



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

**DISEÑO E IMPLEMENTACIÓN DE SISTEMAS DE
CÓMPUTO HETEROGÉNEO BASADOS EN
PROCESADOR RISC-V Y COPROCESADORES EN
LÓGICA RECONFIGURABLE.**

VICENTE RAMÍREZ PÉREZ

**MEMORIA DE TITULACIÓN PARA OPTAR AL
TÍTULO DE INGENIERO CIVIL ELECTRÓNICO**

PROFESOR SUPERVISOR : GONZALO CARVAJAL
PROFESOR CORREFERENTE : ALEJANDRO ESTAY
PROFESOR CORREFERENTE : IOANNIS VOURKAS

4 DE AGOSTO DE 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo:

Diseño e implementación de sistemas de cómputo heterogéneo basados en procesador RISC-V y coprocesadores en lógica reconfigurable

Nombre del candidato(a): Vicente Cristóbal Ramírez Perez

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central Departamento: Departamento de Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Carvajal, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente

DEJO CONSTANCIA que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (marcar una opción):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 02/09/2026

Firma: _____

Estudiante o Candidato(a):

Fecha: 02/09/2026

Firma: _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

AGRADECIMIENTOS

Para cerrar lo que siento ha sido el primer capítulo de mi vida como adulto, no hay nadie a quien quiera agradecer más que a mis padres. Ha sido gracias a su apoyo incondicional que he podido llegar hasta este punto. Espero que algún día pueda devolver, aunque sea en una pequeña parte, todo el cariño, la dedicación y la compañía que me han brindado a lo largo de este camino.

Vicente Cristóbal Ramírez Pérez

El desarrollo de este trabajo contó con el apoyo de los siguientes proyectos ANID:

- FONDECYT 1261452
- AC3E CIA250006

ÍNDICE

AGRADECIMIENTOS	I
ÍNDICE DE FIGURAS	V
1. INTRODUCCIÓN	1
1.1. Motivación y contexto	1
1.2. Planteamiento del problema	2
1.2.1. Objetivos	3
1.3. Alcances y contribuciones	4
1.4. Organización del informe	5
2. ANTECEDENTES	7
2.1. Lógica reconfigurable y cómputo heterogéneo	7
2.1.1. Field-Programmable Gate Arrays	7
2.1.2. Procesadores integrados	8
2.1.3. Buses de interconexión	9
2.1.4. Desafíos técnicos en sistemas heterogéneos sobre FPGA	10
2.2. Arquitectura RISC-V y MicroBlaze V	10
2.2.1. Arquitectura RISC-V	10
2.2.2. MicroBlaze V	11
2.3. High-Level Synthesis para diseño de hardware especializado	12
2.3.1. Herramientas de diseño: Vivado y Vitis	13
2.4. Advanced eXtensible Interface	14
2.5. Uso de memoria en sistemas embebidos basados en FPGA	16
2.5.1. Mapa de memoria	16
2.6. Nociones de desarrollo de firmware	17
2.6.1. Estructura del ejecutable ELF	17
2.6.2. Restricciones de memoria	19
3. METODOLOGÍA Y DESARROLLO	20
	III

3.1. Análisis de material existente	20
3.2. Conceptos a abarcar	22
3.3. Estructura de Guías	23
3.4. Definición preliminar de las guías	24
3.5. Validación	25
3.5.1. Primera etapa de validación	25
3.5.2. Segunda etapa de validación	26
4. MATERIAL DESARROLLADO	27
4.1. Detalle de cada guía	27
4.1.1. Sistema base sobre MicroBlaze V	27
4.1.2. Uso del depurador	30
4.1.3. Uso de Periféricos e Interrupciones externas	30
4.1.4. Periférico personalizado en RTL	32
4.1.5. Periférico personalizado en HLS	33
4.1.6. Memoria externa	34
4.1.7. Acceso directo a memoria	35
4.1.8. Depuración a nivel de hardware	37
4.2. Listado de material	38
4.2.1. Repositorio de archivos fuente y de documentación	38
5. CONCLUSIONES Y TRABAJO FUTURO	40
5.1. Conclusiones	40
5.2. Recomendaciones para trabajo futuro	41
BIBLIOGRAFÍA	42

Índice de figuras

2.1.	Estructura genérica de un sistema heterogéneo sobre FPGA: el procesador accede a los aceleradores y a la memoria a través de un bus de interconexión común. Componentes morados actúan como Maestro y componentes azules actúan como clientes . . .	10
2.2.	Diagrama de flujo HLS sugerido por AMD.	13
2.3.	Transacción de lectura AXI sobre los canales AR y R. Se lee el valor 0xF desde la dirección 0x8	14
2.4.	Transacción de escritura AXI sobre los canales AW y W. Se escribe 0xFFFFFFFF en la dirección 0x4	15
2.5.	Diagrama <i>MicroBlaze V</i> conectado a un periférico UART y la memoria externa a través de AXI.	16
2.6.	Estructura de un archivo ELF.	18
3.1.	Diagrama de flujo de proceso de validación del material	25
4.1.	Conceptos cubiertos por cada guía. Las celdas oscuras marcan la guía que introduce un concepto; las claras, las guías que lo reutilizan.	28
4.2.	Diagrama de implementación de <i>MicroBlaze V</i>	29
4.3.	<i>MicroBlaze V</i> en el diagrama de bloques de Vivado.	29
4.4.	Vista de depuración de Vitis, con funciones y sus tags correspondientes en assembly en rojo.	31
4.5.	Direcciones base de periféricos AXI Uart-Lite y AXI GPIO en Address Map de Vivado.	31
4.6.	Diagrama de una interrupción.	32
4.7.	Diagrama de un Adder tree de 8 entradas de N bits.	32
4.8.	Diagrama de flujo de diseño periférico HLS, con paralelogramos representando insumos, rectángulos procesos y rombos toma de decisiones.	33
4.9.	Representación en punto fijo (8 bits). Código de colores: azul para el bit de signo, amarillo para la parte entera (3 bits) y lila para la parte fraccionaria (4 bits).	34
4.10.	Diagrama de funcionamiento de la Guía 6.	35
4.11.	Secuencia desde la obtención del cuadro hasta su salida en la interfaz VGA	36
4.12.	Secuencia de 3 tramas en un monitor.	37

4.13. Transacción de lectura de los interruptores de la placa Nexys A7 visualizado a través de la perspectiva de depuración de Vivado. Se lee el valor 0xF del registro de estado de AXI GPIO con dirección 0x08.	38
4.14. Página inicial asociada al repositorio de documentación	39

Capítulo 1

INTRODUCCIÓN

El presente informe describe el trabajo realizado en torno al desarrollo y sistematización de un flujo de diseño para sistemas de cómputo heterogéneo basados en FPGA, utilizando herramientas y tecnologías modernas del ecosistema AMD. El objetivo central es establecer una metodología práctica para el diseño, implementación e integración de plataformas que combinen una CPU embebida con unidades de hardware especializado, de modo que se reduzcan las barreras de entrada asociadas a este tipo de desarrollos. Asimismo, el material generado busca servir como base para futuros trabajos de docencia e investigación dentro del Departamento de Electrónica de la Universidad Técnica Federico Santa María.

En particular, el trabajo aborda el uso de la CPU *soft-core MicroBlaze V* de AMD, integrada en una tarjeta de desarrollo basada en FPGA junto con aceleradores descritos mediante *Hardware Description Languages* (HDL) y *High-Level Synthesis* (HLS). A partir de esta plataforma, se revisan los flujos de Vivado y Vitis para el diseño de hardware, la generación de la plataforma embebida, la depuración y ejecución de aplicaciones, y la comunicación software-hardware necesaria para construir un sistema heterogéneo funcional.

1.1. Motivación y contexto

Durante la última década, las FPGA han evolucionado desde dispositivos orientados principalmente a la implementación de bloques de lógica digital personalizada para aceleración de cálculos hacia plataformas capaces de ejecutar sistemas computacionales completos [1]. El aumento sostenido en capacidad lógica, memoria embebida y recursos aritméticos especializados, junto con la incorporación de procesadores integrados y buses de interconexión estandarizados, ha permitido implementar arquitecturas heterogéneas en las cuales una CPU y unidades de hardware especializado cooperan para ejecutar una misma aplicación.

El paradigma de cómputo heterogéneo ha adquirido creciente relevancia en aplicaciones como procesamiento digital de señales, visión computacional, inteligencia artificial y control avanzado, las cuales demandan simultáneamente un elevado desempeño computacional, baja latencia y eficiencia energética [2,3]. Mientras las CPUs de propósito general ofrecen la flexibilidad necesaria para ejecutar algoritmos complejos y coordinar el funcionamiento del sistema, muchas de las operaciones presentes en estas aplicaciones poseen una estructura altamente paralelizable que puede implementarse de manera más eficiente mediante hardware dedicado. En consecuencia, el cómputo heterogéneo busca combinar las ventajas de ambos enfoques, delegando las tareas de control al software y las operaciones computacionalmente intensivas a aceleradores implementados en hardware.

La adopción de este paradigma también se ha visto favorecida por la aparición de nuevas tecnologías de desarrollo. Por una parte, la arquitectura abierta RISC-V ha impulsado la disponibilidad de CPUs *soft-core* abiertas y personalizables, facilitando su incorporación dentro de plataformas FPGA. En paralelo, enfoques de diseño como *High-Level Synthesis* (HLS) han simplificado el desarrollo de periféricos y aceleradores hardware mediante lenguajes de alto nivel, reduciendo parte de la complejidad tradicional del diseño RTL [1]. Finalmente, herramientas de integración como Vivado y Vitis permiten construir plataformas embebidas completas dentro de un flujo relativamente unificado.

Sin embargo, a pesar de la disponibilidad de un amplio soporte tecnológico, el diseño de sistemas heterogéneos sigue siendo no trivial debido a la necesidad de integrar múltiples componentes con naturalezas muy distintas. El desarrollo de un sistema heterogéneo continúa requiriendo combinar diseño de hardware digital, programación de software embebido, configuración de interconexiones, integración de periféricos y mecanismos de comunicación entre CPU y aceleradores. Esta dificultad de integración ha sido caracterizada como uno de los principales desafíos pedagógicos en sistemas embebidos modernos, donde los estudiantes deben aprender a descomponer un sistema en subsistemas cuyas propiedades se sintetizan de forma consistente al integrarlos [4].

La necesidad de comprender los conceptos y el uso de herramientas que evolucionan rápidamente resulta especialmente relevante en contextos docentes y académicos, donde no basta con disponer de herramientas modernas, sino que también es necesario transformarlas en experiencias de aprendizaje estructuradas [4]. El diseño de sistemas heterogéneos sobre FPGA reúne conceptos de arquitectura de computadores, sistemas digitales, diseño de hardware, programación embebida e interfaces de comunicación, por lo que representa un tema de alto valor formativo para estudiantes de especialidad. Para que estas tecnologías puedan incorporarse de manera didáctica en cursos y trabajos futuros, se requiere una metodología que ordene el flujo de desarrollo, explicita las decisiones de diseño y permita reproducir los pasos necesarios para construir una plataforma funcional [5].

En el Departamento de Electrónica de la Universidad Técnica Federico Santa María, y en particular en el grupo en el cual se desarrolla este trabajo, existe amplia experiencia previa en el desarrollo de aceleradores hardware sobre FPGA para distintas aplicaciones. Dentro de esta línea, trabajos recientes han abordado la implementación de aceleradores para control predictivo por modelo mediante síntesis de alto nivel [6], así como la implementación en FPGA de redes neuronales artificiales aplicadas a lazos de control predictivo [7]. Estos desarrollos han permitido explorar distintas estrategias de aceleración hardware para aplicaciones de control avanzado, evaluando principalmente módulos específicos o partes de los algoritmos considerados. No obstante, dichos trabajos se han concentrado mayoritariamente en el acelerador como bloque individual, sin abordar de forma sistemática su integración dentro de una plataforma heterogénea completa que considere CPU, software embebido, mecanismos de comunicación, sincronización y validación conjunta. Avanzar hacia esta integración constituye un paso natural para aproximar estos desarrollos a escenarios de validación más representativos y, al mismo tiempo, generar una base metodológica útil para docencia e investigación.

1.2. Planteamiento del problema

Actualmente, las principales barreras para el desarrollo de sistemas heterogéneos sobre FPGA ya no se encuentran en la disponibilidad del hardware o de las herramientas de desarrollo. Existen plataformas FPGA de bajo costo para uso académico, arquitecturas abiertas como RISC-V que reducen restricciones de licenciamiento, herramientas comerciales como Vivado y Vitis de AMD incluyen licencias gratuitas y capacidades suficientes para uso académico, y enfoques como HLS que simplifican el diseño de aceleradores. No obstante, la construcción de un sistema heterogéneo funcional continúa siendo un proceso complejo para quienes se incorporan al área, imponiendo una barrera que limita la entrada y desacelera la adopción.

Esta complejidad proviene principalmente de la integración de múltiples flujos de trabajo. Mientras el desarrollo de hardware involucra síntesis lógica, temporización, generación de periféricos e integración de buses, el desarrollo de software embebido requiere compilación, depuración, configuración de periféricos y control de la ejecución desde la CPU. Además, ambos dominios deben comunicarse

mediante mapas de memoria, registros de control, interrupciones, interfaces de comunicación y mecanismos de transferencia de datos. En la práctica, una parte importante del esfuerzo no se concentra únicamente en diseñar cada componente por separado, sino en lograr que todos ellos operen correctamente como un sistema.

La dificultad de integración inherente se ve acentuada por la forma en que el conocimiento técnico suele estar documentado. La documentación oficial de las herramientas de diseño y verificación, así como las especificaciones de protocolos como AMBA AXI [8] o las guías de usuario de Vitis HLS [9], están orientadas principalmente a usuarios con experiencia previa que requieren información de referencia. La estructura de estos documentos responde al modelo de manual técnico: completa, exhaustiva y precisa, pero no necesariamente pedagógica. En general, existen numerosos recursos que describen componentes individuales del sistema, tales como procesadores soft core, aceleradores HLS o interconexiones AXI. Sin embargo, la integración de estos elementos en una plataforma funcional requiere conocimientos que rara vez aparecen consolidados en una única fuente. Por ello, quien se aproxima al área por primera vez no siempre encuentra una ruta gradual desde un sistema básico hacia una plataforma heterogénea funcional, sino un conjunto amplio de capacidades cuya articulación queda a su cargo.

La rápida evolución del ecosistema agrava este problema. Las herramientas modifican interfaces, renombran productos y reorganizan flujos de trabajo entre versiones consecutivas. La transición desde Vivado HLS hacia Vitis HLS, así como la reciente introducción de *MicroBlaze V*, han dejado parcial o totalmente desactualizada una porción significativa de los tutoriales y ejemplos previamente disponibles. Como consecuencia, el conocimiento documentado pierde vigencia con rapidez, y cada nueva generación de estudiantes y desarrolladores debe reconstruir buena parte del flujo de trabajo desde fuentes incompletas, fragmentadas o discordantes entre sí.

En consecuencia, el presente trabajo propone el desarrollo de una plataforma de aprendizaje orientada al diseño de sistemas de cómputo heterogéneo sobre FPGA. La plataforma integra guías de trabajo, diseños de referencia, aplicaciones de ejemplo y código fuente reutilizable, organizados de forma progresiva para acompañar al lector desde la construcción de un sistema embebido básico hasta la integración de periféricos personalizados y mecanismos avanzados de transferencia de datos. De esta manera, el foco no se limita a documentar herramientas individuales, sino a proporcionar una ruta estructurada para la adquisición gradual de competencias en diseño e integración hardware/software, además de establecer un punto de partida para futuros trabajos de desarrollo e investigación.

1.2.1. Objetivos

El objetivo general de este trabajo de título consiste en explorar y documentar un flujo de desarrollo estructurado y reproducible para el diseño, implementación e integración de un sistema heterogéneo basado en una CPU *soft-core MicroBlaze V* y aceleradores de cómputo implementados en lógica reconfigurable, orientado a su utilización como base para futuras actividades docentes y de investigación en sistemas embebidos y computación heterogénea.

Los objetivos específicos son:

1. Diseñar e implementar un sistema basado en el procesador *MicroBlaze V* sobre una tarjeta de desarrollo basada en FPGA.
2. Diseñar e incorporar periféricos personalizados mediante HDL y/o *High-Level Synthesis*, definiendo sus interfaces de comunicación, señales de control y parámetros básicos de implementación.
3. Integrar los aceleradores al sistema basado en *MicroBlaze V* mediante interconexiones AXI, considerando comunicación, control, sincronización y acceso desde software embebido.
4. Validar funcionalmente el flujo de integración hardware/software mediante aplicaciones de prueba que permitan verificar la operación conjunta entre CPU, aceleradores y periféricos.

5. Documentar el flujo de trabajo, las decisiones de diseño y las principales consideraciones prácticas, de forma que el material resultante pueda ser reutilizado en docencia y trabajos de desarrollo e investigación.

1.3. Alcances y contribuciones

Este informe apunta a la exposición del trabajo y los resultados obtenidos mediante la generación de material didáctico orientado al área de computación heterogénea. A continuación se listan los alcances y contribuciones de esta memoria:

- El diseño, implementación y simulación de los componentes de hardware se realizan utilizando el lenguaje de descripción de hardware *SystemVerilog* (estándar IEEE 1800). La elección de este lenguaje responde a que forma parte de los contenidos impartidos durante la formación en Ingeniería Civil Electrónica de la Universidad Técnica Federico Santa María, facilitando así la adopción y comprensión del material por parte de futuros estudiantes.
- Las guías desarrolladas consideran un flujo de trabajo orientado al diseño sobre FPGA mediante las herramientas *AMD Vivado 2025.2* y *Vitis 2025.2*, correspondientes a la versión más reciente disponible al momento de la elaboración de este trabajo. La elección de esta versión permite asegurar la vigencia del material desarrollado y evitar dependencias con herramientas o flujos de trabajo que han sido descontinuados. En particular, a partir de la versión 2025.1 AMD eliminó definitivamente el entorno *Vitis Classic*, consolidando el uso del nuevo flujo de desarrollo basado en *Vitis Unified IDE*. Si bien los conceptos, arquitecturas y metodologías presentados son aplicables a otras versiones de las herramientas, algunos procedimientos específicos, configuraciones e interfaces gráficas podrían requerir adaptaciones menores.
- Como plataforma objetivo se utiliza la tarjeta de desarrollo *Nexys A7* de Digilent, basada en una FPGA Artix-7 de AMD. Esta selección se fundamenta en su amplia disponibilidad dentro de la Universidad Técnica Federico Santa María, así como en su bajo costo relativo respecto a otras plataformas de desarrollo, favoreciendo la reproducibilidad de los resultados obtenidos. La elección de esta placa no constituye una limitante para el alcance pedagógico del trabajo, puesto que los diseños desarrollados no buscan explotar características particulares del dispositivo, sino ilustrar el proceso de diseño e implementación de sistemas de cómputo heterogéneo sobre FPGA. En consecuencia, los conceptos, metodologías y arquitecturas presentados son aplicables a otras plataformas FPGA que permitan la incorporación de procesadores *soft-core* y mecanismos equivalentes de interconexión. No obstante, debido al uso de bloques IP propietarios del ecosistema AMD en varias de las guías, la portabilidad directa del material queda restringida a dispositivos compatibles con las herramientas Vivado y Vitis. La adaptación a plataformas de otros fabricantes requeriría reemplazar dichos componentes por alternativas funcionalmente equivalentes, aspecto que se encuentra fuera de los alcances de este trabajo.
- El desarrollo de periféricos personalizados mediante síntesis de alto nivel (HLS) se realiza exclusivamente utilizando *Vitis Unified IDE*, herramienta integrada dentro del ecosistema de desarrollo de AMD. Esta misma plataforma es empleada para el desarrollo de aplicaciones de software, la integración hardware-software y la programación de la tarjeta de desarrollo. Los periféricos implementados tienen un propósito principalmente didáctico y buscan ilustrar el flujo de diseño, generación e integración de componentes personalizados dentro de un sistema heterogéneo basado en FPGA. La evaluación detallada de métricas de rendimiento, utilización de recursos o eficiencia de las implementaciones generadas mediante HLS queda fuera de los alcances de este trabajo y se identifica como una línea de desarrollo futura.

El alcance incluye además la validación funcional de los sistemas desarrollados mediante la ejecución de aplicaciones de prueba. Estas pruebas permiten verificar la correcta interacción entre el procesador y los periféricos personalizados, así como demostrar la viabilidad del flujo de diseño propuesto. De

esta forma, el trabajo se centra en la generación y validación de material docente y de referencia que facilite la adopción de metodologías de diseño de sistemas heterogéneos basados en FPGA en futuros proyectos académicos.

La principal contribución de este trabajo corresponde al desarrollo y documentación de un flujo de diseño reproducible para la construcción de sistemas de cómputo heterogéneo sobre FPGA. Este flujo aborda de manera integrada las distintas etapas involucradas en el desarrollo de este tipo de sistemas, desde la creación de una plataforma embebida basada en un procesador RISC-V hasta la incorporación de aceleradores hardware y su interacción con aplicaciones de software. Para ello, se emplean herramientas y metodologías ampliamente utilizadas en el diseño digital moderno, reuniendo en una única referencia conceptos y procedimientos que habitualmente se encuentran distribuidos entre múltiples fuentes.

La integración de los periféricos personalizados dentro del sistema se documenta de manera detallada, incluyendo la definición de interfaces, los mecanismos de comunicación y las estrategias de sincronización entre hardware y software necesarias para construir una plataforma funcional. Asimismo, las guías presentan ejemplos prácticos que permiten observar cómo las distintas etapas del flujo de diseño se relacionan entre sí dentro de una aplicación completa.

En conjunto, el material desarrollado define una ruta de aprendizaje estructurada que integra tecnologías y herramientas habitualmente estudiadas de forma aislada. Con ello, se busca reducir las barreras de entrada asociadas al diseño de sistemas heterogéneos y proporcionar una base para actividades docentes, laboratorios y futuros trabajos de investigación en el área de sistemas embebidos y cómputo heterogéneo.

1.4. Organización del informe

Los entregables principales de este trabajo consisten en guías de trabajo estructuradas y material de referencia para la enseñanza de diseño de sistemas de cómputo heterogéneo sobre FPGA. Este material se encuentra disponible en los siguientes repositorios de código:

- Guías: Descripción breve y formato de enlace.
- Códigos y demás:

Las versiones entregadas al término de este trabajo de memoria se encuentran probado y validado por el profesor guía y grupos de estudiantes con experiencia en diseño digital. Al quedar disponibles en formato de repositorio, se espera que este material se expanda o actualice en futuras iteraciones.

El resto de este informe sirve como complemento al entregable principal, y se limita a entregar información complementaria sobre aspectos de decisiones de diseño y procedimientos aplicados para el diseño de actividades y resultados que no están disponibles en el repositorio.

Los siguientes capítulos se estructuran de la siguiente manera:

- **Capítulo 1 - Introducción:** Se presentan las motivaciones que dieron origen al trabajo, los objetivos planteados y los alcances considerados, finalizando con una descripción de la estructura del documento.
- **Capítulo 2 - Antecedentes:** Se revisan los conceptos fundamentales relacionados con lógica reconfigurable, cómputo heterogéneo, procesadores RISC-V, síntesis de alto nivel e interfaces de interconexión, además de introducir las herramientas utilizadas durante el desarrollo.
- **Capítulo 3 - Metodología y Desarrollo:** Se describe el proceso seguido para la elaboración del material, incluyendo el análisis de recursos existentes, la definición de criterios de diseño, la planificación de las guías y la metodología utilizada para su validación.

- **Capítulo 4 - Material Desarrollado:** Se presentan las guías elaboradas durante el trabajo, describiendo sus objetivos, contenidos y nivel de complejidad. Asimismo, se documenta la estructura del repositorio generado y el conjunto de recursos puestos a disposición para futuros usuarios.
- **Capítulo 5 - Conclusiones y Trabajo Futuro:** Se analizan los resultados obtenidos, las contribuciones realizadas y las lecciones aprendidas durante el desarrollo del proyecto. Finalmente, se proponen posibles líneas de trabajo futuro para ampliar y complementar el material generado.

Capítulo 2

ANTECEDENTES

Los conceptos presentados en este capítulo constituyen la base teórica necesaria para comprender los ejemplos y sistemas desarrollados en las guías generadas como parte de este trabajo. En particular, se introducen los principales componentes y tecnologías que conforman un sistema de cómputo heterogéneo basado en FPGA.

2.1. Lógica reconfigurable y cómputo heterogéneo

El desarrollo de sistemas de cómputo modernos ha evolucionado hacia arquitecturas heterogéneas, en las cuales distintos tipos de unidades de procesamiento colaboran dentro de un mismo sistema. El objetivo de este enfoque es asignar cada tarea al recurso computacional más adecuado. Por ejemplo, una CPU puede encargarse del control general, la toma de decisiones y la ejecución secuencial del software, mientras que aceleradores hardware pueden ejecutar operaciones repetitivas, paralelizables o de alta demanda computacional.

En un sistema heterogéneo, el desempeño no depende únicamente de la capacidad de cómputo de cada bloque, sino también de la forma en que estos bloques se integran. Aspectos como el movimiento de datos, la latencia de comunicación, la sincronización entre hardware y software, el acceso a memoria y la sobrecarga de control pueden determinar si la aceleración hardware resulta realmente beneficiosa. Por esta razón, el diseño de este tipo de sistemas requiere analizar tanto la arquitectura de cómputo como la arquitectura de interconexión.

Las plataformas basadas en lógica reconfigurable son especialmente atractivas para implementar sistemas heterogéneos, ya que permiten implementar hardware especializado sin fabricar un circuito integrado dedicado. En una FPGA es posible implementar CPUs, periféricos, controladores, memorias locales y aceleradores dentro del mismo dispositivo. Esta flexibilidad permite explorar distintas particiones hardware/software y evaluar diferentes formas de integrar coprocesadores a una plataforma embebida.

2.1.1. Field-Programmable Gate Arrays

Una FPGA, o *Field-Programmable Gate Array*, es un dispositivo digital cuya arquitectura interna puede configurarse después de su fabricación. Su estructura se compone principalmente de bloques lógicos configurables, redes de interconexión programables, bloques de memoria embebida, bloques DSP y recursos de entrada/salida.

Los bloques lógicos configurables permiten implementar funciones combinacionales y secuenciales.

Estos bloques usualmente se construyen en base a *Look-Up Tables* (LUT), flip-flops y multiplexores. Las LUT permiten representar funciones booleanas arbitrarias de un número limitado de entradas, mientras que los flip-flops permiten registrar señales y construir circuitos secuenciales. La red de interconexión programable permite conectar estos recursos para formar arquitecturas digitales más complejas. En dispositivos FPGA modernos, la cantidad de lógica disponible puede variar desde algunos miles hasta varios millones de elementos lógicos equivalentes, dependiendo de la familia y gama del dispositivo. Esta flexibilidad permite implementar desde pequeños sistemas embebidos hasta plataformas heterogéneas completas con múltiples CPUs, memorias y aceleradores hardware integrados en un único chip.

Además de la lógica general, la cual entrega versatilidad para implementar lógica arbitraria, las FPGA modernas incluyen recursos especializados. Los bloques DSP implementan operaciones aritméticas frecuentes, entre estas destaca la operación **multiply-accumulate** (MAC), que combina una multiplicación y una suma en un solo ciclo. Las memorias embebidas, como BRAM o URAM, permiten almacenar datos localmente dentro del dispositivo, reduciendo la dependencia de memorias externas y permitiendo construir buffers, FIFOs, tablas y memorias compartidas entre módulos con estructuras arbitrarias. Estos recursos son fundamentales para implementar unidades de procesamiento especializadas eficientes, especialmente en aplicaciones con procesamiento intensivo de datos.

La principal ventaja de una FPGA es que permite personalizar las estructuras de cómputo para explotar eficientemente patrones de paralelismo espacial y temporal. En una CPU, varias operaciones suelen compartir las mismas unidades funcionales a lo largo del tiempo. En una FPGA, en cambio, es posible instanciar múltiples unidades funcionales dedicadas que operan en paralelo. También es posible construir pipelines profundos, donde distintas etapas del cálculo procesan diferentes datos simultáneamente. Esto puede aumentar considerablemente el *throughput* del sistema cuando el algoritmo posee una estructura adecuada y es posible recibir flujos de datos continuos.

Por otro lado, el uso eficiente de una FPGA requiere considerar restricciones propias del diseño hardware. Entre ellas se encuentran el uso de recursos lógicos, la frecuencia máxima de operación, los caminos críticos, la temporización, el ancho de banda de memoria y la latencia de comunicación entre módulos. Por lo tanto, la implementación de unidades de procesamiento especializadas no consiste únicamente en traducir un algoritmo a hardware, sino en definir una microarquitectura adecuada para los recursos disponibles y para el patrón de datos de la aplicación.

2.1.2. Procesadores integrados

A medida que la capacidad lógica de las FPGA ha aumentado, también ha crecido la complejidad de los sistemas implementados sobre ellas. Actualmente, una FPGA puede contener múltiples aceleradores hardware, periféricos, memorias y módulos especializados operando simultáneamente. En este contexto, resulta necesario incorporar CPUs capaces de coordinar la operación general del sistema, administrar el movimiento de datos y ejecutar tareas de control que no son adecuadas para implementación hardware dedicada. De esta forma, las CPUs integradas permiten transformar la FPGA desde un conjunto aislado de módulos hardware hacia una plataforma programable capaz de ejecutar sistemas heterogéneos complejos.

En sistemas basados en FPGA, las CPUs integradas permiten ejecutar software embebido y coordinar la operación de los distintos bloques hardware. Estas CPUs pueden clasificarse en dos categorías principales: *hard core* y *soft core*.

Una CPU *hard core* corresponde a un procesador implementado físicamente en el silicio del dispositivo. Este tipo de CPU no consume recursos de lógica programable y suele ofrecer mayor frecuencia de operación, mejor eficiencia energética y acceso directo a subsistemas de memoria e interconexión optimizados por el fabricante. Un ejemplo representativo son los dispositivos SoC FPGA, como la familia Zynq de AMD Xilinx, que integra CPUs ARM junto con lógica programable en el mismo chip.

Una CPU *soft core*, en cambio, se implementa usando los recursos reconfigurables de la FPGA. Esto significa que la CPU se sintetiza como cualquier otro módulo digital, utilizando LUT, flip-flops,

BRAMs y otros recursos disponibles. Esta aproximación permite configurar aspectos como periféricos, tamaño de memoria local, interfaces de comunicación, presencia de unidades auxiliares y, en algunos casos, características de la arquitectura del procesador.

La principal ventaja de una CPU *soft core* es su flexibilidad. Puede incorporarse solo cuando el diseño lo requiere, modificarse según las necesidades del sistema o instanciarse varias veces para construir arquitecturas con múltiples CPUs dentro de la misma FPGA. Esta característica resulta útil para exploración arquitectónica, docencia e integración de sistemas donde se desea mantener control sobre la estructura completa de la plataforma. Su principal desventaja es que normalmente alcanza menor frecuencia de operación y consume recursos de lógica programable que podrían utilizarse para aceleradores.

En el contexto actual, RISC-V ha impulsado el interés por CPUs *soft core* abiertas y personalizables. Al ser una arquitectura de conjunto de instrucciones abierta, permite implementar CPUs compatibles sin depender de licencias propietarias de ISA. Esta apertura facilita la investigación, la docencia y la experimentación con extensiones específicas. En ecosistemas comerciales, soluciones como *MicroBlaze V* incorporan una CPU *soft core* basada en RISC-V dentro del flujo de diseño de AMD Xilinx, facilitando su integración con periféricos, memorias e interconexiones estándar.

Dentro de un sistema heterogéneo, la CPU no necesariamente busca ejecutar la parte más intensiva del algoritmo. Su rol suele estar asociado a la inicialización del sistema, configuración de registros, transferencia de datos, manejo de interrupciones, sincronización con aceleradores y ejecución de tareas de control. Los coprocesadores o aceleradores, en cambio, implementan las partes del algoritmo que se benefician de una arquitectura hardware dedicada.

2.1.3. Buses de interconexión

La integración de CPUs, memorias, periféricos y aceleradores requiere mecanismos de comunicación bien definidos. Para esta comunicación se utilizan buses o redes de interconexión, los cuales especifican cómo se transfieren datos, direcciones y señales de control entre módulos.

Un bus de interconexión define, entre otros aspectos, el modelo de comunicación entre maestros y esclavos, el protocolo de lectura y escritura, las señales de validación, la respuesta ante errores y el mecanismo de temporización. En sistemas simples, un bus puede conectar una CPU con varios periféricos mediante transacciones de lectura y escritura. En sistemas más complejos, puede ser necesario soportar múltiples maestros, acceso concurrente a memoria, arbitraje, transferencias por ráfagas y flujos continuos de datos.

Wishbone es un bus abierto ampliamente utilizado en proyectos académicos y diseños *open-source*. Su objetivo principal es facilitar la reutilización de módulos IP mediante una interfaz común. Wishbone define señales simples para dirección, datos, escritura, selección de bytes, reconocimiento y ciclos de transferencia. Su simplicidad lo hace atractivo para diseños educativos y sistemas relativamente pequeños, donde se busca claridad en la integración y bajo costo de implementación.

AXI, por otra parte, forma parte de la especificación AMBA y es ampliamente utilizado en plataformas FPGA comerciales. AXI está diseñado para sistemas de mayor desempeño y complejidad. Una característica importante de AXI es que separa la comunicación en canales independientes para dirección de lectura, datos de lectura, dirección de escritura, datos de escritura y respuesta de escritura. Esta separación permite mayor concurrencia y facilita transferencias eficientes, especialmente cuando se conectan CPUs, memorias y aceleradores.

En sistemas heterogéneos sobre FPGA, la interconexión cumple un rol central porque determina cómo se coordinan los distintos componentes del sistema. Una arquitectura con aceleradores muy eficientes puede no entregar beneficios significativos si el movimiento de datos se transforma en el cuello de botella. Por esta razón, el diseño de buses, interfaces y mecanismos de transferencia es una parte fundamental del desarrollo de plataformas de cómputo heterogéneo.



Figura 2.1: Estructura genérica de un sistema heterogéneo sobre FPGA: el procesador accede a los aceleradores y a la memoria a través de un bus de interconexión común. Componentes morados actúan como Maestro y componentes azulados actúan como clientes

2.1.4. Desafíos técnicos en sistemas heterogéneos sobre FPGA

El desarrollo de sistemas heterogéneos sobre FPGA requiere coordinar componentes que operan bajo modelos de ejecución distintos. Mientras la CPU ejecuta instrucciones de manera secuencial sobre un programa almacenado en memoria, los aceleradores hardware implementan circuitos dedicados capaces de operar de forma concurrente. Como consecuencia, la integración del sistema no se limita a conectar módulos entre sí, sino que exige definir mecanismos adecuados de comunicación, intercambio de datos y sincronización. La Figura 2.1 muestra una estructura genérica de este tipo de sistemas, donde procesador, memoria y aceleradores comparten una infraestructura común de interconexión.

Uno de los principales desafíos consiste en determinar cómo se distribuyen las responsabilidades entre hardware y software. Las tareas de control y coordinación suelen ejecutarse en la CPU, mientras que las operaciones más intensivas pueden implementarse mediante aceleradores especializados. Sin embargo, esta decisión debe considerar no solo el costo computacional de cada tarea, sino también la sobrecarga asociada a la transferencia de datos y la coordinación entre ambos dominios.

La comunicación entre los distintos componentes introduce desafíos adicionales relacionados con las interfaces de interconexión y la organización de memoria. La elección de los mecanismos de acceso a datos condiciona tanto el desempeño del sistema como la complejidad de integración. De forma similar, la distribución de la información entre memorias internas y externas puede transformarse en un factor determinante para el rendimiento global de la plataforma.

Finalmente, la sincronización entre CPU y aceleradores, junto con la validación conjunta de hardware y software, constituye una parte fundamental del desarrollo de sistemas heterogéneos. Estos aspectos influyen directamente en el diseño de la plataforma y serán retomados posteriormente al definir los criterios utilizados para la construcción de las guías desarrolladas en este trabajo.

2.2. Arquitectura RISC-V y MicroBlaze V

2.2.1. Arquitectura RISC-V

Durante los últimos años, las arquitecturas abiertas de conjunto de instrucciones han adquirido una relevancia creciente dentro del desarrollo de sistemas embebidos y plataformas de investigación. Tradicionalmente, gran parte de las CPUs comerciales se han basado en arquitecturas propietarias, donde la especificación de la ISA (*Instruction Set Architecture*) y sus implementaciones se encuentran controladas por una empresa específica. Esta restricción puede limitar la posibilidad de modificar la arquitectura, desarrollar implementaciones personalizadas o estudiar internamente ciertos aspectos del procesador.

En este contexto de ISAs propietarias, RISC-V surgió como una alternativa abierta y modular para el diseño de CPUs y sistemas embebidos [10]. RISC-V corresponde a una ISA basada en la filosofía RISC (*Reduced Instruction Set Computer*), caracterizada por un conjunto de instrucciones relativamente simple, formatos de instrucción regulares y una separación clara entre operaciones aritméticas y acceso a memoria. La arquitectura fue originalmente desarrollada en la Universidad de California, Berkeley,

con el objetivo de disponer de una ISA abierta, extensible y adecuada tanto para investigación como para aplicaciones comerciales.

Una de las principales características de RISC-V es su modularidad. La ISA base define un conjunto mínimo de instrucciones enteras, mientras que funcionalidades adicionales se incorporan mediante extensiones opcionales. Por ejemplo, existen extensiones para multiplicación y división entera, operaciones atómicas, punto flotante y procesamiento vectorial. Esta aproximación permite adaptar la complejidad de una CPU según los requerimientos de cada aplicación.

Otro aspecto relevante es que RISC-V define la ISA, pero no impone una microarquitectura específica. En consecuencia, distintos diseñadores pueden desarrollar implementaciones completamente diferentes manteniendo compatibilidad binaria con el mismo conjunto de instrucciones. Esto ha favorecido el desarrollo de CPUs *soft core*, CPUs académicas, procesadores comerciales y plataformas experimentales basadas en una misma ISA.

Dentro del contexto de FPGA, RISC-V ha impulsado significativamente el desarrollo de CPUs *soft core* abiertas y personalizables. Esta compatibilidad permite reducir la dependencia de arquitecturas propietarias y facilita la integración de CPUs dentro de plataformas heterogéneas reconfigurables.

2.2.2. MicroBlaze V

En el marco de la tendencia hacia ISAs de código abierto, AMD introdujo MicroBlaze V como una evolución de su ecosistema tradicional de CPUs *soft core*. Históricamente, la familia MicroBlaze correspondía a una arquitectura propietaria desarrollada específicamente para FPGA Xilinx. Estas CPUs han sido ampliamente utilizadas en sistemas embebidos sobre FPGA debido a su integración con Vivado, soporte de periféricos AXI y facilidad de incorporación dentro de diseños basados en *IP Integrator*. Sin embargo, al tratarse de una ISA propietaria, el ecosistema de herramientas y las posibilidades de portabilidad se encontraban más limitadas en comparación con alternativas abiertas.

MicroBlaze V mantiene gran parte de la filosofía de integración y flujo de desarrollo del MicroBlaze clásico, pero adopta una ISA basada en RISC-V. Esto permite combinar las ventajas del ecosistema de herramientas de AMD Xilinx con la flexibilidad y apertura del estándar RISC-V. Desde el punto de vista práctico, *MicroBlaze V* se integra directamente dentro del flujo Vivado/Vitis, permitiendo utilizar herramientas ya consolidadas para generación de hardware, desarrollo de software embebido, depuración y configuración de sistemas heterogéneos.

Al igual que otros procesadores *soft core*, *MicroBlaze V* se implementa utilizando lógica programable dentro de la FPGA. Esto implica que la CPU utiliza LUT, flip-flops, BRAMs y otros recursos internos del dispositivo, junto con periféricos e interconexiones definidos durante el diseño del sistema. Esta aproximación permite incorporar una o varias CPUs dentro del mismo dispositivo FPGA, dependiendo de los recursos disponibles y de las necesidades de la aplicación.

Desde el punto de vista arquitectónico, *MicroBlaze V* utiliza una organización tipo Harvard, separando los caminos de instrucción y datos. Esta organización permite reducir conflictos de acceso entre búsqueda de instrucciones y operaciones de lectura o escritura de datos. Asimismo, el procesador soporta integración mediante buses AXI, permitiendo conectar memorias, periféricos y aceleradores hardware dentro de una arquitectura heterogénea estandarizada.

Una ventaja importante de *MicroBlaze V* respecto al MicroBlaze clásico es la compatibilidad directa con el ecosistema abierto RISC-V. Esto permite utilizar herramientas de compilación estándar, soporte de bibliotecas abiertas y una mayor portabilidad de software entre distintas implementaciones compatibles con la ISA. Además, facilita el estudio de arquitecturas abiertas en contextos académicos y de investigación, evitando la dependencia exclusiva de una ISA propietaria.

Desde el punto de vista docente y de exploración arquitectónica, *MicroBlaze V* resulta particularmente interesante porque permite construir plataformas heterogéneas modernas utilizando herramientas ampliamente disponibles en entornos FPGA comerciales. Esto facilita experimentar con integración CPU-acelerador, buses AXI, periféricos embebidos y desarrollo de software sobre

arquitecturas RISC-V, manteniendo un flujo relativamente accesible para estudiantes y desarrolladores.

No obstante, al tratarse de una plataforma reciente, parte del ecosistema asociado a *MicroBlaze V* continúa evolucionando. Algunas herramientas, flujos de integración y ejemplos disponibles todavía presentan limitaciones o documentación parcial, especialmente en comparación con plataformas más maduras. Este desarrollo continuo convierte a *MicroBlaze V* en una plataforma atractiva tanto para exploración tecnológica como para el estudio práctico de metodologías de integración de sistemas heterogéneos sobre FPGA.

2.3. High-Level Synthesis para diseño de hardware especializado

High-Level Synthesis (HLS) corresponde a un enfoque de diseño que permite generar hardware digital a partir de descripciones escritas en lenguajes de alto nivel, tales como C, C++ o SystemC. A diferencia del diseño tradicional mediante lenguajes de descripción de hardware como VHDL o Verilog, HLS permite expresar el comportamiento funcional de un algoritmo en un nivel de abstracción más cercano al software, facilitando a ingenieros sin experiencia profunda en RTL describir y prototipar aceleradores [11].

En el contexto de sistemas heterogéneos sobre FPGA, HLS resulta particularmente útil para el diseño de unidades de hardware especializado, tales como coprocesadores, aceleradores o periféricos personalizados. Estas unidades pueden encargarse de ejecutar partes específicas de una aplicación que presentan alto paralelismo, procesamiento repetitivo o una carga aritmética significativa. De esta forma, HLS complementa el desarrollo de sistemas heterogéneos al facilitar la implementación de bloques hardware dedicados que posteriormente pueden integrarse con una CPU mediante interconexiones estándar.

Es importante destacar que HLS no reemplaza el diseño completo del sistema heterogéneo. Si bien permite generar aceleradores hardware a partir de descripciones en alto nivel, la plataforma final continúa requiriendo la integración de procesadores, memorias, interconexiones y periféricos, así como el desarrollo del software encargado de controlar y coordinar dichos componentes.

El flujo típico de diseño mediante HLS comienza con la descripción funcional del algoritmo en un lenguaje de alto nivel. Posteriormente, el diseñador aplica directivas de optimización que orientan la generación de hardware. Entre las más utilizadas se encuentran el pipelining, el loop unrolling, el particionamiento de arreglos y la definición de interfaces de entrada y salida. Estas directivas permiten modificar la microarquitectura generada, afectando directamente métricas como latencia, throughput, frecuencia máxima de operación y utilización de recursos.

Una característica importante de HLS es que el código sintetizable no se traduce automáticamente en hardware eficiente. El diseñador debe comprender cómo las estructuras del programa influyen en la microarquitectura resultante, ya que decisiones relacionadas con iteraciones, almacenamiento de datos e interfaces afectan directamente el paralelismo disponible y el consumo de recursos.

Desde el punto de vista de integración, los módulos generados mediante HLS pueden exponerse mediante distintas interfaces AXI. Dependiendo de los requerimientos de la aplicación, un acelerador puede utilizar registros de control mediante AXI4-Lite, acceso a memoria mediante AXI4 o interfaces AXI-Stream para el procesamiento continuo de datos.

Por esta razón, el diseño mediante HLS debe considerar no solo el algoritmo implementado, sino también su integración dentro del sistema. El desempeño final depende tanto de la arquitectura interna del acelerador como de factores asociados a la comunicación, acceso a memoria y sincronización con el software. En consecuencia, la definición de interfaces y mecanismos de transferencia de datos constituye una parte central del diseño de hardware especializado mediante HLS.

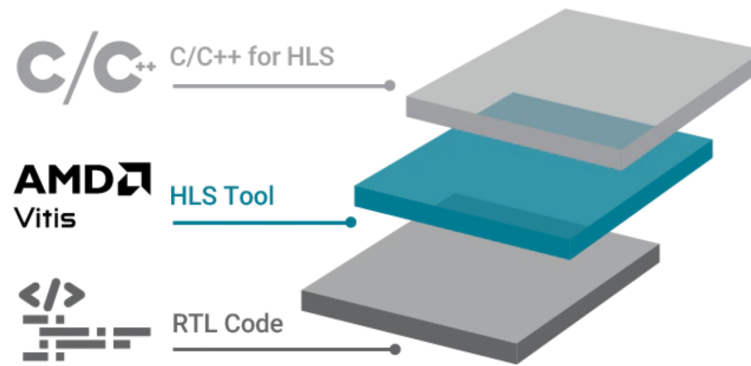


Figura 2.2: Diagrama de flujo HLS sugerido por AMD.

2.3.1. Herramientas de diseño: Vivado y Vitis

El diseño completo de un sistema sobre un dispositivo SoC con lógica programable involucra tres dominios complementarios: el diseño del hardware basado en lenguajes de descripción (HDL), el diseño de aceleradores a partir de descripciones algorítmicas en C/C++, y el desarrollo del software que se ejecuta sobre los procesadores embebidos. AMD ofrece para estos tres dominios un conjunto integrado de herramientas, de las cuales este trabajo utiliza Vivado, Vitis HLS y Vitis Embedded. Cada herramienta cubre uno de los dominios y se comunica con las otras a través de artefactos bien definidos, de modo que el flujo conjunto va desde el código C hasta la imagen de arranque sobre el dispositivo.

Vitis HLS es la herramienta de síntesis de alto nivel (*High-Level Synthesis*) de AMD, y traduce funciones escritas en C o C++ a una descripción RTL sintetizable [9]. El flujo permite validar el comportamiento del módulo mediante simulación a nivel C y luego aplicar directivas (*pragmas*) que guían la síntesis hacia una microarquitectura específica, controlando aspectos como el paralelismo a nivel de tarea, el *pipelining* de los lazos y la interfaz del módulo con el resto del sistema. El resultado se empaqueta como una IP que Vivado puede instanciar dentro del diseño general. El flujo sugerido por AMD se puede apreciar en la Figura 2.2. Este trabajo emplea Vitis HLS para describir, verificar y encapsular los aceleradores algorítmicos que posteriormente se integran en el diseño del FPGA.

Vivado Design Suite es el entorno de diseño integrado que AMD provee para sus FPGAs y SoCs adaptables [12]. Vivado cubre todo el flujo en lógica programable: ingreso del diseño en VHDL o Verilog, integración de bloques mediante IP Integrator, simulación con su simulador HDL integrado, síntesis lógica, ubicación y ruteo (*place and route*), y generación del *bitstream* que configura el dispositivo. En este trabajo, Vivado cumple dos funciones: alberga los módulos descritos directamente en HDL e integra, en un único diseño a nivel de sistema, los aceleradores producidos por Vitis HLS junto con los bloques de IP provistos por AMD.

Vitis Embedded es el componente de la plataforma Vitis destinado al desarrollo de aplicaciones C/C++ que se ejecutan sobre los procesadores embebidos de los dispositivos de AMD: procesadores Arm en las familias Zynq 7000, Zynq UltraScale+ MPSoC y Versal, y procesadores soft MicroBlaze en FPGAs [13]. La herramienta toma como entrada la descripción del hardware exportada desde Vivado y, a partir de ella, genera una plataforma de software con los *drivers*, bibliotecas (BSP) y archivos de cabecera correspondientes a los periféricos presentes en el diseño. Sobre esta plataforma se escribe, compila, depura y perfila la aplicación, y se construye la imagen de *boot* que se carga en el dispositivo. Este trabajo utiliza Vitis Embedded para implementar y validar el software del sistema, incluyendo la comunicación con los aceleradores diseñados en Vitis HLS a través de las interfaces expuestas por Vivado.

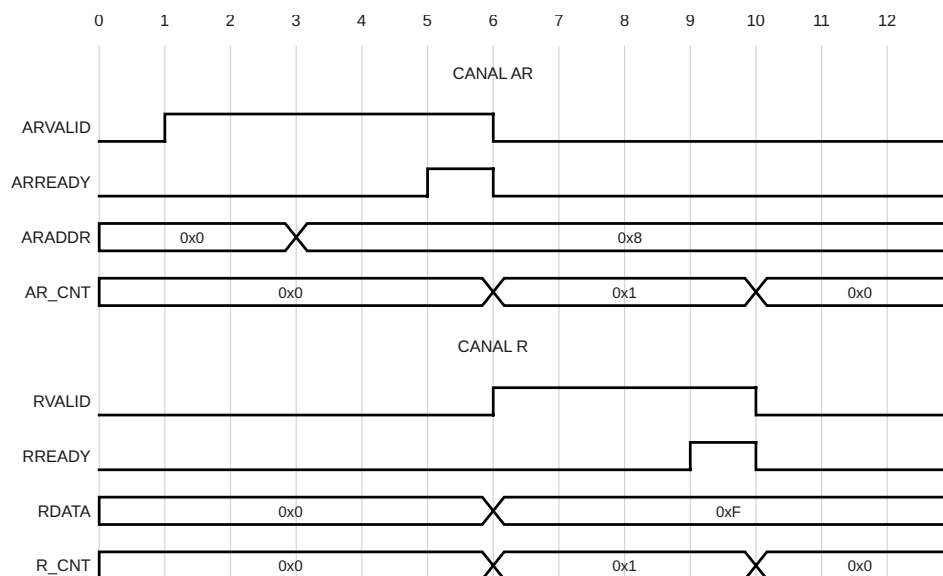


Figura 2.3: Transacción de lectura AXI sobre los canales AR y R. Se lee el valor 0xF desde la dirección 0x8

2.4. Advanced eXtensible Interface

Advanced eXtensible Interface 4 (AXI4) es la cuarta generación de la especificación de interfaz AMBA [8] de ARM. AMD ha adoptado AXI4 como su estándar de interconexión de IP de próxima generación para la familia de dispositivos UltraScale, 7 Series, Zynq 7000 y futuras generaciones, lo que garantiza un ecosistema consolidado de herramientas y componentes reutilizables en sus plataformas de diseño.

AXI está diseñado para sistemas de mayor desempeño y complejidad [14]. Su arquitectura se basa en la separación de la comunicación en canales independientes:

- dirección de escritura (AW)
- datos de escritura (W)
- respuesta de escritura (B)
- dirección de lectura (AR)
- datos de lectura (R)

Esta separación permite que cada canal funcione de forma concurrente, facilitando transferencias eficientes especialmente cuando se integran CPUs, memorias y aceleradores dentro de una misma arquitectura.

Cada canal opera de forma independiente mediante un mecanismo de *handshake* basado en las señales **VALID** y **READY**: quien transmite levanta **VALID** cuando la información es válida y quien recibe levanta **READY** cuando puede aceptarla; la transferencia se concreta en el flanco de reloj en que ambas señales están en alto. Una operación de lectura involucra dos canales: el maestro emite la dirección por AR y el esclavo responde con el dato y un código de estado por R, como se observa en la Figura 2.3. Una operación de escritura involucra tres canales: la dirección viaja por AW, el dato por W y el esclavo confirma la transferencia con un código de respuesta por B, como se ilustra en la Figura 2.4.

Sobre esta base común, la especificación AMBA define tres variantes de AXI orientadas a distintos perfiles de uso. AXI4 está pensado para transacciones de memoria de alto desempeño y soporta

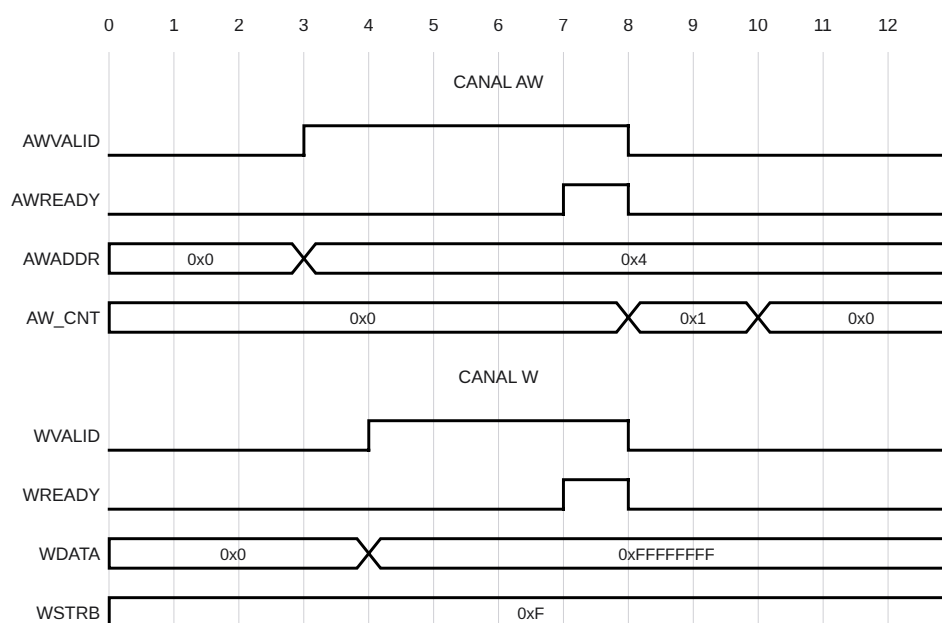


Figura 2.4: Transacción de escritura AXI sobre los canales AW y W. Se escribe 0xFFFFFFFF en la dirección 0x4

Tabla 2.1: Variantes del estándar AXI utilizadas en este trabajo.

Variante	Direccionamiento	Ráfagas	Uso típico
AXI4	Memory-mapped	Sí	Acceso a memoria externa, transferencias DMA
AXI4-Lite	Memory-mapped	No	Registros de control y configuración
AXI4-Stream	No (flujo continuo)	N/A	Video, DSP, cadenas de procesamiento

transferencias por ráfagas. AXI4-Lite es una versión simplificada, usada para acceder a registros de control y configuración de periféricos o aceleradores. AXI4-Stream prescinde de direccionamiento y se orienta a la transferencia continua de datos, lo que la hace adecuada para cadenas de procesamiento, módulos de video o aceleradores que operan sobre flujos. La Tabla 2.1 resume las características principales de cada variante.

La elección de la variante afecta directamente el diseño del sistema. Un acelerador controlado por registros se integra naturalmente mediante AXI4-Lite; uno que procesa grandes bloques de datos suele requerir AXI4 con acceso a memoria o una combinación de AXI4-Stream y un módulo DMA. El diseñador debe considerar, por lo tanto, no solo la funcionalidad del coprocesador, sino también el mecanismo de comunicación más adecuado para alimentarlo con datos, controlar su ejecución y recuperar sus resultados.

2.5. Uso de memoria en sistemas embebidos basados en FPGA

Los dispositivos FPGA disponen de bloques de memoria interna (denominados *Block RAM* o BRAM) distribuidos a lo largo del tejido programable. La cantidad total de BRAM de un dispositivo es fija y constituye un recurso compartido entre todos los bloques del sistema que requieren almacenamiento en chip: memorias de instrucciones y datos del procesador, FIFO de interconexión, búferes de periféricos y otros elementos similares. La BRAM ofrece acceso de baja latencia y ancho de banda elevado, pero su capacidad total resulta insuficiente para aplicaciones de mayor envergadura. Para superar esta limitación, las plataformas de desarrollo incorporan memoria externa al dispositivo; en el caso de la tarjeta Nexys A7 se trata de un módulo DDR2 que se conecta a la FPGA a través de un controlador implementado en la propia lógica programable. La jerarquía resultante define el conjunto de regiones físicas disponibles para alojar las distintas partes del sistema.

2.5.1. Mapa de memoria

Desde el punto de vista del procesador, las regiones físicas de memoria descritas anteriormente y los periféricos del sistema conviven dentro de un único espacio de direcciones plano. Este espacio (denominado mapa de memoria) asigna a cada recurso un rango de direcciones único, de modo que el procesador accede a todos ellos mediante las mismas instrucciones de carga y almacenamiento que utiliza para leer y escribir variables ordinarias. En la plataforma utilizada en este trabajo, el mapa

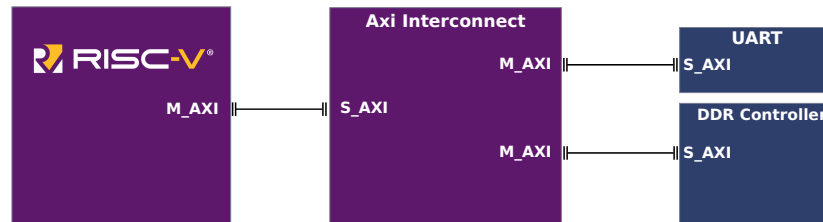


Figura 2.5: Diagrama *MicroBlaze V* conectado a un periférico UART y la memoria externa a través de AXI.

de memoria hace uso de la memoria externa DDR2 y los registros de los periféricos como se puede apreciar en la Figura 2.5.

La técnica que permite incorporar este último tipo de región recibe el nombre de mapeo de periféricos en memoria (*memory-mapped I/O*). En lugar de definir instrucciones dedicadas para el acceso a periféricos, los registros de control y de estado de cada bloque IP se exponen como direcciones del espacio de memoria del procesador. Una escritura sobre la dirección de un registro de control produce, internamente, una transacción sobre el bus AXI hacia el periférico, que altera su estado; análogamente, una lectura sobre un registro de estado retorna el valor actual de dicho registro. Desde código C, el acceso se realiza mediante punteros: el desarrollador declara un puntero **volatile** a la dirección base del periférico y manipula sus campos como si fueran variables comunes en memoria.

La asignación concreta de direcciones a cada bloque del sistema queda definida durante la integración del diseño en Vivado y se exporta junto con la descripción del hardware, de modo que el desarrollo del software pueda referirse a cada periférico a través de macros generadas automáticamente.

2.6. Nociones de desarrollo de firmware

El firmware es el software que se ejecuta directamente sobre el procesador embebido del sistema y se encarga de la inicialización del hardware, la coordinación entre los distintos componentes y la lógica de control de la aplicación. A diferencia del software de propósito general, el firmware opera sin sistema operativo o con uno mínimo, depende directamente de la arquitectura del hardware subyacente y se enfrenta a restricciones estrictas de memoria. Estas características hacen necesario comprender tanto la estructura del ejecutable producido por el compilador como su relación con la memoria física disponible.

2.6.1. Estructura del ejecutable ELF

El producto del proceso de compilación de una aplicación embebida es un archivo en formato ELF (*Executable and Linkable Format*). Este formato define una estructura binaria estandarizada [15] que organiza el código máquina, los datos inicializados, los símbolos de depuración y la información de relocalización en secciones independientes. Entre las secciones más relevantes para el análisis del tamaño del ejecutable se encuentran:

- **.text**: contiene el código ejecutable del programa.
- **.data**: aloja las variables globales y estáticas inicializadas con valores no nulos. Sus valores deben estar presentes en el binario para ser copiados a RAM durante la inicialización.
- **.bss**: reserva espacio para las variables globales y estáticas no inicializadas o inicializadas a cero. No ocupa espacio en el binario, pero sí en la memoria del sistema una vez cargado el programa.
- **.rodata**: agrupa las constantes de solo lectura, tales como cadenas literales y tablas declaradas como **const**.
- **.init**: contiene el código de inicialización que se ejecuta antes de **main**, incluyendo la copia de **.data** hacia RAM y la puesta a cero de **.bss**.

Como se ilustra en la Figura 2.6, el archivo ELF se compone de un encabezado principal que describe la estructura general del binario y referencia dos tablas: la tabla de encabezados de programa, utilizada por el cargador para llevar el ejecutable a memoria, y la tabla de encabezados de sección, utilizada por las herramientas de análisis y depuración para acceder a cada sección individual.

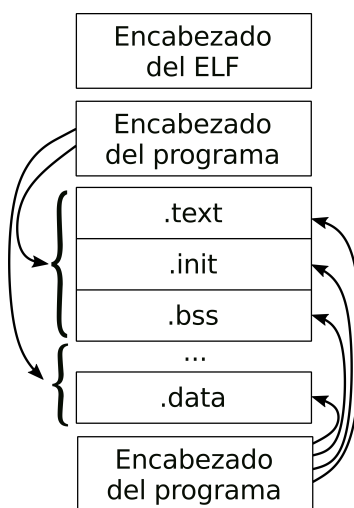


Figura 2.6: Estructura de un archivo ELF.

2.6.2. Restricciones de memoria

En un sistema embebido sin sistema operativo, el ejecutable ELF debe alojarse íntegramente en la memoria física del dispositivo. Esta exigencia implica que la suma de los tamaños de las secciones activas durante la ejecución se mantenga por debajo de la capacidad de la región del mapa de memoria asignada al ejecutable.

En la plataforma utilizada en este trabajo, las regiones candidatas para alojar el ejecutable son los bloques BRAM internos del dispositivo y la memoria DDR2 externa descritas en la Sección 2.5. La elección entre una y otra introduce un compromiso entre velocidad de acceso y capacidad: la BRAM ofrece tiempos de acceso del orden de uno o dos ciclos de reloj pero su tamaño total es limitado, mientras que la DDR2 dispone de capacidad de 128 MB, a costa de latencias del orden de diez a veinte ciclos y de la necesidad de un controlador adicional implementado en el FPGA. Estas restricciones adquieren especial relevancia en sistemas heterogéneos implementados sobre FPGA. Tanto la lógica programable como las memorias internas constituyen recursos finitos que deben repartirse entre el procesador, la infraestructura de comunicación y los aceleradores hardware. En consecuencia, el tamaño del firmware deja de ser únicamente una consideración de software y pasa a influir directamente en la arquitectura hardware que puede implementarse en el dispositivo.

Si el tamaño total del ejecutable excede la capacidad de la región de memoria asignada, el proceso de programación o inicialización puede fallar. Dependiendo de la configuración del sistema, esto puede manifestarse como errores reportados por las herramientas de compilación y enlace, bloqueos durante el arranque o fallos silenciosos que impiden la ejecución correcta de la aplicación.

METODOLOGÍA Y DESARROLLO

En este capítulo se presenta el proceso seguido para el desarrollo del material generado en este trabajo. A partir de los antecedentes expuestos en el Capítulo 2, particularmente los desafíos asociados al diseño de sistemas heterogéneos sobre FPGA, se realizó una revisión de la documentación oficial y de los recursos disponibles para el desarrollo de plataformas basadas en MicroBlaze V y aceleradores hardware. El análisis de este material permitió identificar vacíos tanto en la cobertura de ciertos conceptos como en la forma en que estos son presentados al usuario.

Sobre la base de estas observaciones y de los desafíos técnicos previamente identificados, se definieron los criterios de diseño utilizados para estructurar el material desarrollado. Posteriormente, se estableció una secuencia de ejemplos de complejidad creciente orientada a introducir progresivamente los distintos aspectos involucrados en la construcción de sistemas heterogéneos, desde la implementación de una plataforma base hasta la integración de aceleradores, memorias externas y herramientas de depuración.

Finalmente, se describe la organización de las guías desarrolladas y el proceso de validación utilizado para verificar tanto el funcionamiento técnico de los ejemplos como la utilidad del material como recurso de apoyo para futuros estudiantes y trabajos del área.

3.1. Análisis de material existente

La exploración del material existente partió de un alcance amplio: el desarrollo de sistemas heterogéneos sobre lógica reconfigurable utilizando un procesador basado en lógica reconfigurable de arquitectura RISC-V. Este alcance se acotó a MicroBlaze V, por dos razones. Primero, corresponde a la implementación RISC-V soportada de forma nativa por el flujo de herramientas de AMD, el mismo disponible en el Departamento y compatible con la placa Nexys A7-100T utilizada en este trabajo. Segundo, al estar integrado en dicho flujo, permite concentrar el estudio en la integración hardware-software y no en la puesta en marcha del procesador mismo.

Este acotamiento delimita el conjunto de documentación pertinente: la provista por AMD para las herramientas Vivado y Vitis. Dicha documentación no se organiza en torno al desarrollo de un sistema heterogéneo, sino en torno a cada herramienta por separado, por lo que no existe un documento único que sirva de punto de entrada. Frente a esta dispersión, se ordenaron las guías disponibles según la etapa del flujo de diseño que abordan y se seleccionó un documento por etapa. El resultado de este ordenamiento es el siguiente conjunto de guías de usuario:

- *Vitis High-Level Synthesis User Guide* (UG1399) [9].

- *Vivado Design Suite User Guide: Getting Started* (UG910) [12].
- *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118) [16].
- *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400) [13].
- *MicroBlaze V Processor Embedded Design User Guide* (UG1711) [17].

En la revisión de la documentación oficial se observó que, si bien esta describe de manera detallada las funcionalidades de cada herramienta, su enfoque está orientado principalmente a servir como referencia técnica. Como consecuencia, el proceso completo de desarrollo de un sistema heterogéneo queda distribuido entre múltiples documentos independientes.

Se identificaron las siguientes limitaciones:

- Cada guía aborda una etapa del flujo de forma aislada. La articulación entre etapas, que es precisamente donde reside la dificultad de un sistema heterogéneo, queda a cargo del lector.
- La extensión de los documentos, del orden de cientos de páginas, responde a un uso por consulta puntual y no a una lectura secuencial. Un lector que se aproxima por primera vez al flujo no dispone de un criterio para distinguir qué contenido le resulta necesario.
- Los documentos suponen un conocimiento previo del ecosistema de herramientas y de la terminología asociada. Conceptos como plataforma, dominio o *block design* se emplean sin definición, bajo el supuesto de que el lector ya opera dentro del flujo.
- Los tutoriales asociados describen una secuencia de pasos para alcanzar un resultado determinado, sin explicitar los criterios de diseño que justifican las decisiones tomadas.
- La documentación asume que el usuario reproduce el procedimiento de manera exacta. En consecuencia, entrega poca información sobre errores frecuentes, métodos de depuración o estrategias para identificar problemas cuando el resultado esperado no se obtiene.

Con el objetivo de encontrar material que abordara estas limitaciones, se revisaron además tutoriales, repositorios y proyectos desarrollados por terceros que implementaran sistemas heterogéneos sobre FPGA, particularmente aquellos basados en MicroBlaze V. Durante esta búsqueda se identificaron nuevas dificultades:

- Una gran parte del material disponible utiliza dispositivos de la familia Zynq, donde el procesador se encuentra integrado en silicio. Esto simplifica ciertos aspectos del diseño y dificulta la extrapolación de los ejemplos a plataformas basadas exclusivamente en lógica programable.
- Numerosos tutoriales fueron desarrollados utilizando versiones antiguas de las herramientas de AMD. Los cambios introducidos en Vitis Unified Platform modifican tanto la interfaz de usuario como diversos aspectos del flujo de trabajo, por lo que muchos procedimientos dejan de ser directamente aplicables.
- La mayoría de los proyectos encontrados se enfocan en la construcción de una aplicación final, omitiendo los problemas intermedios que suelen surgir durante el desarrollo y las estrategias utilizadas para resolverlos.

En conjunto, el material disponible presenta una fuerte fragmentación del conocimiento. Un estudiante que se inicia en el desarrollo de sistemas heterogéneos debe consultar múltiples fuentes, relacionar por cuenta propia conceptos pertenecientes a distintas herramientas y resolver de manera independiente los problemas que aparecen durante la implementación. Esta situación dificulta la construcción de una visión global del flujo de trabajo y aumenta considerablemente la barrera de entrada para nuevos usuarios.

Esta dificultad se ve acentuada por el carácter multidisciplinario del desarrollo sobre FPGA. La construcción de un sistema heterogéneo requiere combinar conocimientos de diseño digital, arquitectura de computadores, sistemas embebidos y desarrollo de software, áreas que habitualmente se estudian y ejercen de manera separada. Como consecuencia, un diseñador de hardware puede poseer experiencia en HDL y diseño de circuitos, pero carecer de familiaridad con herramientas de desarrollo de software embebido, depuración o construcción de aplicaciones. De forma análoga, un ingeniero orientado al desarrollo de firmware puede comprender la programación del procesador y la interacción con periféricos, pero no necesariamente los criterios asociados al diseño e integración de aceleradores en lógica reconfigurable. La documentación existente suele asumir que el lector ya dispone de estas competencias, sin ofrecer mecanismos que faciliten la transición entre ambas áreas.

A partir de la identificación de estas limitaciones se definieron los conceptos a abarcar utilizados para la elaboración de las guías propuestas en este trabajo.

3.2. Conceptos a abarcar

La estructura del material desarrollado se definió a partir de los desafíos técnicos asociados al diseño de sistemas heterogéneos sobre FPGA identificados en la Sección 2.1.4, entre ellos la partición hardware/software, la definición de interfaces para aceleradores, la organización de memoria, la sincronización entre procesador y aceleradores, y la validación del sistema. A estos desafíos se suman las limitaciones detectadas en el análisis del material existente presentado en la Sección 3.1, donde se observó la ausencia de un recorrido integrado que permitiera abordar estos conceptos de manera progresiva y dentro de un mismo flujo de desarrollo.

La definición de los contenidos también estuvo influenciada por la experiencia acumulada del autor en cursos de la línea de sistemas digitales, primero como estudiante y posteriormente como ayudante. Esta experiencia permitió identificar las principales dificultades que enfrenta un usuario al aproximarse por primera vez al desarrollo de sistemas embebidos sobre FPGA. Paralelamente, la retroalimentación continua del profesor guía contribuyó a determinar la profundidad y el énfasis con que debía abordarse cada tema.

A partir de estos antecedentes se definieron tres criterios fundamentales que constituyen el núcleo del material desarrollado. Estos criterios reúnen los conocimientos mínimos necesarios para que un lector pueda diseñar e implementar un sistema heterogéneo funcional sobre FPGA.

- **Construcción de un sistema base:** El primer criterio establece la necesidad de disponer de una plataforma de referencia sobre la cual desarrollar el resto del flujo. La integración inicial del procesador constituye una de las principales barreras de entrada para usuarios sin experiencia previa, por lo que resulta necesario presentar de forma explícita el proceso completo de creación de un sistema basado en MicroBlaze V. Este criterio busca que el lector comprenda la relación entre el diseño de hardware realizado en Vivado y el desarrollo de firmware en Vitis, estableciendo una plataforma reutilizable que servirá como base para las etapas posteriores.
- **Integración y utilización de periféricos existentes:** Una vez construida la plataforma base, el segundo criterio introduce los mecanismos de comunicación entre el procesador y el resto de los componentes del sistema. El uso inicial de periféricos ya disponibles permite concentrar la atención en aspectos como el acceso mediante registros mapeados en memoria y el manejo de interrupciones, sin incorporar todavía la complejidad asociada al diseño interno de nuevos bloques de hardware. De esta manera, el lector adquiere una comprensión gradual de la interacción procesador-periférico antes de abordar el desarrollo de componentes propios.
- **Diseño e integración de aceleradores personalizados:** El tercer criterio corresponde al elemento central del diseño heterogéneo: la incorporación de lógica especializada para acelerar tareas específicas. En esta etapa se introduce el desarrollo de periféricos y aceleradores mediante HDL, incluyendo su encapsulamiento como componentes compatibles con la infraestructura AXI del sistema. Adicionalmente, se presenta el flujo basado en High Level Synthesis como alternativa

de diseño, permitiendo contrastar distintos niveles de abstracción para la implementación de funcionalidades equivalentes. Este enfoque busca destacar que la distribución de funciones entre hardware y software constituye una decisión de diseño que puede materializarse mediante diferentes metodologías de desarrollo.

Los criterios anteriores permiten construir sistemas heterogéneos funcionales, pero no abarcan ciertos mecanismos que adquieren relevancia a medida que aumenta la complejidad del sistema o los requerimientos de transferencia de datos. Por esta razón se definieron dos criterios complementarios orientados a profundizar el flujo de desarrollo y aproximarlos a escenarios más representativos de aplicaciones reales. Estos contenidos se incorporan al final de la secuencia de aprendizaje, una vez consolidados los conceptos fundamentales.

- **Integración de recursos avanzados:** Este criterio incorpora elementos que amplían las capacidades del sistema más allá de la configuración mínima necesaria. La utilización de memoria externa DDR2 permite abordar arquitecturas con mayores requerimientos de almacenamiento, mientras que el acceso directo a memoria (DMA) introduce mecanismos eficientes para la transferencia de grandes volúmenes de datos entre aceleradores y memoria. En conjunto, estos recursos permiten desarrollar aplicaciones de mayor complejidad y exponen al lector a desafíos habituales en sistemas heterogéneos de uso práctico.
- **Depuración a nivel de hardware:** El último criterio complementario aborda técnicas de observación y validación del sistema desde la perspectiva del hardware. Su objetivo es proporcionar visibilidad sobre las transacciones que ocurren en la infraestructura de interconexión, permitiendo verificar el comportamiento de los distintos componentes y comprender el funcionamiento interno de los protocolos de comunicación utilizados. Este criterio complementa las herramientas de depuración de firmware introducidas previamente y completa el ciclo de validación del sistema.

La construcción de sistemas heterogéneos requiere combinar conceptos provenientes tanto del desarrollo de hardware como del desarrollo de software embebido. Introducir simultáneamente todos estos elementos incrementa significativamente la complejidad para un lector que se enfrenta por primera vez a este tipo de plataformas. Por este motivo, los criterios definidos anteriormente se materializan en una secuencia progresiva de guías, donde cada una incorpora un nuevo componente del flujo de desarrollo y reutiliza la infraestructura construida en etapas previas. La Sección 3.4 presenta la organización de esta secuencia y la relación entre sus distintas etapas.

3.3. Estructura de Guías

Respecto a los códigos asociados a este trabajo, en este capítulo solo se describen aspectos generales; los detalles específicos de implementación se documentan en el repositorio asociado.

El desarrollo siguió una metodología incremental. La primera etapa implementa una plataforma mínima capaz de ejecutar aplicaciones embebidas básicas. Las etapas posteriores añaden módulos aceleradores conectados al procesador mediante AXI, permitiendo delegar tareas al hardware dedicado. Finalmente, se realizaron pruebas funcionales y mediciones experimentales orientadas a validar la correcta operación conjunta entre software y hardware.

Cada sección del trabajo sigue la siguiente estructura:

- **Contexto:** Define el objetivo del ejemplo, identifica las partes de la placa que se utilizan y enumera los conceptos que el lector debe revisar. Sitúa el ejemplo dentro de la progresión de los tutoriales y fija su alcance.
- **Diseño de Hardware:** Describe paso a paso la construcción del diagrama de bloques que describe el sistema, identificando la funcionalidad de cada uno de los bloques, y el significado

de cada interconexión. Cuando el ejemplo lo requiere, incorpora además el diseño del periférico correspondiente.

- **Diseño de Firmware:** Analiza paso a paso la aplicación a ejecutarse en el procesador. Indica donde se inicializan los periféricos, donde se realizan escrituras y lecturas, entre otros.
- **Validación:** Presenta el resultado esperado junto con los recursos utilizados. Si bien las cifras exactas dependen de la versión de las herramientas y de la configuración de síntesis, las métricas de utilización (tablas de búsqueda (LUT), biestables (flip-flops), bloques de memoria (BRAM) y bloques DSP) sirven como indicador cuantitativo de que el diseño se realizó correctamente.

La combinación de un criterio de selección incremental y una plantilla editorial fija busca un doble objetivo: que cada guía se entienda y reproduzca por sí mismo, y que el conjunto conserve una narrativa coherente a lo largo de la progresión de ejemplos.

3.4. Definición preliminar de las guías

A partir de los focos de complejidad presentados en la Sección 3.2 y la plantilla editorial descrita en la Sección 3.3, se definió en conjunto con el profesor guía de manera preliminar 5 guías separadas en 3 tópicos, organizadas según el foco que abordan:

- **Sistema base.**
 - **Guía 1: Sistema base sobre MicroBlaze V.** Recorrido completo del flujo de diseño embebido, desde la descripción de hardware en Vivado hasta la ejecución de una aplicación simple sobre el procesador.
 - **Guía 2: Uso del depurador.** Validación de la aplicación a nivel de instrucción mediante el depurador de Vitis sobre la plataforma base.
- **Integración de periféricos disponibles en la placa.**
 - **Guía 3: Uso de periféricos e interrupciones externas.** Integración de los periféricos AXI4-Lite disponibles en la placa y manejo de interrupciones a través del AXI Interrupt Controller.
- **Diseño de periféricos y aceleradores propios.**
 - **Guía 4: Periférico personalizado en RTL.** Encapsulado de un módulo descrito en Verilog como periférico AXI4-Lite e integración al sistema base.
 - **Guía 5: Periférico personalizado en HLS.** Implementación del mismo ejemplo funcional de la guía anterior mediante High Level Synthesis, contrastando ambos flujos de descripción.

Las guías no fueron concebidas como actividades independientes, sino como componentes de una secuencia formativa común. Cada guía introduce un conjunto acotado de conceptos y reutiliza conocimientos adquiridos previamente, permitiendo que la complejidad del sistema aumente de manera gradual. Esta organización busca reducir la carga asociada a la incorporación simultánea de múltiples herramientas y tecnologías, favoreciendo una comprensión progresiva del flujo completo de desarrollo de sistemas heterogéneos. E

Para comprobar el correcto funcionamiento de las actividades descritas en cada guía, estas pasaron por un proceso de validación a nivel estructural y de funcionamiento. Este proceso es detallado en la siguiente sección.

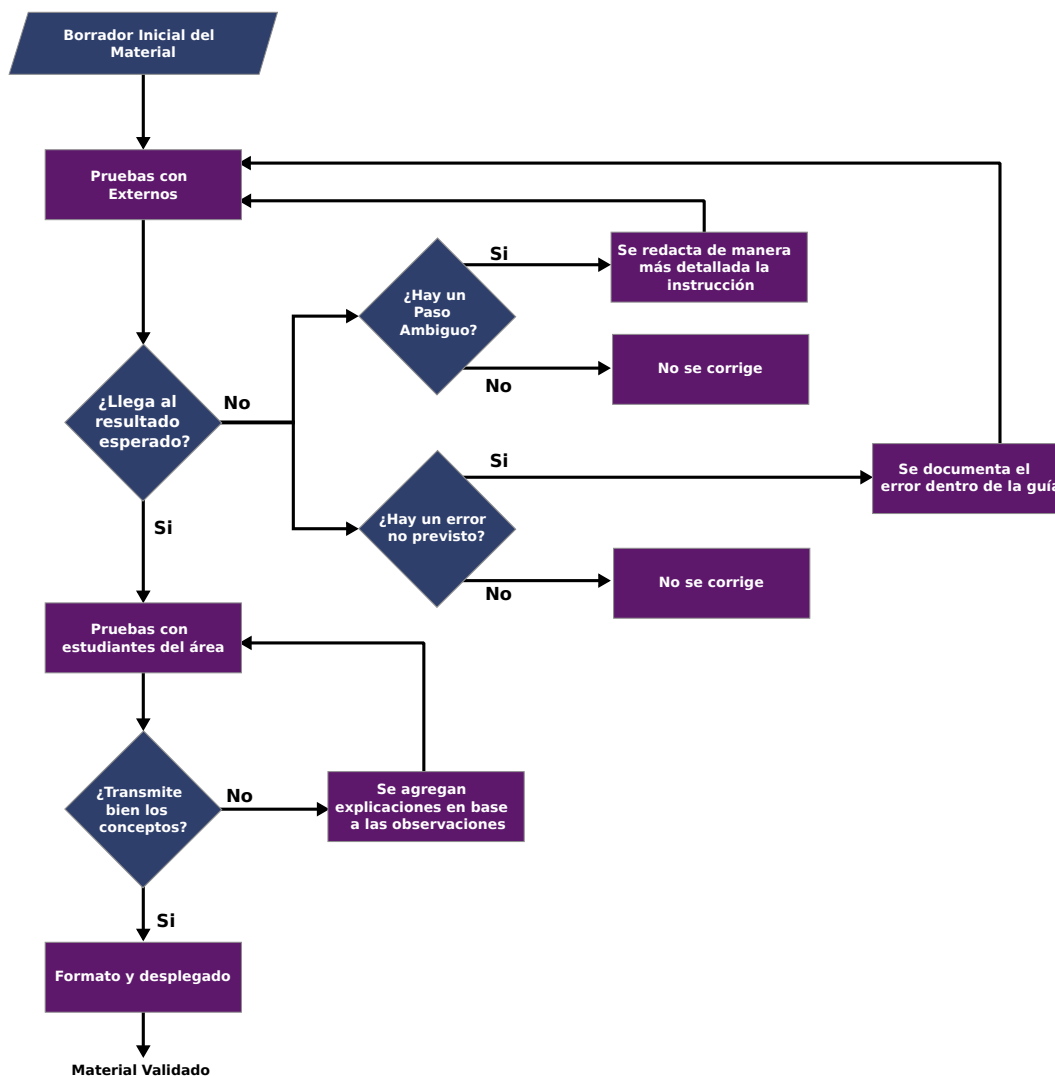


Figura 3.1: Diagrama de flujo de proceso de validación del material

3.5. Validación

La validación del material desarrollado se realizó de manera iterativa durante la elaboración de las guías y contempló tanto aspectos de reproducibilidad como de contenido técnico. El proceso se estructuró en dos etapas complementarias: una primera orientada a verificar que las guías pudieran ser ejecutadas por usuarios externos al proyecto, y una segunda enfocada en evaluar la profundidad técnica y la calidad de las explicaciones incorporadas.

El proceso también se puede apreciar en el diagrama de flujo visto en la Figura 3.1.

3.5.1. Primera etapa de validación

La primera etapa tuvo como objetivo evaluar la reproducibilidad de las guías desarrolladas. Para ello, cada guía fue puesta a prueba por dos usuarios externos al proyecto que no contaban con un dominio completo de los conceptos presentados en el Capítulo 2. La evaluación consistió en solicitar que estos usuarios reprodujeran los ejemplos siguiendo exclusivamente las instrucciones proporcionadas en las guías.

Esta instancia permitió identificar diversos problemas asociados a la claridad del material. Entre ellos destacaron pasos ambiguos, configuraciones omitidas, supuestos implícitos sobre conocimientos previos y diferencias entre el comportamiento esperado y el observado en determinadas versiones de las herramientas. Asimismo, durante el proceso surgieron errores que no habían sido detectados durante el desarrollo original de las guías, lo que permitió documentar su causa probable, su interpretación y mecanismos de resolución.

A partir de estas observaciones se realizaron múltiples correcciones al material, incorporando explicaciones adicionales, capturas de pantalla, diagramas de apoyo y secciones dedicadas a la resolución de problemas frecuentes. Estas modificaciones tuvieron como objetivo reducir la dependencia de conocimientos previos sobre Vivado y Vitis, facilitando que nuevos usuarios pudieran seguir el flujo de trabajo propuesto.

Una vez incorporadas estas correcciones, se solicitó nuevamente a los usuarios externos que realizaran el flujo desde el inicio tras lo cual los usuarios lograron completar satisfactoriamente todas las guías y obtener los resultados esperados en cada una de las actividades planteadas.

3.5.2. Segunda etapa de validación

Una vez verificada la reproducibilidad de los ejemplos, se llevó a cabo una segunda etapa de validación centrada en el contenido técnico de las guías. Esta revisión fue realizada junto al profesor guía y estudiantes con experiencia en el área de sistemas digitales y diseño sobre FPGA.

Durante esta etapa se analizaron las decisiones de diseño adoptadas en cada ejemplo, revisando tanto su corrección técnica como el nivel de detalle con que eran justificadas. Particularmente, se evaluó la forma en que se introducían conceptos como la integración de periféricos personalizados, el uso de interfaces AXI, la interacción entre hardware y software, la organización de memoria y la utilización de aceleradores implementados mediante síntesis de alto nivel.

Las observaciones obtenidas permitieron complementar las guías con explicaciones adicionales acerca de los criterios de diseño utilizados, fortaleciendo la relación entre los procedimientos prácticos y los fundamentos teóricos presentados en el documento. De esta manera, las guías pasaron de describir únicamente una secuencia de pasos para obtener un resultado funcional a explicar también las razones que motivan las decisiones tomadas durante el desarrollo de un sistema heterogéneo.

Como resultado de ambas etapas de validación, el material evolucionó hacia un conjunto de recursos que busca equilibrar la reproducibilidad de los ejemplos con la comprensión de los conceptos involucrados. El contenido fue refinado progresivamente durante el desarrollo del trabajo, incorporando la retroalimentación obtenida en cada iteración hasta alcanzar la versión final presentada en este informe.

MATERIAL DESARROLLADO

El entregable de esta memoria consiste en ocho guías, con su documentación y sus archivos fuente correspondientes, distribuidas en dos repositorios públicos. Las cinco primeras responden a los criterios que definen el núcleo mínimo del flujo de diseño heterogéneo, mientras que las tres últimas desarrollan los criterios complementarios planteados en la Sección 3.2.

El detalle técnico de cada guía no se reproduce en este informe: reside en un sitio web generado a partir del repositorio de documentación, accesible de forma directa al lector. Esta decisión responde a dos objetivos. El primero es evitar duplicar el contenido en dos formatos que tenderían a desincronizarse a medida que el material evolucione. El segundo es mantener el material como un recurso vivo, ya que el repositorio admite correcciones, ampliaciones y nuevas guías sin requerir una reedición del informe, mientras que un documento anexo quedaría congelado en la fecha de entrega.

Lo que sí corresponde exponer aquí es la vista de conjunto que ninguna guía individual entrega. La Figura 4.1 muestra qué guía introduce cada concepto y en cuáles se reutiliza. La lectura por columnas revela que cada guía agrega un único elemento nuevo al flujo de diseño, y la lectura por filas muestra que ese elemento persiste en las guías siguientes: el sistema base sostiene a las ocho, y los periféricos AXI4-Lite introducidos en la Guía 3 reaparecen hasta la Guía 8. Esta estructura acumulativa es el resultado más concreto del trabajo, porque permite que el lector enfrente una dificultad conceptual a la vez sin reconstruir el sistema desde cero en cada iteración.

El material desarrollado se encuentra disponible en el siguiente sitio web y repositorio: https://vicenteramirezp.github.io/MicroblazeV_Examples_Doc/ y los archivos fuente en https://github.com/vicenteramirezp/Tutorial_MicroBlazeV.

Las secciones siguientes describen los repositorios que componen el entregable y el proceso aplicado para validar la funcionalidad del material.

4.1. Detalle de cada guía

4.1.1. Sistema base sobre MicroBlaze V

Esta guía introduce el flujo completo de desarrollo de sistemas embebidos sobre FPGA. Comienza con una breve contextualización de la arquitectura RISC-V, su filosofía de diseño y su relevancia en el ecosistema actual de computación, justificando la elección de *MicroBlaze V* como procesador soft core para el resto del trabajo.

A continuación se presenta la arquitectura interna del procesador *MicroBlaze V*, esquematizada

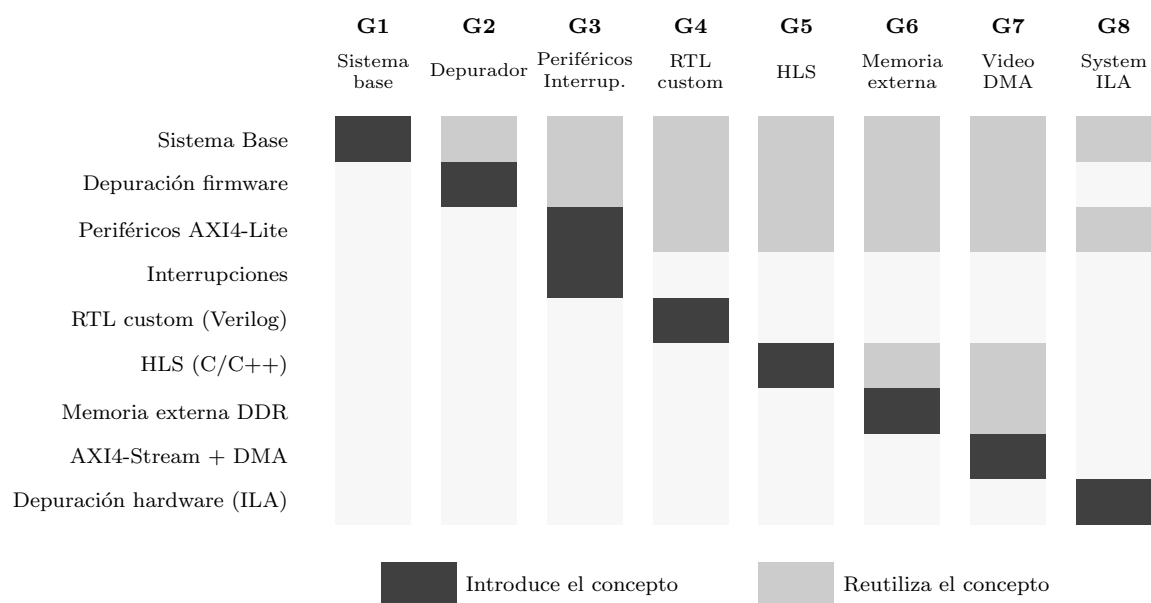
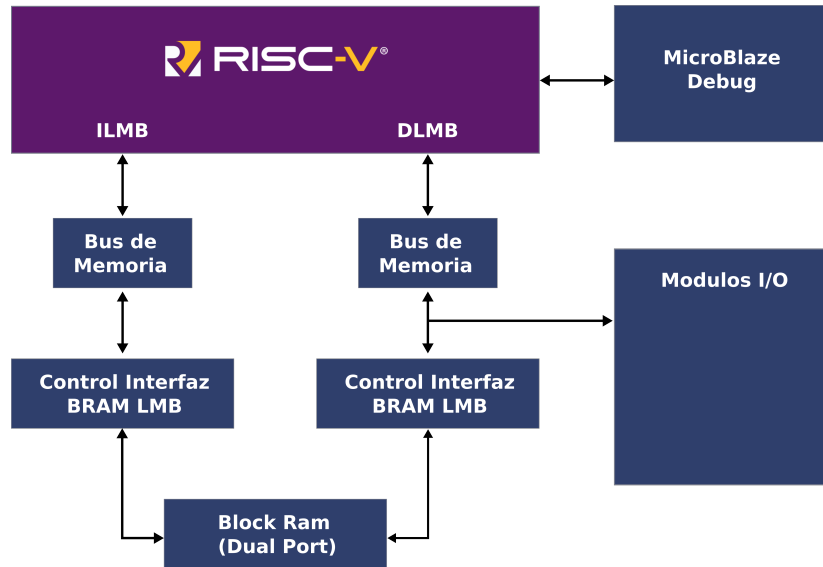
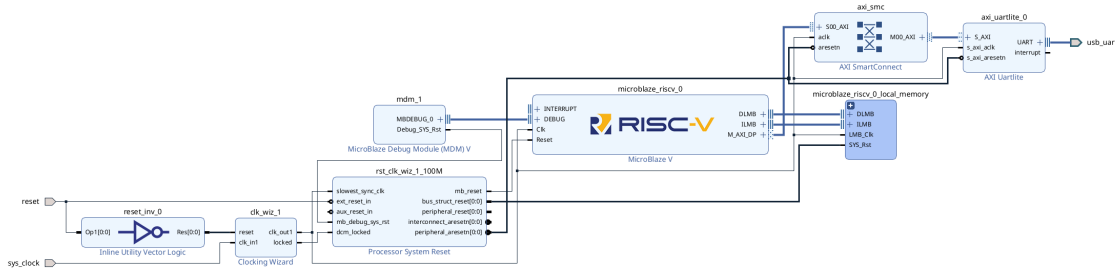


Figura 4.1: Conceptos cubiertos por cada guía. Las celdas oscuras marcan la guía que introduce un concepto; las claras, las guías que lo reutilizan.

Figura 4.2: Diagrama de implementación de *MicroBlaze V*.Figura 4.3: *MicroBlaze V* en el diagrama de bloques de Vivado.

en la Figura 4.2. Se describen los bloques que conforman el sistema base: procesador, buses ILMB y DLMB, controladores LMB-BRAM, memoria local en BRAM y el módulo de depuración MDM enfatizando el esquema que permite solapar el acceso a instrucciones y datos en un mismo ciclo de reloj.

Establecido el marco conceptual, se realiza el primer acercamiento a la herramienta de diseño de sistemas digitales basados en lógica reconfigurable: Vivado. Se ilustra cómo crear un proyecto, qué opciones entrega la herramienta y sus funcionalidades básicas. A continuación se aborda la primera iteración del diseño a nivel de hardware utilizando el editor de diagrama de bloques de Vivado como se ve en la Figura 4.3. Se explica cada paso y la justificación de las decisiones de diseño, incluyendo la importación del procesador, el uso de **Run Block Automation** para la generación automática de los bloques mínimos del sistema (memoria local, *Clocking Wizard* y *Processor System Reset*), y la asignación de los pines de la placa Nexys A7-100T mediante **Run Connection Automation**. Una vez validado el diagrama, se ejecutan las etapas de síntesis, implementación y generación de bitstream. Concluida esta etapa, el hardware se exporta como archivo Xilinx Support Archive (xsa), que encapsula la descripción del sistema para su uso posterior en el flujo de firmware.

Posteriormente se introduce al lector al uso de Vitis como herramienta de desarrollo de firmware. A partir del archivo **xsa** se crea el componente de plataforma, que abstrae el hardware y genera el **Board**

Support Package (BSP) con los controladores de los periféricos disponibles. Sobre esa plataforma se crea luego el componente de aplicación, que contiene el código C/C++ ejecutable sobre el procesador. Como verificación funcional del flujo, se compila y se carga sobre la FPGA una aplicación mínima de tipo *Hello World* que transmite por UART, observándose el resultado en un terminal serial.

Al completar esta guía, el lector ha recorrido el flujo completo Vivado–Vitis para un sistema embebido sobre *MicroBlaze V*, desde la descripción del hardware hasta la ejecución y verificación de una aplicación **bare metal** sobre el procesador. Si bien la aplicación desarrollada es deliberadamente simple, la infraestructura construida constituye la base sobre la que se montan las guías siguientes, en las que se incorporan progresivamente nuevos periféricos y mecanismos de interacción.

4.1.2. Uso del depurador

La Guía 2 utiliza la misma descripción de hardware de la Guía 1 y se concentra en la perspectiva de depuración de Vitis. Su aporte es metodológico: no modifica el diseño de hardware, sino que introduce las herramientas necesarias para verificar el comportamiento de una aplicación paso a paso, tanto a nivel de código fuente como de instrucciones de ensamblador.

Como caso de estudio se desarrolla una aplicación simple en C que implementa un bucle *for* dentro de una función independiente. Este programa permite observar el paso de parámetros por referencia, el acceso a memoria mediante punteros y la ejecución de estructuras iterativas a nivel de procesador.

A nivel de código fuente, la guía presenta las funcionalidades básicas del depurador: control de ejecución (*Step Over*, *Step Into*, *Step Out*, *Continue*), inspección de variables locales, definición de *breakpoints* y seguimiento del *Call Stack*. Con estas herramientas, el estudiante observa cómo evolucionan las variables y cómo se anidan las llamadas a funciones durante la ejecución.

A continuación, el análisis desciende a nivel de instrucción mediante la vista de *Disassembly* y el *Register Inspector*. Al cambiar la granularidad del depurador, el lector avanza instrucción por instrucción sobre código ensamblador RISC-V y verifica el efecto de cada operación sobre los registros del procesador como se ve en la Figura 4.4. Como ejercicio práctico se ejecuta una instrucción donde se guarda un valor en uno de los registros, tras lo cual se confirma, mediante el Register Inspector, la carga del valor.

Al completar la guía, el estudiante dispone de las herramientas necesarias para depurar sistemáticamente sus aplicaciones, correlacionar el código C con su traducción a ensamblador y comprender la interacción entre el firmware y la arquitectura del procesador. Estas capacidades serán fundamentales en las guías posteriores, donde la complejidad de las aplicaciones aumenta y la inspección a nivel de instrucción permite identificar errores que no son visibles desde el código fuente.

4.1.3. Uso de Periféricos e Interrupciones externas

Esta guía ilustra dos mecanismos fundamentales mediante los cuales un procesador interactúa con los periféricos integrados al sistema: el esquema de *memory-mapped I/O* y el sistema de interrupciones. Ambos son piezas centrales en cualquier sistema embebido moderno: el primero permite manipular el hardware con las mismas instrucciones que operan sobre memoria ordinaria, y el segundo libera al procesador de tener que sondear permanentemente a sus periféricos, permitiéndole reaccionar solo cuando ocurre un evento de interés.

A modo de ejemplo se incorpora un periférico GPIO asociado a los botones y LEDs de la placa, manteniendo el periférico UART de la guía anterior. Con esta configuración se muestra cómo ambos periféricos quedan expuestos dentro del mapa de memoria del procesador, dónde se encuentran sus direcciones base, cómo Vivado las asigna durante el diseño del hardware a través de la herramienta **Address Editor** como se ve en la Figura 4.5, y cómo el firmware accede a ellas desde Vitis para operarlos.

Sobre esa base se introduce el concepto de interrupción externa: en el contexto de los

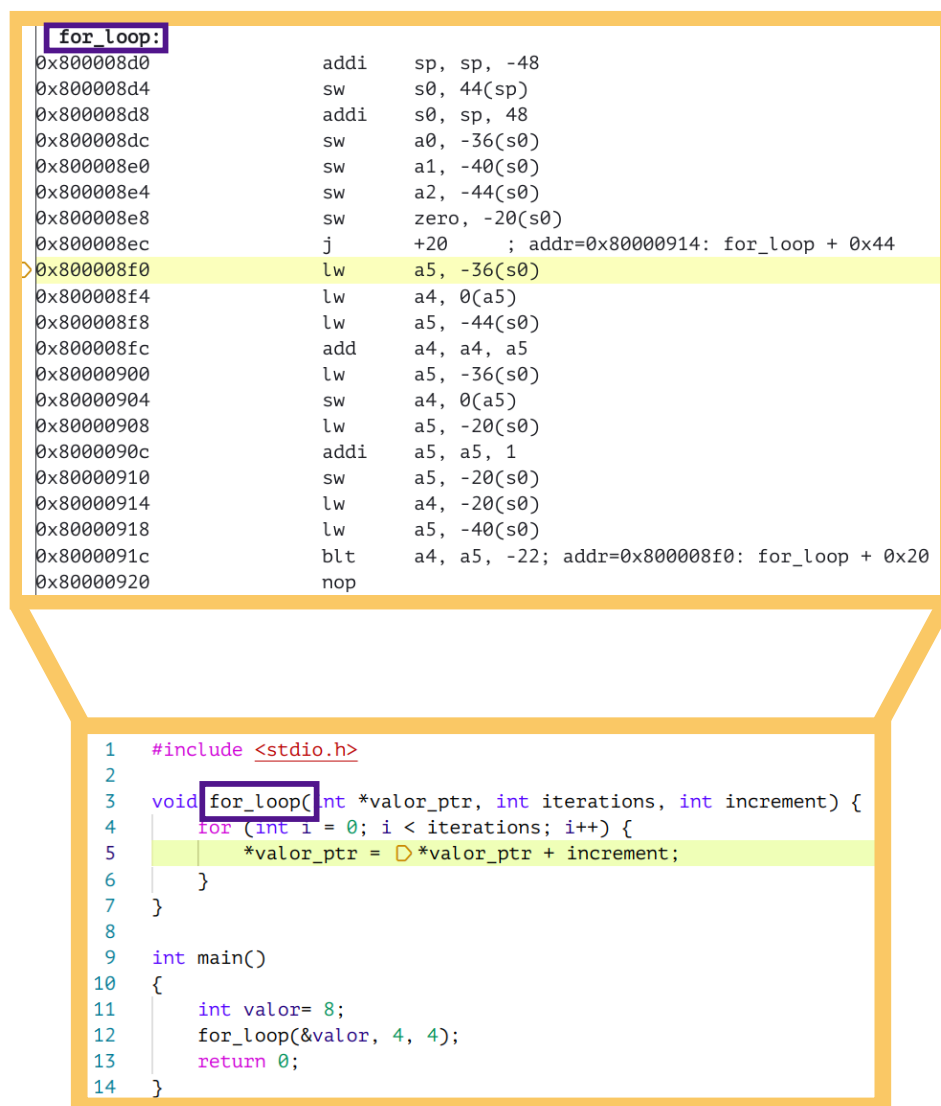


Figura 4.4: Vista de depuración de Vitis, con funciones y sus tags correspondientes en assembly en rojo.

The figure shows a screenshot of the Vivado Address Map tool. It displays a hierarchical view of the device's address space. The 'Address Map' tab is active, showing a table of components and their address ranges. Two components are highlighted with red boxes: '/axi_gpio_0/S_AXI' and '/axi_uartlite_0/S_AXI'. Both are registered as 'Reg' (Register) components with a 64K range. The table also shows other components like '/microblaze_riscv_0/Data' and '/microblaze_riscv_0/Instruction'.

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0 (/microblaze_riscv_0/Data)					
/microblaze_riscv_0					
/microblaze_riscv_0/Data (32 address bits : 4G)					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/microblaze_riscv_0_axi_intc/S_AXI	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
/microblaze_riscv_0_local_memory/dlmb_bram_if_cr/SLMB		Mem	0x0	128K	0x1_FFFF
Network 1 (/microblaze_riscv_0/Instruction)					
/microblaze_riscv_0					
/microblaze_riscv_0/Instruction (32 address bits : 4G)					
/microblaze_riscv_0_local_memory/dlmb_bram_if_cr/SLMB		Mem	0x0	128K	0x1_FFFF

Figura 4.5: Direcciones base de periféricos AXI Uart-Lite y AXI GPIO en Address Map de Vivado.

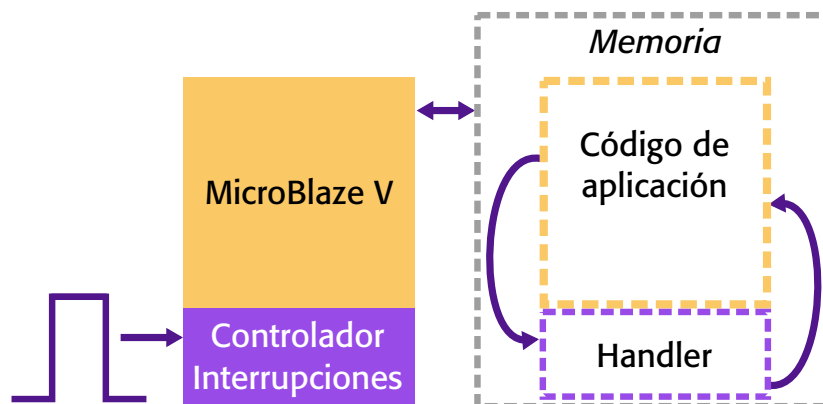


Figura 4.6: Diagrama de una interrupción.

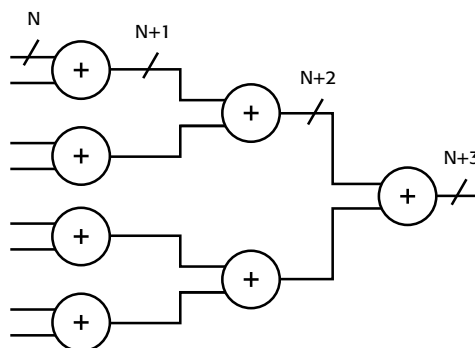


Figura 4.7: Diagrama de un Adder tree de 8 entradas de N bits.

microcontroladores, una señal que un dispositivo externo envía al procesador para desviar momentáneamente su flujo de ejecución y atender una condición, como se ilustra en la Figura 4.6. En esta guía cada pulsación de un botón gatilla una interrupción que el procesador atiende ejecutando una rutina de servicio (ISR), la cual responde enviando un mensaje por UART.

Para observar este mecanismo de forma dinámica se retoma el depurador introducido en la guía previa: colocando breakpoints al inicio de cada rutina de manejo, la ejecución se detiene cada vez que el procesador atiende una interrupción, permitiendo inspeccionar el estado del sistema en el instante exacto en que cambia de contexto.

Hasta este punto, las guías se enfocan en la construcción de sistemas embebidos basados en *MicroBlaze V* y en los mecanismos fundamentales de interacción con periféricos. A partir de la siguiente guía se introduce el desarrollo de hardware personalizado, incorporando coprocesadores implementados directamente en la lógica programable.

4.1.4. Periférico personalizado en RTL

Esta guía introduce el diseño de periféricos personalizados descritos en un lenguaje de descripción de hardware (HDL), específicamente SystemVerilog. A modo de ejemplo se utiliza un Adder Tree de 8 entradas de 16 bits, cuya estructura se aprecia en la Figura 4.7.

Se eligió el Adder Tree como el módulo a implementar porque ilustra una ventaja característica del cómputo en FPGA: procesa varias entradas en paralelo en una sola operación. Un procesador convencional, de naturaleza secuencial, no puede ejecutar esta operación en un solo ciclo.

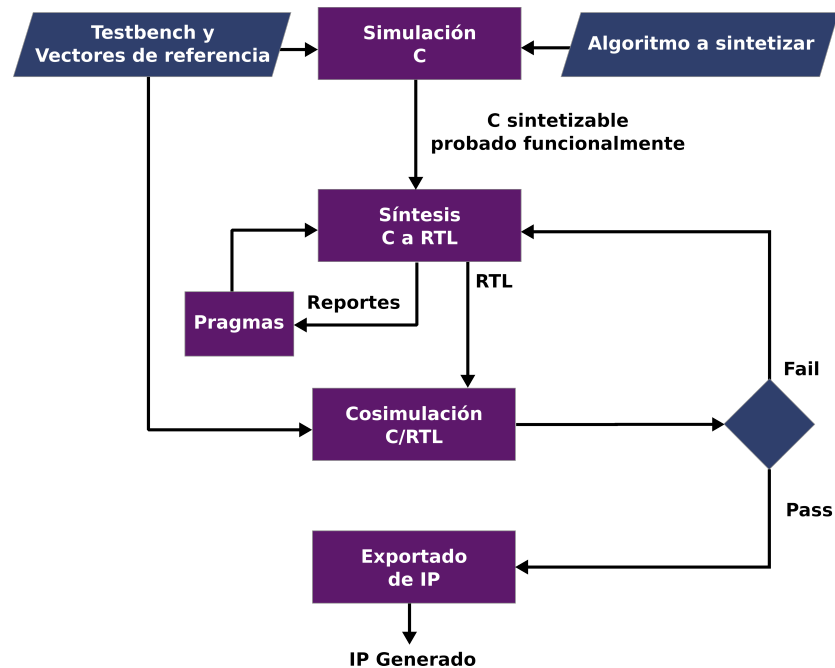


Figura 4.8: Diagrama de flujo de diseño periférico HLS, con paralelogramos representando insumos, rectángulos procesos y rombos toma de decisiones.

Para que el módulo HDL pueda comunicarse con el microprocesador, primero es necesario encapsularlo bajo un protocolo de interconexión compatible con el bus del sistema. Por esta razón la guía dedica una sección a describir, a nivel conceptual, el funcionamiento de la interfaz AXI y la jerarquía maestro–cliente que define las transacciones entre el procesador y los periféricos.

Sobre esta interfaz AXI, la guía muestra cómo empaquetar el módulo como una IP en Vivado: definir sus interfaces y mapear sus puertos internos a registros accesibles vía AXI.

A continuación, se aborda la generación de los drivers asociados al periférico recién empaquetado, de modo que sus registros queden accesibles desde el firmware mediante funciones en C. Con esto la guía continúa el ciclo de co-diseño hardware–software: una descripción RTL propia que termina siendo controlada desde un programa que corre sobre el MicroBlaze V.

4.1.5. Periférico personalizado en HLS

Se introduce el flujo de High-Level Synthesis (HLS) como alternativa al diseño de periféricos con lenguajes de descripción de hardware (HDL) tradicionales. HLS permite describir el comportamiento del módulo en C/C++ y delega a la herramienta la generación del RTL, abriendo un espacio de exploración arquitectural que resultaría impracticable escribiendo Verilog o VHDL a mano.

Para ilustrar este flujo, se sintetiza un Adder Tree funcionalmente equivalente al de la Guía 4, pero descrito en C++, siguiendo las etapas recomendadas por AMD.

- Simulación en C con testbench y vectores de referencia.
- Síntesis C→RTL.
- Cosimulación C/RTL.
- Empaquetado como IP para Vivado.

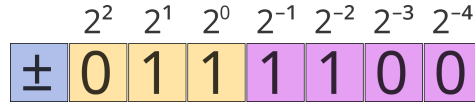


Figura 4.9: Representación en punto fijo (8 bits). Código de colores: azul para el bit de signo, amarillo para la parte entera (3 bits) y lila para la parte fraccionaria (4 bits).

En el diagrama visto en la Figura 4.8 se puede apreciar la metodología en más detalle.

A diferencia de la Guía 4, AMD ofrece un flujo expedito para el empaquetado del módulo: la herramienta consolida los archivos RTL, las restricciones y los drivers de software en un único paquete, sin requerir los pasos manuales descritos en la guía anterior.

Se introducen además los pragmas, directivas que modifican cómo se sintetiza una misma descripción en C sin alterar su comportamiento funcional. En particular, el pragma `HLS_UNROLL` replica el cuerpo del bucle en hardware, materializando sus iteraciones como lógica paralela en lugar de ejecutarlas de forma secuencial. Esta paralelización produce una variación significativa en la latencia del módulo y en su uso de recursos lógicos, evidenciando que una misma descripción en C puede sintetizarse en arquitecturas radicalmente distintas según las directivas aplicadas.

Una vez empaquetada, la IP se integra al sistema base con *MicroBlaze V* en Vivado, y desde el procesador se ejecuta firmware que la controla a través de su interfaz AXI4-Lite, utilizando los drivers que el propio flujo HLS genera. El resultado del cálculo se verifica por UART, mostrando que un algoritmo expresable en C puede materializarse como acelerador dedicado sin salir del entorno de diseño.

Una vez introducidas las metodologías RTL y HLS para la creación de aceleradores, el siguiente paso consiste en abordar uno de los principales desafíos de los sistemas heterogéneos: el manejo eficiente de grandes volúmenes de datos mediante memoria externa y mecanismos especializados de transferencia.

4.1.6. Memoria externa

Esta guía integra los conceptos de las guías previas e incorpora el uso de la memoria externa de la placa. El uso de esta memoria libera recursos lógicos del sistema (particularmente las BRAM) y permite ejecutar aplicaciones con un consumo de memoria mayor al disponible en lógica programable, es decir, un ELF de mayor tamaño.

El objetivo concreto es diseñar una IP en HLS que calcula el producto punto entre dos vectores de 10 elementos. La operación multiplica los vectores elemento a elemento y suma los productos, como describe la ecuación:

$$[A_1, A_2, \dots, A_{10}] \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_{10} \end{bmatrix} = A_1 B_1 + A_2 B_2 + \dots + A_{10} B_{10} \quad (4.1)$$

Para justificar el uso de la memoria externa, cada vector almacena valores en punto fijo de 16 bits. El punto fijo representa números fraccionarios asignando una cantidad fija de bits a la parte entera y otra cantidad fija a la parte fraccional [18], como se ilustra en la Figura 4.9. La librería `ap_fixed.h` de Vitis permite definir estos tipos al diseñar el periférico HLS, especificando el ancho total de la palabra y su división entre parte entera y fraccional.

Para que la aplicación maneje el mismo formato, debe incluir también `ap_fixed.h`. Adicionalmente, representar correctamente estos números al enviarlos por UART obliga a usar la librería `iostream`. Esta dependencia eleva el ELF a aproximadamente 717 KB, valor que supera ampliamente los 128 KB de BRAM disponibles y motiva el uso de la memoria externa.

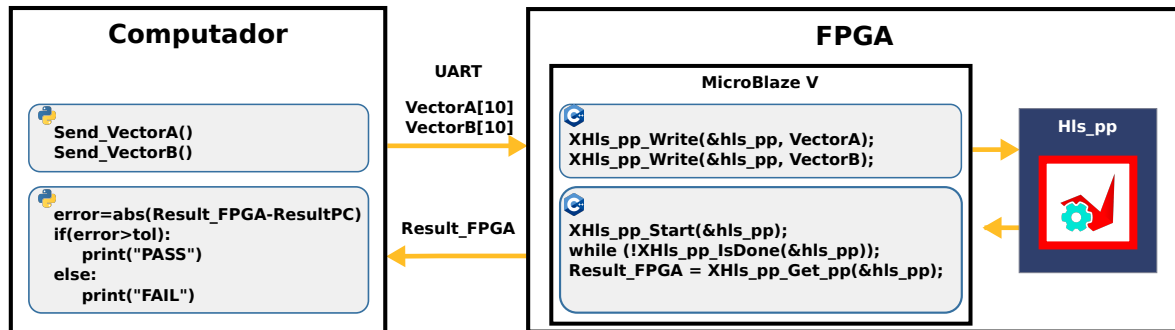


Figura 4.10: Diagrama de funcionamiento de la Guía 6.

La guía está organizada en cuatro etapas. Primero, se diseña en HLS el módulo que calcula el producto punto, siguiendo el procedimiento de la guía anterior. Luego, se introduce la IP *Memory Interface Generator* (MIG) [19], que genera la interfaz específica entre la lógica de la FPGA y el chip de memoria de la placa. A continuación, se integran al procesador la interfaz MIG y la IP de producto punto, y desde la perspectiva de Firmware se examina el mapa de memoria del sistema, prestando atención a la función del *Linker script*: el archivo que define dónde se almacenan las secciones de código generadas durante la compilación.

Finalmente, un script de Python ejecutado en el computador anfitrión detecta la placa, le envía los vectores y compara el resultado recibido con el valor calculado localmente, según el flujo de la Figura 4.10.

4.1.7. Acceso directo a memoria

Esta guía introduce el uso de acceso directo a memoria (DMA), un mecanismo ampliamente utilizado en sistemas embebidos para transferir grandes volúmenes de datos sin intervención continua del procesador. A diferencia de los periféricos estudiados previamente, donde las transacciones eran iniciadas explícitamente por software mediante accesos AXI memory-mapped, ciertas aplicaciones requieren transferencias sostenidas de información que exceden la capacidad práctica de la CPU para gestionar cada movimiento de datos.

Para abordar este problema se incorpora el módulo AXI VDMA [20], un controlador DMA capaz de operar como maestro sobre el bus AXI4. Una vez configurado por software, el módulo puede acceder de forma autónoma a la memoria DDR y transferir bloques de datos sin intervención adicional del procesador. Esta arquitectura permite desacoplar el procesamiento de las tareas de movimiento de datos, reduciendo la carga de la CPU y aprovechando de mejor manera el ancho de banda disponible del sistema.

La guía utiliza como caso de estudio la generación de video VGA, una aplicación que requiere el suministro continuo de datos a una tasa constante. En este contexto, el cuadro almacenado en DDR (referido como framebuffer) es leído por el VDMA y convertido en un flujo AXI4-Stream. A partir de este punto los datos recorren un camino de interconexión dedicado, independiente de las transacciones memory-mapped utilizadas por el procesador, atravesando los bloques de adaptación necesarios hasta llegar al subsistema de video. La organización general del sistema puede apreciarse en la Figura 4.11.

Además de introducir el uso de DMA, la guía permite estudiar la coexistencia de distintos tipos de interconexión AXI dentro de un mismo sistema. Mientras el procesador continúa utilizando transacciones AXI memory-mapped para configuración y control, el flujo de video emplea una ruta AXI4-Stream dedicada para transportar datos de manera continua entre bloques hardware.

En el nivel de software se implementa una estrategia de doble framebuffer. Mientras el VDMA lee una imagen desde memoria para su visualización, la aplicación genera el siguiente cuadro en una región distinta de DDR. Al finalizar cada trama ambos buffers intercambian sus roles, permitiendo actualizar

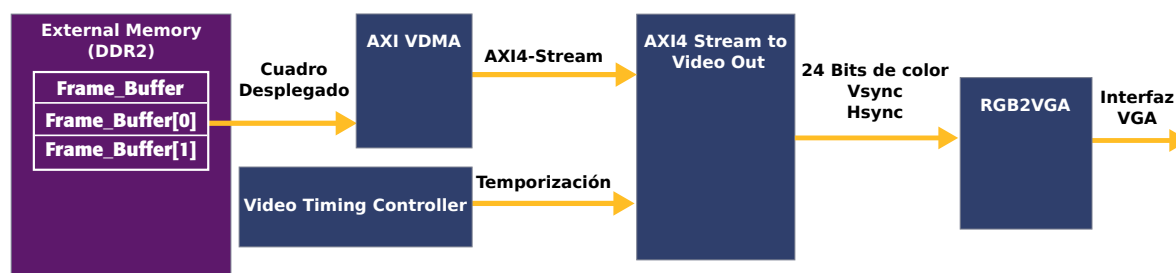


Figura 4.11: Secuencia desde la obtención del cuadro hasta su salida en la interfaz VGA

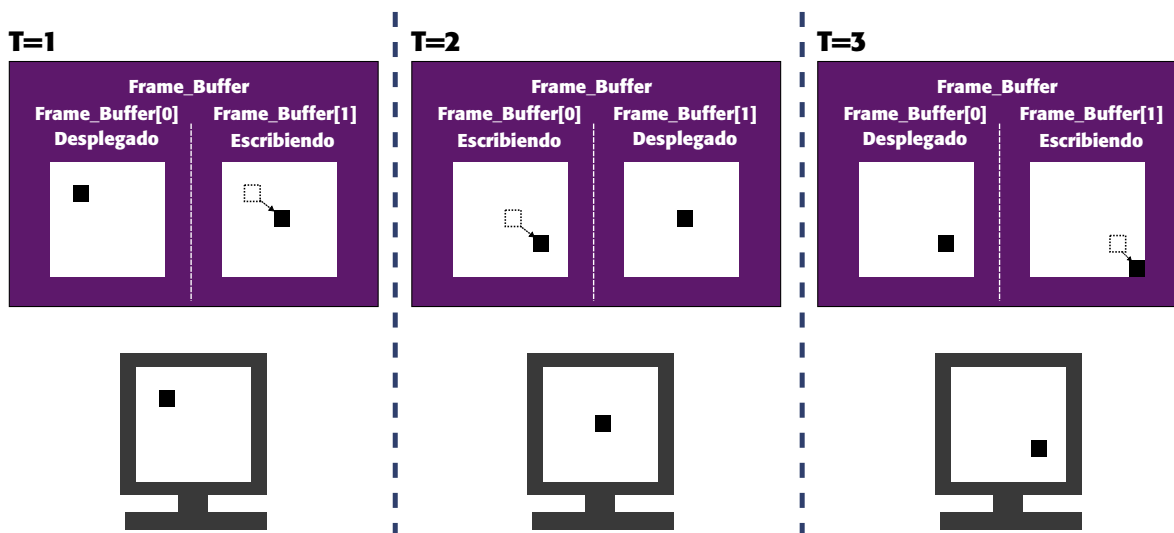


Figura 4.12: Secuencia de 3 tramas en un monitor.

la imagen sin interferir con la transferencia en curso. La secuencia de funcionamiento se muestra en la Figura 4.12.

Finalmente, la última guía incorpora herramientas de depuración a nivel de hardware, permitiendo observar el comportamiento interno del sistema y validar la correcta operación de los mecanismos estudiados previamente.

4.1.8. Depuración a nivel de hardware

Esta guía aborda la depuración de transacciones a nivel de bus en sistemas que utilizan AXI4-Lite. Mientras que las guías anteriores emplean este protocolo de forma transparente mediante las bibliotecas y controladores provistos por Xilinx, en esta etapa se profundiza en los mecanismos de comunicación subyacentes, permitiendo observar cómo las operaciones ejecutadas por el procesador se traducen en transacciones sobre el bus. Para ello se introduce el bloque **System ILA** (*Integrated Logic Analyzer*), el cual permite capturar y analizar señales internas del sistema durante su ejecución.

El objetivo de la guía es construir un sistema basado en *MicroBlaze V* que integre dos periféricos AXI4-Lite: **AXI GPIO**, asociado a los 16 LEDs y 16 interruptores de la tarjeta Nexys A7, y **AXI Timer**. El bloque **System ILA** se conecta a ambas interfaces para monitorear simultáneamente la actividad generada por el procesador, permitiendo visualizar las transacciones correspondientes a los cinco canales del protocolo AXI4-Lite: dirección de escritura (**AW**), datos de escritura (**W**), respuesta (**B**), dirección de lectura (**AR**) y datos de lectura (**R**). De esta forma, el estudiante puede relacionar la estructura conceptual del protocolo con las señales observadas directamente en hardware.

Sobre esta plataforma se desarrolla una aplicación en C utilizando Vitis que ejecuta una secuencia representativa de accesos al sistema: inicialización de los periféricos, configuración de los canales GPIO, escritura de un patrón fijo en los LEDs, lectura del estado de los interruptores, arranque del temporizador y consultas a su valor actual. Cada una de estas operaciones genera una o varias transacciones AXI4-Lite que pueden ser capturadas por el **System ILA**, permitiendo relacionar las llamadas realizadas desde software con la actividad observada sobre la interconexión.

La verificación combina la depuración paso a paso desde Vitis con el modo de auto-gatillado del ILA en Vivado, configurando **WVALID** y **RREADY** como señales gatillantes en flanco de subida. Para cada línea ejecutada, el estudiante contrasta las direcciones y datos observados sobre el bus con los mapas de registros oficiales de **AXI GPIO** y **AXI Timer** como se puede apreciar en la Figura 4.13, cerrando el lazo entre la abstracción ofrecida por el controlador de periféricos y la transferencia concreta de bytes

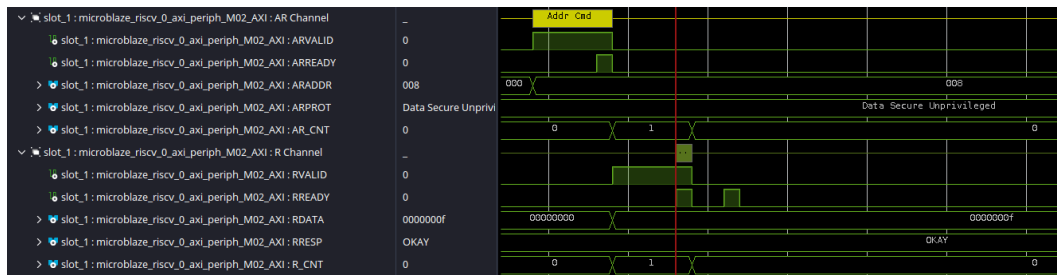


Figura 4.13: Transacción de lectura de los interruptores de la placa Nexys A7 visualizado a través de la perspectiva de depuración de Vivado. Se lee el valor 0xF del registro de estado de AXI GPIO con dirección 0x08.

que ocurre sobre la interconexión.

4.2. Listado de material

El material se distribuye en dos repositorios alojados bajo la cuenta del autor de este trabajo: uno contiene la documentación de las ocho guías y el otro reúne los archivos fuente necesarios para reproducir cada una. El repositorio se encuentra con acceso abierto al momento de la realización de este trabajo. El detalle de cada repositorio se presenta en las dos siguientes subsecciones.

4.2.1. Repositorio de archivos fuente y de documentación

En el repositorio de archivos fuente, el cual al momento de la entrega de este trabajo se encuentra disponible en el enlace: https://github.com/vicenteramirezp/Tutorial_MicroBlazeV es donde se guardan los insumos para realizar cada una de las actividades descritas, separadas en carpetas correspondiente a cada guía.

El repositorio de documentación constituye el principal entregable de este trabajo. Al momento de la entrega de este trabajo, el repositorio se encuentra alojado en Github en la dirección: [github.com/ vicenteramirezp/MicroblazeV_Examples_Doc](https://github.com/vicenteramirezp/MicroblazeV_Examples_Doc), y el sitio desplegado a partir de esta documentación se puede visualizar en el enlace vicenteramirezp.github.io/MicroblazeV_Examples_Doc. En él se documentan ocho guías diseñadas para introducir progresivamente al lector en el flujo de desarrollo de sistemas de cómputo heterogéneo basados en *MicroBlaze V* y lógica reconfigurable. La secuencia de actividades fue estructurada con complejidad incremental, comenzando desde la creación y depuración de aplicaciones embebidas hasta la integración de aceleradores hardware desarrollados mediante HDL y High-Level Synthesis.

Cada guía incorpora el contexto conceptual necesario para comprender los mecanismos involucrados, instrucciones detalladas para reproducir el procedimiento y criterios de validación que permiten verificar la correcta ejecución de los resultados obtenidos. De esta forma, el repositorio no sólo documenta ejemplos funcionales, sino que además proporciona una ruta de aprendizaje reproducible para estudiantes e investigadores que deseen familiarizarse con el desarrollo de plataformas heterogéneas sobre dispositivos FPGA.

La documentación se publica mediante GitHub Pages, permitiendo mantener una única fuente de información permanentemente actualizable. Esta estrategia evita la duplicación de contenido entre el informe escrito y la documentación técnica, facilitando además la incorporación futura de nuevas guías, correcciones y extensiones sin alterar el documento de memoria.

La Figura 4.14 muestra la interfaz del sitio web generado a partir de la documentación. La estructura de navegación permite acceder a las distintas guías, recursos complementarios y material de referencia desde una única plataforma, simplificando la exploración del contenido y favoreciendo su utilización

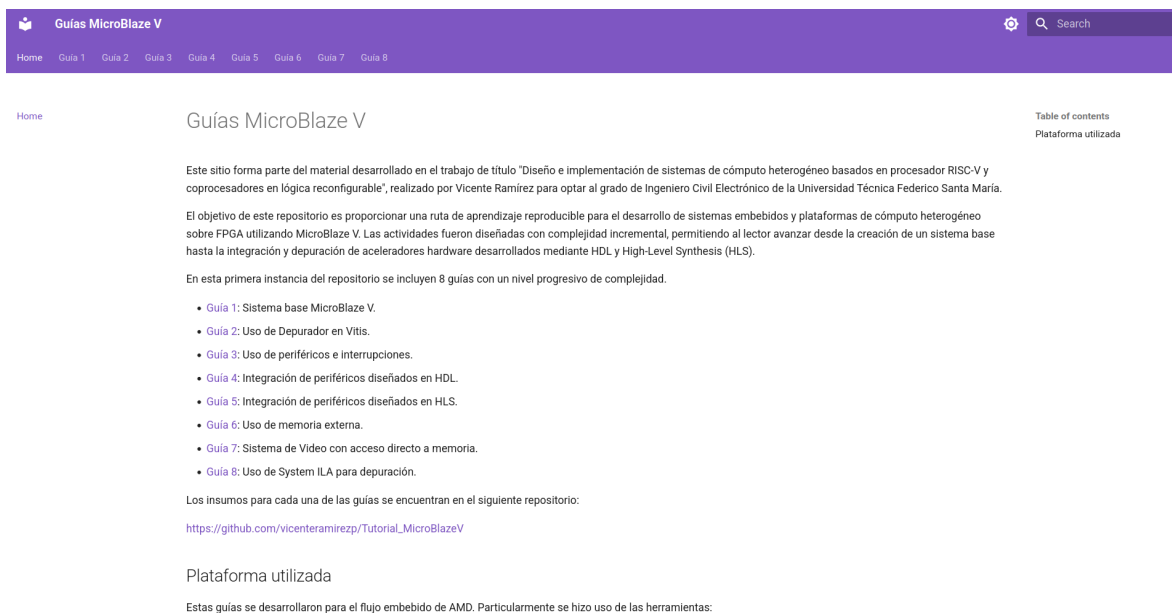


Figura 4.14: Página inicial asociada al repositorio de documentación

como recurso de apoyo para futuras actividades de docencia e investigación.

CONCLUSIONES Y TRABAJO FUTURO

Este capítulo sintetiza los resultados obtenidos en el presente trabajo de título, identifica los aprendizajes alcanzados y reconoce las limitaciones del desarrollo realizado. Después se plantean los posibles trabajos futuros en base al material generado, continuando en la línea de sistemas heterogéneos.

5.1. Conclusiones

Este trabajo presentó el análisis, diseño y desarrollo de material didáctico sobre cómputo heterogéneo en FPGA utilizando el procesador *MicroBlaze V*, con el propósito de servir como base para futuras actividades docentes y trabajos de título en el Departamento de Electrónica de la Universidad Técnica Federico Santa María. El desarrollo abarcó cuatro etapas: diagnóstico del material existente, definición de criterios pedagógicos y técnicos para las guías, construcción de un conjunto de ocho ejemplos progresivos sobre la placa Nexys A7-100T y validación del material en dos escenarios complementarios.

La exploración inicial reveló vacíos relevantes en los recursos disponibles para el flujo embebido sobre FPGA con *MicroBlaze V*. Si bien el ecosistema de AMD provee documentación extensa, esta rara vez organiza su contenido en torno a la integración incremental entre hardware y software, y los tutoriales externos suelen apoyarse en versiones desactualizadas de las herramientas o en plataformas distintas a la Nexys A7. Este diagnóstico delimitó el alcance del trabajo y motivó la elaboración de un conjunto de guías con un hilo conductor único.

Tres decisiones de diseño resultaron determinantes en la cohesión del material. La primera fue adoptar el codiseño hardware–software como hilo conductor, de modo que cada guía vinculara explícitamente su aporte en hardware con una contraparte en software embebido. La segunda fue introducir cada concepto una sola vez y reutilizarlo en las guías posteriores, evitando la duplicación que caracteriza al material disperso disponible en línea. La tercera fue fijar una estructura editorial común a todas las experiencias, lo que dio al lector una expectativa estable sobre dónde encontrar cada tipo de información. Bajo estos criterios, la progresión que parte del sistema base de la Guía 1 y culmina con la depuración a nivel de bus de la Guía 8 emergió de forma natural.

La validación del material en dos rondas complementarias confirmó el objetivo didáctico planteado. En la primera ronda, dos estudiantes ajenos a la línea de cómputo heterogéneo siguieron las guías sin asistencia y reportaron los puntos en que se detuvieron; sus observaciones motivaron revisiones editoriales que consolidaron la redacción actual. En la segunda ronda, el profesor guía y estudiantes

activos en la línea de sistemas digitales evaluaron el material desde la pertinencia técnica y la utilidad didáctica. En ambas rondas los participantes alcanzaron los resultados esperados, lo que evidencia simultáneamente la corrección funcional del material y su accesibilidad para un lector que se aproxima por primera vez al flujo de diseño sobre *MicroBlaze V*.

Este trabajo genera una ruta de aprendizaje reproducible y actualizada para el desarrollo de sistemas heterogéneos basados en *MicroBlaze V*, reduciendo la barrera de entrada identificada al inicio de esta memoria y proporcionando una base concreta para futuros desarrollos académicos en el área.

5.2. Recomendaciones para trabajo futuro

A continuación se presentan recomendaciones para dar continuidad al trabajo desarrollado en esta memoria.

- **Extensión del repositorio de guías.** El repositorio se concibe como una biblioteca dinámica, por lo que se recomienda continuar incorporando guías que cubran funcionalidades aún no abordadas, tales como el uso de protocolos adicionales (SPI, I²C, Ethernet), aceleradores específicos diseñados con HLS o el aprovechamiento de los recursos DSP de la FPGA.
- **Integración de sistemas operativos.** Las guías presentadas operan en un esquema **bare metal**, esquema que pone en evidencia el flujo de datos hardware–software pero deja fuera la gestión de procesos, controladores en el kernel y comunicación entre espacio de usuario y hardware acelerado. Habilitar la ejecución de un sistema operativo sobre *MicroBlaze V*, ya sea un RTOS como Zephyr para aplicaciones de tiempo real o una distribución Linux generada mediante Yocto, sería el siguiente paso natural. Este ascenso en el nivel de abstracción extendería el alcance del material a estos tópicos sin reescribir las guías existentes.
- **Integración en actividades docentes.** El material generado se presta para ser incorporado a actividades didácticas, particularmente al curso de postgrado de la línea. Se sugiere validar el material en un ciclo completo de curso, recolectar retroalimentación estructurada de los estudiantes y ajustar la profundidad y el ritmo de las guías a partir de esa experiencia. Las herramientas y conceptos explorados en estas guías son particularmente relevantes para el curso IPD-432: diseño avanzado de sistemas digitales.

BIBLIOGRAFÍA

- [1] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, “FPGA HLS today: Successes, challenges, and opportunities,” *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 4, pp. 1–42, Aug. 2022.
- [2] J. L. Hennessy and D. A. Patterson, “A new golden age for computer architecture,” *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, Feb. 2019.
- [3] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 4th ed. Springer, 2021.
- [4] S. Pasricha, “Embedded systems education in the 2020s: Challenges, reflections, and future directions,” in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI ’22)*. ACM, 2022, pp. 533–538.
- [5] D. Chaver, S. Harris, L. Piñuel, O. Kindgren, R. Kravitz, J. I. Gómez, F. Castro, K. Olcoz, J. Villalba, A. Grinshpun, F. Gabbay, L. Seed, R. Duarte, M. López, O. Alonso, and R. Owen, “Teaching experiences using the RVfpga package,” arXiv preprint arXiv:2411.14954, 2024.
- [6] A. N. Cortés Neira, “Implementación de aceleradores de cómputo en lógica reconfigurable para aplicaciones de control predictivo por modelo utilizando síntesis de alto nivel,” Tesis de Magíster, Universidad Técnica Federico Santa María, Valparaíso, Chile, mar 2024.
- [7] J. J. Vázquez Cádiz, “Implementación en fpga de redes neuronales artificiales aplicadas a lazos de control predictivo por modelo,” Tesis de Magíster, Universidad Técnica Federico Santa María, Valparaíso, Chile, aug 2025.
- [8] ARM Ltd., *AMBA AXI and ACE Protocol Specification*, ARM Ltd., 2013, standard ARM IHI 0022E.
- [9] Advanced Micro Devices, Inc., *Vitis High-Level Synthesis User Guide*, Advanced Micro Devices, Inc., 2025, uG1399.
- [10] K. Asanović and D. A. Patterson, “Instruction sets should be free: The case for RISC-V,” EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-146, Aug. 2014.
- [11] S. Lahti, P. Sjövall, J. Vanne, and T. D. Härmäläinen, “High-level synthesis for FPGAs: a hardware engineer’s perspective,” *IEEE Access*, vol. 13, pp. 27 530–27 553, 2025.
- [12] Advanced Micro Devices, Inc., *Vivado Design Suite User Guide: Getting Started*, Advanced Micro Devices, Inc., 2025, uG910.
- [13] —, *Vitis Unified Software Platform Documentation: Embedded Software Development*, Advanced Micro Devices, Inc., 2025, uG1400.
- [14] ARM Ltd., *AMBA AXI4-Stream Protocol Specification*, ARM Ltd., 2010, standard ARM IHI 0051A.

-
- [15] Tool Interface Standard (TIS) Committee, *Executable and Linkable Format (ELF) Specification, Version 1.2*, May 1995.
- [16] Advanced Micro Devices, Inc., *Vivado Design Suite User Guide: Creating and Packaging Custom IP*, Advanced Micro Devices, Inc., 2025, uG1118.
- [17] —, *MicroBlaze V Processor Embedded Design User Guide*, Advanced Micro Devices, Inc., 2025, uG1711.
- [18] R. Yates, “Fixed-point arithmetic: An introduction,” Digital Signal Labs, Tech. Rep., Jul. 2020.
- [19] Advanced Micro Devices, Inc., *MIG 7 Series DDR2/DDR3 Product Guide*, Advanced Micro Devices, Inc., 2025, pG150.
- [20] —, *AXI Video Direct Memory Access v6.3 Product Guide*, Advanced Micro Devices, Inc., 2025, pG020.