Universidad Zaragoza
1542

Trabajo Fin de Máster

Sistema de adquisición de datos electrofisiológicos basado en Intan RHD2000 y FPGA.

Electrophysiological data acquisition system based on Intan RHD2000 and FPGA.

Autor

Rodrigo Lozano Puñet

Director

Isidro Urriza Parroqué

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

AGRADECIMIENTOS

El desarrollo de este proyecto ha sido algo costoso y tortuoso, dónde han abundado los problemas y, al principio, la falta de comprensión de los mismos. Para poder solventarlos ha sido imprescindible la ayuda de mi tutor Isidro Urriza a quién le agradezco la ayuda, la comprensión y las ganas de explicar durante estos meses.

Querría agradecer también a Antonio Velarte, el creador del sistema global en el que este proyecto se engloba. Ha sido un gran compañero con el que trabajar, siempre dispuesto a ayudar y a dar su opinión sobre los problemas.

Por último agradecer a mis allegados que, aunque su contribución no ha sido de manera directa, también han sido apoyo y me han ayudado mucho durante la realización de este Máster.

RESUMEN

Este TFM se encuadra dentro de un proyecto mayor, cuyo objetivo es mantener vivo biopsias de tejido cardíaco. Para esta finalidad, es indispensable la monitorización y actuación sobre las señales e impulsos eléctricos que tienen lugar sobre el tejido. Específicamente, este proyecto consiste en el diseño de una plataforma dedicada a la adquisición de las susodichas señales eléctricas.

Este sistema de adquisición esta basado en una FPGA de Xilinx y un dispositivo Intan. El Intan es un dispositivo especializado en adquisición de señales electrofisiológicas. Este dispositivo es el encargado de la lectura de los electrodos y el encapsulado a una comunicación SPI.

Para la realización de este sistema se genera el paquete de trabajo del que surge este proyecto. Diseño *hardware* y el *firmware* del sistema del adquisición de datos, con su posterior optimización y comprobación del correcto funcionamiento.

Índice

Capítulo 1

Introducción

1.1. Contexto del proyecto

Hoy en día, la causa del 25 % de las defunciones anuales en España son las enfermedades del sistema circulatorio, según el INE (Instituto nacional de Estadística) [1]. Esto lo convierte en la principal causa de muerte, seguida de cerca por los tumores. Por estos datos cobra mayor importancia, aún si cabe, la investigación de estas enfermedades y las distintas patologías que se pueden dar en el sistema circulatorio humano.

Con el auge de las nuevas tecnologías, de las que forman parte el *"Big data"* y diversas técnicas de adquisición y tratamiento de datos surgen nuevas metodologías para la investigación. Esto se puede aplicar a muchos campos, siendo uno de ellos la medicina. Este es el principal foco del grupo de investigación "Biomedical Signal Interpretation and Computational Simulation" (BSICoS). Uno de los proyectos que tienen en marcha este grupo (*Equipment for the chronic electro-mechanical stimulation of cell and tissue cultures that allows their maintenance under biomimetic conditions. Force transducer and controller to program duration and amplitude of mechanical stretch [2]*) consiste en la creación de un sistema capaz de mantener un tejido cardíaco vivo una vez echa una biopsia. Esto se hace con el objetivo de un mejor análisis del comportamiento del corazón y de cómo le afectan diferentes fármacos en el largo plazo. Dentro del proyecto, hay una colaboración con el equipo del GEMP (Grupo de electrónica de potencia y microelectrónica) de la Universidad de Zaragoza. Este grupo es el encargado de realizar la parte relacionada con la electrónica dentro del proyecto.

Este TFM (Trabajo de Fin de Máster) se centra en el desarrollo del sistema de adquisición de datos provenientes de unos electrodos colocados sobre el tejido. Estos electrodos miden el potencial eléctrico que existe entre los canales del tejido.

Para realizar la investigación se realizan biopsias de tejidos cardíacos vivos. En primera instancia y durante la fase de desarrollo del proyecto se utilizan tejidos

cardiacos porcinos, pero con el objetivo de que cuando el sistema ya esté desarrollado poder aplicarlo sobre biopsias cardiacas humanas. Este tejido no es superior a unas decenas de micrómetros lo cual no afecta al donante, pero puede hacer algo más complicada la obtención de señales. Para ver como reacciona el tejido del corazón a los diversos fármacos es necesario mantenerlo vivo. Esto se hará mediante un sistema que tiene como base un estímulo mecánico y eléctrico. Una vez que el tejido se mantiene con vida se puede monitorizar su "salud" a través de su comportamiento eléctrico. Los impulsos cardíacos que suceden en nuestro corazón son debidos a un intercambio iónico entre los canales del corazón [3]. Este intercambio genera una diferencia de potencial de 90mV de pico entre los tramos Q-R del la señal del electrocardiograma que se muestra en la Figura 1.1. No solo se quieren detectar los picos, si no que el sistema tiene que tener suficiente resolución tanto a nivel de potencial como a nivel temporal para poder llevar a cabo una correcta monitorización y análisis.

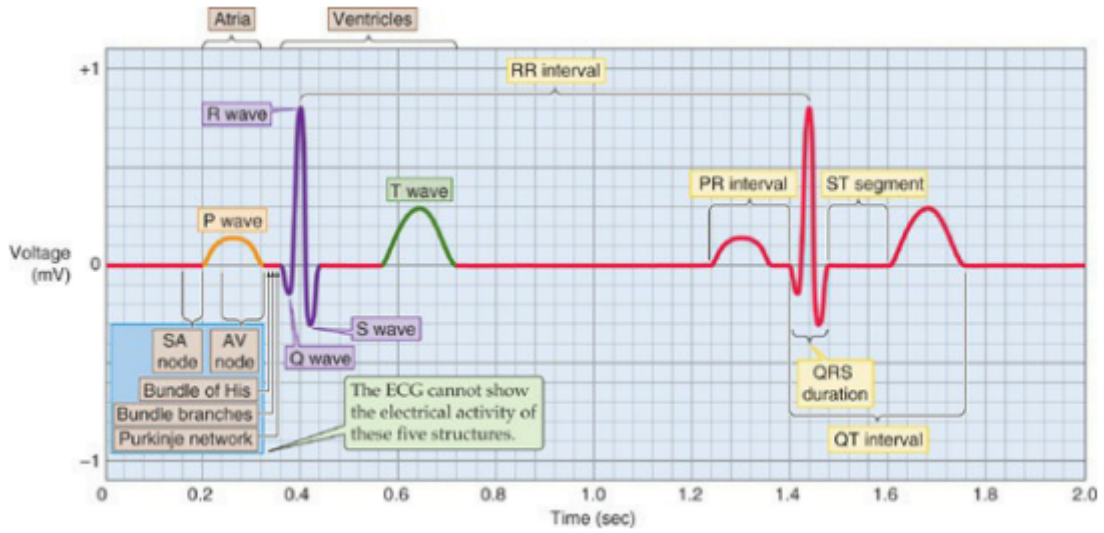

Figura 1.1: Señal de un ECG obtenido de [3]

Este proyecto se hace teniendo en cuenta los Objetivos de Desarrollo Sostenible (ODS) [4] 3.4 (Reducir en un tercio la mortalidad prematura por enfermedades no transmisibles mediante la prevención y el tratamiento y promover la salud mental y el bienestar.), 3.b (Avanzar con el desarrollo y la investigación de vacunas y enfermedades) y el 17.6 (Formar colaboraciones e investigaciones de manera abierta compartiendo conocimiento).

Este sistema ya se ha empezado a desarrollar antes del comienzo de este TFM. Hay un trabajo previo realizado en la elección de los electrodos a utilizar y el posterior diseño de la circuitería analógica posterior para poder obtener una señal de calidad. Además, en la Universidad de Barcelona se está desarrollando el sistema mecánico

encargado de la estimulación mecánica del tejido.

Estos electrónicos irán conectados a un "Electrophysiology Amplifier Chip" de Intan (RHD2000 [5]) para la adquisición de esas señales.

Este es el entorno en el cual comienza el TFM.

1.2. Objetivos

El sistema previamente desarrollado llega hasta la adaptación analógica de la señal a medir. Este TFM se orienta en la creación del sistema de adquisición de datos de tal forma que cumpla con los requisitos temporales y de precisión que requiere la aplicación.

Al Intan le pueden llegar hasta un total de 32 electrodos, que serán muestreados con una resolución de 16 bits. Se tiene frecuencia de muestreo objetivo de 2KHz en cada electrodo. Esto se traduce en una transmisión de aproximadamente 1Mbps. Con esta frecuencia de 2KHz en cada electrodo, las señales obtenidas serán lo suficientemente detalladas para la aplicación que se requiere.

Para la implementación del sistema y la utilización del Intan se han barajado diversas opciones, como un microprocesador específico, un *hard-core* o un *soft-core*. Para llevar a cabo la elección, debido a que queríamos hacerlo en una plataforma flexible como es la FPGA, descartamos la opción del microchip. Con el objetivo de explorar las diferentes opciones y hacer un sistema lo más eficiente posible, nos decantamos por un *soft-core*.

Un *soft-core* es un microprocesador que se implementa utilizando los recursos *harware* de una FPGA. La FPGA será la encargada de transmitir la información obtenida a un dispositivo externo. Existe la opción tanto de trabajar un núcleo de Xilinx (Microblaze) o uno de ARM (Cortex-M3). Utilizaremos el segundo de estos, con el fin de explorar las limitaciones de un sistema basado en este tipo de *soft-core*. Sobre este núcleo se irán montando diversos periféricos encargados de la adquisición y transmisión de los datos.

El diseño del hardware y software necesarios para esta adquisición son el foco principal del trabajo. Teniendo siempre en mente la máxima eficiencia posible con el objetivo de poder hacer un muestreo a 2KHz en los 32 electrodos.

Para conseguir una transmisión de datos rápida y robusta, se elige enviar los datos a través de Ethernet.

1.3. Planificación

Una vez estipulados los objetivos se hace una planificación y se establece una consecución de objetivos parciales para creación el sistema. El diagrama de Gantt del proyecto se muestra en la Figura 1.2. En él se detallan las 5 fases principales en las que se desarrolla el proyecto.

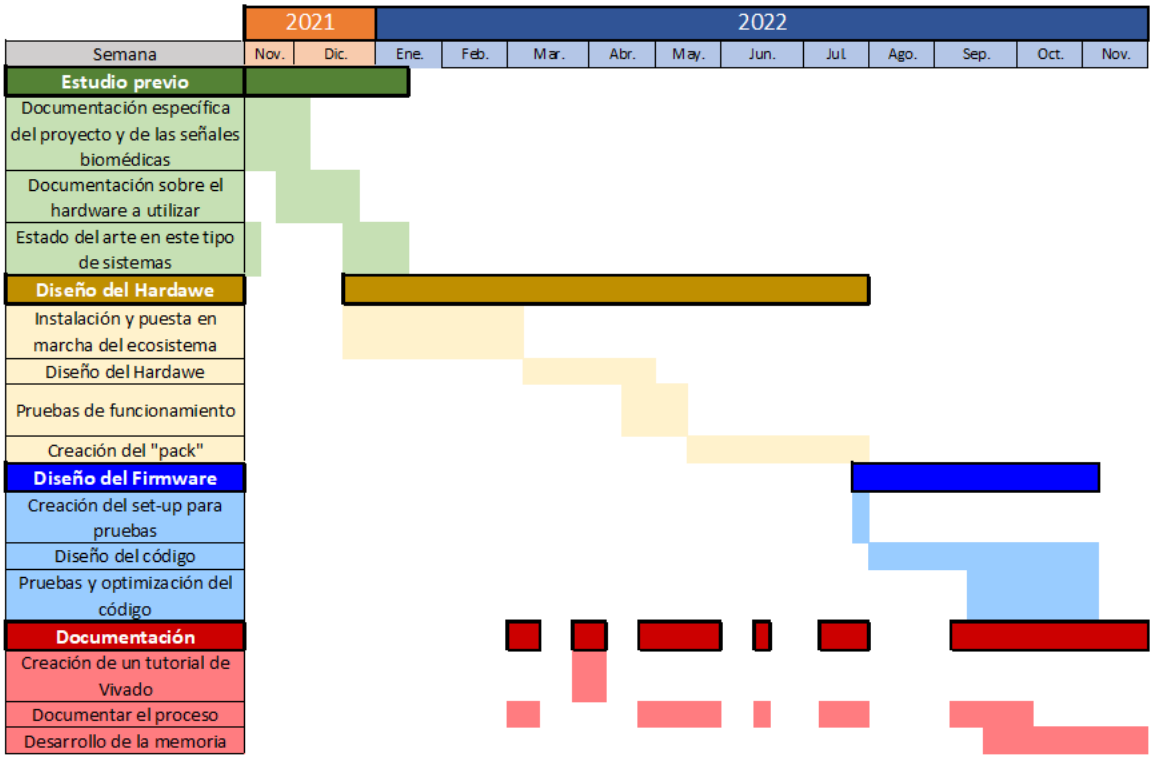

Figura 1.2: Diagrama de Gantt del TFM.

La primera fase consiste, como todo proyecto de investigación y desarrollo, en un estudio del estado previo del sistema y del estado del arte. En este sentido, aunque el entendimiento del carácter de estas señales puede no implicar grandes cambios en nuestro diseño, siempre es importante conocer todas as partes del proyecto. En este caso, los requisitos y objetivos del TFM se establecen según las características de la señal a medir, que son: una resolución de 16 bits y una frecuencia de muestreo de 2KHz. Posteriormente, se realiza el estudio del sistema que se quiere construir como objetivo global del equipo de investigación. El siguiente paso es la documentación del *hardware* que vamos a utilizar, tanto de la arquitectura propia del núcleo y sus características, como los protocolos de comunicación necesarios y las propiedades relevantes del Intan RHD2000. De manera paralela se tiene en cuenta lo que se hace en el estado del arte tanto en lo que respecta en sistemas biomédicos como en la parte más electrónica del sistema. Esto nos dará una visión global del equipo que se quiere montar, los puntos claves del sistema y las zonas más flexibles, dejando claro las prioridades en nuestro

diseño.

Una vez realizado el estudio previo, se realiza el diseño de hardware. Se empieza con una familiarización de la herramienta a utilizar siguiendo diferentes tutoriales [6]. Con estos tutoriales se crean los primeros proyectos sobre la creación y el uso del *soft core*. Posteriormente, se especifica los periféricos a utilizar así como su conexionado creando el diseño *hardware* definitivo. Se pondrán límites temporales a las posibles señales para que no haya problemas de estabilidad. La última fase del diseño *harware* es la creación de un CMSIS-pack (Common Microcontroller Software Interface). Con esto estandarizaremos el formato de la configuración del *harware* y se crearán un conjunto de ficheros para acceder a los periféricos del sistema.

Una vez creado el *hardware*, se pasa a realizar el *firmware*, que contará con unas funciones y ficheros propios para la comunicación con el Intan y otros para la comunicación por Ethernet con el dispositivo externo.

1.4. Herramientas

El sistema de adquisición se desarrollará sobre la FPGA (*Field-Frogrammable Gate Array*) de Xilinx, específicamente, placa de Digilent Arty A7, que se muestra en la Figura 1.3.

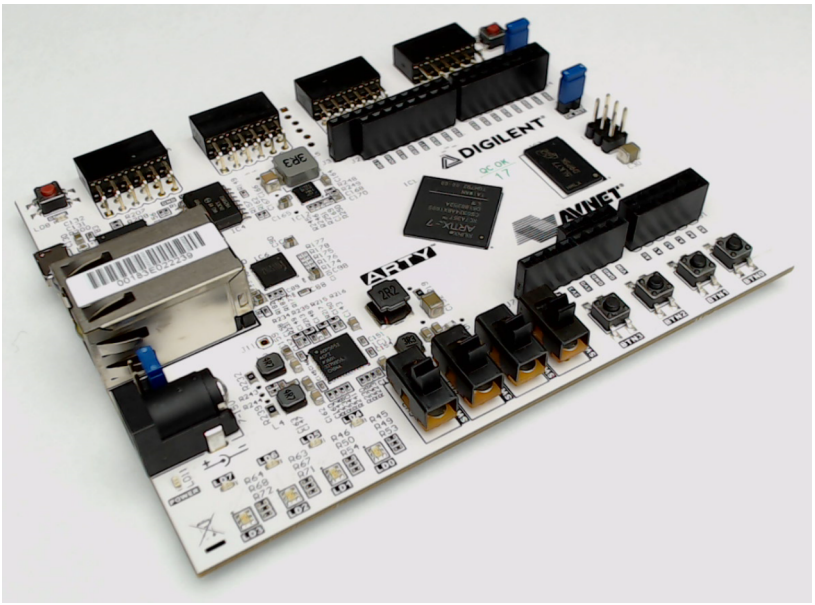

Figura 1.3: Placa Digilent Arty a7-100t.

Para las fases de diseño descritas en la sección 1.3, *hardware y firmware* se utilizarán:

– Vivado 2018.2 para la descripción de hardware. Este es un programa será utilizado con los paquetes proporcionados por ARM en [7].

– Herramientas y plantillas creadas por ARM para la creación de un CMSIS-pack.

– El entorno de programación "Keil-μVision" para la programación y depuración del *firmware*. Este es un programa creado por ARM (diseñadores del Cortex-M3).

1.5. Arquitectura del sistema

El sistema se basa en unos electrodos, que envían las señales y son controlados por el Intan. El Intan se comunica por SPI a una FPGA. Desde la placa de la FPGA se mandan los datos a través de Ethernet para ser recibidos en un ordenador. El flujo de datos y el esquema del sistema se muestran en la Figura 1.4

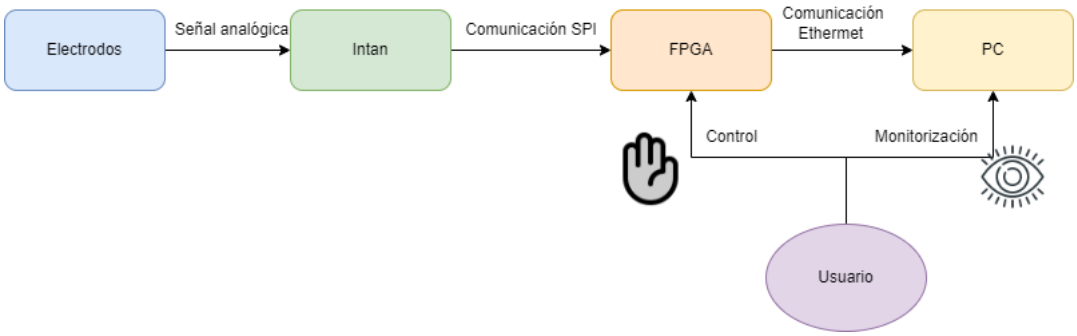

Figura 1.4: Flujo de datos y el esquema del sistema.

Este proyecto se enfoca en el diseño *hardware* y *software* de los bloques de la FPGA y PC de la Figura1.4 y su interacción con el usuario. La elección del dispositivo Intan ha sido elegido con anterioridad, así que sus características propias son requisitos del sistema.

El Intan está desarrollado específicamente para la adquisición de señales electrofisiológicas. Tiene la característica de tener un conversor analógico-digital de prestaciones específicas para el ámbito biológico. El ADC obtiene la señal de un conjunto de AOs (Amplificadores Operacionales) de banda ancha y bajo ruido. Este circuito integrado combina amplificadores, filtros analógicos y digitales. Todo esto trabaja en sincronía para obtener a la salida del circuito integrado las medidas de los electrodos encapsuladas en una comunicación SPI.

Al Intan le pueden llegar como máximo 30 electrodos por los canales principales más otros dos electrodos por los canales auxiliares. Es capaz de muestrear a un máximo de 30KHz en los 32 canales con una precisión de 16 bits. Para nuestra aplicación necesitaremos que muestree a 2KHz.

El sistema de adquisición de datos se basa en un núcleo Cortex-M3 de ARM, el cual está implementado sobre la placa de Diligent Arty A7. Se ha elegido esta placa

debido principalmente a 4 características necesarias para el correcto funcionamiento del sistema:

- Pines a un conector RJ-45 Ethernet, con su consiguiente microchip de adaptación de la capa física (DP83848J) a la comunicación MII *(media-independent interface)*.
- Conector Pmod para la conexión al Intan
- Disponible una comunicación por UART
- Botones e interruptores para una posible interacción con el usuario

A modo de control por parte del usuario, se habilitará el muestreo de datos por medio de un interruptor de dos posiciones disponible en la placa.

Las diferentes partes del proyecto se pueden en la Figura 1.5. Primero se creará el *hardware* necesario y una manera de acceder a la configuración. Se crearán unos drivers que nos servirán como capa HAL (*Hardware Abstraction Layer*) para acceder y configurar los periféricos y por último se creará una capa de aplicación que aplicará la lógica del nuestro sistema.

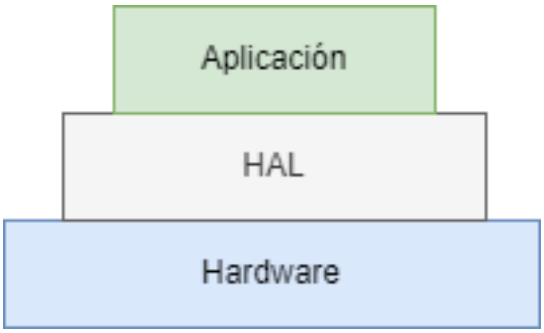

Figura 1.5: Estructura del sistema.

Capítulo 2

Diseño del hardware

Como se ha mencionado en el apartado de herramientas, se utiliza Vivado para el diseño de el hardware.

El diseño del *hardware* podemos subdividir el sistema en varios bloques. En la Figura 2.1 se muestra el diagrama completo de bloques con las subdivisiones en colores según la funcionalidad. En el recuadro rojo se encuentran los bloques relacionados con los relojes y los *resets*, en el amarillo los bloques relacionados con el *Debbugger* (SWD), y por último, en el verde los relacionados con la comunicación AXI con los periféricos.

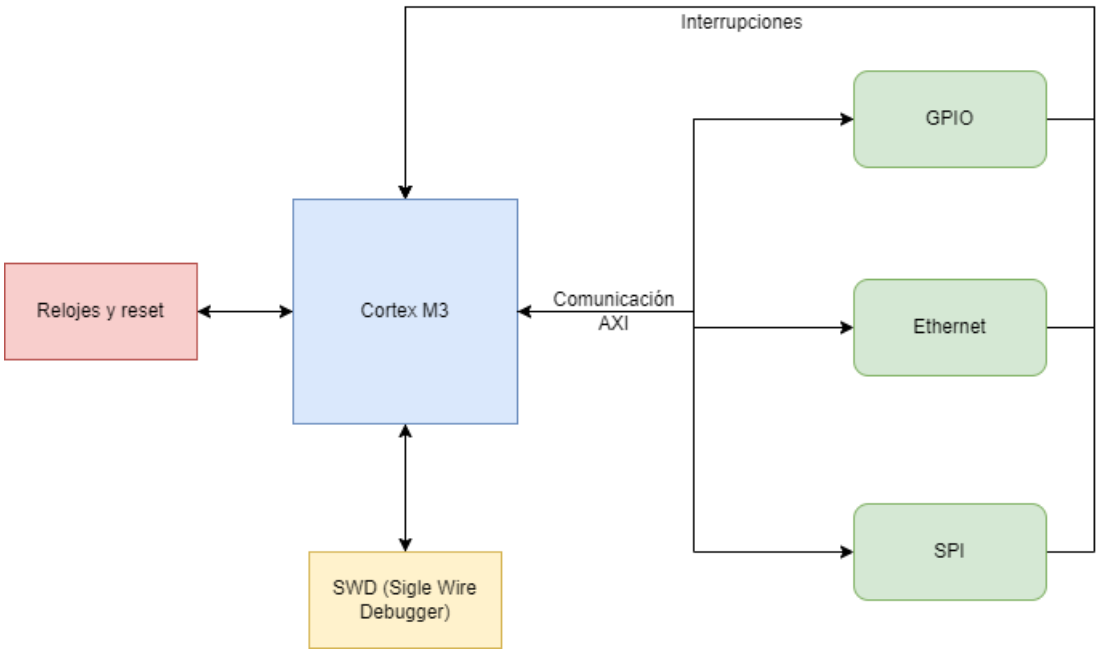

Figura 2.1: Diagrama de bloques del proyecto en Vivado.

El núcleo Cortex-M3 y lo configuraremos de la siguiente manera. Activamos el "Bit-banding"[8]. Se configura para tener 32KB de memoria de programa y de datos.

Segundo, nos vamos a centrar en el bloque de la gestión de los *resets* y los relojes. Si lo analizamos como un sistema en sí mismo, podemos decir que este bloque cuenta como

entradas tres señales externas, que son: el reloj del sistema, el botón de *reset* físico que se encuentra en la placa y la señal "SYSRESETREQ". Las salidas son 3 relojes: uno a 50MHz que irá al núcleo, otro reloj a 25MHz es requerido por el periférico Ethernet y un reloj a 16MHz, que es el máximo que admite el periférico del SPI. Utilizar más frecuencia en el reloj del microcontrolador nos llevará a errores en los tiempos de las señales del sistema en esta placa. Por otro lado, cuenta como salidas 3 *resets* distintos, uno a los periféricos, otro al sistema y otro al pin de *reset* del modo de depuración.

Los resets pueden ocurrir de forma asíncrona, así que primero se han de sincronizar con el sistema. En el caso del botón de *reset* de la placa, se implementará de tal forma que al pulsar el botón, se generará un reinicio tanto en el sistema como en el depurador. Si hay un reinicio proveniente del microcontrolador, se reiniciará los periféricos y el propio microcontrolador. En el caso de tener un reinicio de las dos señales a la vez, se reiniciará también la comunicación AXI.

Los periféricos se conectarán con el núcleo a través de un bloque que nos hace de enlace a la comunicación por protocolo AXI. Utilizamos los IPs de los periféricos SPI, Ethernet y GPIO proporcionados por Xilinx. Los puertos ´clk" de los periféricos se conectan al reloj de 50MHz generado. El reset que se conecta al *reset* de los periféricos es el explicado en el apartado anterior.

Se añade un periférico GPIO para una interacción con la placa por parte del usuario y de una comprobación visual de que el programa está funcionando. Se añade el periférico Ethernet Lite para una comunicación Ethernet de los datos a un dispositivo externo. El periférico SPI es usado por el microcontrolador para adquirir datos del Intan. En el capítulo 3 se profundizará en la comunicación por SPI con el Intan.

Una vez realizada la conexión a los periféricos se define la posición de memoria que ocuparán dentro del Cortex. De esta manera se podrá acceder a los registros de configuración leyendo en las posiciones de memoria determinadas. En el caso del Cortex-M3, el espacio de la memoria dedicado a los periféricos es a partir del 0x4000_0000. Debido a esto, se determina las posiciones de memoria que se muestran en la Figura 2.2.

Se añaden interrupciones provenientes de los periféricos. El número de interrupciones corresponde con el número de periféricos. Estas interrupciones pueden ser configuradas desde el *firmware*.

Una vez descrito el hardware, y enlazado con los pines del microchip, generaremos un fichero de configuración de la FPGA. Este archivo se programará en el núcleo y aplicará la lógica.

Comprobamos si nuestro diseño es realizable y no las señales tienen tiempo suficiente para estabilizarse. En ese aspecto, el "Worst negative Slack (WNS)" es 0.108ns y el

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM3_AXI_0					
CM3_SYS_AXI3 (32 address bits : 4G)					
axi_ethernetlite_0	S_AXI	Reg	0x40E0_0000	64K	0x40E0_FFFF
axi_gpio_0	S_AXI	Reg	0x4011_0000	64K	0x4011_FFFF
axi_quad_spi_1	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
CM3_CODE_AXI3 (32 address bits : 4G)					

Figura 2.2: Posiciones de memoria de los periféricos.

"Worst Hold Slack (WNS)" es 0.071ns en nuestro diseño, que tiene lugar en el núcleo M3. Esto significa que nuestro diseño cumple con los requisitos temporales al ser los dos positivos. Todas las señales consiguen ser estables.

2.1. CMSIS-pack

Una vez hecho el diseño del *hardware* de Vivado y establecidas las direcciones de memoria que van a ocupar los periféricos pasamos a crear el CMSIS-pack. Esto tiene la finalidad de conseguir un diseño modular, que en el caso de requerir una modificación posterior, el cambio sea simple y rápido.

CMSIS es una interfaz que crea una capa de abstracción para los microcontroladores que están basados en los Cortex de ARM. La salida de este proceso es un conjunto de archivos dónde se guardan las descripciones de los periféricos y núcleos de ARM. Esto posteriormente podrá ser leído por el entorno de programación Keil-μVision.

CMSIS-Pack define una manera estandarizada de describir componentes de software, parámetros del dispositivo, información de la placa y el código. La creación de estos archivos se hace siguiendo las instrucciones de ARM en [9].

Existen dos archivos clave en para configurar el pack:

- Un archivo ".pdsc": En el se encuentran detallados los diversos parámetros del núcleo, como el tamaño de la ROM y la RAM, frecuencia de reloj y la posición y gestión de la memoria flash. En este archivo también se detalla posibles códigos de ejemplo para el núcleo o drivers para los periféricos. En nuestro caso no crearemos ni ejemplos de código ni drivers en esta parte. Los dispositivos están ordenados por familia, subfamilia, variante y dispositivo. También se detalla la posición dónde se encontrarán los ficheros que se vayan añadiendo al propio pack.

- Un archivo ".svd": Es un archivo *xml* en el que se detalla parámetros de la memoria como la anchura del bus, la posición de los periféricos así como sus registros.

Una vez creados estos archivos, se crean siguientes ficheros que se muestran en la Figura 2.3

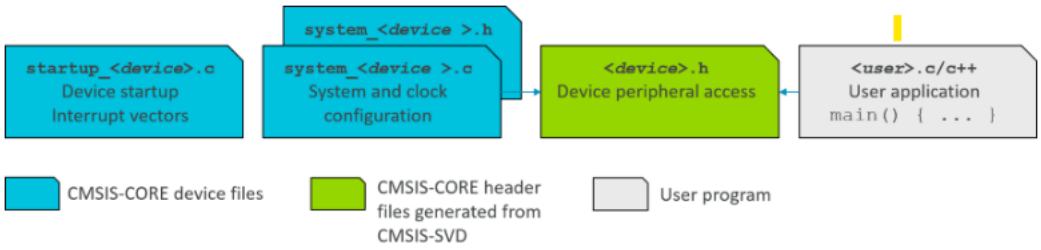

Figura 2.3: Ficheros creados en el CMSIS-pack [9].

– "system_MVCM3xxx.h" y "system_MVCM3xxx.c": En estos ficheros se encuentran las funciones desde el cual inicializa el microprocesador, el *SystemInit*. También se define la variable "__SYSTEM_CLOCK", la frecuencia de reloj del sistema. En nuestro caso tendremos que asegurarnos que está puesta correctamente a 50MHz.

– "MVCM3xxx.h": Se encuentran los recursos del sistema y las máscaras y macros para acceder a los mismos.

– "startup_MVCM3xxx.c": En este ".c" se detallan las interrupciones en el orden de prioridad dentro de un vector.

– "MVCM3xxx_ac6.sct": Es un archivo con la descripción de la distribución de la memoria que se añadirá al proyecto de Keil-μVision.

Una vez creado el pack, damos por finalizada la parte del Diseño de Harware.

Capítulo 3

Aplicación

Una vez realizados el diseño físico y una manera de acceder al mismo, es el momento de desarrollar el código. Para ello se utilizará la plataforma desarrollo Keil-μVision. En ella se incluirán los ficheros del firmware y se configurará para una correcta programación de la FPGA.

Una vez ya se tiene el vehículo de pruebas y se comprueba que funciona correctamente, pasamos a diseñar cómo va a ser la lógica y la estructura de nuestro programa.

3.1. Flujo lógico del programa

Se muestra el flujo del programa en la Figura 3.1.

Solo se realizará el muestreo si el primer interruptor de la placa está activado. De esta manera el usuario podrá tener un control cómodo de la placa.

En primera instancia, se define e periodo de muestreo con un contador propio del núcleo. Esto se hace con el SysTick_Config.

Después se inicializa el periférico del Ethernet deshabilitando sus interrupciones y poniendo a 0 los registros de transmisión y recepción de datos. Seguido a esto se crea una estructura con el esquema de la trama de Ethernet. En ella se escriben la MAC del ethernet del destino, seguida de la MAC propia del chip, que tiene una asignada por defecto. En esta estructura también se define el tipo de ethernet que se explica en [10]. En esta misma estructura es dónde se añadirán los datos a enviar. El resto de parámetros de la trama son añadidos por el propio periférico, esto se detalla en su documentación [11].

El siguiente paso es la configuración del SPI. Aquí se deshabilitan las interrupciones y se configura el SPI como "Master". La comunicación SPI se hace con una anchura de palabra de 16bits, que ha sido especificada en el apartado 2. También pondremos a '1' el "Master_Transaction_Inhibit" que deshabilitará la transacción. Este bit nos servirá

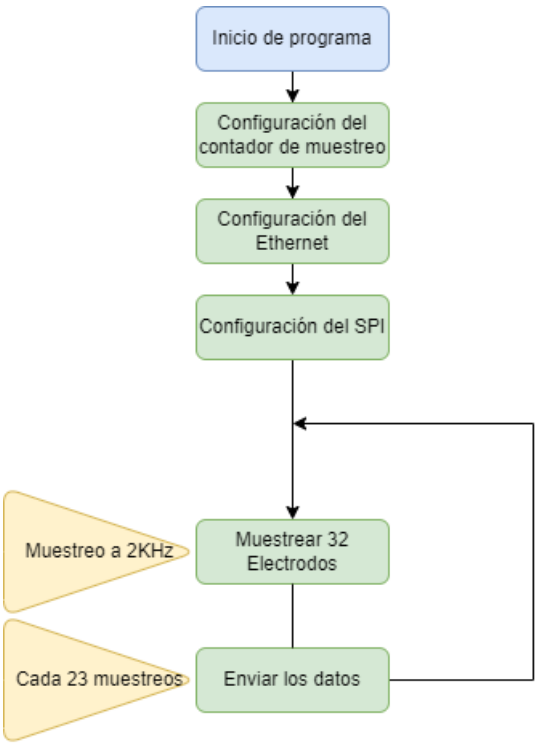

Figura 3.1: Flujo lógico del programa.

como "disparador" de la comunicación.

Una vez llevada a cabo la configuración y preparación previa se entra en el bucle. En él se harán dos tareas: El muestreo de datos y el envío de datos.

Para el muestreo de los electrodos, hay un contador que habilitará el muestreo de datos para que tenga lugar a una frecuencia específica, en este caso, 2KHz.

En el envío de datos, la trama de datos máxima a mandar por Ethernet es de 1500 bytes. Por lo que, se comprueba si se puede almacenar algún muestreo más, cosa que a 64 bytes por muestreo caben 23. Una vez se complete la estructura se enviará.

3.2. Estructura del código

Para que el código sea lo más modular posible, y que un cambio del sistema se pueda aplicar de la manera más rápida posible. De esta manera se propone la estructura de las funciones esenciales que se muestra en la Figura 3.2.

El programa funcionará con un contador interno que hará que entre en la parte del código dónde se muestrea. La principal lógica del programa tendrá lugar en el *main*.

Desde el *main* se llamaran a las funciones de configuración, adquisición y envío de datos tanto del Intan como del Ethernet. El propio *driver* del Intan incluirá las funciones para controlar el SPI, de tal manera que el funcionamiento específico del Intan a bajo nivel no se verá reflejado en el *main*.

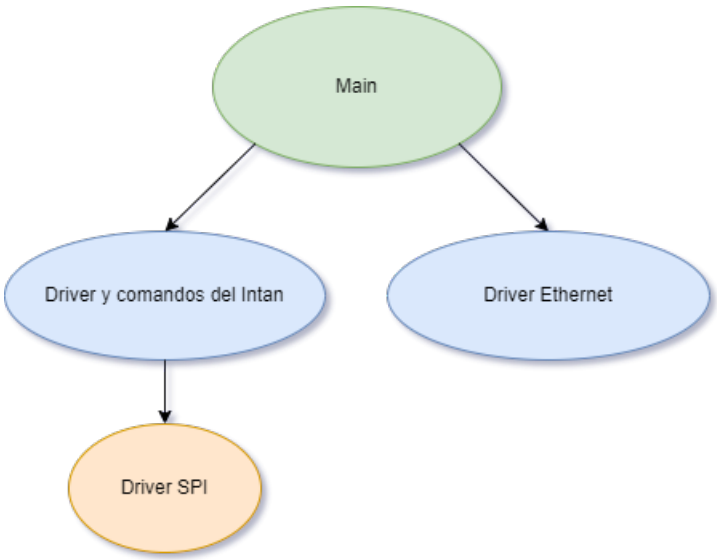

Figura 3.2: Estructura del programa.

Las funciones del Ethernet serán simples en la utilización para que la persona que en un futuro pueda tener que manipular el programa solo se encargue de pasar los datos y la longitud de los datos a enviar. También será necesario conocer la MAC del ordenador de destino, para ello desde la consola de comandos podremos teclear "ip_config" y buscar el apartado de "Ethernet MAC address".

El control de los LEDs y la lectura de los botones e interruptores colocados en la placa no será llevada a cavo por medio de ninguna función. Debido a el fichero *header* hecho con anterioridad en el firmware, se podrá acceder directamente a las posiciones de memoria dónde se ve reflejado el estado de esta interfaz para poder escribir y leer sobre ellos.

3.3. Driver Ethernet

Este controlador se encargará de la gestión de la configuración y del envío de datos por Ethernet. En la Figura 3.3 se puede ver la estructura de la trama de ethernet y en la Figura 3.4 el esquema de funcionamiento del susodicho IP seleccionado en el apartado 2. Esta conexión se hará a nivel de capa OSI2 dentro del modelo OSI de capas.

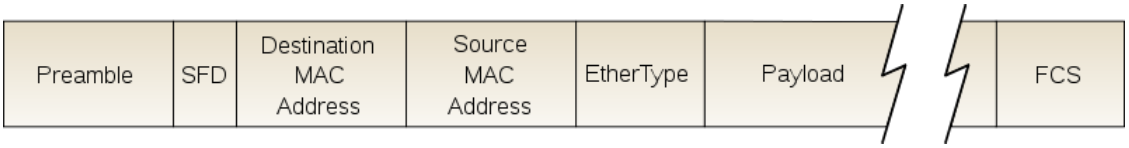

Figura 3.3: Trama de Ethernet. Obtenido de [12]

Con estas dos figuras podemos ver como parte de la trama que se necesita enviar ya

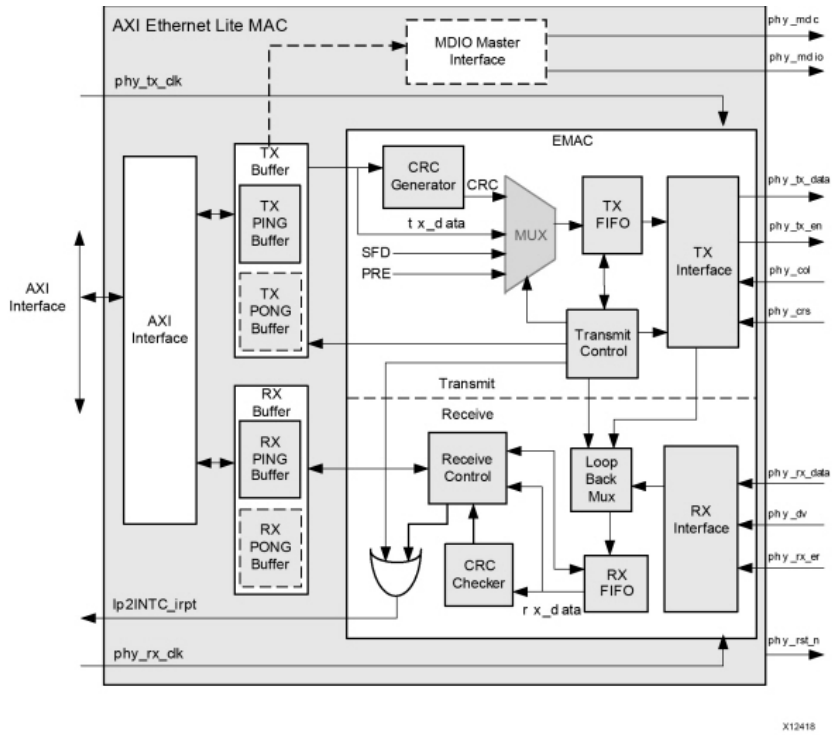

Figura 3.4: Diagrama de bloques del AXI Ethernet obtenido en [12].

es configurada por el propio IP. Tenemos un multiplexor que se encarga de enviar o el *SFD*, *Preamble* y el *CRC* o la trama que se envíen. Por ello nosotros nos encargamos de la escritura de la MAC de origen y destino, del tipo de Ethernet que utilizamos y de los datos a enviar.

Esta trama que está a nuestro cargo será escrita en la posición 0 de la memoria del periférico, como se especifica en su documentación. Debido a la velocidad de transmisión de datos no es necesario el uso de los registros PING-PONG que permitirían un uso más eficiente del periférico. En nuestro caso, utilizaremos solo el registro del PING sobre el cual escribiremos nuestro datos y, habilitando el bit de "TX Ping Control Status" comenzaremos la transmisión. Cuando vayamos a transmitir una nueva trama tendremos que comprobar que la anterior trama ya ha sido enviada y así conseguir no sobre-escribir los datos entre tramas.

El driver del Ethernet constará de la creación de la estructura del *Frame* Ethernet y 4 funciones.

- *ConfigureEthernet*: En esta función se deshabilitan todas las interrupciones propias al envío de datos, se hace un *reset* de los registros donde pudiera haber quedado datos o configuraciones residuales de otros ensayos.

- *DoHeader*: En esta función tiene como entrada la MAC del dispositivo de destino y en él se añaden los datos del tipo de Ethernet que se va a utilizar y la MAC tanto

de destino como de origen de la comunicación que se va a realizar. Estos datos se añaden en un orden determinado por el protocolo a la estructura *Eth_frame* creada.

– *PackFrameData*: Una vez los datos a enviar ya están enviados, esta función teniendo como entrada el *array* de datos y su longitud, los mete dentro de la trama Ethernet. Posteriormente coloca la trama en la posición 0 de memoria del periférico.

– *TransmitRoutine*: Esta función tiene como entrada el la longitud de los datos de la trama, en nuestro sistema este número será constante, pero para hacerlo más modular, en caso de cambiar la longitud a futuro se ha hecho para que sea posible a través de este parámetro. Una vez ya se tiene en la posición de memoria indicada la trama a enviar, en los bytes de "TX_Ping_length " se escribe la longitud de la trama y se pone a 1 el "TX Ping Control Status" que activa la transmisión. se espera a que ese bit se ponga a 0 que indicará el fin de la comunicación para salir de la función.

3.4. Driver SPI

La comunicación al Intan se hará por medio del protocolo SPI. El Intan solo se comunica a través de este protocolo.

La conexión física de los pines de la comunicación se muestra en la Figura 3.5. El pin "SCLK" surge de nuestro diseño hardware y marca la velocidad de reloj del chip SPI. El pin "SS" (Salve select), aunque en nuestro caso solo tendremos un dispositivo conectado nos servirá como bit de marcaje del comienzo y fin de la comunicación. Las señales "MISO" (Master Input Slave Output) y MOSI (Master Output Slave Input) serán por dónde circulen los datos.

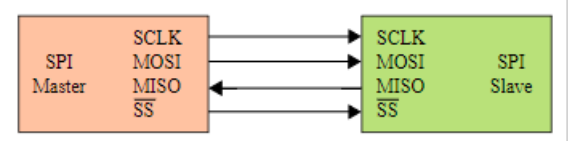

Figura 3.5: Conexión de los pines del SPI [13].

El funcionamiento de la comunicación SPI funciona por desplazamiento de registros. Cuando se transmite una trama del maestro al esclavo, esto provocará que el mismo número de bits transmitidos en esa dirección se muevan también en sentido contrario.

De esta manera podemos decir que no se puede escribir sin leer y viceversa. De esta manera, si ponemos un ejemplo en el que el maestro le envía una ristra de unos al esclavo, que en su registro tiene una ristra de ceros, el estado al empezar la transacción del primer bit sería el estado que se muestra en la Figura 3.6. El resultado final es que se intercambiarían los registros teniendo al final en el registro del esclavo una ristra de unos y en el maestro la ristra de ceros.

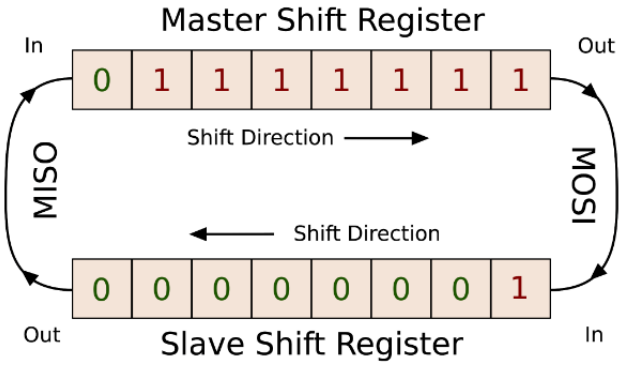

Figura 3.6: Ejemplo de comunicación SPI [14].

Para realizar esta comunicación utilizaremos las funcionalidades del bloque de comunicación SPI de Xilinx. Construiremos dos funciones para comunicarnos.

La primera función inicializa el periférico habilitando los marcadores de interrupción de diversos errores, que utilizaremos para saber si algún registro se ha sobrepasado o sobre vaciado, pero deshabilitaremos las interrupciones asociadas a estos marcadores. Posteriormente escribiremos en el registro de control un reset de los registros de comunicación, habilitaremos el modo maestro, pero dejaremos deshabilitado el bit de transacción de datos. Este último bit es el que utilizaremos como bit de control para el comienzo y el fin de la comunicación. Por último, deshabilitaremos todos los esclavos

La segundo y última función de este driver es el que llamaremos "SPIDataExchange". Tiene como entrada u dato de 16 bits y como salida un dato del mismo tamaño. Este es el tamaño de anchura de palabra que hemos configurado en el diseño de hardware. El tamaño es este ya que el Intan solo admite este tamaño. En esta función lo primero que se hace es comprobar que el registro de datos de salida este vacío, lo cual nos indicará que podemos escribir en él y no sobrescribiremos ninguna comunicación previa. Después de escribir los 16 bits que queremos mandar en este registro, habilitaremos el esclavo que hemos conectado y habilitaremos la comunicación. De esta manera la comunicación se efectuará, y nosotros, esperaremos a que termine comprobando que nuestro registro de envío se queda vacío. Al quedarse vacío, daremos por finalizada la transacción así que la inhabilitaremos para poder leer el registro de datos de entrada sin problema. Antes de leer el registro de datos, comprobaremos que

no está vacío, ya que leer el registro estando vacío provocaría un error. Una vez hayamos leído los datos, deshabilitaremos el esclavo que hemos habilitado y haremos un reset de los marcadores de la comunicación.

Con estas dos funciones ya será posible llevar a cabo la comunicación con el Intan.

3.5. Driver Intan

El dispositivo Intan adquiere los datos de los electrodos y los transmite por una comunicación SPI. El Intan tiene un registro interno en el que pone el lo que tiene que enviar como respuesta a lo que se le está preguntando, pero ese no es el registro que envía en ese intercambio de información, si no que lo enviará en la siguiente comunicación una vez ya haya sido procesado. Esto se ejemplifica bien en la Figura 3.7. La respuesta al comando que se le envía al Intan será recibida dos comunicaciones más tarde.

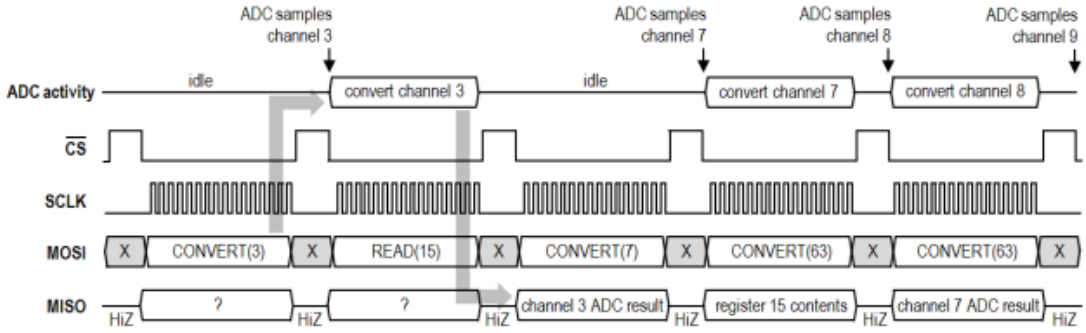

Figura 3.7: Ejemplo de comunicación con el Intan. Obtenido en [5]

Para el control del Intan, se quiere es leer lo que está almacenando en sus registros y su conversor ADC. Se añaden funciones para acceder a los registros y calibración.

En primer lugar vamos a hablar de la función de lectura. El primer paso es preparar los 16 bits que se le pasan al Intan. En los 8 bits menos significativos se escribirá un "0x00" que significa que se quiere leer la información de un registro. En los 8 bits más significativos se escribirá el número correspondiendo de uno de los registros a leer. esta organización se muestra en la hoja de referencia como se muestra en la Figura 3.8. Este registro se le enviará por SPI una vez, pero como la respuesta no la obtendremos en esa comunicación si no dos más tarde, se le vuelve a enviar una comunicación "basura" con el mismo registro para que así nuestro registro del núcleo tenga la respuesta del Intan.

Para la función de escritura se hará algo muy parecido al de lectura. Se cambiará el comando de los 8 bits bajos al dato que se quiere pasar y para no escribir más de la cuenta, se enviará un comando basura que empiezan por "0xC0" para que los registros

Command: READ(R) – Read contents of register R

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
1	1	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	0	0	0	0	0	0	0	0

Result:

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
0	0	0	0	0	0	0	0	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]

Figura 3.8: Registros de lectura del Intan.

roten y así obtener la respuesta de la escritura en el registro de nuestro núcleo. El valor a enviar al Intan para escribir se muestra en la Figura 3.9. Se puede comprobar si el Intan ha respondido a un comando de lectura comprobando que los 8 bits más significativos son todo unos.

Command: WRITE(R,D) – Write data D to register R

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
1	0	R[5]	R[4]	R[3]	R[2]	R[1]	R[0]	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]

Result:

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
1	1	1	1	1	1	1	1	D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]

Figura 3.9: Registros de escritura del Intan. Obtenido en [5]

Por último, la función de lectura de los canales con los electrodos. Los bytes a enviar se muestran en la Figura 3.10. Escribiremos el número de canal que queremos convertir en los 8 bits más significativos y obtendremos el resultado en los 16 bits de respuesta

Command: CONVERT(C) – Run analog-to-digital conversion on channel C

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
0	0	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]	0	0	0	0	0	0	0	H

Result:

MSB 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	LSB 0
A[15]	A[14]	A[13]	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]

Figura 3.10: Registros de lectura de los canales del Intan. Obtenido en [5]

Debido al registro interno que tiene el Intan mientras se procesa la información, a la hora de realizar el muestreo de los 32 electrodos se llevará a cabo una optimización. No enviaremos parámetros "basura" como hacíamos en los anteriores funciones esperando la respuesta del Intan, si no que enviaremos los la conversión de los siguientes canales haciendo así que el Intan trabaje lo más rápido posible y optimizando la comunicación.

3.6. Recepción de datos

Los datos se quieren recibir desde un ordenador, que será conectado directamente a la FPGA. Para poder acceder a los datos que le llegan al ordenador se busca hacer una conexión nodo a nodo entre la placa y el ordenaodor.

Buscamos un programa que nos permita leer las tramas recibidas por el puerto Ethernet y que las filtre por MAC. Se decide leer los datos a través del programa Wireshark, que nos descargaremos e instalaremos a continuación en [15].

Para seleccionar los datos provenientes de la FPGA podemos filtrar por la MAC de nuestra placa (00:0a:35:02:21:8a). Estos datos serán recibidos con el mismo formato con el que lo configuramos en nuestra placa. Tendrá la MAC de destino, la de origen, el tipo de Ethernet y después los datos en "crudo". Esta comunicación debe ser punto a punto, el microchip que lleva la FPGA solo admite este tipo de comunicaciones.

Una vez realizado el ensayo, desde el programa Wireshark se pueden exportar los datos en un JSON que ya podremos leer en Python para la finalidad que queramos.

Con esto ya tendremos los datos en muestreados y listo para el post-procesado en el ordenador.

Capítulo 4

Comprobación de funcionamiento

4.1. Prueba de Tiempos

El objetivo inicial del sistema de adquisición de datos es muestrear con una resolución de 16 bits, 32 electrodos distintos, cada uno de ellos a una frecuencia de 2KHz. Esto resulta en una velocidad de transmisión de datos de 1.024.000 bps.

Para ver la velocidad a la que van cada uno de los bloques del sistema, se hacen test por separado.

Primero se hace una prueba para comprobar la velocidad de la transmisión por Ethernet. En esta prueba se generan unos datos aleatorios que son empaquetados y transmitidos por Ethernet. Se comprueba que la transmisión de Ethernet no es el cuello de botella, ya que es capaz de ir a 32KHz que es el límite que podría dar la comunicación SPI.

La segunda prueba está destinada a conocer la velocidad a la que se pueden transmitir los datos con el Intan. En este caso suprimiremos la comunicación por Ethernet. Comprobaremos los tiempos de adquisición con la ayuda de los cronómetros del depurador y el número de ciclos de reloj necesarios para llevar a cabo la transmisión. Debido a que las pausas con el depurador pueden falsear la medida, se mide los ciclos de reloj antes de muestrear los 32 electrodos y después de muestrearlos. Se tarda 7.1μs en muestrear un canal. Teniendo en cuenta que cuenta que son necesarias 34 transmisiones para un correcto muestreo, nos deja con una frecuencia de muestreo posible de 4KHz. Esta frecuencia está por encima del objetivo de 2KHz.

Debido al cuello de botella que presenta la comunicación por SPI se prueba a hacer un análisis en detalle para saber por qué va a la velocidad que va. Primero se comprueba con un osciloscopio que la velocidad del reloj SPI va a los 16MHz que se le han estipulado en el diseño *hardware*. Efectivamente, la velocidad del reloj es correcta. El siguiente paso es saber dónde se "gastan" más ciclos de reloj. Debido al funcionamiento del depurador, no podemos ir paso a paso midiendo tiempos ya que

al esperar a nuestra iteración con el depurador para avanzar, se falsea la medida de tiempo, así que solo pondremos 3 paradas en la comunicación para la medición. Una parada hasta el inicio de la transacción, después de escribir en los registros los datos a enviar y activar el "Chip Select" del Intan, se tarda 0.8μs en hacer estos pasos. Una segunda parada después de la comunicación, esto se lleva a cabo en 3.86μs. Una tercera parada después de la lectura y de la desactivación del Intan, tarda 0.52μs en realizar estos pasos. Estos 3 tramos del código que representan la comunicación SPI tardan 5.18μs en total. Añadido a estas tareas está el tiempo que se tarda en salir y guardar el dato, 0.64μs, y el llegar otra vez al comienzo de la función 0.14μs. Todo esto hace un total de 5.98μs. Estos tiempos son obtenido poniendo pausas en el código, lo cual falsea la medida en esos tramos. Si lo miramos globalmente la trama tarda en transmitirse 7.1μs. Aunque los valores obtenidos no son demasiado exactos nos permite hacer un análisis cualitativo de cual es el cuello de botella.

Los tiempos medidos en el ensayo muestran que la mayoría del tiempo está siento utilizado en la comunicación y el tiempo restante en la configuración de los registros y gestión de los datos. Sobre el tiempo de la gestión de los registros y las posiciones de memoria no se puede aplicar ninguna optimización mayor, ya que se accede al bit y se escribe sobre este sin ningún tipo de cómputo extra.

El cuello de botella del sistema es la comunicación entre el microcontrolador y el Intan a través del SPI, que limita el muestreo a 4KHz en cada electrodo. Esta comunicación se ve limitada por el funcionamiento del periférico. Lo cual para nuestro sistema es suficiente, el doble de la frecuencia que nos habíamos marcado como objetivo.

Estos datos son obtenidos para la optimización realizada con los mejores resultados.

4.2. Prueba de Muestreo

Se realiza la pruebas de obtención de una onda similar a un electrocardiograma real. Debido a que en el estado en el que se encuentra el proyecto, todavía no se tienen los electrodos para medir ni medidas hechas con los mismos. Se generará la señal con un generador de señales. Este generador se conectará a los canales auxiliares del Intan y se muestreará. El muestreo se hará a 2KHz con una señal comprendida entre los 2 y 0.5V. El resultado del muestreo se puede ver en la Figura 4.1.

Como vemos, la medida es ruidosa y contiene armónicos de alta frecuencia. Se procede a aplicar un filtro paso bajo a la señal para limpiarla. El resultado se muestra en la Figura 4.2.

La señal obtenida, aunque sigue siendo algo ruidosa, es la señal ECG que se quiere medir. A falta de un filtrado más exacto de la señal, se da por un resultado válido del

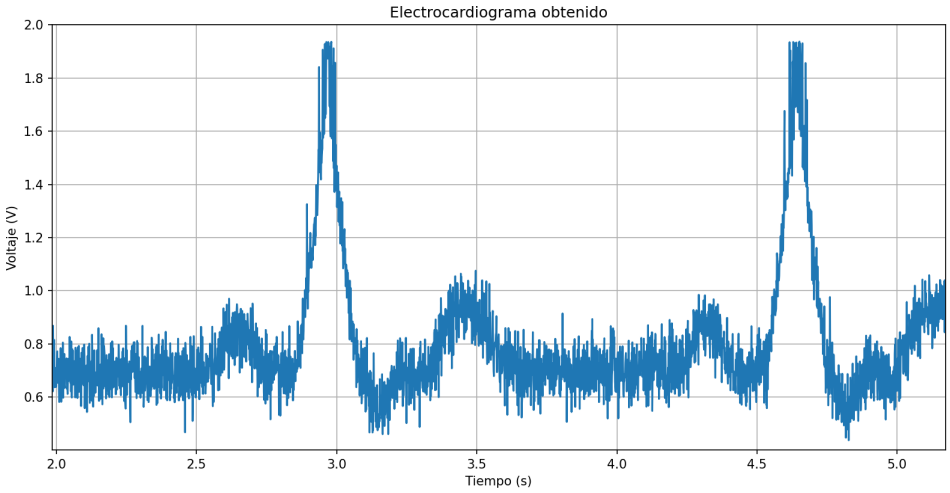

Figura 4.1: ECG muestreado.

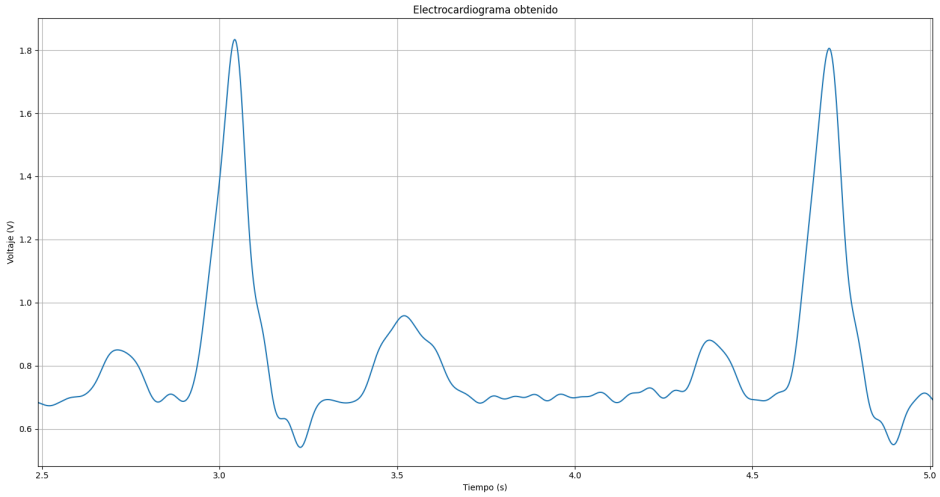

Figura 4.2: ECG tras el filtrado.

sistema.

Capítulo 5

Conclusiones

La realización de un sistema de adquisición de datos como el desarrollado en este TFM es complejo. Se debe bajar a un nivel de detalle con el fin de optimizar el tiempo que hace que surjan detalles que normalmente están encapsulados dentro de las funciones de la herramienta o de las librerías de los fabricantes.

En este proyecto se ha realizado el diseño *hardware* y *firmware* a un nivel de detalle relativamente elevado. Con esto se consiguen muestrear los 32 electrodos a la frecuencia objetivo de 2KHz. Si se quisiera una velocidad de muestre mayor, este sistema llegaría hasta los 4KHz en cada electrodo.

En caso de querer optimizar el sistema a una velocidad de muestreo mayor se propone como futuro trabajo la creación de un periférico propio que esté orientado a nuestra aplicación, al igual que hemos creado nosotros los drivers para obtener un mayor rendimiento. Con esto podríamos hacer un modo ráfaga específico para la característica de las señales en nuestra aplicación.

Capítulo 6

Bibliografía

[1] V. Autores, "Instituto nacional de estadística." https://ine.es/dynt3/inebase/index.htm?padre=8084&capsel=8099, Extraído el 23-09-2022.

[2] BSICoS, "Proyecto cardíaco de bsicos." https://bsicos.i3a.es/equipo-para-la-estimulacion-electro-mecanica-cronica-de-cultivos_celulares-y-tisulares-que-permita-su-mantenimiento-en-condiciones_biomimeticas-transductor-de-fuerza-y-controlador-para-programar-la-d/, Extraído el 26-09-2022.

[3] W. Lederer, "Cardiac electrophysiology and the electrocardiogram," *Medical physiology*, pp. 504–528, 2017.

[4] ONU, "Objetivos de desarrollo sostenible." https://www.un.org/sustainabledevelopment/, Extraído el 23-09-2022.

[5] Intan, "Intan rhd2000 datasheet." https://intantech.com/files/Intan_RHD2000_series_datasheet.pdf, Extraído el 18-11-2022.

[6] Diligent, "Tutorial y proyect base de vivado." https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-vitis, Extraído el 26-09-2022.

[7] ARM, "Arm cortex-m3 designstart fpga-xilinx edition user guide." https://developer.arm.com/documentation/101483/0000/example-software-design/software-update-flow/generating-bit-and-flash-files, Extraído el 18-11-2022.

[8] ARM, "Cortex-m4 technical reference manual: Bit-banding." https://developer.arm.com/documentation/ddi0439/b/Programmers-Model/Bit-banding, Extraido el 24-112022.

[9] ARM, "Pack with device support." https://open-cmsis-pack.github.io/Open-CMSIS-Pack-Spec/main/html/cp_PackTutorial.html#createPack_DFP, Extraído el 24-09-2022.

[10] Wikipedia, "Estructura y tipos de ethernet." https://en.wikipedia.org/wiki/EtherType, Extraído el 02-10-2022.

[11] Xilinx, "Documentación del ip axi ethernet creado por diligent." https://docs.xilinx.com/api/khub/documents/DXkzlCjy38cxQlZIGf4lDA/content?Ft-Calling-App=ft\%2Fturnkey-portal&Ft-Calling-App-Version=3.11.45&filename=ds787_axi_ethernetlite.pdf#page=25&zoom=100,72,700, Extraído el 02-10-2022.

[12] Intan, "Axi ethernet datasheet." https://docs.xilinx.com/api/khub/documents/DXkzlCjy38cxQlZIGf4lDA/content?Ft-Calling-App=ft, Extraído el 18-11-2022.

[13] Intan, "Axi ethernet datasheet." https://es.wikipedia.org/wiki/Serial_Peripheral_Interface, Extraído el 18-11-2022.

[14] Intan, "Axi ethernet datasheet." https://hackaday.com/2016/07/01/what-could-go-wrong-spi/, Extraído el 18-11-2022.

[15] Wireshark, "Wireshark download webpage." https://www.wireshark.org/download.html, Extraído el 24-10-2022.

[16] Zerosquaredio, "Tutorial creación proytecto vivado con cortex-m3." https://github.com/zerosquaredio/arty/blob/main/35t/tutorial1_vivado_2019.1.md, Extraído el 16-10-2022.

Lista de Figuras

Appendices

Apéndice A

Creación de proyecto con Cortex-M3 en Vivado

Este anexo versa sobre la creación de un proyecto en Vivado basado en el Cartex-M3 de ARM. A el programa de Vivado se le añadirán "paquetes" que contienen la información sobre placas y núcleos de Diliget. Para que estos "paquetes" sean compatibles con Vivado tenemos que tener la versión de Vivado de 2018 o 2019.1. Para realizar la preparación nos basaremos en los pasos y los enlaces proporcionados en [16]. De este este Github encontraremos los enlaces para descargarnos: Archivos de las placas base de Diligent, ARM M· DesignStart, un modelo de memoria flash de Micron y otro de Cypress.

Una vez tenemos realizado la descarga y el guardado en una posición conocida de los diferentes ficheros, pasamos a la generación del proyecto Vivado. Iniciamos Vivado y le damos a "Create Project" y guardamos el proyecto en una localización conocida. En la página siguiente a la creación del proyecto, nos saldrá una ventana como la de la Figura A.1. Seleccionaremos lo que se muestra en la susodicha figura.

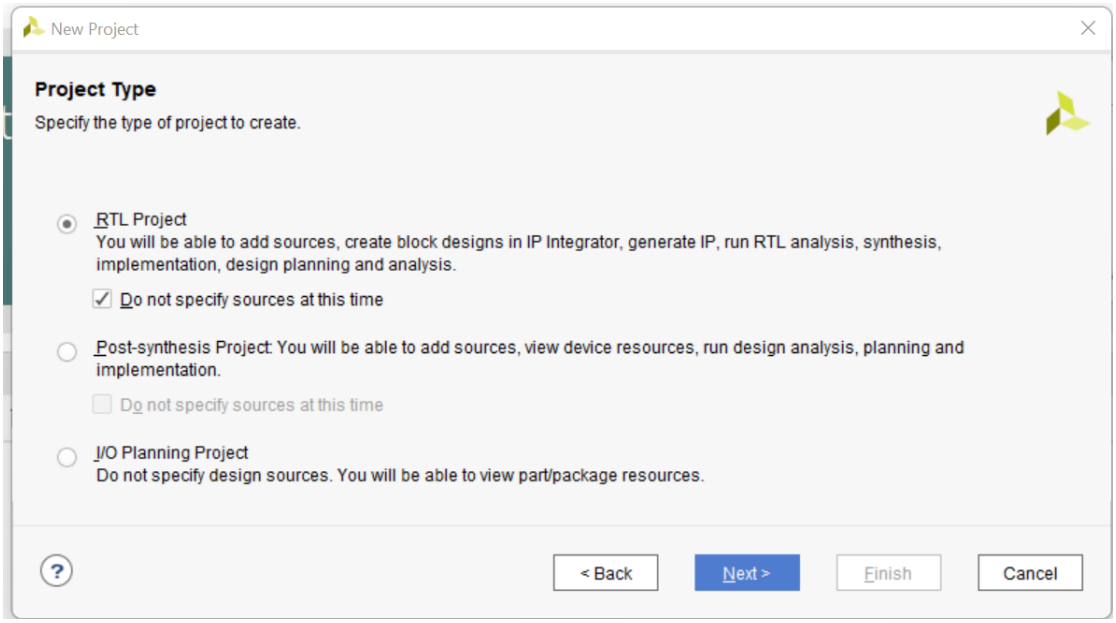

Figura A.1: Configuración del proyecto Vivado.

Una vez pasamos esta página, estaremos en la definición de elementos o placas, en la cual seleccionaremos en nuestro caso el elemento "xc7a100tcsg324-1". Con estos pasos hechos ya tendremos el proyecto Vivado creado.

El siguiente paso es añadir los "paquetes" que nos hemos descargado al principio al proyecto Vivado. Para ello en la barra superior le clickaremos en "Settings". Nos saldrá una ventana como la mostrada en la Figura A.2. Se añade la localización de la carpeta de los archivos de placa Diligent descargados.

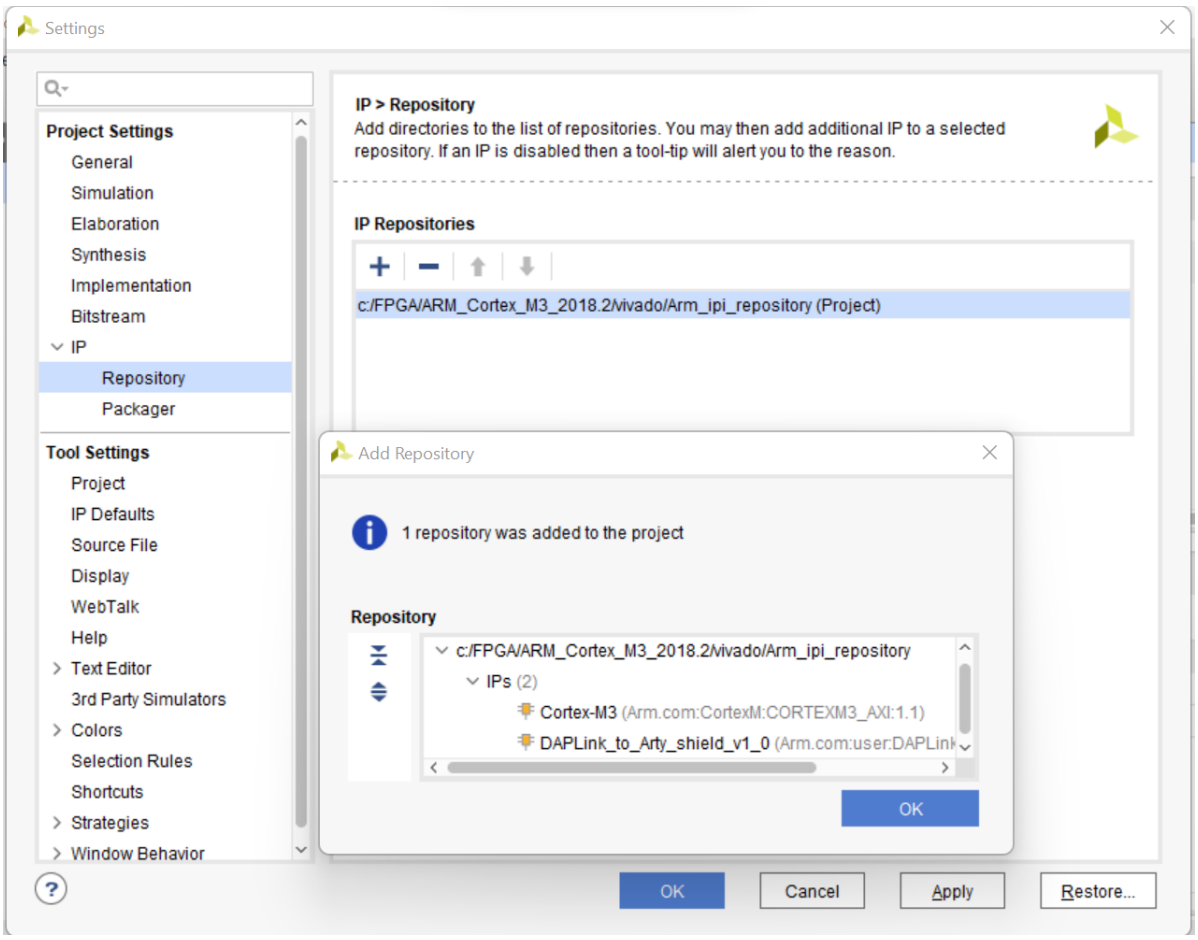

Figura A.2: Configuración del los "paquetes" en el proyecto Vivado.

Una vez puesto en marcha el proyecto con su configuración, podemos pasar al diseño del *hardware*. Esto lo haremos creando un diseño de bloques. Dentro de este diseño estará disponible el botón el "IP integrator". Dentro del diagrama le daremos al botón derecho y añadir bloque en el que podremos añadir los bloques "Cortex-M3", "Clocking Wizard"y "AXI GPIO". Estos serán los bloques que utilizaremos para un proyecto sencillo. Los dos últimos se configurarán como se muestra en la Figura A.3 y la Figura A.4 respectivamente.

Estos periféricos añadidos deben de tener una posición de memoria conocida que se la asignaremos desde el "Address Editor", como se muestra en la Figura A.5.

Posteriormente añadiremos el método de programación que vamos a utilizar, que como se ha visto en la memoria, es el SWD (Single wire debugger). Para ello, le daremos a "Add Sources, Add or create design sources, Add files" y buscaremos el archivo "tri_io_buf.v" dentro de la carpeta con el proyecto base de Diligent que nos hemos descargado, en mi caso esa carpeta se llama FPGA y se encuentra en C:, así que la ruta del archivo es la siguiente: *C:/FPGA/ARM_Cortex_M3_2018.2/hardware/m3_for_arty_a7/block_diagram.* Ahora

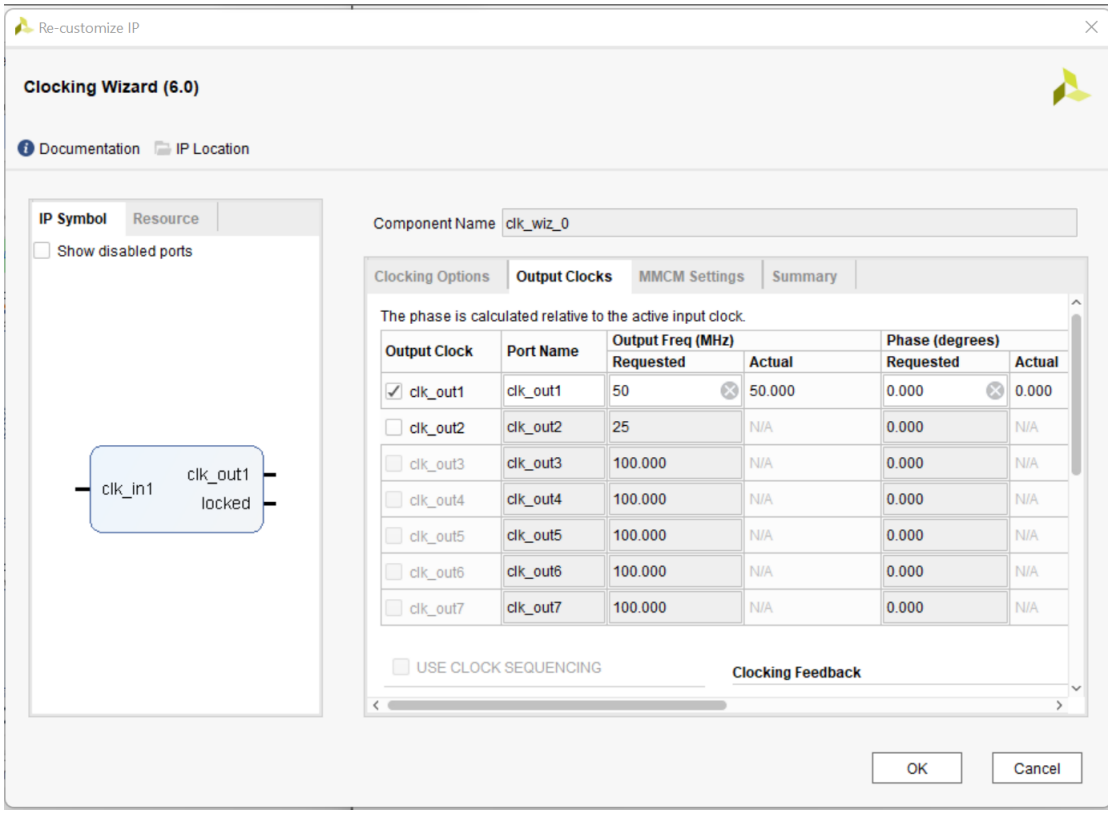

Figura A.3: Configuración de los relojes.

podremos darle a "Añadir módulo" dese el botón derecho en el diagrama de bloques, y veremos el módulo recientemente adjuntado.

Por último añadiremos las restricciones de tiempo en los pines y asignaremos los pines que corresponden al las diversas salidas de nuestra FPGA con el núcleo. Esto lo haremos añadiendo 2 "constraints" que se encuentran en *C:/FPGA/ARM_Cortex_M3_2018.2/hardware/m3_for_arty_a7/constraints*

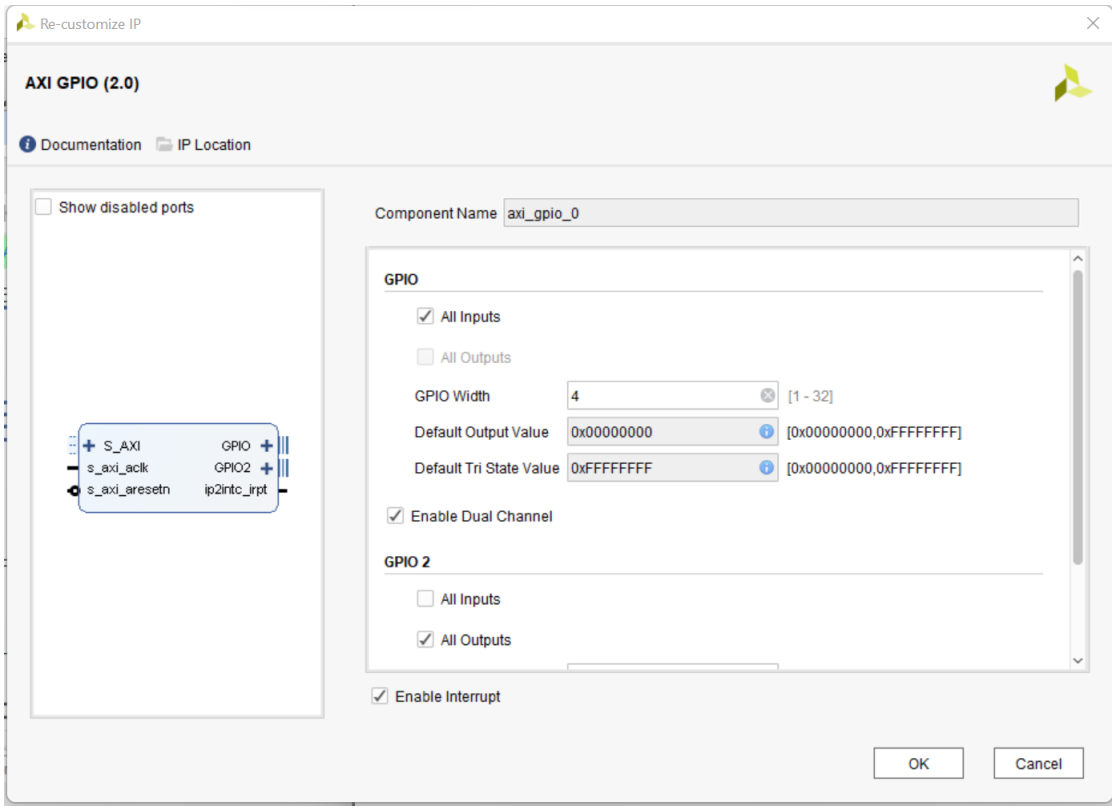

Figura A.4: Configuración del los pines de entrada y salida.

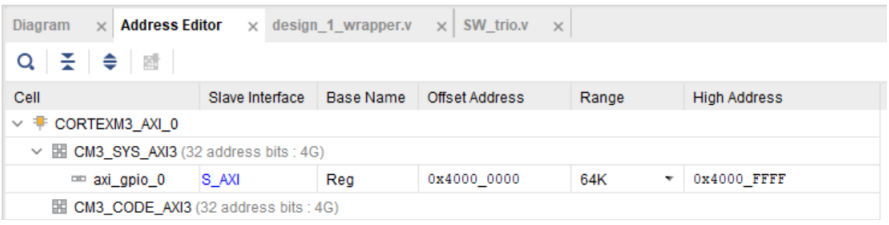

Figura A.5: Ventana de "Address Editor".

Apéndice B

Creación de proyecto Keil para Cortex-M3

En este anexo trataremos la creación de un proyecto Keil una vez se hayan creado los paquetes de nuestro núcleo Cortex-M3.

Primero abriremos Keil y le daremos a "New uVision Project..." como se muestra en la Figura B.1.

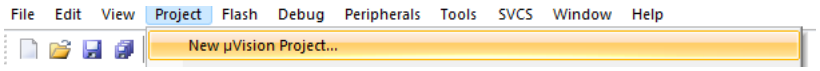

Figura B.1: Creación de un proyecto Keil.

Una vez creado el proyecto, se le añade el núcleo que hemos creado con anterioridad y que se nos mostrará como apareece en la Figura B.2.

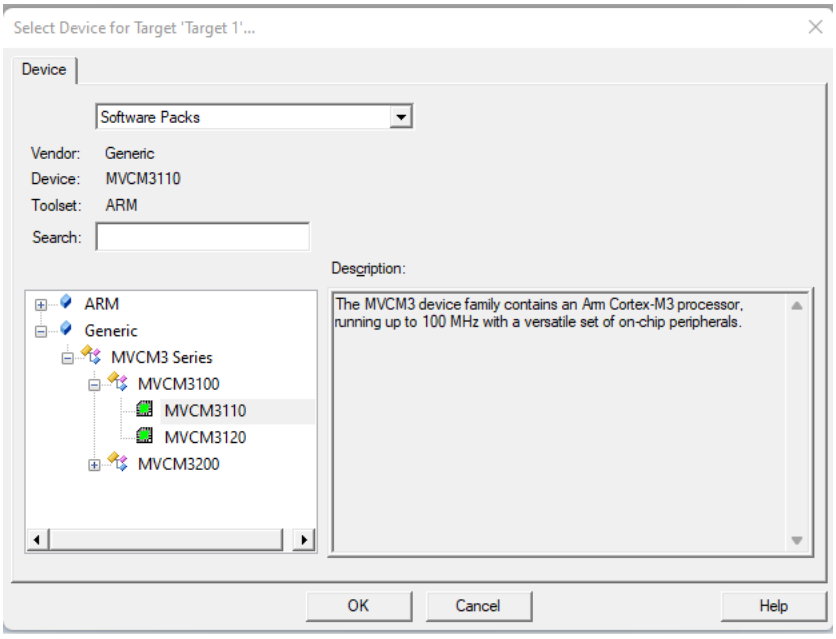

Figura B.2: Selección de núcleo en Keil.

Para que se nos añadan los ficheros en los que estamos interesados, en la ventana que se nos abre después habilitaremos los paquetes que se muestran en la Figura B.3.

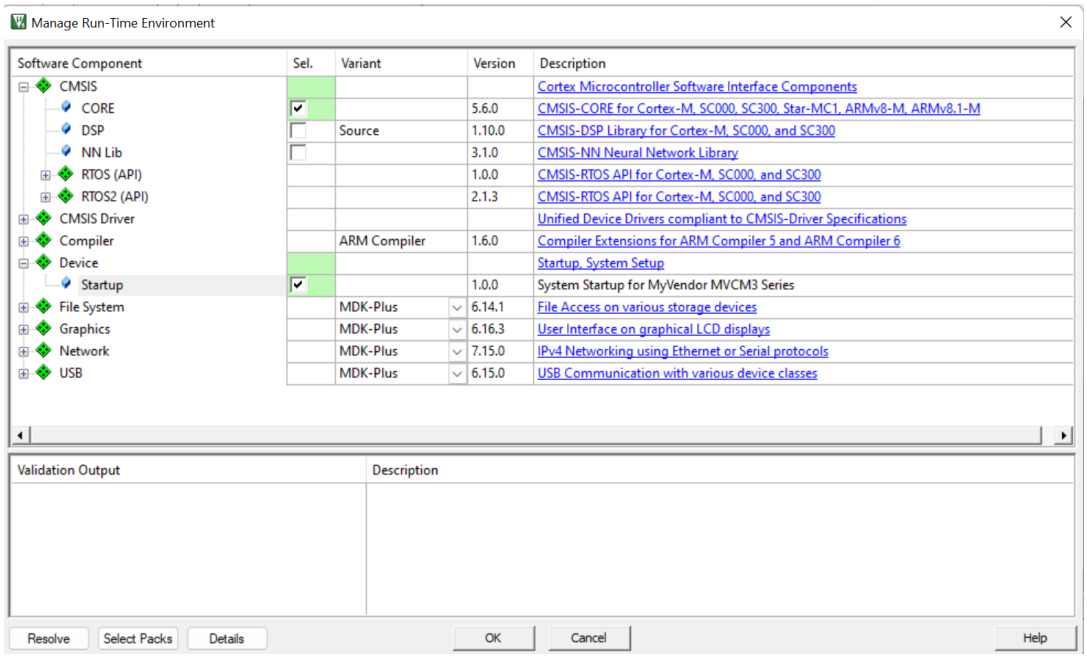

Figura B.3: Selección de ficheros a añadir en Keil.

En este proyecto podremos crear un ".c" que actuará como nuestro fichero principal. A partir de este punto se mostrará el nombre del proyecto y de este fichero como "ProyectoCompleto_28_07".

Para acceder a la configuración propia del proyecto le daremos al icono que es una varita denominado "options for targect". En el poremos acceder a las Figuras B.4, B.5, B.6, B.7, B.8. Para acceder a esta última le deberemos dar a "Settings" dentro de la pestaña "debug", ahí encontraremos la pestaña flash download dónde lo configuraremos como se muestra en la Figura B.8.

El fichero a añadir en la pestaña del linker lo encontraremos en "./RTE/Device/MVCM3110/MVCM3xxx_ac6.sct".

Por último nos debemos de asegurar de 2 cosas:

- El fichero ".h" dento del paquete del núcleo está correctamente actualizado. Este se encuentra en "C:/Users/username/AppData/Local/Arm/Packs/MyVendor/MVCM3/1.0.6/" "Device/Include". Si no se encuentra actualizado lo podremos encontrar en la carptea "SVD" del paquete del núcleo.

- El fichero de "startup.h" se deebe de añadir "typedef void(*VECTOR_TABLE_Type)(void);" en caso de que no lo esté por defecto

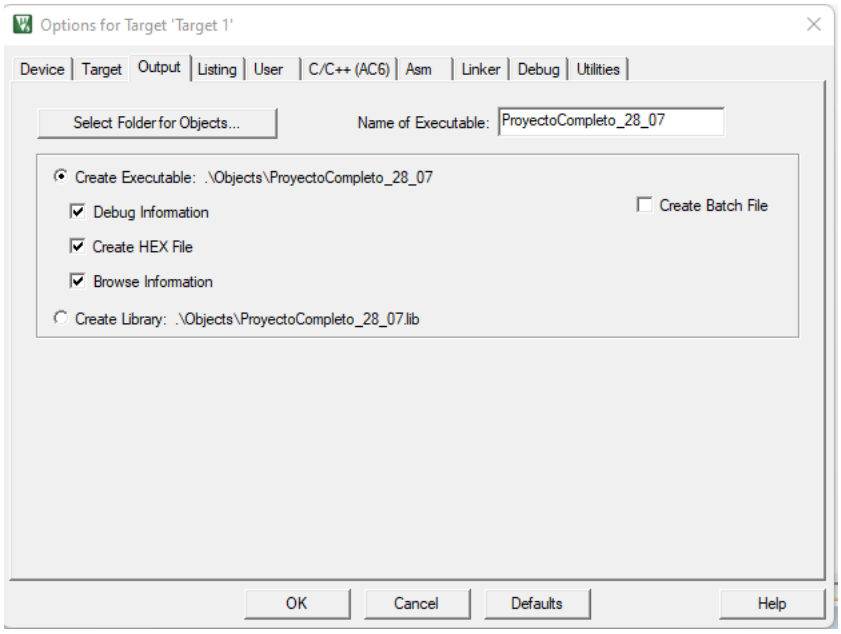

Figura B.4: Selección de salidas del proyecto.

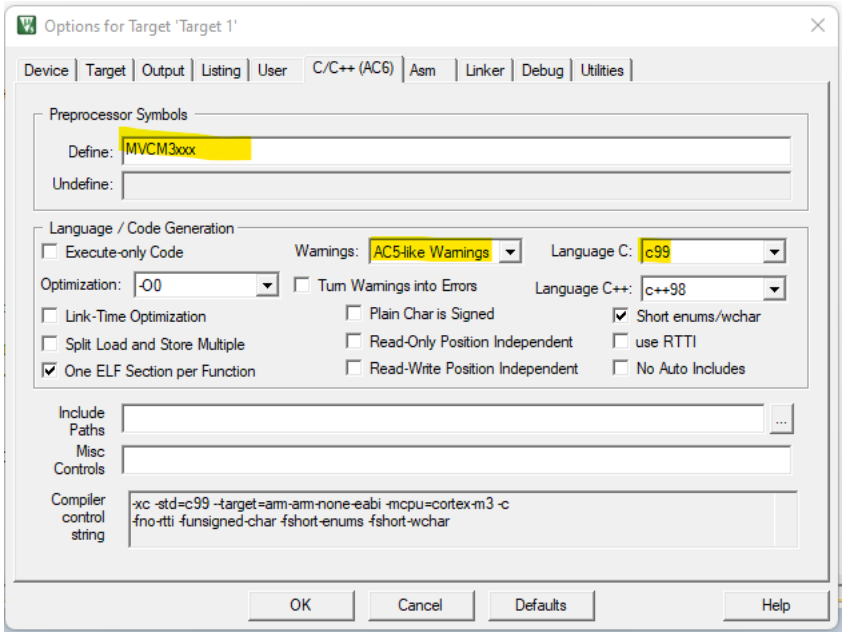

Figura B.5: Configuración del compilador de Keil.

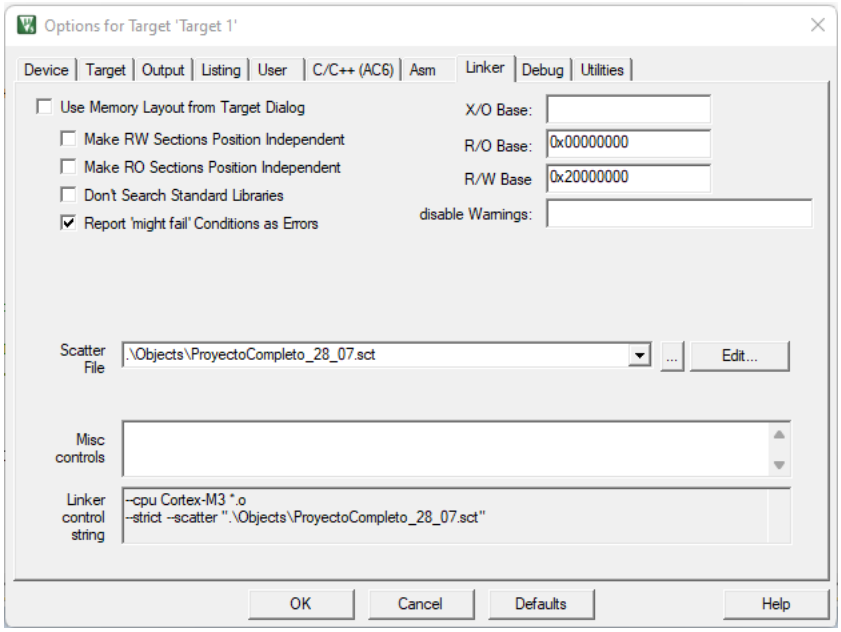

Figura B.6: Configuración de la unión a la placa.

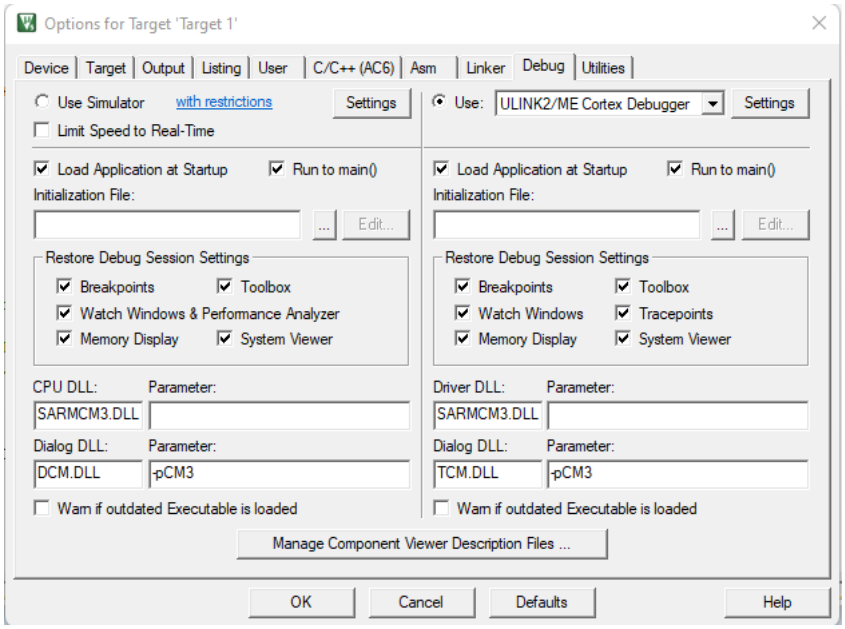

Figura B.7: Configuración del "debugger".

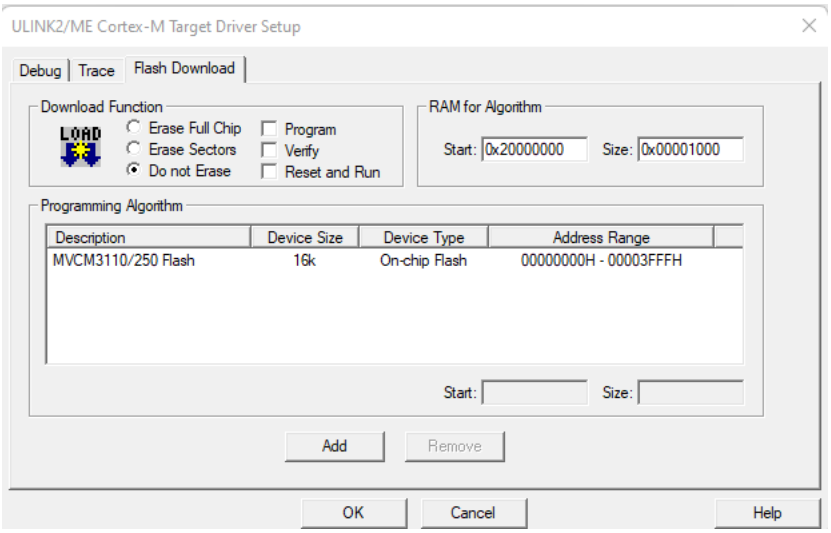

Figura B.8: Configuración de la memoria Flash.