bc11	11c9bc39	11c9bc61	11c9bc89
11c9bb9a	11c9bbc2	11c9bbea	11c9bc12	11c9bc3a	11c9bc62	11c9bc8a
11c9bb9b	11c9bbc3	11c9bbeb	11c9bc13	11c9bc3b	11c9bc63	11c9bc8b
11c9bb9c	11c9bbc4	11c9bbec	11c9bc14	11c9bc3c	11c9bc64	11c9bc8c
11c9bb9d	11c9bbc5	11c9bbed	11c9bc15	11c9bc3d	11c9bc65	11c9bc8d
11c9bb9e	11c9bbc6	11c9bbee	11c9bc16	11c9bc3e	11c9bc66	11c9bc8e
11c9bb9f	11c9bbc7	11c9bbef	11c9bc17	11c9bc3f	11c9bc67	11c9bc8f
11c9bba0	11c9bbc8	11c9bbf0	11c9bc18	11c9bc40	11c9bc68	11c9bc90
11c9bba1	11c9bbc9	11c9bbf1	11c9bc19	11c9bc41	11c9bc69	11c9bc91
11c9bba2	11c9bbca	11c9bbf2	11c9bc1a	11c9bc42	11c9bc6a	11c9bc92
11c9bba3	11c9bbcb	11c9bbf3	11c9bc1b	11c9bc43	11c9bc6b	11c9bc93
11c9bba4	11c9bbcc	11c9bbf4	11c9bc1c	11c9bc44	11c9bc6c	11c9bc94
11c9bba5	11c9bbcd	11c9bbf5	11c9bc1d	11c9bc45	11c9bc6d	11c9bc95
11c9bba6	11c9bbce	11c9bbf6	11c9bc1e	11c9bc46	11c9bc6e	11c9bc96
11c9bba7	11c9bbcf	11c9bbf7	11c9bc1f	11c9bc47	11c9bc6f	11c9bc97
11c9bba8	11c9bbd0	11c9bbf8	11c9bc20	11c9bc48	11c9bc70	11c9bc98
11c9bba9	11c9bbd1	11c9bbf9	11c9bc21	11c9bc49	11c9bc71	11c9bc99
11c9bbaa	11c9bbd2	11c9bbfa	11c9bc22	11c9bc4a	11c9bc72	11c9bc9a
11c9bbab	11c9bbd3	11c9bbfb	11c9bc23	11c9bc4b	11c9bc73	11c9bc9b
11c9bbac	11c9bbd4	11c9bbfc	11c9bc24	11c9bc4c	11c9bc74	11c9bc9c
11c9bbad	11c9bbd5	11c9bbfd	11c9bc25	11c9bc4d	11c9bc75	11c9bc9d
11c9bbae	11c9bbd6	11c9bbfe	11c9bc26	11c9bc4e	11c9bc76	11c9bc9e
11c9bbaf	11c9bbd7	11c9bbff	11c9bc27	11c9bc4f	11c9bc77	11c9bc9f
11c9bbb0	11c9bbd8	11c9bc00	11c9bc28	11c9bc50	11c9bc78	11c9bca0
11c9bbb1	11c9bbd9	11c9bc01	11c9bc29	11c9bc51	11c9bc79	11c9bca1
11c9bbb2	11c9bbda	11c9bc02	11c9bc2a	11c9bc52	11c9bc7a	11c9bca2
11c9bbb3	11c9bbdb	11c9bc03	11c9bc2b	11c9bc53	11c9bc7b	11c9bca3
11c9bbb4	11c9bbdc	11c9bc04	11c9bc2c	11c9bc54	11c9bc7c	11c9bca4
11c9bbb5	11c9bbdd	11c9bc05	11c9bc2d	11c9bc55	11c9bc7d	11c9bca5
11c9bbb6	11c9bbde	11c9bc06	11c9bc2e	11c9bc56	11c9bc7e	11c9bca6
11c9bbb7	11c9bbdf	11c9bc07	11c9bc2f	11c9bc57	11c9bc7f	11c9bca7
11c9bbb8	11c9bbe0	11c9bc08	11c9bc30	11c9bc58	11c9bc80	11c9bca8
11c9bbb9	11c9bbe1	11c9bc09	11c9bc31	11c9bc59	11c9bc81	11c9bca9

Capítulo 5

Sistema de almacenamiento de datos en tiempo real en tarjetas microSD

5.1. General

El capítulo se centra en el desarrollo de un "Sistema de almacenamiento de datos en tiempo real en tarjetas microSD". El sistema de almacenamiento tiene el propósito de respaldar los datos crudos provenientes de la unidad de adquisición (ADC) en un período de 10 minutos.

5.2. Diseño del sistema

El diseño RTL del sistema consta de tres secciones:

- **Módulo Interfaz de Usuario (MIU)**: módulo externo encargado de recibir y eliminar el efecto de rebote de las señales provenientes de los periféricos de la FPGA (botones y switches), para ser enviadas al SHM.
- **Sistema de Empaquetado y Segmentación de Bits ADC (SES)** (Sistema Secundario): encargado de capturar, procesar y almacenar los datos crudos del ADC de la unidad de adquisición y procesamiento de señales.
- **Sistema Host para Múltiples Tarjetas microSD (SHM)** (Sistema Principal/Host): encargado de la comunicación y transmisión de datos con la tarjeta(s) microSD.

Para facilitar la comprensión, se han simplificado los siguientes diagramas mediante el uso de buses de señales. Cada diagrama de bloques viene acompañado (si se requiere) de una tabla la cual detalla las señales contenidas en cada bus. La figura subsiguiente presenta el diagrama de bloques del sistema completo.

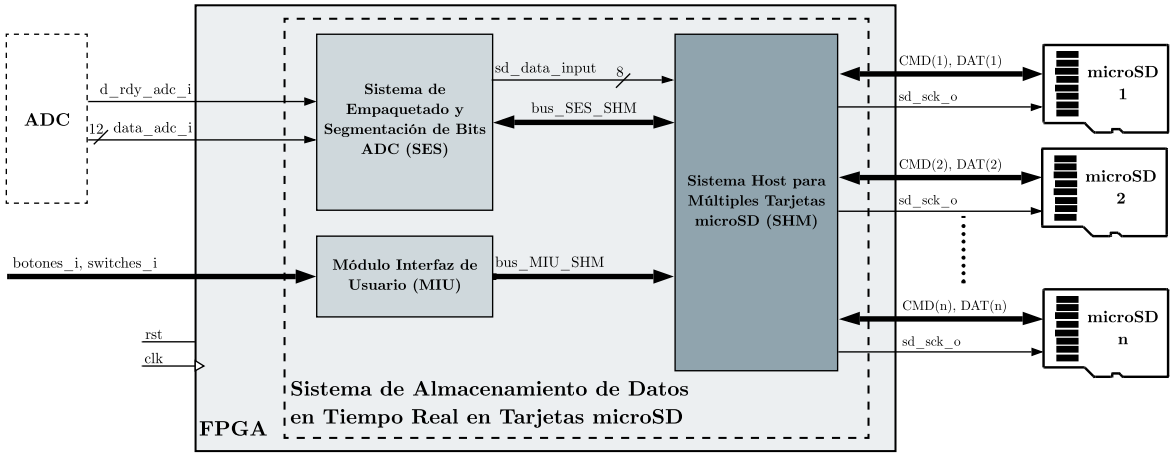


Figura 5.1: Diagrama de Bloques del Sistema de Almacenamiento de Datos en Tiempo Real en Tarjetas microSD

Tabla 5.1: Señales de los Buses de la Figura 5.1

Bus	Señal	Bits	Entrada	Salida
bus_SES_SHM	sd_fifo_wr_push	1	SHM	SES
	sd_start_wr	1	SES	SHM
	sd_fifo_wr_full	1		
	sd_fifo_wr_empty	1		
bus_MIU_SHM	btn_start	1	SHM	1MIU
	btn_select			
	sw_slct_mode	WIDTH_SW		

Las señales `d_rdy_adc_i` y `data_adc_output_i` corresponden a las señales de salida del ADC de la unidad de adquisición y procesamiento de señales.

5.2.1. Módulo Interfaz de Usuario (MIU)

El Módulo Interfaz de Usuario es un módulo externo que procesa las señales provenientes de los periféricos de la FPGA, específicamente botones y switches, y las envía al SHM. Su función principal es mitigar el efecto de rebote (bounce) en estas señales. Para ello, cuenta con dos módulos 'Debounce', cada uno dedicado a manejar las señales de un periférico específico. En la figura 5.2 se puede visualizar el diagrama de bloques del Módulo Interfaz de Usuario (MIU).

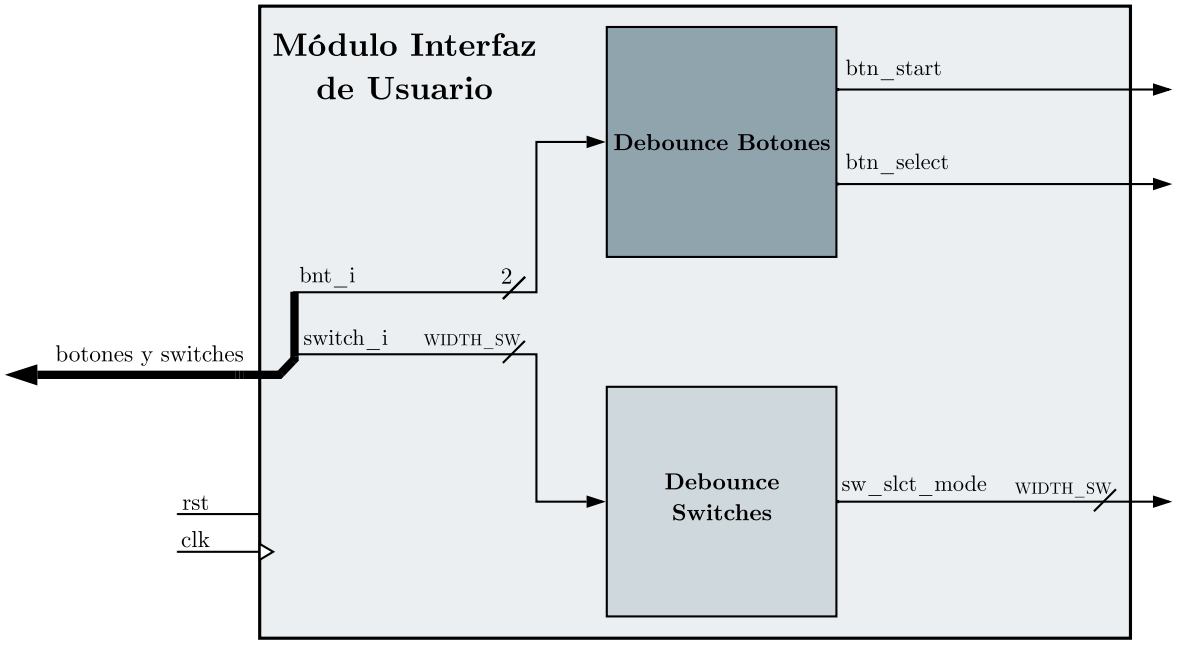


Figura 5.2: Diagrama de Bloques del Módulo de Interfaz de Usuario (`MIU`)

La elección de que este módulo sea externo al SHM se fundamenta principalmente en la versatilidad del diseño del SHM. Ya que SHM se diseñó para adaptarse a cualquier instancia, lo que significa que podría ser implementado en un sistema de mayor tamaño donde las señales que recibe el SHM por parte de este módulo no se originen necesariamente de un periférico, sino que podrían ser señales digitales provenientes del sistema más amplio. En este contexto, el módulo de Interfaz de Usuario no tendría una función específica. Por lo tanto, su exclusión como componente externo resulta más práctica que si estuviera integrado internamente en el SHM.

Debounce Botones

El módulo Debounce Botones es el encargado de generar un pulso estable derivado de la señal de un botón. Su funcionamiento se basa en el diagrama de flujo de la figura 5.3. Cuando se detecta una señal en alto en el botón, el módulo produce un pulso en `btn`, que

es transmitido a SHM. A través de un registro contador restador, con un valor máximo establecido por el parámetro local `SPLITTER`, se impide el envío de pulsos adicionales al SHM, incluso si el botón permanece en alto. Cuando el contador alcanza el valor de 0, el módulo vuelve a leer la señal del botón y repite el proceso. El valor del parámetro `SPLITTER` para efectos de este documento es equivalente a la espera de 0.5 segundos. El módulo presenta dos instancias: la primera envía la señal `btn_start` y la segunda `btn_select`.

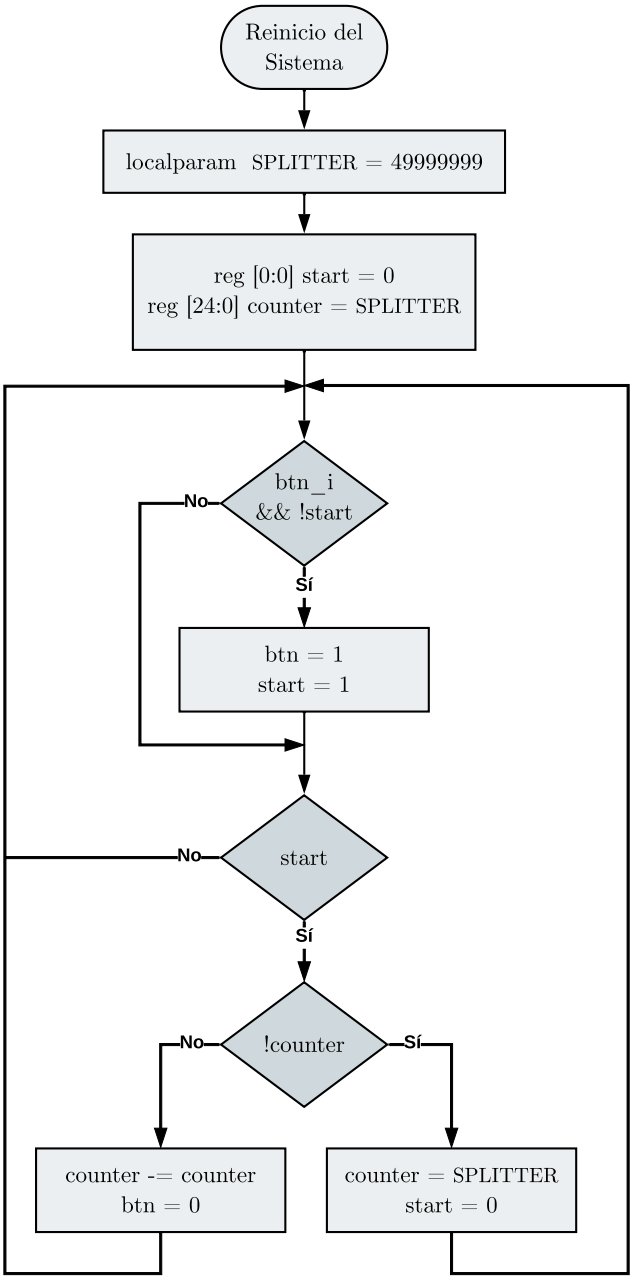


Figura 5.3: Diagrama de Flujo de Debounce Botones

Debounce Switches

El módulo Debounce Switches es el encargado de generar una señal estable derivada de la señal de un switch. La instancia de este módulo es parametrizable mediante el parámetro `WIDTH_SW`. Cada salida de las instancias de estos módulos se concatena en la señal `sw_slct_mode`, de tal forma que la señal tiene una longitud de `WIDTH_SW`.

El funcionamiento del módulo se basa en el diagrama de flujo de la figura 5.4. Utiliza un contador, el cual cada vez que detecta una señal en alto del switch su valor incrementa y decrementa si detecta una señal en bajo. Cuando el contador logra alcanzar su valor máximo o su valor mínimo, se envía un '1' lógico o un '0 lógico' respectivamente en el bit de la señal `sw_slct_mode` correspondiente.

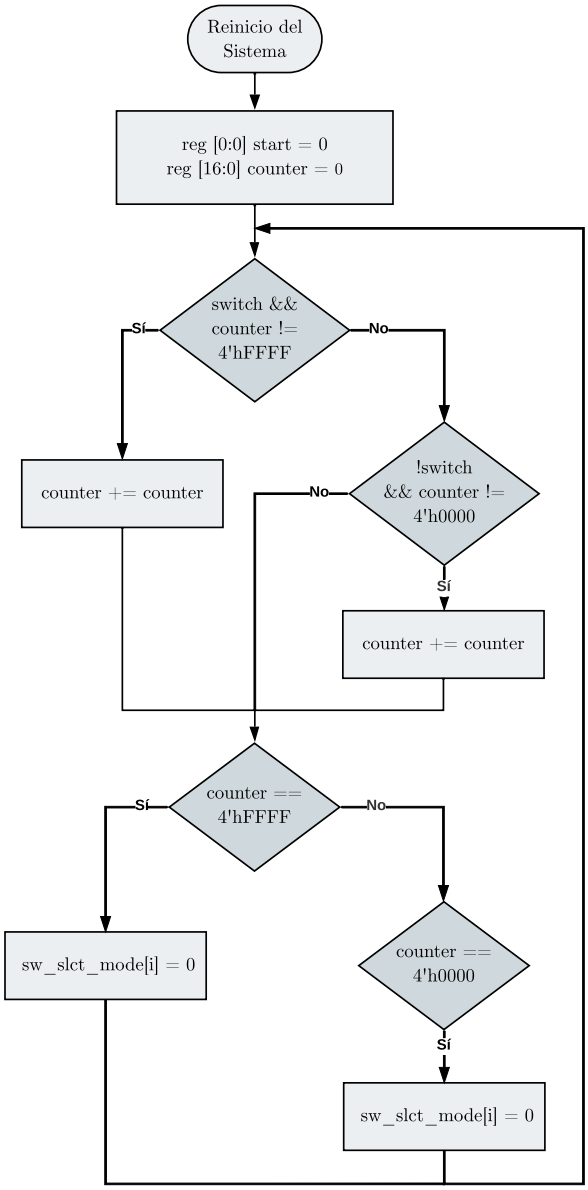


Figura 5.4: Diagrama de Flujo de Debounce Switch

5.2.2. Sistema de Empaquetado y Segmentación de Bits ADC (SES)

El Sistema Secundario es el sistema encargado de transferir los datos externos al SHM, de tal forma que solo baste modificar o reemplazar este sistema para transferir cualquier otro dato externo que el usuario así lo requiera. El Sistema de Empaquetado y Segmentación de Bit (SES) es el Sistema Secundario utilizado en este documento. El SES está diseñado para adaptar y almacenar los datos crudos provenientes del ADC de la unidad de adquisición y procesamiento de señales, de manera que sean compatibles con el SHM para su posterior transferencia.

El SES está conformado por cuatro módulos: Empaquetado y Segmentación, Control de Empaquetado y Segmentación, FIFO ADC y Gestor de FIFOs, como se muestra en la figura 5.5. Adicionalmente, utiliza un mux para habilitar o deshabilitar la entrada de datos del ADC. El mux es controlado por la señal `sd_start_wr` proveniente del CM del Sistema Principal.

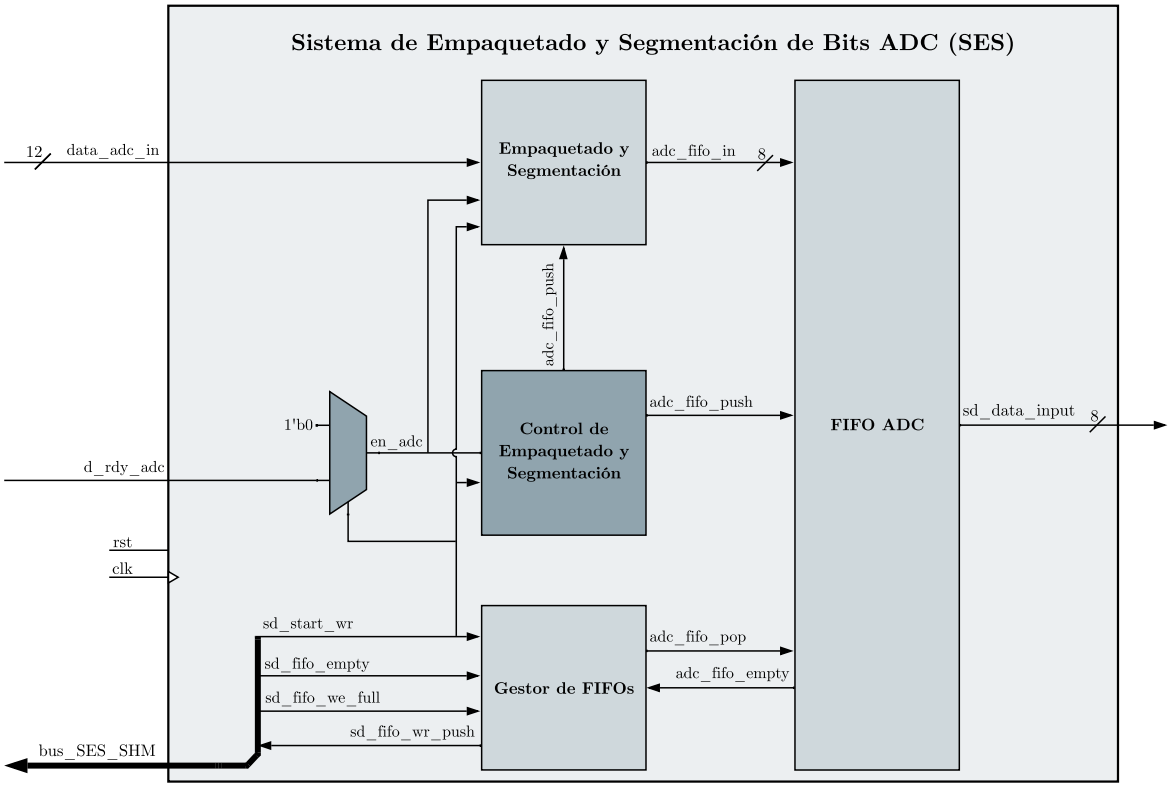


Figura 5.5: Diagrama de Bloques del Sistema de Empaquetado y Segmentación de Bits ADC (SES)

Empaquetado y Segmentación

El módulo de Empaquetado y Segmentación se encarga de adaptar los datos del ADC a un formato compatible con el SHM. Ya que las tarjetas microSD trabajan con datos de 8 bits (1 byte), por ende el SHM mantiene el mismo formato. Sin embargo, los datos del ADC están conformados por 12 bits, por lo que existe una incompatibilidad entre los datos de entrada y los datos que el SHM puede transferir.

El funcionamiento de este módulo se basa en el diagrama de flujo mostrado en la figura 5.6. Una vez iniciado el sistema, cada pulso de la señal `en_adc`, el módulo captura el dato proveniente del ADC y lo almacena en los 12 bits LSB del registro.

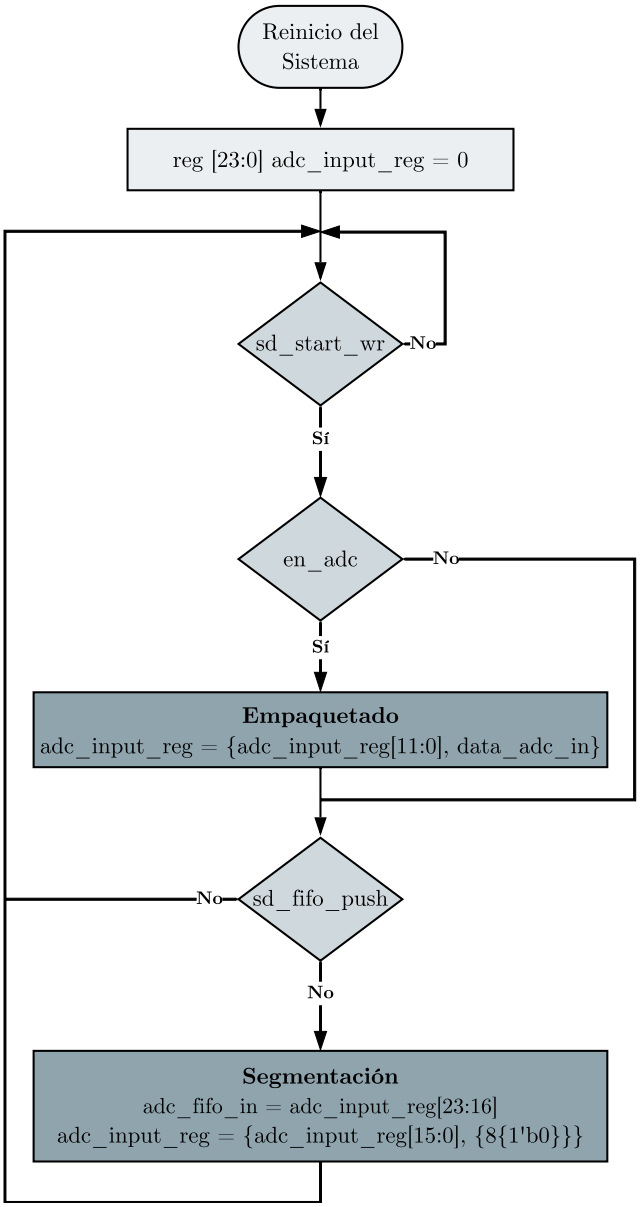


Figura 5.6: Diagrama de flujo del Empaquetado y Segmentación de Bits

Este proceso se repite, pero en esta ocasión, el dato previamente capturado se desplaza a los 12 bits MSB del registro. De esta forma, se logra concatenar dos datos del ADC en un único registro (empaquetado). Posteriormente, cuando la señal `sd_fifo_push` se activa, los 8 bits más significativos del registro se envían a la salida (segmentación). Luego, el registro realiza un desplazamiento a la izquierda de 8 bits y el proceso de segmentación se repite dos veces más. De esta manera, los 24 bits que corresponden a dos datos del ADC se envían como tres datos de 1 byte cada uno.

Control de Empaquetado y Segmentación

El Control de Empaquetado y Segmentación se encarga de controlar el módulo Empaquetado y Segmentación mediante una Máquina de Estados Finita de Mealy. El diagrama de bloque del FSM se muestra en la figura 5.7 y en la tabla 5.2 se describen los estados del FSM:

Tabla 5.2: Descripción de Estados del FSM del Control de Bits Control de Empaquetado y Segmentación

Estado	Descripción
RECEIVE_ DATA	Controla la cantidad de datos capturados por el módulo Empaquetado y Segmentación.
SEND_FIFO	Controla el envío de los datos segmentados y le indica al FIFO ADC cuando debe almacenar un dato.

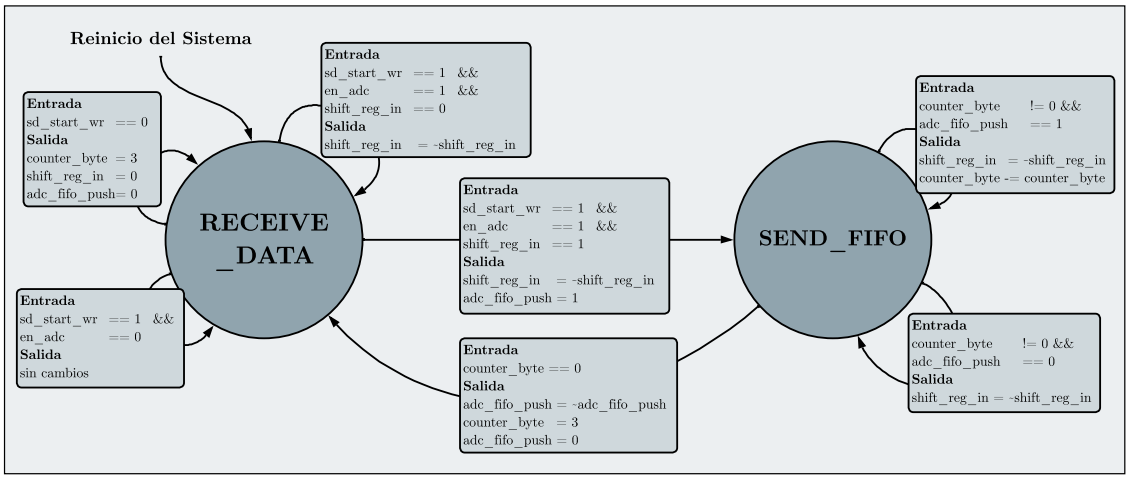


Figura 5.7: Diagrama de Estados del Control de Empaquetado y Segmentación

FIFO ADC

El FIFO ADC corresponde al reservorio donde se almacenan los datos segmentados del ADC y los cuales están listos para ser enviados al SHM y ser transferidos a la tarjeta microSD. Funciona internamente mediante un conjunto de FIFOs en paralelo. Estos FIFOs están diseñados haciendo uso del IP core "FIFO Generator" de Vivado, con un ancho de escritura de 1 byte y una profundidad de escritura de 131072. La cantidad de FIFOs internos se define mediante el parámetro `NUM_FIFOS`. Para efectos de este documento, este parámetro se establece con un valor de 4. Por lo tanto, el módulo FIFO ADC cuenta con un tamaño total de 524.288 kB. La elección de dicho tamaño radica principalmente en el estado `prg` o estado de ocupado que presenta la tarjeta microSD. Este tiempo de ocupado es variable y depende de las especificaciones de la tarjeta. Por tanto, el tamaño del FIFO se determinó mediante resultados experimentales para evitar la pérdida de datos durante este tiempo, en donde el host no puede transferir datos a la tarjeta. En la figura 5.8 se muestran los componentes del módulo.

El módulo utiliza dos registros: el primero asigna un FIFO específico para el ingreso de los datos y el segundo asigna un FIFO específico para el egreso de los datos. El funcionamiento de dichos registros está basado en los diagramas de flujo de la figura 5.9. El FIFO de Escritura funciona mediante un sistema rotativo de FIFOs, en donde cada FIFO rota entre cuatro roles:

- **Captura**: el FIFO está en el proceso de capturar los datos entrantes. El Gestor de Entrada redirecciona las señales de entrada del módulo al FIFO de este estado. Una vez que se ha llenado, pasa al estado "Lleno".
- **Lleno**: el FIFO ha completado su proceso de llenado y está lista para transferir los datos. Una vez que el FIFO en el estado de "Transferencia" ha completado la transferencia de datos, el FIFO de este estado transiciona al estado de "Transferencia". Sin embargo, puede haber más de un FIFO en este estado; de darse esta condición, el sistema rotativo asegura que el FIFO que se transiciona al estado de "Transferencia" es siempre la que precedió a las demás en su llenado.
- **Transferencia**: el FIFO está actualmente en el proceso de transferir los datos. El Gestor de Salida redirecciona las señales de salida del módulo al FIFO de este estado. Una vez que se ha vaciado, pasa al estado "Espera".
- **Espera**: el FIFO está a la espera de comenzar a capturar los datos entrantes. Una vez que el FIFO en el estado de "Captura" ha completado su proceso de captura de datos, este FIFO transiciona al estado de "Captura". Sin embargo, puede haber más de un FIFO en este estado; de darse esta condición, el sistema rotativo asegura que el FIFO que se transiciona al estado de "Captura" es siempre la que precedió a las demás en su vaciado.

Al iniciar el sistema, todos los FIFOs están en el estado de "Espera" a excepción de un, el cual comienza en el estado de 'Captura'.

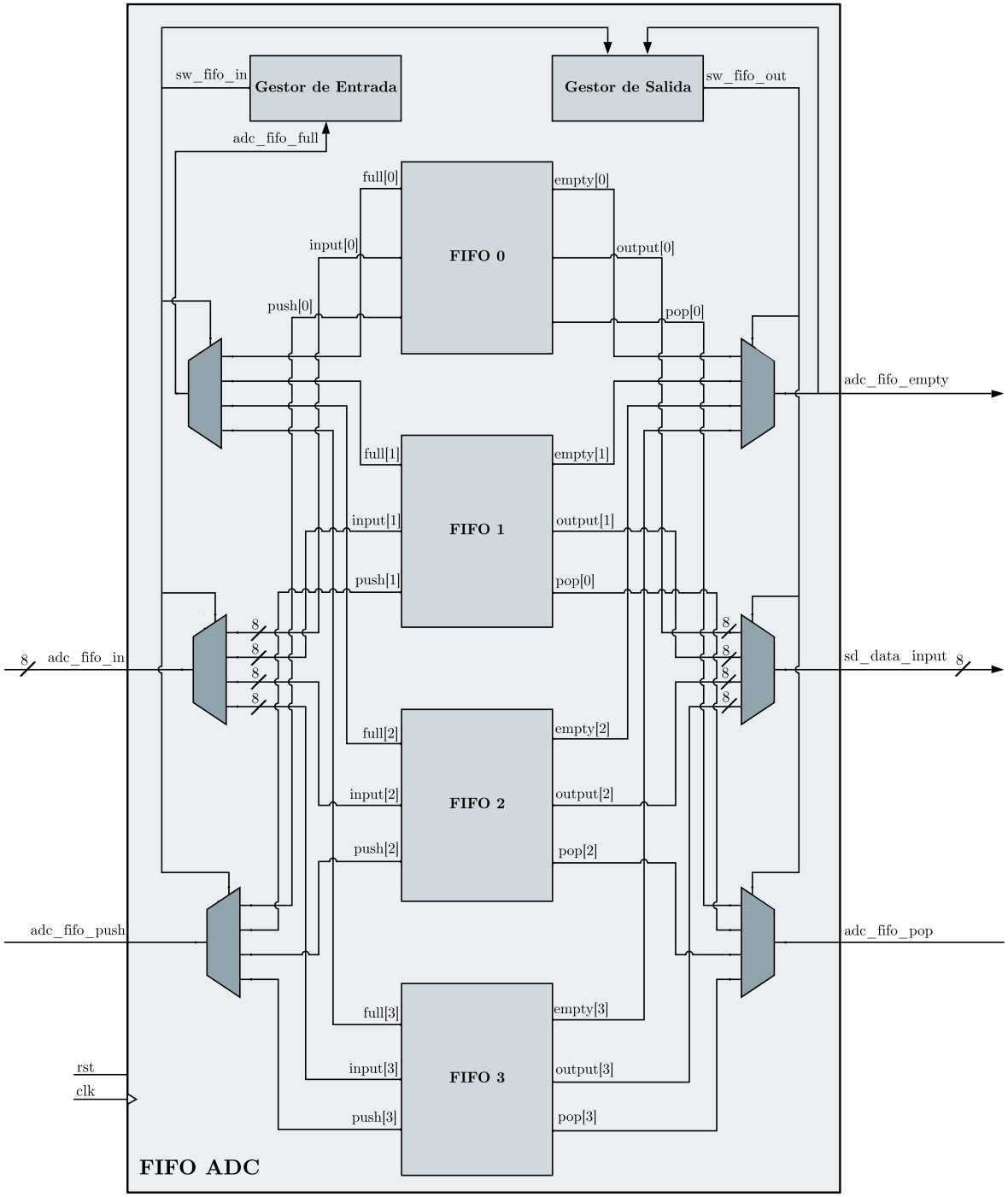
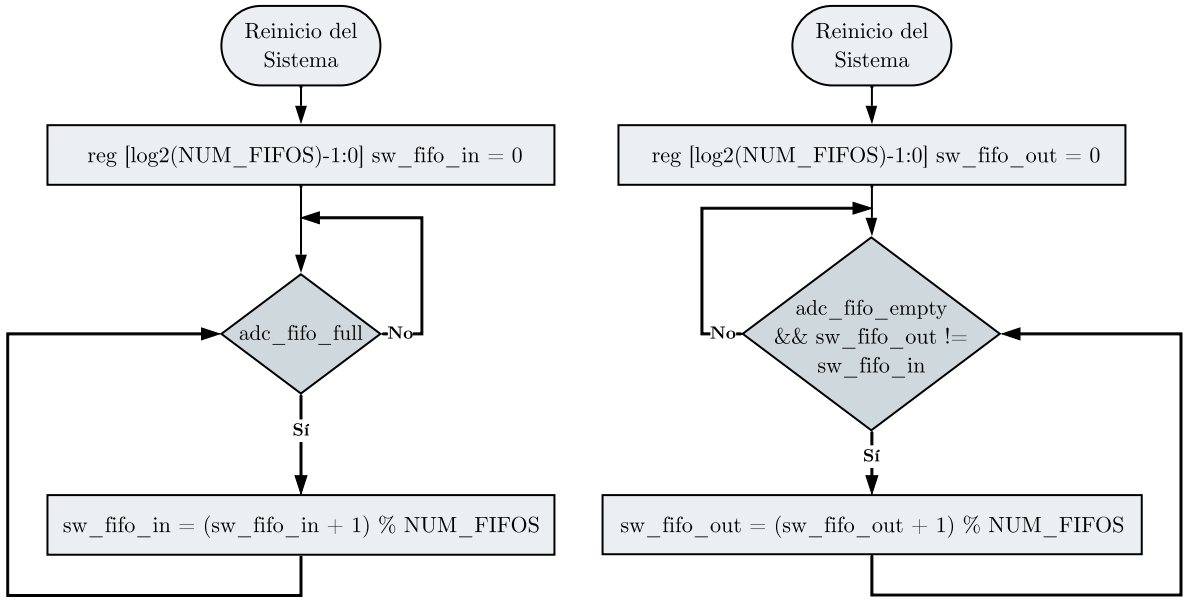


Figura 5.8: Diagrama de Bloques del FIFO ADC



(a) Gestor de Entrada **(b)** Gestor de Salida

Figura 5.9: Diagrama de Flujo de los Gestores Entrada/Salida de FIFO ADC

Gestor de FIFOs

El FIFO ADC del Sistema Secundario y el FIFO de Escritura del Sistema Principal (ver sección 5.2.3) están estrechamente relacionados. Puesto que la salida de datos del FIFO ADC corresponde a la entrada de datos del FIFO de Escritura. Existen dos motivos del porqué se diseñaron dos FIFOs en cascada: el primero se debe a que ambos FIFOs pertenecen a dos sistemas completamente independientes. Se puede dar el caso en que se utilice un Sistema Secundario que no requiera un FIFO de salida. El segundo se debe a que el Sistema Principal necesita asegurar que haya 512 datos disponibles para la transmisión y esto se logra cuando el FIFO de Escritura llega a su capacidad máxima. De esta manera, sin importar el Sistema Secundario, el SHM puede asegurar que la transmisión corresponderá a los datos que el Sistema Secundario brinde.

Por lo cual, el módulo Gestor de FIFOs indica al FIFO ADC cuando debe extraer un dato y, a su vez, le indica al FIFO de Escritura del Sistema Principal cuando hay un dato disponible para almacenar. Su funcionamiento se basa en el diagrama de flujo de la figura 5.10. Si el FIFO ADC contiene datos a transmitir y el FIFO de Escritura no está en su capacidad máxima, el módulo habilita la señal de `adc_fifo_pop`, lo que origina que el FIFO ADC coloque un nuevo dato en la señal `sd_data_input`. En el siguiente flanco ascendente del reloj del sistema, deshabilita la señal `adc_fifo_pop` y habilita `sd_fifo_wr_push`, lo que provoca que el FIFO de Escritura lea y captura el dato de la señal `sd_data_input`. El proceso se repite de forma constante mientras el SHM permita la entrada de datos por parte del Sistema Secundario.

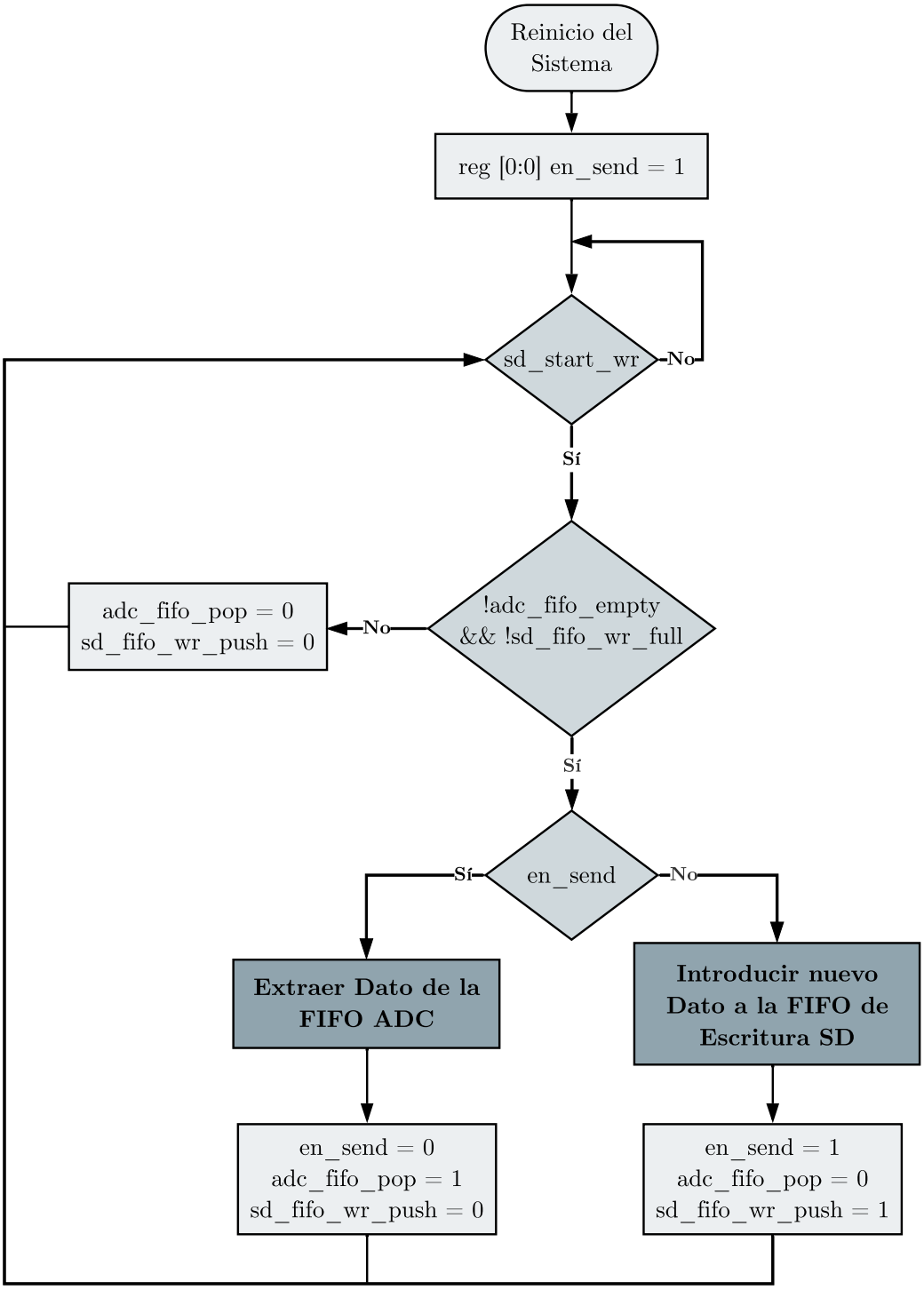


Figura 5.10: Diagrama de Flujo del Gestor de FIFO ADC

5.2.3. Sistema Host para Múltiples Tarjetas microSD (SHM)

El "Sistema host para múltiples tarjetas microSD" (SHM) se diseñó para ser un sistema independiente capaz de almacenar los datos provenientes de sistemas secundarios en una tarjeta microSD, así como para leer los datos provenientes de la misma. En este documento se utiliza este sistema para escribir los datos crudos del ADC segmentados provenientes del sistema secundario: empaquetado y segmentación de bits ADC. El host es compatible con el estándar UHS-I en el modo de velocidad "Default Speed" con una velocidad de bus de 12.5 MB/s como se menciona en el Capítulo 3.4.8. Esto con el fin de contar con un margen apropiado entre la tasa de transferencia de escritura en la microSD y la tasa de transferencia del ADC de la unidad de adquisición y procesamiento de señales. Adicionalmente, el sistema tiene soporte para gestionar un número indefinido de tarjetas microSD SDHC y/o SDXC.

El sistema tiene la capacidad de ejecutar todos los comandos mencionados en el documento "SD Specifications Part 1 Physical Layer Specification" (ver [28]) y realizar operaciones de lectura y escritura. No obstante, el capítulo detalla y hace uso únicamente de los comandos descritos en el Capítulo 3.4.9 y operaciones de escritura.

El SHM se divide en cuatro bloques principales:

- **Unidad Central de Múltiples Tarjetas microSD (UCM)**: gestiona las conexiones a la(s) tarjeta(s) microSD, del mismo modo direcciona las señales del host a la tarjeta seleccionada y viceversa.
- **Unidad de Control de Reloj (UCR)**: genera el reloj `sck` para la(s) tarjeta(s) microSD y trasmite dos pulsos a las unidades adyacentes correspondientes al flanco ascendente y descendente de `sck`.
- **Unidad Central de Comandos (UCC)**: unidad encargada de la transmisión de comandos y la recepción de la línea `CMD`. Contiene el módulo Control Maestro CM el cual controla todas las unidades del sistema.
- **Unidad Central de Datos (UCD)**: unidad encargada de la transmisión y recepción de los datos mediante la línea `DAT`.

La figura 5.11 muestra el diagrama de bloques del Sistema Host para Múltiples Tarjetas microSD (SHM).

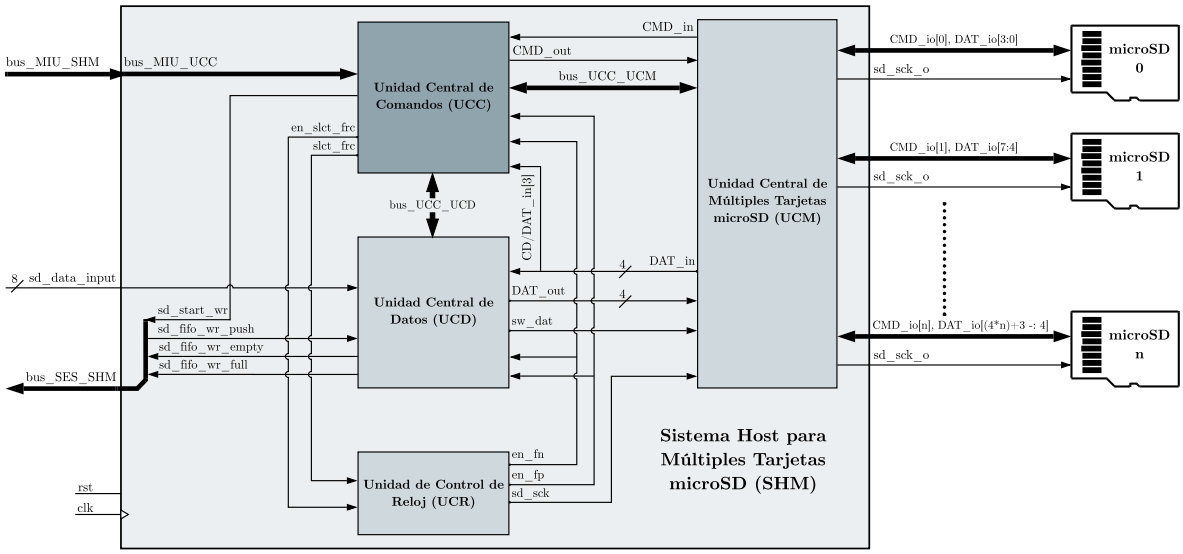


Figura 5.11: Diagrama de Bloques del Sistema Host para Múltiples Tarjetas microSD (SHM)

Tabla 5.3: Señales de los Buses de la Figura 5.11

Bus	Señal	Bits	Entrada	Salida
bus_UCC_UCM	sd_id	log2(SD_LIST)	UCM	UCC
	sw_cmd	1		
	we_log	1		
	sd_log_i	32		
	sd_log_o	32	UCC	UCM
bus_UCC_UCD	finish_data	3	UCC	UCD
	crc_status	5		
	read_busy	1	UCD	UCC
	start_transfer	3		

Unidad Central de Múltiples Tarjetas microSD (UCM)

La Unidad Central de Múltiples Tarjetas microSD es la unidad diseñada para comunicarse con un número teórico indefinido de tarjetas microSD. El UCM es parametrizable mediante el uso del parámetro `SD_LIST`, el cual indica la cantidad de tarjetas microSD que el diseño va a soportar.

La figura 5.12 muestra el diagrama de bloques de UCM. Está conformado por `SD_LIST` (igual a 2, por motivos de simplificación) módulos Conexión Tarjeta. Por medio de un

generate cada módulo crea una conexión de tarjeta mediante el uso de lógica combinacional como se puede observar en la figura 5.13. La señal `sd_id` selecciona la tarjeta que el host quiere operar y el UCM se encarga de direccionar las señales del host a la conexión respectiva. Así mismo, UCM divide las líneas bidireccionales de la tarjeta en dos líneas individuales: la primera destinada a la transmisión serial del host hacia la tarjeta (`CMD_out` y `DAT_out`) y la segunda a la transmisión serial de la tarjeta al host (`CMD_in` y `DAT_in`).

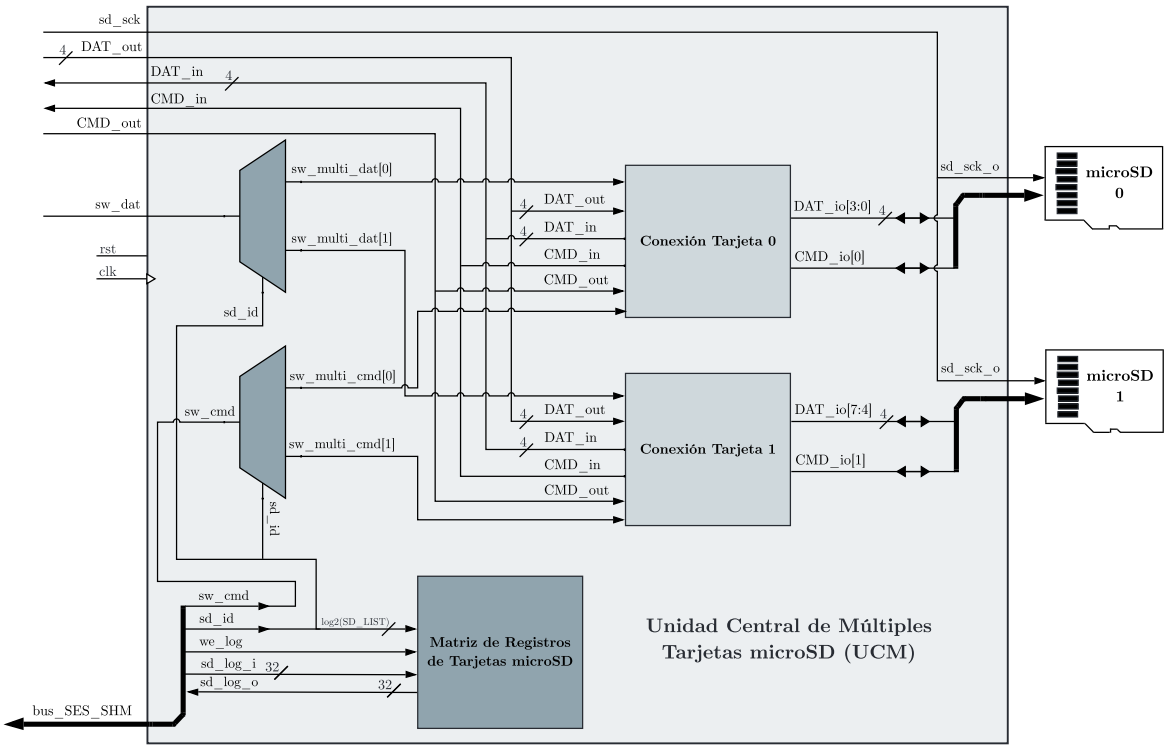


Figura 5.12: Diagrama de Bloques de la Unidad Central de Múltiples Tarjetas microSD (UCM)

Mediante las señales `sw_cmd` y `sw_dat` CC y CD respectivamente, controlan la dirección de transmisión de las líneas `CMD_io` y `DAT_io` del módulo Conexión Tarjeta seleccionado. Un valor en bajo configura las líneas en modo de entrada (alta impedancia), mientras que un valor en alto las configura en modo salida. Las tarjetas no seleccionadas se deben mantener en estado `stby`, tal como se menciona en el Capítulo 3.4.15.

El UCM contiene el bloque Matriz de Registros de Tarjetas microSD, el cual consiste en una matriz de registros de 32 bits. La profundidad de la matriz se define mediante el parámetro `SD_LIST`. Cada registro contiene información de una tarjeta microSD, dicha información es utilizada por el host para conocer el estado de la tarjeta, así como almacenar y leer el `RCA` específico de la tarjeta. El host apunta a un registro de una tarjeta utilizando la señal `sd_id`, que representa el número de fila en la matriz de registros. Con las señales `sd_log_i` y `we_log`, el host escribe en el registro seleccionado y mediante la señal `sd_log_o` accede al contenido del registro. La figura 5.14 muestra una matriz

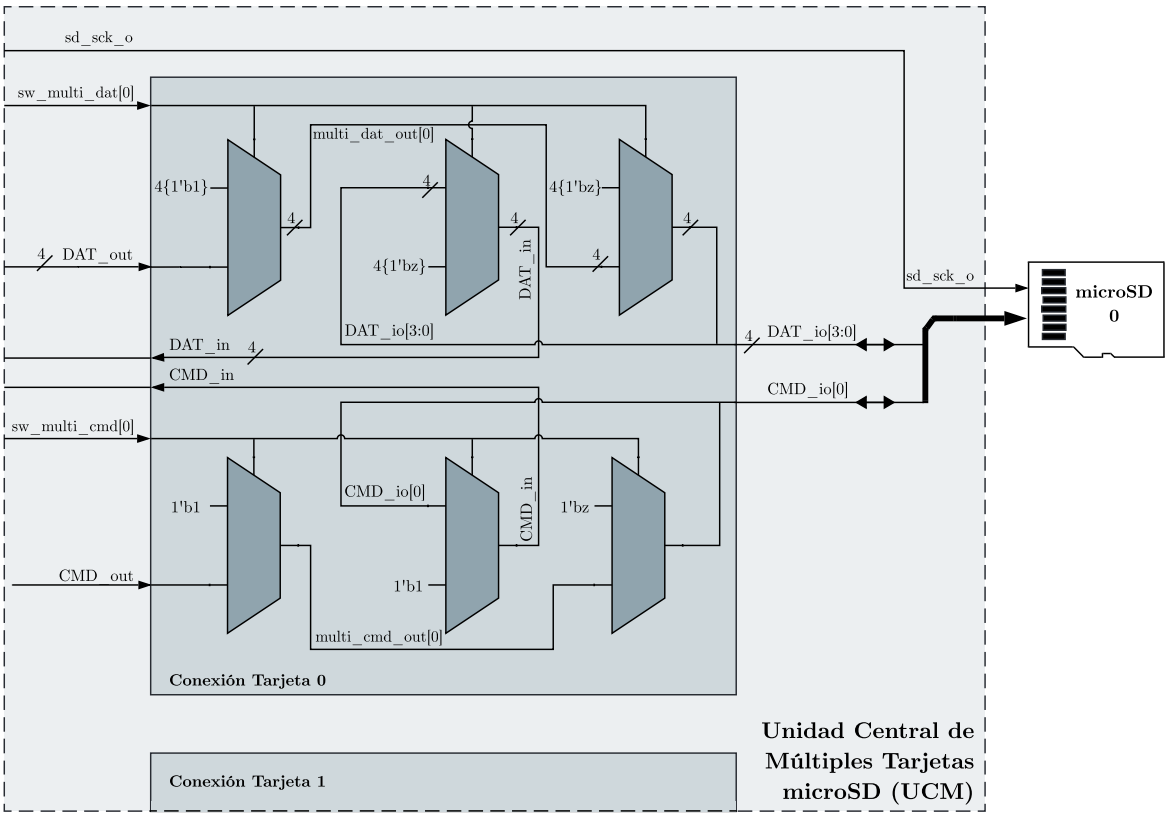


Figura 5.13: Diagrama de Bloques de un Módulo de Conexión Tarjeta

de registros para `n` tarjetas, donde `n` es un número entero. Conjuntamente, la tabla 5.4 presenta los valores de cada columna y su definición.

ID	[31]	[30]	[29]	[28:21]	[20:16]	[15:0]
1	Estado SD 1	CD 1	Mode SD 1	N/A	Última Condición 1	RCA 1
2	Estado SD 2	CD 2	Mode SD 2	N/A	Última Condición 2	RCA 2
	⋮	⋮	⋮	⋮	⋮	⋮
n	Estado SD n	CD n	Mode SD n	N/A	Última Condición n	RCA n

Figura 5.14: Matriz de Registros de Tarjetas microSD

Tabla 5.4: Valores y Definición de la Matriz de Registros de Tarjetas microSD

Campo	Valor (bits)	Definición
Estado SD	1'b0	Tarjeta inactiva (`ina`).
	1'b1	Tarjeta activa (`act`).
CD	1'b0	Tarjeta detectada.
	1'b1	Tarjeta no detectada.
Mode SD	1'b0	Modo 1-bit.
	1'b1	Modo 4-bits.
Última Condición	5{1'bx}	Último CRC status o Error.
`RCA`	16{1'bx}	Dirección Relativa.

Unidad de Control de Reloj (UCR)

La Unidad de Control de Reloj (UCR) genera y establece la frecuencia de reloj que SHM utiliza para la transmisión de datos hacia la microSD. Está compuesto por tres módulos: Selector de Reloj, Generador de Reloj y Detector de Flancos, como se muestra en el diagrama de bloques de la figura 5.15.

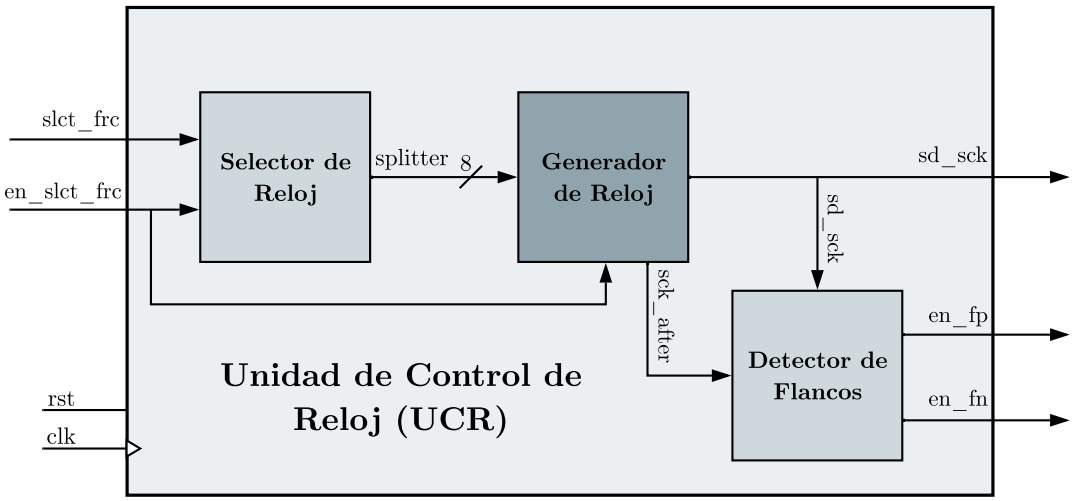


Figura 5.15: Diagrama de Bloques de la Unidad de Control de Reloj (UCR)

El Generador de Reloj utiliza lógica secuencial para generar la señal `sd_sck` mediante el diagrama de flujo de la figura 5.16. La señal `splitter` define el valor máximo del contador, por ende, establece la frecuencia del reloj `sd_sck`. La tabla 5.5 define el valor de `splitter`, la frecuencia asociada a dicho valor y el estado de la señal `sd_sck` según las señales `en_slct_frc` y `slct_frc`.

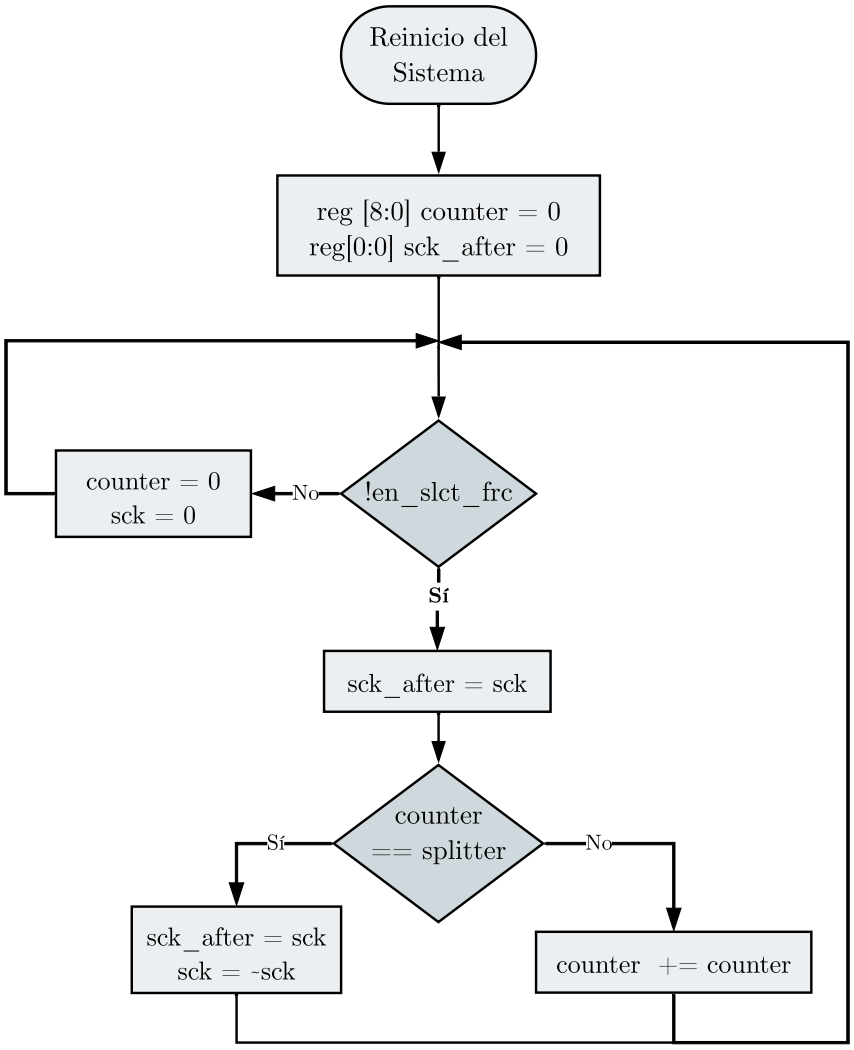


Figura 5.16: Diagrama de Flujo del Generador de Reloj

Tabla 5.5: Frecuencia y Estado del Reloj `sd_sck`

en_slct_frc	slct_frc	splitter		sck
		Valor	Frecuencia	Estado
0	x	Sin cambios	Sin cambios	Habilitado
1	0	'h7c	400 kHz	Deshabilitado
	1	1'h1	25 MHz	

En el Generador de Reloj, el registro `sck_after` retiene el valor anterior de `sd_sck`. El Detector de Flancos lee ambas señales y realiza dos operaciones lógicas `AND`. La primera operación se realiza entre el valor actual de `sd_sck` y el valor negado de `sck_after`, generando un pulso en la señal `en_fp` en el flanco positivo de `sd_sck`. La segunda operación

se realiza entre el valor negado actual de `sd_sck` y la señal `sck_after`, generando un pulso en la señal `en_fn` en el flanco negativo de `sd_sck`.

Las unidades adyacentes utilizan estas señales para preparar la información que la tarjeta muestrea (`en_fn`) e identificar el momento adecuado para capturar la información enviada por la tarjeta (`en_fp`).

Unidad Central de Comandos (UCC)

La Unidad Central de Comandos está conformado por el Control Maestro (CM) y el Centro de Comandos (CC) como se muestra en el diagrama de bloques de la figura 5.17. El CM se detalla más adelante en la sección 5.2.3.

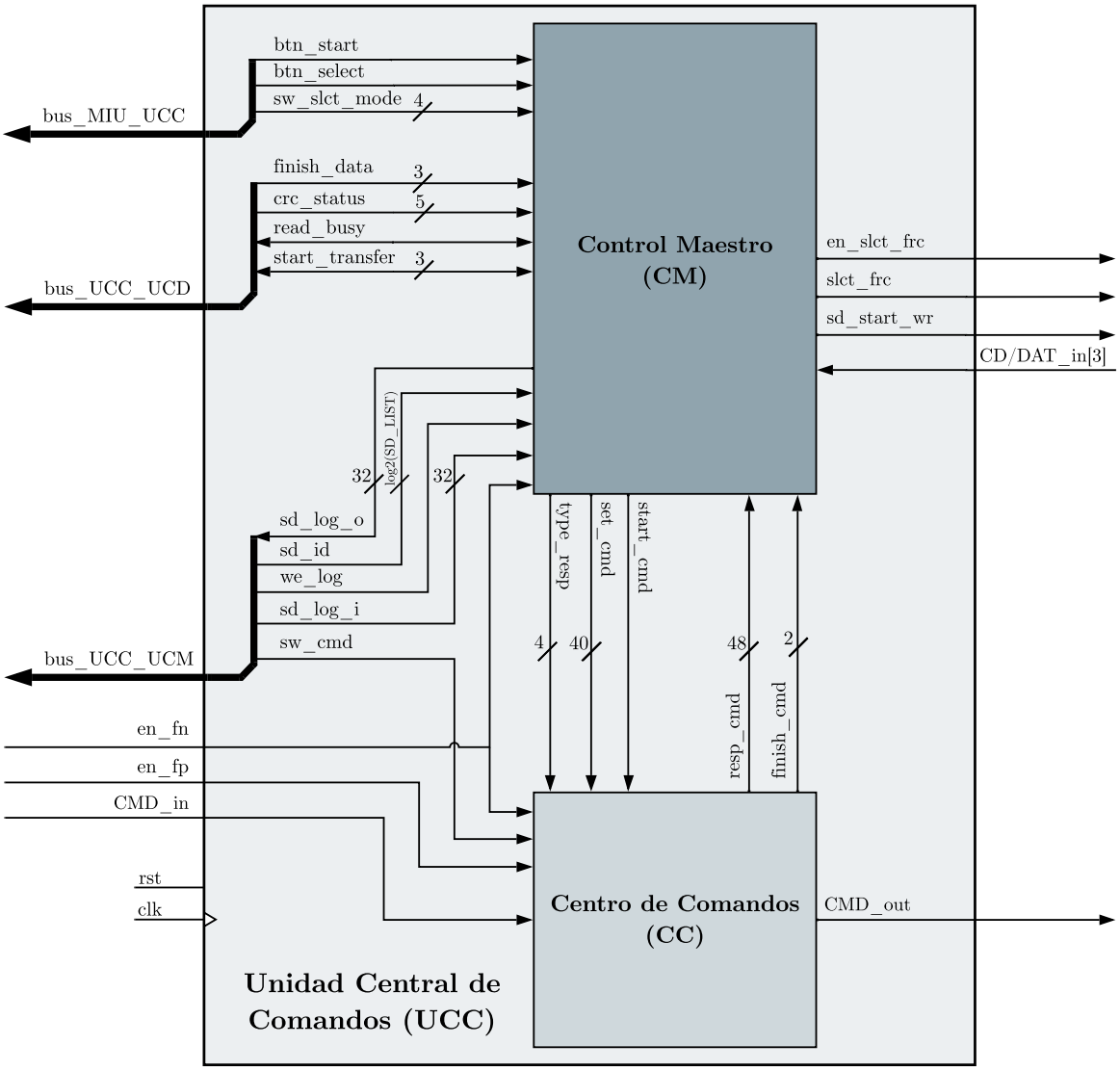


Figura 5.17: Diagrama de Bloques de la Unidad Central de Datos (UCC)

Centro de Comandos (CC)

El Centro de Comandos (CC) es el responsable de llevar a cabo el proceso de envío de comandos y obtención de resultados. Esto incluye la generación y comprobación del CRC7 del comando y respuesta respectivamente. El CC está compuesto por tres módulos: el Control-CMD, CRC7-CMD y Lector-CMD, como se observa en el diagrama de bloques de la figura 5.18.

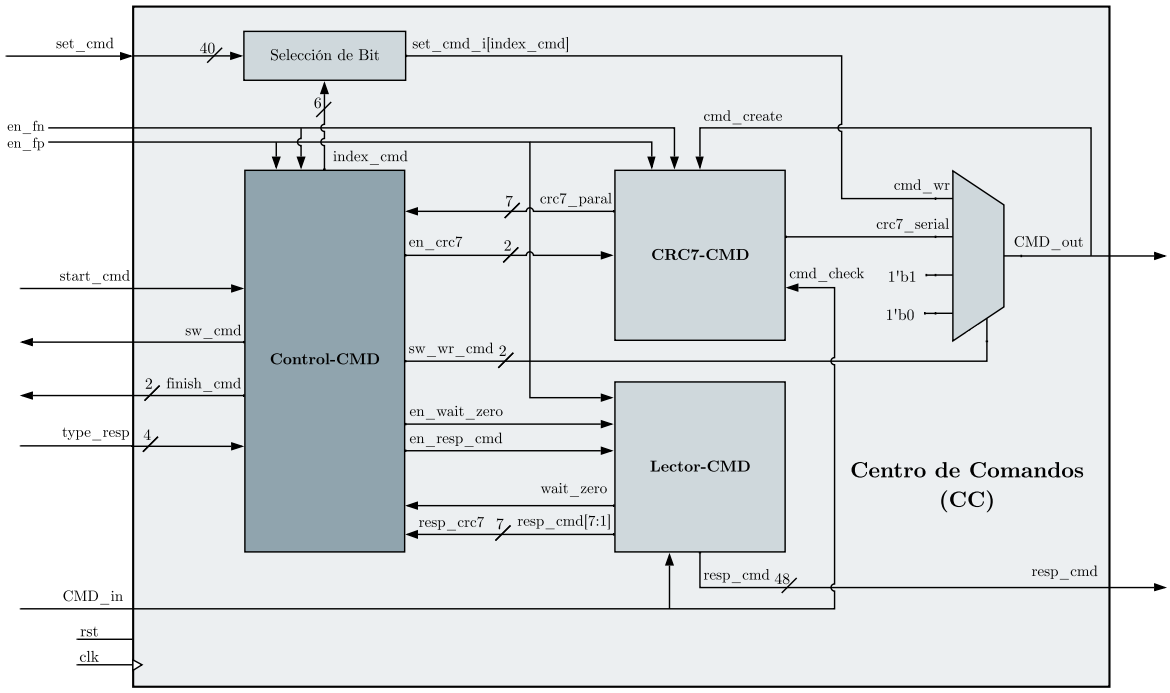


Figura 5.18: Diagrama de Bloques del Control de Comandos (CC)

El CM envía tres señales de relevancia para una correcta transmisión y recepción de comandos. La señal `start_cmd`, que da inicio a la transmisión del comando, y la señal `type_resp`, la cual define el parámetro de respuesta asociado al comando. El CC utiliza esta señal en el Control-CMD para identificar si el comando tiene respuesta o conocer cuál respuesta no presenta CRC7. En la tabla 5.7 se muestra el valor y definición de cada campo de esta señal. Y por último la señal `set_cmd` que corresponde a los primeros 40 bits MSB del comando a transmitir.

Tabla 5.6: Valores y Definición de la Señal `type_resp`

Campo	Valor	Definición
[4:1]	3'bxxx	Establece el tipo de Respuesta.
[0]	0	Comando sin Respuesta.
	1	Comando con Respuesta.

El CC adicional a los tres módulos contiene la lógica denominada Selección de Bit, el cual selecciona el bit correspondiente de `set_cmd` mediante `index_cmd` para transmitir la señal de forma serial a la tarjeta microSD. Además, cuenta con un mux controlado por el Control-CMD mediante la señal `sw_wr_cmd`, el mux selecciona las diferentes secciones del comando a transmitir.

Lector-CMD

El Lector-CMD lee y captura la respuesta de la línea `CMD_in`. Su funcionamiento está basado en el diagrama de flujo de la figura 5.19. Presenta dos fases: la primera fase espera el inicio de una transmisión de respuesta, la cual se da cuando la línea `CMD_in` cambia su estado a bajo. Iniciada la transmisión, entra al segundo estado, en el cual captura la respuesta por parte de la tarjeta. El Lector-CMD captura el valor recibido por medio de un registro de 48 bits con un desplazamiento de 1 bit. Al finalizar la transmisión, el CM maestro lee la respuesta para su interpretación correspondiente y el Control-CMD lee el CRC7 contenido en el registro de captura para su verificación.

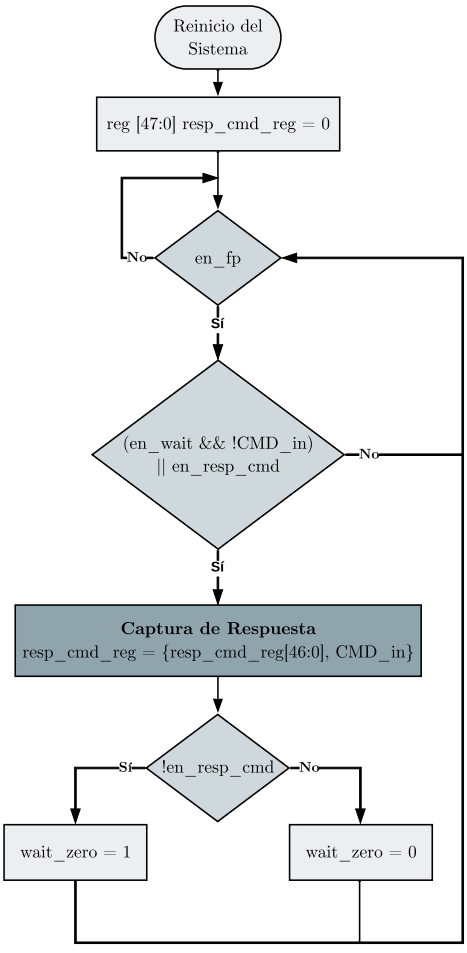


Figura 5.19: Diagrama de Flujo del Módulo Lector-CMD

CRC7-CMD

El módulo CRC7-CMD genera el código de protección del comando a transmitir. De la misma forma, crea un nuevo código de protección a partir de los bits de respuesta para ser cotejados con el código generado por la tarjeta y así verificar la integridad del token de respuesta. El cálculo del código de protección se realiza mediante 7 flip-flops en cascada y operadores lógicos `XOR` como se especifica en el Capítulo 3.4.11. El módulo CRC7-CMD se muestra en la figura 5.20 y funciona mediante dos modos de operación: generador y comprobador de CRC7.

El modo generador crea el CRC7 del comando a transmitir, la señal `cmd_create` se habilita a la entrada, la cual corresponde a la línea `CMD_out`. El módulo captura los bits en el mismo instante del muestreo de la tarjeta, es decir, en el flanco ascendente de `sck`. Finalizada la transmisión de los bits de `set_cmd` se transmite serialmente el código generado a través del mux de CC. La línea de entrada en el momento de la transmisión del CRC7 se deshabilita para no alterar el resultado de CRC7 durante la transmisión.

El modo comprobar genera el CRC7 de la respuesta, la señal `cmd_check` se habilita a la entrada, la cual corresponde a la línea `CMD_in`. El módulo captura los bits en el flanco descendente de `sck`. Obtenida la respuesta sin considerar el CRC7 y el bit de fin, el Control-CMD lee los 7 bits del código CRC7 de forma paralela para ser cotejados por el código CRC7 enviado por la respuesta.

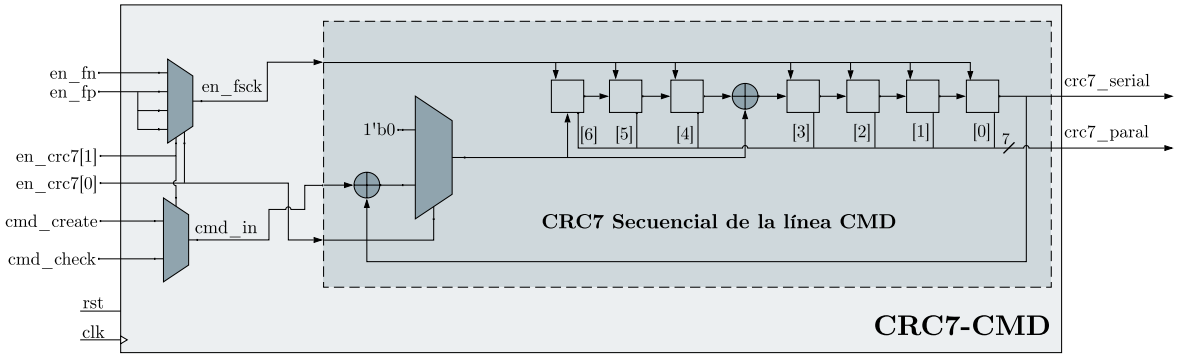


Figura 5.20: Diagrama de Bloques del Módulo CRC7-CMD

Control-CMD

El módulo Control-CMD controla la operación de envío de comando y recepción de respuesta mediante un FSM de Mealy. En la figura 5.21 se muestra el diagrama de estados que describe el funcionamiento del Control-CMD y en la tabla 5.8 se describen los estados del FSM.

Tabla 5.8: Descripción de Estados del FSM de Control-CMD

Estado	Descripción
IDLE	▪ Espera la señal de activación por parte de CM.
SYNCING	▪ Sincroniza el FSM con el reloj sck para asegurar que los bits del comando son desplazados en el flanco descendente y muestreados correctamente por la tarjeta en el flanco ascendente. ▪ Habilita el modo de salida en la conexión de la tarjeta seleccionada en UCM.
SEND_CMD	▪ Decrementa el valor de `index_cmd` para transmitir el siguiente bit de `set_cmd`.
SEND_CRC7	▪ Transmite el código de redundancia cíclica de 7 bits generado a partir de los bits de `set_cmd`.
STOP_BIT	▪ Transmite el bit de fin. ▪ Habilita el modo de entrada en la conexión de la tarjeta seleccionada en UCM.
WAIT_RESP	▪ Espera el bit de inicio de la respuesta por parte de la tarjeta. De no haber respuesta, regresa al estado IDLE.
RECIEVE_RESP	▪ Recibe el contenido de la respuesta por parte de la tarjeta.
NCR_CYCLES	▪ Espera N_{CR} antes de poder enviar otro comando a la tarjeta. ▪ Coteja el CRC7 de la respuesta, si esta lo requiere.

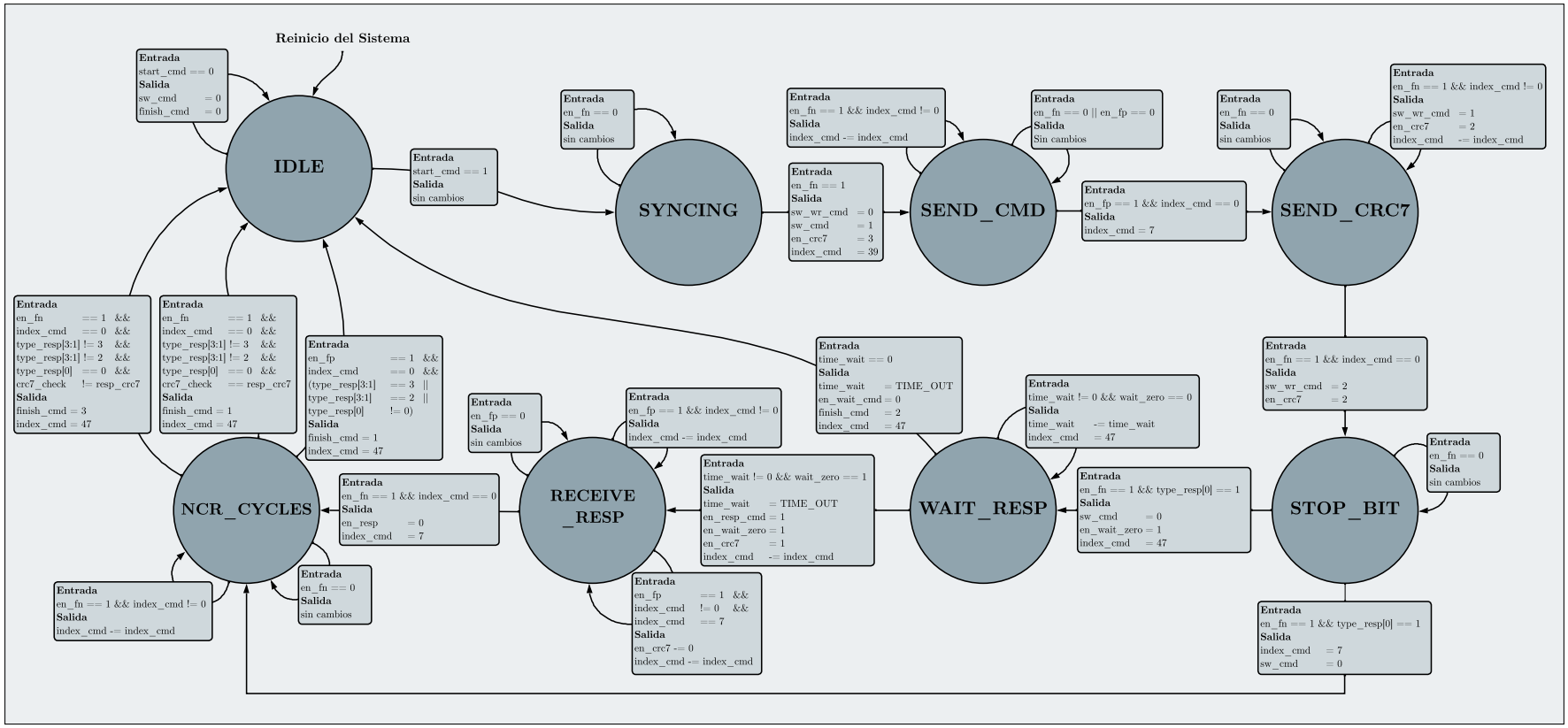


Figura 5.21: Diagrama de Estados de Control-CMD

Unidad Central de Datos (UCD)

La Unidad Central de Datos (`UCD`) recibe los datos externos y los procesa en un bloque de datos para su transmisión a la tarjeta microSD a través de la línea `DAT`. El `UCD` consta de dos componentes principales: el Control de Datos `CD` y un FIFO de Escritura, como se observa en el diagrama de bloques mostrado en la figura 5.22.

El FIFO de Escritura se diseñó haciendo uso del IP core "FIFO Generator" de Vivado, con un ancho de escritura de 1 byte y una profundidad de escritura de 512. Estos valores corresponden al tamaño de un dato (1 byte) y el bloque de datos (512 bytes) con los que se transmiten los datos en una tarjeta microSD. Los datos entrantes del FIFO provienen del exterior del sistema `SHM`, específicamente del Sistema Secundario acoplado a `SHM` durante la implementación del sistema completo. A su vez, es el Sistema Secundario el que proporciona la señal de `sd_fifo_wr_push` al FIFO de Escritura. Los datos del FIFO son extraídos por `CD` mediante el envío de un pulso en la señal `sd_fifo_wr_pop`.

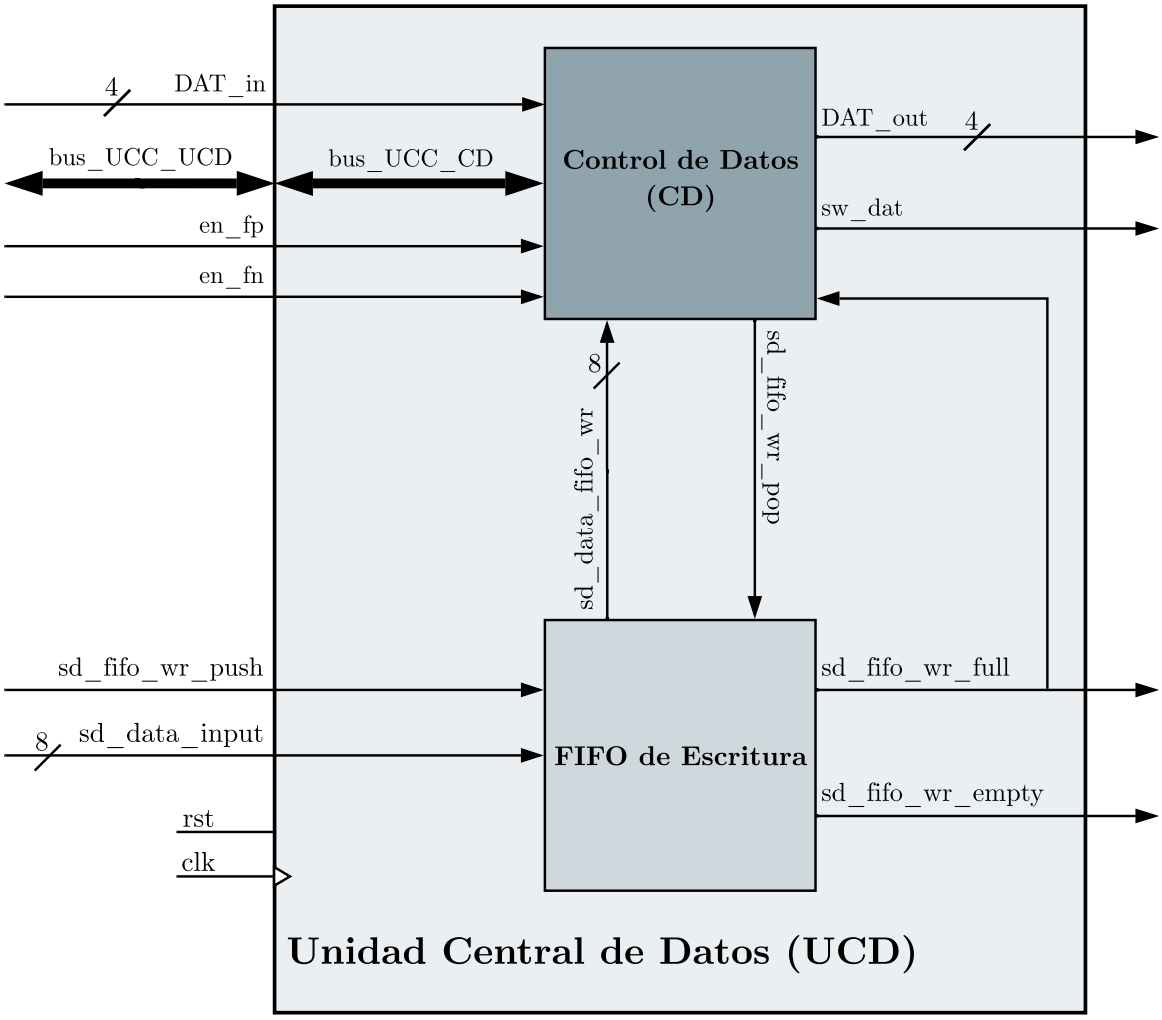


Figura 5.22: Diagrama de Bloques de la Unidad de Control de Datos (UCD)

Control de Datos (CD)

El Control de Datos (CD) transmite los datos provenientes del Sistema Secundario almacenados previamente por el FIFO de escritura mediante bloques de datos de 512 bytes en formato de modo 4-bits (ver Capítulo 3.4.4). En la figura 5.23 se observa el diagrama de bloques de CD, el cual está conformado por los módulos Control-DAT, Lector-DAT, CRC16-DAT.

La señal de entrada `sd_data_fifo_wr` corresponde a un dato de 1 byte proveniente del FIFO de Escritura. Mediante la lógica denominada Selección de Bit, se seleccionan los 4 bits MSB o los 4 bits LSB de la señal `sd_data_fifo_wr` mediante la señal `index` para ser distribuidos a las 4 líneas de `DAT_out` para su transmisión. Adicionalmente, CD cuenta con un mux controlado por el Control-DAT mediante la señal `sw_wr_dat` el mux selecciona las diferentes secciones del bloque de datos a transmitir.

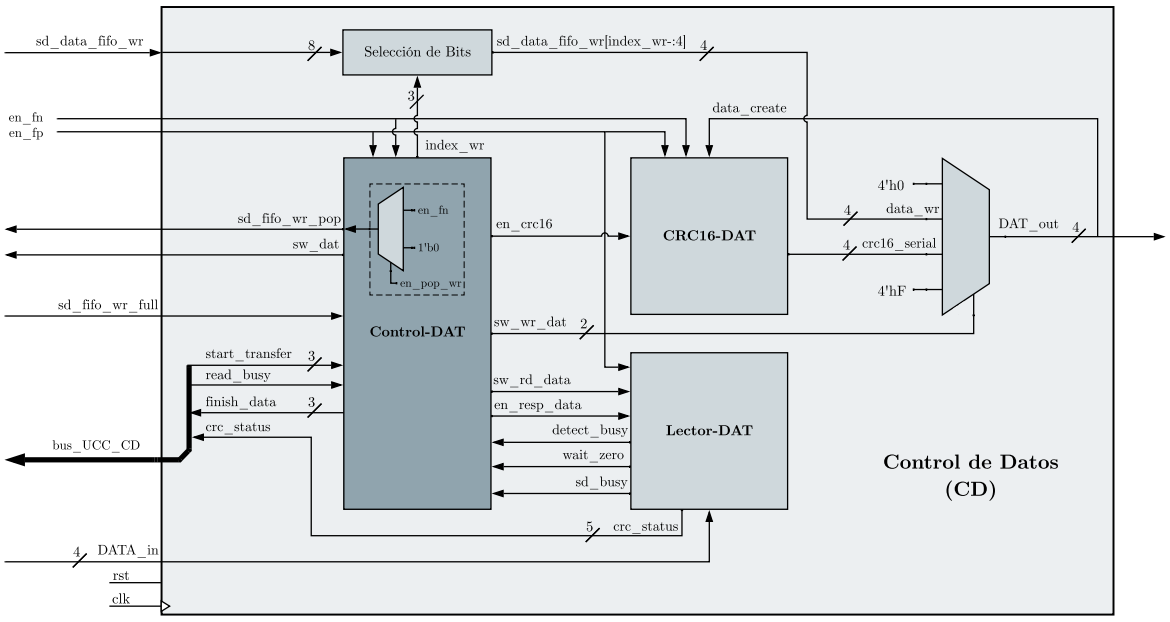


Figura 5.23: Diagrama de Bloques del Control de Datos (CD)

Lector-DAT

El módulo Lector-DAT lee la información proveniente de la tarjeta microSD a través de la línea `DAT_in`. Su funcionamiento se basa en el diagrama de flujo de la figura 5.24. El módulo es capaz de capturar cuatro distintos tipos de señales provenientes de la línea de datos:

- **Captura de datos**: el módulo interpreta los bits entrantes como datos de lectura, la captura se realiza mediante un registro de 8 bits con un desplazamiento de 4 bits hacia la izquierda (fuera del alcance de este documento).

- **Captura de CRC16**: el módulo interpreta los bits entrantes como CRC16 datos de lectura, la captura se realiza por medio de un registro de 16 bits con un desplazamiento de 1 bits hacia la izquierda (fuera del alcance de este documento).
- **Captura de CRC status**: el módulo interpreta los bits entrantes como bit CRC status como resultado de una escritura de bloque de datos. La captura se realiza haciendo uso de un registro de 5 bits con un desplazamiento de 1 bits hacia la izquierda. Este registro es leído por el módulo MC para su interpretación.
- **Leer bit de ocupado**: el módulo lee el bit de la línea `DAT_in[0]`, en caso de ser un '0' lógico, transmite una señal en alto en `sd_busy` y de ser un '1' lógico, transmite una señal en bajo en `sd_busy`.

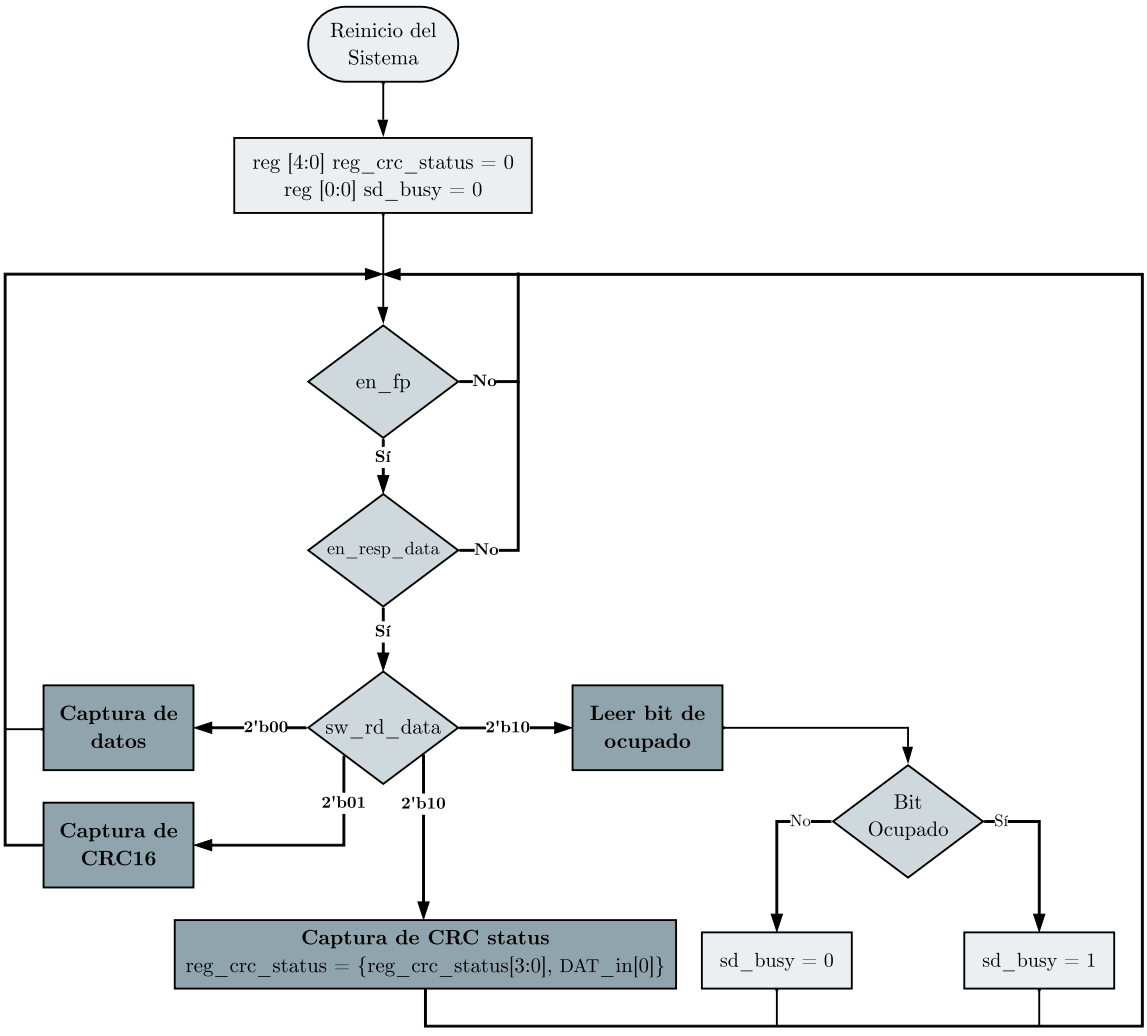


Figura 5.24: Diagrama de Flujo del Módulo Lector-DAT

CRC16-DAT

El módulo CRC16-DAT genera el código de protección del bloque de datos. Lo conforman cuatro módulos CRC16 donde mediante lógica secuencial calculan el código de protección de cada línea `DAT_out`. Un módulo CRC16 está conformado por 16 flip-flops en cascada y operadores lógicos `XOR` como se especifica en el Capítulo 3.4.11. El módulo presenta únicamente el modo generador. Durante este modo la señal `data_create` se habilita a la entrada, la cual corresponde a la línea `DAT_out`. Cada campo de bit es distribuido al módulo CRC16 correspondiente. La captura de bits del bloque de datos se realiza en el mismo instante del muestreo de la tarjeta, es decir, en el flanco ascendente de `sck`.

Al finalizar la transmisión del bloque de datos, se inicia la transmisión del CRC16 de forma serial a través del mux CD. Los cuatro módulos CRC16 desplazan un bit del código de protección a la salida respectiva, en donde se concatenan en la señal de 4 bits `crc16_serial`. Adicionalmente, la entrada se deshabilita para no alterar el resultado de CRC16 durante la transmisión.

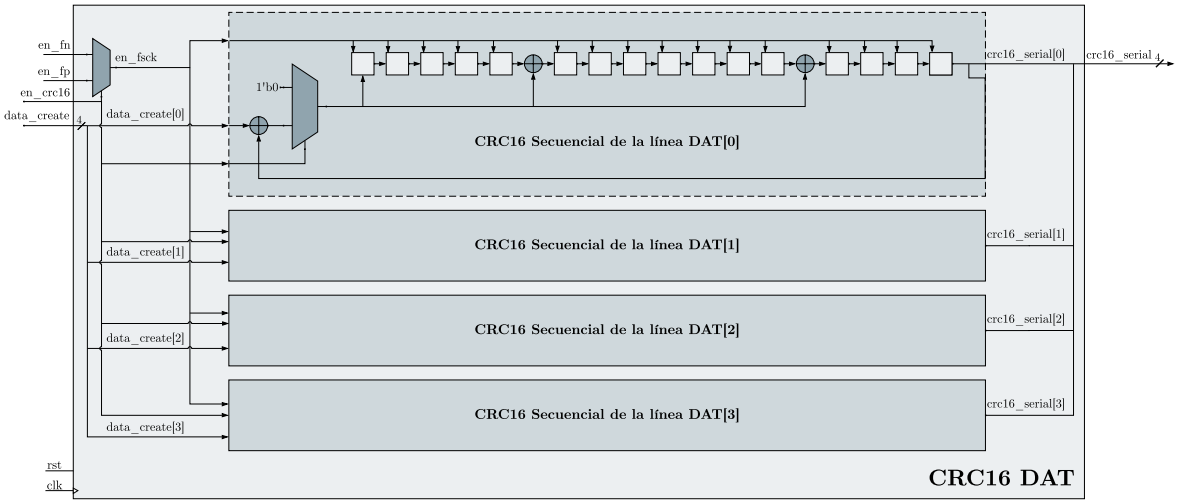


Figura 5.25: Diagrama de Bloques del Módulo CRC16

Control DAT

El módulo Control-DAT controla la operación de escritura de un bloque de datos de la tarjeta microSD mediante un FSM de Mealy. En la figura 5.27 se muestra el diagrama de estados que describe el funcionamiento del Control-DAT y en la tabla 5.9 se describen los estados del FSM.

Es importante mencionar que los bits `start_transfer[2:1]` son utilizados para determinar la operación (lectura o escritura) y el modo de operación SD (1-bit o 4-bit); sin embargo, tanto la operación de lectura como la transmisión en modo 1-bits salen del alcance de este documento, por lo tanto, estos bits siempre corresponden a un '10' lógico.

Tabla 5.9: Descripción de Estados del FSM de Control-DAT

Estado	Descripción
IDLE	▪ Espera la señal de activación por parte de CM mediante la señal `start_transfer[0]`. ▪ Inicia escritura de bloques (`read_busy` = 0)/Inicia la verificación de la señal de ocupado (`read_busy` = 1).
SYNCING	▪ Sincroniza el FSM con el reloj `sck` para asegurar que los bits del comando son desplazados en el flanco descendente y muestreados correctamente por la tarjeta en el flanco ascendente. ▪ Habilita el modo de salida en la conexión de la tarjeta seleccionada en UCM.
STAR_BIT	▪ Transmite el bit de inicio.
SEND_DATA	▪ Alterna el valor de `index` para transmitir los 4 bits MSB o los 4 bits LSB del dato a escribir. ▪ Solicita nuevo dato al FIFO de Escritura. ▪ El funcionamiento de este estado se basa en el diagrama de flujo de la figura 5.26.
WAIT_2CK	▪ Espera dos ciclos del reloj `sck`.
CRC_STATUS	▪ Recibe el CRC status por parte de la tarjeta.
SD_BUSY	▪ Detecta la señal de ocupado de la tarjeta.
WAIT_1CK	▪ Espera un ciclo del reloj `sck`.

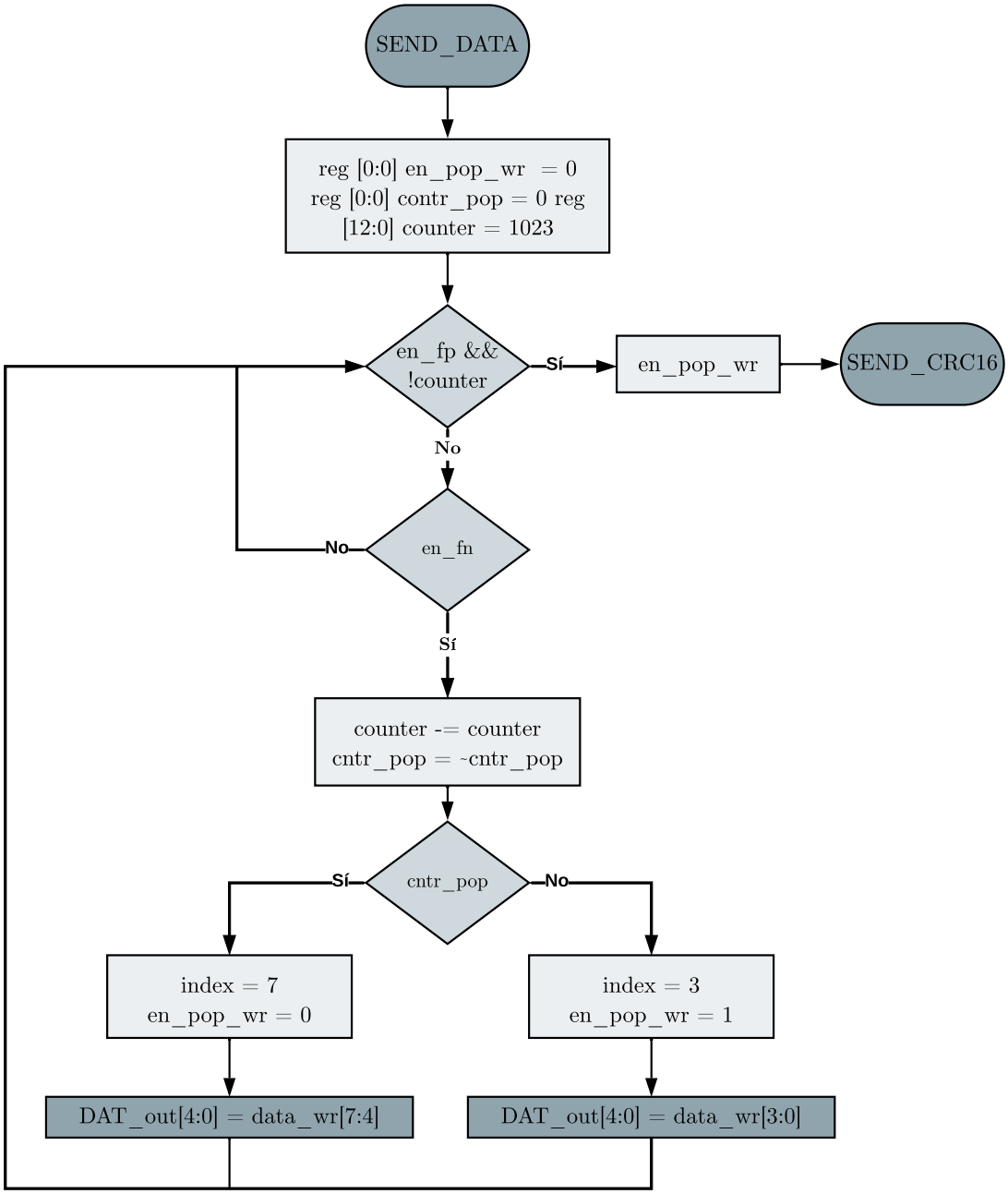


Figura 5.26: Diagrama de flujo del estado SEND_DATA

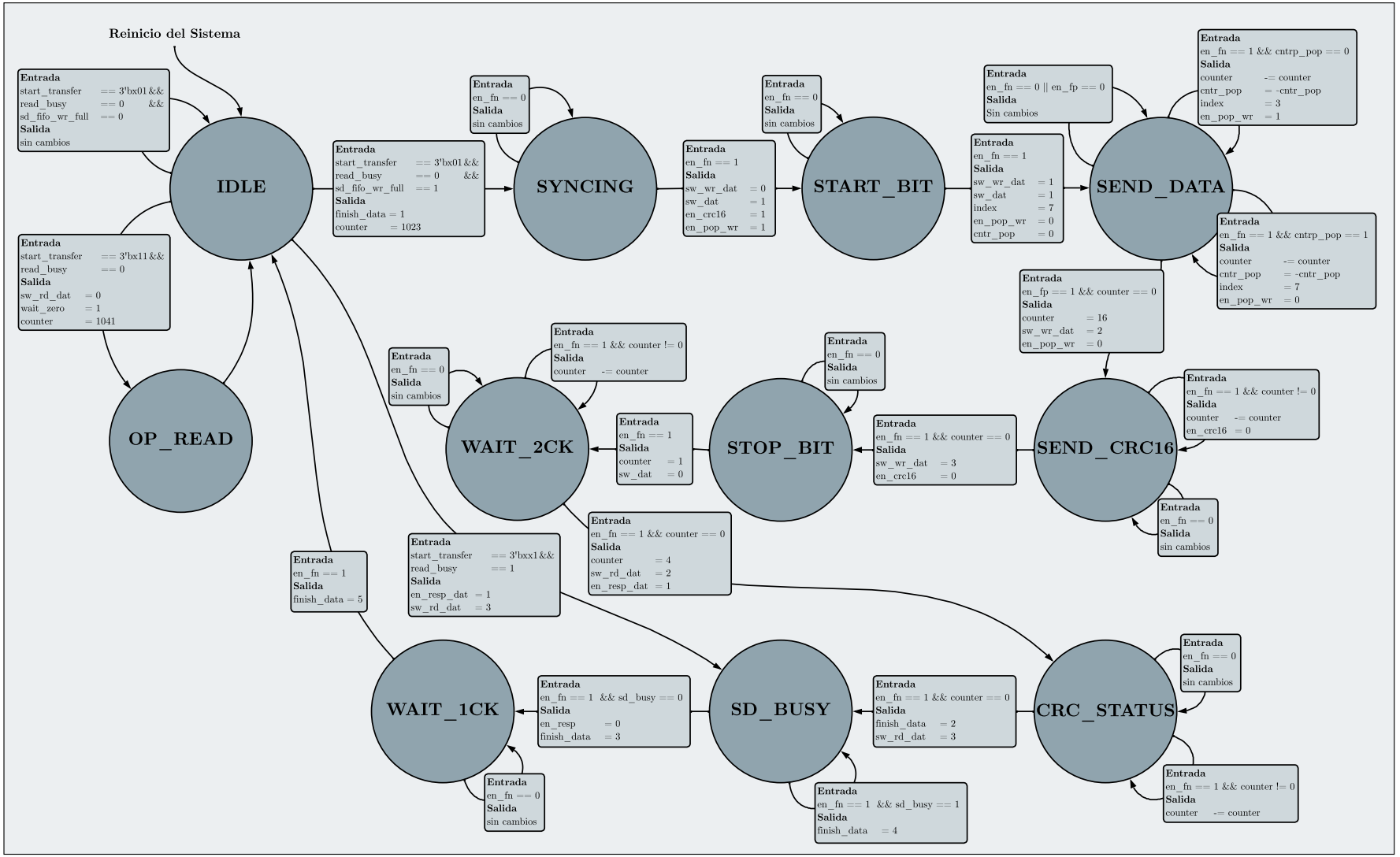


Figura 5.27: Diagrama de Estados de Control DAT

Control Maestro (CM)

El Control Maestro (CM) corresponde al módulo principal del SHM. Controla CC y CD y toma cualquier decisión ante el comportamiento de la tarjeta. Su principal función es la generación de comandos específicos para realizar diversas operaciones en una tarjeta microSD. Transmite los 40 bits MSB del comando al CC mediante la señal `set_cmd` para su transmisión y se encarga de interpretar la respuesta de la tarjeta para determinar la acción a realizar.

La siguiente tabla contiene los comandos con su valor hexadecimal utilizados por CM. Los comandos con una longitud menor a 40 bits, son comandos cuyo argumento depende de alguna respuesta de la tarjeta; por tanto, el argumento es dinámico y se establece durante algún estado del FSM.

Tabla 5.10: Comandos Utilizados por CM

CMD	Valor hexadecimal
CMD0	40'h40_00_00_00_00
CMD2	40'h42_00_00_00_00
CMD3	40'h43_00_00_00_00
CMD7	8'h47
CMD8	40'h48_00_00_01_AA
CMD12	40'h4C_00_00_00_00
CMD25	40'h59_00_00_00_01
CMD55	8'h77
CMD55_ACMD41	40'h77_00_00_00_00
ACMD6	36'h46_00_00_00_0
INQUIRY_CMD41	40'h69_50_00_00_00
FIRT_ACMD41	16'h69_40
ACMD42	36'h6A_00_00_00_0

Su función secundaria es controlar y administrar los procesos de escritura y lectura (fuera del alcance de este documento). El CM inicia el proceso de escritura de un bloque de datos e interpreta las respuestas enviadas por CD para realizar una determinada acción. Adicionalmente le indica al Sistema Secundario cuando SHM está listo para recibir los datos externos.

El CM está constituido por un FSM de Mealy, pero dado la extensión del mismo, es posible dividirlo en dos secciones: Inicialización de Tarjetas y Selección y Escritura de Múltiples Bloques. Los cuales realizan los estados de operación especificados en los Capítulos 3.4.14 y 3.4.15 respectivamente. El FSM está compuesto principalmente de estados "SEND_", los

cuales son estados de envío de un comando establecido e interpretación de respuesta ante ese comando. Previo a un estado "SEND_", el FSM establece los parámetros del comando a transmitir mediante la configuración de las señales que se muestran en la tabla 5.11.

Tabla 5.11: Configuración de Señales Previas a un Estado "SEND_"

Señal	Valor (bits)	Descripción
`start_cmd`	1'b1	Activa el Centro de Comando.
`type_resp`	4{1'bx}	Establece los parámetros de respuesta.
`set_cmd`	40{1'bx}	Contiene el comando a transmitir (sin CRC7 y bit de fin).

Durante el estado "SEND_", en el primer ciclo del reloj del sistema, la señal `start_cmd` se sitúa en bajo. Con el fin de evitar reenviar el comando una vez finalizado CC. Por otra parte, el FSM hace uso de la Matriz de Registros de Tarjetas microSD para leer y escribir los parámetros de la tarjeta seleccionada (ver sección 5.2.3). Para evitar redundancias en las secciones subsiguientes, el documento empleará el término 'archivar' para referirse a la acción de escribir en el registro de la tarjeta seleccionada en la Matriz de Registros, y 'consultar' para denotar la lectura del mismo registro.

Identificación de Tarjetas microSD

En la sección Identificación de las Tarjetas, el FSM accede a cada conexión de tarjeta microSD definida por el parámetro `SD_LIST` y ejecuta el Estado de Operación: Identificación de Tarjeta, establecido en el Capítulo 3.4.14. La figura 5.28 muestra el diagrama de estados de esta sección y en la tabla 5.12 se describen los estados del FSM correspondiente a esta sección.

Tabla 5.12: Descripción de Estados del FSM del Control Maestro - Identificación de Tarjetas

Estado	Descripción
LIST_SD	▪ Selecciona la conexión para iniciar la identificación respectiva de la tarjeta. ▪ Se traslada a SEARCH_CARD una vez finalizó la selección.
CARD_ DETECT	▪ Detecta si la conexión tiene una tarjeta. • Las tarjetas no detectadas son archivadas como `ina` y como tarjeta no detecta; el estado regresa a LIST_SD.
DELAY	▪ Espera 100 ciclos del reloj `sck`.
SEND_CMD0	▪ Restablece la tarjeta transmitiendo el comando `CMD0`. No realiza la captura de respuesta.

Tabla 5.12: Descripción de Estados del FSM del Control Maestro - Identificación de Tarjetas

SEND_CMD8	▪ Comprueba la tensión admitida por la tarjeta mediante la transmisión del comando `CMD8`. ▪ Detecta si la tarjeta es compatible con el voltaje del host. • Las tarjetas no compatibles son archivadas como `ina`; el estado regresa a LIST_SD.
SEND_ CMD55_41	▪ Transmite el comando `CMD55` con un RCA predeterminado. ▪ Se traslada a SEND_FIRT_ACMD41 o SEND_INQUIRY_ CMD41 según se requiera.
SEND_ INQUIRY_ CMD41	▪ Inicia la primera etapa de inicialización. Envía el comando `INQUIRY_CMD41` con un OCR de 0. ▪ Obtiene el OCR enviado por la tarjeta y lo utiliza como argumento para el comando `FIRT_ACMD41`.
SEND_FIRT_ ACMD41	▪ Inicia la segunda etapa de inicialización. Envía el comando `FIRT_ACMD41` con el OCR obtenido en SEND_CMD55_41 y los siguientes valores como argumento: • `HCS`: establecido en 1, el host tiene soporte únicamente para tarjetas microSD SDHC y/o SDXC. • `XPC`: establecido en 0, permite trabajar con 100 mA/3.6V. • `S18R`: establecido en 0, no se solicita un cambio de voltaje. ▪ Detecta el estado de ocupado de la respuesta del comando `FIRT_ACMD41`. • Bit en bajo: reenvía nuevamente el comando `FIRT_ACMD41` manteniendo los valores del argumento. Se traslada a SEND_CMD55_41 para enviar previamente el comando `CMD55`. ◦ Si transcurrió un segundo sin obtener un bit en alto en la respuesta, archiva la tarjeta como `ina`; el estado regresa a LIST_SD. • Bit en alto: comprueba que los valores del argumento de la respuesta sean compatibles con el host.
SEND_CMD2	▪ Solicita el registro CID mediante la transmisión del comando `CMD2`. Lee, pero no captura la respuesta.
SEND_CMD3	▪ Solicita el RCA mediante la transmisión del comando `CMD3`. ▪ Obtiene el RCA de la respuesta y lo archiva. ▪ Archiva la tarjeta como `act`.

Selección y Escritura de Múltiples Bloques

En la sección Selección y Escritura de Múltiples Bloques, el usuario selecciona la tarjeta microSD y la operación de transferencia a realizar. En este documento solo está disponible la operación de escritura de múltiples bloques. Los estados de esta sección del FSM ejecuta

el Estado de Operación: Transferencia de Datos, establecido en el Capítulo 3.4.15. El diagrama de bloques de esta sección se muestra en la figura 5.29 y en la tabla 5.13 se describen los estados del FSM correspondiente a esta sección.

Tabla 5.13: Descripción de Estados del FSM del Control Maestro - Selección y Escritura de Múltiples Bloques

Estado	Descripción
SEARCH_ CARD	▪ El usuario selecciona y confirma la tarjeta mediante la señal `btn_select` y `btn_select` respectivamente. • El estado consulta y verifica si la tarjeta seleccionada esta archivada como `act`. ◦ Obtiene los modos indicados por el usuario mediante las señales de `sw_slct_mode`. ◦ Guarda el registro de la tarjeta para su uso posterior. ◦ Establece `slc_mode[0]` en alto. • Una tarjeta seleccionada en estado `ina` no es accesible y el estado se mantiene.
SEND_CMD7	▪ Mediante `slc_mode[0]` determina la ruta: • `slc_mode[0]` = 1: el comando utiliza el RCA archivado para transferir la tarjeta del estado `stby` al estado `tran`. • `slc_mode[0]` = 0: el comando utiliza el RCA predeterminado para transferir la tarjeta del estado `tran` al estado `stby`.
SEND_CMD55	▪ Transmite el comando `CMD55` con el RCA de la tarjeta seleccionada. ▪ Se traslada a un estado de envío de comando de aplicación específica.
SEND_ACMD6	▪ Establece el modo 4-bits mediante la transmisión del comando `ACMD6`.
SEND_ACMD42	▪ Deshabilita la resistencia pull-up en la línea DAT[3] mediante la transmisión del comando `ACMD42`.
OPERATION_ STATE	▪ Determina la operación a realizar, definida por el usuario mediante la señal `sw_slct_mode`.
SEND_ CMD25	▪ Inicia la escritura de múltiples bloques mediante la transmisión del comando `CMD25`. ▪ Habilita la entrada de datos del ADC en el sistema Secundario. ▪ Inicia la transmisión de bloques mediante la señal `start_transfer`.

Tabla 5.13: Descripción de Estados del FSM del Control Maestro - Selección y Escritura de Múltiples Bloques

MULTIWRITE_ STATE	▪ Mantiene una transferencia de bloques de datos continua. • Monitorea el `crc_status` (no toma ninguna acción) • Controla la cantidad de bloques datos transmitidos. ▪ Detiene la operación de escritura al recibir un pulso en la señal `btn_start` o alcanzado el límite de sectores.
SEND_CMD12	▪ Detiene la transmisión de datos mediante la transmisión del comando `CMD12`. • Solicita mediante `read_busy` a CD verificar la señal de ocupado de la tarjeta.
MULTIWRITE_ CLOSE	▪ Espera que la tarjeta esté en estado de desocupado. ▪ Finaliza la operación de escritura. • Establece `slc_mode[0]` en bajo.

5.2.4. Funcionamiento general del sistema

El Sistema comienza una vez presionado el botón de inicio. Inmediatamente, intenta identificar individualmente todas las tarjetas de cada conexión establecida. Mediante la señal `sd_id` selecciona la tarjeta e inicia el proceso de identificación descrito anteriormente. Las tarjetas que no cumplen los requisitos establecidos por el host o no llegaron a responder son desechadas y clasificadas como `ina`. Aquellas tarjetas que logran pasar la etapa de identificación son clasificadas como `act`.

Una vez identificadas o descartadas todas las tarjetas microSD, el usuario puede elegir la operación de transferencia mediante los switchs de la FPGA. El SHM ofrece la capacidad de definir la cantidad de switches a través del parámetro `WIDTH_SW`. Para los propósitos de este documento, dicho parámetro se ha configurado con un valor de 3. Cada switch determina un modo específico que define cómo se llevará a cabo la operación. El CM está diseñado para soportar un número indefinido de modos. La tabla 5.14 proporciona una descripción del modo correspondiente a cada switch. Dado que el documento está enfocado en la escritura de múltiples bloques mediante el modo de operación SD de 4-bits. La señal `sw_slct_mode` está fijada con el valor 3'b110.

Tabla 5.14: Modo Correspondiente a cada Switch

Swith	Modo	Valor (bits)	Descripción
`sw_slct_mode[0]`	Operación	1'b0	Modo de escritura.
		1'b1	Modo de lectura.
`sw_slct_mode[1]`	Bloque	1'b0	Un bloque.
		1'b1	Múltiples Bloques.
`sw_slct_mode[2]`	Operación SD	1'b0	1-bit.
		1'b1	4-bits.

El usuario puede seleccionar la tarjeta mediante el botón de selección. Cada pulso recibido por medio de este botón incrementa la señal `sd_id`. Alcanzado el valor de `SD_LIST`-1, `sd_id` se restablece a 0. Una vez seleccionada la tarjeta, el usuario presiona nuevamente el botón de inicio, dando lugar así al inicio de la operación de escritura (solo es válido si la tarjeta se encuentra en estado `act`).

El CM realiza la operación establecida en los switches, en este caso la escritura multibloque en modo SD de 4-bits. Completados los preparativos, el CM permite la recepción de datos externos del ADC por medio del Sistema Secundario. Una vez lleno el FIFO de Escritura, se trasmiten los datos mediante un bloque de datos, repitiendo así el proceso. El CM monitorea y controla el proceso de transferencia. Un valor en 2 en la señal `finish_data` le permite obtener el `crc_status` del bloque de datos recién transferido y verificar si la transferencia fue exitosa. Para efectos de este documento, el host no adopta ninguna medida ante un error de `crc_status` y solo se limita a monitorear y archivar el estado. Esto se debe a que el propósito del documento es escribir en tiempo real los datos provenientes de un ADC. Por consiguiente, aunque se pierda un dato, el host no puede reenviar la información; esto se da por dos razones. La primera, una vez transferidos los datos, el host no retiene más la información transferida y la segunda, reenviar nuevamente los datos, repercutiría en la posible pérdida de más datos, dado que se detendría el proceso para reenviar nuevamente el dato no transmitido.

El CM utiliza, además, un registro como contador denominado `cntr_sd_sector`, el cual se utiliza para registrar la cantidad de bloques de datos que se han escrito y así conocer el tamaño de los datos transferidos, esto como una medida de protección que evita exceder el tamaño de almacenamiento de la tarjeta microSD y por consiguiente, evitar la pérdida de datos. Mediante el parámetro `NUM_SEC` se define el número máximo de sectores a escribir. Para efectos de este documento, este parámetro está establecido en un valor de 14062500 sectores, lo que equivale a 7.2 GB de datos transferidos. Este valor es un aproximado de transferir 12 bits de datos del ADC a una velocidad de muestreo de 8 MHz en un período de 10 minutos. El proceso de transferencia de datos se detiene una vez que el usuario presiona nuevamente el botón de inicio o que el host detecte que ha alcanzado el almacenamiento de datos establecidos.

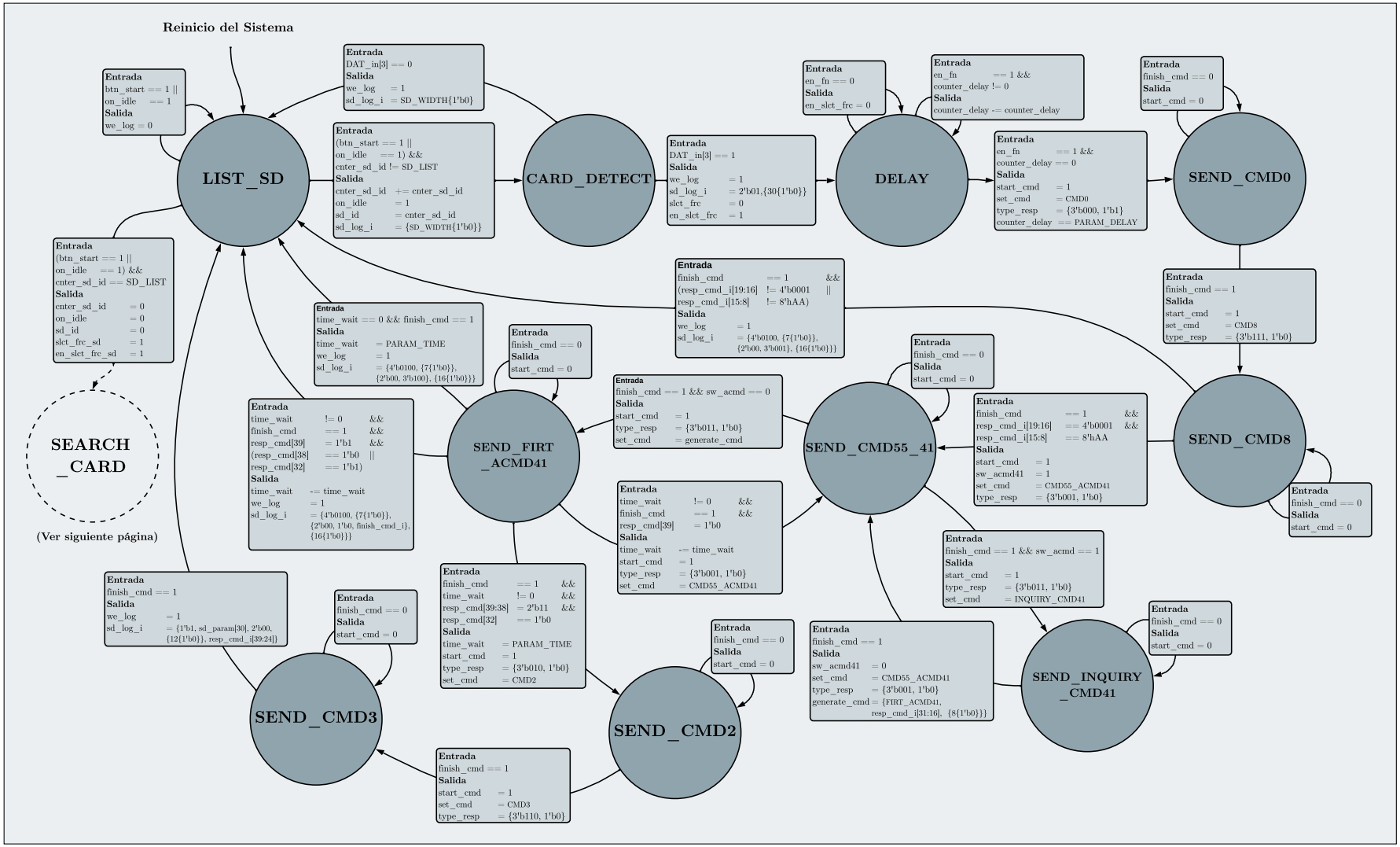


Figura 5.28: Diagrama de Estados de CM - Identificación de Tarjetas

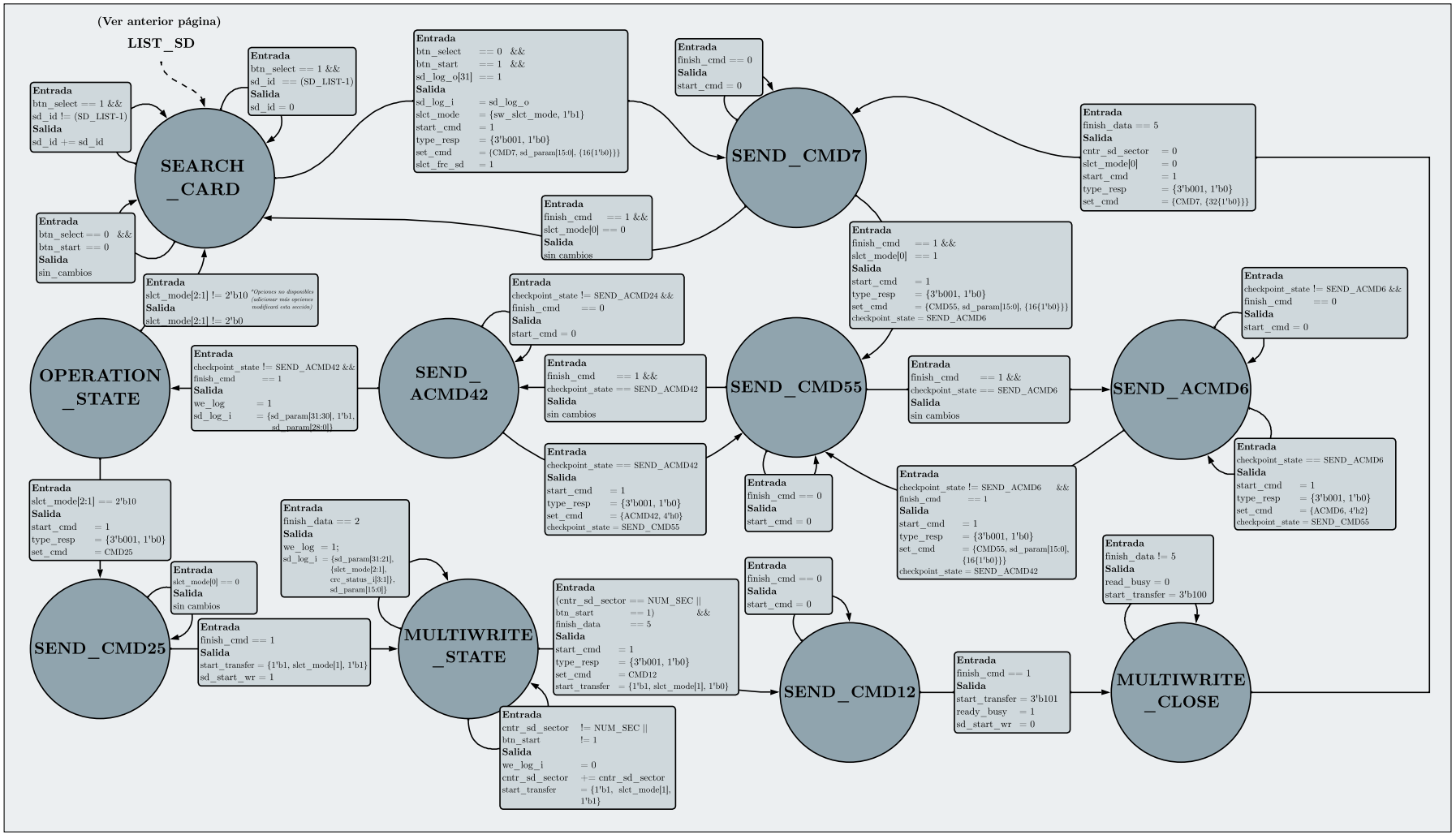


Figura 5.29: Diagrama de Estados de CM - Selección y Escritura de Múltiples Bloques

5.3. Análisis de resultados

5.3.1. General

En esta sección se muestran los resultados obtenidos de la implementación del "Sistema de Almacenamiento de Datos en Tiempo Real en Tarjetas microSD". El sistema se implementó en una FPGA Artix-7 en la plataforma de desarrollo Nexys 4 DDR, cargando los bitstreams del sistema a partir del diseño RTL realizado en el lenguaje de descripción de hardware SytemVerilog en el software Vivado de Xilinx®.

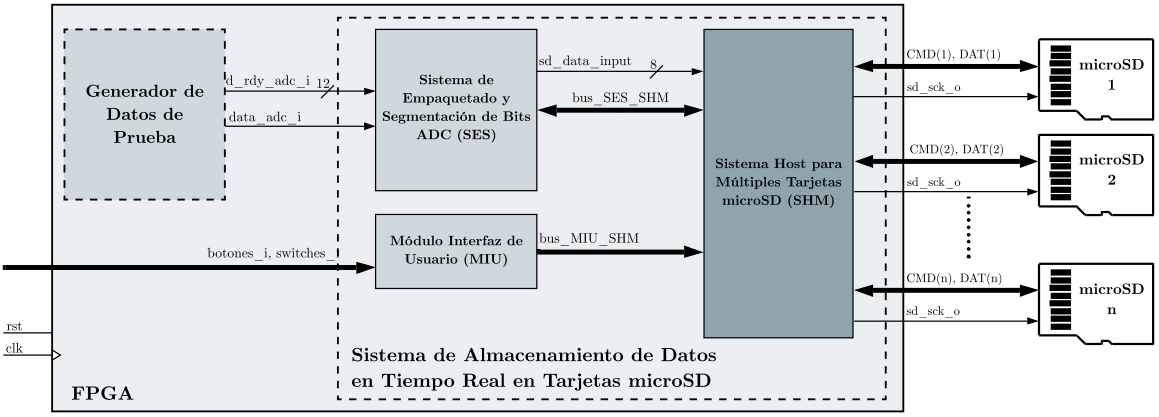


Figura 5.30: Diagrama de Bloques del Sistema de Almacenamiento de Datos en Tiempo Real en Tarjetas microSD con Módulo de Pruebas

Las pruebas se realizaron mediante el uso de un módulo de prueba, el cual simula la salida de un ADC con una resolución de 12 bits y una tasa de muestreo de 8 MHz. Dado que no se cuenta con un ADC de esas especificaciones. El módulo de prueba se observa en la figura 5.30 y consiste en un contador de datos a una frecuencia de 8 MHz (el mismo módulo envía un pulso a la entrada `d_rdy_adc_i` para indicar que hay un dato válido nuevo). Los datos son generados de manera secuencial. Cada dato de 12 bits consta de 3 valores hexadecimales que se describen mediante la función 5.1, donde 'x' representa el valor hexadecimal. El primer dato generado corresponde al valor 12'h123. De esta forma se puede verificar el correcto funcionamiento del sistema. Adicionalmente, los valores de los parámetros utilizados durante las pruebas se pueden visualizar en la tabla 5.15.

$$f(x) = (x + 3) mod 16 \tag{5.1}$$

Los resultados presentados se realizaron en la etapa de implementación mediante el uso del IP core "ILA" de Vivado. Por consiguiente, los resultados corresponden a una prueba física en la FPGA. Es de mencionar que cada unidad de tiempo en los diagramas de tiempo obtenidos por el ILA representan un valor de 10 ns. El reloj del sistema se estableció en 100 MHz en el Archivo de Restricciones, como se muestra a continuación:

Tabla 5.15: Parámetros Utilizados

Párametro	Valor (dec)
SD_LIST	2
WIDTH_SW	3
NUM_FIFOS	4
NUM_SEC	14062500

```
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0
    5} [get_ports clk_pi]
```

Se utilizan en total 9 periféricos de la FPGA para la realización de las pruebas: 3 switches para la definición de modos durante la operación de escritura, 2 botones para el inicio y selección de la tarjeta, 3 LEDs, las cuales indican: el finalizado Estado de Identificación, la tarjeta actualmente seleccionada y el estado de la actual tarjeta; el conector para microSD y 1 puerto Pmod para la conexión de un Pmod microSD. Adicionalmente, la tabla 5.16 detalla los recursos de la FPGA empleados. Se observa cómo el sistema utiliza una gran cantidad de BRAM, esto se debe principalmente al tamaño total del FIFO ADC, dado que los FIFO del IP core de vivado utilizan la BRAM para almacenar los datos. El porqué de la utilización de un FIFO que requiera casi la totalidad de la BRAM se detalla más adelante. La tabla 5.17 presenta los tiempos de mayor interés. Como se observa, los valores se mantienen positivos, por tanto, no hay violaciones de temporización; no obstante, se encuentra al límite en cuanto a la frecuencia de reloj. Adicionalmente, la Figura 5.31 muestra la potencia consumida por el sistema, el cual presenta una potencia mayor que su homólogo de 0.125 W dado al aumento de área utilizado para este sistema.

Tabla 5.16: Recursos de la FPGA utilizados

Recurso	Utilizado	Disponible	Utilizado (%)
LUT	1560	63400	2.46
FF	1333	126800	1.05
BRAM	128.50	135	95.19
IO	32	210	12.24
BUFG	1	32	3.13

Tabla 5.17: Tiempos de la Implementación del Sistema

Setup	Hold	Pulse Width
Wort Negative Slack (WNS): 0.418 ns	Worst Hold Slack (WHS): 0.072 ns	Worts Pulse Width Slack (WPWS): 4.500 ns

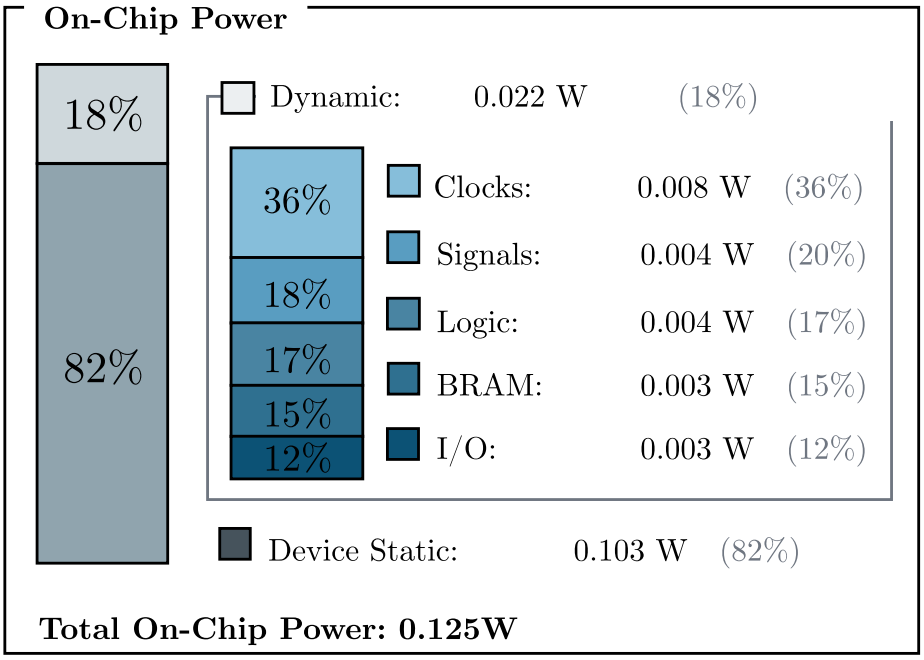


Figura 5.31: Total de Potencia Utilizada

Para las pruebas se utilizó una tarjeta microSD con las siguientes especificaciones:

Tabla 5.18: Especificaciones de la Tarjeta microSD

Fabricante	Modelo	Capacidad de Almacenamiento	Velocidad de Escritura Mínima	Velocidad de Escritura Máxima
Kingston®	CANVAS Select Plus	(256 GB)	V30	85 MB/s

5.3.2. Empaquetado y segmentación de datos

En la figura 5.32 se muestra el inicio de captura de datos del módulo de prueba, se observa cómo la señal `sd_start_wr` habilitó el acceso de la señal `d_rdy_adc_i` al sistema secundario. En consecuencia, se evidencia cómo el sistema comenzó a capturar los datos de prueba para su empaquetado, segmentación y almacenamiento en el FIFO ADC.

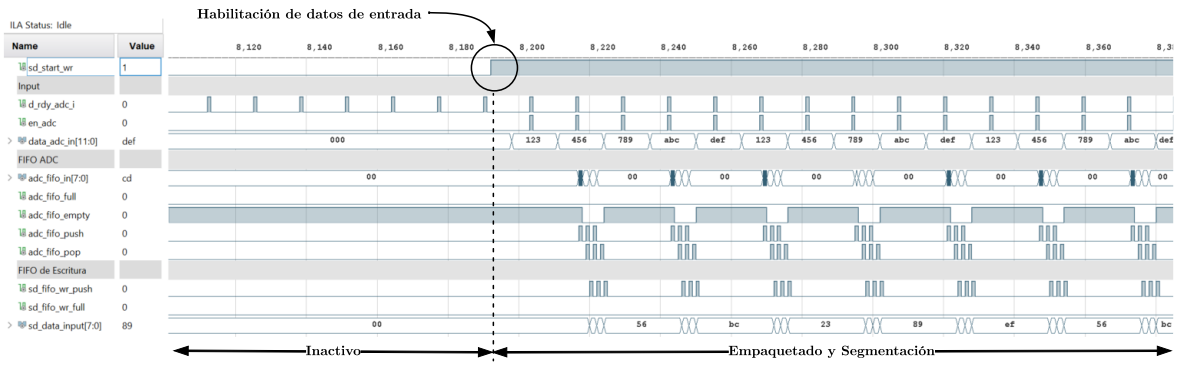


Figura 5.32: Diagrama de Tiempo de Inicio de Captura de Dato

El proceso de empaquetado y segmentación se muestra detalladamente en la figura 5.32. Se observa cómo el sistema secundario recibió y empaquetó dos datos provenientes de la señal `data_adc_in` y posteriormente envió los 24 bits de datos capturados en tres datos segmentados hacia el FIFO ADC. Adicionalmente, se logra observar el funcionamiento del Gestor de FIFO ADC. Inmediatamente después de ingresar un nuevo dato al FIFO ADC, el Gestor de FIFO ADC colocó el dato en la salida del FIFO y le indicó al FIFO de Escritura que capturara dicho dato. Este proceso se repitió por cada dato segmentado que se recibió.

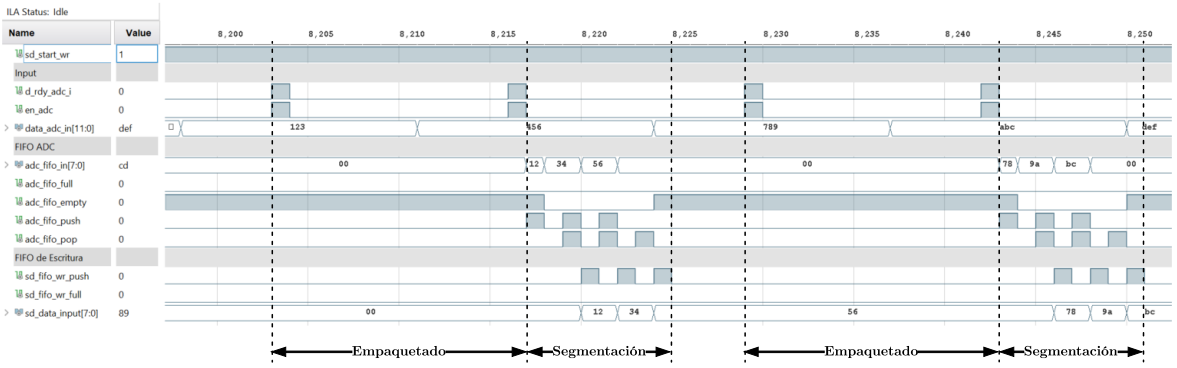


Figura 5.33: Diagrama de Tiempo del Empaquetado y Segmentación de Datos

La figura 5.34 muestra un proceso de llenado de los FIFOs internos del módulo FIFO ADC. Se logra visualizar cómo el FIFO 1 recibió constantemente 3 datos segmentados

hasta que llegó a su capacidad máxima, visualizado mediante una señal en alto de `full_1`. En ese punto se observa cómo el Gestor de entrada cambió su señal de '1' a '2', por lo tanto, redireccionó las señales de entrada del FIFO ADC al FIFO interno 2. Lo anterior se puede validar observando que la siguiente entrada de datos en el FIFO ADC se realiza en el FIFO 2. Asimismo, dado que solo se modificaron las señales de entrada, el FIFO 0 continúa enviando datos sin interrupciones.

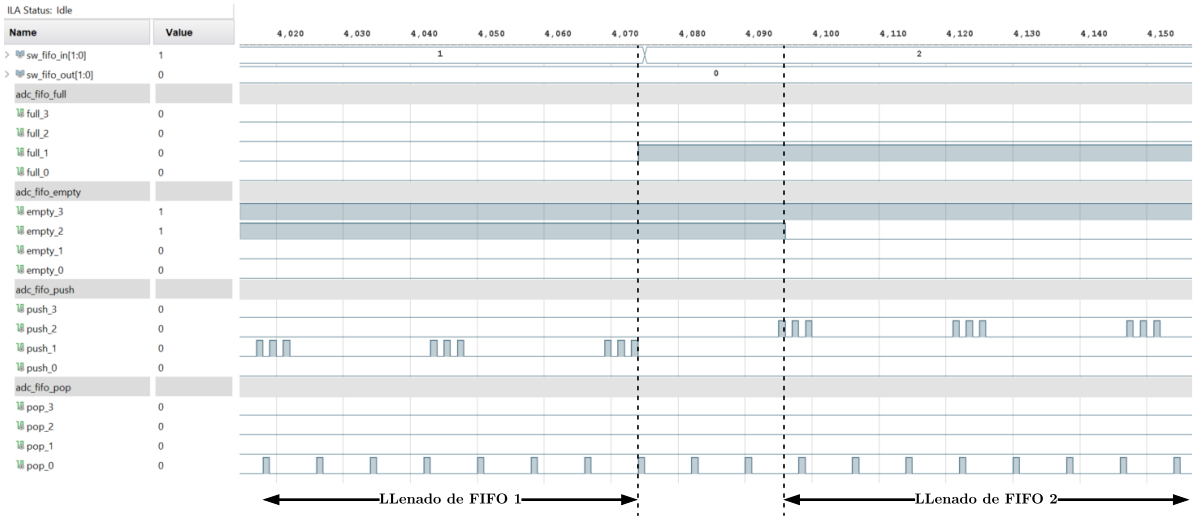


Figura 5.34: Diagrama de Tiempo del Llenado del FIFO interno del FIFO ADC

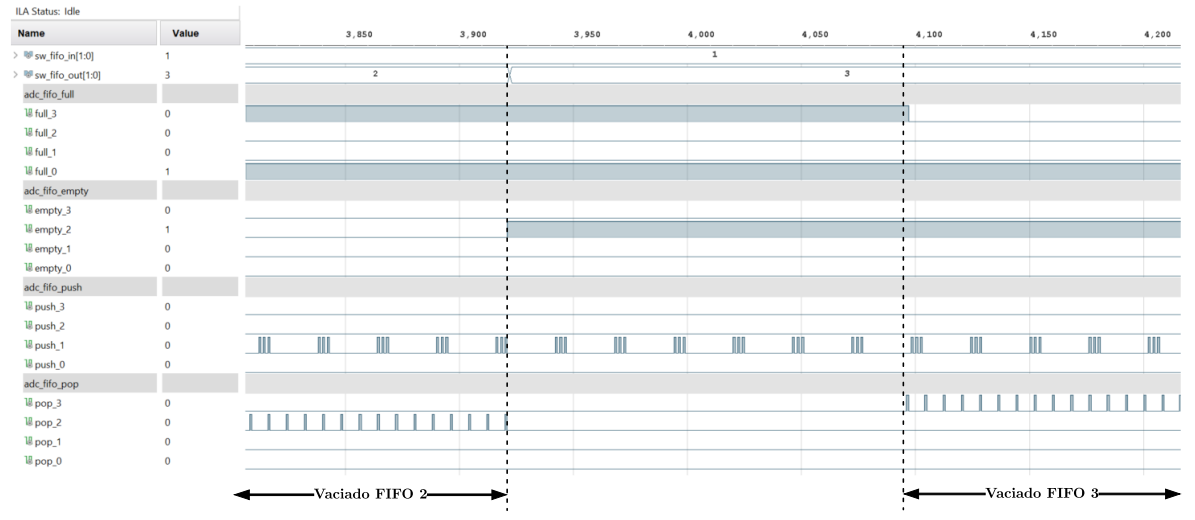


Figura 5.35: Diagrama de Tiempo del Vaciado del FIFO interno del FIFO ADC

La figura 5.35 muestra un proceso de vaciado de los FIFOs internos del módulo FIFO ADC. Se logra visualizar cómo el FIFO 2 envió constantemente los datos hacia el FIFO de escritura. El Gestor de FIFO ADC detectó que el FIFO de Escritura se mantiene

completamente lleno, por tanto, dejó de extraer datos del FIFO interno. Durante ese mismo instante, se puede observar cómo el Gestor de Salida detectó la señal de `empty2` en alto, por lo cual el FIFO 2 carece de datos para transmitir, por lo que redireccionó las salidas del módulo FFIO ADC al FIFO 3. Una vez que el Gestor del FIFO ADC detectó que es posible enviar datos nuevamente al FIFO de Escritura, los datos que son enviados pertenecen ahora al FIFO 3. Por otra parte, dado que solo se modificaron las señales de salida, el FIFO 1 continúa recibiendo datos sin interrupciones.

5.3.3. Comando y respuesta en los estados de operación de la tarjeta microSD

El host dio inicio al sistema seleccionando la tarjeta de la primera conexión y realizó una espera para estabilizar el voltaje de la tarjeta como se especifica en Capítulo 3.4.14. De forma consecutiva transmitió el comando `CMD0` restableciendo la tarjeta. Además, se puede apreciar cómo el host desplazó los bits del comando en el flanco descendente del reloj `sck` para que la tarjeta pudiera muestrearlo correctamente durante el flanco ascendente.

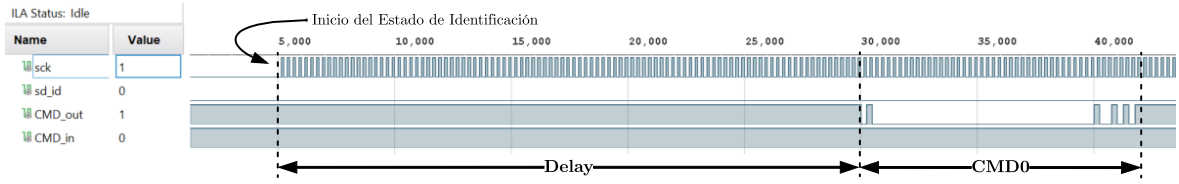


Figura 5.36: Diagrama de Tiempo del Comando CMD0

En la figura 5.37, se muestra el proceso que el host utilizó para comprobar la tensión admitida por la tarjeta. Se observa cómo el host inició la verificación enviando en la línea `CMD` un comando que corresponde al comando `CMD8`. Los argumentos del comando indican que el host opera con una alimentación de 2.7 V a 3.6 V y utilizó un valor de 8'hAA para el `Check Pattern`, como se especifica en el Capítulo 3.4.14.

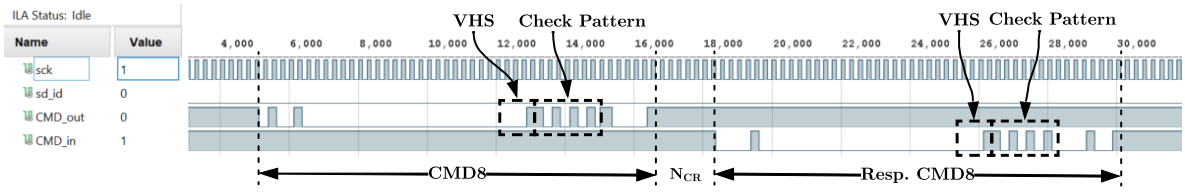
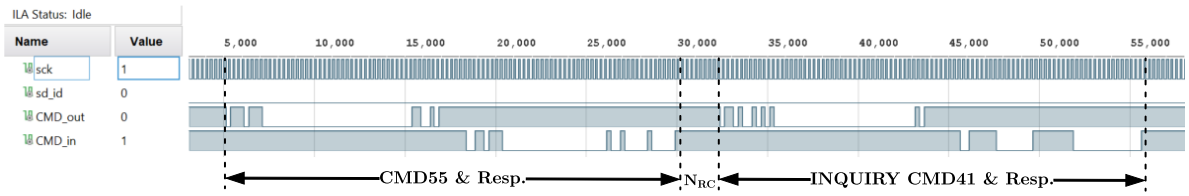


Figura 5.37: Diagrama de Tiempo del Comando CMD8

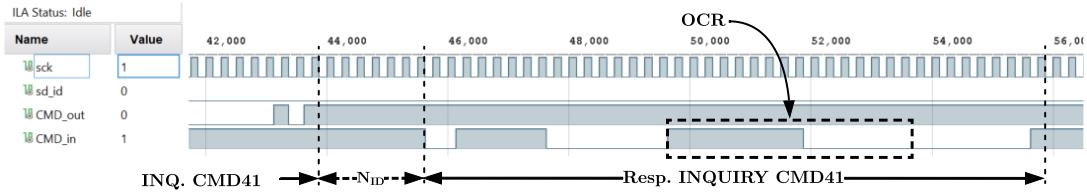
Una vez finalizado el envío del comando, se observa un tiempo N_{CR}, como se especifica en el Capítulo 3.4.12, y posteriormente se recibe una respuesta por parte de la tarjeta. El

envío de un token por parte de la tarjeta evidencia varios aspectos del SHM. Confirma que la estructura del comando enviado por el host es correcta, es decir, presenta un bit de inicio y de transmisión correctos (‘01’ lógico). Además, el índice del comando y los argumentos corresponden al del comando respectivo (`CMD8`). Adicionalmente, se verifica que el CRC7 generado por el módulo CRC7-CMD coincide con el código de protección de los bits MSB del comando. La obtención de una respuesta también indica que el comando se transmitió correctamente, por lo que no hubo errores durante la transmisión del mismo. Al analizar el contenido propio de la respuesta, se observa que el argumento hace eco de los argumentos del comando enviado, lo que significa que la tarjeta es compatible con el voltaje del host.

Posteriormente, el host dio inicio a la primera etapa de inicialización de la tarjeta, la cual se muestra en la figura 5.38. Para ello, transmitió el comando `INQUIRY CMD41`; sin embargo, dado que este es un comando de aplicación específica, previamente envió el comando `CMD55`. Como se puede observar, la tarjeta reconoció el comando `CMD55`, y el siguiente comando lo interpretó como un comando de aplicación especifica. El argumento de `INQUIRY CMD41`, tal como se específica en el Capítulo 3.4.14, concuerda con un OCR predeterminado. La respuesta por parte de la tarjeta a este comando correspondió al OCR que la tarjeta generó.



(a) Transacción del Comando INQUIRY CMD41

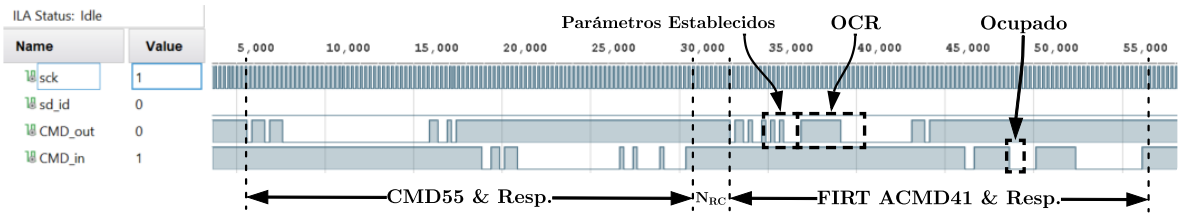


(b) Respuesta del Comando INQUIRY CMD41

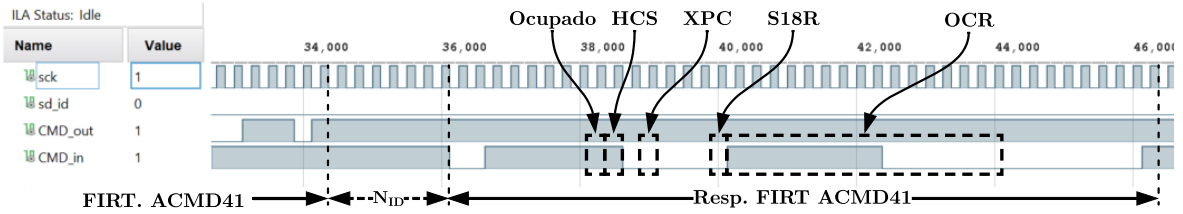
Figura 5.38: Diagrama de Tiempo del Comando INQUIRY CMD41

Para la segunda etapa, la figura 5.39 muestra las diversas fases que el host llevó a cabo. La primera fase transmitió el primer comando `FIRT ACMD41`, utilizando el OCR enviado previamente por la tarjeta y los parámetros establecidos con que el host opera (ver tabla 5.12). Sin embargo, la respuesta ante ese comando fue un bit de ocupado en bajo. Siguiendo las pautas establecidas en el Capítulo 3.4.14, el host retransmitió nuevamente el comando `FIRT ACMD41` (segunda fase) utilizando los mismos valores. Una vez más, la respuesta contenía un bit de ocupado en bajo.

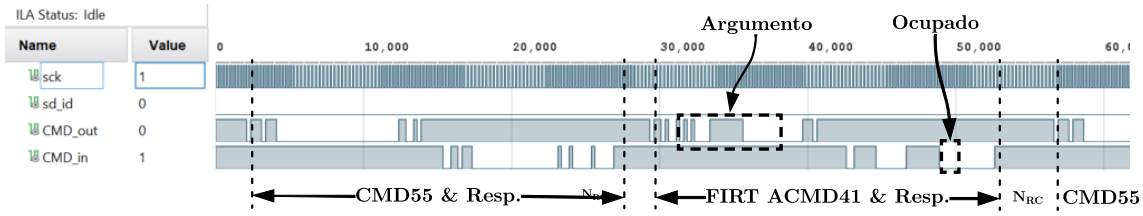
El proceso se repitió hasta que el host finalmente recibió un bit de ocupado en alto (tercera fase). Los argumentos de la respuesta indicaron que la tarjeta es una SDHC o SDXC, lo cual es correcto, ya que la tarjeta es una SDXC (ver tabla 5.18). Además, se observó que la tarjeta va a trabajar con 10 mA/3.6 V (`XPC` = 0' lógico) y no realizó cambios de voltaje (`S18R` = 0' lógico).



(a) Transacción Inicial del Comando FIRT ACMD41



(b) Transacción en Estado Ocupado del Comando FIRT ACMD41



(c) Respuesta del Comando FIRT ACMD41 en Estado Desocupado

Figura 5.39: Diagrama de Tiempo del Comando FIRT ACMD41

En la figura 5.40 se visualiza la solicitud del registro `CID` por parte del host. Se puede notar como la tarjeta reconoció el comando correctamente y transmite una respuesta de 128 bits, en donde 120 bits corresponden al registro `CID` solicitado por la tarjeta.

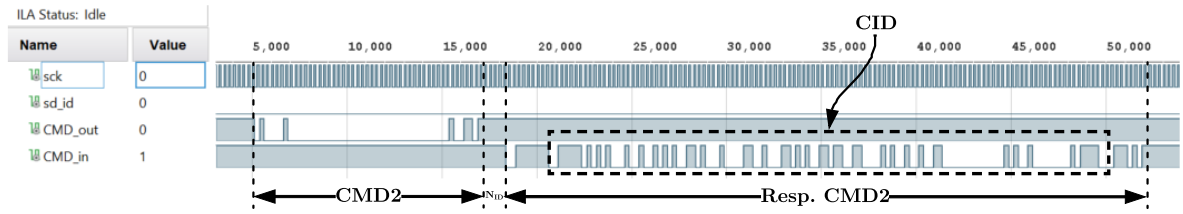
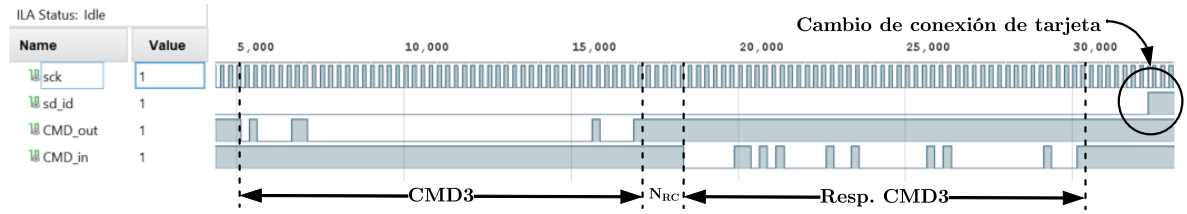
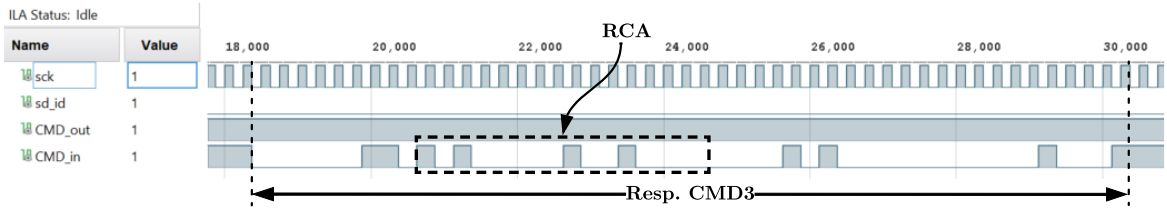


Figura 5.40: Diagrama de Tiempo del Comando CMD2

Consecutivamente, el host solicitó el `RCA` a la tarjeta mediante el comando `CMD3` como se observa en la figura 5.41. La tarjeta reconoció el comando correctamente, por lo cual generó y transmitió el registro RCA. Gracias al RCA, es posible acceder al modo de transferencia de datos de la tarjeta microSD. Además, en este punto, el host finalizó el estado de identificación de dicha tarjeta, asignándola como la tarjeta `act`. También se puede apreciar en la figura cómo el host cambió a la siguiente tarjeta para realizar el mismo proceso con la nueva tarjeta seleccionada.



(a) Transacción del Comando CMD3



(b) Respuesta del Comando CMD3

Figura 5.41: Diagrama de Tiempo del Comando CMD3

Si se observan detalladamente las figuras 5.41 y 5.42, se evidencia cómo el host utilizó el `RCA` obtenido en `CMD3` para ejecutar el comando `CMD7` y acceder al estado de transferencia de la tarjeta en la conexión 0.

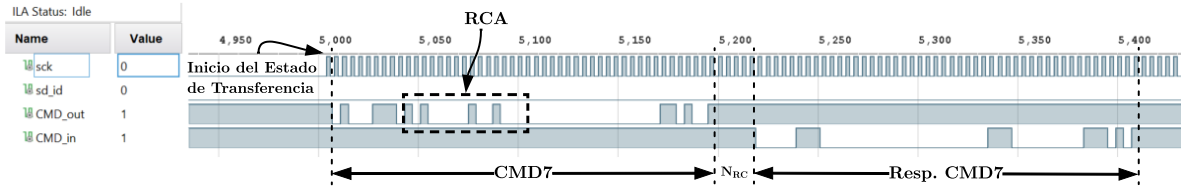


Figura 5.42: Diagrama de Tiempo del Comando CMD7

En la figura 5.43 se muestra cómo el host realizó el cambio a modo 4-bits de la tarjeta, para ello envió el comando `CMD55` y posteriormente transmitió el comando `ACMD6` adjuntando el valor correspondiente en el argumento para habilitar el modo 4-bits, como se menciona en el Capítulo 3.4.9. La respuesta al comando dio confirmación de que la tarjeta había aceptado el cambio.

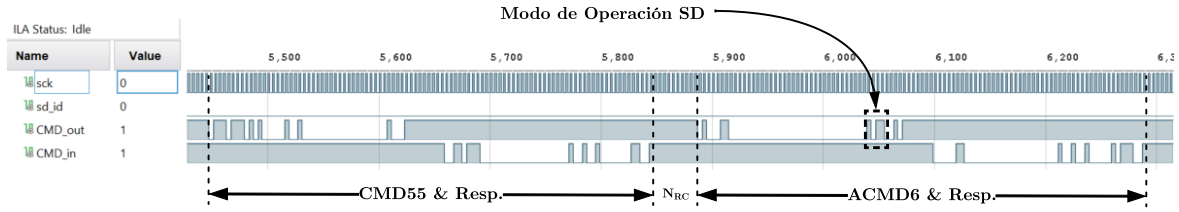


Figura 5.43: Diagrama de Tiempo del Comando ACMD6

Para poder transmitir los datos en las cuatro líneas `DAT`, el host deshabilitó la resistencia pull-up mediante `ACMD42` como se muestra en la figura 5.44 y previamente envió al comando `CMD55` dado que es un comando de aplicación específica.

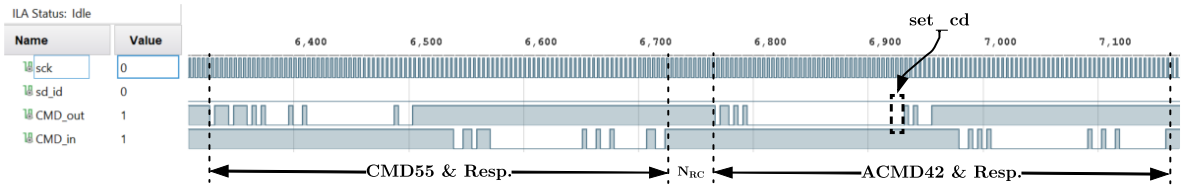


Figura 5.44: Diagrama de Tiempo del Comando ACMD42

Para dar inicio al proceso de escritura de datos en múltiples bloques, el host transmitió el comando `CMD25`, con el valor de '1' en el argumento, como se observa en la figura 5.43. Este valor le indica a la tarjeta microSD que debe iniciar la escritura de datos en el sector número uno. La respuesta ante este comando confirma que la tarjeta está lista para recibir datos.

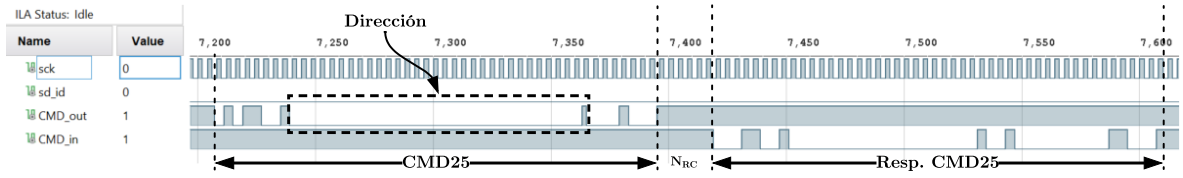


Figura 5.45: Diagrama de Tiempo del Comando CMD25

5.3.4. Transferencia de múltiples bloques de datos

En la figura 5.46, se muestra el inicio de la transferencia de datos después de haber enviado el comando `CMD25`. Se observa cómo la señal `sd_start_wr` se habilitó, permitiendo el ingreso de datos al FIFO de Escritura.

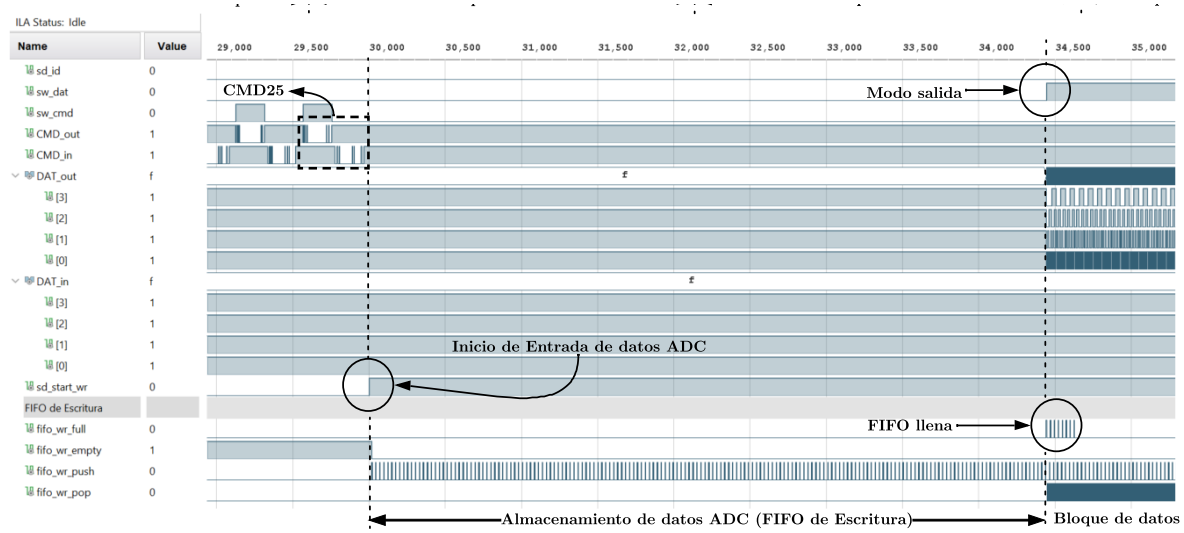


Figura 5.46: Diagrama de Tiempo del Inicio de Transferencia de Datos

Una vez que el FIFO alcanzó su capacidad máxima, se aprecia cómo el UCD habilitó el modo de salida en UCM. Esto permitió que los datos almacenados en el FIFO se transmitieran a la tarjeta microSD a través de la señal `CMD_out`, la cual se envía a la línea `DAT` de la conexión 0. Adicionalmente, se pueden visualizar diversos pulsos en la señal `fifo_wr_full`; este comportamiento se debe al diseño del sistema de almacenamiento de los datos en el FIFO de escritura, el cual se explicará en detalles más adelante.

El contenido del bloque de datos transferido se puede observar en la figura 5.47. Se llega a ver cómo cada bit del segmento de los datos de prueba fue enviado al mismo tiempo mediante las 4 líneas de `DAT`. Asimismo, se puede apreciar la integridad de la secuencia del dato de prueba, lo que demuestra una correcta transferencia de los datos externos.

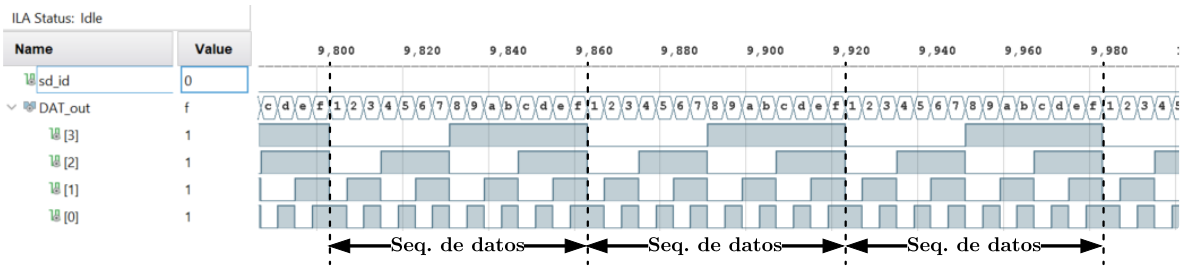


Figura 5.47: Diagrama de Tiempo del Contenido de un Bloque de Datos

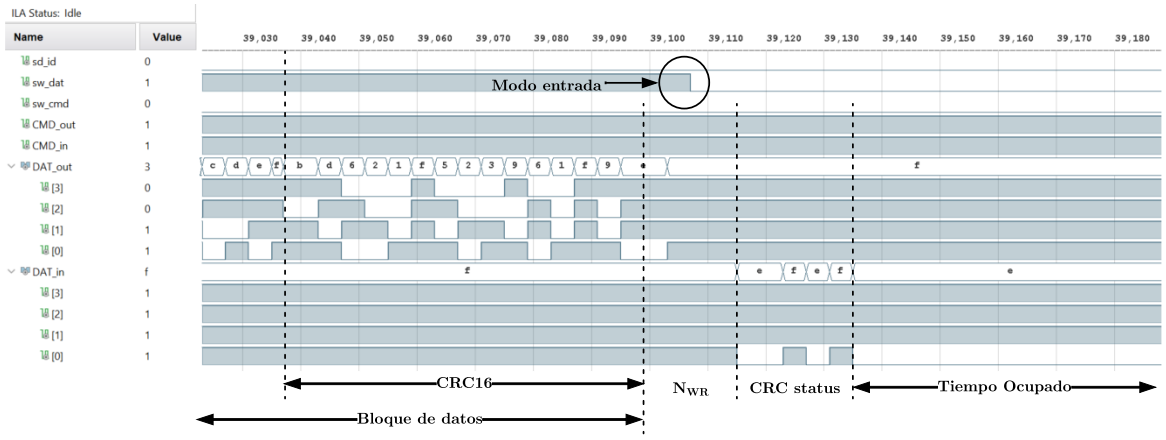


Figura 5.48: Diagrama de Tiempo de CRC16 y CRC status

Una vez finalizada la transferencia del bloque de datos, como se muestra en la figura 5.48, se logra observar cómo el sistema transmite el CRC16 generado mediante el módulo CRC16-DAT. Posteriormente, UCD habilitó el modo de entrada en UCM, lo que permitió el ingreso de `CRC status` por parte de la tarjeta transcurrido un tiempo N_{WR}. El valor de `CRC status` indica que no presentaron errores durante la transmisión, como se menciona en el Capítulo 3.4.11. Por lo tanto, en este punto los datos de prueba contenidos en el bloque de datos fueron almacenados por la tarjeta microSD de forma correcta y sin comprometer la integridad de los datos. Adicionalmente, se valida el funcionamiento correcto del CRC16 generado, puesto que la tarjeta da por buenos tanto los datos transmitidos como el código de protección generado a partir de dichos datos. Finalmente, en la figura se puede apreciar cómo la tarjeta, después de transmitir el `CRC status` entra al estado `prg`, por lo cual envía un bit de ocupado en la línea `DAT[0]`.

En la figura 5.49 se muestra cómo la tarjeta permaneció en estado de `prg`. Mientras tanto, el host monitoreaba el bit de ocupado, esperando una señal en alto. Una vez concluido el estado de `prg`, el host inicia una nueva transferencia de bloque de datos.

Se puede observar cómo el FIFO de Escritura se mantuvo completamente listo apenas finalizó la transmisión del último dato, esto le permitió iniciar una nueva transmisión de forma inmediata una vez que la tarjeta quedó desocupada. Dado que el CM tiene conocimiento de la cantidad de datos que transmite para no exceder los 512 bytes, se logra optimizar el sistema para que las únicas pausas que sufra sean causantes directas de la tarjeta microSD y no del sistema.

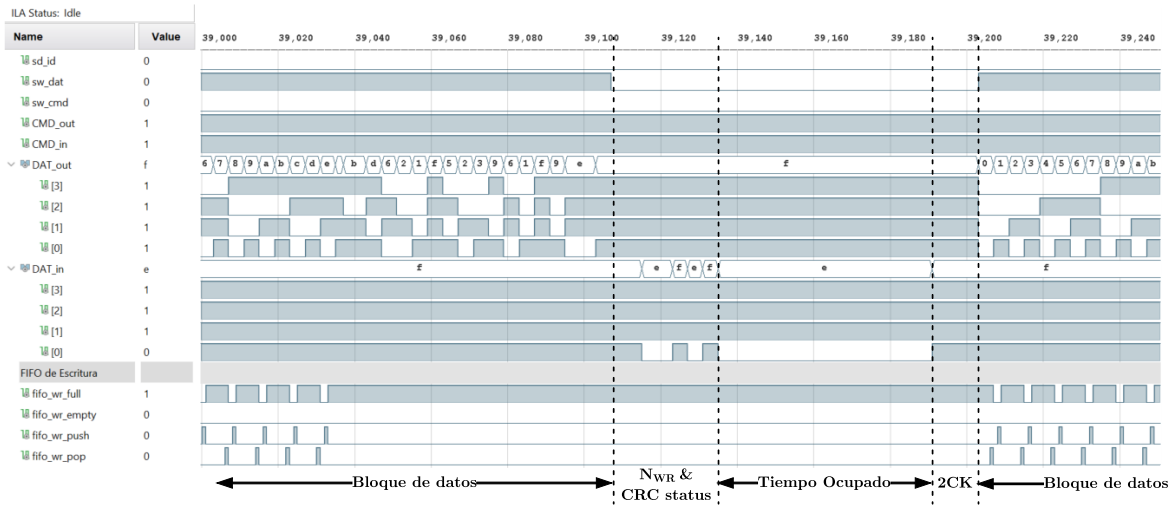


Figura 5.49: Diagrama de Tiempo del Estado de Ocupado de la Tarjeta SD

En la figura 5.50 se evidencia el proceso de envío de múltiples bloques de datos por parte del host. Se puede observar una transferencia continua por parte del sistema, únicamente deteniendo la transmisión por el estado de `prg` de la tarjeta.

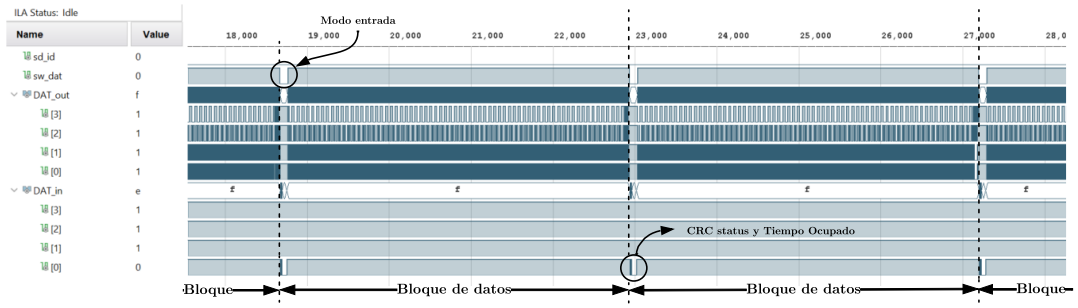


Figura 5.50: Diagrama de Tiempo de un Conjunto de Bloque de Datos

Se muestra que al finalizar un bloque de datos, el host cambió a modo de entrada para leer el `CRC status` y el bit de ocupado. Posteriormente, habilitó el modo de salida y continuó con el siguiente bloque de datos.

Hasta este punto del documento se ha observado que los tiempos de ocupado debido al estado `prg` presentan intervalos cortos. En general, los tiempos de ocupado entre bloques presentaron un aproximado de 560 ns. Sin embargo, durante las pruebas, se observó que el tiempo de ocupado transcurrido en un período de escritura de aproximadamente 3 segundos presentó tiempos de espera muy superiores a los anteriores mencionados. Alcanzando picos máximos registrados de hasta 25 ms. Por lo cual, el FIFO ADC debía tener la capacidad de almacenar todos los datos externos durante los picos máximos sin llegar a un 'overflow'. Considerando que al FIFO ADC le ingresan 3 datos segmentados

del ADC a una frecuencia de 3.84 MHz y un tiempo máximo de ocupado de 25 ms, se obtiene que el FIFO ADC debe poder almacenar durante ese período un mínimo de 288 462 datos. Dado que el IP core "FIFO Generator" tiene una profundidad máxima de 131 072 datos de 1 byte, es necesario un mínimo de 3 FIFOs internos en el módulo FIFO ADC para poder almacenar los datos sin sufrir 'overflow'.

Además, se debe considerar que una transacción de bloques de datos se completa cada 42.96 μs (tomando en cuenta el tiempo de ocupado de 560 ns entre bloques de datos). Por lo cual, por escritura de bloque de datos ingresan 497 nuevos datos segmentados provenientes del ADC, por consiguiente, la cantidad neta de datos que lograría vaciar un FIFO por escritura de bloque de datos es de 15 datos. Por tanto, se tiene que un FIFO ADC utilizando 3 FIFOs internos de 131 072 datos lograría vaciarse en un período de 845 ms. Lo anterior asegura que el FIFO se vaciará antes de que se genere un pico máximo en el tiempo de ocupado, los datos segmentados entrantes del ADC una vez vaciado el FIFO se trasladarán inmediatamente al FIFO de Escritura para su transferencia. Finalmente, se decide optar por 4 FIFOs internos de 131 072 en el módulo FIFO ADC, proporcionando un mayor margen para proteger los datos sobre posibles tiempos de ocupado mayores a los registros durante las mediciones. Dado que el documento "SD Specifications Part 1 Physical Layer Specification" no realiza mención alguna sobre los intervalos del tiempo de ocupado y que las pruebas se realizaron en una única tarjeta, cada tarjeta puede presentar variaciones en los períodos de ocupado. La utilización de 4 FIFOs internos asegura un almacenamiento de 524 288 datos segmentados del ADC y con un tiempo de vaciado de 1.13 s.

En la figura 5.51 se observa cómo una vez se alcanzó la cantidad máxima de bloques de datos transferidos, el host envió el comando `CMD12` una vez finalizó la transmisión del último bloque de datos. La tarjeta aceptó el comando correctamente e inició el proceso de finalización de la operación de escritura de múltiples bloques de datos. Se observa cómo durante el envío de la respuesta, la tarjeta habilita la señal de ocupado, haciendo indicación de que está en estado `prg`.

Se puede observar que el valor de la señal `cntr_sd_sector` corresponde a una transferencia de 14062500 bloques de datos, ya que el contador inicia en 0. Este valor de bloques de datos transferidos equivale a un total de 7.2 GB de datos de prueba. Lo que representa el almacenamiento de datos a una tarjeta microSD de una entrada de datos con una tasa de muestreo de 96 Mbps en un período de 10 minutos.

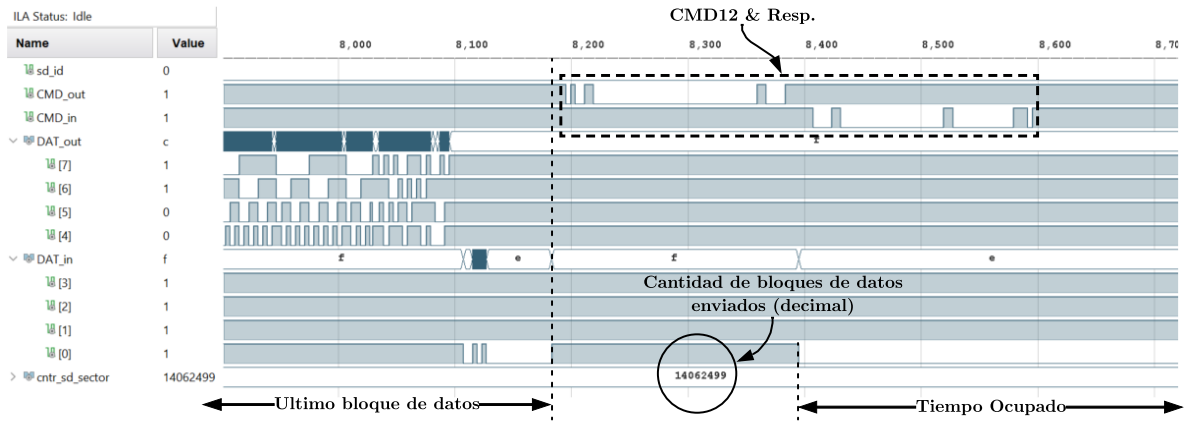


Figura 5.51: Diagrama de Tiempo de Fin de la Operación de Escritura de Múltiples Bloques con una Duración de 10 min

Una vez concluye la señal de ocupado por parte de la tarjeta, el host transmite el comando `CMD7` con un `RCA` predeterminado. Como se observa la tarjeta en la figura 5.52 no transmitió una respuesta al host como se especifica en el Capítulo 3.4.15. Posteriormente el host dio por finalizada la transferencia, poniéndose a la espera de una operación.

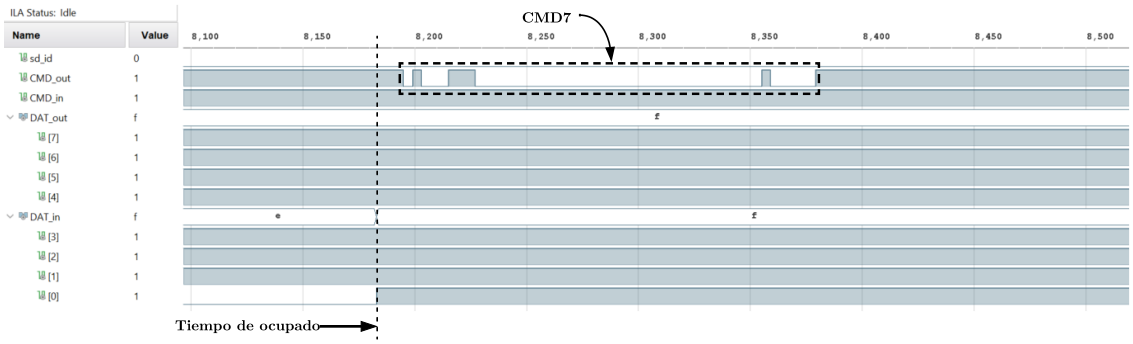


Figura 5.52: Diagrama de Tiempo de Fin de Operación de Transferencia

En las figuras 5.53 y 5.54 se visualiza mediante el uso del software HxD los primeros y últimos sectores escritos por el sistema en la tarjeta microSD. Se puede observar cómo cada sector corresponde a 512 bytes y cómo cada segmento de los datos de pruebas se han escrito correctamente manteniendo la secuencia con que fueron generados.

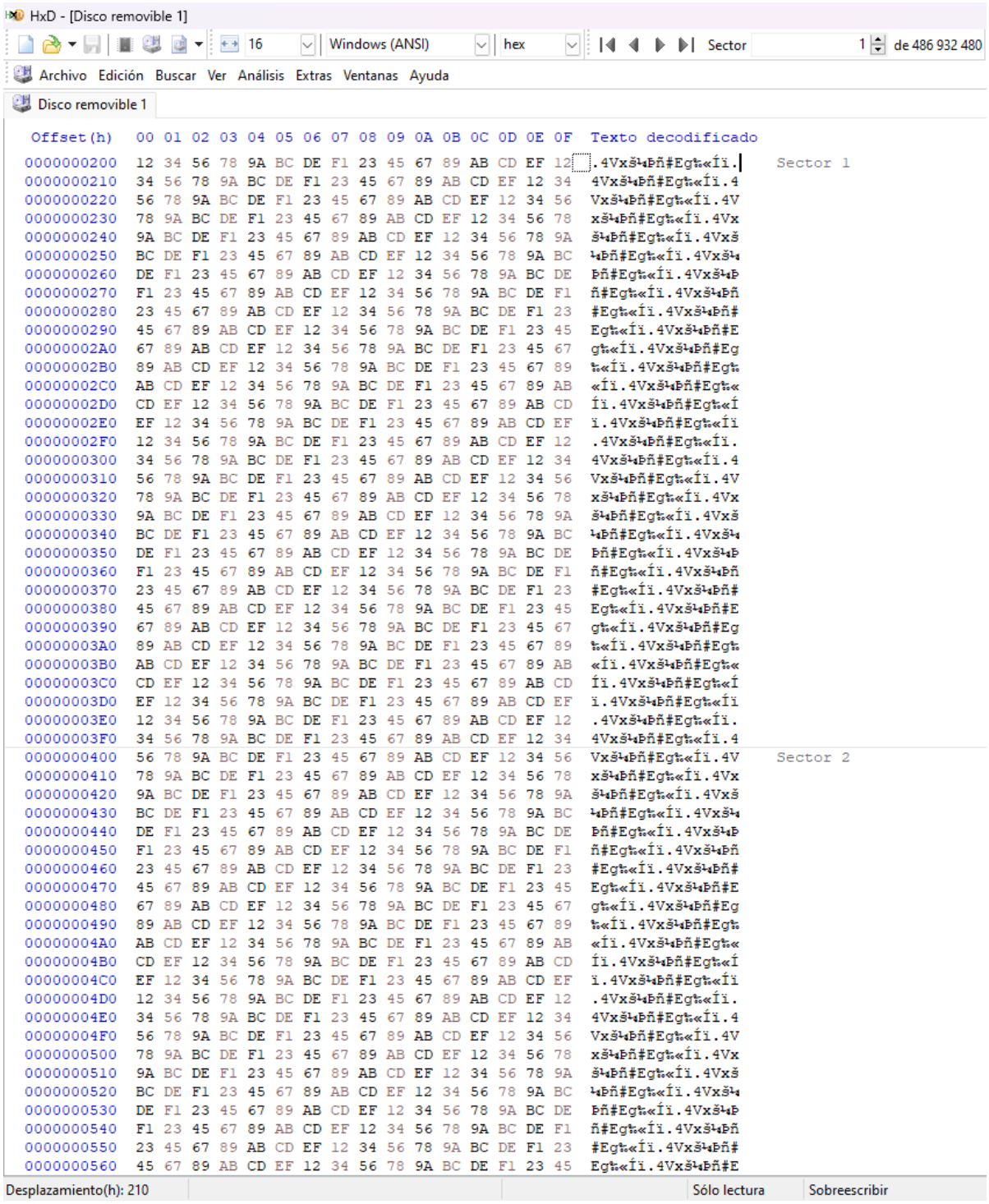


Figura 5.53: Primeros Sectores Escritos por el Sistema

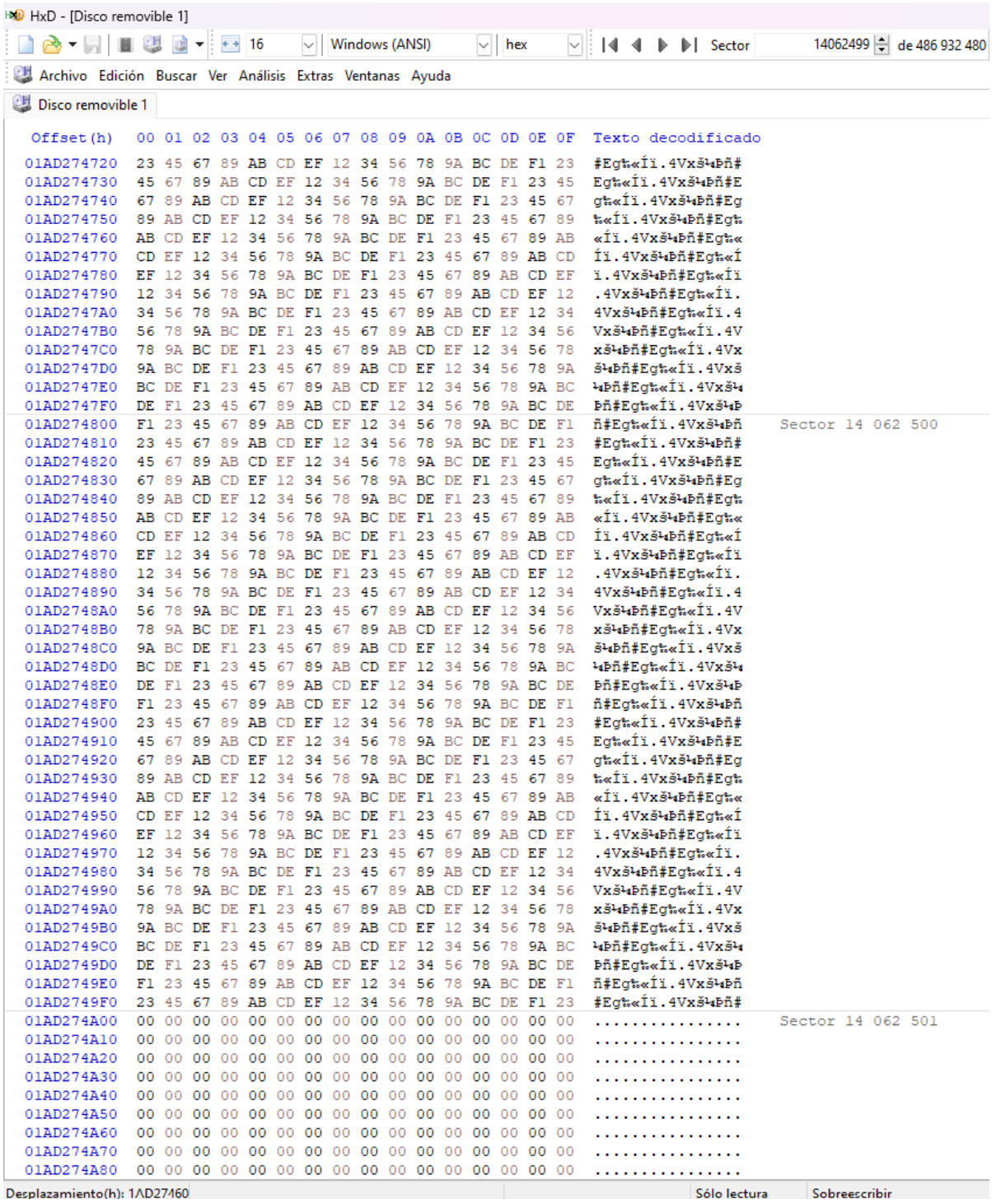


Figura 5.54: Últimos Sectores Escritos por el Sistema

5.3.5. Conexiones a las tarjetas microSD

En las figuras 5.55 y 5.56 se demuestra el funcionamiento del UCM. Como se observa, la señal `sd_id` es utilizada para la selección de la tarjeta, las señales del host por tanto, fueron transmitidas a la tarjeta seleccionada, mientras que la segunda tarjeta se mantuvo a la espera. La señal `sw_cmd` habilitó la línea `CMD` para el envío de comandos o recepción de respuesta y la señal `sw_dat` habilitó la línea `DAT` permitiendo el envío de bloques de datos o lectura de estado de ocupado o `CRC status`. Para la prueba de la segunda conexión se utilizo una microSD maxell® SDHC de 32GB.

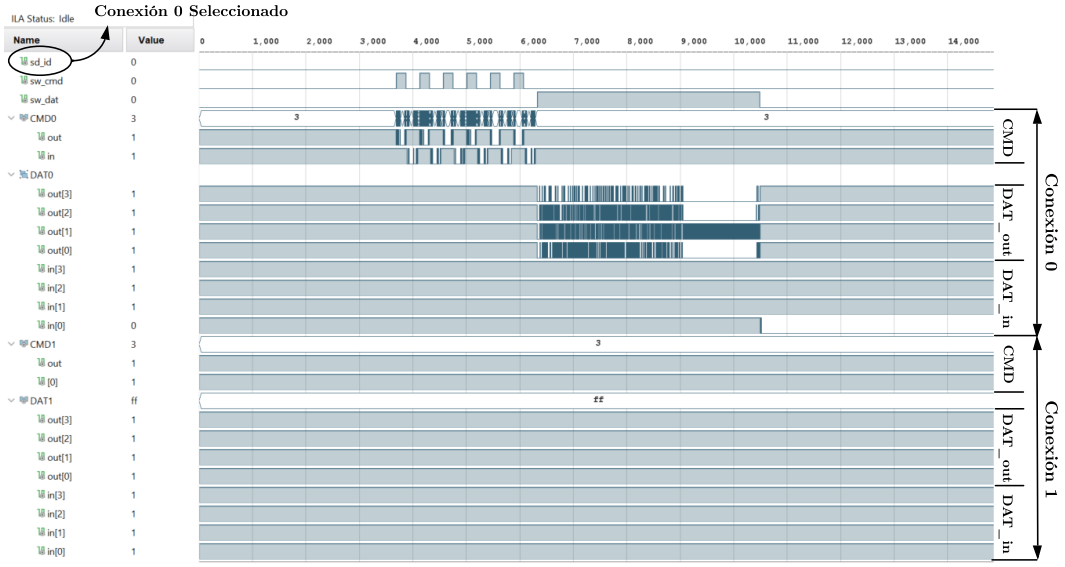


Figura 5.55: Diagrama de Tiempo de la Conexión 0

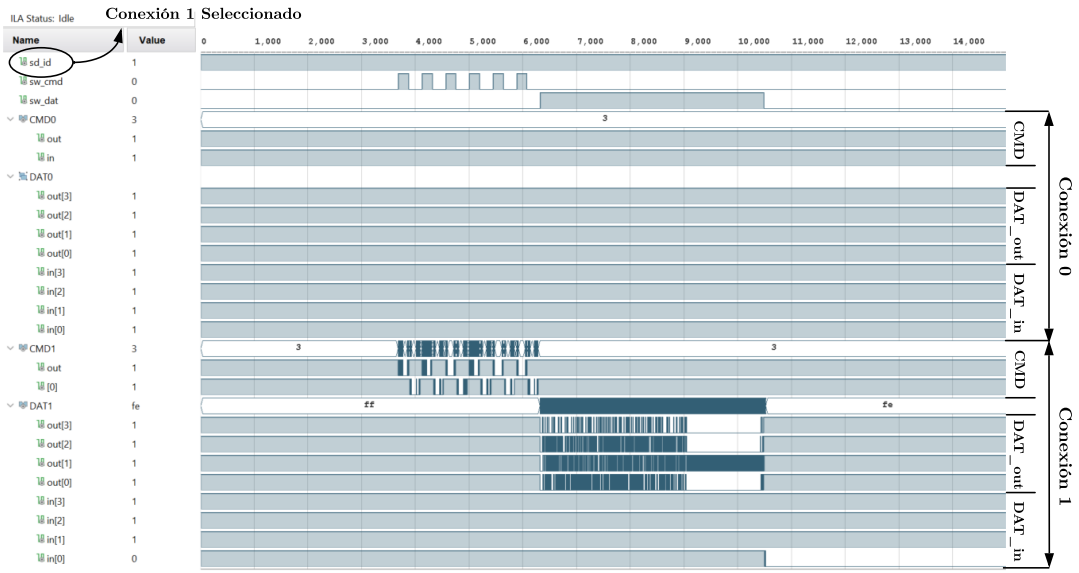


Figura 5.56: Diagrama de Tiempo de la Conexión 1

5.3.6. Comparativa de Rendimiento

Se evidencia cómo la elección de manejar los datos en binario tiene un impacto significativo en el almacenamiento. El sistema implementado en esta sección, para un segundo de datos, requiere 12 MB de almacenamiento. Lo que representa 23 437.5 sectores escritos en una tarjeta microSD (ver figura 5.57).

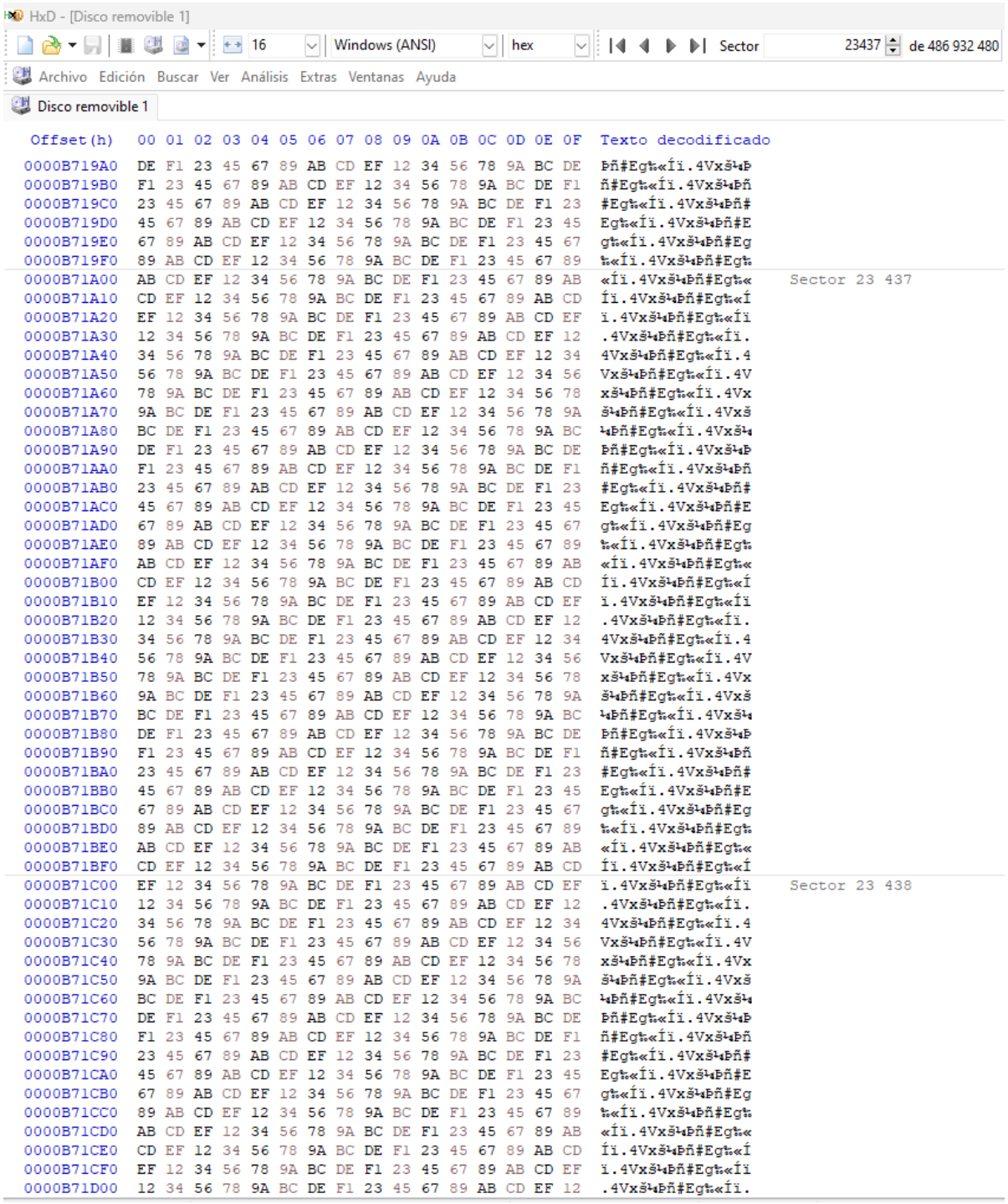


Figura 5.57: Almacenamiento de 12 MB en una tarjeta microSD

Al examinar la tabla 2.2, se observa que los dispositivos requieren entre 115 MB y 360 MB de datos en formato ASCII. Esta cantidad representa una diferencia significativa en tamaño, siendo de 9.6 a 30 veces mayor en comparación con el sistema que implementado. Extrapolando esta tendencia al almacenamiento necesario para una medición de 10 minutos, se obtendría de entre 69 GB y 220 GB, una cantidad impráctica para la mayoría de los sistemas. Además, procesar tal volumen de información demandaría recursos considerables. Por otro lado, el sistema descrito en este capítulo logra almacenar 10 minutos de datos crudos en aproximadamente 7.2 GB, un tamaño óptimo que cualquier tarjeta microSD puede manejar. Este contraste pone de manifiesto la eficiencia del sistema en términos de almacenamiento de datos.

Adicionalmente si observamos la figura 2.1, se tiene que los dispositivos de adquisición no están diseñados para manejar grandes cantidades de muestras en tiempo real durante períodos prolongados. Esto conlleva una pérdida inevitable de datos debido al tamaño limitado de los búferes con los que cuentan. Sin embargo, el sistema implementado está optimizado y diseñado específicamente para manejar velocidades de transferencia requeridas en tiempo real sin perder datos durante el proceso. El enfoque rotativo de los FIFOs permite resguardar los datos en momentos en los que la tarjeta microSD no permite una transmisión constante lo que garantiza un almacenamiento sin pérdida de información.

Capítulo 6

Conclusiones

En conclusión, el documento ha evidenciado que el sistema en FPGA diseñado es una solución eficiente y confiable para la transmisión y almacenamiento de datos biomédicos. Los resultados experimentales han confirmado que el sistema no solo cumple con los objetivos de rendimiento establecidos, sino que los supera, ofreciendo tasas de escritura de datos crudos a 100 Mbps y transmisión de datos procesados a 10 Mbps.

El sistema ha demostrado ser una herramienta robusta y fiable para la gestión eficiente de señales cardiacas, asegurando la integridad y la no pérdida de información en aplicaciones críticas. La capacidad del "Sistema de almacenamiento de datos en tiempo real en tarjetas microSD" para almacenar exitosamente 7.2 GB de datos crudos durante 10 minutos de medición, representa entre 9.6 a 30 veces menos almacenamiento que los sistemas de adquisición de datos, lo que evidencian el potencial para salvaguardar los datos provenientes de un sensor magnético. Además, el uso de tarjetas microSD como método de almacenamiento aporta beneficios significativos en términos de bajo costo y facilidad de movilidad. Esta característica no solo hace que el sistema sea más accesible y económico, sino que también facilita la portabilidad y el intercambio de datos en diferentes entornos y situaciones.

Por otro lado, el "Sistema de transmisión de datos en tiempo real a computadora" ha demostrado ser igual de eficiente. La implementación de un protocolo SPI sencillo pero robusto garantiza una comunicación confiable y segura entre la unidad de hardware y el IC FT232H, lo que es fundamental para mantener la integridad de los datos durante la transmisión. La facilidad de configuración del IC a través del lenguaje de programación Python facilita una recuperación rápida de los datos, permitiendo que estos se almacenen en un formato que demanda poco espacio, optimizando así el uso del almacenamiento disponible. Con un total de 114 MB para una medición de 10 minutos, el sistema no solo demuestra su capacidad para manejar grandes volúmenes de datos, sino que también asegura que el manejo posterior de los mismos sea práctico y eficiente.

La arquitectura modular del "Sistema de almacenamiento de datos en tiempo real en tarjetas microSD" compuesta por bloques independientes interconectados, confiere al sis-

tema una excepcional versatilidad y flexibilidad. Esta estructura permite que el "Sistema Host para Múltiples Tarjetas microSD" se adapte con facilidad a una amplia gama de aplicaciones y problemáticas relacionadas con el almacenamiento de datos. Ya sea que se trate de la captura de datos en tiempo real donde la rapidez y la precisión son esenciales, o en escenarios donde los datos pueden ser recolectados y almacenados para análisis posteriores, el SHM demuestra su capacidad para ajustarse a las necesidades específicas del usuario.

En cuanto a las recomendaciones, se sugiere la optimización del uso de recursos por parte del sistema de almacenamiento. Evaluando formas alternativas para almacenar los datos en tiempos de ocupado o hacer uso de múltiples tarjetas microSD para almacenar los datos en diferentes tarjetas durante los tiempos de ocupado.

Por lo tanto, el documento no solo ha contribuido a demostrar la efectividad de las implementaciones en hardware del almacenamiento y transmisión de datos en tiempo real provenientes de sensores biomagnéticos para uso biomédico, sino que aporta un diseño altamente reutilizable y adaptable mediante mínimas modificaciones que sientan las bases para el desarrollo de nuevos sistemas.

Bibliografía

[1] M. Sosa et al., «Técnicas biomagnéticas y su comparación con los métodos bioeléctricos,» *Revista Mexicana de Física*, vol. 48, n.º 6, págs. 490-500, 2002. dirección: https://www.scielo.org.mx/pdf/rmf/v48n6/v48n6a1.pdf.

[2] S. J. Williamson y L. Kaufman, «Magnetism and Magnetic Materials,» *Journal of Magnetism and Magnetic Materials*, vol. 22, pág. 129, 1981.

[3] D. Halliday, R. Resnick y K. S. Krane, *Física. Vol. 2.* México: Compañía Editorial Continental, 1996.

[4] «ICH GCP Clinical Trials Registry.» Accedido: 04-06-2024. (2021), dirección: https://ichgcp.net/es/clinical-trials-registry/NCT04352816.

[5] A. A. O. Carneiro, A. Ferreira, E. R. Moraes, D. B. Araujo, M. Sosa y O. Baffa, «Biomagnetismo: Aspectos Instrumentales y Aplicaciones,» *Revista Brasileira de Ensino de Física*, vol. 22, pág. 324, 2000.

[6] «Cómo funciona la magnetocardiografía.» Accedido: 04-06-2024. (), dirección: https://your-physicist.com/es/como-funciona-la-magnetocardiografia/.

[7] «Biomagnetic Sensing - Overview.» Accedido: 04-06-2024. (), dirección: https://www.biomagnetic-sensing.de/index.php/overview.

[8] R. Pitaya, «3.1.1.6. STEMlab 125-14 Z7020-LN — Red Pitaya 2.00-30 Documentation,» *Red Pitaya Documentation*, 2024. dirección: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14-Z20/top.html.

[9] Digilent, «Analog Discovery 3: 125 MS/S USB Oscilloscope, Waveform Generator, Logic Analyzer, and Variable Power Supply,» *Digilent*, 2024. dirección: https://digilent.com/shop/analog-discovery-3/.

[10] Saleae, «Product Fact Sheet Saleae Logic 8 USB Logic Analyzer,» *Saleae*, 2024. dirección: https://downloads.saleae.com/specs/Logic+8+Product+Fact+Sheet.pdf.

[11] N. Instruments, «PXIe-4480: Módulo de Sonido y Vibración con 6 Entradas, 1.25 MS/s, Acoplado AC/DC a 3.4 Hz,» *National Instruments*, 2024. dirección: https://www.ni.com/de-de/shop/model/pxie-4480.html.

[12] Z. Instruments, «HF2LI 50 MHz Lock-in Amplifier,» *Zurich Instruments*, 2019. dirección: https://www.zhinst.com/europe/en/products/hf2li-lock-in-amplifier#specifications.

[13] A. Taylor. «Introduction to FPGA.» Accedido: 06-05-2024. (2020), dirección: https://digilent.com/reference/_media/programmable-logic/arty-z7/introduction_to_fpga.pdf.

[14] Digilent Inc., *Nexys4 DDR™ FPGA Board Reference Manual*, Accedido: 06-05-2024, Digilent Incorporated, 1300 Henley Court Pullman, WA 99163, abr. de 2016. dirección: https://digilent.com/reference/_media/nexys4-ddr:nexys4ddr_rm.pdf.

[15] I. C. Society y the IEEE Standards Association Corporate Advisory Group. «IEEE Std 1800™-2012 (Revision of IEEE Std 1800-2009) IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language.» Accedido: 06-05-2024. (2013), dirección: https://ece.uah.edu/~gaede/cpe526/2012%20System%20Verilog%20Language%20Reference%20Manual.pdf.

[16] «Register Transfer Level (RTL) Design and Verification.» Accedido: 06-05-2024. (2024), dirección: https://semiengineering.com/knowledge_centers/eda-design/definitions/register-transfer-level/.

[17] Advanced Micro Devices, Inc. «Integrated Logic Analyzer (ILA).» Accedido: 06-05-2024. (2024), dirección: https://www.amd.com/en/support/kb/faq/ila.

[18] A. K. Oudjida, M. L. Berrandjia, R. Tiar, L. Ahmed y K. Tahraoui, «FPGA implementation of I2C & SPI protocols: A comparative study,» en *2009 IEEE International Conference on Electronics, Circuits and Systems*, Algiers, Algeria, 2009. DOI: 10.1109/ICECS.2009.5410881.

[19] P. Dhaker, «Introduction to SPI Interface,» *Analog Dialogue*, vol. 52, n.º 9, sep. de 2018.

[20] J. Yang, Y. Xiao, D. Li, Z. Li, Z. Chen y P. Wan, «Configurable SPI Interface Based on APB Bus,» 2020. DOI: 10.1109/ASID50160.2020.9271704.

[21] A. Pini. «Por Qué Y Cómo Usar La Interfaz Periférica Serial Para Simplificar Las Conexiones Entre Distintos Dispositivos.» Accedido: 04-06-2024. (2019), dirección: https://www.digikey.com/es/articles/why-how-to-use-serial-peripheral-interface-simplify-connections-between-multiple-devices.

[22] D. Devices, «Understanding SPI Bus: Clock Polarity and Phase (CPOL, CPHA),» mayo de 2020, Accedido: 08-04-2024. dirección: https://deardevices.com/2020/05/31/spi-cpol-cpha/.

[23] Future Technology Devices International Limited, *FT232H Single Channel Hi-Speed USB to Multipurpose UART/FIFO IC*, ver. 2.0, Documento No.: FT_000288 Clearance No.: FTDI #199, Future Technology Devices International Ltd, Glasgow, United Kingdom, 2020.

[24] Adafruit. «Assembly & Wiring | Adafruit FT232H Breakout - Adafruit Learning System.» Accedido: 08-04-2024. (2024), dirección: https://learn.adafruit.com/adafruit-ft232h-breakout/wiring.

[25] Adafruit. «CircuitPython Libraries on any Computer with FT232H.» Accedido: 08-04-2024. (2020), dirección: https://learn.adafruit.com/circuitpython-on-any-computer-with-ft232h/pinouts.

[26] E. Blot. «PyFtdi: FTDI device driver written in pure Python.» Accedido: 08-04-2024. (2024), dirección: https://eblot.github.io/pyftdi/api/spi.html.

[27] D. S. S. Nath, U. M. Krishnan, D. Navaneethan, B. M. Yogavignes y M. S. Sakthekannan, «SD Card Interface Using FPGA for Multimedia Applications,» en *Proceedings of the Sixth International Conference on Electronics, Communication and Aerospace Technology (ICECA 2022)*, IEEE, Chennai, India, 2022, ISBN: 978-1-6654-8271-4.

[28] Panasonic Corporation and SanDisk Corporation and Toshiba Corporation, *SD Specifications Part 1 Physical Layer Specification*, 3.01, SD Card Association, 2010.

[29] Kingston. «Guía de las categorías de velocidad de las tarjetas SD y microSD.» Accedido: 06-05-2024. (2020), dirección: https://www.kingston.com/es/blog/personal-storage/memory-card-speed-classes.

Apéndice A

Interfaz de Usuario en Terminal de Consola para la Realización de Pruebas de Transferencia a una Computadora

```
############################################################

USB_SPI COMMUNICATION

------------------------------------------------------------

Enter the duration in seconds: 10*60
-> Duration: 10 minutes and 0 seconds.

Enter the filename to save the data: Captura_Datos_10min

============================================================

Initiate communication...

------------------------------------------------------------

List of commands:

-> 0xDA: Start SPI communication

Enter the hexadecimal command: DA

------------------------------------------------------------

SPI connected

-> Peripheral response: 0x1e
-> Capturing data for: 10 minutes and 0 seconds.

------------------------------------------------------------

Time's up

-> Saving data to 'Captura_Datos_10min.bin'...

============================================================

Verifying the hexadecimal counter in Captura_Datos_10min.txt...

 -> THE HEXADECIMAL COUNTER IS CORRECT!!!

------------------------------------------------------------

End of SPI communication

############################################################
```

Figura A.1: Interfaz de Usuario en Terminal de Consola

Apéndice B

Tiempos Máximos Reportados del FT232H

Tabla B.1: Tiempos Máximos Reportados del FT232H

8.12E-03	1.00E-02	1.64E-02	1.47E-02
5.12E-03	7.88E-03	5.98E-03	9.12E-03
9.13E-03	9.23E-03	9.34E-03	5.14E-03
1.41E-02	6.56E-03	6.54E-03	7.77E-03
4.92E-03	1.51E-02	4.23E-03	9.12E-03
1.13E-02	**1.79E-02**	8.21E-03	1.44E-02
1.41E-02	1.12E-02	9.12E-03	1.51E-02
8.43E-03	1.51E-02	1.01E-02	1.20E-02
1.01E-02	1.31E-02	6.12E-03	5.12E-03
5.64E-03	8.92E-03	1.51E-02	9.65E-03
6.12E-03	9.53E-03	1.20E-02	9.99E-03
1.32E-02	6.12E-03	7.12E-03	6.71E-03
4.23E-03	9.53E-03	1.41E-02	7.20E-03
8.00E-03	6.35E-03	9.98E-03	8.19E-03
8.56E-03	4.34E-03	6.12E-03	7.84E-03
1.61E-02	7.23E-03	5.12E-03	8.88E-03
1.21E-02	8.21E-03	9.87E-03	7.78E-03
4.12E-03	1.61E-02	1.33E-02	9.12E-03
1.12E-03	1.22E-02	9.53E-03	1.51E-02
5.43E-03	8.12E-03	9.12E-03	1.23E-02
8.12E-03	6.43E-03	6.42E-03	1.23E-02
9.34E-03	7.77E-03	8.12E-03	6.12E-03
6.78E-03	5.34E-03	1.18E-02	4.12E-03
9.12E-03	1.42E-02	1.61E-02	1.39E-02
1.72E-02	9.76E-03	1.21E-02	9.54E-03

Apéndice C

Sistema físico implementado

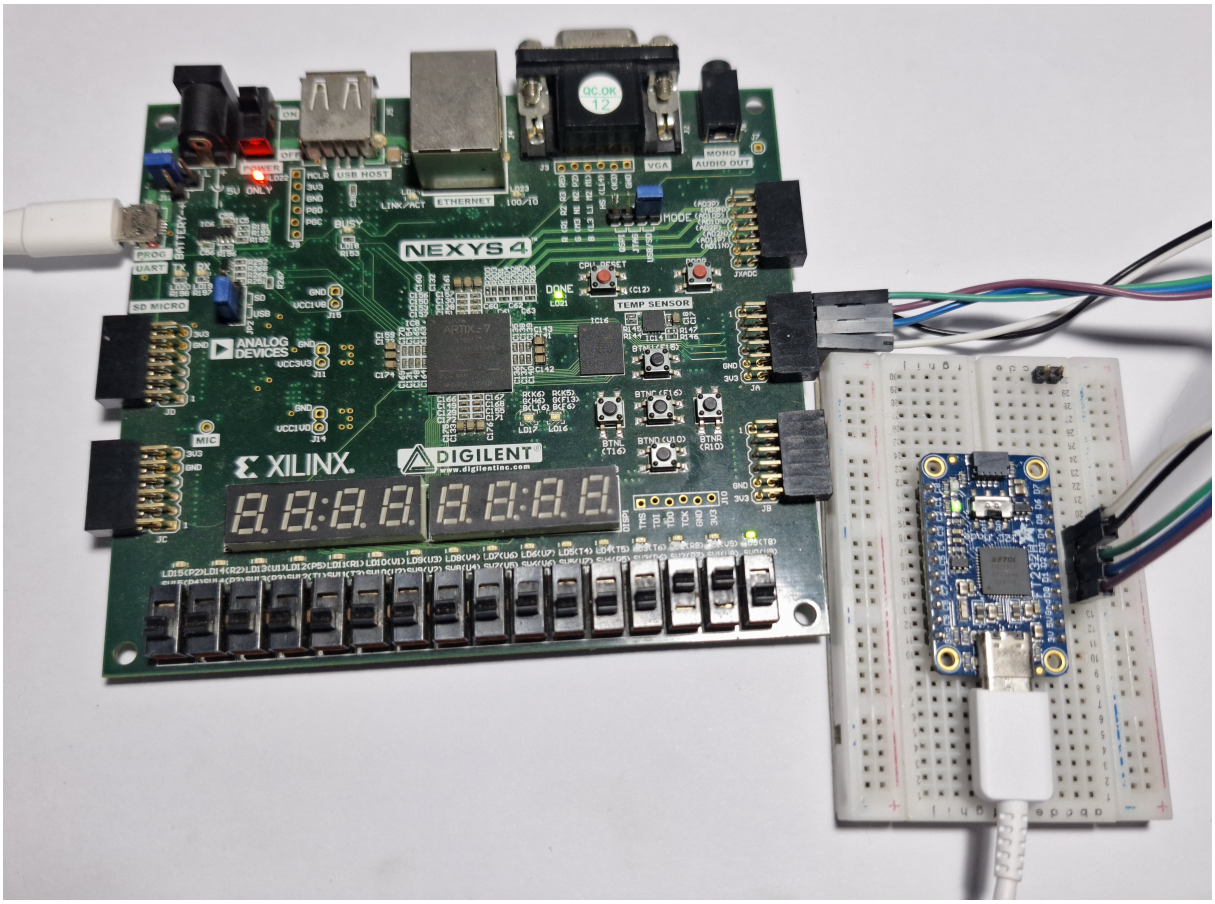


Figura C.1: Sistema de transmisión de Datos en Tiempo Real a Computadora

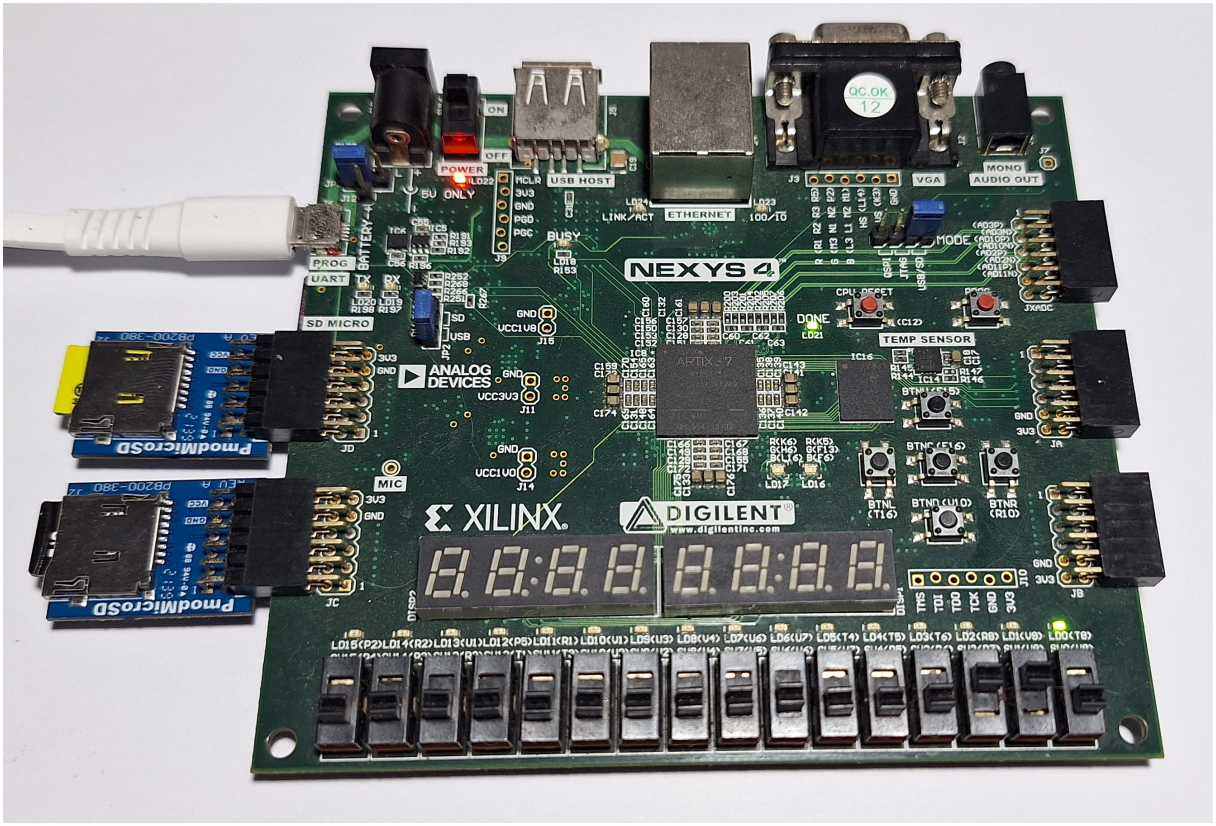


Figura C.2: Sistema de Almacenamiento de Datos en Tiempo Real en Tarjetas microSD

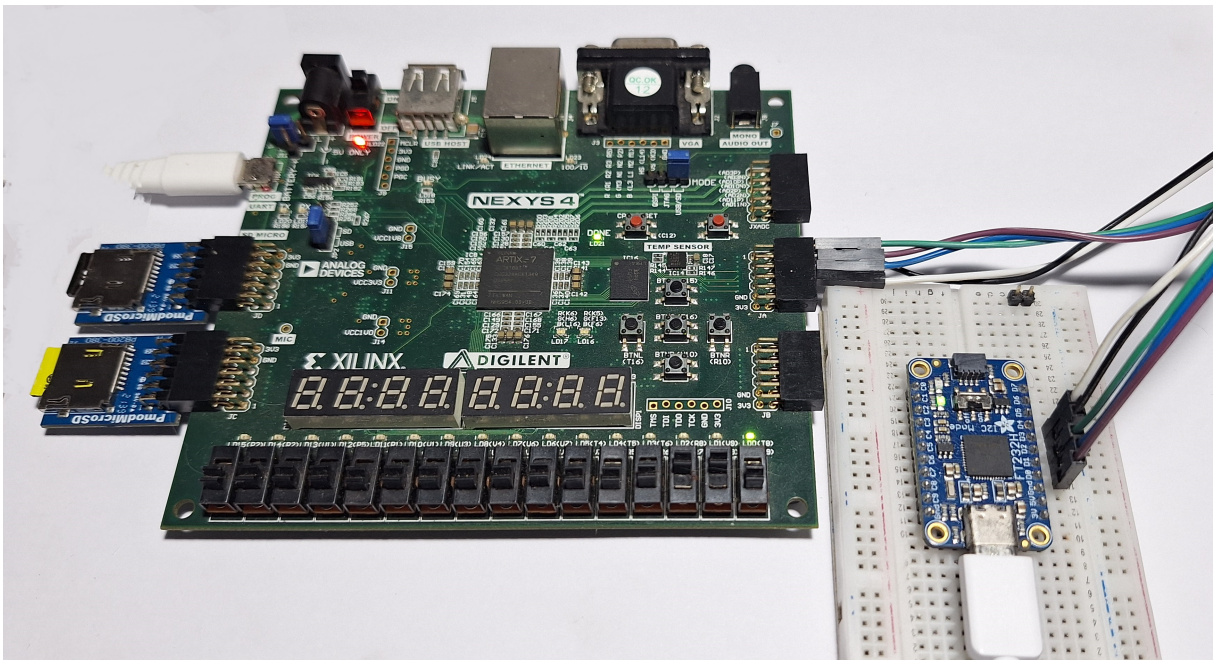


Figura C.3: Sistema de Almacenamiento y Transmisión