

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**MÁSTER UNIVERSITARIO EN INGENIERÍA
DE SISTEMAS ELECTRÓNICOS
TRABAJO FIN DE MÁSTER**

**DISEÑO DE PRÁCTICAS DE
LABORATORIO EN LA PLATAFORMA
NEXYS A7 EMPLEANDO EL
PROCESADOR VIRTUAL XILINX
MICROBLAZE**

**LUCAS DE MIGUEL ANGUITA
2022**

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Título: Diseño de prácticas de laboratorio en la plataforma Nexys A7 empleando el procesador virtual Xilinx Microblaze

Autor: D. Lucas de Miguel Anguita

Tutor: D. Pablo Ituero Herrero

Ponente: D.

Departamento:

MIEMBROS DEL TRIBUNAL

Presidente: D.

Vocal: D.

Secretario: D.

Suplente: D.

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de:
.....

Madrid, a de de 20...

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**MÁSTER UNIVERSITARIO EN INGENIERÍA
DE SISTEMAS ELECTRÓNICOS
TRABAJO FIN DE MÁSTER**

**DISEÑO DE PRÁCTICAS DE LABORATORIO
EN LA PLATAFORMA NEXYS A7
EMPLEANDO EL PROCESADOR VIRTUAL
XILINX MICROBLAZE**

**LUCAS DE MIGUEL ANGUITA
2022**

RESUMEN

Este proyecto surge porque se ha detectado una carencia formativa en los egresados de los planes de estudio actuales en cuanto al empleo de procesadores virtuales (soft-processor) y todas las metodologías y capacidades de diseño asociadas incluyendo su uso junto a bloques descritos en lenguajes de descripción hardware.

Partiendo de los recursos disponibles en el departamento, que cuenta con una gran cantidad de placas Nexys A7, se decidió utilizar esta plataforma, de manera conjunta con las herramientas de desarrollo de Xilinx y su procesador Microblaze. La placa cuenta con numerosas interfaces de entrada y salida y una memoria DDR2, lo que proporciona un amplio abanico de entrenamiento.

El principal reto del proyecto ha sido la puesta en marcha del flujo de trabajo completo que incluye la personalización del procesador virtual, el acceso a diferentes periféricos, la utilización conjunta del procesador virtual y bloques descritos en VHDL y la depuración de código. Para este fin, se han consultado y ordenado diferentes fuentes bibliográficas disponibles en la documentación técnica de Xilinx y diversos sitios de internet. Una vez la nueva metodología se ha asentado y testado bajo distintos escenarios, se han realizado unos nuevos tutoriales basados en la experiencia.

Por lo tanto, el resultado principal de este proyecto será una colección de tutoriales, pensados para realizarse entre 1-2 horas, que vayan avanzando en el diseño hardware utilizando metodologías progresivamente más complejas. Finalmente se planteará una práctica de nivel medio que englobe todas las metodologías cubiertas por los tutoriales.

SUMMARY

This project was born because a lack of training has been detected in the graduates of current curricula in the use of virtual processors (soft-processors) and all the associated design methodologies and capabilities, including their use together with blocks described in hardware description languages.

Based on the resources available in the department, which has a large number of Nexys A7 boards, it was decided to use this platform, in conjunction with Xilinx development tools and its Microblaze processor. The board has numerous input and output interfaces and DDR2 memory, which provides a wide range of training.

The main challenge of the project was the implementation of the complete workflow, including the customization of the virtual processor, the access to different peripherals, the joint use of the virtual processor and the blocks described in VHDL, and the code debugging. For this purpose, various bibliographic sources available in the Xilinx technical documentation and various Internet sites were consulted and ordered. Once the new methodology was established and tested under different scenarios, new tutorials were created based on the experience gained.

Therefore, the main result of this project will be a collection of tutorials, intended to be done in 1-2 hours, that will advance in hardware design using progressively more complex methodologies. Finally, a mid-level practice will be proposed that encompasses all the methodologies covered by the tutorials.

PALABRAS CLAVE

MicroBlaze, Procesador virtual, Módulo IP, FPGA, VHDL, Nexys A7, DDR2.

KEYWORDS

MicroBlaze, Virtual Processor, IP Module, FPGA, VHDL, Nexys A7, DDR2.

AGRADECIMIENTOS

A mi familia.

*A mis amigos y compañeros de universidad y del máster por hacer el paso por la universidad más
ameno y divertido.*

A mi tutor, Pablo Ituero, por haberme ayudado todas las veces que lo he necesitado.

ÍNDICE DEL CONTENIDO

1. INTRODUCCIÓN Y OBJETIVOS.....	1
1.1. Introducción	1
1.2. Objetivos	2
1.3. Metodología	2
1.4. Desarrollo temporal	4
1.5. Metodología docente de las prácticas	5
1.6. Prácticas descartadas.....	5
2. PRÁCTICA 0: INTRODUCCIÓN AL ENTORNO	6
2.1. Introducción	6
2.2. Objetivos de la práctica.....	6
2.3. Desarrollo.....	6
3. PRÁCTICA 1: INTRDODUCCIÓN AL MANEJO DE LA UART.....	19
3.1. Introducción	19
3.2. Objetivos de la práctica.....	19
3.3. Desarrollo.....	19
3.4. Errores comunes.....	23
4. PRÁCTICA 2: INTRODUCCIÓN AL USO DE LA MEMORIA DDR2 DE LA PLACA.....	25
4.1. Introducción	25
4.2. Objetivos de la práctica.....	25
4.3. Desarrollo.....	25
5. PRÁCTICA 3: COMUNICACIÓN ENTRE LENGUAJE RTL Y MICROBLAZE.....	30
5.1. Introducción	30
5.2. Objetivos de la práctica.....	30
5.3. Desarrollo.....	30
6. PRÁCTICA 4: COMUNICACIÓN ENTRE FSM ESCRITAS EN LOS LENGUAJES VHDL Y C	39
6.1. Introducción	39
6.2. Objetivos de la práctica.....	39
6.3. Desarrollo.....	39

7. PRÁCTICA 5: DESARROLLO DE UNA INTERFAZ ENTRE EL ORDENADOR Y LA DDR2.....	49
7.1. Introducción	49
7.2. Objetivos de la práctica.....	49
7.3. Desarrollo.....	49
8. CONCLUSIONES Y LÍNEAS FUTURAS	55
8.1. Conclusiones	55
8.2. Líneas futuras	56
9. BIBLIOGRAFÍA	57
ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES	59
A.1 INTRODUCCIÓN	59
A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO	59
A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS	59
A.4 CONCLUSIONES	60
ANEXO B: PRESUPUESTO ECONÓMICO	61
ANEXO C: ESQUELETO DE EJEMPLO DE CÓDIGO PARA EL MANEJO DE LA UART CON COMANDOS.....	62

ÍNDICE DEL FIGURAS

Figura 1.1 Nexys A7	1
Figura 1.2 Entorno de desarrollo Vivado.....	2
Figura 1.3 Entorno de desarrollo SDK.....	3
Figura 1.4 MicroBlaze	3
Figura 1.5 TeraTerm	4
Figura 1.6 wxHexEditor.....	4
Figura 2.1 RTL Project	6
Figura 2.2 Selección placa Nexys4 DDR.....	7
Figura 2.3 Crear diagrama de bloques	7
Figura 2.4 Nombre diagrama de bloques	8
Figura 2.5 Añadir MicroBlaze	8
Figura 2.6 Run Block Automation.....	8
Figura 2.7 Valores de Run Block Automation.....	9
Figura 2.8 Bloques generados.....	9
Figura 2.9 USB UART	10
Figura 2.10 UART Lite.....	10
Figura 2.11 Configuración por defecto UART Lite.....	11
Figura 2.12 Configuración del reloj de entrada del Clocking Wizard	11
Figura 2.13 Configuración del Reset del Clocking Wizard	11
Figura 2.14 Run Connection Automation.....	12
Figura 2.15 Run Connection Automation Options	12
Figura 2.16 <i>Regenerate Layout</i>	12
Figura 2.17 <i>Layout</i> final práctica 0	13
Figura 2.18 HDL <i>Wrapper</i>	13
Figura 2.19 <i>Manage wrapper</i>	13
Figura 2.20 <i>Export Hardware</i>	14
Figura 2.21 <i>Launch SDK</i>	14
Figura 2.22 Entorno SDK	15
Figura 2.23 <i>Application Project</i>	15
Figura 2.24 <i>New hardware project</i>	16
Figura 2.25 <i>Program FPGA</i>	16
Figura 2.26 Seleccionar archivos <i>Bitstream</i> y <i>ELF</i>	17
Figura 2.27 <i>Setup Serial Port</i>	17
Figura 2.28 <i>Launch on hardware</i>	18
Figura 2.29 Resultado práctica 0.....	18
Figura 3.1 Configuración MicroBlaze práctica 1	19
Figura 3.2 AXI UART Lite y AXI GPIO	20
Figura 3.3 Concat.....	20
Figura 3.4 Configuración Concat.....	20
Figura 3.5 Conexión UART Lite con Concat	21
Figura 3.6 <i>Validate Desing</i>	21
Figura 3.7 <i>New File</i>	21
Figura 3.8 Resultados Terminal practica 1	22
Figura 3.9 Comando 'a'	22
Figura 3.10 Comando 'b' pulsando la tecla 'f' del teclado	23
Figura 3.11 Error en las Librerías	23
Figura 3.12 <i>Generate Linker Script</i>	24

Figura 4.1 DDR2 SDRAM	25
Figura 4.2 Módulo MIG.....	26
Figura 4.3 200MHz clock	26
Figura 4.4 Clk MIG.....	27
Figura 4.5 Rst MIG	27
Figura 4.6 <i>Layout</i> práctica 2	28
Figura 4.7 <i>Memory Tests</i>	28
Figura 4.8 Resultados práctica 2	29
Figura 4.9 LEDs de la placa representando el código ASCII de la letra 'y'	29
Figura 5.1 GPIO LED y Switches.....	30
Figura 5.2 Add Sources.....	30
Figura 5.3 Desing Sources	31
Figura 5.4 <i>Calculator</i>	31
Figura 5.5 Entradas y salidas del módulo RTL.....	32
Figura 5.6 <i>Add module</i>	32
Figura 5.7 Selección de módulos RTL para añadir al diseño	33
Figura 5.8 Bloque RTL.....	33
Figura 5.9 <i>AXI GPIO</i>	33
Figura 5.10 Configuración <i>AXI GPIO</i> nº1	34
Figura 5.11 Configuración <i>AXI GPIO</i> nº2	34
Figura 5.12 Nombres módulos <i>AXI GPIO</i> pt1	35
Figura 5.13 Nombres módulos <i>AXI GPIO</i> pt2	35
Figura 5.14 LED_SW_GPIO	35
Figura 5.15 Conexiones módulo RTL.....	36
Figura 5.16 Run Connection Automation módulos AXI	36
Figura 5.17 <i>Layout</i> práctica 3	36
Figura 5.18 Resultados en el terminal de la práctica 3	38
Figura 5.19 Resultados en la placa de la práctica 3	38
Figura 6.1 Configuración <i>MicroBlaze</i> práctica 4.....	39
Figura 6.2 I/O FSM Práctica 4	40
Figura 6.3 Módulo RTL_FSM.....	40
Figura 6.4 Conexión <i>AXI GPIO</i> con RTL_FSM.....	41
Figura 6.5 <i>Layout</i> práctica 4	41
Figura 6.6 Señales debugueo	42
Figura 6.7 ILA	42
Figura 6.8 Configuración ILA	43
Figura 6.9 ILA probes.....	43
Figura 6.10 Conexión RTL con ILA.....	44
Figura 6.11 ILA clk.....	44
Figura 6.12 <i>Layout</i> con módulo de debugueo.....	45
Figura 6.13 <i>Auto Connect</i>	45
Figura 6.14 Placa conectada	46
Figura 6.15 <i>Add probes</i>	46
Figura 6.16 <i>Probes</i> seleccionadas	46
Figura 6.17 <i>Run Trigger</i>	47
Figura 6.18 Valores de las señales en el <i>Breakpoint</i>	47
Figura 6.19 Resultados en el terminal de la práctica 4	48

Figura 7.1 Configuración <i>UART</i> práctica 5	49
Figura 7.2 <i>Push Buttons</i>	50
Figura 7.3 <i>Layout</i> práctica 5	50
Figura 7.4 Botones Nexys a7	51
Figura 7.5 <i>Baudrate</i> práctica 5	53
Figura 7.6 <i>Send file</i>	53
Figura 7.7 Resultados práctica 5 pt1	54
Figura 7.8 Resultados práctica 5 pt2	54

1. INTRODUCCIÓN Y OBJETIVOS

1.1. INTRODUCCIÓN

Este proyecto surge como una oportunidad para abordar una carencia formativa que se detectó al finalizar mi trabajo de fin de grado. Durante el desarrollo de mi TFG, aparece la necesidad de desarrollar una interfaz eficiente y rápida con la placa Nexys A7, para poder escribir archivos en su memoria DDR2 de forma óptima en términos de tiempo y recursos. Sin embargo, debido a las limitaciones de tiempo y conocimientos en ese momento, no se pudo completar este aspecto del proyecto, lo que supuso un obstáculo importante en su desarrollo.

Ante esta situación, se decidió diseñar un conjunto de prácticas orientadas a cubrir esta carencia formativa, con el objetivo de obtener los conocimientos necesarios para desarrollar interfaces PC-FPGA en proyectos similares. Estas prácticas se centrarán en el empleo de procesadores virtuales (*soft-processor*) y todas las metodologías y capacidades de diseño asociadas, incluyendo su uso junto a bloques descritos en lenguaje de descripción hardware. Con estas prácticas, se pretende alcanzar un nivel de conocimiento y habilidad que permita superar limitaciones de este tipo en proyectos futuros para cualquiera que las realice.

En el presente proyecto el encargado de sintetizar los sonidos es la placa Nexys A7 de Digilent [1] que incorpora una FPGA *Artix-A7*. En concreto se emplea el modelo XC7A100T-1CSG324C, siendo esta referencia la que se utiliza en el software de Xilinx (la sección previa al guion hace referencia al tipo de arquitectura que se monta sobre la plataforma y la sección posterior al guion hace referencia a la plataforma que sustenta todo el hardware necesario para poder emplear dicha arquitectura).



Figura 1.1 Nexys A7

La selección de esta placa se basa en las siguientes razones:

- Está disponible dentro del Departamento de Ingeniería Electrónica.
- Encaja en las necesidades del proyecto.
- El TFG emplea esta misma placa, es decir, usando esta misma placa se podría llegar a un prototipo completo funcional.
- La placa incluye el procesador virtual *Microblaze* [2], indispensable para las necesidades del proyecto.

El proyecto está compuesto por un total de seis prácticas, en las cuales los conocimientos y habilidades necesarias aumentan gradualmente. Cada una de las prácticas se basa en las destrezas y aprendizajes adquiridos en las prácticas anteriores, permitiendo un proceso progresivo y continuo. Estas prácticas están compuestas por:

- **Práctica 0: Introducción al entorno.** Esta práctica tiene como objetivo brindar una introducción al entorno y metodología utilizados en el proyecto, mediante la creación de un "Hola Mundo" básico.
- **Práctica 1: Introducción al uso de la UART.** En esta práctica, se introduce el uso de la UART (*Universal Asynchronous Receiver-Transmitter*) para la comunicación entre el ordenador y la placa Nexys A7.

- **Práctica 2: Introducción al uso de la memoria DDR2 de la placa.** En esta práctica, se profundiza en el uso, configuración, funcionamiento y comunicación de la memoria DDR2 de la placa Nexys A7.
- **Práctica 3: Comunicación entre lenguaje RTL y Microblaze.** Esta práctica introduce el uso de bloques RTL (*Register-Transfer Level*) en el diseño y su comunicación con el procesador virtual Microblaze.
- **Práctica 4: Comunicación entre FSM escritas en los lenguajes VHDL y C.** Esta práctica avanza en el uso de bloques RTL y comunicación, con un enfoque en la comunicación entre FSM (*Finite State Machine*) escritas en los lenguajes VHDL y C, y el procesador virtual Microblaze.
- **Práctica 5: Desarrollo de una interfaz entre el ordenador y la DDR2.** Esta última práctica tiene como objetivo unir y afianzar todos los conocimientos y habilidades adquiridos en las prácticas anteriores, mediante el desarrollo de una interfaz entre el ordenador y la DDR2.

1.2.OBJETIVOS

El objetivo principal de este Trabajo de Fin de Máster es abordar las carencias formativas con relación al empleo de procesadores virtuales (*soft-processor*) y todas las metodologías y capacidades de diseño asociadas, incluyendo su uso en conjunto con bloques descritos en lenguajes de descripción hardware. Para lograr este objetivo, se han llevado a cabo las siguientes tareas:

- Realización de un estudio bibliográfico sobre las metodologías y técnicas necesarias para el desarrollo del proyecto, así como ejemplos de proyectos similares.
- Estudio bibliográfico de los diferentes periféricos disponibles en la placa Nexys A7, con el objetivo de entender su funcionamiento y cómo utilizarlos de manera adecuada.
- Elección de los periféricos más adecuados para componer una interfaz de comunicación PC-Placa, teniendo en cuenta los objetivos y requisitos del proyecto.
- Diseño de las prácticas de forma progresiva, para transmitir correctamente los conocimientos necesarios y asegurar un aprendizaje continuo. Así, cada práctica se basará en los conocimientos adquiridos en las prácticas anteriores.
- También, se llevará a cabo una implementación y simulación de los diferentes bloques y diseños realizados, para comprobar su correcto funcionamiento y asegurar que cumplen con los objetivos establecidos.
- Por último, se realizará una evaluación y análisis de los resultados obtenidos, para determinar el éxito del proyecto y determinar áreas de mejora para futuros proyectos similares.

1.3.METODOLOGÍA

El proyecto se ha desarrollado usando el entorno de desarrollo Vivado [3], específicamente Vivado 2018.1.



Figura 1.2 Entorno de desarrollo Vivado

Vivado es un software de diseño de sistemas electrónicos que ofrece una completa plataforma para el desarrollo y verificación de sistemas mediante un lenguaje de descripción hardware, en este caso: VHDL. Con Vivado, se tiene la capacidad de llevar a cabo todo el ciclo de diseño de un sistema, desde la creación de los ficheros (.vhd) hasta la implementación y generación de un fichero (.bit) listo para ser volcado en la FPGA (*Field-Programmable Gate Array*).

Una vez finalizado el sistema completo en Vivado, es posible introducirlo en una FPGA y comprobar su funcionamiento en un sistema real, más allá de las simulaciones llevadas a cabo en el entorno de Vivado. Esto es gracias a las numerosas herramientas y opciones que Vivado ofrece para la verificación y optimización del sistema antes de su implementación en un sistema real. Además, Vivado permite una mayor flexibilidad y rapidez en el proceso de diseño, ya que se pueden llevar a cabo cambios y mejoras en el sistema de manera dinámica y eficiente.

El propio entorno incluye el kit de desarrollo de software Xilinx (XSDK [4] , el cual es un entorno de diseño integrado para la creación de aplicaciones integradas en cualquiera de los microprocesadores de Xilinx, en este caso, *MicroBlaze*. El entorno está basado en *Eclipse* [5], lo que proporciona una interfaz intuitiva y fácil de usar para los desarrolladores.

Además, el entorno XSDK ofrece una serie de beneficios adicionales, como el acceso a un conjunto muy amplio de bibliotecas y controladores, la posibilidad de integrar FreeRTOS como RTOS para todas las plataformas, y una compatibilidad completa con flujos de depuración y diseño de software. Esto incluye capacidades de depuración conjunta de hardware y software, así como soporte para multiprocesadores. Estas características son esenciales para el desarrollo de sistemas complejos y proporcionan una mayor eficiencia en el proceso de diseño.



Figura 1.3 Entorno de desarrollo SDK

MicroBlaze [2] es una familia de configuraciones de microprocesador RISC de 32 bits/64 bits que ofrece una gran flexibilidad y personalización. Estas configuraciones son preestablecidas y modificables, lo que permite adaptarse a las necesidades específicas de cualquier aplicación.

El procesador *MicroBlaze* puede configurarse de diferentes maneras, permitiendo su uso en código baremetal, como procesador en tiempo real (RTOS) o incluso con Linux incorporado. Esta versatilidad permite a los desarrolladores elegir la opción más adecuada para su proyecto.

Además, *MicroBlaze* cuenta con una gran capacidad de manejo de controladores, lo que permite agilizar y facilitar el uso de periféricos como PWM, UART, DMA o interfaces serie. Esto garantiza una mayor eficiencia en el uso de los periféricos y una mejor adaptabilidad a las necesidades específicas de la aplicación.



Figura 1.4 MicroBlaze

Para poder acceder al terminal serie y realizar las comunicaciones con la placa se usa el programa *TeraTerm* [6]. Este es un emulador de terminal gratuito que permite a los ordenadores personales (PCs) conectarse a sistemas externos para automatizar tareas, recoger datos, etc. *TeraTerm* es compatible con una variedad de terminales de ordenador diferentes, lo que lo convierte en una herramienta versátil para el desarrollo de proyectos electrónicos.

Una de las características clave de *TeraTerm* es su capacidad para vincularse a una conexión TCP/IP o a un puerto serie, lo que permite una gran flexibilidad en la conectividad con el sistema externo. Esto facilita la comunicación entre el ordenador y la placa, lo que permite una mayor eficiencia en el proceso de desarrollo de proyectos.



Figura 1.5 TeraTerm

Finalmente para la comprobación de que las escrituras en memoria se hacen correctamente se usa *wxHexEditor* [7]. Este es un editor hexadecimal compatible con Windows que ha sido diseñado específicamente para archivos grandes, y que admite hasta 2^{64} Bytes. Con este programa se puede verificar que los datos escritos en la memoria de la placa coinciden con los datos del archivo del ordenador, garantizando así la correcta ejecución de la escritura en la memoria.

wxHexEditor es una herramienta esencial para el desarrollo de proyectos que requieren escritura en la memoria de una placa. Su diseño específico para archivos grandes lo convierte en una herramienta ideal para verificar que los datos escritos en la memoria son correctos. Además, el hecho de que sea compatible con Windows hace que sea una herramienta accesible y fácil de usar.



Figura 1.6 wxHexEditor

1.4.DESARROLLO TEMPORAL

Una vez analizadas las diferentes etapas y metodologías utilizadas en este proyecto, es importante hacer un desglose detallado del tiempo dedicado a cada paso.

- Estudio bibliográfico sobre las metodologías y ejemplos de proyectos relacionados con el uso de procesadores virtuales y lenguajes de descripción hardware: Se estima que se dedicaron entre **40 y 50 horas** para realizar una investigación exhaustiva sobre las diferentes metodologías y ejemplos de proyectos existentes en el mercado.
- Estudio bibliográfico de los diferentes periféricos de la placa Nexys A7: Se estima que se dedicaron alrededor de **70 horas** para estudiar todas las características y funciones de los diferentes periféricos de la placa.
- Elección de los periféricos y pruebas para comprobar la viabilidad de uso: Este paso fue crucial para el proyecto, ya que se dedicaron aproximadamente **160 horas** para seleccionar los periféricos adecuados y realizar las pruebas necesarias para comprobar su viabilidad en el proyecto.
- Diseño de las prácticas: Este paso fue clave para transmitir los conocimientos necesarios a los estudiantes, se estima que se dedicaron entre **80 y 90 horas** para diseñar las prácticas de forma progresiva y adecuada.
- Verificación y pruebas de las prácticas: Este paso fue crucial para garantizar que las prácticas funcionaban correctamente, se estima que se dedicaron unas **30 horas** para realizar esta tarea.
- Escritura de la Memoria del Trabajo de Fin de Máster: Por último, se dedicaron unas **70 horas** aproximadamente para escribir la memoria del trabajo, documentando todas las etapas y metodologías utilizadas en el proyecto.

En total, se estima que se dedicaron unas **470 horas** a este proyecto, lo que refleja el gran esfuerzo y dedicación que se ha invertido.

1.5.METODOLOGÍA DOCENTE DE LAS PRÁCTICAS

La metodología con la que se pretende que se realicen estas prácticas es la siguiente: una explicación previa de los objetivos y la teoría necesaria para el desarrollo de la práctica y su posterior realización. Se estima que para cada práctica se necesita un tiempo de:

- **Práctica 0: Introducción al entorno.**
 - Explicación previa: 1 hora.
 - Tiempo de desarrollo: 1 hora.
- **Práctica 1: Introducción al uso de la UART.**
 - Explicación previa: 1 hora.
 - Tiempo de desarrollo: 1 hora.
- **Práctica 2: Introducción al uso de la memoria DDR2 de la placa.**
 - Explicación previa: 1 hora.
 - Tiempo de desarrollo: 2'5 horas.
- **Práctica 3: Comunicación entre lenguaje RTL y Microblaze.**
 - Explicación previa: 0'5 horas.
 - Tiempo de desarrollo: 3 horas.
- **Práctica 4: Comunicación entre FSM escritas en los lenguajes VHDL y C.**
 - Explicación previa: 0'5 horas.
 - Tiempo de desarrollo: 3'5 horas.
- **Práctica 5: Desarrollo de una interfaz entre el ordenador y la DDR2.**
 - Explicación previa: 0'5 horas.
 - Tiempo de desarrollo: 4'5 horas.

Por lo tanto, se estima un tiempo total de aproximadamente **20 horas**.

1.6.PRÁCTICAS DESCARTADAS

El proyecto que se ha llevado a cabo ha tenido en cuenta varios periféricos para su diseño y desarrollo. Sin embargo, se han descartado algunos de ellos debido a diferentes motivos.

En primer lugar, se ha descartado el uso de la tarjeta microSD. El objetivo de incluir este periférico era facilitar el manejo de archivos y evitar la necesidad de utilizar un ordenador para ello. De esta forma, sería mucho más sencillo y rápido añadir o eliminar archivos. Sin embargo, se descartó su uso debido a que las bibliotecas y controladores necesarios para su uso requerían de una licencia de alto costo. Esto significaba que toda la interfaz de comunicación necesaria debía ser diseñada desde cero, lo cual suponía un gran tiempo y dificultad en el diseño de las prácticas.

Por otro lado, también se descartó el uso del conector de Ethernet. La idea detrás de esta decisión era mejorar la velocidad de transmisión de datos, permitiendo así enviar archivos de mayor tamaño y una mayor versatilidad del prototipo. Sin embargo, al comenzar el diseño de las prácticas, se descubrió que los controladores de la interfaz ethernet habían sido eliminados en versiones posteriores a Vivado 2018.1. Aunque se habló con el departamento para obtener acceso a estas bibliotecas, no fue viable su uso debido a las licencias necesarias. Finalmente, se decidió mantener el uso del puerto serie debido a que no merecía la pena el tiempo y dificultad necesarios para el uso de este periférico.

Además de las prácticas relacionadas con los periféricos del lector de tarjeta microSD y el conector de Ethernet, también se han descartado prácticas que incluyan el uso de sistemas operativos en el procesador, como *freeRTOS* por ejemplo. La razón principal de esta decisión es la barrera de dificultad y tiempo que su implementación puede suponer. Aunque el uso de un sistema operativo en el procesador puede ofrecer una mayor flexibilidad y escalabilidad en la gestión de los recursos, en este caso se ha considerado que los sistemas diseñados no requerían su uso debido a que el uso de *baremetal* resultaba suficiente. Además, incluir prácticas relacionadas con el uso de sistemas operativos podría haber desviado el enfoque principal del proyecto y haber dificultado el aprendizaje de los conceptos esenciales. Por tanto, se ha decidido enfocar las prácticas en la comunicación con los periféricos de la placa mediante *baremetal*.

2. PRÁCTICA 0: INTRODUCCIÓN AL ENTORNO

2.1. INTRODUCCIÓN

En esta práctica se va a llevar a cabo un proyecto de introducción al diseño de bloques con el uso del microprocesador virtual *MicroBlaze*, una de las opciones de procesador que ofrece Vivado, el entorno de desarrollo de Xilinx. El objetivo de esta práctica es proporcionar una comprensión básica de cómo trabajar con *MicroBlaze* y las herramientas necesarias para crear un proyecto desde cero.

En esta ocasión se va a llevar a cabo un proyecto sencillo pero con un gran simbolismo, un "Hola Mundo". El proyecto se desarrollará desde cero, es decir, desde la creación del proyecto en Vivado, pasando por el diseño de bloques necesarios para el funcionamiento del procesador hasta finalmente llegar a la creación del código "Hola Mundo" y la programación de la FPGA.

La práctica está diseñada para ser lo más sencilla y fácil de seguir posible. En ella se incluyen explicaciones detalladas de los pasos a seguir y capturas de pantalla para facilitar el aprendizaje del uso del entorno. Sin embargo, se asume que el lector tiene conocimientos básicos de programación en C y conocimientos generales sobre procesadores y FPGAs. A medida que se avance en futuras prácticas, se darán por sabidos estos detalles y se enfocará en aspectos más avanzados del diseño con *MicroBlaze*.

2.2. OBJETIVOS DE LA PRÁCTICA

- I. Familiarizarse con el entorno de diseño bloques de Vivado
- II. Familiarizarse con el entorno *SDK*
- III. Aprender el flujo de trabajo
- IV. Aprender a usar *MicroBlaze*

2.3. DESARROLLO

1. Abra el Vivado y cree un proyecto nuevo, lo llamaremos *holaMicroBlaze*.
2. En la siguiente ventana seleccionar que *RTL Project*. Seleccionar la opción de no especificar fuentes, no se van a necesitar por el momento.

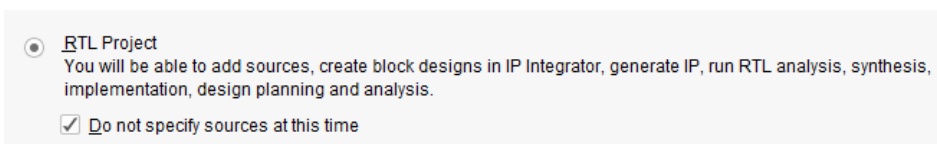


Figura 2.1 RTL Project

3. Selecciona la placa Nexys 4 DDR (Para poder realizar este paso se proporciona el archivo de configuración de la placa. Para que Vivado tenga acceso a este archivo tendrá que meterlo en la siguiente ruta: *C:\Xilinx\Vivado\2018.1\data\boards*). En este proyecto no es crucial tener la configuración de la placa, pero en futuros proyectos este paso ahorrará mucho trabajo a la hora de configurar los diferentes módulos IP.

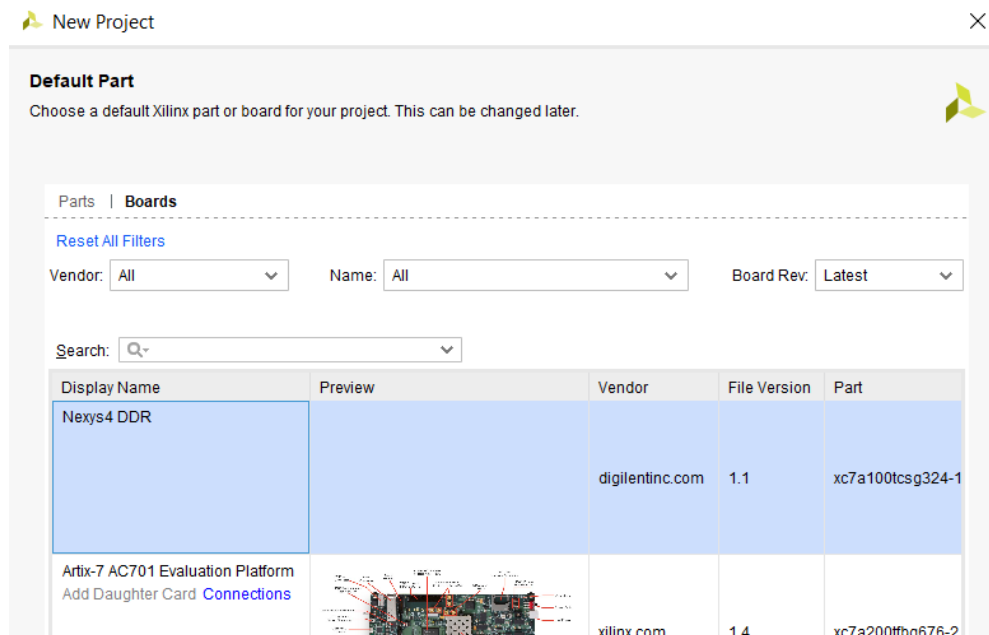


Figura 2.2 Selección placa Nexys4 DDR

4. Una vez creado el proyecto hay que crear un diagrama de bloques.

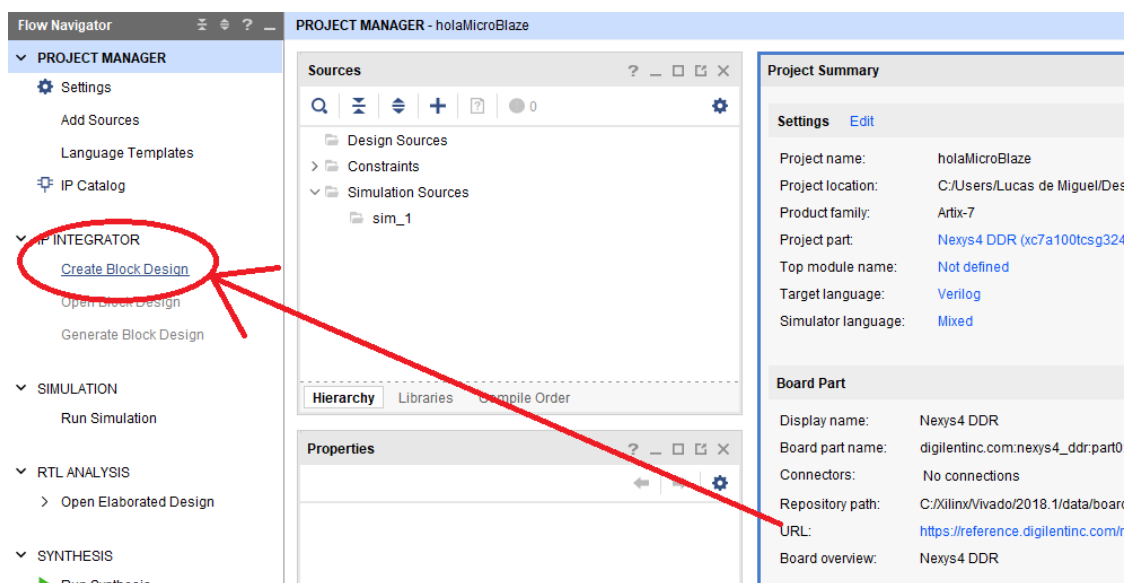


Figura 2.3 Crear diagrama de bloques

5. A la hora de poner el nombre al diagrama es muy recomendable poner un nombre muy representativo para una mejor organización del espacio de trabajo. En este caso: *holaMicro*, por ejemplo.

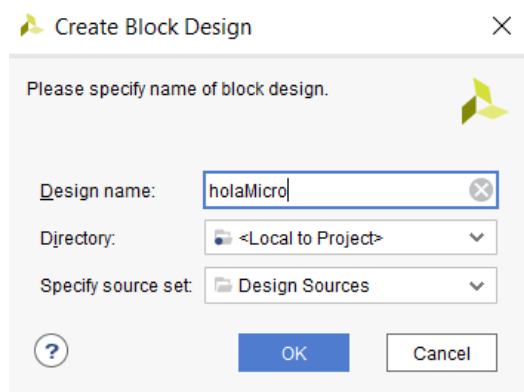


Figura 2.4 Nombre diagrama de bloques

6. Añadir al diseño, presionando el botón '+', el módulo IP: *MicroBlaze*.

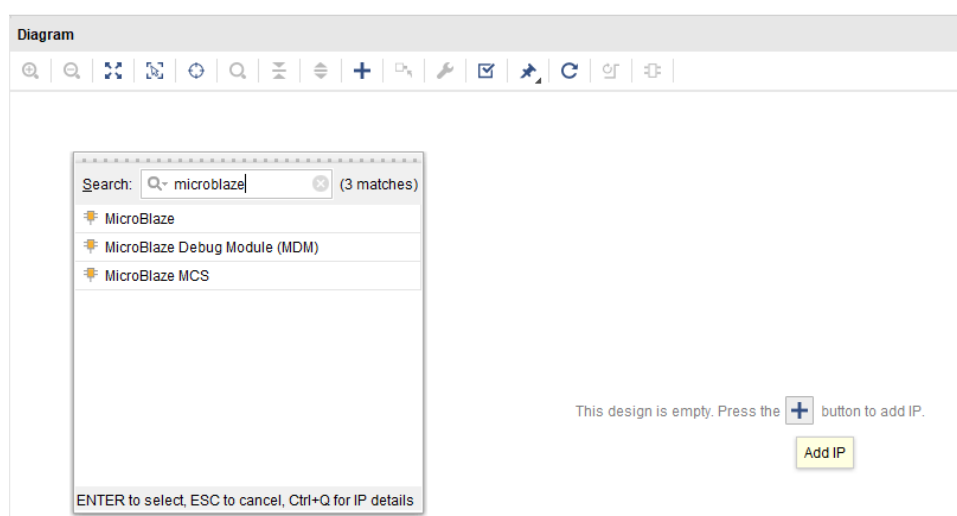


Figura 2.5 Añadir MicroBlaze

7. Una vez añadido MicroBlaze pulse en *Run Block Automation*. De esta forma se añadirán automáticamente los bloques necesarios para que funcione correctamente el módulo.

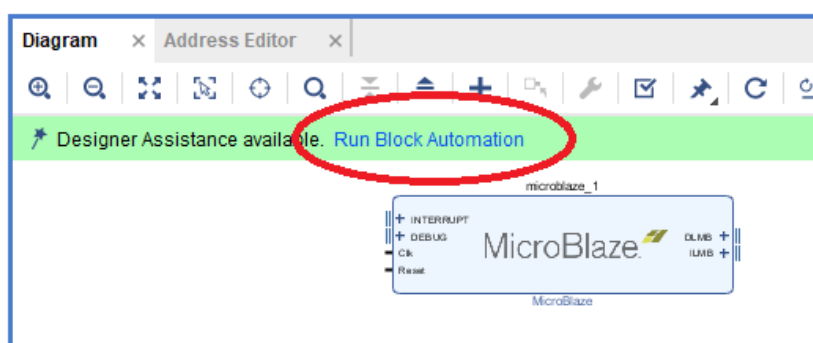


Figura 2.6 Run Block Automation

8. En la ventana saliente se configuran los módulos que van a ser añadidos (memoria local, caché, el reloj, etc.). Para esta práctica se mantendrán los valores por defecto.

Options

Preset:	None
Local Memory:	8KB
Local Memory ECC:	None
Cache Configuration:	None
Debug Module:	Debug O...
Peripheral AXI Port:	Enabled
Interrupt Controller:	<input type="checkbox"/>
Clock Connection:	New Clocking Wizard (100 M...

Figura 2.7 Valores de Run Block Automation

Se generarán los siguientes bloques:

- La memoria local: *microblaze_0_local_memory*
- *Processor System Reset*: para el manejo del Reset del sistema
- *Clocking Wizard*: para configurar el reloj.
- *MicroBlaze Debug Module* [8]

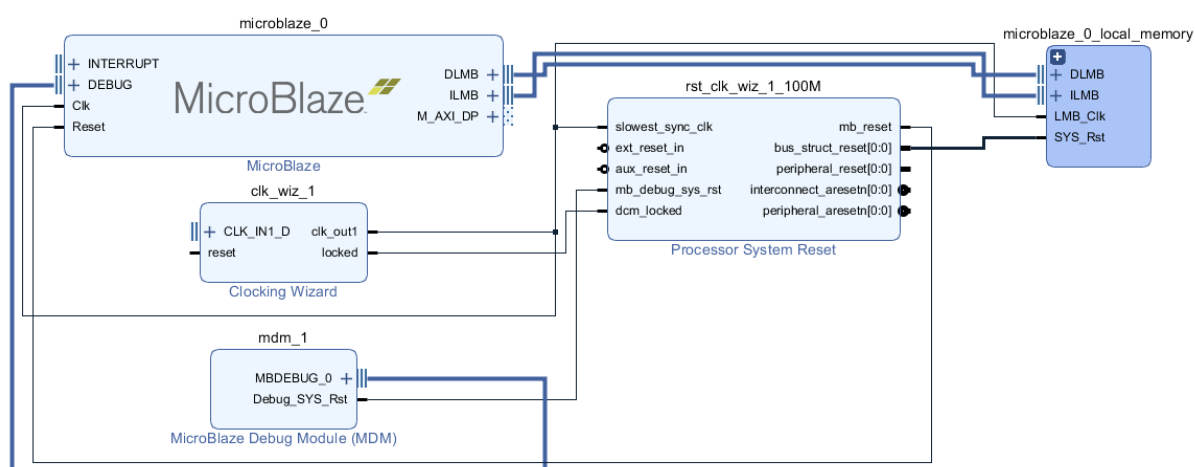


Figura 2.8 Bloques generados

9. En la pestaña *Board* aparecen todos los periféricos disponibles en la placa Nexys4 DDR gracias al paso 3. Seleccione el módulo *USB UART* para añadirlo al diseño.

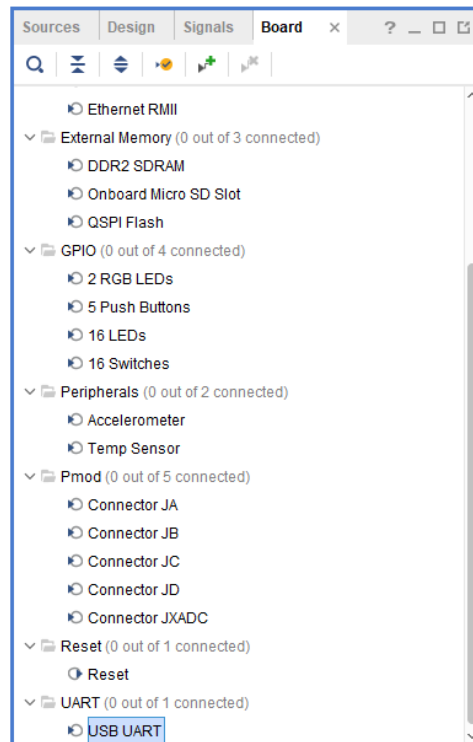


Figura 2.9 USB UART

10. Seleccione el módulo IP: *AXI Uartlite*. Deje la configuración del módulo por defecto.

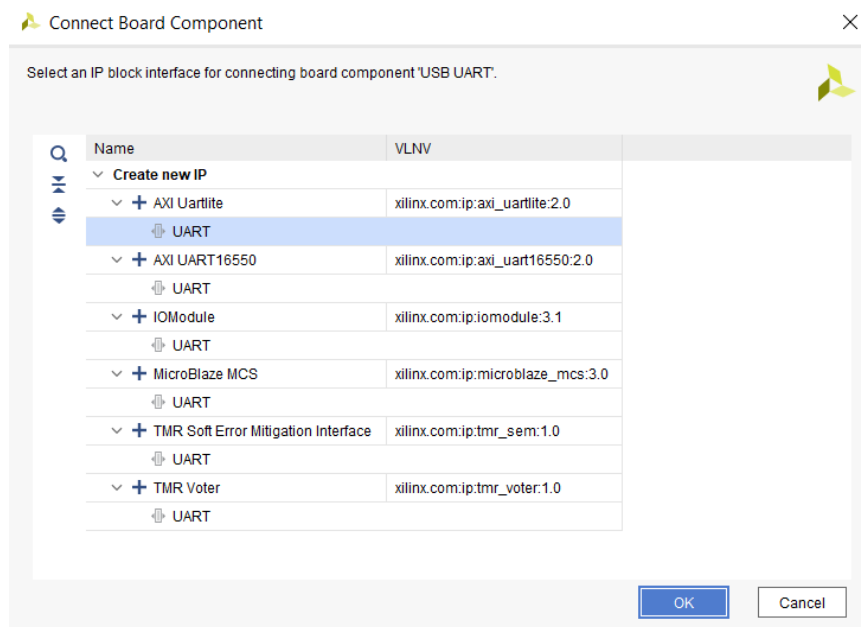


Figura 2.10 UART Lite

Board **IP Configuration**

☒ Auto AXI CLK Frequency 100.0 [10-300]MHz

Baud Rate 9600

Data Bits 8 [5 - 8]

Parity

☒ No Parity ☐ Odd ☐ Even

Figura 2.11 Configuración por defecto UART Lite

11. Seleccione el *Clocking Wizard*. Configure el reloj de entrada en *sys_clock* para indicar que se usará el reloj de 100MHz del sistema y el reset como activo a nivel bajo.

Board **Clocking Options** **Output Clocks** **MMCM Settings** **Summary**

Associate IP interface with board interface

IP Interface	Board Interface
CLK_IN1	sys clock
CLK_IN2	Custom
EXT_RESET_IN	Custom

Clear Board Parameters

Figura 2.12 Configuración del reloj de entrada del Clocking Wizard

Board **Clocking Options** **Output Clocks** **MMCM Settings** **Summary**

Output Clock	Sequence Number	Frequency	Phase	Source	Signaling
clk_out2	1	100.000	0.000	N/A	BUFG
clk_out3	1	100.000	0.000	N/A	BUFG
clk_out4	1	100.000	0.000	N/A	BUFG
clk_out5	1	100.000	0.000	N/A	BUFG
clk_out6	1	100.000	0.000	N/A	BUFG
clk_out7	1	100.000	0.000	N/A	BUFG

☐ USE CLOCK SEQUENCING

Clocking Feedback

Source	Signaling
<input checked="" type="radio"/> Automatic Control On-Chip	<input checked="" type="radio"/> Single-ended
<input type="radio"/> Automatic Control Off-Chip	<input type="radio"/> Differential
<input type="radio"/> User-Controlled On-Chip	
<input type="radio"/> User-Controlled Off-Chip	

Enable Optional Inputs / Outputs for MMCM/PLL

☒ reset ☐ power_down ☐ input_clk_stopped

☒ locked ☐ clkfbstopped

Reset Type

☐ Active High ☒ Active Low

Figura 2.13 Configuración del Reset del Clocking Wizard

12. Para realizar de forma automática algunas conexiones y generar un módulo *AXI Interconnect* (el cual se encarga de las comunicaciones AXI) pulse *Run Connection Automation*.

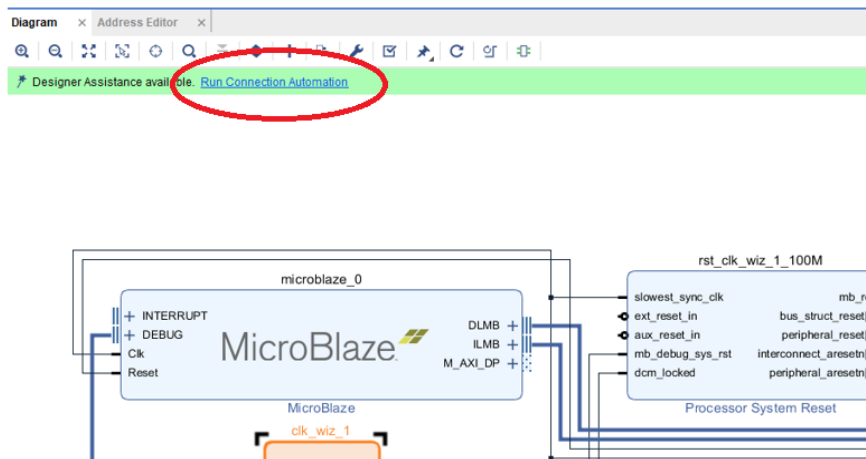


Figura 2.14 Run Connection Automation

Saldrá la siguiente ventana, donde puede seleccionar que módulos quiere que se conecten de forma automática, para este paso deje todos seleccionados:

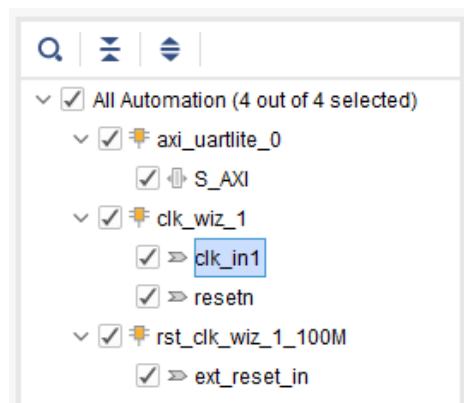


Figura 2.15 Run Connection Automation Options

13. Para obtener un *Layout* de los bloques más limpio pulse *Regenerate Layout* y automáticamente se colocarán los bloques de la forma más eficiente en función de las conexiones existentes. Se obtiene el diagrama de la Figura 2.17.

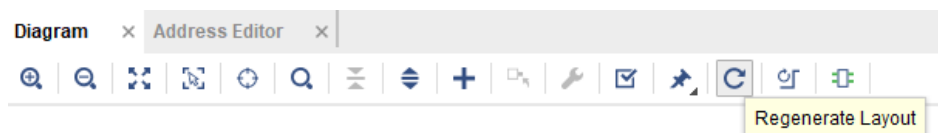


Figura 2.16 Regenerate Layout

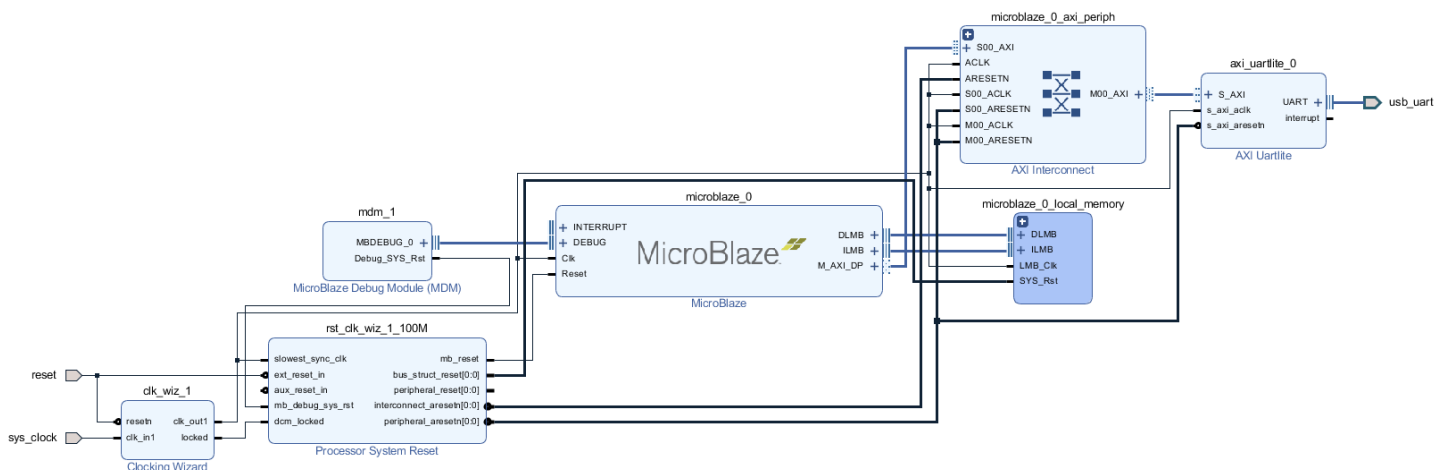


Figura 2.17 Layout final práctica 0

14. En este momento se tiene un archivo no sintetizable por lo que hay que crear un *wrapper* de código RTL que describa el diseño de bloques anterior.

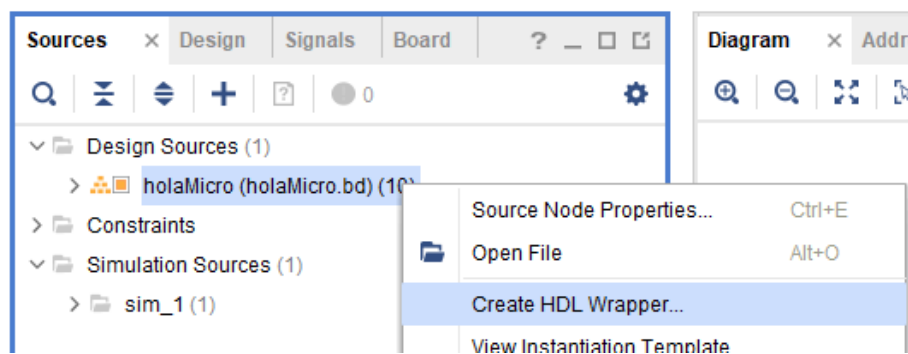


Figura 2.18 HDL Wrapper

En la ventana saliente seleccionar la segunda opción para que sea el propio Vivado el que actualice este archivo y evitar así problemas si es necesario cambiar el diseño de bloques en el futuro.

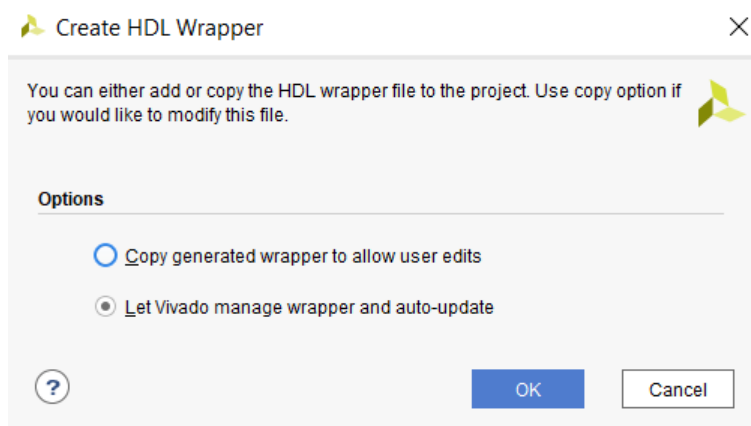


Figura 2.19 Manage wrapper

15. Generar bitstream. (*Nota: La primera vez tarda mucho en sintetizar e implementar todos los módulos del diagrama, si se cambia o se añade algo en el diagrama será lo único que se sintetice*)
16. Una vez generado el *bitstream* exporte el hardware seleccionando la opción *include bitstream*.

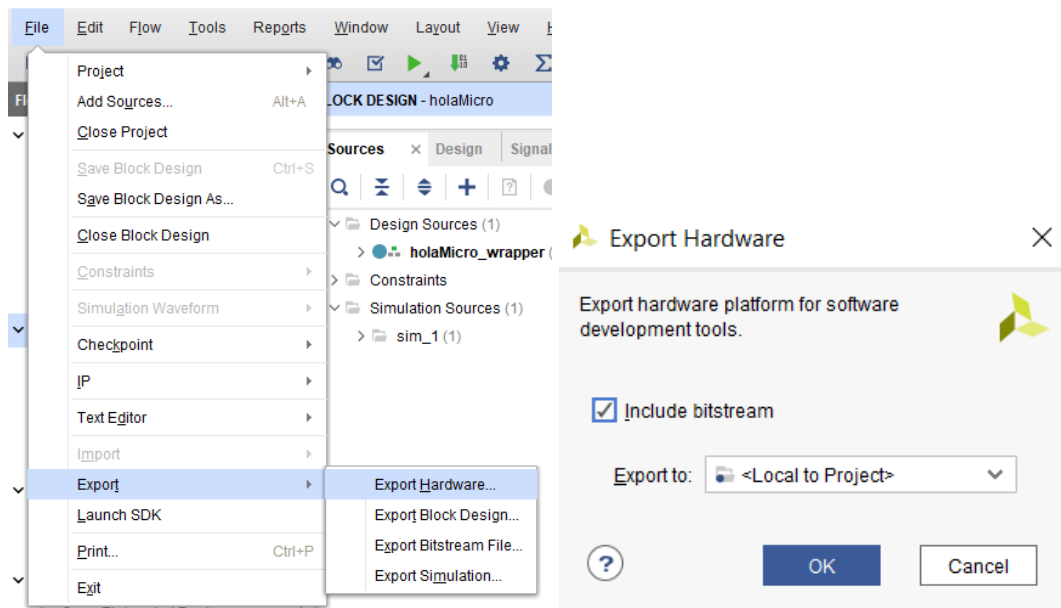


Figura 2.20 Export Hardware

17. Una vez exportado el hardware lance el SDK.

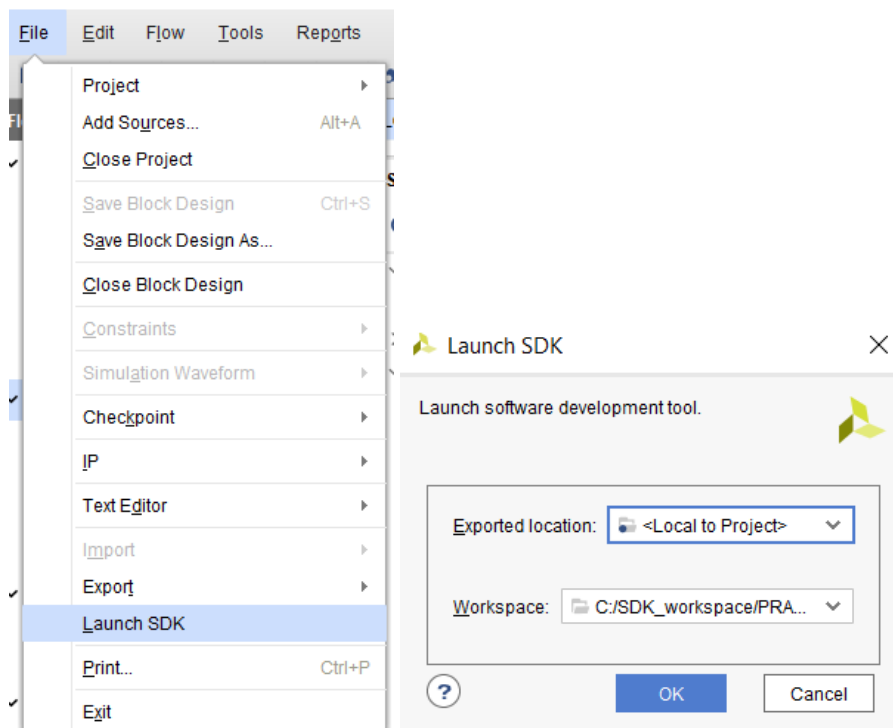


Figura 2.21 Launch SDK

Nota: el workspace seleccionado debe ser una ruta que no contenga espacios.

18. El Vivado abrirá el programa SDK, basado en eclipse, pensado para el desarrollo de aplicaciones C/C++. Seleccione *Create Application Project*.



Figura 2.22 Entorno SDK

19. Al crear un nuevo proyecto tendrá que seleccionar la plataforma hardware donde se llevará acabo.

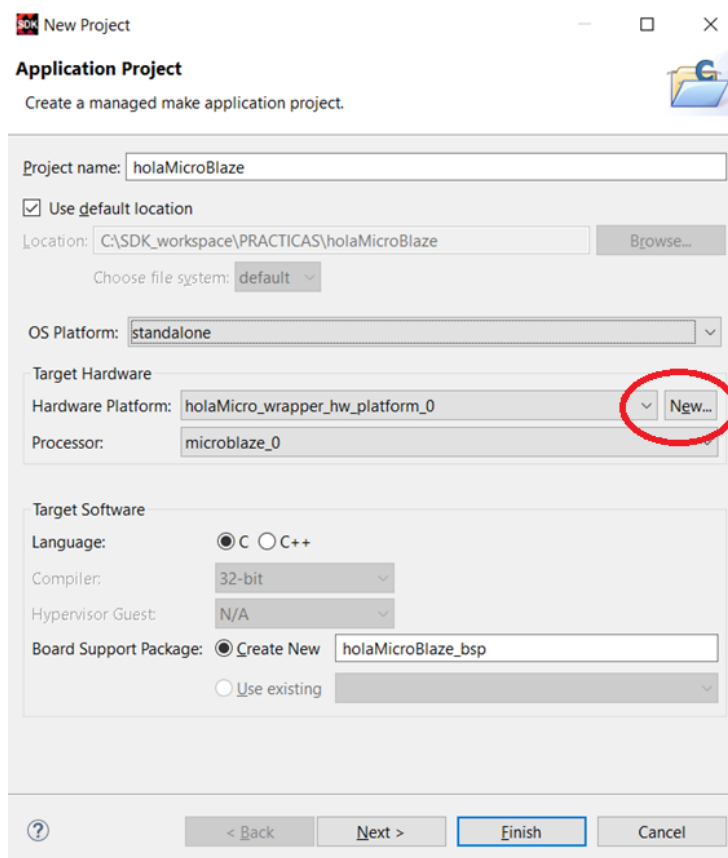


Figura 2.23 Application Project

El hardware diseñado anteriormente lo encontrará en la ruta marcada:

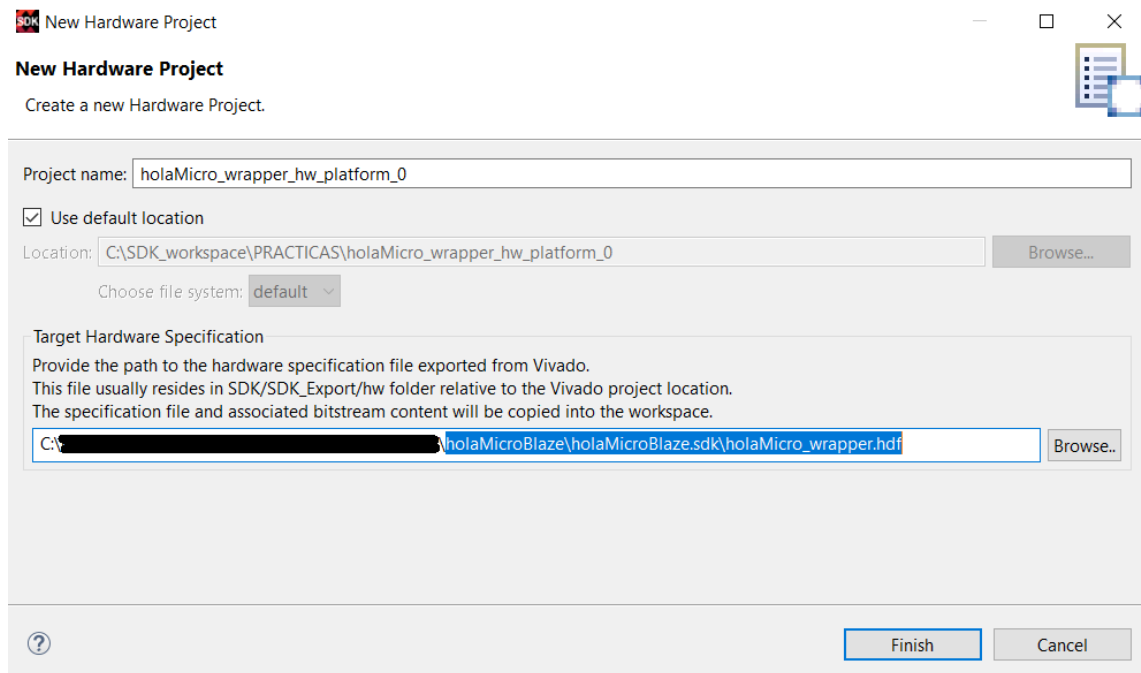


Figura 2.24 *New hardware Project*

20. Pinchando en *Next* aparecerán una serie de proyectos de ejemplo. Seleccionamos *Hello World*.
21. Para poder correr el programa en la placa primero hay que programar la FPGA. Es decir, el bitstream y más archivos necesarios.

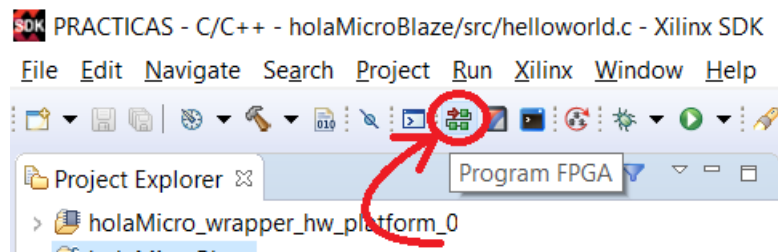


Figura 2.25 *Program FPGA*

Hay que especificar el archivo bitstream y ELF que se enviarán a la memoria de la placa. Asegúrese de que sean los correctos.

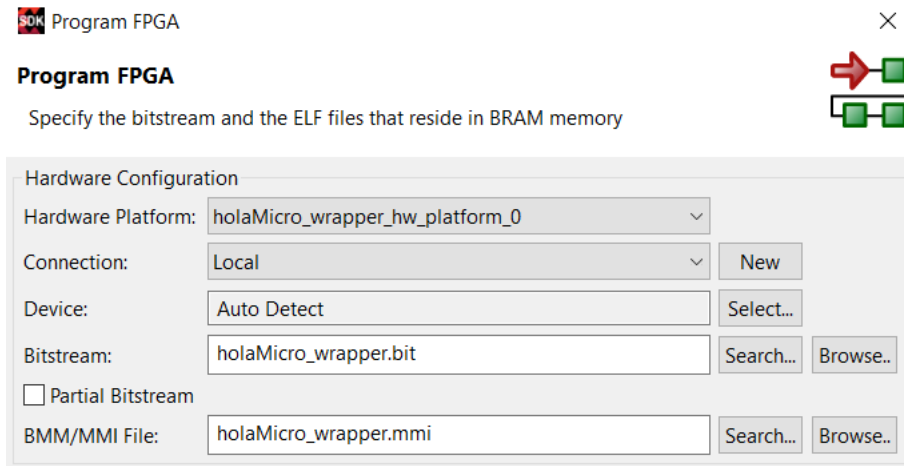


Figura 2.26 Seleccionar archivos *Bitstream* y *ELF*

22. Para poder visualizar el puerto serie usaremos el programa *Tera Term*. Comprobamos que la velocidad es la misma que la configuración de *UART Lite*.

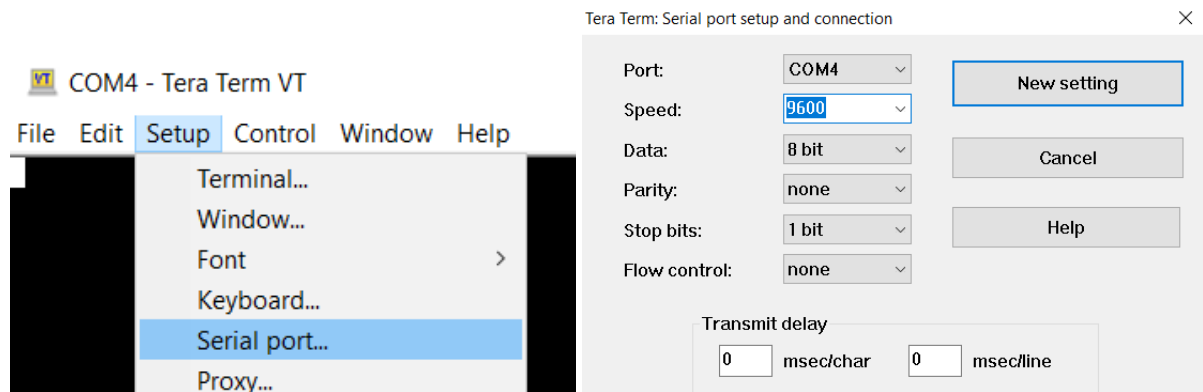


Figura 2.27 *Setup Serial Port*

23. Para correr el programa selecciona la cuarta opción: *Launch on Hardware*.

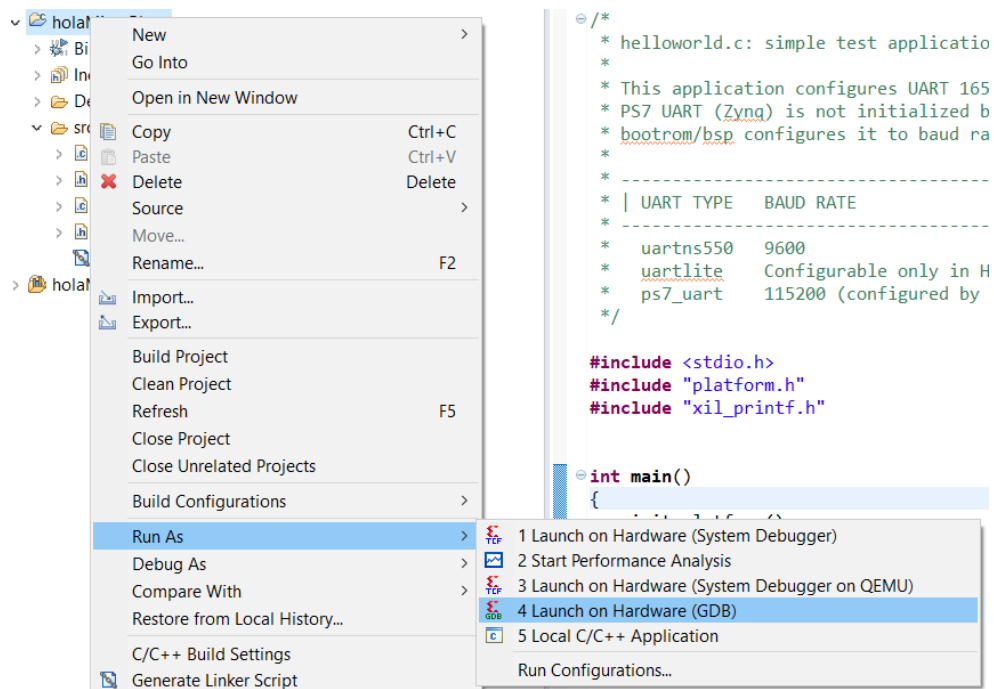


Figura 2.28 Launch on hardware

24. Finalmente se obtendrá el siguiente resultado así por finalizada la práctica.

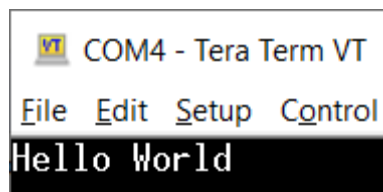


Figura 2.29 Resultado práctica 0

3. PRÁCTICA 1: INTRODUCCIÓN AL MANEJO DE LA UART

3.1. INTRODUCCIÓN

En esta práctica se va a llevar a cabo un proyecto de introducción al diseño de bloques con el uso del microprocesador virtual *MicroBlaze*. Se llevará a cabo la creación de una interfaz básica entre la placa y el ordenador. El objetivo principal será recibir comandos del ordenador a través del teclado y actuar en consecuencia. Esto incluirá la configuración de los diferentes periféricos necesarios, como la UART, para lograr la comunicación entre la placa y el ordenador.

3.2. OBJETIVOS DE LA PRÁCTICA

- I. Aprender un manejo más avanzado de la *UART*
- II. Aprender a configurar la *UART*
- III. Aprender a leer comandos del teclado con la *UART*
- IV. Generar una interfaz sencilla entre placa y ordenador

3.3. DESARROLLO

1. Realizar pasos del 1 al 7 de la practica 0 para crear un nuevo proyecto y un diagrama de bloques.
2. Esta vez el *MicroBlaze* va a necesitar una memoria local de 32KB debido a que se van a usar más librerías que en la anterior práctica. También hay que seleccionar la opción del controlador de interrupciones, para manejar las interrupciones que genere la *UART*.

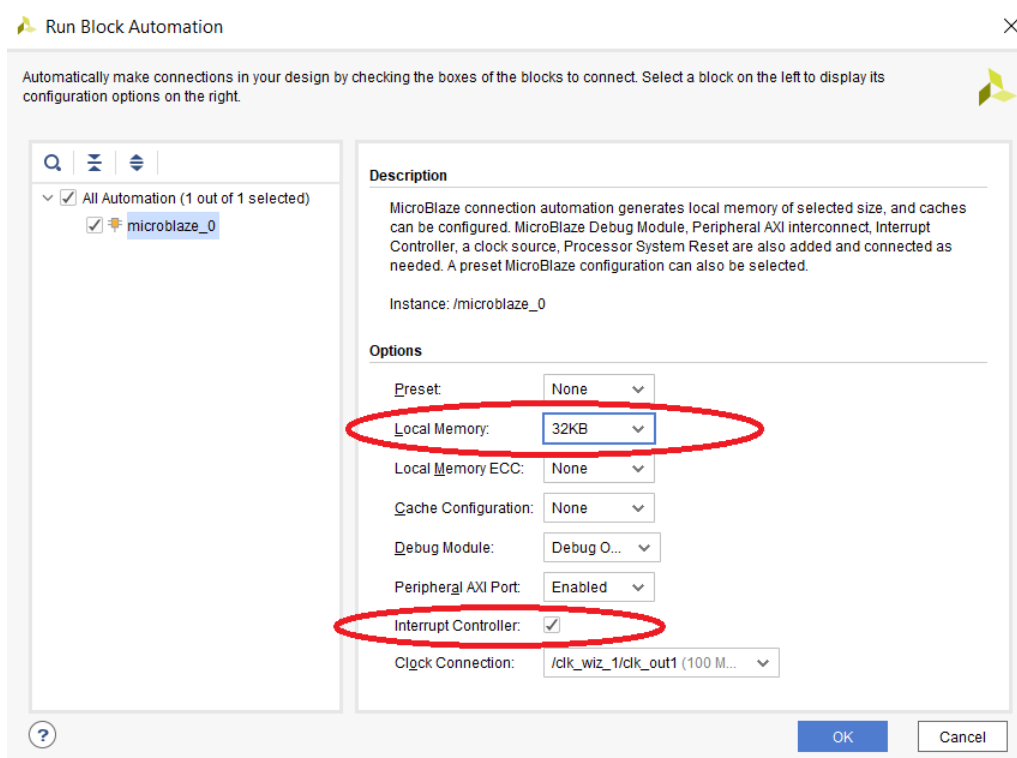


Figura 3.1 Configuración MicroBlaze práctica 1

- En la pestaña *Board* añada los periféricos: *16 LEDs* y *USB UART*. Deje la configuración que viene por defecto para la placa.

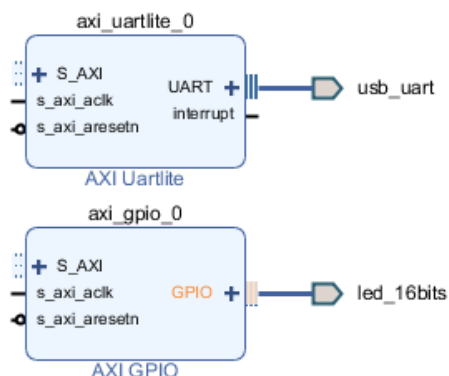


Figura 3.2 AXI UART Lite y AXI GPIO

- Selecione el *Clocking Wizard*. Configure el reloj de entrada en *sys_clock* para indicar que se usará el reloj de 100MHz del sistema y el reset como activo a nivel bajo.
- Pulse *Run Connection Automation* y seleccione todos los módulos.
- Reconfigure el módulo *concat* con un único puerto de entrada.

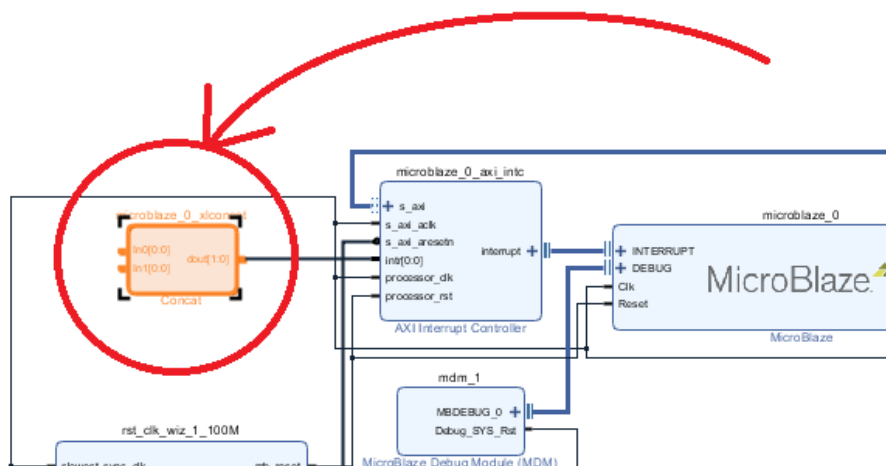


Figura 3.3 Concat

Number of Ports	1	[1 - 32]
Auto In0 Width	1	[1 - 4096]
Auto In1 Width	1	[1 - 4096]

Figura 3.4 Configuración Concat

7. Conecte la salida *interrupt* del módulo *UART Lite* a la entrada de *Concat*. Ahora las interrupciones generadas por el módulo *UART Lite* están gestionadas de forma automática por el módulo *AXI Interrupt Controller*.

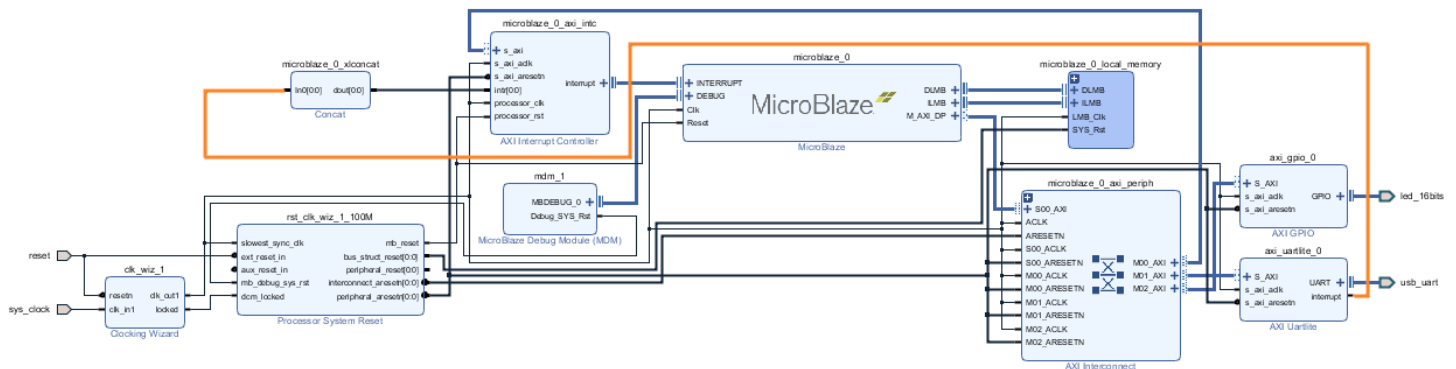


Figura 3.5 Conexión UART Lite con Concat

8. Para comprobar que no hay errores o warnings en el diseño pulse *Validate Desing*.

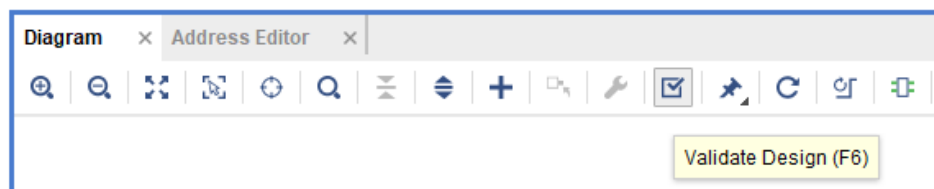


Figura 3.6 *Validate Desing*

9. Si no hay ningún error en el diseño realice los pasos del 13 al 19 de la práctica 0.
10. Seleccione la opción de *Empty Project* y cree un archivo nuevo al que llamaremos *uart_led.c*, por ejemplo.

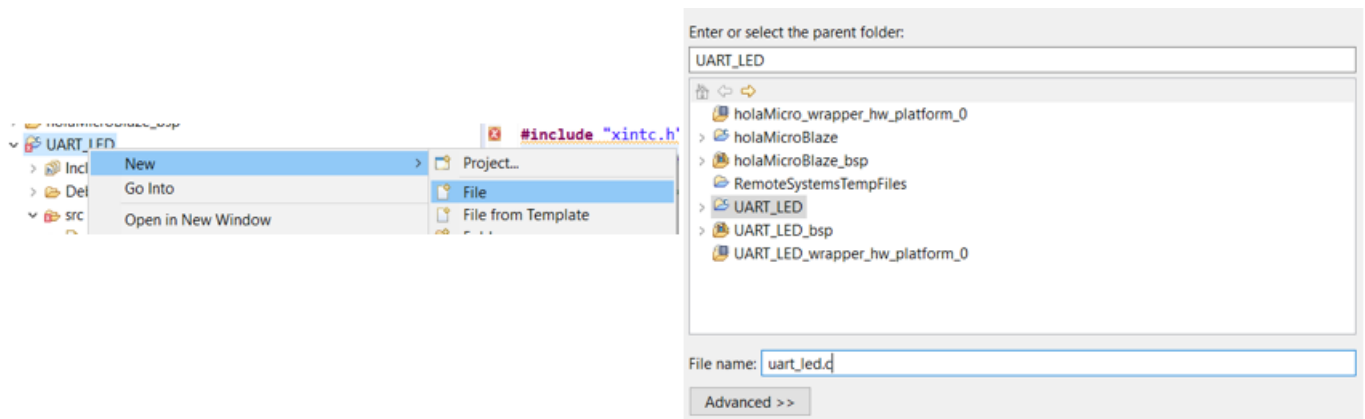


Figura 3.7 *New File*

11. El código de este archivo debe ser realizado por el alumno. Se recomienda usar el esqueleto proporcionado en el “Anexo C: Esqueleto de ejemplo de código para el manejo de la UART con comandos” de este documento. Este código tiene zonas sin rellenar para que sean escritas por el alumno.

El código desarrollado tiene que ser capaz de detectar que tecla ha sido pulsada en el teclado del ordenador y actuar en consecuencia. Habrá mínimo dos comandos:

- Comando ‘a’: Cuando se pulse la letra ‘a’ los LEDs de la placa tendrán que cambiar de valor. Si están apagados se encenderán y si están encendidos se apagarán.
- Comando ‘b’: Cuando se pulse la letra ‘b’ se tiene que solicitar pulsar otra letra del teclado y encender los LEDs con el código ASCII de la tecla pulsada.

12. Para poder probar el código tendrá que realizar los pasos 21, 22 y 23 de la práctica 0.

13. Los resultados esperados son los siguientes:

```

COM4 - Tera Term VT
File Edit Setup Control Window Help

Serial Port Properties -----
Device ID : 0
Baud Rate : 9600
Data Bits : 8
Base Addr : 40600000

Options -----
a - Change the LED
b - Press a key to see its ASCII value show up on the LEDs

GPIO LED is off, turning on! ← COMANDO: 'a'

Options -----
a - Change the LED
b - Press a key to see its ASCII value show up on the LEDs

Enter a key to see its value show up on the LEDs ← 1. COMANDO: 'b'
Programming LED with ASCII representation of : f ← 2. TECLA 'f' PRESIONADA
  
```

Figura 3.8 Resultados Terminal practica 1

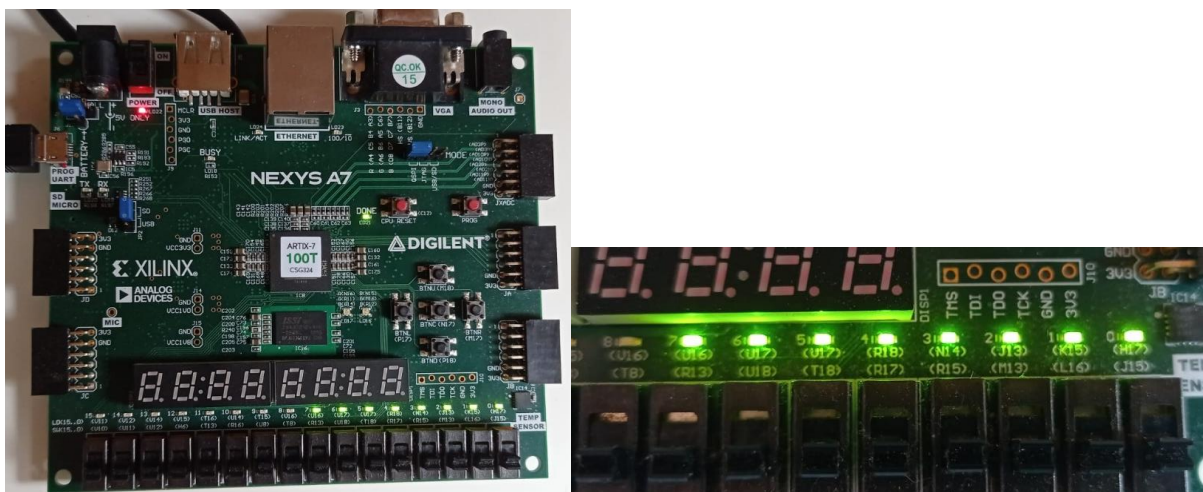


Figura 3.9 Comando 'a'

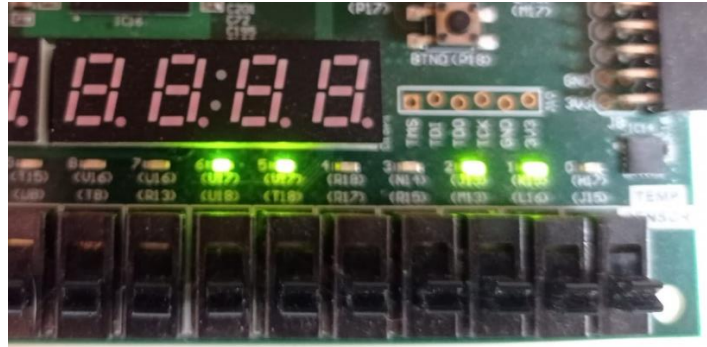


Figura 3.10 Comando 'b' pulsando la tecla 'f' del teclado

3.4.ERRORES COMUNES

Si en algún momento es necesario rediseñar los bloques en el diagrama por cualquier motivo hay que volver a generar el bitstream y exportar el hardware. Cuando se realiza este paso la mayoría de las veces se generan errores en el SDK con las librerías.

Se puede solucionar de la siguiente manera:

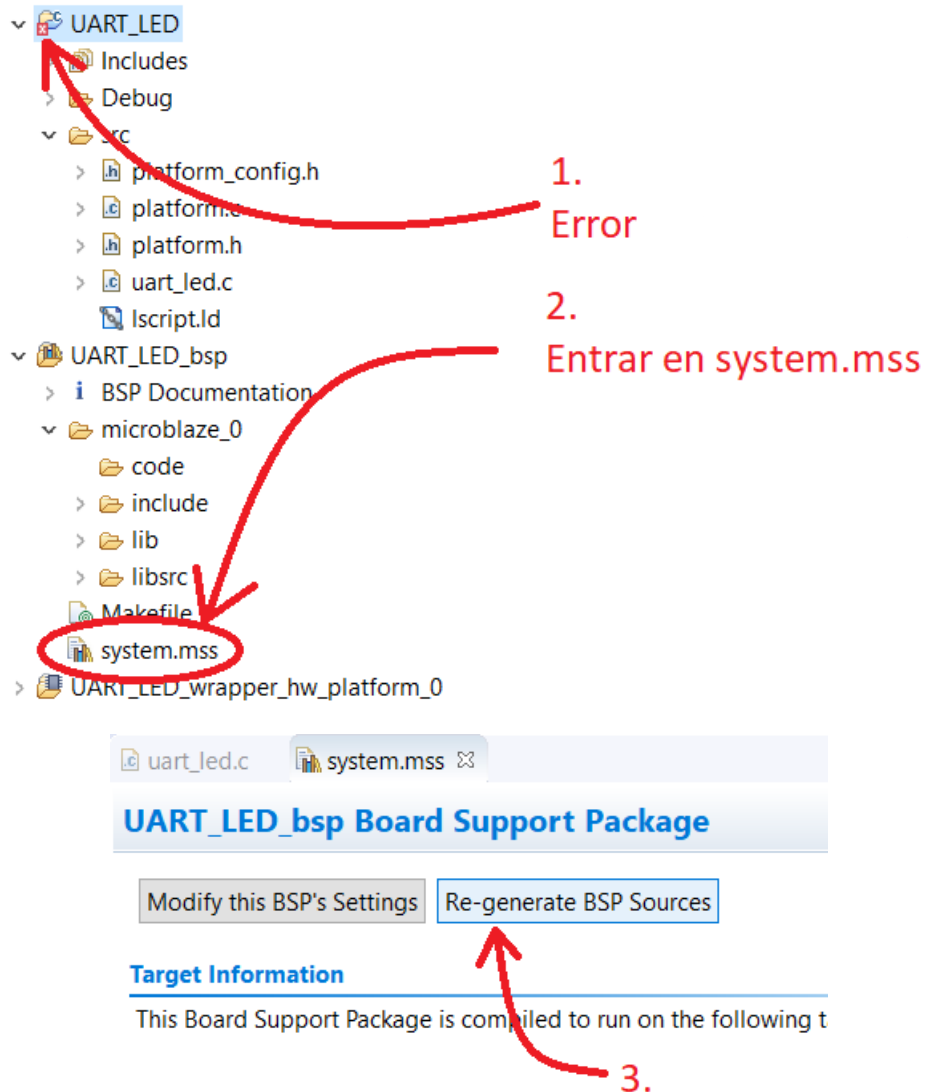


Figura 3.11 Error en las Librerías

Si el error no desaparece tras esto significa que el error es del *Linker*. El problema en esta situación reside en que el *Linker Script* no ha podido actualizarse de forma automática por lo que habrá que hacerlo manualmente.

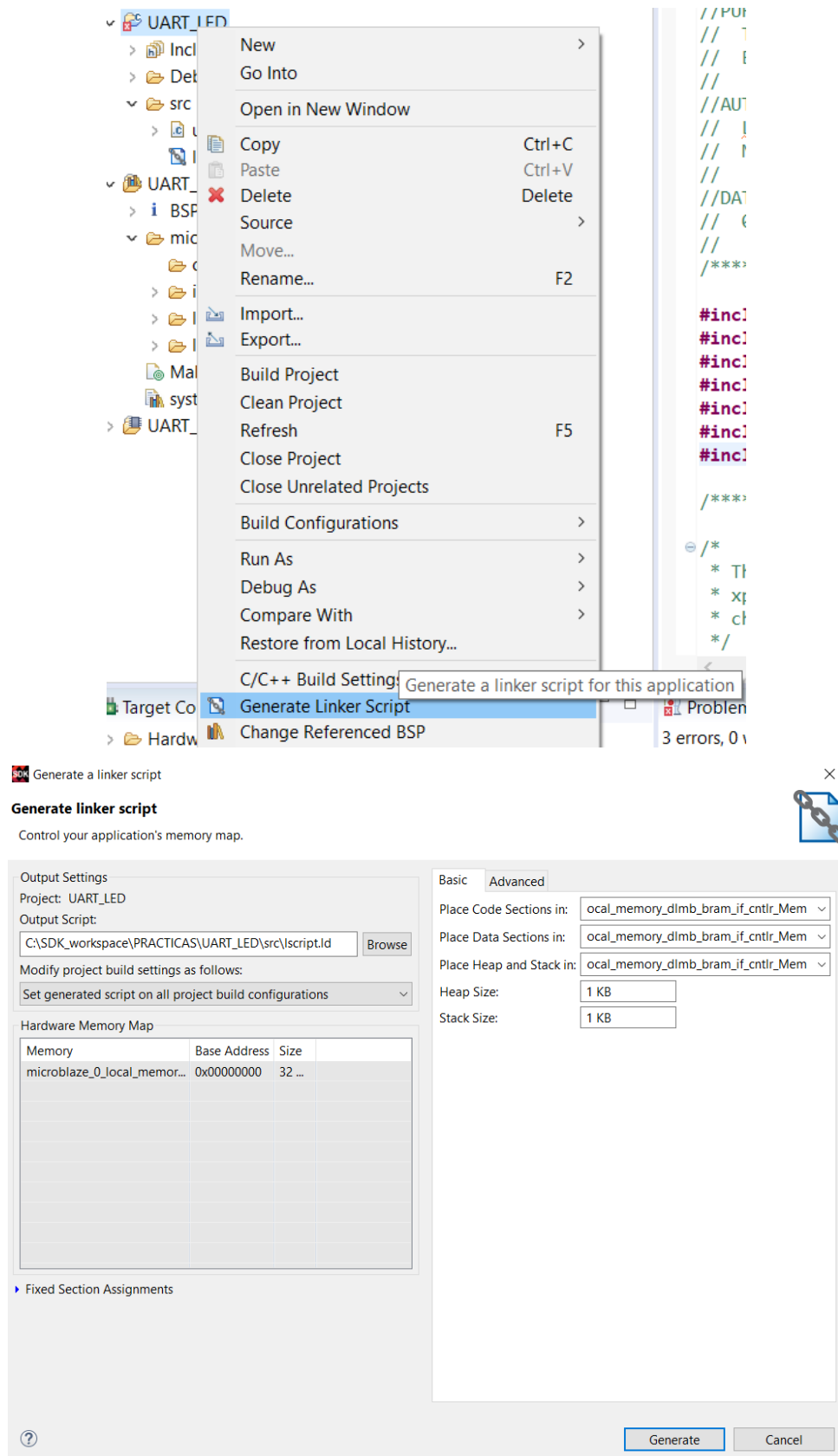


Figura 3.12 *Generate Linker Script*

4. PRÁCTICA 2: INTRODUCCIÓN AL USO DE LA MEMORIA DDR2 DE LA PLACA

4.1. INTRODUCCIÓN

Una vez que se ha adquirido el conocimiento necesario para manejar la interfaz UART mediante comandos escritos en el teclado del ordenador, se ha diseñado una práctica básica para llevar a cabo operaciones de escritura y lectura sencillas en la memoria DDR2 de la placa Nexys A7. Esta práctica busca proporcionar una comprensión práctica de cómo funciona la UART y cómo se puede utilizar para comunicarse con un dispositivo externo, en este caso, la memoria DDR2 de la placa Nexys A7. Además, esta práctica es una excelente manera de familiarizarse con los conceptos básicos de la comunicación de datos y de aprender a utilizar herramientas específicas para llevar a cabo estas tareas. En general, esta práctica es una excelente oportunidad para aplicar los conocimientos adquiridos sobre el manejo de la UART y mejorar habilidades en el campo de la comunicación de datos.

4.2. OBJETIVOS DE LA PRÁCTICA

- I. Comprender el funcionamiento de la memoria DDR2 de la placa
- II. Aprender a realizar lecturas y escrituras en memoria
- III. Coordinar las comunicaciones entre la *UART* y la DDR2

4.3. DESARROLLO

1. Realice los pasos del 1 al 7 de la práctica 1. Es decir, cree un proyecto nuevo y en el diagrama de bloques añada *MicrBlaze*, *USB UART* y *16 LEDs*; Configure el *Clocking Wizard* y conecte los módulos de las interrupciones apropiadamente.
2. En la pestaña *Board* seleccione el módulo *DDR2 SDRAM* y añádalo al diagrama.

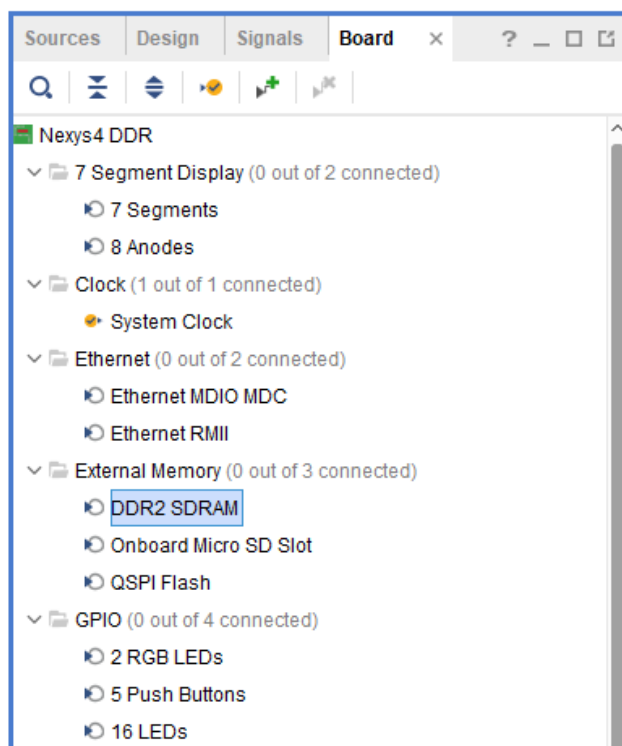


Figura 4.1 DDR2 SDRAM

Se añadirá un módulo MIG (*Memory Interface Generator*) preconfigurado para la memoria DDR2 de la placa. Para hacer esta configuración manualmente consultar *Using MIG in the Vivado Design Suite de la guía Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v4.2* [9], donde viene una serie de pasos detallados en los que se explica cómo llevar a cabo esta configuración.

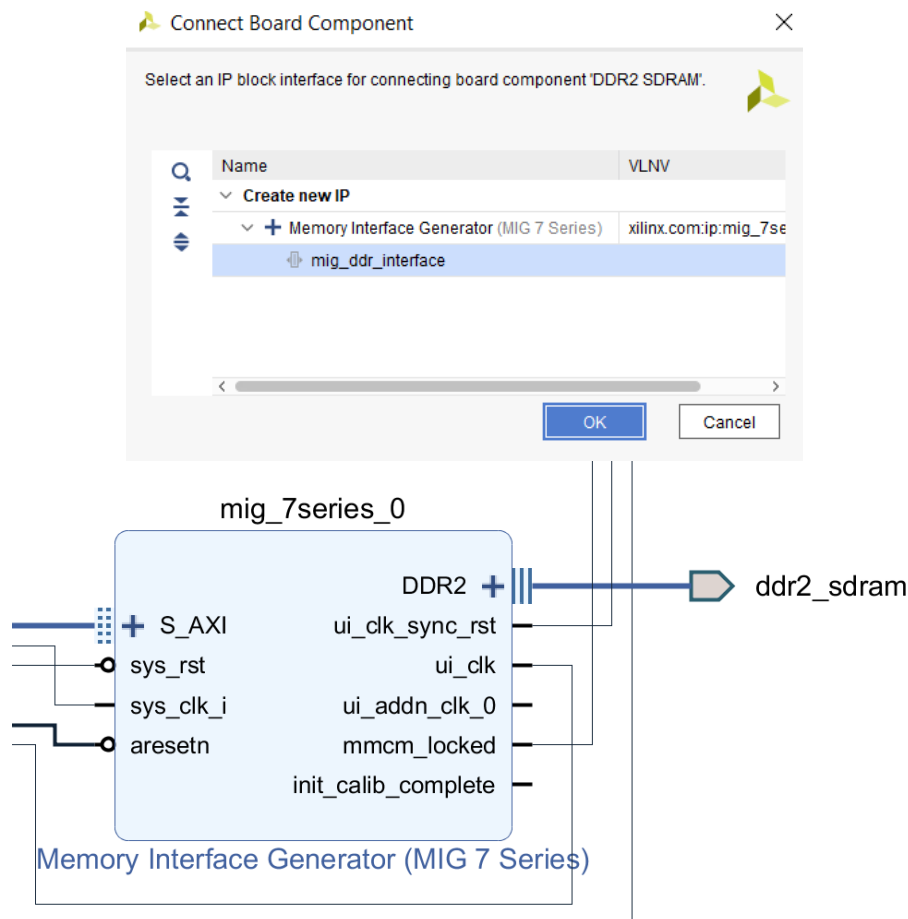


Figura 4.2 Módulo MIG

- Para que la memoria DDR2 funcione correctamente el módulo *MIG* necesita un reloj de 200MHz. Por lo tanto habrá que reconfigurar el *Clocking Wizard* para añadir una salida de un reloj de 200MHz.

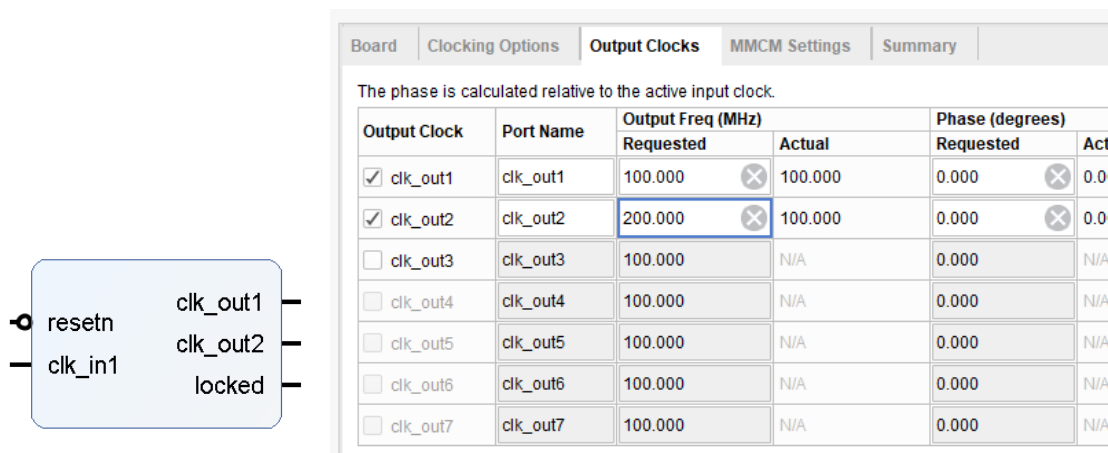


Figura 4.3 200MHz clock

4. Pulse *Run Connection Automation*. Seleccione todos los módulos y reconfigure los parámetros de entrada del *MIG* para que el reloj de entrada sea el reloj de 200MHz configurado anteriormente y que el *reset* sea el del sistema.

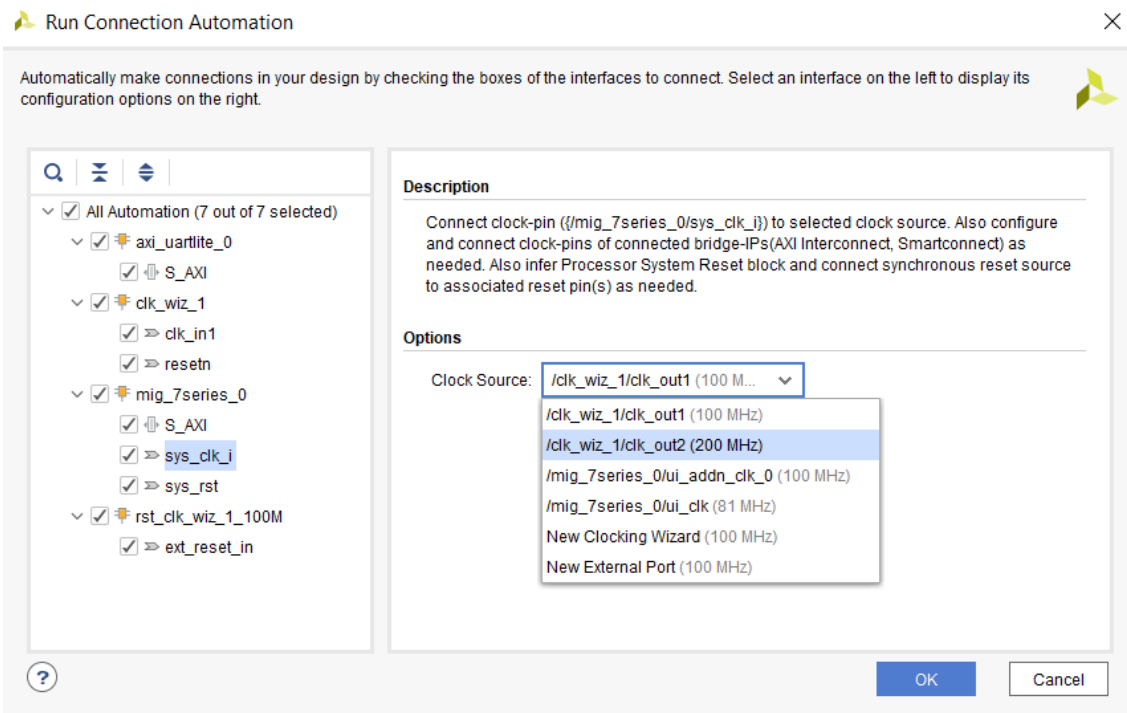


Figura 4.4 Clk MIG

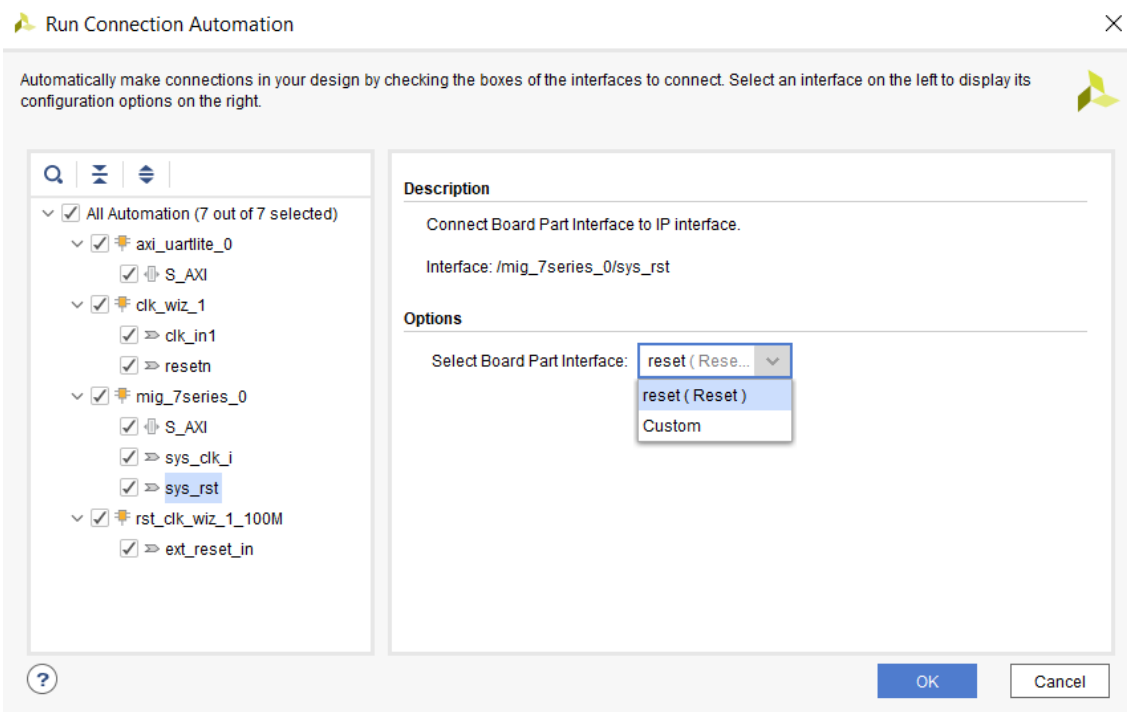


Figura 4.5 Rst MIG

El resultado será parecido al siguiente:

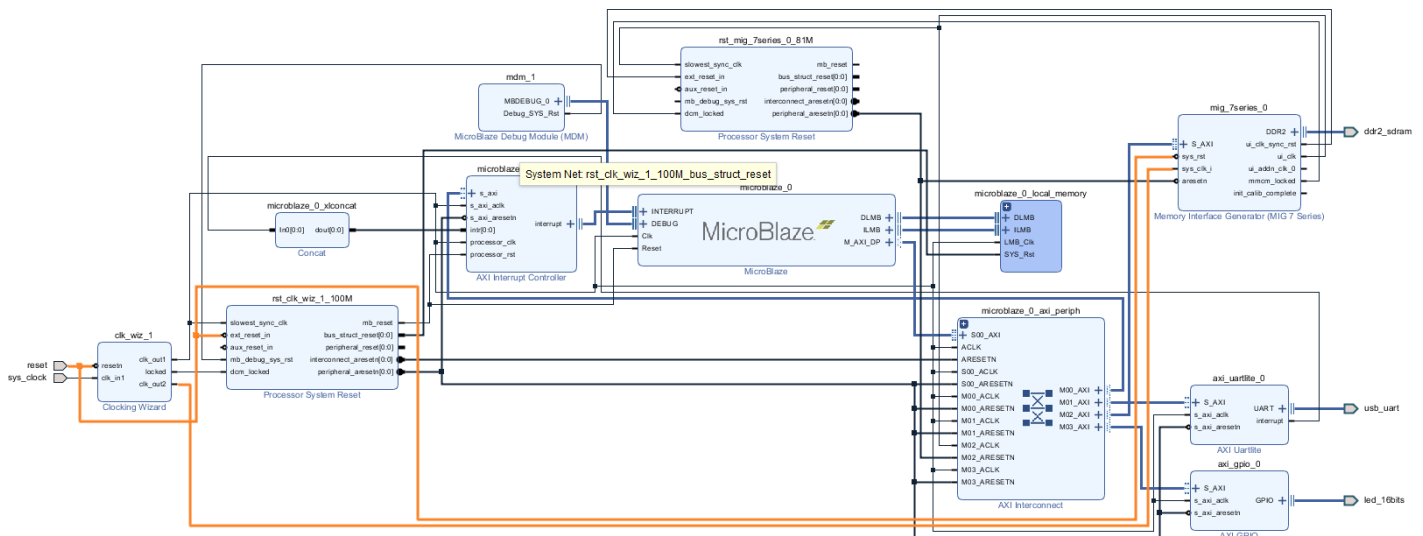


Figura 4.6 Layout práctica 2

5. realice los pasos 8 y 9 de la práctica 1.
6. Seleccione el template: *Memory test*. Seleccionando este ejemplo se añadirán las librerías necesarias para la comunicación con la memoria de forma automática.

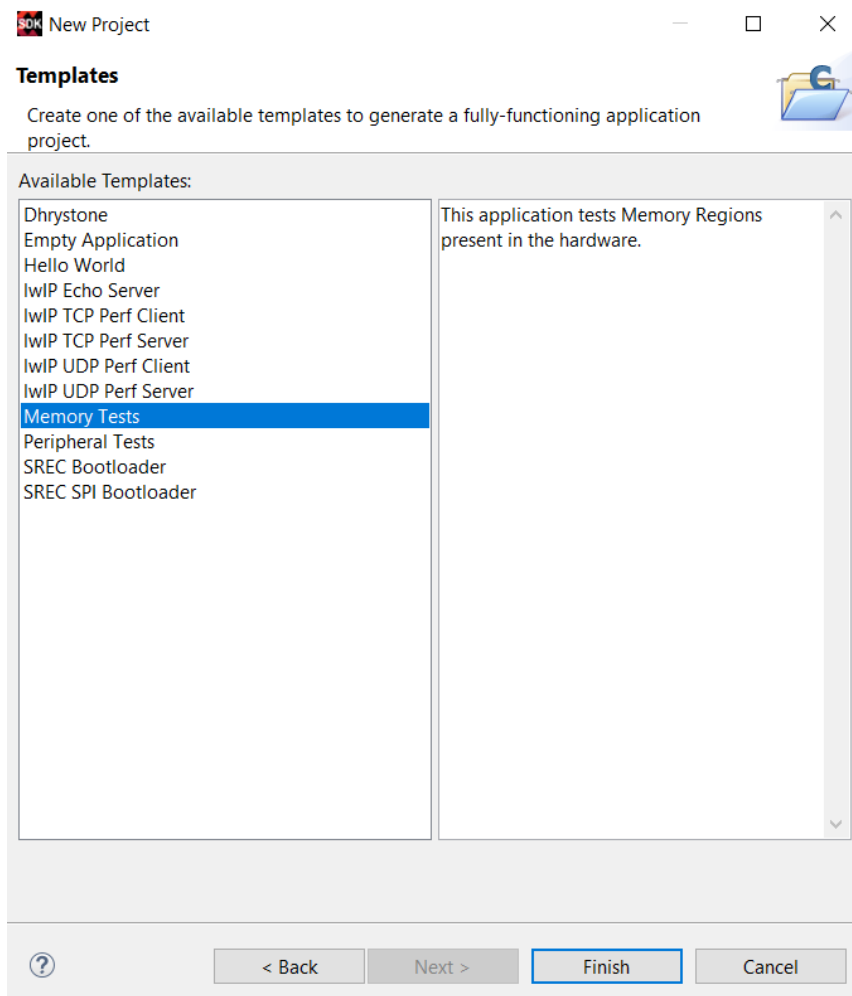


Figura 4.7 Memory Tests

7. Reemplace el contenido del archivo *Memorytest.c* por el esqueleto del “Anexo C: Esqueleto de ejemplo de código para el manejo de la UART con comandos”.

(Nota: es recomendable echar un vistazo a este código antes de eliminarlo, ya que contiene las claves para escribir y leer datos de la memoria)

Esta vez el código a desarrollar tiene que, aparte de ser capaz de detectar que tecla ha sido pulsada en el teclado del ordenador y actuar como en la anterior práctica, realizar lecturas y escrituras en la memoria DDR2. Habrá mínimo dos comandos:

- Comando ‘a’: Cuando se pulse la letra ‘a’ se tendrá que leer el valor de alguna dirección de la memoria, la primera por ejemplo. Este valor tendrá que mostrarse en los LEDs de la placa.
- Comando ‘b’: Cuando se pulse la letra ‘b’ se tiene que solicitar pulsar otra letra del teclado y escribir el código ASCII de la tecla pulsada en la misma dirección que se elija para el comando anterior.

8. Los resultados esperados tienen que ser similares a los siguientes:

```

VT COM4 - Tera Term VT
File Edit Setup Control Window Help

Serial Port Properties -----
Device ID : 0
Baud Rate : 9600
Data Bits : 8
Base Addr : 40600000

Options -----
a - Read the Addr: 80000000; and show its ASCII value in up on the LEDs
b - Press a key to write its ASCII value in the Addr: 80000000

Enter a key to write its ASCII value in the DDR2
Actual Data : 00000079
Wrote Data : 00000079

Options -----
a - Read the Addr: 80000000; and show its ASCII value in up on the LEDs
b - Press a key to write its ASCII value in the Addr: 80000000

Actual Data : 00000079
Programming LED with ASCII representation of : y

```

Figura 4.8 Resultados práctica 2

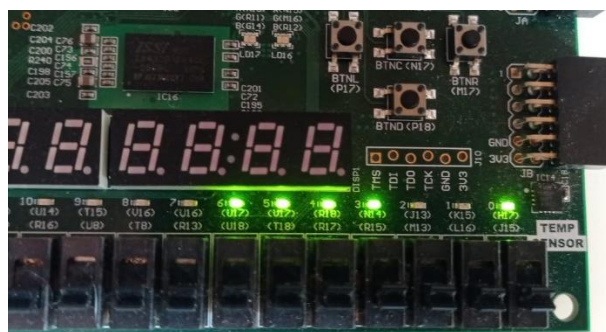


Figura 4.9 LEDs de la placa representando el código ASCII de la letra 'y'

5. PRÁCTICA 3: COMUNICACIÓN ENTRE LENGUAJE RTL Y MICROBLAZE

5.1. INTRODUCCIÓN

En esta práctica se utilizarán los conocimientos y habilidades adquiridos en la práctica anterior y se combinarán con el uso de bloques RTL personalizados para un uso específico. El objetivo principal de esta práctica es aprender a integrar estos bloques personalizados en un diseño existente y comprender cómo se realizan las comunicaciones entre estos bloques y el resto del sistema.

5.2. OBJETIVOS DE LA PRÁCTICA

- I. Consolidar los conocimientos y habilidades de la anterior práctica
- II. Añadir módulos RTL al diseño de bloques
- III. Comunicación entre módulos RTL y *MicroBlaze*

5.3. DESARROLLO

1. Realice los pasos 1 al 4 de la práctica 2.
2. En la pestaña *Board* añada los periféricos de los *LED* y de los *Switches*. Se añadirá un módulo *AXI GPIO* con dos canales automáticamente.

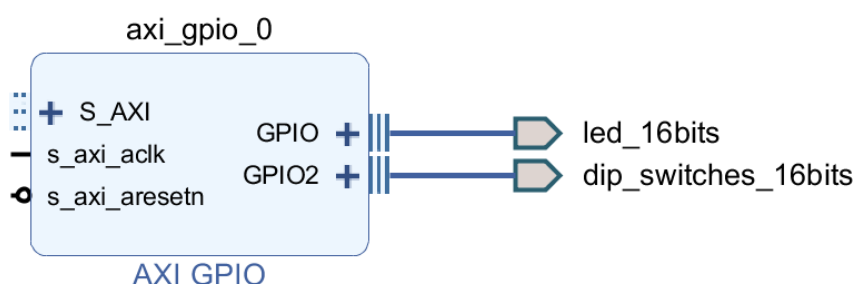


Figura 5.1 GPIO LED y Switches

3. Añada un módulo RTL al diseño. En la pestaña *Sources* pulse en *Add Sources*.

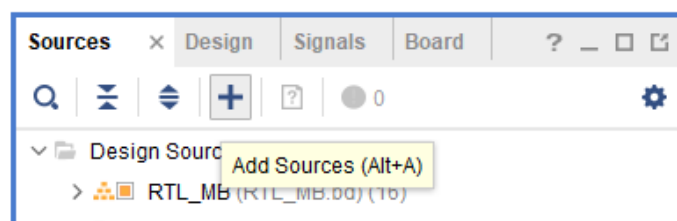


Figura 5.2 Add Sources

Seleccione la opción *desing sources*.

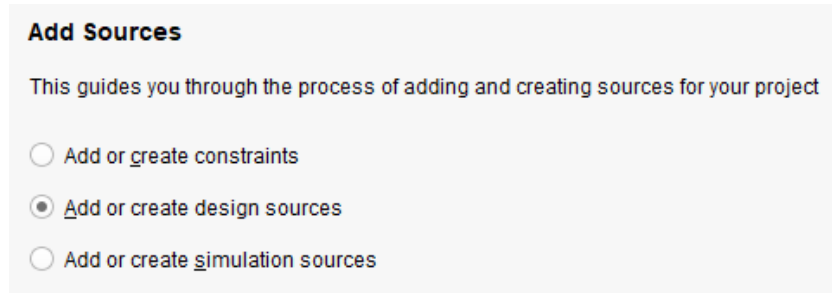


Figura 5.3 Desing Sources

Seleccione el tipo de archivo como VHDL. Al archivo lo llamaremos *Calculator*.

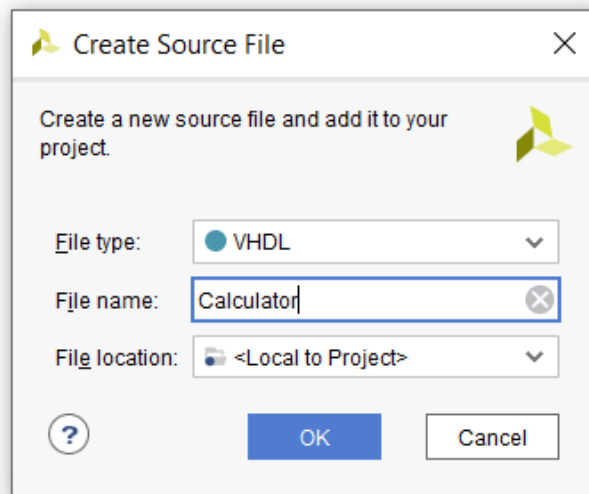


Figura 5.4 *Calculator*

Este módulo tiene el simple objetivo de recibir datos mediante *AXI*, realizar una operación simple con ellos, tipo *AND*, *OR* o una multiplicación. En la presente practica se realizará una multiplicación por 2. Se puede realizar con las siguientes entradas y salidas: *Data_in* y *Data_out* serán los datos que entren y los calculados respectivamente; *valid* será la señal que confirma que los datos recibidos son correctos cuando tenga valor '1'; finalmente *rdy* indicará que el módulo ya ha realizado el cálculo y puede comenzar con otro.

Module Definition

Entity name:

Architecture name:

I/O Port Definitions

+ - ↑ ↓

Port Name	Direction	Bus	MSB	LSB
Data_in	in	<input checked="" type="checkbox"/>	15	0
valid	in	<input type="checkbox"/>	0	0
rdy	out	<input type="checkbox"/>	0	0
Data_out	out	<input checked="" type="checkbox"/>	15	0

Figura 5.5 Entradas y salidas del módulo RTL

- Una vez escrito el código, clicando en el botón derecho del ratón en el diagrama de bloques encontrará la opción para añadir el módulo al diseño.

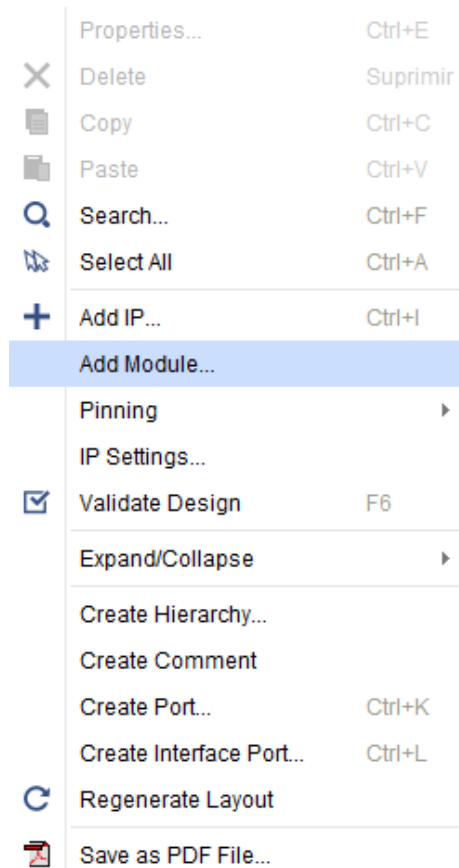


Figura 5.6 Add module

Aparecerán las opciones de los módulos RTL disponibles para añadir al diseño. En este caso solo aparecerá el que acabamos de diseñar.

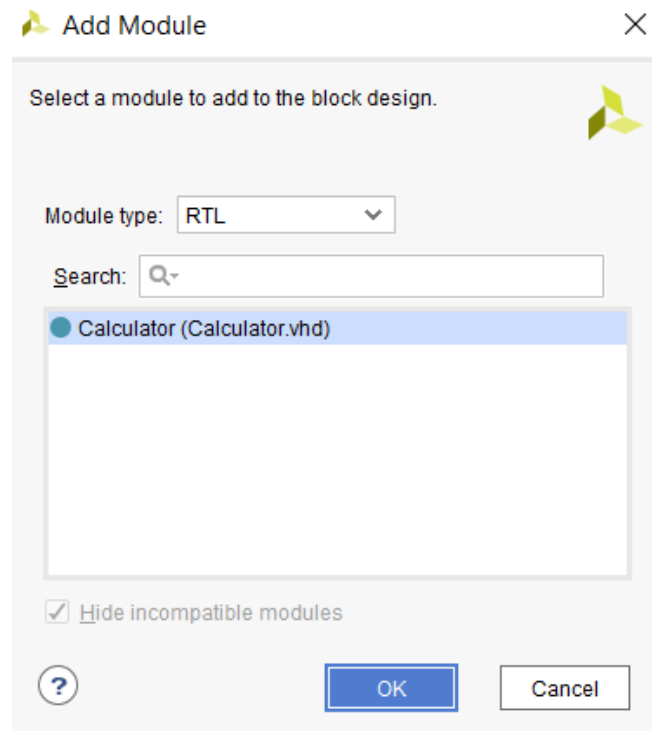


Figura 5.7 Selección de módulos RTL para añadir al diseño

Le aparecerá el siguiente bloque en el diseño:

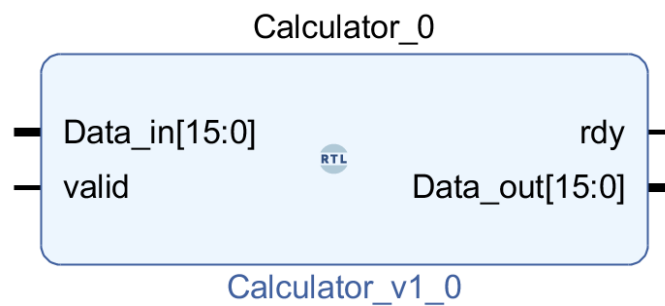


Figura 5.8 Bloque RTL

5. Para poder comunicarse mediante AXI se van a añadir módulos *AXI GPIO*.

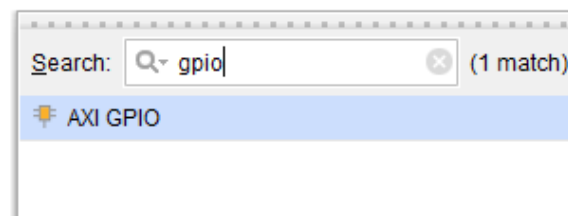


Figura 5.9 AXI GPIO

Cada módulo *AXI GPIO* puede tener hasta dos canales por lo que habrá que añadir dos módulos con las siguientes configuraciones.

Board IP Configuration

GPIO

☐ All Inputs

☒ All Outputs

GPIO Width: 16 [1 - 32]

Default Output Value: 0x00000000 [0x00000000, 0xFFFFFFFF]

Default Tri State Value: 0xFFFFFFFF [0x00000000, 0xFFFFFFFF]

☒ Enable Dual Channel

GPIO 2

☒ All Inputs

☐ All Outputs

GPIO Width: 16 [1 - 32]

Default Output Value: 0x00000000 [0x00000000, 0xFFFFFFFF]

Default Tri State Value: 0xFFFFFFFF [0x00000000, 0xFFFFFFFF]

☐ Enable Interrupt

Figura 5.10 Configuración AXI GPIO n°1

Board IP Configuration

GPIO

☐ All Inputs

☒ All Outputs

GPIO Width: 1 [1 - 32]

Default Output Value: 0x00000000 [0x00000000, 0xFFFFFFFF]

Default Tri State Value: 0xFFFFFFFF [0x00000000, 0xFFFFFFFF]

☒ Enable Dual Channel

GPIO 2

☒ All Inputs

☐ All Outputs

GPIO Width: 1 [1 - 32]

Default Output Value: 0x00000000 [0x00000000, 0xFFFFFFFF]

Default Tri State Value: 0xFFFFFFFF [0x00000000, 0xFFFFFFFF]

☐ Enable Interrupt

Figura 5.11 Configuración AXI GPIO n°2

Para facilitar el trabajo futuro en *SDK* cambiaremos los nombres de los módulos. El primero será para los datos por lo que le llamaremos *DATA_gpio* y el segundo para las señales de *valid* y *rdy* por lo que un nombre podría ser *valid_rdy_gpio*.

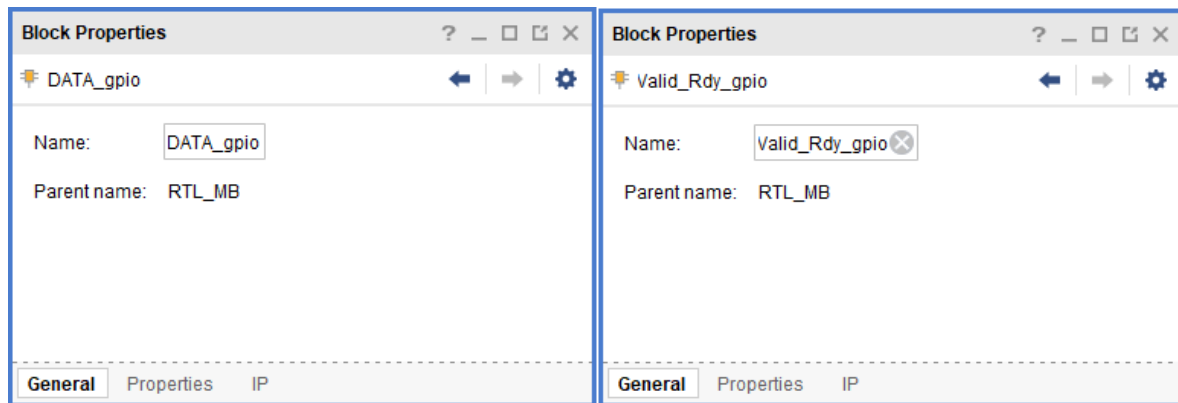


Figura 5.12 Nombres módulos AXI GPIO pt1

Quedando el siguiente resultado:

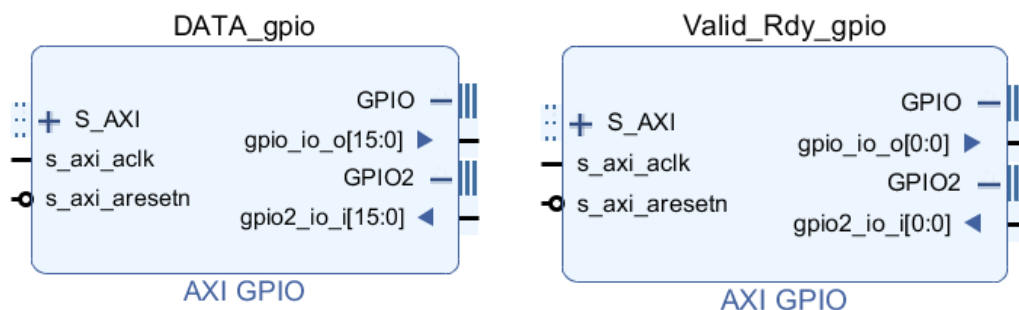


Figura 5.13 Nombres módulos AXI GPIO pt2

Por la misma razón también cambiaremos el nombre del módulo que conecta con los *LEDs* y *Switches* de la placa.

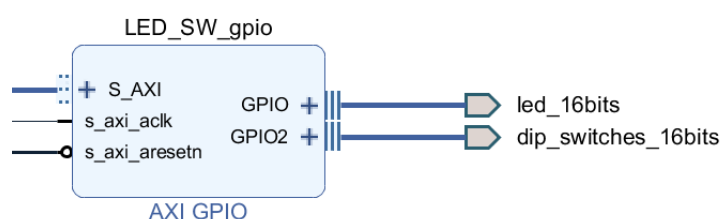
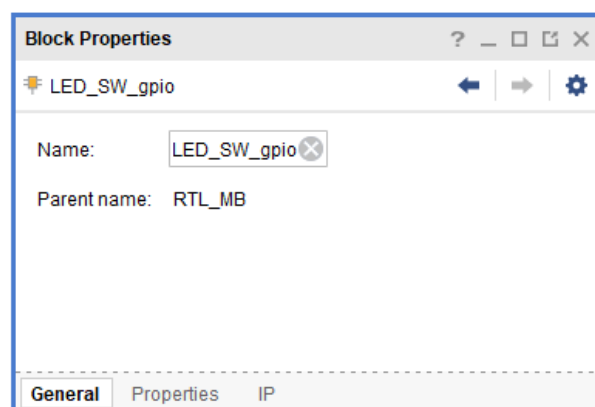


Figura 5.14 LED_SW_GPIO

-
- The diagram illustrates the RTL schematic of the AXI GPIO block. It consists of two main components: **DATA_gpio** and **Valid_Rdy_gpio**, both labeled as **AXI GPIO**. Each component has three inputs: **S_AXI** (with a bus width of 1024), **s_axi_aclk**, and **s_axi_aresetn**. **DATA_gpio** has two outputs: **gpio_io_o[15:0]** (GPIO output) and **gpio2_io_i[15:0]** (GPIO2 input). **Valid_Rdy_gpio** has two outputs: **gpio_io_o[0:0]** (GPIO output) and **gpio2_io_i[0:0]** (GPIO2 input). The **Calculator_0** block takes **Data_in[15:0]** and **valid** as inputs and outputs **rdy** and **Data_out[15:0]**. The **Calculator_v1_0** block is also present but not connected to any signals.

Figura 5.15 Conexiones módulo RTL





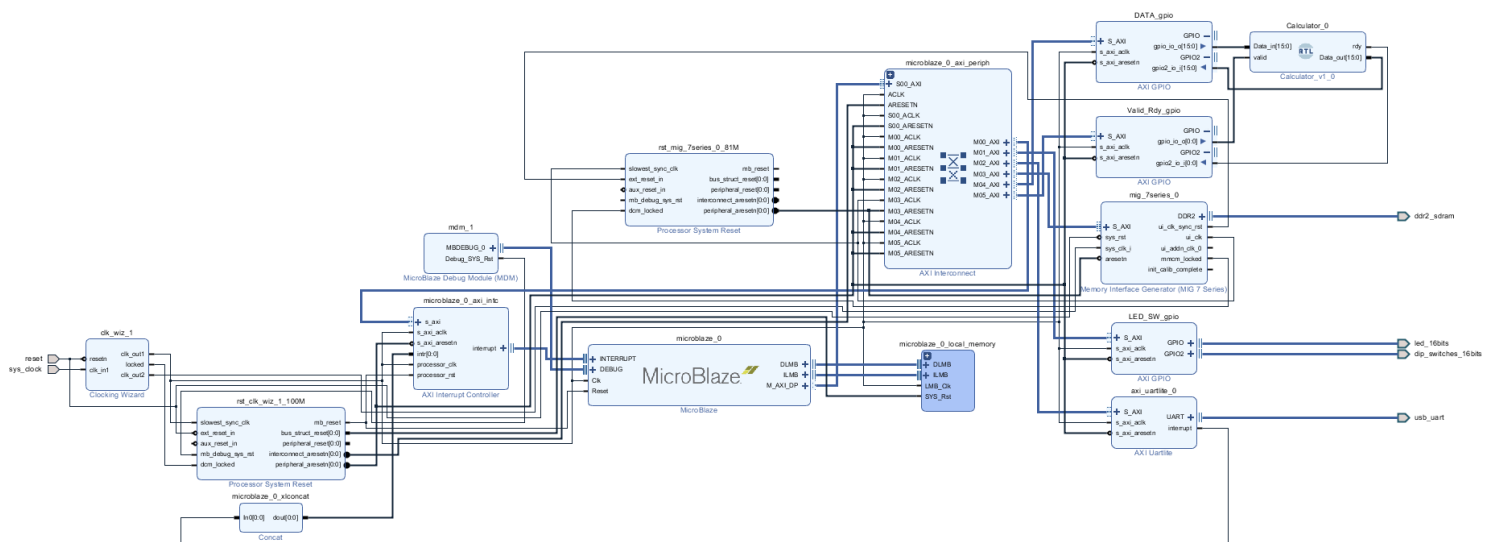
- ☒ All Automation (2 out of 2 selected)
 - ☒  DATA_gpio
 - ☒  S_AXI
 - ☒  Valid_Rdy_gpio
 - ☒  S_AXI

Figura 5.16 Run Connection Automation módulos AXI

Se obtendrá un resultado como este:

Figura 5.17 *Layout* práctica 3

8. Realice el paso 5 de la práctica 2.
9. Una vez en *SDK* cree un proyecto vacío.
10. El código por desarrollar tendrá dos comandos, por lo que habrá que recurrir al “*Anexo C: Esqueleto de ejemplo de código para el manejo de la UART con comandos*” añadiendo las siguientes líneas para poder acceder a los módulos *GPIO*.

```
#define GPIO1_ID                                XPAR_GPIO_0_DEVICE_ID
#define INPUT                                    0xFF
#define OUTPUT                                    0x00

XGpio Gpio1; /* The Instance of the GPIO Driver */
Status = XGpio_Initialize(&Gpio1, GPIO1_ID);
if (Status != XST_SUCCESS) {
    xil_printf("Gpio Initialization Failed\r\n");
    return XST_FAILURE;
}

XGpio_SetDataDirection(&Gpio1, *Channel*, *INPUT/OUTPUT*);

u32 Data;
Data = XGpio_DiscreteRead(&Gpio1, *Channel*);
XGpio_DiscreteWrite(&Gpio1, *Channel*, Data);
```

Los comandos tendrán que hacer lo siguiente:

- Comando ‘a’: escribirá en la primera dirección de la memoria DDR2 el valor que haya en los *Switches* de la placa.
- Comando ‘b’: leerá el valor guardado en la primera dirección de la memoria DDR2 y se lo proporcionará al módulo RTL para que opere con estos datos. El resultado de esta operación tendrá que verse en los *LEDs* de la placa.

11. Los resultados de la práctica son los siguientes.

```
Options -----
a - Write the SW value in the DDR address
b - Operate with the value stored in the address

SW Value : 00004AA7
Wrote Data : 00004AA7
Addr : 80000000

Options -----
a - Write the SW value in the DDR address
b - Operate with the value stored in the address

Addr : 80000000
DDR2 Data : 00004AA7
LED Value : 0000954E
```

Figura 5.18 Resultados en el terminal de la práctica 3



Figura 5.19 Resultados en la placa de la práctica 3

6. PRÁCTICA 4: COMUNICACIÓN ENTRE FSM ESCRITAS EN LOS LENGUAJES VHDL Y C

6.1. INTRODUCCIÓN

En esta práctica se buscará afianzar los conocimientos adquiridos en la práctica anterior en cuanto a comunicación entre módulos RTL y *MicroBlaze*. Además, se buscará lograr una comunicación coordinada entre dos máquinas de estados, una funcionando en VHDL y otra en lenguaje C, es decir, una funcionando en la FPGA como bloque RTL y otra funcionando en el soft-core *Microblaze*. Para lograr esto, se añadirán módulos de depuración con el objetivo de monitorear el valor de determinadas señales en tiempo real, lo que ayudará a garantizar que la comunicación entre los módulos sea correcta tanto en cuanto a los valores de las señales como en cuanto al momento y duración de estas.

En esta práctica se aprenderá a utilizar herramientas específicas de depuración y monitoreo para analizar el comportamiento de los módulos y detectar cualquier problema en la comunicación. Además, se aprenderá a solucionar problemas relacionados con la comunicación entre módulos, lo que es esencial para el diseño y desarrollo de sistemas embebidos y otros sistemas digitales complejos.

6.2. OBJETIVOS DE LA PRÁCTICA

- I. Afianzar conocimientos de comunicación entre módulos RTL y *MicroBlaze*
- II. Diseñar máquinas de estados en diferentes lenguajes
- III. Comunicación entre máquinas de estados
- IV. Aprender a debuggear en tiempo real sistemas mixtos (VHDL y C)

6.3. DESARROLLO

1. Realice los siete primeros pasos de la práctica 1. Es decir, crear el proyecto y el diagrama de bloques.
2. Añada un módulo *MicroBlaze* y configure el *Run Block Automation* de la siguiente manera.

Options

Preset:	None ▼
Local Memory:	64KB ▼
Local Memory ECC:	None ▼
Cache Configuration:	None ▼
Debug Module:	Debug O... ▼
Peripheral AXI Port:	Enabled ▼
Interrupt Controller:	<input checked="" type="checkbox"/>
Clock Connection:	New Clocking Wizard (100 M... ▼)

Figura 6.1 Configuración *MicroBlaze* práctica 4

3. En la pestaña *Board* añada los periféricos de la memoria DDR2 y la UART.
4. Realice el paso 3 de la práctica 3. Esta vez el código será una máquina de estados finitos. Esta máquina tendrá que solicitar y enviar datos a la memoria DDR2. Tendrá que solicitar las dos primeras direcciones de memoria, con los datos que le sean proporcionados realizar una

operación sencilla estilo *AND* y el resultado tendrá que ser escrito en la tercera dirección de la memoria DDR2.

Se proponen las siguientes entradas y salidas para diseñar esta máquina.

- *Valid* indicará que los datos recibidos son correctos.
- *OP_req* será una señal que solicite una nueva operación.
- *Data_in* y *Data_out* serán los datos que entren y salgan de la memoria.
- *Addr* será la dirección de memoria necesaria.
- *Ready* indicará que se ha finalizado la operación y la máquina de estados está disponible para una nueva operación.

```
entity RTL_FSM is
  Port ( clk : in STD_LOGIC;
        rst : in STD_LOGIC;
        valid : in STD_LOGIC;
        OP_req : in STD_LOGIC;
        Data_in : in STD_LOGIC_VECTOR (31 downto 0);
        Addr : out STD_LOGIC_VECTOR (31 downto 0);
        ready : out STD_LOGIC;
        Data_out : out STD_LOGIC_VECTOR (31 downto 0)
        );
end RTL_FSM;
```

Figura 6.2 I/O FSM Práctica 4

5. Realice el paso 4 de la práctica 3.

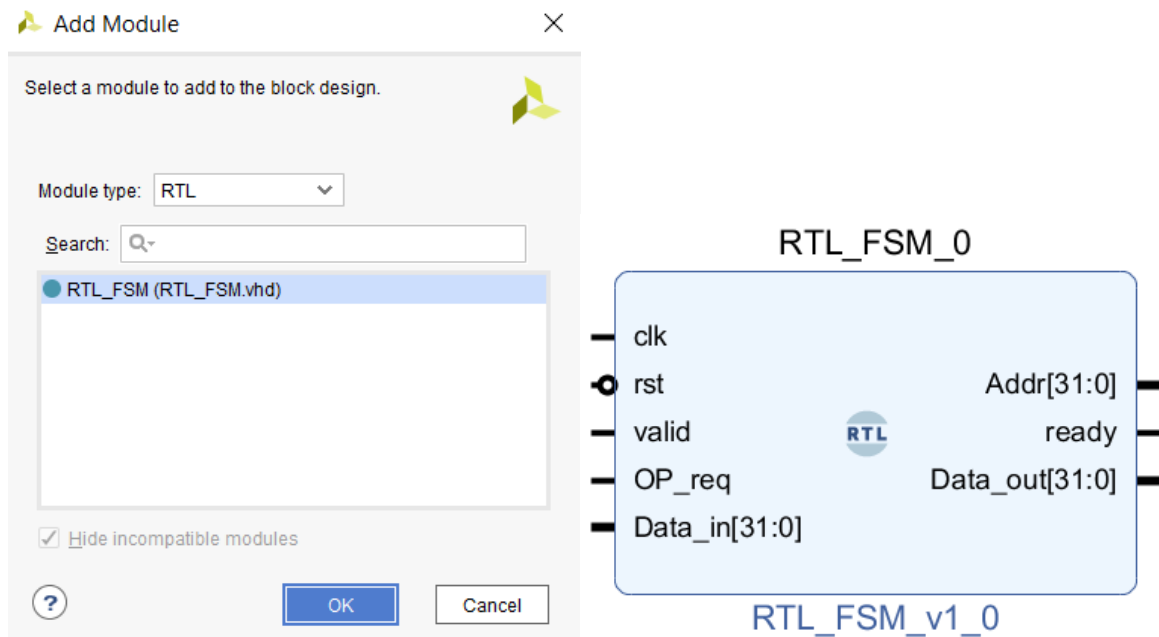


Figura 6.3 Módulo RTL_FSM

6. Realice los pasos 5 y 6 de la práctica 3 y conecte los puertos a los *AXI GPIO* de la siguiente manera.



The diagram illustrates a MicroBlaze SoC architecture. At the center is the **MicroBlaze** core, which interfaces with an **AXI Interconnect**. This interconnect manages data flow between the core and various peripheral blocks:

- MicroBlaze_0_axi_periph**: Provides AXI interfaces for the core, including S_AXI, M00_AXI, M01_AXI, M02_AXI, M03_AXI, M04_AXI, M05_AXI, and M06_AXI.
- MicroBlaze_0_axi_intc**: An **AXI Interrupt Controller** that manages interrupts for the core.
- MicroBlaze_0_axi_concat**: A concatenation block for AXI signals.
- MicroBlaze_0_axi_uartlite**: An **AXI UARTlite** block for serial communication.
- MicroBlaze_0_axi_gpio**: An **AXI GPIO** block for general-purpose input/output.
- MicroBlaze_0_axi_rdy_gpio**: An **AXI GPIO** block for ready signals.
- MicroBlaze_0_axi_valid_addr_gpio**: An **AXI GPIO** block for valid address signals.
- MicroBlaze_0_axi_uartlite_0**: An **AXI UARTlite** block for serial communication.
- MicroBlaze_0_axi_usb_uart**: An **AXI USB UART** block for USB communication.
- MicroBlaze_0_axi_dsp**: An **AXI DSP** block for digital signal processing.
- MicroBlaze_0_axi_dma**: An **AXI DMA** block for direct memory access.
- MicroBlaze_0_axi_sram**: An **AXI SRAM** block for static random access memory.
- MicroBlaze_0_axi_flash**: An **AXI Flash** block for non-volatile storage.
- MicroBlaze_0_axi_nand**: An **AXI NAND** block for NAND flash storage.
- MicroBlaze_0_axi_nor**: An **AXI NOR** block for NOR flash storage.
- MicroBlaze_0_axi_rom**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_0**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_1**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_2**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_3**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_4**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_5**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_6**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_7**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_8**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_9**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_10**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_11**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_12**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_13**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_14**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_15**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_16**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_17**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_18**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_19**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_20**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_21**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_22**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_23**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_24**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_25**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_26**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_27**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_28**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_29**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_30**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_31**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_32**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_33**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_34**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_35**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_36**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_37**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_38**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_39**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_40**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_41**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_42**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_43**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_44**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_45**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_46**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_47**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_48**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_49**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_50**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_51**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_52**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_53**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_54**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_55**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_56**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_57**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_58**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_59**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_60**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_61**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_62**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_63**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_64**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_65**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_66**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_67**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_68**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_69**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_70**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_71**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_72**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_73**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_74**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_75**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_76**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_77**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_78**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_79**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_80**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_81**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_82**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_83**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_84**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_85**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_86**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_87**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_88**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_89**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_90**: An **AXI ROM** block for read-only memory.
- MicroBlaze_0_axi_rom_91**: An **AXI ROM** block for read-only memory.
- MicroBl**

Figura 6.5 *Layout* práctica 4

8. Realice el paso 5 de la práctica 2.
9. Una vez en *SDK* diseñe una máquina de estados usando la librería *fsm.h* que le será proporcionada. La máquina tendrá que comunicarse con la máquina de estados RTL:
 - Solicitando las operaciones.
 - Gestionando las solicitudes de lectura y escritura, es decir, leer y escribir datos de la memoria y proporcionarlos al RTL.
10. Para poder “debuguear” el sistema y conseguir un correcto funcionamiento del sistema vamos a tener que recurrir a módulos de “debugueo”. Añadiremos, por ejemplo, las siguientes señales:
 - Para comprobar que los estados cambian según lo esperado: *st*. Codificaremos los estados y sacaremos esos valores en función del estado en el que se encuentre la máquina.
 - Y para comprobar que los datos se están guardando correctamente: *data1_debug* y *data2_debug*.

```
entity RTL_FSM is
  Port ( clk : in STD_LOGIC;
        rst : in STD_LOGIC;
        valid : in STD_LOGIC;
        OP_req : in STD_LOGIC;
        Data_in : in STD_LOGIC_VECTOR (31 downto 0);
        Addr : out STD_LOGIC_VECTOR (31 downto 0);
        ready : out STD_LOGIC;
        Data_out : out STD_LOGIC_VECTOR (31 downto 0);

        -- ONLY FOR DEBUGGING
        st : out STD_LOGIC_VECTOR (3 downto 0);
        data1_debug : out STD_LOGIC_VECTOR (31 downto 0);
        data2_debug : out STD_LOGIC_VECTOR (31 downto 0));
end RTL_FSM;
```

Figura 6.6 Señales debugueo

Al diagrama bloques añadiremos un módulo *ILA*. Nos dirá el valor de la señal en tiempo de ejecución.

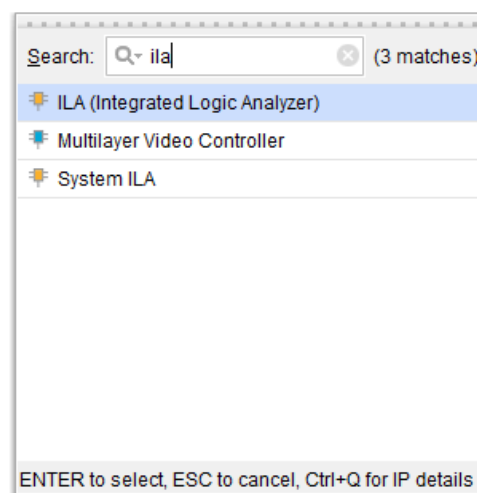


Figura 6.7 ILA

Y lo configuraremos de la siguiente manera (si ha añadido más señales para debuguear tendrá que aumentar el número de *Probes*).

To configure more than 64 probe ports use Vivado Tcl Console

General Options | Probe_Ports(0..7) | Probe_Ports(8..8)

Monitor Type

☒ Native ☐ AXI

Number of Probes: 9 [1...512]

Sample Data Depth: 2048

☒ Same Number of Comparators for All Probe Ports

Number of Comparators: 2

☐ Trigger Out Port

☐ Trigger In Port

Input Pipe Stages: 0

Trigger And Storage Settings

☒ Capture Control

☒ Advanced Trigger

GUI configuration mode is limited to 64 probe ports.

Figura 6.8 Configuración ILA

Configure las sondas del *ILA* en función de las señales que quiera comprobar. En este caso queremos mirar todas las entradas y salidas del módulo RTL.

- clk
- probe0[0:0]
- probe1[0:0]
- probe2[0:0]
- probe3[31:0]
- probe4[31:0]
- probe5[31:0]
- probe6[3:0]
- probe7[31:0]
- probe8[31:0]

To configure more than 64 probe ports use Vivado Tcl Console

General Options | **Probe_Ports(0..7)** | Probe_Ports(8..8)

Probe Port	Probe Width [1..4096]	Number of Comparators	Probe Trigger or Data
PROBE0	1	2	DATA AND TRIGGER
PROBE1	1	2	DATA AND TRIGGER
PROBE2	1	2	DATA AND TRIGGER
PROBE3	32	2	DATA AND TRIGGER
PROBE4	32	2	DATA AND TRIGGER
PROBE5	32	2	DATA AND TRIGGER
PROBE6	4	2	DATA AND TRIGGER
PROBE7	32	2	DATA AND TRIGGER

Figura 6.9 ILA probes

Conectamos coherentemente las señales del módulo RTL al ILA.

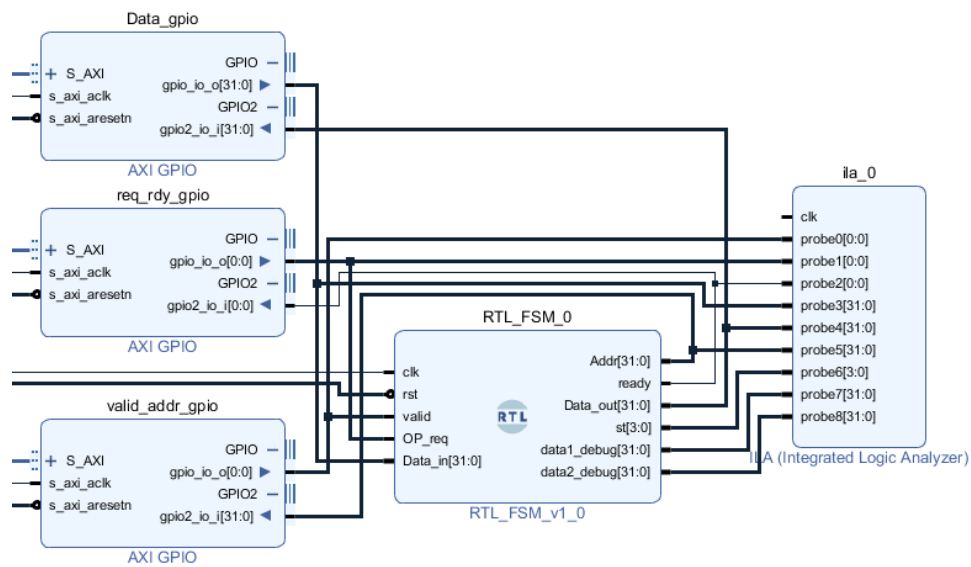


Figura 6.10 Conexión RTL con ILA

Conectamos el reloj del *ILA* al reloj del sistema:

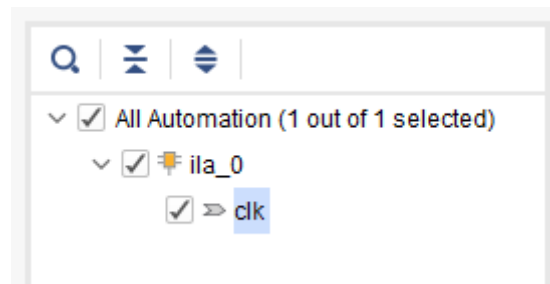


Figura 6.11 ILA clk

Se obtendrá un resultado similar al siguiente:

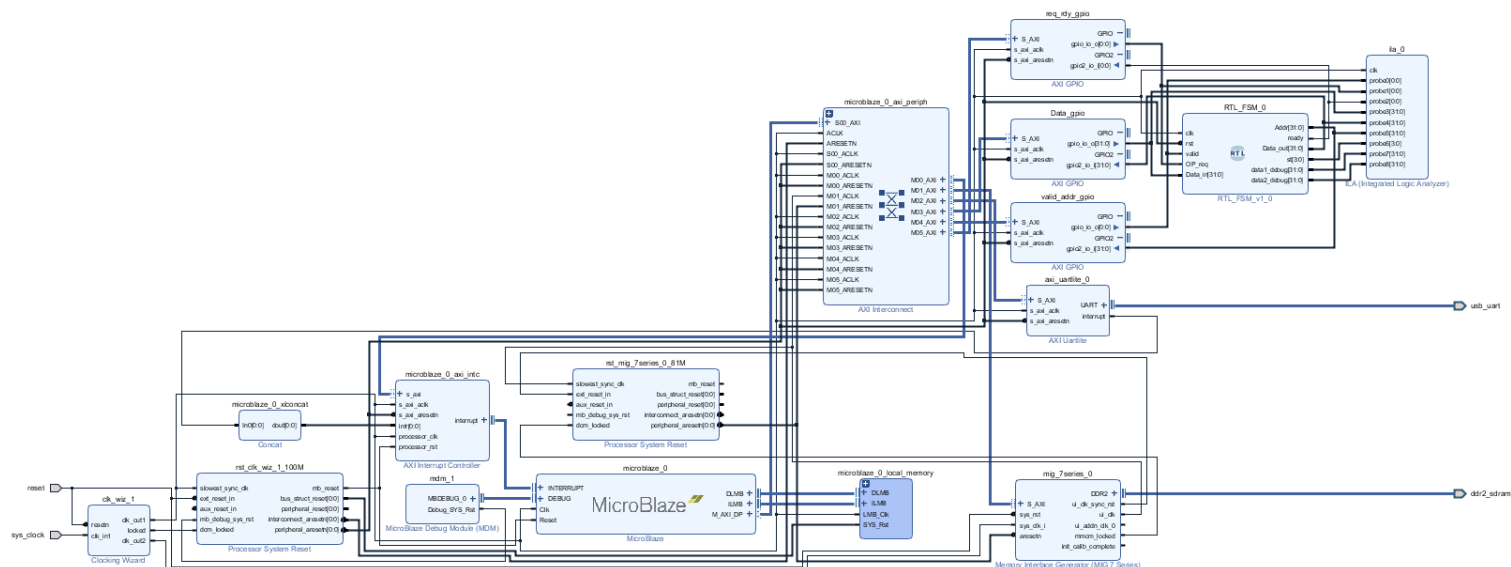


Figura 6.12 Layout con módulo de debugeo

- Una vez configurado el módulo de debugeo abrimos el *Hardware manager* y conectamos con la placa con la opción *Auto Connect*.

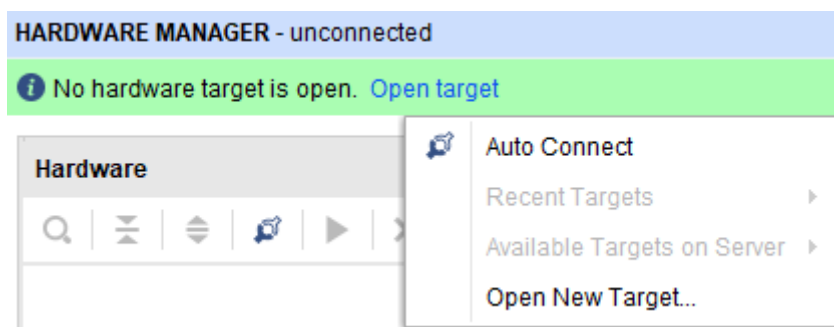


Figura 6.13 Auto Connect

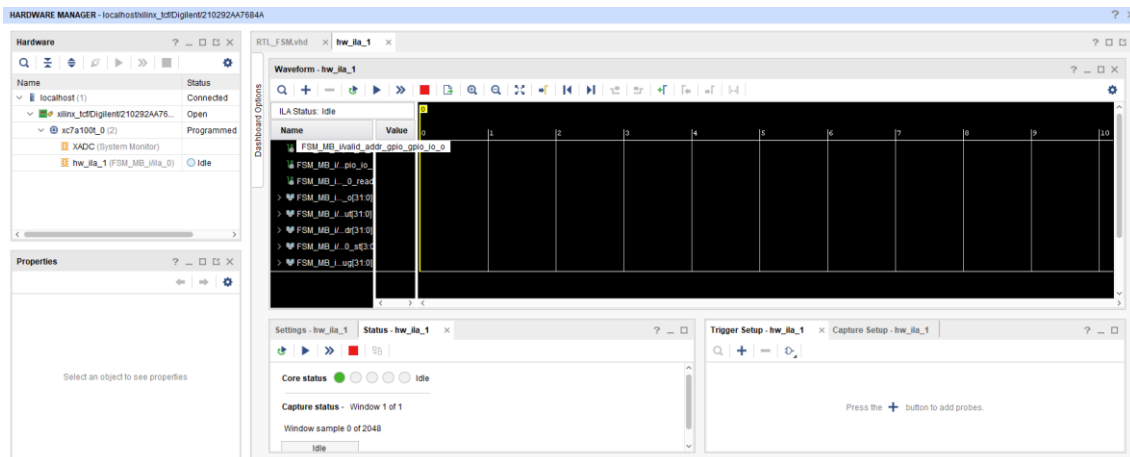


Figura 6.14 Placa conectada

12. Para poder ver los valores de las señales hay que añadir *Probes*. Que básicamente son todas las entradas que se han configurado previamente del módulo *ILA*. Para añadir *Probes* hay que pulsar la siguiente ventana que se encuentra en la esquina inferior derecha de la pantalla.

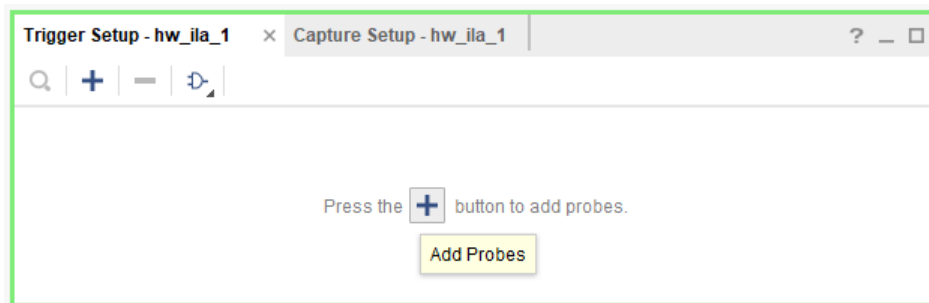


Figura 6.15 Add probes

Y seleccionamos todas las señales que deseamos ver por pantalla.

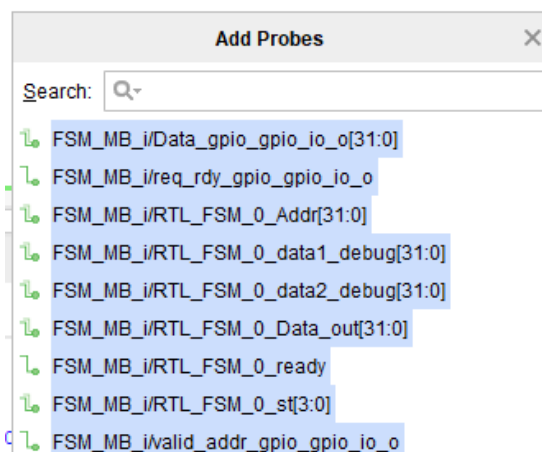


Figura 6.16 Probes seleccionadas

13. Una vez configurado el *Hardware Manager* comenzamos a debugear desde el *SDK*. Para ello debemos colocar inteligentemente *breakpoints* a lo largo del código. Es decir, hay que colocar los *breakpoints* en los puntos donde queramos ver los valores de las *Probes* anteriormente

configurados. Cuando el código se encuentre parado y queramos ver los valores de las señales en la FPGA:

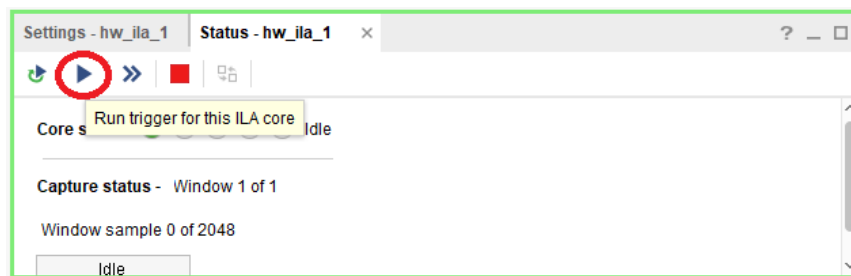


Figura 6.17 Run Trigger

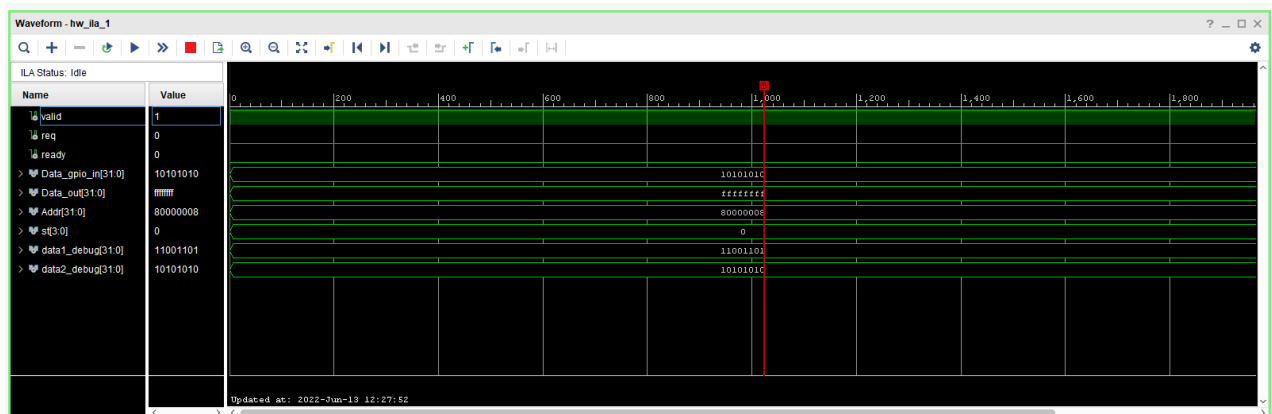


Figura 6.18 Valores de las señales en el Breakpoint

14. Una vez terminado el debugueo se obtendrán los siguientes resultados:

```
Serial Port Properties -----
Device ID : 0
Baud Rate : 9600
Data Bits : 8
Base Addr : 40600000

Addr : 80000000
Data : 11001101

Addr : 80000004
Data : 10101010

Data DDR2: 9999CCCC
Addr DDR2: 80000008

Options -----
a - Start DDR test
b - See ddr data values

Starting DDR test

Options -----
a - Start DDR test
b - See ddr data values

Addr 1 : 80000000
Data 1 : 11001101

Addr 2 : 80000004
Data 2 : 10101010

Data : 10001000
Data DDR2: 10001000
Addr DDR2: 80000008
```

Datos de las 3 primeras direcciones de memoria

Máquinas de estado funcionando

Resultado esperado

Resultado de las fsm (escrito en memoria)

Figura 6.19 Resultados en el terminal de la práctica 4

7. PRÁCTICA 5: DESARROLLO DE UNA INTERFAZ ENTRE EL ORDENADOR Y LA DDR2

7.1. INTRODUCCIÓN

En esta práctica se buscará rematar todos los conocimientos adquiridos en las prácticas anteriores, y se desarrollará una interfaz sencilla que permitirá una comunicación entre el ordenador y la placa. El objetivo principal de esta práctica es crear un sistema que permita enviar un archivo desde el ordenador a la memoria DDR2 de la placa Nexys mediante órdenes enviadas desde el terminal del ordenador.

En esta práctica se aplicará todo lo aprendido en las prácticas anteriores, como el uso de la interfaz UART para la comunicación entre el ordenador y la placa, el uso de bloques RTL personalizados para implementar funciones específicas, y la comunicación coordinada entre módulos mediante el uso de máquinas de estados. Además, se aprenderá a utilizar herramientas específicas para la transferencia de archivos y a realizar la configuración necesaria para que el sistema funcione correctamente.

En resumen, esta práctica es una excelente oportunidad para aplicar todos los conocimientos y habilidades adquiridos en las prácticas anteriores y mejorar habilidades en el campo de la comunicación entre dispositivos, el uso de bloques RTL personalizados y la transferencia de archivos. Es una oportunidad para poner en práctica todo lo aprendido y desarrollar un sistema completo y funcional.

7.2. OBJETIVOS DE LA PRÁCTICA

- I. Comunicar el ordenador con la memoria DDR2 de la placa
- II. Leer datos de la memoria en función de órdenes recibidas mediante los *Push Buttons* de la Nexys
- III. Escribir archivos provenientes del ordenador en la memoria DDR2

7.3. DESARROLLO

1. Realice los pasos del 1 al 7 de la práctica 1.
2. Añadiremos un módulo *UARTLite* y la memoria DDR2 de la placa.
3. Configure el módulo *UART* de la siguiente manera.

The screenshot shows the 'IP Configuration' window for a UART module. It has two tabs: 'Board' and 'IP Configuration'. The 'IP Configuration' tab is active. It contains the following settings:

- AXI CLK Frequency:** A text box with '100.0' and a range '[10-300]MHz'.
- Baud Rate:** A dropdown menu showing '115200'.
- Data Bits:** A text box with '8' and a range '[5 - 8]'.
- Parity:** A section with three radio buttons: 'No Parity' (selected), 'Odd', and 'Even'.

Figura 7.1 Configuración *UART* práctica 5

4. Añadiremos los *Push Buttons* de la placa.

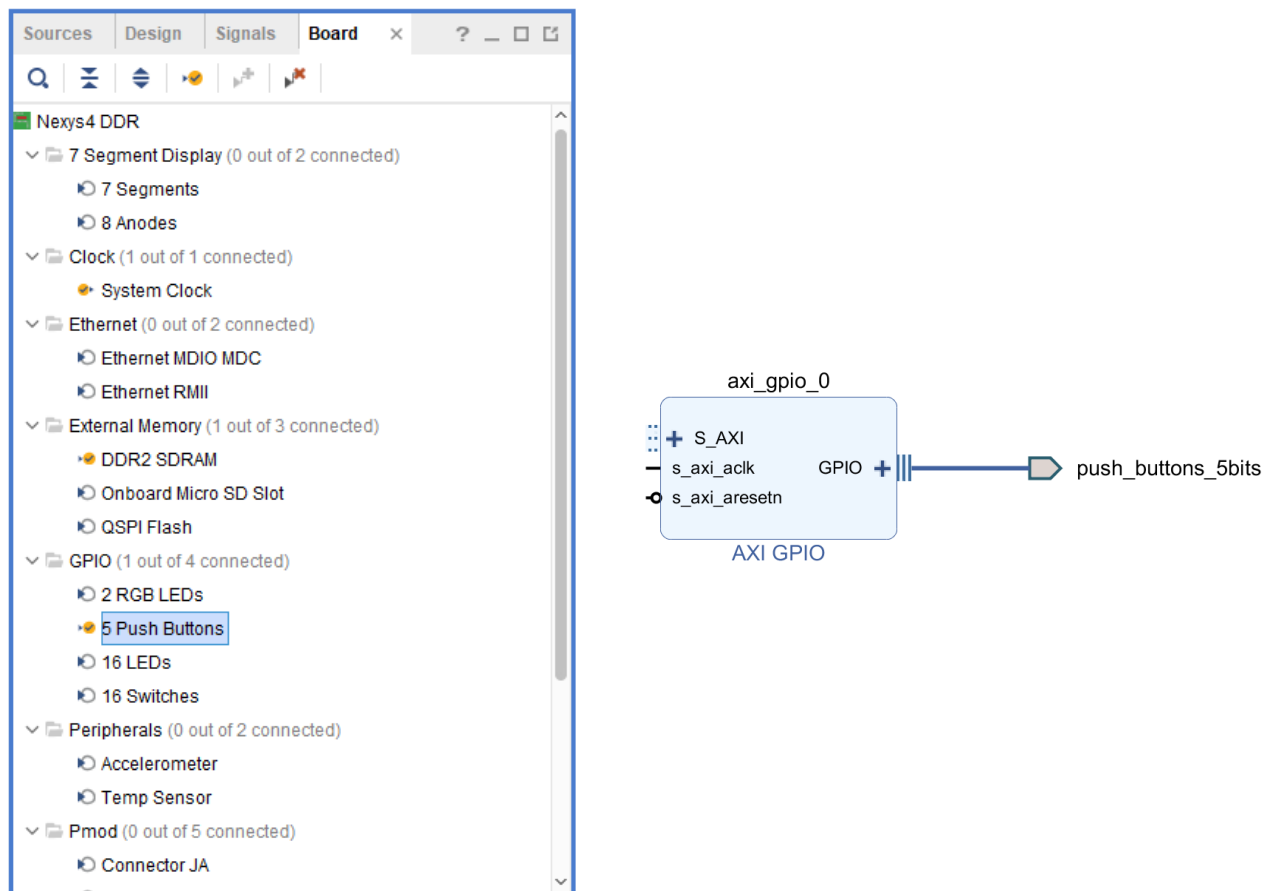


Figura 7.2 *Push Buttons*

Obteniendo el siguiente resultado:

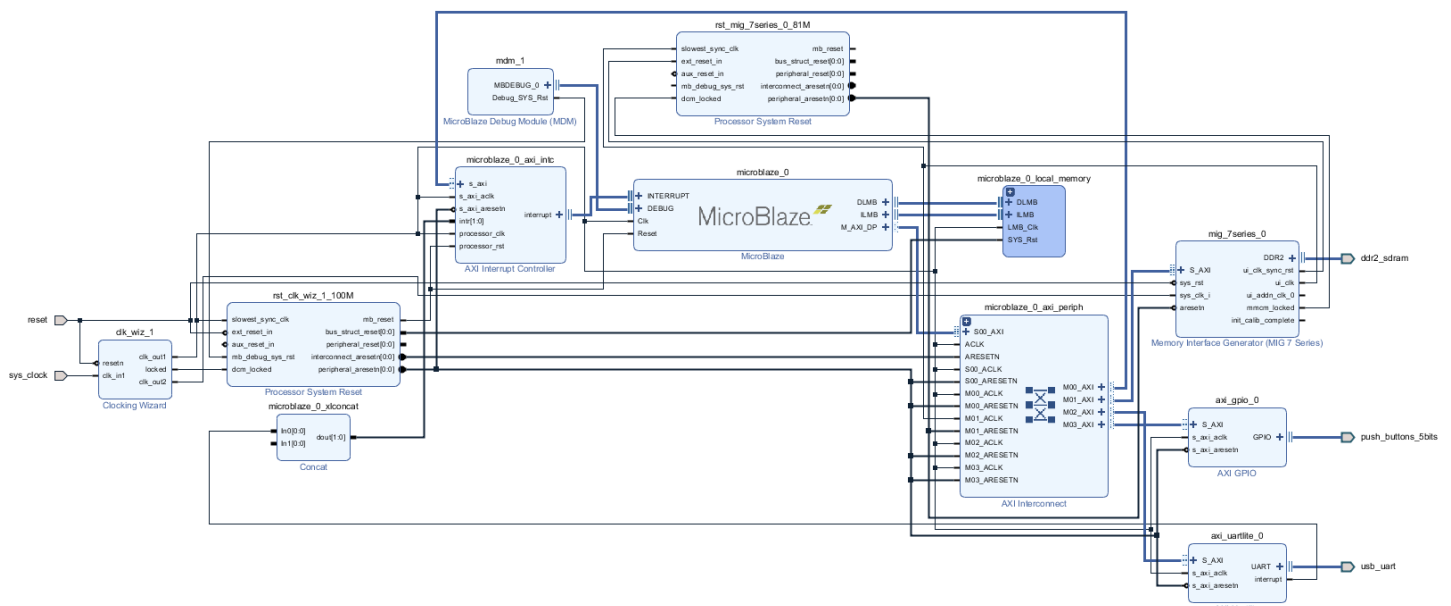


Figura 7.3 *Layout* práctica 5

5. Para poder diferenciar que botón ha sido pulsado hay que tener en cuenta la siguiente codificación de la placa:
- Ningún botón: 0x 00
 - BTNL: 0x 04
 - BTNU: 0x 02
 - BTNR: 0x 08
 - BTND: 0x 10
 - BTNC: 0x 01

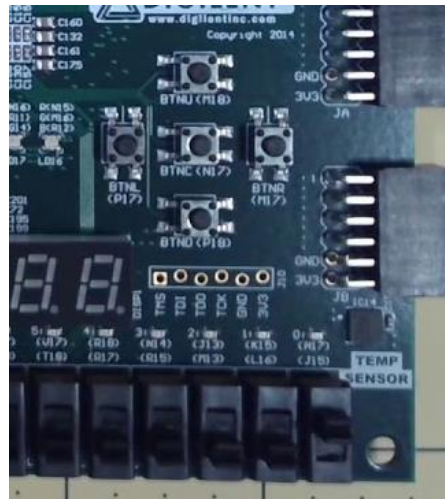


Figura 7.4 Botones Nexys a7

6. Para poder recibir datos por la *UART* hay que añadir el siguiente esqueleto a nuestro código:

```
int UartLitePolled(u16 DeviceId)
{
    int Status;
    unsigned int ReceivedCount = 0;
    int Index;

    /*
     * Initialize the UartLite driver so that it is ready to use.
     */
    Status = XUartLite_Initialize(&UartLite, DeviceId);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Initialize the the receive buffer bytes to zero.
     */
    for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
        RecvFIFO[Index] = 0;
    }

    /*
     * Receive the number of bytes which is transfered.
     * Data may be received in fifo with some delay hence we continuously
     * check the receive fifo for valid data and update the receive buffer
     * accordingly.
     */
}
```

```

    */

    while (1) {
        ReceivedCount += XUartLite_Rcv(&UartLite,
                                       RecvFIFO + ReceivedCount,
                                       TEST_BUFFER_SIZE - ReceivedCount);

        //codigo a rellenar por el alumno
        //...

    }

    return XST_SUCCESS;
}

```

7. En este punto hay que obtener un código que permita leer y escribir datos en memoria en función del botón que se pulse. El siguiente esqueleto de código se puede seguir como ejemplo para decodificar que botón de la placa ha sido pulsado:

```

while(1){

    xil_printf("\n\r");
    xil_printf("Options ----- \n\r");
    xil_printf("BTNC - See DDR2 data values\n\r");           //0x01
    xil_printf("BTNR - Start loading data into the DDR\n\r"); //0x08
    xil_printf("\n\r");

    //Check to make sure there is a new command to run
    while (button == 0){ //button = 0 when no button is pressed
        button = XGpio_DiscreteRead(&GPIO_buttons, 1);
        //Insertar código antirebotes
    };

    if (button == 0x01){
        //Codigo a rellenar por el alumno
    } else if (button == 0x08){
        //Send file to the DDR2
        xil_printf(" -> Select a file from to send to the FPGA\n\r");
        Status = UartLitePolled(UARTLITE_DEVICE_ID);
    } else {
        xil_printf("Not a valid command!\n\r");
    }

    //Increment to the next command
    button = 0;
}

//End of program
return XST_SUCCESS;
}

```

8. Una vez se haya obtenido un código funcional abrimos el *Tera Term* y configuramos la velocidad de la *UART* a 115200.

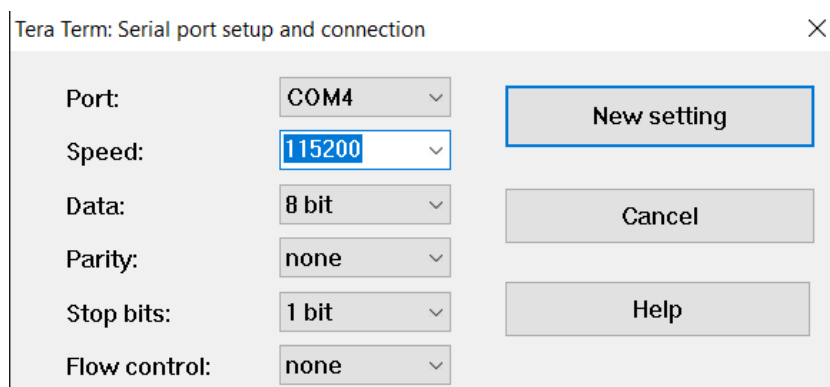


Figura 7.5 Baudrate práctica 5

9. Para poder enviar archivos a la placa hay que llamar a la función *UartLitePolled*, el cual espera a que una *FIFO*, de tamaño a escoger en función del archivo, se llene. Esta función de quedará esperando a que se comience a enviar un archivo, el cual seleccionaremos como en la siguiente figura:

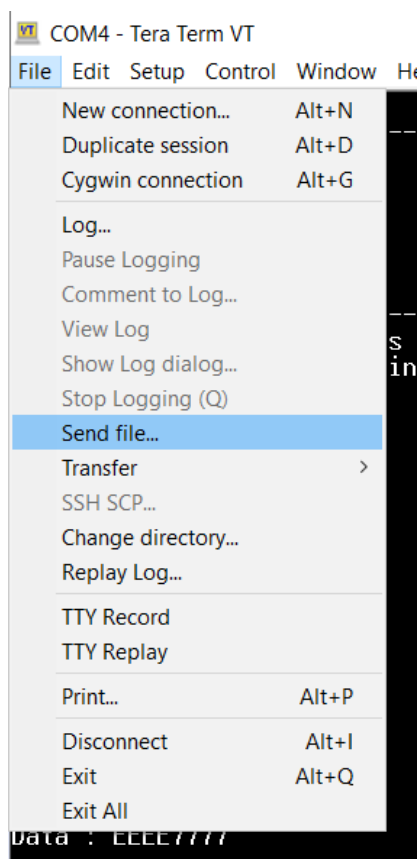


Figura 7.6 Send file

10. Finalmente se obtendrán los siguientes resultados:

```
Serial Port Properties -----
Device ID : 0
Baud Rate : 115200
Data Bits : 8
Base Addr : 40600000

Options -----
BTNC - See DDR2 data values
BTNR - Start loading data into the DDR

Addr : 80000000
Data : 33339999

Addr : 80000004
Data : EEEE7777

Addr : 80000008
Data : 9999CCCC

Addr : 8000000C
Data : 44442222

Addr : 80000010
Data : 1111BBBB

Addr : 80000014
Data : EEEE7777

Addr : 80000018
Data : 9999CCCC

Addr : 8000001C
Data : 44442222

Addr : 80000020
Data : 1111BBBB

Addr : 80000024
Data : EEEE7777

Options -----
BTNC - See DDR2 data values
BTNR - Start loading data into the DDR

-> Select a file from to send to the FPGA

16384 Bytes received
DATA TRANSFER IS COMPLETED!!
```

**Botón BTNC pulsado,
Valores por defecto en la
memoria**

**Botón BTNR,
Se ha enviado un archivo
a la memoria**

Figura 7.7 Resultados práctica 5 pt1

```
Options -----
BTNC - See DDR2 data values
BTNR - Start loading data into the DDR

Addr : 80000000
Data : 4D495250

Addr : 80000004
Data : 20415245

Addr : 80000008
Data : 54524150

Addr : 8000000C
Data : 41430045

Addr : 80000010
Data : 5480C350

Addr : 80000014
Data : 204F4C55

Addr : 80000018
Data : 51203A31

Addr : 8000001C
Data : 74206575

Addr : 80000020
Data : 61746172

Addr : 80000024
Data : 20656420
```

**Valores que se han
escrito en las primeras
direcciones tras recibir el
archivo**

Figura 7.8 Resultados práctica 5 pt2

8. CONCLUSIONES Y LÍNEAS FUTURAS

8.1. CONCLUSIONES

En conclusión, este proyecto se enfoca en abordar una carencia formativa en el ámbito de los sistemas *System on Chip* (SoC) mediante la realización de prácticas de laboratorio introductorias. El objetivo principal es proporcionar una base sólida y una introducción valiosa para el diseño de estos sistemas, los cuales son cada vez más necesarios y utilizados en ambientes profesionales.

Una vez completadas las prácticas, se puede comprender la razón de la creciente popularidad de los sistemas basados en FPGA y SoC. Como se ha podido observar, estos sistemas ofrecen una gran flexibilidad, ya que se pueden configurar para realizar una amplia variedad de funciones en base a las necesidades del proyecto. Además, se destaca la eficiencia de diseño y capacidad de procesamiento que se obtiene al combinar estos dos dispositivos en una sola plataforma.

A través de las prácticas, se abordan diferentes conceptos y técnicas relacionadas con el diseño de sistemas basados en FPGA y SoC, incluyendo el uso de bloques descritos en lenguaje de descripción de hardware y debugging y depuración. Al finalizar las prácticas, se habrán adquirido conceptos y técnicas importantes para el diseño de este tipo de sistemas, tales como el uso de la UART para la comunicación serie, el uso de la memoria DDR para almacenar datos, el diseño de interfaces PC-FPGA y el diseño de bloques con el uso del procesador virtual Microblaze.

Sin embargo, es importante tener en cuenta que el diseño de sistemas de comunicación, seguridad, alimentación y diseño de sistemas con requerimientos de tiempo real son áreas importantes y comunes en proyectos de sistemas SoC y FPGA, pero no se encuentran cubiertos en las prácticas diseñadas en este proyecto. Estos temas podrían ser una posible línea de trabajo futura para seguir profundizando en el diseño de sistemas SoC, ya que son aspectos clave en la mayoría de los proyectos que requieren de estas tecnologías.

En resumen, este proyecto proporciona una introducción valiosa al diseño de sistemas SoC y FPGA, y proporciona una base sólida para el desarrollo de futuros proyectos en esta área. No obstante, es importante seguir profundizando en el aprendizaje y adquirir conocimientos en áreas y aplicaciones específicas para poder aprovechar al máximo las ventajas de estas tecnologías en proyectos profesionales.

8.2.LÍNEAS FUTURAS

El conocimiento adquirido a través de este proyecto es altamente valioso ya que puede ser aplicado en una variedad de proyectos futuros relacionados con el uso de FPGA y SoC, así como con las metodologías de diseño asociadas. Estos proyectos pueden ser en el campo de la automatización industrial, la robótica, el control de motores y la automatización de procesos, así como en campos como el procesamiento de señales, el procesamiento de imágenes y el procesamiento de datos en tiempo real. Estos campos requieren altas velocidades de procesamiento y una alta flexibilidad en el diseño.

Algunos ejemplos de proyectos en los que se puede aplicar el conocimiento adquirido incluyen:

- Diseño de sistemas de comunicación: Se podría desarrollar un proyecto que involucre el diseño de un sistema de comunicación basado en la placa Nexys A7, como una red de bus para la comunicación entre varios dispositivos conectados a la placa. También se podrían implementar protocolos de comunicación como I2C, SPI o Ethernet para mejorar la velocidad y la fiabilidad de la comunicación, así como mecanismos de detección y corrección de errores para garantizar la integridad de los datos transmitidos.
- Depuración y debugging: Se podría desarrollar un proyecto que involucre la depuración y solución de problemas en el diseño de sistemas basados en FPGA. Se podrían utilizar herramientas de depuración y análisis de señales en tiempo real para identificar y solucionar problemas en el diseño, como problemas de sincronización, problemas de rendimiento, etc. Además, se podrían implementar mecanismos de diagnóstico y monitoreo en tiempo real para detectar problemas en el sistema y facilitar su solución.
- Diseño de sistemas con requerimientos de tiempo real: Se podría desarrollar un proyecto que involucre el diseño de sistemas con requerimientos de tiempo real utilizando la placa Nexys A7, como un sistema de control de movimiento para un robot. Es esencial cumplir con requerimientos de tiempo real para garantizar un movimiento preciso y suave, por lo que se podrían implementar mecanismos de sincronización de relojes para garantizar la precisión temporal en el sistema, y utilizar herramientas de depuración para medir y mejorar el rendimiento del sistema en tiempo real.

Además, el conocimiento adquirido en este proyecto también podría ser aplicado en campos como la inteligencia artificial, donde se requiere un procesamiento de datos rápido y eficiente, y en proyectos relacionados con el internet de las cosas, donde se requiere una comunicación confiable y una capacidad de procesamiento en tiempo real.

Resumiendo, el conocimiento adquirido en este proyecto es altamente valioso y versátil, ya que se puede aplicar en una variedad de campos y proyectos futuros, desde la automatización industrial hasta el procesamiento de datos en tiempo real y la inteligencia artificial.

9. BIBLIOGRAFÍA

- [1] Digilent, «digilent.com,» [En línea]. Available: <https://digilent.com/reference/programmable-logic/nexys-a7/start>.
- [2] Xilinx, «xilinx.com,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/microblaze.html>. [Último acceso: 2022].
- [3] Xilinx, «xilinx.com,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vivado.html>. [Último acceso: 2022].
- [4] Xilinx, «xilinx.com,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/legacy-tools/sdk.html>. [Último acceso: 2022].
- [5] Wikipedia, «Wikipedia,» [En línea]. Available: [https://es.wikipedia.org/wiki/Eclipse_\(software\)](https://es.wikipedia.org/wiki/Eclipse_(software)). [Último acceso: 2022].
- [6] Softonic, «softonic,» [En línea]. Available: <https://tera-term.softonic.com/>. [Último acceso: 2022].
- [7] Sourceforge, «sourceforge.net,» [En línea]. Available: <https://sourceforge.net/projects/wxhexeditor/>. [Último acceso: 2022].
- [8] Xilinx, «xilinx.com,» [En línea]. Available: https://www.xilinx.com/products/intellectual-property/opb_mdm.html. [Último acceso: 2022].
- [9] Xilinx, Diciembre 5 2018. [En línea]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/mig_7series/v4_2/ug586_7Series_MIS.pdf.
- [10] P. P. Chu, FPGA prototyping by VHDL examples, Wiley, 2007.
- [11] «GitHub,» [En línea]. Available: <https://github.com/chasep255/Nexys-4-DDR-Ethernet-Mac>. [Último acceso: 2022].
- [12] «GitHub,» [En línea]. Available: <https://github.com/Digilent/Nexys-A7-100T-DMA-Audio>. [Último acceso: 2022].
- [13] «Digilent Forums,» [En línea]. Available: <https://forum.digilent.com/topic/2688-accessing-ddr2-on-nexys-4-ddr/>. [Último acceso: 2022].
- [14] «dwjbosman.github.io,» [En línea]. Available: <https://dwjbosman.github.io/nexys-4-ddr-microblaze-with-ddr-ram-and-flash-bootloader-support>. [Último acceso: 2022].
- [15] «docs.xilinx.com,» [En línea]. Available: <https://docs.xilinx.com/v/u/en-US/xapp1026>. [Último acceso: 2022].
- [16] A. Xilinx, «YouTube,» [En línea]. Available: <https://www.youtube.com/watch?v=bsvpJYCDmCQ>. [Último acceso: 2022].
- [17] «lancesimms.com,» [En línea]. Available: https://lancesimms.com/Xilinx/VC707_Microblaze_UART_to_LED_Example_Part4.html. [Último acceso: 2022].
- [18] W. Knitter, «hackster.io,» [En línea]. Available: <https://www.hackster.io/whitney-knitter/add-custom-ip-modules-to-vivado-block-design-77042d#:~:text=settings%20to%20it,->

,Add%20the%20RTL%20Module%20to%20the%20Block%20Design,option.. [Último acceso: 2022].

- [19] BOPV, «YouTube,» [En línea]. Available: <https://www.youtube.com/watch?v=MbteffkRi8Y>. [Último acceso: 2022].
- [20] K. Eissa, Modeling of a Multi-core MicroBlaze System ay RTL and TLM Abstraction leves in SystemC, 2013.
- [21] G. I. y. B. E. Aguayo E, Tutorial Xilinx MicroBlaze, Madrid: UAM, 2005.
- [22] F. M. V. J. S. Rod Jesman, MicroBlaze Tutorial: Creating a simple embedded system and adding custom peripherals using Xilinx EDK software tools, ECASP.
- [23] Xilinx, «GitHub.com,» [En línea]. Available: https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/uartlite/examples/xuartlite_polled_example.c. [Último acceso: 2022].
- [24] Xilinx, «Xilinx.com,» [En línea]. Available: <https://www.xilinx.com/video/hardware/logic-debug-in-vivado.html>. [Último acceso: 2022].
- [25] «support.xilinx.com,» [En línea]. Available: https://support.xilinx.com/s/question/0D52E00006hpSinSAE/how-to-read-the-data-via-uartlite?language=en_US. [Último acceso: 2022].
- [26] Xilinx, «xilinx.github.io,» [En línea]. Available: https://xilinx.github.io/embeddedsw.github.io/uartlite/doc/html/api/group__uartlite__v3__7.html#gaa30ed8bfedba1acdc67dc94e77d5a41d. [Último acceso: 2022].
- [27] Digilent, «GitHub,» [En línea]. Available: https://github.com/Digilent/Nexys-A7?_ga=2.94331549.1206465457.1655118148-1177350316.1607859484. [Último acceso: 2022].
- [28] Digilent, «GitHub,» [En línea]. Available: <https://github.com/Digilent/vivado-library>. [Último acceso: 2022].
- [29] MathWorks, «Mathlab,» [En línea]. Available: <https://www.mathworks.com/products/matlab.html>. [Último acceso: 2022].
- [30] «Digilent,» [En línea]. Available: <https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-getting-started-with-microblaze-servers/start>. [Último acceso: 2022].

ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

A.1 INTRODUCCIÓN

Este proyecto busca abordar una carencia formativa en el uso de SoC y metodologías de diseño asociadas, con el objetivo de desarrollar una interfaz para escribir archivos en la memoria de la placa Nexys A7 de manera rápida y eficiente. El objetivo es obtener los conocimientos necesarios para abordar proyectos similares que requieran interfaces PC-FPGA y requieran de altas velocidades de procesamiento y una alta flexibilidad en el diseño, aspectos que cubren los SoC.

A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Un proyecto que se desarrolle con los conocimientos adquiridos con estas prácticas tiene impactos económico, social, ético, tecnológico y ambiental. El impacto económico se refiere a los costos y beneficios del proyecto, mientras que el impacto social se refiere a cómo el proyecto afecta a la sociedad. El impacto ético se refiere a cómo el proyecto afecta a los valores morales, el impacto tecnológico se refiere a cómo el proyecto afecta a la innovación y el impacto ambiental se refiere a cómo el proyecto afecta al medio ambiente. En general, el proyecto tiene un impacto económico positivo, puede tener un impacto social positivo o negativo dependiendo de cómo se utilice y tiene un impacto ético positivo al hacer el conocimiento más accesible, tiene un impacto tecnológico positivo al mejorar la capacidad de procesamiento y la flexibilidad en el diseño, y un impacto ambiental positivo al permitir una mayor eficiencia en el uso de los recursos.

A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

El **impacto económico** es un factor clave en este proyecto. El desarrollo de un sistema SoC es un proceso complejo que requiere un gran esfuerzo en términos de diseño, programación y pruebas. Además, los conocimientos avanzados necesarios para llevar a cabo este tipo de proyectos pueden representar un desafío adicional en términos de tiempo y, por lo tanto, coste del proyecto. Sin embargo, también pueden ofrecer un gran retorno de inversión en términos de mejora de la eficiencia y productividad.

El **impacto social** de este proyecto es difícil de determinar con precisión. Los proyectos que se pueden llevar a cabo con los conocimientos adquiridos son variados y pueden tener un impacto social directo o indirecto, ya sea a través de la creación de empleo o mejora de la calidad de vida. Por ejemplo, un sistema de automatización industrial puede mejorar la seguridad en el trabajo y un sistema de control de motores puede mejorar la eficiencia energética.

En cuanto al **impacto ético**, este proyecto busca proporcionar una base sólida y valiosa de conocimientos para aquellos interesados en el mundo de los SoC, con el objetivo de reducir la barrera de entrada y hacerlo más accesible. Por lo tanto, estas prácticas son de gran utilidad tanto para principiantes como para aquellos con experiencia en el campo de las FPGA pero no en el campo de los SoC.

Por otra parte, el **impacto tecnológico**: Los proyectos que utilizan SoC pueden mejorar la capacidad de procesamiento y la flexibilidad en el diseño, lo que permite innovar en una variedad de campos, como

la robótica, el procesamiento de señales, el procesamiento de imágenes y el procesamiento de datos en tiempo real.

Por último, es importante mencionar el **impacto ambiental** del proyecto. El uso de un SoC permite una mayor flexibilidad en el diseño de sistemas, lo que significa que se pueden adaptar las características del proyecto según las necesidades. Esto significa que si se decide cambiar alguna característica del proyecto, se podrá rediseñar y reutilizar la plataforma, lo que permite ahorrar recursos significativamente. Además estos proyectos pueden tener un impacto ambiental positivo en más situaciones, ya que la automatización y la eficiencia energética pueden reducir el consumo de recursos y la huella de carbono.

A.4 CONCLUSIONES

En conclusión, el proyecto tiene un impacto económico positivo, puede tener un impacto social positivo o negativo dependiendo de cómo se utilice, tiene un impacto ético positivo al hacer el conocimiento más accesible, tiene un impacto tecnológico positivo al mejorar la capacidad de procesamiento y la flexibilidad en el diseño, y un impacto ambiental positivo al permitir una mayor eficiencia en el uso de los recursos.

ANEXO B: PRESUPUESTO ECONÓMICO

COSTE DE MANO DE OBRA (coste directo)

Horas	Precio/hora	Total
470	30 €	14,100 €

COSTE DE RECURSOS MATERIALES (coste directo)

	Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador personal (Software incluido).....	1.000,00 €	12	5	200,00 €
Nexys A7	240,00 €	10	5	40,00 €

COSTE TOTAL DE RECURSOS MATERIALES

240,00 €

GASTOS GENERALES (costes indirectos)

15%

sobre CD

2.151,00 €

BENEFICIO INDUSTRIAL

6%

sobre CD+CI

989,46 €

SUBTOTAL PRESUPUESTO

17480,46 €

IVA APLICABLE

21%

3.670,90 €

TOTAL PRESUPUESTO

21.151,36 €

ANEXO C: ESQUELETO DE EJEMPLO DE CÓDIGO PARA EL MANEJO DE LA UART CON COMANDOS

```
#include "xparameters.h"
#include <xil_printf.h>
#include <stdio.h>
#include "xil_exception.h"
#include "xuartlite.h"
#include "xintc.h"
#include "xgpio_1.h"

/***** Constant Definitions *****/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define UARTLITE_DEVICE_ID      XPAR_UARTLITE_0_DEVICE_ID
#define INTC_DEVICE_ID          XPAR_INTC_0_DEVICE_ID
#define UARTLITE_INT_IRQ_ID     XPAR_INTC_0_UARTLITE_0_VEC_ID
#define GPIO_REG_BASEADDR      ? // Direccion GPIO del LED

/*
 * The following constant controls the length of the buffers to be sent
 * and received with the UartLite device.
 */
#define TEST_BUFFER_SIZE       500
#define LED_CHANNEL             ? // Chanel en el IP del GPIO del LED

/***** Function Prototypes *****/

int SetupUartLite(u16 DeviceId);

int SetupInterruptSystem(XUartLite *UartLitePtr);

void SendHandler(void *CallBackRef, unsigned int EventData);

void RecvHandler(void *CallBackRef, unsigned int EventData);

static int GpioOutputExample(u32 LED_Value);

/***** Variable Definitions *****/

XUartLite UartLite;           /* The instance of the UartLite Device */
XUartLite_Config *UartLite_Cfg; /* The instance of the UartLite Config */
XIntc InterruptController;    /* The instance of the Interrupt Controller */

/*
 * The following buffers are used in this example to send and receive data
 * with the UartLite.
 */
u8 SendBuffer[TEST_BUFFER_SIZE];
u8 ReceiveBuffer[TEST_BUFFER_SIZE];

/* Here are the pointers to the buffer */
u8* ReceiveBufferPtr = &ReceiveBuffer[0];
```

```

u8* CommandPtr      = &ReceiveBuffer[0];

/*
 * The following counters are used to determine when the entire buffer has
 * been sent and received.
 */
static volatile int TotalReceivedCount;
static volatile int TotalSentCount;

int main()
{
    //Variable definitions
    int Status=0;
    long i=0;

    //Set up the UART and configure the interrupt handler for bytes in RX buffer
    Status = SetupUartLite(UARTLITE_DEVICE_ID);

    //Get a reference pointer to the Uart Configuration
    UartLite_Cfg = XUartLite_LookupConfig(UARTLITE_DEVICE_ID);

    //Print out the info about our XUartLite instance
    xil_printf("\n\r");
    xil_printf("Serial Port Properties ----- \n\r");
    xil_printf("Device ID : %d\n\r", UartLite_Cfg->DeviceId);
    xil_printf("Baud Rate : %d\n\r", UartLite_Cfg->BaudRate);
    xil_printf("Data Bits : %d\n\r", UartLite_Cfg->DataBits);
    xil_printf("Base Addr : %08X\n\r", UartLite_Cfg->RegBaseAddr);
    xil_printf("\n\r");

    while(i==0){

        //Print out the current contents of the buffer
        xil_printf("\n\r");
        xil_printf("Options ----- \n\r");
        xil_printf("a - Change the LED\n\r");
        xil_printf("b - Press a key to see its ASCII value show up on the LEDs\n\r");
        xil_printf("\n\r");

        //Check to make sure there is a new command to run
        while (ReceiveBufferPtr <= CommandPtr){};

        if (*CommandPtr == 'a'){
            /*
             * CODIGO A RELLENAR POR EL ALUMNO
             */
        } else if (*CommandPtr == 'b'){
            /*
             * CODIGO A RELLENAR POR EL ALUMNO
             * SE RECOMIENDA USAR GpioOutputExample()
             * se encuentra al final del código
             */
        }
        else {
            xil_printf("Not a valid command!\n\r");
        }
    }
}

```

```

        //Increment to the next command
        CommandPtr++;

    }

    //End of program
    return Status;
}

/*****
/**
 *
 * This function does a minimal test on the UartLite device and driver as a
 * design example. The purpose of this function is to illustrate
 * how to use the XUartLite component.
 *
 * This function sends data and expects to receive the same data through the
 * UartLite. The user must provide a physical loopback such that data which is
 * transmitted will be received.
 *
 * This function uses interrupt driver mode of the UartLite device. The calls
 * to the UartLite driver in the handlers should only use the non-blocking
 * calls.
 *
 * @param      DeviceId is the Device ID of the UartLite Device and is the
 *              XPAR_<uartlite_instance>_DEVICE_ID value from xparameters.h.
 *
 * @return     XST_SUCCESS if successful, otherwise XST_FAILURE.
 *
 * @note
 *
 * This function contains an infinite loop such that if interrupts are not
 * working it may never return.
 *
 *****/
int SetupUartLite(u16 DeviceId)
{
    int Status;
    int Index;

    /*
     * Initialize the UartLite driver so that it's ready to use.
     */
    Status = XUartLite_Initialize(&UartLite, DeviceId);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Perform a self-test to ensure that the hardware was built correctly.
     */
    Status = XUartLite_SelfTest(&UartLite);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Connect the UartLite to the interrupt subsystem such that interrupts can
     * occur. This function is application specific.

```

```

    */
    Status = SetupInterruptSystem(&UartLite);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Setup the handlers for the UartLite that will be called from the
     * interrupt context when data has been sent and received, specify a
     * pointer to the UartLite driver instance as the callback reference so
     * that the handlers are able to access the instance data.
     */
    XUartLite_SetSendHandler(&UartLite, SendHandler, &UartLite);
    XUartLite_SetRecvHandler(&UartLite, RecvHandler, &UartLite);

    /*
     * Enable the interrupt of the UartLite so that interrupts will occur.
     */
    XUartLite_EnableInterrupt(&UartLite);

    /*
     * Initialize the send buffer bytes with a pattern to send and the
     * the receive buffer bytes to zero to allow the receive data to be
     * verified.
     */
    for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
        SendBuffer[Index] = 1;
        ReceiveBuffer[Index] = 0;
    }

    return XST_SUCCESS;
}

/*****
**
*
* This function is the handler which performs processing to send data to the
* UartLite. It is called from an interrupt context such that the amount of
* processing performed should be minimized. It is called when the transmit
* FIFO of the UartLite is empty and more data can be sent through the UartLite.
*
* This handler provides an example of how to handle data for the UartLite,
* but is application specific.
*
* @param      CallbackRef contains a callback reference from the driver.
*              In this case it is the instance pointer for the UartLite driver.
* @param      EventData contains the number of bytes sent or received for sent
*              and receive events.
*
* @return      None.
*
* @note        None.
*
*****/
void SendHandler(void *CallbackRef, unsigned int EventData)
{
    TotalSentCount = EventData;
}

```

```

/*****/
/**
 *
 * This function is the handler which performs processing to receive data from
 * the UartLite. It is called from an interrupt context such that the amount of
 * processing performed should be minimized. It is called data is present in
 * the receive FIFO of the UartLite such that the data can be retrieved from
 * the UartLite. The size of the data present in the FIFO is not known when
 * this function is called.
 *
 * This handler provides an example of how to handle data for the UartLite,
 * but is application specific.
 *
 * @param CallbackRef contains a callback reference from the driver, in
 * this case it is the instance pointer for the UartLite driver.
 * @param EventData contains the number of bytes sent or received for sent
 * and receive events.
 *
 * @return None.
 *
 * @note None.
 */
*****/
void RecvHandler(void *CallbackRef, unsigned int EventData)
{
    XUartLite_Rcv(&UartLite, ReceiveBufferPtr, 1);
    ReceiveBufferPtr++;
    TotalReceivedCount++;

    //If we've reached the end of the buffer, start over
    if (ReceiveBufferPtr >= (&ReceiveBuffer[0] + TEST_BUFFER_SIZE)){
        xil_printf("Resetting Receive Buffer. Please enter a new command!\n\r");
        ReceiveBufferPtr = &ReceiveBuffer[0];
        CommandPtr = &ReceiveBuffer[0];
        TotalReceivedCount = 0;
    }
}

/*****/
/**
 *
 * This function setups the interrupt system such that interrupts can occur
 * for the UartLite device. This function is application specific since the
 * actual system may or may not have an interrupt controller. The UartLite
 * could be directly connected to a processor without an interrupt controller.
 * The user should modify this function to fit the application.
 *
 * @param UartLitePtr contains a pointer to the instance of the UartLite
 * component which is going to be connected to the interrupt
 * controller.
 *
 * @return XST_SUCCESS if successful, otherwise XST_FAILURE.
 *
 * @note None.
 */
*****/
int SetupInterruptSystem(XUartLite *UartLitePtr)
{

```



```

    int Status;

    /*
     * Initialize the interrupt controller driver so that it is ready to
     * use.
     */
    Status = XIntc_Initialize(&InterruptController, INTC_DEVICE_ID);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Connect a device driver handler that will be called when an interrupt
     * for the device occurs, the device driver handler performs the
     * specific interrupt processing for the device.
     */
    Status = XIntc_Connect(&InterruptController, UARTLITE_INT_IRQ_ID,
                          (XInterruptHandler)XUartLite_InterruptHandler,
                          (void *)UartLitePtr);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Start the interrupt controller such that interrupts are enabled for
     * all devices that cause interrupts, specific real mode so that
     * the UartLite can cause interrupts through the interrupt controller.
     */
    Status = XIntc_Start(&InterruptController, XIN_REAL_MODE);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    /*
     * Enable the interrupt for the UartLite device.
     */
    XIntc_Enable(&InterruptController, UARTLITE_INT_IRQ_ID);

    /*
     * Initialize the exception table.
     */
    Xil_ExceptionInit();

    /*
     * Register the interrupt controller handler with the exception table.
     */
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                                (Xil_ExceptionHandler)XIntc_InterruptHandler,
                                &InterruptController);

    /*
     * Enable exceptions.
     */
    Xil_ExceptionEnable();

    return XST_SUCCESS;
}

```

```

/*****
/**
 *
 * This function does a minimal test on the GPIO device configured as OUTPUT.
 *
 * @param      None.
 *
 * @return      - XST_SUCCESS if the example has completed successfully.
 *               - XST_FAILURE if the example has failed.
 *
 * @note       None.
 *****/
static int GpioOutputExample(u32 LED_Value)
{
    //Change the value of the LED accordingly
    if (LED_Value == 0xFF){
        xil_printf("GPIO LED is off, turning on!\n\r");
    } else if (LED_Value == 0x00){
        xil_printf("GPIO LED is on, turning off!\n\r");
    } else {
        xil_printf("Programming LED with ASCII representation of : %c\n\r",
LED_Value);
    }

    // Set the LED to the requested state
    XGpio_WriteReg((GPIO_REG_BASEADDR),
        ((LED_CHANNEL - 1) * XGPIO_CHAN_OFFSET) +
        XGPIO_DATA_OFFSET, LED_Value);

    // Return
    return XST_SUCCESS;
}

```