Universidad Zaragoza
1542

# Trabajo Fin de Grado

## Diseño de un periférico AXI-4 Lite para la adquisición de señales electro-fisiológicas con el integrado INTAN RHD22xx

## Design of an AXI-4 Lite peripheral for electro-physiological signal acquisition with the INTAN RHD22xx integrated circuit


Autor:

Antonio José Castel Santolaria

Director:

Isidro Urriza Parroqué


Ingeniería de Tecnologías y Servicios de la Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2023

# Agradecimientos

Gracias a todas las personas que me han acompañado y apoyado a lo largo de este camino. A todas las personas que estaban desde el principio y las que se han sumado a lo largo de estos cuatro años. A mi familia y amigos por su apoyo incondicional y por la confianza depositada durante todo este tiempo. A todos los que han compartido su tiempo conmigo, ayudándome a crecer no solo en el ámbito profesional si no también como persona. En especial a mi profesor y tutor, Isidro Urriza, por sus conocimientos, su tiempo, apoyo y paciencia que me ha brindado para poder completar este proyecto.

Gracias.

# Diseño de un periférico AXI-4 Lite para la adquisición de señales electro-fisiológicas con el integrado INTAN RHD22xx


## Resumen

En este trabajo se realiza el proceso de diseño de un periférico que actúa como intermediario en la comunicación entre un microprocesador *soft-core* y el circuito integrado RHD22xx desarrollado por INTAN. El periférico automatiza las tareas más comunes del RHD22xx reduciendo así la carga computacional del microprocesador.

En la etapa de diseño se sigue una arquitectura modular, implementando en bloques separados las tareas más comunes del RHD22xx: inicialización, comunicación directa RHD22xx-microprocesador y conversión de canales seleccionados. Se ha diseñado un bloque que incluye una interfaz SPI (Serial Peripheral Interface) para la comunicación con el RHD22xx.

En el apartado de verificación se realizan distintos bancos de pruebas tanto en el simulador de HDL (Hardware Description Language) incluido en Vivado, utilizando para ello modelo HDL comportamental del RHD22xx. Asi como del sistema real implementando el microprocesador, distintos periféricos y el periférico diseñado sobre una FPGA (Field Programable Gate Array). Se han incluido y verificado restricciones temporales, asegurando el correcto funcionamiento de la interfaz serie diseñada.

Se ha desarrollado un conjunto de rutinas en código C que permiten controlar el periférico desde el microprocesador.



## Abstract

The developed work details the design process of a peripheral that acts as an intermediary in the communication between a *soft-core* microprocessor and the RHD22xx integrated circuit developed by INTAN. The peripheral performs the most common tasks of the RHD22xx thus reducing the computational load of the microprocessor.

In the design stage, a modular architecture has been followed, implementing in separate blocks the most common tasks of the RHD22xx: initialization, direct communication RHD22xx - Microprocessor and conversion of selected channels. A block has been designed than includes an SPI interface (Serial Peripheral Interface) for communication with the RHD22xx.

In the verification stage, different testbenches are carried out booth in the HDL (hardware Description Language) simulator included in Vivado using a behavioral HDL model of the RHD22xx. As well as the real system, implementing the microprocessor, different peripherals and the peripheral designed on a FPGA (Field Programable Gate Array). Timming restrictions have been included and verified, ensuring the correct operation of the serial interface implanted in the peripheral.

A set of routines in C code has been developed allows the microprocessor to control de designed peripheral.

# Índice

# 1 Introducción

## 1.1 Planteamiento del problema

En los últimos años, la adquisición de señales electrofisiológicas ha sufrido un aumento de la tecnología fomentado, en parte, por la necesidad de reducir el equipo necesario para la toma de señales débiles de una forma precisa. Podemos encontrar en el mercado una gran variedad de soluciones, entre ellas la serie de microchips hacia las que va destinada este proyecto, la serie RHD22xx de INTAN Technologies [2].

La comunicación con esta gama de circuitos integrados se realiza de forma serializada, por lo que los comandos se envían de forma individual. Esto consume recursos del microprocesador que controla al RHD22xx haciendo que, en procesos que necesiten una gran cantidad de comandos, tenga que enviar cada comando de manera individual.

El objetivo de este proyecto es la creación de un periférico que permita descargar en lo posible al microprocesador de las tareas relacionadas con la comunicación con el microchip. Permitiendo además optimizar la gestión de las distintas funciones del RHD22xx. Además, se implementará un sistema formado por un microprocesador, los periféricos necesarios y el periférico diseñado sobre una FPGA (*Field Programable Gate Array*) para la evaluación del diseño propuesto en un entorno cercano a la realidad.

## 1.2 Elementos del sistema

Un periférico es un sistema incluido junto a un microprocesador, que permite realizar una tarea específica de forma concurrente a la CPU (*Central Processing Unit*). Además, es posible controlar y configurar su funcionamiento a través de las instrucciones que se ejecutan en la CPU. Los periféricos pretenden cubrir necesidades concretas del sistema reduciendo carga computacional del microprocesador, desde la gestión de la comunicación a través de diversas interfaces hasta la gestión de los relojes.

El periférico es controlado desde un microprocesador. A la hora de diseñar el sistema que va a ser implementado en la FPGA, encontramos dos tipos microprocesadores: *hard-core* y *soft-core*. El microprocesador *hard-core* se fabrica como un bloque hardware en el mismo silicio que la FPGA, mientras que el *soft-core* se implementa dentro de la FPGA utilizando los bloques lógicos de esta. Los microprocesadores *hard-core* permiten velocidades de funcionamiento mucho más altas, son fijos y no pueden ser modificados. Los microprocesadores s*oft-core*, al estar implementados sobre la FPGA pueden ser modificados para realizar funciones específicas, pueden utilizar varios núcleos en función de los recursos de la FPGA y presentan una velocidad de funcionamiento menor que la de los *hard-core*, limitada por la estructura de la FPGA.

En este proyecto, se ha decido el uso de un microprocesador *soft-core* debido a su capacidad de modificación. Se valoran las distintas opciones de microprocesadores de este tipo: MicroBlaze de Xilinx y el Cortex-M1 de ARM. Ambos presentan una arquitectura RISC (*Reduced Instruction Set Computer*) de 32 bits. Se opta finalmente

por el Cortex-M1 de ARM aunque podría utilizarse con cualquier microprocesador que presente una interfaz AXI4 [4].

Para la generación de los distintos elementos del sistema, microprocesador y periféricos, se ha utilizado el catálogo de IPs (*Intellectual Properties*) incluido en Vivado [7]. Para la creación del periférico del proyecto, se ha hecho uso del gestor de IPs de Vivado, que permite la generación y creación de nuevas IPs y periféricos. Generar el periférico con esta herramienta limita el uso de este a las FPGAS del mismo fabricante. En el apartado Flujo de diseño se incluye una breve descripción el proceso seguido para la generación del periférico y el resultado obtenido.

## 1.3 RHD22xx

La serie de circuitos integrados INTAN RHD22xx están desarrollados para la detección y captura de señales eléctricas débiles provenientes de tejidos vivos en distintas aplicaciones de monitorización. El RHD22xx cuenta con bancos de amplificadores de bajo ruido con ancho de banda programable y una arquitectura que combina un conversor analógico-digital (ADC) de 16 bits y distintos sensores de control. La estructura del RHD22xx permite la conexión directa de 16, o 32, electrodos. La comunicación con el RHD22xx es completamente digital y se realiza por medio de una interfaz serie (SPI) [2]. Al transformar las señales de los electrodos directamente en datos digitales, un solo RHD22xx permite reemplazar toda la parte analógica y de conversión utilizada en un sistema tradicional de captura de señales electrofisiológicas [1].

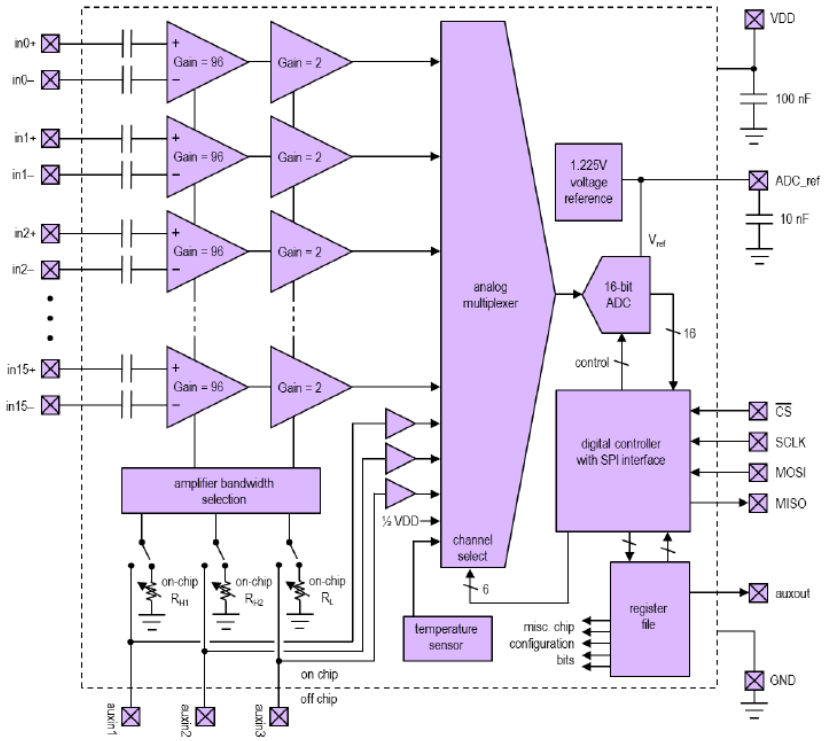


*Figura 1-1. Diagrama simplificado del RHD2216 [2, p.4].*

El RHD22xx incluye un módulo de procesado digital de señal (DSP) para implementar un filtro paso alto de un solo polo y frecuencia de corte variables en cada uno de los canales muestreados. Tiene por objetivo la eliminación de voltajes DC residuales asociados a los amplificadores analógicos. Este bloque puede activarse y configurarse desde los registros de configuración. A la hora de enviar comandos de conversión es posible habilitarlo o deshabilitarlo solo para esa conversión [2, pp. 32-33].

El integrado está preparado para la respuesta de 5 comandos enviados por la interfaz SPI: conversión ADC de un canal determinado, escritura en un registro, lectura de un registro, limpiar la calibración del ADC y calibración del ADC. Estos comandos de 16 bits se envían a través de la interfaz serie comenzando por el bit más significativo. A continuación, se detallan los comandos:

- **CONVERT(C)**: Este comando realiza la conversión ADC del canal seleccionado, *C*. La respuesta recibida contiene 16 bits con el resultado de la conversión. Es posible convertir los canales 0-31 que corresponden a los amplificadores del chip y los canales 32-62 que corresponden a sensores auxiliares y canales externos. Al enviar el comando de conversión del canal 63 se comienza un ciclo que convierte todos los canales de manera sucesiva desde el canal 0 al 62. Modificando el LSB del comando es posible activar o desactivar la eliminación de offset del filtro paso alto del módulo DSP.


*Figura 1-2. Comando y respuesta de conversión*

- **CALIBRATE()**: Inicia el proceso de calibración del ADC. Tras su envío es necesario continuar con el envío de 9 comandos que no serán ejecutados por el circuito integrado. Estos comandos se utilizan para mantener la transmisión de la señal de reloj. Sólo debe enviarse una vez durante la inicialización del chip.

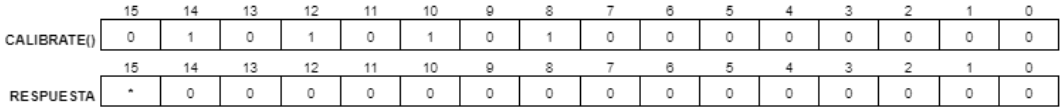


*Figura 1-3. Comando y respuesta de calibración*

- **CLEAR()**: Elimina los parámetros de calibración adquiridos en la calibración del chip.

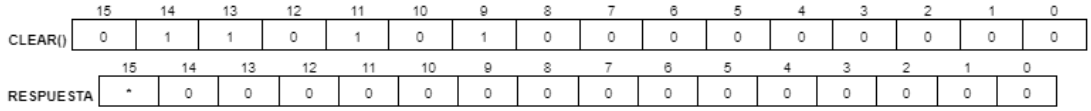


*Figura 1-4. Comando y respuesta de limpieza*

- **WRITE(addr, data)**: Escribe 8 bits de datos, *data*, en el registro indicado en el campo *addr*.

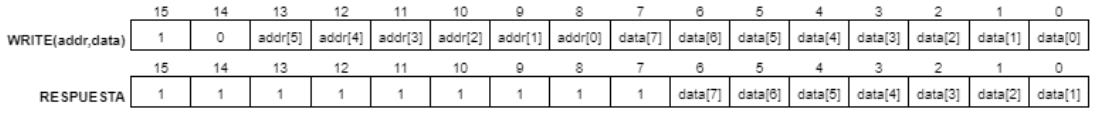


*Figura 1-5. Comando y respuesta de escritura*

- **READ(addr)**: Devuelve los 8 bits de datos almacenados en el registro indicado en el campo *addr*. Los datos se envían en el byte menos significativo de la respuesta.

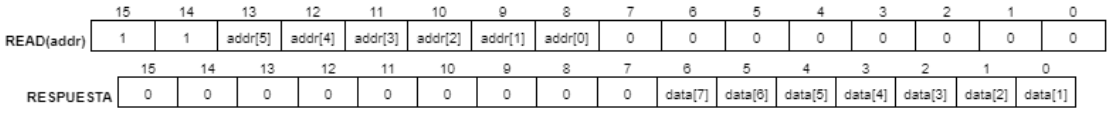


*Figura 1-6. Comando y respuesta de lectura*

Como ya se ha comentado, la comunicación con el periférico se realiza por medio de una interfaz serie. La comunicación tiene forma esclavo-maestro, donde el RHD22xx toma el rol de esclavo. Debido al funcionamiento del chip la respuesta a un comando enviado por el periférico es enviada a través de la interfaz serie con un retraso de 2 comandos como se aprecia en la Figura 1-7.

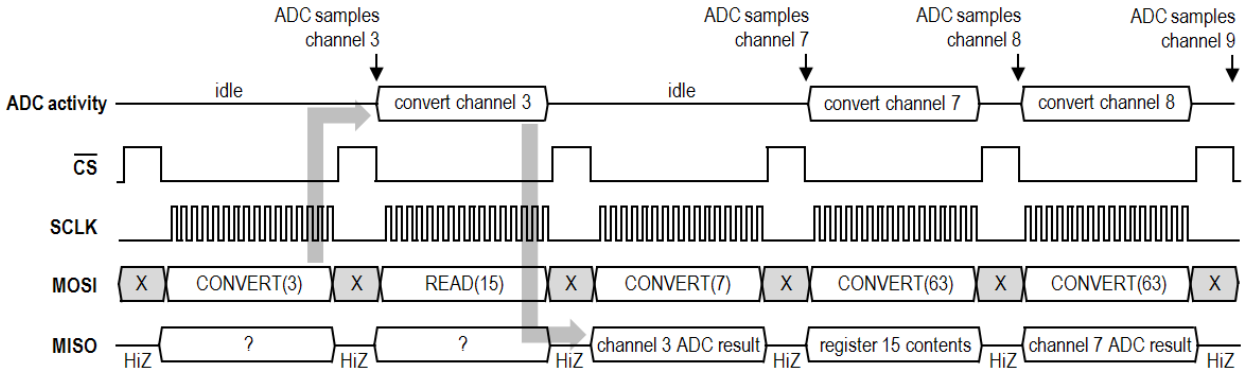


*Figura 1-7. Ejemplo de comunicación con el circuito integrado [2, p. 15].*

La distribución de los registros en el periférico mantiene la siguiente distribución:

- Los 18 primeros registros (@0x00 – @0x11) almacenan las distintas configuraciones del chip. Son registros de lectura y escritura, permitiendo su modificación desde la interfaz SPI. En la Tabla 1-1 se encuentra el nombre de cada registro y su dirección.
- Encontramos dos grupos de registros de solo lectura (ROM) en los que se almacena información sobre el RHD22xx. El primer bloque (@0x28-@0x2C) contiene los caracteres ASCII “INTAN”. En el segundo bloque (@0x3C-@0x3F) se almacena información sobre el microchip.

| Address | Acces type (Read/Write) | Register name |
|---|---|---|
| 0x00 | R/W | ADC Configuration and Amplifier Fast Settle |
| 0x01 | R/W | Supply Sensor and ADC Buffer Bias Current |
| 0x02 | R/W | MUX Bias Current |
| 0x03 | R/W | MUX Load, Temperature Sensor, and Auxiliary Digital Output |
| 0x04 | R/W | ADC Output Format and DSP Offset Removal |
| 0x05 | R/W | Impedance Check Control |
| 0x06 | R/W | Impedance Check DAC |
| 0x07 | R/W | Impedance Check Amplifier Select |
| 0x08 – 0x0D | R/W | On-Chip Amplifier Bandwidth Select |
| 0x0E-0x11 | R/W | Individual Amplifier Power |
| 0x28 – 0x2C | R | Company Designation |
| 0x3C | R | Die Revision |
| 0x3D | R | Unipolar/Bipolar Amplifiers |
| 0x3E | R | Number of Amplifiers |
| 0x3F | R | Intan Technologies Chip ID |

*Tabla 1-1. Dirección, tipo de acceso y nombre de los registros del INTAN RHD22xx*

Finalmente, hay que destacar que el RHD22xx permite la utilización de la interfaz SPI de forma estándar o utilizar señalización diferencial. Sin embargo, la placa que integra el RHD22xx utilizada en este proyecto solo permite la comunicación por medio de señales diferenciales de bajo voltaje (LVDS).

## 1.4 Alcance y objetivos

El siguiente proyecto tiene por alcance cubrir los objetivos que se detallan a continuación:

- Diseño de un periférico que facilite la comunicación entre el circuito integrado INTAN RHD22xx y un microprocesador por medio de las interfaces SPI y AXI4-Lite respectivamente.
- Diseño de un periférico que permita automatizar las tareas realizadas habitualmente en el RHD22xx, liberando de carga computacional al microprocesador que controla el sistema, siguiendo una arquitectura modular que permita la implementación de los distintos procesos como bloques independientes.
    - Diseño de un sistema de envío de comandos en la que el periférico solo realiza la traducción entre las distintas interfaces, AXI4-Lite y SPI, almacenando el contenido de la respuesta en un registro, funcionando de manera transparente.
    - Implementación del proceso de inicialización del RHD22xx desde el periférico, donde el usuario solo deba introducir la configuración del integrado y lanzar el proceso.
    - Diseño e implementación del proceso que permita realizar peticiones de conversión de canales desde una lista introducida por el usuario hacia el RHD22xx. El periférico permite un fácil acceso a la respuesta de estas conversiones y la capacidad para realizar este proceso de manera periódica.

## 1.5 Cronograma

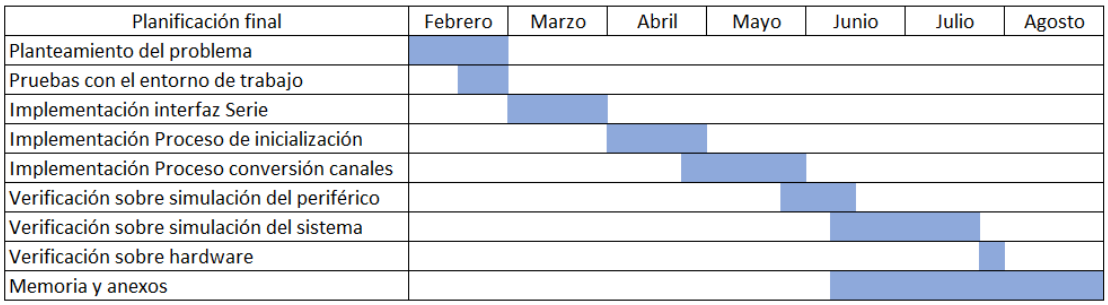


*Figura 1-8. Diagrama de Gantt del proyecto*

## 1.6 Herramientas utilizadas

Para la realización del proyecto se han utilizado las siguientes herramientas:

- **Vivado 2018.2**: Software de síntesis y análisis de diseños de descripción de hardware (HDL) preparado para el desarrollo de sistemas en FPGAs.
- **Sublime Text, Build 4151**: Editor de texto y de código, permitiendo el uso con gran variedad de lenguajes.
- **Keil uVision 5**: Software para el desarrollo de código C y C++ preparado para el desarrollo en sistemas embebidos, permitiendo compilar el programa y depurar el código.
- **FromElf image converter**: Conversor de imágenes ELF en otros formatos, como hexadecimal orientado a byte, para su carga en memoria o simulación en simuladores HDL.
- **INTAN RHD2216**: Circuito integrado desarrollado por INTAN y utilizado para validar el funcionamiento del periférico implementado.
- **SN65LVDT14**: Circuito integrado desarrollado por Texas Instruments para adaptar una interfaz serie estándar sobre señales diferenciales de bajo voltaje.
- **Placa ARTY A7**: Placa de desarrollo de la marca Digilent que contiene una FPGA xc7a100tcsg324-1, Artix-7, de Xilinx. Los pines de la placa se conectan al SN65LVDT14 para la comunicación con el RHD2216.
- **J-link edu**: Depurador desarrollado por Segger utilizado para la carga del programa en C sobre el microprocesador implementado en la FPGA y para la comunicación con el ordenador

## 1.7 Estructura de la memoria

En la presente memoria se recoge todo el proceso de diseño llevado a cabo a lo largo del proyecto. El documento presenta 2 capítulos principales: Diseño y verificación. En el capítulo 1 se detalla el diseño del periférico en VHDL y las soluciones propuestas. En el capítulo de verificación se expone el proceso de validación de las soluciones y se verifica la implementación del periférico diseñado. Para completar el documento se incluye una conclusión del trabajo.

# 2 Diseño

## 2.1 Especificaciones

Una vez introducido el problema, es necesario desarrollar las especificaciones que presentará el periférico a diseñar:

- Para la comunicación con el microprocesador se utilizará la interfaz AXI4-Lite.
- Integra una interfaz serie (SPI) maestra para la comunicación con el RHD22xx.
- El periférico utiliza la señal de reset de la interfaz AXI4-Lite, activa en bajo, como señal de reset del periférico.
- El sistema funcionará con un reloj de frecuencia $f_{clk}$ igual a 96 MHz. El reloj de la interfaz serie deberá funcionar a una frecuencia $f_{sclk}$ igual a 24 MHz, siendo esta la frecuencia máxima de funcionamiento del RHD22xx.
- Siguiendo la distribución de memoria del RHD22xx, el periférico contará con 64 registros de 32 bits, accesibles desde la interfaz AXI4-Lite para su lectura o escritura dependiendo de la función que desempeñen. Los registros contendrán una copia de la información almacenada en el RHD22xx y serán utilizados para el control y configuración del periférico.
- El control del periférico se realizará mediante la escritura desde la interfaz AXI4-Lite en uno de los registros. Asimismo, para comprobar el estado del periférico bastará con leer el mismo registro desde la interfaz AXI4-Lite.
- El periférico debe realizar 3 procesos distintos: Comunicación directa, proceso de inicialización y conversión de una serie de canales. La comunicación directa deberá enviar por la interfaz SPI los comandos escritos en el registro habilitado y almacenar la respuesta inmediata en otro de los registros. El bloque de inicialización enviará la configuración, escrita en los registros habilitados del periférico desde el microprocesador, al RHD22xx, deberá completar de manera automática un proceso de inicialización basado en el ejemplo incluido en la hoja de características del RHD22xx y realizar una copia de los registros de este en el periférico, permitiendo así su lectura en paralelo. Finalmente, el bloque encargado de la conversión de canales utilizará un registro para seleccionar los canales que interesa convertir y realizará el proceso. El periférico sólo permitirá seleccionar los 32 canales por defecto del RHD22xx, no permitirá la selección de los canales auxiliares ni de los sensores del RHD22xx. Además, contará con un temporizador para poder realizar la conversión de manera periódica.
- Presenta la generación de interrupciones, por medio de 2 puertos preparados para ello, que se conectarán al gestor de interrupciones del microprocesador. Estas interrupciones indicarán la llegada de la respuesta de un comando y la finalización de un bucle de conversión de canales.

## 2.2 Flujo de diseño

El periférico se genera mediante el gestor de IPs del programa Vivado de Xilinx, lo que limita el proyecto a usar las FPGAs de esta marca. A través del asistente se genera una IP con interfaz AXI4-Lite y un banco 64 registros [3]. A continuación, se diseña la interfaz SPI y se añade al periférico base. Se realizan pruebas independientes de la interfaz SPI para el ajuste de los tiempos.

Para facilitar el diseño de periférico, el resto de los procesos se implementan en bloques separados y tras probar cada bloque de manera individual, se integran en el periférico como se puede ver en la Figura 2-5 del apartado Descripción de los bloques. Cada vez que se añade un nuevo bloque al periférico, la máquina de estados encargada del control del periférico es modificada. Una vez el periférico reúne todos los bloques, se procede a la prueba del conjunto en simulación y sobre hardware.

## 2.3 Protocolos de comunicación

Como ya se ha comentado, para la comunicación con el integrado se utilizan la interfaz SPI y para la comunicación con el microprocesador una interfaz AXI4-Lite. Además, para la comunicación entre los distintos bloques que integran el periférico se utilizan interfaces AXI4-Stream.

### 2.3.1 AXI4-Lite

La interfaz AXI4-Lite es parte del protocolo AXI (*Advanced eXtensible Interface*) desarrollado por ARM y perteneciente a la especificación AMBA (*Advanced Microcontroller Bus Architecture*) [4].

En la comunicación por medio de AXI4-Lite intervienen 5 canales:

- Canal de dirección de lectura
- Canal de datos de lectura
- Canal de dirección de escritura
- Canal de datos de escritura
- Canal de respuesta de escritura

Los 2 primeros canales se utilizan en el proceso de lectura y los 3 siguientes permanecen en reposo hasta el proceso de escritura. Todos los canales utilizan un método de *handshake* utilizando 2 señales para ello: *VALID* y *READY* como se indica en la Figura 2-2.

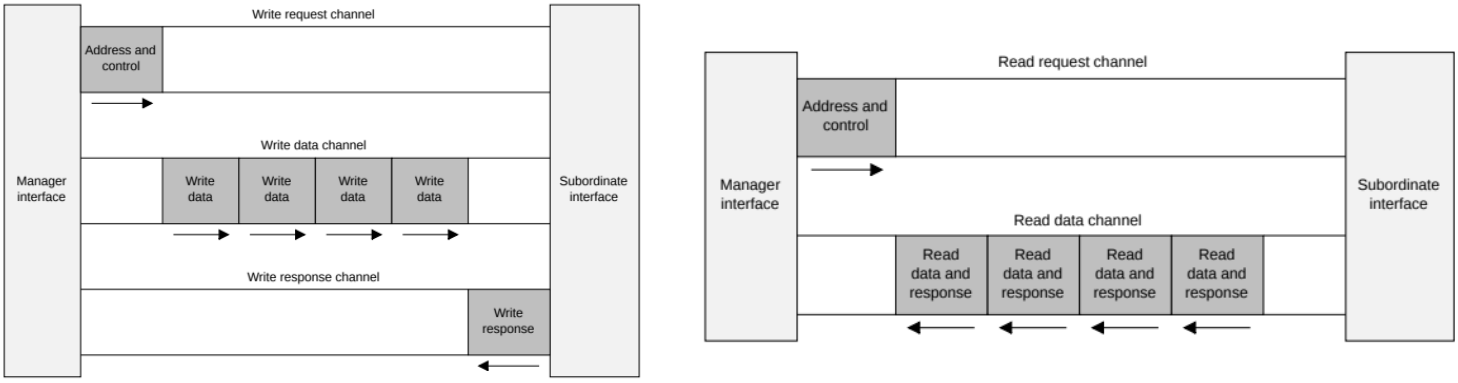


*Figura 2-1. Arquitectura de canales de la interfaz AXI4-Lite para el proceso de escritura (izquierda) y de lectura (derecha) [4, p.27].*

En la Figura 2-1, la interfaz generada toma el rol de interfaz subordinada ya que el periférico actua como esclavo en la comunicación, recibe peticiones de escritura y lectura en sus registros. Estos procesos comienzan con la llegada de la dirección del registro a acceder por el canal de lectura o escritura. Si la petición es de escritura a continuación se reciben los datos por el canal indicado y tras terminar la escritura el periférico responde indicando el estado de esta. En el caso de una petición de lectura se devuelven los datos por el canal indicado.

La especificación de AXI4-Lite incluye varias señales opcionales que añaden distintas funcionalidades a la interfaz. En este proyecto se opta por una interfaz simplificada por lo que solo se utilizan las señales indispensables. En la Tabla 2-1 se muestran las señales utilizadas en el proyecto.

| Señales globales | Canal dirección escritura | Canal datos escritura | Canal respuesta escritura | Canal dirección lectura | Canal datos lectura |
|---|---|---|---|---|---|
| ACLK<br>ARESETn | AWVALID<br>AWREADY<br>AWADDR | WVALID<br>WREADY<br>WDATA | BVALID<br>BREADY<br>BRESP | ARVALID<br>ARREADY<br>ARADDR | RVALID<br>RREADY<br>RDATA |

*Tabla 2-1. Desglose de las señales utilizadas en la interfaz AXI4-Lite*

El proceso de *handshake* [4] indica cuando los datos transmitidos por la interfaz son válidos y cuando el receptor los ha recibido. Cada una de las señales del proceso es generada por una de las partes involucradas en la comunicación: En el caso de la señal *VALID* la genera el extremo emisor de los datos. Esta señal indica que los datos que encontramos en el canal de datos son correctos. La señal *READY* la genera el extremo receptor. Se encarga de indicar que el receptor está disponible para la captura de los datos y la ha realiza. Una vez ambas señales se encuentran activas simultáneamente y se produce un flanco de subida en la señal de reloj, la transmisión de los datos se ha completado y el emisor puede cambiar los datos del canal.

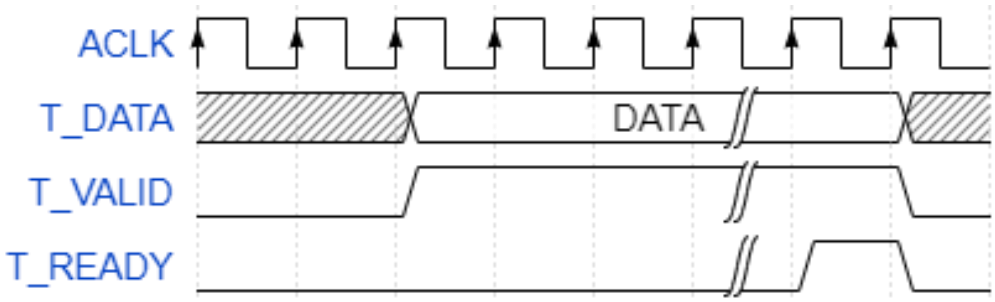


*Figura 2-2. Ejemplo del protocolo de handshake*

### 2.3.2 AXI4-Stream

Para la comunicación entre los bloques con el bloque encargado de la gestión de la interfaz SPI se utilizan interfaces AXI4-Stream [5]. Esta interfaz pertenece al protocolo AXI al igual que la anterior. Para la comunicación se utiliza una versión reducida con tres señales: un bus de datos de 16 bits y las dos señales necesarias para el *handshake, VALID* y *READY*, que ocurre de la misma forma que en la interfaz AXI4-Lite como se muestra en la Figura 2-2.

El control de la transmisión de datos a través de esta interfaz se realiza por medio de máquinas de estados finitos (MEF) que se detallaran en secciones posteriores.

### 2.3.3 Serial Peripheral Interface (SPI)

La comunicación con el RHD22xx se realiza a través de una interfaz serie. SPI es un estándar de comunicaciones para la transferencia de datos entre circuitos integrados. Consta de 4 puertos [6]:

- **Clock (SCLK)**: Transporta la señal de reloj de la interfaz. Dirección: Maestro → Esclavo.
- **Master Input Slave Output (MISO)**: Transporta bit a bit el comando enviado por el maestro. Dirección: Maestro → Esclavo.
- **Master Output Slave Input (MISO)**: Transporta bit a bit la respuesta al comando anterior enviada por el esclavo. Dirección: Esclavo → Maestro.
- **Chip Select (CS)**: Permite seleccionar el chip al que va dirigida la comunicación en caso de utilizar la misma interfaz SPI para controlar varios dispositivos. Dirección: Maestro → Esclavo.

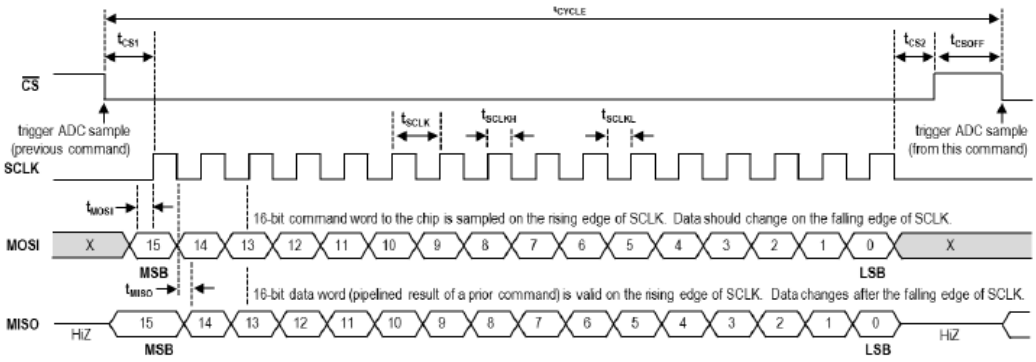


*Figura 2-3. Diagrama temporal de la interfaz serie [2, p. 15]*

SPI permite 4 modos de trabajo en función de dos parámetros de la señal *SCLK*: la polaridad en reposo y la fase utilizada para el cambio de las señales *MOSI* y *MISO*. En el problema planteado, como puede verse en la Figura 2-3, el reloj se mantiene a nivel bajo en reposo, los datos de las señales *MOSI* y *MISO* cambian con el flanco de bajada de *SCLK*.

En el problema planteado nuestro periférico actúa como dispositivo maestro y el RHD22xx actúa como esclavo [2]. Esto hace que el periférico sea quién inicie la comunicación en todo momento cambiando el estado de la señal *CS*, enviando la señal de reloj *SCLK* y la ráfaga de bits por el puerto *MOSI*. El periférico deberá encargarse de capturar los datos recibidos por el puerto *MISO*.

La señal *SCLK* es la que se utiliza como señal de reloj en el RHD22xx, se fija a la frecuencia máxima especificada en la hoja de características: $f_{sclk} = 24$ MHz [2, p. 15]. Esta señal solo se encontrará activa durante el tiempo que dure la transmisión de información.

La comunicación se realiza en series de 16 bits en ambas direcciones a través de los puertos *MOSI* y *MISO*, comenzando por el bit más significativo, según las especificaciones del RHD22xx [2] estas señales cambian en el flanco de bajada de la señal *SCLK*. Cada serie va precedida de un cambio a nivel bajo de la señal *CS* y al finalizar vuelve a nivel alto.

## 2.4 Diseño del sistema

El sistema completo está formado por un microprocesador S*oft-core* conectado al periférico diseñado por medio de una interfaz AXI4-Lite y de 2 interrupciones. La interfaz serie del periférico tiene conectado un microchip de Texas Instruments, el circuito integrado SN65LVDT14, que realiza la conversión de la interfaz SPI estándar a señales diferenciales de bajo voltaje (LVSD). El SN65LVDT14 está conectado por medio de un cable al RHD22xx. Para completar el sistema se incluirán otros periféricos que permiten cubrir distintas funciones.

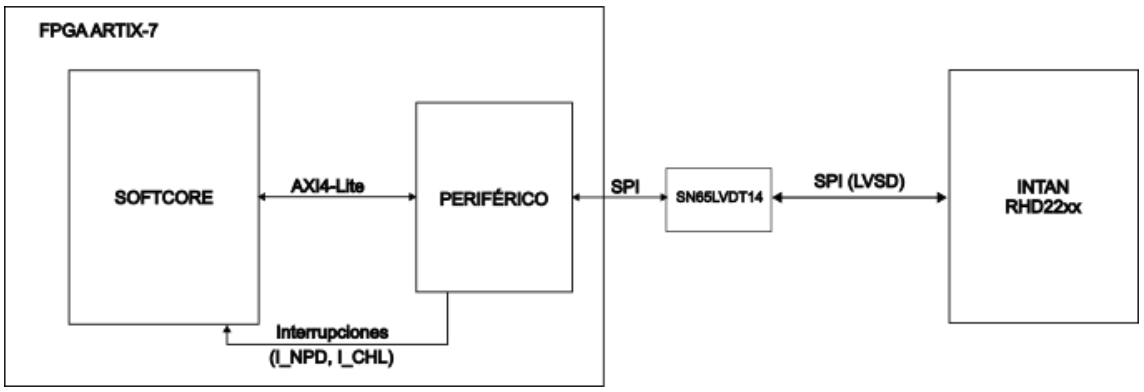


*Figura 2-4. Diagrama simplificado del sistema completo*

### 2.4.1 Periférico

El periférico cuenta con dos interfaces de comunicación principales nombradas anteriormente: AXI4-Lite para la comunicación con el microprocesador y la interfaz SPI para la comunicación con el RHD22xx. Además, cuenta con 2 puertos de interrupción pensados para ser conectadas con el gestor de interrupciones del microprocesador, de

esta manera el periférico avisa de la llegada de una respuesta y la finalización de un bucle de conversión de forma activa, sin necesidad de leer el registro de control.

El periférico cuenta con un banco de 64 registros, distribuidos siguiendo el mapa de memoria del RHD22xx, indicado en la sección RHD22x. Los registros con direcciones que el RHD22xx no utiliza son asignados en el periférico para el control del periférico, configurar y almacenar información de los distintos procesos.

Los registros del periférico utilizados son los siguientes:

- **Registros de configuración del INTAN RHD22xx (Direcciónes *@0x00-@0x44*)**: 18 registros utilizados para configurar el circuito integrado durante el proceso de inicialización.
- **Registro de comando con prioridad *(@0x48*)**: Utilizado para el envío de comandos directos. Los comandos escritos sobre este registro son enviados al circuito integrado durante el proceso de comunicación directa.
- **Registro de respuesta al comando con prioridad *(@0x4C*)**: Utilizado para la recepción de la respuesta a los comandos enviados. Sobre este registro se escribe la respuesta inmediata al comando con prioridad enviado.
- **Registro de estado y control *(@0x50*)**: Registro desde el que se controla todo el funcionamiento del periférico, el inicio de los procesos y la finalización de estos.
- **Registro de selección de canales *(@0x54*)**: Utilizado para la selección de los canales a convertir cuando se realiza un bucle de conversión.
- **Registros de conversión de canales *(@0x58-@0x94*)**: 16 registros utilizados para la recepción y almacenamiento de los resultados de la conversión de canales. En cada registro se almacena el resultado de la conversión de 2 canales.
- **Registros de sólo lectura del INTAN *(@0xA0-@0xB0, @0xF0-@0xFC*)**: Registros que contienen una copia de los mismos registros del circuito integrado.

El mapa de registros del periférico y la descripción de estos se encuentra ampliada en la documentación del periférico, incluida en los documentos anexos.

Para el cálculo de la frecuencia de reloj del periférico, $f_{clk}$, se han tenido en cuenta las condiciones impuestas por los demás elementos que forman el sistema completo. En primer lugar, el RHD22xx fija el reloj de la interfaz serie a una frecuencia máxima $f_{sclk} = 24\ MHz$ [2, p. 15]. Por otro lado, al implementar Cortex M1 en la FPGA, la frecuencia máxima de funcionamiento asciende hasta los $f_{\mu P} = 110\ MHz$.

Con las condiciones fijadas, se valoran dos opciones para la gestión de los relojes:

- Cada elemento del sistema implementado en la FPGA (microprocesador y periféricos) funcionará a una frecuencia de reloj diferente. A partir del reloj interno de la FPGA, se utiliza un MMCM (*Mixed-Mode Clock Module*) para generar los distintos relojes, incluyendo el reloj de la interfaz serie, a las frecuencias deseadas.
- A partir del reloj interno de la FPGA, utilizando un MMCM se genera un único reloj para todo el sistema implementado en la FPGA. El reloj de la interfaz serie se genera a partir del reloj del sistema, por lo que la frecuencia del reloj del sistema deberá ser múltiplo exacto de la $f_{sclk}$.

Tras valorar ambas opciones, se opta por utilizar el mismo reloj en todo el sistema facilitando el diseño y evitando los problemas que aparecen al intercambiar información entre sistemas con distintos relojes, conocido como CDC (*Clock Domain Crossing*). Tras elegir el método de gestión de relojes, se aplican las condiciones del problema planteado obteniendo una frecuencia de reloj del sistema igual a $f_{clk} = 96\ MHz$.

$$f_{clk} \leq 110\ MHz$$

$$f_{clk} = f_{sclk} \cdot M = 24\ MHz \cdot M$$

$$f_{clk} = 24\ MHz \cdot 4 = 96\ MHz < 110\ MHz$$

*Ecuación 1. Cálculo de la frecuencia de reloj del sistema*

### 2.4.2 Descripción de los bloques

Para facilitar el diseño del periférico se ha dividido en los distintos bloques que componen la siguiente figura. La interfaz SPI se diseñará como un bloque separado y cada uno de los procesos se diseña de manera independiente. El diagrama de bloques del periférico se puede ver en la Figura 2-5.

Como ya se ha comentado, el periférico se genera utilizando el gestor de IPs de vivado [3]. Se genera un periférico con interfaz esclava AXI4-Lite y 64 registros sobre el que se realizarán todas las modificaciones necesarias. El bloque de control y el de comunicación directa se implementan sobre el periférico.

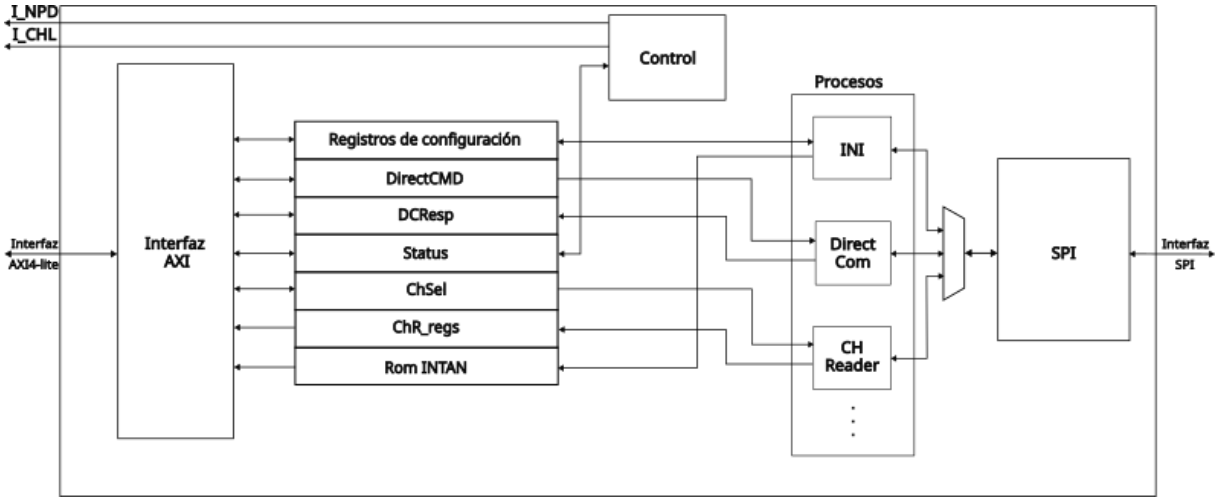


*Figura 2-5. Diagrama de bloques del periférico*

#### *2.4.2.1 Bloque SPI*

La interfaz serie se concibe bajo un bloque independiente encargado de la gestión de la transmisión y recepción.

Este bloque es el encargado de la comunicación con el circuito integrado, enviando y recibiendo la información por la interfaz SPI. Los datos recibidos y enviados son paquetes de 16 bits. Además, genera la señal *SCLK* de la interfaz serie un reloj de $f_{sclk} = 24$ MHz a partir de la señal de reloj del sistema, $f_{clk} = 96$ MHz.


*Figura 2-6. Bloque SPI*

El bloque SPI, Figura 2-6, cuenta con 2 interfaces AXI4-Stream, para la recepción y el envío de comandos y respuestas, que le permiten comunicarse con el resto de los bloques del periférico, además de la interfaz serie que se comunica con el exterior. Utiliza también reset activo en bajo y trabaja con la señal de reloj del periférico.

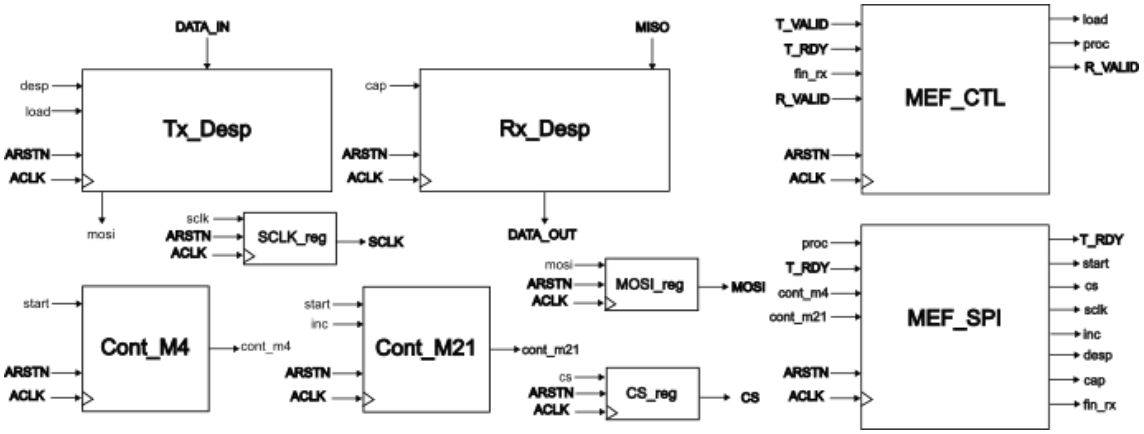


*Figura 2-7. Componentes que conforman el bloque SPI*[1]

Se ha decidido controlar el bloque mediante 2 máquinas de estados finitos (*MEF_CTL* y *MEF_SPI*), una encargada del control de la interfaz serie mientras la otra se encarga de la gestión general del bloque. El uso de 2 MEFs simplifica la implementación permitiendo separar la gestión de las interfaces AXI4-Stream del proceso de envío y recepción por la interfaz SPI.

Como se muestra en la Figura 2-7, Se utilizan también 2 registros de desplazamiento de 16 bits (*Tx_Desp* y *Rx_Desp*), encargados de almacenar las palabras enviadas y recibidas; un contador módulo 4 (*Cont_M4*) encargado de contar los ciclos del reloj del sistema que se utilizan en cada periodo de la señal *SCLK* y un contador módulo 21

[1] En los siguientes diagramas de bloques y máquinas de estado finitas se adopta el uso de minúsculas para indicar las señales internas del bloque al que pertenecen los componentes. Los puertos del bloque se denotan en mayúsculas.

(*Cont_M21*) para controlar la duración de la comunicación serie. Son necesarios 3 registros para retrasar las salidas *MOSI, CS* y *SCLK*.

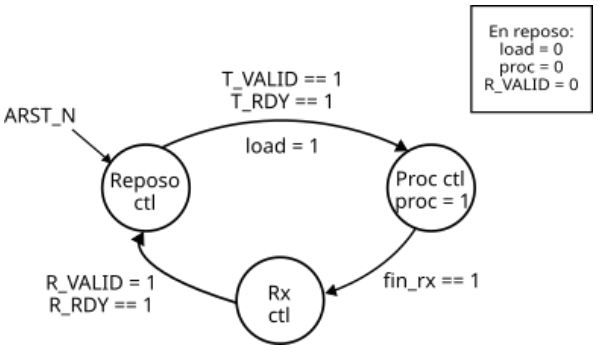


*Figura 2-8. Máquinas de estados de control*

El proceso comienza cuando se recibe un dato válido por la interfaz AXI4-Stream de entrada (*DATA_IN, T_VALID, T_RDY*), el proceso de envío y recepción de estas interfaces recae en la máquina encargada del control del bloque. Al recibir un nuevo dato, la máquina de control lo almacena en el registro de desplazamiento para su transmisión (*Tx_desp*) y cambia de estado e inicia el proceso de transmisión del comando por la interfaz serie, indicado con la señal *proc*.

Al terminar el proceso de envío, controlado por la máquina de estados representada en la Figura 2-10, se indica con la señal *fin_rx* y al detectarla, la máquina de control inicia el proceso de enviar el dato por la interfaz AXI4-Stream de salida (*DATA_OUT, R_VALID, R_RDY*).

Para el diseño de la máquina encargada de controlar la comunicación serie se realiza la primera aproximación utilizando los tiempos indicados en la hoja de características del RHD22xx [2, p. 15], indicados en la Tabla 2-2. Estos tiempos se han ajustado para trabajar con el reloj del sistema, $f_{clk} = 96\,MHz$. Como se define en las especificaciones el reloj de la interfaz serie trabajará a la frecuencia máxima permitida $f_{sclk} = 24\,MHz$, la cuarta parte de la frecuencia del sistema.

**$T_A$ = 25°C, $V_{DD}$ = 3.3V unless otherwise noted.**

| SYMBOL | PARAMETER | MIN | MAX | UNIT | COMMENTS |
|---|---|---|---|---|---|
| $t_{SCLK}$ | SCLK Period | 41.6 | | ns | Maximum SCLK frequency is 24 MHz |
| $t_{SCLKH}$ | SCLK Pulse Width High | 20.8 | | ns | |
| $t_{SCLKL}$ | SCLK Pulse Width Low | 20.8 | | ns | |
| $t_{CS1}$ | $\overline{CS}$ Low to SCLK High Setup | 20.8 | | ns | |
| $t_{CS2}$ | SCLK Low to $\overline{CS}$ High Setup | 20.8 | | ns | |
| $t_{CSOFF}$ | $\overline{CS}$ High Duration | 154 | | ns | |
| $t_{MOSI}$ | MOSI Data Valid to SCLK High Setup | 10.4 | | ns | |
| $t_{MISO}$ | SCLK or $\overline{CS}$ Falling Edge to MISO Data Valid | | 12 | ns | |
| $t_{CYCLE}$ | Total Cycle Time Between ADC Samples | 950 | | ns | Maximum sample rate is 1.05 MS/s, or 30 kS/s per channel for 35 multiplexed channels. |

*Tabla 2-2. Requisitos temporales para la comunicación con el RHD22xx [2, p.15].*

Para la creación de la máquina de estados, se ha realizado un diagrama temporal, Figura 2-9, cumpliendo los tiempos de la hoja de características al mismo tiempo que se realizan los cambios de las señales en función del reloj del sistema, *CLK*. En el diagrama se plantean los estados utilizados y a continuación se diseña la máquina de estados.

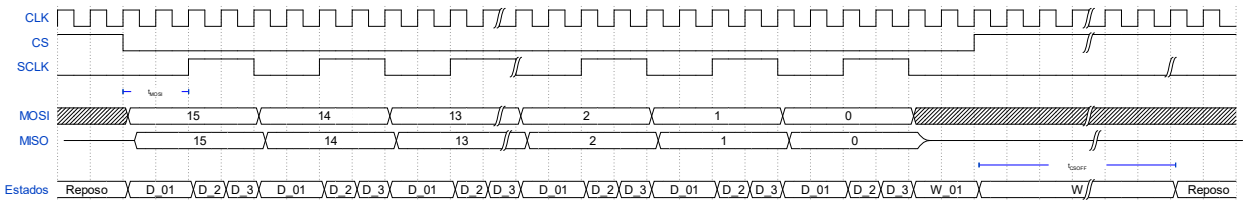


*Figura 2-9. Diagrama temporal de la interfaz Serie implementada*

Al comparar el diagrama temporal utilizado para la máquina de estados, Figura 2-9, el diagrama temporal de la hoja de características del RHD22xx mostrado en la Figura 2-3 y las restricciones temporales de la Tabla 2-2. Los únicos tiempos que han sido modificados son los indicados en la figura anterior:

- $t_{MOSI}$: Tiempo entre la aparición del dato en la señal *MOSI* y el primer flanco de subida de la señal *SCLK*, se ha ampliado. El dato se coloca en la señal *MOSI* 2 ciclos de reloj antes del flanco de subida de la señal *SCLK*, frente al ciclo de reloj indicado como tiempo mínimo en la hoja de características.
- $t_{CSOFF}$: Tiempo entre el intercambio de mensajes a través de la interfaz SPI. En la hoja de características se da un tiempo mínimo de 154 ns, al no ser un múltiplo exacto del periodo de *CLK*, se ha aproximado a 18 ciclos de reloj.

Para conseguir ajustar el tiempo de las señales, ha sido necesario registrar las señales, *cs*, *sclk* y *mosi* antes de ser enviadas por sus respectivos puertos, *CS, SCLK* y *MOSI*.

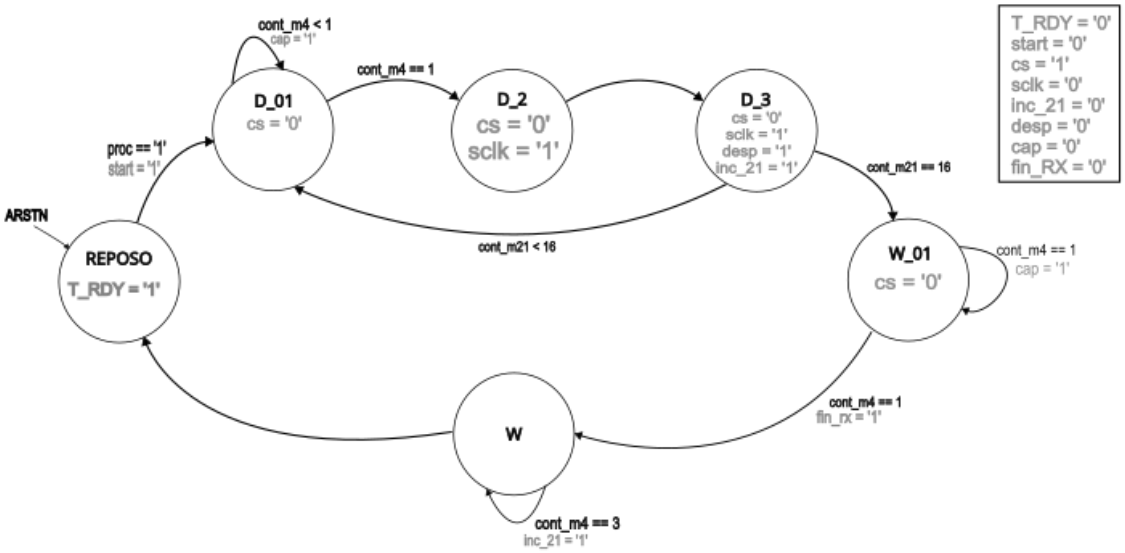


*Figura 2-10: Máquina de estados para el envío y recepción de la comunicación serie*

Tras almacenar el comando a enviar en el registro paralelo e indicar con la señal *proc*, desde la MEF de control, que la transmisión puede comenzar, la máquina encargada de la interfaz serie cambia del estado de reposo al primer estado, *D_01*. Este estado, con duración de 2 ciclos de reloj, mantiene la señal *sclk* en reposo y la señal *cs* a nivel bajo. En el segundo ciclo del estado se realiza la captura de la señal que entra por el puerto *MISO* a través de la señal *cap*, como la salida del puerto *SCLK* está retrasada se consigue capturar el dato en el último ciclo de la señal *clk* antes del flanco de bajada. En el siguiente estado, *D_2*, se cambia la señal *sclk* a un nivel alto. En el estado *D_3* se incrementa el contador módulo 21 usando la señal *inc_21*, encargado de contar el número de periodos de la señal *SCLK* que han tenido lugar y se activa la señal *desp* para desplazar el registro que contiene el comando (*Tx_desp*) y poder enviar el siguiente bit más significativo. Como la salida del registro de desplazamiento se encuentra registrada, el cambio de bit coincidirá con el flanco de bajada de la señal *SCLK*.

Este ciclo de 3 estados: *D_01, D_2* y *D_3*. Se repite 16 veces para capturar y enviar los 16 bits del comando y de la respuesta. Una vez terminado los 16 ciclos encontramos un estado *W_01* utilizado para cumplir los tiempos entre la señal *SCLK* y *CS*. En el siguiente estado, *W*, todas las señales se encuentran en el estado de reposo, sin embargo, es necesario hacer cumplir el tiempo entre el envío de comandos indicado en la hoja de características, durante este periodo se incrementa el contador módulo 21, que nos indicará el momento de finalización. Cuando el tiempo se ha cumplido la máquina indica con la señal *fin_rx* la finalización del proceso y vuelve al estado *REPOSO*.

Una vez generado el diseño, es necesario especificar una serie de restricciones temporales para asegurar una correcta comunicación en todo momento, si bien en la simulación no se tendrán en cuenta nos permitirán analizar si se cumplen los tiempos en etapas posteriores de la verificación.

#### *2.4.2.2 Bloque de inicialización*

Como se ha comentado en las especificaciones, el periférico debe ser capaz de realizar de manera independiente y automática el proceso de inicialización del RHD22xx. Para realizar este proceso se ha diseñado un bloque que se ha integrado dentro del periférico. Se entiende como proceso de inicialización el proceso que se ejecuta al iniciar el circuito integrado y lo prepara para su correcto funcionamiento según la configuración que el usuario introduce. Tras el proceso se realiza la copia de la memoria del RHD22xx en el periférico, obteniendo así una imagen de la memoria del RHD22xx a la que se puede acceder de manera paralela.

El proceso de inicialización básico definido en la hoja de características es la escritura de los 18 registros de configuración y realizar la calibración [2, p. 36]. Sin embargo, el proceso de inicialización que realiza el bloque está divido en cuatro partes y está basado en el ejemplo presentado de la hoja de características del RHD22xx:

1. Inicio de la configuración
2. Envío de la configuración
3. Calibración
4. Lectura de los registros

En un primer momento se realiza el envío de 2 comandos de lectura del registro 63 con los que se busca iniciar la comunicación. El registro es de solo lectura, por lo que contiene un valor fijo conocido.

A continuación, se realiza la escritura de todos los registros de configuración. El usuario ha introducido la configuración deseada en los 8 bits menos significativos de los registros 0 a 17 del periférico. Estos registros se encuentran conectados al bloque y son enviados al circuito integrado.

Tras la escritura de la comunicación se realiza el proceso de calibración. El bloque envía el comando **CALIBRATE()** seguido de 9 comandos de lectura, **READ(63)**, que no tienen ningún efecto, solo se utilizan para completar el proceso de calibración.

Para terminar el proceso, se realiza una lectura de todos los registros del circuito integrado con el objetivo de almacenar su contenido en los respectivos registros del periférico. De esta manera, cuando el usuario acceda a los registros del periférico tendrá acceso al contenido de los registros del circuito integrado y se podrá verificar si el proceso se ha realizado correctamente. Los registros de configuración se sobrescriben con la configuración que se ha escrito en los registros del RHD22xx, que deberá ser la misma si no hay errores.

Al finalizar, debido a la forma de comunicación del integrado, explicada en secciones anteriores. Se añaden 3 comandos de lectura de relleno para poder leer el contenido de todos los registros.

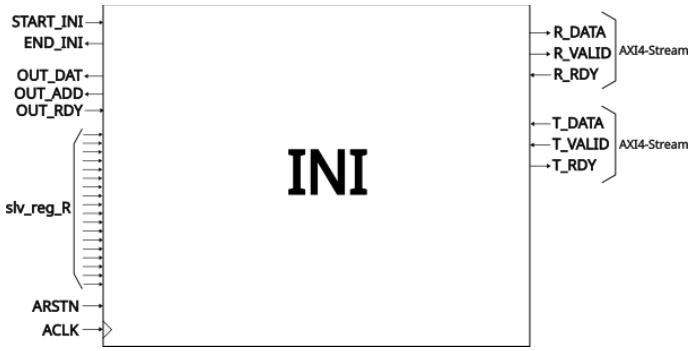


*Figura 2-11. Bloque Inicialización*

Para comunicarse con el resto del periférico, como se ve en la Figura 2-11, el bloque cuenta con 2 interfaces AXI4-Stream para la comunicación con el bloque SPI para la transmisión (*T_DATA, T_VALID, T_RDY*) y recepción (*R_DATA, R_VALID, R_RDY*), 2 señales conectadas al registro de control para iniciar el proceso (*START_INI*) e indicar su finalización (*END_INI*), 3 señales utilizadas para guardar las respuestas de las lecturas en los registros indicados (*OUT_DAT, OUT_ADD, OUT_RDY*) y utiliza también 18 señales conectadas a los 8 bits menos significativos de los respectivos registros de configuración para su lectura (*slv_reg_R*). Finalmente utiliza la señal de reloj del sistema (*ACLK*), a 96 MHz, y una señal de reset síncrono a nivel bajo (*ARSTN*).

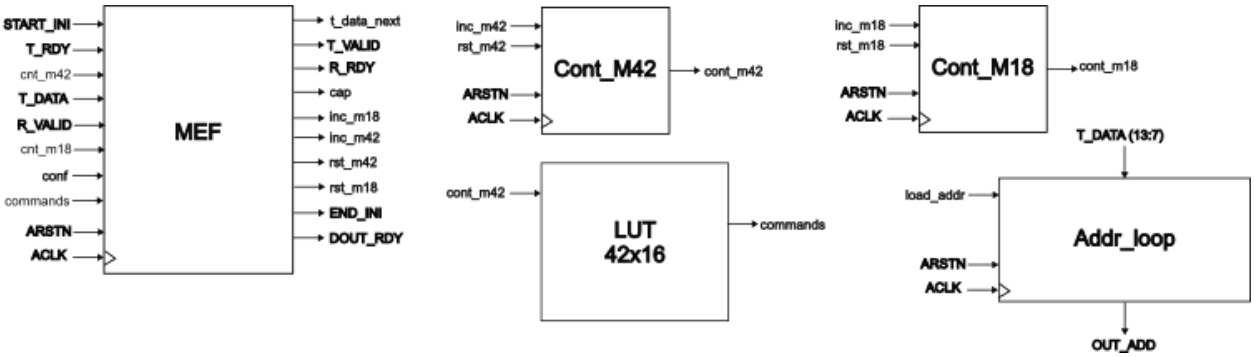


*Figura 2-12. Componentes principales que forma el bloque INI*

Los comandos que se enviarán se almacenan en una LUT (*Look-Up Table*) de 42 palabras de 16 bits, los únicos comandos no almacenados son la escritura en los registros de configuración. Se utiliza un contador módulo 42 como puntero de la LUT. Además, como se puede ver en la Figura 2-12, se utiliza un contador módulo 18 para controlar el envío de los registros de configuración. Los contadores, junto a una máquina de estados finitos, permiten controlar todo el proceso de inicialización. También se utilizan 3 registros de carga paralela de 6 bits para ajustar la dirección con el contenido del registro en el momento de lectura.

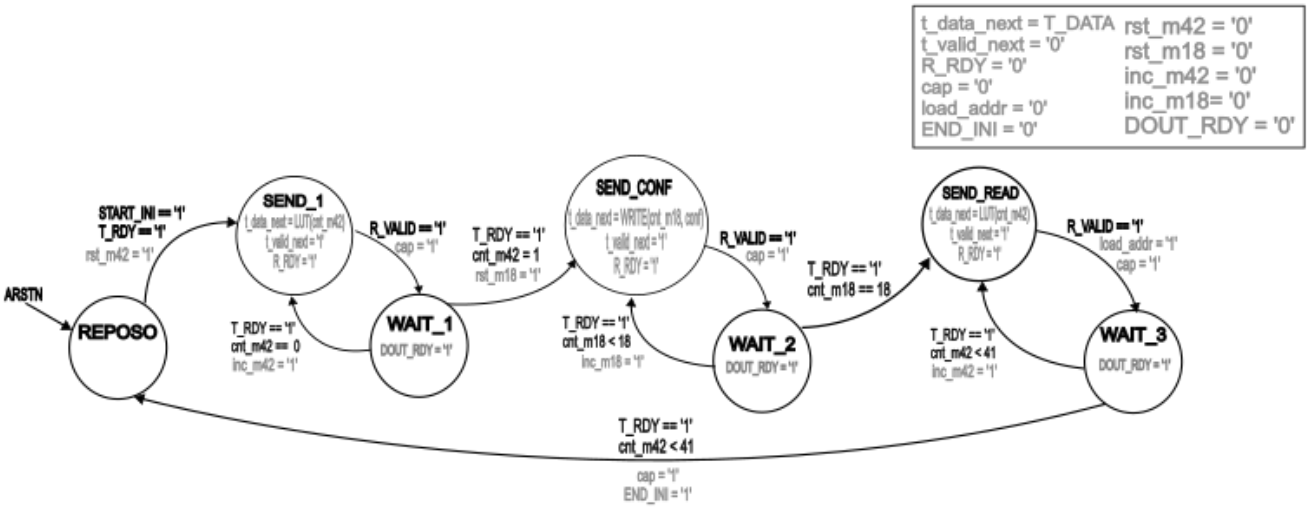


*Figura 2-13. Máquina de estados del proceso de inicialización*

La máquina sale del estado *REPOSO* en el momento que llega la señal *START_INI* y el bloque SPI está preparado para recibir comandos. En el siguiente estado, *SEND_1*, comienza el envío del primer comando de lectura, **READ(63)**, colocándolo en la señal *t_data_next* que sirve de entrada a un registro conectado a *T_DATA*. Tras indicar que el dato recibido del bloque SPI es válido con *t_valid_next*, conectado a otro registro que tiene como salida *T_VALID*, se pasa al estado *WAIT_1*. En este estado se decide el comando a enviar a continuación: si todavía no se han enviado 2 comandos de lectura se vuelve al estado *SEND_1* y se incrementa el puntero de la LUT, de lo contrario se pasa al estado *SEND_CONF* para el envío de la configuración del integrado.

El envío de la configuración se realiza por medio del comando **WRITE(CNT_M18, CONF)**, la dirección de escriturar viene dada por el contador módulo 18 mientras que el contenido a escribir viene dado por las señales *slv_reg_R*. El proceso de envío de la configuración es similar al explicado para los comandos de lectura, exceptuando la condición para el cambio de estados que se realiza en función del contador módulo 18.

Al terminar la escritura de la configuración se procede al envío de todos los comandos almacenados en la LUT desde la última posición del puntero, para ello se utilizan los estados *SEND_READ* y *WAIT_3* con un funcionamiento similar a las parejas de estados anteriores. El primer comando enviado es **CALIBRATE()**, seguido del envío de 9 comandos de lectura, **READ(63)**, utilizados para hacer llegar la señal de reloj al circuito y completar la calibración. Tras el envío de cada comando, en el estado *WAIT_3* se incrementa el puntero de la LUT.

Los siguientes 27 comandos almacenados en la LUT son comandos de lectura, **READ(addr)**, para realizar la lectura de todos los registros del circuito integrado. A la hora de almacenar el contenido del registro del integrado en el registro indicado del periférico el bloque envía parejas de datos y dirección del registro al que corresponde por las señales *OUT_DAT* y *OUT_ADD*. Conseguir que las parejas sean correctas supone un problema debido el retraso de 3 comandos en la respuesta del RHD22xx. Para solucionarlo se implementa un sistema de 3 registros de carga paralela conectados en serie (*Addr_loop*). Cada vez que se envía un comando de lectura, el campo de dirección de este se almacena en el primer registro, se carga con la señal

*load_addr*. La salida del tercer registro se conecta a la salida del bloque *OUT_ADD*. De esta manera, al retrasar la salida de la dirección del comando durante el envío de 3 comandos se consigue que la pareja dirección – comando en los puertos *OUT_ADD* y *OUT_DAT* sea la correcta. Debido a este retraso, al final de la LUT es necesario añadir 3 comandos de lectura del último registro para conseguir recibir el contenido de todos los registros. En la Figura 2-14 se muestra el diagrama temporal de las señales intermedias del bloque.

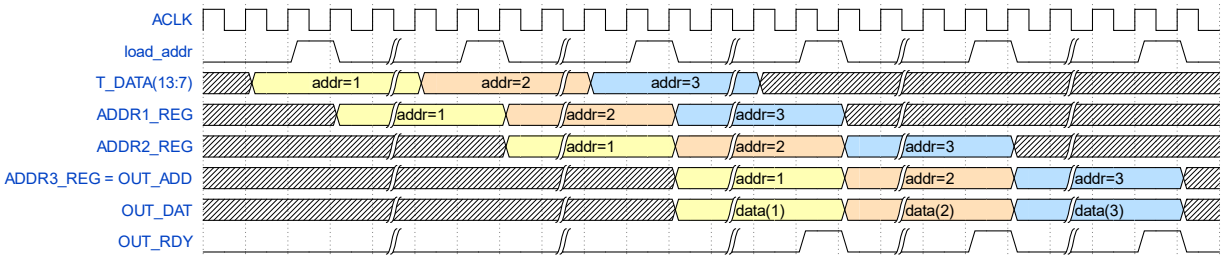


*Figura 2-14. Diagrama temporal del retraso de la dirección con el bloque ADDR_LOOP*

Mientras se realiza la lectura de los registros, el bloque devuelve al periférico parejas de datos con la dirección del registro, en las señales *OUT_DAT* y *OUT_ADD*. Cuando la pareja es válida el bloque dispara la señal *OUT_RDY* para indicarlo. Desde el periférico se almacena el dato recibido en su correspondiente registro.

#### *2.4.2.3 Bloque de comunicación directa*

Para el envío de mensajes al circuito integrado de manera directa, que se envían sin modificar y no pertenecen a ningún proceso del periférico, se ha diseñado el siguiente proceso.

A diferencia de los anteriores bloques, este ha sido diseñado directamente sobre el periférico debido a su simplicidad ya que, entendemos por comunicación directa el envío de un comando y la recepción de la respuesta inmediata.

El proceso hace uso de 2 registros del periférico sobre los que lee el contenido de los comandos (dirección @0x48) y escribe las respuestas (dirección @0x4C).

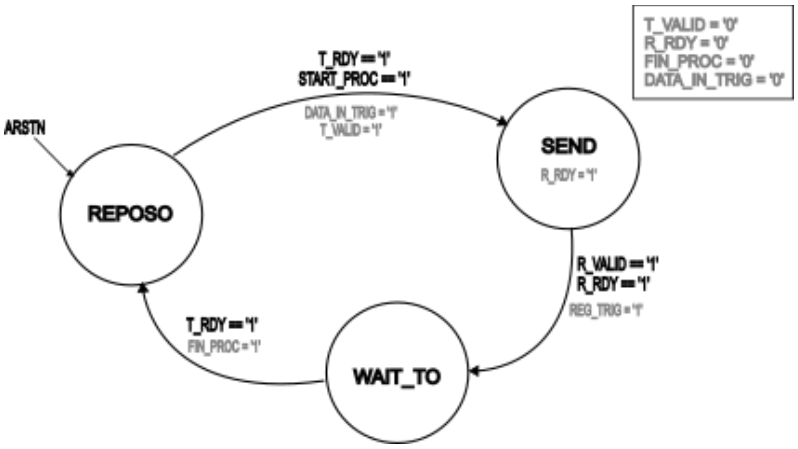


*Figura 2-15. Máquina de estados del proceso de comunicación directa*[2]

El bloque es controlado por una máquina de estados finitos que se encarga de gestionar el envío y recepción del comando almacenado en el registro y de su posterior respuesta a través de las interfaces AXI4-Stream del bloque SPI.

#### *2.4.2.4 Bloque de conversión de canales (CH_READER)*

Como se ha definido en las especificaciones, es necesario implementar un proceso que permita el muestreo de los canales seleccionados por el usuario de forma automática. El proceso se implementa en este bloque, aunque también se realizan ciertas funciones desde el periférico.

El circuito integrado cuenta con 32 canales de conversión más varios canales y sensores auxiliares, en este proceso solo se tratará la conversión de los 32 canales. Para la selección de estos hacemos uso de uno de los registros del periférico (*CHSel*, *@0x54*). El registro consta de 32 bits asociado cada uno al canal de la posición en la que se encuentra el bit: el bit 0 corresponde al canal 0, el bit 1 corresponde al canal 1, el bit 2 corresponde al canal 2 …


*Figura 2-16. Bloque conversión de canales (CH_READER)*

[2] Al ser implementada sobre el periférico directamente, todas las señales se escriben en mayúscula.

Como se ve en la Figura 2-16, el bloque cuenta con 2 interfaces AXI4-Stream para comunicarse con el bloque SPI, 2 señales para el control del bloque (*START_CH*, *END_CH*), un bus de 32 bits conectado al registro de selección de canales (*CHANNELS*), 3 señales que devuelven el número de canal (*CHR_ADD*), el resultado de la conversión (*CHR_DATA*) y la validez de la pareja de dato y canal (*CHR_RDY*). Se incluye una señal que permite activar o desactivar la eliminación de offset en las conversiones (*OFF_REM*). Finalmente utiliza la señal de reloj del periférico (*ACLK*) y la señal de reset del mismo (*ARSTN*).

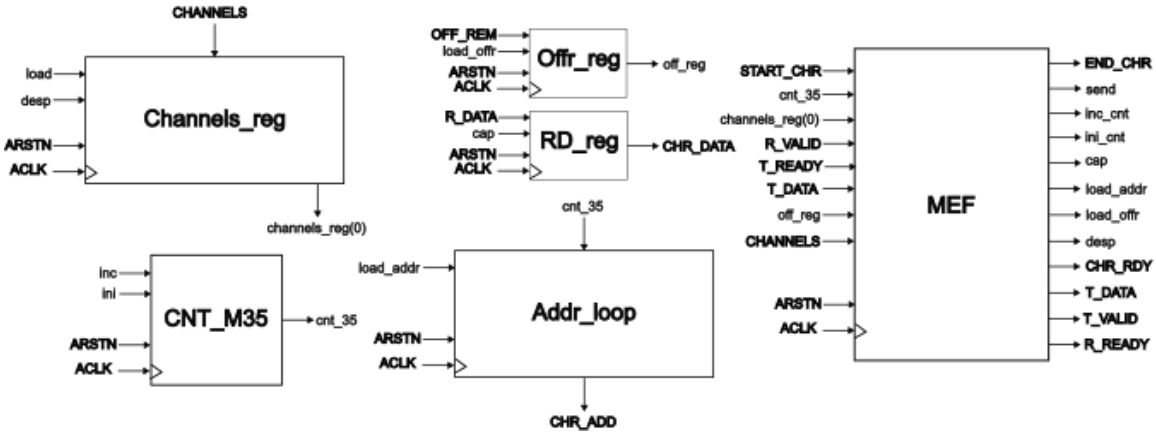


*Figura 2-17. Bloques principales que forma el bloque CH_READER*

Cabe destacar, como se aprecia en la Figura 2-17, el uso de un registro de desplazamiento de 32 bits para almacenar la selección de canales, un contador módulo 35 que, junto a la máquina de estados finitos de la Figura 2-18, se encargan del control y una serie de 3 registros de carga paralela. También se utilizan dos registros encargados de almacenar los datos recibidos de la interfaz SPI, *RD_reg,* y de almacenar la configuración del bit de Offset Removal*, Offr_reg*.

Como se ve en la Figura 2-18 el proceso se inicia al indicarlo la señal *START_CH*, en ese momento el contenido del registro de selección de canales se carga en un registro de desplazamiento (activado por la señal *load*), se almacena en otro registro (offr_reg) el bit de *offset removal* y se reinicia el retraso de las direcciones de salida y el contador con las señales *load_addr* e *ini* respectivamente. En el estado *CHECK* se evalúa el bit menos significativo del registro de canales (*channels_reg(0)*) que, en caso de tomar el valor 1 indicará que el canal asociado a ese bit debe ser muestreado. El contador módulo 35 indica el número del canal que se está muestreando, se incrementa tras cada iteración del estado *CHECK* para indicar el número de canal que se quiere muestrear. En el estado *SEND_CH* se realiza el envío del comando, como número de canal se utiliza el contador módulo 35, además se utiliza el bit de *off_reg*. El estado *WAIT_1* se utiliza para almacenar la respuesta recibida.

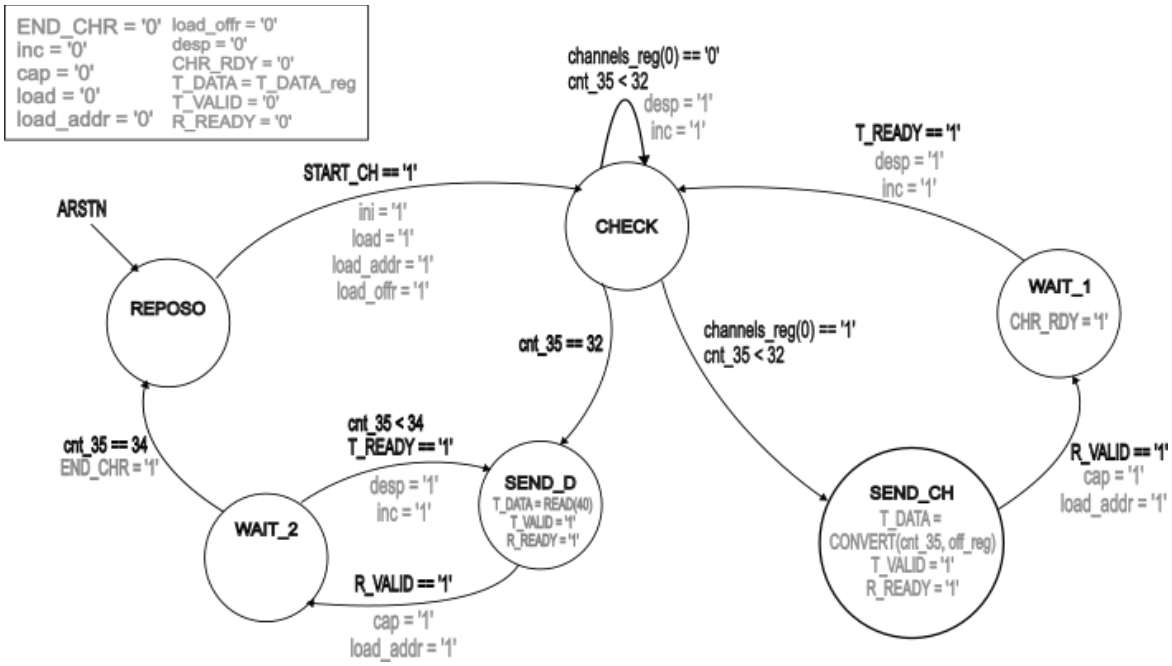


*Figura 2-18. Máquina de estados del proceso de conversión de canales*

Este proceso se repite con los 32 bits del registro. Al llegar al último bit, como ha ocurrido en el proceso de inicialización debido al retraso en la respuesta a los comandos, es necesario el envío de comandos de lectura para conseguir obtener la respuesta a todos los comandos de conversión. El proceso para conseguir parejas de datos y direcciones del canal a la salida del bloque se ha realizado de la misma forma que en el proceso de inicialización, por medio de 3 registros de carga paralela conectados en serie (activados por la señal *load_addr*).

En el caso de que se active la eliminación de offset en las conversiones, esta se aplica en todos los canales. Esto supone que el bloque modifica el comando enviado al SPI, el LSB (*Least Significant Bit*) cambia a 1 en caso de activarse.

Fuera del bloque también se han implementado distintos procesos para poder almacenar los resultados de las conversiones y para realizar conversiones de manera periódica, se detalla en los siguientes apartados.

Como los resultados de las conversiones tienen una longitud de 16 bits, para facilitar el acceso y reducir el número de registros utilizados, se ha decidió almacenar la conversión de 2 canales en cada registro. Los 16 bits menos significativos corresponden a los canales pares, mientras que los más significativos corresponden a los impares. Los registros tienen direcciones sucesivas.

### 2.4.3 Control

Todo el control del periférico se realiza por medio de la escritura de uno de los registros, el registro *Status (@0x50)*. Este registro se comunica con el bloque de control del periférico.

En la Figura 2-19, se detalla el contenido del registro de control.

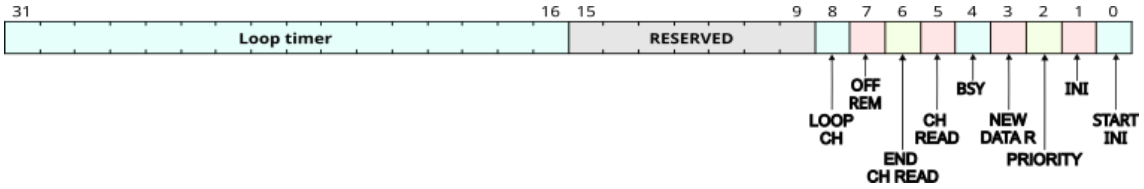


*Figura 2-19. Diagrama registro de Estado y control*

Los bits 0 y 1 están asignados al proceso de inicialización. El bit 0 (*START INI*), de escritura, al ser activado inicia el proceso. Durante todo el proceso el bit 1 (*INI*) se mantiene en nivel alto, vuelve a 0 al finalizar el proceso.

Los bits 2 y 3 están asociados al proceso de comunicación directa. El bit 2 (*PRIORITY)* es de escritura y al ser activado inicia el proceso. Al finalizar el proceso el bit 3 (*NEW DATA R*) se activa indicando que hay una nueva respuesta. Este bit está conectado a una de las interrupciones.

El bit 4 (*BSY*) es de sólo lectura y se encarga de indicar cuando el periférico se encuentra procesando.

Los siguientes 4 bits: *CH READ, END CH READ, OFF REM* y *LOOP CH*. Están asociados al proceso de conversión de canales. Siendo el primero de estos el encargado de iniciar el proceso al ser activado. El segundo indica cuando termina el proceso de conversión y los nuevos resultados están almacenados en los registros, además está conectado a la interrupción. El tercer bit se activa y desactiva el *offset removal* de la conversión de los canales. El cuarto bit se encarga de indicar que el bucle de conversión de canales está activo.

Para indicar el periodo del bucle de conversión se utilizan los 16 bits más significativos del registro. Se debe introducir el periodo en función de los ciclos de reloj del sistema lo que permite un periodo de muestreo máximo

$$T_{s\,max} = 2^{16} \cdot T_{clk} = 2^{16} \cdot 10.4\ (ns) = 681.6\ (\mu s)$$

*Ecuación 2. Cálculo del periodo máximo del bucle de conversión de canales*

La información sobre el registro de control se encuentra ampliada en la documentación del periférico que se incluye en los anexos.

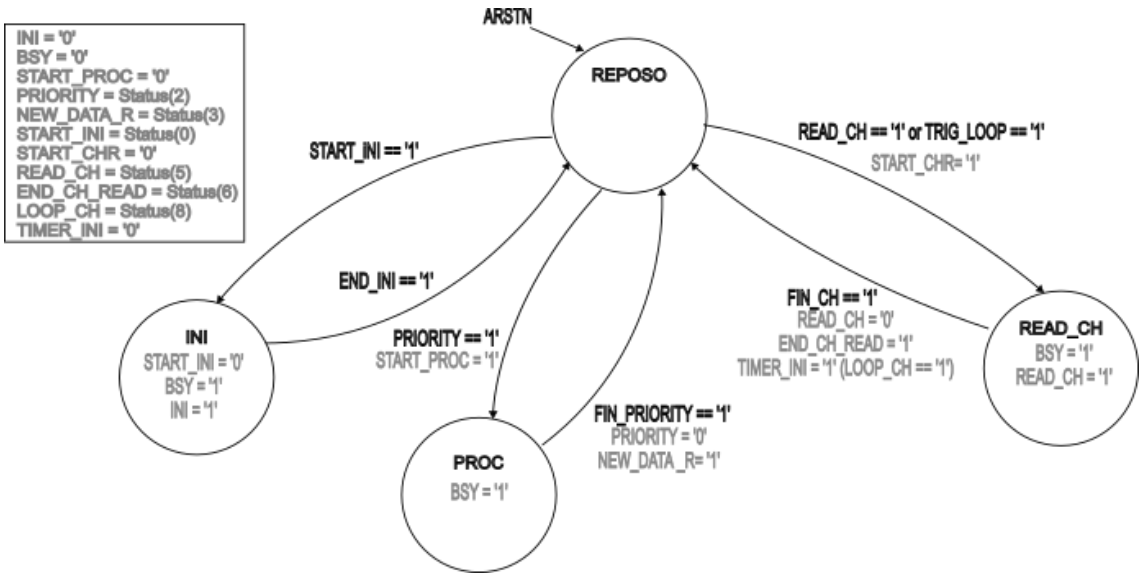


*Figura 2-20: Máquina de estados de control del periférico*[3]

La MEF encargada del control, Figura 2-20, se comunica con el registro *Status (@0x50)* en todo momento, utilizando sus bits como entradas y sobrescribiendo los bits del registro cuando es necesario. Al seleccionar cada proceso la MEF entra en el estado indicado, manteniendo las señales de estado en el valor indicado. Las señales *START_INI*, *START_PROC* y *START_CHR* son las encargadas de iniciar el proceso en sus respectivos bloques. Siempre que la MEF entra en el estado correspondiente a un proceso, se activa la señal *BSY* para indicar que el periférico está procesando.

Las interrupciones están conectadas a los bits del registro de estado. Cabe resaltar que estos bits no vuelven a su estado predeterminado tras la lectura de los registros que almacenan la respuesta al comando con prioridad o los resultados de la conversión de los canales. Es necesario reiniciarlas escribiendo sobre el registro *Status (@0x50)* desde la interfaz AXI4-Lite.

[3] Como se implementa directamente en el periférico, no se ha diferenciado entre señales internas y puertos del bloque.

# 3 Verificación

## 3.1 Simulación de los bloques

Para asegurar el correcto funcionamiento de los bloques, estos han sido sometidos a diferentes bancos de pruebas. Otra de las ventajas que presenta la implementación de los distintos bloques por separado es la capacidad de probar por separado los distintos procesos, permitiendo una verificación más exhaustiva. Todas las pruebas de simulación se han realizado en la herramienta de simulación HDL incluida en Vivado 2018.2 [7].

Para poder realizar las pruebas simuladas de una manera más cercana a la realidad, todas las pruebas se han realizado utilizando el bloque VHDL *RHD2000_fake.vhd* [8]. Este bloque modela el comportamiento del RHD22xx permitiendo respuesta a los comandos de lectura y escritura, calibración y conversión de canales. Las respuestas que el bloque presenta frente a los distintos comandos han sido modificadas para poder facilitar la verificación de los comandos enviados:

- En la versión original del bloque, la respuesta al comando de lectura presenta una respuesta distinta a 0 sólo en los casos que se accede a los registros de sólo lectura (registros 40-44 y 60-63), tras la escritura en el resto de registros el bloque actualiza su valor. Para las pruebas del sistema completo se modifica, devolviendo como respuesta el número de registro como respuesta en todos los casos para comprobar la lectura de todos los registros sin necesidad de escribir antes en ellos.
- En el caso del comando de escritura, la respuesta presenta la misma estructura que el integrado original: Los 8 bits más significativos a nivel alto seguidos de los datos que se querían escribir. El bloque que emula el periférico almacena los datos escritos y devolverá la respuesta en caso de escribir sólo en los registros de configuración (registros 0-17).
- Para el proceso de conversión, en un primer momento, el bloque devolvía el resultado del muestreo de una señal sinusoidal almacenada en una LUT. Sin embargo, este comportamiento ha sido modificado para que la respuesta solo contenga el número del canal a convertir con lo que se consigue comprobar de una manera simple si se está convirtiendo el canal que interesa. Se modifica también el resultado al indicar que se ha activado la eliminación de offset en la conversión, esto se notifica modificando los bits 10 bits más significativos de la respuesta del canal.

El primer bloque verificado es la interfaz SPI. En las primeras versiones se prueba de manera separada al periférico, Figura 3-1, introduciendo los vectores de prueba a través de la interfaz AXI4-Stream. Los comandos enviados son peticiones para leer los 5 registros de sólo lectura que contienen, en código ASCII, las letras *INTAN*. Tras varias pruebas se consiguen ajustar el correcto funcionamiento de la máquina de estados y se ajustan pequeños fallos en las señales. Por medio de la simulación también se consiguen asegurar que se cumplen los tiempos marcados en la hoja de características del RHD22xx [2], Figura 1-7.

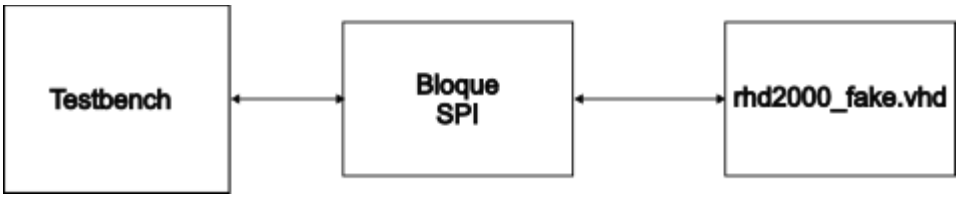


*Figura 3-1. Diagrama de bloques del banco de pruebas del bloque SPI*

Para terminar de verificar el bloque de la interfaz serie, se realiza una versión del banco de pruebas con la capacidad de ser sintetizable, código HDL capaz de ser traducido a un mapa de puertas lógicas, de esta manera se prueba el bloque sobre hardware real permitiendo la valoración y verificación de los tiempos de la interfaz. Este procedimiento se detalla en la sección Pruebas sobre hardware.

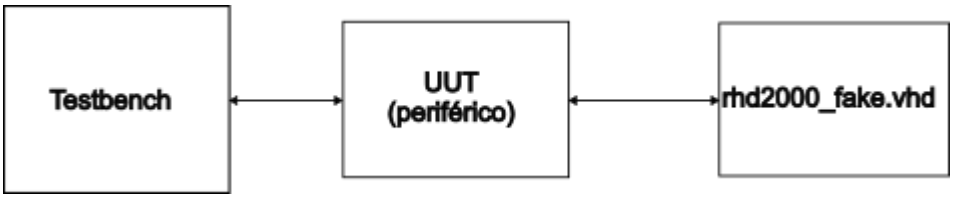


*Figura 3-2. Diagrama de bloques del banco de pruebas de la comunicación directa*

Una vez verificado el bloque de la interfaz, este se añade al periférico y se procede a la prueba del siguiente bloque, la comunicación directa. Como se describe en la Figura 3-2 la UUT (*Unit Under Test*) es el periférico completo por lo que el banco de pruebas genera las señales necesarias para la comunicación a través de la interfaz AXI4-Lite. También se utiliza el módulo que emula el comportamiento del INTAN RHD22xx. Dentro del banco de pruebas se implementan los siguientes procesos encargados de la generación de estímulos actuando como interfaz AXI4-Lite maestra:

- **ADDR_WRITE**: Encargado de la generación de estímulos para el canal de dirección de escritura de la interfaz AXI4-Lite. Se envían 2 comandos de escritura desde el banco de pruebas por lo que se generan estímulos con las dos direcciones, el registro *DirectCMD (@0x48)* y el registro *Status (@0x50)*.
- **DATA_WRITE**: Encargado de la generación de estímulos para el canal de datos de escritura de la interfaz AXI4-Lite. Se escribe sobre 2 registros, primero se añade el comando a enviar al integrado y después se indica iniciar el proceso de comunicación directa sobre el registro de control, *Status (@0x50)*.
- **ADDR_READ**: Genera los estímulos del canal de dirección de lectura. Se realizan dos operaciones de lectura, la lectura del registro de control, *Status* (*@0x50*) y la del registro de respuesta de la comunicación directa, *DCResp (@0x4C).*
- **DATA_READ***:* Genera los estímulos del canal de datos de lectura. Se encarga de la generación de la señal *READY* para el proceso de *handshake* del canal.
- **B_PROC***:* Encargado de la generación de la señal *READY* para el handshake del canal de respuesta de escritura.
- **CLK_PROC***:* Encargado de la generación de la señal de reloj del sistema ($f_{clk} = 96\ MHz$).
- **RST_PROC***:* Encargado de la generación de la señal de reset activa a nivel bajo.

En una primera versión del banco de pruebas se envían 4 instrucciones: primero se escribe en el registro *DirectCMD (@0x48)* el comando a enviar; se indica en el registro de control, *Status (@0x50),* que hay un comando con prioridad; se lee el registro *Status (@0x50)* para comprobar el funcionamiento de los bits de estado y finalmente se lee el contenido del registro de respuesta *DCResp (@0x4C)*.

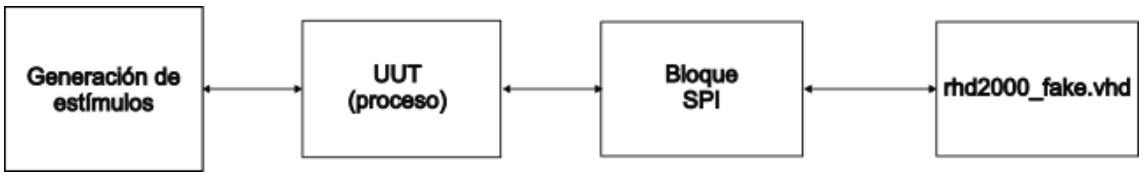


*Figura 3-3. Diagrama de bloques del banco de pruebas para la verificación de los procesos de manera independiente*

Para comprobar el bloque de inicialización se implementa un banco de pruebas por separado en el que se conecta a los estímulos directamente con los puertos del bloque como se indica en la Figura 3-3 Se utiliza además el bloque SPI y el bloque que emula al circuito integrado para comprobar el funcionamiento. Los registros de configuración del periférico se sustituyen por señales que contienen el número de cada registro.

Al incluir el bloque en el periférico para realizar la verificación del conjunto bloque y periférico, encontramos la dificultad de ajustar los tiempos de todas las señales de la interfaz AXI4-Lite correctamente para escribir en los distintos registros. Para solucionarlo se decide diseñar un nuevo banco de pruebas, Figura 3-4, basado en el descrito en el utilizado para la prueba de la comunicación directa añadiendo un proceso y modificando la generación de estímulos:

- **PROC_SEQUENCER**: Proceso encargado de la lectura de un fichero que contiene las instrucciones para la generación del estímulo deseado. Utiliza el paquete *std.textio*.
- **ADDR_WRITE, DATA_WRITE, ADDR_READ**: El estímulo a generar viene dado por el proceso anterior, que lo traduce del fichero con las instrucciones.
- **DATA_READ, B_PROC, CLK_PROC, RST_PROC**: Los procesos se mantienen igual que en el banco de pruebas de la comunicación directa.

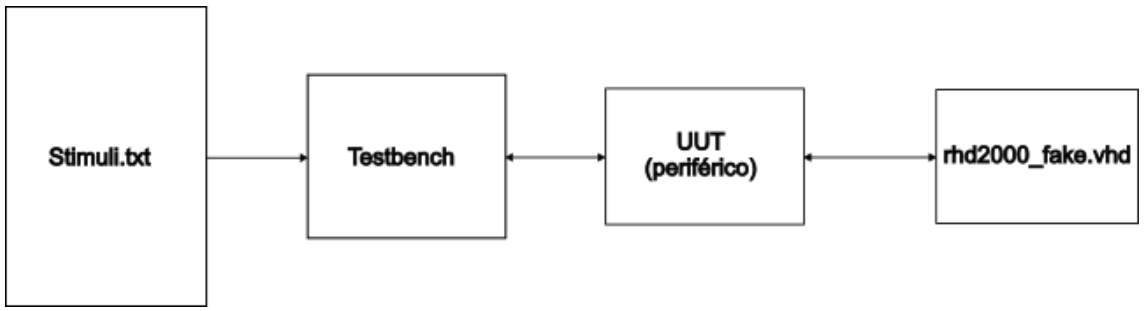


*Figura 3-4. Diagrama de bloques del banco de pruebas del periférico utilizando un fichero para la introducción de comandos.*

El paquete de VHDL *std.textio* forma parte de la librería *std*, permite la lectura y escritura de ficheros de texto ASCII desde la simulación del banco de pruebas. Desde el banco de pruebas se realiza la lectura del fichero, que contiene la información del comando a ejecutar, y la generación de las señales de la interfaz AXI4-Lite. Se han diseñado una serie de ficheros que contienen las instrucciones para ejecutar y comprobar las distintas funciones del periférico. Todos los ficheros mantienen la misma estructura y contienen las instrucciones a enviar al periférico emulando el comportamiento del procesador.

Las instrucciones del fichero de texto siguen el siguiente formato:

| TYPE | ADDR | DATA |
|---|---|---|

*Tabla 3-1. Formato de las instrucciones del fichero Stimuli.txt*

El campo *Type* permite indicar si el comando a enviar es de lectura (R) o escritura (W) en la interfaz AXI4-Lite. De esta manera el banco de pruebas selecciona las señales a generar.

A continuación, el campo *ADDR* indica la dirección del registro a leer o escribir. Está formado por una cadena de 6 bits.

Para finalizar, en caso de ser un comando de escritura, *DATA* indica el contenido a escribir en el registro. Debe ser una cadena en binario de 32 bits. En caso de ser un comando de lectura, este campo quedará en blanco.

Los distintos campos se separan por un espacio en blanco. También se permite el uso de comentarios comenzando la linea con el símbolo almohadilla (#). Los ficheros comienzan con una pequeña cabecera indicando el formato y todos son nombrados de la forma *Stimuli_NombreDelProceso.txt*

Este formato de banco de pruebas permite la simplificación de la descripción de los estímulos a generar, aunque no se ha implementado una instrucción para mantener el canal en reposo durante un determinado tiempo y poder volver a continuar el envío de estímulos. Esto hace que sea necesario generar estímulos hasta la finalización de los procesos para poder leer el resultado que nos interesa.

Para la prueba del proceso de inicialización, el fichero de instrucciones comienza escribiendo en el registro de control el inicio del proceso. Este comando lo siguen una serie de comandos de lectura del mismo registro para mantener la comunicación hasta que el proceso finalice. Al finalizar el proceso se leen los distintos registros del periférico que contienen la información de sólo lectura del RHD22xx. También se leen distintos registros con el objetivo de comprobar la corrección de errores.

Para la verificación del bloque encargado de la conversión de canales se ha seguido el mismo procedimiento que los anteriores bloques, un banco de pruebas del bloque por separado seguido de una prueba del periférico.

En el caso de la prueba del bloque la estructura del banco de pruebas se describe en la Figura 3-3, utilizando como UUT el bloque *CH_READER*. Se repite este banco de pruebas con distintos vectores de canales, de manera pseudoaleatoria, con el objetivo

de cubrir el mayor número de configuraciones posibles. Con este banco de pruebas se asegura el correcto funcionamiento de las señales de salida del bloque, que contienen la dirección y los datos de cada conversión.

En el caso del banco de pruebas del periférico, se realiza el mismo proceso que para el bloque, pero mediante la escritura en los registros como se indica en la Figura 3-4. Al finalizar el proceso se realiza la lectura de los 16 registros que contienen el resultado de la conversión. El objetivo de este banco de pruebas es la comprobación del correcto almacenamiento de los resultados de la conversión.

Además, se realiza otro fichero de texto que contiene las instrucciones para realizar la prueba de la conversión de canales de manera periódica. De esta manera quedan verificados los dos procesos asociados al bloque de conversión de canales.

Tras la implementación de este formato de banco de pruebas, se decide volver a probar la comunicación directa generando un fichero de instrucciones para el proceso.

Al completar las distintas pruebas de los bloques por separado y de los distintos procesos desde el periférico, se procede a realizar pruebas por simulación del sistema completo, contando con una versión simulada del microprocesador.

## 3.2 Simulación del sistema completo

Para la simulación del sistema completo es necesario la utilización de distintas herramientas. Se ha creado un proyecto de Vivado [9] que contiene un Microcontrolador (MCU). Este MCU contiene un Cortex-M1 de ARM como CPU y se le añaden distintos periféricos, entre ellos el descrito en el proyecto. Las instrucciones que ejecutar son escritas en código C utilizando el programa KeiluVision. Para ello, se define la estructura de los registros del periférico [10], permitiendo así el acceso a estos desde el código.

Para simular el programa diseñado en un simulador HDL como el incluido en Vivado, es necesario realizar el siguiente proceso, que comienza con la programación del fichero:

- Desde el software KeiluVision [18] se realiza la *build* del programa a ejecutar en el microprocesador.
- El juego de instrucciones, en formato AXF (*Arm eXecutable Format*) se convierte utilizando la herramienta *fromelf.exe* incluida en la suite KeiluVision en formato hexadecimal orientada a bytes (formato de memoria de *Verilog*) [11]. Esto nos permite cargar las instrucciones en los modelos de memoria del simulador HDL.
- Finalmente se lanza la simulación desde Vivado 2018.2.

Para facilitar las distintas pruebas de los procesos se han implementado diversas funciones que se encargan de realizar todo lo necesario para cada proceso. Estas funciones se adjuntan en el anexo Anexo C. Funciones en código C para la verificación. En la Figura 3-5 y en la Figura 3-6 se incluyen los diagramas de flujo de las distintas funciones.

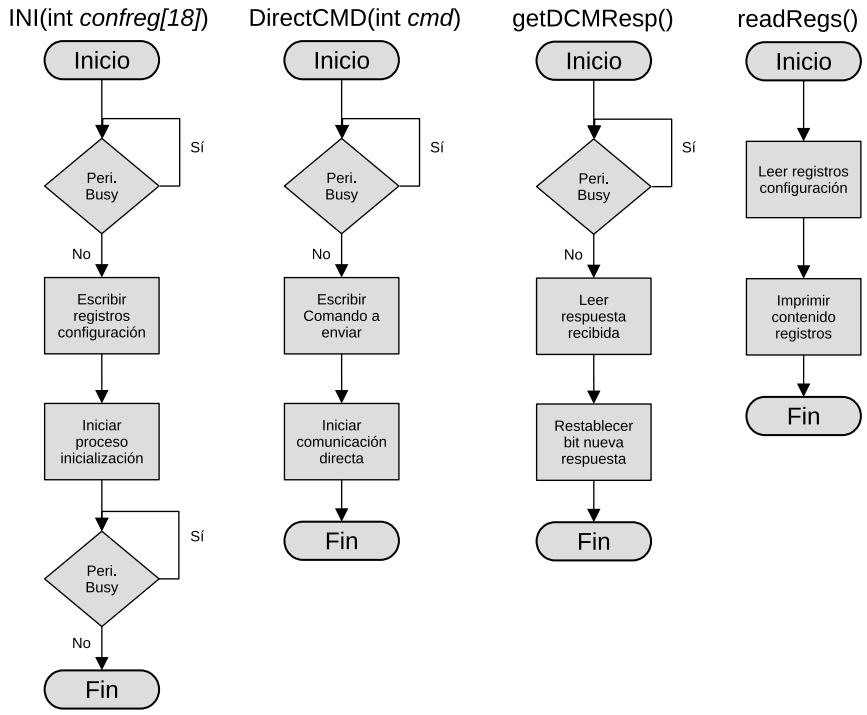


*Figura 3-5: Diagrama de flujo de las funciones implementadas I*

El primer proceso en ser verificado de esta manera es el proceso de inicialización. Para ello se implementa una función, *INI(int confreg[18])*, encargada de escribir la configuración los registros del periférico correspondientes y lanzar el proceso. Como se comenta en el apartado Simulación de los bloques, el bloque *RHD2000_fake.vhd* se modifica para devolver un valor por defecto en cada uno de los registros tras un comando de lectura. Esto nos permite comprobar la lectura de todos los registros. También se implementa una función, *readRegs(),* que permite leer el contenido de los 18 primeros registros del periférico *(@0x00 - @0x44*). Estos registros corresponden a los registros de configuración del RHD22xx.

Para la comprobación del bloque de comunicación directa se implementan dos funciones encargadas de la escritura del comando e inicio del proceso, *DirectCMD(int cmd)*, y de la lectura de la respuesta, *getDCMResp()*. La función encargada de la lectura de la respuesta también se encarga de volver a poner a nivel bajo el bit de nueva respuesta, esto reinicia también la interrupción. Este proceso de reiniciar el bit indicador de nueva respuesta y la interrupción no se realiza de manera automática desde el periférico, si no que se debe hacer desde el programa corriendo en el microprocesador.

Para la verificación se envían 3 comandos de lectura de uno de los registros de sólo lectura, en concreto el registro 40, y se procede a la lectura de las 3 respuestas. Dado el funcionamiento del circuito, la respuesta correcta llegará junto al envío del tercer comando.

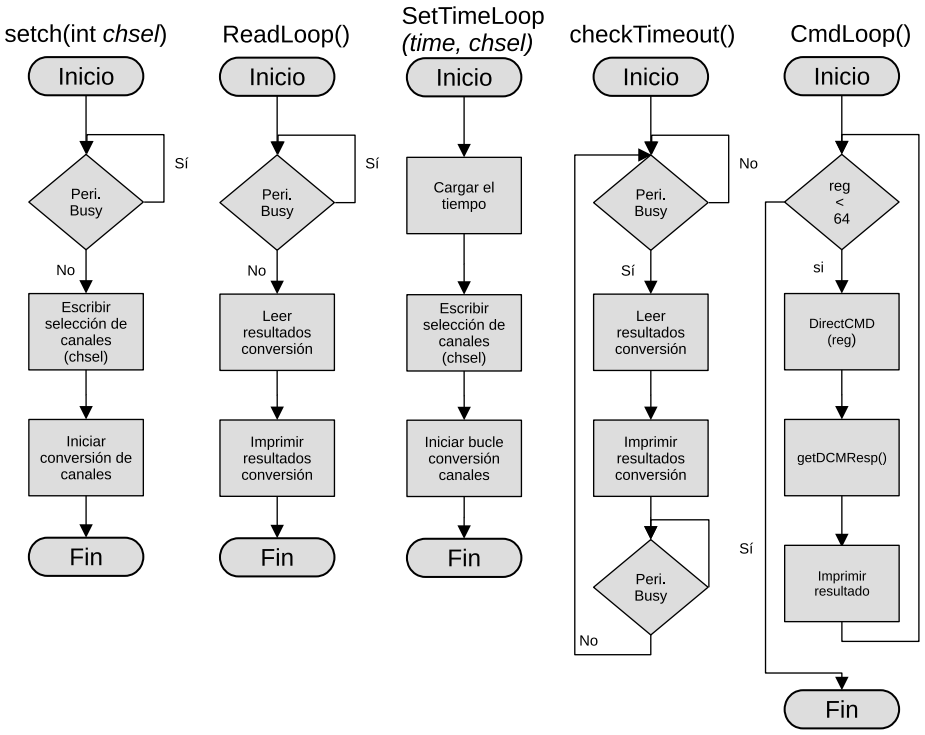


*Figura 3-6. Diagrama de flujo de las funciones implementadas II*

También se implementa una función que realiza la lectura de todos los registros por medio de la comunicación directa de forma automática, *CmdLoop()*. De esta manera se comprueba el funcionamiento de este proceso al encontrarse con una serie de comandos continuos.

La verificación del proceso de conversión de canales se realiza por medio de tres funciones. Por un lado, una función encargada de guardar la selección de los canales y de iniciar el proceso, *setch(int chSel)*. Por otro lado, utilizamos una función, *ReadLoop()*, que permite la lectura de los 16 registros que almacenan el resultado de la conversión. También se programa una función, *setTimeLoop(int time, int chsel)*, encargada de configurar el bucle periódico de conversión de canales.

Ha sido necesario modificar también el bloque que emula el comportamiento del circuito integrado para facilitar la búsqueda de fallos en el almacenamiento de la respuesta de la conversión. Se realizan pruebas generando la selección de canales de una manera pseudoaleatoria, buscando cubrir la mayor cantidad de condiciones posibles.

A la hora de leer la respuesta de las conversiones, como ya se ha comentado se utiliza una función que realiza todo el proceso. Cabe destacar que, al igual que ocurre en el caso de la lectura de la respuesta con prioridad, la función también realiza el reinicio del bit que indica el final del proceso de conversión y la interrupción que indica lo mismo.

Se realizan también pruebas para cubrir el funcionamiento del bucle de conversión de canales. Al conseguir el correcto funcionamiento del bloque sin activar la periodicidad, el objetivo de la prueba es verificar si el periodo es correcto y es capaz de realizar la

conversión de manera continua. Para ello también se ha implementado una función, encargada de indicar cuando comienza un nuevo proceso de conversión.

Para finalizar las pruebas sobre simulación, se implementa un pequeño programa que emula el comportamiento del sistema real, comenzando con una inicialización y seguida por la ejecución del resto de los procesos de manera arbitraria. El orden de los procesos es modificado varias veces para probar distintos escenarios.

## 3.3 Síntesis e implementación del periférico

Al mismo tiempo que se han realizado las pruebas por simulación se han realizado distintas pruebas de Síntesis e implementación, tanto del bloque SPI de manera independiente como del periférico. Durante el proceso de síntesis, la descripción HDL del circuito es traducida al nivel de puertas lógicas. Durante la implementación se determinan los recursos de la FPGA que serán utilizados en función, entre otras cosas, de cumplir las restricciones que se le añaden al sistema una vez se implemente sobre hardware. Realizar pruebas de síntesis nos permite encontrar errores que no aparecen al lanzar la simulación HDL como la aparición de *latches*, elementos de memoria que no utilizan señal de reloj, no deseados. También permite realizar pruebas para la comprobación de las restricciones temporales, así como indicar los recursos utilizados por el bloque.

El reloj de la FPGA tiene una frecuencia de funcionamiento $f_{FPGA} = 100\ MHz$ [12], que es superior a la utilizada por el sistema. Para generar el reloj del sistema $f_{clk} = 96\ MHz$ se añade un IP encargado de la gestión de relojes, el MMCM (*Mixed-Mode Clock Manager*) [13]. Esta IP permite la generación, a partir del reloj de la placa, de varias señales de reloj a distintas frecuencias. En este caso solo se utiliza para la generación de una señal. El IP genera sus propias restricciones temporales en las que se implementa la restricción del reloj de la placa.

Una vez diseñado el bloque SPI y comprobado el bloque SPI es necesario ajustar las condiciones temporales a cumplir, ya que el sistema real presentará retardos.

A la hora de generar las restricciones temporales se genera como reloj virtual el reloj del sistema. *SCLK* también se genera como un reloj virtual, teniendo este la cuarta parte de la frecuencia del reloj del sistema. Estos relojes se referencian al puerto de salida del MMCM.

El cambio de dato de la señal *MISO* se realiza con el flanco de bajada de la señal *SLCK*. Sin embargo, presenta un pequeño retraso hasta que el dato es válido por lo que la captura del nuevo dato de llegada se retrasa y se realiza en el último flanco de *CLK*, de esta manera se deja más tiempo que el recomendado por el datasheet del RHD22xx para que el dato pueda cambiar.

Como ya se ha comentado, para la comunicación con el RHD22xx se utilizan señales diferenciales de bajo voltaje sobre las que se envía la comunicación SPI. El *SN65LVDT14* de Texas Instruments [14] realiza la adaptación de señales SPI sobre señales diferenciales de bajo voltaje (LVDS). Consta de un *driver* que convierte las señales diferenciales en una señal y 4 receptores que realizan el proceso contrario.

Para la conexión también se utiliza el cable recomendado en el datasheet del *RHD22xx* que transporta las señales LVDS. Se utiliza un cable de 0.9 metros de longitud.

La introducción de estos elementos en el sistema incluye una serie de retardos en la transmisión que deben de tenerse en cuenta a la hora del diseño de la interfaz SPI, por lo que se han añadido restricciones temporales en el diseño de la interfaz.

Tras consultar las respectivas hojas de características, se definen los máximos retrasos que encontramos en cada pieza del sistema:

$$Delay_{cable}(ns) = 0.149\ (m/ns) * 0.9\ (m) = 6.04\ (ns)$$

*Ecuación 3. Cálculo del retardo introducido por el cable [15]*

En el caso del retardo introducido por el *SN65LVDT14* [14], la hoja de características nos indica un retardo que oscila entre 1 ns y 3.8 ns en el caso del receptor. En el caso del *driver* oscila entre 2.9 ns y 0.9 ns.

Los retardos de entrada camino de los datos llegados por el puerto *MISO*, datos de entrada al periférico que cambian en el flanco de bajada del reloj *SCLK*, se definen como el tiempo máximo y mínimo que se utiliza desde la salida del flanco de reloj *SCLK* hasta la llegada del nuevo dato al periférico. El retardo a tener en cuenta viene dado por la siguiente fórmula:

$$Input\ Delay = 2\ Delay_{cable} + Delay_{driver} + Delay_{recptor} + t_{MISO}$$

*Ecuación 4: Cálculo del retardo de entrada en el MISO*

Con los valores obtenidos de las hojas de características se calculan los retardos de llegada máximos y mínimos. Este retardo oscilará entre los 24.02 ns y los 30.78 ns.

Finalmente se configura el camino desde el reloj del periférico a través del puerto *MISO* como un camino multiciclo, puesto que el puerto trabaja con la señal *SCLK*.

En el camino de salida de los datos, puerto *MOSI*, se configura el retardo máximo de salida con el tiempo mínimo del *MOSI* obtenido de la hoja de características. También se configura el camino desde el reloj del sistema a través del puerto como un camino multiciclo.

En los documentos anexos, en la sección Anexo B. Restricciones temporales, se incluyen los ficheros que contienen las restricciones temporales utilizados para la prueba del bloque SPI asi como del periférico completo.

También se sitúa la restricción física que sitúa los Flip-Flops de los puertos *MOSI*, *MISO* y *SCLK* en los IOBs (*Input Output Block*) asociados a cada puerto. Estos bloques se encuentran al lado del pin de entrada lo que permite registrar la señal al momento que llega al PAD de la FPGA con lo que se consigue reducir los caminos de entrada y de salida, reduciendo también el retardo asociado a estos caminos.

Tras realizar la síntesis e implementación con las restricciones temporales anteriores los informes temporales indican que no se viola ninguna condición.

Tras realizar la implementación del periférico se ha generado un informe sobre las restricciones temporales. Este informe nos permite conocer el *Slack* de los distintos caminos. El *Slack* es la diferencia entre el tiempo requerido, de *Setup* en este caso, y el tiempo real que tarda el dato en llegar al final del camino de datos. En caso de querer conocer el *Slack* de *Hold* la diferencia se realizaría al revés. Un valor positivo nos indica que las restricciones temporales se cumplen. En la Tabla 3-2 se incluyen los *Slacks* obtenidos en el informe que se obtiene lanzando el comando *report_timing_summary* en la consola TCL (Tool Command Language) de Vivado. *CLK96* representa el reloj del sistema y *SCPICLK* el reloj de la interfaz serie, *SCLK*.

| From clock | To clock | Worst Setup Slack | Worst Hold Slack |
|---|---|---|---|
| CLK96 | CLK96 | 1.376 ns | 0.048 ns |
| SCPICLK | CLK96 | 1.376 ns | 0.777 ns |
| CLK96 | SCPICLK | 10.165 ns | N/A |

*Tabla 3-2. Resultados extraídos del informe temporal del periférico tras la implementación*

La implementación del periférico también permite conocer los recursos utilizados sobre la FPGA una vez el diseño se integre en esta. La siguiente tabla, Tabla 3-3, muestra los recursos utilizados por cada bloque y por el periférico completo, se ha obtenido por medio del comando TCL *report_utilization -hierarchical*.

| Instance | Module | Total LUTs | Logic LUTs | Flip-Flops |
|---|---|---|---|---|
| Intan_RHD2xxx_v1_0 | (top) | 1174 | 1174 | 2169 |
| Peri_inst | Intan_RHD2xxx_v1_0_s00_AXI | 1174 | 1174 | 2169 |
| (Peri_inst) | Intan_RHD2xxx_v1_0_s00_AXI | 738 | 738 | 1965 |
| CH_READER_B | CH_READER | 199 | 199 | 90 |
| INI_B | INI | 183 | 183 | 66 |
| SPI_B | SPI | 54 | 54 | 48 |

*Tabla 3-3: Resumen de los recursos utilizados por el periférico en la implementación.*

## 3.4 Pruebas sobre hardware

Antes de realizar las pruebas del sistema completo indicado en la Figura 2-4, se realizó la prueba del bloque SPI de manera individual sobre componentes físicos. Esto se debe a la necesidad de verificar los diversos tiempos de la interfaz con los del sistema real.

Como ya se ha comentado en la sección Diseño del sistema, el sistema real consta de una *FPGA Artix-7*, sobre la que se implementa tanto el microprocesador como el periférico, conectado a través de la interfaz SPI al RHD22xx por medio de un cable y el circuito integrado *SN65LVDT14*.

La prueba del bloque SPI de manera independiente al resto del sistema se realiza en las primeras fases del proyecto dada la importancia del bloque, tras haber completado la verificación por simulación, síntesis e implementación del bloque. Por ello es necesario verificar el funcionamiento de la comunicación real con el RHD22xx y además obtener ciertos tiempos, como el retardo de salida de la señal *SCLK*.

Para llevar a cabo la prueba, se ha implementado un nuevo banco de pruebas sintetizable, el utilizado durante simulación utilizaba estructuras no sintetizables. Para ello se diseña un banco de pruebas como un componente nuevo, que se incluye junto al bloque SPI y un MMCM para la gestión de relojes en la FPGA.

Como se ha comentado en la sección anterior, el MMCM, utilizado en todas las pruebas sobre hardware, se encarga de reducir la frecuencia de $f_{FPGA} = 100\,MHz$ [12] del reloj de la placa a los $f_{clk} = 96\,MHz$ que utiliza el sistema.

El bloque encargado de las pruebas consta de una LUT en la que se almacenan todos los comandos a enviar. Por medio de una máquina de estados y un contador se realiza el envío y la recepción de los comandos a través de las interfaces AXI4-Stream del bloque SPI, el contador se utiliza como puntero de la LUT. El banco de pruebas sintetizable sigue la estructura de la Figura 3-1 en la que se cambia el RHD22xx_fake.vhd por el RHD22xx real y el banco de pruebas pasa a ser el banco de pruebas sintetizable.

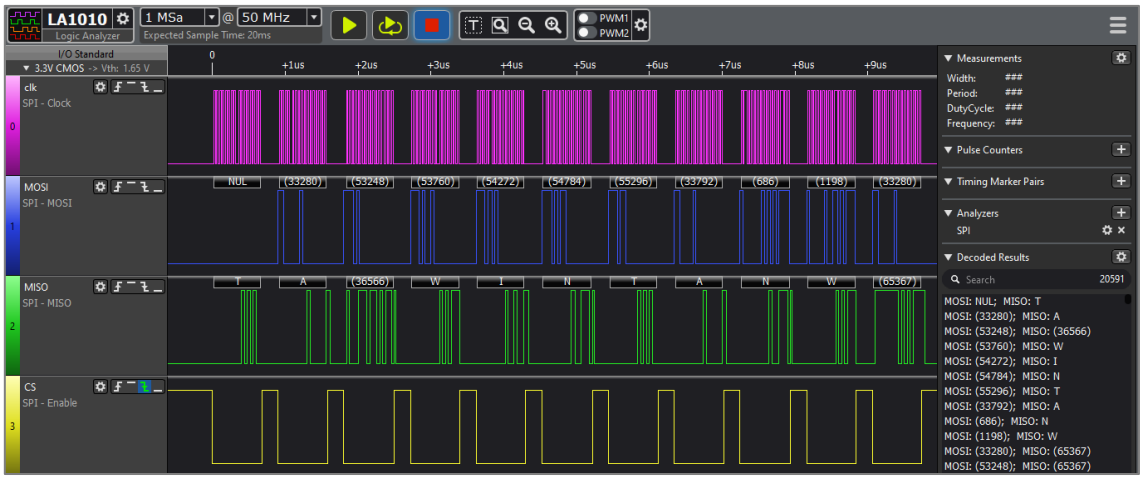


*Figura 3-7: Extracto de la comunicación serie capturado con un analizador lógico*

Los datos son analizados gracias a un analizador lógico conectado a los puertos de la interfaz serie. Estos se convierten a caracteres ASCII y son interpretados como se puede ver en la Figura 3-7.

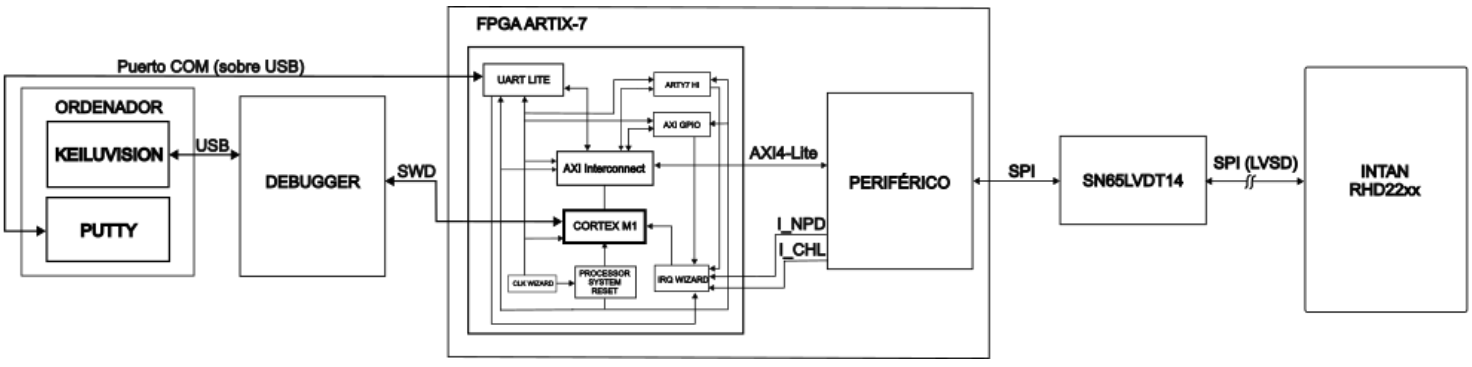


*Figura 3-8: Diagrama de los bloques importantes para la prueba sobre hardware del sistema completo*

La prueba del resto de bloques se realiza una vez el periférico está completo. Como se puede ver en el diagrama de bloques de la Figura 3-8, para la prueba del sistema se incluye el procesador y el periférico dentro de la FPGA. Por medio del *debugger J-link edu,* de la marca Segger*,* es posible conectarse con el microprocesador a través del puerto serie. Esto nos permite comprobar desde el ordenador, a través de una consola, el resultado de las instrucciones ejecutadas. Además, desde KeiluVision y a través del *debugger,* es posible elegir y modificar el programa que se ejecuta sobre el microprocesador. En la Figura 3-9 encontramos el sistema que corresponde al diagrama de bloques descrito.

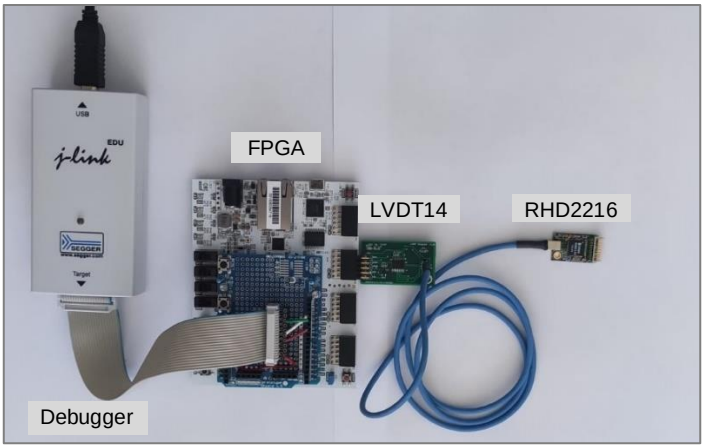


*Figura 3-9. Sistema utilizado para las pruebas sobre Hardware*

Para las pruebas sobre el sistema físico, aunque en un primer momento se utilizan los mismos programas que los utilizados en simulación, se ha implementado un código en C que mantiene la estructura de las pruebas realizadas por simulación.

El programa comienza con el proceso de inicialización y la comprobación de que este ha sido correcto, indica por pantalla el contenido de los registros de sólo lectura copiados en el periférico y de manera automática indica si es correcto. A continuación, se realiza la comprobación de la lectura directa, mediante instrucciones sueltas y con un bucle de lectura de todos los registros tras el que, por medio de los datos impresos en el terminal, se comprueba que todos los registros contengan los datos esperados. Para finalizar se realiza la comprobación de la conversión de canales utilizando distintos vectores de selección de canal y finalmente se realiza la conversión de canales de manera periódica.

Este programa se ejecuta varias veces variando los distintos parámetros de las funciones, buscando comprobar todos los fallos posibles.

Aunque las pruebas sobre el sistema físico presentan un gran apoyo a la hora de encontrar fallos y testear el sistema, en este caso presentan ciertas limitaciones que nos impiden comprobar el funcionamiento completo del periférico. Es en el caso de la conversión de canales, durante las pruebas en simulación se ha comprobado el funcionamiento por medio de un bloque al que se le ha dotado del comportamiento que interesaba frente al comando de conversión. Sin embargo, el circuito integrado real no tiene esa capacidad, está calibrado para trabajar con señales biológicas, difíciles de emular en el laboratorio por sus características. Paralelo a este proyecto, se están desarrollando los electrodos para utilizarlos junto con el RHD22xx que, una vez terminados, se podrán usar para la verificación del periférico y realizar las modificaciones necesarias.

# 4 Conclusión

A lo largo de este proyecto se ha realizado el diseño y la verificación de un periférico que interactúa como intermediario en la comunicación microprocesador-RHD22xx. Durante la fase de diseño se han extraído las especificaciones a partir de las distintas hojas de características del resto de componentes del sistema y del problema planteado. Se han separado los distintos procesos en bloques independientes para facilitar el trabajo.

El periférico permite automatizar tareas que se realizan de forma continua en el RHD22xx, liberando de carga computacional al microprocesador que lo controla. El periférico ha sido diseñado siguiendo una arquitectura modular, facilitando la implementación de los procesos en bloques separados y permitiendo añadir nuevas funciones en caso de ser necesario.

En el diseño del bloque SPI, que contiene dicha interfaz, se ha adaptado el diseño en función de los requisitos temporales del sistema y se ha realizado una verificación exhaustiva, tanto por simulación como sobre el sistema físico, del cumplimiento de estas condiciones.

Se ha diseñado un modo de comunicación directa, en la que el periférico actúa de manera transparente realizando solo el intercambio de comandos y respuesta entre el microprocesador y el RHD22xx. Además, se implementa la generación de una interrupción al finalizar el proceso, con lo que se consigue avisar al microprocesador de forma activa.

En el diseño del bloque de inicialización, se opta por el diseño de una rutina basada en la recomendada por la guía de usuario. Se decide añadir la lectura de todos los registros del RHD22xx y su escritura en los registros del periférico, con el objetivo de tener una imagen de la memoria del RHD22xx en el periférico y poder acceder de forma paralela a cada registro.

A la hora de diseñar el bloque de conversión de canales se utiliza un registro para la selección de los canales. Se decide compartir cada registro que almacena la respuesta a la conversión entre 2 canales, reduciendo el número de registros utilizados y facilitando y agilizando la lectura desde la interfaz AXI4-Lite. La generación de una interrupción al finalizar la conversión de los canales le permite al periférico avisar de forma activa al microprocesador de la finalización del proceso.

En la fase de verificación se desarrolla un banco de pruebas que simplifica la generación de los estímulos de la interfaz AXI4-Lite para el envío de comandos al periférico, a través de la lectura de un fichero de texto con los comandos a enviar. Se realizan pruebas por medio de simulación, tanto de los bloques independientes, del periférico y del sistema completo, permitiendo así una búsqueda y corrección de errores exhaustiva. Las pruebas sobre hardware añaden una capa más de verificación.

## 4.1 Lineas futuras

Se podrían realizar cambios en el diseño del periférico implementando nuevas funcionalidades como la detección de errores al establecer la comunicación y verificar la escritura de los registros en el proceso de inicialización.

Además, gracias a la arquitectura expandible del periférico se podría implementar un bloque encargado de la captura y almacenamiento en un bloque de memoria de los resultados de la conversión de canales. El bloque permitiría configurar el proceso de conversión de canales periódico, permitiendo ajustar de forma más precisa el muestreo de los canales y la duración del bucle, además de almacenar los resultados directamente en un bloque de memoria. De esta forma se conseguiría una reducción mayor de la carga computacional del microprocesador.

Otra linea de trabajo es el desarrollo de una librería de funciones en C para facilitar la utilización del periférico. Cubriendo de esta manera la comunicación con el periférico y mejorando sus funciones. La implementación de este sistema mejoraría la interacción a nivel usuario.

También sería conveniente avanzar en la verificación del sistema sobre el soporte físico realizando pruebas del proceso de conversión de canales utilizando los electrodos, que se encuentran en desarrollo, para la captura de los estímulos y verificar así el correcto funcionamiento del periférico y del sistema completo.

# 5 Bibliografía

[1] INTAN Technologies (2023), “RHD electrophysiology Amplifier Chips” [Online]. Disponible en: https://intantech.com/products_RHD2000.html

[2] INTAN Technologies, "Digital Electrophysiology Interface Chips", RHD2000 Series Datasheet, diciembre 2012 [Actualizado agosto 2022].

[3] Xilinx (2023, junio 25), "Create and Packaging Custom IP"

[4] ARM, "AMBA(R) AXI and ACE Protocol Specification", junio 2003 [actualizado marzo 2020].

[5] ARM, "AMBA(R) AXI-Stream Protocol Specification", marzo 2010 [actulaizado abril 2021].

[6] Analog Devices (2015), "SPI Interface", Application note No.1248

[7] Xilinx (2018), "Vivado Design Suite User Guide".

[8] I. Urriza Parroqué (2023), "rhd2000_fake.vhd" [VHDL].

[9] I. Urriza Parroqué (2023), "M1_mi" [Proyecto de Vivado]

[10] I. Urriza Parroqué (2023), "keil.m1_arty7_1062_DFP.1.5.5.pack" (v.1.5.5) [uVision Software pack]

[11] ARM (2014), "Fromelf User Guide".

[12] Digilent (2023), "Arty A7 Reference Manual", [Online]. Disponible en: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual.

[13] Xilinx (2009), "Mixed-Mode Clock Module (MMC) Module (v1.00a)"

[14] Texas Instruments, "SN65LVDTxx multi-channel LVDS transceivers", SN65LVDT14 Datasheet, abril 2002 [revisado febrero 2019]

[15] INTAN Technologies, "RHD SPI Interface Cable/Connector Specification", febrero 2013 [actualizado mayo 2021]

[16] Xilinx (2012), "AXI Reference Guide".

[17] Xilinx (2022), "Using Constraints".

[18] ARM (2022), "uVision User's Guide V5.38a".

# 6 Lista de figuras

# 7 Lista de tablas



# 8 Lista de acrónimos

**AMBA:** Advanced Microcontroller Bus Architecture

**AXI:** Advanced eXtensible Interface

**CDC:** Clock Domaing Crossing

**CPU:** Central Processing Unit

**FPGA:** Field Programable Gate Array

**HDL:** Hardware Description Language

**IP:** Intellectual Property

**IOB:** Input Output Block

**LSB:** Least Significant Bit

**LUT:** Look-Up Table

**LVDS:** Low Voltage Differential Signaling

**MCU:** Microcontroller Unit

**MEF**: Máquina de Estados Finitos

**MMCM:** Mixed-Mode Clock Module

**RISC:** Reduced Instruction Set Computer

**ROM:** Read Only Memory

**SPI:** Serial Peripheral Interface

**TCL:** Tool Command Language

**UUT:** Unit Under Test

# ANEXOS

## Anexo A. Diseño de un periférico AXI-4 Lite para la adquisición de señales electro-fisiológicas con el integrado INTAN RHD22xx AXI4-Lite – INTAN RHD22xx

A continuación, se incluye la documentación del periférico diseñado. Esta documentación amplía la información recogida en la memoria sobre el periférico.

# Periférico para comunicación AXI4-Lite – INTAN RHD22xx

Última revisión: 30/08/2023

## Introducción

El periférico permite una comunicación con el circuito integrado INTAN RHD22xx por medio de la interfaz serie (SPI). Permite la comunicación para su control y acceso a sus registros con la interfaz AXI4-Lite de la especificación AMBA (*Advance Microcontroller Bus Architecture*).

El periférico descrito en este documento está pensado para su utilización junto al circuito integrado INTAN RHD22xx [1]. El datasheet del circuito integrado es referenciado a lo largo del documento y amplía la información del funcionamiento de la comunicación y del mapa de memoria.

## Características

- Interfaz AXI4-Lite para el acceso a los registros y control del periférico.
- Interfaz SPI para la comunicación con el integrado RHD22xx.
- Proceso de inicialización del integrado predefinido.
- Proceso de conversión de los canales seleccionados con opción periódica.
- Permite la comunicación directa con el circuito integrado.
- Generación de interrupciones para el aviso de llegada de respuestas y final del bucle de conversión.

# Descripción general

Los módulos del periférico se muestran en la siguiente figura y son descritos a continuación.

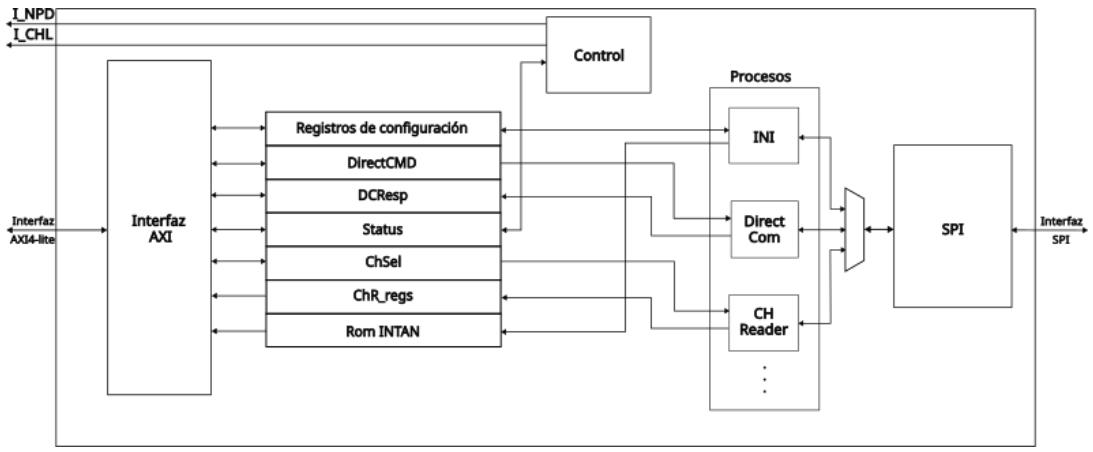


*Figura 0-1: Diagrama de bloques del periférico*

- **Interfaz AXI:** El bloque implementa la interfaz esclava de AXI4-Lite [2] para el acceso a los registros.
- **Registros:** Este bloque está formado por 64 registros de 32 bits utilizados para el control del periférico y de los distintos procesos que implementa. Los registros siguen la distribución del mapa de memoria del RHD22xx y almacenan una copia de la memoria de este. El mapa de los registros se detalla en la sección Descripción de los registros.
- **Control:** Este bloque se encarga del control y la gestión del periférico. Se controla por medio de la escritura en determinados bits del registro *Status*. Es el encargado de la generación de interrupciones, control de los distintos procesos y de la gestión del bucle de conversión de canales.
- **Procesos:** Este bloque engloba los distintos procesos que ejecuta el periférico:
  - **INI** → Bloque encargado del proceso de inicialización.
  - **Direct com** → Bloque encargado de la transmisión y recepción de comandos y respuestas de manera directa a través de la interfaz SPI.
  - **CH Reader** → Bloque encargado del proceso de conversión de los canales seleccionados.
- **SPI:** El bloque implementa la interfaz SPI maestra para la comunicación con el circuito integrado RHD22xx.

## Resumen de las características

El periférico presenta las siguientes características:

- Realiza el proceso de inicialización de manera independiente a partir de la configuración indicada en los registros por medio de la interfaz AXI4-Lite. Realiza también una calibración y copia de los registros del RHD22xx.
- Presenta un proceso de conversión de canales seleccionados configurable y con capacidad para realizarlo de manera periódica con un periodo configurable.
- Genera 2 interrupciones para la notificación de la recepción de respuestas y para el final del bucle de conversión.
- La distribución de los registros mantiene la estructura de la memoria del RHD22xx.
- El periférico trabaja con una frecuencia de reloj igual a $f_{clk} = 96\ MHz$.
- La interfaz serie trabaja a una frecuencia igual a $f_{sclk} = 24\ MHz$.
- Periodo máximo de la conversión de canales periódica igual a $T_{max} = 681.6\ \mu s$.

# Especificaciones

## Descripción de los puertos

Los puertos del periférico se describen en la Tabla 0-1.

*Tabla 0-1: Descripción de señales I/O*

| Nombre de la señal | Interfaz | I/O | Estado inicial | Descripción |
|---|---|---|---|---|
| **Señales SPI** | | | | |
| MISO | SPI | I | - | *Master Input Slave Output*. Recepción de datos. (Esclavo → Maestro) |
| MOSI | SPI | O | 0 | *Master Output Slave input.* Transmisión de datos. (Maestro → Esclavo) |
| SCLK | SPI | O | 0 | Salida del reloj ($f_{sclk} = 24\ MHz$) de la interfaz Serie. (Maestro → Esclavo) |
| CS | SPI | O | 1 | Chip Select, SPI. (Maestro → Esclavo) |
| **Señales de la interfaz AXI4-Lite** | | | | |
| S00_axi_* | S_AXI | - | - | Consultar el documento "AXI reference guide (UG761)" [3] para una descripción completa de las señales de la interfaz AXI4. |
| **Señales del sistema** | | | | |
| s_axi_aclk | Sistema | I | - | Reloj de la interfaz AXI. Utilizado como reloj del periférico. Frecuencia de trabajo: $f_{clk} = 96MHz$ |
| s_axi_arestn | Sistema | I | - | Reset AXI, activo en bajo. |
| I_NPD | Interrupción | O | 0 | Señal de interrupción de nueva respuesta. |
| I_CHL | Interrupción | O | 0 | Señal de interrupción de finalización del bucle de conversión de canales. |

## Descripción de los registros

La Tabla 0-2 muestra todos los registros del periférico, su dirección, tipo de acceso y la descripción de su función.

*Tabla 0-2: Mapa de direcciones de los registros*

| Dirección | Nombre del registro | Tipo de acceso | Descripción |
|---|---|---|---|
| 0x00 – 0x44 | Registros de configuración | R/W | Registros de configuración del RHD22xx. |
| 0x48 | DirectCMD | W | Comando transmisión directa hacia el Integrado. |
| 0x4C | DCResp | R | Respuesta directa del Integrado al comando enviado. |
| 0x50 | Status | R/W | Registro de estado y control del periférico. |
| 0x54 | ChSel | R/W | Selector de canales para la conversión en bucle. |
| 0x58 | ChR_reg0 | R | Repuesta de la conversión de los canales 0 y 1. |
| 0x5C | ChR_reg1 | R | Repuesta de la conversión de los canales 2 y 3. |
| 0x60 | ChR_reg2 | R | Repuesta de la conversión de los canales 4 y 5. |
| 0x64 | ChR_reg3 | R | Repuesta de la conversión de los canales 6 y 7. |
| 0x68 | ChR_reg4 | R | Repuesta de la conversión de los canales 8 y 9. |
| 0x6C | ChR_reg5 | R | Repuesta de la conversión de los canales 10 y 11. |
| 0x70 | ChR_reg6 | R | Repuesta de la conversión de los canales 12 y 13. |
| 0x74 | ChR_reg7 | R | Repuesta de la conversión de los canales 14 y 15. |
| 0x78 | ChR_reg8 | R | Repuesta de la conversión de los canales 16 y 17. |
| 0x7C | ChR_reg9 | R | Repuesta de la conversión de los canales 18 y 19. |
| 0x80 | ChR_reg10 | R | Repuesta de la conversión de los canales 20 y 21. |
| 0x84 | ChR_reg11 | R | Repuesta de la conversión de los canales 22 y 23. |
| 0x88 | ChR_reg12 | R | Repuesta de la conversión de los canales 24 y 25. |
| 0x8C | ChR_reg13 | R | Repuesta de la conversión de los canales 26 y 27. |
| 0x90 | ChR_reg14 | R | Repuesta de la conversión de los canales 28 y 29. |
| 0x94 | ChR_reg15 | R | Repuesta de la conversión de los canales 30 y 31. |
| 0x98 – 0x9C | Reservado | N/A | |
| 0xA0 – 0xB0 | INTAN | R | Registro de solo lectura para datos del integrado. |
| 0xB4 – 0xEC | Reservado | N/A | |
| 0xF0 – 0xFC | INTAN ROM | R | Registros de solo lectura para datos del integrado. |

### Registros de configuración

Los registros de configuración mantienen el mismo orden que los registros de configuración del RHD22xx. Ver “INTAN RHD2000 series datasheet” [1] para una descripción completa de estos registros. La información de estos registros se copia al registro correspondiente durante el proceso de inicialización y, al finalizar, se sobrescriben con el contenido de los registros del RHD22xx.

### Registro de comando con prioridad (DirectCMD)

Sobre este registro se escribe el comando recibido desde la interfaz AXI4-Lite, a la espera de enviarlo hacia el circuito integrado por la interfaz SPI, sin ser procesado en el periférico. Es un registro de escritura y lectura.

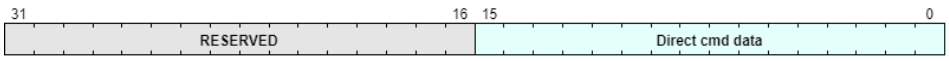


*Figura 0-1. DirectCMD (@0x48, bits de datos = 16).*

### Registro de respuesta al comando con prioridad (DCResp)

Este registro de sólo lectura contiene la respuesta inmediata del RHD22xx obtenida tras el envío del comando escrito en el registro *DirectCMD*. Tiene como valor por defecto es 0x00.

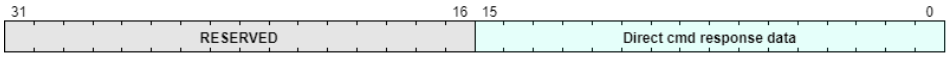


*Figura 0-2. DCResp (@0x4C, bits de datos = 16).*

### Registro de estado y control (Status)

Este registro contiene la información del estado del periférico y la escritura en algunos bits permite configurar e iniciar los distintos procesos: inicialización, comando directo y bucle de conversión. Registro de Escritura y lectura.

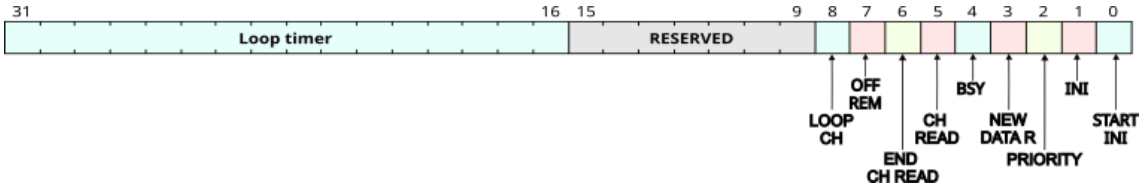


*Figura 0-3. Status (@0x50)*

En la Tabla 0-3 se describen los bits del registro.

*Tabla 0-3: Definición de los bits del registro de estado y control*

| Bits | Nombre | Tipo de acceso | Valor inicial | Descripción |
|---|---|---|---|---|
| 0 | START_INI | Write | 0h | Escribir 1 da inicio al proceso de inicialización. Este bit se limpia de manera automática tras su escritura.<br>0 = Reposo<br>1 = comenzar inicialización |
| 1 | INI | Read | 0h | Indica que el proceso de inicialización está corriendo.<br>0 = Proceso de inicialización desactivado<br>1 = Proceso de inicialización corriendo |
| 2 | PRIORITY | Write | 0h | Escribir 1 comienza el proceso de comunicación directa, transmite el contenido del registro *DIRECTCMD*. Se limpia de manera automática al comenzar el proceso.<br>0 = Reposo<br>1 = Enviar comando con prioridad |
| 3 | NEW_DATA_R | Read / Write | 0h | Indica la recepción de una respuesta al comando enviado con prioridad[1]. Es necesario poner a 0 el campo una vez se ha leído el registro *DCResp*. Conectado a la interrupción *I_NPD*<br>0 = No hay respuesta nueva<br>1 = Nueva respuesta recibida |

[1] Se entiende por respuesta inmediata los bits recibidos por la entrada MISO del SPI durante la transmisión del comando con prioridad. Los datos recibidos no contienen la respuesta al comando enviado de forma inmediata. Para más información ir al *"INTAN RHD2000 series datasheet"* [1].

| Bits | Nombre | Tipo de acceso | Valor inicial | Descripción |
|---|---|---|---|---|
| 4 | BSY | Read | 0h | Indica si el periférico se encuentra procesando o puede aceptar nuevos procesos.<br>0 = Reposo<br>1 = Periférico ocupado |
| 5 | CH_READ | Write | 0h | Escribir 1 comienza el proceso de conversión de los canales seleccionados en el registro ChSel. Este bit se limpia automáticamente al comenzar el proceso.<br>0 = Reposo<br>1 = Comienzo proceso de conversión de canales |
| 6 | END_CH_READ | Read / Write | 0h | Indica la finalización del proceso de conversión de canales y la disposición de nuevos datos. Es necesario poner a 0 el bit una vez se hayan leído los nuevos datos de los registros. Conectado a la interrupción I_CHL.<br>0 = no hay nuevos datos disponibles<br>1 = fin de la conversión y nuevos datos disponibles |
| 7 | OFF_REM | Write | 0h | Indica si el proceso CH READ va a realizarse aplicando Offset Removal[2]. Esto se aplicará en todos los canales selecionados en ChSel.<br>0 = Offest Removal desactivado<br>1 = Offest Removal activado |
| 8 | LOOP_CH | Write | 0h | Escribir 1 indica el comienzo del bucle de conversión de canales periódico. Escribir 0 detiene el bucle.<br>0 = Bucle detenido<br>1 = Bucle de conversión de canales iniciado |
| 15-9 | RESERVED | N/A | - | Reservado |
| 31-16 | LOOP_TIMER | Write | 0x00 | Bits que indican el periodo, en ciclos de reloj, entre el lanzamiento del proceso de conversión de canales cuando LOOP_CH está activo. |

### Registro de selección de canales (ChSel)

El registro de selección de canales, *ChSel*, es un registro de escritura que permite seleccionar los canales que se quieren convertir durante el bucle de conversión de canales. Escribir a 1 un bit en el registro implica que el canal que coincide con la posición del bit se convertirá (i.e: bit 0 corresponde al canal 0).

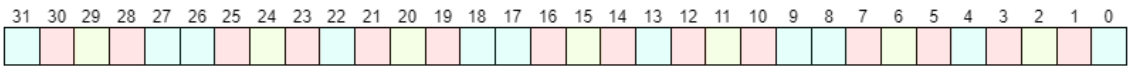


*Figura 0-4: ChSel (@0x54, cada bit corresponde al canal con el mismo número).*

[2] Para más información sobre la conversión de canales y sobre el Offset removal consultar la "INTAN RHD2000 series datasheet" [1].

### Registros de conversión de canales (ChR_reg0 – ChR_reg 15)

Los registros de conversión de canales contienen el resultado de realizar la conversión de los canales indicados en el registro *ChSel*. Son registros de sólo lectura, la escritura en estos registros no tiene efecto. Todos los registros denominados *ChR_regN*, con N de 0 a 15 siguen la misma estructura: Contienen el contenido de 2 canales, los bits [31:16] corresponden al canal impar y los bits [15:0] al canal par.

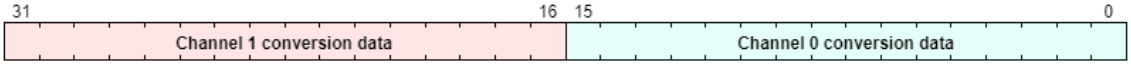


*Figura 0-5: ChR_regN (@0x58-@0x94)*

### Registros de solo lectura (INTAN, INTAN ROM)

Los registros de sólo lectura, *INTAN* e *INTAN ROM*, son registros que mantienen la misma dirección que los registros en el circuito integrado. Durante el proceso de inicialización copian el contenido de los registros del INTAN RHD22xx y no se modifican. Para más información sobre el contenido de los registros ver "INTAN RHD2000 series datasheet" [1].

## Procesos

En los siguientes apartados se detallan los procesos que realiza el periférico.

### Comunicación Directa

Este proceso permite el envío de comandos directamente al RHD22xx, sin ser modificados por el periférico. El comando a enviar se almacena en el registro *DirectCMD (@0x48)* para su envío.

La respuesta inmediata, que entendemos por la respuesta que se recibe por el puerto *MISO* durante la transmisión del comando, se almacena en el registro *DCResp (@0x4C)* para su lectura.

El proceso genera una interrupción para avisar de la llegada de la nueva respuesta, también activa el bit *NEW_DATA_R* en el registro *Status (@0x50)*.

### Inicialización

Este proceso se encarga de la inicialización del RHD22xx. Se basa en el proceso descrito en la hoja de características del RHD22xx [1]. Solo debe ejecutarse una vez al comienzo de la comunicación.

Antes de lanzar el proceso, es necesario escribir en los registros de configuración *(@0x00 – @0x44)* la configuración deseada para el RHD22xx.

Este proceso realiza las siguientes funciones:

1. Inicio de la comunicación.
2. Escritura de los registros de configuración.
3. Calibración del ADC del RHD22xx.
4. Copia del contenido de todos los registros del RHD22xx a los registros del periférico.

Durante el tiempo que dura el proceso, el bit *INI* del registro *Status (@0x50)* se mantiene a nivel alto.

### Conversión de selección de canales

El proceso se encarga de la conversión de una serie de canales definidos por el usuario en el registro *CHSel (@0x54)*.

El resultado del proceso se almacena en los registros indicados para ello.

El proceso ofrece la posibilidad del eliminar el offset asociado a los amplificadores analógicos de los canales muestreados en todos los canales seleccionados por el usuario. Para más información sobre el "offset removal" ver "INTAN RHD2000 series datasheet" [1].

Es posible realizar esta conversión de manera periódica. El periodo máximo entre conversiones es de 681.6 us. El periodo del bucle debe introducirse en el registro *Status (@0x50)*.

Al terminar cada proceso de conversión se genera una interrupción y se activa el bit *END_CH_LOOP* en el registro *Status (@0x50)*.

## Recursos utilizados

El periférico utiliza los siguientes recursos medidos en una FPGA Artix-7.

| Recursos del dispositivo | Slices | Slice Flip-Flops | LUTs |
|---|---|---|---|
| Recursos utilizados | 802 | 2169 | 1174 |

# Guía de diseño

Este capítulo incluye información adicional y guías para facilitar el diseño y trabajo con el periférico.

## Relojes

El periférico opera con el reloj en *s_axi_aclk*. La frecuencia de trabajo del periférico es igual a $f_{clk} = 96\ MHz$. Como la frecuencia difiere de la del reloj interno de las FPGA, se recomienda el uso de un MMCM [13] para la generación de la señal de reloj.

El INTAN RHD22xx opera con el reloj del SPI, *SCLK*. La frecuencia de este reloj es igual a $f_{sclk} = 24\ MHz$.

## Reset

El periférico utiliza como reset la señal *s_axi_aresetn*. Es una señal activa en bajo asíncrona al reloj, *s_axi_aclk*. El reset se realiza de manera síncrona.

## Flujo de trabajo

Para controlar el periférico programe los bits deseados en el registro *Status (@0x50)*.

Para realizar el proceso de inicialización es necesario escribir primero la configuración deseada en los registros de configuración[3]. Durante este proceso el periférico envía el contenido de estos registros al circuito integrado y realiza la calibración. Además, se copia el contenido de los registros del circuito integrado a los registros del periférico. Al finalizar este proceso, se recomienda la lectura y verificación de los registros del periférico, para comprobar la correcta inicialización del *RHD22xx*. Durante este proceso se sobrescriben los registros de configuración (@0x00-@0x44), la comparación entre el contenido de los registros y la configuración inicial permite comprobar si hay errores en el proceso. En la Tabla 0-1 se muestra un ejemplo de configuración del proceso de inicialización.

*Tabla 0-1:Ejemplo de configuración y uso del proceso de inicialización*

| | |
|---|---|
| READ(@0x50) | Leer registro *Status* y comprobar que el bit *BSY = 0* |
| WRITE(@0x00, confReg0) | Escribir la configuración en el registro de configuración 0 |
| WRITE(@0x04, confReg1) | Escribir la configuración en el registro de configuración 1 |
| ... | |
| WRITE(@0x44, confReg17) | Escribir la configuración en el registro de configuración 17 |
| WRITE(@0x5C, START_INI = 1) | Escribir el bit *START_INI* del registro *Status* para comenzar el proceso |
| READ(@0x50) | Leer registro *Status* y comprobar que el bit *INI = 1* para indicar que el proceso ha comenzado |

[3] El proceso de inicialización implementado sigue la estructura propuesta en "INTAN RHD2000 series datasheet" [1].

Puede escribir los comandos con prioridad en el registro *DirectCMD (@0x48)* y enviarlos modificando el registro de control. Los datos con prioridad recibidos se pueden leer en el registro *DCResp (@0x4C)* cuando se activa la señal correspondiente. En la siguiente tabla, Tabla 0-2, se muestra el proceso a seguir para el envío de este tipo de comandos.

*Tabla 0-2: Ejemplo de uso del proceso de comandos con prioridad*

| | |
|---|---|
| READ(@0x50) | Leer registro *Status* y comprobar que el bit *BSY = 0.* |
| WRITE(@0x48, cmd) | Escribir el comando a enviar en el registro *DirectCMD.* |
| WRITE(@0x50, PRIORITY = 1) | Escribir el bit PRIORITY en el registro *Status* para comenzar el proceso. |
| READ(@0x50) | Leer el registro *Status* hasta que el bit *NEW_DATA_R* = 1. |
| READ(@0x4C) | Leer la respuesta al comando en el registro *DCResp.* |
| WRITE(@0x50, NEW_DATA_R = 0) | Escribir el bit *NEW_DATA_R* = 0 en el registro *Status* para indicar haber leído el nuevo dato. |

Puede realizar la conversión de varios canales eligiéndolos en el registro *ChSel* e iniciando el proceso. Al finalizar la conversión de los canales, el resultado se almacena en los registros *ChR_reg.* Para realizar esta conversión de manera periódica hay que introducir el periodo deseado en el registro *Status (@0x50).* En la siguiente tabla, Tabla 0-3, se muestra un ejemplo de uso del comando, asi como el proceso para ejecutarlo correctamente.

*Tabla 0-3: Ejemplo de uso y configuración del proceso de conversión de canales seleccionados*

| | |
|---|---|
| READ(@0x50) | Leer registro *Status* y comprobar que el bit *BSY = 0* |
| WRITE(@0x54, chsel) | Escribir en el registro *ChSel* la selección de canales a convertir |
| WRITE(@0x5C, OFF_REM)[4] | Escribir el bit *OFF_REM* del registro *Status* con la opción deseada. |
| WRITE(@0x5C, LOOP_TIMER) | Escribir en el registro *Status* el periodo entre la ejecución de las conversiones. |
| WRITE(@0x5C, LOOP_CH = 1) | Escribir el bit *LOOP_CH* = 1 del registro *Status* para iniciar la conversión de manera periódica. |
| WRITE(@0x5C, CH_READ = 1)[5] | Escribir el bit CH_READ = 1 del registro *Status* para iniciar la conversión de canales una sola vez. |
| READ(@0x5C) | Leer el registro *Status* hasta que el bit *END_CH = 1*, indicando que la conversión ha finalizado. |
| READ(ChR_reg) | Leer los registros que contienen la respuesta a la conversión de los canales deseados |
| WRITE(@0x5C, END_CH_READ = 0) | Escribir el bit *END_CH_READ = 0* en el registro *Status* para indicar que ya se han leído los resultados de la conversión. |

[4] En función de si se desea utilizar o no está opción, la instrucción se deberá ejecutar o no.
[5] Este comando **solo** ha de ejecutarse cuando la conversión de canales no se realice de manera periódica, utilizando el temporizador del periférico. En ese caso los dos comandos anteriores **no hay que ejecutarlos**.

## Restricciones del periférico

Como el periférico utiliza pines externos, es recomendable añadir las restricciones de los retardos, basadas en los retardos de la placa sobre la que se implementa. También se recomienda situar los registros de los puertos *MISO, MOSI* y *SCLK* en los PADs añadiendo las siguientes restricciones.

```
1. # Sitúa en el pad los flipflops de MISO, MOSI y SCLK
2. set_property IOB TRUE [get_ports MISO]
3. set_property IOB TRUE [get_ports MOSI]
4. set_property IOB TRUE [get_ports SCLK]
5.
```

Como el periférico está pensado para utilizarse en conjunto con el *RHD22xx* y distintos elementos para su conexión, es recomendable añadir restricciones temporales basadas en los retardos acumulados por los elementos exteriores a la placa.

Debido a los dos relojes utilizados por el periférico, se recomienda la generación de 2 relojes virtuales de $f_{scpilk} = 24\ MHz$ y $f_{clk96} = 96\ MHz$ para configurar las restricciones temporales del periférico.

A continuación, se incluye un ejemplo del fichero de restricciones temporales del periférico:

```
 1. # ****************************************************************************
 2. # Timing
 3. # ****************************************************************************
 4.
 5. # SETTINGS
 6.
 7. # CLOCK del sistema (F = 96MHz)
 8. set sysClockPeriod 10.417;
 9.
10. # Timing de los componentes externos
11.
12. # INTAN RHD2000 timing
13. set tMISO_max 12;
14. set tMISO_min 0;
15. set tMOSI_min 10.4;
16.
17. # Cable delay (v=0.149 m/ns)
18. set cable_max 6.04; # 0.9 m
19. set cable_min 6.04; # 0.9 m
20.
21. # SN65LVDT14
22. #   receiver lvds to lvttl
23. set lvdt14_rec_max 3.8
24. set lvdt14_rec_min 1
25. #   driver lvttl to lvds
26. set lvdt14_drv_max 2.9
27. set lvdt14_drv_min 0.9
28.
29. # -------------------------------------------------
30. # Clocks
31. # -------------------------------------------------
32.
33. # Reloj 96 Mhz
34. create_generated_clock -name CLK96  [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0} ]
35.
36. # Reloj SPI (F=24MHz) CLK96/4
37. create_generated_clock -name SCPICLK -divide_by 4 -source [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0}] [get_ports {SCLK}] -invert
```

```
38.
39. # ------------------------------------------------
40. # SPI  Timing
41. # ------------------------------------------------
42. #  Output timing
43. set_output_delay -max   1.1 -clock [get_clocks CLK96] [get_ports {SCLK} ]
44. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {SCLK} ]
45.
46. set_output_delay -max -datapath_only -clock [get_clocks CLK96] [get_ports {CS} ]
47. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {CS} ]
48.
49. # Camino de captura MISO
50. set miso_id_max [expr
$lvdt14_drv_max+$cable_max+$tMISO_max+$cable_max+$lvdt14_rec_max]
51. set miso_id_min [expr
$lvdt14_drv_min+$cable_min+$tMISO_min+$cable_min+$lvdt14_rec_min]
52.
53. set_input_delay -max $miso_id_max -clock_fall -clock [get_clocks {SCPICLK}] [get_ports
{MISO}]
54. set_input_delay -min $miso_id_min -clock_fall -clock [get_clocks {SCPICLK}] [get_ports
{MISO}]
55. set_multicycle_path 4 -setup      -to [get_clocks CLK96] -through [get_ports {MISO}] -
from [get_clocks {SCPICLK}]
56. set_multicycle_path 3  -hold -end -to [get_clocks CLK96] -through [get_ports {MISO}] -
from [get_clocks {SCPICLK}]
57.
58. # Camino de Salida MOSI
59. set_output_delay $tMOSI_min -clock [get_clocks {SCPICLK}] [get_ports {MOSI} ]
60. set_multicycle_path 2 -setup -start -from [get_clocks CLK96] -through [get_ports
{MOSI}] -to [get_clocks {SCPICLK}]
61. set_multicycle_path 1  -hold -start -from [get_clocks CLK96] -through [get_ports
{MOSI}] -to [get_clocks {SCPICLK}]; # Revisar!!
62.
63. # ------------------------------------------------
64.
```

Todos los valores introducidos como variables deben de ser determinados en función de los elementos utilizados y los retardos de la placa utilizada.

# Referencias

Los siguientes documentos contienen material complementario al presente en este documento.

[1] INTAN Technologies, "Digital Electrophysiology Interface Chips", RHD2000 Series Datasheet, diciembre 2012 [Actualizado agosto 2022]..

[2] ARM, "AMBA(R) AXI and ACE Protocol Specification", junio 2003 [actualizado marzo 2020].

[3] Xilinx (2012), "AXI Reference Guide".

[4] Xilinx (2009), "Mixed-Mode Clock Module (MMC) Module (v1.00a)"

[5] Texas Instruments, "SN65LVDTxx multi-channel LVDS transceivers", SN65LVDT14 Datasheet, abril 2002 [revisado febrero 2019]

[6] Xilinx (2022), "Using Constraints".

# Anexo B. Restricciones temporales

En este anexo se incluyen los archivos *timing.xdc* y *M1_arty7_timing.xdc*.

Estos archivos con formato *xdc (Xilinx Design Constraints)* contienen comandos con semántica de *TCL (Tool Command Language)* y son los encargados de indicar las restricciones físicas, como los pines físicos a los que se le asignan los puertos descritos en VHDL, como las restricciones temporales del circuito.

En el archivo *timing.xdc* se encuentran las restricciones temporales utilizadas para las pruebas del bloque SPI sobre el soporte físico.

```
# nota: https://support.xilinx.com/s/article/63174?language=en_US
# ****************************************************************************
# Timing
# ****************************************************************************

# SETTINGS

# CLOCK de la placa (F= 100MHz)
set mainClockPeriod 10;

# CLOCK del sistema (F = 96MHz)
set sysClockPeriod 10.417;

# Timing de los componentes externos

# INTAN RHD2000 timing
set tMISO_max 12;
set tMISO_min 0;
set tMOSI_min 10.4;

# Cable delay (v=0.149 m/ns)
set cable_max 6.04; # 0.9 m
set cable_min 6.04; # 0.9 m

# SN65LVDT14
#   receiver lvds to lvttl
set lvdt14_rec_max 3.8
set lvdt14_rec_min 1
#   driver lvttl to lvds
set lvdt14_drv_max 2.9
set lvdt14_drv_min 0.9

# ----------------------------------------------------
# Clocks
# ----------------------------------------------------

# Reloj de 100MHz
# create_clock -name MAINCLK -period  $mainClockPeriod [get_ports CLK]; # incluido en
el xdc generado en el IP

# Reloj 96 Mhz
create_generated_clock -name CLK96  [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0} ]

# Reloj SPI (F=24MHz) CLK96/4
create_generated_clock -name SCPICLK -divide_by 4 -source [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0}] [get_ports {SCLK}] -invert

# ----------------------------------------------------
# SPI  Timing
# ----------------------------------------------------
#  Output timing
#    nota: al situar los registros en IOBS, no es necesario configurar el delay de
salida Â¿?
```

```
51. set_output_delay -max   1.1 -clock [get_clocks CLK96] [get_ports {SCLK} ]
52. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {SCLK} ]
53.
54. set_output_delay -max -datapath_only -clock [get_clocks CLK96] [get_ports {CS} ]
55. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {CS} ]
56.
57. # Camino de captura MISO
58.
59. set miso_id_max [expr
$lvdt14_drv_max+$cable_max+$tMISO_max+$cable_max+$lvdt14_rec_max]
60. set miso_id_min [expr
$lvdt14_drv_min+$cable_min+$tMISO_min+$cable_min+$lvdt14_rec_min]
61.
62. set_input_delay -max $miso_id_max -clock_fall -clock [get_clocks {SCPICLK}] [get_ports
{MISO}]
63. set_input_delay -min $miso_id_min -clock_fall -clock [get_clocks {SCPICLK}] [get_ports
{MISO}]
64. set_multicycle_path 4 -setup      -to [get_clocks CLK96] -through [get_ports {MISO}] -
from [get_clocks {SCPICLK}]
65. set_multicycle_path 3  -hold -end -to [get_clocks CLK96] -through [get_ports {MISO}] -
from [get_clocks {SCPICLK}]
66. #report_timing -delay_type max -to [get_cells {SPI_01/RX_DESP_REG_reg[0]}]  -through
[get_ports {MISO}]
67.
68. # Camino de Salida MOSI
69. set_output_delay $tMOSI_min -clock [get_clocks {SCPICLK}] [get_ports {MOSI} ]
70. set_multicycle_path 2 -setup -start -from [get_clocks CLK96] -through [get_ports
{MOSI}] -to [get_clocks {SCPICLK}]
71. set_multicycle_path 1  -hold -start -from [get_clocks CLK96] -through [get_ports
{MOSI}] -to [get_clocks {SCPICLK}]; # Revisar!!
72. # report_timing -delay_type max -from [get_clocks {SYSCLK}] -through [get_ports
{MOSI}]
73.
74.
75.
76. # ----------------------------------------------------
77.
```

En el archivo *M1_arty7_timing.xdc* encontramos las restricciones temporales utilizadas al implementar las pruebas sobre el sistema completo. Se incluyen tanto las restricciones de la interfaz SPI como las del resto del microcontrolador implementado.

```
  1. # ****************************************************************************
  2. # Timing
  3. # ****************************************************************************
  4.
  5. # Master clock frequencies derived from clock wizard
  6.
  7. # ----------------------------------------------------
  8. # Clocks
  9. # ----------------------------------------------------
 10.
 11. # Rename main clocks for clarity
 12. # create_generated_clock -name cpu_clk  [get_pins
{m1_for_arty_a7_i/Clocks_and_Resets/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT0} ]
 13. # create_generated_clock -name qspi_clk [get_pins
{m1_for_arty_a7_i/Clocks_and_Resets/clk_wiz_0/inst/mmcm_adv_inst/CLKOUT1} ]
 14. set cpu_clk "clk_96MHz_M1_mi_clk_wiz_0"
 15. set spi_clk "clk_96MHz_M1_mi_clk_wiz_0"
 16. # ----------------------------------------------------
 17. # Input clocks
 18. # ----------------------------------------------------
 19. # Support upto 20MHz SWD
 20. set swclk_period 50.0
 21. create_clock -period $swclk_period -name SWCLK [get_ports {SWCLK_p9_0}]
```

```
# ----------------------------------------------------
# Internal timings
# ----------------------------------------------------
# The DAP is asynchronous to the CPU, (SWCLK and cpu_clk(clk_out1_M1_mi_clk_wiz_0)).
# However need to ensure that all signals pass across the relevant CDC structures quickly enough
# This should be within 2 cycles of the fastest clock, (cpu_clk).  This is currently 110MHz, ~9ns.
# We only wish to constrain the acutal datapath, we do not need to consider clock skew and jitter
# as these are asychronous clocks
# Set to be less that cpu_clk period for guaranteed transistion times.
set cdc_cpuclk_swclk_max 9.0
set_max_delay -from [get_clocks $cpu_clk] -to [get_clocks SWCLK] -datapath_only $cdc_cpuclk_swclk_max
set_max_delay -from [get_clocks SWCLK] -to [get_clocks $cpu_clk] -datapath_only $cdc_cpuclk_swclk_max

# ----------------------------------------------------
# Debug signals
# ----------------------------------------------------

# Large input Tsu, as clock insertion delay is a lot shorter than datapath input delay.
set sw_in_tsu 10
set sw_in_max_delay [expr {$swclk_period - $sw_in_tsu}]
set sw_in_th  -1
set sw_out_tsu 5
set sw_out_th  -5

set debug_od 5.0
set debug_id 5.0

# SWDIO
# SWDIO is driven at both ends by posedge clk.  The clock is sourced from the DAPLink board
# For input signals it could be either side of rising edge
# For output signals need to ensure the whole round trip is less than the period
set_input_delay  -clock [get_clocks SWCLK] -add_delay -max $sw_in_max_delay [get_ports {SWDIO_p7_0_tri_io}]
set_input_delay  -clock [get_clocks SWCLK] -add_delay -min $sw_in_th [get_ports {SWDIO_p7_0_tri_io}]
set_output_delay -clock [get_clocks SWCLK] -add_delay -max $sw_out_tsu [get_ports {SWDIO_p7_0_tri_io}]
set_output_delay -clock [get_clocks SWCLK] -add_delay -min $sw_out_th [get_ports {SWDIO_p7_0_tri_io}]
set_output_delay -clock [get_clocks SWCLK] -add_delay -min $debug_od [get_ports {SWDO*}]

# ----------------------------------------------------
# Untimed ports
# ----------------------------------------------------
# Following ports have no timing requirement to any output or on-board clock.
# Set to small delays to give timing closure
set untimed_od 0.5
set untimed_id 0.5
# ----------------------------------------------------
#  Virtual slow clocks
# ----------------------------------------------------
create_clock -period 100.0 -name slow_out_clk
set_max_delay -from [get_clocks $cpu_clk] -to [get_clocks slow_out_clk] -datapath_only $cdc_cpuclk_swclk_max
# Use a virtual slow clock for the untimed IO
# UART
set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports uart*rxd]
set_output_delay -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports uart*txd]

# Switch inputs
```

```
 77. set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
*switches*]
 78. set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
push_buttons*]
 79.
 80. # Reset
 81. set_input_delay  -clock [get_clocks $cpu_clk] -add_delay $untimed_id [get_ports
RESET*]
 82. # Prevent reset from timing from cpu_clk to qspi_clk
 83. set_false_path -from [get_ports RESET*] -to [get_clocks $spi_clk]
 84.
 85. # CPU Halted,Lockup
 86. set_output_delay  -clock [get_clocks $cpu_clk] -add_delay $untimed_id [get_ports
{HALT*}]
 87. set_output_delay  -clock [get_clocks $cpu_clk] -add_delay $untimed_id [get_ports
{LOCKUP*}]
 88.
 89.
 90. # Output LEDs
 91. # set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
led_4bits*]
 92. # set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
rgb_led*]
 93. set_output_delay -clock [get_clocks slow_out_clk] -add_delay $untimed_od [get_ports
led_4bits*]
 94. set_output_delay -clock [get_clocks slow_out_clk] -add_delay $untimed_od [get_ports
rgb_leds*]
 95.
 96. # GPIO 8bits
 97. set_input_delay  -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
gpio_8bits*]
 98. set_output_delay -clock [get_clocks slow_out_clk] -add_delay $untimed_id [get_ports
gpio_8bits*]
 99.
100.
101.
102. # SETTINGS
103.
104. # CLOCK de la placa (F= 100MHz)
105. set mainClockPeriod 10;
106.
107. # CLOCK del sistema (F = 96MHz)
108. set sysClockPeriod 10.417;
109.
110. # Timing de los componentes externos
111.
112. # INTAN RHD2000 timing
113. set tMISO_max 12;
114. set tMISO_min 0;
115. set tMOSI_min 10.4;
116.
117. # Cable delay (v=0.149 m/ns)
118. set cable_max 6.04; # 0.9 m
119. set cable_min 6.04; # 0.9 m
120.
121. # SN65LVDT14
122. #   receiver lvds to lvttl
123. set lvdt14_rec_max 3.8
124. set lvdt14_rec_min 1
125. #   driver lvttl to lvds
126. set lvdt14_drv_max 2.9
127. set lvdt14_drv_min 0.9
128.
129. # ---------------------------------------------------
130. # Clocks
131. # ---------------------------------------------------
132.
133. # Reloj de 100MHz
134. # create_clock -name MAINCLK -period  $mainClockPeriod [get_ports CLK]; # incluido en
el xdc generado en el IP
```

```
135.
136. # Reloj 96 Mhz
137. create_generated_clock -name CLK96  [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0} ]
138.
139. # Reloj SPI (F=24MHz) CLK96/4
140. create_generated_clock -name SCPICLK -divide_by 4 -source [get_pins
{CLK_MANAGER_1/inst/mmcm_adv_inst/CLKOUT0}] [get_ports {SCLK}] -invert
141.
142. # ------------------------------------------------
143. # SPI  Timing
144. # ------------------------------------------------
145. #  Output timing
146. #    nota: al situar los registros en IOBS, no es necesario configurar el delay de
salida ¿?
147. set_output_delay -max   1.1 -clock [get_clocks CLK96] [get_ports {INTAN_SCLK} ]
148. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {INTAN_SCLK} ]
149.
150. set_output_delay -max -datapath_only -clock [get_clocks CLK96] [get_ports {INTAN_CS}
]
151. set_output_delay -min  -2.0 -clock [get_clocks CLK96] [get_ports {INTAN_CS} ]
152.
153. # Camino de captura MISO
154.
155. set miso_id_max [expr
$lvdt14_drv_max+$cable_max+$tMISO_max+$cable_max+$lvdt14_rec_max]
156. set miso_id_min [expr
$lvdt14_drv_min+$cable_min+$tMISO_min+$cable_min+$lvdt14_rec_min]
157.
158. set_input_delay -max $miso_id_max -clock_fall -clock [get_clocks {SCPICLK}]
[get_ports {INTAN_MISO}]
159. set_input_delay -min $miso_id_min -clock_fall -clock [get_clocks {SCPICLK}]
[get_ports {INTAN_MISO}]
160. set_multicycle_path 4 -setup      -to [get_clocks CLK96] -through [get_ports {MISO}]
-from [get_clocks {SCPICLK}]
161. set_multicycle_path 3  -hold -end -to [get_clocks CLK96] -through [get_ports {MISO}]
-from [get_clocks {SCPICLK}]
162. #report_timing -delay_type max -to [get_cells {SPI_01/RX_DESP_REG_reg[0]}]  -through
[get_ports {INTAN_MISO}]
163.
164. # Camino de Salida MOSI
165. set_output_delay $tMOSI_min -clock [get_clocks {SCPICLK}] [get_ports {INTAN_MOSI} ]
166. set_multicycle_path 2 -setup -start -from [get_clocks CLK96] -through [get_ports
{INTAN_MOSI}] -to [get_clocks {SCPICLK}]
167. set_multicycle_path 1  -hold -start -from [get_clocks CLK96] -through [get_ports
{INTAN_MOSI}] -to [get_clocks {SCPICLK}]; # Revisar!!
168. # report_timing -delay_type max -from [get_clocks {SYSCLK}] -through [get_ports
{INTAN_MOSI}]
169.
```

# Anexo C. Funciones en código C para la verificación

A continuación, se incluyen todas las funciones implementadas en código C utilizadas para la verificación del sistema, tanto por simulación como sobre hardware.

```c
/**
* @brief
*
* función que lanza el proceso de inicialización.
* Configura los registros y lanza el proceso de inicialización.
*
* @param confreg[18] son los registros de configuración
* @return void
*/
void INI(int confreg[18])
{
      while(RHD2000->Status_b.BSY){
                LedSetColor(LD4,GREEN);
                LedSetColor(LD5,OFF);
      }

      // Escribir en los registros de configuración (0 al 17)
      RHD2000->R00 = confreg[0];
      RHD2000->R01 = confreg[1];
      RHD2000->R02 = confreg[2];
      RHD2000->R03 = confreg[3];
      RHD2000->R04 = confreg[4];
      RHD2000->R05 = confreg[5];
      RHD2000->R06 = confreg[6];
      RHD2000->R07 = confreg[7];
      RHD2000->R08 = confreg[8];
      RHD2000->R09 = confreg[9];
      RHD2000->R10 = confreg[10];
      RHD2000->R11 = confreg[11];
      RHD2000->R12 = confreg[12];
      RHD2000->R13 = confreg[13];
      RHD2000->R14 = confreg[14];
      RHD2000->R15 = confreg[15];
      RHD2000->R16 = confreg[16];
      RHD2000->R17 = confreg[17];

    //  Inicializa RHD2000

    RHD2000->Status_b.START_INI=1;
    while (RHD2000->Status_b.INI)
    {
        LedSetColor(LD4,GREEN);
        LedSetColor(LD5,OFF);
    }
}
```

```c
/**
* @brief
* Función que envia un comando directo al INTAN
* @param cmd comando a enviar
*/
void DirectCMD(int cmd)
{
      while(RHD2000->Status_b.BSY){
                LedSetColor(LD4,GREEN);
        LedSetColor(LD5,OFF);
      }

      RHD2000->DirectCMD = cmd; // Escribimos cmd
      RHD2000->Status_b.PRIORITY=1; // cmd con prioridad
}
```

```
16.
```

```
1. /**
2. * @brief
3. * Función que lee el registo de respuesta directa.
4. * Pone el indicador de nueva respuesta a 0 al leeer
5. * @return resp lectura de la respuesta
6. */
7. int getDCMResp(void)
8. {
9.       while(RHD2000->Status_b.BSY){
10.                LedSetColor(LD4,GREEN);
11.                LedSetColor(LD5,OFF);
12.      }
13.
14.      int resp = RHD2000->DCResp; // Lee el registro de respuesta
15.      RHD2000->Status_b.NEW_DATA_R=0; //Debería funcionar.
16.      return resp;
17. }
18.
```

```
1. /**
2. * @brief
3. * Función que realiza la lectura de los registros de configuración
4. * @param data contenido del registro
5. * @return void
6. */
7. void readRegs()
8. {
9.       char confReg[18] = {RHD2000->R00, RHD2000->R01, RHD2000->R02, RHD2000->R03,
10.                          RHD2000->R04, RHD2000->R05, RHD2000->R06, RHD2000->R07,
11.                          RHD2000->R08, RHD2000->R09, RHD2000->R10, RHD2000->R11,
12.                          RHD2000->R12, RHD2000->R13, RHD2000->R14, RHD2000->R15,
13.                          RHD2000->R16, RHD2000->R17};
14.      xil_printf("\n REGISTROS CONFIGURACION \n");
15.      for(int i = 0; i < 18; i++)
16.      {
17.                xil_printf("%s\n\n", confReg[i]);
18.      }
19. }
20.
```

```c
/**
* @brief
* Función que se encarga de leer todos los registros mediante lectura directa.
*/
void CmdLoop()
{
  int resp = 0;
  xil_printf(">>>> Lectura directa de los registros\n");
  for(int i = 0; i < 64; i ++)
  {
      DirectCMD(0xc000 + 0x100*i);
      resp = getDCMResp();
      if(i-2 >= 0)
      {
                xil_printf(">>>> reg %i: ", i-2);
                if(resp >= 65 && resp <= 90)
                {
                          xil_printf("%c\n", resp);
                }
                else
                {
                          xil_printf("%i\n", resp);
                }
      }
      RHD2000->Status_b.NEW_DATA_R = 0;
  }
}

```

```c
/**
* @brief
* Función que se encarga de configurar un bucle de conversión.
* Selecciona los canales y lo inicia
* @param chSel contenido del registro de selección de canales
*/
void setch(int chSel)
{
  while(RHD2000->Status_b.BSY){
        LedSetColor(LD4,GREEN);
        LedSetColor(LD5,OFF);
  }

  RHD2000->ChSel = chSel;
  RHD2000->Status_b.CH_READ = 1;
}

```

```c
/**
* @brief
* Función que se encarga de leer los 16 registros que contienen el
* resultado de la conversión de los canales deseados
*/
void ReadLoop()
{
	while(RHD2000->Status_b.BSY)
	{
	  LedSetColor(LD4,GREEN);
	  LedSetColor(LD5,OFF);
	}
	int Read[16] = {RHD2000->ChR_reg0, RHD2000->ChR_reg1,
				RHD2000->ChR_reg2, RHD2000->ChR_reg3,
				RHD2000->ChR_reg4, RHD2000->ChR_reg5,
				RHD2000->ChR_reg6, RHD2000->ChR_reg7,
				RHD2000->ChR_reg8, RHD2000->ChR_reg9,
				RHD2000->ChR_reg10, RHD2000->ChR_reg11,
				RHD2000->ChR_reg12, RHD2000->ChR_reg13,
				RHD2000->ChR_reg14, RHD2000->ChR_reg15};
	for(int i = 0; i < 16; i ++)
	{
		xil_printf("Channel %i ", i);
		if(i%2 != 0)
		{
			xil_printf("%i\n", Read[i]&(0xffff0000));
		}
		else
		{
			xil_printf("%i\n", Read[i]&(0xffff));
		}
	}
	RHD2000->Status_b.END_CH = 0;
	xil_printf("%i\n", RHD2000->Status);
}

```

```c
/**
* @brief
* Función que permite configurar el periodo del bucle de conversión
* El periodo es fijo
* @param time Periodo en ciclos de reloj entre ciclos de conversión
* To-Do: Poder pasar el tiempo como parámetro
*/
void setTimeLoop(int time, int chsel)
{
		RHD2000->Status = 0x00FF0000 | RHD2000->Status; // Cargamos tiempo
		RHD2000->ChSel = chsel;			// Introducimos los canales
		RHD2000->Status_b.CH_LOOP = 1;		// Iniciamos
}

```

```
/**
* @brief
* Función que se encarga de comprobar cuando se pone el bucle de conversión de canales
* a funcionar, es decir, cuando el timeout se termina.
* Cada vez que BSY se active significa que comienza un bucle.
* To-Do: Reescribir de forma más precisa
*/
void checkTimeOut()
{
      while(1)
      {
                if(RHD2000->Status_b.BSY == 1)
                {
                          xil_printf(">>>> Bucle de conversion en proceso\n");
                          while(RHD2000->Status_b.BSY)
                          {
                            LedSetColor(LD4,GREEN);
                            LedSetColor(LD5,OFF);
                          }
                }
      }

}

```