Universidad de Alcalá

Departamento de Electrónica

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA ELECTRÓNICA

Trabajo Fin de Máster

Implementación de dispositivos ARM sobre FPGAs de Xilinx.

Autor: Francelly Katherine Cano Ladino

Director/es: Ignacio Bravo Muñoz

2020

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior
Departamento de Electrónica

Máster Universitario en Ingeniería Electrónica

Trabajo Fin de Máster

Implementación de dispositivos ARM sobre FPGAs de Xilinx

Autor: Francelly Katherine Cano Ladino.

Director/es: Ignacio Bravo Muñoz.

Comisión Evaluadora:

Presidente: Francisco Javier Meca Meca

Vocal 1º: Santiago Cóbreces Álvarez

Vocal 2º: Álvaro Hernández Alonso

Fecha

A mi familia y a mi yo atemporal.

Tabla de contenido

Índice de figuras

Índice de tablas

Resumen

Este proyecto se centra en la implementación de microprocesadores Cortex-M1 de la plataforma DesignStart FPGA de ARM para entornos de XILINX. Estos microprocesadores son de libre acceso por lo cual es interesante estudiar su comportamiento en una FPGA de la familia Artix-7, en concreto la Nexys 4 DDR. De esta forma, es posible conocer de primera mano como es el desarrollo de nuestro propio SoC con uno de los procesadores más sencillos de ARM.

Palabras clave: FPGA, Cortex-M1, ARM, DesignStart FPGA, XILINX.

Abstract

This project is focused on the implementation of Cortex-M1 microprocessors of ARM's DesignStart FPGA platform for XILINX environments. These microprocessors are open-source, therefore it is interesting to study their behavior in an FPGA of the Artix-7 family, specifically the Nexys 4 DDR. In this way, it is possible to know first-hand what the development of our own SoC is like with one of the simplest ARM processors.

Keywords: FPGA, Cortex-M1, ARM, DesignStart FPGA, XILINX.

I. Contenido

1.1. Introducción

1.1.1. Presentación

La FPGA (Field Programmable Gate Array) es una matriz de puertas reprogramables. Está basada en una matriz configurable de bloques lógicos configurables que se conectan entre sí por medio de interconexiones programables, permitiendo al diseñador crear circuitos hardware. Esta es una de las características más atractivas de la FPGA, que permiten generar nuevos dispositivos o componentes lógicos programables.

Por otro lado, los procesadores son uno de los elementos más flexibles dadas sus prestaciones a la hora de su programación de lenguajes a alto nivel. A la hora de clasificar los micrprocesadores, existen diferentes categorías. Así una de ellas es la que emplea el tipo de implementación para su clasificación. Estos pueden ser clasificados como *soft* o *hard core.*Esta clasificación es la más habitual cuando se trata de denominar los tipos de microprocesadores que pueden ser implementados en una FPGA. Los ***soft processors***, ***soft core processor*** o ***soft microprocessors***, están implementados de forma dinámica por los recursos lógicos de la FPGA, como bloques lógicos (LBs), interfaces de interconexión y bloques de memoria RAM, convirtiéndolos en una arquitectura flexible. Este tipo de procesador consta de un núcleo, un conjunto de periféricos on-chip, memoria e interfaces de memoria off-chip.

La otra alternativa son los *hard processor*, los cuales ya tienen fijada un área de silicio en específico desde su proceso de fabricación. Una de las ventajas que tienen este tipo es el menor consumo de potencia, aunque esto también puede provocar limitaciones ya que al ser un sistema fijo no todos los recursos serán utilizados en las aplicaciones o también pueden ser insuficientes. Por ello es interesante estudiar más acerca de los *soft processor***,** por su flexibilidad permiten "personalizar" el sistema según el funcionamiento que requiera. La escalabilidad y la reducción de la obsolencia también forman parte de sus ventajas [1] .Aunque también se debe de contemplar los elementos limitantes como que no todos los fabricantes permitan la migración de los recursos lógicos entre ellos o también la autoría de las IP de los soft processor, no esté disponible o accesibles.

Los *soft core* se pueden dividir en dos grupos. Primero los que forman parte de los núcleos patentados, que son asociados a un fabricante, por ejemplo Microblaze de Xilinx. Los segundo son los *open-source cores*, que es una tecnología independiente y permite implementarse en dispositivos de diferentes fabricantes. En este proyecto nos vamos a centrar en el segundo grupo, en concreto los ***Cortex-M*** *de* ***ARM.***

1.1.2. Objetivos y metodología

El objetivo de este proyecto es realizar un estudio sobre el funcionamiento e implementación de varios núcleos de ARM sobre la FPGA usando para ello microprocesadores de núcleo blando que se implementan directamente con la

lógica integrada de la FPGA. Esto nos permite descubrir alternativas por ejemplo a los núcleos como Microblaze o PicoBlaze para Xilinx.

ARM permite a los diseñadores de diseñar sus propias soluciones SoC, utilizando por ejemplo microprocesadores del tipo Cortex-M. Mediante su plataforma ARM DesignStart que ofrece las IP sin necesidad de costes o licencias, se pueden desarrollar soluciones que ofrezcan un producto diferente. También tiene un gran interés académico, ya que es interesante conocer como se pueden hacer diseños SoCs.

Para poder realizar este trabajo se ha estructurado en tres partes:

- Estudio teórico de microprocesador ARM Cortex-M1 en el entorno de Xilinx. Conocer sus características y aspectos más importantes
- Determinar cuales de los diferentes tipos de comunicación entre núcleos es la más apropiada para poder lograr la implementación de más de un núcleo.
- Implementación dese uno hasta cuatro núcleos, incluyendo memorias externas y periféricos.

1.1.3. Estructura del documento

En esta introducción también nos interesa describir la estructura de todo este documento.

En el capitulo 2 se explica la temática de los ***Open Source Core***, y se explica con un poco más de detalle cual microprocesador se seleccionó. Por otra parte en el capitulo 3, se explican las tarjetas FPGA de Xilinx, que se pueden utilizar.

En el capitulo 4, se introduce un poco en el interfaz de comunicación que tiene este microprocesador, se explica en líneas generales como funcionan.

Continuando, encontramos que en el capitulo 5, se encuentra una explicación más extensa de la memoria de este núcleo, su mapa de memoria así como sus interfaces. Además, se explican dos memorias externas que se utilizarán en nuestra implementación, la memoria Flash y una memoria DDR2.

En el capitulo 6 y 7, se explican los diferentes tipos de comunicaciones ***on-chip***, este es un tema muy importante porque nos ayudará a entender como funcionarán los núcleos que se implementan dentro de un sistema. También se explica como es la metodología para hacer un banco de pruebas a una implementación como esta.

El capitulo 8, explica todos los detalles de la implementación que cosas se hicieron y como se hicieron, junto con los resultados que se obtuvieron. Además, el capitulo 9, se tienen las conclusiones que se han obtenido con esta implementación y que mejoras se pueden realizar en un futuro trabajo.

Al final se tienen un pliego de condiciones para realizar este trabajo, como el presupuesto que se necesita. Una parte fundamental está en el manual de memoria que ayudará a comprender todo lo necesario para hacer esta implementación.

2.1. Open Source Cores

Los procesadores Cortex-M de ARM están pensados para aplicaciones de microcontroladores, es decir, para aplicaciones de sistemas embebidos o de IoT (Internet de las cosas). Las ventajas que añaden al incluirlos dentro de la FPGA es que proporcionan un buen funcionamiento en un área pequeña de silicio y un fácil desarrollo software. [1]

Los diseños SoC basados en los Cortex-M, son utilizados ampliamente desde aplicaciones de transporte, comunicaciones o del automóvil. Dado que el sector IoT está en crecimiento, requiere una gran oferta de este tipo de diseños SoC.

La familia de procesadores Cortex-M de ARM posee tres arquitecturas con sus diferentes variedades, dentro de ellas la más sencilla es la ARMv6-M, en la cual podemos encontrar los procesadores *Cortex-M0, Cortex-M0+* y el *Cortex-M1*. También cabe destacar de la arquitectura ARMv7-M, el procesador *Cortex-M3.* Estos procesadores anteriormente mencionados se pueden integrar dentro de un amplio rango de diseños SoC. Fueron diseñados para ser optimizados para la mayoría de los dispositivos de FPGA, ya que son pequeños y permiten una alta frecuencia de operación, al mismo tiempo pueden ser portables entre diferentes tipos de FPGA y también es compatible con otros procesadores Cortex-M, como por ejemplo los procesadores Cortex-M0 y Cortex-M0+.

La empresa ARM dentro de su ecosistema de programas (ARM Flexible Access y ARM DesignStart) permite con **DesignStart FPGA** usar de forma rápida y gratuita, los soft CPU IP como el Cortex-M1 y el Cortex-M3 para investigación, evaluación o uso comercial. Por una parte el Cortex-M1 se utiliza más para aplicaciones más sencillas, las cuales requieren comunicación y control. Por otra parte el Cortex-M3 se utiliza para un amplio abanico de aplicaciones embedded e IoT. [2]

Cortex-M1	**Cortex-M3**
Acceso a depuración a toda la memoria y registros en el sistema	Unidad de protección de memoria (MPU)
Puerto de acceso a depuración (DAP)	Punto de ruptura (Breakpoint) y Parche Flash (FPB)
Unidad BreakPoint (BPU)	Data watchpoint y Unidad de rastreo (DWT)
Data WatchPoint (DW)	Unidad Instrumental Trace Macrocell (ITM)
	Embedded Trace Macrocell (ETM)
	Bus de acceso (AHB-AP)
	Interfaz de rastreo AHB (Interfaz HTM)
	Unidad Trace Port Interface (TPIU)
	Controlador Interrupción Wake-up (WIC)

	Puerto para depuración con interfaz AHB-AP
	Constante de control AHB

Tabla 1. Comparación entre los dos procesadores.

Centrándonos en el Cortex-M1, es un soft CPU core pensado para ser utilizado en aplicaciones embebidas. Dispone de una arquitectura ARMV6-m RISC (Reduced Instruction Set Computer) de 32 bits, esta arquitectura sólo admite instrucciones del tipo Thumb/Thumb-2, incluye también entre otras funcionalidades como el controlador de interrupciones vectorizadas (NVIC), lógica de depuración configurable e interfaz TCM (Tightly Coupled Memory) para conectar memorias. El interfaz de comunicación que usa es el AMBA AHB-Lite. Este procesador puede utilizarse tanto con Microsemi, Intel o Xilinx, que es nuestro caso y operar a frecuencias desde 70 a 200 MHz según la tarjeta que se utilice.

El paquete para la prueba del procesador Cortex-M1 que proporciona DesingStart FPGA, ofrece las características básicas mostradas en la Figura 1.

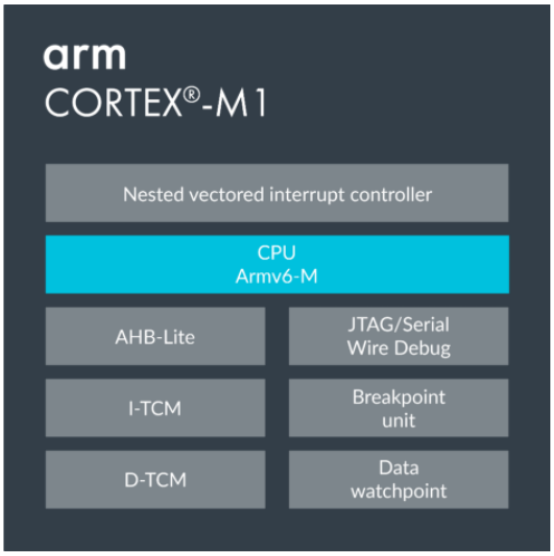


Figura 1. Soft core processor Cortex-M1 [2].

De forma detallada estos son los principales aspectos de estos núcleos:

- 32 interrupciones como máximo.
- Interfaces ITCM (Instruction Tightly Coupled Memory) hasta 1MB y DTCM(Data Tightly Coupled Memory) también hasta 1 MB. Ofrece la posibilidad de no utilizar la DTCM, en ese caso la ITCM deberá compartir espacio para la memoria de datos y de programa.
- Depuración por línea serie (SW), JTAG o combinación de ambas.
- Extensiones configurables: OS, formato Little-endian, soporte de depuración integrado.

Como se había dicho anteriormente, el Cortex-M1 utiliza como interfaz de comunicación el bus AMBA AHB-Lite, pero para poderlo usar con el resto de herramientas que ofrece Xilinx, el procesador que ofrece DesignStart FPGA, integra

un bridge entre AHB y AXI, de esta manera se pueden utilizar las IPs estándar de Vivado.

Describiendo un poco más en profundidad este microprocesador. Vamos a enumerar las funcionalidades más representativas que dispone.

- **Instrucciones:** Utiliza la arquitectura ARMV6-m RISC, este microprocesador ejecuta un subconjunto de instrucciones *Thumb-2*, las cuáles incluyen instrucciones Thumb de 16-bit y unas cuantas de Thumb2 de 32 bits (BL,MRS,MSR,ISB,DSB y DMB). El procesador puede operar en dos estados, uno de ellos es el estado ***Thumb state***, es el modo de operación normal que ejecuta el conjunto de instrucciones de los dos tipos nombrados. Por otra parte, el estado ***debug state***, cuando se detiene una depuración.

  Este tipo de operación permite escribir código eficiente para un núcleo que tenga memoria limitada y se suele utilizar en aplicaciones embedded.

  Cuando trabaja con instrucciones de 16 bits, permiten acceder al doble de densidad en memoria del código ARM de 32 bits estándar. Este procesador no soporta instrucciones tipo ARM. [3]
- **Registros**: Contiene trece registros de 32 bits, definidos entre los registro bajos (R0-R7) y los registros altos (R8-R12). Los registros más importantes son: *Stack Pointer* (SP), Registro de enlace (LR), Contador de Programa (PC) y el registro de estado del programa (xPSR).
- **Interfaz de memoria:** Tiene dos interfaces de memoria. Una de ellas es para el acceso a memoria ITCM, memoria del programa. La otra es para acceso a la memoria DTCM, memoria de datos.
- **NVIC:** Se pueden configurar hasta 32 interrupciones. El procesador configura de forma automática el número de interrupciones.
- **Depuración:** Hay dos formas de configuración posible. La primera, se puede hacer una depuración amplia o reducida, la IP dispone para esta forma se tienen 3 configuraciones diferentes (JTAG, Serial Wire y JTAG con Serial Wire)Figura 2. La segunda es no disponer de la depuración Figura 3.

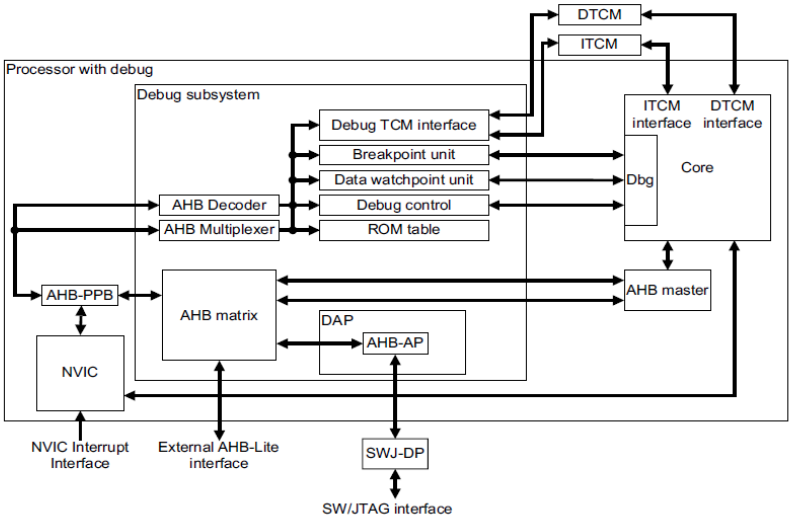


Figura 2.Diagrama de bloques del microprocesador con la opción de depuración. [4]

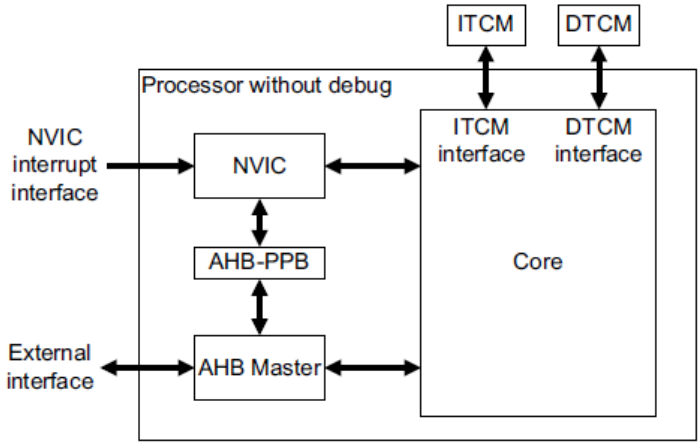


Figura 3. Diagrama de bloques del microprocesador sin depuración. [4]

La interfaz de la IP del Cortex-M1 tiene la apariencia que vemos en la Figura 4, se va a explicar brevemente las señales que tiene, para la configuración que viene por defecto.

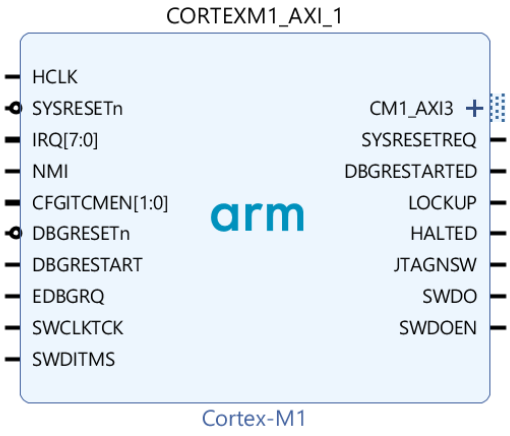


Figura 4. IP del Cortex-M1.

1. Señales de reloj y reset:

Nombre	Tipo	Descripción
HCLK	Entrada	Señal de reloj principal del procesador.
SYSRESETn	Entrada	Señal de reset del sistema. Resetea tanto el procesador como la parte no depurada para el NVIC.
DBGRESETn	Entrada	Resetea la lógica de depuración.

2. Señales de la interfaz de interrupciones :

Nombre	Tipo	Descripción
IRQ	Entrada	Señales externas de interrupción.
NMI	Entrada	Non-maskable interrupt

3. Resto de señales

Nombre	Tipo	Descripción
CFGTCMEN[1:0]	Entrada	Habilita el Alias ITCM. En el primer bit se fija el Upper Alias Enable y en el bit 1 fija al Lower Alias Enable.
DBGRESTART	Entrada	Señal de reset externa.
DBGRESTARTED	Salida	Señal de acuerdo para la señal DBGRESTART.
EDBGRQ	Entrada	Señal de externa de solicitud de depuración.
SYSRESETRED	Salida	Solicita que el controlador de reset del sistema reinicie el núcleo.
LOCKUP	Salida	Indica que el núcleo está bloqueado.
HALTED	Salida	Indica la detención del modo de depuración.
SWCLKTCK	Entrada	Señal de la interfaz de depuración, puede ser la señal de reloj del JTAG o de la depuración Serie. Se utiliza con el depurador Daplink.
SWDITMS	Entrada	Datos de entrada de la depuración serie o selección del modo de test del JTAG.
JTAGNSW	Salida	Esta señal define el bloque SWJ está en modo JTAG(1) o SW (0).
SWDO	Salida	Datos de salida de el modo de depuración serie.
SWDOEN	Salida	Señal de salida de habilitación modo serie.

3.1. Tarjetas FPGA

Para poder utilizar los soft cores de la licencia de ARM DesignStart FPGA, se pueden utilizar dentro de las familias que tiene Xilinx: Spartan ,Artix y Zynq , así como podemos ver en la Figura 5.

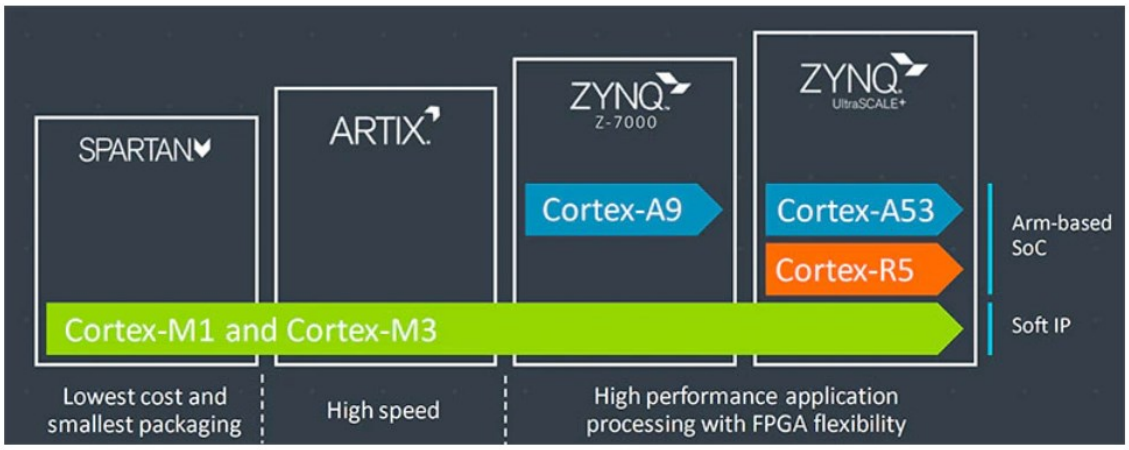


Figura 5. Familia de tarjetas para trabajar con los microprocesadores Cortex-M [5].

La integración de los soft core con los SoC Zynq, que tienen procesadores de hard core como ***Cortex-A*** y ***Cortex-R,*** tiene sentido ya que ayudan a optimizar el funcionamiento del hardware de estos SoCs. Al estar pensado para realizar tareas de microcontroladores, estas tareas pueden necesitar una sincronización determinista o ejecutar tareas críticas con sistemas operativos de tiempo real (RTOS), que resultarían excesivas si son ejecutadas en procesadores de alta gama, como los nombrados anteriormente. [5]

3.1.1. Spartan 7

Dentro de la familia Spartan, nos centraremos en la línea Spartan 7. Los dispositivos de esta nueva versión de la familia Spartan ofrecen el mejor rendimiento por vatio de su clase. Cuentan con un soft processor *MicroBlaze* que es capaz de ejecutar más de 200 DMIPS con un soporte de memoria DDR3 de 800Mb/s, construido en una tecnología de 28 nm. Además, ofrecen también un ADC (Analog Digital Converter) integrado y funciones de seguridad. [6]

ARM y Xilinx proponen utilizar las placas Arty S7 que incluyen la Spartan 7, tenemos las variedades [7]:

- *Arty S7*: Spartan-7 25/Spartan-7 50.

3.1.2. Zynq 7000 SoC

Esta familia integra la programabilidad software de un procesador de ARM con la capacidad de programación hardware de una FPGA, permite el análisis clave y aceleración hardware mientras integra CPU, DSP, ASSP, y funcionalidad de señal mixta en un solo dispositivo. Mejora los niveles de potencia y rendimiento superando el de procesadores discretos y sistemas FPGA.

La familia tiene disponibles dispositivos, los Zynq-7000S de un solo núcleo y el Zynq-7000 de doble núcleo. La familia Zynq 7000 es la mejor en cuanto a rendimiento por watio, es una plataforma SoC totalmente escalable. [8]

Como se había dicho antes ARM y Xilinx proponen utilizar estos kits de desarrollo [7].

- Arty Z7: Zynq-7000 Z7-10/Zynq-7000 Z7-20.
- MiniZed: Zynq-700 7S.

3.1.3. Zynq UltraScale

Estos dispositivos proporcionan procesadores de 64 bit escalables al mismo tiempo que combinan control en tiempo real, con motores para procesamiento gráfico, vídeo, formas de onda, etc. Esta construido sobre un procesador común en tiempo real y una plataforma equipada con una lógica programable. Tiene tres variantes que incluyen un procesador de aplicación dual (CG), de aplicación cuádruple y dispositivos GPU (EG). Crea posibilidades para aplicaciones en 5G, ADAS de próxima generación e IoT de aplicación industrial. [9]
También se recomenda este kit de desarrollo [7]:

- Ultra 96: Zynq UltraScale+ ZU3EG.

3.1.4. Artix-7

Esta familia está optimizada para aplicaciones de baja potencia, requiere de transceptores serie, alto rendimiento DSP y rendimiento lógico. Utiliza el soft processor MicroBlaze y tiene soporte para memoria DDR3 de 1.066Mb/s. La familia Artix-7 está muy bien valorada para aplicaciones que son sensibles al coste y la potencia consumida. [10] [11]

Las placas Arty A7(Artix-7 35T o Artix-7 100T), son las que se han utilizado en proyectos con los procesadores Cortex-M1 y Cortex-M3. En este proyecto, sin embargo, se utilizará la placa Nexys 4 DDR.

Como se utilizará una FPGA de esta familia de Xilinx, se explicará brevemente una descripción general de la misma.

La FPGA al estar compuesta por células lógicas e interruptores programables; las células lógicas son unos elementos muy importantes en los dispositivos de la Artix-7, estas células contienen un LUT (*lookup table*), que pueden ser configurados cada uno como un LUT de 6 entradas o como un 5 LUTs de 2 entradas. Estas células lógicas pueden implementar funciones aritméticas y también circuitos de multiplexación, los cuales son usados para crear multiplexores. Los LUTs también son usados para recursos de memoria como se explicará más adelante. [11]

Los dispositivos de esta familia también cuentan con algunos tipos de macro células. Los controladores de reloj MMCM (Mixed-Mode Clock Manager) son macro células que proporciona una gran cantidad de frecuencias a partir de una única entrada del oscilador, permitiendo ajustar el cambio de fase o reducir el skew de las señales de

reloj. Los bloques BRAM (Block Random Access Memory), también son macro células de 36 Kb de doble puerto síncrono, más adelante se explicarán. Los DSP (Digital Signal Processing) son unas macro células formadas por un multiplicador de 25x28 y un acumulador de 48 bit. El bloque de entrada y salida (IOB), estas macro células están asociadas a la interfaz física de la FPGA. Entre otras macro células, el XADC (Xilins Analog-To-Digital Converter), contiene dos convertidores de 12 bits; o los bloques para la interfaz de gigabit ethernet y PCI. [11]

En la Figura 6 se comparan los diferentes dispositivos de la familia Artix-7 indiciando sus recursos más significativos.

Device	Num. of LCs	Num. of 36-Kb BRAMs	BRAM bits	Num. of DSP slices	Num. of MMCMs
XC7A15T	16,640	25	900K	45	5
XC7A35T	33,280	50	1,800K	90	5
XC7A50T	52,160	75	2,700K	120	5
XC7A75T	75,520	105	3,780K	180	6
XC7A100T	101,440	135	4,860K	240	6
XC7A200T	215,360	365	13,140K	740	10

Figura 6. Diferentes dispositivos de la familia Artix-7. [11]

3.1.4.1. Nexys 4 DDR

La tarjeta Nexys 4 DDR de Digilent está diseñada con el dispositivo Artix-7 100T y tiene una gran variedad de periféricos integrados. Es una FPGA grande cuyo número de pieza es ***XC7A100T-1CSG324C***.

Los principales periféricos y componentes son los siguientes:

1. Conector de alimentación para fuente de alimentación externa (opcional)
2. Puerto compartido USB JTAG y UART.
3. Artix XC7A100T.
4. Puerto Pmod (JD)
5. Puerto Pmod (JC)
6. Dieciséis Leds
7. Dieciséis switches
8. Sensor de temperatura
9. Display LED de siete segmentos.
10. Puerto Pmod (JB)
11. Cinco interruptores de botón.
12. Puerto Pmod (JA)
13. Botón de reset para el soft core processor
14. Puerto Pmod y entrada analógica (conectado al XADC)
15. Entrada Jack de audio
16. Puerto VGA
17. Conector Ethernet
18. Puerto host USB (conectado al USB ratón/teclado)

19. Interruptor de encendido.

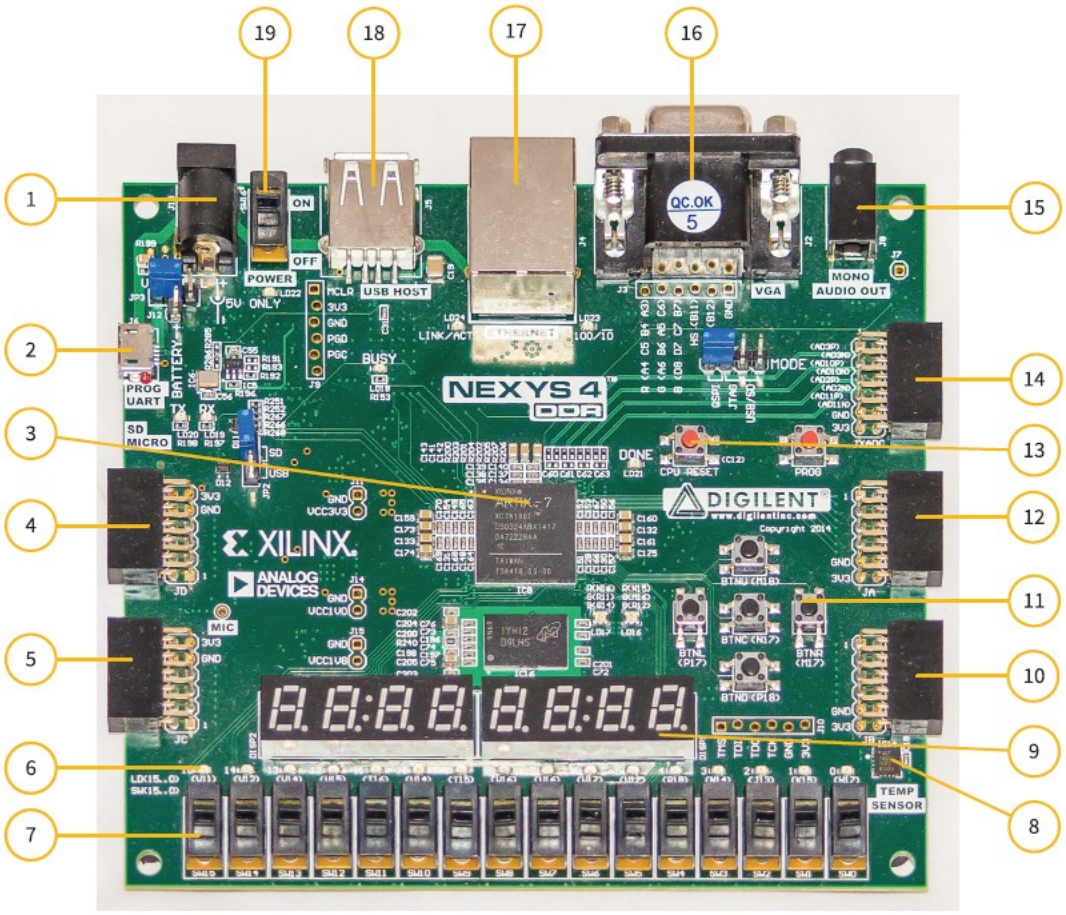


Figura 7. Placa Nexys 4 DDR. [11]

Especificaciones claves	
Células lógicas	15.850 (slices), cada una con 6 LUTs de entrada y 8 flip-flops.
Bloques RAM	4.860 Kbits
Controlador de reloj	6, cada uno con PLL
Bloques DSP	240
Reloj interno	450MHz+
DDR2	128MiB
Células RAM	16 MB
Ethernet	10/100 PHY

Tabla 2. Tabla de de especificación de la Nexys 4 DDR.

Uno de las características más importante de la Nexys 4 DDR que nos interesa conocer es su memoria, por este motivo se va a explicar con más detalle.

En una FPGA sus elementos lógicos contienen un registro y debido a esto pueden proporcionar memoria de un bit. Sin embargo, se requieren muchos transistores para implementar dichos elementos lógicos y en cambio sólo son necesarios entre 5

o 6 transistores para construir una célula SRAM (Static RAM) de un bit. Para hacer un uso de recursos más eficientes, las FPGA modernas se construyen con módulos de almacenamiento prefabricados.

En las FPGAs de Xilinx hay dos tipos de memoria embebida: memoria RAM distribuida y bloques de memoria RAM (BRAM). El primer tipo de memoria está hecha de células lógicas (LUT). Para entender mejor la idea de este tipo de células, por ejemplo, 6 LUTs de entrada pueden ser configurados como un $2^6 x1$ módulo RAM, y múltiples LUTs pueden ser conectados en cascada para obtener un módulo mucho más grande. [11]

En la Nexys 4 DDR se pueden proporcionar más de 1.188 Kbits de la memoria distribuida, que si es comparada con la memoria de bloques BRAM es pequeña (4.860 Kbits) o con una memoria externa. También, la lectura de datos de este tipo de memoria no es registrado (es asíncrona). Por estos motivos, esta memoria distribuida no es muy utilizada en las FPGAs y por ser poco portable. [11]

La memoria de bloques RAM (BRAM) es un módulo de memoria prefabricada dentro de la FPGA y se encuentra separada de las células lógicas. En un dispositivo Artix, cada BRAM consiste de 32 $Kbits$ (2^{15}) de datos, más 4 Kbits de paridad adicionales, en total 36 Kbits. En el caso concreto del Artix-7 XCA100T tiene 135x36 Kbits bloques BRAM, en total se pueden utilizar 4.860 Kbits para datos.

Este tipo de memoria se puede encontrar en todas las FPGAs modernas. La interfaz que usa es síncrona, y gracias a esto no es necesario un controlador de memoria adicional. Las ventajas que tiene es que son bloques flexibles y se pueden configurar para realizar un único o doble acceso a los puertos. [11]

La Nexys 4 DDR tiene una memoria flash (Spansion S25FL125S). Esta memoria externa se comunica con el terminal host mediante una interfaz SPI. Según las especificaciones de la tarjeta, casi 4 MB son utilizados para el fichero de configuración del Artix-7, dicho fichero es cargado durante el encendido de la FPGA. Dejando el resto de la memoria para almacenar datos del usuario.

Memoria	
Flip-Flops (para registros)	13 KB, en células lógicas y buffers I/O
RAM distribuida	150 KB, construida mediante células lógicas.
BRAM	540 KB, configurado como módulos de 270 32Kb
Externa SDRAM	128 MB, configurado como un único rango 8M-por 16 DDR SDRAM
Flash	16 MB

Tabla 3. Memorias disponibles en la Nexys 4 DDR.

4.1. Interfaz de comunicación: Buses on-chip

El microprocesador Cortex-M1, utiliza la arquitectura de bus de Von Neumann. Esto; lo que significa que se tiene un único bus para el acceso a memoria, para este microprocesador se proporcionan interfaces para la memoria TCM de forma que simplifica la integración de la memoria en la FPGA y ofrece una frecuencia de reloj más alta. [12]

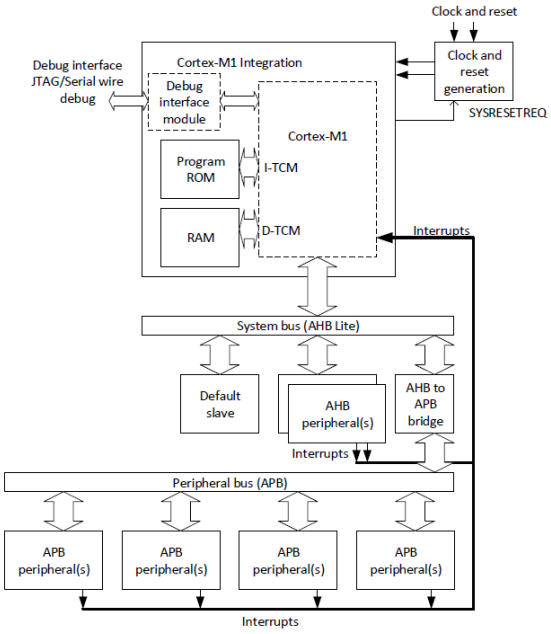


Figura 8.Diagrama de bloques básico del Cortex-M1. [10]

Para que los sistemas embebidos alcancen una comunicación eficiente entre el microprocesador y los periféricos, interesa que la infraestructura de comunicación, en este caso el bus ce comunicación asegure que la información se intercambia de forma correcta. Además, es interesante que las IPs utilizadas puedan ser reutilizables, por lo que interesa que tengan un protocolo estándar para poder integrarlos dentro del sistema.

Para los microprocesadores de ARM, el protocolo más extendido es *AMBA* (Advanced Microcontroller Bus Architecure). Es un protocolo muy utilizado ya que está disponible sin necesidad de pagar licencia, tiene un amplio soporte y muchas empresas lo utilizan para sus IPs. Este protocolo está pensado para una comunicación ***on-chip***.

Las características más importantes que tienes son:

- Realiza operaciones síncronas.
- Su estructura evita a los buffer tristate por lo que; no dispone de señales bidireccionales.

Dentro del amplio abanico de protocolos que tiene AMBA, los más importantes son: AXI3,AXI4,AXI4-Lite, AXI4-Stream y AHB. Vamos a ver una breve descripción de forma general de los más importantes:

4.1.1. APB (Advanced Peripheral Bus)

El APB es un bus simple que se utiliza principalmente para conexiones con periféricos. La mayoría de los sistemas que lo utilizan son de 32 bits. El ancho de banda del bus no está limitado, pero normalmente se usa para periféricos de 32 bits. [12]

Las ventajas que tiene de utilizarlo son:

- Cuando los sistemas tiene muchos periféricos, si se conectan por ejemplo con un bus tipo AHB , su frecuencia máxima tiende a reducirse por motivos como la lógica de decodificar las direcciones. Por este motivo, resulta más útil emplear este tipo de bus.
- Si los periféricos funcionan a una frecuencia de reloj distinta.
- Como su interfaz es muy básica, el diseño de los periféricos es más simple.

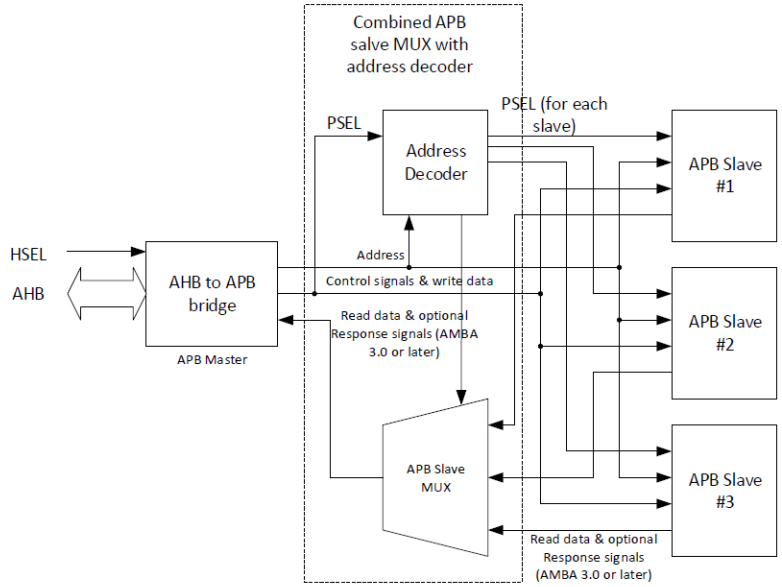


Figura 9.Ejemplo de un subsistema APB [10].

En la mayoría de los casos el sistema APB tiene un bus *bridge* como el bus maestro que conecta con el bus principal del procesador. Además, el APB tiene un multiplexor de esclavos y un decodificador de direcciones, en algunos casos esto puede ser una unidad combinada.

Es frecuente que para los APB esclavos las muestras de datos durante la escritura se realicen en el último ciclo de la transferencia, como es el caso de la versión AMBA 2. También se puede dar el caso de muestrear el dato de escritura en el primer ciclo de la transferencia porque el master APB debe proporcionar un dato de escritura valido a los esclavos incluso en su primer ciclo. Para la transferencia de lectura, el master APB sólo tiene que leer el valor del dato de retorno de lectura en su último ciclo o cuando la señal PREADY (Anexo 1) es un nivel alto. En el caso de que reciba un dato erróneo, deberá ser descartado por el bus master.

4.1.2. AHB (Advanced High-performance Bus)

Este protocolo multi master y multi esclavo, que fue diseñado por primera vez en las especificaciones del AMBA 2. Está diseñado para trabajar con microprocesadores embedded con baja latencia. Los sistemas que lo emplean, normalmente son de 32 bit, aunque también hay casos de 64 bit, puede trabajar con diferentes tamaños.

Para sistemas de alto rendimiento, es el caso de los módulos como: el microprocesador o memorias on-chip, que utilizan frecuencias de reloj altas, se asegura el ancho de banda. Se emplea un puente para conectarse con buses de bajo ancho de banda como el APB, se puede ver en la Figura 8, donde están conectados los periféricos.

La versión AMBA 3, o mejor conocida como AHB Lite. Elimina del bus las señales de solicitud y de admitido. Este es el protocolo que utiliza el Cortex-M1. Sin embargo, la versión de este procesador es una versión adaptada pensada para poder utilizarse con las IPs de Xilinx; por lo tanto, este Cortex-M1 incluye a su vez un puente para pasar al protocolo AXI (AHB to AXI bridge).

Este puente lo que permite es que se pueda establecer una comunicación entre esclavos AXI y masters AHB o interconectar si la señal de reloj y reset es común. [13]

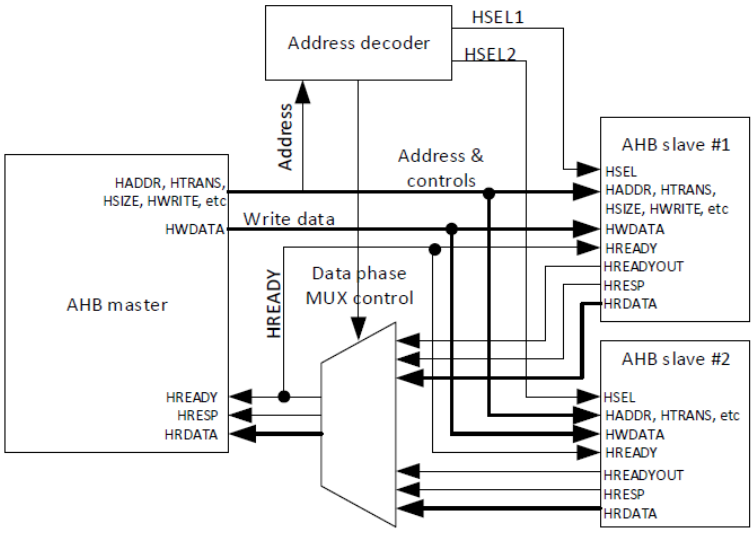


Figura 10. Un sistema sencillo AHB con un master y dos esclavos [12].

Además de los esclavos y el master, un sistema AHB contiene un multiplexor de control y de direcciones, un decodificador y en el caso de que se tengan más de un master un árbitro. En ese caso, si un maestro necesitará tomar el control del bus, enviaría primero una solicitud al árbitro. Entonces, para aceptar la solicitud de acceso se dará primero al que tenga la prioridad más alta. Un número de señales de control se utilizan para poder definir la dirección, el ancho y el tipo de datos de transferencia. Estos elementos que hemos explicado se pueden apreciar en la Figura 10.

Por otra parte, el decodificador recibe las señales de dirección del maestro y lo decodifica en las señales seleccionadas del esclavo. El esclavo responde al maestro mediante la señal *HRESP* (Anexo 2) y empieza a transferirse el dato entre el esclavo y el maestro. [12]

4.1.3. AXI (Advanced eXtensible Interface)

ARM lo introdujo en la versión AMBA 3, en concreto se le dio el nombre de AXI3. Este protocolo proporciona una solución muy eficiente para comunicaciones de sistemas y periféricos de alta frecuencia.

El protocolo AXI es el más extendido, permitiendo una comunicación entre SoC implementados en FPGAs o ASICs. Esta versión del protocolo es de muy alto rendimiento y proporciona soluciones en aquellos casos la transferencia de datos se accede a una determinada dirección dentro de un espacio de memoria.

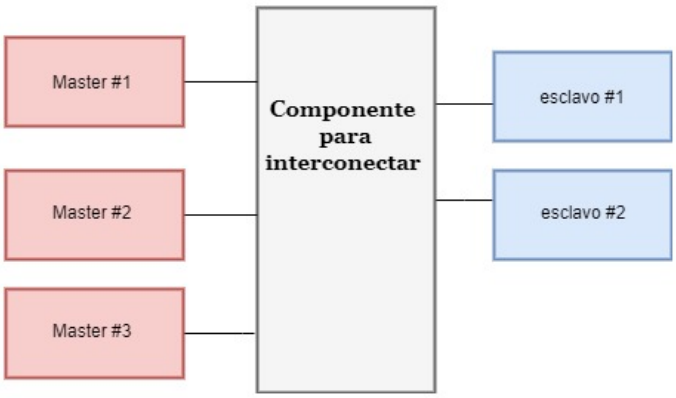


Figura 11. AXI interconnect

Como podemos ver en la Figura 11, tanto AHB como AXI comparten similitudes, ambos utilizan una configuración master-esclavo, conectados mediante un componente para interconectar, donde se transfieren los datos. Sin embargo, la diferencia que tienen es AXI utiliza una arquitectura de canal punto a punto, por lo que se utiliza canales independientes para las direcciones, señales de control, datos de lectura y escritura. De esta forma, se permiten transferencias bidireccionales entre el master y el esclavo, con ayuda de señales de transacción.

Cuando se trabaja con un sistema que tiene múltiples master y múltiples esclavos, AXI proporciona más flexibilidad que en el caso de AHB. Por ejemplo, si el sistema tiene múltiples maestros que intentar comunicarse con un solo esclavo, el AXI tiene un árbitro que encamina el datos entre las interfaces del master y el esclavo. Esta arbitración se puede implementar usando prioridades, una arquitectura round-robin o la que se ajuste al diseñador. También permite que un conjunto de masters o esclavos se agrupen juntos, de forma que son visto como uno solo. [1]

Otras características interesantes del protocolo AXI:

Una pareja de master-esclavo pueden operar a diferentes frecuencias.

Si un master inicia una transacción con un periférico que es lento y luego con uno más rápido, no es necesario esperar que la primera transacción se complete para poder atender a las siguientes.

Permite estepas pipeline. Los canales son independientes de cada uno para enviar información en una sola dirección.

Para la comunicación entre el master y el esclavo se hace mediante cinco canales diferentes:

- Read address
- Read data
- Write address
- Write data
- Write response

4.1.3.1. Transacción READ (Lectura)

La transacción de lectura, el master inicia con poniendo la dirección del esclavo dentro de la señal *ARADDR* y confirma que el la dirección es válida en *ARVALID*. A continuación, el esclavo confirma con la señal ARREADY, pero esto lo envía cunado la información la tiene disponible. Por otra parte, cuando se recibe los datos para confirmar que se reciben lo hace mediante la señal de ready. Para que una transferencia ocurra las señales ARREADY y ARVALID deben de estar activas. Estas transacciones han tenido lugar en el canal de *read address.*

Em el canal *read data*, se desarrolla el resto de la transacción. Cuando el master está preparado para los datos confirma con la señal *RREDAY*. El esclavo por su parte pone los datos en la señal *RDATA* y los confirma con *RVALID*. El origen es el esclavo y el destino el master.

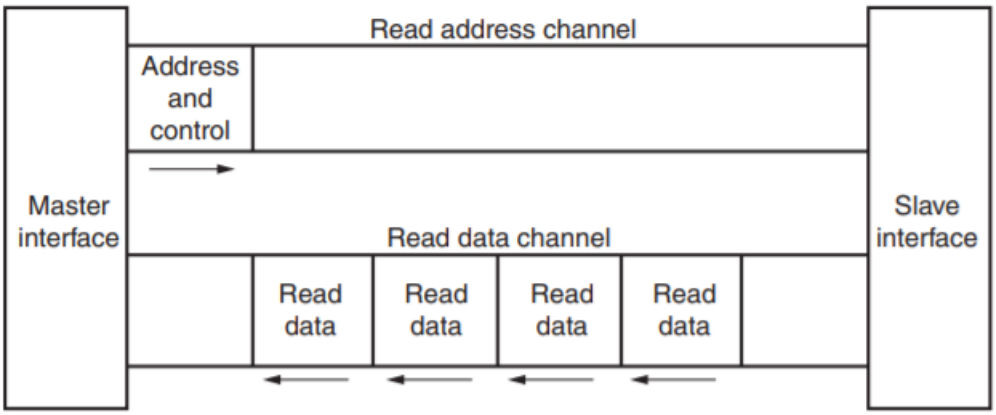


Figura 12. Transferencias de lectura. [14]

4.1.3.2. Transacción Write (Escritura)

En esta transacción intervienen los tres canales restantes. La etapa de direccionamiento es igual a la de lectura. La primera parte de esta transacción ocurre en el canal *write Address.* El maestro por su parte pone la señal del esclavo en la señal *AWADDR* y la valida con la señal *AWVALID*. Por otra parte el esclavo confirma que está preparado para recibir la dirección y se hace esa primera transferencia.

Después en el canal *write data*, el maestro pone el dato (*WDATA*) en el bus y lo valida con WVALID. Cuando el esclavo está preparado, se confirma con la señal *WREADY* y empieza la transferencia. Cuando el master confirma se ha recibido el último dato de la transferencia se activa WLAST.

La última parte, transferencia sucede en el canal *Write Response*. El esclavo puede confirmar que la transacción de escritura se ha completado con éxito (Anexo 6).

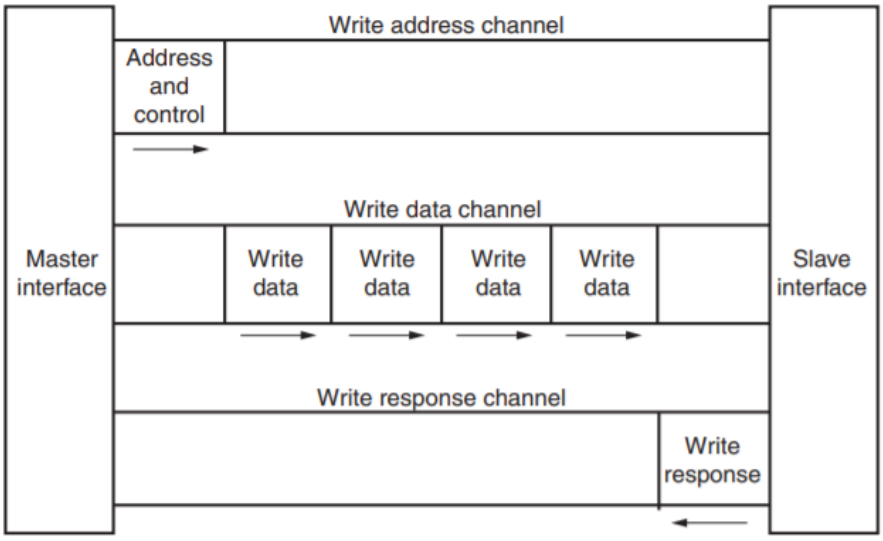


Figura 13. Transferencias de escritura [14]

5.1. Memoria

El tema de este trabajo consiste en implementar núcleos ARM dentro de una FPGA. Para ello será necesario estudiar las características para manejar la memoria por parte de los núcleos, de forma que se pueda almacenar su memoria de datos y de programa. Por otra parte, se analizarán diferentes tipos de memorias físicas que se emplearan en este trabajo.

5.1.1. Mapa de memoria

Los procesadores Cortex-M tienen definido un mapa de memoria que tiene asignado regiones en rangos de memoria. Conocer el mapa de memoria del procesador, permite acceder a los periféricos, memorias externas, la región disponible a la depuración, sean accesibles mediante un programa en C. El mapa de memoria permite hacer más optimo el uso de los recursos que tiene disponible.

El diagrama que se ve en la Figura 14, se tienen seis zonas de gran importancia. Para empezar:

La primera región desde el rango de direcciones 0x00000000 a 0x1FFFFFFF, se usa principalmente para la memoria del programa ***CODE***, también tiene una tabla de vectores de excepción después del encendido del procesador.

La siguiente región que va desde la dirección 0x20000000 a la dirección 0x3FFFFFFF se utiliza como una **RAM estática**; este tipo de memoria RAM puede ser utilizada para almacenar programas, incluyendo el *stack*, o para almacenar datos de lectura o escritura.

La región comprendida entre 0x40000000 hasta 0x5FFFFFFF se utiliza para los **periféricos**. Es usada fundamentalmente para periféricos, y también para almacenamiento de datos. Los periféricos conectados pueden disponer del protocolo AHB-Lite o del APB, como se vio en la Figura 8. [15]

Siguiendo con la próxima región entre la dirección 0x60000000 hasta 0x9FFFFFFF se utiliza como región de **memoria externa**. Se utiliza para almacenar datos, en algunos casos puede ser utilizada también como un espacio de memoria continuo de 1GB.

La región para **periféricos externos** está comprendida dentro del rango de direcciones desde 0xA0000000 hasta la 0xDFFFFFFF. Se utiliza principalmente para periféricos o puertos I/O. Esta región no permite la ejecución de programas, pero se puede utilizar para el almacenamiento de datos generales.

La última región destinada que incluye **periféricos privados**, el controlador de interrupciones vectorizadas, el espacio de control del sistema entre otros va desde la dirección 0xE0000000 hasta la 0xFFFFFFFF. Dentro de este rango se encuentra el espacio de control del sistema SCS (System Control Space), contiene los registros de control de interrupciones, registros de control del sistema, registros de control de depuración, entre otros. El SCS incluye también el SysTick. [15]

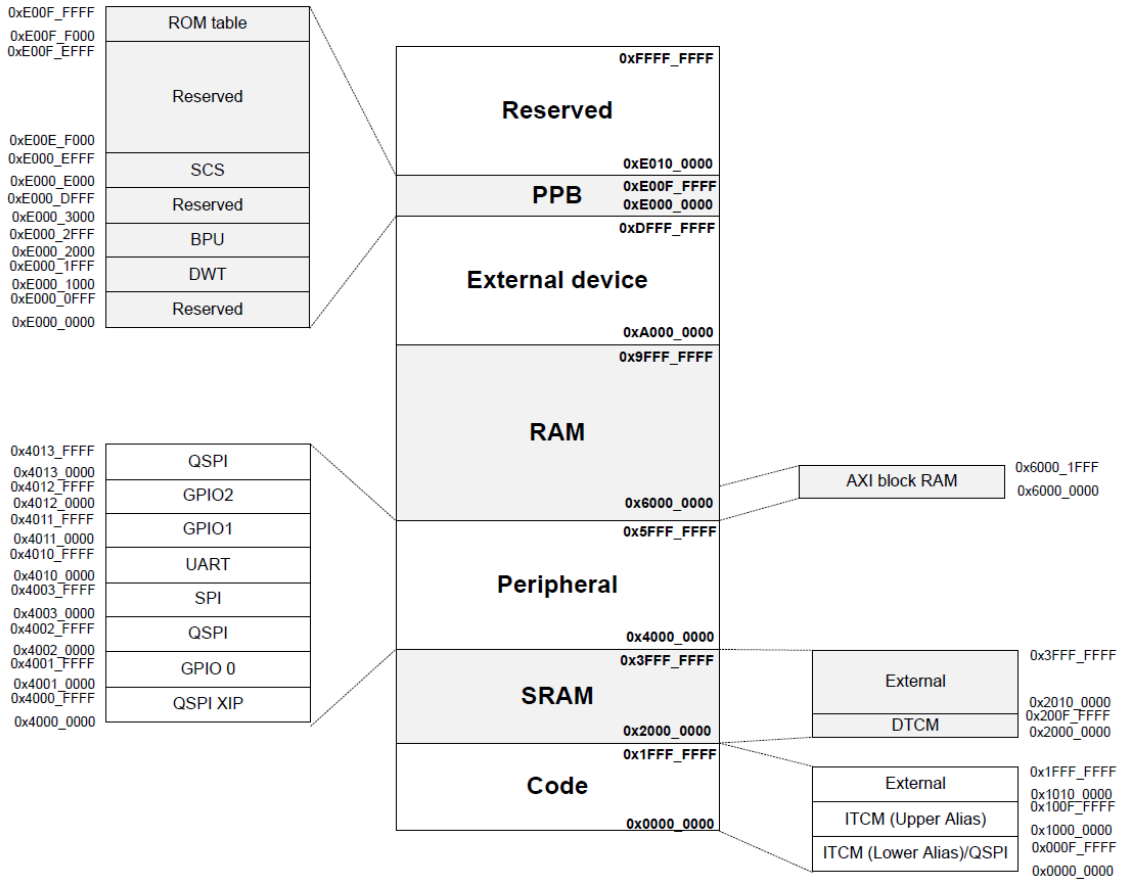


Figura 14. El mapa de memoria del Cortex-M1 [4]

El procesador Cortex-M1 tiene la particularidad de que las instrucciones TCM y los datos TCM están ya fijados en una dirección de memoria, ambos pueden ser de tamaños configurables [12].

5.1.2. TCM

El procesador Cortex-M1 de ARM ha sido desarrollado para implementarse en FPGAs. Por lo que proporciona interfaces que habilitan la conexión de la memoria al procesador, la interfaz TCM (Tightly Coupled Memory). Esta interfaz ha sido diseñada para proporcionar una baja latencia en memoria y que permita ser usada por el procesador sin la imprevisibilidad que genera la memoria cache. Este tipo de interfaz se utiliza para almacenar rutinas criticas (rutinas de atención de interrupciones, tareas de tiempo real). Incluso también puede ser utilizada para almacenar datos, pila de datos de interrupciones o datos que no se pueden almacenar en cache. [12] [16]

Para este procesador se disponen de dos interfaces TCM: la interfaz ITCM que está destinada principalmente a las instrucciones de memoria (accesos a datos), y la interfaz DTCM que es ante todo para transferencia de datos. Entenderemos la interfaz ITCM como ***code memory*** o memoria de programa, y la interfaz DTCM como ***data memory*** o memoria de datos.

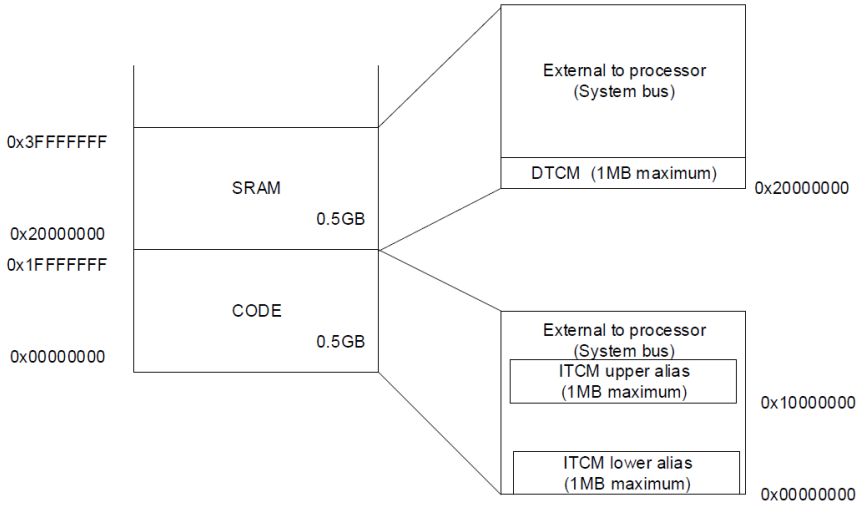


Figura 15. Mapa de memoria, zona TCM del Cortex - M1 [12].

El máximo tamaño que pueden tener estas interfaces es de hasta 1 MB. Además, estas interfaces están diseñadas para utilizar bloques internos de la (***BRAM***).

Los procesadores Cortex-M tienen por lo menos dos tipos de memoria. Una memoria **no volátil**, como por ejemplo la memoria flash utilizada principalmente para almacenar el programa (***code memory***). Por otro lado está la memoria RAM, para leer o escribir datos (***data memory***)Figura 15.

Los datos que se almacenan en la memoria del programa representan datos constantes o valores de inicialización para tiempos de ejecución que se almacenarán por ejemplo en la memoria SRAM. En cuanto a la memoria de datos, es principalmente una memoria de lectura/escritura, que se utiliza para diferentes tipos de datos como locales, globales, memoria estática o almacenamiento de memoria dinámica. [12]

Como se había explicado antes la interfaz de memoria que utiliza el Cortex-M1, se puede configurar hasta un tamaño de 1 MB. Esto en algunos casos se puede quedar pequeño en determinadas situaciones. Debido a esto se tienen memorias Flash o memorias RAM que pueden almacenar la información que se necesite.

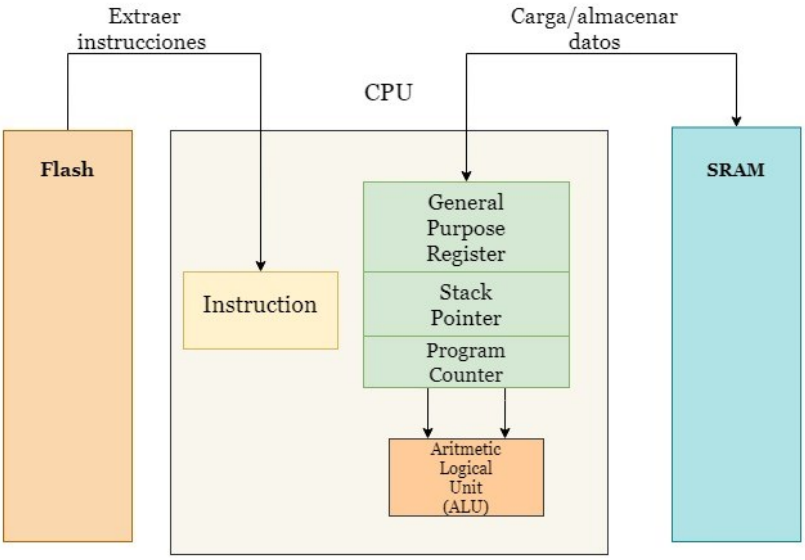


Figura 16. Diagrama de bloques de un microcontrolador

En la figura 16 se puede comprender la idea a la de que se puede transferir información desde los dos puntos entre estos dos tipos de memorias para tenerla información en el procesador.

5.1.3. Memoria Flash

La memoria Flash, es una memoria no-volátil, un tipo de memoria sólo de lectura programable y borrable, una EEPROM (Electrically Programmable Read-Only Memory), quiere decir que no pierde información cuando el dispositivo se apaga. Las memorias flash se utilizan porque mejoran su eficiencia por área utilizada y un menor consumo de energía, ya que no necesitan una memoria de configuración externa. Este tipo de memorias ofrecen una serie de ventajas, siendo la más importante su no volatilidad. Esta característica permite eliminar la necesidad de pedir recursos externos para almacenar o cargar datos de configuración cuando la SRAM, por ejemplo, está siendo usada. Igualmente, la memoria flash puede funcionar inmediatamente después del encendido en lugar de esperar a que se carguen los datos de configuración. El tiempo de inicio es más rápido, el patrón de configuración de la flash se carga casi instantáneamente, mientras que la SRAM todavía estará cargando la configuración. Esto representa una ventaja cuando su aplicación necesita que se inicie o reinicie lo más rápido posible. [17]

Hay dos tipos de memoria flash, que se clasifican dependiendo del método de escritura. Memoria flash ***NOR*** y la ***NAND***. Una de las características que representa a cada tipo es, por ejemplo, que para el tipo NAND se requiere una alta tensión y en cambio para el tipo NOR lo que se requiere es una alta corriente, en la puerta flotante que se ve en la Figura 17. [18]

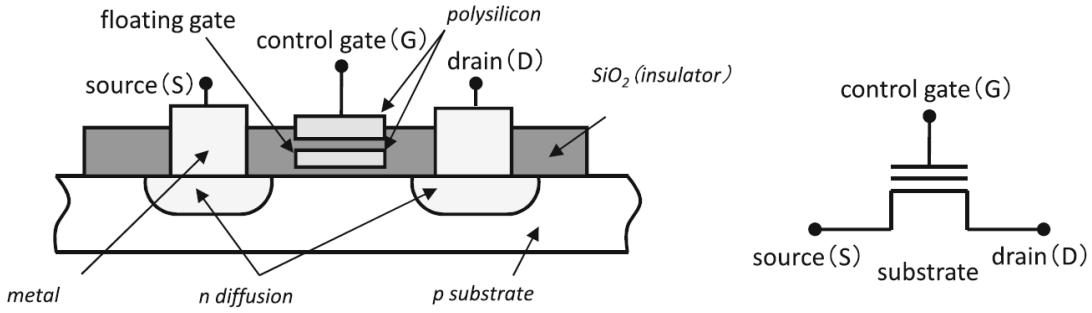


Figura 17. Estructura de una memoria Flash [18]

5.1.3.1. Memoria Flash Spansion S25FL128S

La memoria flash S25FL128S, es la memoria no volátil que viene incorporada en la Nexys 4 DDR. Es una memoria fabricada por Cypress, tiene un tamaño de 16 MB, una velocidad de x4 y se alimenta con una tensión de 3.3V. Como se había explicado anteriormente, esta memoria externa para comunicarse con el terminal host utiliza la interfaz de comunicación SPI. [19]

Una transacción SPI flash básica consiste en el envío de un comando de 8 bits, seguido de un argumento especifico de comando (depende del comando), por lo que el esclavo deberá devolver algún tipo de respuesta. Durante el proceso anteriormente explicado, tanto las líneas MISO y MOSI que contienen datos válidos van en direcciones opuestas al mismo tiempo [20]. O el caso de que el controlador flash esté enviando datos al chip flash o los datos van en la dirección inversa. [20]

Comprobando el datasheet del fabricante, se puede ver que todas las transacciones están hechas por secuencias de 8 bits. Es algo muy común en los chips de las flash basadas en el protocolo SPI.

También se produce la particularidad que las señales del protocolo SPI como MOSI y MISO sean rediseñadas en señales bidireccionales. El maestro puede enviar en ambos bits durante el periodo de argumentos del comando, y entonces podrá recibir en ambos bits durante el periodo de respuesta. A esta forma de trabajar se le conoce como *Dual SPI* (x2) o DSPI. Muchos integrados flash, incluyendo el que posee la Nexys 4 DDR, pueden incluso tener otros dos puertos, un pin de reinicio y otro pin de protección de escritura lógica negativa. Estos también pueden ser reutilizados en otras señales bidireccionales de datos, creando de esta forma un bus de datos bidireccionales de 4 hilos para una comunicación incluso más rápida. A esta forma de trabajar se le llama Quad SPI (x4) o QSPI. Es este el funcionamiento que utiliza la flash integrada en la placa que utilizamos. [20]

Nombre del pin convencional	**Nombre del pin usado en esta memoria**	**Nombre del pin en la FPGA**	**Función**
SCK	CLK	E9	Reloj para las instrucciones flash SPI y datos
$\overline{\boldsymbol{SS}}$	CS	L13	Chip Select
MOSI	DQ0	K17	Maestro de salida, esclavo de entrada. Puede ser también un pin adicional de datos de salida x2 o x4
MISO	DQ1	K18	Maestro entrada, esclavo de salida.
$\overline{\boldsymbol{W}}$	DQ2	L14	Protege porciones de escritura de la

			memoria Flash SPI. Puede ser también un pin adicional de datos de salida x4.
$\overline{\boldsymbol{HOLD}}$	DQ3	M14	Mantiene o pausa sin necesidad de deseleccionar el dispositivo. Puede ser también un pin adicional de datos de salida x4.

Tabla 4. Nombre de los pines de la Flash SPI. [21]

El funcionamiento a nivel general de esta memoria flash se describe a continuación. Después de encender la FPGA y una vez se completa la auto inicialización, que es un tiempo power-on reset [22], se transmite la orden a la memoria QSPI flash para recuperar los datos de configuración.

Para seleccionar la SPI flash, uno de los parámetros importantes a tener en cuenta es la densidad, para que la memoria sea capaz de almacenar los datos según las necesidades del diseño, por ejemplo cuando se tienen múltiples bitstream. La densidad mínima será acorde al tamaño del bitstream de configuración de la FPGA, en el caso de la Nexys 4DDR, es de 32 Mb, Tabla 5. Se permite comprimir el bitstream, aunque no es muy recomendable; ya que a pesar de que el tamaño de la SPI flash es determinado, la comprensión puede variar, puesto que el diseño del usuario tiene un tamaño aún sin determinar. [21]

Otro parámetro importante a tener en cuenta es especificar o configurar la frecuencia con la que se trabajará. Dependiendo del grado de velocidad con el que se piense trabajar y las limitaciones de la memoria externa, es importante conocer la frecuencia. Ya que determinará como de rápido se hará la lectura.

Si buscamos como es el esquema interno de la memoria SPI flash en la Nexys 4 DDR, nos encontramos con el esquema de la Figura 18. Antes se había explicado de forma básica los puertos que intervienen, pero como se ve de forma más amplia en este caso, vemos que intervienen más.

Dispositivo	**Longitud del bistream de configuración (bits)**	**Tamaño mínimo de la memoria flash de configuración (Mb)**	**JTAG/Dispositivo IDCODE[31:0] (hex)**	**JTAG Longitud instrucciones (bits)**
7A100T	30.606.304	32	X3631093	6

Tabla 5. Longitud Bitstream. [23]

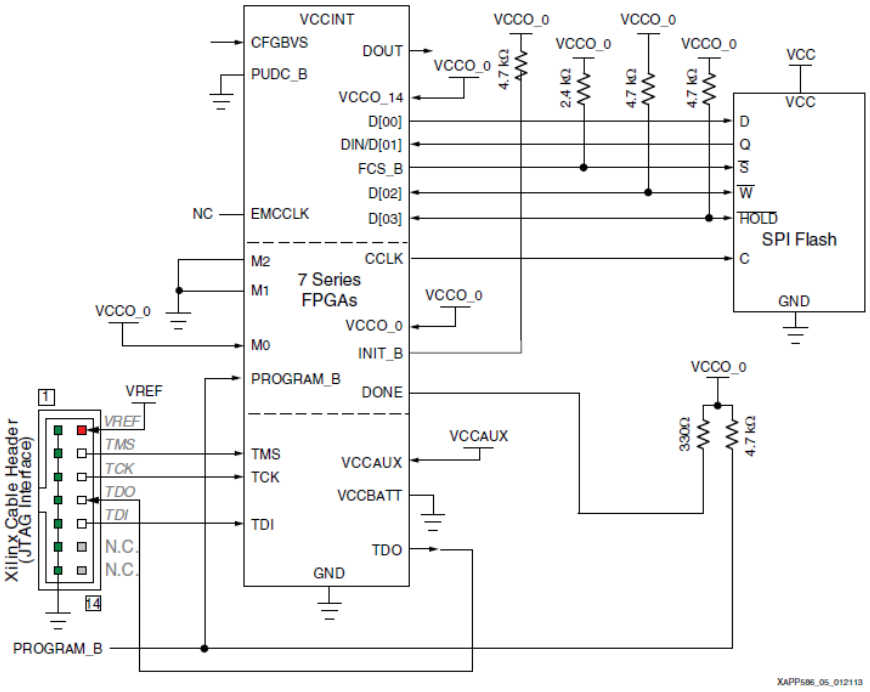


Figura 18. Esquema de la configuración modo SPIX4

Después de realizar la auto inicialización, una vez activa la señal INIT y la FPGA muestrea los pines de M[2:0] para determinar qué modo de configuración utilizar. Con el modo M[2:0]=001, la FPGA empieza a trabajar con las señal de reloj a una frecuencia aproximada de 3 MHz. Justo después, el pin FCS_B se mantiene a nivel bajo, seguido de OPCODE que realiza una instrucción rápida de lectura x1 y la dirección en el pi D[00].

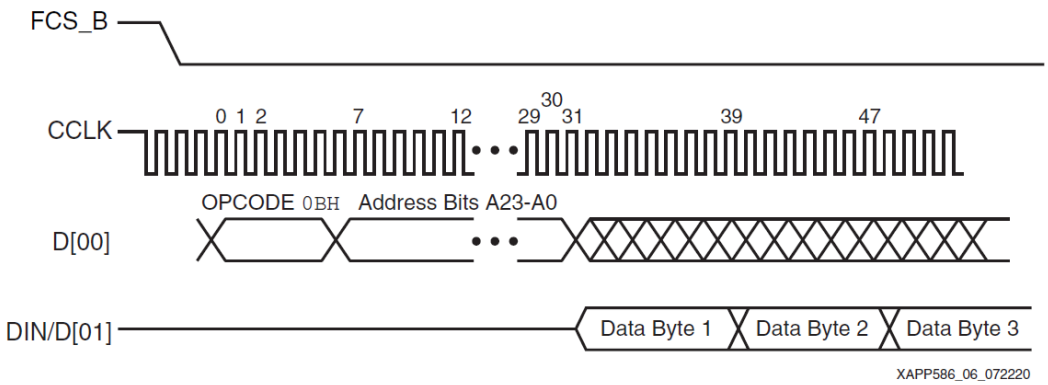


Figura 19. Funcionamiento de la memoria Flash. [21]

Como se ha dicho anteriormente, el modo en el que empieza a trabajar la memoria es en el modo básico x1. Pero después, se leen las nuevas opciones x2 o x4, reloj externo. La FPGA realiza una reajuste a la configuración inicial para tener estas nuevas opciones disponibles.

El comportamiento por defecto es que el dato es capturado por la FPGA en el flanco de subida de la señal CCLK, y el dato pasa a la salida de la SPI flash en el flanco de

bajada de CCLK. Este comportamiento puede ser modificado de capturar el dato en el flanco de bajada habilitando la opción SPI_FALL_EDGE.

Nombre del puerto	Tipo	Descripción
M[2:0]	Entrada	Determina el modo de configuración de la FPGA. M[2:0]=001 modo maestro de la SPI flash
DIN/D[01]	Entrada	Recibe datos desde el puerto MISO de la memoria.
D[00]	Entrada/Salida	Al inicio de la configuración de la FPGA, este puerto conduce al puerto MOSI y entrega una dirección de lectura y la dirección. Si está en el modo x1, este puerto será solo de salida. En el modo x2 y x4, es bidireccional.
D[02]	Entrada	Recibe el bit 2 de datos, del modo x4.
D[03]	Entrada	Recibe el bit 3 de datos del modo x4.
INIT_B	Bidireccional, entrada, salida y open-drain	A nivel bajo durante el encendido de la FPGA, indica que se está realizando la auto-inicialización. Después de esto también puede ser utilizado para demorar la configuración. Cuando es muestreado, pasa a Open-drain. Durante la carga del bitstream, también actúa como un indicador de error CRC.
PROGRAM_B	Entrada	
PUDC_B	Entrada	Controla las resistencias pull-up de I/O durante la configuración. 0= Resistencias pull-up 1=3-state
EMCCLK	Entrada	Una entrada para alimentar de forma externa el reloj de configuración. Está internamente conectado al puerto CCLK de la FPGA.
CCLK	Entrada/Salida	Fuente de reloj de la configuración inicial de todos los modos de configuración excepto el JTAG.
FCS_B	Salida	Conectado al pin SS de la SPI flash y habilitado durante la configuración.
DOUT	Salida	Utilizada únicamente en el modo x1
DONE	Salida/Open-drain	Activa a nivel alto indica cuando la configuración esta completada 0=FPGA no configurada 1= FPGA configurada

CFGBVS	Entrada	Para la familia Artix-7, este pin determina la tensión estándar soportada en los bancos I/O de configuración.
TDI	Entrada	Puerto de datos de entrada al JTAG
TMS	Entrada	Puerto de seleccionar el modo al JTAG
TCK	Entrada	Puerto de la señal de reloj
TDO	Salida/Open-drain	Puerto de datos de salida.

Tabla 6. Señales de la memoria Flash. [17]

En este caso en concreto, al trabajar con una placa que utiliza como núcleo un Artix-7. En el caso de este proyecto se trabaja con una frecuencia de 100 MHz, y para la señal *ext_spi_clk* se trabajará con una frecuencia de 50 MHz.

Familia	**Grado de velocidad**	**Frecuencia máxima del interfaz AXI4-Lite (MHz)**	**Frecuencia máxima del interfaz AXI4 (MHz)**	**Ext_spi_clk Frecuencia máxima (MHz)**
Artix-7	-1	120	150	60
	-2	140	180	70
	-3	160	200	80

Tabla 7. Frecuencias recomendadas para la memoria Flash. [21]

5.1.4. Memoria DDR

En el caso de que se necesiten más cantidad de almacenamiento de memoria RAM, una buena idea sería utilizar memorias *off-chip* como la DDR (Double-Data-Rate) SDRAM. Este es un tipo de memoria forma parte de las memorias dinámicas RAM (DRAM), que comparadas con las memorias estáticas (SRAM) tienen un rendimiento menor y un consumo mayor de potencia, a cambio son más densas y baratas. La DRAM es una memoria síncrona (SDRAM), lo que significa que este dispositivo depende de una señal de reloj. Las memorias DDR SDRAM es una evolución de las memorias SDRAM, la novedad que tienen es que las transacciones se hacen en ambos flancos de reloj, lo que significa que duplica el ancho de banda. [24]

La evolución de las memorias DDR, pasa desde la SDRAM hasta la DDR4 SDRAM

Memoria	**Datos transferidos por ciclo de reloj**	**Frecuencia de reloj del bus(MHz)**	**Velocidad de datos (MT/s)[1]***	**Velocidad de transferencia (GB/s)**
SDRAM	1	66-133	100-166	0.8-1.3
DDR SDRAM	2	100-200	266-400	2.1-3.2
DDR2 SDRAM	4	200-400	533-800	4.2-6.4
DDR3 SDRAM	8	400-1066	1066-1600	8.5-14.9
DDR4 SDRAM	8	1066-1600	2133-3200	17-21.3

Tabla 8. Memorias DDR SDRAM. [25]

En este proyecto se utilizó una memoria DDR2 que tiene incorporado la Nexys 4 DDR.

5.1.4.1. Memoria DDR2

La memoria DDR2 SDRAM es una interfaz DRAM síncrona de doble velocidad de datos, que transfiere datos tanto en los flancos a nivel bajo y alto de la señal de reloj. La frecuencia de reloj de la DDR2 SDRAM son altas como se puede ver en la Tabla 8, gracias a esto se incrementa la velocidad de datos de la memoria. [24]

Las señales de dirección, reloj y la de comandos son sencillas porque son señales unidireccionales. Las señales de datos y de validación son bidireccionales. El controlador de memoria las maneja durante la operación de escritura y la DDR2 SDRAM las controla durante la operación de lectura.

La DDR2 SDRAM también mejora la integridad de las señales de datos y de validación mediante el ODT, una señal ODT puede habilitar una terminación on-die y tiene la capacidad de programar valores on-die como (50Ω,75Ω,150Ω,etc.). [26]

Esta memoria consigue reducir potencia del sistema durante su operación alrededor de 1.8V, que significa el 72% de una DDR SDRAM de 2.5V. Para algunas

[1] MT/s acrónimo de megatransfers/second (número de ciclos que se usan para transferir información).[ref]

implementaciones, el número de columnas en una fila se reduce, por lo que se reduce la potencia cuando se activa una fila para una lectura o escritura. Otro beneficio de la reducción de potencia es que se reducen las oscilaciones en las tensiones de los niveles lógicos. [26]

Una de las novedades que introduce la DDR2 SDRAM frente a las versiones anteriores es que la latencia es aditiva, lo que significa que al controlador de memoria le aporta flexibilidad para enviar los comandos de Read y Write antes del comando ACTIVATE. También mejora el ancho de banda, los ocho bancos incrementan la flexibilidad del acceso a memoria intercalando diferentes operaciones en los bancos.

5.1.4.1.1. Memoria DDR2 de Micron

La Nexys 4 DDR incorpora una memoria DDR2 del fabricante Micron *MT47H64M16HR-25*. Dentro de las especificaciones que nos proporciona el datasheet, el tamaño de la memoria es de 128 MB. Además, tiene una configuración *64M16* (8 Meg x 16 x 8 Banks); se alimenta con una tensión de 1.8 V, se limita la corriente con una resistencia de 50 Ω. También utiliza la tecnología ODT (on-die terminations) para adaptar las impedancias de la línea de transmisión que se encuentra dentro del chip de la memoria [19].

Un módulo de memoria integrado en una FPGA es una macro célula que está separada de las células lógicas normales que tiene la FPGA. Hay diferentes métodos para incorporar estos módulos al diseño en el que se quiere tener [11]:

1. Una instalación HDL
2. Mediante una bloque de controlador de memoria
3. Por una template interfaz HDL (rtl)

En los dos primeros métodos son específicos de Xilinx. El primero de ellos consiste en copiar el segmento de código para instanciar y manualmente modificar los parámetros correspondientes para obtener la configuración deseada. Esta forma por lo general, tiende al error. El segundo método, las herramientas de Xilinx guían al usuario para configurar y seleccionar los correspondientes atributos y añadirlo al diseño. El último consiste en utilizar templetes HDL (rtl) para introducirse dentro del módulo de la memoria interna [11].

Gracias a los recursos que ofrece Xilinx, se utilizará la IP del controlador de memoria, *Memory Interface Generator (MIG),* que es una interfaz entre la lógica del usuario y la interfaz física de la memoria. En la Figura 20, el microprocesador solicita las correspondientes transacciones de lectura o escritura; el controlador de memoria se encargará entonces de *manejar* estas solicitudes, lo traduce a los comandos correspondientes de la memoria DDR2 SDRAM y termina accediendo al dato que está en la memoria. [11]

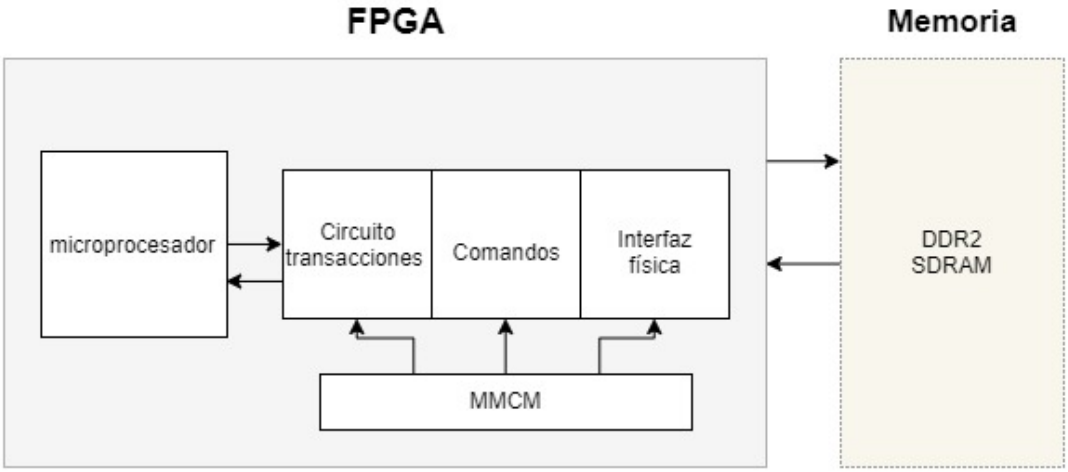


Figura 20. Diagrama de un controlador de memoria. [11]

Analizando el datasheet de la memoria, se puede encontrar una descripción general de la memoria. La memoria DDR2 SDRAM utiliza la arquitectura *4n-precarga*, que en líneas generales consiste en acceder en una lectura o escritura, a una transferencia de datos de un ciclo de reloj de un ancho de 4n bits.

La memoria trabaja con un reloj diferencial (*CK* Y *CK#*);un flanco a nivel alto de *CK* será cuando *CK* este *HIGH* y *CK#* este en *LOW*. Los comandos son registrados en cada flanco a nivel alto de CK.

La señal bidireccional de validación (DQS,DQS#) es transmitida externamente, junto con los datos. Como es una señal bidireccional, es transmitida por la memoria durante las lecturas y por el controlador de memoria durante las escrituras.

El dato por lo tanto será registrado en los dos flancos de la señal DQS, en cuanto el dato de salida es referenciado tanto en los dos flancos de DQS y en el de los de la señal CK.

Los accesos de escritura y lectura de la memoria están orientados en modo ráfaga; el acceso inicia con una ubicación de memoria seleccionada y continua para un número de ubicaciones en un programa en secuencia. El acceso inicia con el registro del comando *ACTIVE*, el cual será seguido por el comando *READ* y *WRITE.* Para acceder al banco y la fila se necesita que la dirección que es registrada coincide también con la del comando *ACTIVATE.* Por último, para seleccionar y acceder al banco y la columna inicial para poder hacer un acceso en ráfaga, la dirección registrada debe de coincidir con la usada en los comandos *READ* y *WRITE*. [27]

Las señales más importantes de la memoria las tenemos descritas en la Tabla 9.

Señal	**Tipo**	**Descripción**
A[12:0]	Entrada	Dirección de entrada. Proporciona la dirección de la fila para el comando ACTIVATE, la dirección de columna y el bit de precarga A10 ara los comandos de READ/WRITE.

BA[2:0]	Entrada	Dirección del banco de entrada. Se encarga de definir en qué banco se tiene que aplicar los comandos ACTIVATE,READ,WRITE, PRECHARGE.
CK,CK#	Entrada	Reloj. Señal de reloj diferencial.
CKE	Entrada	Señal de habilitación del reloj. Si se registra HIGH activa y si está a LOW desactiva el circuito del reloj en la memoria.
CS#	Entrada	Chip Select.
ODT	Entrada	Terminación On-die. Habilita cuando está HIGH las terminaciones de la resistencia interna de la memoria.
RAS#,CAS#,WE#	Entrada	Entradas comando.
DQ[15:0]	I/O	Datos de entrada/salida. Bus bidireccional de 64 Megx16
DQS,DQS#	I/O	Dato de validación. Es de salida con un dato de lectura y de entrada con un dato de escritura para un operación síncrona de origen.

Tabla 9. Señales más representativas de la memoria. [27]

También se describe brevemente las funciones de los principales comandos de la memoria, en la Tabla 10.

Comando	Descripción
DESELECT	CS# HIGH. Evita que nuevos comandos sean ejecutados por la memoria.
LOAD MODE	Los registros de modo son cargados mediante las direcciones de los bancos y las direcciones de entrada.
ACTIVATE	Se utiliza para abrir una fila en concreto de un banco para un acceso secuencial.
READ	Este comando se utiliza para inicializar un acceso de lectura en ráfaga en una fila activa.
WRITE	Se utiliza para inicializar un acceso de escritura en ráfaga en una fila activa.
PRECHARGE	Este comando se utiliza para desactivar una fila activa de un banco en particular o una fila abierta en todos los bancos.
REFRESH	Se utiliza durante una operación normal de la memoria DDR2 SDRAM.
SELF REFRESH	Se utiliza para retener un dato en la DDR" SDRAM, incluso si el resto del sistema está apagado.

Tabla 10. Descripción de los comandos más importantes. [27]

En la figura 22, se ve de forma más concreta la estructura que tiene la memoria MT47H64M16.

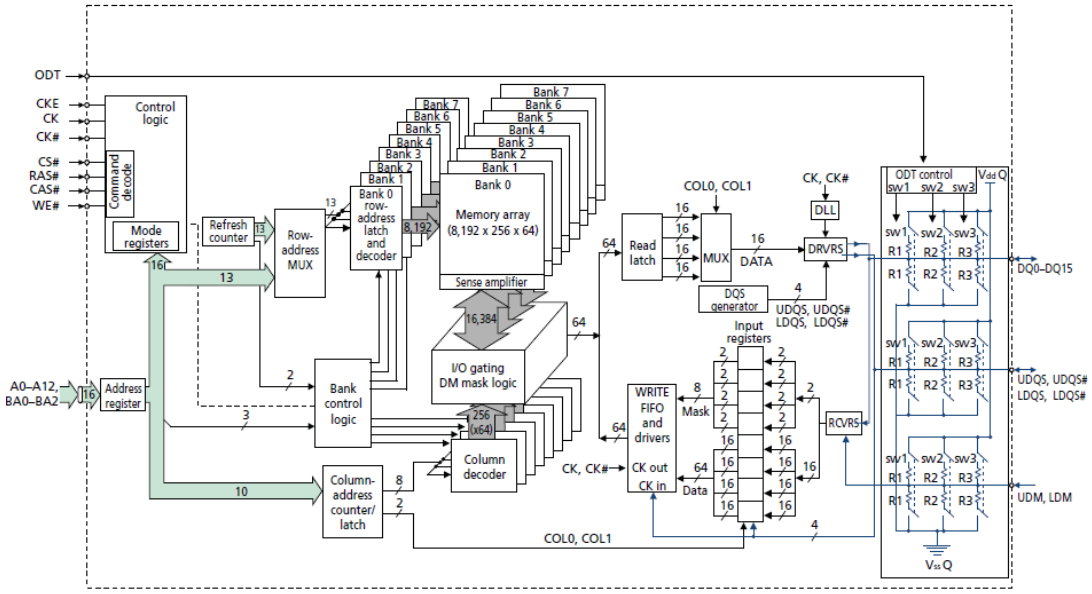


Figura 21. Diagrama de bloques de la memoria MT47H64M16. [27]

5.1.4.1.2. Memory Interface Generator MIG

Habíamos visto antes como sería la interfaz de un controlador de memoria, en la Figura 22, se puede ver la interfaz real para la FPGA que estamos usando.

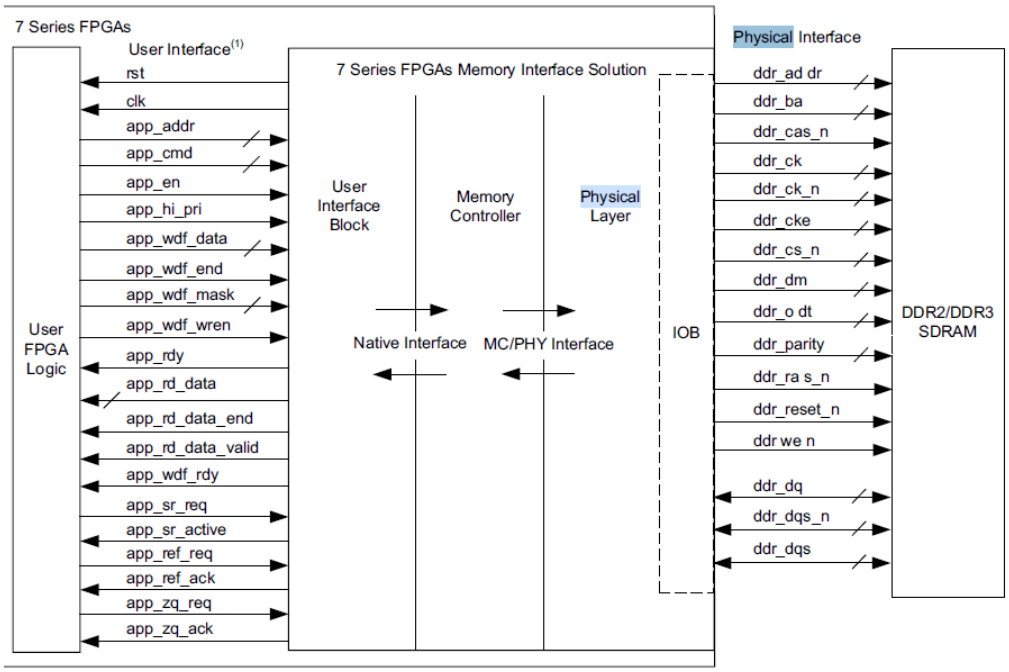


Figura 22. Interfaz del controlador de memoria para las FPGAs de la serie 7. [28]

La configuración del MIG (Memory Interface Generator), se ha realizado mediante la ventana de configuración que aparece dentro del catálogo de IPs de Xilinx. Estos son los pasos que se han tomado:

Lo primero que aparece del asistente para configurar la memoria, es lo que se puede ver en la Figura 23. Comprobamos que se ha seleccionado el número del dispositivo de la Nexys 4 DDR.

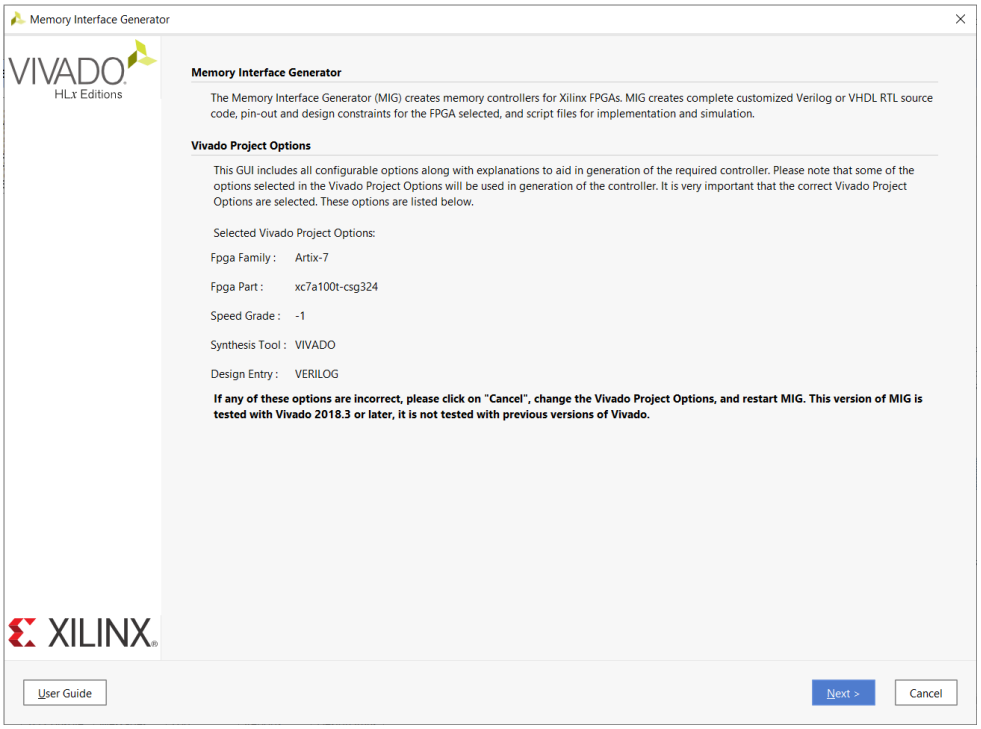


Figura 23. Primera ventana del asistente de configuración.

En la Figura 24, vemos que la interfaz de comunicación que utilizamos será el AMBA AXI4. También es posible especificar el nombre del componente.

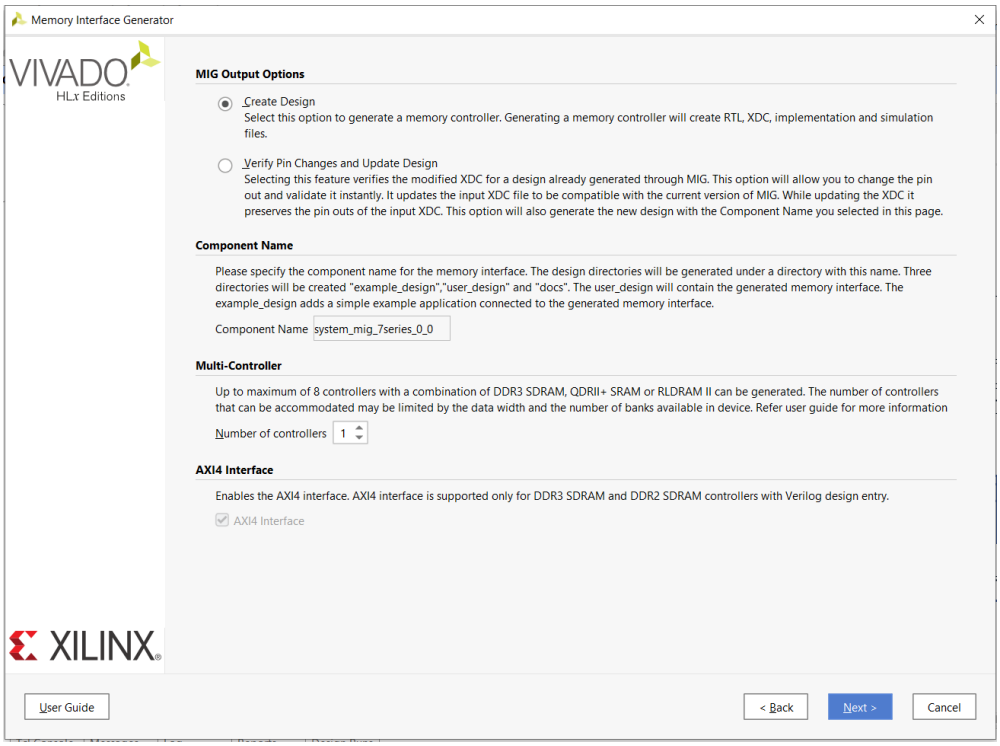


Figura 24. Asistente de configuración: nombre del componente e interfaz.

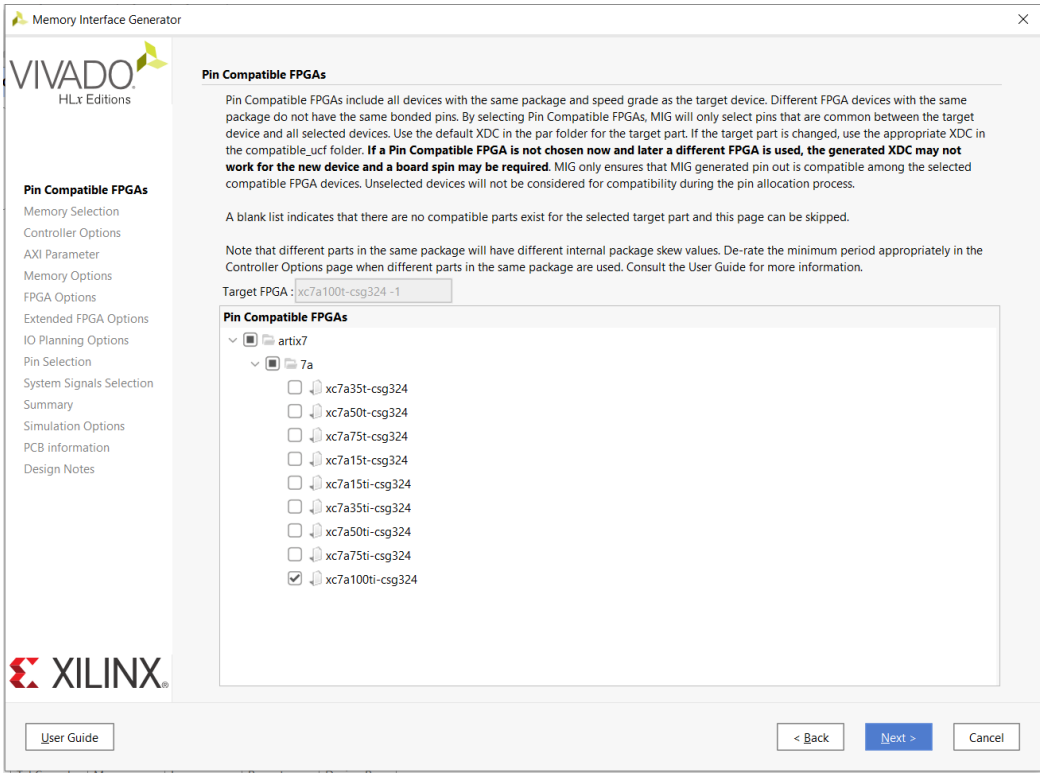


Figura 25. Asistente de configuración: seleccionar los pines compatibles con la FGPA.

En la Figura 25, se tiene que elegir los pines compatibles con la FPGA que estamos utilizando, en este caso es la última de la familia Artix 7, *XC7A100TI-CSG324.*

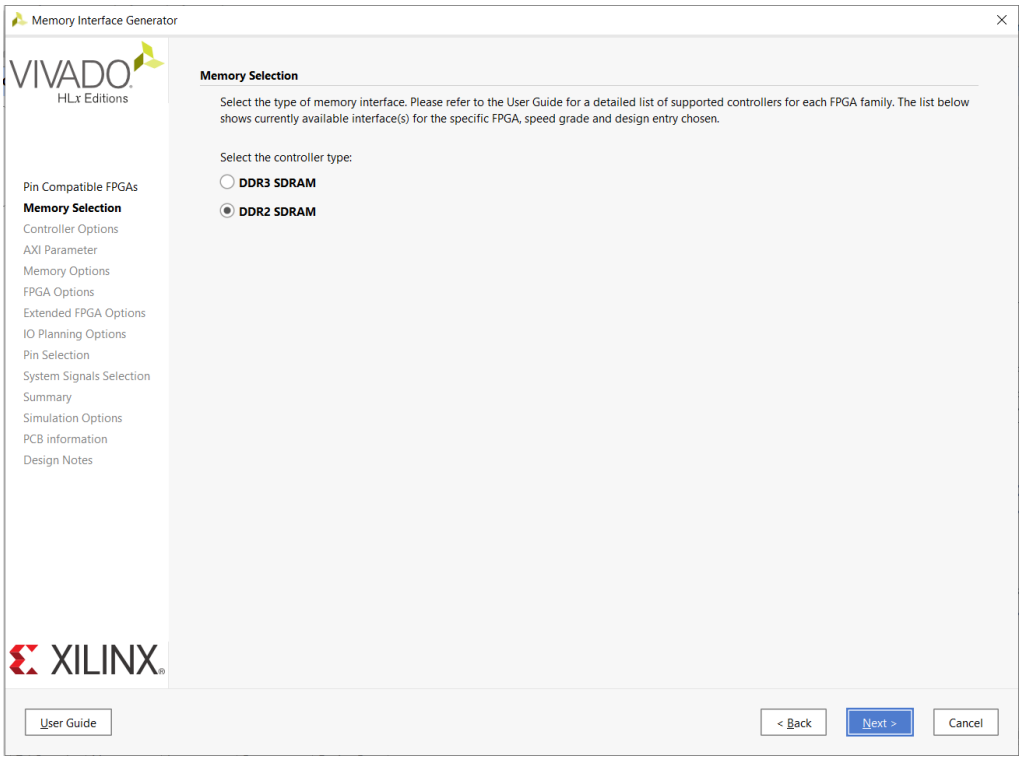


Figura 26. Asistente de configuración: Seleccionamos memoria DDR2 SDRAM.

El asistente de configuración, nos da la oportunidad, como se puede ver en la Figura 26, de seleccionar un controlador para una memoria DDR3 o DDR2, para este proyecto se selecciona para una memoria DDR2 SDRAM.

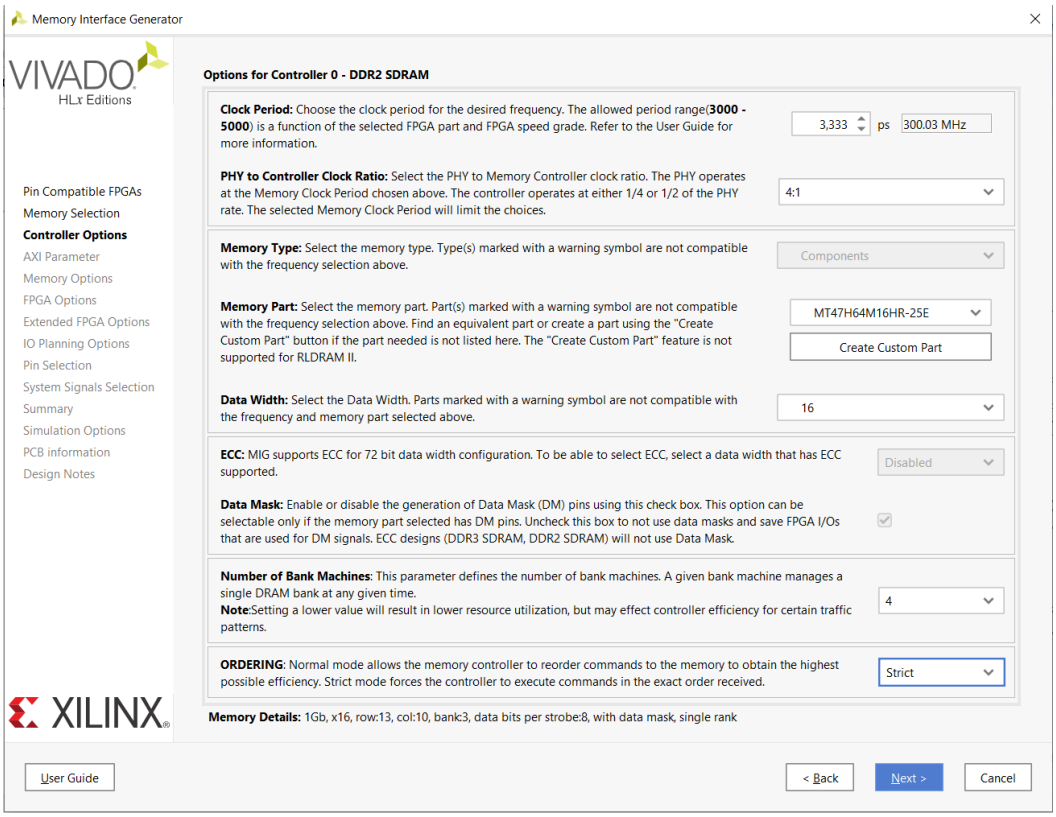


Figura 27. Asistente de configuración: Opciones del controlador.

Para este controlador se va a utilizar una frecuencia de 300.03 MHz, que está dentro del límite posible para utilizar en esta FPGA; la lógica interna de la FPGA es sincronizada por un recurso del reloj global, en este caso a un cuarto de la frecuencia del reloj de la DDR2 SDRAM. Se selecciona la memoria *MT47H64M16HR-25E*, el ancho de bus es de 16 bits. El reordenamiento de los comando a la memoria se hará en modo *Strict*, lo que significa que se ejecutan en el mismo orden en el que fueron recibidos, Figura 27.

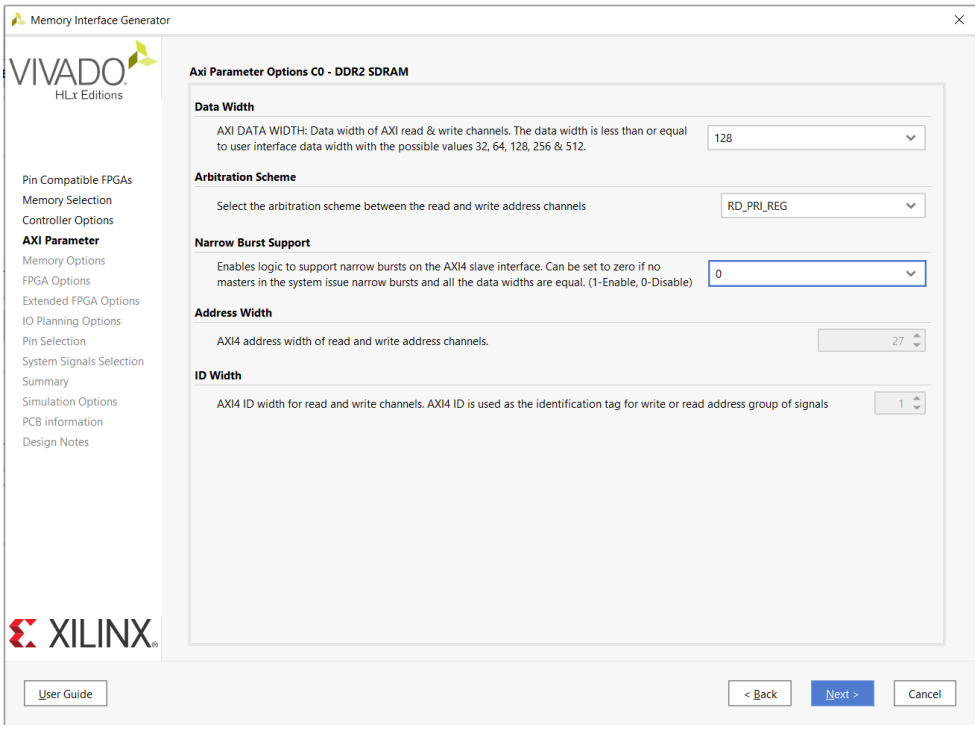


Figura 28. Asistente de configuración: Parámetros para AXI.

En la Figura 28, se puede configurar ciertos parámetro de la interfaz AXI4. El ancho de las direcciones los datos del AXI es de 128. Se deselecciona la opción de *Narrow Burst Support*, para que permita a la interfaz AXI4 guardar recursos y mejorar los tiempos.

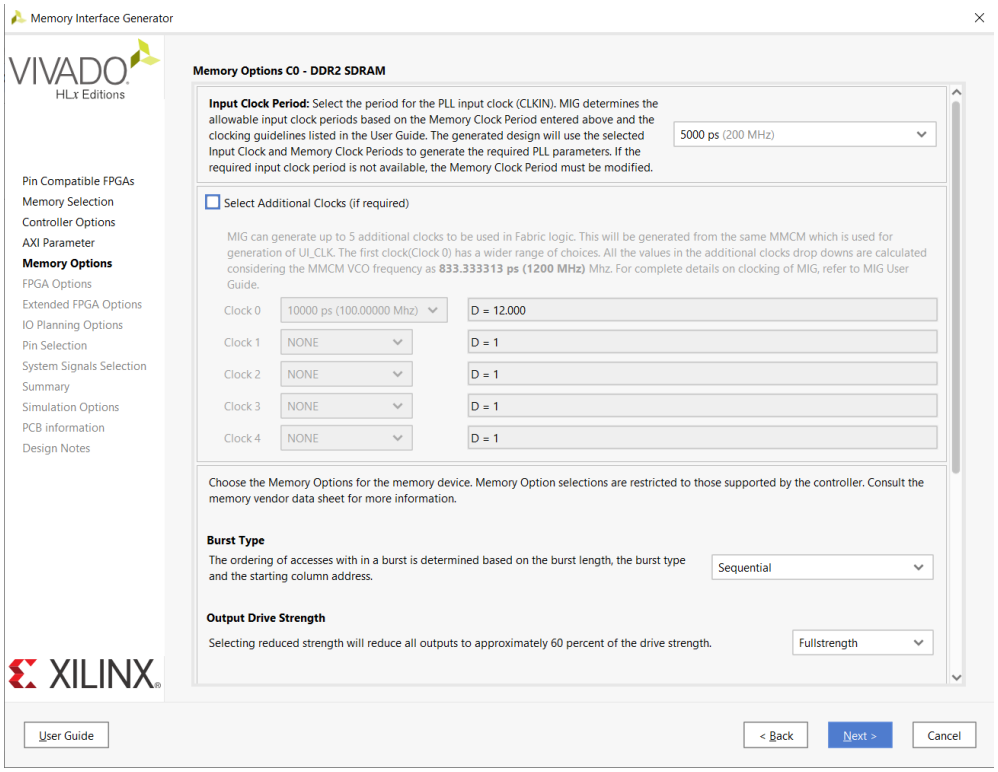


Figura 29. Asistente de configuración: Opciones para la memoria DDR2 SDRAM.

Para la entrada de reloj que se utilizará para memoria será de una frecuencia de 200 MHz. Como se ha hecho en el manual de usuario. El tipo de ráfaga es secuencial. En la opción de *Output Drive Strength*, se selecciona la opción Fullstrength, que sirve para fijar la impedancia de salida del driver de la memoria, Figura 29.

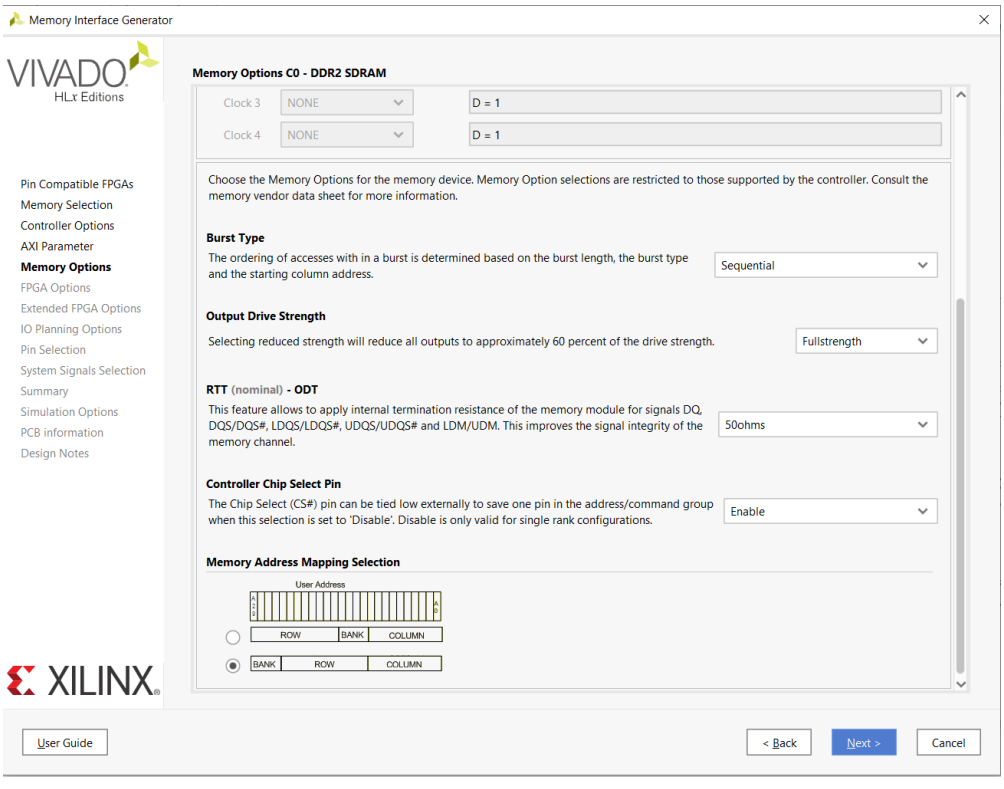


Figura 30. Asistente de configuración: Opciones para la memoria DDR2 SDRAM segunda parte.

También se ha seleccionado para la adaptación de impedancias ODT, resistencias de 50 Ω. El Chip Select se deja habilitado y por último la configuración del mapa de dirección de memorias, se selecciona el *bank-row-columnk,* es la más utilizada.

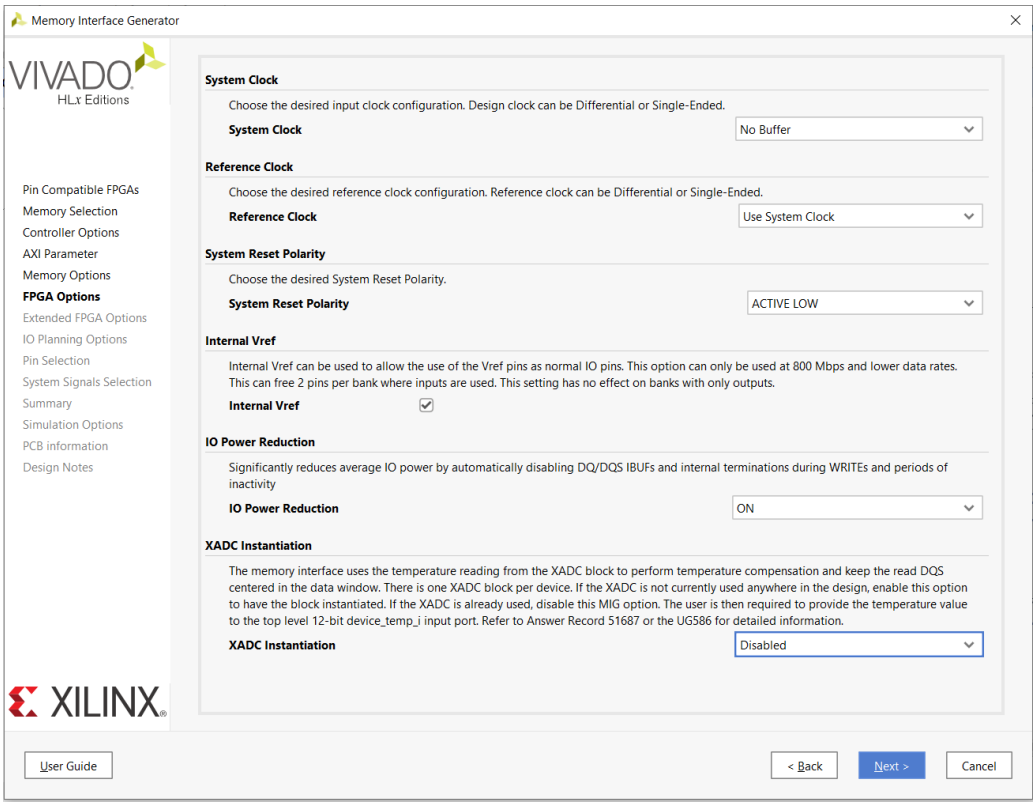


Figura 31. Asistente de configuración: Opciones para configurar la FPGA.

Las opciones que se permiten configurar en la Figura 31, se dejan las que vienen por defecto.

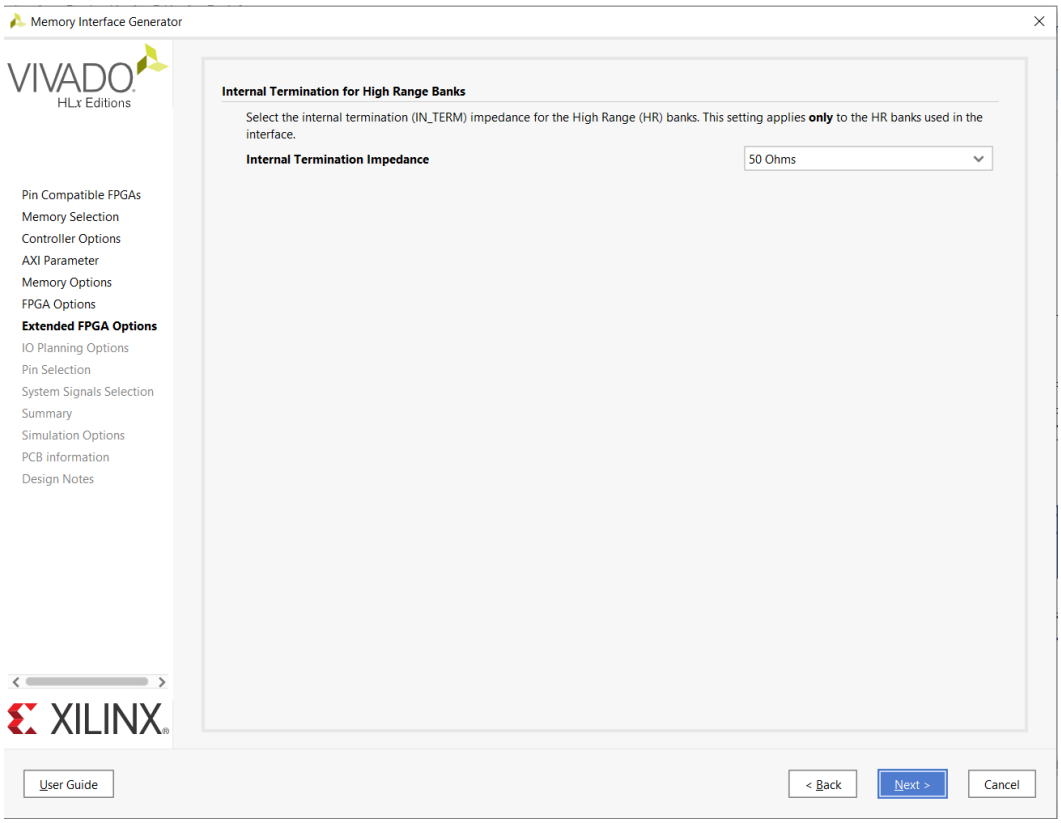


Figura 32.Asistente de configuración: Opciones extendidas para la FPGA.

Terminando con las configuraciones, en la Figura 32 se deja como impedancia interna, la resistencia de 50Ω.

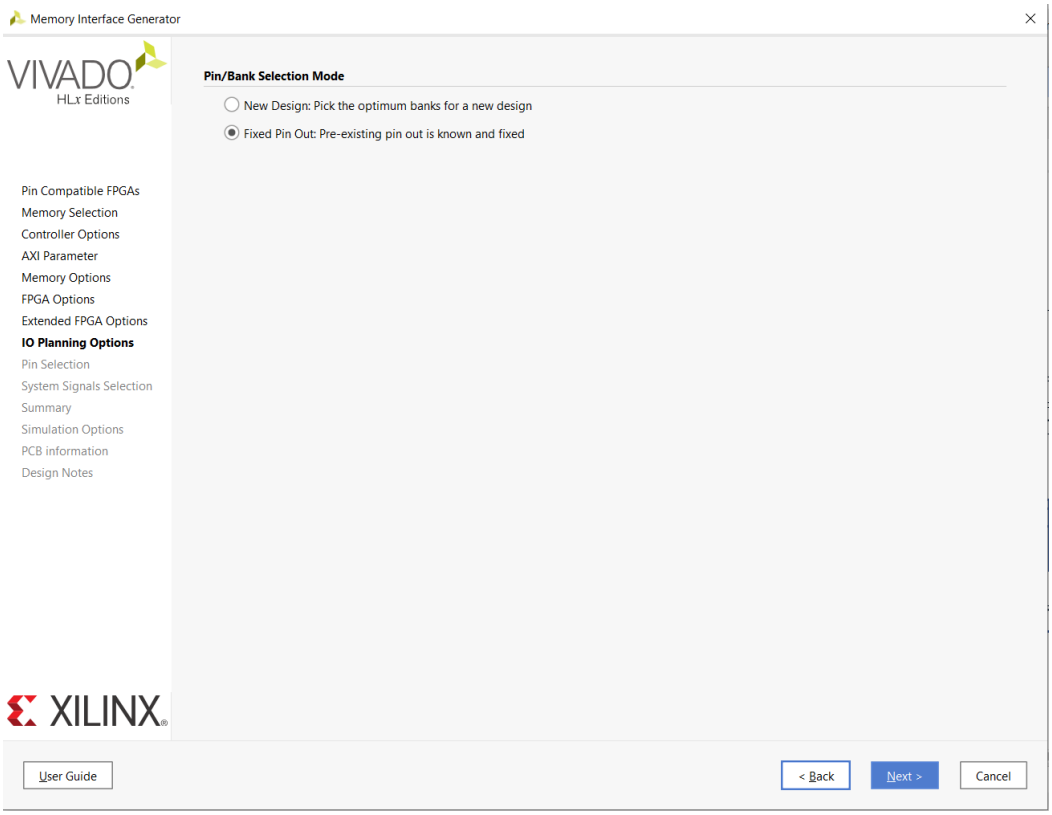


Figura 33. Asistente de configuración: Opciones para los pines de la memoria con la FPGA.

Los pines que se utilizarán, los podemos añadir en el fichero .xdc como se ha hecho en el manual de usuario, para este controlador se añadieron y posteriormente se validaron, como se puede ver en la Figura 34.

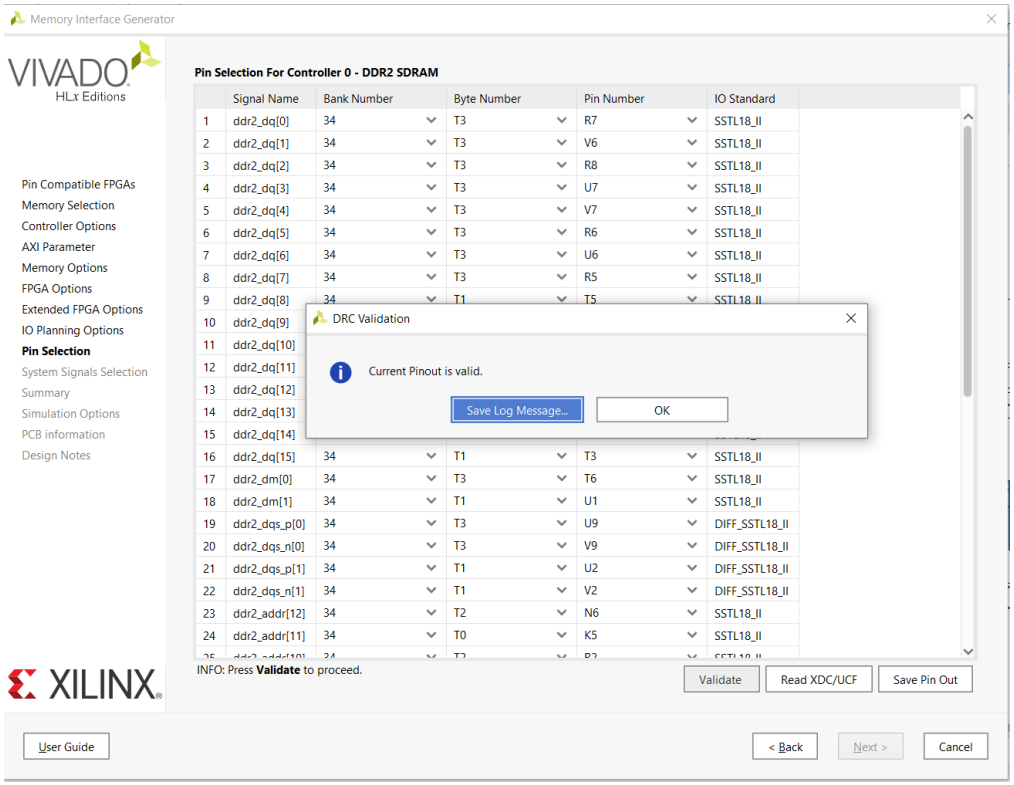


Figura 34.Asistente de configuración: Selección de los pines y validación.

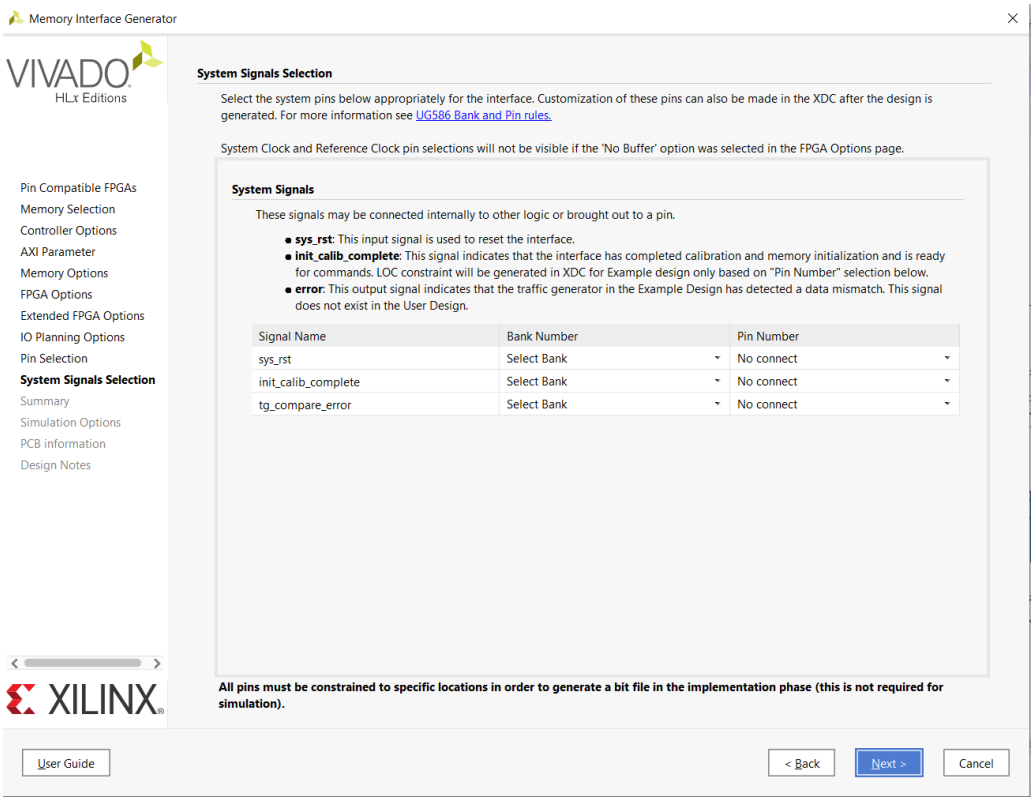


Figura 35. Asistente de configuración: Selección de las señales del sistema.

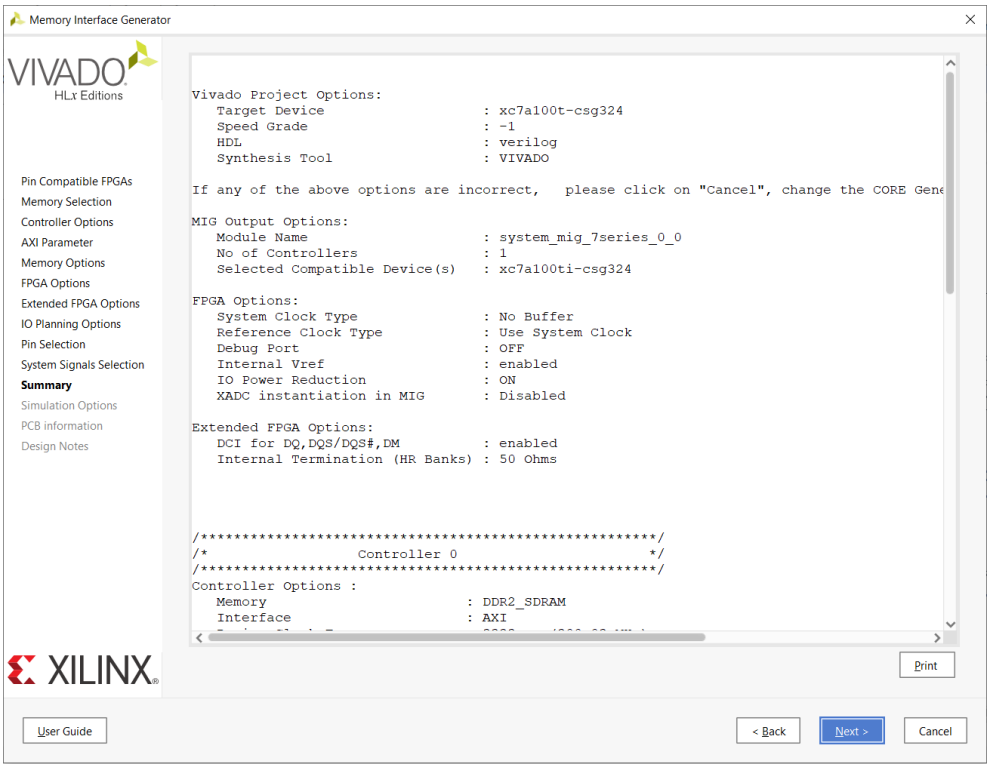


Figura 36. Asistente de configuración: Resumen.

En la Figura 36, se puede ver un resumen de todas las configuraciones que se han hecho.

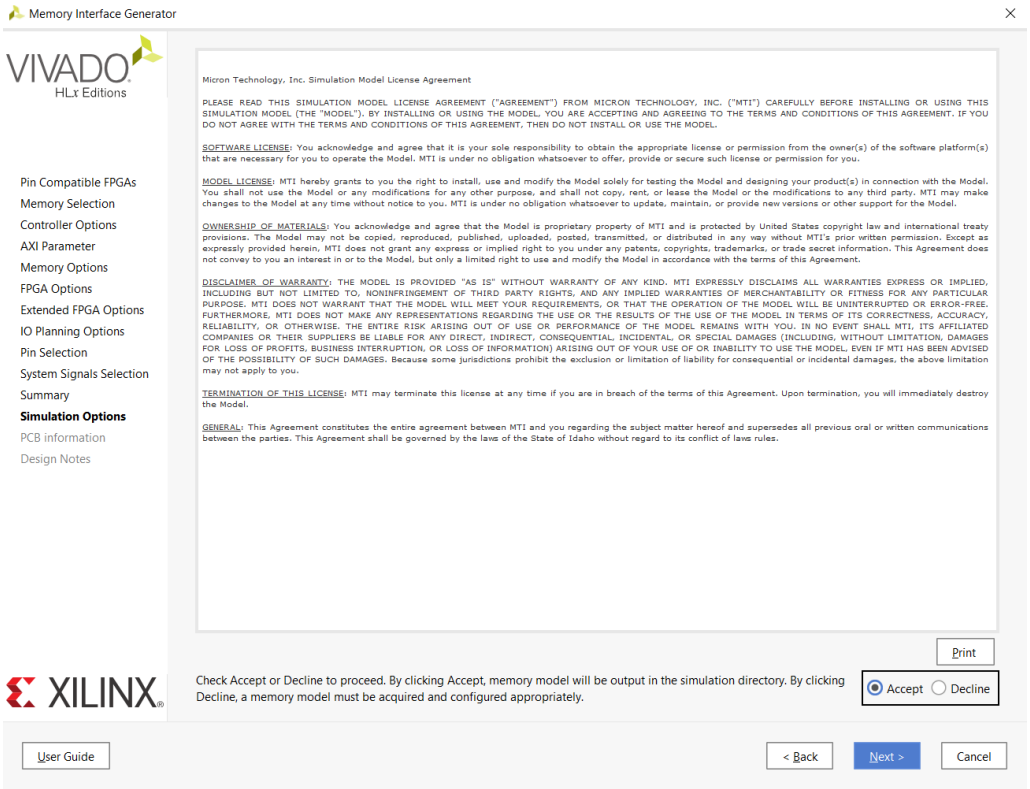


Figura 37. Asistente de configuración: Opciones de simulación y aceptar los términos de la IP.

Para poder hacer uso de la IP, es necesario aceptar los términos de la misma, como lo enseña la Figura 37.

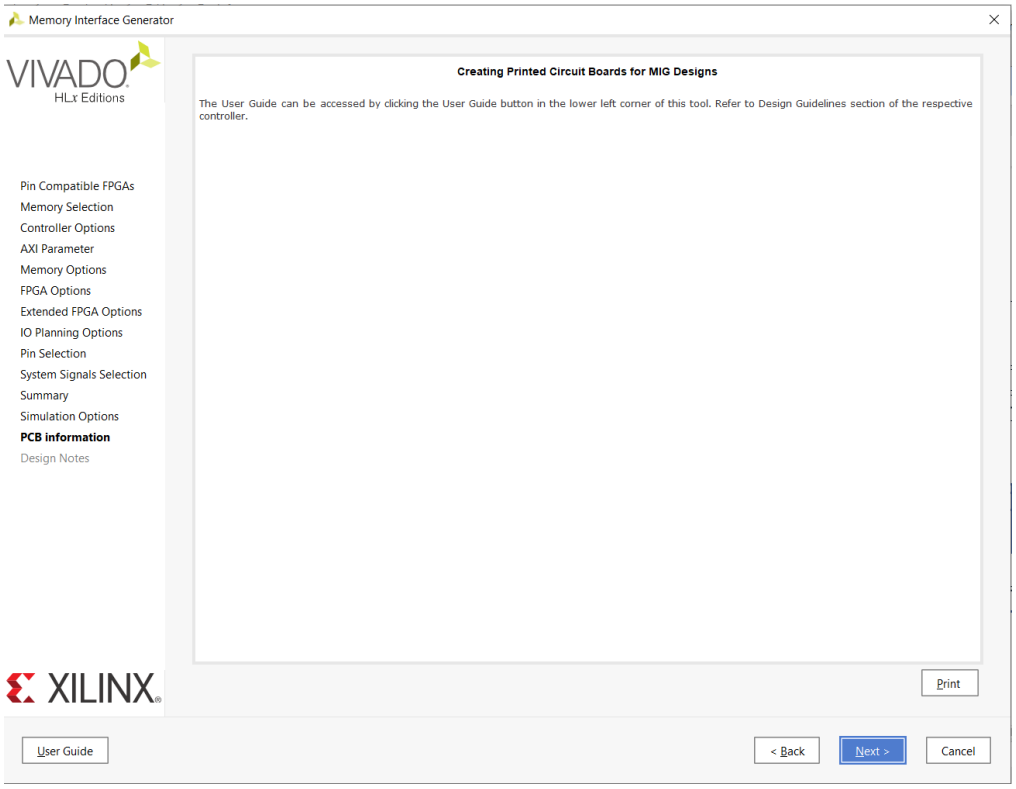


Figura 38. Asistente de configuración: Información de la PCB.

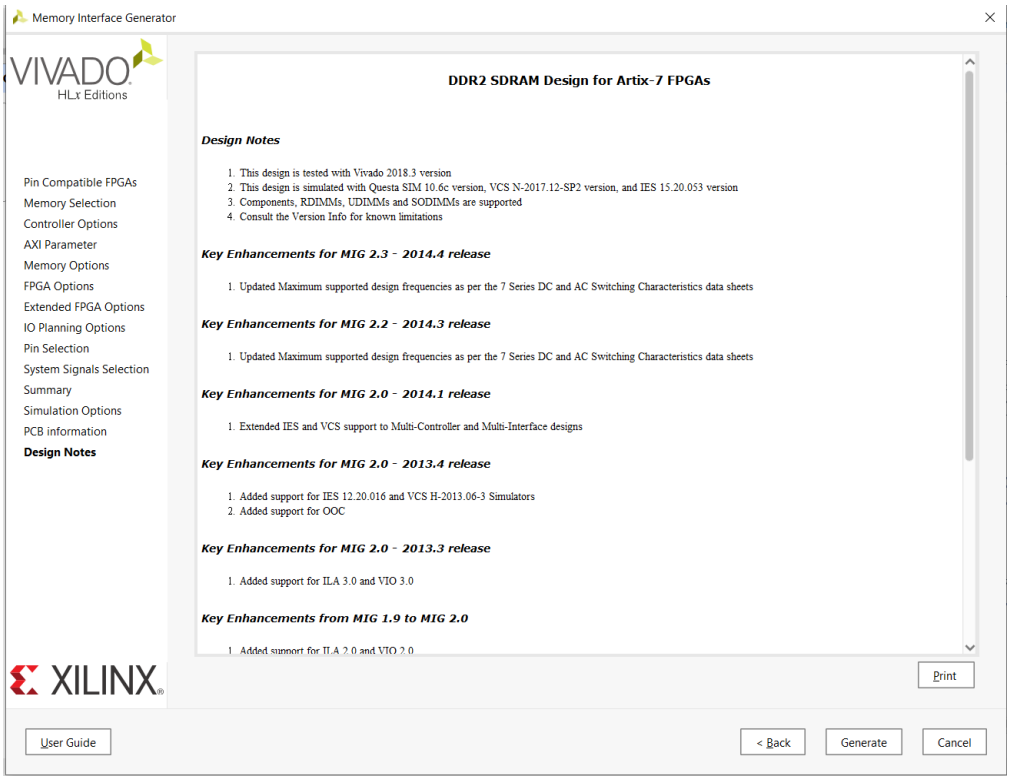


Figura 39. Asistente de configuración: Notas del diseño.

Lo que enseña la Figura 39 será la última que enseña el asistente de configuración del MIG, al dar a generar ya podemos proceder a la integración de este controlador en nuestro proyecto.

6.1. Comunicaciones on-chip

Son muchos los sistemas que requieren una comunicación entre elementos externos. Por este motivo, es necesario una comunicación que permita a los sistemas embedded conectarse a otros sistemas o elementos, de forma que se construya una infraestructura más compleja que esté controlada por su propio subsistema embedded.

Las interfaces de comunicación son sistemas estandarizados que proporcionan ventajas de cara al control de un sistema. La estandarización de estas interfaces permiten que sean escalables, modúlales y más confiables. [1]

Para encontrar una solución de comunicación *on-chip*, se deben de tener en cuenta algunos factores, como por ejemplo: la velocidad (relaciona el rendimiento y la latencia), la fiabilidad (si puede producirse perdidas de datos), compatibilidad con la red y otros más. Considerando estos factores, encontrar una solución a priori puede resultar complejo.

Por ejemplo, las comunicaciones internas entre los recursos son muy versátiles que minimizan los problemas de conectar muchos elementos cuando es requerido, de forma directa, haciendo necesario que compartir recursos o distribuir datos entre diferentes métodos se necesite una arbitración. las comunicaciones que son estandarizadas permiten este tipo de comportamiento. Así que el desarrollo de múltiples buses o estructuras "puente", de esto se ha hablado en la parte del bus AXI que describe la conectividad entre microprocesadores embedded. Sin embargo, considerando los otros factores como la escalabilidad, este tipo de estructura puede tener problemas en grandes diseños. Debido, a que tiende a empeorar a medida que aumenta el número de módulos conectados, ya sea porque se producen saturaciones durante el acceso a los datos, cuando varios de los módulos están deseando acceder al mimos recurso, esto puede conllevar a rebajar el rendimiento. [11]

Existen otros mecanismos que pueden resolver este tipo de problemas, cuando la complejidad es mayor donde hay un gran número de módulos incrementado, puede resultar útiles las estructuras como los *Network-on-chip* (NoCs) o *Messege Passing Interface (*MPI*)*.

Vamos a analizar diferentes tipos de conexiones para comunicaciones internas de módulos dentro de una FPGA.

6.1.1. Network on Chip

Un NoC consiste en una serie de enlaces conectados por unos router que interconectan múltiples núcleos. Los enlaces habitualmente son pares de canales bidireccionales conectados a un par de routers. Para poder hacer el acceso a los núcleos se hace a través de los routers. [11]

Pero para entender mejor lo que se ha explicado anteriormente, se va a explicar de que está compuesto una estructura NoC, como se ve en la Figura 40.

Un NoC se compone de tres bloques principales. El primero y más importante son los ***enlaces*** que conectan físicamente a los nodos y son los que implementan la comunicación. Estos enlaces de comunicación están formados por un conjunto de

cables conectados a dos routers en una red. Estos enlaces pueden consistir en uno o más canales físicos o lógicos, y cada canal se compone de dicho conjunto de cables. El número de cables por canal es uniforme por toda la red y se le conoce como ancho de banda del canal. Los enlaces pueden definir el rendimiento más puro y el consumo de energía de un NoC.

El segundo bloque es el ***router***, el cual implementa el protocolo de comunicación. El protocolo consiste en una serie de reglas definidas durante el diseño e implementadas en el router, que manejan las situaciones comunes durante la transmisión de paquetes. Este router está compuesto por un número de puertos de entrada (conectado a canales compartidos NoC), a puertos de salida, a una matriz de interruptores conectados a los puertos tanto de entrada como de salida. Y a un puerto local de acceso al núcleo de la IP que está conectada a su router. Este bloque también tiene un bloque lógico que implementa por ejemplo [1]:

- *Control de flujo* que se caracteriza por el movimiento de paquetes a lo largo del NoC, implicando problemas globales y locales.
- *Algoritmo de enrutamiento*, que es la lógica que selecciona un puerto de salida para poder reenviar un paquete que llega a un puerto de entrada, la selección se realiza según la cabecera del paquete.
- *Lógica de arbitración*, esto consiste en seleccionar un puerto de entrada cuando varios paquetes llegan de forma simultánea solicitando el mismo puerto de salida.
- *Switching* define como se transmiten los datos desde el nodo de origen al destino.
- *Buffering* el método que se utiliza para almacenar la información en el router cuando hay una congestión en la red y el paquete no puede ser reenviado.

El último bloque es un **adaptador de red** (NA) o interfaz de red (NI). Este bloque lo que hace es conectar la lógica entre la red y las IP, estas pueden tener diferentes protocolos de interfaces respecto con la red. Este bloque es importante porque permite la separación entre la comunicación y la computación, y también permite la reutilización de la infraestructura del núcleo y la comunicación.

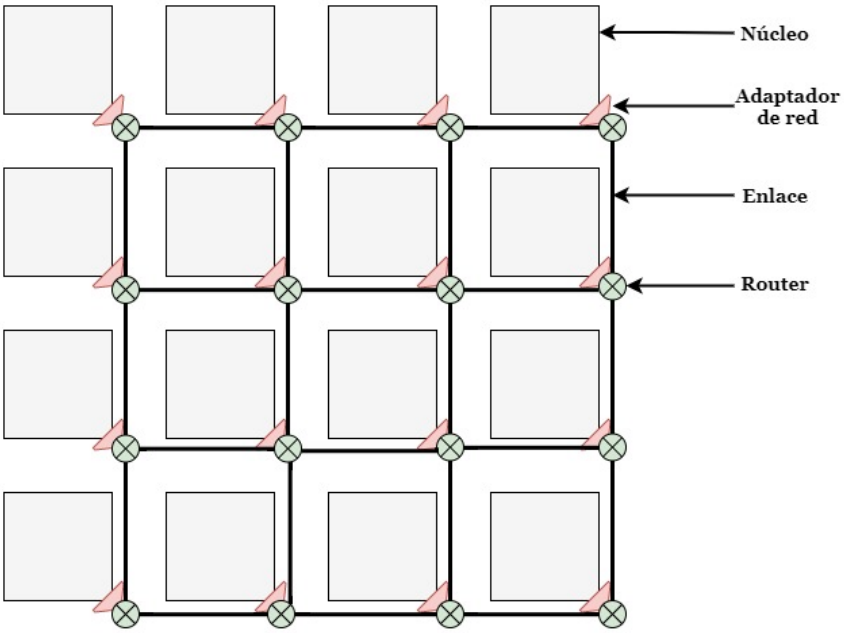


Figura 40, Estructura de un NoC

La principal ventaja que presenta la estructura NoC es que no se produce degradación de la velocidad con el tamaño del sistema desde que todas las conexiones se puedan hacer de forma local, y mientras sea posible conseguir el paralelismo, dando la oportunidad que muchos nodos se puedan comunicar en paralelo utilizando múltiples caminos. Y según las reglas que se establecen en el protocolo se puede predecir que se puede contar con suficiente ancho de banda para las comunicaciones entre nodos en el NoC.

En la actualidad, los fabricantes de FPGA no proporcionan muchas herramientas para el diseño de este tipo de estructuras, y las herramientas de terceros puede que no sean compatibles o sean aptos para diseños específicos con FPGAs.

Dentro del abanico de herramientas que ofrece ARM, la que ayuda implementar este tipo de soluciones se llama *CoreLink Network Interconnect (NIC)*, para poder utilizarla hace falta disponer de la licencia *ARM Flexible Access*, que tiene acceso tanto para *Starups* como para investigación, en el momento que se estuvo realizando este proyecto la licencia para investigación no estaba disponible en nuestra región.

6.1.2. Conexiones Point-to-Point (P2P)

Cuando las interconexiones son para un número pequeño de bloques, las conexiones punto a punto es un tipo de conexión que puede ser una buena solución. En este caso, no se tienen conexiones compartidas, lo que significa que los bloques están siempre preparados para comunicar. Una de las características a este tipo de comportamiento es que el rendimiento de la comunicación aumenta, incluso también asegura un comportamiento predecible del sistema.

A pesar de las ventajas anteriormente explicadas, este tipo de comportamiento sencillo acarrea más problemas que ventajas. Un de los problemas es la falta de estandarización, porque presenta menos estándares que los otros dos tipos de comunicación que se explican aquí. Como se había visto en el caso de los NoC. Sin embargo, puede funcionar en protocolos de transacciones simples, siendo eficiente en transacciones síncronas y asíncronas. Por ejemplo, cuando se trata de una comunicación asíncrona, las memorias FIFO se deben de ubicar en ambos lados de la comunicación, por lo que utilizarán dos relojes, tendrán doble puerto para realizar el acceso a las dos regiones.

Este tipo de estructura puede ser fácilmente implementado en un modelo de flujo de datos. En este modelo, las partes que lo integran siguen unas directrices de activación autónoma, de forma que el modelo empieza a trabajar siempre que todos los datos de entrada de una o más conexiones estén disponibles. Al terminar se envían a través de las conexiones de salida P2P. Como se ha explicado, nos podemos dar cuenta que no se necesita un control central. Este tipo de conexión no necesita un direccionamiento, también es posible recibir datos etiquetados de diferentes tipos de elementos de la misma conexión.

Podría considerarse que el comportamiento predecible y la falta de control central, podrían ser muy buenas características. Sin embargo, la principal desventaja que se tiene es que este tipo de solución tiende a incrementar el área de comunicación, por lo tanto no es muy escalable [1].

Dentro de las conexiones P2P, encontramos una muy popular que se llama *Messege Passing Interface (MPI)*, este tiene la ventaja que si presentar un interfaz estándar que representa una de sus mayores características y ventajas.

6.1.2.1. Messege Passing Interface (MPI)

Es una librería estándar que se usa para desarrollar programas de paso de mensajes para el procesamiento de sistemas. Fue desarrollada para especificar una interfaz que permitiera a programadores escribir aplicaciones que fueran portables para diferentes arquitecturas paralelas. [29]

El objetivo del MPI es establecer un estándar de paso de mensajes que sea portable, eficiente y flexible. MPI no es un estándar ISO o IEEE, aun así se ha convertido en un estándar industrial para escribir programas de paso de mensaje en plataformas HPC (High-Processing Computing).

El objetivo principal del MPI es ser portable a través de diferentes máquinas. Al utilizar un lenguaje de programación como C, permite otorgarle al MPI ese grado de portabilidad. De forma que si se utiliza el mismo código fuente de paso de mensajes, este puede ser ejecutado por diferentes sistemas. Por este mismo motivo también consigue un grado de flexibilidad en el código desarrollado. El MPI puede trabajar en sistemas homogéneos y heterogéneos, nos referimos a procesadores de diferentes arquitecturas. La implementación del MPI se hará de forma automática sin ser necesario ninguna conversión de datos y utilizará el protocolo de comunicación correcto.

El estándar MPI no especifica de forma específica las operaciones con memoria compartida. Tampoco tiene facilidades para la depuración, un soporte específico para hilos o funciones I/O.

Dado que los cálculos en las FPGAs se pueden hacer tanto a nivel software en un procesador embedded o por mecanismos hardware. La implementación MPI puede hacerse a nivel software

El mecanismo básico del MPI consiste en transmitir datos entre un par de procesos. Por ejemplo, uno de ellos se encarga de enviar datos y el otro en recibir, como ya se había explicado anteriormente a este mecanismo se le conoce como comunicación Point-to-Pont. Este estándar proporciona funciones de enviar y recibir que permiten la comunicación de datos con una etiqueta asociada.

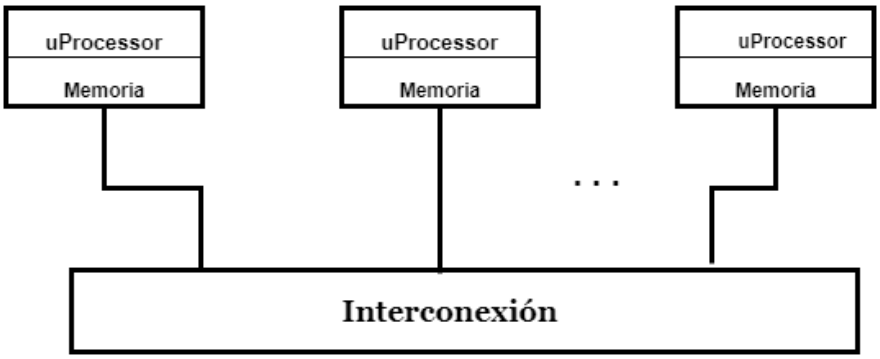


Figura 41. Arquitectura hardware para implementar el protocolo MPI a nivel software. [29]

Este protocolo puede tener hasta 100 funciones, siendo las más básicas *MPI_Init, MPI_Comm_size, MPI_Send, MPI_Recv* y *MPI_Finalize,* proporcionando herramientas esenciales para la resolución de la mayoría de problemas.

La forma básica de iniciar consiste en [29] :

1. la inicialización del MPI.
2. Se procede a obtener la información acerca de la cantidad y el rango de procesadores.
3. Después se analiza el número de interacciones.

Cuando se refiere a el rango, se hace referencia al número de procesadores que ejecutan la aplicación. [30]

Las funciones punto a punto **síncronas** como *MPI_Send* y *MPI_Recv* son comúnmente consideradas como las funciones principales de la transferencia de datos en múltiples procesadores. Hay ciertos modos de comunicación para implementar dichas funciones:

- **El modo síncrono**: donde ambos procesadores (el que envía y recibe) hacen el acuerdo de la transferencia (handshake) y esperan que cada uno inicie la transferencia de datos.
- **El modo buffered**: donde el procesador que envía, escribe os datos en el buffer y no espera que al procesador que recibe para iniciar la transferencia.
- **El modo estándar**: donde está arriba de la implementación MPI y determina donde serán almacenados los mensajes o donde no.
- **El modo preparado:** donde las operaciones de enviar sólo funcionan si se ha publicado con anterioridad la solicitud de recibido. [31]

Cada una de esas funciones se identifica por un rango del procesador de destino, etiqueta y comunicador, siempre estos tres parámetros. El comunicador es un objeto especifico que fija el contexto de la comunicación entre los grupos de procesadores.

Por otra parte, para las funciones **asíncronas** *MPI_Isend* y *MPI_Irecv* y las funciones colectivas más comunes, como *MPI_Barrier, MPI_Reduce Reduce* y MPI_Bcast, también son funciones importantes definidas en el MPI-Forum.

6.1.3. Interconexión basada en buses

En términos generales un *bus* es un conjunto de cables interconectados, de forma estandarizada, de al menos dos elementos. Una interconexión de este tipo involucra dos cosas: aspectos *hardware* y un protocolo lógico. Los aspectos hardware determinan la capa física de interconexión, aspectos eléctricos y también la topología. El protocolo lógico trata con los problemas temporales, el tipo de transacción soportada y el modo de arbitración.

"Un bus puede contener más de un master y más de un esclavo. El bloque máster se encarga de iniciar la transacción, proponer el tipo de acceso (escritura o lectura), realizar la petición de transacción y proporcionar una dirección. Por otra parte, el bloque esclavo se encarga de responder a las peticiones que le llegan del master y recibir los datos, si se trata de una transacción de escritura, o de proporcionarlo si es una lectura, y proporcionar la dirección especificada por el master. En el caso de que más de un master realiza una petición, se debe de seguir un modelo de arbitración, que permita decidir cuál de los solicitantes obtiene la aprobación para la transacción". [1]

La topología del bus también es una especificación física importante. Se explicarán cuatro de ellas. La primera es una de las más simples, la *single-shared* (sólo permite una transacción al mismo tiempo). Estas pueden ser multimaster y permitir transacciones complejas. La desventaja que tiene es que cuando aumentan los bloques conectados a este tipo de topología puede llegar a saturarse, empieza a perder rendimiento ya que aumenta las longitudes de las conexiones.

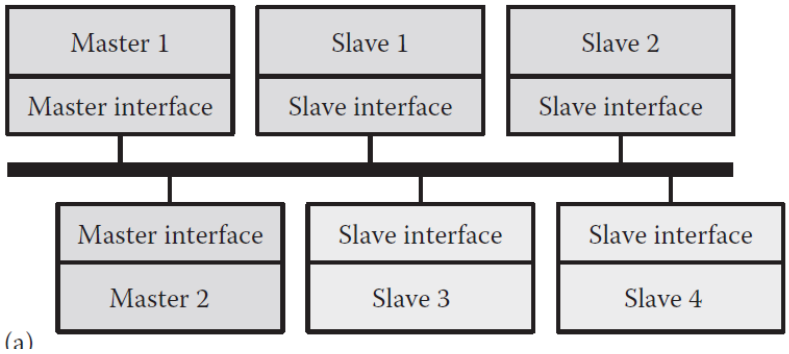


Figura 42. Topología single-shared. [1]

Las siguientes topologías son la *crossbar switch* y la *ring-based*. La primera topología mencionada tiene como característica que incrementa los recursos utilizados porque hay más caminos que conectan los masters con los esclavos. Por otra parte, la segunda topología puede añadir altas latencias mientras se reduce el uso de los recursos.

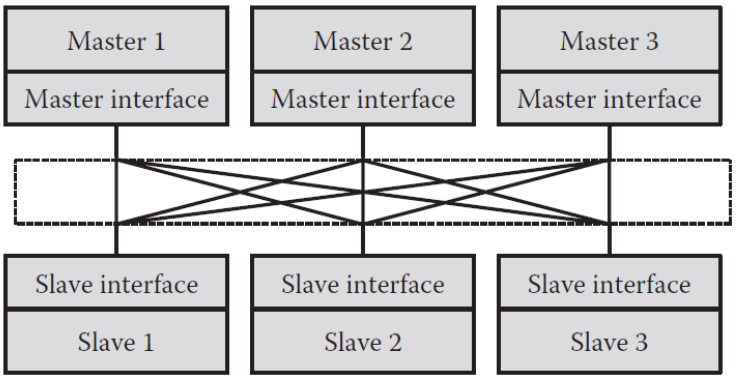


Figura 43.Topología Crossbar. [1]

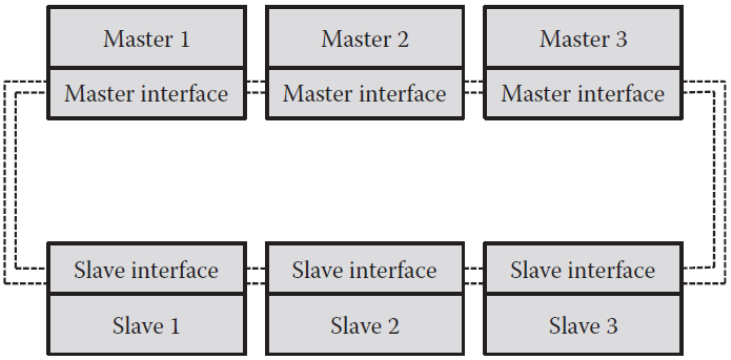


Figura 44.Topología Ring-based. [1]

La topología *bridge*, es muy práctico cuando el número de bloques interconectados es alto. Este tipo de topología permite un acceso paralelo, por lo que se incrementa el ancho de banda. Se puede dar el caso los elementos conectados puedan operar a diferentes velocidades, por lo que el bus se adaptará. Ese es el caso de los buses más sencillos que tiene una velocidad baja para el acceso de periféricos, y también se tienen buses más complejos que funcionan con velocidades más altas. [1]

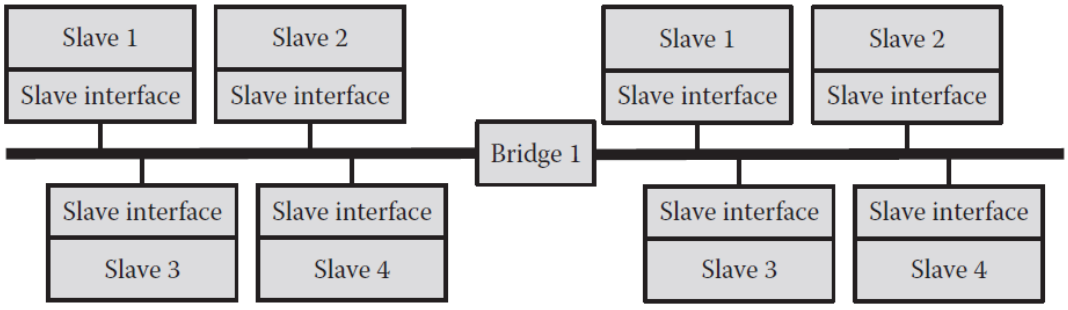


Figura 45. Topología bridge. [1]

La sincronización o *timing* también es una característica importante de los buses. Hay dos tipos, pueden ser síncronos o asíncronos. En el caso de que sea síncrono, todos los tiempos están sincronizados a la señal de reloj del master, este tipo de bus proporciona un acceso más rápido comparado con los buses asíncronos. Por otra parte, los buses asíncronos, al no tener una señal de reloj, el control se lleva mediante el evento de unas señales específicas. Este tipo de bus tiene como desventaja que su modo de control es más complejo y requiere un protocolo entre

el master y el esclavo; y como ventaja es que proporciona más sincronización ya que asegura la compatibilidad con los elementos conectados.

La arbitración es un aspecto también muy importante. Por ejemplo, en el caso de tener que manejar varios masters, para empezar una transacción tendrán que realizar una solicitud de arbitración (AR). El árbitro lo mantiene un ciclo ARB, para decidir qué master le dará el acceso según como sea la configuración. Cuando se le da el acceso al master, entonces puede enviar una solicitud al esclavo (RQ), y este pasará a estado ocupado, hasta que tenga los datos preparados por lo que se envía un ACK y se termina la transacción.

7.1. Benchmarks

Las ventajas que proporciona la interconexión de diferentes bloques que cumplen funcionalidades determinadas según el sistema, hacen que las comunicaciones on-chip necesiten como las NoC, P2P, Interconexión basada en buses, necesiten de utilizar BenchMarks (bancos de pruebas) para evaluar de forma imparcial el funcionamiento de la estructura interconectada.

Ayuda a los desarrolladores a determinar qué tipo de topología funciona mejor, el bus que se utiliza, como hacer las interconexiones es una parte fundamental del diseño. No existen muchas metodologías estándar, cada fabricante o diseñadores tienen la suya propia, por lo tanto además de ser una tarea necesaria, también es muy compleja porque tiene que adaptarse en cierta medida al sistema que se tiene que probar.

"Debida a la complejidad que requieren los diseños para las comunicaciones on-chip, en especial para la NoC. Definir cómo afecta el tráfico de datos a su funcionamiento. Para conocer y evaluar el funcionamiento de dichas conexiones es necesario mediante simulaciones de tráfico de datos reales". [32]

Para poder conocer que impacto tienen las aplicaciones de diferentes áreas, se necesita un modelo a partir de los resultados de las simulaciones.

"Lo principal del método que se utilice debe consistir en crear un modelo del tráfico de la simulación de la interconexión que se ha implementado. Este tráfico que se ha creado se usará entonces para estimular el modelo de la red con su simulador apropiado. Con esta simulación de forma aproximada se conocerá el funcionamiento para un tráfico de datos en específico". [33]

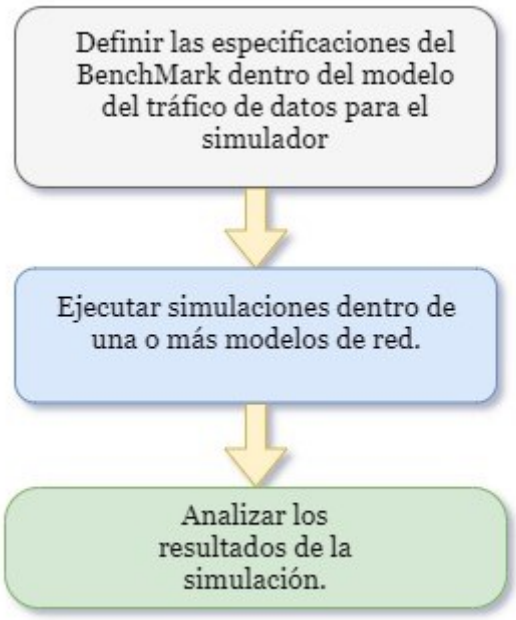


Figura 46. Diagrama de flujo del BenchMark. [32]

En la Figura 46, vemos que el método del BenchMark consiste en tres pasos. El primero consiste en conocer las especificaciones del modelo de tráfico para el simulador, dependiendo también de la metodología para la simulación; Se debe ejecutar el modelo de tráfico como estímulos sobre el modelo hardware según la red de interconexión establecida; por último se recolectan los datos de las simulaciones.

8.1. Implementación y resultados

En este proyecto lo que se quiere hacer es una implementación de varios núcleos en una FPGA de Xilinx. Para poder llevar a cabo esta tarea se ha analizado todos los parámetros importantes que tienen los Cortex-M1.

Dado que los ejemplos que proporciona ARM donde utiliza este microprocesador, han empleado otras FPGAs, se empezará como primer paso conocer el entorno de trabajo para la Nexys 4. El inicio de este trabajo empieza desde realizar un diseño muy básico, simplemente con un solo núcleo, hasta una configuración NoC con cuatro núcleos, pasando por varios diseños con diferentes periféricos. Para que fuera los más simple posible, la configuración que se optó para este microprocesador fue hacerlo sin depuración, de esta forma las señales de reset y de reloj que necesitaría el núcleo no serán necesarias. Sólo es necesario implementar las necesarias para un funcionamiento normal sin depuración, más adelante se detallará como se realizó.

Una vez que se tiene el primer diseño, podemos ir añadiendo complejidad de manera que podamos abrir camino para llegar a una implementación de cuatro núcleos. El siguiente paso consiste en añadir un segundo procesador, donde se estudian qué características particulares tiene este nuevo diseño.

Partiendo del diseño anterior, nos interesa conocer como funcionarían memorias externas como las tipo Flash y tipo memoria DDR2. Debido a que si queremos trabajar hasta con cuatro procesadores, es interesante conocer como compartirían este tipo de recursos los diferentes núcleos.

Es muy importante seguir este proceso, con lo que se explica en el manual de usuario, ya que se explica paso a paso las medidas necesarias, para generar los ficheros necesarios y de esta forma se pueda programar la FPGA.

El último paso, consiste en estudiar el comportamiento de cuatro procesadores con la memoria DDR2 incidiendo en cómo se han configurado los núcleos así como; qué se ha hecho a nivel software para hacer el un ejemplo de comprobación.

Como dato interesante, se han estudiado tres formas diferentes que proporcionan las IP del catálogo de Vivado, para la interconexión de nuestros núcleos con los periféricos mediante el bus AXI.

Cada paso que se ha ido detallando, se ha hecho una observación y análisis de los resultados que ofrece la implementación del hardware en Vivado, para ayudarnos a conocer cuantos recursos han sido utilizados de la FPGA y que posibilidades se tendrían de añadir más núcleos o de que manera se tendrían que configurar los interfaces de memoria para no sobrepasar los recursos disponibles de la Nexys 4 DDR.

Las pruebas que se hacen en cada diseño desde el más básico al más complejo que es la implementación de 4 núcleos. Consiste en ver que se genera de forma correcta los fichero para programar la FPGA; cuando se hacen pruebas con periféricos, ver si

las pruebas que se hacen son las esperadas, tanto para la UART, GPIO, memorias externas; la red que se establece entre varios núcleos funciona.

La información que se extrae de todas estas pruebas nos ayudarán a determinar qué se tiene que tener en cuenta para el siguiente diseño que se desee implementar. Además, permite determinar si es posible usar la Nexys 4 DDR, entender si es posible aplicar los conceptos teóricos que se explicaron anteriormente como las comunicaciones on-chip, el mapa de memoria que se explicó funciona según lo esperado; permite comprobar este tipo de aspectos. Nos ayuda a saber como podemos usar los recursos internos de la FPGA, sin sobrepasar sus máximos. En definitiva nos ayudará a conocer como debemos de tratar los Cortex-M1 dentro de un entorno de trabajo con una Nexys 4 DDR de Xilinx. Nos interesa saber si el diseño de un SoC muy básico es una idea plausible para el entorno que ofrece Xilinx.

En esta parte del proyecto, se explicará la implementación que se llevó a cabo y los resultados que se han obtenido.

8.1.1. Implementación de procesadores Cortex-M1

Para poder hacer una implementación de un procesador Cortex-M1 dentro del entorno de Xilinx en nuestra FPGA, se tiene que seguir una serie de pasos que se especifica en la Figura 47.

Como se puede ver el primero consiste en crear el proyecto en Vivado, donde vendrán el diseño hardware con el microprocesador y los periféricos conectados, aquí se generará un bitstream, que se explica todo con detalle en la parte del manual de usuario, para poder exportarlo al SDK. Hay que tener especial cuidado en esta parte del proceso ya que además de generar el bitstream también se genera el fichero de descripción hardware (HDF) y aun más importante el fichero MMI (Memory Map Info), que es el que define las ubicaciones de los bloques BRAM que se utilizan para instanciar el procesador en la FPGA. Cada vez que se actualiza el diseño hardware, se tendrá que actualizar también este fichero.

El siguiente paso consiste en crear el BSP (Board Support Package) en el SDK de Xilinx. Este BSP contendrá los drivers de los periféricos que se utilizaron en la etapa del diseño hardware, con esto se podrá utilizar el API y los drivers para la configuración del procesador.

La aplicación se creará en KEIL utilizando el BSP que se generó anteriormente en el SDK. Aquí se incluirán todos los drivers que se crearon anteriormente en el SDK, y que permitirán hacer el desarrollo de la aplicación. La forma de hacerlo se explica también en el manual de usuario. Al construir la aplicación se crearán unos ficheros muy importantes que son el ELF y el HEX, estos como se ha visto en los pasos del manual de usuario son los necesarios junto el fichero MMI, se creará un fichero BIT que el que programará la FPGA. Además, se puede generar un fichero MCS que es el que se descarga dentro de la memoria flash como se ha explicado.

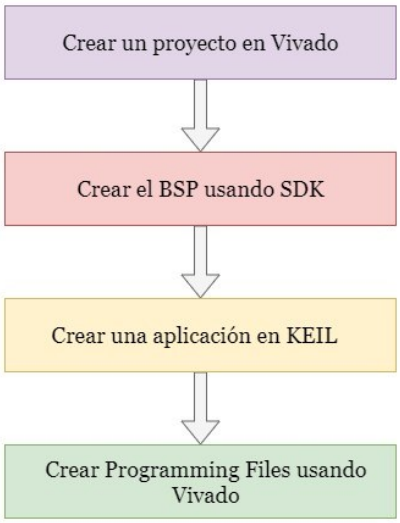


Figura 47. Diagrama de flujos para realizar la implementación.

8.1.1.1. Configuración del procesador Cortex-M1

Para obtener el primer diseño básico que nos permita conocer el funcionamiento de este núcleo con nuestra FPGA, se parte de este diagrama de bloques que será el punto de partida para los siguientes pasos.

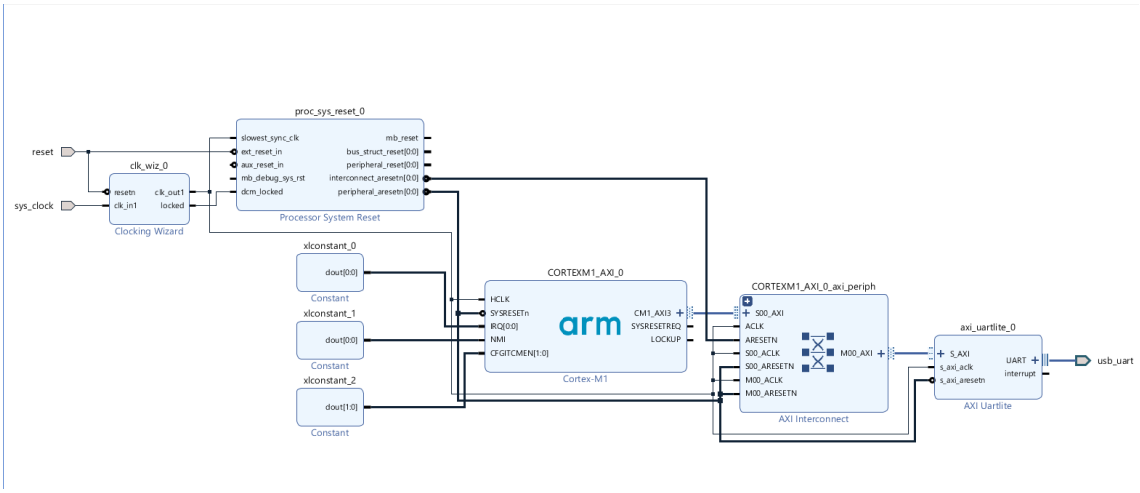


Figura 48. Diagrama de bloques inicial.

Como se ha dicho anteriormente, hay dos posibles configuraciones del Cortex-M1. La configuración que se ha elegido ha sido ***sin depuración***, las opciones para llegar a esta configuración de la IP son las mismas que se ven en las Figura 110, Figura 111 y Figura 112 del manual de memoria del primer ejemplo. El resultado de la IP configurada de esta forma queda así:

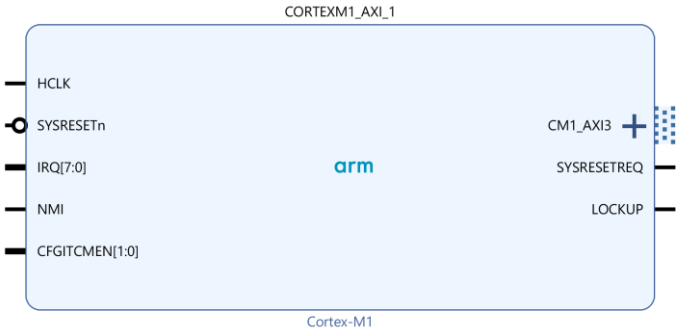


Figura 49. Cortex-M1 configurado modo sin depuración.

Según se puede ver, las únicas señales que quedan en la IP son las siguientes:

1. HCLK
2. SYSRESETn
3. IRQ[7:0]
4. NMI
5. CFGITCMEN[1:0]
6. CM1_AXI3
7. SYSRESETREQ
8. LOCKUP

Para la configuración o el manejo que se le ha dado a cada una de ellas, se explicará a continuación:

1.1. HCLK:

Esta señal como ya se ha explicado antes, es la señal de reloj del procesador. Este trabaja con una frecuencia de reloj de 100 MHz, pero para generarla se usa la IP *clock wizard*, que funciona como un PLL y de una frecuencia de entrada que utilizamos de 12 MHz de la Nexys 4 DDR, aunque la frecuencia de *sys_clk* sea de 100 MHz Single-Ended, se ha hecho de esta forma, al final se genera a la salida una frecuencia de 100 MHz y será la que se conecte a esta señal.

1.2. SYSRESETn:

Esta es la señal de reset del sistema, y solo es para el procesador. Si hubiera sido el caso de que hubiéramos configurado el procesador para depuración, se tiene una señal reset independiente. También se parte de una señal de reset externa que tiene nuestra placa, y la ponemos activa a nivel **bajo**, hay que avisarle al *clock wizard* que nuestra señal reset está activa de esta forma.

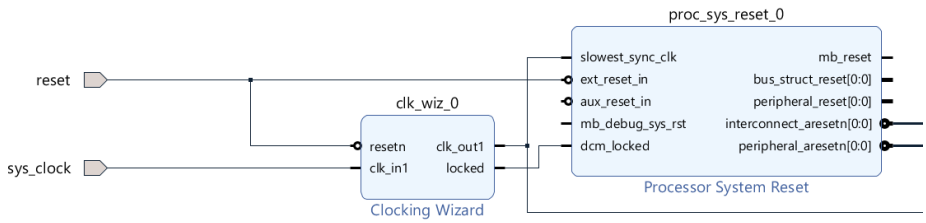


Figura 50. Bloque de reloj del procesador.

Como se ve en la anterior figura, en la configuración automática (*Run Connection Automation*) que permite Vivado se añade un bloque adicional

que se llama *Processor System Reset.* De este bloque se saca la señal que irá conectada a la señal *SYSRESETn,* estamos hablando de la señal *peripheral_aresetn[0:0]*. Como último comentario, la señal *interconnect_aresetn[0:0]* será la que se utilice en el Bus AXI interconnect.

1.3. IRQ[7:0]:
Esta señal corresponde a las señales externas de interrupción. Aquí es donde debe conectarse las interrupciones de los periféricos si es el caso. El valor de número de interrupciones se actualiza de forma automática, nosotros no utilizamos ninguna interrupción. Lo que se hizo fue conectar un bloque *Constant*, dándole un ancho de 1 bit y valor 0. Al final el valor que se queda en la configuración de número de interrupciones del Cortex-M1 es de 1.

1.4. NMI:
Esta señal está relacionada con la configuración de las interrupciones del microprocesador. Esta señal está estrechamente relacionado con el registro de control de interrupción y su valor se sitúa en el bit 31 del registro, como no se han utilizado interrupciones se le da el valor 0 usando también el bloque *Constant*. El NMI es la interrupción de más alta prioridad, si en esta posición de dicho registro se pone 1, significa que se queda pendiente del valor de NMI pero si es 0 como nuestro caso se queda sin efecto, esto es muy importante a nivel de las interrupciones.

1.5. CFGITCMEN[1:0]:
Esta señal está relacionada con la interfaz de memoria ITCM, en concreto con el registro de control auxiliar, cuyos bits (3 y 4) llamados ***ITCMLAEN*** y ***ITCMUAEN***, lo que hacen es que cuando esta *ITCMLAEN* a 1, todas las instrucciones y acceso a datos están en la región 0x00000000 hasta el tamaño máximo de la ITCM, mapeados en memoria en esta interfaz de memoria. Cuando está fijado a 0, se mapea en memoria en la interfaz AHB-Lite externa. Por otra parte, cuando *ITCMUAEN* está puesta a 1, se mapea en la región 0x10000000 de la interfaz de memoria ITCM, y cuando está a cero tiene el mismo comportamiento que *ITCMLAEN* cuando tiene el mismo valor.

1.6. SYSRESETREQ:
Esta señal lo que hace es solicitar al controlador externo del sistema que realiza el reset , en realizar un reset al núcleo y generar un reset a la depuración.
Dependiendo de las necesidades del diseño, el reset del sistema puede ser externo e independiente a SYSRESETREQ, por lo que se tiene que asegurar que esta señal no esté conectada con SYSRESETn de forma combinacional. Esta señal no se utiliza en el proyecto.

1.7. LOCKUP:
Esta señal indica que el procesador ha sido bloqueado. Este estado ocurre cuando el sistema pasa cuando ocurre una condición irrecuperable. Por

ejemplo, cuando el Cortex-M1 se le fija una prioridad -1 o -2. Para dejar este estado si la señal NMI se toma y se le fija la prioridad de -1.

En la Figura 48 se puede ver como debe de quedar el diagrama con todas las configuraciones comentadas. El mapeo en memoria de los periféricos utilizados queda de la siguiente manera, tal como se ve en la Figura 51.

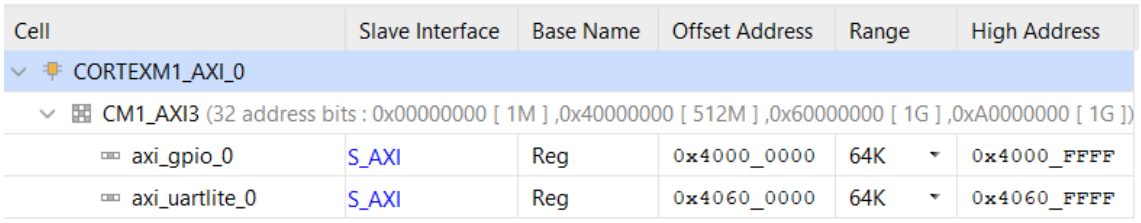


Figura 51. Mapa de direcciones del Cortex-M1.

Para saber un poco más en profundidad de los recursos que ha utilizado el Cortex-M1 de la Nexys 4 DDR, se utilizarán las herramientas que tiene disponible Vivado.

Recurso	Utilizado	Disponible	Nivel de uso%
LUT	2244	63400	3.54
LUTRAM	70	19000	0.37
FF	2249	126800	1.77
BRAM	16	135	11.85
DSP	3	240	1.25
IO	20	210	9.52
MMCM	1	6	16.67

Tabla 11. Tabla de recursos de un microprocesador con dos periféricos (GPIO y UART).

En la Tabla 11, se puede ver un resumen de los recursos utilizados en un ejemplo donde se usa un Cortex-M1 y dos periféricos que son los leds 16 bits y la UARTLite. Para comprender un poco estos resultados, recordaremos en la sección que se habló de la Nexys 4 DDR, que esta posee 15.850 *slices*, cada uno con 4 LUTs y 8FF, además de 2 F7MUX y 1 F8MUX. Esto lo que indica son los recursos utilizados ya en la etapa de ***implementation***.

La primera resección nos cuenta cuanto de estos estos elementos han sido utilizados. Hay que tener en cuenta que si se llega a aproximar al 100%, estaríamos ante un problema en el diseño porque este es demasiado grande para el ***XC7A100T***.

En 1 de cada 4 slices, los LUTs pueden ser utilizados como memoria RAM distribuida, eso es a lo que llama y vemos en la tabla ***LUTRAM***. Por otra parte, cada LUT tiene una conexión asociada con un Flip-Flop en la slice, por este motivo hay 4 parejas LUT-FF en cada slices. Según el caso, para ubicar se pueden utilizar un solo LUT de

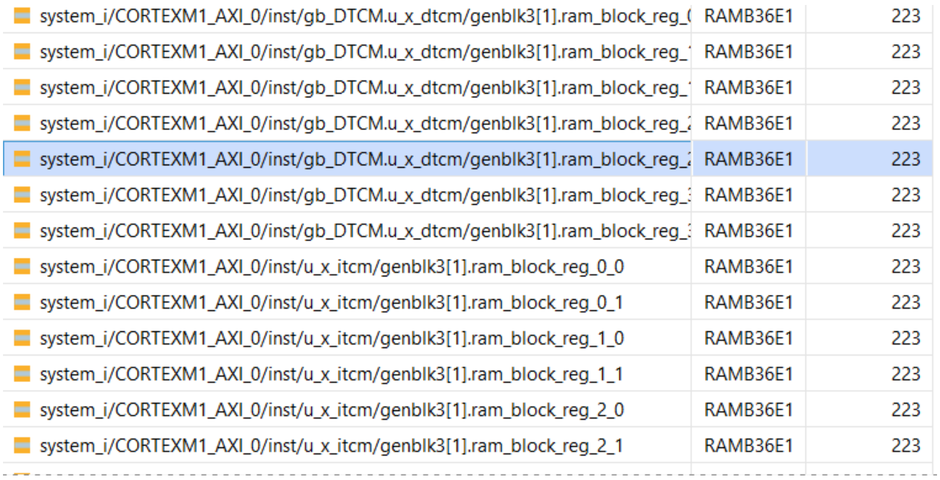
la pareja, un solo FF, o ambos. Toda esta información se obtiene a través de ***Report Utilization***.

Después de esta explicación se puede entender un poco mejor la Tabla 11, como se ve todos los recursos que aparecen ninguno supera el 100%, por lo que esto nos indica que el diseño se puede colocar perfectamente en nuestra placa.

Dentro de los recursos más utilizados han sido el bloque de memoria RAM (BRAM) que ya explicamos anteriormente. De los 135 bloques disponibles de BRAM, se han utilizado 16, esto nos indica que han sido utilizados tanto para la interfaz ITCM como para la DTCM. Si hacemos un cálculo podemos llegar a entender que podemos tener en nuestro diseño hasta 8 microprocesadores Cortex-M1, pero esto implica llegar a niveles muy altos de uso. El bloque RAM que se usa es del tipo ***RAMB36***, que tiene un almacenamiento de ***36Kb.*** Si recordamos por ejemplo la interfaz de memoria ITCM se ha configurado como **32KB**, por eso se han utilizado 8 BRAM del tipo RAMB36, que tendrán de almacenamiento 36.864 bytes en total, es suficiente para los 32.768 bytes de almacenamiento que pedimos. Con esta conclusión llegamos a que podemos tener 8 Cortex-M1, en el mismo proyecto si consideramos por ejemplo el número de BRAM del tipo RAMB36 que se puede utilizar, es decir, si aumento el tamaño de la ITCM y la DTCM se necesitarán más bloques BRAM y por tanto esto me condicionará a valorar el número de procesadores que puedo tener dentro del mismo diseño.

La frecuencia máxima de este primer diseño es de $F_{MAX} = \frac{1}{T-WNS} = \frac{1}{(10-0.894)ns} = 109.8MHz$

Si hacemos una búsqueda de los bloque BRAM utilizados, nos aparecerá la siguiente información que se puede ver en parte en la Figura 52.



system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_(	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_'	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_'	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_i	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_i	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_:	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/gb_DTCM.u_x_dtcm/genblk3[1].ram_block_reg_:	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_0_0	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_0_1	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_1_0	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_1_1	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_2_0	RAMB36E1	223
system_i/CORTEXM1_AXI_0/inst/u_x_itcm/genblk3[1].ram_block_reg_2_1	RAMB36E1	223

Figura 52. Bloques de memoria BRAM.

Otro de los recursos que tiene un valor de utilización muy alto, es el MMCM (Mixed-Mode Clock Manager). Nos indica la información que hemos utilizado un bloque de los 6 disponibles. Este bloque es el que hemos utilizado para generar la señal de reloj que conectamos al Cortex-M1 desde el Clock Wizad.

El ejemplo que se utilizó para validar el proyecto es el “Hello World” que ya se enseñó el resultado también en el manual de usuario.

8.1.1.2. Dos procesadores en el mismo proyecto

Para integrar un procesador inicial al ejemplo anterior, se han seguido las mismas configuraciones anteriores, y la forma de hacerlo también se explica en el manual de usuario.

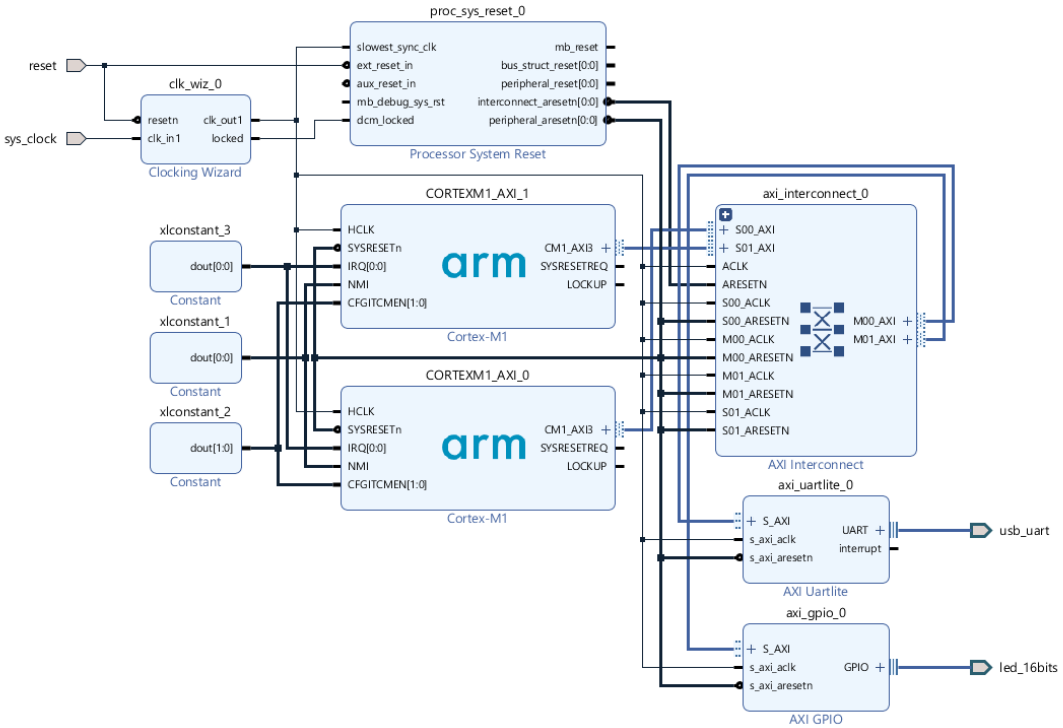


Figura 53. Diagrama de bloques con dos microprocesadores.

En esta parte se explicará también los recursos utilizados, y dos funcionalidades adicionales que consiste en añadir la memoria flash que incorpora la Nexys 4 DDR , y de la misma manera la DDR2.

El mapa de memoria de los dos microprocesadores utilizando los mismos periféricos que la última vez, se ve en la Figura 54.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM1_AXI_0					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
CORTEXM1_AXI_1					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF

Figura 54. Mapa de memoria de los dos procesadores.

La asignación de memoria de los periféricos se hace de forma automática por parte de Vivado, pero el diseñador también puede hacerlo de forma manual. Los valores que se ven son acordes al mapa de memoria que se explicó anteriormente en la parte teórica.

La utilización de recursos con dos microprocesadores, se enseña en la Tabla 12.

Recurso	Utilizado	Disponible	Nivel de uso%
LUT	4213	63400	6.65
LUTRAM	129	19000	0.68
FF	4038	126800	3.18
BRAM	32	135	23.70
DSP	6	240	2.50
IO	20	210	9.52
MMCM	1	6	16.67

Tabla 12.Tabla de recursos de dos microprocesador con dos periféricos (GPIO y UART).

Al haber añadido un Cortex-M1 más al diseño, nos aparece en la parte de la implementación los resultados de los recursos utilizados que se ven en la Tabla 12.

Como se había explicado anteriormente, y como cabría esperar el recurso que ha aumentado en su utilización es el BRAM. Al configurar de la misma forma el nuevo procesador como el primero, que se explicó anteriormente se ve que de los 16 bloques se han pasado a 32 bloques de memoria RAM, llegando a alcanzar casi el 25% de la capacidad total que ofrece la Nexys 4 DDR. Por lo demás los valores siguen con un valor esperado no superan el 10 % de la capacidad permitida. Este diseño no presentó problemas con la utilización máxima de los recursos.

Para probar que se podía trabajar con dos procesadores se ha explicado un poco en la sección de buses n-Chip que también está en la parte de implementación y resultados. Lo que se vio durante las pruebas fueron que hay que tener cuidado si ambos procesadores quieren utilizar al mismo tiempo el mismo periférico, porque el resultado que se obtiene no sea el que se espera. Por ejemplo, si es la UART le mensaje que se imprimiera era una concatenación de los mensajes que querían imprimir los dos periféricos. Por otra parte, si eran los leds que se querían encender el resultado que se enseñaba era solo uno de los dos.

A esta conclusión se ha llegado en base a los resultados que se obtuvieron en el apartado 8.1.2. Cuando se hicieron las pruebas de que un periférico se utilizaba al mismo tiempo, por ejemplo en la UART, los mensajes de dos núcleos se concatenaban, como se ve en la Figura 86. Para que se imprimieran de forma

independiente, se necesita que mientras uno imprime un mensaje por la UART otro realice otra tarea. Por este motivo, se realizan tareas intermedias para que se puedan ver los resultados enseñados. Esto es importante, ya que se necesitará tenerlo en cuenta para las próximas pruebas.

Cuando se realizó el proyecto de dos procesadores se añadieron también al diseño la memoria flash y la DDR2. A continuación se explicará cómo se implementó y que resultados se obtuvieron.

La frecuencia máxima de este diseño es de $F_{MAX} = \frac{1}{(10-1.064)ns} = 111.9MHz$

8.1.1.3. Memoria Flash

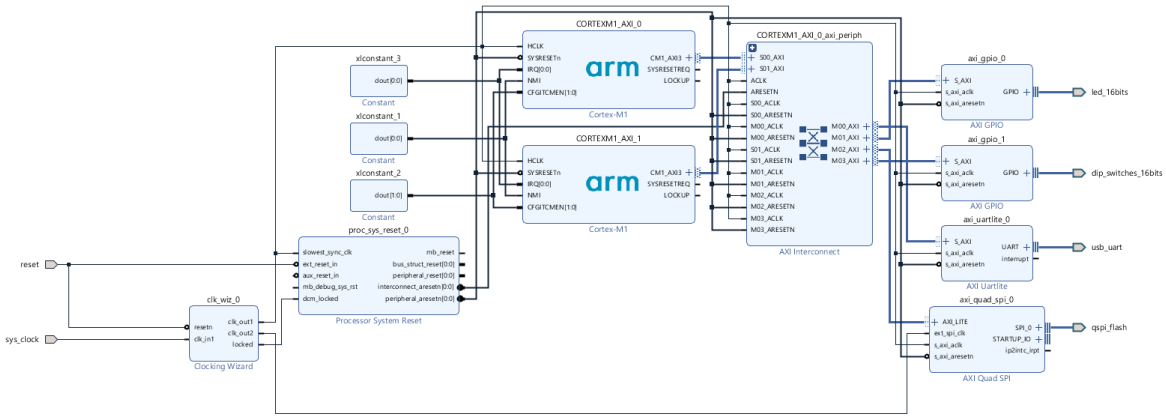


Figura 55. Diagrama de bloques con la memoria QUAD SPI Flash.

El diseño anterior con el bloque de la memoria QUAD SPI Flash es el que se puede ver en la Figura 55. El mapa de memoria con el nuevo periférico, tiene la siguiente Figura 56. Como se ve se añade en la región de los periféricos, si vemos el mapa de memoria que habíamos explicado en la parte de teoría. El bloque de la memoria flash ocupa 64Kb.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM1_AXI_0					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_gpio_1	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
CORTEXM1_AXI_1					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
axi_gpio_1	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF

Figura 56. Asignación en memoria del QSPI Flash.

Vamos a analizar los recursos utilizados que nos da la parte de implementación de Vivado, como hemos hecho anteriormente.

Recurso	Utilizado	Disponible	Nivel de uso%
LUT	4666	63400	7.36
LUTRAM	147	19000	0.77
FF	4652	126800	3.67
BRAM	32.5	135	24.07
DSP	6	240	2.50
IO	25	210	9.52
MMCM	1	6	16.67

Tabla 13. Tabla de recursos de dos microprocesador con GPIO,UART y QSPI Flash.

Según los resultados obtenidos comparando con los anteriores es que ha incrementado ligeramente el uso de los LUTs, pero en general se mantiene todo igual salvo que en el bloque de memorias ha aumentado porque se usa un bloque del tipo RAMB18 para la SPI, son sólo 18Kb. En líneas generales se mantiene igual sin pasarse de los niveles anteriores.

La frecuencia máxima de este diseño es de $F_{MAX} = \frac{1}{(10-0.794)ns} = 108.62MHz$

En el manual de usuario que se dedica a explicar cómo se hace toda la configuración necesaria para la memoria Flash, no se volverá a explicar cómo hacerlo nos centraremos más en los resultados y resaltar algunas de las configuraciones.

Como se trabaja con una memoria Quad SPI Flash, su velocidad es de (x4) por lo que se tiene que especificar que el ancho del bus debe ser de 4, esto lo hemos especificado en el fichero de *constrains.* Además, también podemos utilizar la propiedad de *CONFIG_MODE*, que tiene tres opciones de configuración de la SPI, en este caso se selecciona la SPIx4.

Al momento de generar el bitstream, como se ha explicado en el manual de usuario de utilizar el fichero ***make_prog_files.tcl*** primero se genera el fichero BIT, y a partir de este se genera el fichero MCS.

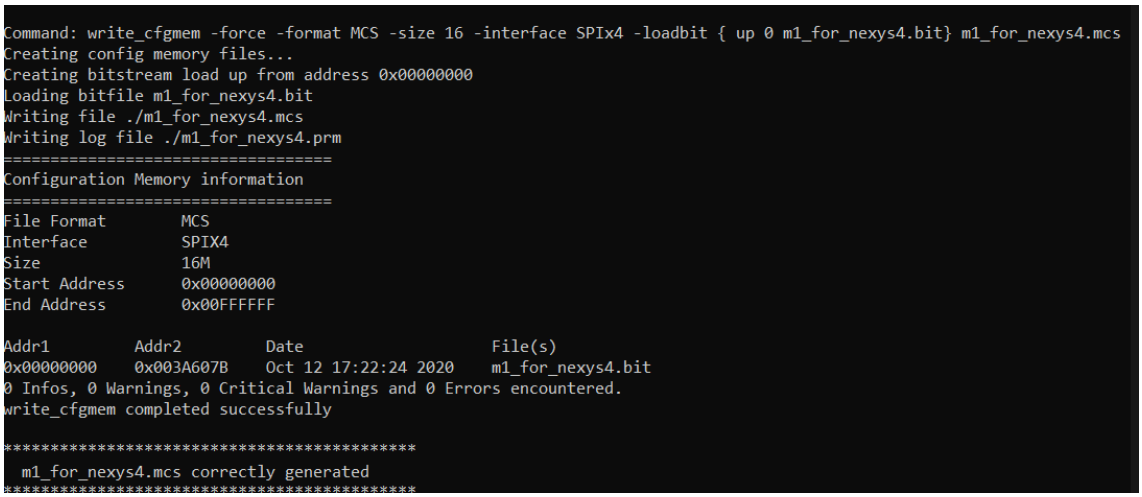


Figura 57. Información que aparece en la ventana Shell al generar el bitstream.

Como se puede ver en la Figura 57, la dirección de memoria donde se almacena el fichero MCS, se sabe que se pueden generar distintas imágenes que podemos cargar en la memoria Flash. En este caso, sólo se hizo una que se guardará en la dirección 0x00000000, la cual como se ve en la Figura 57, nos enseña el resultado de generar dicho fichero tiene un tamaño de 16MB. Lo que se ha hecho para demostrar su funcionamiento, es que un microprocesador se encarga de utilizar la UART y otro utiliza los GPIOs.

Si seguimos los pasos que se explican en el manual de memoria para generar el fichero MCS y cargarlo en la FPGA, podemos hacer la comprobación que buscamos.

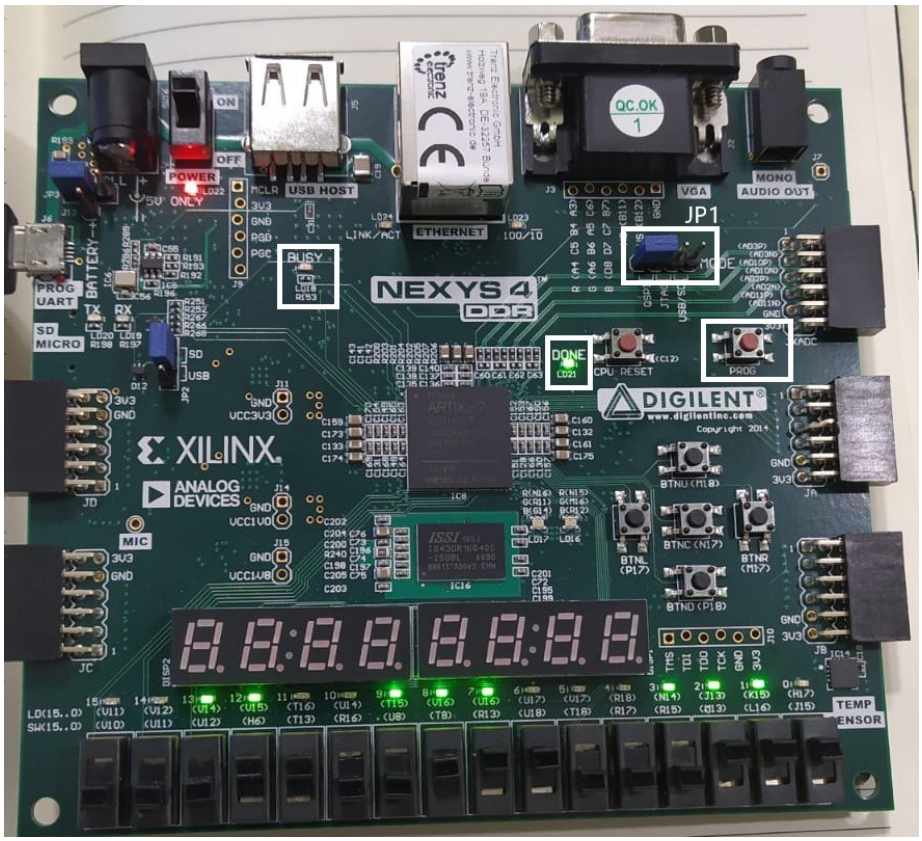


Figura 58. Resultado en la FPGA sobre la memoria Flash.

En la Figura 58, se puede ver la explicación de cómo se puede probar que se ha cargado la imagen del bitstream dentro de la FPGA. Siguiendo las indicaciones que se han mencionado, si se mantiene el jumper del JP1 en la posición del QSPI, y pulsamos el botón PROG, se podrá comprobar que se almaceno el bitstream que se ha generado.

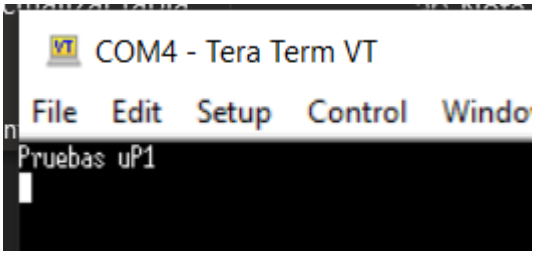


Figura 59. Resultado en la UART.

Se comprueba tanto en las Figura 58 y Figura 59, que el código utilizado en la etapa software, a partir del cual se generaron los ficheros ELF y HEX, y que posteriormente se utilizan para generar el fichero BIT, que a su vez nos ayuda a generar el fichero MCS, para cargar la imagen del bitstream en la memoria QSPI, ha funcionado.

Las pruebas para validar el resultado, ha sido desconectar la FPGA. Cuando está de nuevo encendida, si el jumper **JP1** está en la posición QSPI y se pulsa el botón **PROG,** se encienden los leds de BUSY y DONE en ese orden, y se vuelve a cargar la imagen que se utilizó, que en este caso ha sido imprimir por la UART un mensaje y el ejemplo

de los leds con los switches. Si el jumper estuviera en la posición JTAG, ya no se vería el mismo resultado.

8.1.1.4. Memoria DDR

Una vez hechas las comprobaciones con la memoria Flash, es el turno de hacerlas con la memoria DDR2. Para empezar a estudiar los efectos que ha tenido el uso de dicha memoria y los resultados que se han obtenido, vamos a analizar los recursos utilizados.

Recurso	Utilizado	Disponible	Nivel de uso%
LUT	13055	63400	20.59
LUTRAM	1261	19000	6.64
FF	13342	126800	10.52
BRAM	32.5	135	24.07
DSP	6	240	2.50
IO	71	210	33.81
MMCM	2	6	33.33
PLL	1	6	16.67

Tabla 14. Recursos utilizados incluyendo la DDR2

Comparando con los últimos resultados vemos que cuanto a los LUTs, LUTRAM y FF, han aumentado su uso considerablemente. Se ha pasado de tener un uso del 7.36% en los LUTs, para tener ahora un 20.59%, al aumentar la complejidad de este proyecto, estos LUTs o Flip-Flops no se han utilizado para crear memoria distribuida, el aumento se debe a que el diseño a pasado a ser más complejo y por lo tanto los bloques lógicos de la FPGA aumentan. Para comprobarlo si hacemos una búsqueda en los LUTs, vemos que para el bloque del controlador de memoria de la DDR2, es el bloque que más LUTs utiliza, Figura 60.

La frecuencia máxima de este diseño es de $F_{MAX} = \frac{1}{(10-0.192)ns} = 101.95MHz$

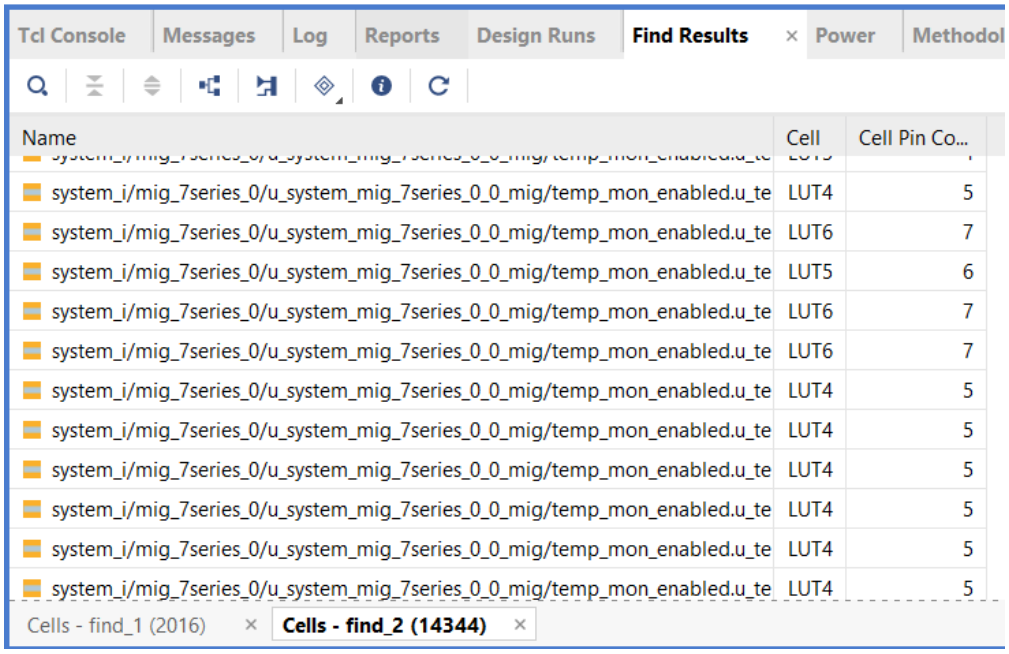


Figura 60. LUTs utilizadas en el bloque MIG.

El diagrama de bloques donde tiene al controlador de memoria de la DDR2, se puede ver en la Figura 61.

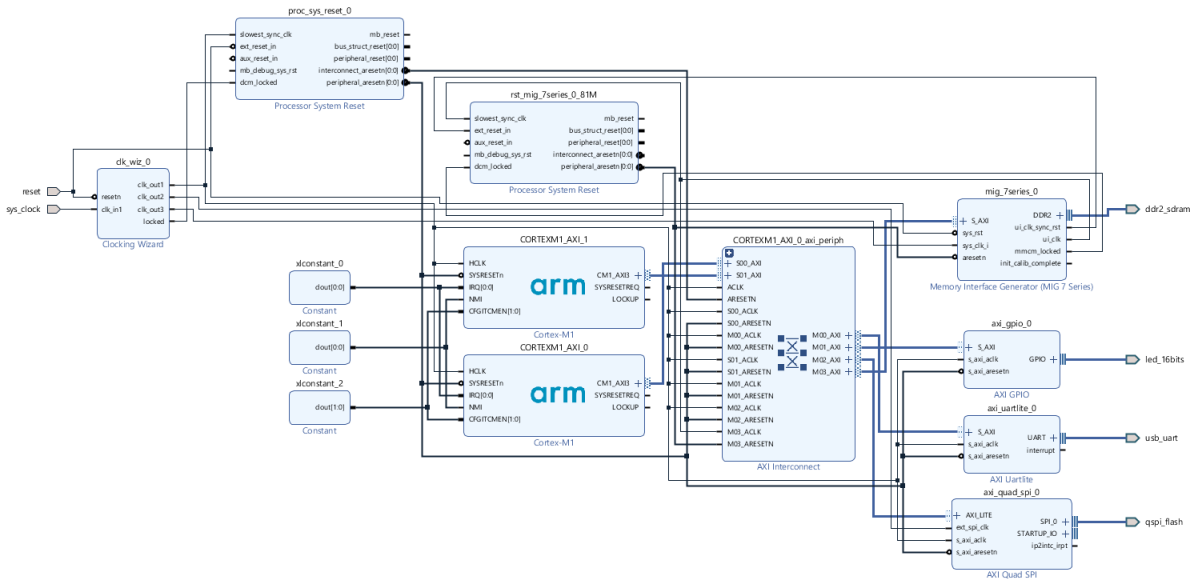


Figura 61. Diagrama de bloques con la memoria DDR2.

La configuración de esa IP ya se explicó en la parte teórica de la memoria DDR2, una parte muy importante es la asignación en memoria de esta memoria externa en el mapa de memoria de los microprocesadores. Esta información será vital para hacer la comprobación en dicha memoria.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM1_AXI_0					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
CORTEXM1_AXI_1					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

Figura 62. Asignación en memoria de MIG.

Como se puede ver en la Figura 62, la MIG se ha ubicado en la dirección 0x80000000, que si recordamos a lo que se explicó del mapa de memoria del Cortex-M1 en la parte teórica, las memorias externas como la DDR2 se tendrán que ubicar en el rango de memorias de la RAM (0x60000000 -0x9FFFFFFF). En dicha imagen se puede ver que gracias a la asignación automática que tiene vivado lo ha hecho en dicho rango, y que además el tamaño de la memoria es de 128MB, justo el tamaño que tiene la memoria DDR2 que viene incluida en la Nexys 4 DDR.

Para hacer la comprobación a nivel software, vamos a añadir en KEIL en las opciones de tarjeta, en el área de memoria lectura /escritura la dirección donde está nuestra memoria DDR2 y el tamaño que tiene, como se ve en la Figura 63. Se hace para ambos microprocesadores.

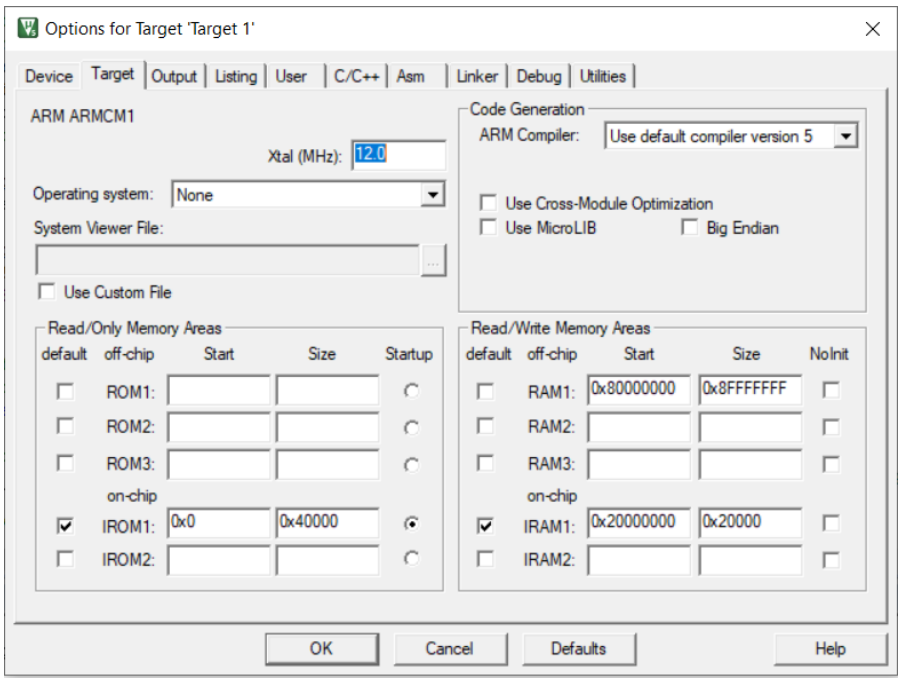


Figura 63. Añadir la ubicación de la memoria DDR2 y su tamaño.

Una vez realizado esta pequeña modificación, podemos hacer la comprobación. En el primer microprocesador vamos a escribir en la memoria DDR2, esto lo hacemos mediante la dirección de dicha memoria que la obtenemos por el fichero xparameters.h, XPAR_MIG_7SERIES_0_BASEADDR. Vamos a usar un puntero que apunta a esta dirección para guardar un array con ocho valores aleatorios, empezado por dicha posición de memoria de la DDR2. Cuando se terminé de escribir se imprime por la UART un mensaje que avise de que se ha terminado de escribir, y después se encenderán los leds.

En el segundo procesador se realizará la lectura, como habíamos visto en el mapa de memoria de ambos microprocesadores, los dos tienen la DDR2 en la misma posición con el mismo tamaño, esta asignación es muy importante porque nos índice que ambos acceden y tienen la misma dirección. En el momento de generarse los drivers, este valor también será el mismo, por lo que es muy útil a la hora de hacer la comprobación. La lectura ahora consiste en que a partir de la dirección de memoria de la DDR2, con el puntero voy extrayendo la información que guardó el primer procesador. Se tiene el mismo array con el mismo contenido que se usó para la escritura, en este caso se compara si lo que se extrae de la DDR2 es lo mismo que se tiene en el array, en el caso de que no coincida se aumenta una variable que la utilizamos como flag para indicar con un mensaje que la lectura ha dado un error.

En la Figura 64 se ve el código que se utilizó para el primero procesador y en la Figura 65 para el segundo.

```
// Specify as volatile to ensure processor reads values back from DDR2
// and not local storage
 volatile u32 *pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;

// Test data for DDR
 u32 DDR_data[8] = {0x01234567, 0x89abcdef, 0xdeadbeef, 0xfeebdaed, 0xa5f03ca5, 0x87654321, 0xfedc0ba9, 0x01020408};

 // ******************************************************
 // Test th DDR
 // ******************************************************

 // Write to DDR
 for( i=0; i< (sizeof(DDR_data)/sizeof(u32)); i++)
     *pDDRmemory++ = DDR_data[i];
 readbackError = 0;

 print("Escrita la DDR\n\r");
 // Reset the pointer
 pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;


 /****   Indicate it has finished write part****/

XGpio_SetDataDirection(&leds,1,0x00000000);
XGpio_DiscreteWrite(&leds,1,0x0000000A);
```

Figura 64. Código de ejemplo para la DDR2.

```
/*************   Test DDR2   *************/
// Reset the pointer
  pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;
// Readback


for( i=0; i< (sizeof(DDR_data)/sizeof(u32)); i++)
  {
    sprintf(debugStr,"Lectura memoria DDR2 %x\r\n",*pDDRmemory);
    print(debugStr);
    if ( *pDDRmemory++ != DDR_data[i] )
      readbackError++;

    }

  if ( readbackError )
      print( "ERROR - ddr readback corrupted.\r\n" );
  else
      print( "DDR readback correct\r\n" );
```

Figura 65. Código utilizado para la DDR2.

Al programar la FPGA con el fichero BIT, el resultado que se obtiene es el siguiente:

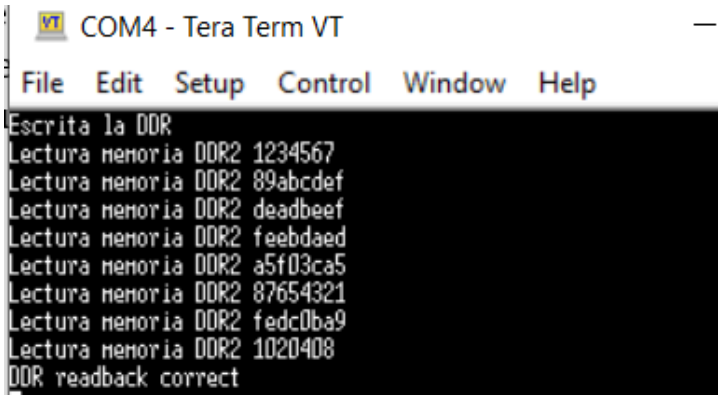


Figura 66. Resultado correcto de la comprobación de la DDR2.

Si nos fijamos en la Figura 64, el contenido del array coincide con lo que se extrae del puntero que señala la dirección de la DDR2. Se comprueba que se realiza la escritura y la lectura es correcta.

Para comprobar que si los dos arrays utilizados son distintos, la lectura tendría que dar un error. En el array del segundo microprocesador se hace un cambio:

```
 // Test data for DDR
u32 DDR_data[8] = {0x01020408,0x01234567, 0x89abcdef, 0xdeadbeef, 0xfeebdaed, 0xa5f03ca5, 0x87654321, 0xfedc0ba9 };
```

Al generar nuevamente el bitstream, el resultado que nos aparece en Tera Term es el siguiente, la lectura que se hace es con el array sin modificar del primer procesador, pero el del segundo procesador si ha sido modificado por lo tanto aparece el error “esperado”.

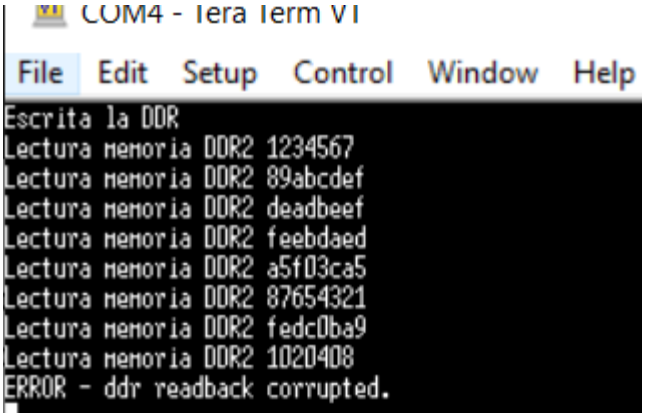


Figura 67. Resultado incorrecto en la lectura de la DDR2.

Por último se quiere explicar el uso de los leds, como la interconexión está basada en el bus AXI, para poder arbitrar las tareas que hacen los dos Cortex-M1 con los periféricos, la memoria y que sea de forma ordenada. Tiene que haber tareas intermedias durante la escritura y la lectura para que los procesadores les dé tiempo en realizar dichas acciones. Si no se llegarán a poner tareas intermedias, la escritura no le daría suficientemente tiempo en terminarse y cuando se quiere hacer la lectura nos generará un error. Por este motivo, se ha implementado de esta manera, para que la comprobación salga como se espera.

Dado a que los núcleos intentan acceder al mismo periférico, para que el resultado que nos salga coherente con lo que buscamos. Es necesario asignar tareas intermedias a los núcleos para que no accedan al mismo recurso y se presenten problemas como el que se ve en Figura 68. Ambos núcleos intentan acceder al mismo tiempo a la UART, el mensaje que se obtiene es una concatenación de los caracteres de ambos.

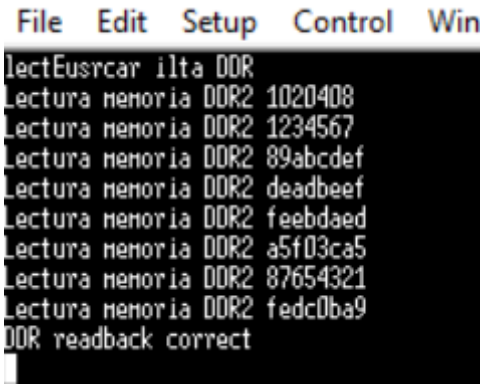


Figura 68. Error acceso al mismo tiempo de la UART.

8.1.1.5. Sistema con cuatro procesadores

En el Anexo 9, se puede ver la figura final de forma más amplia con la integración de cuatro procesadores en total, en la Figura 69 se enseña hacernos una idea.

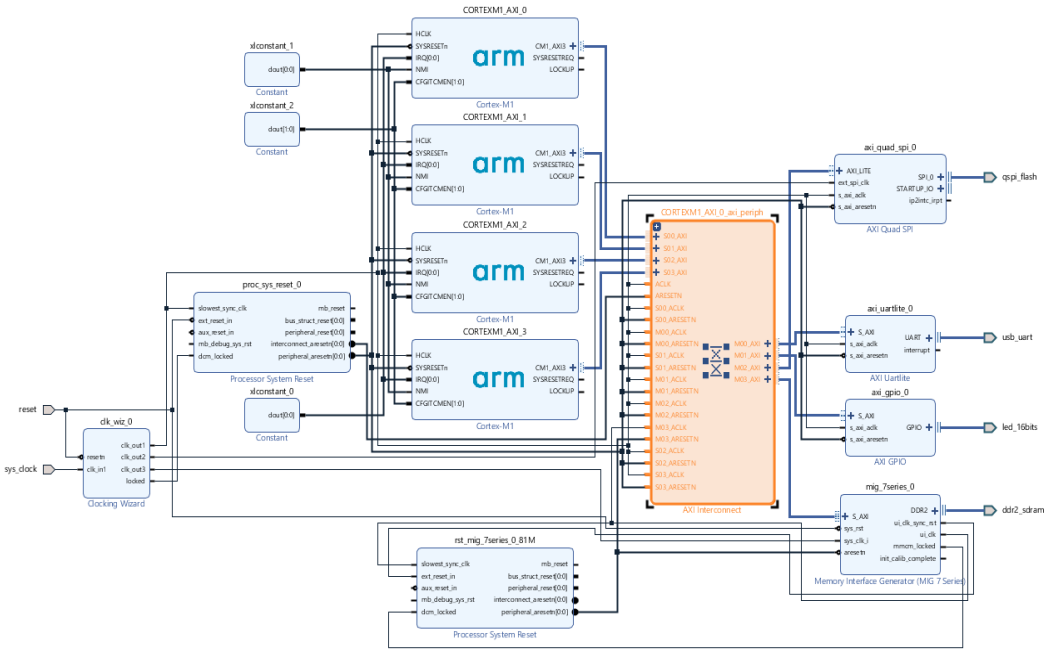


Figura 69. Sistema con cuatro microprocesadores.

La asignación en memoria de los perifericos y las memorias en cada uno de los Cortex-M1 queda de la siguiente forma:

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM1_AXI_0					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
CORTEXM1_AXI_1					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
CORTEXM1_AXI_2					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF
CORTEXM1_AXI_3					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_quad_spi_0	AXI_LITE	Reg	0x44A0_0000	64K	0x44A0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	128M	0x87FF_FFFF

Figura 70. Mapa de memoria de los microprocesadores.

Como se ha visto en los casos anteriores, la asignación automática pone la misma dirección de cada periférico y memoria, en cada microprocesador. Como se había visto en el caso anterior, cuando se comprobó el funcionamiento de la DDR2 se vio que tener la misma dirección de memoria favorecía la comprobación de dicha memoria, para este caso también se hará una comprobación con la DDR2 en los cuatro Cortex-M1.

Ahora al contar con tantos microprocesadores, será necesario establecer una prioridad entre ellos, para ello como se explicará en la parte de ***Buses on-chip 8.1.2***, se tendrá que fijar una prioridad, la que se le asigne el valor más alto será la más prioritaria.

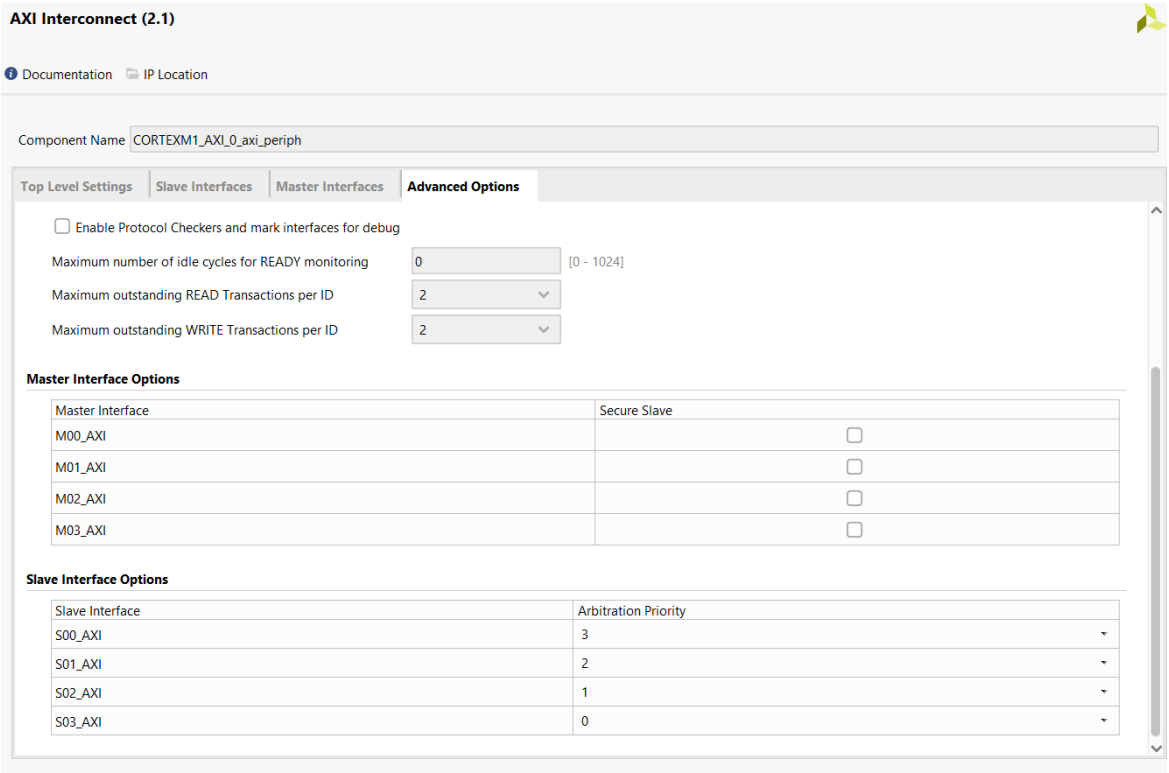


Figura 71. AXI interconnect, asignación de prioridades.

Más adelante se verá cómo afecta esta prioridad y las razones de porqué se hizo esta asignación. Para adelantar, al que se le da el número más alto será el más prioritario. Nos interesa que se asigne de esta forma para que la escritura en la DDR2 la realice primero el primer microprocesador, es decir, la tarea de escritura sea la primera en hacer para que los demás núcleos puedan leer el contenido que se hizo en dicha escritura.

En cuanto a los recursos que se utilizaron para esta implementación los vemos resumidos en la siguiente Tabla 15:

Recurso	Utilizado	Disponible	Nivel de uso%
LUT	17665	63400	27.86
LUTRAM	1377	19000	7.25
FF	17484	126800	13.80
BRAM	64.50	135	47.78
DSP	12	240	5.00
IO	71	210	33.81
MMCM	2	6	33.33
PLL	1	6	16.67

Tabla 15. Recursos utilizados por 4 up.

Si recordamos lo que teníamos en la anterior Tabla 14, se puede ver que tanto los recursos como los LUTs, LUTRAM y FF, no han aumentado significativamente ya que

la memoria DDR2 es compartida entre todos los procesadores. Lo que sí ha aumentado son los bloques de memoria BRAM, de 32.5 bloques se ha pasado a 64,5, los dos nuevos microprocesadores también han sido configurados de la misma forma que los anteriores, por lo tanto el tamaño de las dos interfaces de memoria será también de 32 KB. Como habíamos explicado anteriormente, para poder lograr tener este tamaño de memoria con los bloques BRAM de la FPGA, se necesitaban ocho bloques para cada interfaz de memoria (ITCM y DTCM). Como se ve en la Tabla 15, se indica que el número de bloques que se tienen en este diseño son de 64.50, llegando casi a la mitad del 50% del total disponible de la FPGA. Esto nos hace reflexionar como ya lo hicimos en su momento, es que sería posible añadir otros procesadores más sin sobre pasar el tamaño de las memorias de los microprocesadores.

La frecuencia máxima de este diseño es de $F_{MAX} = \frac{1}{(10-0.326)ns} = 103.36MHz$

Para probar el funcionamiento en conjunto de este nuevo sistema que hemos implementado, lo que se ha hecho es que uno de los microprocesadores se encargará de escribir una lista de valores en un array de valores aleatorios que se almacenarán desde la dirección de memoria de la DDR2, el mismo procedimiento que habíamos hecho en el caso anterior.

Antes, la lectura de dicha escritura en la DDR2 lo hacia un solo núcleo. Ahora, la novedad consiste en que los otros tres procesadores se encargarán de leer sólo una parte determinada, en este caso serán ocho elementos del array original, que tiene en total veinticuatro elementos. La configuración en KEIL se hizo igual como en la explicación de la memoria DDR2.

Los resultados los vemos en la UART y los leds, para ello vamos a enseñar capturas de la escritura y la lectura de la DDR2.

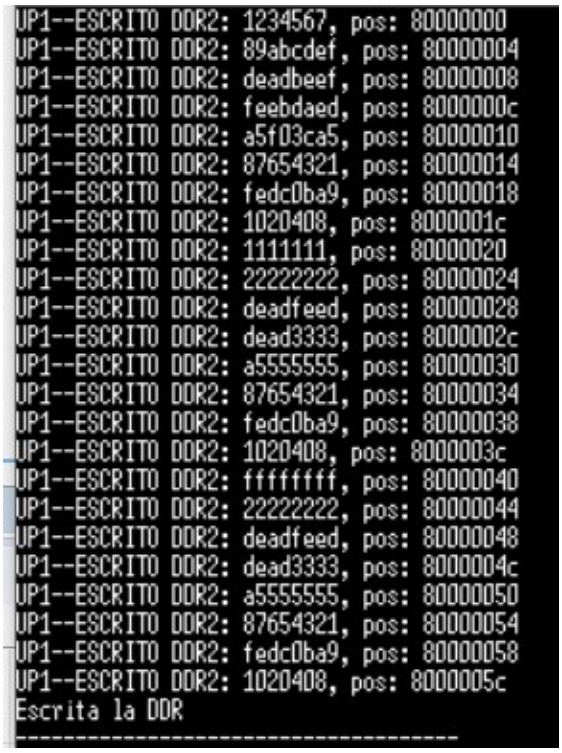


Figura 72. Resultado de escribir en la DDR2 del primer microprocesador.

Figura 73. Primer microprocesador.

Como se ve en las figuras anteriores, la escritura la hace el primero microprocesador y almacena los valores aleatorios de un array de tamaño 24. La primera dirección es la correspondiente a la que encontramos en *xparameters.h* asociada a la memoria DDR2, la dirección 0x80000000 y la última posición corresponde a 0x8000005c. Para indicar que ha terminado la lectura se encienden los leds con el valor 0x0000000A,Figura 73. Esto se utiliza para que el diseñador que dispone de la FPGA pueda saber que núcleo se ha ejecutado Figura 73. Primer microprocesador.. En cada núcleo se utiliza un valor arbitrario en los leds que nos indicarán además que la UART que se ha realizado la lectura.

La prioridad más alta se le asigno a este, ya que es el encargado de escribir y se espera que complete el primero la escritura para poder hacer a continuación la escritura, las demás asignaciones de prioridad se hicieron en el orden de que realizarían las lecturas, como se explicará ahora.

Para la lectura de los primeros 8 valores del array que se escribió, se utiliza el segundo procesador. Este los identifica con un array que tiene esos valores que buscamos, después de una comparación con los valores escritos, los imprimimos por la UART, también se avisa de que se ha terminado la lectura con el encendido de los leds, que toman el valor de 0x00001111.

```
UP2--Lectura memoria DDR2: 1234567
UP2--Lectura memoria DDR2: 89abcdef
UP2--Lectura memoria DDR2: deadbeef
UP2--Lectura memoria DDR2: feebdaed
UP2--Lectura memoria DDR2: a5f03ca5
UP2--Lectura memoria DDR2: 87654321
UP2--Lectura memoria DDR2: fedc0ba9
UP2--Lectura memoria DDR2: 1020408
```

Figura 74. Lectura del segundo microprocesador.

Figura 75. Segundo microprocesador.

La lectura de la primera parte, como se ve en las figuras anteriores se hace según lo esperado, aparecen los ocho primero valores del array.

```
UP3--Lectura memoria DDR2: 1111111
UP3--Lectura memoria DDR2: 22222222
UP3--Lectura memoria DDR2: deadfeed
UP3--Lectura memoria DDR2: dead3333
UP3--Lectura memoria DDR2: a5555555
UP3--Lectura memoria DDR2: 87654321
UP3--Lectura memoria DDR2: fedc0ba9
UP3--Lectura memoria DDR2: 1020408
```

Figura 76. Lectura del tercer microprocesador.

Figura 77. Tercer microprocesador.

```
UP4--Lectura memoria DDR2: ffffffff
UP4--Lectura memoria DDR2: 22222222
UP4--Lectura memoria DDR2: deadfeed
UP4--Lectura memoria DDR2: dead3333
UP4--Lectura memoria DDR2: a5555555
UP4--Lectura memoria DDR2: 87654321
UP4--Lectura memoria DDR2: fedc0ba9
UP4--Lectura memoria DDR2: 1020408
```

Figura 78.Lectura del último microprocesador.

Figura 79. Cuarto microprocesador.

En general las tres lecturas se hicieron de la forma esperada, cada una leyó los ocho valores del array general que les correspondía.

Para realizar la escritura y lectura vamos a presentar de forma sencilla como se implementó el código en KEIL.

Lo primero consistió en definir el array con los valores aleatorios que se querían escribir[2].

```
// Test data for DDR
 u32 DDR_data[24] = {0x01234567, 0x89abcdef, 0xdeadbeef, 0xfeebdaed, 0xa5f03ca5, 0x87654321, 0xfedc0ba9, 0x01020408,
                     0x01111111, 0x22222222, 0xdeadfeed, 0xdead3333,0xa5555555, 0x87654321, 0xfedc0ba9, 0x01020408,
                    0x0fffffff, 0x22222222, 0xdeadfeed, 0xdead3333, 0xa5555555, 0x87654321, 0xfedc0ba9, 0x01020408};


        // ******************************************************
        // Test th DDR
        // ******************************************************

        // Write to DDR
        for( i=0; i< (sizeof(DDR_data)/sizeof(u32)); i++){
            *pDDRmemory++ = DDR_data[i];

        }
        //Read what were keep in memory
        pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;
        for( i=0; i< (sizeof(DDR_data)/sizeof(u32)); i++){

            sprintf(debugStr,"UP1--ESCRITO DDR2: %x, pos: %x\r\n",*pDDRmemory,pDDRmemory);
            print(debugStr);
            *pDDRmemory++;
        }

        readbackError = 0;

        print("Escrita la DDR\n\r");
        print("-----------------------------------\n\r");
        // Reset the pointer
        pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;


        /****   Indicate it has finished write part****/

      XGpio_SetDataDirection(&leds,1,0x00000000);
      XGpio_DiscreteWrite(&leds,1,0x0000000A);
```

Figura 80. Escritura en la DDR2,UP1.

Para la lectura, se utilizó de forma adicional un array que contenía solo el trozo que se quería leer y uno auxiliar donde guardar los resultados cuando se hacía la comparación.

```
u32 DDR_data[8] = {0x01234567, 0x89abcdef, 0xdeadbeef, 0xfeebdaed, 0xa5f03ca5, 0x87654321, 0xfedc0ba9, 0x01020408};
u32 DDR_data_original[24] = {0x01234567, 0x89abcdef, 0xdeadbeef, 0xfeebdaed, 0xa5f03ca5, 0x87654321, 0xfedc0ba9, 0x01020408,
u32 DDR_data_READ[8];
```

[2] El contenido es el mismo que se ve representado en la Figura 72.

El código que se utilizó es el mismo para el resto. Por lo tanto, sólo se explicará uno de ellos.

```
// Reset the pointer
  pDDRmemory = (u32 *)XPAR_MIG_7SERIES_0_BASEADDR;
// Readback

  for(int j = 0; i < 80000000; i++);

  for( i=0; i< (sizeof(DDR_data_original)/sizeof(u32)); i++)
    {

      if ( *pDDRmemory = DDR_data[i] ){
          DDR_data_READ[i]=*pDDRmemory;
          *pDDRmemory++;
      }

    }
    for( i=0; i< (sizeof(DDR_data)/sizeof(u32)); i++)
    {
        sprintf(debugStr,"UP2--Lectura memoria DDR2: %x\r\n",DDR_data_READ[i]);
        print(debugStr);
    }

    print("-------------------------------------\n\r");


  XGpio_SetDataDirection(&leds,1,0x00000000);
  XGpio_DiscreteWrite(&leds,1,0x00001111);
```

Figura 81. Lectura de la DDR2.

Se resetea la primera dirección de la DDR2, es decir, se pone en 0x80000000 al puntero que va recorriendo la DDR2. Posteriormente se espera un tiempo con un bucle for, de forma que simula una tarea intermedia para dar tiempo a la UART, como ya se explicó anteriormente. Se realiza una comparación entre lo que está en la DDR2 con el contenido que queremos leer, el contenido que está almacenado en la DDR2 coincide con el trozo que queremos leer para cada núcleo y se guarda ese trozo de elementos guardados en un array para poder imprimirlo posteriormente y enseñarlo en Tera Term.

8.1.2. Buses On-chip:

La arquitectura de bus que se ha elegido, como ya se explicaba anteriormente es la AMBA. En concreto la AXI, ya que el microprocesador Cortex-M1 aunque utiliza el protocolo AHB, tiene incluido internamente un puente "bridge" entre AHB y AXI.

Dentro de nuestro proyecto utilizamos, lo que se conoce como Infraestructura IP, disponible en las herramientas de Vivado, proporciona funciones como enrutamiento y verificación de datos. Vamos a ver en más detalle este bloque.

La IP AXI Interconnect, lo que hace principalmente es conectar el mapa de memoria de los masters y los esclavos. Esta IP permite cualquier mezcla entre los bloques master y esclavo para su conexión, estos bloques pueden ser distintos tanto en el ancho de datos, frecuencia de reloj y el protocolo AXI que utilizan (AXI3,AXI4 y AXI4-Lite). La ventaja que introduce es que si se produce el caso de algunas de estas diferencias, la IP automáticamente se encarga de crear la infraestructura necesaria para la conexión y se realice la conversión necesaria.

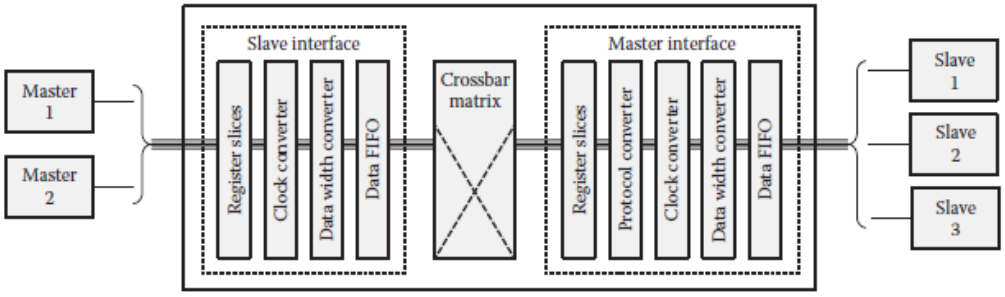


Figura 82. Diagrama de bloques del AXI Interconnect.

Esta infraestructura de conexión cuenta con varios modelos para ser usados configurados según las necesidades del diseñador. Dado que el propósito de este proyecto, es implementar más de un procesador Cortex-M1 en la FPGA, vamos a ver un ejemplo de cómo funcionarían dos procesadores maestro usando el AXI Interconnect y cómo funcionarían usando módulos separados como el *crossbar* o el *smartconnect.*

8.1.2.1. Modulo AXI Crossbar

Vivado dispone de una IP llamada AXI Crossbar. Sin embargo, el AXI Interconnect contiene dentro ya una instancia del AXI Crossbar, nos interesa conocer su comportamiento por separado.

El AXI Interconnect dispone tanto de dieciséis espacios entre masters y esclavos para aceptar transacciones entre ellos. El modo de conexión es Shared-Address. Aunque permite configurar de forma automática los protocolos AXI, soportando tanto AXI3, AXI4 y AXI-Lite. El protocolo que tiene que tener tanto el master como el esclavo debe ser el mismo, por lo que surge un inconveniente entre compatibilidades, y el diseñador deberá utilizar otro bloque que resuelva esta cuestión, en nuestro caso lo que se utilizó por el lado de los procesadores fueron unos bloques que se encargan de convertir los protocolos, llamado *AXI Protocol Converter*.

Entre otras características más destacables de esta IP tenemos:

- El modo de acceso compartido está optimizado en área.
- Admite múltiples transacciones pendientes.
- La arbitración que utiliza es de prioridad fija y round-robin. Cuenta con hasta dieciséis niveles configurables de prioridad.

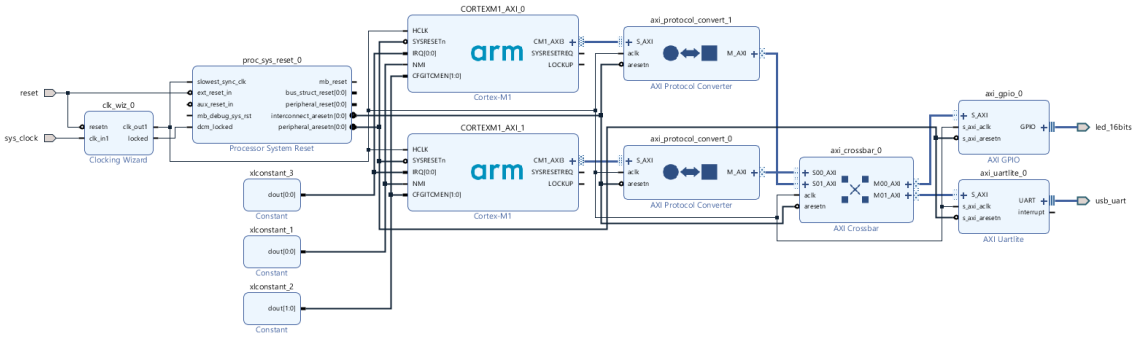


Figura 83. Diagrama de bloques con el AXI Crossbar y dos procesadores.

Se ha dejado que las configuraciones se realicen automáticamente, entre los dos procesadores y los dos periféricos que se ha añadido.

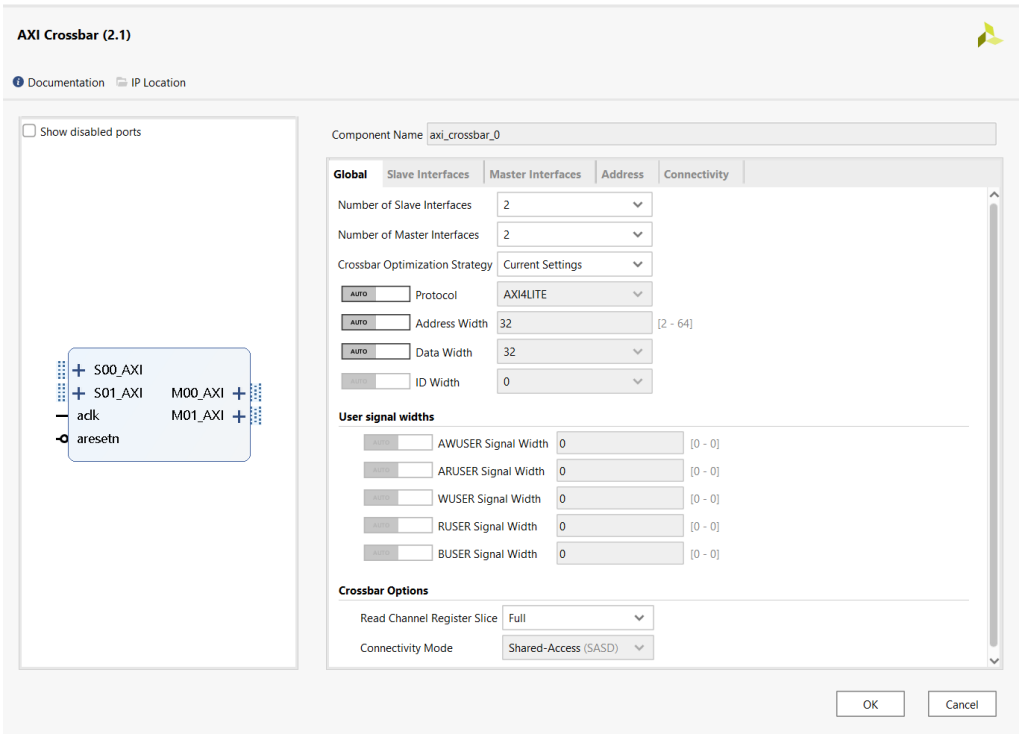


Figura 84. Configuración del AXI Crossbar.

- Si se hace la prueba de que ambos procesadores utilicen al mismo tiempo el mismo recurso, por ejemplo la UART. El resultado que se obtiene es el siguiente:

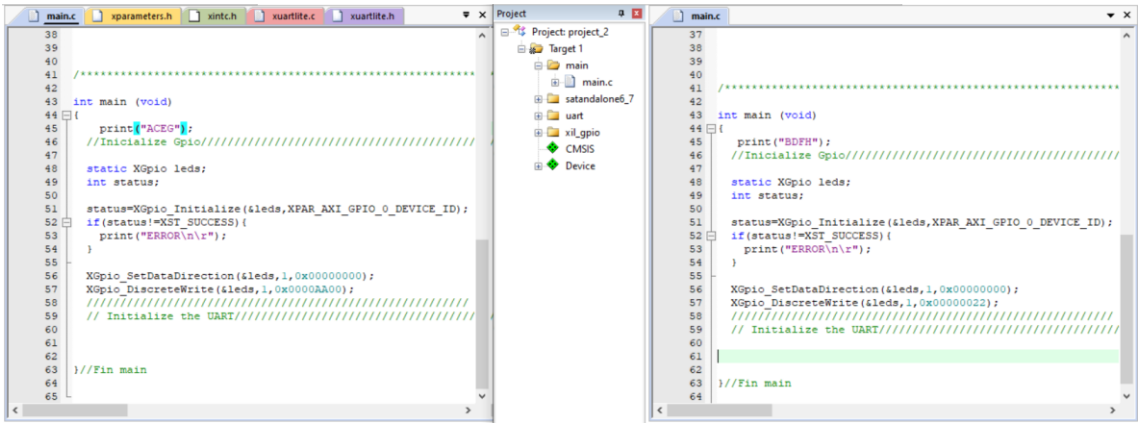


Figura 85. Código del procesador 1 a la izquierda y del procesador 2 a la derecha.

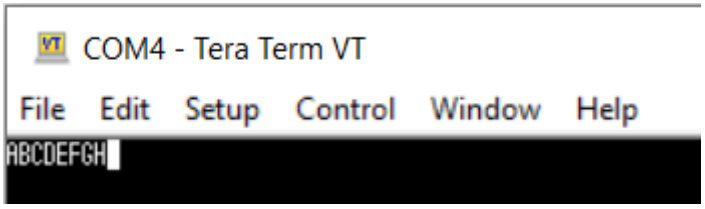


Figura 86. Resultado de utilizar al mismo tiempo la UART.

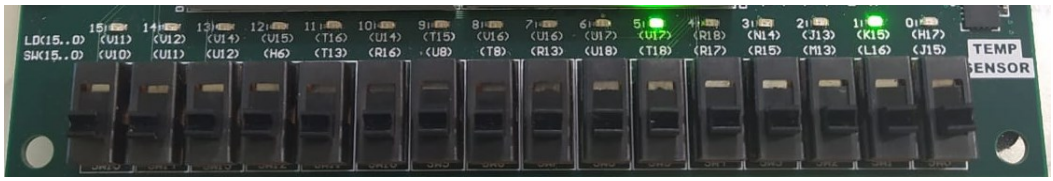


Figura 87. Resultado de los leds usados al mismo tiempo.

Como se ve en las Figura 86 y Figura 87, el resultado de utilizar al mismo tiempo el mismo periférico es que ambos procesadores van imprimiendo un carácter por turno hasta que termina el último. En la configuración del AXI Crossbar, la prioridad de round-robin está de la siguiente forma: para el procesador 1 se tiene la primera prioridad 0 y para el segundo se tiene un nivel de prioridad 1. Por ese motivo el primero en empezar es el código que se ejecuta del segundo microprocesador.

- Si se hace la prueba de que cada microprocesador, utiliza un periférico de forma independiente, el resultado es el siguiente:

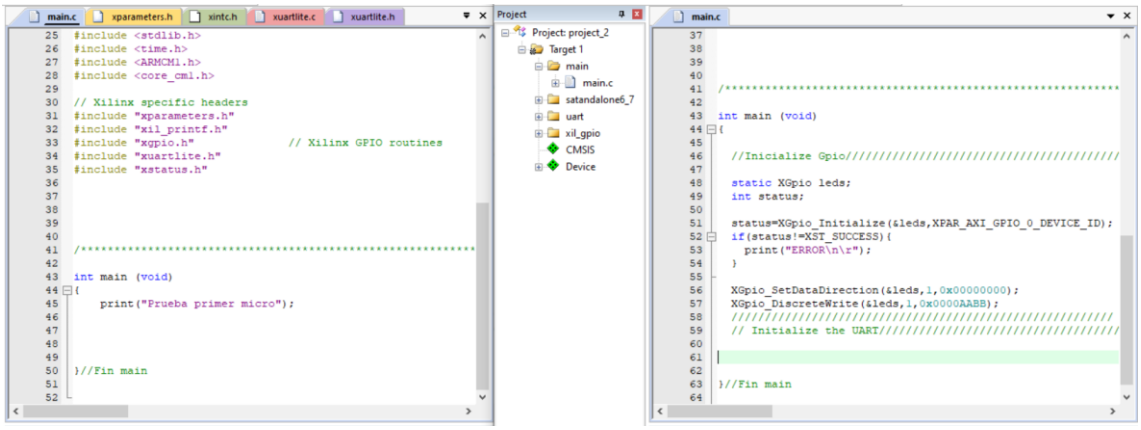


Figura 88.Procesador 1 a la izquierda y procesador 2 a la derecha.

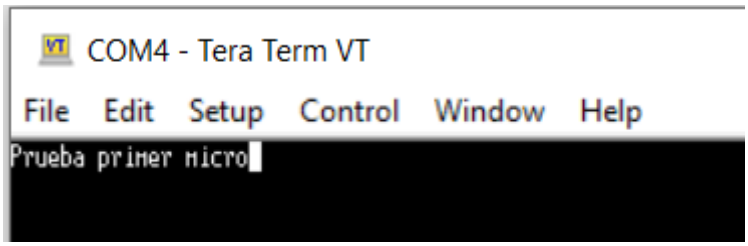


Figura 89. Resultado en la UART.

Figura 90. Resultado en los leds.

Como se puede ver en las Figura 89 y Figura 90, el resultado que se obtiene es que cada procesador ejecuta en cada periférico una tarea y está se completa satisfactoriamente.

8.1.2.2. Modulo AXI SmartConnect

El bloque IP AXI SmartConnect conecta más de un maestro mapeado en memoria con otro esclavo también mapeado en memoria. Este bloque tiende ser más compacto en los diseños que el AXI Interconnect v2. También permite utilizar interfaces como AXI3, AXI4 y AXI4-Lite.

Para configurar el AXI SmartConnect, no ofrece muchas opciones para configurar como si lo hacia el AXI Crossbar, sólo tiene disponibles las que aparecen en la Figura 91.

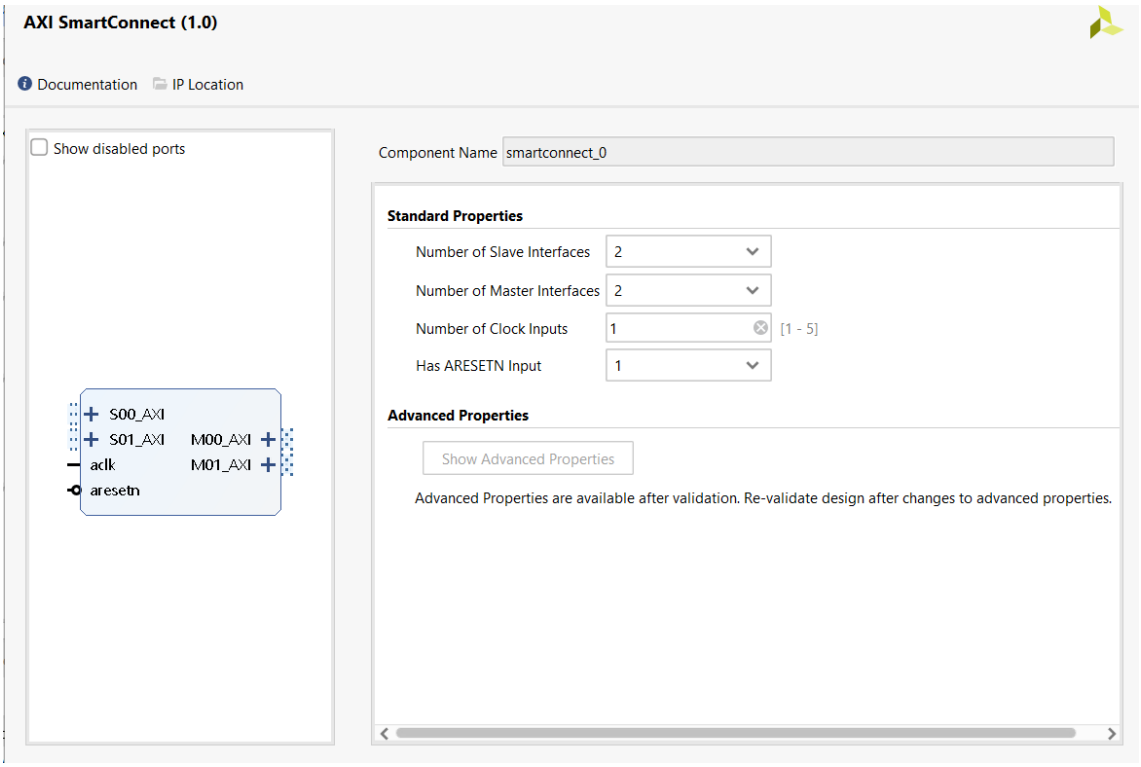


Figura 91. Configuración del AXI SmartConnect

Entre otras de sus características tenemos que:

- Se pueden utilizar hasta 16 master y esclavos en una instancia.
- Si las transacciones el ancho de datos es diferente, se hace una conversión automáticamente con esta IP.
- La arbitración: si cada canal tiene un destino independiente. La transferencia de dos o más puntos tanto de origen como destino, esa transferencia puede ocurrir concurrentemente para cualquiera de los canales.

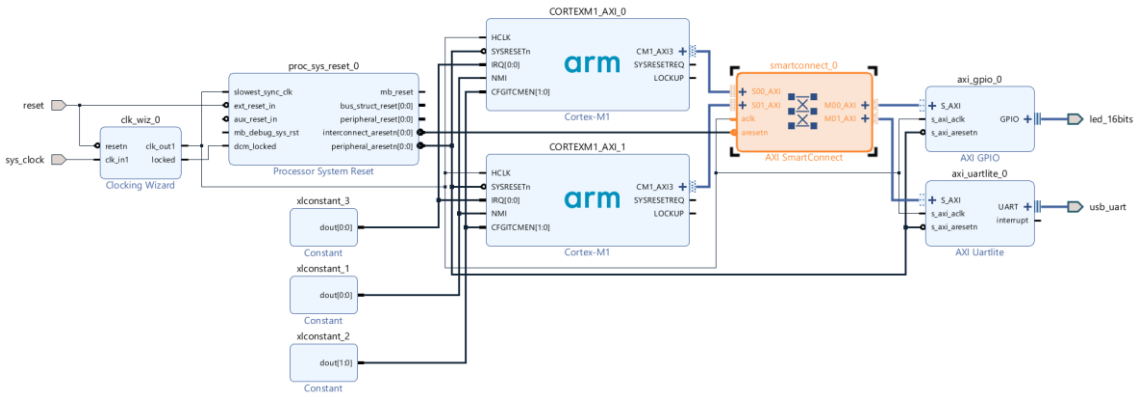


Figura 92. Diagrama de bloques con el AXI SmartConnect

- Cuando se hace la prueba de que cada procesador acceden al mismo recurso, se obtiene el mismo resultado que en la Figura 86. Lo mismo pasa cuando se hace de forma separada, se obtiene el mismo resultado como en las Figura 89 y Figura 92.

8.1.3. Modulo AXI Interconnect

El módulo AXI es el que se ha utilizado para la implementación. La razón de porque se seleccionó es que integra la mayoría de las funcionalidades que se han explicado en esta parte. Por lo tanto, la mayoría de las configuraciones necesarias las hace de forma automática y cuando se necesitó asignar prioridad, en el momento de utilizar 4 microprocesadores, sólo fue necesario acceder a las configuraciones avanzadas y hacerlo. Es la IP más completa para nuestra implementación, porque también permite y es compatible con más de un protocolo o versión del bus AXI.

9.1. Conclusión

El desarrollo de este proyecto nos ha permitido conocer como es la metodología de trabajo con microprocesadores open-source. Hemos visto que dentro de los dispositivos lógicos configurables como las FPGAs, es posible trabajar con dos tipos de núcleos (hard o soft). En este trabajo se ha profundizado en los *soft processor,* en concreto en el Cortex-M1 que es proporcionado por ARM desde su plataforma DesignStart FPGA. Ésta ofrece la IP de forma gratuita, sin condiciones de licencia para que pueda usarse fácilmente en proyectos.

Dentro del paquete que ofrece ARM vienen integrado dos diseños, para utilizarse en dos FPGAs en concreto (Arty A7 y Arty S7). Nosotros hemos podido conseguirlo en otra FPGA, que también pertenece a la familia pero que es una de las más grandes de la familia en cuanto a recursos.

Para poder implementar el primer diseño con esta IP dentro de nuestra FPGA, se tuvo que pasar por una fase de estudio del microprocesador y comprender como es el diagrama de flujo que se tiene que seguir para poder implementar una aplicación desde el nivel hardware hasta el nivel software. Ya que su línea de trabajo no es la misma a la que estamos acostumbrados, como por ejemplo cuando se trabaja con soft processor como *MicroBlaze* o SoCs como *Zynq*. Para poder trabajar con el Cortex-M1, se necesitaron herramientas como Vivado, SDK y KEIL. En esta primera etapa se vió que es posible trabajar con los microprocesadores de diferentes maneras para poder al final programar la FPGA y ver los resultados.

Durante el continuo aprendizaje se apreció a más bajo nivel, como trabaja la memoria de la FPGA cuando se implementan este tipo de microprocesadores. Se conoce que al menos las FPGAs cuentan con dos bloques de memoria, una es la memoria distribuida y la otra son los bloques de memoria RAM (BRAM). En este aspecto se vio que podría ser un factor limitante de cara a añadir más procesadores al sistema, ya que si se configuraba el tamaño de las interfaces de memoria en un tamaño mayor, consumiría más bloques BRAM de la FPGA y no podrían integrarse más microprocesadores.

Para poder generar el fichero BIT y programar la FGPA, era necesario durante la etapa de implementación de Vivado era necesario generar un fichero MMI que nos ayuda a describir el diseño de los BRAM utilizados por las interfaces de memoria del microprocesador (ITCM y DTCM). Se comprobó en los siguientes diseños, cuando se les añadía un poco más de complejidad que estos bloques de memoria no eran estáticos, se debía de regenerar dicho fichero para poder al final generar el bitstream.

Para generar los drivers de los periféricos utilizados, se partía desde la definición hardware que se hace en Vivado, en SDK se generaban los BSP necesarios para poder utilizarlos en KEIL. Posteriormente, en KEIL generamos los ficheros ELF y HEX, que junto con el fichero MMI, conseguimos generar el bitstream.

Se ha resumido como es el proceso, para poder programar nuestro diseño en la FPGA. Dentro de las facilidades que ofrece Vivado, se ha aprendido a trabajar de otra forma para conseguir un mismo resultado, el bitstream.

Por otra parte, se ha estudiado como puede ser la comunicación entre procesadores. Se estudiaron tres tipos: NoC, Point-to-Point y la comunicación por buses. A pesar de que la primera es la más importante y eficiente, se optó por trabajar con la comunicación por buses, ya que para la implementación que se quería realizar era la más práctica y sencilla de realizar con las IP que proporciona Xilinx. En este caso se utilizó la IP *AXI Interconnect* , donde se vieron las propiedades y funciones que ofrece. Como por ejemplo permite asignar prioridades entre los procesadores, de forma que trabajen de una forma determinada según las necesidades del diseñador. Este comportamiento se pudo comprobar y fue muy interesante entender cómo funcionaba, cuando compartían una memoria externa. Dentro de las tres tipos de conexión aparte del NoC, era la que tenía un comportamiento más eficiente que si se comparaba con la Point-to-Point, según sus características.

9.1.1. Futuro trabajo

Una de las características que tenía el Cortex-M1 era que permitía hacer depuración, de tres formas: JTAG, modo Serie o la combinación de ambas. En la hojas de información de este microprocesador, al haberse hecho las pruebas para dos tipos de FPGAs, se invitaba a los diseñadores a probar el dispositivo DAPTLINK, el cuál también tenía su propia IP y es de fácil integración en el diagrama de bloques de Vivado.

Puesto que el elemento más extendidos para depuración son los *SEGGER J-Link*, también es posible realizar la depuración con ellos. La única desventaja que cuenta, es que para poder usarlo, el diseñador debe de realizar una serie de configuraciones importantes en el diseño hardware para poder hacer las pruebas. Sería interesante, poder conseguir un diagrama de bloques configurado de manera que los microprocesadores estén preparados para la depuración.

Al principio de este trabajo se mencionó que ARM también tenía disponible la IP del Cortex-M3. Sería interesante trasladar las ideas y conceptos que se aplicaron en los diseños con el Cortex-M1. Se explicó que era un microprocesador con más completo que el propio Cortex-M1.

También sería interesante probar otro tipo de comunicación además de los buses, como la comunicación NoC. Realizar un diseño donde se pueda implementar esta comunicación, ya que era el mejor de los tres tipos de comunicación on-chip que se explicaron; ya que una de sus principales ventajas es que se conseguía mejorar el ***paralelismo***. Un factor muy importante cuando se trabajan con varios núcleos.

II. Pliego de condiciones

-Instalación

Para poder llevar a cabo el proyecto, es necesario que se sigan las instrucciones que se dan. También es importante que se utilicen los mismos elementos, tanto *hardware* y *software*. Y en el caso de que no sea posible, intentar que los elementos sean lo más compatibles a los que sean utilizado.

-Técnicas

Las condiciones técnicas de este proyecto son tanto a nivel hardware como nivel software, se va a proceder a detallar cuales son:

- **Hardware**

Para la parte hardware es necesario utilizar la tarjeta **Nexys 4 DDR**, junto con un cable **usb** tipo **micro-b.**

Las características del ordenador, en el cual, se ha realizado el proyecto son las siguientes:

HP Omen 15 con procesador Intel core i7-8750H (2,2 GHz, 6 núcleos/12 hilos), memoria RAM 12 GB, unidad SSD 256 GB y disco duro 1 TB.

- **Software**

Los requisitos generales son: Sistema operativo Windows 10

Se ha utilizado:

1. Vivado 2019.1 versión Webpack (64 bit).
2. IP Cortex-M1 de ARM DesignStart FPGA.
3. Nexys 4 DDR xdc.
4. Ficheros de las Tarjetas Digilent desde GitHub
5. Keil:
   MDK-ARM Professional Version:5.26.2.0
   C compiler: ARMcc.exe V5.06
   Assembler: ARMasm.exe V5.06
   Linker: ARMLinker.exe V5.06
   Library Manager: ARMAr.exe V5.06
   Hex Converter: FromElf.exe V5.06
   CPU DLL: SARMCM3.DLL V5.25.2.0
   Dialog DLL: DARMCM1.DLL V1.19.1.10
   Target DLL: UL2CM3.DLL V1.161.6.0
   Dialog DLL: TARMCM1.DLL V1.14.0.0
   Flex license: MDK-ARM Professional (mdk_pro)

III. Presupuesto

Se va a proceder a estimar el coste que ha tenido este proyecto. Para hacer el cálculo se ha dividido en tres partes: software, hardware y mano de obra.

Recursos Software				
Concepto	Coste (€)	Amortización (años)	Tiempo de uso (meses)	Total (€)
KEIL MDK Professional	3970.00	1	5	1654.16
Vivado 2019.1 versión Webpack	0	-	5	0
IP Cortex-M1 de ARM DesignStart FPGA	0	-	5	0
Windows 10 Home	145	-	5	0
Sublime Text 3	0	-	5	0
Editor de diagramas	0	-	5	0
Total				1654.16

Tabla 16. Coste en los recursos hardware.

Recursos Hardware				
Concepto	Coste (€)	Amortización (años)	Tiempo de uso (meses)	Total (€)
Nexys 4 DDR	326.77	-	5	
Ordenador HP Omen 15	1000	5	5	83.33
Total				83.33

Tabla 17.Coste en los recursos software.

Mano de obra			
Concepto	Coster por hora (€/h)	Horas	Coste
Implementación	35	352	12320
Redacción	15	88	1320
Total			13640

Tabla 18. Coste en mano de obra.

Con las tres tablas anteriores podemos calcular el presupuesto total del proyecto.

$$CT = 1654.16 + 83.33 + 13640 = 15377.49€$$

Si aplicamos el 13% de los gastos generales y el 6% de beneficio industrial. Nos queda que: $P_{GG+BI} = 11417.49 \cdot (1.19) = 18299.213€$

El presupuesto total del proyecto contando con el 21% de IVA nos queda en:

$$P_F = \mathbf{22142.04}€$$

IV. Manual de usuario.

En este apartado se va a detallar como poner en marcha este proyecto desde cero. Una vez se ha asegurado que dispone de las condiciones técnicas, se procede a continuar con la explicación.

4.1. Descarga de la IP de ARM

Para poder descargar la IP ARM, es necesario registrarse en la página web de ARM. Una vez allí, en el apartado de ARM DesingStart FPGA, se selecciona descargar la IP del Cortrex-M1. Lo que aparece es lo mismo que se ve en la Figura 93.

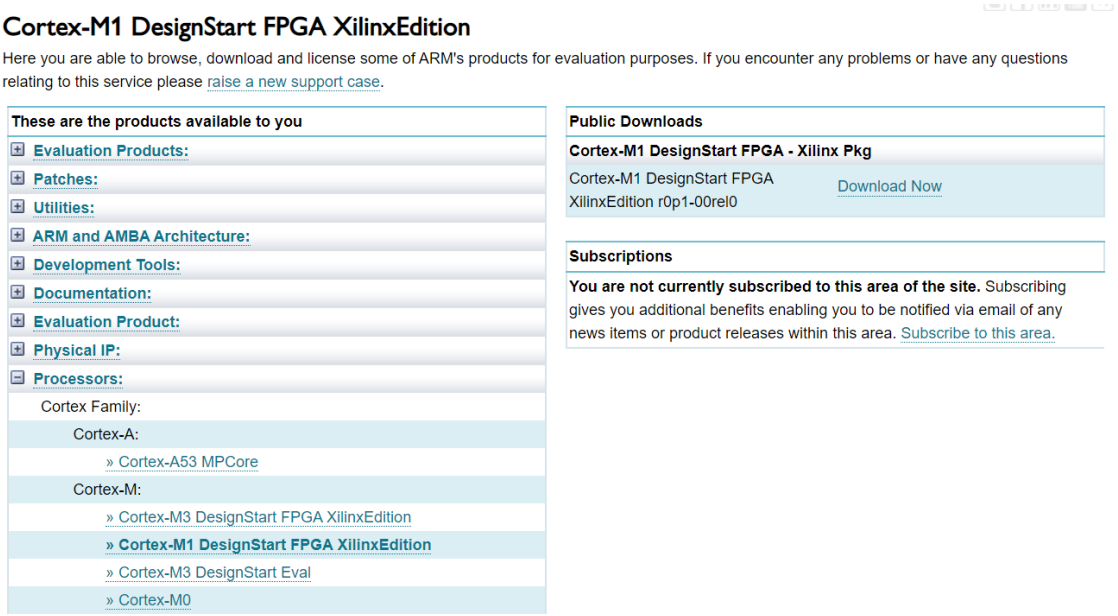


Figura 93. Descargar la IP de ARM

Una vez, se haya completado la descarga, aparecerá en el lugar que ha seleccionado para guardar la IP, un fichero con el siguiente nombre:

AT472-BU-98000-r0p1-00rel0.tgz

Al descomprimirlo, aparecen cinco carpetas, un pdf y un fichero formato txt.

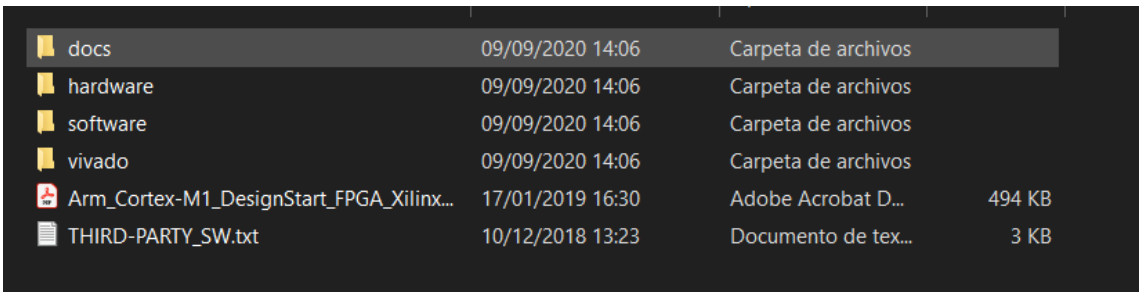


Figura 94. Directorio de la IP de ARM

4.2. Instalación de tarjetas

En el caso de que en la versión de vivado no disponga de la Tarjeta Nexys 4 DDR, se deberá instalar de forma manual.

Para ello se debe ir a la página web https://github.com/Digilent/vivado-boards, y descargar los ficheros, en la parte que pone Code, como se ve en la Figura 95.

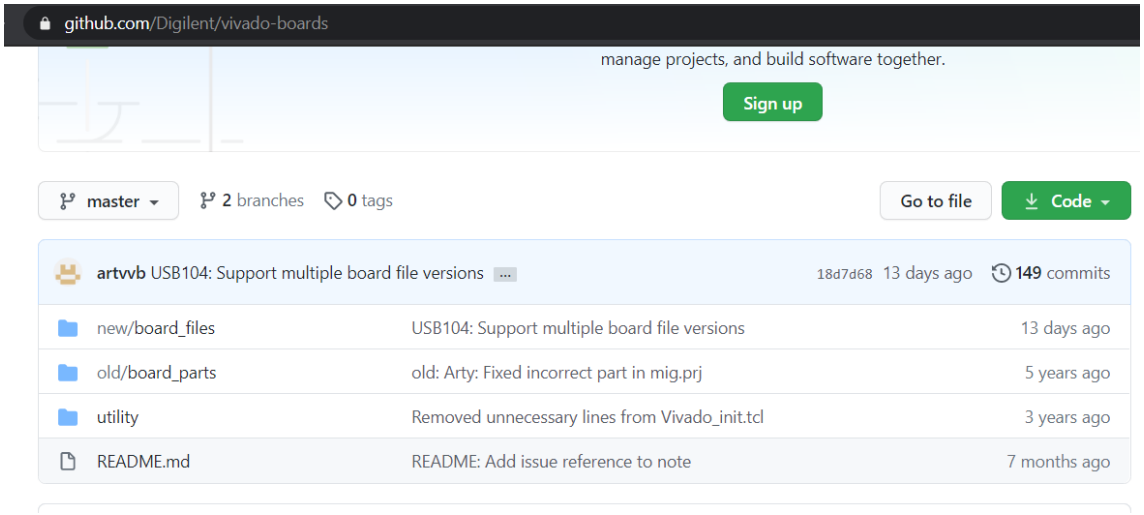


Figura 95. Descarga de las tarjetas de Digilent

Después de la descarga, se debe buscar en el path donde tiene instalado Xilinx, en el caso de este proyecto, el path es: D:\Xilinx\Vivado\2019.1\data\boards\board_files

Por otra parte, en el directorio donde guardo la descarga que se hizo en GitHub, se copia la carpeta de la Nexys 4 DDR.

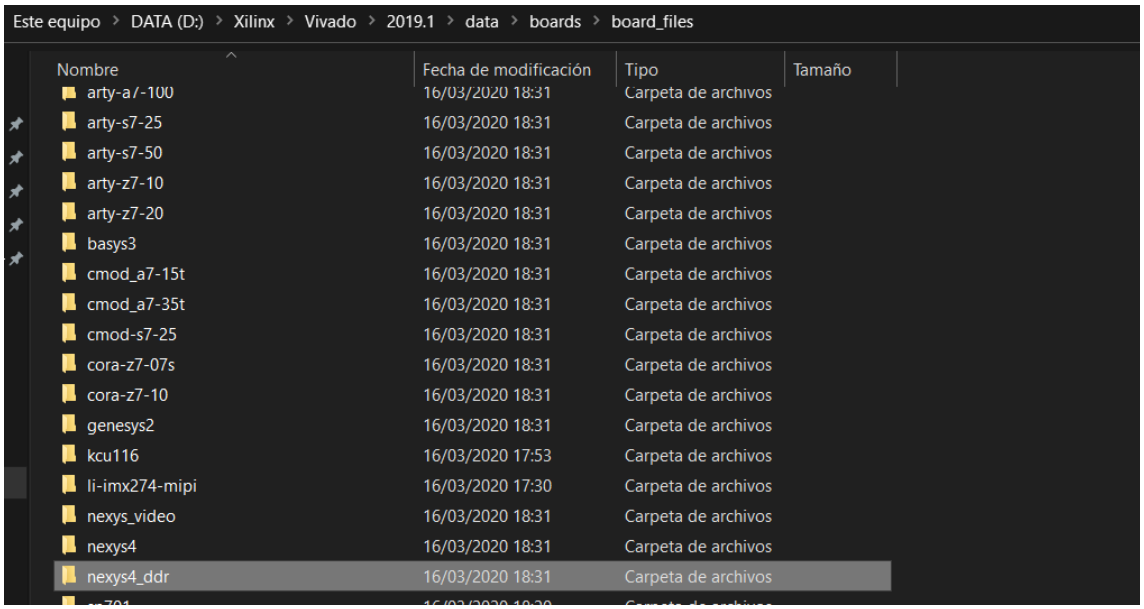


Figura 96. Directorio donde están las tarjetas de Digilent

4.3. Vivado 2019.1

Completado los pasos anteriores, ya se está en posición de abrir vivado para iniciar el proyecto. Antes de hacerlo cree una carpeta donde almacenará todo lo relacionado al proyecto, tiene que incluir tres carpetas, llamadas: ***hw,sw, vivado***. La carpeta **vivado**, deberá copiarla del directorio donde se guardó la IP de ARM, la misma que se ve en la Figura 97, y pegarla en el nuevo directorio donde se va a trabajar con vivado.

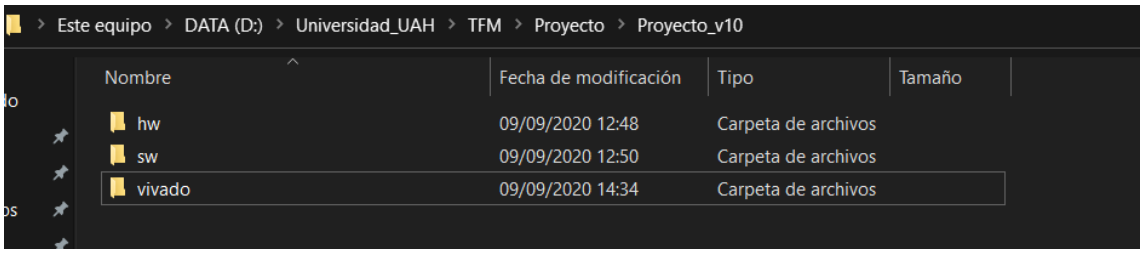


Figura 97. Directorio para trabajar con vivado.

Ahora, el siguiente paso consiste en abrir vivado 2019.1 y añadir en *Repositories*¸ la IP de ARM que hemos descargado y se encuentra en la carpeta vivado de la Figura 98. Para ello se tiene que acceder a la sección *Settings* en la pestaña de *Tools*.

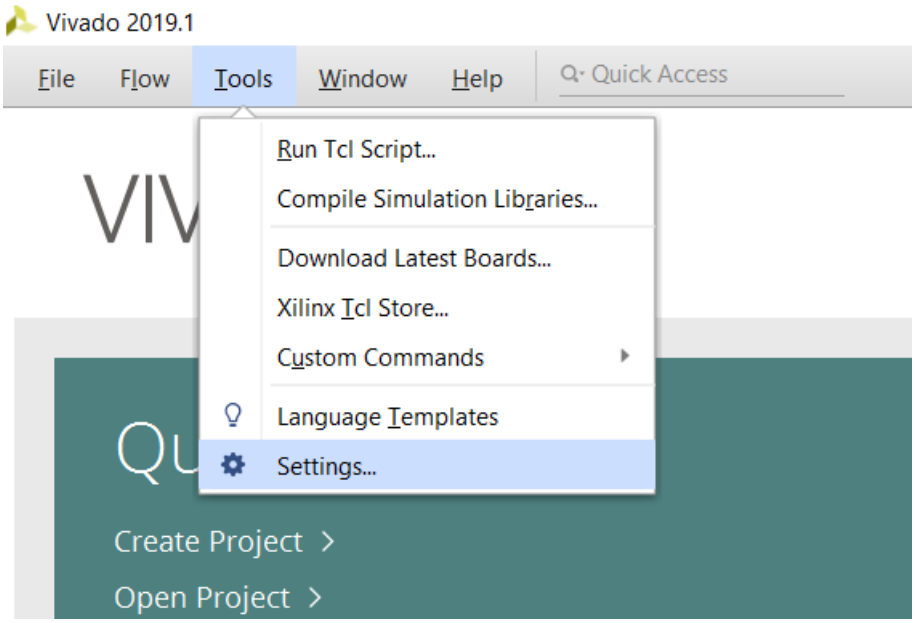


Figura 98. Página principal de Vivado 2019.1

En la parte de *settings*, se tiene que buscar *IP Defaults*, allí se tiene que buscar la IP de ARM, que está en la carpeta vivado que está dentro del directorio, como se ve en la Figura 98. Para asegurar que esta IP está dentro de nuestro proyecto, se volverá a incluir, pero esto se explicará más adelante.

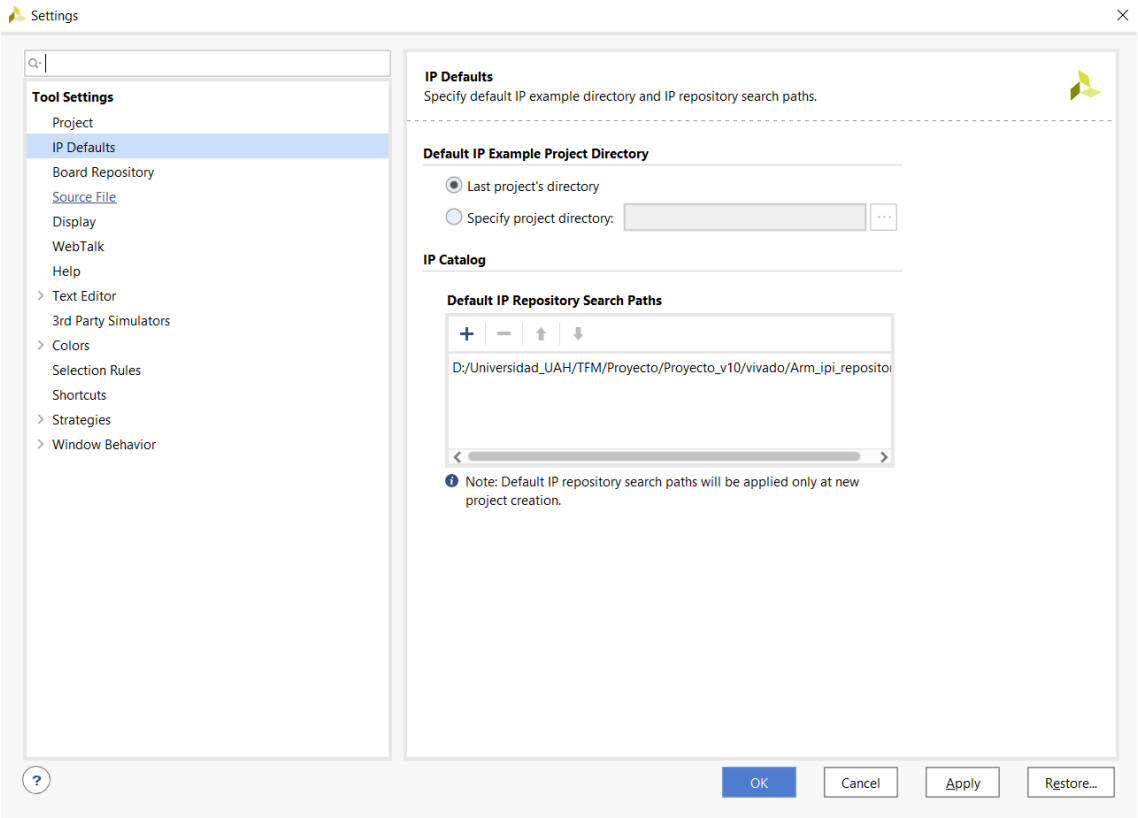


Figura 99. Incluir la IP de ARM en vivado

4.4. Crear proyecto nuevo

Para crear un proyecto nuevo, se hace el procedimiento habitual, que se explicará a continuación.

Lo primero que se tiene que hacer es seleccionar el path del proyecto, y asignar un nombre. Como se ve en la Figura 100, se le ha asignado como *Project_1*, en el path que se había creado y concuerda con el de la Figura 97. Este paso se puede ver en la Figura 100 y se pasa a la siguiente parte con *next*.

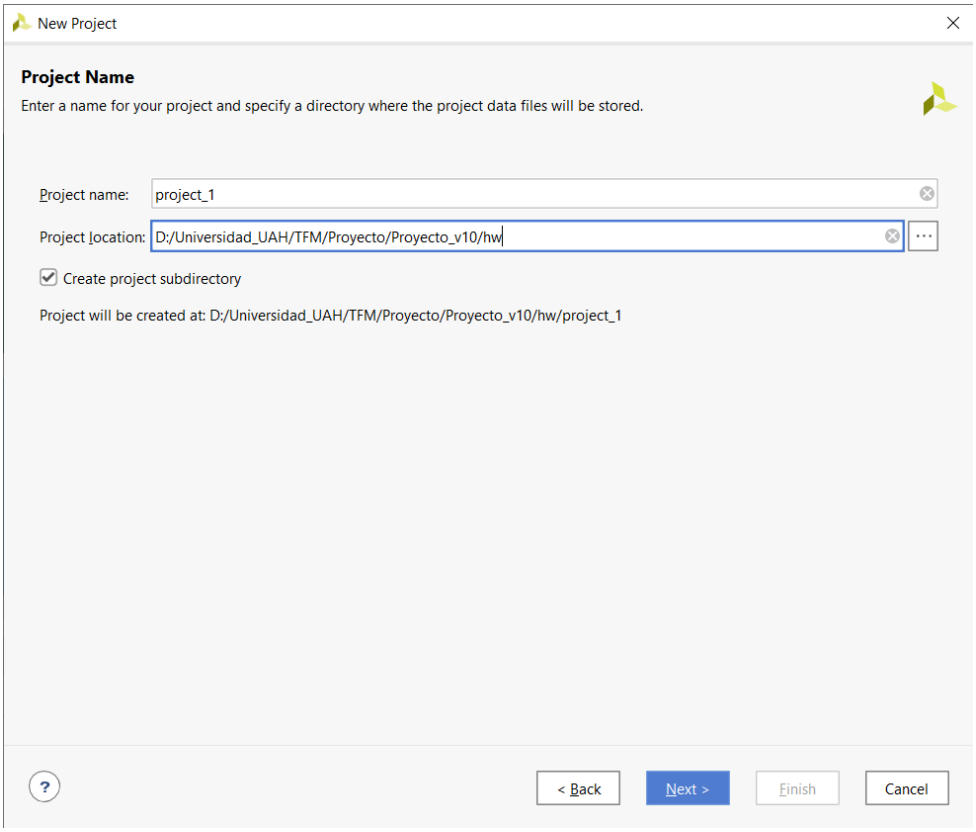


Figura 100. Primera parte de crear un proyecto nuevo.

En las Figura 101,Figura 102 y Figura 103, se dejan las características que se tiene por defecto.

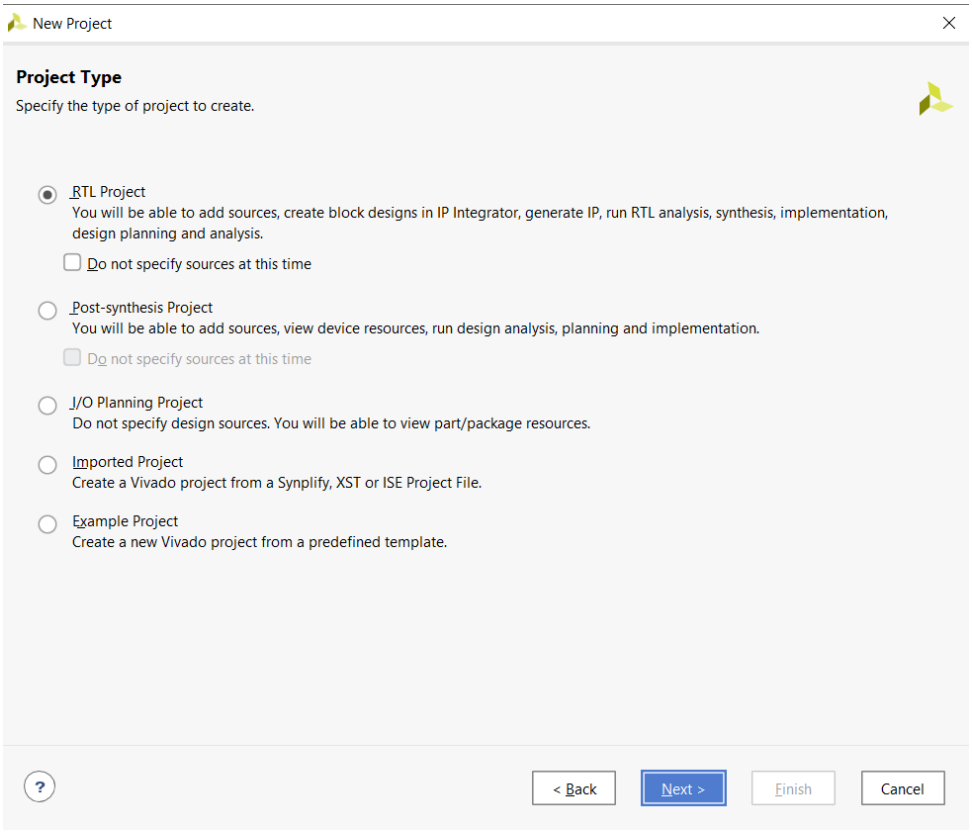


Figura 101. Segunda parte de crear un nuevo proyecto

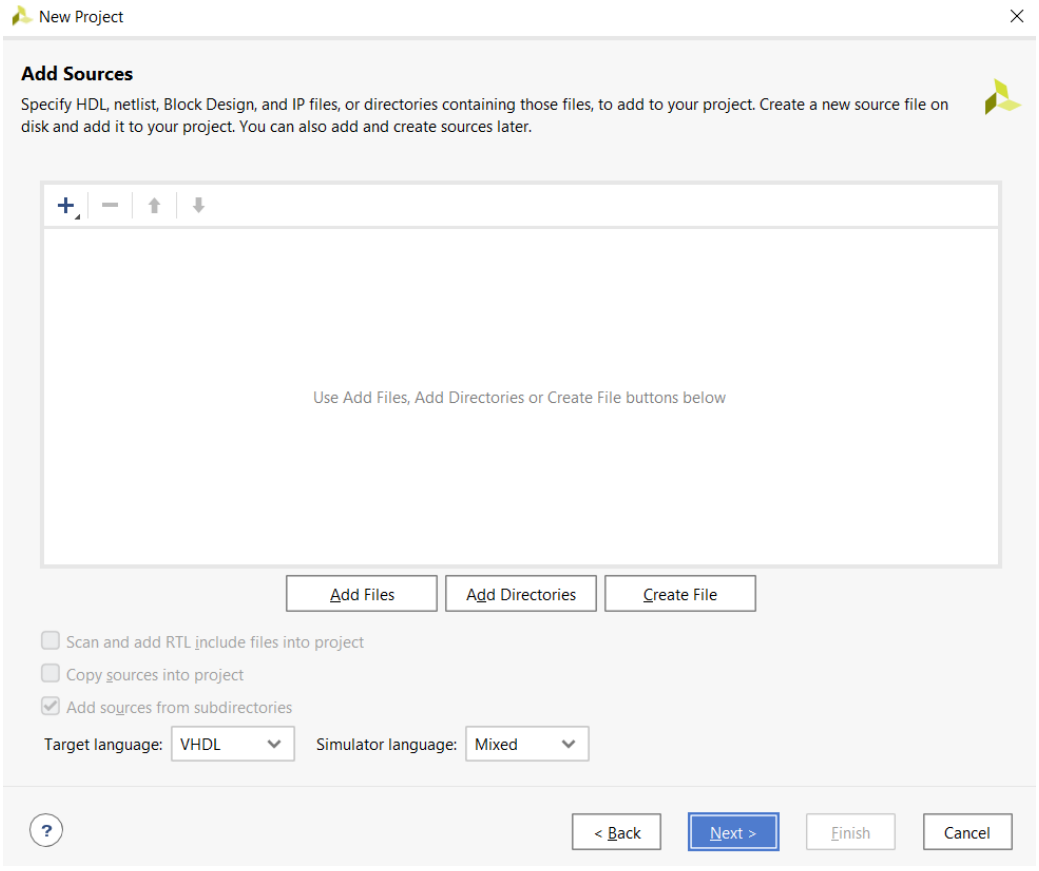


Figura 102. Tercera parte de crear el proyecto.

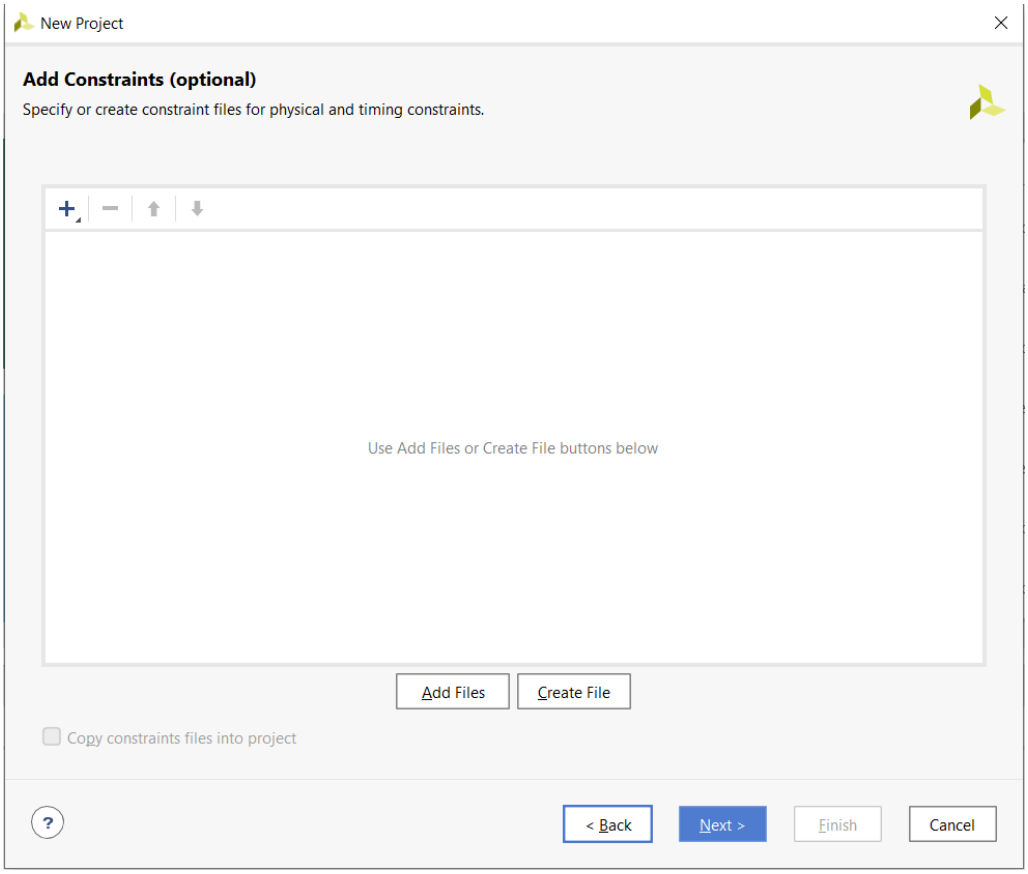


Figura 103. Cuarta parte de crear el proyecto nuevo.

En la Figura 104, se debe seleccionar la tarjeta con la que se va a trabajar, en este caso es la Nexys4 DDR.

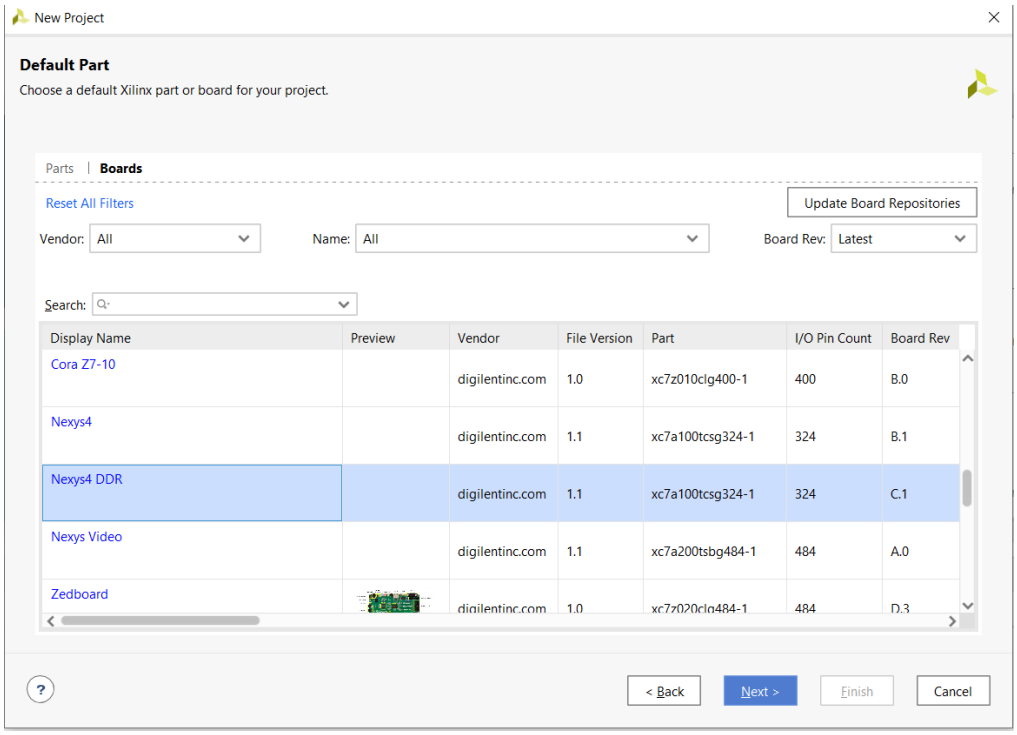


Figura 104. Quinta parte de crear un proyecto nuevo.

La última parte como se ve en la Figura 105, consiste en un resumen de todos los pasos anteriores. Seleccionamos *finish* y se termina por esta parte.

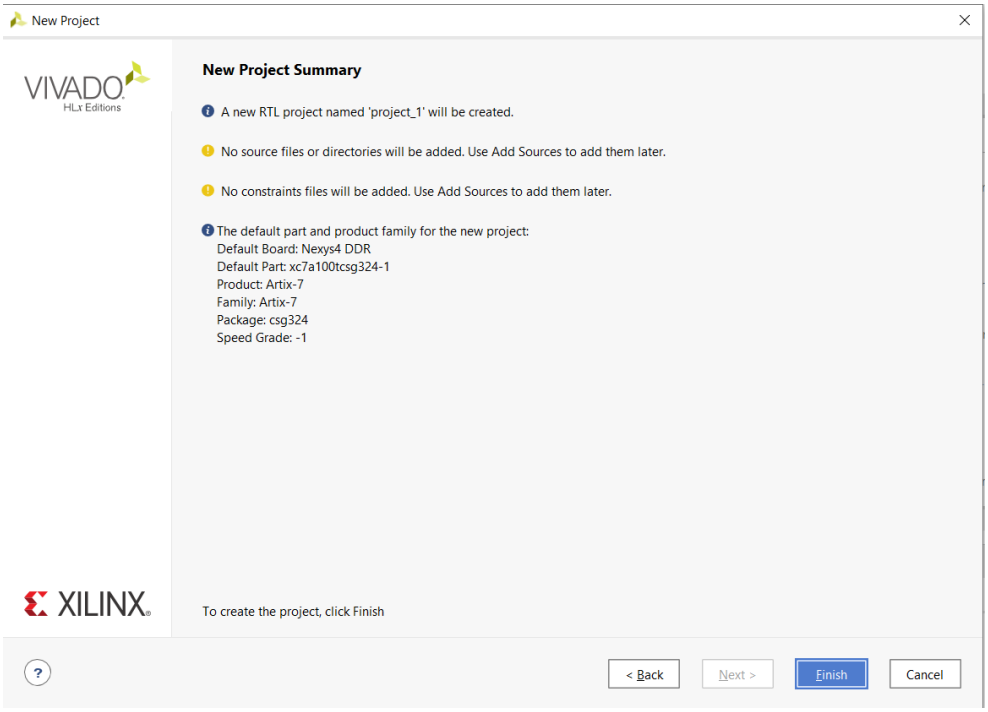


Figura 105. Última parte de crear un proyecto nuevo

El resultado de todo lo anterior, se puede ver en la Figura 106. Posteriormente, es necesario crear un *Block Design*; para esto seleccionamos *Create Block Design*, y le asignamos el nombre de *system.*

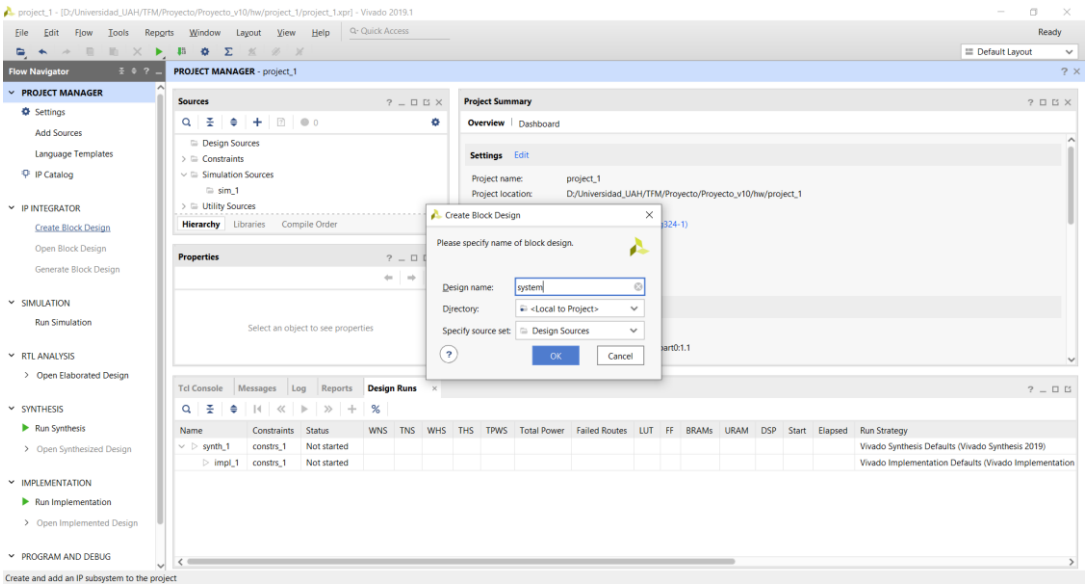


Figura 106.Resultado de crear el proyecto nuevo.

Este paso volvemos añadir la IP, para asegurarnos de no tener ningún problema. Seleccionamos *Tools*, seguido de *Settings.*

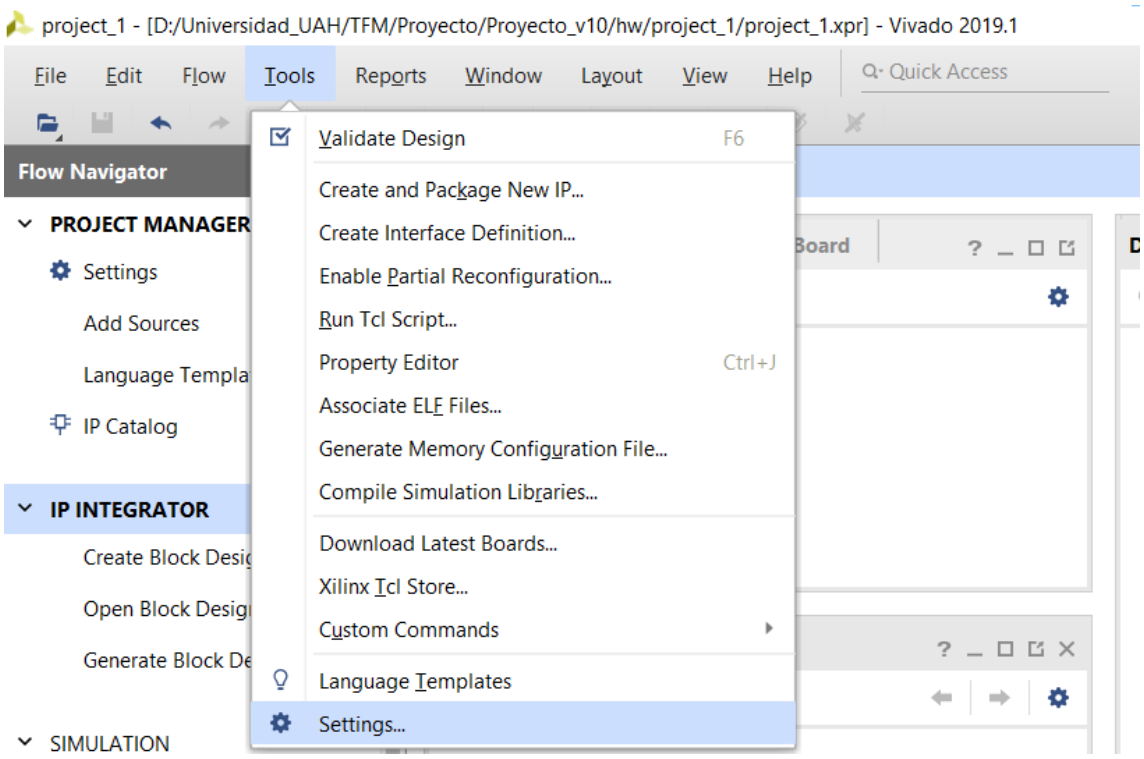


Figura 107. Añadir IP de ARM

Buscamos en la sección de *IP Repository*, y añadimos la IP de ARM, como se ve en la Figura 108. Le damos *apply* y terminamos esta parte.

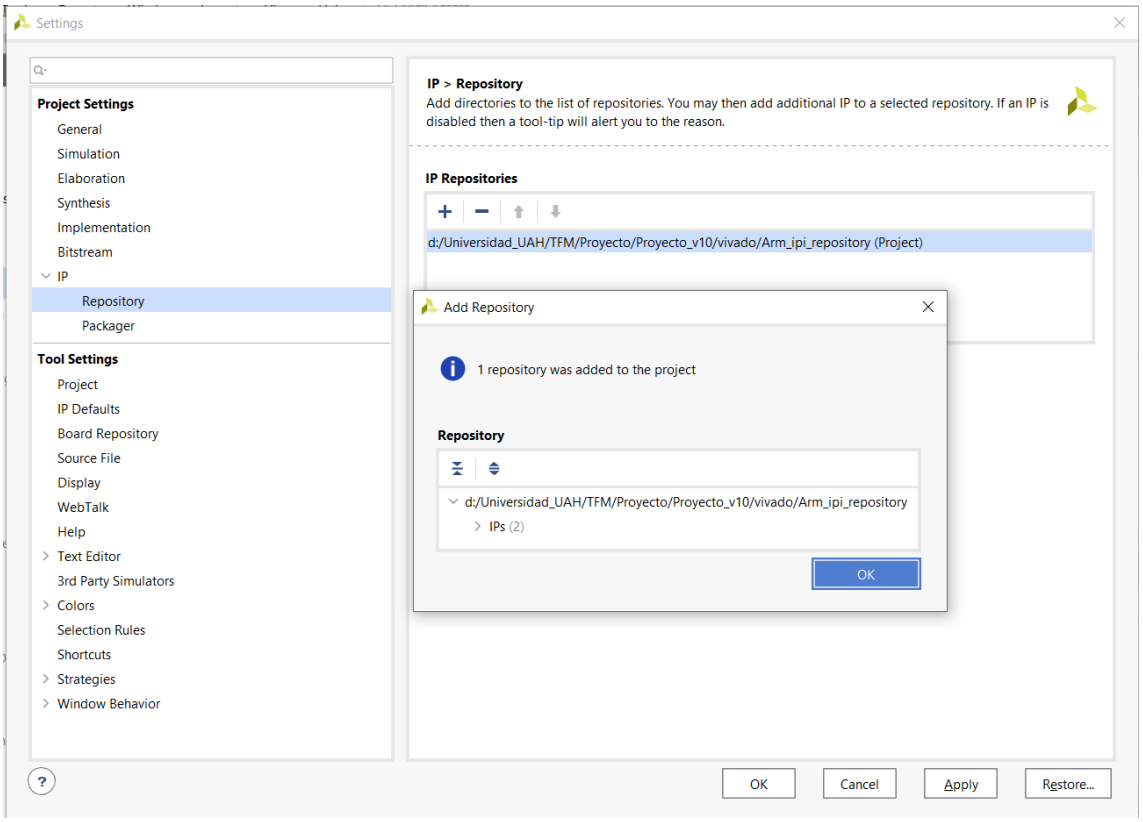


Figura 108. Paso final de añadir IP de ARM

4.5. Ejemplo de "Hello World"

Para empezar con un proyecto básico como el "Hello World", vamos a buscar al procesador Cortex-M1 que ya debe de estar disponible, al haber agregado la IP al repositorio de vivado. Como es un proyecto sencillo vamos a configurarlo de forma que no utilice muchos recursos y sea lo más sencillo posible.

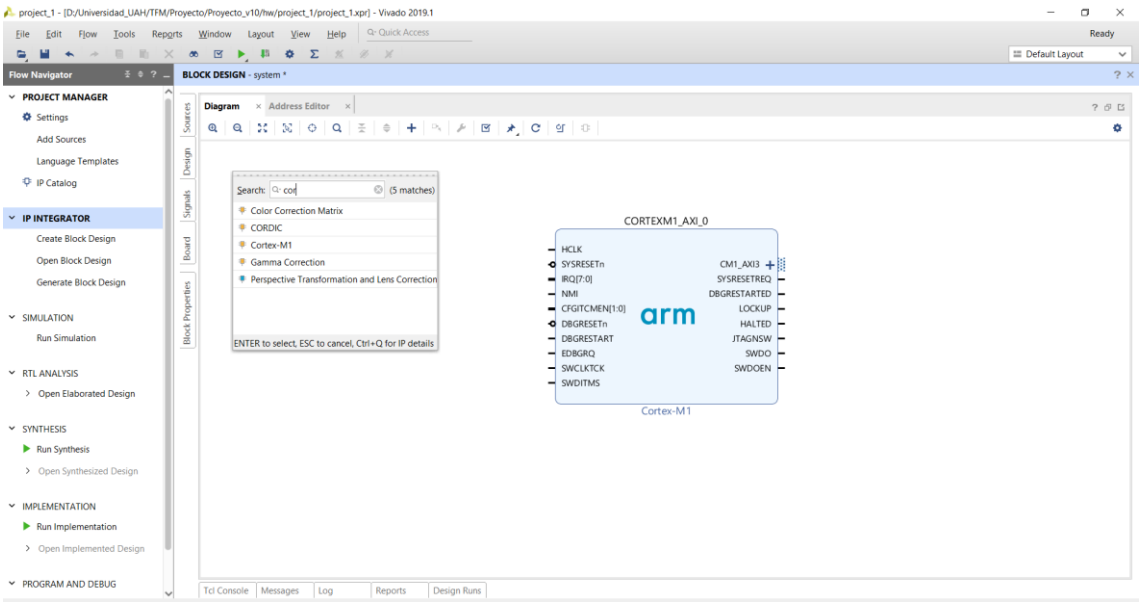


Figura 109. Selección del Cortex-M1

Para que cumpla la simpleza que se ha mencionado, este procesador no debe tener la opción a depuración y el tamaño de la memoria de instrucciones y de datos tendrán un tamaño de 32 Kb.

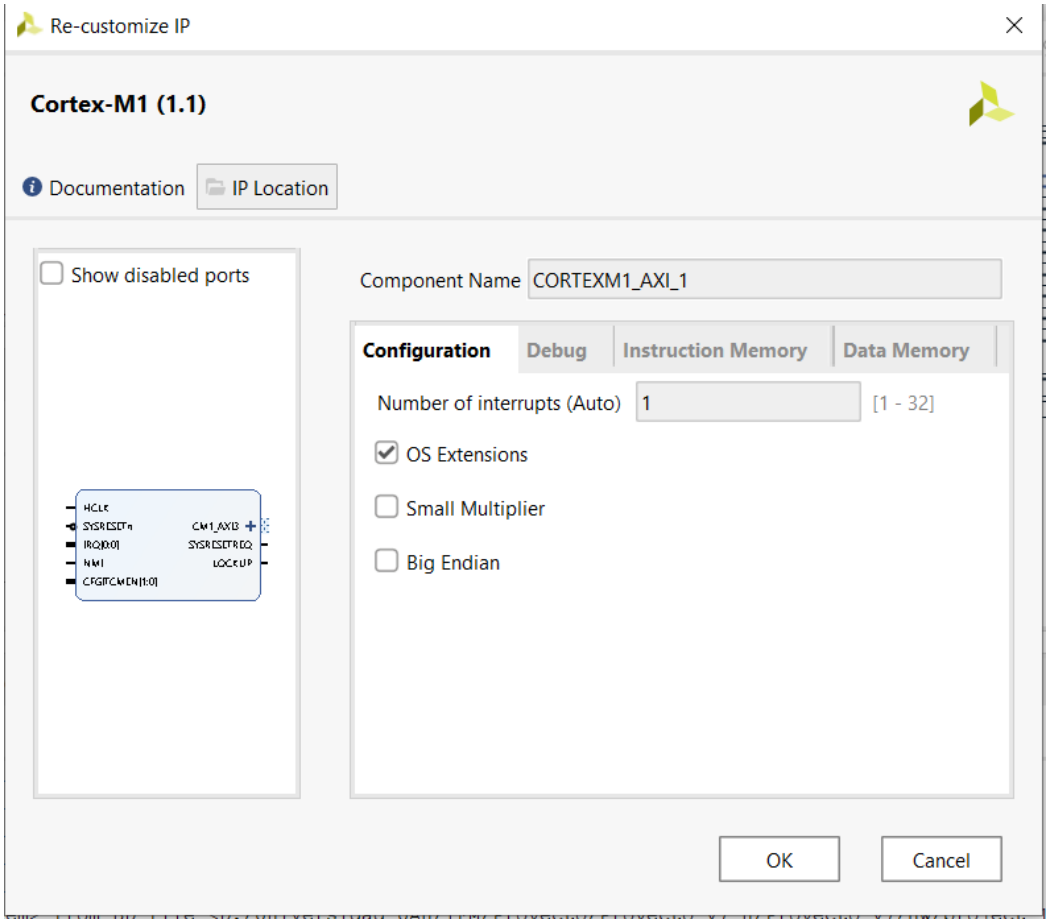


Figura 110. Configuración del Cortex-M1

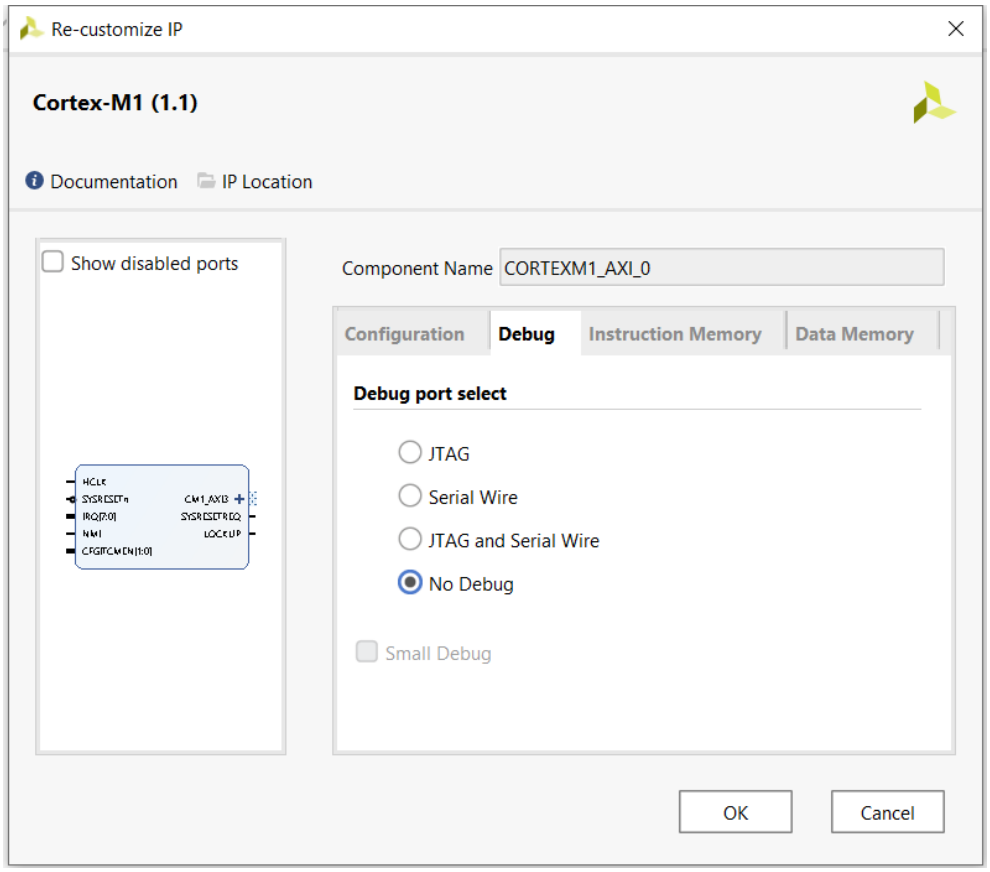


Figura 111. Configuración del Cortex -M1 Debug

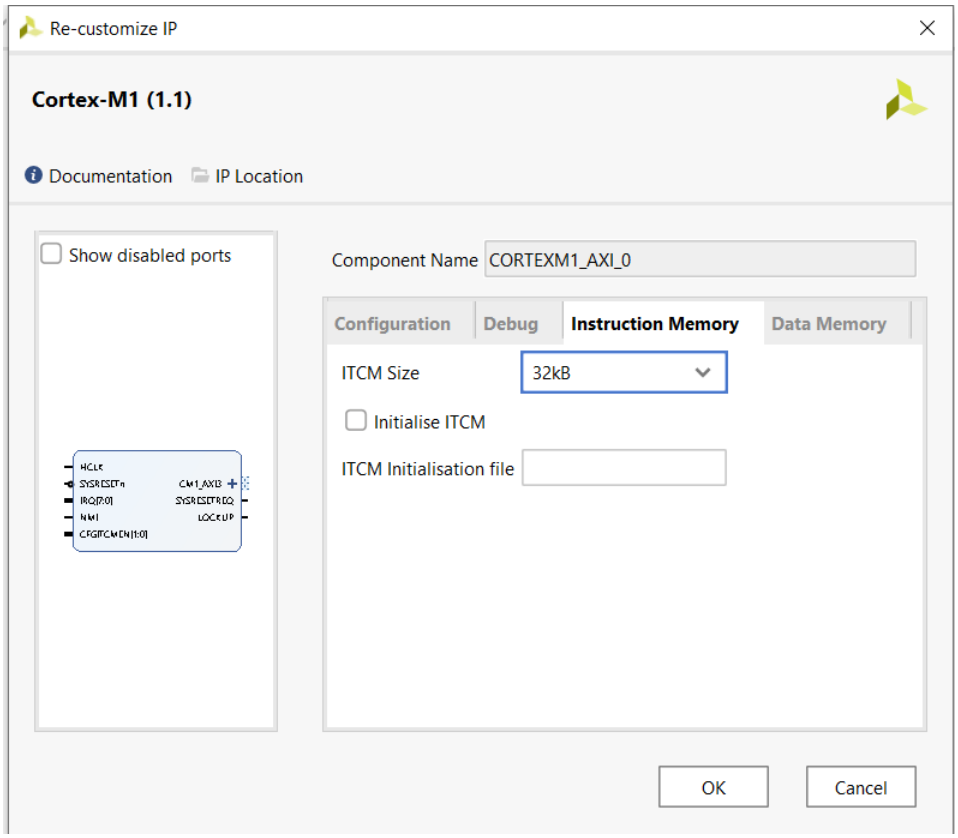


Figura 112. Configuración del Cortex-M1 memoria de instrucciones

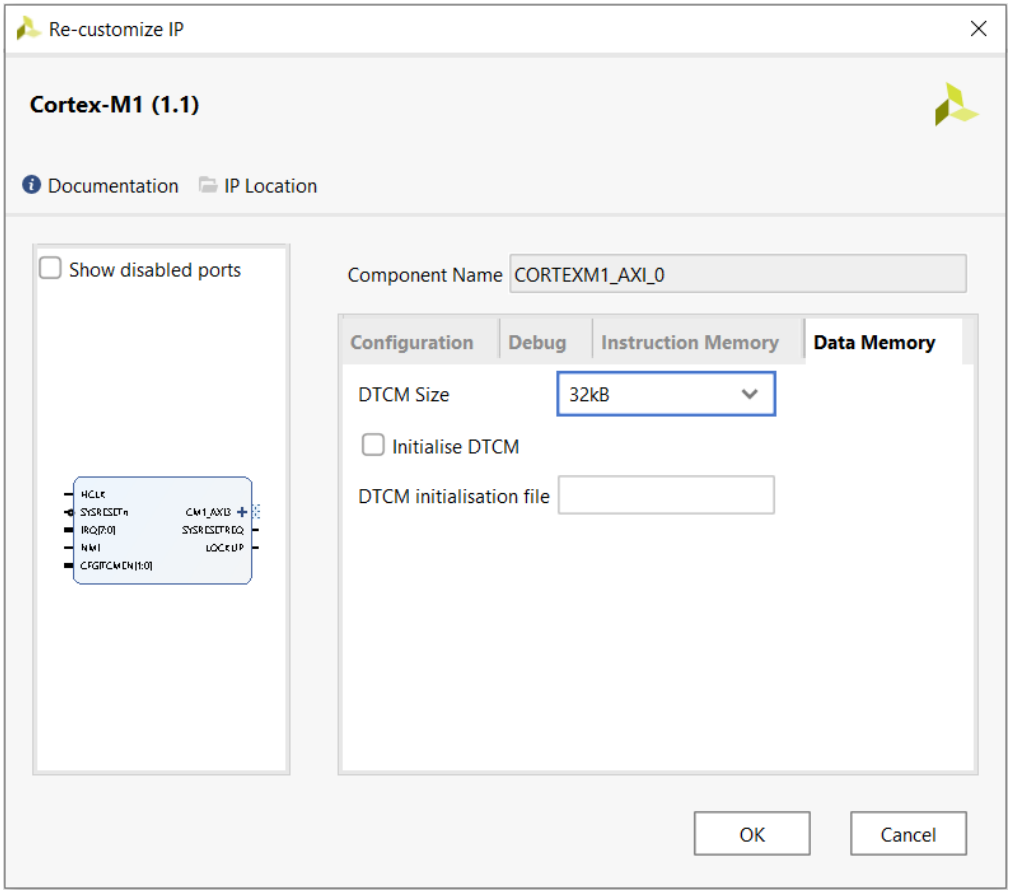


Figura 113. Configuración del Cortex-M1 memoria de datos

Ahora necesitamos añadir la señal de reloj y el reset, para ello vamos a usar una señal de reloj de entrada de 12 MHz, para que mediante el **clock wizard**, se tenga a la salida una señal de reloj, cuya frecuencia sea de 100 MHZ, y la señal de reset debe ser activa a nivel bajo. Estas dos señales las podemos agregar al diagrama de bloques de dos formas. La primera puede ser manual, creando los puertos de forma manual o desde la pestaña ***Board***. Para hacerlo más sencillo, se va a utilizar el segundo método.

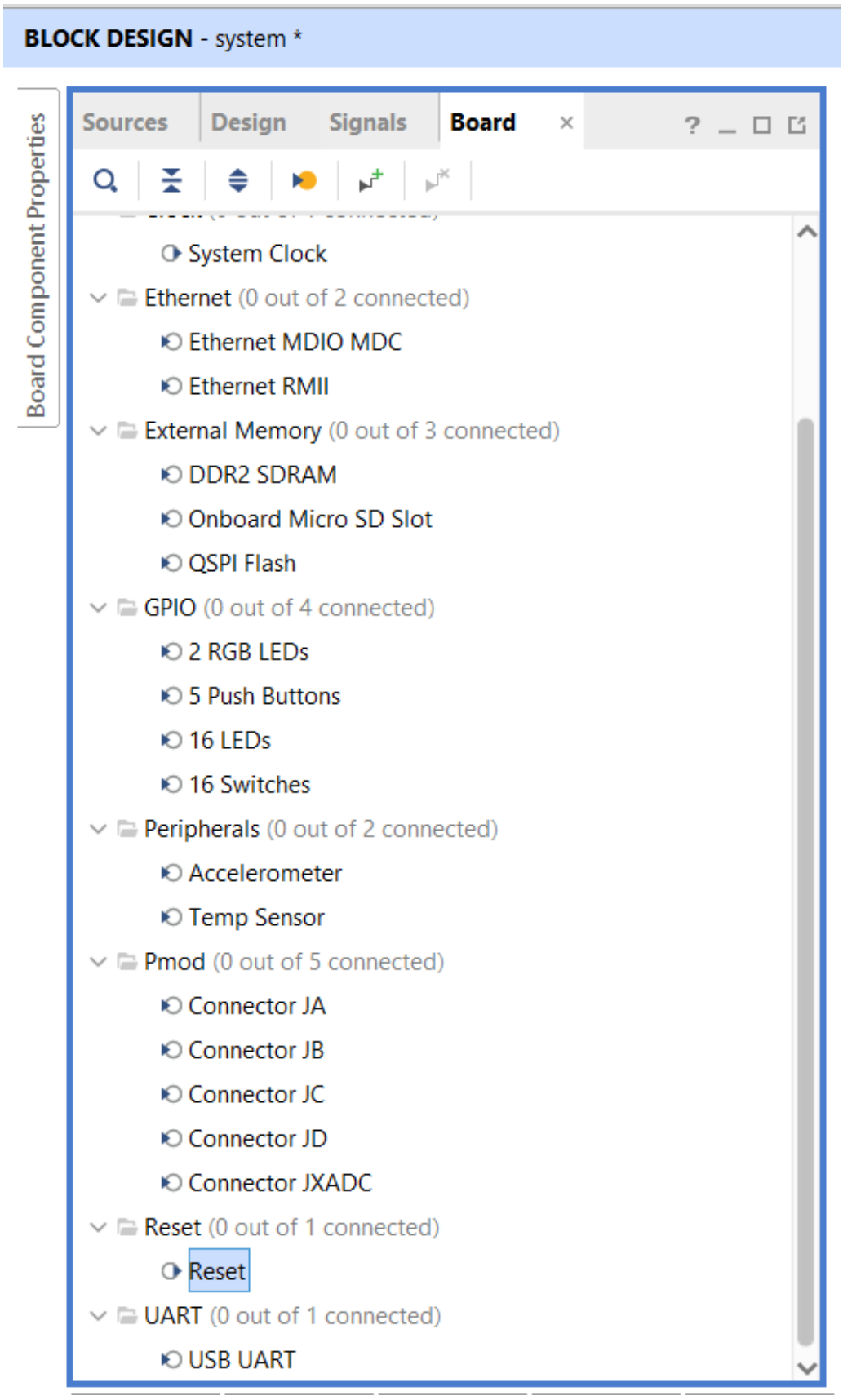


Figura 114. Selección del Reset de la zona de componentes de la tarjeta.

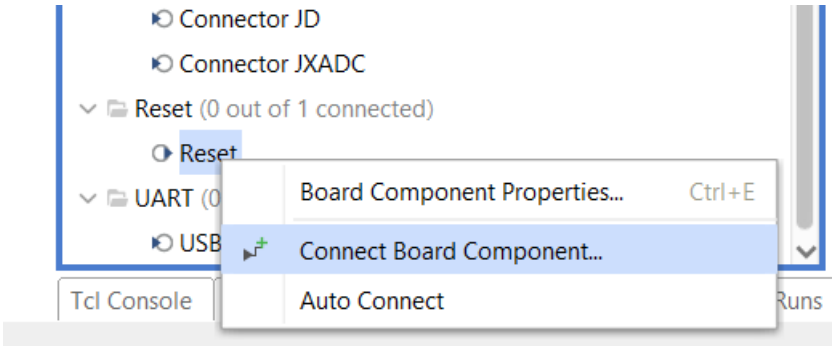


Figura 115. Conectar el reset con el diagrama de bloques

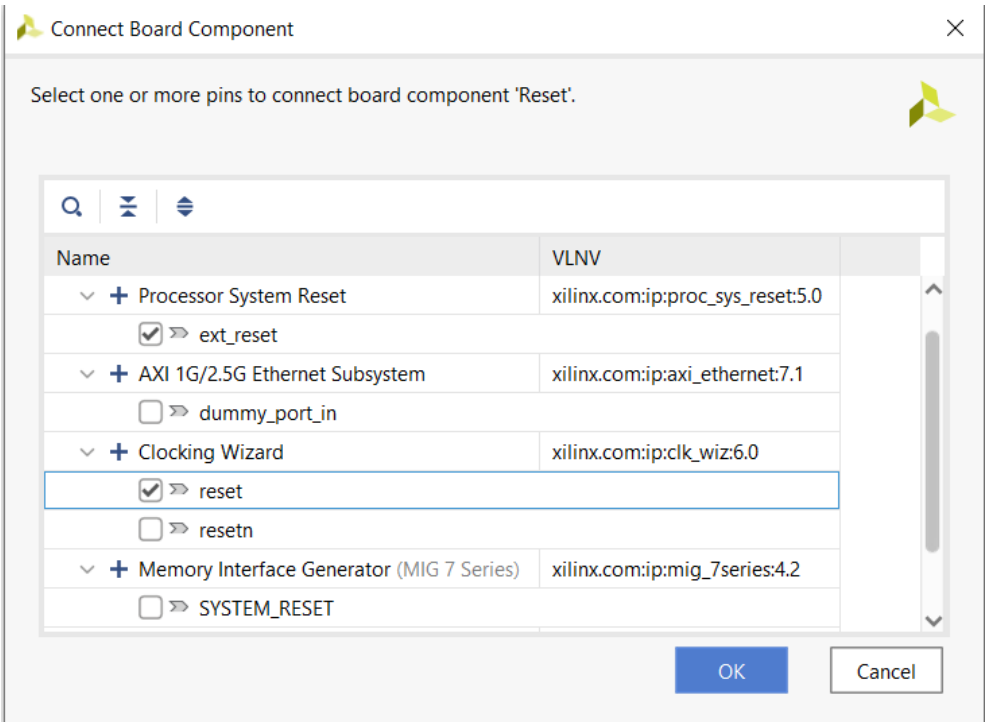


Figura 116. Conectar el reset con el Clock Wizard.

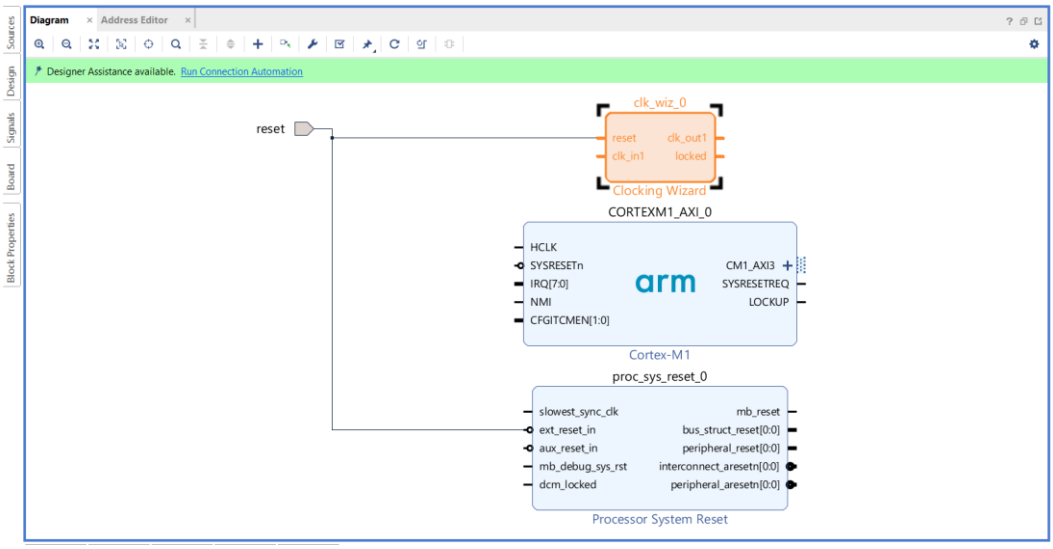


Figura 117. El resultado de los pasos anteriores

En las figuras anteriores se ha configurado la señal reset. Ahora se procede a configurar la señal de reloj.

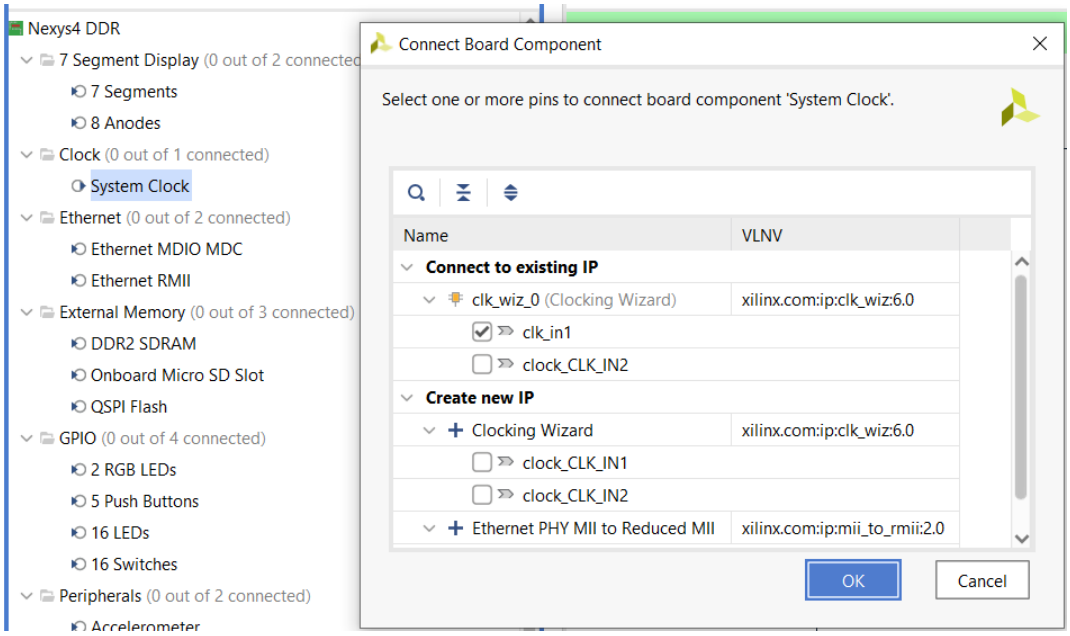


Figura 118. Seleccionamos la señal system clock.

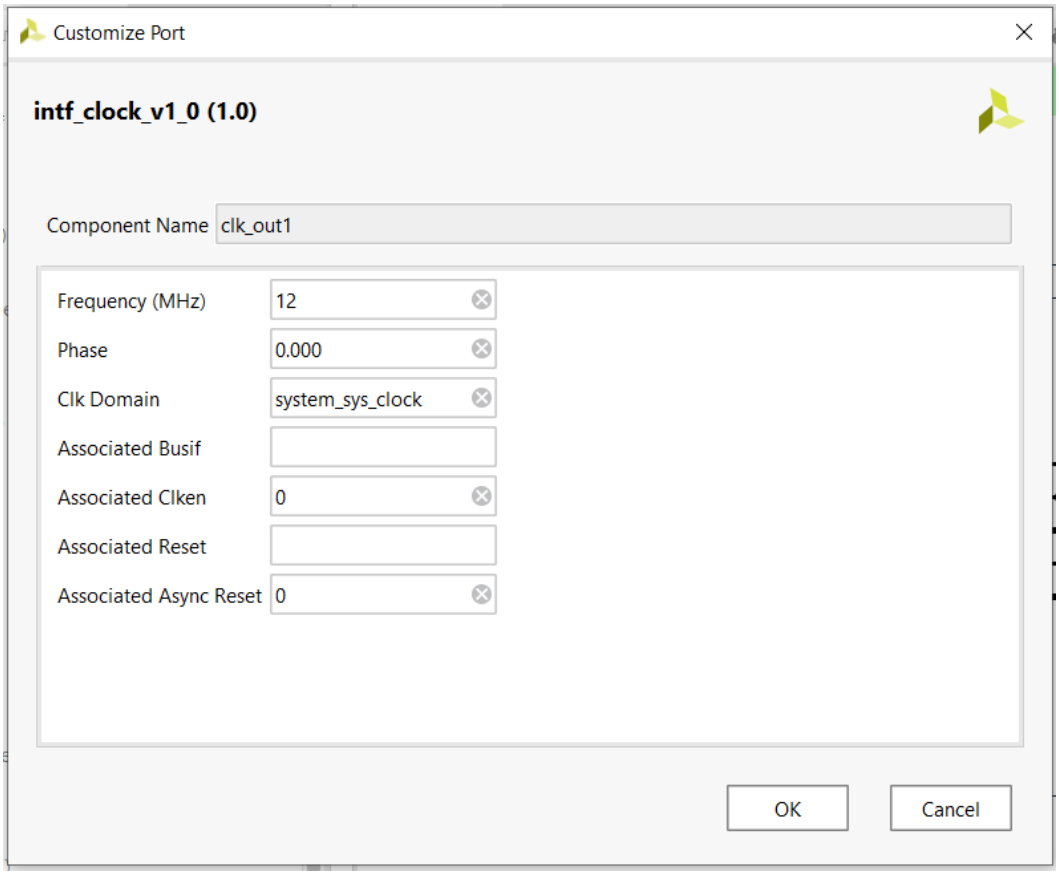


Figura 119. Configuramos la frecuencia.

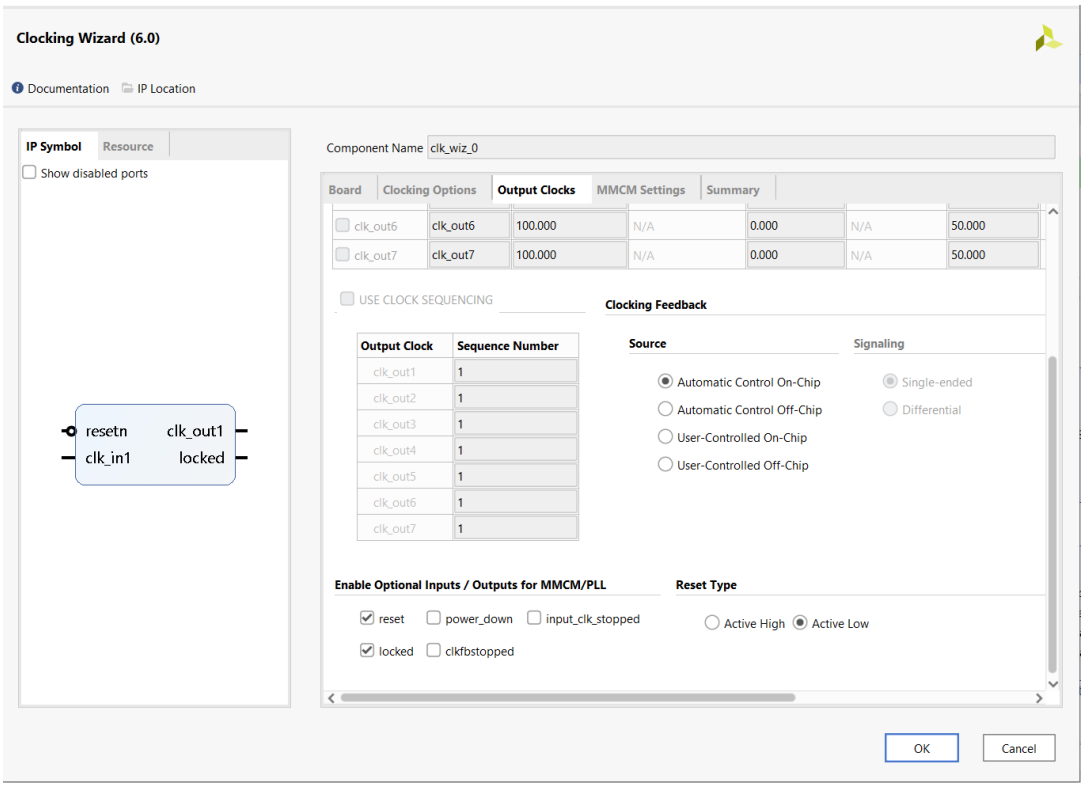


Figura 120. Configuramos el reset activo a nivel bajo.

En la Figura 120, hay que asegurarse que tanto la frecuencia de salida de la señal de reloj sea de 100 MHz. Además, hay que verificar que el reset esté activo a nivel bajo.

Dado que el ejemplo que se quiere hacer es un "Hello world", se emplea en este proyecto la UART. Entonces, se hace lo mismo que en los casos anteriores, se selecciona la UART, como se ve en la Figura 114, dentro de las disponibles nos quedaremos con la AXI UART LITE. Para continuar, se le da a *Run Connection Automotion* y automáticamente nos conecta la UART Lite con el Cortex-M1 mediante el bus AXI. En este paso hay que tener especial cuidado, ya que el puerto interconnect_aresetn[0:0] del *Processor System Reset* debe de estar conectado el puerto *ARESETN* del AXI Interconnect. Para el resto de los puertos reset deben de estar conectados al puerto *peripheral_aresetn[0:0]*.

Para terminar con el diagrama de bloques, el puerto *CFGITCMEN* se le da un valor 0x03, y tanto para los puertos *IRQ* y *NMI* son puestos a nivel bajo. Después de validar el diseño nos tiene que quedar parecido a lo que tenemos en la Figura 121. Asimismo, se tiene que comprobar que se le ha asignado una posición de memoria a la UART Lite en el Cortex-M1, vivado lo suele hacer automáticamente.

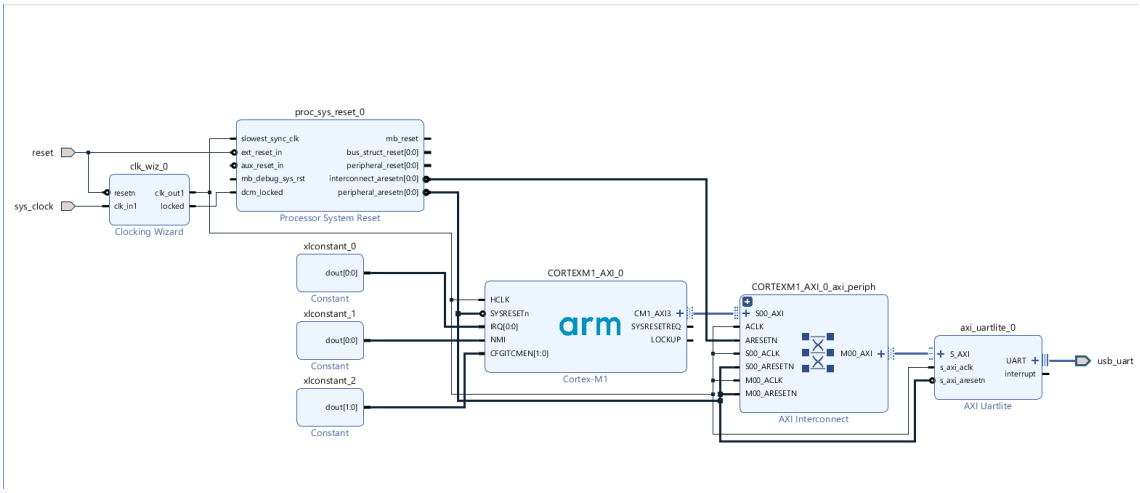


Figura 121. Diagrama de bloques final

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
CORTEXM1_AXI_0					
CM1_AXI3 (32 address bits : 0x00000000 [1M] ,0x40000000 [512M] ,0x60000000 [1G] ,0xA0000000 [1G])					
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF

Figura 122. Asignación de memoria de la UART Lite.

Como un proyecto normal, el siguiente paso consiste en crear el HDL wrapper del diseño, ejecutar la *synthesis* y posteriormente la *implementation*. Cuando se termine de ejecutar la implementación y antes de seguir para generar el bitstream, es necesario crear el fichero MMI. Para ello se utilizará un fichero TCL que nos permitirá obtener el fichero MMI y así crear el bitstream.

Entonces abrimos la implementación y damos *ctrl+F* para buscar los bloques de memoria ram de la interfaz TCM.

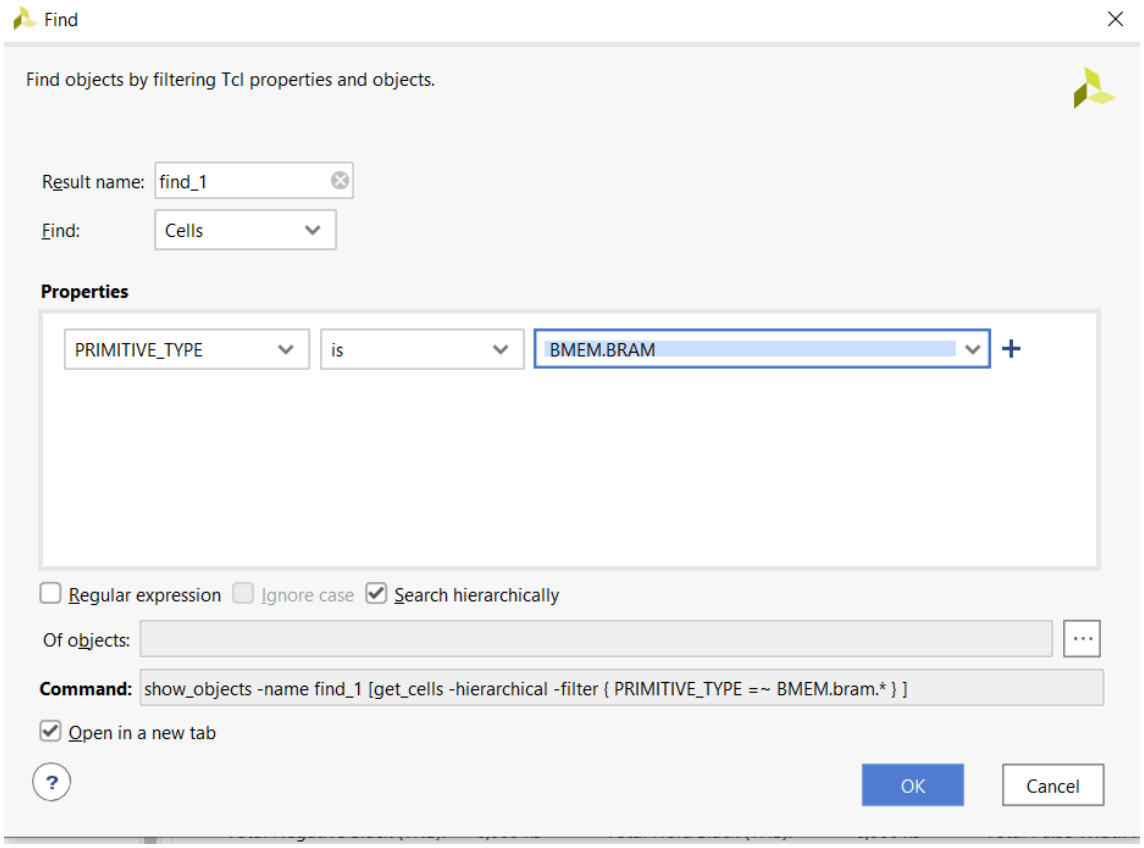


Figura 123. Buscamos los bloques BRAM

El resultado de la búsqueda se puede ver en la Figura 124, se pueden contar ocho bloques RAM para la interfaz DTCM y otros ocho para la interfaz ITCM, para poder crear nuestro bitstream necesitamos los bloques RAM de ITCM.

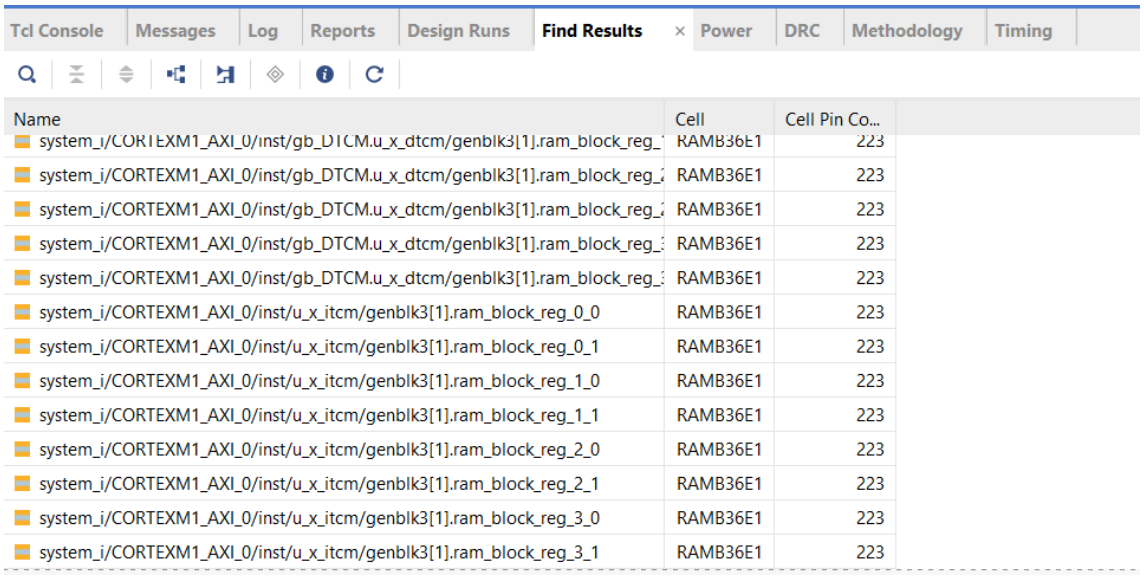


Figura 124. Bloques RAM

Para crear el fichero MMI, se parte del fichero que viene dado en la carpeta comprimida que se descargó anteriormente de ARM. Concretamente, del fichero llamado *make_mmi_file.tcl* que se encuentra dentro del path \AT472-BU-98000-

r0p1-00rel0\hardware\m1_for_arty_a7\m1_for_arty_a7. Dado ese proyecto se realizó con un *ARTY A7*, se tendrá que realizar unos pequeños cambios para la trabajar con la Nexys 4 DDR. Si abrimos el fichero anteriormente mencionado con un editor de textos como Notepad++ o Sublime Text, editamos las siguientes partes.

1. Cambiar el set part por el correspondiente a la Nexys 4 DDR: xc7a100tcsg324-1
2. En la parte que se encarga de escribir la cabecera del fichero MMI en XML, se tiene que cambiar el *AddressSpace Name*, este debe de coincidir con el que tenemos en la Figura 124, en este caso será:
   system_i.CORTEXM1_AXI_0.inst.u_x_itcm

Teniendo el fichero tcl modificado lo guardamos en la carpeta project_1 que se encuentra dentro del directorio en el que estamos trabajando. Dentro de la consola TCL que tenemos al haber abierto la implementación anteriormente, lo que se hace es buscar la carperta Project_1 donde guardamos el fichero *make_mmi_file.tcl* y ejecutamos el siguiente comando: *source make_mmi_file.tcl*. Si se ha seguido los anteriores pasos, se genera un fichero llamado *m1.mmi*, que si se abre con un editor de textos contiene los ocho bloques de memoria de la interfaz ITCM para generar nuestro bitstream.

Con este paso terminado podemos generar el bitstream. Una vez terminado exportamos el Hardware al SDK. Es importante no incluir el bitstream, hacerlo como lo enseña la Figura 125. Para terminar lanzamos el SDK en la misma carpeta donde se exporto el hardware.

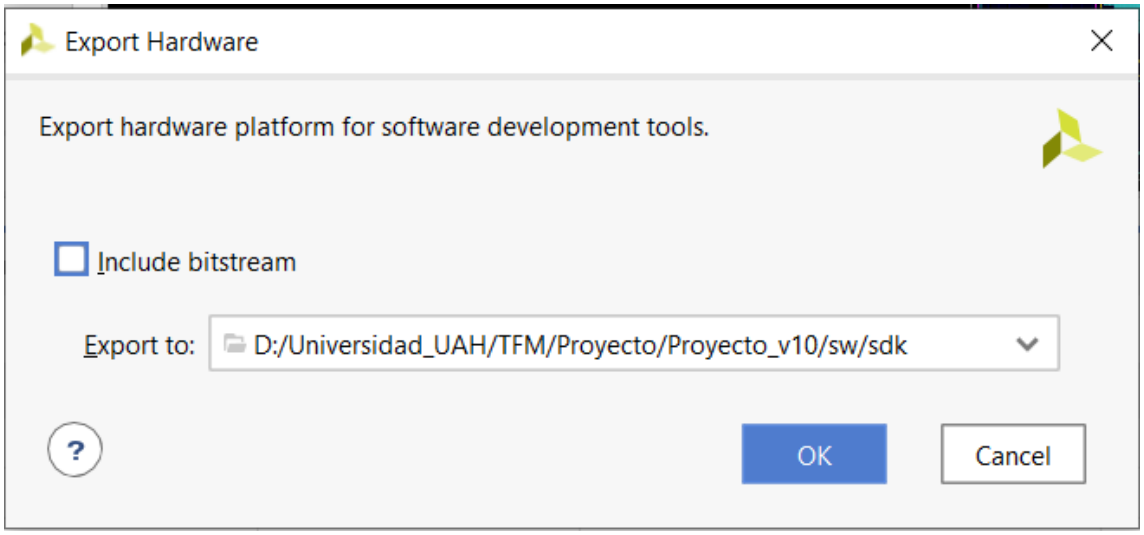


Figura 125. Exportar el Hardware

Ahora lo que se va a hacer es crear un *Borad Support Package* que luego utilizaremos en keil. Antes de esto se tiene que agregar al *Repository* de Xilinx, la carpeta que se

llama *ARM_sw_repository* que tenemos en la carpeta *Vivado.* Entonces, en Xilinx > Repositories, en Global Repositories agregamos dicha carpeta.

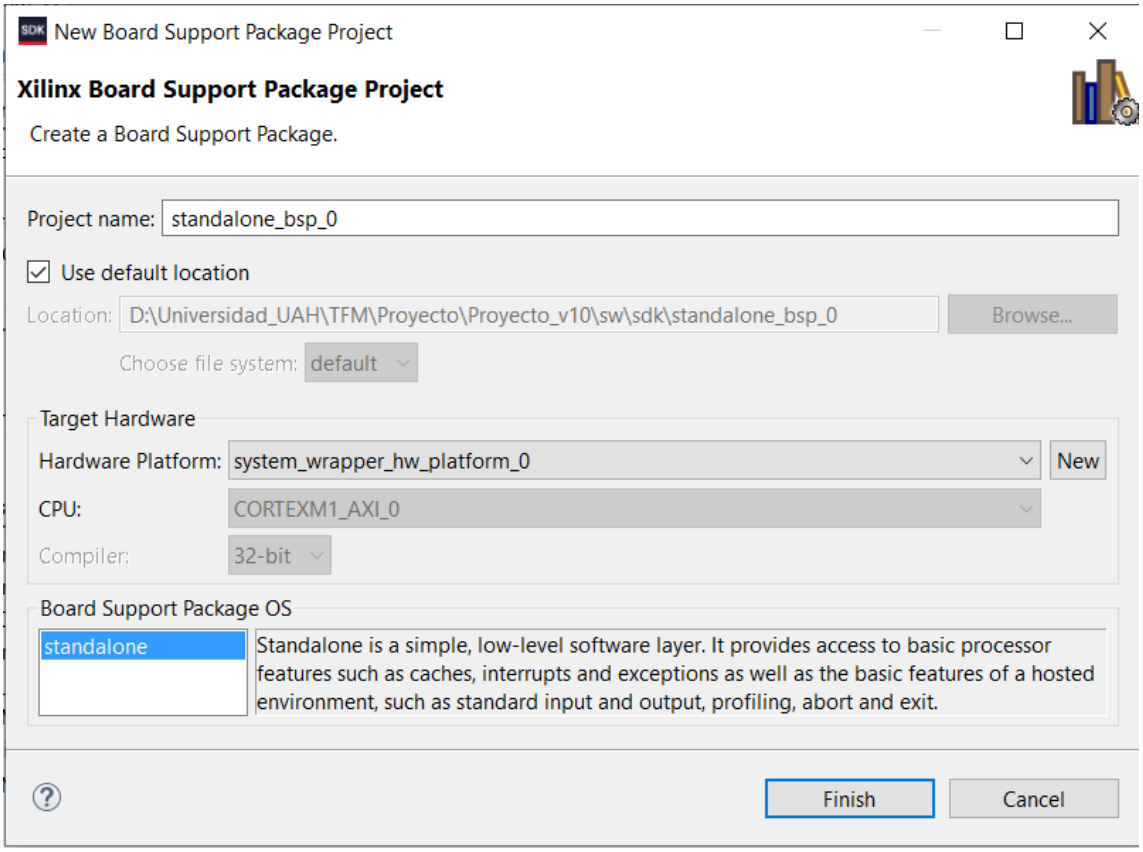


Figura 126. Crear el BSP

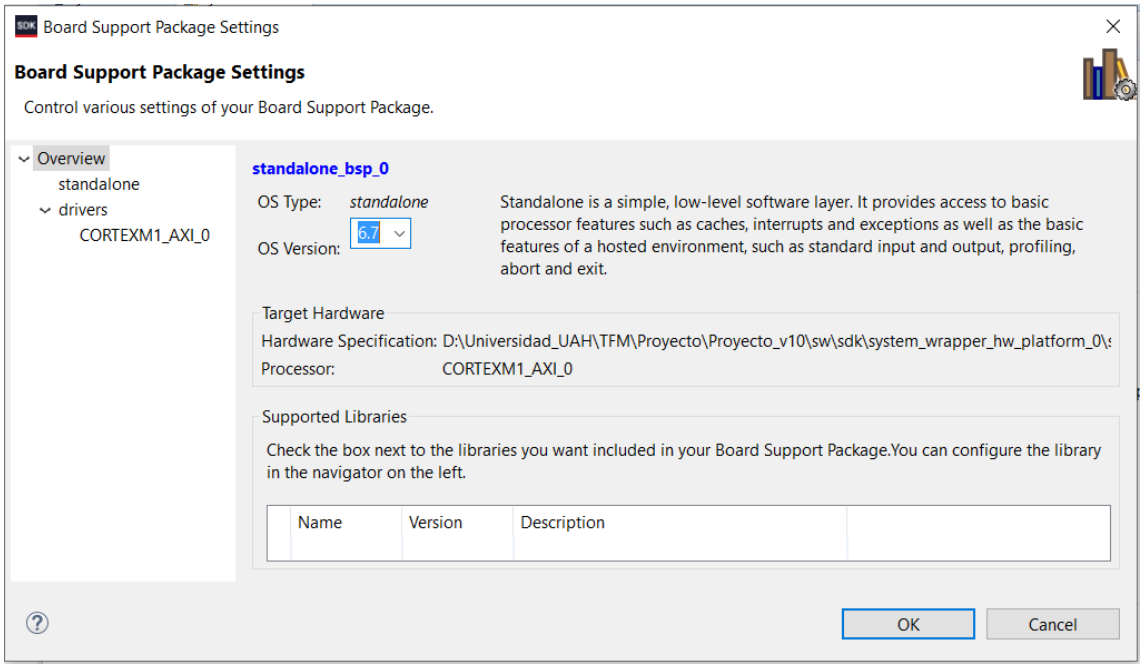


Figura 127. Configurar por standalone v6.7

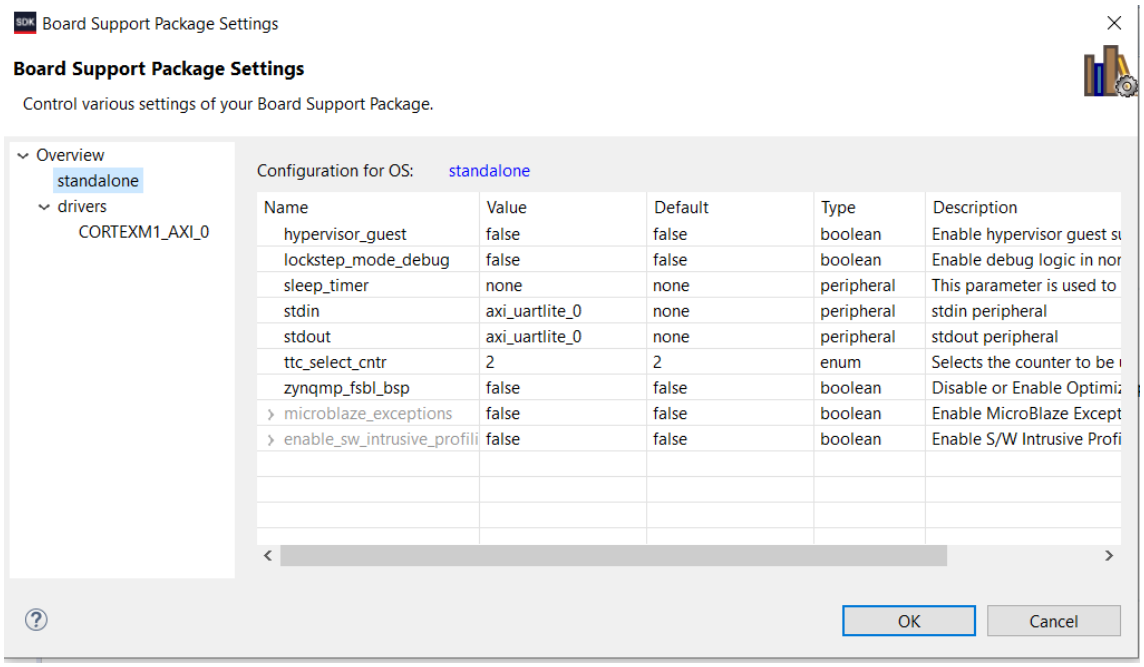


Figura 128. Configurando el bare-metal SO

Se tiene que seleccionar standalone v6.7 y tener especial cuidado en la parte de *stdin* y *stdout*, que sea AXI_uartlite_0 para que no tengamos problemas en el futuro. Una vez generado los drivers, podemos pasar a la parte de KEIL.

El proyecto que se cree de KEIL lo guardaremos en la carpeta keil que está en sw. Como nombre se pone p*roject_1*. Aparecerá una ventana para seleccionar el procesador Cortex-M1 como se ve en la Figura 129, posteriormente se tendrá que elegir dentro de los componentes el *core (en* CMSIS) y el *startup* (Device)como se ve en la Figura 130. Si se abre la pestaña O*ptions for Target*, nos aparece en primer lugar lo mismo que se ve en la Figura 131, como esto es un proyecto sencillo se deja tal como está. En la Figura 132, en la pestaña *Output* se elige *Create HEX file.* Dentro de los comando que se puede ejecutar cuando se hace la compilación, antes o después, como se ve en la Figura 133, se tiene que marcar los dos últimos y añadir los correspondientes comando, esto sirve para generar los ficheros ELF y HEX, que luego se utilizarán en la creación del bitstream. La Figura 134, se ve cómo se han cargado los correspondientes drivers para poder ejecutar el proyecto. Por último, se añade la sentencia que se ve en la Figura 135.

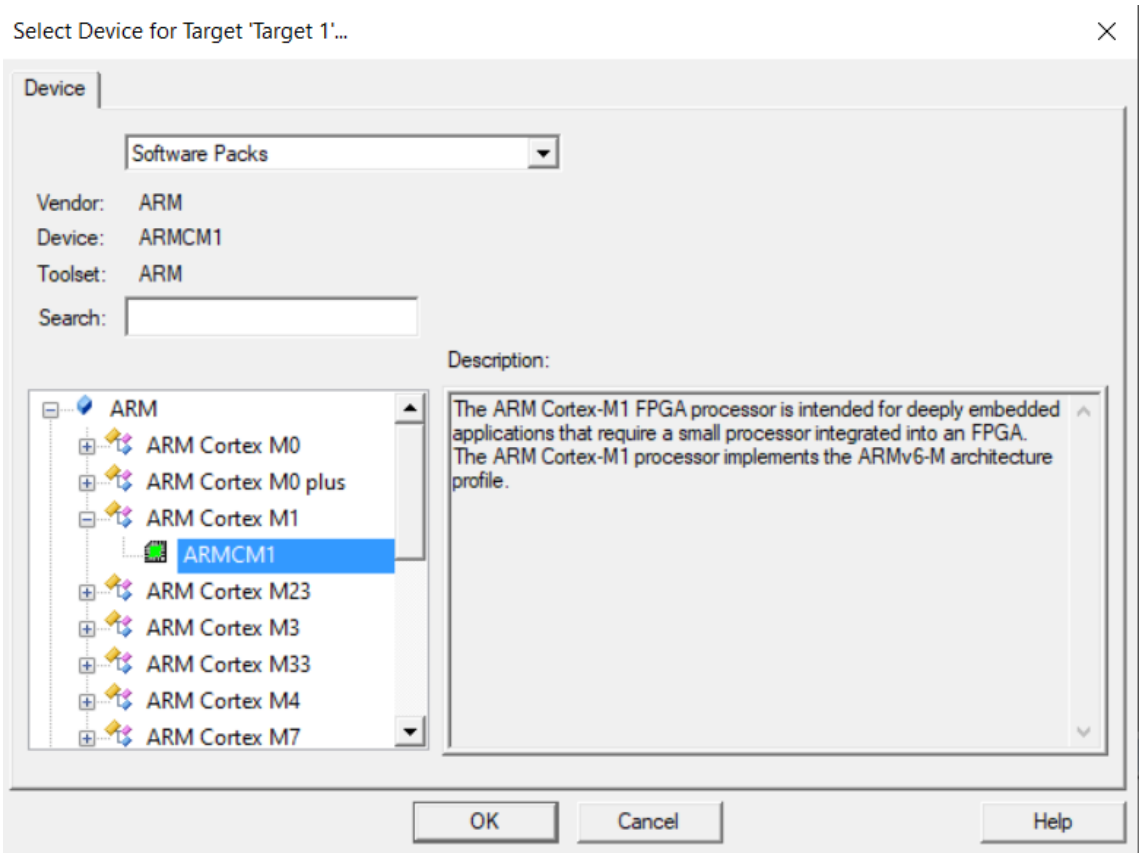


Figura 129. Seleccionamos el Device ARMCM1.

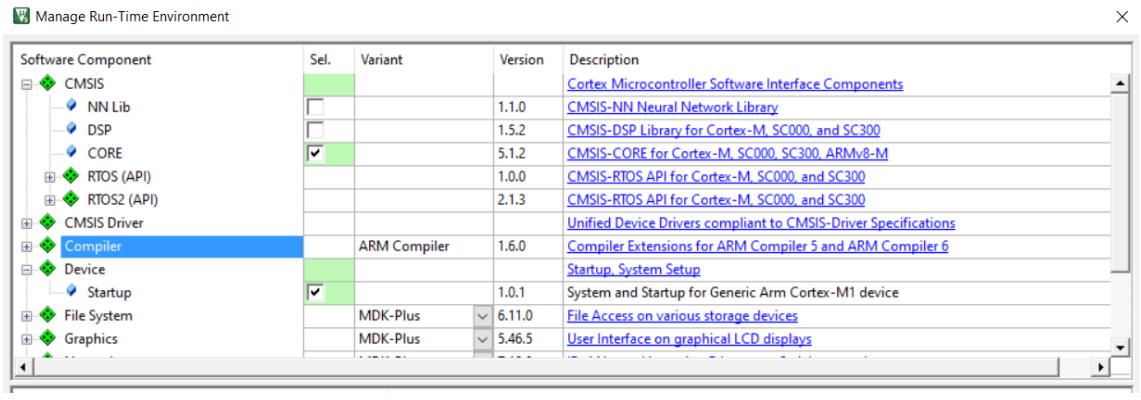


Figura 130. Se selecciona el core y el startup

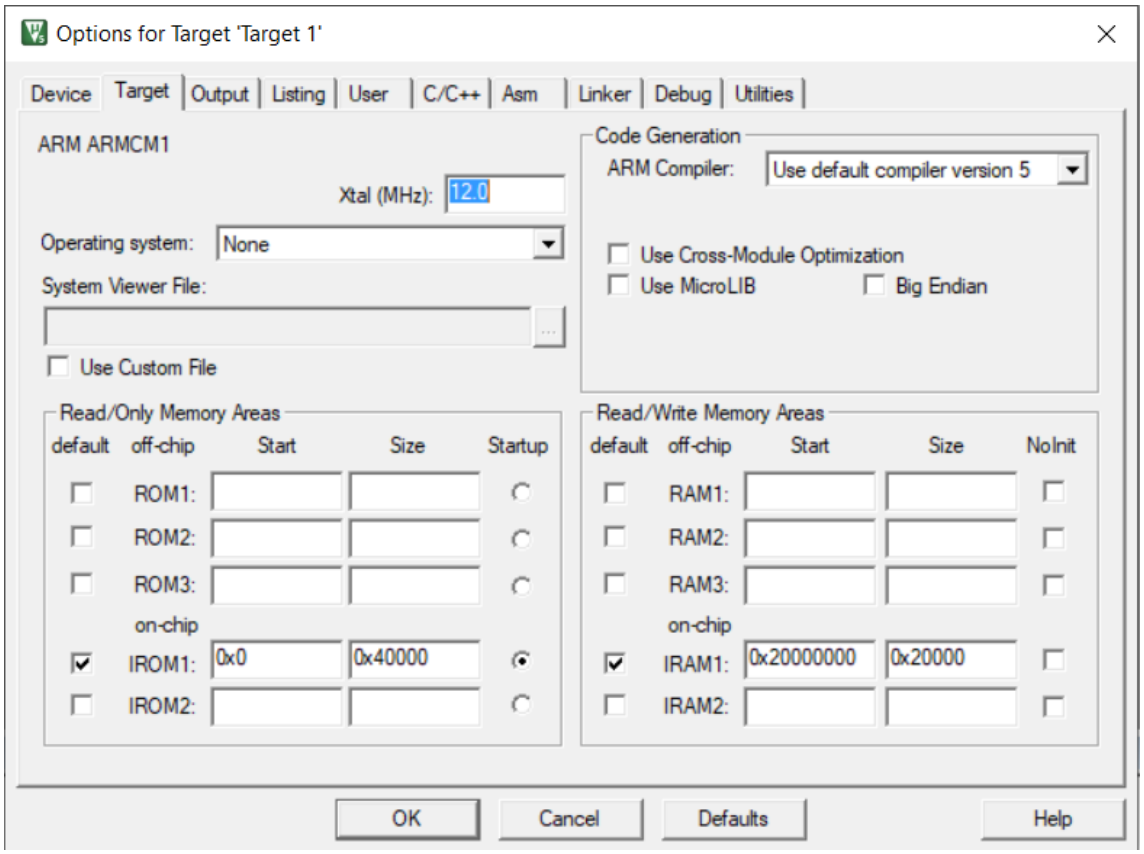


Figura 131. El área de memoria.

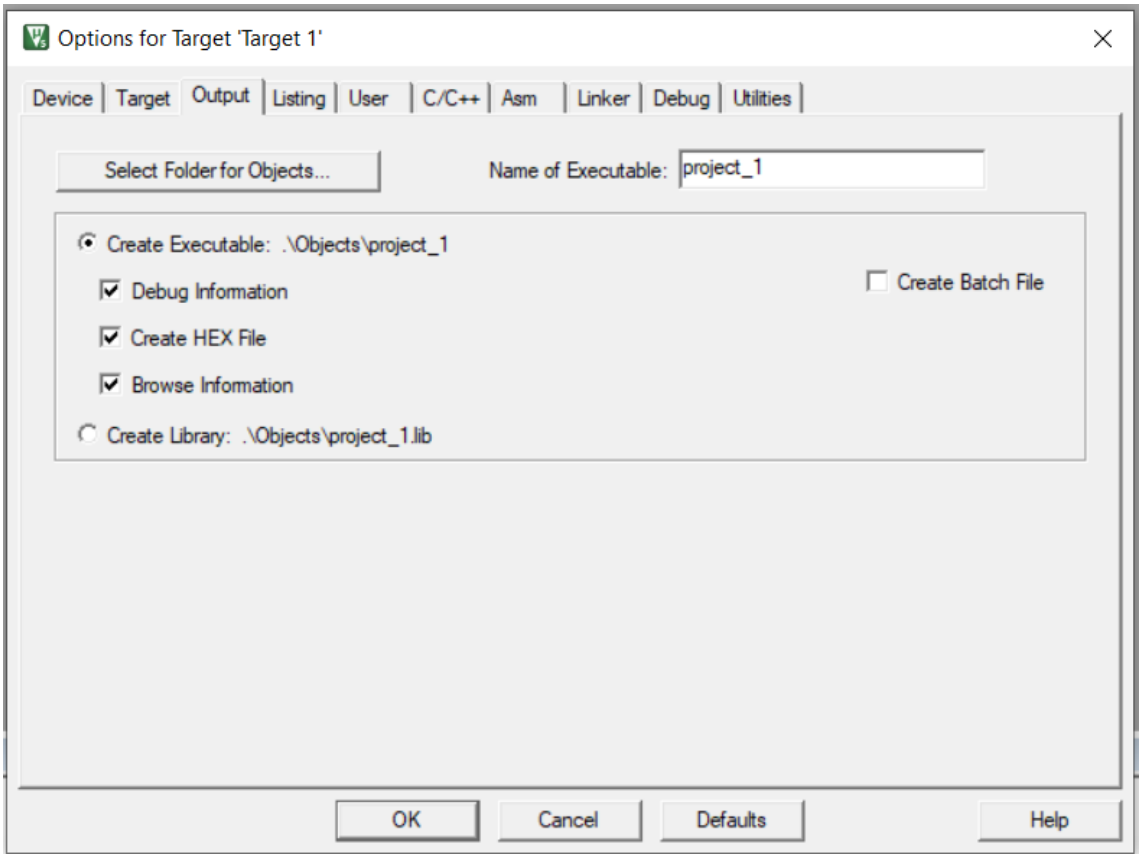


Figura 132. Esperamos que a la salida se genere el fichero HEX

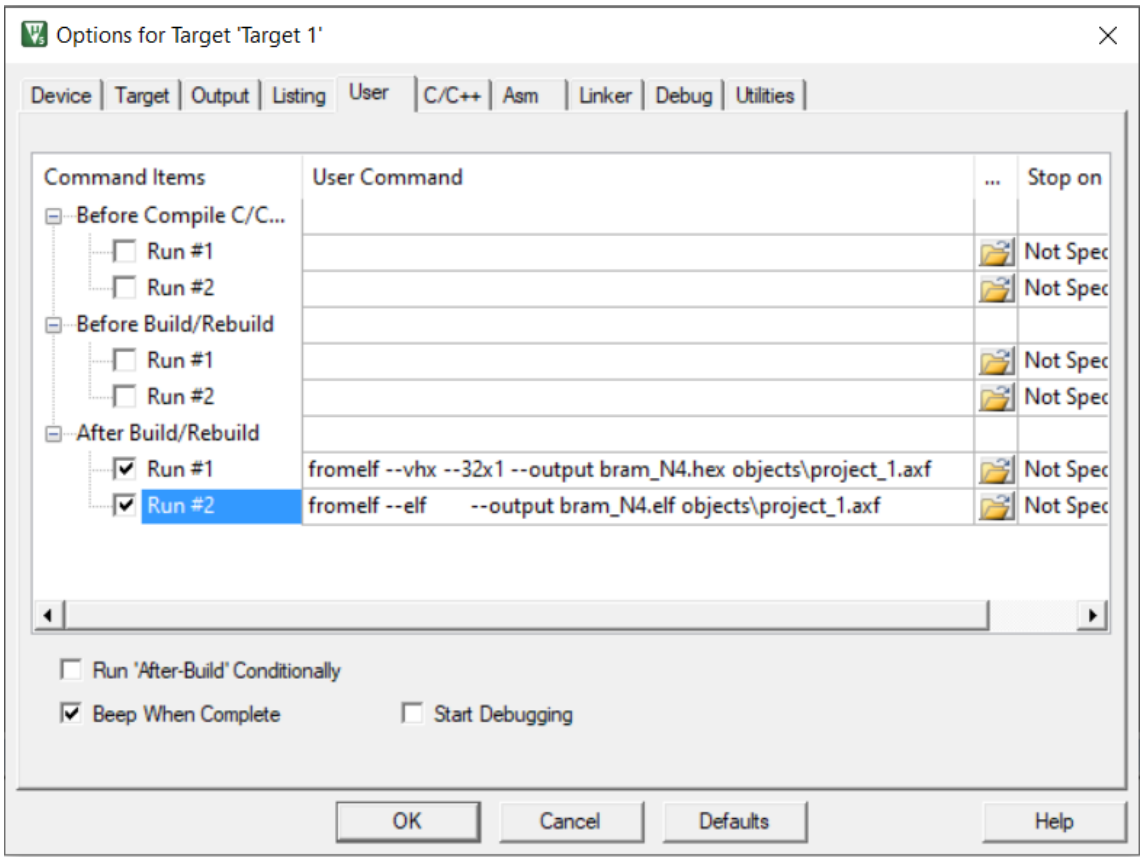


Figura 133. Después del Build se ejecutan los siguientes comandos

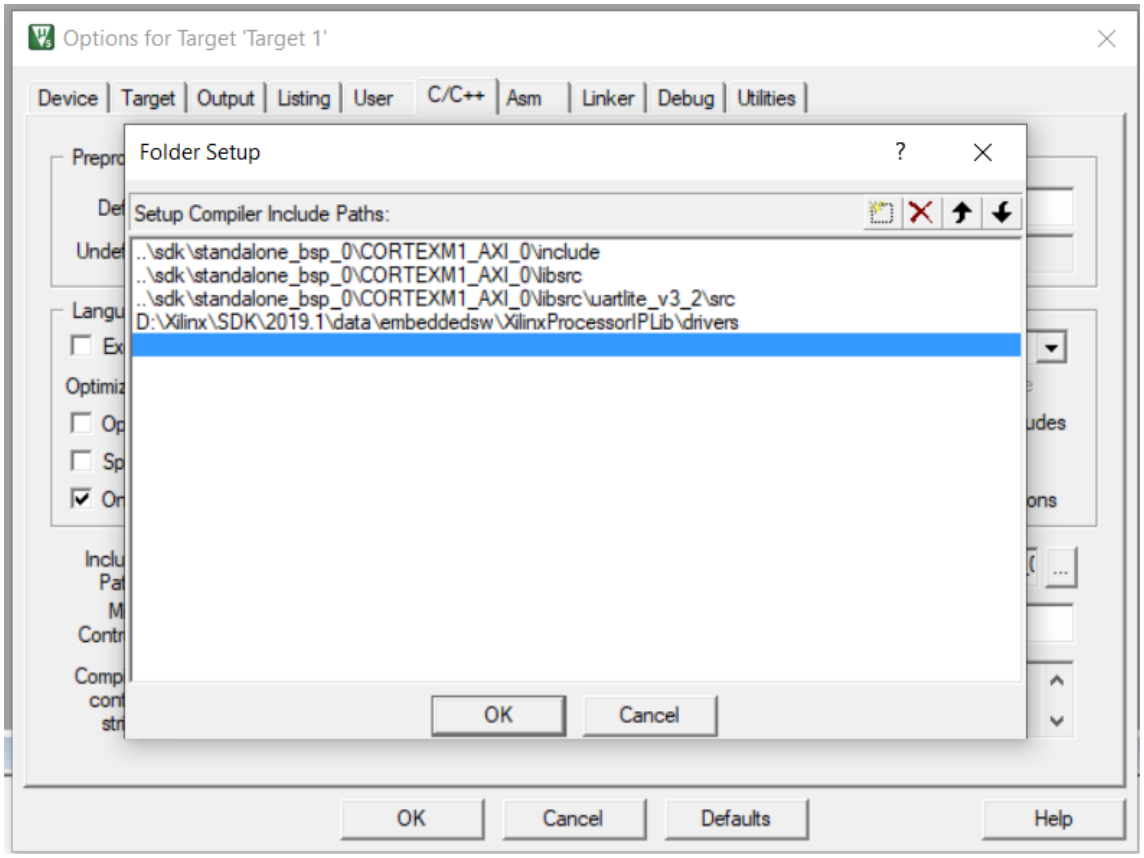


Figura 134. Se agregan los drivers necesarios.

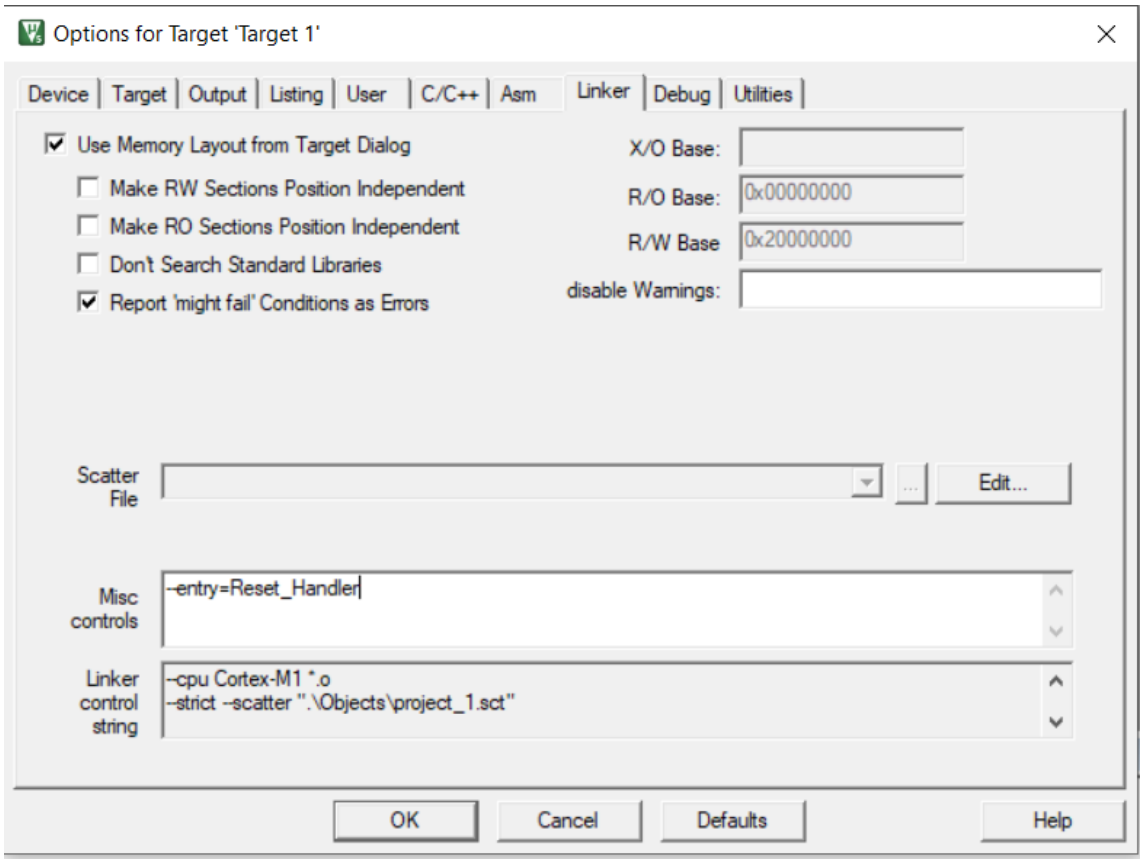


Figura 135. Configuración final.

Lo siguiente que se tiene que hacer es incluir también los drivers en el directorio del proyecto, de esta forma KEIL puede crear los archivos que estamos buscando. Estos drivers los encontramos dentro de la carpeta libsrc, que está en el BSP que se creó en el SDK. Para terminar se crea un main, cuya única línea de código es un print que imprima "Hello World".

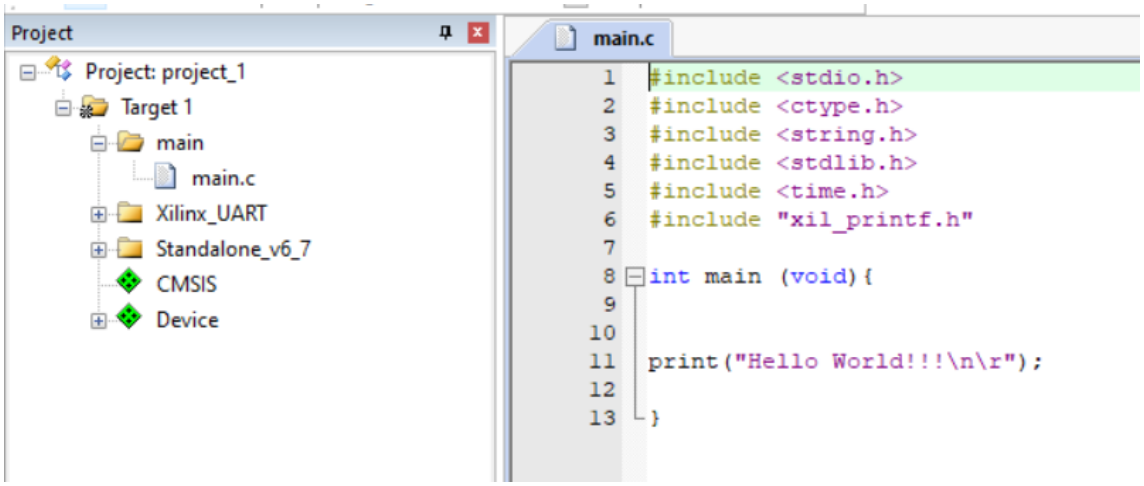


Figura 136. El resultado de todo el proceso en KEIL

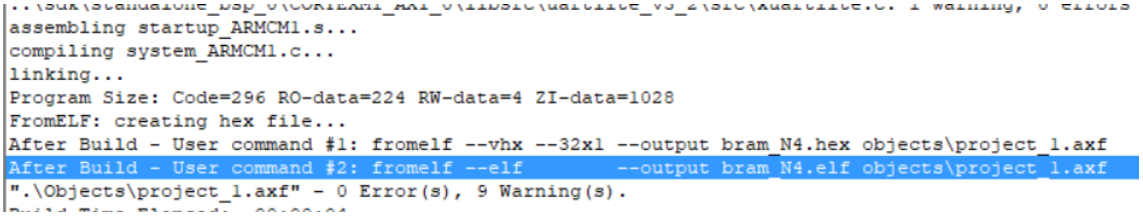


Figura 137. Resultado que enseña la consola.

Como se ve en la Figura 137, se ha compilado y se ha generado los dos ficheros que necesitamos. Estos están en la carpeta KEIL, junto donde reside el proyecto que hemos hecho en este software. Para continuar, se tendrán que copiar los dos ficheros generados, y los pegamos donde la carpeta Project_1 de vivado, en la zona hardware.

Es necesario volver al siguiente path AT472-BU-98000-r0p1-00rel0\hardware\m1_for_arty_a7\m1_for_arty_a7 para copiar los ficheros *make_prog_files.tcl* y *make_prgo_files.bat*, luego llevarlos también a la carpeta de *Project_1*. En el fichero *make_prog_files.tcl* se tiene que editar algunas partes para que se ajuste a lo que se está buscando.

1. Se tiene que cambiar el nombre del fichero elf_file, se cambia por: ./bram_N4.elf
2. Se tiene que cambiar el nombre del fichero bit de origen: ./project_1.runs/impl_1/system_wrapper.bit
3. Se comenta la línea *set reference_bit_file*
4. La línea que da nombre a los ficheros de salida, lo cambiamos por: m1_for_nexys4ddr.bit

Cuando se tengan todos estos cambio hechos, se busca en la barra de herramientas-> Xilinx Design Tools-> Vivado 2019.1 Tcl Shell, y lo llevamos al path de Project_1 de la carpeta hw, donde están todos los ficheros necesarios para crear el bitstream.

En esta parte escribimos: *make_prog_files.bat.* Si todo sale como se espera, saldrá el siguiente mensaje, como se ve en la Figura 138.

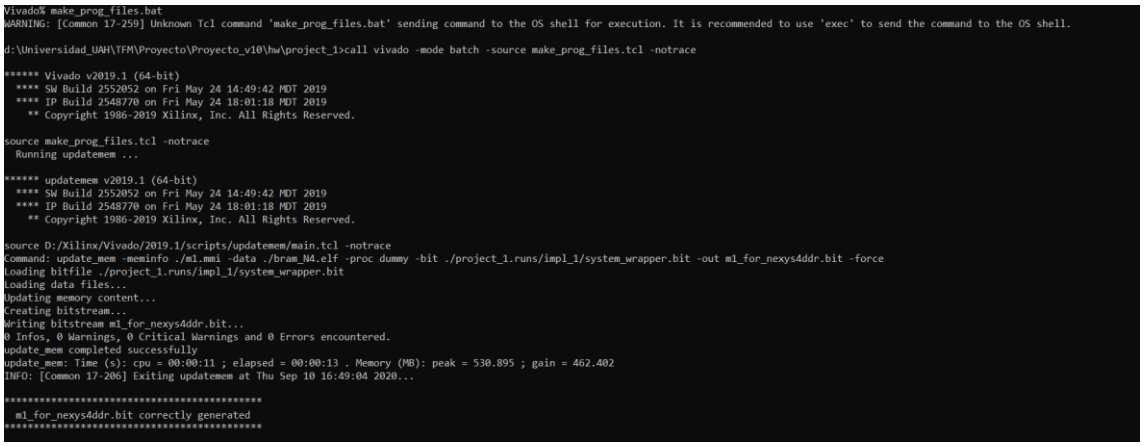


Figura 138. Resultado de crear el bitstream.

Ahora para probarlo en la Nexys 4DDR, en Vivado se tiene que abrir *Open Hardware Manager->Open Target->Auto-Connect->Program Device*, lo que se necesita ahora es cargar el bitstream que se ha generado en el paso anterior. También es necesario tener abierto el Tera Term, como usamos la UART Lite, no hace falta configurar nada, con los valores por defecto vale.

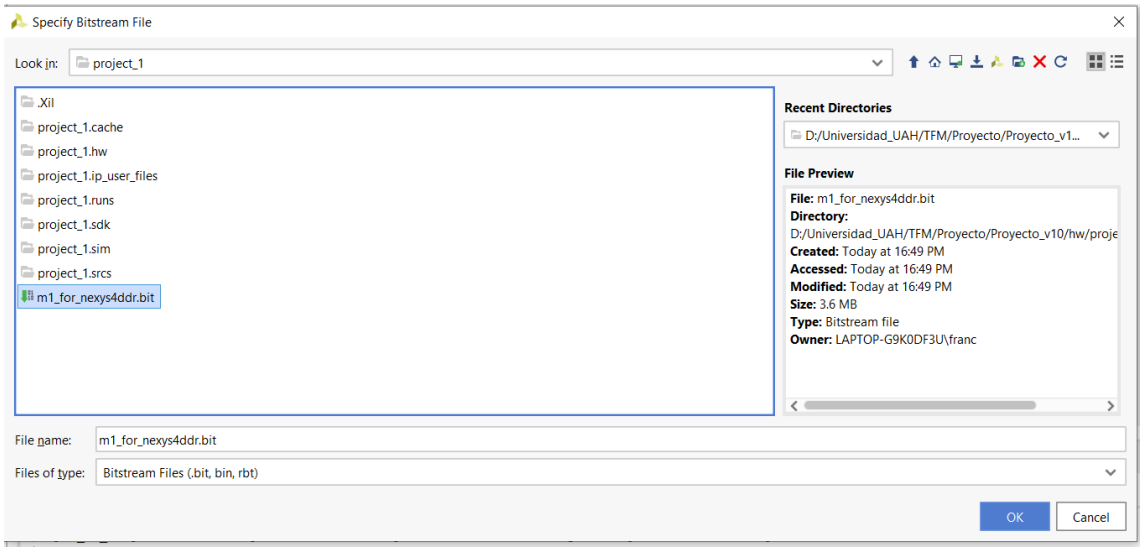


Figura 139. Cargar el bitstream.

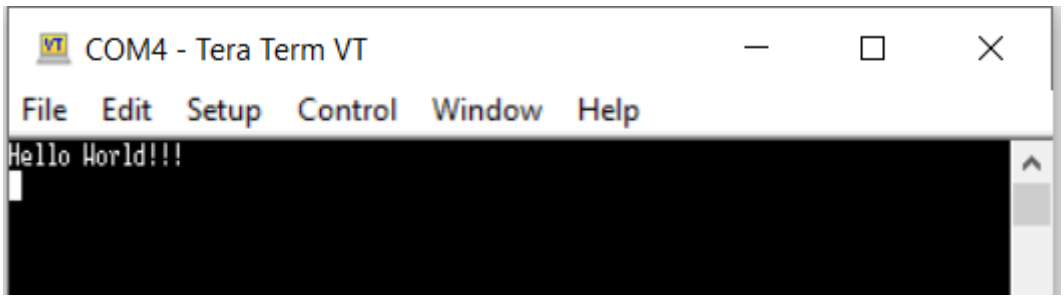


Figura 140. El resultado del projecto.

4.6. Añadir un segundo procesador

Para agregar otro procesador al proyecto básico del “Hello world”, se tiene que seguir siguientes indicaciones.

Como lo que se busca es un proyecto sencillo en el que se capten las ideas importantes, el segundo procesador también va a tener las mismas características que tiene el primero. Entonces, se configurará igual.

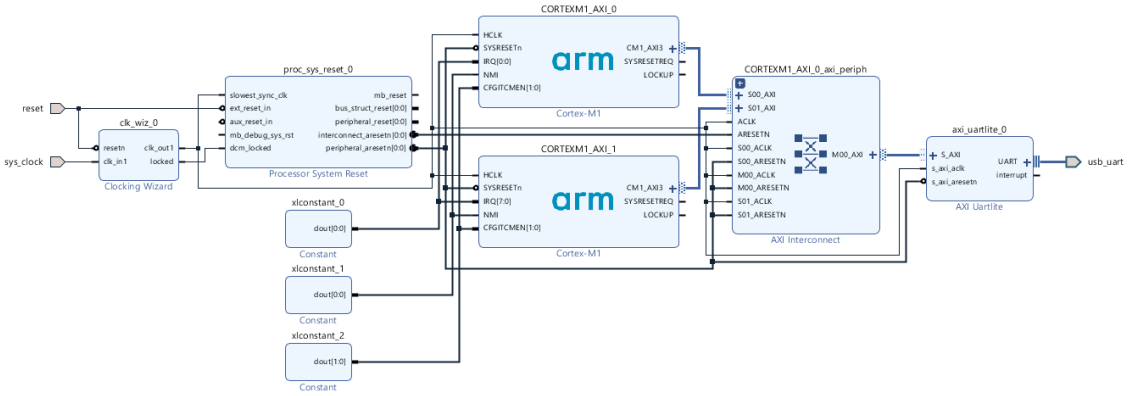


Figura 141. Segundo Cortex-M1

Se va a añadir también otro periférico para el ejemplo, en este caso los dieciséis leds de la Nexys 4 DDR. Se hace el mismo proceso, como en el caso de la UART Lite.

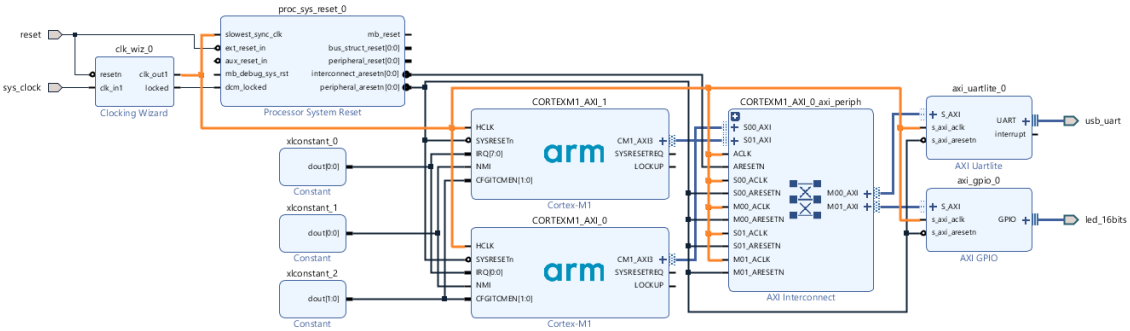


Figura 142. Nuevo periférico al proyecto.

Como se ha añadido un nuevo periférico del tipo GPIO, tendremos que añadir los pines de los leds al fichero de *constrains*, para obtenerlos se puede acudir a la página de GitHub donde está el fichero xdc de la Nexys 4 DDR.

El siguiente paso consiste en ejecutar la *synthesis* y luego la *implementation*. Cuando termine la *implementation*, vamos a crear el fichero MMI. Este fichero se tiene que actualizar cada vez que se hace algún cambio en la parte hardware.

Al buscar los bloques BRAM, como se explicó anteriormente, nos damos cuenta que se tienen el doble de bloques que en la anterior vez, ocho de cada tipo para cada

procesador. Entonces, se tiene que realizar unos ajustes en el fichero make_mmi_file.tcl.

Para evitar confusiones, se van a crear dos ficheros MMI, uno para cada Cortex y luego se unirán de forma manual.

En el fichero para el primer procesador se modificará esta parte:

```
48  # Find all the ITCM RAMs, place in a list
49  set itcm_ram [get_cells -hier -regexp {.*CORTEXM1_AXI_0.*itcm.*ram_block_reg.*} -filter {REF_NAME =~ RAMB36E1}]
```

De esta forma solo buscará los bloques BRAM del primer procesador. Donde se escribe la cabecera del fichero, se tendrá que agregar también al *InstPath:*

system_i/CORTEXM1_AXI_1

```
83  puts $fp "    <Processor Endianness=\"ignored\" InstPath=\"system_i/CORTEXM1_AXI_0\">"
```

Para el segundo se haría lo mismo, tanto para los bloques BRAM,para el InstPath y cambiar el nombre del fichero *MMI*.

Ahora, si se ejecutan en la consola tcl, se deberán de obtener dos ficheros MMI diferentes. Cuando estén generados, en el primer fichero MMI, se tendrá que pegar los bloques que memoria BRAM del otro, como el fichero es de tipo XML, se deberá pegar justo después de </Processor>, es decir, los bloques de memoria están encapsulados dentro de <Processor> y </Processor>. En la Figura 143, se ve la terminación de uno y la continuación del otro.

```
48      </Processor>
49      <Processor Endianness="ignored" InstPath="system_i/CORTEXM1_AXI_1">
50          <AddressSpace Name="system_i.CORTEXM1_AXI_1.inst.u_x_itcm" Begin="0" End="32767">
```

Figura 143. Creación del fichero MMI único.

Generamos el bitstream, exportamos el hardware ya que hemos añadido cosas. En el SDK tras actualizar el BSP que ya habíamos creado, será necesario crear otro para el segundo Cortex añadido. Después de crearlo se pasa a la parte de KEIL.

Para este nuevo procesador, se creará un proyecto independiente al anterior con su correspondiente carpeta que la llamaremos, por ejemplo keil1. La configuración será la misma que se hizo para el primer caso, salvo que para generar los ficheros ELF y HEX tendrán otro nombre

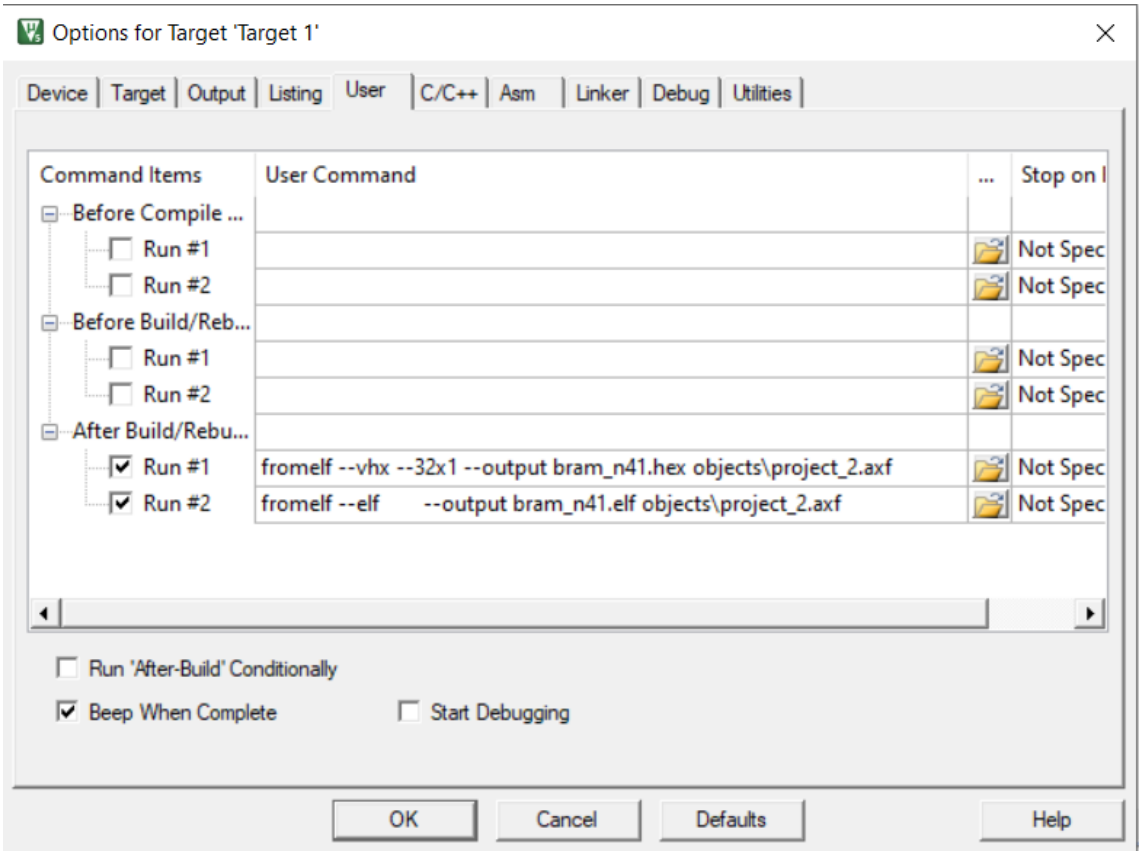


Figura 144. Configuración para obtener ficheros ELF y HEX

En el main lo que se ha hecho es probar los leds, cuando compilamos se generan los ficheros ELF y HEX, por lo que los copiamos y los pegamos en la carpeta Project_1 de la parte hardware, como se hizo en el anterior caso.

Antes de generar el bitstream, se tiene que modificar el fichero *make_prog_files.tcl*. Los cambios que se realizan son:

- Añadir el nuevo fichero ELF:
  ./bram_n41.elf
- Agregar en la línea 72 del fichero lo siguiente:
  En --proc system_i/CORTEXM1_AXI_0 --proc system_i/CORTEXM1_AXI_1.
  Que quede como en la siguiente figura.

```
72  lf_file1 --bit $source_bit_file --proc system_i/CORTEXM1_AXI_0 --proc system_i/CORTEXM1_AXI_1 --out $output_bit_file} result]
73
74  --bit $source_bit_file --proc system_i/CORTEXM1_AXI_0 --proc system_i/CORTEXM1_AXI_1  --out $output_bit_file} result]
```

- Y también esto: --data $elf_file1 a las dos líneas anteriores.

Al realizar estos cambios, ya estamos preparados para ejecutar el bitstream con el fichero BAT en el TCL Shell de Vivado.

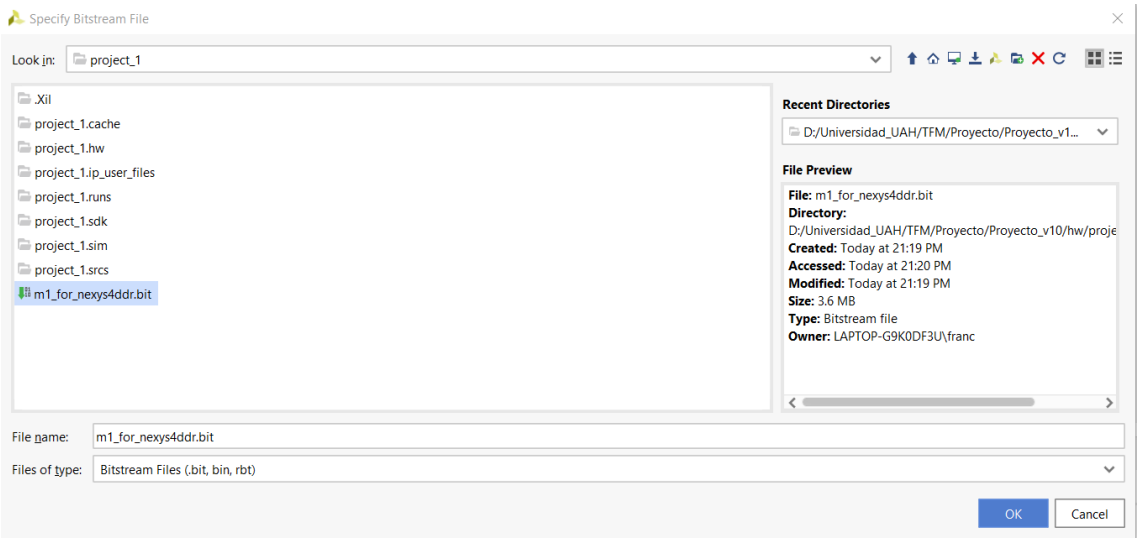


Figura 145. Nuevo bitstream para este caso.

4.7. Memoria Flash

Para integrar la memoria flash a nuestro proyecto con dos procesadores se tendrán que seguir las siguientes indicaciones.

Lo primero que se tiene que hacer es crear una nueva señal de salida de reloj, este reloj tiene que tener una frecuencia de 50 MHz. Posteriormente, se tendrá que buscar la IP de AXI QUAD SPI dentro del catálogo de Vivado. Entonces, en el puerto ***ext_spi_clk*** se conectará a esa señal de reloj que se ha creado anteriormente.

La configuración de la IP se debe de hacer de la siguiente forma. En la Figura 146, se fija como interfaz de la tarjeta *QSPI Flash*, que es la memoria Flash que queremos utilizar. En segundo lugar, como se puede ver en la Figura 147, se elige de modo ***Quad***, y de fabricante ***Spansion***. Cuando se tenga ésta parte hecha, se puede continuar al darle a *Run Connection Automation*, por lo que quedaría lo mismo que en la Figura 148.

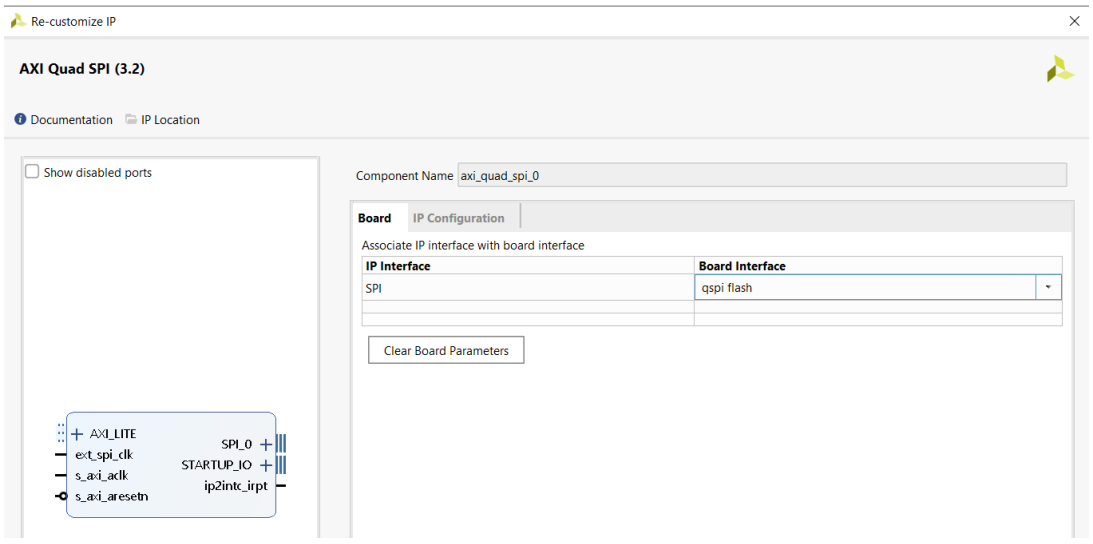


Figura 146. La interfaz de la tarjeta QSPI Flash

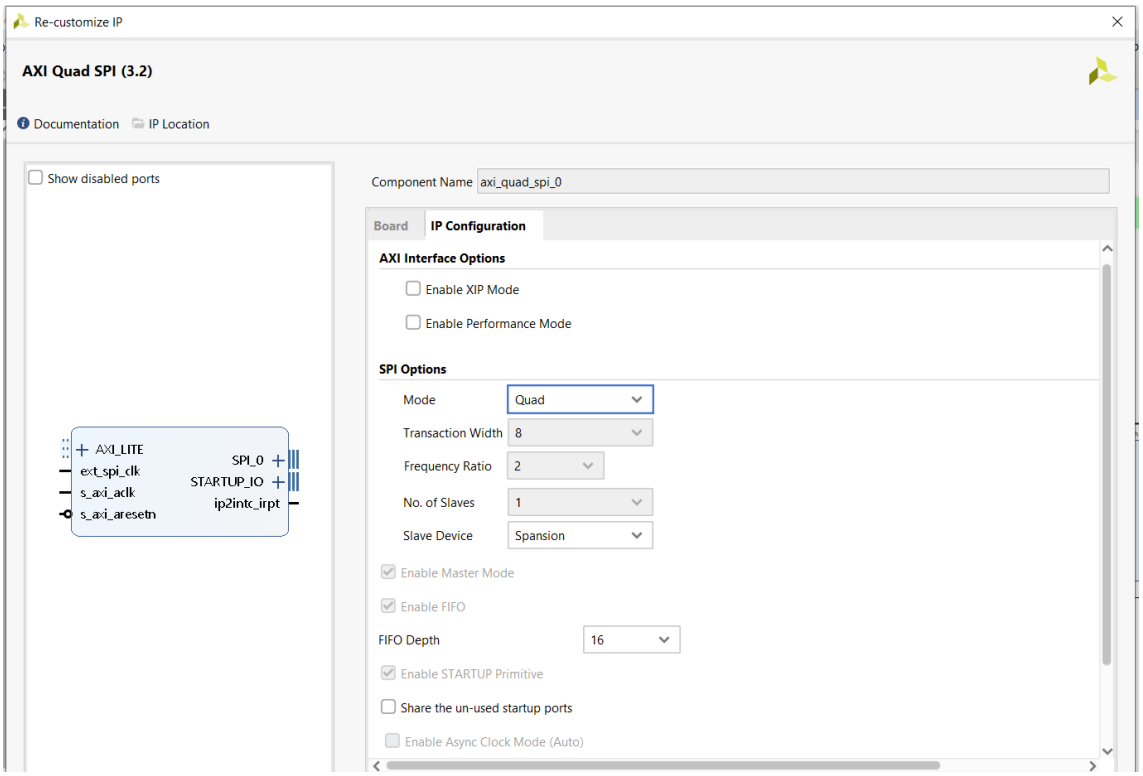


Figura 147. Configuración de la IP

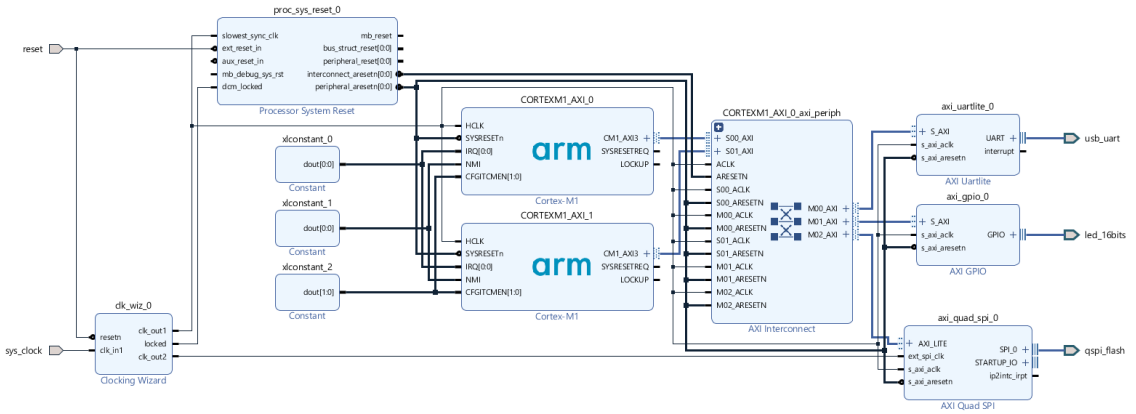


Figura 148. Resultado de añadir la IP AXI QUAD SPI

Para configurar la memoria flash es importante añadir algunos parámetros nuevos en los *constrains*. Como es una operación normal de cargar el bitstream, no se está haciendo para este caso un Multiboot, lo que se debe de añadir a parte de los pines para la SPI es unos parámetros de configuración de la QSPI flash.

Figura 149. Contrains para la memoria flash

Hay dos formas de agregar estos parámetros de configuración al constrains. El primero es hacerlo de forma manual como se ve en la Figura 149. La segunda es hacerla mediante las herramientas que proporciona Vivado en la *implementación,* exactamente en *Edit Device Property.*

Siguiendo con el procedimiento habitual, ahora se tiene que ejecutar la *synthesis* y la *implementation.* Se siguen los mismos pasos de actualizar la memoria MMI como se ha explicado ya anteriormente, se exporta el hardware y se actualizan los drivers.

Ahora se puede generar el bitstream y cargarlo también en la memoria flash. Se hace el mismo procedimiento que antes, con la Shell de vivado para generar el fichero .bit y ahora el MCS que se carga en la flash.

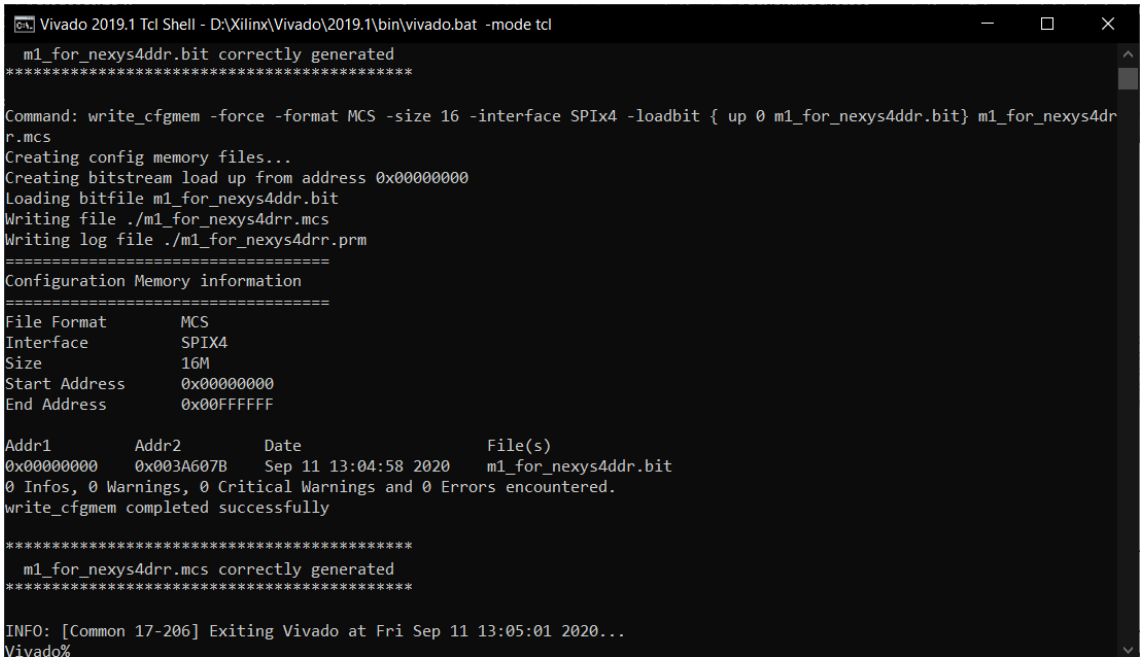


Figura 150. Resultado de generar los dos ficheros.

Ahora para cargar el bitstream dentro de la memoria flash se tienen que hacer dos pasos. Al abrir el *Open Hardware Manager*, lo primero que se debe hacer es tener el jumper ***JP1*** en la posición QSPI de la tarjeta, y programar la tarjeta con el fichero .bit como en los casos anteriores.

Una vez programada la tarjeta con el fichero .bit, pasaremos a la parte de cargar el fichero .mcs. Para ello se tiene que dar a la opción de *Add Configuration Memory Device* como se ve en la Figura 151.

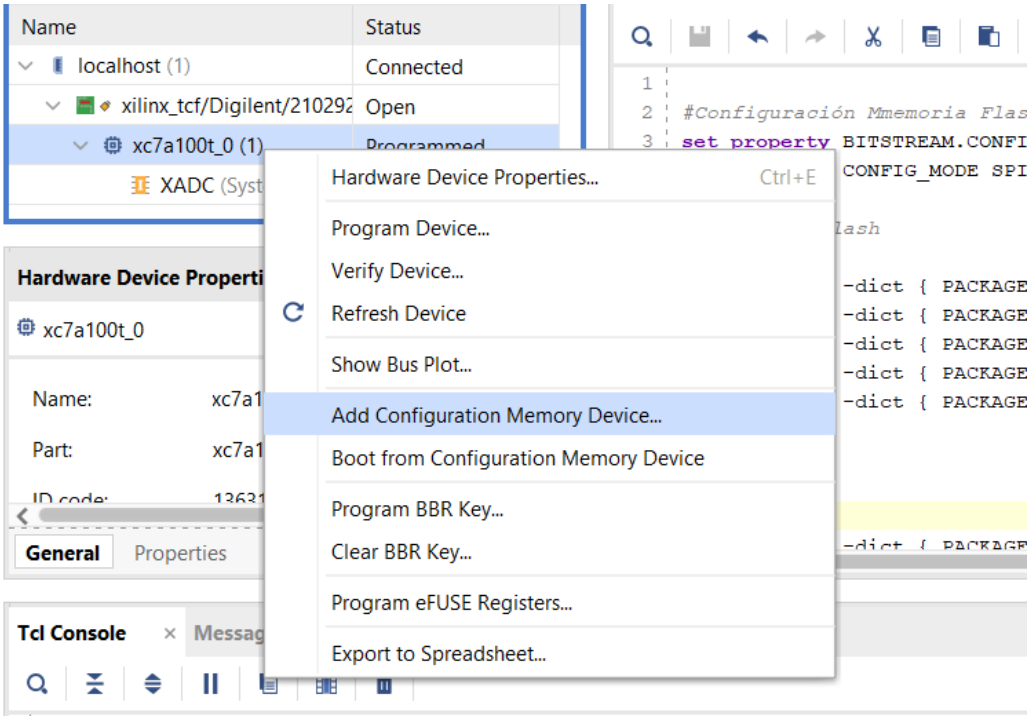


Figura 151. Abrir la configuración de memoria del dispositivo

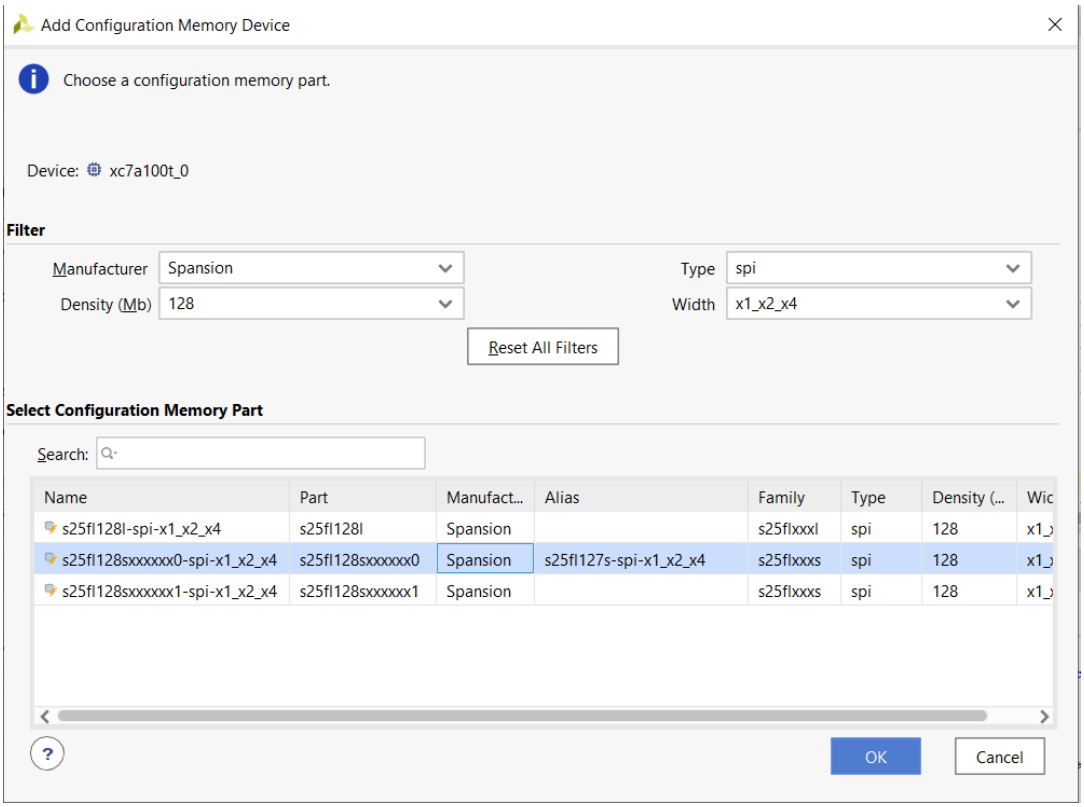


Figura 152. Ventana para seleccionar la memoria flash

En la ventana que aparece en la Figura 152, se tiene que buscar la memoria flash con la que se quiere trabajar, sus características son:

- Manufacter: Spansion
- Type: SPI
- Density(Mb): 128
- Width: x1_x2_x4
- Family: s25flxxxs
- Alias: s25fl127s_spi_x1_x2_x4

Aparecerá un mensaje como el que se ve en la Figura 153, se pulsa ok para continuar con la configuración.

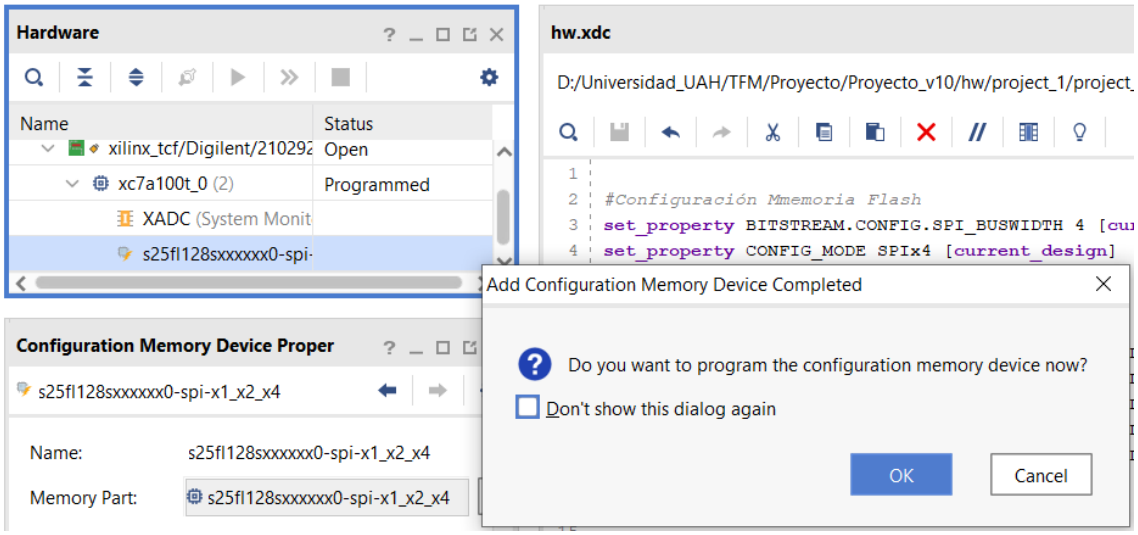


Figura 153. Ventana que continua la configuración.

Vuelve a aparecer una ventana como se ve en la Figura 154, aquí es donde se carga el fichero .mcs que se creó anteriormente. Antes de seguir debemos verificar que el jumper ***JP1*** está en la posición JTAG. Cuando ya esté seleccionado el fichero .mcs y el jumper en su sitio, se pulsa ok para seguir.

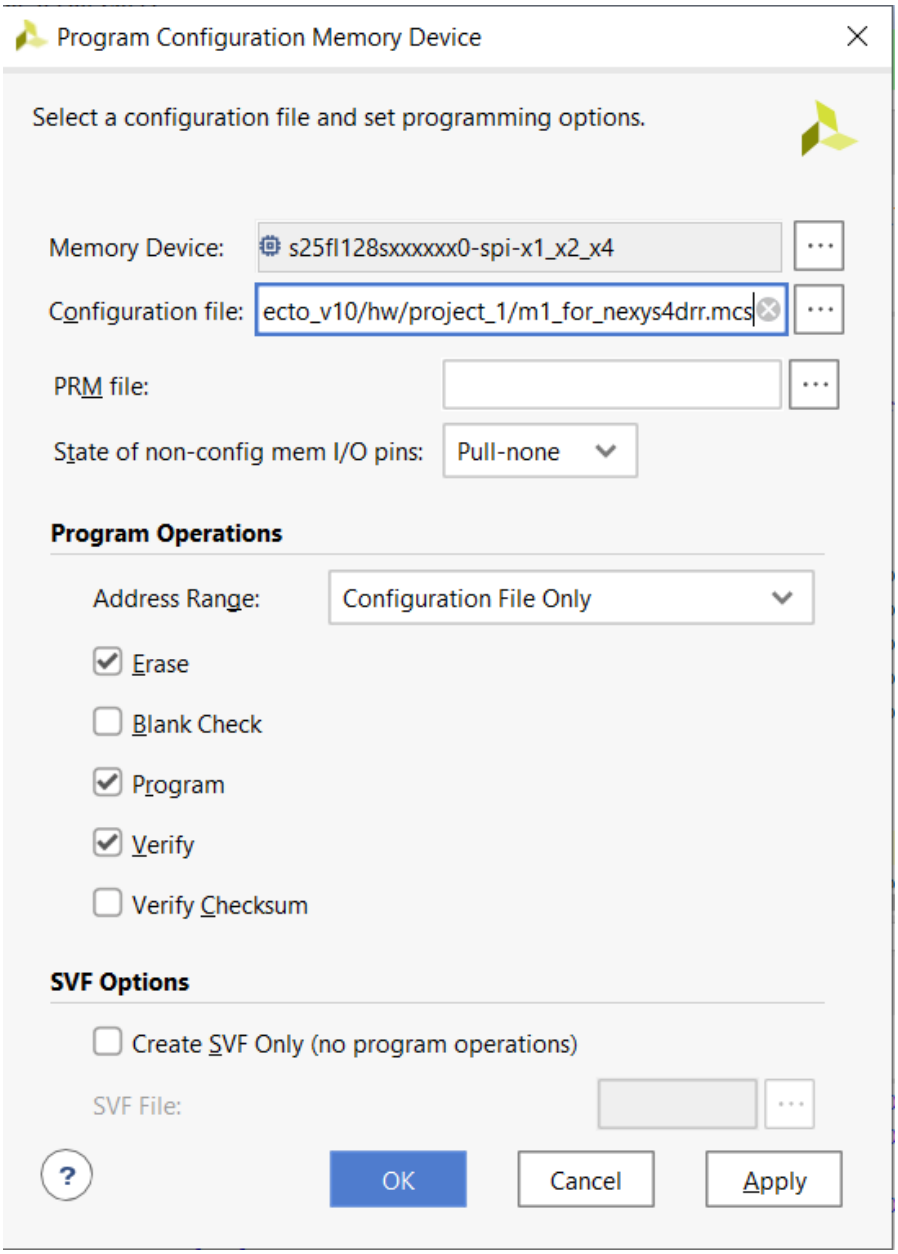


Figura 154. Cargar fichero mcs

Si todo ha salido bien, aparece un mensaje como el de la Figura 155.

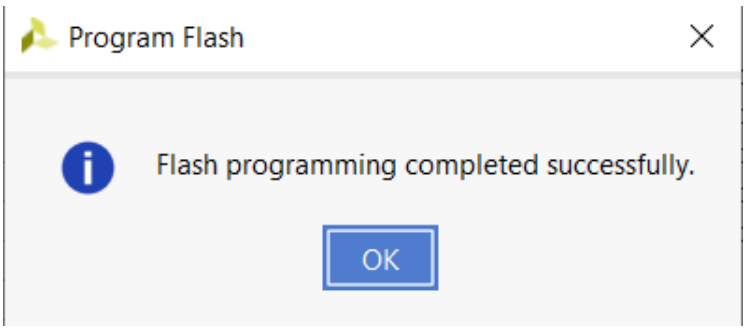


Figura 155. Programación de la flash correcta

Para comprobar que todo ha salido según lo esperado, se apaga la tarjeta y se desconecta. Se pone el jumper ***JP1*** en la posición QSPI, se enciende de nuevo la tarjeta, se pulsa el botón que tiene la tarjeta llamado ***PROG***, se enciende un led ***BUSY*** y luego un led ***DONE***, a continuación se ejecuta el mismo bitstream que hemos cargado.

4.8. Memoria DDR2

La memoria DDR2 utilizará una frecuencia de 200 MHz, por este motivo que de la IP *Clock Wizard* se saca otra señal de reloj de salida con esa frecuencia. Para añadir la parte de la memoria DDR2 es necesario utilizar la IP de Xilinx llamada MIG (Memory Interface Generator).

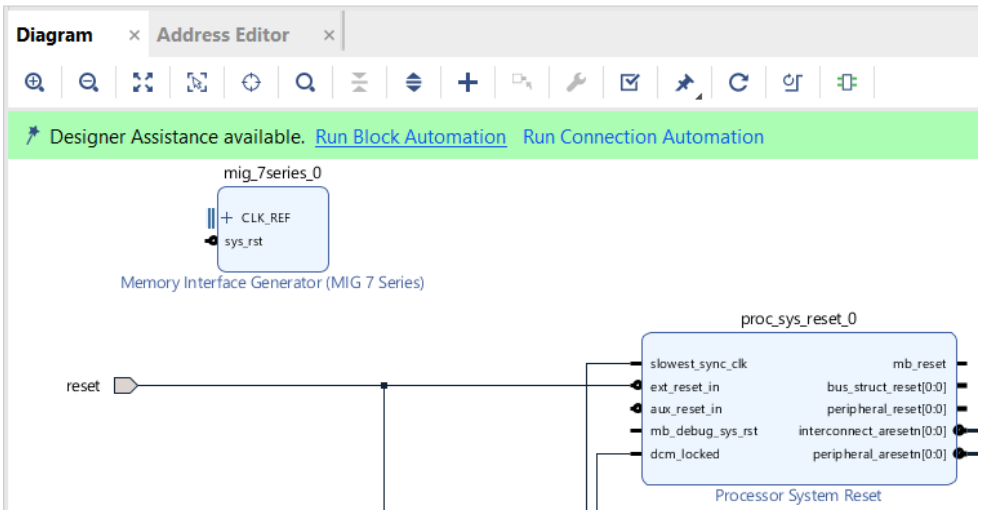


Figura 156. IP del MIG

Si nos fijamos en la Figura 156, aparece la opción de *Run Block Automation,* se tiene que dar click, y luego OK.

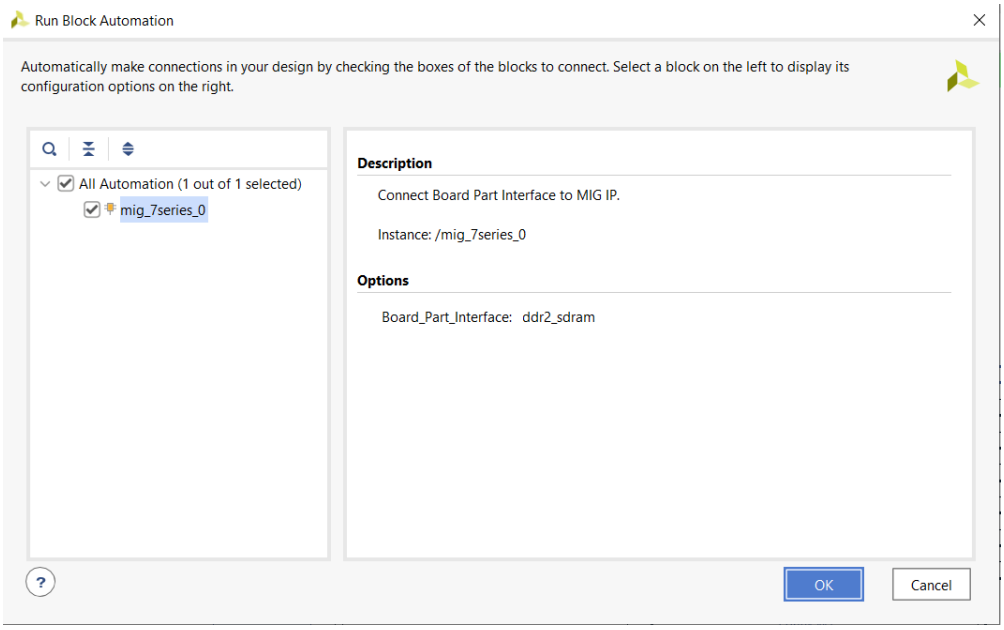


Figura 157. Ventana de Run Block Automation

Justo después de dar OK, aparecerá una nueva ventana con un mensaje de error. No es un error grave, tiene fácil solución. Este mensaje aparece porque faltan aún elementos por configurar, de momento se puede ignorar.

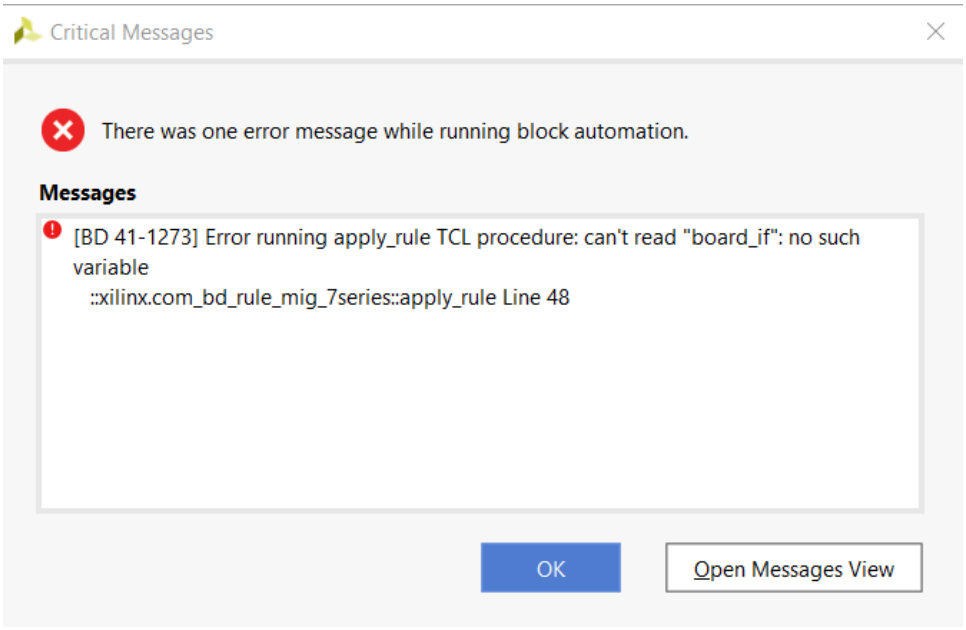


Figura 158. Mensaje de error del Run Block Automation

La señal de reloj que se generó de 200 MHz se tiene que conectar al puerto *sys_clk_i*, una vez hecho este paso se puede seleccionar la opción de *Run Connection Automation.*

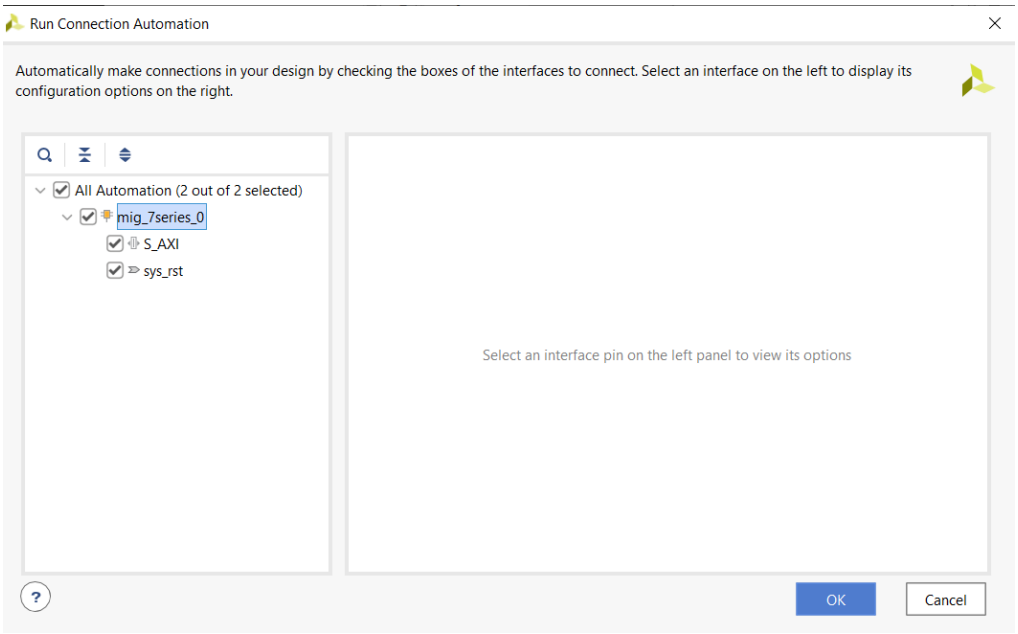


Figura 159. Run Connection Automation para la MIG

Para solucionar el problema anterior, se puede hacer añadiendo los pines necesarios en el fichero XDC. Después de esto, seleccionamos en la pestaña *Board* la *DDR2 SDRAM* y la conectamos a la MIG.

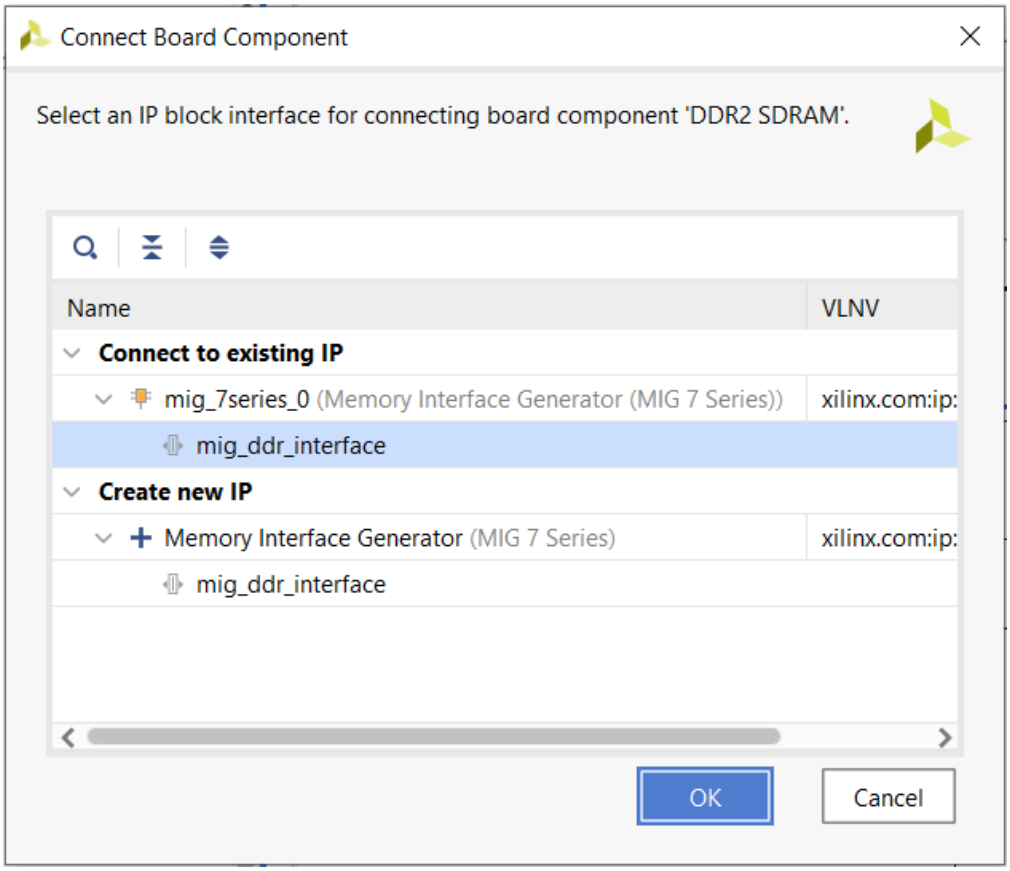


Figura 160. conectamos la DDR2 a la MIG

Para comprobar que se han resuelto los problemas y el diseño es correcto, damos a la opción de validar el diseño (F6). La configuración de la MIG se encuentra en el apartado de la memoria DDR2. A continuación, se hace la *synthesis* y la *implementation.*

Los siguientes pasos son los mismos que antes, se exporta el hardware al SDK. Y las comprobaciones a nivel software se verán más adelante.

4.9. Agregar dos micros más

Ahora el objetivo es añadir dos microprocesadores más a todo el diseño que se lleva hecho. Para ello sólo hace falta seguir las mismas indicaciones que se hizo para el caso, en el que se añadió un microprocesador más. Recordemos que se tienen que generar ahora dos ficheros más MMI, para los dos nuevos microprocesadores; y que al final los tendremos que añadir a un único fichero.

Luego en la parte del SDK, de tendrá que hacer lo mismo de crear un BSP para cada micro. Y por último tendremos que crear un proyecto Keil para los dos nuevos procesadores, para que estos nos generen los ficheros .hex y .elf y al final se pueda hacer el bitstream general.

V. Anexo

5.1. Interfaz de comunicación

5.1.1. APB (Advanced Peripheral Bus):

Señales básicas del protocolo APB:

Señales	Dirección	Descripción
PCLK	Fuente de reloj -> Todos los bloques APB	Señal de reloj común. El flanco de subida de PCLK se realizan las transferencias en el APB.
PRESETn	Fuente reset -> Todos los bloques APB	Señal de reset común activo a nivel bajo. Normalmente está conectado directamente a la señal de reset del sistema.
PSEL	Decodificador de direcciones-> Esclavo	Selecciona el esclavo. En el puente APB se genera esta señal para cada periférico esclavo del bus. Indica que el esclavo se ha seleccionado y que sus datos son solicitados.
PADDR[n:0]	Maestro->Esclavo	Bus de direcciones. Puede ser hasta 32 bits de ancho.
PENABLE	Maestro->Esclavo	Control de transferencia. Esta señal indica el siguiente y subsiguiente ciclos de la transferencia.
PWRITE	Maestro->Esclavo	Escribe comando de control (Dirección)(1=Escribe,0=Lee).
PPROT[2:0]	Maestro->Esclavo	Control de transferencia de control, presente en la versión AMBA 4. Indica el nivel de protección de la transacción, ya sea una transacción de acceso a datos o de instrucciones.
PSTRB[n-1:0]	Maestro->Esclavo	Verificación de la escritura. Esta señal indica que bytes se deben de actualizar durante la lectura. No debe de estar activa durante la lectura.
PWDATA [31:0]	Maestro->Esclavo	Escribe los datos. Este bus es dirigido por el puente del bus de los periféricos durante la escritura cuando PWRITE está activo.
PRDATA[31:0]	Maestro<-Esclavo	Lectura de los datos. Del esclavo seleccionado se toman los datos durante la transferencia de lectura y mientras la señal PWRITE esté a nivel bajo.
PSLVERR	Maestro<-Esclavo	Esta señal indica si hay un fallo durante la transferencia.
PREADY	Maestro<-Esclavo	El esclavo utiliza esta señal para indicar que está preparado para una transferencia.

Anexo 1.Lista de señales de APB.

5.1.2. AHB (Advanced High-performance Bus)

Señales	Dirección	Descripción
HCLK	Fuente de reloj -> Todos los bloques AHB	Todas las señales están relacionadas con el flanco de subida.
HRESETn	Fuente reset -> Todos los bloques APB	La señal de reset es activa a nivel bajo.
HSEL	Decodificador de direcciones-> Esclavo	Cada esclavo AHB tiene su propia señal y esta señal indica que la transferencia actual está relacionada con dicho esclavo.
HADDR[31:0]	Maestro->Esclavo	Es la señal de direcciones del sistema de 32 bits.
HTRANS[1:0]	Maestro->Esclavo	Esta señal indica el tipo de la transferencia actual, puede ser NONSEQUENTIAL, SEQUENTIAL o IDLE.
HWRITE	Maestro->Esclavo	1=Escritura y 0=Lectura.
HSIZE[2:0]	Maestro->Esclavo	Esta señal indica el tamaño de la transferencia, que normalmente es de un byte, media palabra o una palabra.
HBURST[2:0]	Maestro->Esclavo	Esta señal indica que la transferencia forma parte de una ráfaga.
HPROT[3:0]/[6:0]	Maestro->Esclavo	La señal de control indica que si la transferencia es una búsqueda de código de operación o un acceso a datos, y si la transferencia esta en modo de supervisor o el modo de acceso del usuario.
HMASTLOCK	Maestro->Esclavo	Indica si el control de la transferencia está bloqueado.
HMASTER[3:0]		Indica la identidad del master del bus actual.
HWDATA[31:0]	Maestro->Esclavo	El bus de datos de escritura es usado para transferir datos desde el master al bus de los esclavos durante las operaciones de escritura.
HRDATA[31:0]	Maestro<-Esclavo	El bus de datos de lectura es usado para los datos de transferencia de lectura.
HRESP[1:0]/HRESP	Maestro<-Esclavo	La transferencia de respuesta proporciona información adicional en el estado de la misma.
HREADY	Maestro<-Esclavo	Cuando esta a nivel alto indica que la transferencia se ha completado en el bus. Si por lo contrario, está a nivel bajo es que espera una transferencia.

Anexo 2. Lista de señales de APB.

5.1.3. AXI (Advanced eXtensible Interface)

Se van a enumerar las señales más típicas que se utilizan tanto en los protocolos AXI3 y AXI4.

Señales globales

Señales	Dirección	Descripción
ACLK	Fuente de reloj -> Todos los bloques AXI	Señal global de reloj.
ARESETn	Fuente reset -> Todos los bloques APB	Señal de reset activa a nivel bajo.

Anexo 3.Señales globales.

Canal Write Address

Señal	Fuente	Descripción
AWID	Master	Escribe la dirección del master. Esta señal indica la identificación asociada para el grupo de señales de este canal.
AWADDR	Master	Dirección de escritura. La dirección de escritura proporciona la dirección de la primera transferencia de escritura cuando es modo ráfaga.
AWLEN	Master	Longitud de la ráfaga.
AWPROT	Master	Tipo de protección de la transferencia y también el nivel de seguridad.
AWVALID		Dirección de escritura valida. Esta señal indica que la dirección de escritura y la información de control es válida.
AWREADY	Esclavo	Esta señal indica que el esclavo está preparaso para aceptar una dirección y las señales de control asociadas. La dirección de escritura está preparada.

Anexo 4.Señales del canal write address, las más importantes.

Canal Write data

Señal	Fuente	Descripción
WID	Master	Esta señal indica la identificación asociada para el grupo de señales de este canal.
WDATA	Master	Dato de escritura.
WSTOBES	Master	Señales de validación de escritura. Indica cual canal de bytes es un dato valido.
WLAST	Master	Indica la última transferencia en modo de escritura en ráfaga.

WUSER	Master	Señal de usuario.
WVALID	Master	Esta señal indica que el dato escrito es válido y que las señales de control están disponibles.
WREADY	Esclavo	Indica que el esclavo puede aceptar datos de escritura.

Anexo 5.Señales del canal write data.

Canal Write Response

Señal	Fuente	Descripción
BID	Esclavo	Esta señal indica la identificación asociada para el grupo de señales de este canal.
BRESP	Esclavo	Esta señal indica el estado de la transacción de escritura.
BUSER	Esclavo	Señal de usuario.
BVALID	Esclavo	Esta señal indica que el canal tiene una respuesta de escritura válida.
BREADY	Master	Indica que el master puede aceptar una respuesta de escritura.

Anexo 6.Señales del canal write response.

Canal Read Address

Señal	Fuente	Descripción
ARID	Master	Esta señal indica la identificación asociada para el grupo de señales de este canal.
ARADDR	Master	La dirección de lectura proporciona la dirección de la primera transferencia de lectura cuando es modo ráfaga.
ARLEN	Master	Indica el número exacto de la longitud de la transferencia en ráfaga.
ARPROT	Master	Tipo de protección de la transferencia y también el nivel de seguridad.
ARVALID	Master	Esta señal indica que la dirección de lectura y la información de control es válida.
ARREADY	Esclavo	Esta señal indica que el esclavo está preparado para aceptar una dirección y las señales de control asociadas.

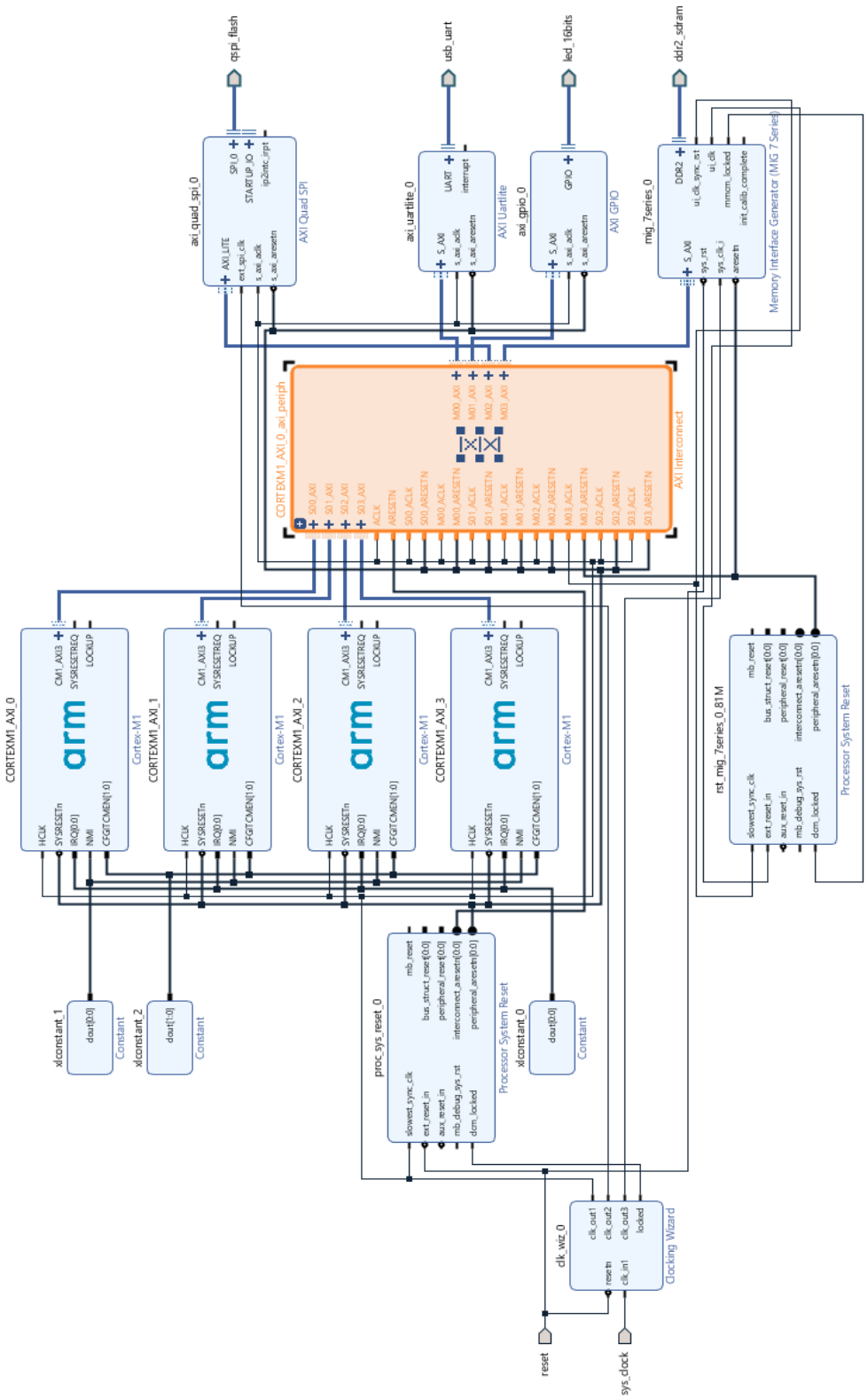
Anexo 7.Señales del canal read address.

Canal Read Data

Señal	Fuente	Descripción
RID	Esclavo	Esta señal indica la identificación asociada para el grupo de señales de este canal.
RDATA	Esclavo	Dato de lectura.
RRESP	Esclavo	Esta señal indica el estado de la transferencia de lectura.
RLAST	Esclavo	Indica la última transferencia en modo de lectura en ráfaga.
RUSER	Esclavo	Señal de usuario.
RVALID	Esclavo	Esta señal indica que el canal esta señalando a un dato de lectura válido.
RREADY	Master	Indica que el master puede aceptar datos de lectura e información de respuesta.

Anexo 8.Señales del canal read data.

5.2. Sistema final.



Anexo 9. Sistema final.

Bibliografía

[1] E. d. l. T. A. y. M. D. V. P. Juan José Rodríguez Andina, Fundamentals, Advanced Features,and Applications in Industrial Electronics, CRC Press, 2017.

[2] ARM, «ARM Developer,» ARM, [En línea]. Available: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m1. [Último acceso: 13 Octubre 2020].

[3] ARM, «Technical Reference Manual Cortex™-M1 Revision: r0p1,» ARM, 2006-2008.

[4] «Cortex™-M1 Technical Reference Manual Revision: r1p0,» ARM, 2006-2008.

[5] ElectronicDesign, «ElectronicDesign,» [En línea]. Available: https://www.electronicdesign.com/industrial-automation/article/21807078/free-access-to-softcore-cortexm-designs-for-xilinx-fpga-users. [Último acceso: 13 Octubre 2020].

[6] XILINX, «XILINX,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/fpga/spartan-7.html. [Último acceso: 13 Octubre 2020].

[7] P. Burr y S. George, «XILINX,» 2018. [En línea]. Available: https://www.xilinx.com/publications/events/developer-forum/2018-frankfurt/bringing-the-benefits-of-cortex-m-processors-to-fpga.pdf. [Último acceso: 16 Octubre 2020].

[8] XILINX, «Zynq 7000,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Último acceso: 13 Octubre 2020].

[9] XILINX, «Zynq Ultrascale,» [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html#productAdvantages. [Último acceso: 13 Octubre 2020].

[10] XILINX, «Artix-7,» [En línea]. Available: https://link.springer.com/book/10.1007/978-3-030-11961-4. [Último acceso: 13 Octubre 2020].

[11] P. P. Chu, FPGA PROTOTYPING BY VHDL EXAMPLES Xilinx MicroBlaze MCS SoC Second Edition, WILEY, 2017.

[12] J. YIU, System-on-Chip Design with Arm® Cortex®-M Processors Reference Book, Cambridge,: ARM Education Media, 2019.

[13] ARM, «PrimeCell® Infrastructure AMBA™ 2 AHB™ to AMBA 3 AXI™ Bridges,» ARM, 2004.

[14] XILINX, «AXI Reference Guide,» XILINX, 2011.

[15] J. Yiu, The Definitive Guide to Arm® Cortex®-M0 and Cortex-M0+ Processors (Second Edition), Newnes, 2015.

[16] ARM, «Cortex-M1 TCM Initialization Considerations for System Designers,» 2009.

[17] I. Kuon, R. Tessier y J. Rose, FPGA Arquitecture: Survey and challenges, Massachusetts: Now the essence of knowledge, 2008.

[18] H. Amano, Principles and Structures of FPGAs, Singapore: Springer, 2018.

[19] Digilent, «Nexys4 DDR™ FPGA Board Reference Manual,» Abril 2016. [En línea]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start. [Último acceso: 15 Octubre 2020].

[20] Gisselquist Technology,LLC, «Gisselquist Technology,LLC,» 16 Agosto 2018. [En línea]. Available: https://zipcpu.com/blog/2018/08/16/spiflash.html. [Último acceso: 15 Octubre 2020].

[21] A. Yang, «Using SPI Flash with 7 Series FPGAs,» XILINX, 2020.

[22] Cypress Semiconductor Corporation, «S25FL128S, S25FL256S,» Cypress, San Jose, 2015.

[23] XILINX, «7 Series FPGAs Configuration User Guide,» XILINX, 2018.

[24] Q. Jian, Y. Liu Liansheng:Peng y L. Datong, «Optimized FPGA-based DDR2 SDRAM controller,» de *IEEE 11th International Conference on Electronic Measurement & Instruments*, Harbin, 2013.

[25] Transcend, «Transcend,» [En línea]. Available: https://www.transcend-info.com/Support/FAQ-296%208.Referencia%20memoria%20DDR2. [Último acceso: 15 Octubre 2020].

[26] Tektronix, «SDRAM Memory Systems: Architecture Overview and Design Verification,» Junio 2013. [En línea]. Available: https://download.tek.com/document/54W_21473_3_HR_Letter.pdf. [Último acceso: 16 Octubre 2020].

[27] Micron, «DDR2 SDRAM,» Micron, 2007.

[28] XILINX, «Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions User Guide,» 5 Diciembre 2018. [En línea]. Available: https://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v4_2/ug586_7Series_MIS.pdf. [Último acceso: 16 Octubre 2020].

[29] B. Schmidt, J. Gonzalez-Dominguez, C. Hundt y M. Schlarb, «Message Passing Interface,» de *Parallel Programming*, Elsevier , 2018, p. 315–364.

[30] F. Pires, M. Vestias y H. Neto, «An Implementation of MPI on FPGA for Distributed Memory Multiprocessing,» Lisboa, 2016.

[31] F. Pires, «Distributed-Memory Multiprocessors in FPGAs,» Lisboa, 2015.

[32] D. Wiklund, S. Sathe y D. Liu, «Benchmarking of On-Chip Interconnection Networks,» IEEE, Linköping, Sweden.

[33] S. Emo, L. Vesa, K. Kimmo y H. Timo, «OVERVIEW OF BUS-BASED SYSTEM-ON-CHIP INTERCONNECTIONS,» IEEE, Tampere, Finland.

Universidad de Alcalá

Departamento de Electrónica

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá