Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación versátil de un microprocesador en FPGA aplicado a un sistema de comunicación de campo cercano (NFC).

Versatile implementation of microprocessor in FPGA applied to a near field communication (NFC) system.

Autor

Christian M. Narváez A.

Director

Isidro Urriza Parroqué

Escuela de Ingeniería y Arquitectura

2023-2024

Resumen

El objetivo de este trabajo de fin de grado ha sido desarrollar un sistema hardware basado en una implementación de un microprocesador dentro de una FPGA (Field Programmable Gate Array), para aplicarlo en el desarrollo de aplicaciones basadas en comunicaciones de campo cercano (NFC). Este proyecto consistirá en la ejecución de dos ramas principales.

Por un parte, se desarrollará un hardware específico, implementado un microprocesador de tipo Soft-Core denominado Cortex-M3 [1], desarrollado por la empresa ARM Holding [2]. Este microprocesador será configurado y adaptado a los requerimientos de la aplicación, realizando simulaciones con Vivado para corroborar los resultados esperados.

Por otra parte, con el objetivo de verificar este diseño hardware, se hará un desarrollo Software que permitirá analizar el funcionamiento del Soft-Core Cortex-M3. Además, se desarrollará una aplicación para inicializar, configurar y detectar la presencia de tarjetas NFC mediante el control del periférico X-NUCLEO NFC03A1 [3], cuyo resultado permitirá validar la flexibilidad del microprocesador para adecuarse a los requerimientos de un periférico externo.

Finalmente, se realizará el montaje del escenario de prueba donde se interconectará la FPGA junto al periférico NFC, donde se podrá observar el funcionamiento mediante interacción por consola y encendido de un LED al detectar una tarjeta NFC.

Tabla de contenidos

1. Introducción

1.1. Contexto

La creciente demanda de soluciones a medida ha llevado al desarrollo de microprocesadores personalizados, que permitan incorporar periféricos adaptados, para llevar a cabo tareas específicas que los periféricos convencionales no permiten realizar.
Este trabajo de fin de grado se desarrolla, con la colaboración del departamento de ingeniería electrónica y comunicaciones, la implementación de un microprocesador embebido en una FPGA (Soft-Core), que permita tener esta flexibilidad para adecuarse a los requisitos de un periférico específico. Así mismo, este tipo de microprocesadores, se presentan como una plataforma de pruebas que permita depurar el desarrollo de un sistema embebido sin la necesidad de su fabricación en silicio, presentando un significativo ahorro económico y logístico a sus desarrolladores.
En el contexto de los microprocesadores Soft-Core, el Cortex-M3 [1] de ARM Holdings, destaca por lograr un equilibrio entre rendimiento y eficiencia energética. Con su arquitectura de 32 bits, facilita el acceso eficiente a datos y memoria, junto con una óptima gestión de interrupciones que proporciona una mayor capacidad de respuesta al sistema.
El Cortex-M3 dispone de su modelo IP (Intelectual property) para el desarrollo de aplicaciones en FPGA del fabricante AMD-Xilinx [2], por lo que facilita su integración con periféricos modificados como UART, SPI e I2C, para comunicarse y controlar periféricos externos.

1.2. Objetivos

Este proyecto de fin de grado tendrá como objetivos el diseño de una plataforma hardware, basada en un microprocesador Soft-Core (Cortex-M3), que permita controlar el integrado CR95HF [4]. Este integrado es capaz de leer tarjetas NFC de distintos tipos, permitiendo recibir o enviar datos mediante su interfaz SPI o UART.
Otro de los objetivos planteados es el desarrollo de una aplicación software que permita verificar el correcto funcionamiento del diseño hardware, permitiendo comunicarnos con el integrado CR95HF [4] y detectar la presencia de tarjetas NFC.
Para lograr estos objetivos se han planteado las siguientes tareas:

- Estudio de las características de un microprocesador Soft-Core Cortex-M3.
- Estudio de las características del integrado CR95HF.

- Aprendizaje del uso de herramientas para el desarrollo hardware proporcionadas por el fabricante ARM, para su uso en la plataforma de vivado [2].
- Aprendizaje del uso de herramientas para el desarrollo de la parte software del proyecto.
- Configuración e implementación de la parte hardware del sistema en función de los requisitos del integrado CR95HF [4] que incorpora el periférico X-NUCLEO NFC03A1 [3].
- Desarrollo de la aplicación software que permita verificar la implementación hardware del sistema.
- Simulación del entorno hardware para verificar la correcta ejecución de las aplicaciones software desarrolladas.
- Desarrollo de la aplicación software que permita la comunicación con el lector NFC para detectar la presencia de tarjetas NFC.

1.3. Metodología y planificación

Las herramientas utilizadas para la ejecución de las tareas definidas son las siguientes:

- **Desarrollo Hardware**: Para el desarrollo hardware se ha empleado el software Vivado [5] de AMD-Xilinx, donde se emplea el lenguaje de descripción hardware VHDL y verilog.
- **Desarrollo Software:** Para el desarrollo software se ha utilizado la plataforma de keil uVision5 [6] y el lenguaje de programación utilizado es C.

Al iniciar el proyecto, se realizó la planificación de la Figura 1, incluida también en la sección en el apartado 9.2 de los anexos para su visualización más detallada.

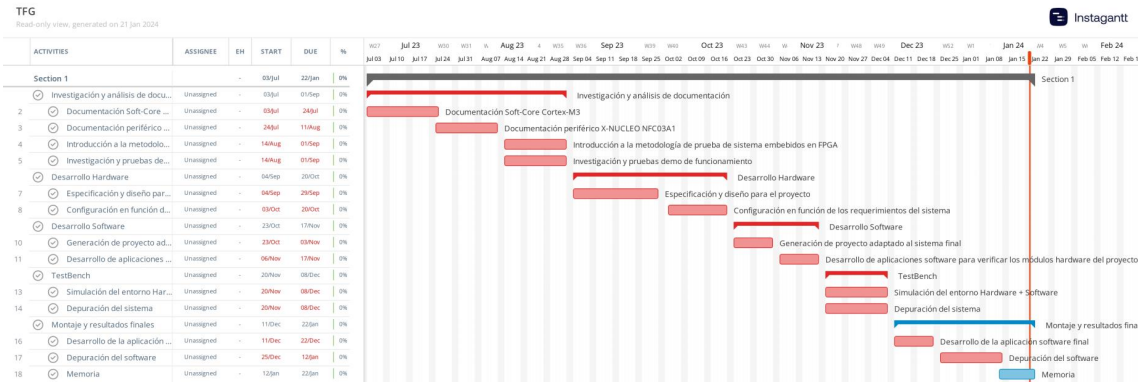


Figura 1: Planificación

La metodología para llegar a cumplir los objetivos propuestos se recoge en los siguientes bloques:

- **Estudio y análisis de documentación:**
  - Documentación Soft-Core Cortex M3: Se analiza la documentación proporcionada por el fabricante ARM para observar las capacidades de este microprocesador.
  - Documentación periférico X-NUCLEO NFC03A1: Se analiza la documentación del periférico para determinar las restricciones del sistema
  - Introducción a la metodología de prueba de sistema embebidos en FPGA: Se analiza el método de prueba propuesto por el fabricante para generar aplicaciones software con su microprocesador Soft-Core.
  - Investigación y pruebas demo de funcionamiento: Ejecución y testeo de los ejemplos proporcionado por el fabricante.
- **Desarrollo Hardware**
  - Especificación y diseño para el proyecto: Se especifican los componentes mínimos que se necesitan incorporar al proyecto para cumplir el objetivo final.
  - Configuración en función de los requerimientos del sistema: Se configuran los componentes del sistema para adecuarnos a los requerimientos finales.
- **Desarrollo Software**
  - Generación de proyecto adaptado al sistema final: Se genera un proyecto en Keil uVision 5 con los recursos hardware de nuestro sistema
  - Desarrollo de aplicaciones software para verificar los módulos hardware del proyecto: Se generan aplicaciones software que permiten verificar de forma individual cada uno de los componentes del sistema.
- **TestBench**
  - Simulación del entorno Hardware + Software: Se simula el sistema para detectar fallos.
  - Depuración del sistema: Se corrigen los fallos detectados tanto en la configuración hardware como en el software.
- **Montaje y resultados finales**
  - Desarrollo de la aplicación software final: Se desarrolla la aplicación destinada a la detección de tarjetas NFC mediante el periférico NFC.
  - Depuración del software: Se corrigen los errores detectados de la aplicación software
  - Memoria: Se elabora la memoria técnica del proyecto

1.4. Contenido de la memoria

El contenido de esta memoria se encuentra desarrollada en 5 apartados que recogen todas las tareas realizadas para cumplir con los objetivos planteados.
En el apartado 1, se realiza una introducción de las herramientas que se utilizarán para desarrollar el trabajo de fin de grado, así como los objetivos propuestos para cumplir con la motivación del proyecto. Además, se especifica la metodología empleada y su planificación.
En el apartado 2, se realiza una lista de los acrónimos utilizados a lo largo de la memoria.
En el apartado 3, se realiza el estado del arte de los tipos de microprocesadores destinados a ofrecer soluciones a medida para sistema embebidos.
En el apartado 4, se caracteriza el microprocesador Soft-Core Cortex-M3 que va a ser utilizado para el desarrollo del proyecto, así como los bloques que componen el sistema.
En el apartado 5, se configura los componentes del sistema hardware para que se adapten a los requerimientos del periférico NFC03A1. Se describe el desarrollo para generar las aplicaciones software que permitirán verificar el sistema en su conjunto. Se realizan las simulaciones del sistema antes de proceder a su montaje y se describe el proceso de asignación de pines entre periférico NFC03A1 y la placa FPGA. Por último, se recogen los resultados de las pruebas.
En el apartado 6, se finaliza la memoria con las conclusiones que se recogen de las pruebas realizadas al sistema y los objetivos conseguidos del proyecto.
En el apartado 7, se enumeran algunas de las líneas de desarrollo futuras tras evaluar los resultados conseguidos en el desarrollo del proyecto.
Finalmente, en el apartado 8 y 9 se recogen las referencias utilizadas a lo largo de la memoria y los anexos que proporcionan información adicional del proyecto.

2. Acrónimos

Abreviatura	Significado
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integred Circuit
IP	Intellectual property
NFC	Near field comunication
SoCs	System on a chip
ASICs	Aplication Specific Integrated Circuits
BSP	Board Support Package
SWD	Serial Wire Debug
FPGA	Field Programmable Gate Array
RISC	Reduced Instruction Set Computer
CPU	Central Processing Unit

3. Estado del arte

Históricamente la plataforma más utilizada para albergar los SoCs (System on a Chip) han sido las ASICs (Application Specific Integrated Circuits), debido a su bajo consumo energético y su gran rendimiento. No obstante, su costoso proceso de desarrollo y fabricación hace que solo sean rentables en el caso de producciones masivas. Las FPGA, por el contrario, al ser dispositivos configurables ofrecen, la posibilidad de implementar diseños personalizados a un coste mucho más reducido [7]. Esto se puede ratificar en la Figura 2, extraída del análisis llevado a cabo por anysilicon [8], donde se puede observar esta relación.

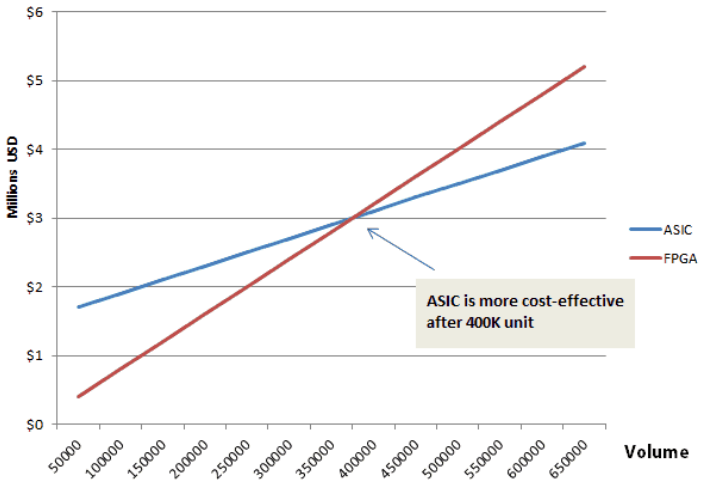


Figura 2: Relación del coste total por volumen de desarrollos ASICs vs FPGA

Los microprocesadores embebidos en FPGA (Soft-Core) destacan por su flexibilidad y personalización para el desarrollo y pruebas de sistemas específicos. Por una parte, esta flexibilidad permite adaptarse a cambios en los requisitos del sistema, reduciendo costes económicos y optimizando los recursos hardware. Por otra parte, la reprogramación de las FPGAs elimina la necesidad de fabricar los microprocesadores modificados, por lo que permite un ahorro significativo de coste económicos, logísticos y temporales. Así mismo, se presenta como una herramienta de depuración para los desarrolladores de microprocesadores, dado que permite testear distintas arquitecturas sin la necesidad de recurrir a costes de fabricación.

3.1. Microprocesadores Hard-Core

Los microprocesadores hardcore son unidades de procesamiento integradas en hardware, diseñadas para ejecutar tareas específicas y embebidas directamente en un dispositivo. En comparación con los microprocesadores de propósito general, los microprocesadores hardcore están altamente optimizados para aplicaciones especializadas, como controladores embebidos o procesamiento de

señales. Dado que están integrados a nivel de hardware en el silicio, proporcionan una mayor eficiencia y reducen el consumo de energía. En contraste con los microprocesadores Soft-Core, son implementaciones no configurables, por lo que no permiten su reprogramación mediante herramientas software, perdiendo flexibilidad, pero ganando rendimiento en la ejecución de sus aplicaciones.

3.2. Microprocesadores Soft-Core

A diferencia de los microprocesadores Hard-Core, los microprocesadores Soft-Core son configurables mediante software y pueden adaptarse para cumplir con requisitos específicos de aplicaciones embebidas, ofreciendo una mayor flexibilidad en sus soluciones.
A continuación, se analizarán diversas opciones de microprocesadores Soft-Core que se encuentran disponibles en el mercado. Estos desarrollos se han llevado a cabo por distintos fabricantes por lo que cada familia de microprocesadores presenta distintas características.

3.2.1. Alternativas en el mercado

Actualmente nos encontramos desarrollos de fabricantes conocidos como AMD-Xilinx, Intel y ARM, por lo que analizaremos las características de estos microprocesadores para realizar una comparación de prestaciones.

MicroBlaze (AMD-Xilinx)

Es un microprocesador Soft-Core para FPGAs de AMD-Xilinx y sistemas embebidos en versiones de 32 y 64 bits [9]. Los desarrolladores eligen este Soft-Core para sus proyectos debido a su alta configurabilidad y adaptación a los sistemas. En la Figura 3, se puede observar este Soft-Core tras importarlo a la herramienta de Vivado.

Figura 3: Ejemplo del Soft-Core MicroBlaze en vivado

Algunas de las características principales de este microprocesador se encuentran recogidas en el documento [9]. Algunas destacables son:

- **Tamaño de la memoria caché configurable:** el tamaño de la caché repercute directamente sobre el rendimiento del microprocesador

haciendo que aumente o disminuya la velocidad de acceso a memoria y en el rendimiento general del sistema. Al permitir ajustar el tamaño de la caché a cada aplicación, se puede buscar un equilibrio entre velocidad de ejecución y utilización de recursos.

- **Unidades de operación:** La elección de las unidades de operación y extensiones afecta directamente a las capacidades del microprocesador en términos de cálculos y procesamiento. Configurar estas unidades de manera adecuada, según las necesidades específicas de la aplicación, puede aumentar la eficiencia de procesamiento y permitir la realización de operaciones específicas de manera más rápida y efectiva.
- **Interfaces de memoria y periféricos:** es posible configurar estas características para garantizar la conectividad y la interacción con los componentes del sistema. Esto abarca aspectos como la velocidad de reloj, los protocolos de comunicación y las capacidades de manejo de datos

NIOS II(Intel)

Familia de microprocesadores Soft-Core, desarrollado inicialmente por Altera y posteriormente continuado por Intel [10]. Se encuentra diseñado para sistemas embebidos y en tiempo real.
Una de las limitaciones de este Soft-Core es que únicamente puede ser utilizado en FPGAs desarrolladas por Intel.
En su documento guía de referencia [10], podemos encontrar los siguientes tipos de núcleos, recogidos en la Tabla 1.

Núcleo	Características destacadas	Aplicaciones principales
Nios II/f	Tamaño pequeño consiguiendo una eficiencia de espacio.	Sistemas de recursos limitados
Nios II/e	Equilibrio entre rendimiento y eficiencia.	Sistemas embebidos en general
Nios II/s	Uso eficiente de los recursos de la FPGA	Sistemas con limitaciones de FPGA

Tabla 1: Familia de microprocesadores Soft-Core de Intel

Algunas de sus principales características son:

- **Alto grado de configuración:** permite ajustar aspectos como la memoria cache, el conjunto de instrucciones y las unidades de operación según las necesidades de la aplicación
- **Gestión de interrupciones y excepciones:** permite establecer prioridades y máscaras a las interrupciones para manejar los eventos de forma más eficiente

- **Conjunto de instrucciones variadas:** soporta instrucciones compactas, incluidas aquellas que requieren de un mayor procesamiento, como pueden ser los cálculos en punto flotante. De esta forma, se puede adaptar de forma más precisa a la aplicación.

Cortex-M (ARM)

Familia de microprocesadores Soft-Core, desarrollado por ARM y diseñados para sistemas embebidos [1]. Cada tipo de microprocesador dentro de la familia M, está diseñado para abordar diferentes necesidades y aplicaciones. Este Soft-Core es soportado en todas las FPGAs independientemente del fabricante.
En la Figura 4 se puede observar este Soft-Core importado a la herramienta de Vivado.

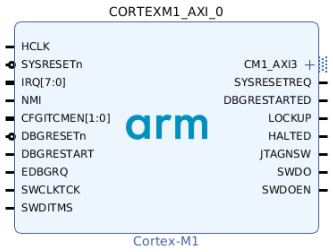


Figura 4: Ejemplo del Soft-Core Cortex M1 desde vivado

En la tabla Tabla 2, se recogen las características y el uso planteado para cada uno de ellos.

Núcleo	Eficiencia Energética	Características	Aplicaciones Principales
Cortex-M0	Alta	Bajo consumo de energía	Sensores y Dispositivos IoT
Cortex-M0+	Muy Alta	Alta eficiencia y ofrece un mayor rendimiento que el M0.	Dispositivos portátiles, Sensores
Cortex-M1	Alta	Integración en FPGAs (Modificable)	Lógica programable en FPGA
Cortex-M3	Alta	Implementa una arquitectura RISC y un pipeline de 3 etapas	Sistemas embebidos
Cortex-M4	Alta	Añade instrucciones de punto flotante y DSP	Procesamiento de señales de audio
Cortex-M7	Alta	Mayor rendimiento y DSP que el Cortex-M4.	Aplicaciones de control

Tabla 2: Familia Cortex-M desarrollados por ARM

Alguna de sus características principales del Cortex-M3, que se encuentran recogidas en el documento [1], son las siguientes:

- **Migración a un sistema ASIC (*Application Specific Integrated Circuit*)**: La familia Cortex-M es flexible para migrar entre diferentes sistemas [2].
- **Arquitectura RISC**: El Cortex-M3 utiliza una arquitectura *"Reduced Instruction Set Computer"* (RISC), lo que significa que se basa en un conjunto de instrucciones reducidas y optimizadas para ejecutar tareas de forma eficiente. Las instrucciones son simples y se ejecutan en un solo ciclo de reloj, lo que permite mejorar la velocidad de ejecución y reduce la latencia.
- **Pipeline de Tres Etapas**: El Cortex-M3 emplea un pipeline de tres etapas en su unidad central de procesamiento (CPU). Esto permite que varias instrucciones se procesen de manera concurrente, mejorando aún más la eficiencia y el rendimiento del procesador. La división en tres etapas (buscando, decodificando y ejecutando) permite una ejecución más rápida de instrucciones.
- **Ahorro de Energía y Optimización para Sistemas Embebidos**: El Cortex-M3 se ha diseñado específicamente para sistemas embebidos con limitaciones de energía. Incorpora características como el modo de bajo consumo y la capacidad de apagar partes del procesador cuando no se utilizan, lo que contribuye a una mejor eficiencia energética.

Tabla comparativa

En la Tabla 3, se recogen las características de cada uno de los microprocesadores Soft-Core.

	Cortex-M	MicroBlaze	NIOS II
FPGA compatibles	Multifabricante	AMD Xilinx	Intel
Versiones	32-bit	32 y 64 bit	32-bit
Migración a sistemas ASIC	Soportado	No soportado	No soportado
Entorno de desarrollo	Vivado (DesingStart)	Vivado	Quartus Prime (Intel)

Tabla 3: Comparativa de microprocesadores Soft-Core

La implementación hardware que se va a realizar en este proyecto de finde grado será un Cortex-M debido a alta compatibilidad con múltiples fabricantes de FPGAs y porque nos proporciona las herramientas necesarias para trabajar con

vivado mediante el software DesingStart [11]. Dentro de la familia M se selecciona el M3, pero los resultados conseguidos en este proyecto también son replicables tanto con un M1 (sin realizar modificaciones) como con el Soft-Core MicroBlaze (realizando modificaciones).
En el apartado 3.3 se describen algunas de las características de la arquitectura ARM y en el capítulo 4 se detallará más en profundidad este Soft-Core.

3.3. Arquitectura ARM

La arquitectura ARM intenta ofrecer principalmente tres tipos de perfiles al desarrollo de un sistema:
A (Application): Alto rendimiento ofreciendo un diseño que permite correr aplicaciones complejas de un sistema.
R (Real-Time): Sistemas con requerimientos de tiempo real a la hora procesar sus servicios
M (Microcontroller): Microcontrolador de dimensiones reducidas con alta eficiencia.

Existen varias familias de microprocesadores definidas, permitiendo que cada tipo de integrado se especialice en un conjunto específico de funciones. Esto asegura que el sistema en su totalidad satisfaga los requisitos funcionales establecidos inicialmente. Por lo que es común hallar sistemas compuestos por múltiples microprocesadores que incorporan la arquitectura ARM, como se puede observar en la Figura 5.

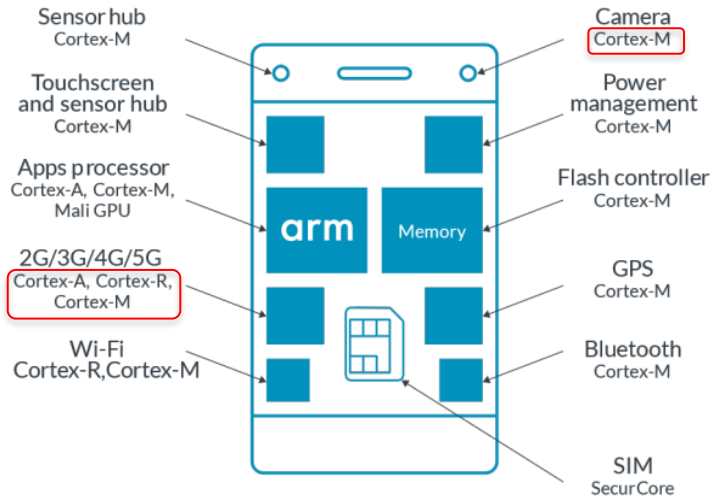


Figura 5: Esquema conceptual de un sistema que implementa diferentes perfiles de la arquitectura ARM

4. Cortex-M3

El microprocesador Cortex-M3 se destaca por su eficiencia y bajos consumos de energía, lo que lo convierte en una buena opción para sistemas embebidos con aplicaciones específicas.
Se encuentra diseñado para optimizar los recursos hardware, por lo que destaca en la capacidad de mantener bajos consumos sin comprometer la velocidad de respuesta, gracias a la implementación de un controlador de interrupciones vectorial anidado (NVIC), como se observa en la Figura 6.
En un sistema de interrupciones genérico, tras detectar la interrupción, el microprocesador distribuirá la petición hacia el controlador global de interrupciones, quien le proporcionará la dirección de memoria de la interrupción a donde debe dirigirse. En este caso, el bloque NVIC le indicará directamente esta dirección, sin la necesidad de realizar otra petición adicional. Además, siempre se mantiene la prioridad a cero de cualquier interrupción (prioridad máxima) asegurando una respuesta rápida ante cualquier evento.

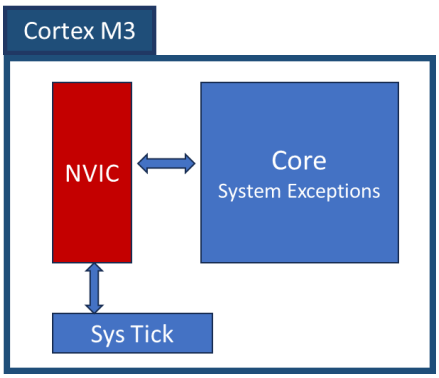


Figura 6: Composición del núcleo Cortex-M3

Para depurar, hacer trazabilidad y analizar el microprocesador Cortex-M3 se dispone de tres bloques dedicados a estos usos [1]:

ETM (Embedded Trace Macrocell): Proporciona la capacidad de rastrear y analizar la ejecución en tiempo real, capturando información detallada sobre las instrucciones y el flujo de ejecución del software.

AHB TRACE MACROCELL: La macro celda de rastreo AHB permite capturar información sobre las transacciones y el flujo de datos en el bus AHB, permitiendo observar la comunicación entre los diferentes componentes del sistema.

ADVANCED HIGH PERFORMANCE BUS ACCESS PORT (AHB-AP): permite la conexión entre las unidades de depuración y rastreo, como la ETM, al bus AHB. Mediante esta interfaz se accede a los registros internos de las unidades de depuración, permitiendo la configuración y el control de la ETM.

4.1. Opciones de configuración del Cortex-M3 DesingStart

ARM proporciona el programa DesingStart [11], que nos permite utilizar las versiones de los microprocesadores Cortex M1 y M3 con las herramientas de Xilinx. En su interior contiene:

- **Arm_ipi_repository:** Repositorio de librerías IP para importar en Vivado.
- **Arm_sw_repository:** Repositorio que proporciona los ficheros BSP (Board Support Package) del Cortex-M3 y un ejemplo de desarrollo de aplicación.

De esta forma, se podrá explorar las opciones de configuración de este Soft-Core al importar su librería en Vivado. A continuación, se enumeran sus opciones.

Vector de interrupciones (Number of interrupts)

El vector NVIC de interrupciones actúa como una tabla que apunta a las subrutinas de interrupción, permitiendo gestionar prioridades durante su ejecución. Este vector puede trabajar con hasta 240 interrupciones. En la Figura 7, se puede observar los parámetros de configuración relacionados con los vectores de interrupción.

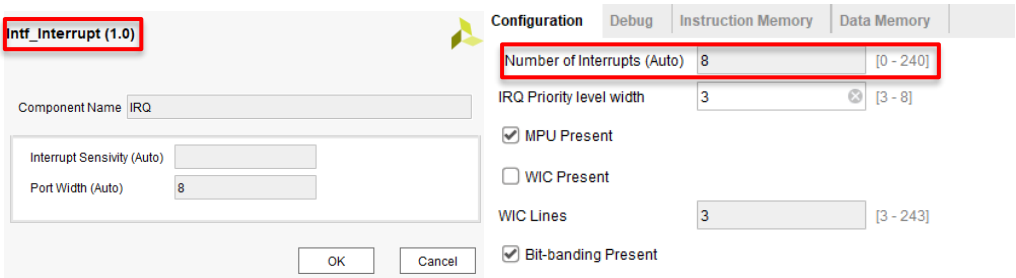


Figura 7: Vector de interrupciones Cortex-M3 DesingStart

Unidad de protección de memoria (MPU)

Parte del núcleo que permite proteger las regiones de memoria en los sistemas embebidos, definiendo permisos específicos para esas zonas, con el objetivo de prevenir accesos no autorizados y garantizar la integridad del sistema. En la Figura 8, se puede observar la ubicación del ajuste dentro del Cortex M3.

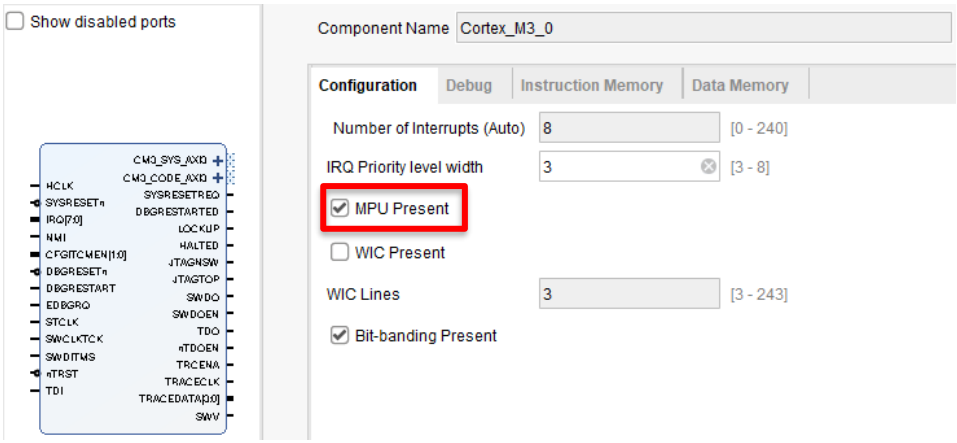


Figura 8: Composición del núcleo Cortex M3

Esta característica es especialmente útil cuando es necesario segmentar y proteger diferentes partes del software o cuando se ejecutan tareas múltiples con un constante flujo de información en entornos de tiempo real. Esta opción se emplea en sistemas operativos de tiempo real (RTOS).

Wake-up Interrupt Controller

Este bloque implementa un controlador de interrupciones que permite "despertar" al microprocesador de un estado de bajo consumo. En la Figura 9, se puede observar la ubicación del ajuste dentro del bloque IP.

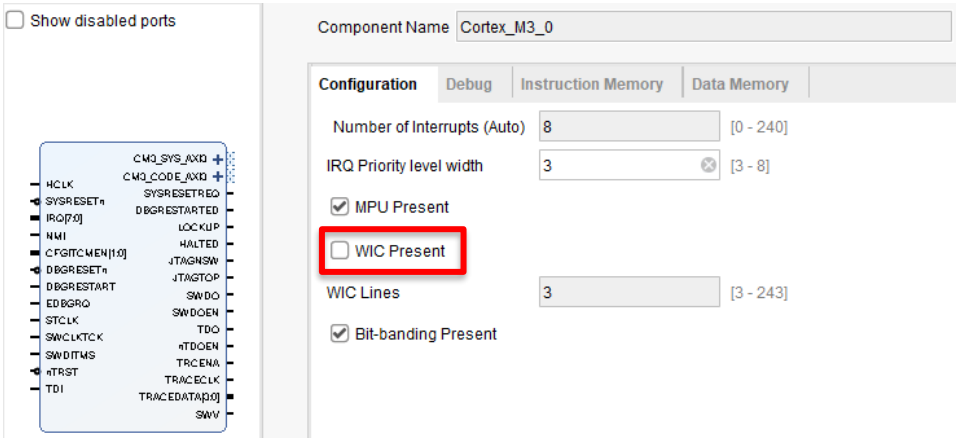


Figura 9: WIC núcleo Cortex M3

El controlador WIC puede gestionar los eventos que se producen en los periféricos del microprocesador, como puede ser la detección de pulsos en los GPIOs o la recepción de datos mediante una de las interfaces de comunicación.
Este controlador permite determinar aquellos eventos que despierten al núcleo y gestiona la correspondiente función de interrupción para su ejecución.
Esta funcionalidad permite disminuir el consumo energético del núcleo durante los periodos de inactividad.

Flash patch and breakpoint (FPB)

La FPB es una parte esencial para la depuración y el análisis de software en sistemas embebidos, ya que permite establecer puntos de interrupción en el código, que se encuentra ejecutando dentro del microprocesador.
El FPB también permite sustituir instrucciones de la memoria flash por otras de forma temporal, dicho procedimiento se denomina "parcheo".
Estas funcionalidades permiten depurar el sistema de forma más precisa y disminuir el tiempo de depuración. En la Figura 10, se puede observar las distintas opciones de configuración

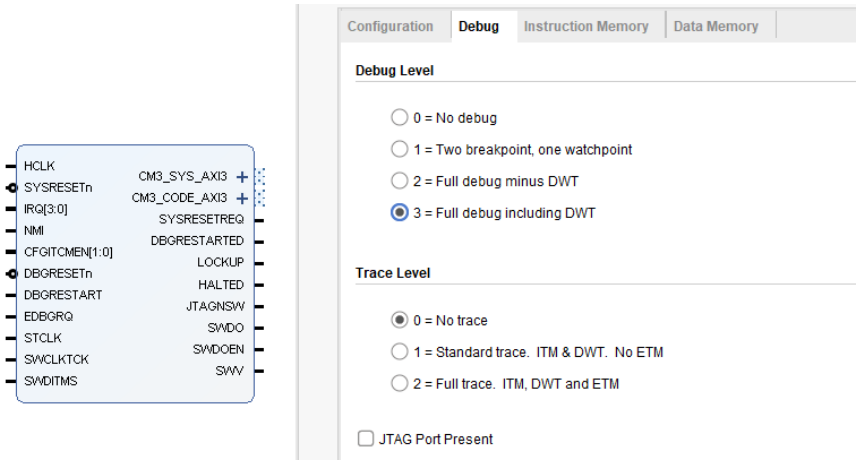


Figura 10: Configuración del nivel del debug y el nivel de trazabilidad

Memoria de instrucción

La memoria de instrucción contiene las instrucciones críticas del programa. Se encuentra ubicada en una región cerca del microprocesador para permitir un acceso rápido y mejorar la velocidad de respuesta. Esta memoria contendrá la aplicación software desarrollada para el microprocesador. En la Figura 11, se puede observar los ajustes de configuración, permitiendo seleccionar el tamaño y su inicialización tras la implementación del micro.

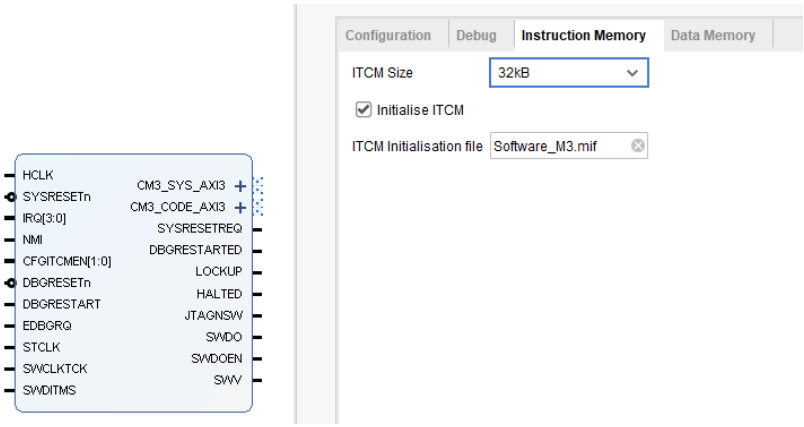


Figura 11: Instrucción de memoria Cortex-M3

4.2. Descripción de los componentes del sistema

Para la especificación de los componentes que conformarán nuestra plataforma hardware será necesario tener en cuenta las restricciones que nos marca nuestro integrado CR95HF [4].
Para este trabajo de fin de grado se ha especificado el desarrollo de una aplicación software que permita detectar tarjetas NFC, por lo que la plataforma hardware se debe adecuar para cumplir con este objetivo.
El integrado CR95HF se controla mediante su interfaz SPI o UART, según se configure. Por lo que será necesario disponer de estos bloques IP en nuestro sistema. La velocidad de la comunicación dependerá de la interfaz seleccionada.
Por otra parte, para inicializar el periférico y seleccionar esta interfaz de comunicación, es necesario generar señales en determinadas entradas digitales del periférico, por lo que será necesario disponer de un bloque IP que controle las entradas/salidas de nuestra placa FPGA.
Todos estos bloques IP, necesitarán entenderse con el microprocesador Corte-M3, por lo que será necesario un módulo de interconexión entre ellos.
Para el desarrollo de la aplicación software será necesario disponer de un método para depurar el sistema, por lo que será necesario disponer de un bloque IP que permita esta funcionalidad.
Por último, será necesario incorporar los componentes básicos de nuestro microprocesador que son una señal de reloj, un módulo de reset.
De esta forma, se utilizan los siguientes módulos IP (intellectual property), desarrollados por Xilinx, para cumplir con las características del sistema.

- **Señal de reset**: Bloque IP Processor System Reset
- **Señal de reloj**: Bloque IP Clocking Wizard
- **Interconexión Núcleo-periféricos**: Bloque IP Axi-Interconnect
- **Control de las salidas digitales**: Bloque IP AXI GPIO
- **Control de la comunicación serial:** Bloque IP AXI Uartlitle
- **Control de la comunicación SPI:** Bloque IP AXI Quad SPI
-

Cada uno de estos bloques IP es configurado en función de las restricciones de nuestro sistema en el apartado 5.1.
En cuanto al módulo del debugger, no se encuentra implementado, en la librería de Xilinx, un bloque IP que cumpla con estas características, por lo que en el apartado 5.1.7 se especifica el desarrollo e implementación de este bloque IP.
En la Figura 12 se recogen todos los componentes que conformarán nuestra plataforma hardware.

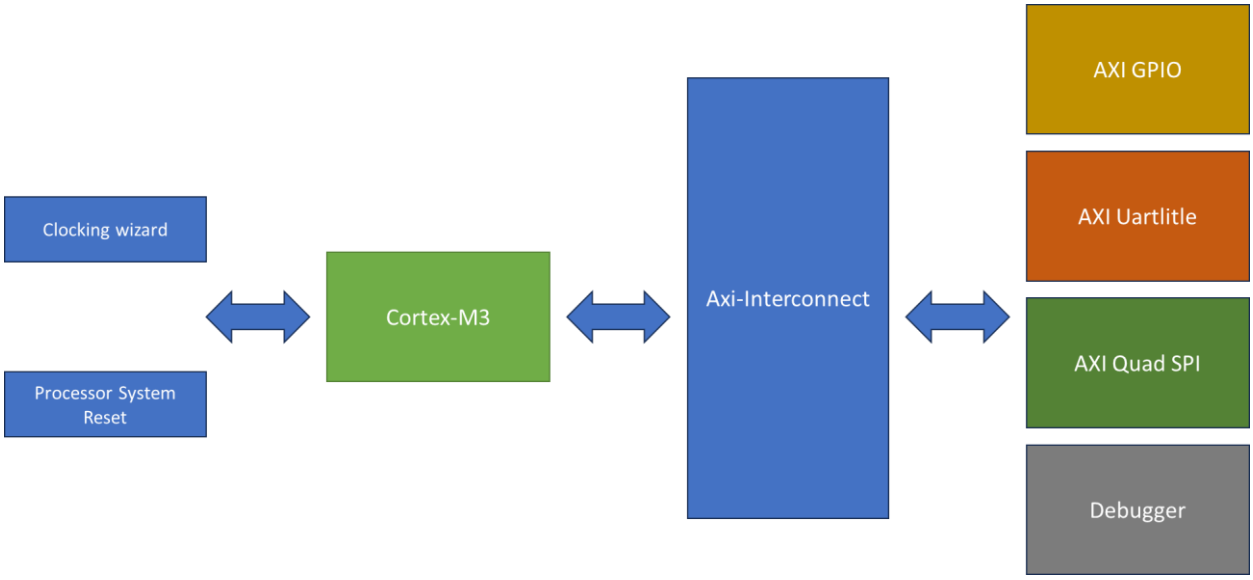


Figura 12: Componentes del sistema

5. Implementación del sistema

Tras especificar los bloques IP que están involucrados en el sistema final, se pasa a configurar aquellos componentes IP que requieren adaptarse para cumplir con los requerimientos de comunicaciones que tiene el integrado CR95HF y que nos permita cumplir con la aplicación software de detección de tarjetas NFC.
La etapa de desarrollo se ha basado principalmente en dos aspectos fundamentales:

- **Desarrollo Hardware**: se selecciona y configura los componentes necesarios que integran el sistema final para su correcto funcionamiento a la hora de desempeñar el objetivo inicialmente planteado.
- **Desarrollo Software**: se desarrolla una aplicación software que, por una parte, permitirá establecer las comunicaciones necesarias con los periféricos externos para la transferencia de información mediante el protocolo NFC y que, por otra parte, permita verificar la correcta implementación del hardware implementado en la etapa anterior.

De esta forma, abordaremos el desarrollo del proyecto de fin de grado desde dos perspectivas distintas, donde podremos ir evaluando esa correlación que deben tener ambas partes para conseguir una correcta ejecución de la aplicación desarrollada.
El material empleado para el desarrollo de nuestro sistema consistirá en:

- FPGA: Artix-7
- Periférico NFC: X-Nucleo-NFC03A1 [3] (Incorpora el integrado CR95HF)
- Debugger: ULINK 2

5.1. Desarrollo Hardware

El hardware desarrollado tiene como objetivo implementar un microprocesador que permita establecer una comunicación mediante el protocolo SPI con periféricos externos, establecer una comunicación serial, controlar E/S digitales y finalmente que permita depurar el sistema en su conjunto.
Teniendo en cuenta estos requerimientos, se ha optado por implementar el Soft-Core Cortex-M3 que se ha detallado en el apartado 4 y los módulos IP descritos en el apartado 4.2. Una de las razones por la que se ha optado por este microprocesador es porque ARM nos proporciona el software DesingStart nos proporciona las herramientas necesarias para su desarrollo en Vivado.
El desarrollo en vivado consta de 4 fases:

- **Diseño**: se establecen los bloques IP que serán necesarios para el correcto funcionamiento del microprocesador.

- **Síntesis del diseño**: Convierte esas descripciones lógicas en descripciones sintetizables que se corresponderán al hardware físico implementable.
- **Implementación del diseño**: se asignan los recursos físicos de la FPGA, se realiza el enrutamiento interno y se asignan las restricciones temporales.
- **BitStream**: se genera un fichero de configuración que contiene toda la información específica del diseño generado para programar la FPGA.

5.1.1. Cortex-M3

Para la primera etapa de diseño se ha tenido en cuenta los diferentes ajustes del Cortex-M3 que se detallan en el apartado 4.
En la aplicación final no será crítico la lectura y escritura en regiones de memoria del micro, se ha decidido deshabilitar la protección de memoria (MPU) y dado que necesitamos que el micro se inicialice nada más conectar, se desactiva el control manual del encendido (WIC).
Con respecto al debugger del micro, se selecciona un nivel de depuración alto y sin trazabilidad. Por último, se especifica el fichero ".mif" que contendrá la aplicación software que ejecutará el micro, desde memoria interna, una vez esté en funcionamiento.
En la Figura 13 se puede observar la configuración final del Cortex-M3.

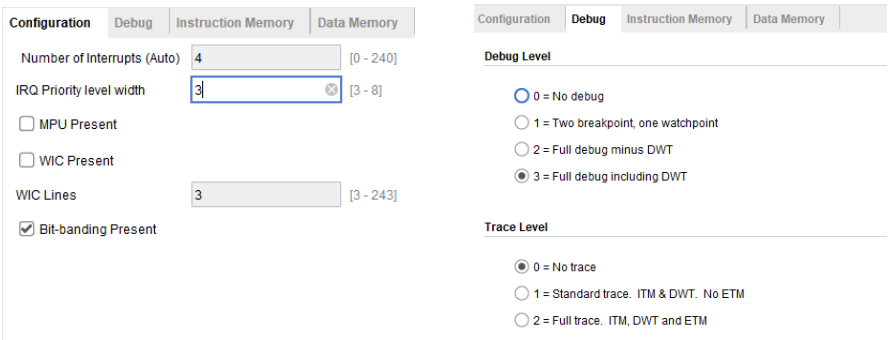


Figura 13: Configuración núcleo Cortex-M3

En el documento técnico del Cortex-M3 [1] se especifica las funcionalidades de sus E/S del bloque IP de este núcleo.
Para que el Corte-M3 opere correctamente, es necesario indicarle la memoria a utilizar en el arranque del sistema (Memoria interna/ Memoria externa). Teniendo en cuenta que para la aplicación final no se contempla una memoria externa, hay que especificarle desde arranque desde memoria. Para ello se fija la posición de la entrada CFGITCMEN [1:0] del bloque IP al valor [1 1].
Por otra parte, se establece el registro de interrupciones enmascaradas a 0, dado que no se hará uso de esa característica del micro. Este conexionado se puede observar en la Figura 14.

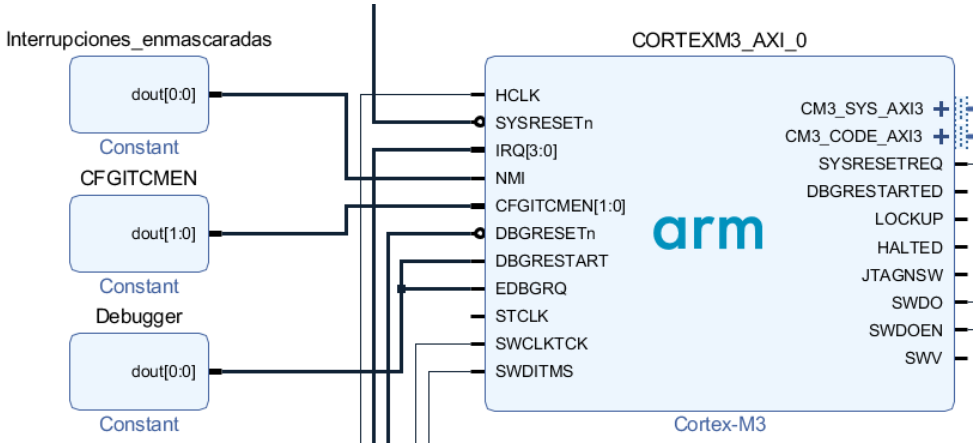


Figura 14: Señales para el arranque del microprocesador

5.1.2. Señal de Reloj

A continuación, se incluye el módulo del reloj [12] que nos permitirá seleccionar la frecuencia a la que trabajará nuestro micro. Para facilitar la sincronización entre diferentes bloques que tengan diferentes frecuencias de reloj, todos los relojes sean múltiplos enteros del reloj principal.
Una restricción del sistema es la velocidad máxima que soporta el SPI del Lector NFC, por lo que es necesario realizar una revisión del DataSheet para conocer y adecuarnos a la velocidad del periférico.
En la documentación del periférico X-NUCLEO NFC03A1 [3] se observa que su desarrollo está basado en el integrado CR95HF [4], por lo que la velocidad del SPI estará limitada por ese núcleo. De esta manera, se hace una revisión de la documentación y se observa que la frecuencia máxima de trabajo para el SPI es 2 MHz, como se observa en la Figura 15.

Symbol	Parameter	Condition	Min.	Max.	Unit
f_{SCK} $1/t_{c(SCK)}$	SPI clock frequency			2.0	MHz
V_{IL}	Input low voltage			0.3 x V_{PS}	V
V_{IH}	Input high voltage		0.7 x V_{PS}		
V_{OL}	Output low voltage			0.4 x V_{PS}	
V_{OH}	Output high voltage		0.7 x V_{PS}		

Figura 15: Frecuencia máxima a la que trabaja el SPI del lector NFC

De esta manera, la frecuencia máxima a la que debe trabajar el controlador del SPI es 2 MHz. Teniendo en cuenta las opciones de configuración de este bloque IP, si especificamos el ratio de frecuencia a 16x1, con un reloj de entrada de 32 MHz conseguiremos una frecuencia de trabajo de 2 MHz para el SPI.
De esta forma, se ha determinado que la frecuencia de trabajo que se debe configurar en el núcleo debe ser 32 MHz, como se puede observar en la Figura 16.

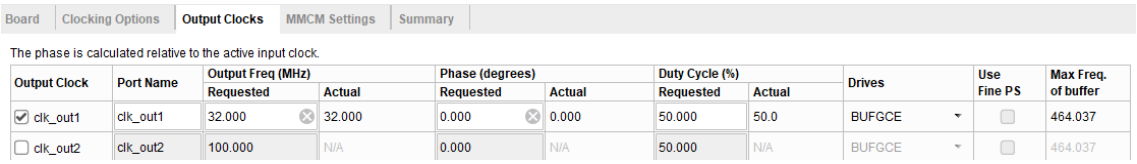


Figura 16: Frecuencia de reloj del módulo CLK

La frecuencia máxima que soporta el Corte-M3 es de 150 MHz en una FPGA Artix-7, por lo que nos encontramos dentro del margen de trabajo del micro.

5.1.3. SPI

Con respecto al módulo SPI [13], se configura como "*single SPI*" (1 canal de comunicación) y se selecciona la velocidad 1x16 que indica que el reloj de entrada se subdividirá por 16, de esta forma, el módulo SPI tendrá una frecuencia de trabajo de 2 MHz para adecuarse al integrado CR95HF. En la documentación [4] se observa que los comandos para el envío y recepción de información mediante el SPI son mayores de 16 bytes, por lo que el tamaño de la FIFO se selecciona de 256 bytes. En la Figura 17, se puede observar la configuración final del controlador SPI.

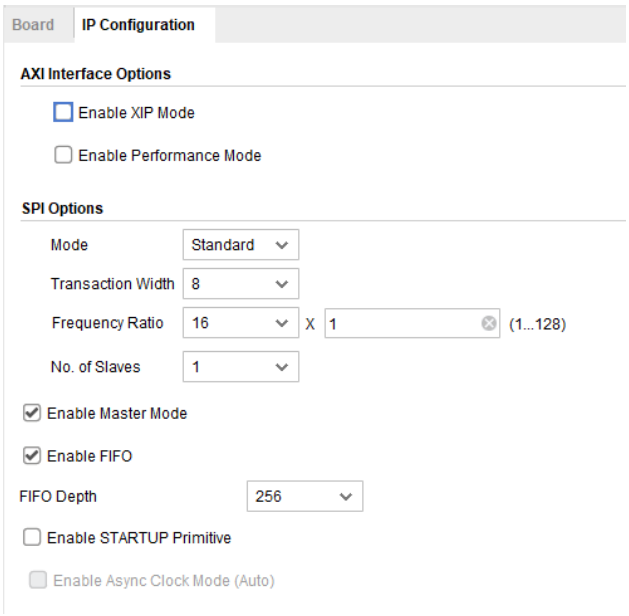


Figura 17: Configuración Bloque Axi Quad SPI

5.1.4. GPIO

Lo siguiente en añadir son los módulos GPIO [14] para controlar los pines de entrada y salida del sistema.
Inicialmente se analiza la documentación del lector NFC [3] y se extrae que los Pines que nos permitirán realizar la aplicación software son:

- *IRQ_In*: interrupción de entrada para despertar el núcleo CR95HF del lector NFC
- *Interface PIN*: Selección de la interfaz de comunicación

- *IRQ_Out:* Interrupción de salida para indicar el estado del núcleo CR95HF.

De esta manera, se establecen dos instancias por cada Bloque GPIO para controlar los siguientes pines:

- 3 pines para controlar las señales *"IRQ_In"*, *"Interface pin" y "IRQ_Out"* del lector NFC (Control_SPI_lectorNFC).
- 4 pines para controlar los Leds que incorpora el lector NFC (leds_lectorNFC).
- 4 pines para el control de los Leds de propia placa de la FPGA (leds_4bits).
- 12 pines para el control de los Leds RGB de la propia placa FPGA (rgb_led).

En la Figura 18, se puede observar los dos bloques finales incorporados en el proyecto.

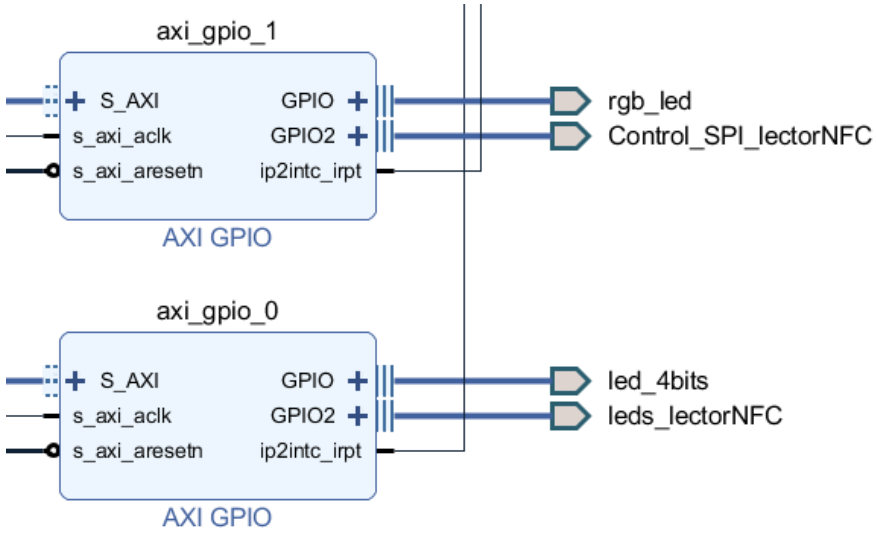


Figura 18: Módulos GPIO

5.1.5. Señal de Reset

En todo sistema, es necesario establecer un mecanismo para resetear los componentes que lo conforman, en nuestro proyecto será mediante la incorporación del bloque IP "*Processor System Reset*" [15].

El comportamiento esperado del reset es el siguiente:

- Reset mediante botón: Resetear todo el sistema del microprocesador mediante el botón físico de la placa FPGA.
- Reset mediante Debugger: Resetear el sistema del microprocesador y periféricos mediante señal Reset del Debugger.

De esta forma, se establecen los dos bloques IP de la Figura 19.

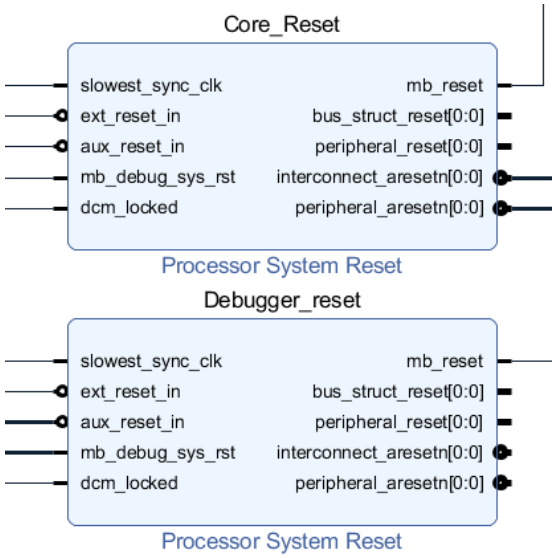


Figura 19: Módulos de Reset del Core y Debugger

Con el objetivo de permitir estas dos funciones descritas, se realiza el conexionado de la siguiente forma:

- **Debugger_Reset**: Se conecta la entrada "*ext_reset_in*" al botón físico de la FPGA, por lo que cuando se pulse, se activará la señal *"mb_reset"* que se conectará al pin "System_Reste_In" del Cortex M3.
- **Core_Reset**: Se conecta la entrada "*ext_reset_in*" al botón físico de la FPGA y se conecta la entrada "*aux_reset_in*" al pin SWRSTn del Debugger, de esta forma, conseguiremos que se active la señal "*mb_reset*" cuando se presiona el botón y cuando se envía la orden de Reset desde el bloque IP del Debugger. La señal "*mb_reset*" se conectará al pin "DEBUGRESETn" del núcleo.

Cuando se realiza un reset del núcleo hay que tener en cuenta que es necesario seguir una secuencia determinada para realizarlo de forma correcta. Para ello, primero se reiniciarán los periféricos conectados al bloque del reset (*interconnect_aretn* y *peripheral_aresetn*) y posteriormente se reiniciará el micro (*mb_reset*).

Cabe resaltar que este bloque IP también implementa la sincronización de la señal de reset con la señal de reloj.

5.1.6. Comunicación Serial

Para poder comunicarnos con el microprocesador mediante la interfaz serie se añade el bloque IP *Axi Uarlite* [16]. Dado que para la aplicación final no existen requerimientos especiales, se deja por defecto la velocidad de la comunicación a 9600 baudios/s.

5.1.7. Debugger

Para debuggear el sistema utilizaremos el protocolo SWD [17] donde únicamente se utiliza 1 pin para enviar o recibir datos, por lo que será necesario incorporar un bloque IP que implemente un buffer triestado, cuyo funcionamiento se observa en la Figura 20.

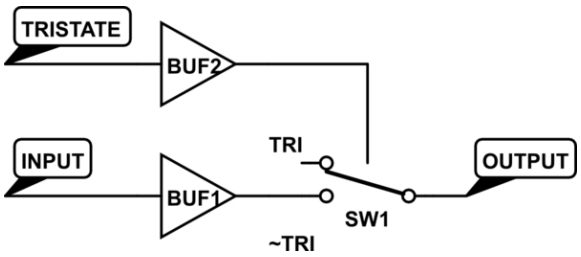


Figura 20: Concepto Buffer Triestado

Este bloque IP no se encuentra en la librería del núcleo, por lo que diseña en VHDL y se incorpora al proyecto.

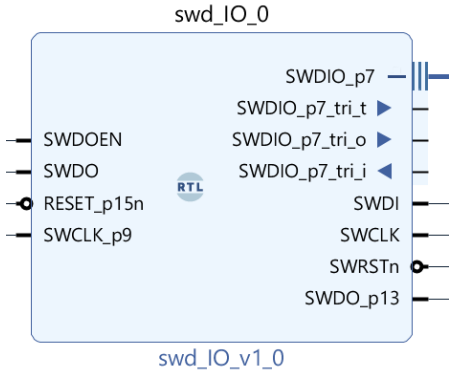


Figura 21: Bloque IP Debugger

En la Figura 21 se puede observar en conjunto de E/S que tiene este bloque IP:

- **SWDOEN:** Señal de habilitación para la salida de datos SWD.
- **SWDI:** Señal de entrada para los datos SWD.
- **SWDO:** Señal de salida para los datos SWD.
- **SWCLK:** Señal de reloj para la interfaz SWD.
- **SWRSTn:** Señal de reinicio activa en bajo para la interfaz SWD.
- **RESET_p15n:** Señal de reinicio activa en bajo.
- **SWDIO_p7_tri_t:** Señal de datos SWD para configurar el buffer triestado.
- **SWDIO_p7_tri_o:** Salida del buffer triestado para la señal de datos SWD.
- **SWDIO_p7_tri_i:** Entrada del buffer triestado para la señal de datos SWD.
- **SWDO_p13:** Señal de datos SWD.
- **SWCLK_p9:** Señal de reloj SWD.

El funcionamiento esperado es el siguiente:

- La señal SWDOEN se activa en bajo cuando se habilita la salida de datos por SWDO, en caso contrario, no se permite la salida de datos por SWDO. El sentido de la comunicación es desde el micro al debugger.

- Por otra parte, la entra SWDI se utiliza para enviar datos desde el debugger hacia el micro.

De esta manera, se diseña en VHDL este comportamiento:

- **SWDIO_p7_tri_t <= not SWDOEN**: para que se pueda sacar datos por SWDIO_p7_tri_o o permitir la entrada de datos por SWDIO_p7_tri_i
- **SWDI <= SWDIO_p7_tri_i:** Se asocia el pin del micro de entrada con el pin del debugger.
- **SWDIO_p7_tri_o <= SWDO:** Se asocia el pin del micro de salida con el pin del debugger.

De esta forma, desde un único pin, controlado por la señal de entrada SWDOEN, podremos enviar o recibir información del debugger.

5.1.8. Interconexión de bloques

Por último, se añade el bloque Axi interconect [18], que permite la comunicación entre los módulos que componen el sistema (GPIO, UART, SPI y Debugg) y el núcleo.
Se configuran las interfaces Máster de forma que los registros se habiliten de forma automática. En la Figura 22 se puede observar la configuración final para nuestro sistema.

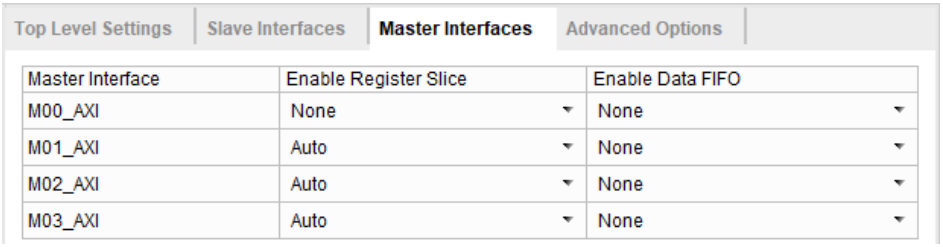


Figura 22: Configuración del módulo AXI interconect tras añadir los bloques IP

En el anexo 9.3 se puede obtener más características de los bloques IP utilizados en el proyecto.
De esta manera, el sistema final se puede observar en la Figura 23.

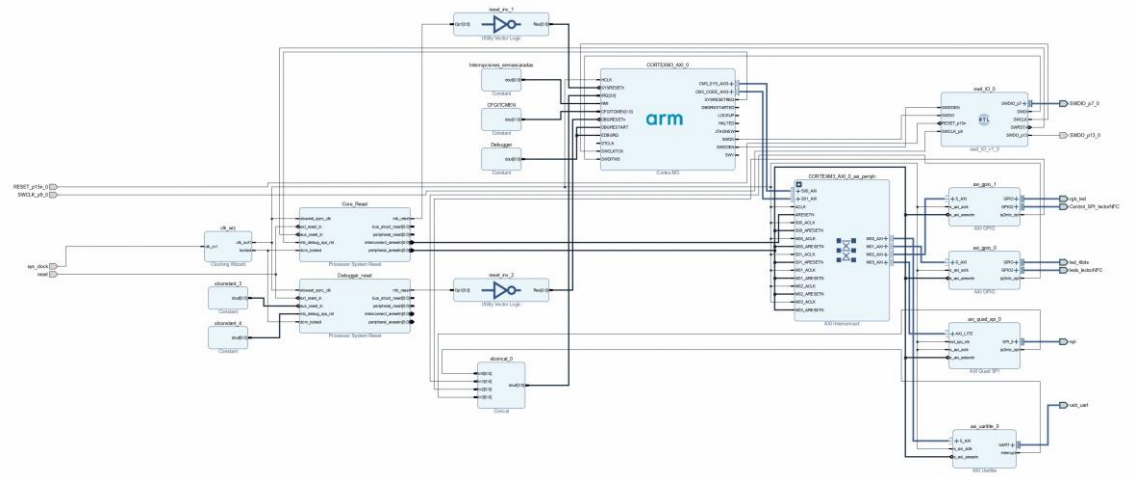


Figura 23: Sistema Hardware final

Tras especificar el diseño, se pasa a la etapa de síntesis donde el software de vivado transforma los bloques de la etapa de diseño en módulos sintetizables que son los necesarios para la etapa de implementación.

5.1.9. Fichero de restricciones

Por último, se desarrolla el fichero de restricciones donde se especifica la ubicación de los pines de entrada/salida del sistema con los pines físicos de la placa FPGA y las restricciones temporales del sistema. Durante esta etapa, es fundamental disponer de una visión general del conexionado, dado que en este fichero se determina cómo se interconectarán los pines para realizar el montaje final.

La FPGA con la que se está desarrollando el proyecto es una Artix-7 [19] preparada para trabajar con periféricos de Arduino. En la Figura 24, se puede observar la distribución de los pines físicos en la placa.

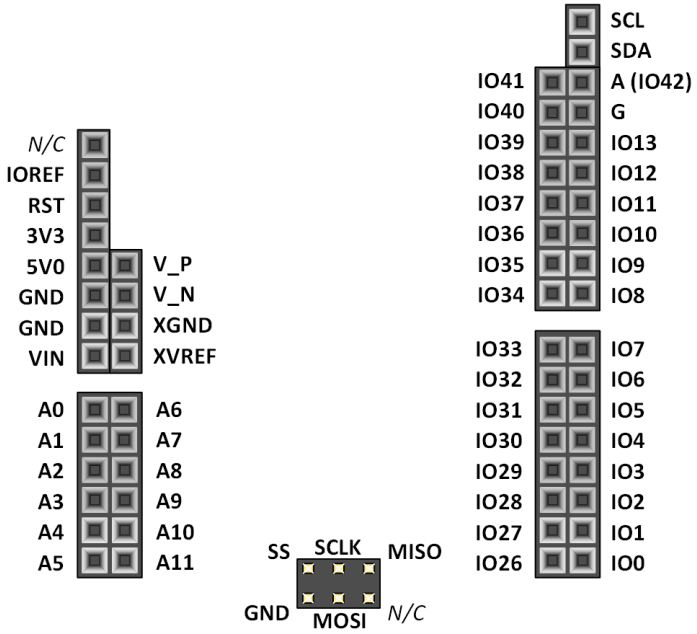


Figura 24: Conector Arduino/chipKIT de la plaza Arty A7

De la misma forma, con la documentación del lector NFC [3] podemos determinar la posición de cada uno de los pines para su conexión con la FPGA.

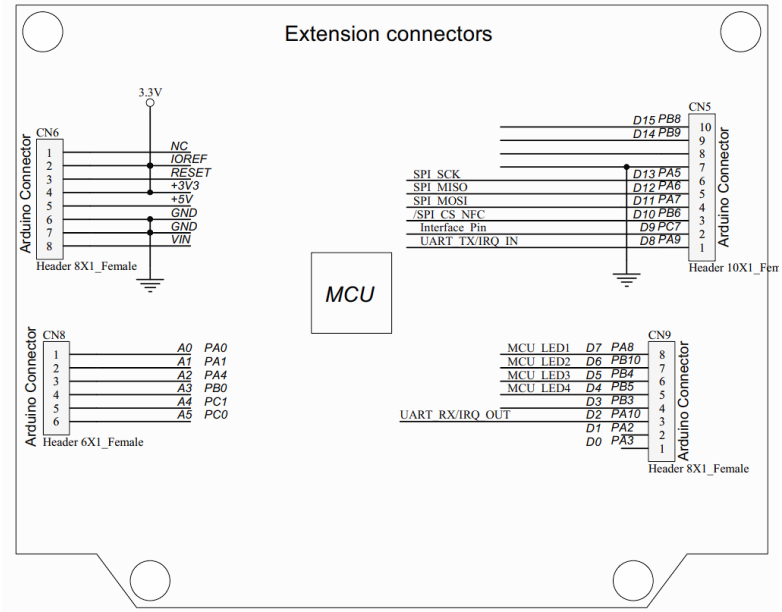


Figura 25: Conectores X -NUCLEO NFC03A1

Podemos observar en la Figura 25 que la compatibilidad entre los conectores de la Artix-7 y el lector NFC es adecuada para su montaje.

De esta forma, conocida la posición de los conectores del lector NFC y su correspondencia con los pines de la placa FPGA, debemos ubicar cada una de las salidas de nuestro sistema en cada uno de los pines que se corresponda al lector NFC, por lo que en el fichero de restricciones se realizará las siguientes asociaciones de la Figura 26.

```
# PMOD JD DEBUGGER
set_property PACKAGE_PIN D4 [get_ports {RESET_p15n_0}]; # JD2
set_property PACKAGE_PIN F4 [get_ports {SWCLK_p9_0}]; # JD1
set_property PACKAGE_PIN E2 [get_ports {SWDIO_p7_0_tri_io}]; # JD7
set_property PACKAGE_PIN F3 [get_ports {SWDO_p13_0}]; #JD4

# SPI PINES

set_property PACKAGE_PIN U18 [get_ports spi_io0_io]; # MOSI
set_property PACKAGE_PIN R17 [get_ports spi_io1_io]; # MISO
set_property PACKAGE_PIN P17 [get_ports spi_sck_io]; # SCK
set_property PACKAGE_PIN V17 [get_ports spi_ss_io]; # CS

# Control Lector NFC

set_property PACKAGE_PIN N15 [get_ports {Control_SPI_lectorNFC_tri_o[0]}]; # IRQ_IN
set_property PACKAGE_PIN M16 [get_ports {Control_SPI_lectorNFC_tri_o[1]}]; # INTERFACE_SELECT
set_property PACKAGE_PIN P14 [get_ports {Control_SPI_lectorNFC_tri_o[2]}]; # IRQ_OUT
set_property PACKAGE_PIN T11 [get_ports {Control_SPI_lectorNFC_tri_o[3]}]; # LIBRE D3

# LED's Lector NFC

set_property PACKAGE_PIN R12 [get_ports {leds_lectorNFC_tri_o[0]}]; #
set_property PACKAGE_PIN T14 [get_ports {leds_lectorNFC_tri_o[1]}]; #
set_property PACKAGE_PIN T15 [get_ports {leds_lectorNFC_tri_o[2]}]; #
set_property PACKAGE_PIN T16 [get_ports {leds_lectorNFC_tri_o[3]}]; #
```

Figura 26: Fichero de restricciones en Vivado

Para el caso del bloque IP del debugger, su conexionado se debe redirigir hacía uno de los bloques PMOD para facilitar el montaje del sistema posteriormente. El bloque IP del debugger tiene una señal de reloj como entrada, por lo que será necesario ubicar esa entra digital en un PIN dedicado a señales de reloj en la placa FPGA. Estos pines se denominan SRCC y junto con la documentación [20] se selecciona el pin adecuado para esta señal. La distribución de los pines en la FPGA se puede observar en Figura 27.

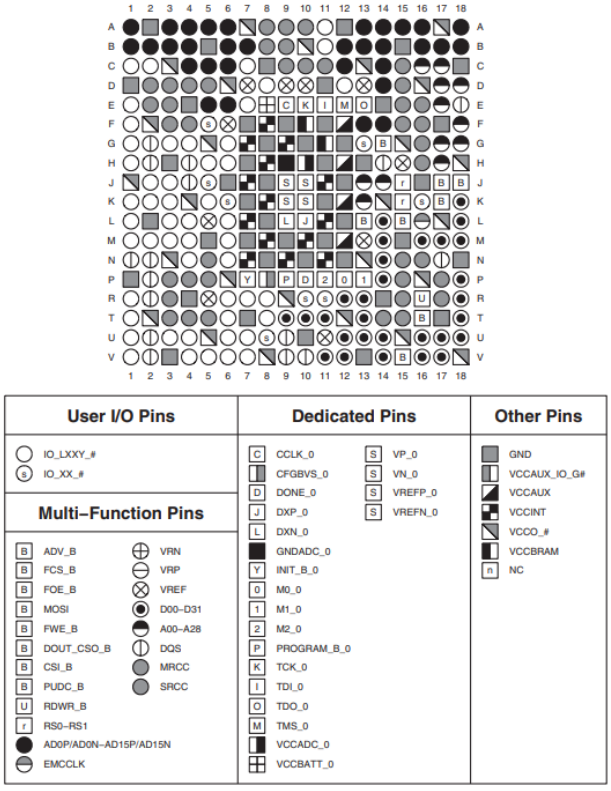


Figura 27: Pines FPGA

Por último, se genera la síntesis y la implementación del proyecto para exportar el hardware, como se puede observar en la Figura 28.

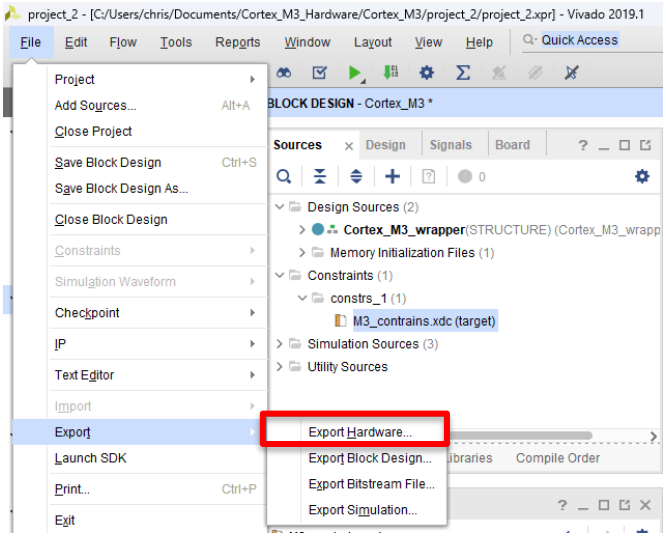


Figura 28: Exportar Hardware en vivado

5.2. Desarrollo Software

Para el desarrollo software se emplean dos herramientas: Vivado SDK y Keil uVision 5.

Desde el software de vivado podemos lanzar la herramienta Vivado SDK que permite generar los ficheros BSP (Board support package) [21], que contienen la configuración y los controladores necesarios del hardware, específicamente diseñado en el apartado 5.1.

Dentro de este proyecto se puede observar el fichero *"system.hdf"* donde se recogen bloques IP incluidos al proyecto, así como las direcciones de memoria donde se ubican.

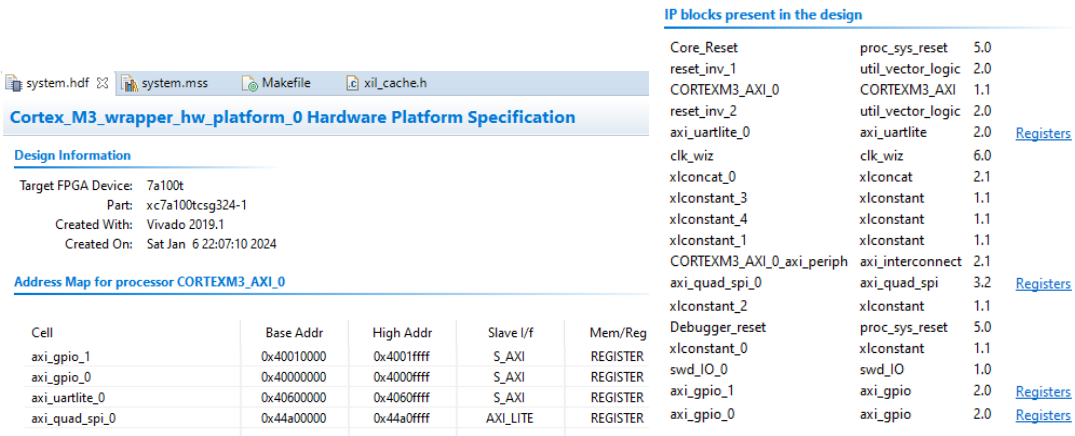


Figura 29: Fichero "system.hdf"

Por otra parte, podemos observar que en el fichero *"system.mss"* se recogen los drivers incluidos al proyecto.

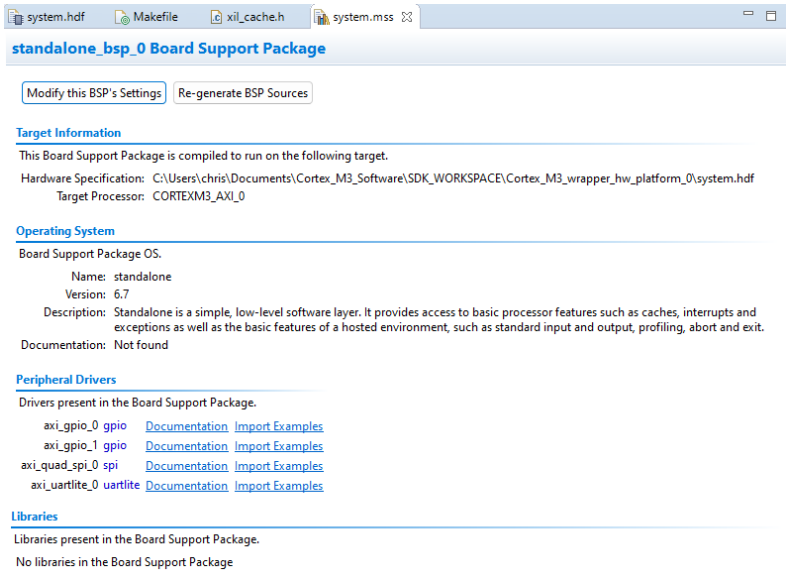


Figura 30: Fichero "system.mss"

Tras generar el BSP del hardware, se continúa con el desarrollo de la parte software mediante el programa keil uVision5.
Al generar el nuevo proyecto, se selecciona como CMSIS (Cortex Microcontroller Software Interface Standard) un ARMCM3, dado que se está implementando un ARM Cortex M3.

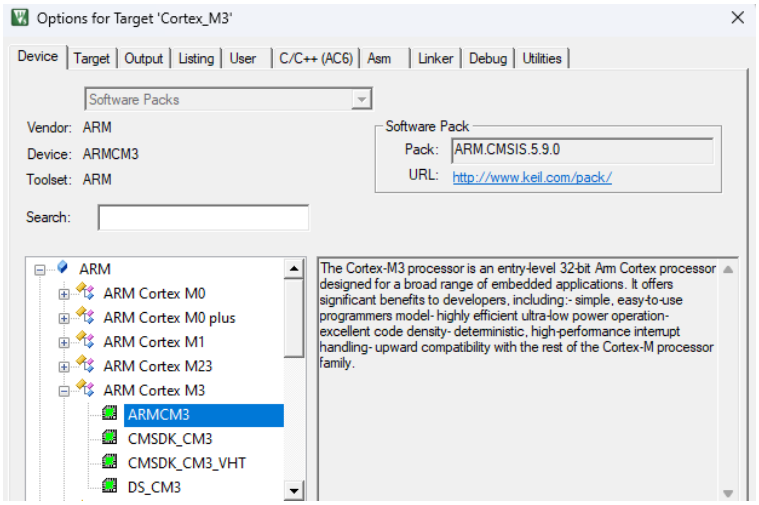


Figura 31: Configuración proyecto Keil

Se importan los ficheros BSP del Vivado SDK y las librerías de los drivers de Xilinx asociado a los bloques IP GPIO, SPI y UART.
Como compilador del proyecto se utiliza la última versión disponible de keil y se seleccionan los ficheros de salida tras la compilación. El fichero que contiene el programa fuente y el que se cargará al Cortex-M3 es el de extensión *".hex"*.

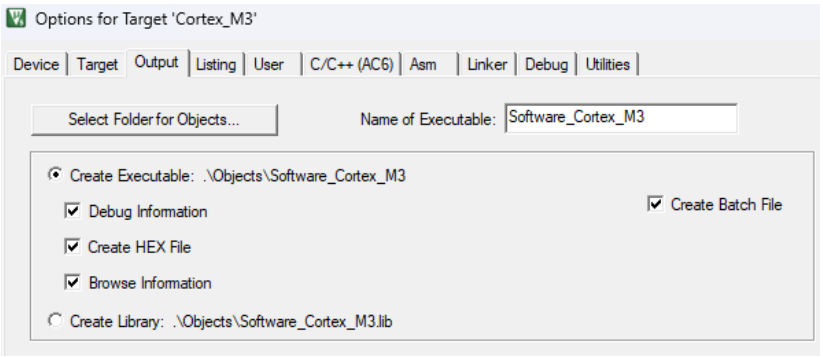


Figura 32: Ficheros de salida keil

Tras configurar todo el proyecto dentro de Keil, pasamos a desarrollar la aplicación principal que ejecutará el núcleo del Cortex M3. Esta aplicación tendrá como objetivo inicializar los periféricos GPIO y los bloques de comunicación SPI y UART, para poder controlar las entradas digitales del lector NFC, comunicarnos por la interfaz SPI y obtener una respuesta del microprocesador por interfaz serie tras la detección de una tarjeta NFC.

Como hemos podido analizar en el apartado 5.1 para poder trabajar con el integrado CR95HF, es necesario configurar correctamente la interfaz SPI y activar las señales digitales correspondientes para poner en funcionamiento el periférico.

Para inicializar el integrado CR95HF, es necesario cumplir la secuencia de arranque que se proporciona en la documentación del núcleo [4].

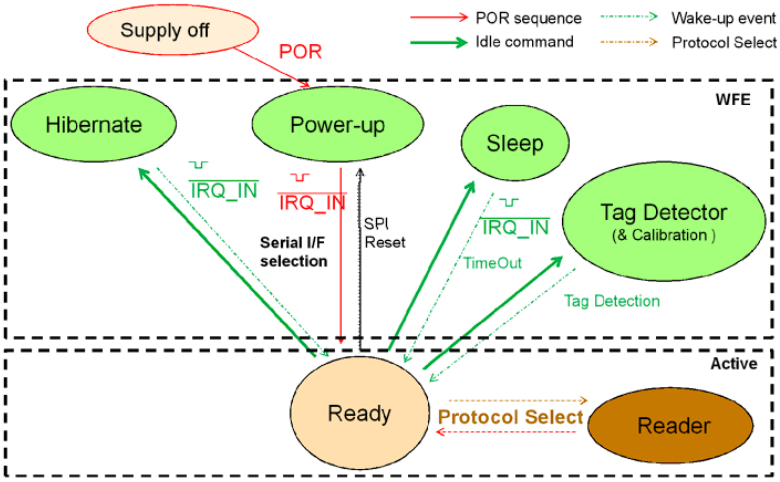


Figura 33: Secuencia de arranque del lector NFC [4]

En la Figura 33 y Figura 34, se observa que es necesario generar la señal IRQ_IN a '0' durante un tiempo mayor de t1 (10 us).

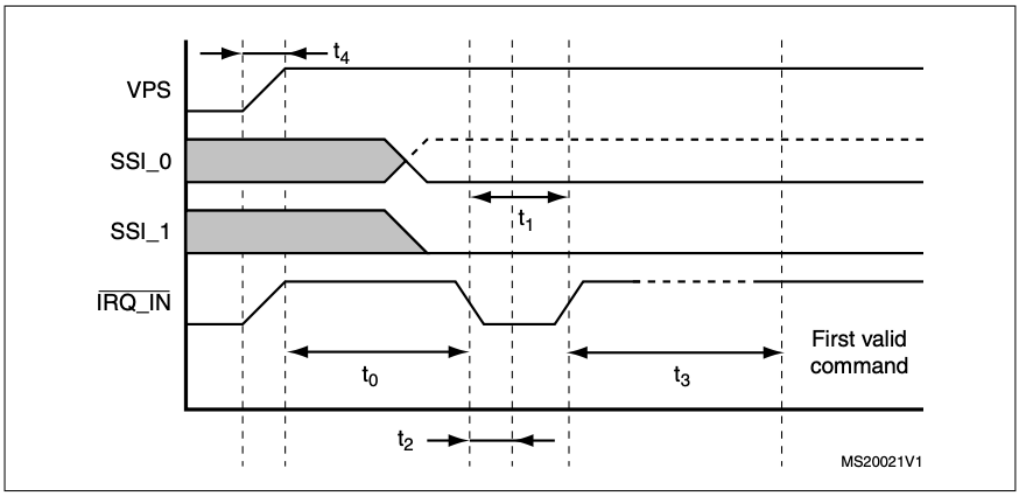


Figura 34: Start up lector NFC [4].

En nuestro sistema, el pin "*IRQ_IN*" se encuentra conectado a una de las salidas digitales de la FPGA asociada a un bloque GPIO, por lo que, para generar ese pulso, únicamente será necesario activar esa salida digital a '1' durante un tiempo mínimo de 10 us. La señal "*SSI_0*" de la Figura 34, selecciona la interfaz por la que se comunicará nuestro periférico ('1' SPI '0' UART). En nuestro sistema tenemos conectado ese pin a una salida digital de la FPGA asociada a un bloque GPIO, por lo que para seleccionar la interfaz SPI será necesario colocar un '1' en esa salida digital.

Una vez inicializado el periférico pasamos a comunicarnos con él mediante la interfaz SPI.

En la documentación del periférico [4] se especifica que la interfaz SPI trabaja únicamente como esclavo y que tras el envío de cada comando es necesario realizar un retorno de la señal SPI_SS. De esta manera es necesario configurar al driver del SPI de nuestro sistema acorde a estas características. Con la documentación proporcionada por Xilinx [13] se localiza el registro de control que nos permitirá configurar el modo de operación del SPI.

De esta forma, se escribe sobre este registro y se coloca un uno en la posición *"Master"* y un cero en la posición de *"Master Transaction Inhibit"*. De esta manera, el SPI del Cortex-M3 actuará como Máster y colocará la señal *"SPI_SS"* activa en alto tras finalizar la transferencia de datos.

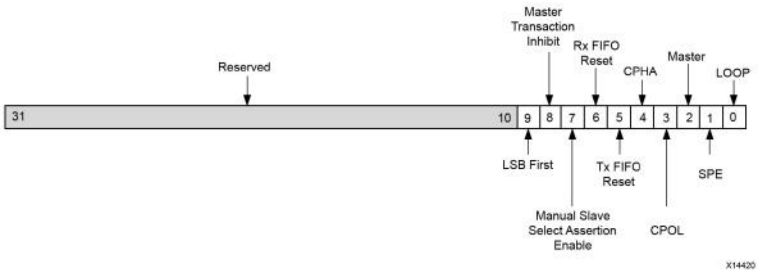


Figura 35: Registro de control SPI Cortex M3 [13]

A continuación, tras inicializar y configurar la interfaz SPI se procede a generar el envío de comandos con el formato que se indica en la documentación del integrado CR95HF [4] que incorpora el lector NFC.
El funcionamiento general de la comunicación se basa en el esquema de la Figura 36.

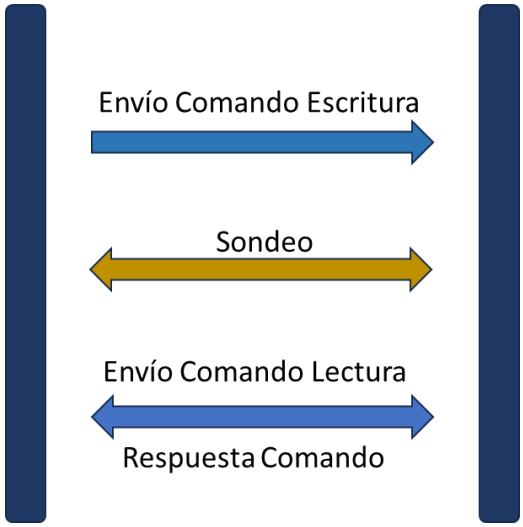


Figura 36: Proceso de comando/respuesta del lector NFC

De esta manera, para establecer esta comunicación hacia el lector NFC, es necesario enviar los datos con la siguiente estructura:

Byte de control	Comando	Longitud	Datos
Tipo y dirección de la comunicación	Orden para ejecutar	Longitud de los Datos enviados	Datos transmitidos

El lector NFC contempla 4 valores del bit de control:

- **0x00:** Envío de datos al lector NFC.
- **0x03:** Sondeo al lector NFC.
- **0x02:** Leer datos del lector NFC.
- **0x01:** Reiniciar el lector NFC.

Los valores, que utilizaremos en esta esta aplicación, para el campo comando serán:

- **0x01:** Información general del lector NFC
- **0x07:** Conmuta el CR95HF a un modo de bajo consumo denominado "Espera de Evento" (WFE) (Encendido, Hibernación, Reposo o Detección de Etiqueta)

Especificada la forma en la que vamos a enviar los datos para que nos entienda el lector NFC, generamos varias funciones que se seguirán el esquema de la Figura 36.

Envío comando escritura

Para enviar un comando de escritura se seguirá la estructura enunciada anteriormente:

Byte de control	Comando	Longitud	Datos
0x00	0x01 0x07	Longitud de los Datos enviados	Datos transmitidos

Se definen dos buffers que nos permitirán almacenar los bytes que necesitamos transmitir y los bytes que recibiremos del lector NFC.
Con la función *XSpi_Transfer()* podemos indicarle al SPI del microprocesador que envíe "n" bytes que se encuentran en el buffer *QSPI_base_tx_buf* y que la respuesta en el buffer *QSPI_base_tx_buf*.
Hay que tener en cuenta que, en el momento del envío del comando, la información que recibimos del lector NFC por la entrada MISO, no se corresponderá a la repuesta de nuestro comando ya que será información que se encontraba en el buffer del periférico. En la Figura 37 y Figura 38 se puede observar el resultado final de las funciones que contempla el envío del comando "**0x01**" y "**0x07**".

```
233 int WriteSPICommandIdle(XGpio gpio_1)
234 {
235     int status;
236     // Comando de escritura hacia el CR95HF
237     // 0x00: Byte de control de escritura
238     // 0x07: Comando de configuración de encendido por detección de TAG
239     // 0x0E: Longitud del comando
240     // 0x0A21007901180020606074843F08: Parámetros de configuración
241     QSPI_base_tx_buf[0] = 0x00;
242     QSPI_base_tx_buf[1] = 0x07;
243     QSPI_base_tx_buf[2] = 0x0E;
244     QSPI_base_tx_buf[3] = 0x0A;
245     QSPI_base_tx_buf[4] = 0x21;
246     QSPI_base_tx_buf[5] = 0x00;
247     QSPI_base_tx_buf[6] = 0x79;
248     QSPI_base_tx_buf[7] = 0x01;
249     QSPI_base_tx_buf[8] = 0x18;
250     QSPI_base_tx_buf[9] = 0x00;
251     QSPI_base_tx_buf[10] = 0x20;
252     QSPI_base_tx_buf[11] = 0x60;
253     QSPI_base_tx_buf[12] = 0x60;
254     QSPI_base_tx_buf[13] = 0x74;
255     QSPI_base_tx_buf[14] = 0x84;
256     QSPI_base_tx_buf[15] = 0x3F;
257     QSPI_base_tx_buf[16] = 0x08;
258
259     //print("Se envia el Control Byte + comando + longitud_Datos + Datos");
260     status = XSpi_Transfer(&SGSPI, QSPI_base_tx_buf, QSPI_base_rx_buf, 17);
261
262
263     return status;
264 }
```

Figura 37: Comando de escritura comando 0x07

```
int WriteSPIManufacturer(XGpio gpio_1)
{
    int status;
   // Comando de escritura hacia el CR95HF
    QSPI_base_tx_buf[0] = 0x00;
    QSPI_base_tx_buf[1] = 0x01;
    //Inicializamos registros de transmisión y recepción
    for(int i=2;i<32;i++){
      QSPI_base_tx_buf[i] = 0x00;
    }
    //Transmisión de datos
    status = XSpi_Transfer(&SGSPI, QSPI_base_tx_buf, NULL, 3);
    if (status==XST_SUCCESS){
      XGpio_DiscreteWrite(&gpio_1, 1, 0x4); // LED RGB
    }
    return status;
}
```

Figura 38: Comando de escritura comando 0x01

Sondeo

El lector NFC necesitará un tiempo de procesamiento para atender a nuestro comando, por lo que es necesario realizar un sondeo al lector NFC para conocer su estado.
Con se observa en la Figura 39, se enviará el byte de control **"0x03"** hasta recibir un flag de respuesta, indicando que el lector NFC ya está disponible para ser leído, en este caso se activará el bit 3 (posición 4) cuando se encuentre listo.

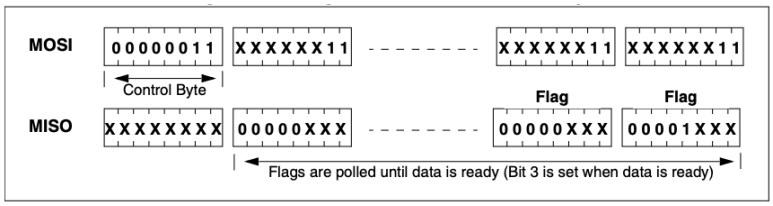


Figura 39: Formato del comando de Sondeo [4]

Mientras se espera a que lector NFC esté listo, se puede ir ejecutando otra parte del código si es necesario.
En la Figura 40 se puede observar el resultado final del código.

```
int PollingSPI(XGpio gpio_1)
{
    //Estado de la función de transferencia de datos del SPI
    int status;
    //Variable que indica si se ha detectado el FLAG para poder leer datos del lector NFC
    int resultado=0;
    //Comando de sondeo al lector NFC
    QSPI_base_tx_buf[0] = 0x03;
    //Inicialización buffer de transmisión y recepción
    for(int i=1;i<32;i++){
      QSPI_base_tx_buf[i] = 0x03;
      QSPI_base_rx_buf[i] = 0x00;
    }
    //Esperamos a que el Lector NFC nos devuelva el Flag para leer
    while(resultado==0){
      //Cada petición de sondeo está retrasado 3000 ciclos de reloj
      delay_cycles1(3000);
      //Enviamos el comando de sondeo
      status = XSpi_Transfer(&SGSPI, &QSPI_base_tx_buf[0], &QSPI_base_rx_buf[0],2);
      //Detectamos si hemos recibido un 1 en la posición 4
      //que indica que se puede enviar una petición de lectura al Lector NFC
      resultado=QSPI_base_rx_buf[1]&0x08  ;
      if(resultado!=0)
      {
        //Encendemos El LED RGB
        XGpio_DiscreteWrite(&gpio_1, 1, 0x01);
      }
    }
    return status;
}
```

Figura 40: Comando de Sondeo hacia el lector NFC

Envío de comando lectura/ respuesta

Por último, se envía el comando de lectura mediante el valor del byte de control **"0x02".** De esta manera, el formato de los datos a enviar será de la siguiente manera:

Byte de control	CMD	Longitud
0x02	**0x00**	**00**

Para la respuesta del comando "**0x01"** (Información general del lector NFC) la respuesta contendrá 18 bytes que se almacenarla en el buffer de recepción *QSPI_base_rx_buf*.

Direction	Data	Comments
CR95HF to Host	0x00	Result code
	0x01	Length of data
	<Data>	Data (Wake-up source): 0x01: Timeout 0x02: Tag detect 0x08: Low pulse on IRQ_IN pin 0x10: Low pulse on SPI_SS pin
CR95HF to Host	0x82	Error code
	0x00	Length of data

Figura 41: Códigos de detección del lector NFC [4]

En el caso del comando "**0x07"** (Espera de Evento), las posibles respuestas son las especificadas en Figura 41, con una longitud de 1 byte. Si la respuesta obtenida es **"0x000102"** (Detección de tarjeta) encendemos un LED RGB de la placa FPGA. En la Figura 42 se puede observar el código que modela este comportamiento.

```
int ReadSPICommandIdle(XGpio gpio_1)
{

    //Estado de la función
    int status;
    //Variable para saber si ha detectado una tarjeta
    int detectar=0;
    //Comando 0x02 para leer datos provenientes del Lector NFC
    QSPI_base_tx_buf[0] = 0x02;
    //Inicializamos buffer
    for(int i=1;i<32;i++){
      QSPI_base_tx_buf[i] = 0x00;
      QSPI_base_rx_buf[i] = 0x00;

    }
    //Enviamos comando y recibimos respuesta
    status = XSpi_Transfer(&SGSPI, QSPI_base_tx_buf, QSPI_base_rx_buf,4);
    //Comprobamos si que el FLAG de respuesta corresponde al 0x000102 Detección de TAG
    if(QSPI_base_rx_buf[1]==0x00 && QSPI_base_rx_buf[2]==0x01 && QSPI_base_rx_buf[3]==0x02){
      XGpio_DiscreteWrite(&gpio_1, 1, 0x02);  // Encendemos LED RGB
      delay_cycles1(3200000);                 // Encendemos el LED RGB indicando que se ha detectado una tarjeta
      XGpio_DiscreteWrite(&gpio_1, 1, 0x00);  // Apagamos LED RGB
      detectar=1;                             // Devolvemos el valor 1 indicando que se ha detectado un tarjeta
    }
    return detectar;
}
```

Figura 42: Lectura de la respuesta del comando 0x07

Para finalizar nuestra aplicación software, se genera un bucle principal donde se realizan las siguientes funciones:

- Mensajes por interfaz serie UART
- Esperar a recibir interacción con el usuario
- Iniciar proceso de detección
    - Enviamos el comando **0x07**
    - Sondeamos el lector NFC
    - Leemos respuesta

De esta forma, el bucle principal de nuestro software queda como en la Figura 43.

```
    while ( 1 )
    {
      if(inicio==1){
          //Enviamos el comando por el módulo SPI
          status = WriteSPICommandIdle(gpio_1);
          //Esperamos a que lector esté listo
          status = PollingSPI(gpio_1);
          //Leemos la respuesta
          status = ReadSPICommandIdle(gpio_1);
          //En caso que la respuesta corresponda a una Tarjeta NFC
          if(status==1){
            //Volvemos al estado inicial
            inicio=0;
            //Indicamos que hemos detectado una tarjeta
            print("Tarjeta detectada\r\n");
          }
        }
      else{
          //Mensaje por pantalla para empezar a detectar tarjetas
          print("INICIAR DETECCION DE TARJETAS Y/N:\r\n");
          do {
          //Esperamos a recibir la respuesta por teclado
            caracter = recibirCaracter();
          //Lo almacenamos en un buffer
            buffer[indice++] = caracter;
          //Imprimimos por pantalla el caracter introducido por el usuario
            print(&caracter);
          } while (caracter != '\r' && caracter != '\n' ); //Esperamos a que el usuario presione enter
          print("\r\n");
          //Reiniciamos contador
          indice=0;
          //Si la respuesta es 'Y' iniciamos, si la respuesta es 'N' volvemos a preguntar
          if(buffer[0]=='N' | buffer[0]=='n'){
            inicio=0;}
          else if(buffer[0]=='Y' | buffer[0]=='y'){
            inicio=1;}
          else {
            print("Valor introducido no valido\r\n");
            inicio=0;}
      }
    }
}
```

Figura 43: Bucle principal para Detectar tarjetas NFC

5.3. Simulación

Una vez ejecutadas ambas partes del proyecto, se continua con la simulación del resultado final.

Para la etapa de simulación se emplea el software de vivado, donde se puede simular el proyecto en conjunto con la parte software generada en la plataforma keil, por lo que conseguiremos ver los cambios en las señales internas del microprocesador, las señales de salida mediante los GPIOS y la comunicación mediante la interfaz SPI.

Una de estas señales es el bloqueo interno (LOCKUP) del microprocesador, que se activará cuando existe algún error tanto en el hardware como en el software. Al simular la ejecución del Cortex-M3 también podemos ver los valores iniciales y el cambio en los valores de los registros durante su ejecución, como se puede observar en la Figura 44 con el registro de control del SPI.

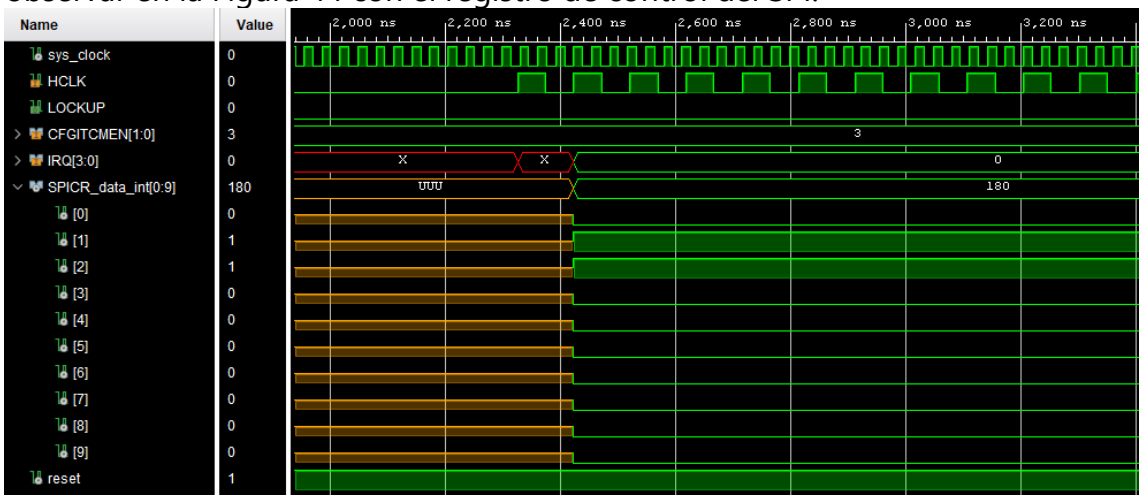


Figura 44: Simulación del Inicio Cortex M3

Por otra parte, también se puede observar los cambios generados en los GPIOS para realizar la inicialización del lector NFC descrito en la Figura 34.

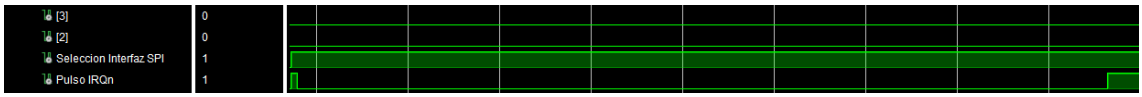


Figura 45: Inicialización Lector NFC

Con respecto a las comunicaciones a través del SPI, en la Figura 43 y Figura 47, se puede observar la ejecución de los comandos y la forma de cómo se van a transmitir a la hora de conectar el Lector NFC.

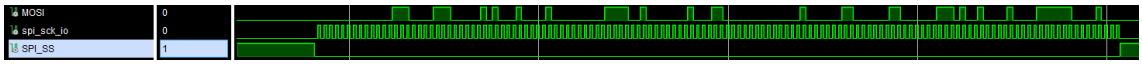


Figura 46: Envío de comandos por el SPI del Cortex-M3

Figura 47: Comando de sondeo a través del SPI del Cortex-M3

Con estas simulaciones se puede evaluar el comportamiento final del sistema a nivel físico y verificar esa correcta correlación entra la parte hardware y la parte software del sistema.

5.4. Montaje

Una de las últimas fases del proyecto, tras haber configurado el hardware, diseñado el software y realizar simulaciones del resultado, es montar el periférico del lector NFC en conjunto con la FPGA.
Al generar el fichero de restricciones descrito en el 5.1 ya hemos especificado la ubicación de cada uno de las entradas y salidas de nuestro sistema. De esta forma, en la Figura 48, se puede observar la conexión final entre la FPGA y el lector NFC.

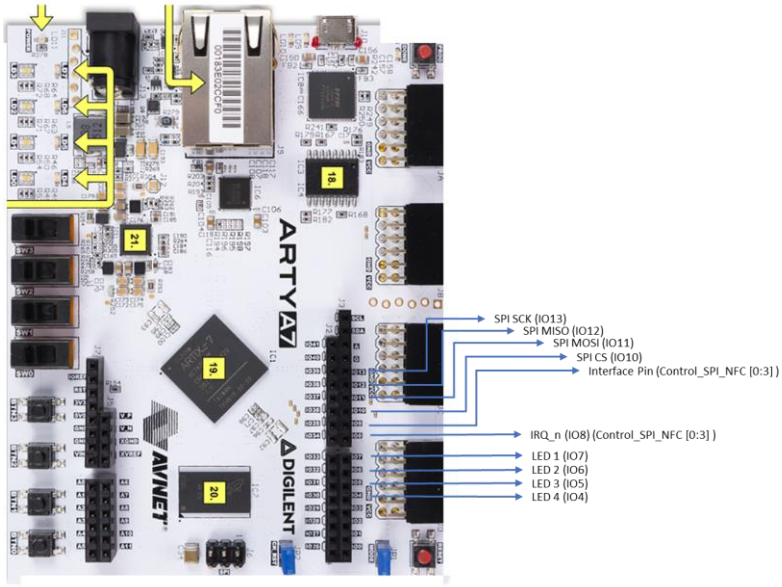


Figura 48: Distribución del conexionado Lector NFC

De esta forma, ya se dispondrá de los pines necesarios para comunicarnos e inicializar el lector NFC.
La conexión del debugger se realiza por el módulo PMOD JD de la placa FPGA. En la Figura 49 se puede observar su conexionado final.

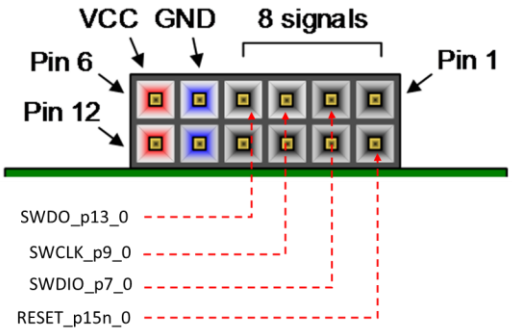


Figura 49: Distribución conexionado del debugger a través de módulo PMOD de la FPGA

Realizando este conexionado, el montaje final del sistema se puede visualizar en la Figura 50.

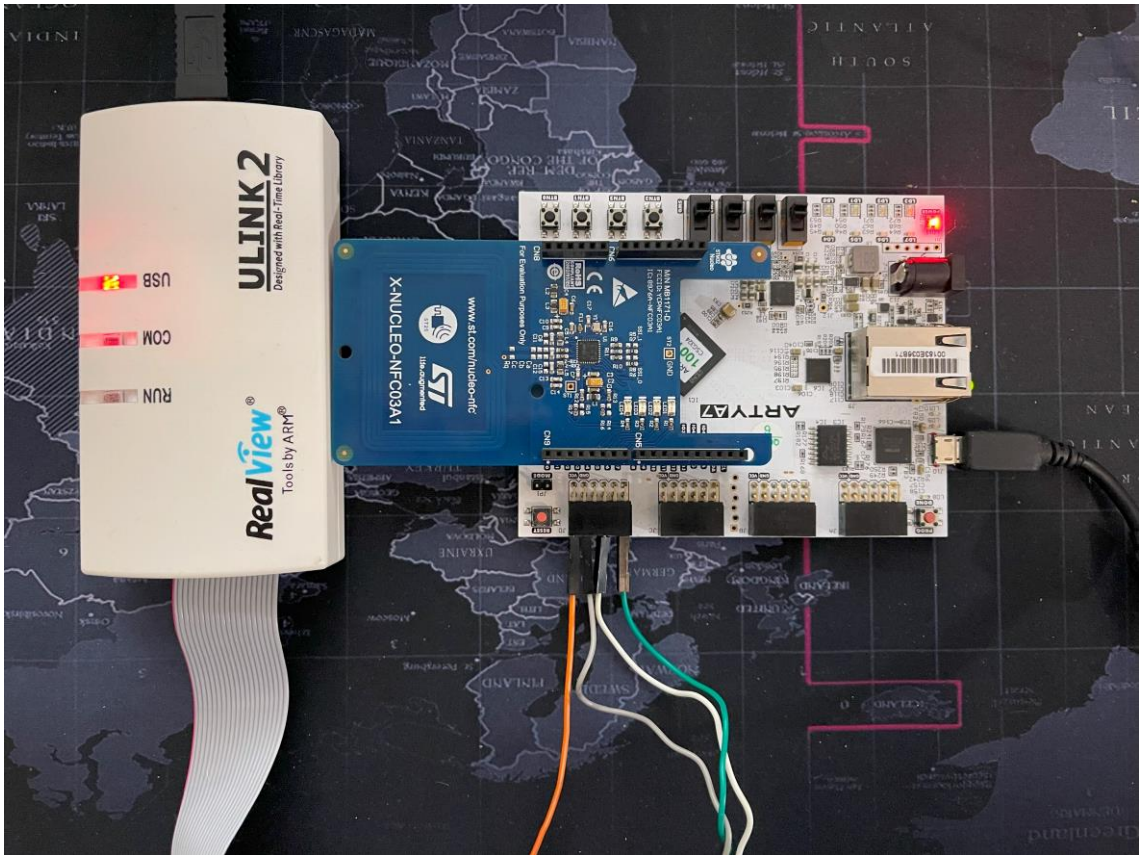


Figura 50: Montaje del sistema final

5.5. Resultados

La última fase del proyecto es verificar los resultados obtenidos durante la simulación y evaluar el desempeño del sistema a la hora de detectar la presencia de tarjetas NFC con su antena RFID.
Durante las primeras pruebas de ejecución, se pudo observar que el desempeño no era el correcto debido a errores durante las etapas previas.
Uno de los problemas que tuvo un mayor tiempo de análisis fue debido a que, en la etapa de simulación, se observaba un correcto funcionamiento del sistema, pero a la hora de cargar el BitStream a la FPGA, el resultado no era el esperado.
El problema se detecta a partir de realizar una simulación tras la sinterización del sistema, de esta manera, se tenía un resultado más preciso de lo que teníamos físicamente, por lo que se consigue evaluar y determinar que el error provenía de una mala configuración hardware.
Tras analizar diferentes problemas detectados, se puedo extraer un entendimiento más consensuado de los procesos internos del micro y esa correlación entre la configuración hardware y software del sistema en su conjunto.

De esta manera, depuramos el sistema final y conseguimos los siguientes resultados:

- **Control de las entradas/salidas digitales:** Se verifica la correcta actuación de las salidas digitales asociadas a los bloques GPIO. Se genera una aplicación software que encienda los leds tanto de la propia placa FPGA como del lector NFC. Esto se puede observar en la Figura 51.

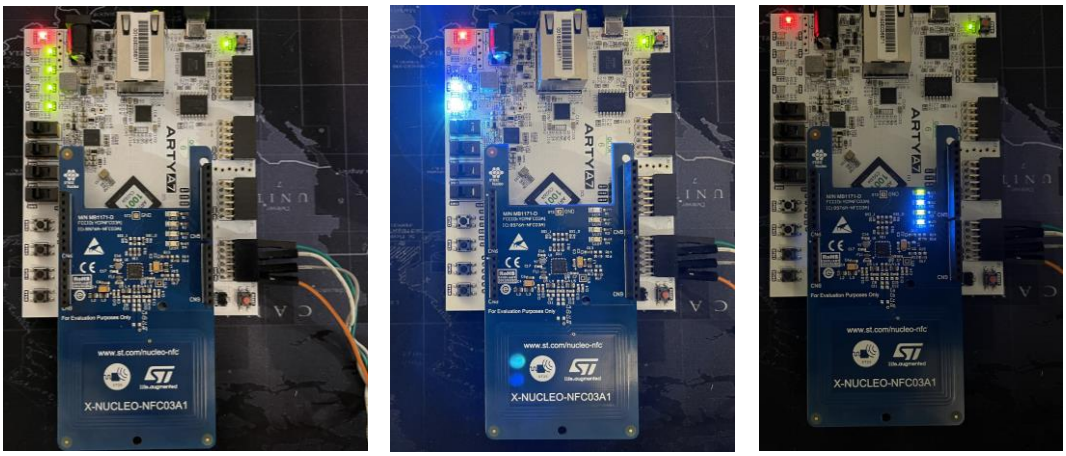


Figura 51: Actuación salidas digitales

- **Comunicación por puerto serie**: Se verifica que la comunicación por puerto serial se realiza correctamente. Mediante el software "*putty.exe*", establecemos una comunicación desde nuestro PC hacia el microprocesador Cortex-M3 que se encuentra implementado en el interior de la FPGA. De esta forma, se genera una aplicación software que permite al usuario introducir palabra por teclado y en consecuencia encender un Led. El resultado de esta conexión se puede observar en la Figura 52.

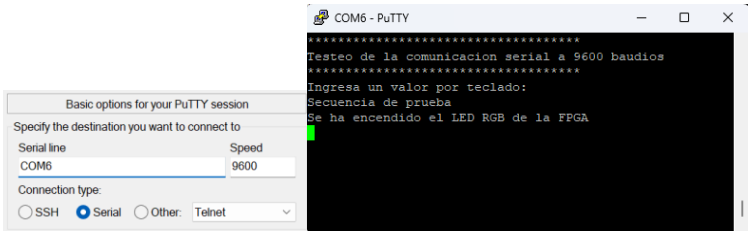


Figura 52: Comunicación por puerto serie

- **Comunicación por la interfaz SPI y detección de tarjetas NFC:** Se verifica la correcta actuación del bloque IP que implementa la comunicación mediante la interfaz SPI. Para ello se genera la aplicación software descrita en el apartado 5.2, donde se hace uso de la interfaz del puerto serial para recibir una orden del usuario y comenzar la detección. Se inicia todo el proceso descrito en la Figura 36 y se culmina volviendo al punto de partida y solicitando nuevamente una interacción al usuario. En la Figura 53, se puede observar los mensajes recibidos por el puerto

serial y la actuación del Led RGB de la FPGA tras acerca una tarjeta en el lector NFC.

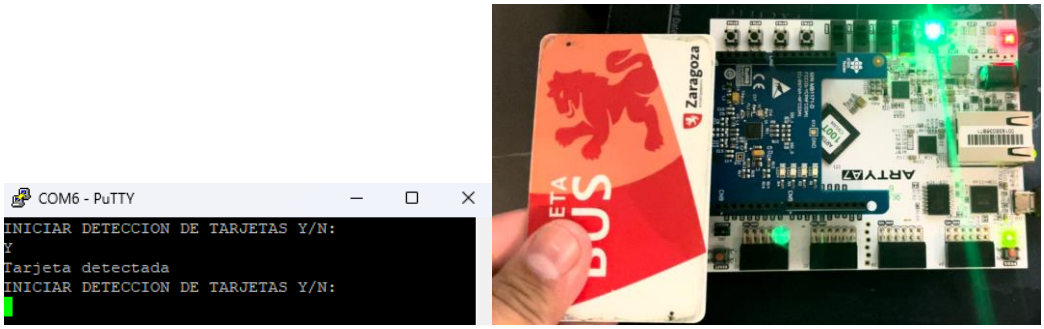


Figura 53: Detención de tarjetas NFC

De esta manera, por una parte, se consigue verificar el correcto funcionamiento del hardware, permitiendo ejecutar todos los periféricos introducidos en el sistema y por otra parte, se consigue una correcta ejecución del software donde la interacción con el lector NFC permite determinar la presencia de una tarjeta NFC.

6. Conclusiones

Este proyecto de fin de grado se centró en la implementación de un microprocesador Soft-Core customizado para trabajar con el periférico X-NUCLEO NFC03A1, abordando la problemática tanto desde el desarrollo hardware como desde el software, para cumplir con los requisitos de comunicación y detección de tarjetas NFC impuestos por el periférico.

En la fase de desarrollo hardware, se implementó el microprocesador Soft-Core Cortex-M3, seleccionando módulos IP específicos para la comunicación SPI, control de GPIO, reloj, reset, comunicación serial y debugger. Tras su identificación, se configuraron los bloques IP de manera precisa para asegurar el correcto funcionamiento con el periférico del lector NFC.

Se configuraron aspectos del Cortex-M3, como la desactivación de la protección de memoria (MPU) y la desactivación del control manual del encendido (WIC) para la inicialización automática del micro. Se ajustaron parámetros relacionados con el debug, la memoria y las interrupciones enmascaradas.

Se consideraron las limitaciones del lector NFC para configurar el controlador del SPI a 2MHz, lo que derivó a establecer la señal de reloj a una frecuencia de 32 MHz para conseguir trabajar con una única señal de reloj y que todas las demás sean múltiplos enteros de esta.

Se incorporaron módulos GPIO para controlar las señales de arranque y Leds del lector NFC y durante la fase de implementación se incluyó la generación de un fichero de restricciones que especifica la ubicación de los pines de entrada/salida en la FPGA, seleccionando y se mapeando los pines de acuerdo con los conectores indicados del lector NFC.

En el desarrollo software, se utilizó Vivado SDK y Keil uVision 5 para generar los archivos BSP y desarrollar la aplicación principal. Se implementaron funciones para enviar comandos al lector NFC, realizar sondeos y leer las respuestas, estableciendo una secuencia de inicialización y detección de tarjetas NFC.

La simulación del sistema en Vivado permitió verificar la correlación entre la parte hardware y software, observando cambios en señales internas, registros y comunicación SPI, pudiendo observar la ejecución del sistema desde una perspectiva más de bajo nivel.

Por último, en el montaje final y durante las pruebas, se identificaron errores en la configuración hardware que afectaban al desempeño del sistema. Se depuraron los problemas y se verificó el correcto funcionamiento, incluyendo el control de salidas digitales y la comunicación con el lector NFC.

Esto ha permitido alcanzar el objetivo inicial del proyecto, logrando customizar y desarrollar un sistema completo para la detección de tarjetas NFC, superando los desafíos en la configuración hardware y software, demostrando un desempeño adecuado en las pruebas realizadas.

7. Desarrollos Futuros

Con los resultados conseguidos durante la ejecución de este proyecto, se ha podido observar las siguientes líneas de desarrollo en el software:

- Implementación de una aplicación software que permita leer la información contenida en una tarjeta NFC, así mismo, que permita escribir en una tarjeta NFC.
- Implementación de una aplicación software que permita detectar el tipo de tarjeta NFC:
    - ISO/IEC 14443- Type A and B tags
    - ISO/IEC 15693 tags
    - ISO/IEC 18000-3M1 tags
    - NFC Forum tags: Types 1, 2, 3 and 4

Con respecto a la plataforma Hardware, se pueden plantear las siguientes líneas de desarrollo:

- Implementación de una plataforma hardware, con similares características, utilizando otros microprocesadores Soft-Core para cumplir con el objetivo de detección y lectura de tarjetas NFC, para analizar el rendimiento de las soluciones propuestas y comparar las metodologías de trabajo entre fabricantes.
- Estudio e implementación de bloques IP, diseñados desde cero, que permita trabajar con otros integrados con unos requerimientos más restrictivos.

8. Bibliografía

[1] ARM Ltd, «Cortex-M3, Technical Reference Manual, r2p0,» 2010. [En línea]. Disponible: https://developer.arm.com/documentation/ddi0337/e/.

[2] ARM Ltd, «Arm Cortex-M on FPGA,» [En línea]. Disponible: https://www.arm.com/resources/free-arm-cortex-m-on-fpga.

[3] STMicroelectronics, «X-NUCLEO-NFC03A1, Data Brief,» 2018. [En línea]. Disponible: https://www.st.com/resource/en/data_brief/x-nucleo-nfc03a1.pdf.

[4] STMicroelectronics, «CR95HF, DataSheet-production data,» 2017. [En línea]. Disponible: https://www.st.com/resource/en/datasheet/cr95hf.pdf.

[5] AMD-Xilinx Inc, «Vivado Design Suite (UG912), Properties Reference, Guide,» 2022. [En línea]. Disponible: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug912-vivado-properties.pdf.

[6] ARM Ltd, «Keil, uVision User's Guide,» 2019. [En línea]. Disponible: https://www.keil.com/support/man/docs/uv4cl/.

[7] J. G.-C. Barrena, Contributions to the Fault Tolerance of Soft-Core Processors Implemented in SRAM-based FPGA Systems, Bilbao, 2018.

[8] anysilicon, «anysilicon,» [En línea]. Disponible: https://anysilicon.com/fpga-vs-asic-choose/.

[9] AMD-Xilinx Inc, «AMD, MicroBlaze V,» 2024. [En línea]. Disponible: https://www.xilinx.com/products/design-tools/microblaze-v.html.

[10] Intel Corporation, «Nios® II, Processor Reference Guide,» 2023. [En línea]. Disponible: https://www.intel.com/content/www/us/en/docs/programmable/683836/current/introduction.html.

[11] ARM Ltd, «Arm®-M3 DesignStart FPGA - Xilinx edition,» 2018. [En línea]. Disponible: https://developer.arm.com/documentation/101483/0000/introduction/cortex-m3-designstart-fpga-xilinx-edition-package?lang=en.

[12] AMD-Xilinx Inc, «Clocking Wizard, LogiCORE IP Product Guide, Vivado Design Suite (PG065),» 2022. [En línea]. Disponible: https://docs.xilinx.com/r/en-US/pg065-clk-wiz.

[13] AMD-Xilinx Inc, «AXI Quad SPI Product Guide, LogiCORE IP Product Guide, Vivado Design Suite (PG153),» 2022. [En línea]. Disponible: https://docs.xilinx.com/r/en-US/pg153-axi-quad-spi/References.

[14] AMD-Xilinx Inc, «AXI GPIO v2.0, LogiCORE IP Product Guide, Vivado Design Suite(PG144),» 2016. [En línea]. Disponible: https://docs.xilinx.com/v/u/en-US/pg144-axi-gpio.

[15] AMD-Xilinx Inc, «Processor System Reset Module v5.0, LogiCORE IP Product Guide, Vivado Design Suite(PG164),» 2015. [En línea]. Disponible: https://docs.xilinx.com/v/u/en-US/pg164-proc-sys-reset.

[16] AMD-Xilinx Inc, «AXI UART Lite v2.0, LogiCORE IP Product Guide, Vivado Design Suite (PG142),» 2017. [En línea]. Disponible: https://docs.xilinx.com/v/u/en-US/pg142-axi-uartlite.

[17] ARM Ltd, «ARM Debug Interface v5, Architecture Specification,» 2006. [En línea]. Disponible: https://developer.arm.com/documentation/ihi0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-ARM-Serial-Wire-Debug--SWD--protocol.

[18] AMD-Xilinx Inc, «AXI Interconnect LogiCORE IP Product, LogiCORE IP Product Guide, Vivado Design Suite (PG144),» 2016. [En línea]. Disponible: https://docs.xilinx.com/v/u/en-US/pg144-axi-gpio.

[19] Digilent Inc, «Arty A7 Reference Manual,» 2024. [En línea]. Disponible: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual.

[20] AMD-Xilinx Inc, «7 Series FPGAs Packaging and Pinout Files,» [En línea]. Disponible: https://www.xilinx.com/support/package-pinout-files/artix-7-pkgs.html.

[21] AMD Xilinx Inc, «MicroBlaze, Processor Reference, Guide, Vivado Design Suit (UG984),» 2021. [En línea]. Disponible: https://www.amd.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2021_2/ug984-vivado-microblaze-ref.pdf.

9. ANEXOS

9.1. ANEXO I: Software implementados para verificar el Hardware

Comando de escritura 0x07 para detectar una tarjeta NFC

```
int WriteSPICommandIdle(XGpio gpio_1)
{
    int status;
    // Comando de escritura hacia el CR95HF
    // 0x00: Byte de control de escritura
    // 0x07: Comando de configuración de encendido por detección de TAG
    // 0x0E: Longitud del comando
    // 0x0A21007901180020606074843F08: Parámetros de configuración
    QSPI_base_tx_buf[0] = 0x00;
    QSPI_base_tx_buf[1] = 0x07;
    QSPI_base_tx_buf[2] = 0x0E;
    QSPI_base_tx_buf[3] = 0x0A;
    QSPI_base_tx_buf[4] = 0x21;
    QSPI_base_tx_buf[5] = 0x00;
    QSPI_base_tx_buf[6] = 0x79;
    QSPI_base_tx_buf[7] = 0x01;
    QSPI_base_tx_buf[8] = 0x18;
    QSPI_base_tx_buf[9] = 0x00;
    QSPI_base_tx_buf[10] = 0x20;
    QSPI_base_tx_buf[11] = 0x60;
    QSPI_base_tx_buf[12] = 0x60;
    QSPI_base_tx_buf[13] = 0x74;
    QSPI_base_tx_buf[14] = 0x84;
    QSPI_base_tx_buf[15] = 0x3F;
    QSPI_base_tx_buf[16] = 0x08;

    //print("Se envia el Control Byte + comando + longitud_Datos + Datos");
    status = XSpi_Transfer(&SGSPI, QSPI_base_tx_buf, QSPI_base_rx_buf, 17);


    return status;
}
```

Comunicación de puerto serie e interacción con usuario

```
    // Test
    print ("************************************\r\n");
    print ("Testeo de la comunicacion serial a 9600 baudios\r\n");
    print ("************************************\r\n");

    print("Ingresa un valor por teclado:\r\n");

    // Leer un valor hasta que se presione Enter ('\r' o '\n')
    char caracter;
    //Buffer de 128 caracteres
    char buffer[128];
    int indice = 0;
    do {
      //Esperamos a recibir algún caracter por teclado
      caracter = recibirCaracter();
      //Lo almacenamos en un buffer de entrada
      buffer[indice++] = caracter;
      //Imprimimos por pantalla el caracter introducido por el usuario
      print(&caracter);
    } while (caracter != '\r' && caracter != '\n' );
    print("\r\n");
    XGpio_DiscreteWrite(&gpio_1, 1, 0x01);  // LED RGB
    print("Se ha encendido el LED RGB de la FPGA\r\n");
```

Detección de Tarjeta NFC e interacción con usuario mediante el puerto serie.

```
78      /*Iniciar detección
79        Inicio=0 No iniciar
80        Inicio=1 Iniciar
81      */
82      int inicio=0;
83      while ( 1 )
84      {
85        if(inicio==1){
86            //Enviamos el comando por el módulo SPI
87            status = WriteSPICommandIdle(gpio_1);
88            //Esperamos a que lector esté listo
89            status = PollingSPI(gpio_1);
90            //Leemos la respuesta
91            status = ReadSPICommandIdle(gpio_1);
92            //En caso que la respuesta corresponda a una Tarjeta NFC
93            if(status==1){
94              //Volvemos al estado inicial
95              inicio=0;
96              //Indicamos que hemos detectado una tarjeta
97              print("Tarjeta detectada\r\n");
98            }
99          }
100       else{
101           //Mensaje por pantalla para empezar a detectar tarjetas
102           print("INICIAR DETECCION DE TARJETAS Y/N:\r\n");
103           do {
104           //Esperamos a recibir la respuesta por teclado
105             caracter = recibirCaracter();
106           //Lo almacenamos en un buffer
107             buffer[indice++] = caracter;
108           //Imprimimos por pantalla el caracter introducido por el usuario
109             print(&caracter);
110           } while (caracter != '\r' && caracter != '\n' ); //Esperamos a que el usuario presione enter
111           print("\r\n");
112           //Reiniciamos contador
113           indice=0;
114           //Si la respuesta es 'Y' iniciamos, si la respuesta es 'N' volvemos a preguntar
115           if(buffer[0]=='N' | buffer[0]=='n'){
116             inicio=0;}
117           else if(buffer[0]=='Y' | buffer[0]=='y'){
118             inicio=1;}
119           else {
120             print("Valor introducido no valido\r\n");
121             inicio=0;}
122       }
```

Lectura de la respuesta del comando 0x07

```
50  int ReadSPICommandIdle(XGpio gpio_1)
51  {
52
53      //Estado de la función
54      int status;
55      //Variable para saber si ha detectado una tarjeta
56      int detectar=0;
57      //Comando 0x02 para leer datos provenientes del Lector NFC
58      QSPI_base_tx_buf[0] = 0x02;
59      //Inicializamos buffer
60      for(int i=1;i<32;i++){
61        QSPI_base_tx_buf[i] = 0x00;
62        QSPI_base_rx_buf[i] = 0x00;
63
64      }
65      //Enviamos comando y recibimos respuesta
66      status = XSpi_Transfer(&SGSPI, QSPI_base_tx_buf, QSPI_base_rx_buf,4);
67      //Comprobamos si que el FLAG de respuesta corresponde al 0x000102 Detección de TAG
68      if(QSPI_base_rx_buf[1]==0x00 && QSPI_base_rx_buf[2]==0x01 && QSPI_base_rx_buf[3]==0x02){
69        XGpio_DiscreteWrite(&gpio_1, 1, 0x02);  // Encendemos LED RGB
70        delay_cycles1(3200000);                 // Encendemos el LED RGB indicando que se ha detectado una tarjeta
71        XGpio_DiscreteWrite(&gpio_1, 1, 0x00);  // Apagamos LED RGB
72        detectar=1;                             // Devolvemos el valor 1 indicando que se ha detectado un tarjeta
73      }
74      return detectar;
75  }
```

Sondeo del lector NFC

```
int PollingSPI(XGpio gpio_1)
{
    //Estado de la función de transferencia de datos del SPI
    int status;
    //Variable que indica si se ha detectado el FLAG para poder leer datos del lector NFC
    int resultado=0;
    //Comando de sondeo al lector NFC
    QSPI_base_tx_buf[0] = 0x03;
    //Inicialización buffer de transmisión y recepción
    for(int i=1;i<32;i++){
      QSPI_base_tx_buf[i] = 0x03;
      QSPI_base_rx_buf[i] = 0x00;
    }
    //Esperamos a que el Lector NFC nos devuelva el Flag para leer
    while(resultado==0){
      //Cada petición de sondeo está retrasado 3000 ciclos de reloj
      delay_cycles1(3000);
      //Enviamos el comando de sondeo
      status = XSpi_Transfer(&SGSPI, &QSPI_base_tx_buf[0], &QSPI_base_rx_buf[0],2);
      //Detectamos si hemos recibido un 1 en la posición 4
      //que indica que se puede enviar una petición de lectura al Lector NFC
      resultado=QSPI_base_rx_buf[1]&0x08  ;
      if(resultado!=0)
      {
        //Encendemos El LED RGB
        XGpio_DiscreteWrite(&gpio_1, 1, 0x01);
      }
    }
    return status;
}
```

Código para recibir los valores por puerto serie

```
static XUartLite UART0_instance;

char recibirCaracter() {
    char caracter;
    // Esperamos hasta que haya datos en el registro de recepción
    while (XUartLite_IsReceiveEmpty(UART0_instance.RegBaseAddress)) {
    }
    // Lee un solo byte del registro de recepción
    caracter = XUartLite_RecvByte(UART0_instance.RegBaseAddress);
    // Devuelve el caracter leido
    return caracter;
}
```

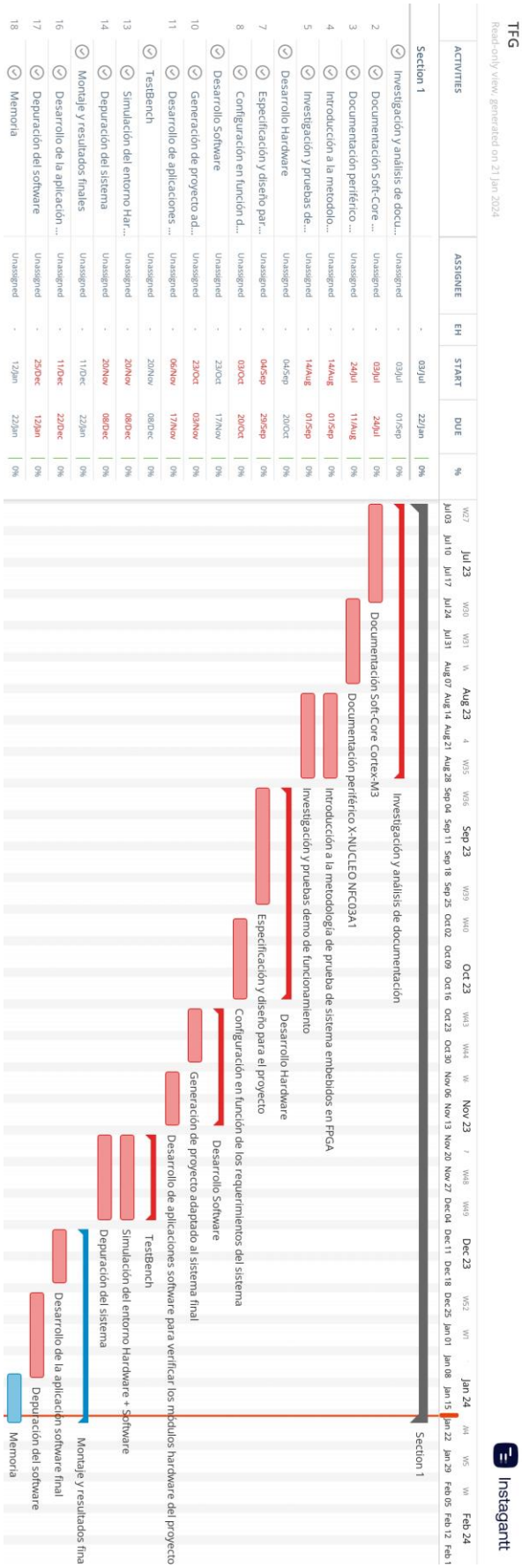
Código para controlar los GPIOS

```
    // Inicializar el GPIO 0
    status = XGpio_Initialize(&gpio_0, XPAR_AXI_GPIO_0_DEVICE_ID);
    if (status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    // Inicializar el GPIO 1
    status = XGpio_Initialize(&gpio_1, XPAR_AXI_GPIO_1_DEVICE_ID);
    if (status != XST_SUCCESS) {
      return XST_FAILURE;
    }

    XGpio_DiscreteWrite(&gpio_0, 1, 0xF); //Encendido de LED's FPGA
    XGpio_DiscreteWrite(&gpio_1, 1, 0xFF); //Encendido de LED's RGB FPGA
```

9.2. ANEXO II: Planificación



9.3. ANEXO III: Características módulos IP del proyecto

9.3.1. Processor System Reset

El bloque del reset permite generar un pulso digital para reiniciar los sistemas conectados a él. El esquemático de este bloque IP y su diagrama funcional es el mostrado en la Figura 54.

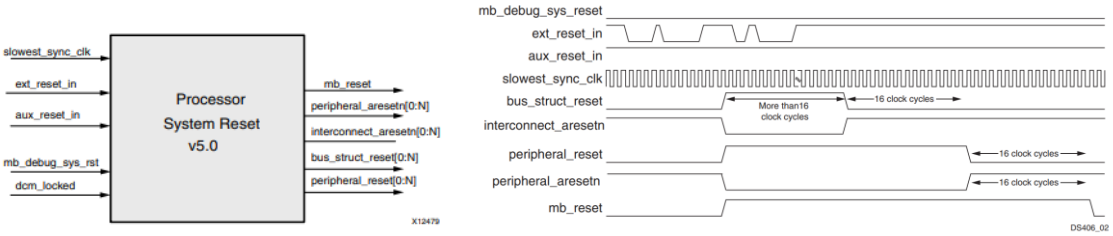


Figura 54: Bloque System Reset

- **Ext Reset Active Width**: Controla el número de ciclos de reloj que tiene que estar activa el pulso de entrada "*ex_reset_in*" para ser detectada como una señal de reset válida y actuar en consecuencia. En la Figura 55 se puede observar este ajuste desde el software de Vivado.

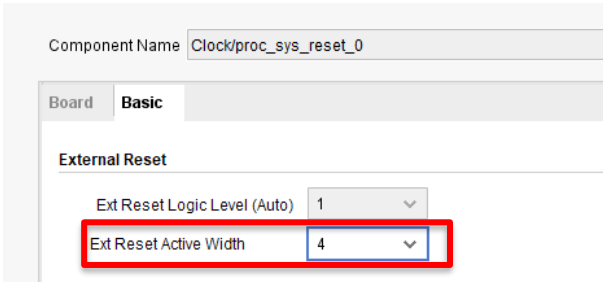


Figura 55: Configuración del Bloque System Reset

El restablecimiento se activa en seis o siete ciclos de reloj después de que la entrada se ha activado, permaneciendo activo durante cinco relojes. Después de que la señal "*ext_reset_in*" ha pasado inactivo durante cinco ciclos de reloj, comienza la secuencia para salir del reinicio.

- **ext_reset_in**: Señal de entrada que indica el comienzo del reset. Debe mantenerse durante al menos 5 ciclos de reloj.
- **mb_debug_sys_rst**: Señal de entrada que tiene un comportamiento similar a la señal *"ext_reset_in"*, aplicando las mismas condiciones de activación. La señal resultante es "*mb_reset*" y se utiliza para debugear el sistema.
- **Bus_structure**: Señal de salida utilizada para interconectar módulos y suministrarles la señal de reset de forma síncrona.

- **Interconect_areset**: Señal de salida utilizada para interconectar módulos y suministrarles la señal de reset de forma asíncrona.
- **dcm_locked**: Señal de referencia para conocer el estado de la señal de reloj de entrada.

La configuración final de este bloque IP aplicado a nuestro sistema de detección de tarjetas NFC se especificará en el apartado 5.1

9.3.2. Clocking Wizard

En la Figura 56 se puedese observar el bloque de reloj "Clocking Wizar". Este bloque IP permite generar diferentes circuitos de sincronización para la salida de relojes en frecuencia por fase y duty cycle.

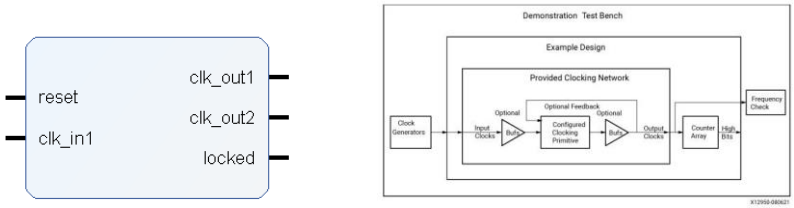


Figura 56: Diagrama de bloques y modelo del asistente de sincronización

Para conseguir estos circuitos de sincronización se utiliza dos métodos de funcionamiento:

- **MMCM**: Mixed mode clock manager
- **PLL**: Phase-locked loop

Por defecto, todos los puertos son opcionales a excepción de la entrada de un reloj interno y un reloj de salida.

Puerto	E/S	Descripción
Clk_in1	I	Entrada del reloj primario
Clk_in2	I	Entrada del reloj secundario
Clk_in_sel	I	Selector de reloj de entrada
Clk_out1	O	Salida de reloj 1
Clk_out2	O	Salida de reloj 2
Locked	O	Señal de control que permite indicar que la señal de reloj de salida es estable y puede ser utilizada por los bloques siguientes.

9.3.3. AXI interconect

Este bloque IP permite la comunicación entre el núcleo y los diferentes componentes del sistema, facilitando la transferencia de datos de forma eficiente y estandarizada. En la Figura 57 se pueden ver los ajustes de alto nivel donde se

especifica el número de interfaces esclavo/maestro que contemplará el desarrollo.

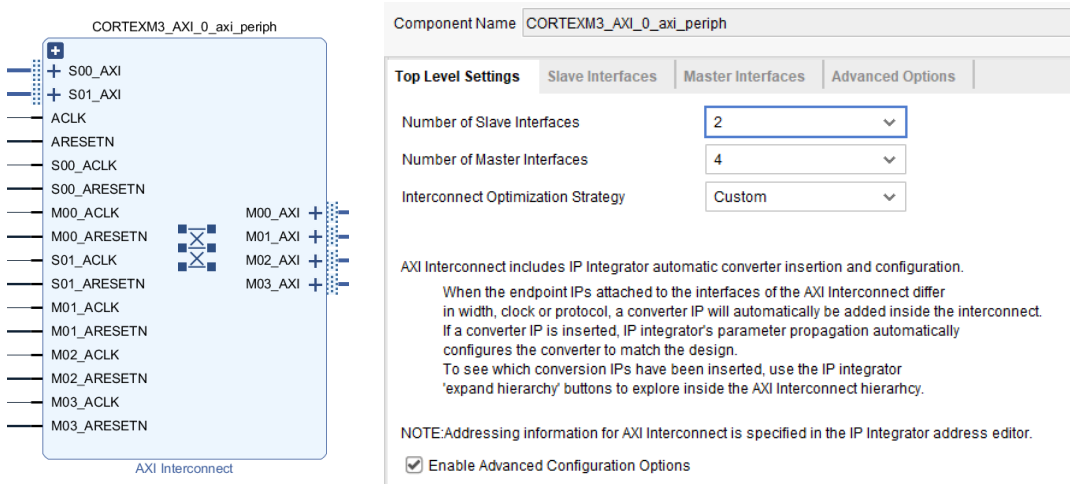


Figura 57: Opciones de configuración bloque "Axi interconect"

En la Figura 58, se puede observar que es posible configurar el modo de operación con el resto de los bloques del sistema conectado a él, permitiendo habilitar segmentos de registro por cada conexión o habilitar una FIFO de datos,

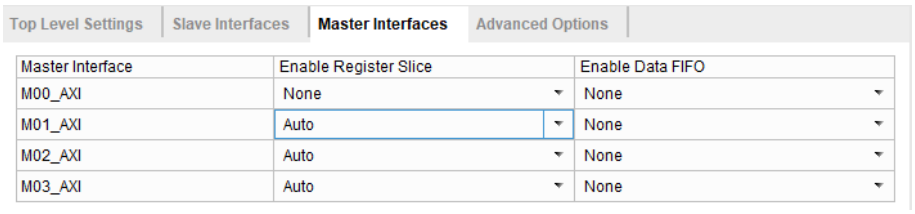


Figura 58: Opciones de configuración de las conexiones máster

En la Figura 59, se puede observar los ajustes avanzados del bloque IP, donde es posible seleccionar el ancho de los datos asociado a los registros de entrada y salida, así como el número de ciclos que estará inactivo hasta que llegue al estado Ready del micro.

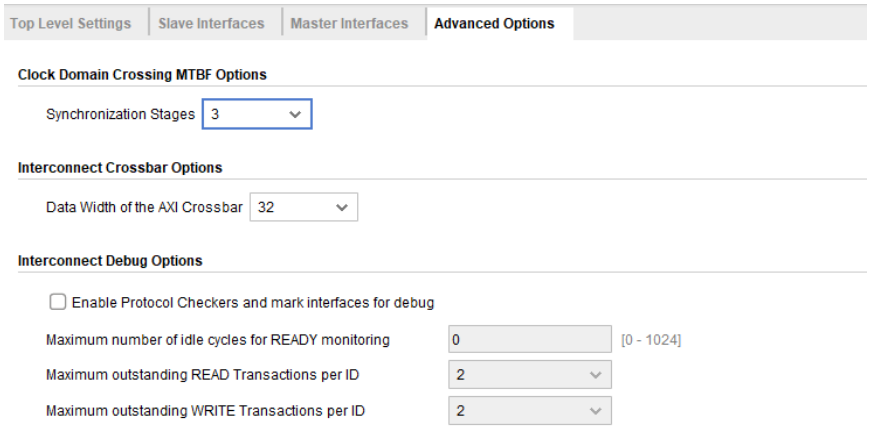


Figura 59: Opciones de configuración bloque "Axi interconect"

9.3.4. AXI Uartlite

El bloque AXI UART (Universal Asynchronous Receiver Transmitter) Lite, permite establecer una interfaz de control para la transferencia de información mediante una comunicación serial Asíncrona.

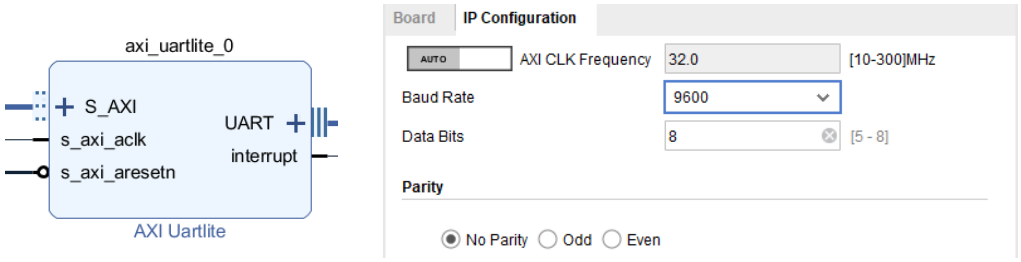


Figura 60: Opciones de configuración bloque "Axi Uartlite"

En la Figura 60, se pueden observar los diferentes ajustes de configuración como la velocidad de la comunicación, el número de bits y la inclusión del bit de paridad.

9.3.5. AXI Quad SPI

Este bloque IP implementa un sistema SPI (Serial peripheral interfaz) para establecer una comunicación asíncrona y transferir datos mediante un esquema maestro-esclavo.

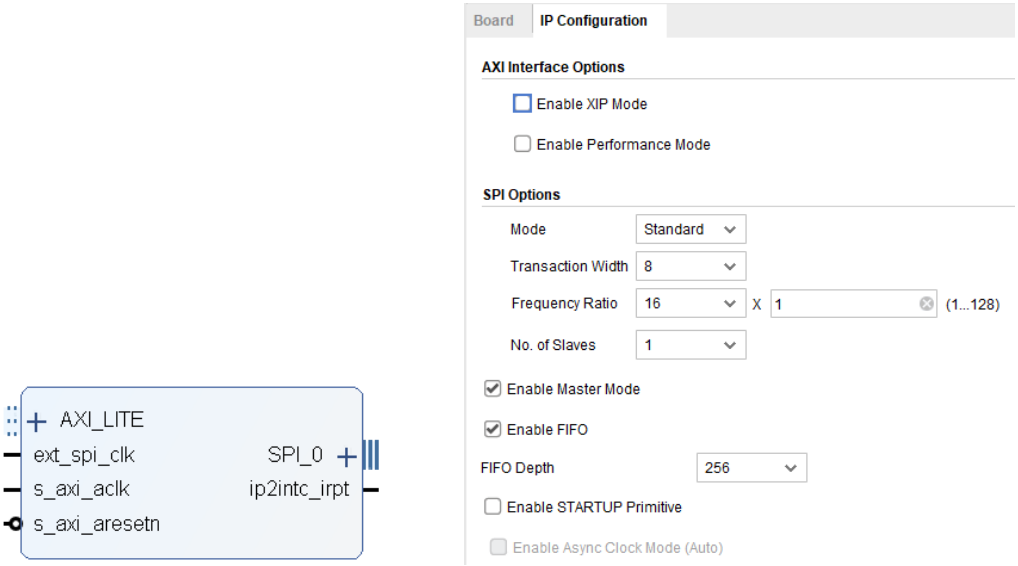


Figura 61: Opciones de configuración bloque "Axi Quad SPI"

En la Figura 61 se observan los distintos ajustes de configuración, donde destaca el modo de funcionamiento (Standard, dual, Quad), el ratio de frecuencia (Relación de la frecuencia de trabajo con la frecuencia del reloj de entrada) que permite establecer la velocidad de la comunicación, el modo de operación (Maestro), la habilitación de una FIFO para la transferencia de datos y el tamaño de esta.

9.3.6. AXI GPIO

Módulo IP que implementa el control de las entradas/salidas digitales de propósito general de las que dispone la placa de desarrollo.

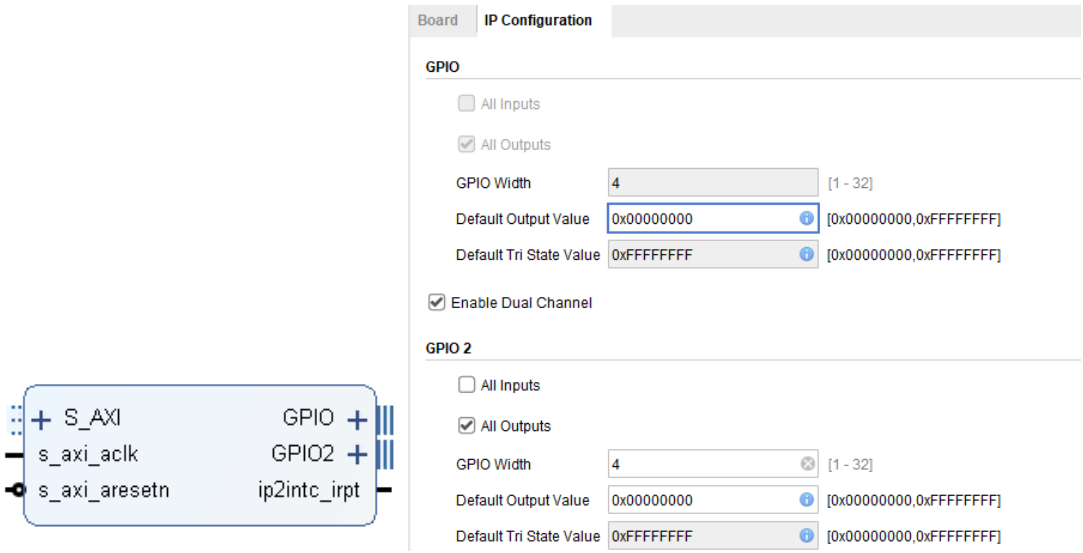


Figura 62: Opciones de configuración bloque "Axi GPIO"

En la Figura 62, se puede observar que es posible configurar por cada instancia GPIO 2 canales de control. También es posible establecer el sentido de la comunicación (entradas / salidas).