

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)



Trabajo Fin de Máster

Estudio, análisis y experimentación con núcleos RISC-V para SoC FPGA

Autor: Romarey Lourdes Santos Ruiz
Tutor: Dr. Pedro Pérez Carballo
Fecha: Septiembre 2025

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)



Trabajo Fin de Máster

**Estudio, análisis y experimentación con núcleos RISC-
V para SoC FPGA**

HOJA DE FIRMAS

Alumno/a: Romarey Lourdes Santos Ruiz Fdo.:

Tutor/a: Dr. Pedro Pérez Carballo Fdo.:

Fecha: Septiembre 2025

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)



Trabajo Fin de Máster

Estudio, análisis y experimentación con núcleos RISC- V para SoC FPGA

HOJA DE EVALUACIÓN

Calificación:

Presidente Dr. Juan Antonio Montiel Nelson Fdo.:

Secretario Dr. Luis Hernández Acosta Fdo.:

Vocal Dr. Roberto Sarmiento Rodríguez Fdo.:

Fecha: Septiembre 2025

Resumen

El trabajo presentado realiza un análisis de la arquitectura RISC-V, destacando sus principales aspectos, entre ellos su ISA abierta, modular y libre. RISC-V se presenta como una alternativa a las arquitecturas propietarias. Ello fomenta la posibilidad de innovar en el campo de las arquitecturas de ordenador, reduciendo costos, reduciendo el consumo de potencia y mejorando la seguridad, aspectos claves para las aplicaciones actuales. RISC-V se originó hace 15 años en la Universidad de California, Berkeley, y la primera versión de la ISA se publicó en mayo de 2011. La flexibilidad de la ISA ha impulsado su adopción en un amplio rango de aplicaciones, desde soluciones muy simples para dispositivos IoT hasta aplicaciones de HPC en centros de datos. RISC-V está recibiendo una gran aceptación en la industria, como es el caso de NVIDIA que recientemente ha publicado sus avances en la integración de RISC-V en sus sistemas de procesamiento basado en GPU.

El diseño de RISC-V, se apoya en los principios de RISC (*Reduced Instruction Set Computer*), utilizando instrucciones simples y eficientes, con una coordinación eficiente entre la arquitectura *hardware* y el compilador. La arquitectura base, RV32I, soporta enteros de 32 bits y puede ser extendida para dar soporte a otros tipos de datos más complejos (punto flotante, por ejemplo) y a instrucciones más costosas como por ejemplo la división. Para ello utiliza distintos tipos de formatos de instrucción con una decodificación muy regular, la gestión de memoria en formato *little-endian* y una la estructura de registros coherente con el modelo de procesamiento. Además, incluye distintos modos de funcionamiento en cuanto al nivel de privilegios (Modo Usuario, Modo Supervisor, y Modo Máquina) que garantiza la protección del sistema en función del tipo de aplicación (*baremetal*, RTOS, Linux, etc.).

La estructura modular de la ISA de RISC-V, está soportada en el mecanismo de extensiones que permite añadir nuevas instrucciones en función de los requerimientos del caso de uso del procesador. Por ejemplo, si se precisan operaciones de multiplicación, se incluye la extensión necesaria, lo cual tiene impacto en las prestaciones de la arquitectura. RISC-V *International Foundation* ha establecido un mecanismo de estandarización de las extensiones, identificando con un código las extensiones estandarizadas, entre las que podemos citar las siguientes:

- M: añade soporte a operaciones de multiplicación y división de enteros.
- A: soporta operaciones atómicas para sincronización en sistemas multiprocesador.
- F y D: introduce instrucciones para el procesamiento de punto flotante de precisión simple y doble.
- C: soporta instrucciones comprimidas de 16 bits para reducir el tamaño del código.
- V: aporta instrucciones para el procesamiento vectorial, necesaria para aplicaciones de computación de alto rendimiento, IA y procesamiento de imágenes.
- B: instrucciones de manipulación de bits, útiles en criptografía y DSP.

Además, la ISA de RISC-V facilita la creación de extensiones personalizadas utilizando los campos disponibles en los formatos de instrucción. En la literatura revisada se citan casos de uso, como, por ejemplo, la aceleración de algoritmos de IA, lo que subraya la capacidad de RISC-V para adaptarse a las necesidades de mercados emergentes

El ecosistema *software* incluye herramientas de desarrollo, como los compiladores GCC y Clang/LLVM, junto con simuladores como QEMU y Spike para el desarrollo y la depuración de aplicaciones. Además, la compatibilidad con distribuciones de Linux y sistemas operativos en tiempo real (RTOS) como FreeRTOS y Zephyr asegura que RISC-V sea una plataforma viable para una variedad de entornos.

Se estudian varios núcleos RISC-V de código abierto, incluyendo los siguientes:

- VexRiscv: este núcleo RISC-V se ha diseñado con una visión parametrizable, está orientado a ser implementado en dispositivos FPGA, capaz de ejecutar Linux.
- Rocket Core: se trata de una arquitectura RISC-V de 5 etapas de segmentación que soporta la especificación RISC-V RV64GC. Está orientada a aplicaciones de alto rendimiento,
- BOOM: al igual que Rocket Core, está orientado a ejecutar RV64GC y su principal característica es la capacidad de ejecutar instrucciones fuera de orden.
- PicoRV32, NEORV32 y CVA5: se trata de núcleos de tamaño reducido, optimizados para bajo consumo y con periféricos integrados, especialmente orientados a FPGA.
- CVA6: procesador de 64 bits de alto rendimiento de 6 etapas de segmentación, siendo capaz de soportar un sistema operativo completo.

Para tener una visión clara de las facilidades aportadas por distintos núcleos RISC-V, se ha realizado el estudio y la experimentación de varios núcleos. Se han implementado en dispositivos FPGA, lo que proporciona una evaluación práctica de su rendimiento y eficiencia. En este sentido, se han implementado los distintos *cores*, algunos de ellos con soporte para RV32 y otros para RV64.

El núcleo NEORV32 aporta facilidad de uso y modularidad. Como ejemplo de parametrización, se ha activado un controlador DMA para transferencias de datos eficientes disponible en NEORV32. También se ha evaluado la capacidad de integración de instrucciones personalizadas utilizando la CFU, lo que se demostró, para un algoritmo de cifrado XTEA, que redujo el tiempo de ejecución en un factor de 6 en comparación con respecto a la implementación *software*. La disponibilidad de una amplia documentación y un entorno *software* con un notable desarrollo facilitan su utilización para aplicaciones empuotradas.

El núcleo AIRISC está optimizado para aplicaciones embebidas. Ha alcanzado la frecuencia de 100 MHz con un bajo impacto en la FPGA. Presenta una baja ocupación de recursos, con una arquitectura de 5 etapas de pipeline que utiliza interfaces AHB-Lite. La estrategia de utilización de memoria es diferente al resto de los núcleos, al

utilizar todos los bloques BRAM disponibles, lo que incrementa ligeramente el consumo de potencia.

El núcleo CVA5 es un núcleo RISC-V de 32 bits que tiene un bajo impacto en el uso de recursos de la FPGA, que al igual que los casos anteriores, funciona a 100 MHz en una placa basada en dispositivo FPGA Artix. Sin embargo, el soporte *software* no está tan bien desarrollado como en el caso de NEORV32, lo que le posiciona en desventaja en la utilización para sistemas empotrados.

Por su parte, CVA6 se implementó en una Placa de prototipado Genesys 2 que incluye un dispositivo FPGA Kintex de la serie 7 de AMD. También se ha realizado su síntesis lógica para tecnologías ASIC (CMOS TSMC 65nm). Se ha demostrado su capacidad de integración y la facilidad para arrancar y ejecutar sistemas operativos complejos como Linux en la FPGA, lo que confirma su aptitud para aplicaciones de alto rendimiento y grado industrial.

Una vez desarrollado el trabajo, se puede concluir que la arquitectura RISC-V es una alternativa viable y en crecimiento, destacando la facilidad de uso de *cores* como NEORV32 para fines educativos e industriales, y la robustez de CVA6 para aplicaciones más exigentes, como es el caso de aceleración de IA y de aplicaciones vectoriales. En cualquier caso, es preciso profundizar en el ecosistema *software*, para incrementar la disponibilidad de librerías portadas a los procesadores referenciados.

Las líneas de trabajo futuro propuestas tratan de profundizar en la evaluación experimental de otros núcleos, tanto de código abierto (como Sargantana) como comerciales (de empresas como Andes Technologies y Ventana Micro Systems). Entre los elementos a analizar de forma más profunda es preciso citar las interfaces con aceleradores *hardware* en CVA6 y la evaluación de los requisitos hardware para la integración de RTOS en *cores* de 32 bits. Y finalmente, con objeto de tener una evaluación efectiva de las prestaciones es preciso comparar de manera rigurosa el rendimiento de estos núcleos con los de arquitecturas competidoras, como por ejemplo ARM, utilizando aplicaciones del mundo real.

Abstract

The present work analyzes the RISC-V architecture, highlighting its key aspects, including its open, modular, and free ISA (Instruction Set Architecture). RISC-V is presented as an alternative to proprietary architecture. This encourages innovation in the field of computer architecture, reducing costs, power consumption, and improving security, which are all key aspects for current applications. RISC-V originated 15 years ago at the University of California, Berkeley, and the first version of the ISA was published in May 2011. The flexibility of the ISA has driven its adoption in a wide range of applications, from very simple solutions for IoT devices to HPC (High-Performance Computing) applications in data centers. RISC-V is gaining significant industry acceptance, as seen with NVIDIA, which recently published its advancements in integrating RISC-V into its GPU-based processing systems.

The design of RISC-V is based on RISC (Reduced Instruction Set Computer) principles, using simple, efficient instructions with efficient coordination between the hardware architecture and the compiler. The base architecture, RV32I, supports 32-bit integers and can be extended to support more complex data types (e.g., floating-point) and more costly instructions, such as division. It uses different instruction formats with very regular decoding, little-endian memory management, and a register structure consistent with the processing model. Additionally, it includes different operating modes regarding privilege levels (User Mode, Supervisor Mode, and Machine Mode), which guarantees system protection based on the application type (bare-metal, RTOS, Linux, etc.).

The modular structure of the RISC-V ISA is supported by an extension mechanism that allows new instructions to be added based on the requirements of the processor's use case. For example, if multiplication operations are needed, the necessary extension is included, which impacts the architecture's performance. The RISC-V International Foundation has established a standardization mechanism for extensions, identifying standardized extensions with a code. These include:

- M: Adds support for integer multiplication and division operations.
- A: Supports atomic operations for synchronization in multiprocessor systems.
- F and D: Introduce instructions for single and double-precision floating-point processing.
- C: Supports 16-bit compressed instructions to reduce code size.
- V: Provides instructions for vector processing, necessary for high-performance computing, AI, and image processing applications.
- B: Bit manipulation instructions, useful in cryptography and DSP.

Furthermore, the RISC-V ISA facilitates the creation of custom extensions using the available fields in the instruction formats. The reviewed literature cites use cases, such as accelerating AI algorithms, which underscores RISC-V's ability to adapt to the needs of emerging markets.

The software ecosystem includes development tools like GCC and Clang/LLVM compilers, along with simulators like QEMU and Spike for application development and

debugging. Additionally, compatibility with Linux distributions and real-time operating systems (RTOS) like FreeRTOS and Zephyr ensures that RISC-V is a viable platform for a variety of environments.

In this work, several open-source RISC-V cores are studied, including the following:

- VexRiscv: This RISC-V core is designed with a configurable vision, is aimed at being implemented on FPGA (Field-Programmable Gate Array) devices, and is capable of running Linux.
- Rocket Core: This is a 5-stage pipelined RISC-V architecture that supports the RV64GC specification. It is aimed at high-performance applications.
- BOOM: Similar to Rocket Core, it is aimed at running RV64GC, and its main feature is the ability to execute instructions out of order.
- PicoRV32, NEORV32, and CVA5: These are small cores optimized for low power consumption and with integrated peripherals, especially suited for FPGAs.
- CVA6: A high-performance 64-bit processor with a 6-stage pipeline, capable of supporting a full operating system.

To get a clear view of the capabilities offered by different RISC-V cores, a study and experimentation of several cores were conducted. They were implemented on FPGA devices, which provided a practical evaluation of their performance and efficiency. In this regard, the different cores were implemented, some with support for RV32 and others for RV64.

The NEORV32 core provides ease of use and modularity. As an example of parameterization, a DMA (Direct Memory Access) controller was enabled for efficient data transfers available in NEORV32. The ability to integrate custom instructions using the CFU (Custom Function Unit) was also evaluated, which for an XTEA encryption algorithm, was shown to reduce execution time by a factor of 6 compared to the software implementation. The availability of extensive documentation and a well-developed software environment facilitate its use for embedded applications.

The AIRISC core is optimized for embedded applications. It has reached a frequency of 100 MHz with a low impact on the FPGA. It shows low resource utilization, with a 5-stage pipeline architecture that uses AHB-Lite interfaces. The memory utilization strategy is different from the other cores, as it uses all available BRAM blocks, which slightly increases power consumption.

The CVA5 core is a 32-bit RISC-V core that has a low impact on FPGA resource usage, and like the previous cases, it operates at 100 MHz on a board based on an Artix FPGA device. However, the software support is not as well-developed as in the case of NEORV32, which puts it at a disadvantage for use in embedded systems.

Meanwhile, CVA6 was implemented on a Genesys 2 prototyping board that includes a Kintex 7 series FPGA from AMD. Its logical synthesis for ASIC (Application-Specific Integrated Circuit) technologies (TSMC 65nm CMOS) was also performed. Its integration capability and the ease of booting and running complex operating systems

like Linux on the FPGA were demonstrated, confirming its suitability for high-performance and industrial-grade applications.

Upon completing the work, it can be concluded that the RISC-V architecture is a viable and growing alternative, highlighting the ease of use of cores like NEORV32 for educational and industrial purposes, and the robustness of CVA6 for more demanding applications, such as AI acceleration and vector applications. In any case, it is necessary to further develop the software ecosystem to increase the availability of libraries ported to the referenced processors.

The proposed future work aims to delve deeper into the experimental evaluation of other cores, both open-source (like Sargantana) and commercial (from companies like Andes Technologies and Ventana Micro Systems). Among the elements to be analyzed in more depth are the interfaces with hardware accelerators in CVA6 and the evaluation of hardware requirements for the integration of RTOS in 32-bit cores. Finally, to have an effective performance evaluation, it is necessary to rigorously compare the performance of these cores with those of competing architectures, such as ARM, using real-world applications.

Tabla de Contenidos

Capítulo 1. Introducción	1
1.1. Introducción	1
1.2. RISC-V	2
1.3. Objetivos de RISC-V	4
1.4. Ventajas de RISC-V	4
1.5. Aplicaciones.....	5
1.6. Objetivos del trabajo	6
1.7. Estructura del documento.....	7
Capítulo 2. Arquitectura ISA RISC-V	9
2.1. Introducción	9
2.2. Instrucciones RV32I	9
2.3. Tipos de datos y organización de la memoria	11
2.4. Registros RV32I.....	12
2.5. <i>Loads y stores</i>	13
2.6. Modelo de memoria.....	13
2.7. Interrupciones	14
2.8. Registros de control y estado (CSR)	14
2.9. Ejemplos de programas para RISC-V	16
2.10. Conclusiones.....	17
Capítulo 3. Ecosistema de extensiones	19
3.1. Introducción	19
3.2. Codificación de extensiones	19
3.3. Ordenación de bytes de las instrucciones.....	21
3.4. Especificación oficial de extensiones	21
3.4.1. Extensiones ISA no privilegiadas	22
3.5. Extensiones ISA RISC-V recientes para diferentes aplicaciones.....	31
3.5.1. Internet de las cosas (IoT)	31
3.5.2. Inteligencia Artificial (IA)	33
3.5.3. Criptografía	36
3.5.4. Computación de alto rendimiento	38
3.5.5. Compresión de datos.....	40
3.5.6. Seguridad	40
3.6. Conclusiones.....	42
Capítulo 4. Metodología <i>software</i> RISC-V	43
4.1. Simuladores y emuladores	43
4.1.1. Jupiter	43

4.1.2. Venus.....	44
4.1.3. RARS.....	45
4.1.4. Spike	45
4.1.5. Ripes	46
4.1.6. QEMU.....	46
4.2. Herramientas de desarrollo.....	47
4.2.1. Compiladores C/C++	47
4.2.2. IDEs.....	47
4.3. Sistemas Operativos y RTOS.....	48
4.4. Conclusiones.....	49
Capítulo 5. Revisión de IPs disponibles para RISC-V.....	51
5.1. Núcleos de CPU y SoC.....	51
5.1.1. VexRiscv	51
5.1.2. Rocket Core	52
5.1.3. BOOM.....	52
5.1.4. PicoRV32.....	52
5.1.5. NEORV32.....	53
5.1.6. NaxRiscv	55
5.1.7. NOEL-V	55
5.1.8. ORCA RISC-V.....	57
5.1.9. SERV	58
5.1.10. VROOM.....	59
5.1.11. Ibex	60
5.1.12. CVE2	60
5.1.13. CVA6	61
5.1.14. Sargantana	62
5.2. IPs RISC-V comerciales.....	63
5.2.1. Procesadores de SiFive	63
5.2.2. Synopsys ARC-V	66
5.2.3. AIRISC CORE	67
5.3. Núcleos RISC-V orientados a FPGA.....	68
5.3.1. CVA5.....	68
5.3.2. AMD MicroBlaze V.	70
5.4. Plataformas basadas en RISC-V	72
5.4.1. X-HEEP	72
5.4.2. Cheshire	73
5.4.3. Sonata System	73
5.4.4. PolarFire SoC FPGA.....	74
5.5. Conclusiones.....	75
Capítulo 6. Implementación de los núcleos RISC-V.....	77
6.1. Implementación de NEORV32.....	77

6.1.1. Configuración de las herramientas <i>software</i>	77
6.1.2. Desarrollo del <i>hardware</i>	80
6.1.3. Integración de módulos externos en el diseño	80
6.2. Implementación de AIRISC	89
6.2.1. Configuración del <i>core</i> AIRISC	90
6.2.2. Desarrollo del <i>hardware</i>	92
6.2.3. Implementación y análisis de prestaciones	94
6.3. Implementación de CVA5	97
6.3.1. Configuración de las herramientas y del bloque IP	97
6.3.2. Diseño de la plataforma	98
6.3.3. Implementación y análisis de prestaciones	98
6.4. Implementación de CVA6	100
6.4.1. Implementación FPGA	100
6.4.2. Implementación ASIC	102
6.5. Análisis de los resultados	106
6.5.1. Comparación de área/recursos	107
6.5.2. Comparación de las prestaciones temporales	109
6.5.3. Comparación de consumo de potencia	109
6.6. Conclusiones	110
Capítulo 7. Conclusiones y trabajos futuros	111
7.1. Conclusiones	111
7.2. Trabajos futuros	112
Referencias	113

Índice de Figuras

Figura 1. Resultados de publicaciones científicas [9]	2
Figura 2. RISC-V core de 6 etapas de pipeline [21]	3
Figura 3. Crecimiento de la utilización de RISC-V en diferentes ámbitos de aplicación [31]	6
Figura 4. Formatos de instrucción base RISC-V codificados[33].....	11
Figura 5. Funcionamiento memoria principal RISC-V[35]	14
Figura 6. Ejemplo de codificación de las instrucciones RISC-V	16
Figura 7. Código de operación de las instrucciones RISC-V [40].....	20
Figura 8. Codificación de la instrucciones auipc [40].....	21
Figura 9. Representación en memoria de la instrucción auipc [40]	21
Figura 10. Estructura de la ISA RISC-V [39]	22
Figura 11. Operaciones de multiplicación para la extensión M	22
Figura 12. Formato de instrucción para la operación de división	22
Figura 13. Formato de instrucciones load/store atómicas.....	23
Figura 14. Instrucciones atómicas	23
Figura 15. Instrucciones punto flotante de precisión simple	23
Figura 16. Instrucciones de punto flotante de precisión doble	24
Figura 17. Instrucciones de punto flotante de precisión cuádruple.....	24
Figura 18. Instrucciones comprimidas.....	24
Figura 19. Formato para las extensiones Zicntr y Zihpm.....	25
Figura 20. Formato para las instrucciones de la extensión ZICSR	26
Figura 21. Formato para las instrucciones de la extensión ZIFENCEI	26
Figura 22. Formato de instrucción para la extensión Zimop	27
Figura 23. Formato de instrucciones MOP	27
Figura 24. Formato de instrucciones Zcmop	28
Figura 25. Formato de instrucciones Zawrs.....	28
Figura 26. Formato para extensión Zfh y Zfhmin	29
Figura 27. Diseño hardware de muestreo gaussiano discreto[43]	31
Figura 28. Diagrama general del ecosistema Scale4Edge [47]	33
Figura 29. Comparación de la densidad de código de varias redes neuronales artificiales para ARM, x86, RV32I y RV32I con instrucciones personalizadas [48]	34
Figura 30. Formato de instrucción personalizado basado en la extensión P [49]	35
Figura 31. Instrucciones personalizadas para el training de una DNN (Adaptada de [50]).....	36
Figura 32. Interpretación de la extensión de las instrucciones [50].....	36
Figura 33. Resultados obtenidos (tiempo de compresión: menor => más rápido) [51].....	37
Figura 34. Rendimiento medio de los algoritmos PQC Kyber y Dilithium [52]	37
Figura 35. Resultados de desempeño obtenidos [53]	38
Figura 36. Resultados de las arquitecturas de 32 bits y comparación con arquitecturas de referencia de 32 bits [54]	38
Figura 37. Comparación entre los enfoques de verificación convencionales y el enfoque propuesto[55]	39
Figura 38. Análisis de potencia obtenidos[56]	40
Figura 39. Estadísticas sobre la relación de aceleración debido a instrucciones de adyacencia vectorial[57].....	40
Figura 40. Potencia global de los submódulos obtenidos[58].....	41
Figura 41. Comparación del rendimiento de cifrado/descifrado en el número de instrucciones para diferentes tamaños de mensajes entre la línea base, la implementación optimizada basada en ISA, la implementación asistida por ISE y diferentes implementaciones de vectorización [59]	41
Figura 42. Jupiter GUI Mode.....	43
Figura 43. Ejecución de Venus en VS Code.....	44
Figura 44. Interfaz del simulador RARS [64]	45
Figura 45. Simulador Ripes [68].....	46
Figura 46. IDE Freedom Studio [75].....	48

Figura 47. Diferentes implementaciones de BOOM [96]	52
Figura 48. Núcleo PicoRV32 conectado a una lógica personalizada [98]	53
Figura 49. Estructura del procesador NEORV32 [99]	53
Figura 50. Jerarquía de buses de NEORV32 [99]	54
Figura 51. Arquitectura general del procesador NaxRiscv [103]	55
Figura 52. SoC NOEL-V	55
Figura 53. Flujo de datos internos de SERV	58
Figura 54. Arquitectura VROOM	59
Figura 55. Arquitectura del núcleo Ibex	60
Figura 56. Diagrama de bloques arquitectura CVE2 [120]	60
Figura 57. Estructura de desarrollo de microprocesadores RISC-V de OpenHW Group	61
Figura 58. Arquitectura CVA6 [125]	61
Figura 59. Utilización de CVA6 en aplicaciones industriales	62
Figura 60. Arquitectura de Sargantana [127]	62
Figura 61. Implementación del procesador Sargantana en tecnologías GF 22nm FD-SOI [126]	63
Figura 62. Arquitectura de la serie E3 de SiFive	63
Figura 63. Diagrama de bloques de los núcleos E31[129] y E51[130]	64
Figura 64. Tabla comparativa de rendimientos [131]	64
Figura 65. Cluster de procesadores de SiFive	65
Figura 66. Diagrama de bloques de ARC-V RPX-105 [140]	67
Figura 67. Arquitectura AIRISC Core Complex [142]	68
Figura 68. Arquitectura de CVA5 [143]	69
Figura 69. Linux arrancado sobre CVA5 usando OpenSBI	69
Figura 70. Diagrama de bloques del procesador MicroBlaze V [145]	70
Figura 71. Ejemplo de plataforma basada en MicroBlaze V	71
Figura 72. Arquitectura X-HEEP [148]	72
Figura 73. Arquitectura de Cheshire [151]	73
Figura 74. Diagrama de bloques de la placa de desarrollo Sonata[153]	74
Figura 75. Diagrama de bloques PolarFire SoC Icicle Kit[157]	75
Figura 76. Hello World ejecutado en NEORV32	78
Figura 77. Resultados de la ejecución de ordenación por inserción	80
Figura 78. Diseño de bloques implementado en Vivado	81
Figura 79. Implementación de NEORV32	81
Figura 80. Prestaciones del bloque IP NEORV32	82
Figura 81. Detalles de los resultados de prestaciones de la implementación del bloque IP NEORV32	82
Figura 82. Impacto en la utilización de recursos del IP NEORV32	82
Figura 83. Terminales de E/S del diseño	83
Figura 84. Detalles de la implementación final del diseño	83
Figura 85. Análisis temporal para una frecuencia de referencia de 100 MHz	83
Figura 86. Análisis de potencia del diseño basado en NEORV32	84
Figura 87. Resultados de implementación final de NEORV32	84
Figura 88. Detalles de los IOB utilizados	84
Figura 89. Resultado de la ejecución del ejemplo demo_bram	85
Figura 90. Resultado test demo_dma	85
Figura 91. Integración del algoritmo de cifrado ligero XTEA	86
Figura 92. Aplicación XTEA usando transferencias DMA	89
Figura 93. Diagrama de bloques de AIRISC	90
Figura 94. Esquema de interconexión AIRISC	91
Figura 95. Resultado testbench RV32IMFC	91
Figura 96. Resultado testbench RV32M	91
Figura 97. Resultado testbench RV32F	92
Figura 98. Mapa de memoria AIRISC [159]	93
Figura 99. Señales del bus AHB-Lite [159]	93
Figura 100. AIRISC con la interfaz AHB	94
Figura 101. Layout del núcleo AIRISC	95
Figura 102. Layout del pipeline de AIRISC	95

Figura 103. Utilización de recursos utilizados por la plataforma basada en AIRISC	95
Figura 104. Histograma para el análisis temporal	96
Figura 105. Análisis temporal de AIRISC	96
Figura 106. Potencia disipada por AIRISC	96
Figura 107. Pipeline 5 etapas CVA5[143]	97
Figura 108. Configuración del bloque CVA5	98
Figura 109. Diseño de bloques CVA5	98
Figura 110. Layout del core CVA5 implementado	99
Figura 111. Utilización de recursos utilizados por la plataforma basada en CVA5.....	99
Figura 112. Requerimientos temporales del diseño (setup, hold y pulso)	99
Figura 113. Ruta crítica del diseño (Control/Status Registers)	99
Figura 114. Análisis de potencia del diseño implementado	100
Figura 115. Memoria de program de CVA5	100
Figura 116. Prestaciones de la implementación de CVA6 para Genesys 2	101
Figura 117. Layout de la implementación de CVA6 para la placa AMD Kintex Genesys2	101
Figura 118. Modos de funcionamiento en CVA6 durante el proceso de arranque	101
Figura 119. OpenSBI arrancando en el microprocesador CVA6	102
Figura 120. Ejecución de aplicaciones para el arranque de DHCP en la interfaz Ethernet.....	102
Figura 121. Jerarquía del diseño en Synopsys Design Vision.....	103
Figura 122. Interfaz de E/S de CVA6	103
Figura 123. Bloques principales de CVA6 sintetizados en Design Compiler	103
Figura 124. Bloque contadores de prestaciones de CVA6.....	104
Figura 125. Histograma de Slack del diseño	104
Figura 126. Ruta crítica (resumen)	105
Figura 127. Distribución del tipo de células estándar utilizadas en el diseño	105
Figura 128. Distribución del área utilizada por las células estándar utilizadas en el diseño	105
Figura 129. Ruta crítica (resumen)	106
Figura 130. Distribución de la potencia estimada en la fase de síntesis	106
Figura 131. Utilización de recursos para distintos cores: (a) datos agregados, (b) detalle de LUTs y FFs y (c) detalles de BRAMs y DSPs.	108
Figura 132. Potencia disipada (en Watios)	110
Figura 133. Potencia disipada de los cores de 32 bits	110

Índice de Tablas

Tabla 1. Tipos de datos nativos de RV32I	11
Tabla 2. Conversión de tipos de datos de C a RV32I	12
Tabla 3. Registros de RISC-V RV32I (adaptado de [34]).....	12
Tabla 4. Registros de punto flotante (FP) (adaptado de [34])	13
Tabla 5. Registros de la extensión vectorial (adaptado de [34])	13
Tabla 6. Combinaciones de modos y privilegios soportados RISC-V[36].....	14
Tabla 7. Extensiones ratificadas para RISC-V [42]	29
Tabla 8. Alternativas de integración de aceleradores hardware en NEORV32 (adaptado de [102])	54
Tabla 9. Recursos utilizados por NOEL-V para dispositivos Arty de AMD [105]	56
Tabla 10. Datos de utilización de recursos de NOEL-V para dispositivos Kintex [105].....	57
Tabla 11. Resultados de rendimiento	57
Tabla 12. Resultados de implementación para distintas configuraciones y dispositivos FPGA	58
Tabla 13. Frecuencia de funcionamiento de MicroBlaze-V	71
Tabla 14. Recursos utilizados en dispositivos Artix 7	71
Tabla 15. Comparación de resultados de implementación de distintos cores.....	107
Tabla 16. Frecuencias de funcionamiento de los cores RISC-V analizados	109
Tabla 17. Potencia disipada por las implementaciones realizadas (en Watios)	110

Listado de código fuente

Código 1. Código C de ordenamiento por inserción [38]	17
Código 2. Instrucciones ejecutadas en la ordenación por inserción [38]	17
Código 3. Instalación de RISC-V GNU toolchain	78
Código 4. Código fuente del programa de ordenación de datos adaptados a NEORV32.....	79
Código 5. Función xtea_encrypt_dma_text	87
Código 6. Función xtea_decrypt_dma_text	88
Código 7. Transferencia de datos por DMA	88

ACRÓNIMOS

ADAS	<i>Advanced Driver Assistance Systems</i>
AI	<i>Artificial Intelligence</i>
ALU	<i>Arithmetic and Logic Unit</i>
AMD	<i>Advanced Micro Devices</i>
AMO	<i>Atomic Memory Operation</i>
ARM	<i>Advanced RISC Machine</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
ASIL	<i>Automotive Safety Integrity Level</i>
AXI	<i>Advanced Extensible Interface</i>
BHT	<i>Branch History Table</i>
BRAM	<i>Block Random Access Memory</i>
CISC	<i>Complex Instruction Set Computer</i>
CMOS	<i>Complementary MOS</i>
CPU	<i>Central Processing Unit</i>
CSR	<i>Control-Status Register</i>
CSRRC	<i>Control-Status Register Read and Clear</i>
CSRRS	<i>Control-Status Register Read and Set</i>
CSRRW	<i>Control-Status Register Read and Write</i>
CTO	<i>Chief Technology Officer</i>
DDR	<i>Double Data Rate</i>
DMA	<i>Direct Memory Access</i>
DNN	<i>Deep Neural Network</i>
DSP	<i>Digital Signal Processing</i>
DUT	<i>Device Under Test</i>
EPI	<i>European Processor Initiative</i>
FPGA	<i>Field Programmable Gate Array</i>
FPU	<i>Floating Point Unit</i>
GCC	<i>GNU Compiler Collection</i>
GPIO	<i>General Purpose Input/Output</i>
GPP	<i>General Purpose Processor</i>
HAL	<i>Hardware Abstraction Layer</i>
HDL	<i>Hardware Description Level</i>
HW	<i>Hardware</i>
IBM	<i>International Business Machines</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
INTEL	<i>Integrated Electronics</i>
IoT	<i>Internet of Things</i>
IP	<i>Intellectual Property</i>
IPMS	<i>Institute for Photonic MicroSystems</i>
ISA	<i>Instruction Set Architecture</i>
IUMA	<i>Instituto Universitario de Microelectrónica Aplicada</i>
JTAG	<i>Join Test Action Group</i>

KVM	<i>Kernel-based Virtual Machine</i>
LLVM	<i>Low Level Virtual Machine</i>
LUT	<i>Look-Up Table</i>
MDU	<i>Multiply-Divide Unit</i>
MIF	<i>Memory Initialization File</i>
MIPS	<i>Microprocessor without Interlocked Pipeline Stages</i>
MMCM	<i>Mixed Mode Clock Manager</i>
MMU	<i>Memory Management Unit</i>
MXU	<i>Matrix Extension Unit</i>
PMP	<i>Physical Memory Protection</i>
PWM	<i>Pulse Width Modulation</i>
RARS	<i>RISC-V Assembler and Runtime Simulator</i>
RISC	<i>Reduced Instruction Set Computer</i>
RISC-V	<i>Reduced Instruction Set Computer - Version 5</i>
RTL	<i>Register-Transfer Level</i>
RTOS	<i>Real Time Operating System</i>
RVTSO	<i>RISC-V Total Store Ordering</i>
RVWMO	<i>RISC-V Weak Memory Ordering</i>
SDK	<i>Software Development Kit</i>
SIMD	<i>Single-Instruction Multiple-Data</i>
SoC	<i>System-on-Chip</i>
SPI	<i>Serial Peripheral Interface</i>
SW	<i>Software</i>
TFM	<i>Trabajo Fin de Máster</i>
TLB	<i>Translation Lookaside Buffer</i>
TVM	<i>Tensor Virtual Machine</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
VCIX	<i>Vector Coprocessor Interface Extension</i>
VHDL	<i>VHSIC Hardware Description Language</i>
VM	<i>Virtual Machine</i>
WHS	<i>Worst Hold Slack</i>
WNS	<i>Worst Negative Slack</i>
WPWS	<i>Worst Pulse Width Slack</i>
X-HEEP	<i>eXtendable Heterogeneous Energy-Efficient Platform</i>

Capítulo 1. Introducción

1.1. Introducción

En la actualidad, pueden encontrarse una gran variedad de procesadores comerciales para el desarrollo de sistemas electrónicos complejos, donde en muchos casos es necesario utilizar técnicas de codiseño *hardware/software* para obtener rendimientos optimizados para un ámbito de aplicaciones concreto. La mayoría de estos procesadores son núcleos que incorporan una arquitectura de conjunto de instrucciones (ISA-*Instruction Set Architecture*) cerrada, protegida y propietaria, donde su utilización requiere licencias de explotación o condicionados al uso de componentes y sistemas de desarrollo comerciales.

Por ejemplo, las compañías que disponen de especificaciones de arquitectura de juego de instrucciones (ISA) como ARM[1], IBM[2] e Intel[3] tienen patentes sobre aspectos optimizados de sus ISAs que impiden a otros usuarios utilizarlos sin licencias. Las negociaciones se llevan a cabo durante 6-24 meses y son muy costosas, lo que impiden a los académicos y a otros usuarios su uso sin incurrir en un alto coste de acceso. Asimismo, una licencia de ARM no permite realizar el diseño personalizado de un núcleo en cuanto a la modificación de su ISA, ya que sólo se pueden utilizar sus diseños (sólo las grandes empresas tienen licencias que permiten nuevos núcleos de ARM) e incluso para la fundación OpenPOWER[4] se debe pagar a IBM para usar su ISA. Esta situación ahoga a la competencia y a la innovación, impidiendo que muchos usuarios diseñen y puedan compartir sus núcleos compatibles con la ISA. También, es importante destacar que esta industria se beneficia de mercados libres, así como de *software* libre de código abierto. Finalmente, los ISA propietarios no están garantizados a durar, es decir, si una compañía desaparece, la posibilidad de supervivencia de la ISA es prácticamente inviable (caso de OpenSPARC)[5].

Todo ello justifica la necesidad de disponer de una ISA abierta para el diseño de procesadores donde se alcance una mayor innovación a través de la competencia del mercado, incluyendo las implementaciones de ISAs abiertas que se puedan confrontar contra las propietarias; diseños de núcleos abiertos compartidos, lo que significaría un plazo más corto de comercialización, un menor costo de reutilización, menos errores al ser revisado por más usuarios y transparencia, que haría difícil, por ejemplo, que se añadan puertas traseras; y procesadores cada vez más asequibles para más dispositivos, lo que ayuda a expandir IoT.

Un ejemplo de tales iniciativas es RISC-V, ISA abierta y modular, flexible, que permite incluir extensiones en función de las necesidades de diseño. Ello facilita su utilización en un amplio rango de diseños, desde aquellos muy simples orientados a IoT

a otros más complejos para computación de altas prestaciones. Por este motivo, Europa ha apostado por esta arquitecturas y la iniciativa para crear un microprocesador en Europa (EPI - *European Processor Initiative*) incluye RISC-V en su *roadmap*[6].

SPARC V8 [7], RISC y MIPS [8] fueron las primeras ISAs abiertas y libres de regalías, pero estas propuestas no se han consolidado en el mercado. En 2010 se crea RISC-V que elimina la mayor parte de las restricciones impuestas por las ISAs propietarias, permitiendo agregar un subconjunto de instrucciones personalizadas a los usuarios y las empresas. Desde entonces, RISC-V ha creado múltiples implementaciones y realizaciones de silicio ya sea en ASIC (Circuito Integrado de Aplicación Específica) como en FPGA. La evolución de esta arquitectura se refleja cuando se observa el número de publicaciones científicas que hacen referencia a RISC-V en los últimos cinco años [9]. En la Figura 1, se puede observar una comparación de la cantidad de publicaciones científicas que incluyen núcleos RISC-V, MIPS, núcleos comerciales para ARM Cortex-M1, BA22-DE, Intel NIOS II y Xilinx MicroBlaze y para los núcleos de código abierto OpenRISC [10], LEON3, OpenSPARC y Mirco32. Para la comparación, se observa que el creciente interés en los núcleos RISC-V para FPGA es claramente visible.

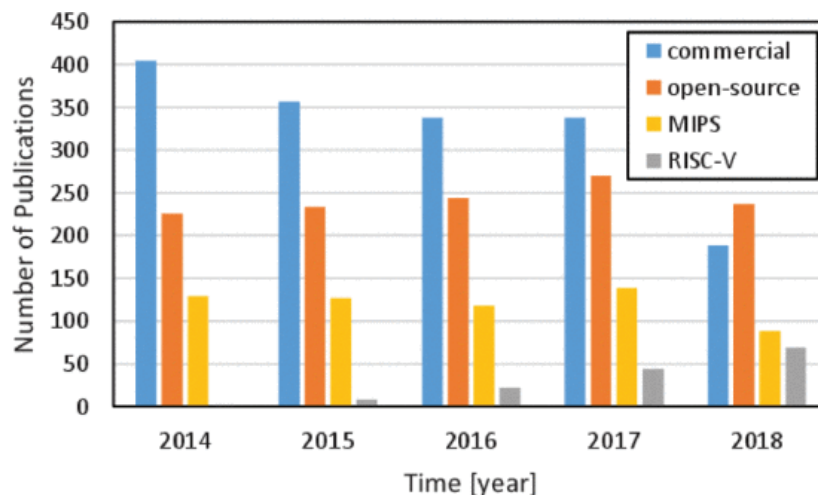


Figura 1. Resultados de publicaciones científicas [9]

La situación actual de esta arquitectura de conjunto de instrucciones (ISA) engloba una gran popularidad en una amplia gama de aplicaciones, desde sistemas embebidos hasta centros de datos. Si tenemos en cuenta las implementaciones, podemos encontrar una variedad tanto comerciales como de código abierto. Entre las comerciales se encuentran el RISC-V 64 de Andes Technology[11], la XPU de SiFive[12] y Western Digital[13] y las de código abierto, HiFive1[14], Pulpino[15] y Rocket Chip[16].

1.2. RISC-V

RISC-V[17] es una ISA emergente que ha sido desarrollada en la Universidad de California, Berkeley y soportado por la RISC-V *Foundation*[18]. La Fundación RISC-V está compuesta por más de 325 compañías. Debido a que la ISA es abierta y libre de derechos, los núcleos de procesador o el sistema en chips (SoCs) de RISC-V pueden ser diseñados y liberados libremente. Por esta razón, se han liberado los núcleos de procesador y los SoC que operan en las FPGAs como Rocket chip[16] y BOOM[19].

Además, varios programas informáticos, incluidos algunos compiladores y sistemas operativos (SO), comienzan a dar soporte a la ISA[20].

En la Figura 2 se muestra un diagrama de bloques de la arquitectura para una implementación con un *pipeline* de 6 etapas [21].

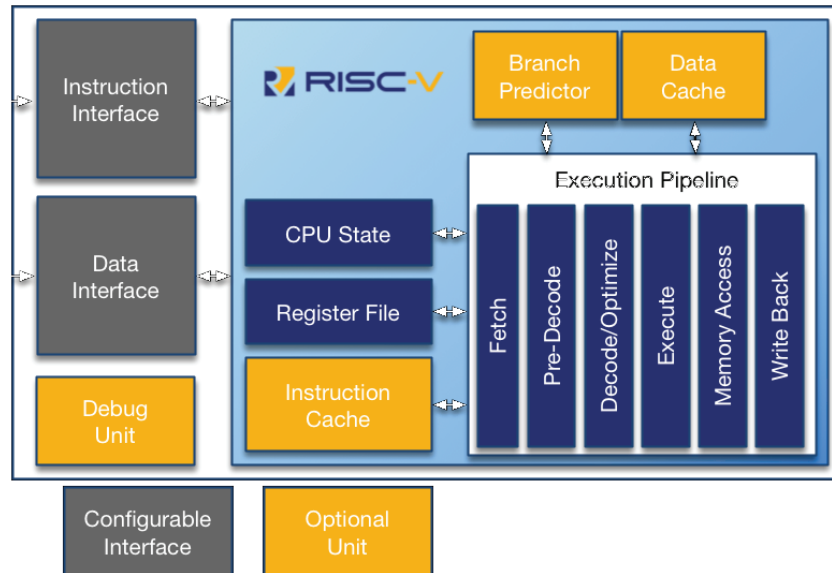


Figura 2. RISC-V core de 6 etapas de pipeline [21]

Linux también está disponible en RISC-V, dando soporte a múltiples aplicaciones. Se prevé un crecimiento de RISC-V con soporte de Linux. Ejemplo de este tipo de sistemas es Freedom U500 VC707 Dev Kit [22] de SiFive [23] aunque se requiere una FPGA de alta gama.

Por lo tanto, mediante el desarrollo de un sistema basado en RISC-V que puede ser fácilmente utilizado en una pequeña FPGA y soportar Linux, se espera que sea utilizado en varios dominios de aplicación.

El conjunto de instrucciones se organiza en tres especificaciones: RV32I, RV64I y RV128I, que definen los conjuntos de instrucciones de 32, 64 y 128, respectivamente y los conjuntos vectoriales de RV32V y RV64V. En todo caso se diferencia en el ancho de los registros de la arquitectura. En RISC-V [24] es posible extender la ISA para que puedan ser agregadas a la ISA de base denominada RV32I, que soporta el espacio de direcciones de 32 bits. Los diseños pueden seleccionar los conjuntos de instrucciones soportados dependiendo de la forma de utilizar los sistemas. En este sentido se definen distintas extensiones: la extensión M soporta la multiplicación y división de números enteros en *hardware*, la extensión C incluye las instrucciones comprimidas y la extensión F para las instrucciones de punto flotante son definidas. La anotación de RV32IM indica la ISA que apoya a RV32I y la extensión M. Para soportar Linux en un sistema RISC-V, no sólo es necesario soportar RV32IM sino también RV32GC (G=IFAMD) con la extensión A para las instrucciones atómicas.

1.3. Objetivos de RISC-V

RISC-V fue originalmente diseñada para apoyar la investigación y la educación en arquitectura de computadoras[24], pero ahora se espera que también se convierta en una arquitectura estándar libre y abierta para las implementaciones de la industria. Los objetivos en la definición de RISC-V incluyen:

- Una ISA completamente abierta que está disponible gratuitamente para la educación y la industria.
- Una ISA real adecuada para la implementación directa de *hardware* nativo, no sólo para traducción simulada o binaria.
- Una ISA que evita usar arquitecturas sobredimensionadas para un estilo particular de microarquitectura (por ejemplo, microcodificado, en orden, desacoplado, fuera de orden) o la tecnología de implementación (por ejemplo, la tecnología de ASIC, FPGA), pero que permite la implementación eficiente en cualquiera de estos.
- Una ISA separada en una pequeña base entera ISA, utilizable por sí mismo como base para la personalización aceleradores o con fines educativos, y extensiones estándar opcionales, para apoyar a los desarrolladores de *software* de propósito.
- Apoyo a la norma revisada de 2008 IEEE-754 para punto flotante.
- Una ISA que soporta extensiones de la ISA a nivel de usuario y variantes especializadas.
- Variantes de espacio de direcciones de 32 y 64 bits para aplicaciones, núcleos de sistemas operativos e implementaciones de *hardware*.
- Una ISA con soporte para implementaciones mononúcleo o multinúcleo altamente paralelas, incluyendo multiprocesadores heterogéneos.
- Instrucciones opcionales de longitud variable tanto para ampliar el espacio de codificación de instrucciones disponible como para soportar una codificación de instrucciones densa opcional para mejorar el rendimiento y el tamaño del código estático, y la eficiencia energética.
- Una ISA totalmente virtualizable para facilitar la utilización con sistemas hipervisor.
- Una ISA que simplifica los diseños de los experimentos con el nuevo nivel de supervisor y el nivel de hipervisor ISA.

1.4. Ventajas de RISC-V

Los beneficios clave de la tecnología RISC-V presentan ventajas significativas en distintas etapas del desarrollo del sistema[25]:

- **Arquitectos de *software*/ingenieros de *firmware*/desarrolladores de *software*.** Las instrucciones básicas y las extensiones opcionales que han sido aprobadas también son libres. Debido a la estabilidad de la ISA, el desarrollo de *software* se puede aplicar con confianza a RISC-V sabiendo que se preservará su inversión. El *software* escrito para RISC-V se ejecutará en todos los núcleos RISC-V similares. Debido a que RISC-V ISA está abierto, esto se traduce en que los ingenieros *hardware* tienen más flexibilidad sobre la implementación del procesador. Con esta capacidad, es posible modificar la implementación final del *hardware* en función de las necesidades del *software*.
- **CTO/Diseñadores de chips/Ingenieros de sistemas.** La innovación es el habilitador clave de RISC-V. Debido a que el ISA está abierto, es el equivalente de que todos tengan una licencia de microarquitectura. Se pueden optimizar diseños en términos de potencia, rendimiento, seguridad, etc., manteniendo una compatibilidad total con otros diseños. Debido a que existe un control significativamente mayor sobre la implementación *hardware*, es posible tomar decisiones en momentos iniciales del diseño que condicionan los resultados finales. RISC-V también admite instrucciones personalizadas para diseños que requieren aceleración particular o funciones especiales.
- **Diseñadores de sistemas e integradores.** Las ISAs abiertas de RISC-V puede proporcionar varios beneficios adicionales. Por ejemplo, si los ingenieros están implementando un núcleo *soft* RISC-V en una FPGA, a menudo el código fuente RTL está disponible. Dado que RISC-V está libre de regalías, esto crea una flexibilidad significativa para portar un diseño basado en RISC-V desde una FPGA a un ASIC u otra FPGA sin ninguna modificación de *software*. Los diseñadores que se preocupan por la seguridad desde una perspectiva de confianza también apreciarán RISC-V[6]. Cuando el código fuente RTL está disponible, esto permite una inspección profunda.

1.5. Aplicaciones

Desde la creciente adopción de RISC-V en la industria, existen numerosas aplicaciones en marcha que utilizan esta arquitectura, con empresas como Amazon Web Services[26], Google[27], Samsung[28] o incluso IBM[29]. Se calcula que más de 200 empresas desarrollan actualmente productos basados en RISC-V y se espera que siga creciendo en los próximos años. Entre ellas podemos ver RISC-V en sistemas embebidos por su bajo consumo y su reducido tamaño; centros de datos por su rendimiento y eficiencia energética; es cada vez más popular en computación en la nube por proporcionar potencia de cálculo a aplicaciones de alto rendimiento; inteligencia artificial que permite proporcionar potencia de cálculo para una amplitud de aplicaciones de desarrollo automático; criptografía aportando mejoras en eficiencia y seguridad en algoritmos de cifrado, instrucciones de firma digital o autenticación y otros protocolos de seguridad; robótica por su compatibilidad con una amplia gama de sensores y actuadores.

Por ejemplo, en [20] diseñan un sistema de comunicación RISC-V portátil y con capacidad para Linux que soporta RV32IMAC e implementado sobre una FPGA a partir de una descripción *hardware* realizada en Verilog HDL. Igualmente, debido a su sencillez, se plantean su utilización en aplicaciones médicas implantables[30]. En la Figura 3 se muestra la proyección realizada para la utilización del microprocesador RISC-V en diferentes ámbitos de aplicaciones [31].

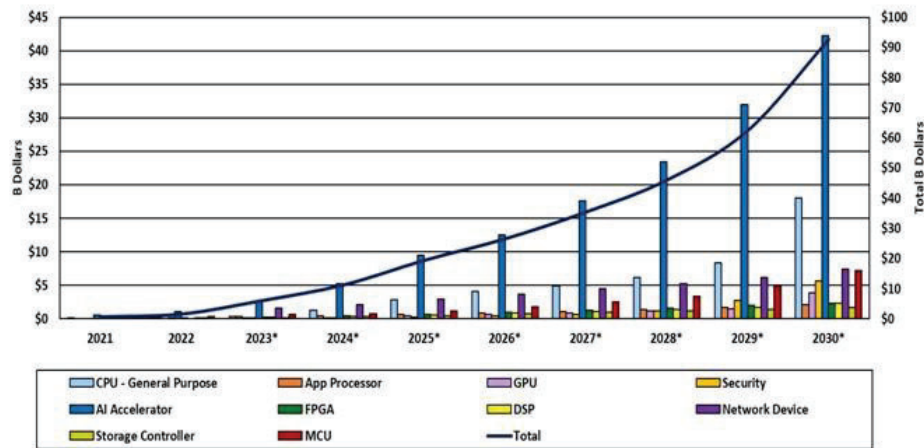


Figura 3. Crecimiento de la utilización de RISC-V en diferentes ámbitos de aplicación [31]

La arquitectura RISC-V está en constante desarrollo y en los últimos años, se han producido varios avances importantes entre los que cabe destacar estas versiones que incluyen nuevas instrucciones y funciones que aportan una mejora de la eficiencia, el rendimiento y la seguridad.

- Publicación de la ISA RISC-V versión 2.2, lanzada en 2022. Esta versión introduce una serie de mejoras como el soporte para instrucciones vectoriales de 12 bits y la virtualización.
- Publicación de la ISA RISC-V versión 3.0, lanzada en 2023. Se incluyen mejoras para el soporte de instrucciones de aprendizaje automático y una mejora en seguridad.

En conclusión, la arquitectura RISC-V está bien definida y documentada. Existe una gran variedad de implementaciones y existen diversas herramientas y recursos para ayudar a los desarrolladores a trabajar con RISC-V. Todo lo que se necesita es un pc, una placa de desarrollo y un conjunto de *software* disponible gratuitamente para transformar programas C en implementaciones equivalentes de FPGA.

1.6. Objetivos del trabajo

El principal objetivo de este TFM consiste en el estudio de la arquitectura RISC-V para diferentes dispositivos FPGA. En base a esto, se pretende llevar a cabo los siguientes objetivos operativos:

- O1. Conocer las ventajas, limitaciones y características principales de la ISA RISC-V, así como definir los principales ámbitos de aplicación
- O2. Conocer el ecosistema de soluciones *software* necesario para el desarrollo de aplicaciones
- O3. Seleccionar distintas implementaciones existentes y estudiar la idoneidad en cuanto a eficiencia, ocupación y el consumo energético de dicha arquitectura en dispositivos FPGA avanzados
- O4. Obtener resultados y realizar comparaciones de las plataformas implementadas bajo distintas condiciones de optimización
- O5. Documentar y presentar el trabajo realizado

1.7. Estructura del documento

Este documento se estructura de la siguiente manera:

- Capítulo 1. En este capítulo se presenta de manera introductoria las características más relevantes de RISC-V y la necesidad del uso de esta arquitectura.
- Capítulo 2. Se recogen las distintas arquitecturas ISA RISC-V disponibles, instrucciones base y extensiones, nivel de privilegios, tipos de formato de instrucción y cómo se interpreta cada bit.
- Capítulo 3. El siguiente capítulo ofrece una visión detallada del universo de extensiones de RISC-V, explicando desde las extensiones base y estándar hasta su uso en diferentes ámbitos.
- Capítulo 4. Este capítulo explora la metodología *software* asociada a RISC-V. Se abordarán los simuladores y emuladores que ofrecen evaluar y depurar programas sin necesidad de *hardware* físico, así como las herramientas de desarrollo, compiladores y entornos integrados que mejoran la experiencia del desarrollador. También se incluye el soporte de sistemas operativos, desde distribuciones Linux compatibles hasta sistemas en tiempo real para dispositivos integrados.
- Capítulo 5. Para este capítulo se enumeran y detallan los diferentes *cores* disponibles para la arquitectura agrupándolas según si son *open-source*, comerciales o están basadas en plataformas.
- Capítulo 6. En este capítulo se implementan los *cores* NeoRV32, AIRISC y CVA5 y CVA6.
- Capítulo 7. Este último capítulo recoge las conclusiones obtenidas de este trabajo.

Capítulo 2. Arquitectura ISA RISC-V

Uno de los aspectos competitivos de la especificación ISA de RISC-V es su capacidad de incluir la funcionalidad necesaria, reduciendo de forma significativa los requisitos energéticos necesarios. Para ello parte de una arquitectura de referencia y añade, mediante extensiones a la arquitectura base, la funcionalidad requerida. En este capítulo se estudia la arquitectura básica de RISC-V y sus extensiones, mostrando tanto los formatos de instrucciones como los mecanismos de integración de extensiones disponibles.

2.1. Introducción

Una arquitectura del juego o conjunto de instrucciones (ISA - *Instruction Set Architecture*) representa un modelo abstracto de una especificación de las instrucciones de una CPU. Una ISA define el conjunto de instrucciones, tipos de datos, registros, acceso a memoria, modos de direccionamiento de memoria y entradas/salidas, visibles por el programador.

La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la microarquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Representa la interfaz más importante en un sistema informático, ya que contiene todas las instrucciones que la CPU puede entender y ejecutar. La ISA es el representa la capa de menor nivel del *software* del sistema y el nivel de control del *hardware*.

Dependiendo del contenido semántico de las instrucciones incluidas, es posible clasificar la ISA como *Reduced Instruction Set Computer* (RISC) o *Complex Instruction Set Computer* (CISC).

Tal y como adelantábamos en el capítulo 1, los primeros proyectos RISC provinieron de IBM, Stanford y UC-Berkeley a finales de los años 1970 y principios de 1980. Estos proyectos se desarrollaron bajo el concepto de arquitectura de conjunto de instrucciones reducidas. La principal característica es que poseen tiempos de ejecución de ciclo único, ejecución simultánea de partes o fases de instrucciones con el fin de procesar instrucciones de manera más eficiente y un gran número de registros para evitar grandes cantidades de interacciones con la memoria.

2.2. Instrucciones RV32I

RISC-V define una ISA básica nivel de usuario orientada a enteros, extensible y modular, con soporte para distintas variantes. RV32I aporta un conjunto de instrucciones básicas y varias extensiones que se pueden combinar para implementar

versiones especializadas[32]. Dependiendo del *hardware* y de la aplicación a ejecutar se usará una u otra extensión, tomando como referencia el conjunto básico. A continuación, se presentan las variantes de la ISA de RISC-V y algunas de sus extensiones:

- ISA básica:
 - RV32I: Instrucciones de enteros de 32 bits
 - RV32E: Instrucciones de enteros de 32 bits RV32E para microprocesadores empotrados
 - RV64I: Instrucciones de enteros de 64 bits
 - RV128I: Instrucciones de enteros de 128 bits RV128I
- Extensiones:
 - M: Extensión estándar para multiplicación y división de números enteros
 - A: Extensión estándar para instrucciones atómicas
 - F: Extensión estándar para punto flotante de precisión simple
 - D: Extensión estándar para coma flotante de doble precisión
 - G: Abreviación de las extensiones IMFD
 - Q: Extensión estándar para punto flotante de precisión cuádruple
 - L: Extensión estándar para punto flotante decimal
 - C: Extensión estándar para instrucciones comprimidas
 - B: Extensión estándar para manipulación de bits
 - J: Extensión estándar para idiomas traducidos dinámicamente
 - T: Extensión estándar para memoria transaccional
 - P: Extensión estándar para instrucciones SIMD empaquetadas
 - V: Extensión estándar para operaciones vectoriales
 - N: Extensión estándar para interrupciones a nivel de usuario
 - H: Extensión estándar para hipervisor

RV32I, con tan sólo 40 instrucciones, implementa las operaciones necesarias para conseguir la funcionalidad básica con enteros de 32 bits. Está codificada en un formato de instrucción de 32 bits, incluye instrucciones para suma, resta, operaciones lógicas a nivel de bit, *load/store*, saltos y *branches*.

En la Figura 4 se muestran los tipos de formatos de instrucción utilizados y los campos en los que se organiza:

- Tipo R: formato de instrucción para instrucciones que operan con registros
- Tipo I: formato para operaciones con inmediatos y *load* para carga de datos de memoria a registros
- Tipo S: formato *store*, que carga datos de registro a memoria
- Tipo SB: formato *branch*
- Tipo U: formato inmediato superior
- Tipo J: formato de saltos no condicionales

Los campos de los formatos de instrucción son los siguientes:

- *opcode* (código de operación): son los bits que especifican la instrucción a ejecutar

2.3. Tipos de datos y organización de la memoria

- *rd* (registro *de* destino): especifican el registro de destino que contendrá el resultado de la operación
- *rs1* (registro fuente 1): especifican el primer registro de origen
- *rs2* (registro fuente 2/inmediato): especifican el segundo registro de origen o un valor inmediato
- *funct* (*función*): se trata de distintos campos específicos (disponibles para algunas instrucciones) y especifican una variante de la instrucción

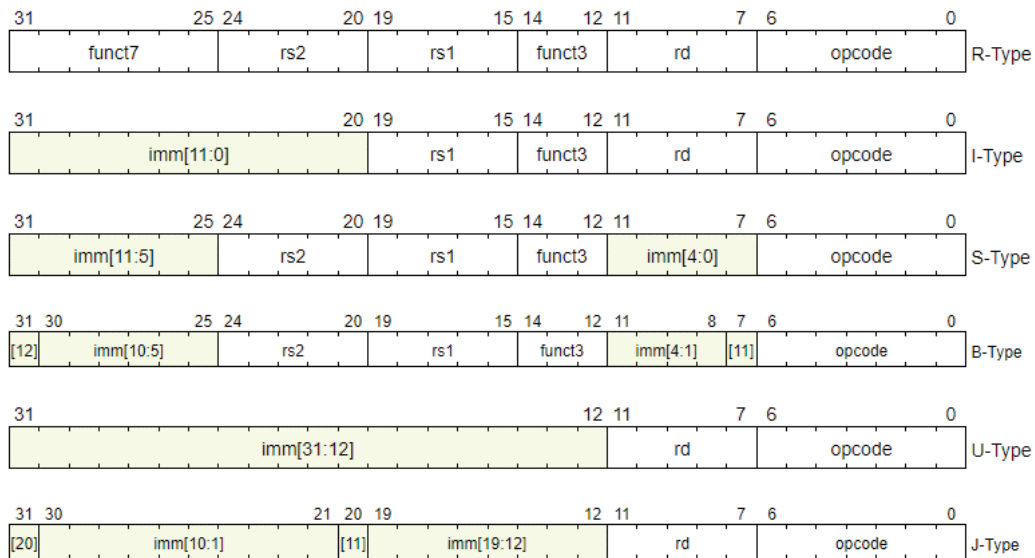


Figura 4. Formatos de instrucción base RISC-V codificados[33]

2.3. Tipos de datos y organización de la memoria

Los tipos de datos nativos de ISA (Tabla 1) son tipos de datos que ISA puede procesar de forma natural. La siguiente tabla muestra los tipos de datos nativos RV32I y sus respectivos tamaños en bytes[32].

Tabla 1. Tipos de datos nativos de RV32I

Nombre del tipo de datos RV32I nativo	Tamaño en bytes
byte	1
unsigned byte	1
halfword	2
unsigned halfword	2
word	4
unsigned word	4

Al traducir un programa escrito en lenguaje C al código ensamblador RV32I, los tipos de datos C deben convertirse a tipos de datos nativos RISC-V. La siguiente tabla muestra las asignaciones de tipos de datos nativos de C a tipos de datos nativos RV32I. Todos los punteros en C (por ejemplo, *int*, *char* y *void**) representan direcciones de memoria y están asignados al tipo de datos de palabra sin signo (Tabla 2).

Tabla 2. Conversión de tipos de datos de C a RV32I

C	RV32I	Tamaño en bytes
bool	byte	1
char	byte	1
unsigned char	unsigned byte	1
short	halfword	2
unsigned short	unsigned halfword	2
int	word	4
unsigned int	unsigned word	4
long	word	4
unsigned long	unsigned word	4
void*	unsigned word	4

2.4. Registros RV32I

El ISA sin privilegios RV32I contiene treinta y dos registros y el PC, de un ancho de 32 bits (Tabla 3). El registro x0 es un registro especial que se puede usar para descartar el resultado de una operación, ya que siempre se lee 0. Los registros (x1-x31) son de propósito general, aunque la interfaz binaria de aplicación (ABI) especifica un propósito para cada uno de ellos y permiten trabajar con números enteros. Esto significa que algunos registros contengan datos temporales o guardados, punteros, direcciones de retorno, etc. El contador de programa (PC) contiene la dirección de la siguiente instrucción que se debe ejecutar.

Tabla 3. Registros de RISC-V RV32I (adaptado de [34])

Registro	Denominación ABI	Descripción	Preservado por
x0	zero	Valor constante 0	Inmutable
x1	ra	Dirección de retorno	Caller
x2	sp	Puntero de pila	Callee
x3	gp	Puntero global a los datos (<i>Global Pointer</i>)	N/A*
x4	tp	Puntero de hilo de procesamiento (<i>Thread Pointer</i>)	N/A*
x5 - x7	t0-t2	Registros temporales	Caller
x8	s0/fp*	Callee-saved/Frame pointer*	Callee
x9	s1	Callee-saved	Callee
x10 - x17	a0-a7	Argumentos de funciones/Valor retorno	Caller
x18 - x27	s2-s11	Callee-saved	Callee
x28 - x31	t3-t6	Temporales	Caller
pc	pc	Contador de programa	N/A*
fp*: el <i>frame pointer</i> es opcional. N/A* indica que no es aplicable			

Para el caso de usar extensiones de punto flotante, estas disponen de sus propios registros, tal como se recoge en la Tabla 4.

Tabla 4. Registros de punto flotante (FP) (adaptado de [34])

Nombre	Denominación ABI	Descripción	Preservado entre llamadas
f0 - f7	ft0 - ft7	Registros temporales	No
f8 - f9	fs0 - fs1	<i>Callee-saved registers</i>	Sí
f10 - f17	fa0 - fa7	Argumentos	No
f18 - f27	fs2 - fs11	<i>Callee-saved registers</i>	Sí
f28 - f31	ft8 - ft11	Registros temporales	No

Finalmente, para las extensiones vectoriales, existe otro conjunto de registros dedicados para la transferencia de datos (Tabla 5).

Tabla 5. Registros de la extensión vectorial (adaptado de [34])

Nombre	ABI	Descripción	Preservado entre llamadas
v0-v31	N/A	Registros temporales	No
vl	N/A	Longitud del vector	No
vtype	N/A	Tipo de datos del vector	No
vxrm	N/A	Modo de redondeo para las operaciones de punto fijo	No
vxsat	N/A	Flag de saturación para operaciones de punto fijo	No

2.5. Loads y stores

Una arquitectura *load/store* requiere que los valores se carguen/almacenen explícitamente desde/en la memoria antes de operar con ellos. RISC-V es una arquitectura de carga/almacenamiento de 32 bits (*lw*, *sw*) además de bytes y *halfwords*, *signed* o *unsigned* (*lb*, *lbu*, *lh*, *lhu*, *sb*, *sh*). Por lo tanto, para realizar operaciones aritméticas, aunque el dato no sea de un tamaño de 32 bytes, este operará correctamente rellenando con cero.

2.6. Modelo de memoria

RISC-V es una arquitectura que contiene un bus de datos y de direcciones de 32 bits por lo que permite direccionar $2^{32} \times 8b = 2^{30} \times 32b$ (4 GiB). Su memoria está preparada para trabajar con datos de 8, 16 y 32 bits e instrucciones de 32 bits y todos ellos se encuentran alineados y con ordenación *little-endian*, esto significa que el byte menos significativo de una palabra se almacena en la dirección menos significativa de la memoria. En la Figura 5 se representa un modelo de la memoria principal en la que se enfoca en la gestión de accesos de lectura y escritura. Tal y como se puede apreciar, la dirección de acceso “a” y el dato de salida “d” tienen un tamaño de 32 bits, permitiendo acceder a diferentes tamaños de datos (“B” byte, “H” media palabra, “W” palabra). Para controlar el acceso, se usan las señales de habilitación WE, OE y CE y ERR para detectar errores de alineamiento. Por tanto, en este modelo se puede comprobar cómo el sistema detecta y responde a accesos mal alineados, así como se controla el flujo de datos entre el procesador y la memoria.

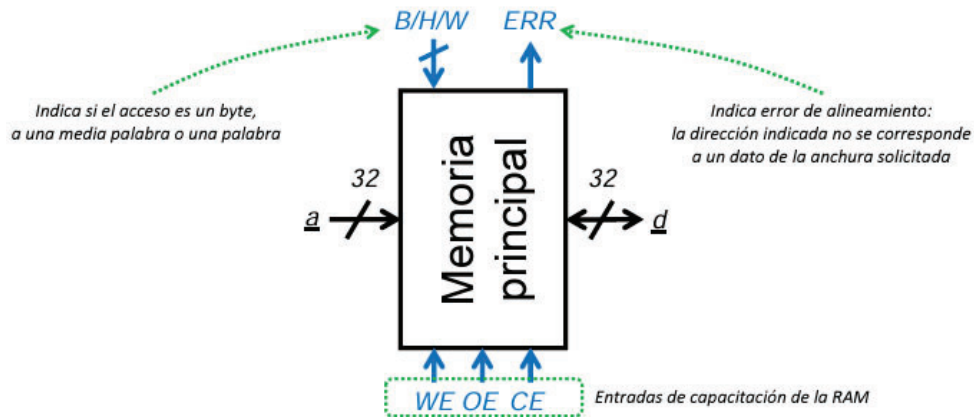


Figura 5. Funcionamiento memoria principal RISC-V[35]

2.7. Interrupciones

En cualquier momento, un subproceso *hardware* RISC-V se ejecuta en algún nivel de privilegio codificado como modo en uno o más registros de control y estado (CSR). Los niveles de privilegio se usan para proporcionar protección entre diferentes componentes de la pila. La ISA de RISC-V define tres niveles de privilegio (Tabla 6):

- **Modo Usuario/Aplicación:** para los procesos de usuario destinados al uso de aplicaciones normales y sistemas operativos
- **Modo Supervisor:** para el *kernel* y/o controladores de dispositivos, ofrecer soporte a los sistemas operativos
- **Modo Máquina:** para el gestor de arranque y/o *firmware*, ejecuta el código más fiable, tiene los privilegios más altos y es el único nivel de privilegio obligatorio para una plataforma de *hardware* RISC-V

Cada nivel de privilegio tiene un conjunto básico de extensiones ISA privilegiadas con extensiones y variantes opcionales. Asimismo, especifica que el microprocesador puede implementar sólo un subconjunto de estos niveles de privilegio.

Tabla 6. Combinaciones de modos y privilegios soportados RISC-V[36]

Niveles	Modos soportados	Utilización
1	M	Sistemas empotrados simples
2	M, U	Sistemas empotrados seguros
3	M, S, U	Sistema ejecutando un sistema operativo tipo Linux

2.8. Registros de control y estado (CSR)

Los **registros de control y estado (CSR)** son un banco independiente de registros con un espacio de direcciones independiente de 12 bits, lo que limita su número a 4.096. Suelen contener información diversa de la CPU, como temporizadores, contadores, *flags*, información del fabricante y otros datos[37].

Los CSR son registros especiales diseñados para controlar y supervisar el funcionamiento del procesador. Sirven como mecanismo a través del cual el *software*

interactúa con el *hardware* para ajustar la configuración, gestionar excepciones y adquirir información sobre el estado del procesador. Comprender los CSR es esencial para cualquier programador RISC-V, ya que permiten un control preciso sobre el comportamiento del procesador y posibilitan una programación eficiente del sistema.

La **extensión Zicsr** contiene instrucciones destinadas a la manipulación de CSRs, como CSRRW (CSR *read and write*), CSRRS (CSR *read and set*), y CSRRC (CSR *read and clear*). Estas instrucciones permiten al *software* leer y modificar los valores de los registros de control y estado.

Existen varios Registros de Control y Estado (CSR) importantes en la arquitectura RISC-V, cada uno de los cuales cumple una función específica. Estos son algunos de los CSR más importantes:

- **mstatus (Registro de Estado de Máquina).** Este CSR es sin duda el más importante, ya que controla y supervisa el modo de funcionamiento y el nivel de privilegios de la máquina. Contiene campos para controlar el estado de activación/desactivación de interrupciones, establecer el nivel de privilegio (modo M, modo S, modo U) y varias banderas que influyen en el comportamiento del procesador. Es fundamental para las transiciones de nivel de privilegio y la gestión de interrupciones.
- **mepc (Contador de Programa de Excepción de Máquina).** Almacena el valor del contador de programa de la instrucción que causó una excepción o interrupción en modo máquina. Determina dónde debe reanudarse la ejecución del programa después de gestionar la excepción.
- **mtvec (Dirección Base del Vector de excepciones (*Traps*) en modo Máquina).** Especifica la dirección base del manejador de excepciones (*traps*) para modo máquina, determinando a dónde debe saltar el procesador cuando se produce una excepción.
- **mcause (Registro de Causa de Máquina).** Este CSR proporciona información sobre la razón de la excepción o interrupción más reciente. Distingue entre excepciones e interrupciones y proporciona un código que indica la causa específica, como un fallo de página o una interrupción de *software*.
- **misa (Machine ISA Register).** Especifica las extensiones del conjunto de instrucciones soportadas por el procesador, permitiendo al *software* determinar las capacidades del procesador RISC-V. También codifica el ancho de bits de la ISA base (RV32, RV64 o RV128).

Estos CSR son fundamentales para el funcionamiento de un procesador RISC-V, ya que permiten el control del modo de privilegio, el manejo de excepciones y proporcionan información sobre las capacidades del procesador y los eventos recientes. Hay muchos más CSR que cumplen otras funciones. Los citados son los más críticos para el procesador básico. En [38] se muestra una descripción detallada de los registros de control y estado.

mostrado en el código C, mediante el bucle *for* recorre la lista de elementos a ordenar uno por uno e inserta cada elemento en su posición correcta en $a[j] = x$. En el Código 2 se muestra el código desensamblado mediante objdump.

```
void insertion_sort(long a[], size_t n)
// a[]: Array to be sorted
// n: size of the array
{
    for(size_t i = 1, j; i < n; i++){
        long x = a[i];
        for(j = i; j > 0 && a[j-1] > x; j--){
            a[j] = a[j-1];
        }
        a[j] = x;
    }
}
```

Código 1. Código C de ordenamiento por inserción [38]

```
# RV32I (19 instrucciones, 76 bytes, o 52 bytes con RVC)
# Registros: n => a1, a[0] => a3, i => a4, j => a5, x => a6
0: 00450693 addi a3,a0,4 # a3 apunta a a[1]
4: 00100713 addi a4,x0,1 # i = 1
Outer Loop:
8: 00b76463 bltu a4,a1,10 # si i < n, saltar a Continue Outer loop
Exit Outer Loop:
c: 00008067 jalr x0,x1,0 # retornar de función
Continue Outer Loop:
10: 0006a803 lw a6,0(a3) # x = a[i]
14: 00068613 addi a2,a3,0 # a2 apunta a a[j]
18: 00070793 addi a5,a4,0 # j = i
Inner Loop:
1c: ffc62883 lw a7,-4(a2) # a7 = a[j-1]
20: 01185a63 bge a6,a7,34 # si a[j-1] <= a[i], saltar a Exit Inner Loop
24: 01162023 sw a7,0(a2) # a[j] = a[j-1]
28: fff78793 addi a5,a5,-1 # j--
2c: ffc60613 addi a2,a2,-4 # decrementar a2 para apuntar a a[j]
30: fe0976e3 bne a5,x0,1c # si j != 0, saltar a Inner Loop
Exit Inner Loop:
34: 00279793 slli a5,a5,0x2 # multiplicar a5 por 4
38: 00f507b3 add a5,a0,a5 # a5 es ahora el byte address de a[j]
3c: 0107a023 sw a6,0(a5) # a[j] = x
40: 00170713 addi a4,a4,1 # i++
44: 00468693 addi a3,a3,4 # incrementar a3 para apuntar a a[i]
48: fc1ff06f jal x0,8 # saltar a Outer Loop
```

Código 2. Instrucciones ejecutadas en la ordenación por inserción [38]

2.10. Conclusiones

En este capítulo se explican los principios básicos de la ISA de RISC-V, que incluye los tipos de datos, los registros, los formatos de instrucciones, el modelo de memoria, aspectos básicos de las interrupciones y del modelo de registros de control y estado. El capítulo concluye con un ejemplo de programa compilado desde C a lenguaje ensamblador para RISC-V usando la ABI estándar.

Capítulo 3. Ecosistema de extensiones

3.1. Introducción

Tal y como se ha visto en capítulos anteriores, RISC-V es una arquitectura de conjunto de instrucciones (ISA) abierta y flexible que admite explícitamente extensiones personalizadas específicas de dominio. Actualmente, esta ISA popular para procesadores integrados permite adaptarse a diferentes necesidades y aplicaciones. Sin embargo, aún existe una brecha entre las capacidades de RISC-V y a medida que la tecnología avanza, surgen nuevas demandas en diferentes áreas como inteligencia artificial, criptografía o computación de alto rendimiento [39].

Las extensiones permiten a la arquitectura mantenerse relevante y competitiva por lo que se hace necesario introducir continuamente nuevas extensiones para satisfacer las necesidades cambiantes de la industria, mejorar la eficiencia energética de los procesadores y ofrecer soluciones personalizadas. Para ello, la organización de estándares de RISC-V se encarga de investigar e introducir las nuevas propuestas de extensiones personalizadas de RISC-V brindando una base sólida y flexible para la innovación en ciertos escenarios.

Sin embargo, aunque RISC-V ha logrado un éxito significativo en los últimos años, su estándar ISA aún está en desarrollo y necesita ser mejorado aún más para diversos escenarios.

RISC-V está alcanzando su madurez en el campo de los procesadores integrados, pero todavía está en sus inicios en otras áreas como la computación en la nube, la computación de alto rendimiento, etc. Además, el conjunto de instrucciones también debe enriquecerse aún más para adaptarse a los requisitos de los escenarios de aplicación específicos del dominio, como la inteligencia artificial, la computación gráfica, etc.

3.2. Codificación de extensiones

Las extensiones se usan para ampliar y acelerar las capacidades de una CPU de dos maneras: añadiendo nuevas instrucciones o implementando un acelerador *hardware*. Las extensiones son parte del flujo de instrucciones y tienen una latencia extremadamente baja por lo que la sobrecarga de usar la instrucción puede ser mínima o nula[40].

RISC-V tiene una codificación de instrucciones muy regular dado que se permiten extensiones arbitrarias. RISC-V utiliza una codificación de longitud variable, con todas las instrucciones codificadas en 32 bits o en 16 bits para un subconjunto comprimido.

Suponiendo que los 2 bits menos significativos de cada instrucción de 32 bits son siempre 1 1. Los 5 bits siguientes son el código de operación principal que controla cómo se decodifica la instrucción (Figura 7):

Major opcode		
000	00	11
000	01	11
000	10	11
000	11	11
001	00	11
001	01	11
001	10	11
001	11	11
010	00	11
010	01	11
010	10	11
010	11	11
011	00	11
011	01	11
011	10	11
011	11	11
100	00	11
100	01	11
100	10	11
100	11	11
101	00	11
101	01	11
101	10	11
101	11	11
110	00	11
110	01	11
110	10	11
110	11	11
111	00	11
111	01	11
111	10	11
111	11	11

LSB 1 1 means non-compressed

Figura 7. Código de operación de las instrucciones RISC-V [40]

- Algunas instrucciones incluyen más campos específicos que las incluidas en el código de operación. Por ejemplo, OP (0110011) tiene dos campos de código de operación adicionales, que añaden otros 10 bits. Este código de operación principal se comparte con la ISA base, la extensión *Multiply*, la extensión Zicond, la extensión *Packed SIMD* y algunas otras. No existe una forma general de determinar si un código de operación en particular corresponde a la ISA base o a una extensión únicamente analizando el *opcode*, si no se tiene a la vista toda la decodificación de toda la instrucción.
- Algunos códigos de operación importantes corresponden por completo a extensiones específicas. Por ejemplo, AMO contiene instrucciones atómicas (la extensión A), y OP-V originalmente estaba reservado, pero ahora lo utiliza la extensión Vector.
- Grandes secciones del espacio del código de operación se utilizan para algunas funciones complejas, como la multiplicación-suma fusionada.
- custom-0custom-3: los usuarios pueden agregar sus extensiones propietarias, sin necesidad de que sean estandarizadas. Básicamente, custom-* es una

situación de libertad para todos (excepto que la ISA RV128I de 128 bits propuesta eventualmente usará custom-2 y custom-3).

Es muy importante tener en cuenta que las extensiones (no personalizadas) no son partes distintas y separadas del espacio de decodificación. RISC-V International inserta extensiones en los espacios entre las instrucciones base existentes. Las extensiones principales como Vector que tienen su propio código de operación principal (OP-V) también tienen instrucciones en LOAD-FP.

3.3. Ordenación de bytes de las instrucciones

Las instrucciones siempre se almacenan en formato *little endian*, con el byte menos significativo primero en la memoria. Esto se aplica incluso en la máquina RISC-V *big endian* (teórica). Las herramientas de desensamblado como objdump muestran los bytes como *big endian*. Por lo tanto, una instrucción como auipc aparece en la documentación y objdump como se muestra en la Figura 8.

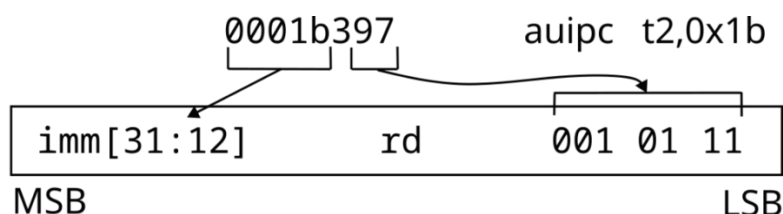


Figura 8. Codificación de la instrucciones auipc [40]

Una vez que se ha ensamblado la instrucción, su representación en memoria se muestra en la Figura 9.

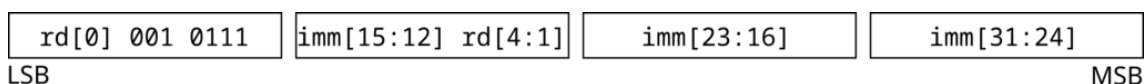


Figura 9. Representación en memoria de la instrucción auipc [40]

3.4. Especificación oficial de extensiones

RISC-V ISA consta de conjuntos de instrucciones básicas y extensiones ISA modulares, y también admite explícitamente extensiones personalizadas específicas del dominio [39]. El conjunto de instrucciones básicas RISC-V solo contiene el conjunto de instrucciones para operaciones generales con números enteros (por ejemplo, suma y resta). Las extensiones RISC-V ISA contienen una variedad de extensiones modulares para satisfacer diferentes necesidades informáticas. RISC-V International ha introducido nuevas extensiones de conjuntos de instrucciones para mejorar la funcionalidad de RISC-V, como la extensión V para el cálculo vectorial y la extensión H para la virtualización. Además de los estándares de extensión de conjuntos de instrucciones desarrollados, los investigadores también han llevado a cabo una serie de estudios para personalizar las extensiones de conjuntos de instrucciones para una variedad de escenarios para acelerar aplicaciones específicas o tareas informáticas.

A continuación, se detalla el progreso de las especificaciones ISA RISC-V recogidas por la RISC-V *International Foundation* que consta de dos partes: ISA no

privilegiada y la ISA privilegiada como se muestra en la Figura 10. La ISA sin privilegios incluye el conjunto básico de instrucciones de números enteros y varias extensiones ISA sin privilegios. La ISA con privilegios contiene las ISA básicas de nivel de máquina y de nivel de supervisor, así como las extensiones de memoria virtual y la extensión de hipervisor. En este trabajo nos centraremos en estudiar las extensiones ISA sin privilegios publicadas en *Volume 1, Unprivileged Specification* [41] versión 20240411 y actualizar la información aportada en el trabajo *RISC-V Instruction Set Architecture Extensions: A survey* [39].

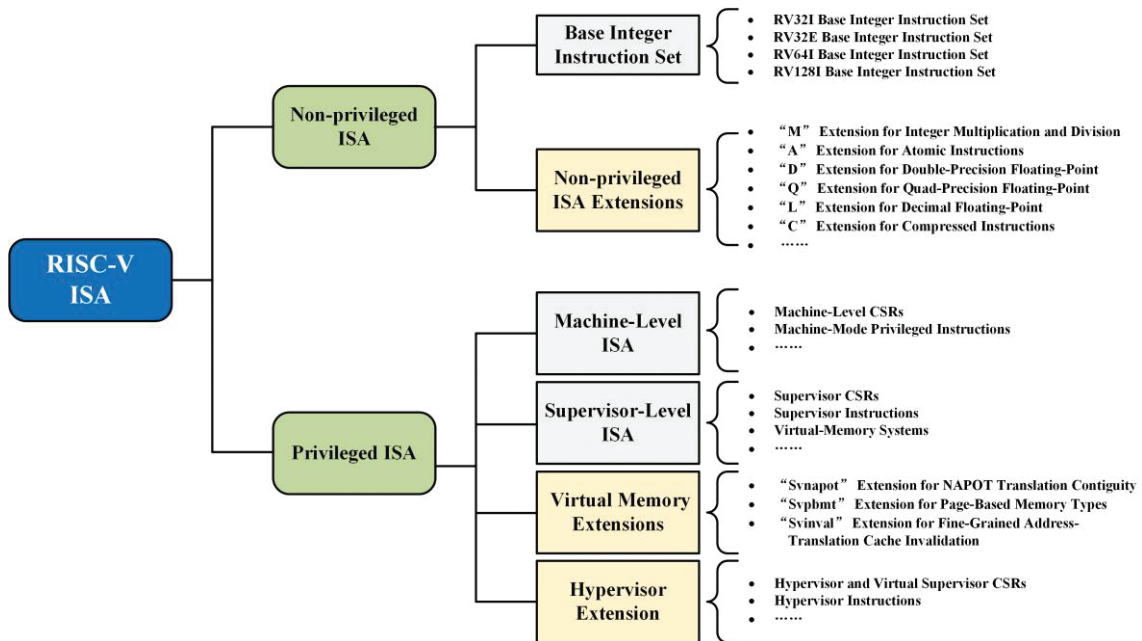


Figura 10. Estructura de la ISA RISC-V [39]

3.4.1. Extensiones ISA no privilegiadas

3.4.1.1. EXTENSIÓN M

La extensión M es la extensión del conjunto de instrucciones que soporta la multiplicación y división de enteros. Multiplica o divide los valores almacenados en dos registros enteros, incluidas las instrucciones de multiplicación y división de enteros con y sin signo (Figura 11 y Figura 12).

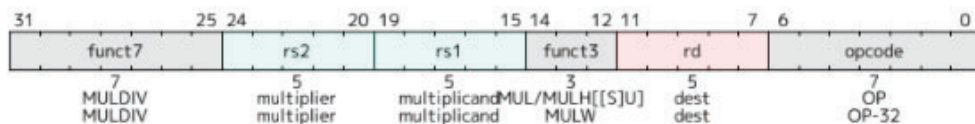


Figura 11. Operaciones de multiplicación para la extensión M

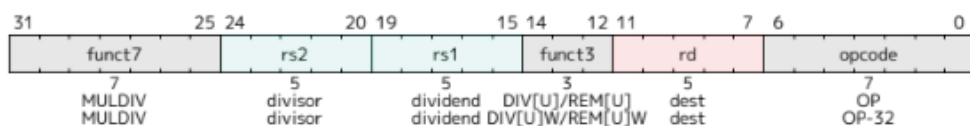


Figura 12. Formato de instrucción para la operación de división

MUL realiza una multiplicación binaria de $XLEN \times XLEN$ -bits de rs1 por rs2 y coloca los bits $XLEN$ inferiores en el registro de destino. MULH, MULHU y MULHSU realizan la

misma multiplicación pero devuelven los bits XLEN superiores del producto completo de $2 \times \text{XLEN}$ -bits, para la multiplicación para valores con $\text{rs1 signed} \times \text{rs2 signed}$, $\text{rs1 unsigned} \times \text{rs2 unsigned}$ y $\text{rs1 signed} \times \text{rs2 unsigned}$.

DIV y DIVU realizan una división entera XLEN bits por XLEN bits con signo y sin signo de rs1 por rs2, redondeando hacia cero. REM y REMU proporcionan el resto de la operación de división correspondiente. Para REM, el signo de un resultado distinto de cero es igual al signo del dividendo.

3.4.1.2. EXTENSIÓN A

La extensión A, también conocida como extensión del conjunto de instrucciones atómicas, consta de instrucciones que leen, modifican y escriben en memoria de forma atómica para proporcionar sincronización entre varios *cores* RISC-V que se ejecutan en el mismo espacio de memoria. La extensión A proporciona dos tipos de instrucciones atómicas, que son instrucciones de memoria de búsqueda y operación atómicas e instrucciones de reserva de carga/almacenamiento condicional (Figura 13 y Figura 14).

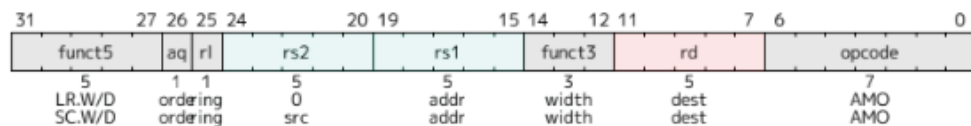


Figura 13. Formato de instrucciones load/store atómicas

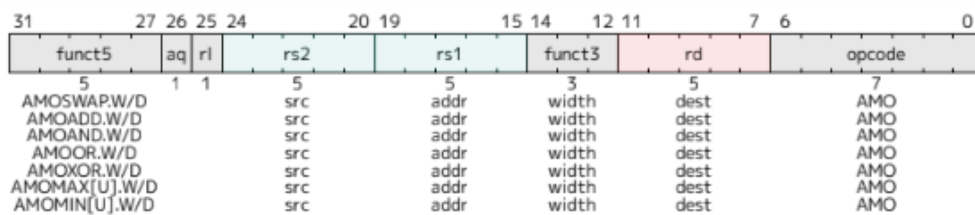


Figura 14. Instrucciones atómicas

3.4.1.3. EXTENSIÓN F

La extensión F agrega 32 registros de punto flotante, f0-f31, cada uno de 32 bits de ancho, y un registro de control y estado de punto flotante fcsr, que contiene el modo operativo y el estado de excepción de la unidad de punto flotante. La mayoría de las instrucciones de punto flotante operan sobre valores en el archivo de registro de punto flotante. Las instrucciones de carga y almacenamiento de punto flotante transfieren valores de punto flotante entre registros y memoria. También se proporcionan instrucciones para transferir valores hacia y desde el archivo de registro de enteros (Figura 15).



Figura 15. Instrucciones punto flotante de precisión simple

3.4.1.4. EXTENSIÓN D

La extensión estándar del conjunto de instrucciones de punto flotante de doble precisión, denominada "D", que agrega instrucciones computacionales de punto flotante de doble precisión que cumplen con el estándar aritmético IEEE 754-2008. La extensión D depende del subconjunto de instrucciones de precisión simple base F (Figura 16).

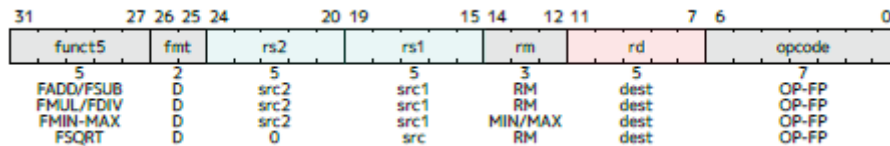


Figura 16. Instrucciones de punto flotante de precisión doble

3.4.1.5. EXTENSIÓN Q

La extensión Q es la extensión ISA de punto flotante binario de precisión cuádruple. En la extensión Q, el registro de punto flotante se extiende a 128 bits. En particular, la extensión Q agrega nuevas instrucciones de carga, almacenamiento, conversión y movimiento de precisión cuádruple, e instrucciones de clasificación, comparación y cálculo de punto flotante de precisión cuádruple. La extensión D es un requisito previo para usar la extensión Q (Figura 17).

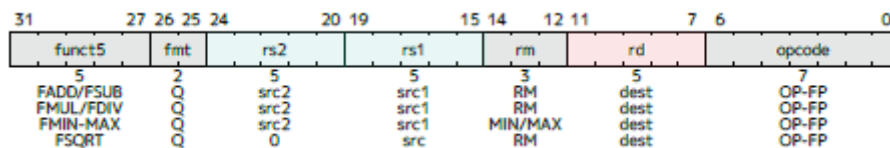


Figura 17. Instrucciones de punto flotante de precisión cuádruple

3.4.1.6. EXTENSIÓN L

La extensión L se utiliza para soportar la aritmética de punto flotante decimal. Se utiliza principalmente para guardar valores de punto flotante decimal de 64 y 128 bits en registros y mover estos valores hacia y desde la memoria con instrucciones de carga y almacenamiento. La extensión L se encuentra actualmente en la etapa de borrador y no ha sido ratificada por la Fundación Internacional RISC-V.

3.4.1.7. EXTENSIÓN C

La extensión C es la extensión que reduce el tamaño del código estático y dinámico añadiendo codificaciones de instrucciones cortas de 16 bits para operaciones comunes (Figura 18). Además, la extensión C se puede añadir a cualquiera de las ISA base (RV32, RV64, RV128).

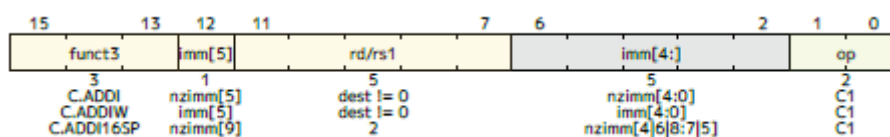


Figura 18. Instrucciones comprimidas

3.4.1.8. EXTENSIÓN Zc*

Zc* es un grupo de extensiones que definen subconjuntos de las extensiones C existentes (Zca, Zcd, Zcf) y extensiones nuevas que solo contienen codificaciones de 16 bits.

3.4.1.9. EXTENSIONES Zicntr y Zihpm PARA CONTADORES

La extensión de contadores proporciona un conjunto de hasta 32 contadores y temporizadores de 64 bits a los que se puede acceder a través de registros CSR de solo lectura específicos. Los primeros tres contadores (CYCLE, TIME e INSTRET) se utilizan específicamente para el conteo cíclico, el reloj en tiempo real y el retiro de instrucciones, mientras que los contadores restantes se pueden utilizar para el conteo de eventos programables (Figura 19).

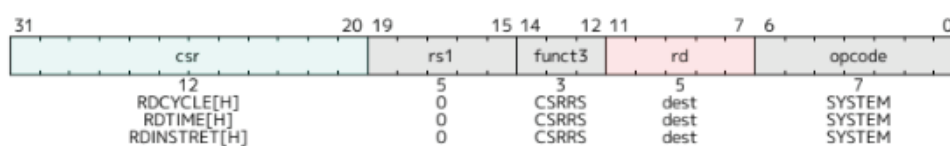


Figura 19. Formato para las extensiones Zicntr y Zihpm

3.4.1.10. EXTENSIÓN B

La extensión B es la extensión del conjunto de instrucciones de manipulación de bits, que consta de instrucciones como zba, zbb, zbc, zbs, etc. Estas instrucciones pueden admitir la manipulación de bits, incluidos recuentos de bits, desplazamientos, rotaciones, inserciones, extracciones y permutaciones, etc.

3.4.1.11. EXTENSIÓN J

La extensión J está diseñada para soportar lenguajes traducidos dinámicamente, como Java, JavaScript, etc. Estos lenguajes normalmente se implementan a través de traducción dinámica. Esta extensión tiene como objetivo proporcionar instrucciones adicionales para soportar verificaciones dinámicas, recolección de basura y aceleración JIT, etc.

3.4.1.12. EXTENSIÓN T

La extensión T es la extensión de la memoria transaccional. Proporcionará instrucciones para soportar operaciones de memoria transaccional y agregará un pequeño búfer de memoria transaccional de capacidad limitada para soportar operaciones atómicas que involucran múltiples direcciones.

3.4.1.13. EXTENSION P

La extensión P es la extensión del conjunto de instrucciones de datos múltiples y una sola instrucción empaquetada (*Packed-SIMD*). Proporciona instrucciones SIMD/DSP de punto fijo/entero y puede soportar computación paralela de datos eficiente, como procesamiento de audio, voz e imágenes, etc. La extensión P está diseñada principalmente para dispositivos integrados o DSP.

3.4.1.14. EXTENSIÓN V

La extensión de vector base tiene como objetivo proporcionar soporte general para la ejecución paralela de datos dentro del espacio de codificación de instrucciones de 32 bits, con extensiones de vector posteriores que admiten una funcionalidad más rica para ciertos dominios.

3.4.1.15. EXTENSIÓN ZICSR

La extensión Zicsr es la extensión del conjunto de instrucciones del registro de control y estado (CSR) y proporciona instrucciones para leer, modificar y escribir CSR de forma atómica. RISC-V define un espacio de direcciones independiente de CSR para cada núcleo. RISC-V tiene un espacio de CSR de 12 bits, es decir, 4096 CSR como máximo. La extensión Zicsr proporciona 6 instrucciones para operar CSR en este espacio (Figura 20).

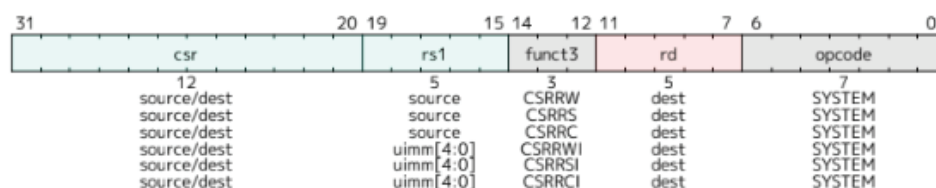


Figura 20. Formato para las instrucciones de la extensión ZICSR

3.4.1.16. EXTENSIÓN ZIFENCEI

La extensión Zifencei es la extensión de *fetch* para la obtención de instrucciones. Solo especifica una instrucción, FENCE.I, que ofrece sincronización explícita entre las escrituras en la memoria de instrucciones y la obtención de instrucciones en el mismo hilo *hardware*. Actualmente, esta instrucción es el único mecanismo estándar para garantizar que los almacenamientos visibles para un *hart* también sean visibles para sus recuperaciones de instrucciones (Figura 21).

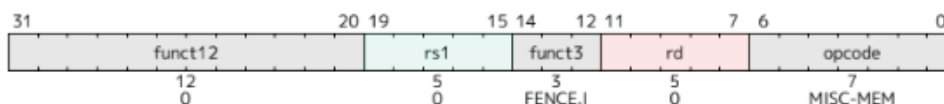


Figura 21. Formato para las instrucciones de la extensión ZIFENCEI

3.4.1.17. EXTENSIÓN ZAM

La extensión Zam es la extensión del conjunto de instrucciones atómicas desalineadas. Agrega soporte estandarizado para operaciones de memoria atómica (AMO) desalineadas a la extensión A. Al utilizar la extensión Zam, las AMO desalineadas solo necesitan ejecutarse de manera atómica con respecto a otros accesos a la misma dirección y del mismo tamaño.

3.4.1.18. EXTENSIÓN Ztso

La extensión Ztso es la extensión del conjunto de instrucciones de ordenación de almacenamiento total. En la especificación oficial de RISC-V se define un modelo básico de consistencia de memoria, que se denomina ordenación de memoria débil RISC-V (RVWMO). De manera similar al modelo de memoria débil utilizado por ARMv8, RVWMO tiene reglas de ordenación más flexibles, lo que proporciona más libertad para la implementación *hardware* y la optimización del rendimiento. La extensión Ztso

proporciona un modelo de memoria de ordenación de almacenamiento total RISC-V (RVTSO), que es una extensión de RVWMO. De manera similar a los modelos de memoria fuerte utilizados por SPARC y x86, RVTSO tiene reglas de ordenación estrictas.

3.4.1.19. EXTENSIÓN Zihintntl

Las instrucciones NTL son HINT que indican que los accesos explícitos a la memoria de la instrucción inmediatamente posterior (en adelante, "instrucción de destino") exhiben una localidad temporal de referencia deficiente. Se proporcionan cuatro variantes: La instrucción NTL.P1, NTL.PALL, NTL.S1 y NTL.ALL. Estas instrucciones se pueden utilizar para evitar la contaminación de la memoria caché al transmitir datos o atravesar grandes estructuras de datos, o para reducir la latencia en las interacciones entre productor y consumidor.

3.4.1.20. EXTENSIÓN Zihintpause

La instrucción PAUSE es un HINT que indica que la tasa de retiro de instrucciones del *hardware* actual debe reducirse o pausarse temporalmente. La duración de su efecto debe ser limitada y puede ser cero. PAUSE está codificado como una instrucción FENCE con $\text{pred}=\text{W}$, $\text{succ}=0$, $\text{fm}=0$, $\text{rd}=\text{x0}$ y $\text{rs1}=\text{x0}$.

3.4.1.21. EXTENSIÓN Zimop

La extensión Zimop introduce el concepto de instrucciones que pueden ser operaciones (MOP). Las MOP se definen inicialmente para escribir cero en $\text{x}[\text{rd}]$, pero están diseñadas para ser redefinidas por extensiones posteriores para realizar alguna otra acción. La extensión Zimop define un espacio de codificación para 40 MOP. La extensión Zimop define 32 instrucciones MOP denominadas MOP.R.n, donde n es un número entero entre 0 y 31, ambos inclusive. A menos que otra extensión las redefina, estas instrucciones simplemente escriben 0 en $\text{x}[\text{rd}]$. Su codificación permite que futuras extensiones las definan para leer $\text{x}[\text{rs1}]$, así como para escribir $\text{x}[\text{rd}]$ (Figura 22).

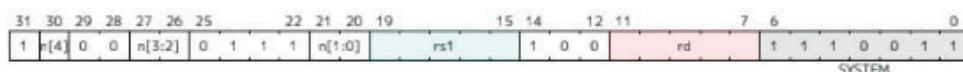


Figura 22. Formato de instrucción para la extensión Zimop

Además, la extensión Zimop define también 8 instrucciones MOP denominadas MOP.RR.n, donde n es un número entero entre 0 y 7, ambos inclusive. A menos que otra extensión las redefina, estas instrucciones simplemente escriben 0 en $\text{x}[\text{rd}]$. Su codificación permite que futuras extensiones las definan para leer $\text{x}[\text{rs1}]$ y $\text{x}[\text{rs2}]$, así como para escribir $\text{x}[\text{rd}]$ (Figura 23).

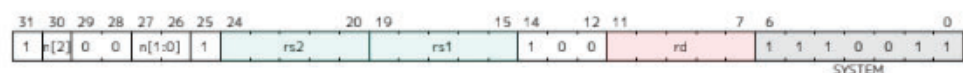


Figura 23. Formato de instrucciones MOP

3.4.1.22. EXTENSIÓN Zcmop

Zcmop define 8 instrucciones MOP de 16 bits denominadas C.MOP.n, donde n es un entero impar entre 1 y 15, ambos inclusive. C.MOP.n está codificada en el espacio de codificación reservado correspondiente a C.LUI. A diferencia de las MOP definidas en la

extensión Zimop, las instrucciones C.MOP.n están definidas para no escribir ningún registro. Su codificación permite que futuras extensiones las definan para leer el registro x[n]. La extensión Zcmop requiere la extensión Zca (Figura 24).

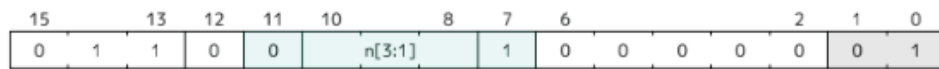


Figura 24. Formato de instrucciones Zcmop

3.4.1.23. EXTENSIÓN Zicnd

La extensión Zicnd proporciona una solución de beneficios y flexibilidad para soportar operaciones aritméticas condicionales y de selección/movimiento condicional, mientras se mantiene fiel a la filosofía de diseño de RISC-V. Las instrucciones siguen el formato de instrucciones de tipo R con 3 operandos (es decir, 2 operandos de origen y 1 operando de destino). Usando estas instrucciones, se pueden implementar secuencias sin ramificaciones (normalmente en secuencias de dos instrucciones) sin la necesidad de fusión de instrucciones, disposiciones especiales durante la decodificación de instrucciones arquitectónicas u otras disposiciones microarquitectónicas.

3.4.1.24. EXTENSIÓN Zawrs

La extensión Zawrs define un par de instrucciones que se utilizarán en bucles de sondeo que permiten que un núcleo entre en un estado de bajo consumo de energía y espere a que se almacene en una ubicación de memoria. Las instrucciones de la extensión Zawrs solo son útiles junto con la instrucción LR, que proporciona el componente Zalrsc de la extensión A (Figura 25).



Figura 25. Formato de instrucciones Zawrs

3.4.1.25. EXTENSIÓN Zacas

Comparar e intercambiar (CAS) proporciona una manera fácil y generalmente más rápida de realizar operaciones de sincronización de subprocesos cuando se admite como una instrucción *hardware*. CAS se usa generalmente en algoritmos sin bloqueo y sin espera. Esta extensión propone instrucciones CAS para operar en valores de datos de 32 bits, 64 bits y 128 bits (solo RV64). La instrucción CAS admite la operación de comparación e intercambio atómico de C++11.

3.4.1.26. EXTENSIÓN Zfh y Zfhmin

La extensión estándar Zfh y Zfhmin para instrucciones de punto flotante binario de precisión media de 16 bits cumplen con el estándar IEEE 754-2008. Zfh permite que un valor de precisión media sea encajonado en NaN dentro de un valor de precisión simple (que puede ser encajonado en NaN de manera recursiva dentro de un valor de precisión doble o cuádruple cuando la extensión D o Q está presente). La expectativa para Zfhmin es que se utilice principalmente el formato de precisión media para el almacenamiento, realizando la mayoría de los cálculos con mayor precisión. La

extensión Zfh y Zfhmin dependen de la extensión de punto flotante de precisión simple, F (Figura 26).

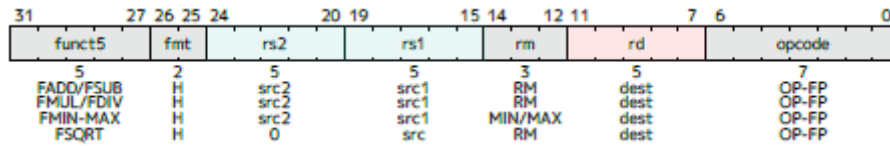


Figura 26. Formato para extensión Zfh y Zfhmin

3.4.1.27. EXTENSIONES Zfinx, Zdinx, Zhinx y Zhinxmin

La extensión Zfinx proporciona instrucciones similares a las de la extensión estándar de punto flotante F para instrucciones de punto flotante de precisión simple, pero que operan en los registros x en lugar de los registros f. Al eliminar los registros f, la extensión Zfinx reduce sustancialmente el costo de las implementaciones RISC-V simples con soporte de conjunto de instrucciones de punto flotante. La extensión Zdinx proporciona instrucciones de punto flotante de doble precisión análogas. La extensión Zhinx proporciona instrucciones de punto flotante de precisión media análogas. La extensión Zhinxmin proporciona un soporte mínimo para instrucciones de punto flotante de precisión media de 16 bits que operan en los registros x. La extensión Zdinx, Zhinx y Zhinxmin requieren de la extensión Zfinx.

Como se puede observar, la arquitectura RISC-V permite una amplia flexibilidad para adaptarse a una extensa gama de aplicaciones, y listar todas las extensiones personalizadas existentes es una tarea difícil. Muchas extensiones son desarrolladas por instituciones de investigación o empresas de forma privada. No obstante, en [42] se puede encontrar una descripción general y el estado de varias extensiones que se encuentran actualmente en desarrollo. La Tabla 7 muestra las extensiones ratificadas hasta la fecha de escritura de este documento.

Tabla 7. Extensiones ratificadas para RISC-V [42]

Nombre de la especificación	Fecha ratificación	Nuevas extensiones o perfiles
<i>Load/Store Pair for RV32 (Zilsd & Zclsd)</i>	Feb. 25	Zilsd, Zclsd
<i>The RISC-V Debug Specification</i>	Feb. 25	Sdext, Sdtrig
<i>RISC-V Control Transfer Records</i>	Nov. 24	Smctr, Ssctr
<i>RISC-V Pointer Masking</i>	Oct. 24	Smmppm, Smnpm, Ssnpm, Supm, Sspm
<i>The RISC-V Instruction Set Manual Volume II: Privileged Architecture</i>	Oct. 24	Sm1p13, Ss1p13
<i>Double Trap</i>	Ago. 24	Ssdbltrp, Smdbltrp
<i>RISC-V Quality-of-Service (QoS) Identifiers</i>	Jun. 24	Sqosid
<i>Obviating Memory-Management Instructions after Marking PTEs Valid</i>	Jun. 24	Svvptc
<i>Resumable Non-Maskable Interrupts</i>	Jun. 24	Smrnmi
<i>Shadow Stacks and Landing Pads</i>	Jun. 24	Zicfiss, Zicfilp
<i>BF16 Extensions</i>	Jun. 24	Zfbfmin, Zvfbfmin, Zvfbfwma
<i>Zaamo and Zalrsc Extensions</i>	Abr. 24	Zaamo, Zalrsc
<i>B Standard Extension for Bit Manipulation Instructions</i>	Abr. 24	B

Nombre de la especificación	Fecha ratificación	Nuevas extensiones o perfiles
<i>Byte and Halfword Atomic Memory Operations (Zabha)</i>	Abr. 24	Zabha
<i>RISC-V Supervisor Counter Delegation</i>	Mar. 24	Smcdeleg, Ssccfg
<i>Mayo-Be-Operations</i>	Mar. 24	Zimop, Zcmop
<i>RISC-V Indirect CSR Access (Smcsrind/Sscsrind)</i>	Feb. 24	Smcsrind, Sscsrind
<i>RISC-V Integer Conditional (Zicond) operations extension</i>	Nov. 23	Zicond
<i>Hardware Updating of PTE A/D Bits (Svadu)</i>	Nov. 23	A quarrel
<i>RISC-V Cycle and Instret Privilege Mode Filtering (Smcntrpmf)</i>	Nov. 23	Smcntrpmf
<i>Atomic Compare-and-Swap (CAS) Instructions (Zacas)</i>	Nov. 23	Zacas
<i>RISC-V Cryptography Extensions Volume II: Vector Instructions</i>	Sep. 23	Zvbb, Zvbc, Zvkb, Zvkg, Zvkn, Zvkc, Zvkned, Zvkng, Zvknha, Zvknhb, Zvks, Zvksc, Zvkse, Zvksg, Zvksh, Zvkt
<i>"Zfa" Standard Extension for Additional Floating-Point Instructions</i>	Sep. 23	Zfa
<i>RISC-V Advanced Interrupt Architecture</i>	Jun. 23	Smaia, Ssaia
<i>"Zvfh/Zvfhmin:" Vector Extension for Half-Precision Floating-Point Arithmetic/Vector Extension for Minimal Half-Precision Floating-Point Arithmetic</i>	Jun. 23	Zvfh, Zvfhmin
<i>"Zihintntl" Non-Temporal Locality Hints</i>	May. 23	Zihintntl
<i>RISC-V Code Size Reduction</i>	Abr. 23	Zca, Zcb, Zcd, Zce, Zcf, Zcmp, Zcmt
<i>RISC-V Profiles</i>	Mar. 23	RVA20, RV120, RVA22
<i>"Zicntr" and "Zihpm" Counters</i>	Mar. 23	Zicntr, Zihpm
<i>RV32E and RV64E Base Integer Instruction Sets</i>	Ene. 23	RV32E/RV64E
<i>"Ztso" Standard Extension for Total Store Ordering</i>	Ene. 23	Ztso
<i>RISC-V Wait-on-Reservation-Set (Zawrs) extension</i>	Nov. 22	Zawrs
<i>Zmmul Extension</i>	Jun. 22	The Kite
<i>PMP Enhancements for memory access and execution prevention on Machine mode (Smepmp)</i>	Nov. 21	Smepmp
<i>RISC-V Base Cache Management Operation ISA Extensions</i>	Nov. 21	Zicbom, Zicbop, Zicboz
<i>RISC-V Bit-Manipulation ISA-extensions</i>	Nov. 21	Zba, Zbb, Zbc, Zbs
<i>RISC-V Count Overflow and Mode-Based Filtering Extension</i>	Nov. 21	Sscofpmf
<i>RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions</i>	Nov. 21	Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh, Zkn, Zks, Zkt, Zk, Zkr
<i>RISC-V State Enable Extension</i>	Nov. 21	The smstate
<i>RISC-V "stimecmp / vstimecmp" Extension</i>	Nov. 21	Sstc
<i>RISC-V Vector Extension</i>	Nov. 21	For 32x, for 32f, for 64x, for 64f, for 64d, for, zvl32b, zvl64b, zvl128b, zvl256b, zvl512b, zvl1024b, zvl, zvl
<i>The RISC-V Instruction Set Manual Volume II: Privileged Architecture</i>	Nov. 21	Sm1p12, Ss1p12, Sv57, Hypervisor, Svinval, Svnapot, Svpbmt
<i>"Zfh" and "Zfhmin" Standard Extensions for Half-Precision Floating-Point</i>	Nov. 21	Zfh, Zfhmin

Nombre de la especificación	Fecha ratificación	Nuevas extensiones o perfiles
"Zfinx", "Zdinx", "Zhinx", "Zhinxmin": <i>Standard Extensions for Floating-Point in Integer Registers</i>	Nov. 21	Zfinx, Zdinx, Zhinx, Zhinxmin
"Zhintpause" <i>Pause Hint</i>	Feb. 21	Zhintpause
<i>The RISC-V Instruction Set Manual Volume I: Unprivileged ISA</i>	Dic. 19	A, D, F, RV32I, RV64I, Zaamo, Zalrsc, Zicsr, Zifencei
<i>The RISC-V Instruction Set Manual Volume II: Privileged Architecture</i>	Dic.19	C, M, Q, Sm1p11, Ss1p11, B32, B39, B48

3.5. Extensiones ISA RISC-V recientes para diferentes aplicaciones

Además de las especificaciones ISA vistas anteriormente, existe una variedad de nuevas investigaciones que proponen extensiones personalizadas de RISC-V para diferentes aplicaciones encontradas en la bibliografía como internet de las cosas, inteligencia artificial, criptografía, computación de alto rendimiento, compresión de datos y seguridad. Siguiendo la clasificación de los escenarios encontrados, analizaremos las aportaciones de los estudios relacionados a continuación:

3.5.1. Internet de las cosas (IoT)

El Instituto Nacional de Estándares y Tecnología (NIST) ha propuesto a FALCON como uno de los algoritmos de firma digital estandarizados contra ataques cuánticos en 2022 [43]. En comparación con otros esquemas de criptografía post-cuántica (PQC), FALCON está diseñado para futuras aplicaciones de Internet de las cosas (IoT) debido al proceso de verificación de firma más rápido y con menor sobrecarga de transmisión. En este trabajo, proponen una extensión personalizada basada en el marco escalar-vectorial RISC-V para una implementación eficiente de FALCON. Este trabajo consiste en un diseño conjunto *hardware/software* para rutinas completas de generación y verificación de firmas FALCON. Además, incluyen un bloque *hardware* de muestreo gaussiano FALCON y un núcleo específico de dominio basado en la extensión vectorial RISC-V (RVV). La arquitectura propuesta acelera las operaciones del núcleo en FALCON, como el muestreo gaussiano discreto (Figura 27), la transformada teórica de números (NTT), la NTT inversa y las operaciones polinómicas.

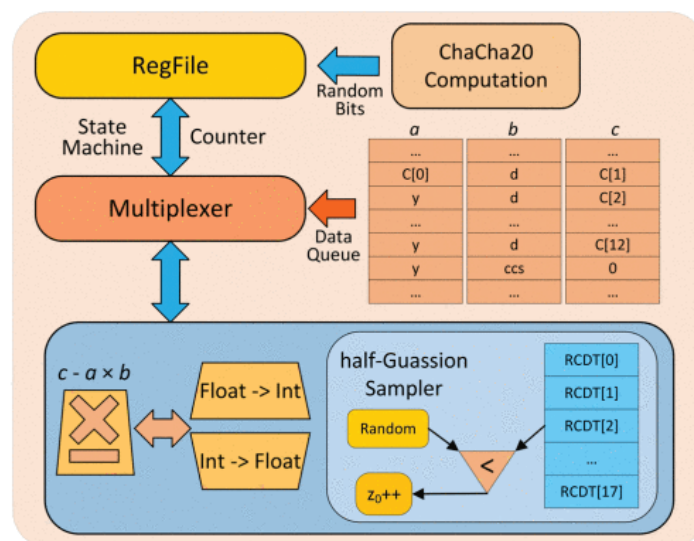


Figura 27. Diseño hardware de muestreo gaussiano discreto [43]

Los resultados experimentales fueron implementados en un *hardware* personalizado utilizando Chisel HDL 3.6.0 [44] y se evaluó el diseño *hardware* en la placa FPGA Xilinx UltraScale + ZCU104 utilizando Xilinx Vivado 2019.2. En comparación con la implementación de referencia, los resultados en la plataforma de simulación gem5-RTL [45] presentan una aceleración de la generación y verificación de firmas de hasta x18 y x6,9.

El trabajo presentado en [46] “*A RISC-V ISA Extension for Ultra-Low Power IoT Wireless Signal Processing*” aborda la creación de una extensión del conjunto de instrucciones para el RISC-V ISA (RV32IM) de código abierto dedicado a los transceptores IoT inalámbricos definidos por *software*, con objeto de trabajar en rangos muy bajos de potencia (*Ultra Low Power*). Las instrucciones personalizadas están adaptadas a las necesidades de aritmética compleja de enteros de 8/16/32 bits que suelen requerir las modulaciones en cuadratura. La extensión propuesta ocupa solo dos códigos de operación principales y la mayoría de las instrucciones están diseñadas para tener un costo de energía cercano a cero. Se utilizan tanto un modelo de precisión de instrucción (IA) como un modelo de precisión de ciclo (CA) de la nueva arquitectura para evaluar seis bancos de pruebas de procesamiento de banda base de IoT que incluyen demodulación FSK y detección de preámbulo LoRa.

Los resultados de la simulación muestran mejoras en el recuento de ciclos del 19 al 68 por ciento. Las simulaciones posteriores a la síntesis para una tecnología FDSOI de 22 nm de destino muestran menos del 1 por ciento de consumo de energía y del 28 por ciento de sobrecarga de área, respectivamente, en relación con un diseño RV32IM de referencia. Las simulaciones de potencia muestran un consumo máximo de potencia de 380 mW para la demodulación LE de Bluetooth y 225 mW para la detección de preámbulo LoRa (BW = 500 kHz, SF = 11).

En [47] se presenta una plataforma *hardware* RISC-V escalable para el procesamiento inteligente de sensores. Para ello, presentan un chip de demostración para una aplicación de detección de eventos de audio industrial desarrollado como parte del proyecto Scale4Edge. El proyecto tiene como objetivo habilitar un ecosistema integral basado en RISC-V para ensamblar de manera eficiente dispositivos de borde bien diseñados. En la Figura 28 se muestra una descripción general de los diferentes componentes y su interacción.

El ecosistema Scale4Edge cubre componentes y herramientas escalables y los extiende para componentes de borde específicos de la aplicación en tres niveles:

- 1) nivel de instrucción de CPU definido por la arquitectura del conjunto de instrucciones (ISA) RISC-V,
- 2) nivel de *software* definido por el último estándar de lenguaje de programación C C11 con compiladores y bibliotecas abiertas a estándares complementarios como MISRA-C, y
- 3) nivel de sistema operativo y *firmware* a través de servicios de sistema, interfaces de configuración y controladores.

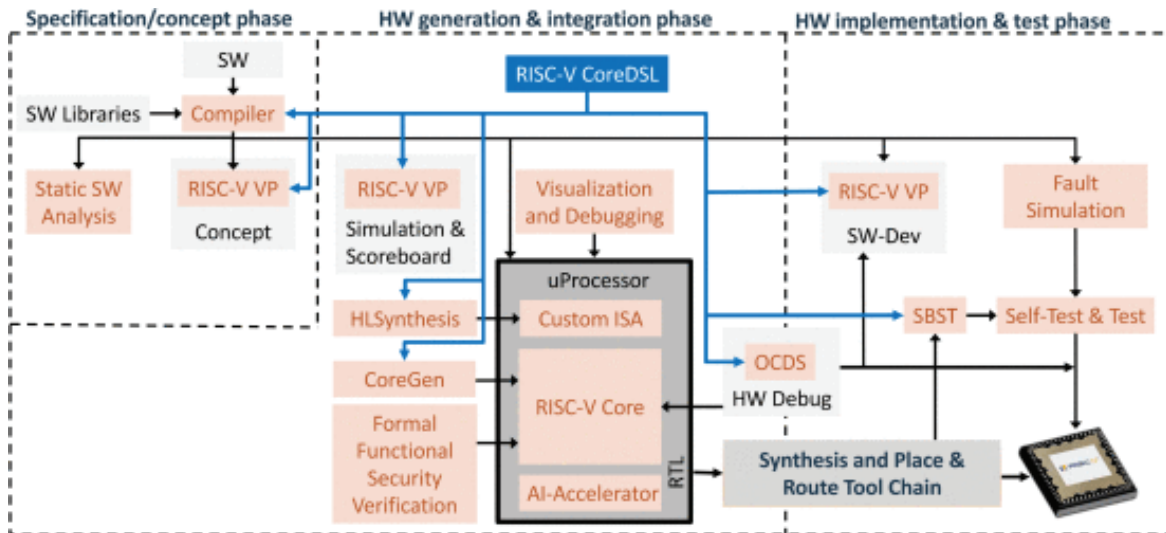


Figura 28. Diagrama general del ecosistema Scale4Edge [47]

La plataforma del ecosistema es personalizable para la aplicación individual a través de la ISA RISC-V, que puede definir instrucciones personalizadas opcionales, por ejemplo, para admitir instrucciones no interrumpibles para aplicaciones individuales. En consecuencia, Scale4Edge se basa en un *hardware* ampliamente escalable que aborda diferentes arquitecturas de canalización, arquitecturas multinúcleo, coprocesadores y aceleradores *hardware*, como para aplicaciones de IA y DSP.

El chip está fabricado con tecnología 22FDX de GlobalFoundries y contiene una CPU RISC-V con extensiones de arquitectura de conjunto de instrucciones (ISAX) personalizadas para un procesamiento rápido de IA y DSP, un acelerador de red neuronal de bajo consumo y un PLL escalable para cumplir con los requisitos de procesamiento en tiempo real. Mediante la integración automatizada de estos componentes *hardware* especializados, logran una aceleración de x2,15 al tiempo que reducen el consumo en un 27% en comparación con la solución no optimizada.

3.5.2. Inteligencia Artificial (IA)

La demanda de soluciones basadas en inteligencia artificial (IA) está aumentando exponencialmente en todos los campos de aplicación, incluidos los dispositivos de bajo consumo en el borde [48]. Sin embargo, debido a sus capacidades computacionales limitadas, estos dispositivos, que ejecutan unidades centrales de procesamiento (CPU) adaptadas a aplicaciones integradas, generalmente no están optimizados para ejecutar redes neuronales complejas. Proporcionar extensiones *ad-hoc* a la arquitectura del conjunto de instrucciones de un procesador RISC-V puede ser una solución viable para abordar este problema.

En [48] se propone el uso de la cadena de herramientas *PyTorch Graph Lowering (Glow)*-LLVM para comprender el impacto del código compilado de los modelos de IA en una máquina RISC-V y ampliar su conjunto de instrucciones para mejorar el rendimiento en tiempo de ejecución. Este enfoque permite la creación de perfiles de código, la detección de cuellos de botella computacionales y el aprovisionamiento de las mejoras de CPU necesarias para implementarse en el *backend* de LLVM antes de la implementación *hardware*.

Después de perfilar redes neuronales artificiales (RNA) conocidas cuantificadas a int8 (particularmente, un solo perceptrón, RESNET18, VGG11 y LENET5), han identificado y diseñado tres instrucciones adicionales, que llamaron LWM, LWA y LWS, que indican respectivamente *Load Word-and-Multiply*, *-Add* y *-Subtract*. Como resultado, se obtiene una descripción del procesador orientada a la IA de borde, significativamente mejorada en términos de tiempo de inferencia y densidad de programa, lista para ser diseñada por *hardware*.

Para imágenes RGB de 128×128, las extensiones personalizadas permitieron una velocidad hasta 13 veces mayor en comparación con RV32I y 5 veces mayor en comparación con RV32IM, con un máximo de 11,7 % de reducción en el tamaño del código. Junto con estos resultados, se destaca sistemáticamente los principales pasos metodológicos para incluir nuevas instrucciones en un *backend* LLVM. En la Figura 29, se puede comprobar los resultados obtenidos que demuestran la densidad de código para las redes comparadas.

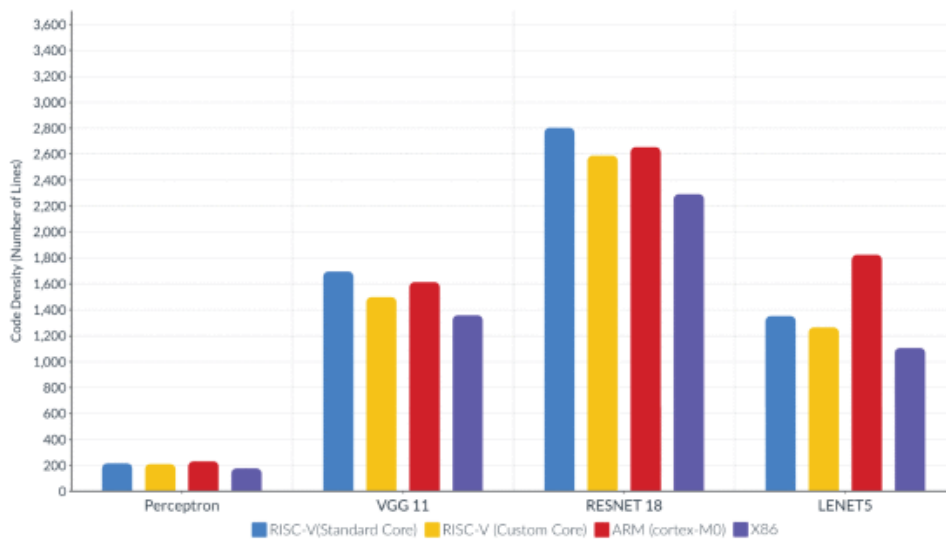


Figura 29. Comparación de la densidad de código de varias redes neuronales artificiales para ARM, x86, RV32I y RV32I con instrucciones personalizadas [48]

En los últimos años, se ha investigado sobre el uso de *hardware* personalizado para acelerar el aprendizaje profundo en dispositivos de borde [49]. Sin embargo, el flujo de extremo a extremo del aprendizaje profundo incluye preprocesamiento y posprocesamiento. Los aceleradores *hardware* de aprendizaje profundo no pueden acelerar estas operaciones, lo que, en consecuencia, se convierte en un cuello de botella de rendimiento en el flujo de ejecución. En este estudio, proponen métodos de optimización para mejorar el preprocesamiento y posprocesamiento en los dispositivos de borde. Para este propósito, se adopta *Tensor Virtual Machine* (TVM), un marco de compilación de aprendizaje automático de extremo a extremo. TVM proporciona un lenguaje *front-end* tipo script que permite a los usuarios escribir programas para preprocesamiento y posprocesamiento.

Se proponen estrategias de reescritura para mejorar el rendimiento de los operadores escritos en script híbrido a través de la extensión RISC-V *Packed SIMD* (extensión P). La extensión P define instrucciones específicas de subpalabra de instrucción única y múltiples datos (SIMD) que permiten realizar cálculos complejos de

manera eficiente en dispositivos de borde. En este estudio se diseñan instrucciones personalizadas basadas en la extensión RISC-V P para reescribir estrategias con el fin de acelerar las operaciones de aprendizaje profundo. Los resultados experimentales indican que los métodos usados mejoran el rendimiento en un factor de 1,28 a 15,29. La Figura 30, muestra el formato de instrucciones personalizado a partir de la extensión P que mejora el preprocesamiento y posprocesamiento de las operaciones.

	31	25	24	20	19	15	14	12	11	7	6	
Custom Instruction	funct7			rs2		rs1		funct3		rd		opcode
plus32	1			index		base addr		6		destination		11
psu32	1			source		index		6		base addr		43
quot8	1			divisor		dividend		0		destination		11

	31	20	19	15	14	12	11	7	6	
Custom Instruction	imm[31:20]			rs1		funct3		rd		opcode
plse8	stride			base addr		3		destination		11
psse8	stride			source		7		base addr		43

	31	15	14	12	11	7	6	
Custom Instruction	imm[31:15]			funct3		rd		opcode
pcsv8	Immediate[31:15]			7		destination		11

Figura 30. Formato de instrucción personalizado basado en la extensión P [49]

Durante la última década, se ha descubierto que las redes neuronales profundas (DNN) tienen un mejor rendimiento en las tareas de percepción visual y comprensión del lenguaje a medida que aumenta su tamaño [50]. Sin embargo, esto se produce a costa de un alto consumo de energía y de una gran necesidad de memoria para entrenar modelos con un alto número de parámetros. Como el entrenamiento de las DNN requiere un amplio rango dinámico para representar tensores, normalmente se utilizan formatos de punto flotante.

En este trabajo, se utiliza un formato de punto flotante de bloque (BFP) que reduce significativamente el tamaño de los tensores y el consumo de energía de las unidades aritméticas. Trabajos previos sobre el entrenamiento de DNN basado en BFP seleccionan empíricamente el tamaño de bloque y la precisión necesaria para mantener la precisión de los datos durante el entrenamiento. Para que el entrenamiento basado en BFP sea más eficiente, proponen un tamaño de bloque dinámico y escala de precisión (DBPS) para el entrenamiento de DNN. También se presenta un acelerador *hardware*, llamado núcleo DBPS, que admite el control DBPS configurando unidades aritméticas con instrucciones personalizadas extendidas en un procesador RISC-V en la placa Xilinx Virtex-7 VC707. Como resultado, el tiempo de entrenamiento y el consumo de energía se reducen en un 67,1% y un 72,0%, respectivamente, sin perjudicar la precisión del entrenamiento. La Figura 31, muestra un conjunto de instrucciones personalizadas que se extienden para configurar y ejecutar el núcleo DBPS para un entrenamiento DNN eficiente y en la Figura 32 se muestra el esquema de interpretación de las instrucciones extendidas.

Custom Instruction	func7	Acción
dbps_config	0 0 0 0 0 0 0	Configura el coprocesador
mvin_man_w	0 0 0 0 0 0 1	Carga de datos desde cache: mantisa, exponente, ...
mvin_exp_w	0 0 0 0 0 1 0	
mvin_man_i	0 0 0 0 0 1 1	
mvin_exp_i	0 0 0 0 1 0 0	
mvout	0 0 0 0 1 0 1	Escritura de datos a cache
bfp_gemm	0 0 0 0 1 1 0	Procesamiento de la DNN
batch_norm	0 0 0 0 1 1 1	
relu_pool	0 0 0 1 0 0 0	
flush	0 0 0 1 0 0 1	Inicialización controlador

Figura 31. Instrucciones personalizadas para el training de una DNN (Adaptada de [50])

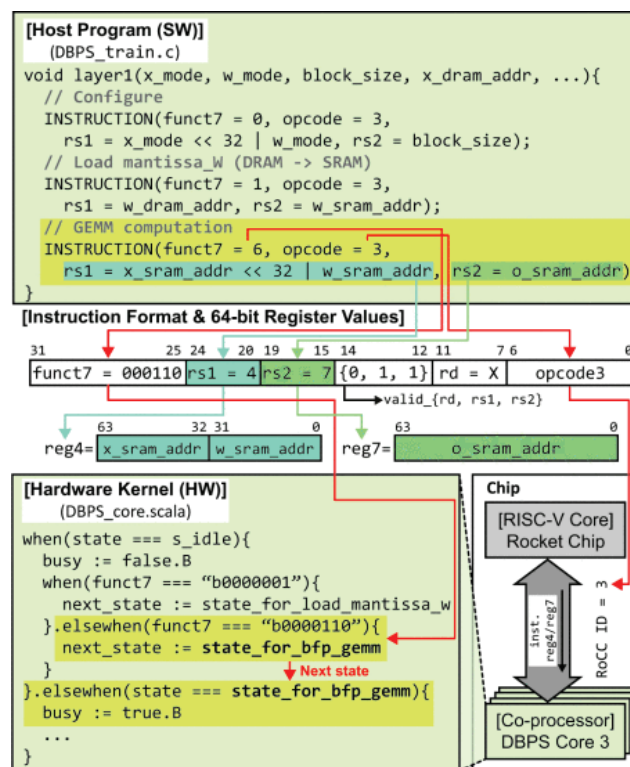


Figura 32. Interpretación de la extensión de las instrucciones [50]

3.5.3. Criptografía

En [51] se propone un diseño *hardware* para acelerar el algoritmo hash Blake3, una función hash criptográfica, en la arquitectura RISC-V, utilizando tanto la extensión vectorial (V1.0) como una instrucción Blake3 especialmente diseñada. El diseño aprovecha el paralelismo y la simplicidad de BLAKE3 para lograr un alto rendimiento y una baja latencia, implementando la función de compresión central utilizando una arquitectura de conjunto de instrucciones personalizada que admite operaciones vectoriales y computación constante. La función de compresión Blake3 se acelera en la arquitectura basada en el diseño de código de *hardware/software*. La capacidad de extensión de la instrucción personalizada RISC-V proporciona estabilidad y una plataforma fácil de usar. El diseño logra una aceleración 10 veces mayor para la función

de compresión en comparación con un solo núcleo de 2,7 GHz Xeon 8280 y un 10% más rápido que una implementación AVX512 (Figura 33).

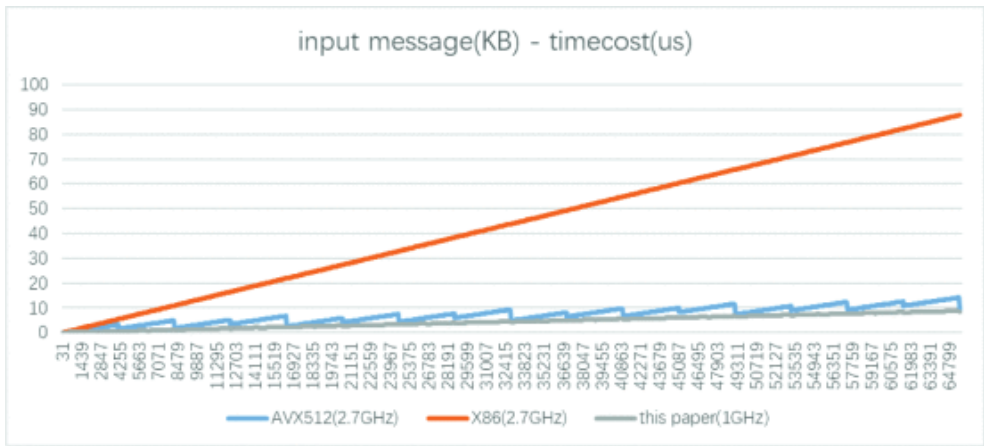


Figura 33. Resultados obtenidos (tiempo de compresión: menor => más rápido) [51]

En datos [52] se presenta una extensión criptográfica construida sobre la extensión vector para la aplicación eficiente de criptografía postcuántica basada en retículas, ofreciendo instrucciones personalizadas que pueden realizar operaciones vectorizadas en polinomios de longitud variable y ancho de datos. Para ello, utilizan una arquitectura de micro operaciones para simplificar la ejecución de instrucciones vectoriales de latencia variable y se proponen unidades aritméticas modulares fracturables para soportar operaciones en ancho variable para los coeficientes. Sobre esta base, se diseña una unidad vectorial, logrando una aceleración significativa en comparación con otros trabajos que utilizan la multiplicación de polinomios basada en transformadas de teoría de números.

Esta extensión criptográfica se integra además en el simulador gem5 para evaluar CRYSTALS-Kyber y CRYSTALS-Dilithium; los resultados superan las implementaciones de última generación con una mejora de más de 2,3x en el recuento de ciclos como se puede apreciar en la Figura 34. Para obtener la latencia de operación precisa y la sobrecarga de recursos, implementan la unidad vectorial personalizada propuesta utilizando Chisel y evalúan el diseño en la plataforma de prototipado Xilinx UltraScale+ ZCU111 con Xilinx Vivado 2019.2.

Design	CPU Model	CoreMark/MHz ^a	Instruction Set Architecture	Cycle Count (kCCs)							
				CRYSTALS-Kyber-512-CCA-KEM				CRYSTALS-Dilithium-II			
				KeyGen	Encaps	Decaps	Total Cycles ^b Ratio	KeyGen	Sign	Verify	Total Cycles ^b Ratio
This Work	Minor CPU	2.34	RISC-V+Enhanced RVV	22.8	23.8	31.1	1	82.7	159.2	84.9	1
RISQ-V [8]	CV32E40P	2.74	RISC-V + Scalar ISE	150.1	193.1	204.9	7.2	—	—	—	—
Sapphire [6]	Unknown	2.33 ^c	RISC-V + Scalar ISE	74.5	131.7	142.3	5.0	167.4	634.8	229.5	3.5
VPQC [9]	SCR1	2.24	RISC-V + Vector ISE	18.6	45.9	80	2.3	—	—	—	—
TCHES'22 [18]	Cortex-A72	5.4	Armv8 + Neon	62.5	80.7	76.4	2.9	269.7	649.2	272.8	3.8

^a Under GCC Compiler Optimization Option -O2.

^b Total Cycles refer to the cycles required for an encapsulation and decapsulation, or a signature signing and verification, as the key generation can be done offline.

^c Sapphire [6] does not present the CPU model, but mentions that the CPU performance is similar to Arm Cortex M0; thus the CoreMark score of Arm Cortex M0 is used.

Figura 34. Rendimiento medio de los algoritmos PQC Kyber y Dilithium [52]

En este estudio, el sistema NTRU, se diseñó y ejecutó en un procesador RV32IMC de código abierto [53]. Se desarrollaron nuevos candidatos a instrucciones y se integraron en la unidad funcional del procesador con un método de perfilado funcional. Para ello, en primer lugar, se diseña el criptosistema NTRU en lenguaje C como una

aplicación *software*. A continuación, se aplica un perfilado a nivel funcional con un método clásico para determinar los bloques utilizados con más frecuencia. Se diseñan e integran en el núcleo del procesador nuevas instrucciones que implementan las operaciones de los bloques detectados. Los diseños se prueban en FPGA y se comparan con otros para todas las combinaciones de instrucciones. Según los resultados obtenidos, aunque la utilización de recursos aumenta ligeramente el requisito de diseño, ha proporcionado una mejora importante en términos de rendimiento. Para resolver este problema, se utiliza una tarjeta FPGA, Nexys 4 DDR para implementar todo el proyecto y ejecutar el código C. Los resultados muestran en la Figura 35 que el criptosistema NTRU se puede acelerar en un 32,3% con este método simple.

Added Instruction	Clock Cycles	Improvement
VANILLA	329,281,688	—
MOD	255,087,137	−22.53%
ADD	310,181,046	−5.80%
EQU	345,059,480	+4.79%
MOD + ADD	224,556,529	−31.80%
MOD + EQU	321,277,395	−2.43%
ADD + EQU	312,095,182	−5.22%
MOD + ADD + EQU	222,705,262	−32.37%

Figura 35. Resultados de desempeño obtenidos [53]

3.5.4. Computación de alto rendimiento

SHA-3 se considera una de las funciones hash estandarizadas más seguras [54]. Se basa en la permutación Keccak-f[1.600], que opera sobre un estado interno de 1.600 bits, representados principalmente como una matriz de $5 \times 5 \times 64$ bits. Si bien las implementaciones existentes procesan el estado secuencialmente en fragmentos de típicamente 32 o 64 bits, la permutación Keccak-f[1.600] puede beneficiarse mucho de la aceleración mediante la paralelización. En este artículo se explora el potencial de la paralelización de Keccak-f[1.600] en procesadores basados en RISC-V a través de extensiones de vectores personalizadas para arquitecturas de 32 y 64 bits. Se analizan la permutación Keccakf[1.600], compuesta por cinco asignaciones de pasos diferentes, y se proponen diez instrucciones de vectores personalizadas para acelerar el cálculo. Se implementan estas extensiones en un procesador SIMD descrito en SystemVerilog. En la Figura 36 se compara el rendimiento de sus diseños con arquitecturas existentes basadas en procesadores de conjuntos de instrucciones específicos de la aplicación (ASIP) vectorizados y se demuestra que sus diseños presentan un alto rendimiento gracias a las instrucciones vectoriales personalizadas implementadas.

Implementation	Execution time		Throughput (bits /cycle)	Area (slices)	Throughput/Area (bits /(cycle × slices))
	cyc/rnd	cyc/byte			
LEON3 [13]	-	369	21.68×10^{-3}	8 648	2.51×10^{-6}
MIPS Native [14]	-	178.1	44.92×10^{-3}	6 595	6.81×10^{-6}
MIPS Coprocessor [14]	-	137.9	58.01×10^{-3}	7 643	7.59×10^{-6}
OASIP [12]	-	291.5	27.44×10^{-3}	981	27.97×10^{-6}
DASIP [12]	-	130.4	61.36×10^{-3}	1 522	40.31×10^{-6}
32-bit with LMUL=8 (EleNum=5, S N=1)	147	18.1	441.98×10^{-3}	6 359	69.5×10^{-6}
32-bit LMUL=8 (EleNum=15, S N=3)	147	18.1	$1\,325.97 \times 10^{-3}$	23 408	56.65×10^{-6}
32-bit LMUL=8 (EleNum=30, S N=6)	147	18.1	$2\,651.93 \times 10^{-3}$	48 036	55.2×10^{-6}

Figura 36. Resultados de las arquitecturas de 32 bits y comparación con arquitecturas de referencia de 32 bits [54]

Para verificar de manera eficiente un diseño de procesador RISC-V con extensiones de instrucciones personalizadas, en este trabajo se propone un *framework* de verificación que combina los beneficios de la abstracción de alto nivel de la simulación C/C++ y el modelado con precisión de ciclo de las cosimulaciones C-RTL[55]. Para ello asignan el núcleo RISC-V y los aceleradores de instrucciones personalizados generados por HLS, junto con sus modelos *software* C/C++ correspondientes, al mismo dispositivo FPGA con los procesadores acelerados que les permiten ejecutarse simultáneamente. Un monitor y verificador global verifica cuidadosamente los resultados tanto del *hardware* como del *software* en tiempo real (Figura 37). Si se detecta una discrepancia, se captura una instantánea de todo el *hardware* y se reconstruye la simulación en simuladores *software* externos para una depuración detallada. A través de una serie de experimentos de referencia, los resultados muestran una mejora significativa del rendimiento en comparación con los enfoques convencionales de x1419 a x9011.

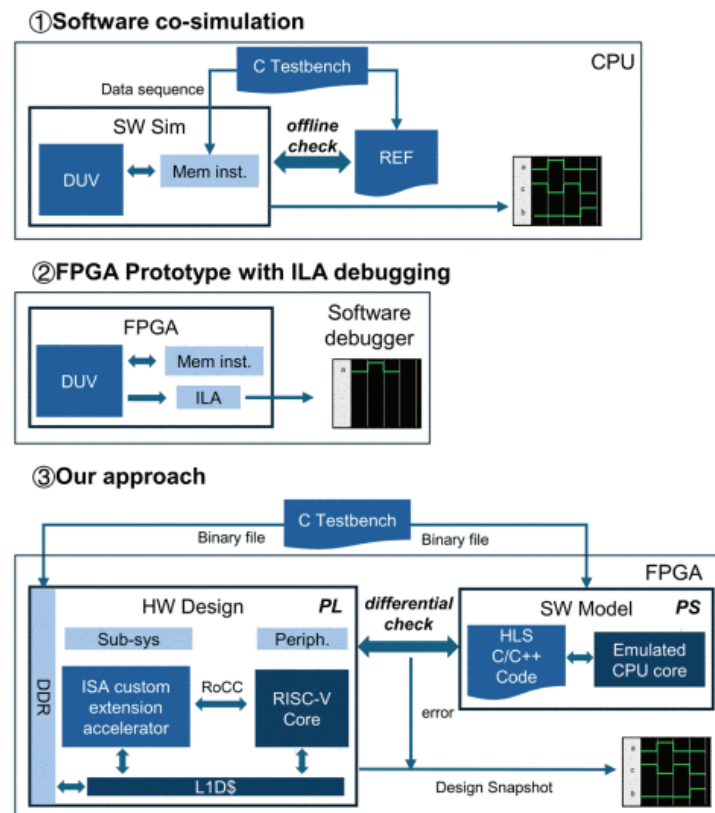


Figura 37. Comparación entre los enfoques de verificación convencionales y el enfoque propuesto[55]

En RISC-V, el sistema numérico se representa en varios formatos de aritmética de números reales [56]. La representación de números reales se realiza siguiendo el estándar de punto flotante IEEE 754. Tiene muchos impactos como el redondeo, el exceso de No es un número (NaN), cero con signo. Al combinar estos aspectos utilizando el formato de número posit, se desarrolla la unidad de procesamiento posit.

El sistema numérico Posit es una alternativa a la representación de punto flotante IEEE754. Modifica el estándar RISC-V ISA que permite la conversión de posiciones de 8 o 16 bits a números de coma flotante IEEE de 16 bits para obtener un rango más dinámico dentro del número dado de bits. El modelo propuesto consiste en

hacer el chip más pequeño y reducir el consumo de energía. Los conceptos de baja precisión presentan una solución mejorada que las estrategias de computación aproximada que intentan aceptar una calidad de respuesta inferior. En comparación con los números flotantes del mismo tamaño, las posiciones de alta precisión ofrecen decimales más exactos. Las posiciones evitan el error ilógico de asignar un valor al NaN (no es un número). En la Figura 38, se muestran que los resultados obtenidos del consumo de energía de las entradas en la unidad sumadora RISC V con representación de número posit se reduce a 217 mW, mientras que la unidad sumadora RISC V con representación de número de punto flotante requería 257 mW.

With Floating point (mW)	With Posit (mW)	Percentage Reduction
257	217	15.56%

Figura 38. Análisis de potencia obtenidos[56]

3.5.5. Compresión de datos

En [57] realizan una mejora de la extensión Vector RISC-V (RVV) para la compresión de datos. Para ello analizan cinco algoritmos de compresión comunes y diseñan una clase de instrucciones de adyacencia vectorial para la aceleración de la vectorización para bucles de puntos calientes en algoritmos de compresión que no pueden usar la vectorización RVV. El diseño de una unidad aritmética vectorial desacoplada para estas instrucciones que puede completar cálculos con bucles dependientes de datos de una manera no bloqueante. El procesador vectorial de código abierto Ara se utiliza para implementar estas ideas usando System Verilog y se sintetiza e implementa en el Alveo U50. Los resultados muestran en la Figura 39 que el trabajo aporta una aceleración máxima de 13,24x en el recuento de ciclos.

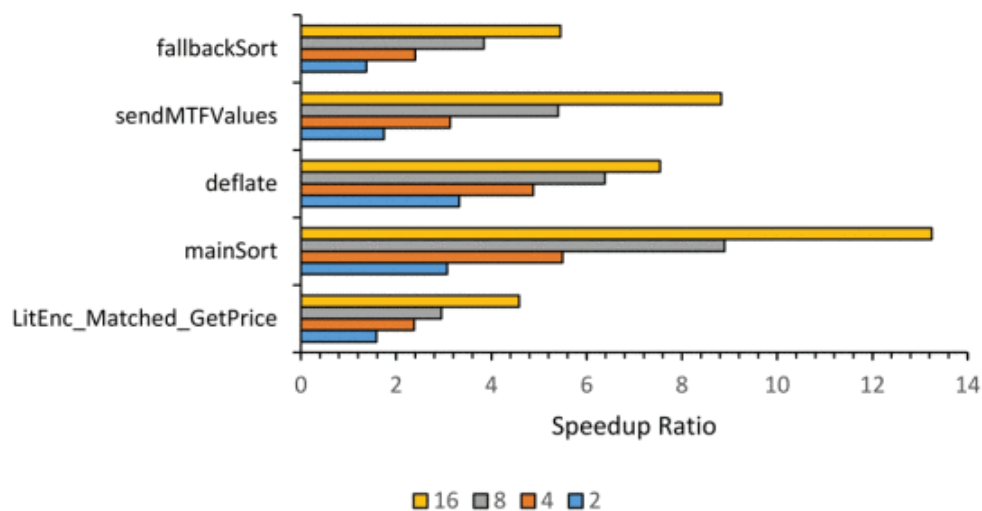


Figura 39. Estadísticas sobre la relación de aceleración debido a instrucciones de adyacencia vectorial[57]

3.5.6. Seguridad

Las máquinas virtuales (VM) deben ser extremadamente eficientes y, por lo tanto, utilizan motores complejos como un compilador JIT para acelerar el bucle habitual

de interpretación de bytecode [58]. Su uso de tareas de bajo nivel y críticas para la seguridad las convierte en objetivos de elección. La aplicación de un aislamiento de memoria de grano fino de bajo coste ha sido un foco de investigación importante como contramedida a los ataques JIT más avanzados. El aislamiento de memoria divide los componentes de una aplicación con comunicación controlada y acceso verificado a otros recursos. Este artículo vincula las instrucciones de seguridad en RISC-V desde el código generado por el compilador JIT al procesador y se presenta una aplicación de la extensión de seguridad RIMI a un procesador *softcore* RISC-V y un estudio del impacto de las instrucciones añadidas en el rendimiento y la intrusión del procesador. Se presenta cómo las instrucciones personalizadas vinculadas a la comprobación de dominios aplicada por *hardware* podrían proteger el código y los datos JIT. Presentan soluciones incrementales y sus instrucciones personalizadas correspondientes. El código de máquina generado y el RISC-V Rocket extendido tienen un bajo coste tanto en rendimiento como en intrusión como se puede observar en la Figura 40.

	Bare Rocket	Modified Rocket	
ALU	3.584	3.543	(- 1.14%)
BPU	0.177	0.176	(- 0.56%)
CSRs	2.087	2.315	(+ 10.92%)
div	10.235	10.507	(+ 2.66%)
ibuf	4.327	4.674	(+ 8.02%)
Overall power	30.919	32.285	(+ 4.42%)

Figura 40. Potencia global de los submódulos obtenidos[58]

ChaCha es un cifrado de flujo de alto rendimiento diseñado con el objetivo de garantizar altos márgenes de seguridad y, al mismo tiempo, lograr un alto rendimiento en plataformas *software*[59]. En este documento se propone un ISE liviano para admitir ChaCha en arquitecturas RISC-V. Este enfoque apunta a sistemas informáticos integrados, como dispositivos de borde de IoT que no admiten un motor vectorial. El ISE propuesto está diseñado para acelerar el cálculo de la función de bloque ChaCha y alinearse con los principios de diseño de RISC-V. Se demuestra que los ISE propuestos ayudan a mejorar la eficiencia de la función de bloque ChaCha. La implementación asistida por ISE del cifrado ChaCha acelera al menos 5,4 veces y 3,4 veces en comparación con la implementación de referencia de OpenSSL y la implementación optimizada basada en ISA, respectivamente. Para cifrar mensajes cortos, la implementación asistida por ISE obtiene un rendimiento comparable en comparación con las implementaciones que utilizan extensiones de vector con una sobrecarga de área muy alta como se muestra en la Figura 41.

Message size	Baseline OpenSSL	RV64I V ₄	RV64IB V ₅	ISE V ₃	128 bit Vector Vector1	128 bit Vector Vector2	256 bit Vector Vector1	256 bit Vector Vector2
64 bytes	2825	1768	1129	523	2001	607	2001	615
128 bytes	5555	3486	2207	989	2001	1182	2001	615
256 bytes	11015	6922	4363	1921	2001	2332	2001	1191
512 bytes	21935	13794	8675	3785	3748	4632	2001	2343
1024 bytes	43775	27538	17299	7716	7242	9232	3748	4647

Figura 41. Comparación del rendimiento de cifrado/descifrado en el número de instrucciones para diferentes tamaños de mensajes entre la línea base, la implementación optimizada basada en ISA, la implementación asistida por ISE y diferentes implementaciones de vectorización [59]

3.6. Conclusiones

En este capítulo se han presentado las principales extensiones de la ISA RISC-V que están disponibles en la actualidad, tanto desde el punto de vista de las extensiones estandarizadas o propuestas para arquitecturas especializadas. Se revisan distintos tipos de extensiones, indicando los mecanismos que soportan y el proceso de creación de las extensiones indicadas.

Capítulo 4. Metodología *software* RISC-V

En el dominio *software*, la comunidad de RISC-V ha generado una variedad de herramientas y recursos sofisticados tales como simuladores, emuladores, sistemas operativos específicos que facilitan el desarrollo y la acogida de esta arquitectura abierta. En los siguientes apartados se detallan algunos de los sistemas que permiten adaptar el conjunto de instrucciones según las necesidades de una aplicación específica optimizando el rendimiento y la eficiencia.

4.1. Simuladores y emuladores

4.1.1. Jupiter

Júpiter [60][61] es un ensamblador y simulador para la arquitectura RISC-V. Esta herramienta de código abierto orientada a la educación está escrita en Java y permite simular las instrucciones de enteros ISA (extensión I) más las extensiones M y F (RV32IMF). Además, incluye todas las pseudoinstrucciones descritas en el manual del conjunto de instrucciones a nivel de usuario. En la Figura 42, se puede apreciar la interfaz en GUI Mode.



Figura 42. Jupiter GUI Mode

Entre sus características se puede destacar:

- Facilidad de uso, dado que se diseñó con enfoque en la educación y para cualquier persona que desee aprender RISC-V. Tiene dos modos de operación (interfaz de línea de comandos e interfaz gráfica de usuario) diseñados de forma intuitiva y fáciles de usar.

- Modularidad, permite compilar y simular varios archivos a la vez, basta con especificar una etiqueta global que sea el punto de partida principal del programa. Esto permite la modularidad y facilita la creación de proyectos y laboratorios.
- Retroalimentación, el simulador muestra diferentes tipos de errores, tales como: errores de sintaxis, intentos de acceso a memoria reservada o intentos de escritura en memoria de sólo lectura.
- Multiplataforma, se encuentra disponible para Linux (Ubuntu), macOS y Windows.

4.1.2. Venus

Venus [62] es un simulador de conjunto de instrucciones RISC-V creado para la educación y basado en web por la Universidad de California Berkeley. Se caracteriza por:

- Simular extensiones RV32IM
- Depurar en un solo paso con la función deshacer y depuración de puntos de interrupción
- Permite visualizar el código de maquina y las instrucciones originales en paralelo
- Varios ecalls, incluyendo print y sbrk
- Visualización del contenido de la memoria desde la vista “Venus: Open Memory”
- Sistema básico de gestión de paquetes externos

Venus está integrado con una extensión de VS Code [63] que permite ejecutar todo dentro de VS Code (Figura 43).

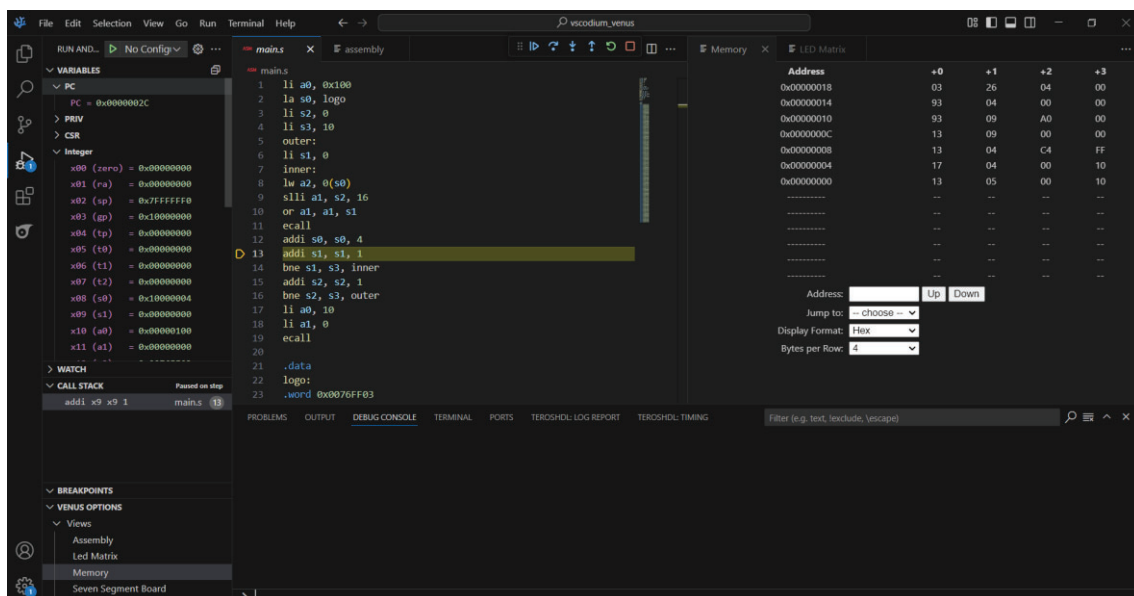


Figura 43. Ejecución de Venus en VS Code

4.1.3. RARS

El ensamblador y simulador RARS (*RISC-V Assembler and Runtime Simulator*) [64] está escrito en Java y se basa en el simulador original MARS 4.5, adaptado para la ejecución de programas en lenguaje ensamblador RISC-V. Su objetivo principal es ser un entorno eficaz para los usuarios que se inician en RISC-V. Sus principales características son las siguientes:

- Soporta las arquitecturas base RISC-V IMFDN (RISCV32 y RISCV64).
- Permite múltiples llamadas al sistema que coinciden con el comportamiento de MARS o SPIKE.
- Soporta la depuración mediante puntos de interrupción y/o *breakpoints* y/o *breaks*.
- Compara la expansión de código de pseudo-instrucción a código máquina con pasos intermedios.
- Soporta en ensamblado de múltiples archivos usando archivos abiertos o un directorio.

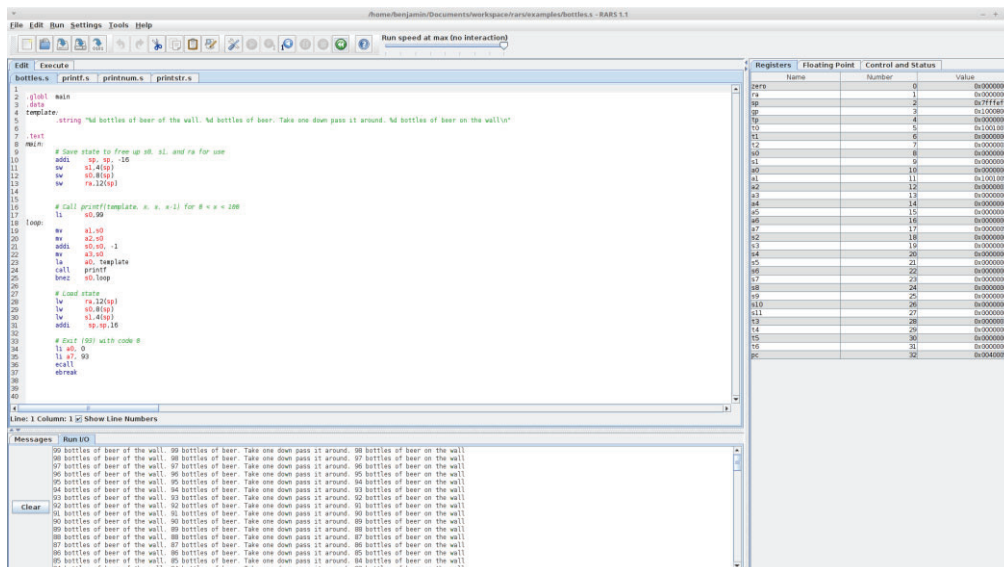


Figura 44. Interfaz del simulador RARS [64]

4.1.4. Spike

El simulador ISA Spike [65] desarrollado por RISC-V, sirve como modelo de referencias para sus procesadores capaz de implementar un modelo funcional de uno o más *cores* RISC-V.

Spike soporta las especificaciones de la ISA base de 32 y 64 bits (RV32I y RV64I y RV32E y RV64E), así como las distintas extensiones. En [66] se listan todas las extensiones soportadas, incluyendo la versión actualizada. Los proyectos se versionan para indicar cuándo se ha ampliado la API o se ha vuelto incompatible. En ese sentido, Spike sigue el esquema de versiones SemVer [67], en el que:

1. Los números de versión principales se incrementan cuando se realizan cambios en la API que son incompatibles con versiones anteriores,
2. Los números de versión secundaria se incrementan cuando se agregan nuevas API.
3. Los números de versión de parche se incrementan cuando se corrigen errores de manera compatible con versiones anteriores.

El simulador Spike puede ejecutar programas RISC-V en un *host* x86 o ARM.

4.1.5. Ripes

Ripes[68] es un simulador visual que se puede usar en línea para editar código ensamblador. Este simulador se utiliza para: ejecutar el código de máquina en una variedad de microarquitecturas (basadas en RV32IMC/RV64IMC), conocer la influencia de los diferentes diseños de caché en el rendimiento, compilar y ensamblar el código C y el código ensamblador para obtener un código máquina ejecutable y la posibilidad de interactuar un procesador con la E/S asignada a la memoria (Figura 45).

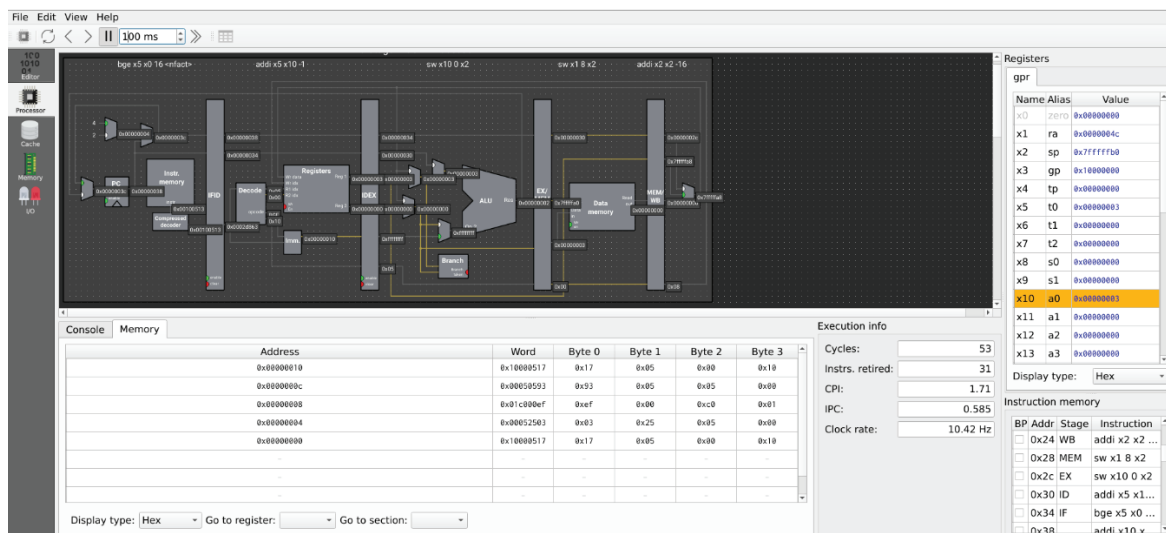


Figura 45. Simulador Ripes [68]

4.1.6. QEMU

QEMU[69] es un emulador y virtualizador de máquina versátil con espacio de usuario genérico de código abierto capaz de emular una máquina completa en *software* sin necesidad de soporte de virtualización de *hardware*. QEMU también puede integrarse con los hipervisores Xen y KVM para proporcionar *hardware* emulado mientras permite que el hipervisor administre la CPU. Con el soporte del hipervisor, QEMU puede lograr un rendimiento casi nativo para las CPU. Cuando QEMU emula las CPU directamente, es capaz de ejecutar sistemas operativos hechos para una máquina (por ejemplo, una placa ARMv7) en una máquina diferente (por ejemplo, una placa de PC x86_64).

QEMU también es capaz de proporcionar virtualización de la API del espacio de usuario para interfaces de *kernel* de Linux y BSD. Este emulador tiene como objetivo

adaptarse a una variedad de casos de uso. Puede ser invocado directamente por usuarios que deseen tener control total sobre su comportamiento y configuración. También tiene como objetivo facilitar la integración en capas de gestión de nivel superior, proporcionando una interfaz de línea de comandos estable y una API de monitorización.

4.2. Herramientas de desarrollo

4.2.1. Compiladores C/C++

GCC (GNU Compiler Collection) [70] es una familia de compiladores que se puede usar para compilar *software* en una amplia gama de plataformas incluido RISC-V. Incluye interfaces para C, C++, Objective-C, Fortran, Ada, Go, D y Modula-2, así como bibliotecas para estos lenguajes (libstdc++,...). El compilador cruzado de C y C++ RISC-V GNU Compiler Toolchain [71] admite dos modos de compilación: una cadena de herramientas genérica ELF/Newlib y una cadena de herramientas Linux-ELF/glibc más sofisticada.

Clang/LLVM [72] contiene un conjunto de herramientas para la construcción de compiladores, optimizadores y entornos de tiempo de ejecución optimizados. Este proyecto se diferencia por varios componentes: un núcleo denominado LLVM y el *frontend* Clang usados por los lenguajes similares a C. LLVM contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para procesar representaciones intermedias y convertirlas en archivos de encabezado necesarios para procesar representaciones intermedias y convertirlas en archivos de objeto. Por otro lado, Clang compila código C, C++, Objective-C y Objective-C++ en código binario LLVM y, desde allí, en archivos de objetos mediante LLVM. Otros componentes incluyen la biblioteca estándar C++ libcpp, el enlazador LLD y más.

4.2.2. IDEs

GNU MCU Eclipse RISC-V Embedded GCC [73][74] es una distribución de herramientas GCC para dispositivos RISC-V, que complementa la distribución oficial RISC-V mantenida por SiFive. Este compilador se caracteriza por proporcionar binarios para todas las plataformas principales (Windows 64/32 bits, GNU/Linux 64/32 bits, macOS), permite una instalación uniforme y portátil dado que la cadena de herramientas se encuentra disponible como un xPack binario y se puede instalar fácilmente con xpm. Asimismo, la cadena de herramientas se puede usar fácilmente en entornos de pruebas por su soporte mejorado para el uso de integración continua.

Freedom Studio SDK [75] se trata de una herramienta completa para el desarrollo de *software* en procesadores RISC-V de SiFive. Freedom Studio incluye los complementos, herramientas y visualizadores para proporcionar información *software* crucial sobre el núcleo del SoC. Como características se puede destacar que posee una colección de compiladores GNU (GCC) integrada y optimizada para RISC-V brindando un soporte de compilación para C, C++, Fortran y otros lenguajes; proporciona un simulador QEMU que realiza pruebas y verificaciones de *software* sin *hardware* real, mejorando la eficiencia del desarrollo; contienen el sistema operativo en tiempo real (RTOS) Linaro Lite para conseguir un diseño de sistema integrado eficiente; proporciona paquetes de

soporte de placa preconfigurados para una variedad de placas de desarrollo, permitiendo a los desarrolladores comenzar fácilmente.

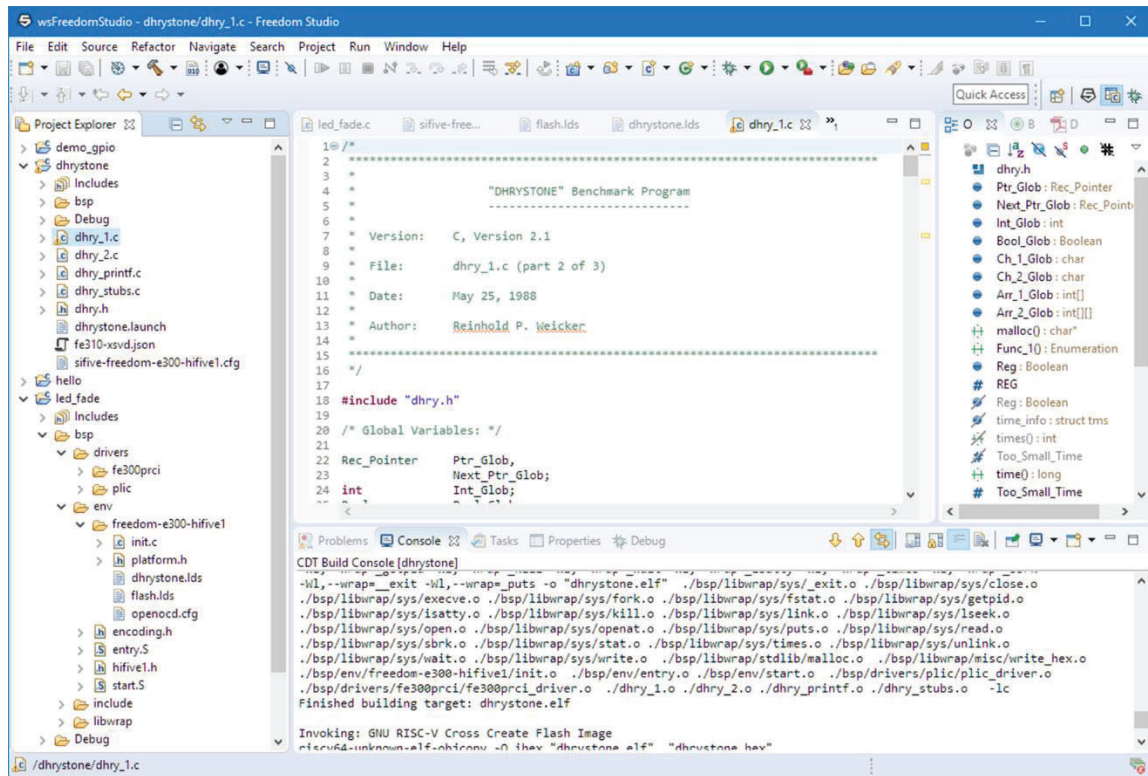


Figura 46. IDE Freedom Studio [75]

4.3. Sistemas Operativos y RTOS

Linux Foundation [76] lanza el proyecto del ecosistema de *software* RISC-V (RISE[77]) con la finalidad de acelerar el desarrollo de *software* de código abierto para RISC-V mediante un conjunto de herramientas. En cuanto a los sistemas operativos Linux ofrece un conjunto muy amplio de distribuidores compatibles con RISC-V: Fedora[78], Debian[79], OpenMandriva[80], openSUSE[81], Gentoo[82], Parabola GNU/Linux-libre[83], Ataraxia Linux[84], Slackware Linux[85] y Ubuntu[86].

FreeRTOS [87][88] es un sistema operativo en tiempo real (RTOS) líder en el mercado para microcontroladores y microprocesadores pequeños. Está distribuido libremente bajo la licencia de código abierto MIT, incluye un núcleo y un conjunto creciente de bibliotecas adecuadas para su uso en todos los sectores de la industria lo que facilita la integración con RISC-V. El portado para RISC-V de FreeRTOS es extensible lo que permite proporcionar un puerto de base que maneja los registros comunes a todas las implementaciones de RISC-V y un conjunto de macros que se deben implementar para manejar las características y extensiones específicas de la implementación de *hardware*, como registros adicionales.

El sistema operativo RTOS escalable Zephyr [89] admite múltiples arquitecturas *hardware*, está optimizado para dispositivos con recursos limitados y está diseñado teniendo en cuenta la seguridad. Se basa en un núcleo de tamaño reducido diseñado para usarse en sistemas con recursos limitados: desde simples sensores ambientales

integrados hasta puertas de enlace inalámbricas IoT. El *kernel* Zephyr admite múltiples arquitecturas, tales como ARM (Cortex-A, Cortex-R, Cortex-M), Intel x86, ARC, Nios II, Tensilica Xtens y RISC-V, SPARC, MIPS y una gran cantidad de placas compatibles.

4.4. Conclusiones

En este capítulo se describen distintas herramientas disponibles para abordar el proceso de compilación y depurado de las aplicaciones desarrolladas para RISC-V. En la literatura se puede encontrar un conjunto significativo de herramientas desarrolladas principalmente para docencia e investigación. Es importante destacar la necesidad de consolidar distintos IDE, librerías y *stack* para proporcionar un entorno estable que consolide el ecosistema *software* necesario para que RISC-V avance como plataforma de referencia, especialmente en el ámbito de la IA. El proyecto RISE [77] en el que participan la mayor parte de las empresas del sector realiza la labor de definición de estándares en este sentido.

Capítulo 5. Revisión de IPs disponibles para RISC-V

En los últimos años, las aplicaciones informáticas actuales con un uso intenso de computación superan con creces los recursos de los procesadores de un solo núcleo. Los avances en arquitecturas *hardware* especializadas han ayudado a la industria y a la comunidad investigadora a abordar la potencia de cálculo necesaria para aplicaciones y algoritmos de inteligencia artificial (IA) más mejorados y con uso intensivo de computación que ya han alcanzado un crecimiento sustancial.

Para abordar este problema, los procesadores multinúcleo junto con técnicas de computación paralela se han convertido en una solución para aumentar la potencia informática. Asimismo, los multiprocesadores son esenciales para soportar nuevas tecnologías y nuevos desafíos en las aplicaciones científicas. En particular, la arquitectura RISC-V a través de la contribución de una comunidad de investigadores y usuarios comerciales han generado un ecosistema de procesadores de alto rendimiento.

En este capítulo, se detalla los diferentes núcleos que admiten *hardware* basado en RISC-V para crear una clasificación de sistemas en chip (SoC) y núcleos de CPU de código abierto, comerciales, procesador basado en FPGA dado que la mayoría de los procesadores se dirigen a ASIC y basados en plataformas [90].

5.1. Núcleos de CPU y SoC

5.1.1. VexRiscv

VexRiscv [91] es una implementación central de procesador RISC-V altamente parametrizable compatible con el conjunto de instrucciones base RV32I, que opcionalmente se puede ampliar hasta el conjunto de instrucciones RV32IMAFDC. Está particularmente optimizado para FPGA, ya que elimina la necesidad de utilizar IP específicas del proveedor y tiene una utilización baja de LUTs para mejorar la portabilidad. El flujo de diseño parte de SpinalHDL[92] y se exporta a VHDL o Verilog. También se puede parametrizar para que sea compatible con FPU de 32 o 64 bits. Se proporcionan dos SoC basados en VexRiscv: Briey SoC y Marex SoC. Por ejemplo, ambos SoC generarán diseños más complejos con funcionalidad SDRAM y VGA capaces de ejecutar sistemas operativos Linux.

Adicionalmente, en la lista de *cores* presentada por RISC-V se encuentra VEXRiscv Plugins para B y K [93] un complemento rápido que permite agregar un subconjunto de la extensión B y K al núcleo de VexRiscv. Este complemento se ha creado como parte de la Iniciativa Europea de Procesadores (EPI) y actualmente es únicamente compatible

para RV32. Finalmente, la arquitectura VexRiscv es muy modular y ofrece soporte de complementos opcionales para funcionalidades adicionales, como aceleración de cifrado/descifrado AES, función de depuración JTAG y muchas otras.

5.1.2. Rocket Core

Rocket Core [16] es un núcleo escalar RISC-V desarrollado en la Universidad de California, Berkeley (UCB) e implementa las extensiones ISA RV32G y RV64G. También se admiten FPU opcionales y controladores de memoria caché con tamaños configurables. Los núcleos Rocket se generan mediante el generador de chips Rocket, que está basado en el lenguaje Scala y utiliza el compilador Chisel [94]. Rocket Custom Co-Processor (RoCC) es un coprocesador de arquitectura vectorial personalizado que se implementa como una extensión RISC-V no estándar. Con Rocket Chip Generator, se pueden organizar múltiples instancias de núcleos, controladores de caché y aceleradores de Rocket en mosaicos contiguos para crear una variedad de diseños de SoC.

5.1.3. BOOM

Berkeley Out-of-Order Machine (BOOM) [95] es otro procesador de código abierto derivado de UCB. La arquitectura original se basa en Rocket. La última implementación es SonicBOOM[96] (BOOM v3), que solucionó los cuellos de botella de iteraciones anteriores tal como se identificaron en la unidad de búsqueda de instrucciones, el *backend* de ejecución y la unidad de carga/almacenamiento. SonicBOOM se basa en FPGA, se fabrica comercialmente a 1 GHz y logró un rendimiento de nivel comercial. En la Figura 47 se muestran las implementaciones desarrolladas con una mejora en los saltos de ciclos.

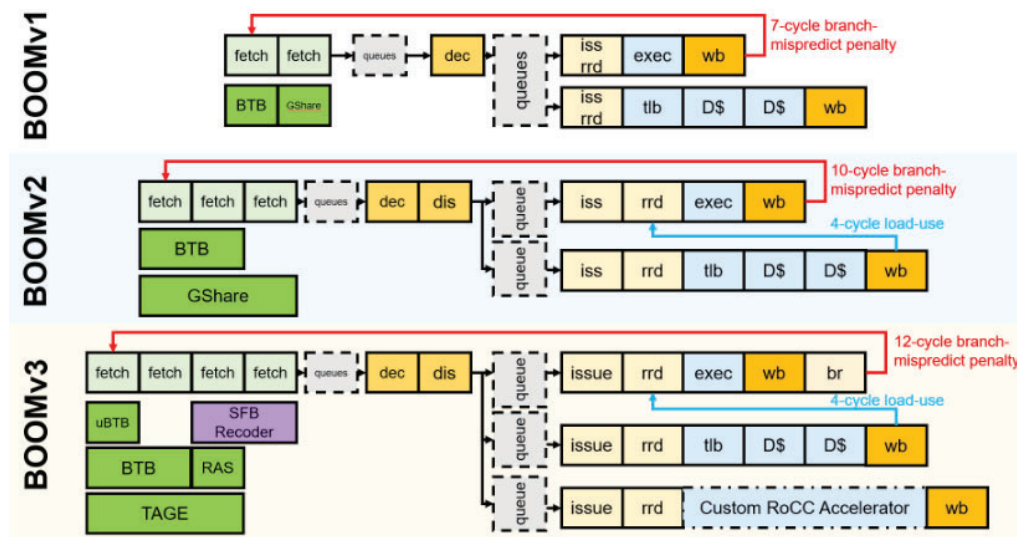


Figura 47. Diferentes implementaciones de BOOM [96]

5.1.4. PicoRV32

PicoRV32 [97] admite hasta un conjunto de instrucciones RV32IMC y ocupa poco espacio (<2K LUT). Este diseño RISC-V también puede implementar el núcleo RV32E para diseños aún más pequeños e integrados (MCU). La extensión M es compatible

opcionalmente a través del coprocesador Pico (PCPI) para proporcionar instrucciones sin ramificación a través de una implementación de coprocesador independiente. PicoRV32 tiene un tamaño altamente optimizado para implementaciones FPGA y ASIC (Figura 48).

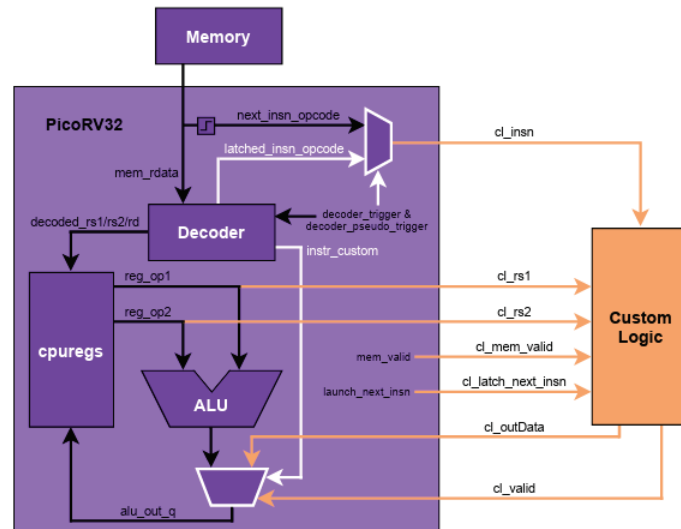


Figura 48. Núcleo PicoRV32 conectado a una lógica personalizada [98]

5.1.5. NEORV32

NEORV32 [99] es un núcleo que implementa la ISA RISC-V y distintas extensiones altamente configurable y con todas las funciones con documentación completa y múltiples ejemplos de implementación de FPGA para ayudar a principiantes y usuarios experimentados. El núcleo NEORV32 se puede integrar con varios periféricos (UART, SPI, 1-Wire, PWM, Watchdog) y un bus de comunicación (Wishbone/AXI) para construir un SoC personalizable. El proyecto apunta a aplicaciones de microcontroladores y es compatible con Zephyr [100] y FreeRTOS [101]. En la Figura 49 se muestra la arquitectura del procesador.

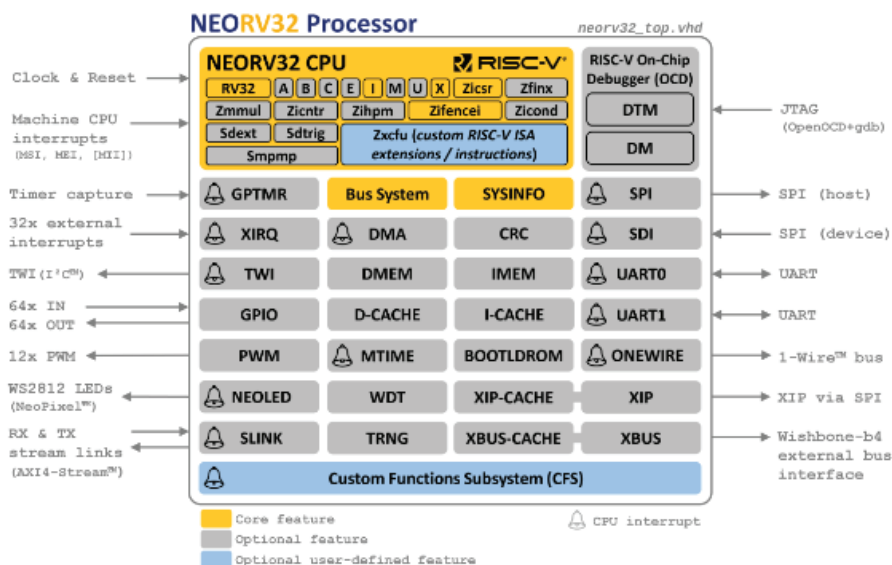


Figura 49. Estructura del procesador NEORV32 [99]

NEORV32 soporta distintos mecanismos para la integración de módulos externos capaces de soportar la ejecución mediante aceleración hardware de las extensiones del juego de instrucciones de RISC V. En la Tabla 8 se muestran distintas capacidades soportadas en función del mecanismo de integración utilizado. En la Figura 50 se muestra la jerarquía de buses del procesador NEORV32, mostrando la ubicación del XBUS. Veremos más adelante que para la implementación como bloque de Vivado, el XBUS se muestra como una interfaz AXI4 (el uso de la cache es configurable) y la interfaz SLINK (ofrece canales independientes TX y RX para enviar y recibir datos en forma de *streaming*, y que contienen FIFO locales) como AXI4Stream.

Tabla 8. Alternativas de integración de aceleradores hardware en NEORV32 (adaptado de [102])

	<i>Custom Functions Unit (CFU)</i>	<i>Custom Functions Subsystem (CFS)</i>	<i>External Bus Interface (XBUS)</i>
Ubicación del Código RTL	Integrado CPU	Integrado procesador	Externo al procesador
Complejidad HW	Pequeña	Media	Grande
Independencia de operación de la CPU	no	Sí	Sí
Interfaz con la CPU	Acceso a través del fichero de registros	Mapeado en memoria a través de un bus	Mapeado en memoria a través de un bus
Mecanismo de acceso de bajo nivel	Instrucciones a medida	load/store	load/store
Latencia de acceso	Baja	Media	Media/alta
Interfaces de E/S Externas	No soportadas, integradas en la CPU	Sí, capacidad limitada	Sí, definidas por el diseñador
Gestión de excepciones	Sí	No	No
Capacidad de interrupciones	No	Sí	Definidas por el diseñador

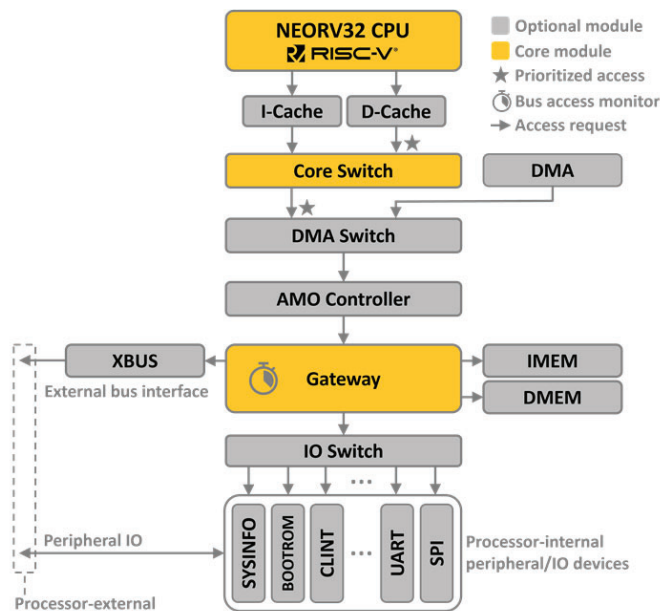


Figura 50. Jerarquía de buses de NEORV32 [99]

5.1.6. NaxRiscv

NaxRiscv [103] es un *kernel* de código abierto que admite la ejecución fuera de orden que incluye distintas extensiones ISA, incluidas instrucciones RV32I y RV64I. El núcleo se construye usando SpinalHDL y ofrece soporte para Linux y FreeRTOS. Las implementaciones *hardware* están actualmente integradas en Litex [104] y pueden realizarse para dispositivos FPGA. Con este núcleo RISC-V, se puede lograr una baja utilización de recursos y una alta frecuencia de operación. En una configuración estándar de un RV64 NaxRiscv funcionando a 137 MHz en una FPGA Xilinx Artix 7, se lograron 2,97 DMIPS. En la Figura 51 se muestra el diagrama general de la arquitectura con dos configuraciones posibles: a la izquierda con tres unidades de ejecución y a la derecha con dos.

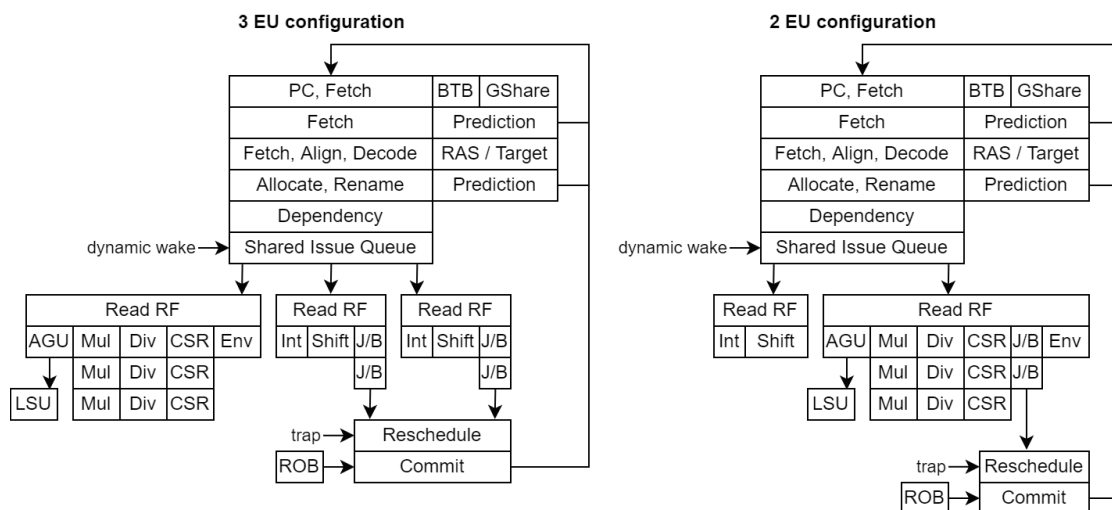


Figura 51. Arquitectura general del procesador NaxRiscv [103]

5.1.7. NOEL-V

NOEL-V [105] es un núcleo basado en RISC-V disponible con una licencia de código abierto de una empresa comercial. Está diseñado para soportar arquitecturas RV32 y RV64 para aplicaciones integradas. En la Figura 52 se muestra el diagrama de bloques de NOEL-V.

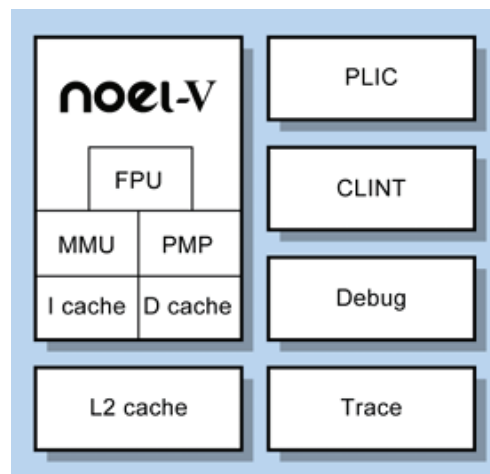


Figura 52. SoC NOEL-V

El núcleo del procesador NOEL-V y los periféricos están contenidos en una biblioteca de bloques IP (GRLIB) que se puede utilizar para crear un SoC. Una lista de configuraciones preestablecidas, así como configuraciones personalizadas, está disponible a través de GRLIB. Sin embargo, algunas de las configuraciones solo están disponibles bajo una licencia comercial, como GRFPU para habilitar unidades de procesamiento de coma flotante (FPU) de alto rendimiento o FT-FPGA, que incluye funciones de tolerancia a fallos para implementaciones FPGA o ASIC.

En [106] se introduce una plataforma MPSoC basada en NOEL-V RV64 con configuración de procesador de propósito general (GPP). La arquitectura incluye un *pipeline* con dos unidades de proceso que despachan instrucciones en orden (*dual-issue in-order*), e incluye unidades de punto flotante (soporta *floats* y *doubles*), cuatro unidades de enteros segmentadas, soporte para multiplicaciones y divisiones de enteros, soporte para instrucciones atómicas, una unidad de gestión de memoria (MMU), una unidad para protección de memoria física (PMP), unidad de predicción de saltos, *stack* y cache de instrucciones y datos de L1 configurables. El MPSoC incluye números interfaces de comunicación y periféricos integrados.

Con respecto a los recursos lógicos utilizados por NOEL-V, tal como sea extraído de la información disponible en [105], en la Tabla 9 se muestran los recursos utilizados para la implementación de dispositivos de Arty (Serie 7) de AMD/Xilinx para una frecuencia de 40 MHz. La Tabla 10 recoge resultados para la serie Kintex funcionando a 100 MHz tanto en configuraciones de 32 y de 64 bits.

Tabla 9. Recursos utilizados por NOEL-V para dispositivos Arty de AMD [105]

Configuration name	MC32-Lite	GP32-Lite	MC64
Processor	NOEL-V	NOEL-V	NOEL-V
Architecture	32-bit/single issue	32-bit/dual issue	64-bit/single issue
RISC-V Extensions*	RV32-IMA	RV32-IMAFDBC	RV64-IMAFDBC
Number of cores	1	1	1
Level-1 cache	8+8 KiB	16+16 KiB	8+8 KiB
HW Mul / Div	Yes	Yes	Yes
PMP	Yes	Yes	Yes
MMU	No	Yes	No
Floating-point Unit	No	nanoFPU _{nv}	nanoFPU _{nv}
Debug Support Unit	Yes	Yes	Yes
Standard Peripherals	Yes	Yes	Yes
Slice LUT	23850 (36.2%)	41715 (66.8%)	34880 (55 %)
Slice Registers	11454 (9.0%)	15985 (12.6 %)	14891 (11.7 %)
BRAMs	29 (21.5%)	47 (34.8 %)	31 (22.9%)
DSPs	4 (1,6%)	6 (2.5%)	18 (7.5%)

5.1. Núcleos de CPU y SoC

Tabla 10. Datos de utilización de recursos de NOEL-V para dispositivos Kintex [105]

Configuration name	GP64F	GP64	GP64-Lite	GP32F	GP32-Lite	MC32	MC32-Lite
Processor	NOEL-V	NOEL-V	NOEL-V	NOEL-V	NOEL-V	NOEL-V	NOEL-V
Architecture	64-bit/dual issue	64-bit/dual issue	64-bit/dual issue	32-bit/dual issue	32-bit/dual issue	32-bit/single issue	32-bit/single issue
RISC-V extensions*	RV64-IMAFDBCH	RV64-IMAFDBCH	RV64-IMAFDBC	RV32-IMAFDBCH	RV32-IMAFDBC	RV32-IMAFDBC	RV32-IMA
Number of cores	1	1	1	1	1	1	1
Level-1 cache	16+16 KiB	16+16 KiB	16+16 KiB	16+16 KiB	16+16 KiB	8+8 KiB	8+8 KiB
HW Mul / Div	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PMP	Yes	Yes	No	Yes	No	Yes	No
MMU	Yes	Yes	Yes	Yes	Yes	Yes	No
Floating-point Unit	GRFPU _{nv}	nanoFPU _{nv}	nanoFPU _{nv}	GRFPU _{nv}	nanoFPU _{nv}	nanoFPU _{nv}	No
Debug Support Unit	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Standard Peripherals	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CLB LUTs	73687 (22.2%)	65004 (19.6%)	53391 (16.1%)	64091 (19.3%)	43518 (13.1%)	31843 (9.6%)	24082 (7.2%)
CLB Registers (FF)	30865 (4.7%)	27402 (4.1%)	23202 (3.5%)	26670 (4.2%)	17674 (2.7%)	14808 (2.2%)	13001 (2.0%)
BRAMs	38 (3.5%)	35.5 (3.3%)	35.5 (3.3%)	34.5 (3.2%)	31.5 (2.9%)	21.5 (2.0%)	20.5 (1.9%)
DSPs	42 (1.5%)	18 (0.7%)	18 (0.7%)	30 (1.1%)	6 (0.2%)	6 (0.2%)	4 (0.1%)

5.1.8. ORCA RISC-V

ORCA [107] implementa un subconjunto de la ISA RISC-V (RV32I y RV32IM) optimizada para FPGA, diseñado como host para dar soporte al procesamiento de matrices a alta velocidad. Puede operar como un núcleo procesador que incluye caches de datos e instrucciones. Puede ser configurado para usar interconexiones Avalon de Altera, WISHBONE de Open Cores o Local Memory Bus (LMB) de AMD. El proyecto, inicialmente desarrollado por VectorBlox [108], está disponible como código abierto y actualmente lo mantiene la comunidad. Las implementaciones de ORCA han logrado un rendimiento significativo en relación con la proporción de recursos utilizado. En la Tabla 11 y en la Tabla 12 se muestran resultados de rendimiento de los diferentes *cores* probados en el dispositivo Cyclone IV de Altera.

Tabla 11. Resultados de rendimiento

Core	FMAX	LUT4s
RV32E	138 MHz	1700
RV32I	133.53 MHz	1900
RV32IM	97 MHz	2500
RV32IM (sw div)	101 MHz	2100

Tabla 12. Resultados de implementación para distintas configuraciones y dispositivos FPGA

	Intel Cyclone IV (CPU only)		Lattice iCE40 Ultra (Full sys)	
	Area (LUTs)	Speed (MHz)	Area (LUTs)	Speed (MHz)
ORCA Tiny (no counters, no exceptions, sw mul/div, 5 stage pipeline, 8 cycle shifter)	1501	114	2274	33
ORCA General Purpose (32b counter, exceptions, hw mul, sw div, 5 stage pipeline, 3 cycle shifter)	1968	106	3035	18
ORCA High Performance (64b counter, exceptions, hw mul, hw div, 5 stage pipeline 3 cycle shifter)	2342	106	n/a	n/a
ORCA-GP+LVE	2602	82	n/a	n/a
ORCA-GP-MXP (V8-64k)	26047	96	n/a	n/a

5.1.9. SERV

Una de las implementaciones más pequeñas de RISC-V fue presentada por el proyecto SERV [109]. Se trata de un núcleo de CPU de bits en serie, lo que significa que la ruta de datos interna tiene un bit de ancho. El flujo de datos interno de SERV se muestra en la Figura 53. Para cada instrucción, los datos se leen del archivo de registro o de los campos inmediatos de la palabra de instrucción y el resultado de la operación se almacena nuevamente en el archivo de registro. La lectura y escritura de la memoria se maneja a través del módulo de interfaz de memoria. SERV es un núcleo de código abierto que admite Litex para una variedad de placas FPGA y además para las plataformas Servant [110], CoreScore [111], Observer [112] y Subservient [113]. SERV es compatible con Zephyr RTOS y Linux. Además, el núcleo SERV puede agregar opcionalmente una unidad de multiplicación y división (MDU) correspondiente a la extensión M ISA.

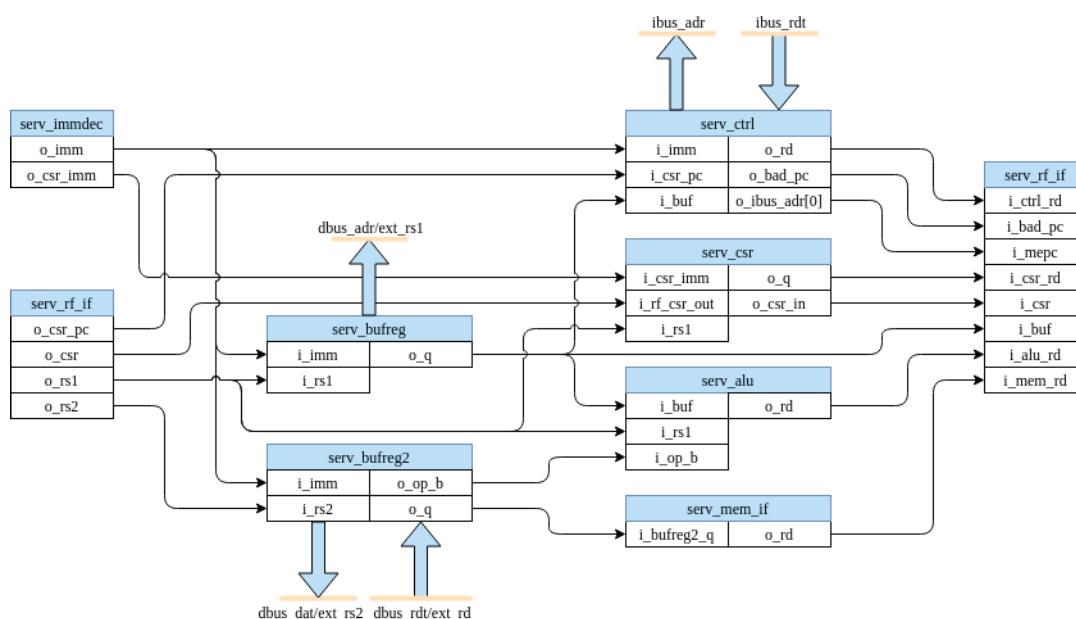


Figura 53. Flujo de datos internos de SERV

5.1.10. VROOM

VROOM [114] es una implementación RISC-V relativamente nueva y aún en desarrollo. VROOM es un núcleo de alta gama que presenta las extensiones ISA RV64IMAFDCHB(V) y ofrece múltiples hilos de procesamiento de 2 vías (SMT). VROOM está disponible bajo la licencia pública GNU v3 (GPL3) y ha sido probado con Linux en instancias FPGA de AWS. La Figura 54, se muestra la arquitectura general del CPU.

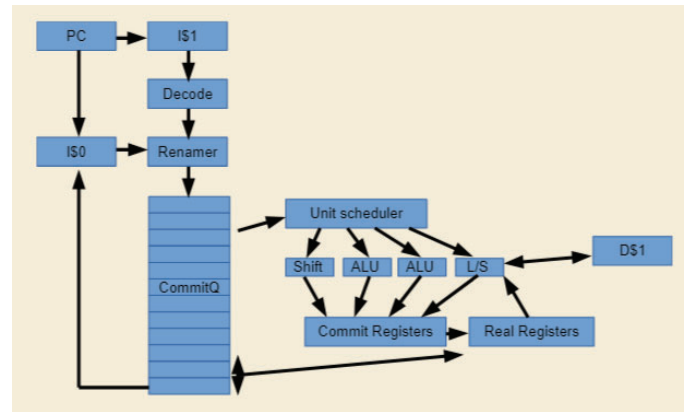


Figura 54. Arquitectura VROOM

Como se aprecia en la Figura 54, la arquitectura ejecuta parte de la instrucción en orden y parte fuera de orden:

- **En orden**
 - FETCH tiene una FSM para leer instrucciones de la memoria principal.
 - DECODE recibe bytes de instrucción y los decodifica en microoperaciones (uops).
 - UCODE generalmente solo pasa uops desde DECODE. Las uops de la decodificación con *trap_to_ucose set* harán que el microse secuenciador tome el control y entregue uops desde la ROM hasta que se alcanza el final de la memoria.
 - RENAME cambia el nombre de los orígenes y los destinos del registro, y envía el uop renombrado a ALLOC.
 - ALLOC asigna ROBID y asigna al RS apropiado.
- **Fuera de orden**
 - EXE ejecuta uops enteras desde su RS cuando sus dependencias están disponibles y vuelve a escribir en el ROB.
 - MEM ejecuta uops de memoria desde su RS cuando sus deps están disponibles y vuelve a escribir en el ROB. Las operaciones de *store* se completan poco después de la emisión (*issue*).
- **En orden:**
 - RETIRE recibe las reescrituras de la parte OoO de la ruta de datos y las vuelve a escribir.

Con respecto a los saltos, estos se resuelven en EXE pero no se toman hasta RETIRE. Actualmente, las ramas siempre se predicen *Not-Taken*.

5.1.11. Ibex

Ibex[115] es un núcleo de CPU RISC-V de 32 bits de clase microcontrolador que admite las extensiones RV32IEMCB ISA. Ibex se desarrolló anteriormente bajo el nombre Zero-riscy[116] como parte de los núcleos de la plataforma PULP [117] y actualmente ha contribuido a lowRISC[118]. Es un núcleo de CPU de producción con múltiples implementaciones de FPGA. La Figura 55, muestra la arquitectura del núcleo.

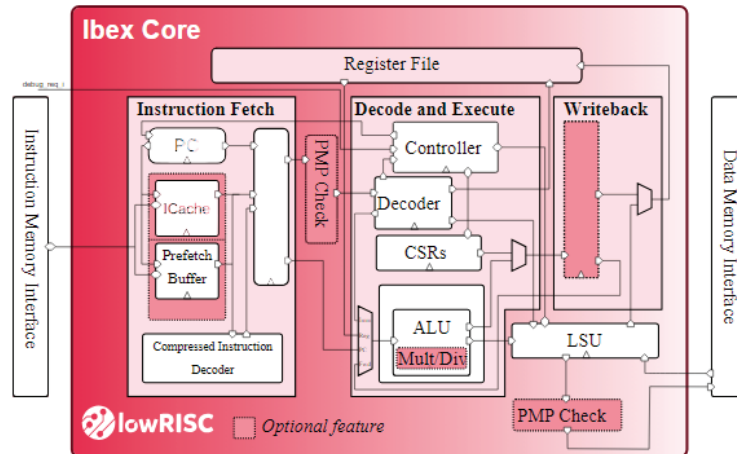


Figura 55. Arquitectura del núcleo Ibex

5.1.12. CVE2

CVE2 [119][120] es un pequeño núcleo de CPU de 32 bits de 2 etapas de pipeline desarrollado por el grupo OpenHW [121] que está basado en trabajos previos de los núcleos Ibex[115] y Zero-riscy[122]. CV32E20 se diseñó para ser de bajo coste, está escrito en *SystemVerilog*, es altamente parametrizable y admite extensiones Entero (I) o Embebido (E), Multiplicación y División de Enteros (M) e instrucciones Comprimadas (C) de 16 bits. Este núcleo se caracteriza por simplificar el diseño y reducir el consumo energético, además, permite parametrizar diversas configuraciones tales como el número de registro (16 o 32), tipo de unidad de multiplicación/división (rápida, lenta, ciclo único), activación de extensiones opcionales, número y granularidad de regiones PMP (*Physical Memory Protection*). A continuación, la Figura 56 muestra la estructura general del CVE2.

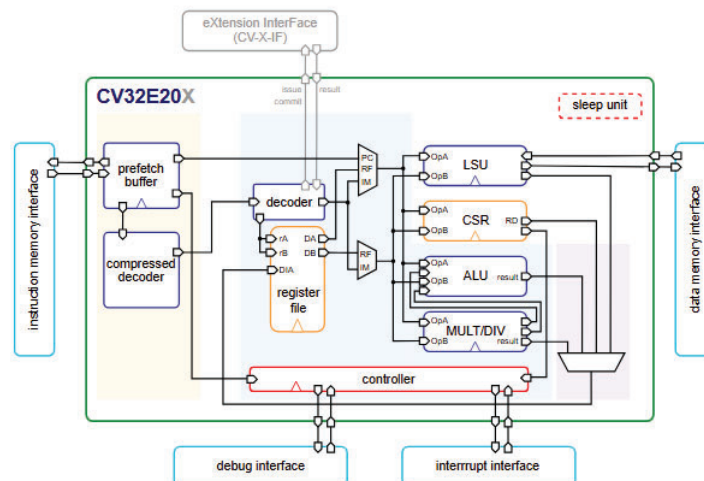


Figura 56. Diagrama de bloques arquitectura CVE2 [120]

5.1.13. CVA6

CVA6 [123][124] es un procesador RISC-V altamente configurable, forma parte de la familia de procesadores *CORE-V* del OpenHW Group. Se trata de una CPU de 6 etapas de *pipeline* que implementa el conjunto de instrucciones RISC-V de 64 bits que apunta a implementaciones tanto de ASIC como de FPGA. Implementa completamente las extensiones I, M, A y C especificadas en el *Volumen I: User-Level ISA V 2.3*, así como el borrador de la extensión de privilegios 1.10. Sigue la especificación de OpenHW Group para la implementación de procesadores basados en RISC-V, tal como se muestra en la En la Figura 57 se muestra el *stack* utilizado por OpewnHW Group para la implementación de procesadores basados en RISC-V y en la Figura 58 se puede observar la arquitectura del procesador.

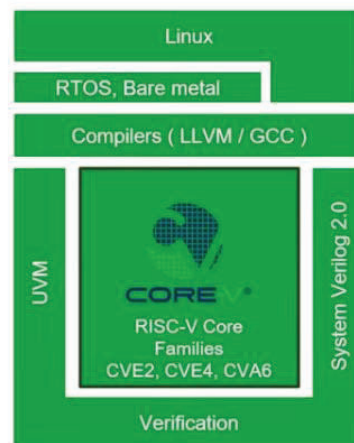


Figura 57. Estructura de desarrollo de microprocesadores RISC-V de OpenHW Group

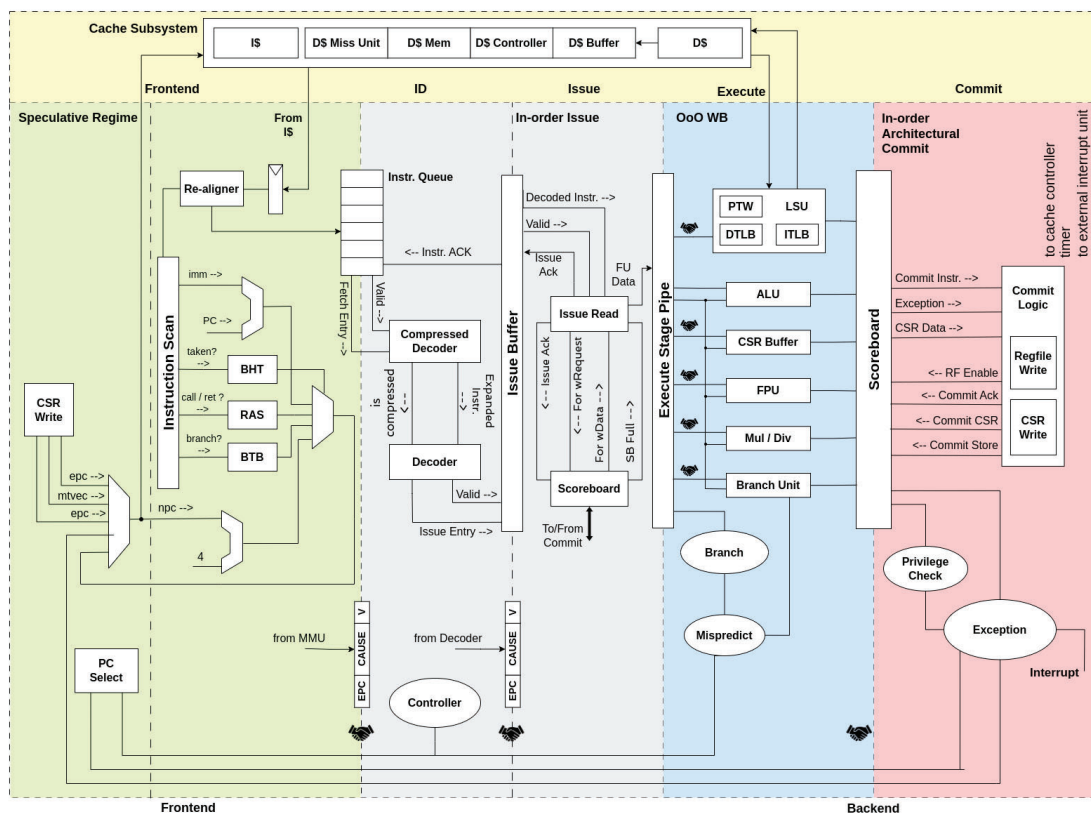


Figura 58. Arquitectura CVA6 [125]

Implementa tres niveles de privilegio M, S, U para soportar completamente un sistema operativo tipo Unix. Además, cumple el borrador de la especificación de depuración externa 0.13. Tiene un tamaño configurable, TLB independientes, PTW por *hardware* y predicción de saltos (búfer de destino de saltos y tabla de historial de saltos). CVA6 está escrito en *SystemVerilog* y es altamente parametrizable.

CVA6 es un procesador *open source*, listo para ser utilizado en distintas aplicaciones industriales. Por ejemplo, BOSCH ha anunciado su utilización en distintos sistemas ADAS para la ayuda en la conducción (Figura 59), adaptando su arquitectura para acelerar distintas aplicaciones., en este caso redes neuronales profundas.

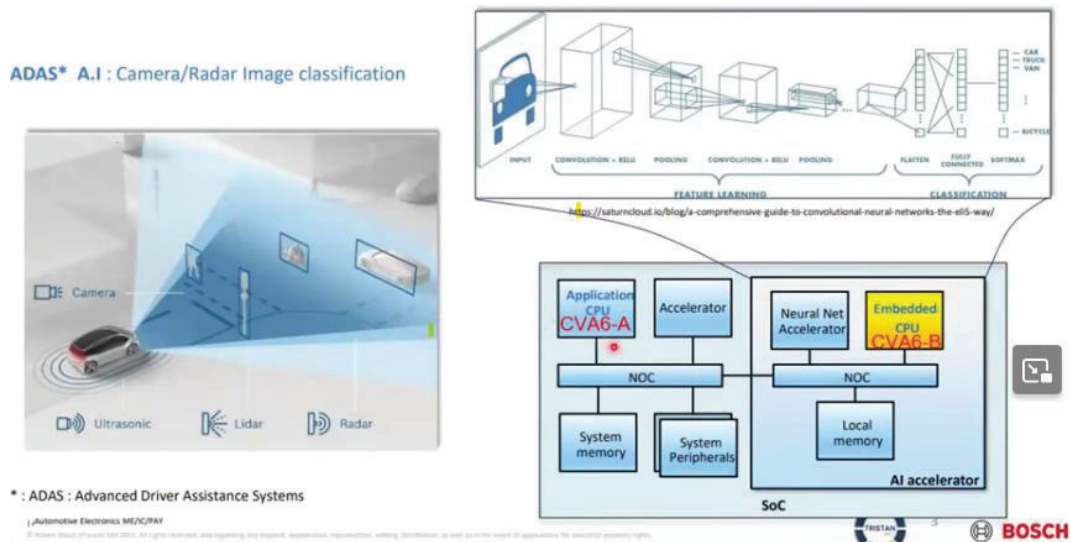


Figura 59. Utilización de CVA6 en aplicaciones industriales

5.1.14. Sargantana

Sargantana [126][127] es un diseño desarrollado conjuntamente por investigadores del BSC, CIC-IPN, CNM, UAB, UB, UPC y URV que pertenece a la tercera generación de los procesadores Lagarto. Sargantana, un procesador de 64 bits basado en RISC-V que implementa RV64G ISA, un subconjunto de la extensión de instrucciones vectoriales (RVV 0.7.1) e instrucciones personalizadas específicas de la aplicación.

Presenta un *pipeline* de 7 etapas optimizada que implementa escritura de fuera de orden, cambio de nombre de registros y un acceso a memoria sin bloqueo. Asimismo, Sargantana cuenta con una unidad *Single-Instruction Multiple-Data* (SIMD) que acelera aplicaciones de dominios específicos (Figura 60). En la Figura 61, se muestra un ejemplo de la implementación del procesador en tecnologías GF 22nm FD-SOI.

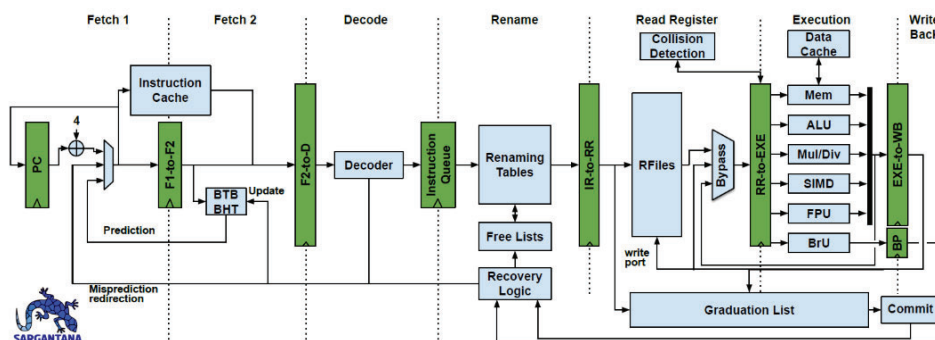


Figura 60. Arquitectura de Sargantana [127]

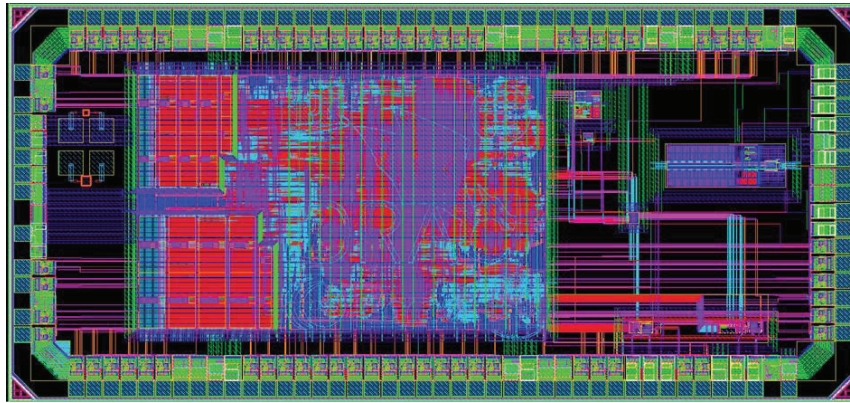


Figura 61. Implementación del procesador Sargantana en tecnologías GF 22nm FD-SOI [126]

5.2. IPs RISC-V comerciales

5.2.1. Procesadores de SiFive

SiFive [128] ofrece una amplia gama de núcleos de procesador basados en las arquitecturas RV32 y RV64 con diferentes extensiones que aportan flexibilidad para diseñar arquitecturas de procesador para diferentes clases de potencia, área y el rendimiento en función a las necesidades de la aplicación que se vaya a usar. Los núcleos IP RISC-V de SiFive están catalogados en 4 familias de procesadores disponibles:

En primer lugar, el conjunto de procesadores SiFive Essential ofrecen núcleos altamente personalizables en el rango comprendido desde procesadores de aplicaciones heterogéneas de alto rendimiento hasta microcontroladores de baja potencia. Entre ellos se incluyen microcontroladores, procesamiento en tiempo real y dispositivos IoT. Los modelos 2-Series, 6-Series, 7-Series y U-Series están incluidos entre la propuesta de SiFive. Por ejemplo, E31 [129], es un núcleo de procesador RISC-V distribuido bajo una licencia comercial. También está disponible una licencia de evaluación gratuita para uso de FPGA, pero el código fuente RTL está oculto.

El núcleo E31 (Figura 62Figura 63) presenta soporte de expansión RV32IEMAFRC ISA con coherencia de hasta 8 núcleos en un solo diseño. Se dice que es más eficiente que ARM Cortex-M4 y se considera adecuado para una variedad de aplicaciones de IoT.

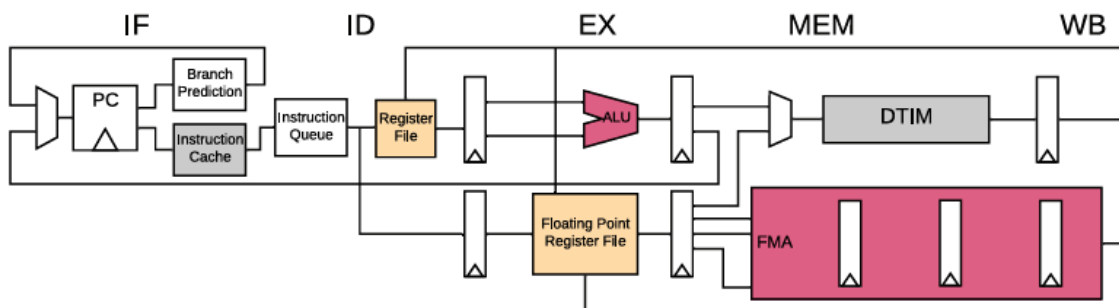


Figura 62. Arquitectura de la serie E3 de SiFive

La unidad de ejecución de la serie E3 incluye una ruta de datos para la ejecución de una única instrucción, en orden. La segmentación consta de cinco etapas: búsqueda

de instrucciones (IF), decodificación de instrucciones y recuperación de registros (ID), ejecución (EX), acceso a la memoria de datos (MEM) y escritura de registros (WB). La ruta de datos tiene una tasa de ejecución máxima de una instrucción por ciclo de reloj, de modo que la mayoría de las instrucciones tienen una latencia de resultado de un ciclo.

Por otro lado, el core E51 [130] (Figura 63) es un núcleo RISC-V integrado que se utiliza, por ejemplo, como subsistema de monitoreo en el SoC PolarFire FPGA. Se admite una arquitectura de 64 bits con extensión RV64IMAC ISA y se puede integrar con hasta 8 núcleos en total coherencia. En la Figura 64, se muestra una tabla comparativa de rendimientos de cada núcleo.

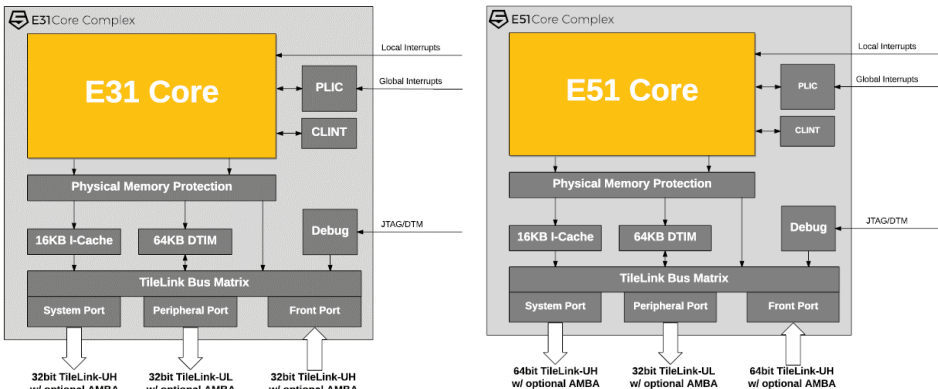


Figura 63. Diagrama de bloques de los núcleos E31[129] y E51[130]

	TSMC 28nm HPC		UMC 55nm LP	
Frequency (worst)	200 MHz	870 MHz	200 MHz	340 MHz
Implementation Details	9t tt 0p9v 25C	12t tt 0p9v 25C	7t tt 1p2v 25C	7t tt 1p2v 25C
Core Complex Area (mm ²)*	0.082	0.101	0.232	0.267
Core Only Area (mm ²)**	0.053	0.065	0.149	0.183
Core Complex Power (mW)	4.1	26	11.2	24
	TSMC 28nm HPC		UMC 55nm LP	
Frequency (worst)	200 MHz	870 MHz	200 MHz	340 MHz
Implementation Details	9t tt 0p9v 25C	12t tt 0p9v 25C	7t tt 1p2v 25C	7t tt 1p2v 25C
Core Complex Area (mm ²)*	0.142	0.163	0.404	0.45
Core Only Area (mm ²)**	0.082	0.09	0.233	0.28
Core Complex Power (mW)	5.3	36	15	32

Figura 64. Tabla comparativa de rendimientos [131]

SiFive Performance, proporcionan un rendimiento excepcional en una huella energéticamente eficiente. Incluyen la serie P650/670 (procesadores de aplicaciones vectoriales fuera de orden de cuatro emisiones), la serie P550 (procesadores de aplicaciones vectoriales fuera de orden de tres emisiones), la serie P450/P470 (procesadores de aplicaciones vectoriales fuera de orden altamente eficientes) y la serie P270 (procesadores vectoriales RISC-V de rendimiento eficiente).

SiFive Intelligence, estos procesadores fueron diseñados para cargas de trabajo de inteligencia artificial, manejo de flujo de datos, detección de objetos, procesamiento de voz y recomendaciones. Ofrecen capacidades de cómputo vectorial escalables y entre ellos destaca el modelo X280 con extensiones RVV de 512 bits de ancho y soporte para Linux. Con la nueva extensión de interfaz de coprocesador vectorial SiFive (VCIX) se utiliza como un host informático de IA para proporcionar una programación flexible en combinación con el acelerador Google MXU (multiplicador de matriz sistólica) en el centro de datos [132].

Por último, la familia SiFive Automotive que incluye una amplia gama de procesadores RISC-V optimizados para aplicaciones automotrices, con un enfoque en seguridad, rendimiento y energía. Se incluyen en soluciones para sistemas avanzados de asistencia al conductor (ADAS), infoentretenimiento (IVI), *Body* y *Powertrain*. Cumplen con los estándares de seguridad ISO26262 y están diseñados para satisfacer las necesidades específicas de los vehículos. En esta gama se encuentra la Serie E6-A de 32 bits, S7-AD de 64 bits, X280-A y P870-A. En la Figura 65 se muestra un ejemplo del *cluster* de lo procesadores.

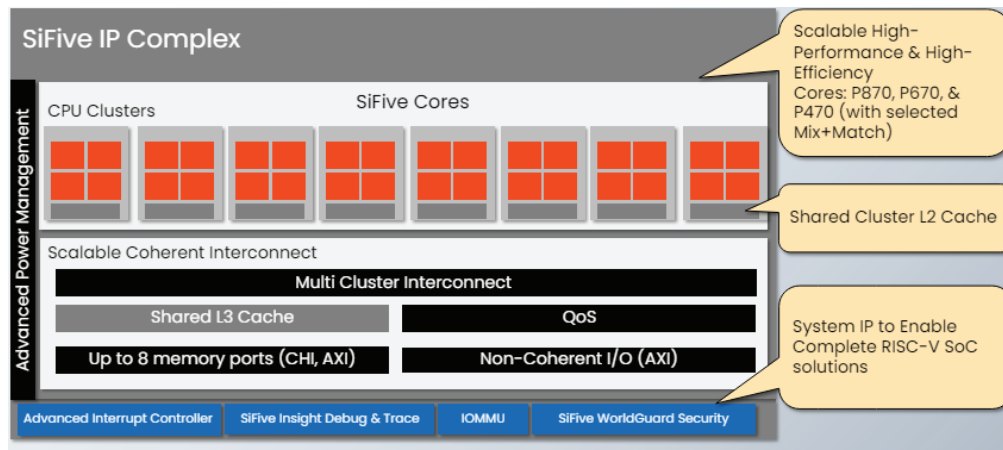


Figura 65. Cluster de procesadores de SiFive

C910 [133] es un núcleo de CPU RISC-V de alto rendimiento que admite hasta 4 núcleos a una frecuencia operativa máxima de 2,5 GHz y alcanza 6 DMIPS/MHz. La extensión RV64IMAFDC ISA es compatible con una canalización superescalar desordenada. C910 fue desarrollado por T-Head, una empresa de Alibaba. Admite la expansión de vectores ISA y tiene una serie de instrucciones no estándar para acelerar diversas tareas. C910[134] es compatible con varias variantes de Linux portadas a la arquitectura RISC-V, como las adaptaciones de Debian y Fedora. Recientemente, OpenC910[135], una versión de código abierto del C910, fue lanzada bajo la licencia permisiva Apache 2.0.

GD32VF103 [136] es un SoC MCU RISC-V de 32 bits con soporte de expansión RV32IMAC-ISA y una frecuencia operativa máxima de 108 MHz. Se basa en una arquitectura de un solo núcleo llamada Bumblebee Core[137], desarrollada conjuntamente por Nuclei y que contiene el núcleo N205[138]. El diseño incluye una gama de entradas ADC y salidas DAC con capacidad de 12 bits, así como otras E/S como UART y SPI.

5.2.2. Synopsys ARC-V

La familia de procesadores escalable basados en RISC-V ofertados por Synopsys [139] ofrecen procesadores vectoriales, procesadores neuronales y procesadores de visión y subsistemas. Estos procesadores se categorizan en 3 opciones. ARC-V incluye opciones de procesadores RISC-V de alto rendimiento, rango medio y ultra bajo consumo, así como versiones de seguridad funcional (Figura 66). Estos núcleos ofrecen una eficiencia óptima en términos de potencia y rendimiento para una amplia variedad de cargas de trabajo de aplicaciones. Dentro de esta gama se ofertan las series:

- ARC-V RMX de 32 bits incluye los procesadores RMX-100, con un *pipeline* de 3 etapas, y RMX-500 con un *pipeline* de 5 etapas. Están optimizados para aplicaciones integradas de bajo consumo. Los procesadores ofrecen soporte DSP opcional para una mayor eficiencia en el procesamiento de señales. Los procesadores RMX cumplen con las normas ISO 26262 ASIL (*Automotive Safety Integrity Levels*) B y D FuSa (*Functional Safety*), así como cumplimiento de ciberseguridad en vehículos ISO 21434.
- ARC-V RHX-100 de 32 bits consta de procesadores superescalares, de un solo núcleo y multinúcleo optimizados para aplicaciones eficientes en tiempo real. Incluye soporte para aceleradores coherentes y virtualización de *hardware* en tiempo real, así como extensiones RVV opcionales (RHX-100V/105V). Para aplicaciones críticas para la seguridad, la serie RHX-100 ofrece versiones compatibles con la seguridad funcional de todos los procesadores de la serie.
- ARC-V RPX-100 de 64 bits son procesadores multinúcleo con soporte de perfil definido RISC-V de 64 bits para modos de usuario y supervisor. El ARC-V RPX-105 multinúcleo admite SMP Linux y caché L3 compartida. Ofrece configuraciones de hasta 16 núcleos y está optimizado para un rendimiento de procesamiento de host eficiente para una variedad de aplicaciones. La serie RPX-100 también ofrece versiones de sus procesadores que cumplen con las normas de seguridad funcional.

El procesador IP especializado ARC IP está optimizado para obtener la mejor potencia, rendimiento y área para dominios de aplicaciones dedicados. En esta gama se encuentran las series ARC VPX DSP IP, ofrecen diseño SIMD/VLIW para procesamiento paralelo y múltiples unidades vectoriales de punto flotante para alta precisión. La IP de la unidad de procesamiento neuronal ARC NPX ofrece unidades de procesador neuronal escalables (MAC de 1K a 96K) y admite las últimas redes de IA (por ejemplo, transformadores).

El ARC NPX6 NPU IP está diseñado para la cobertura de algoritmos de aprendizaje profundo que incluyen tareas de visión por computadora como detección de objetos, mejora de la calidad de la imagen y segmentación de escenas, y para aplicaciones más amplias. Los procesadores EV están diseñados para integrarse perfectamente en un SoC y pueden usarse con cualquier procesador host y operar en paralelo con el host. EV7x integran unidades de procesamiento escalar, DSP vectorial y red neuronal profunda (DNN) para un procesamiento de visión rápido y de alta precisión. Los procesadores de

visión ARC EV7x integran hasta cuatro unidades de procesamiento de vectores (VPU) mejoradas y un acelerador DNN con hasta 3520 MAC.

La última gama del conjunto de procesadores, la familia ARC Classic de 32/64 bits basados en los conjuntos de instrucciones ARCV2 y ARCV3, flexibles y probados ofrece las series:

- ARC EM incluye ARC EM4 y EM6 basados en ARCV2 ISA, así como EM5D, EM7D, EM9D y EM11D mejorados con DSP. Los núcleos EM ultracompactos presentan una densidad de código excelente, un tamaño pequeño y un consumo de energía muy bajo, lo que los hace ideales para aplicaciones integradas y profundamente integradas sensibles al área y de energía crítica.
- ARC SEM incluye los procesadores de seguridad ARC SEM110 y SEM120D basados en ARCV2 ISA para proteger contra ataques de HW, SW y canal lateral y SecureShield para crear TEE.

ARC HS se diferencian por tener una CPU de alto rendimiento, CPU + DSP y configuraciones de uno y varios núcleos.

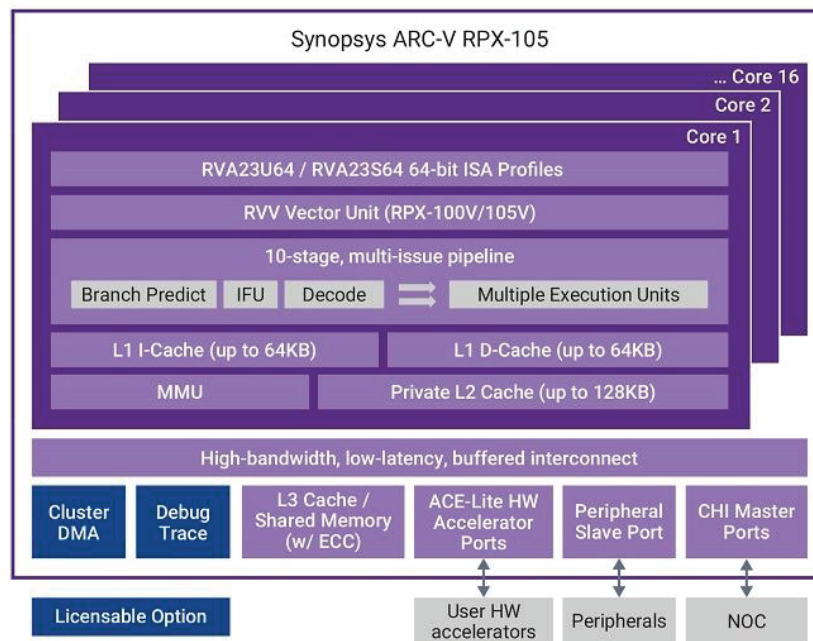


Figura 66. Diagrama de bloques de ARC-V RPX-105 [140]

5.2.3. AIRISC CORE

El Instituto Fraunhofer de Microsistemas Fotónicos IPMS [141] ofrece módulos IP listos para usar e independientes de la plataforma. AIRISC Core Complex [142] es un núcleo de procesador de 32 bits basado en tecnología RISC-V y optimizado para aplicaciones de IA integradas. Este núcleo ha sido probado con ASIC y FPGA y permite construir un SoC.

La CPU implementa una arquitectura *Harvard* con un *pipeline* de cuatro etapas e interfaces AHB-Lite separadas para la recuperación de instrucciones y el acceso a datos. La ISA básica del núcleo RV32I_Zicsr se puede ampliar mediante una interfaz de coprocesador unificada (PCPI) y, además, dispone de coprocesadores para operaciones de multiplicación y división de enteros.

AIRISC incluye un conjunto de periféricos comunes junto con una amplia variedad de extensiones opcionales para crear un sistema en chip (SoC) personalizable (Figura 67). Fraunhofer IMS utiliza su IP integrado RISC-V para tareas de sensores bajo una licencia de código abierto, que también permite su uso para productos comerciales. Soporta la integración en dispositivos FPGA mediante la utilización de flujos de diseño de los principales fabricantes de FPGAs.

La conversión a un ASIC es posible posteriormente en un tiempo récord gracias al código base idéntico y no requiere una licencia comercial para el núcleo básico. El núcleo AIRISC [142] está completamente implementado en *Verilog/SystemVerilog*. El paquete de código abierto AIRISC incluye, además de la descripción *Verilog* para el núcleo, el soporte necesario para un fácil desarrollo de *software* (*Board Support Package*) y una aplicación de ejemplo básica. El IMS también mantiene un catálogo de funciones complementarias para acelerar las tareas de detección, tales como procesamiento de imágenes, inferencia de modelos de IA, etc.

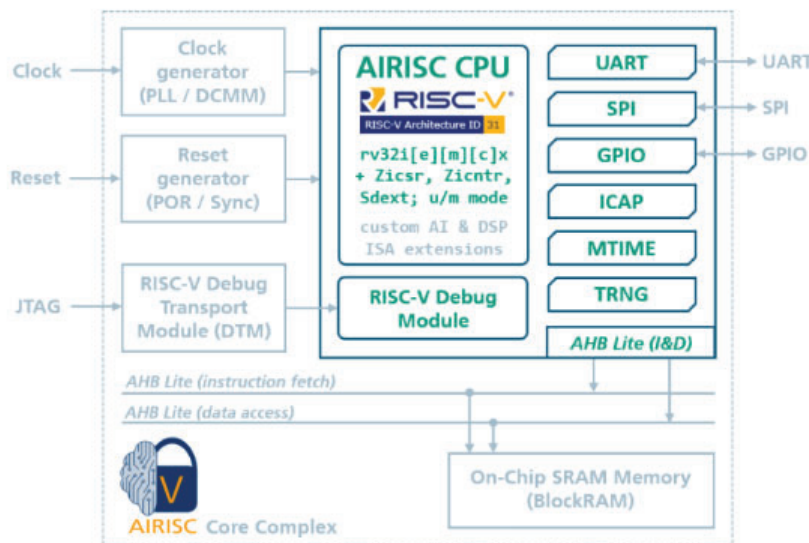


Figura 67. Arquitectura AIRISC Core Complex [142]

5.3. Núcleos RISC-V orientados a FPGA

5.3.1. CVA5

CVA5 [143] es un procesador RISC-V de 32 bits diseñado y optimizado concretamente para FPGAs por el grupo OpenHW. Este procesador está desarrollado en *SystemVerilog* lo que facilita su integración y modificación en entornos de diseño, es compatible con extensiones RV32IMAFD lo que incluye Enteros (I), Multiplicación/División(M), Atómicas (A), Punto flotante en simple y doble precisión (F

y D). Su arquitectura es derivada del proyecto Taiga [144] de la Universidad Simon Fraser.

CVA5 se caracteriza por una arquitectura de 5 etapas tipo *Application Class*, siendo ideal para aplicaciones exigentes en rendimiento (alcanza hasta 250 MHz en FPGAs UltraScale+ de un solo núcleo) (Figura 68). La estructura interna del CVA5 posibilita la ejecución simultánea de instrucciones y admite elementos de ejecución con diferentes tiempos de respuesta, lo que simplifica la integración de nuevas funcionalidades. En la Figura 68, los bloques marcados en rojo son configurables (pueden estar presentes o no), de tal forma que es posible reducir el uso de recursos del dispositivo FPGA.

CVA5 soporta Linux de forma experimental mediante la utilización de la plataforma LiteX (Figura 69). Utiliza *Open Supervisor Binary Interface* (OpenSBI), que es una implementación *open source* de la especificación realizada por RISC-V para SBI.

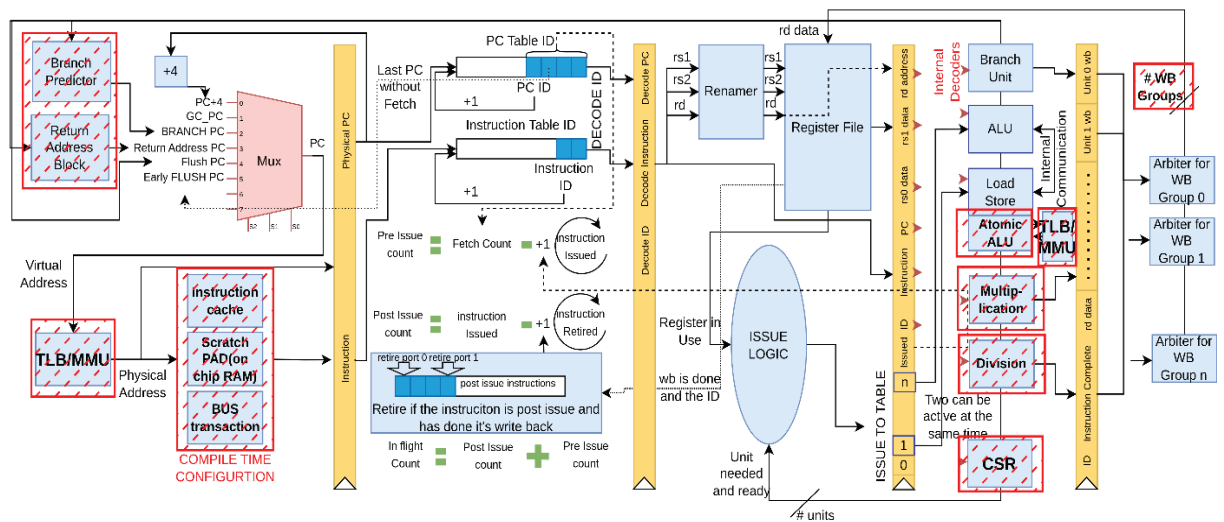


Figura 68. Arquitectura de CVA5 [143]

```
[ 4.027908] Freeing initrd memory: 8192K
[ 4.037652] Freeing unused kernel image (initmem) memory: 180K
[ 4.038055] Kernel memory protection not selected by kernel config.
[ 4.038364] Run /init as init process
Starting syslogd: OK
Starting klogd: OK
Running sssd: OK
Saving 256 bits of non-creditable seed for next boot
Starting network: OK
Starting crond: OK
crond[64]: crond (busybox 1.36.1) started, log level 8

Welcome to Buildroot
buildroot login: root

LINUX
ON
CVA5

32-bit RISC-V Linux running on LiteX / CVA5.

login[65]: root login on 'console'
root@buildroot:~# echo Hello From CVA5
Hello From CVA5
root@buildroot:~# pwd
/root
root@buildroot:~#
```

Figura 69. Linux arrancado sobre CVA5 usando OpenSBI

5.3.2. AMD MicroBlaze V.

El procesador MicroBlaze V de AMD [145] es un procesador *soft-core* basado en la arquitectura de juego de instrucciones de 32 bits de RISC-V. Está diseñado para ser modular, eso significa que se puede usar el bloque de instrucciones que se necesite, haciendo que el procesador pueda ser más sencillo y eficiente. El MicroBlaze V presenta una arquitectura RISC (*Reduced Instructions Set Computer*) y está optimizado para ser implementado en las FPGA y SoC de AMD. En la Figura 70 se muestra el diagrama de bloques del procesador.

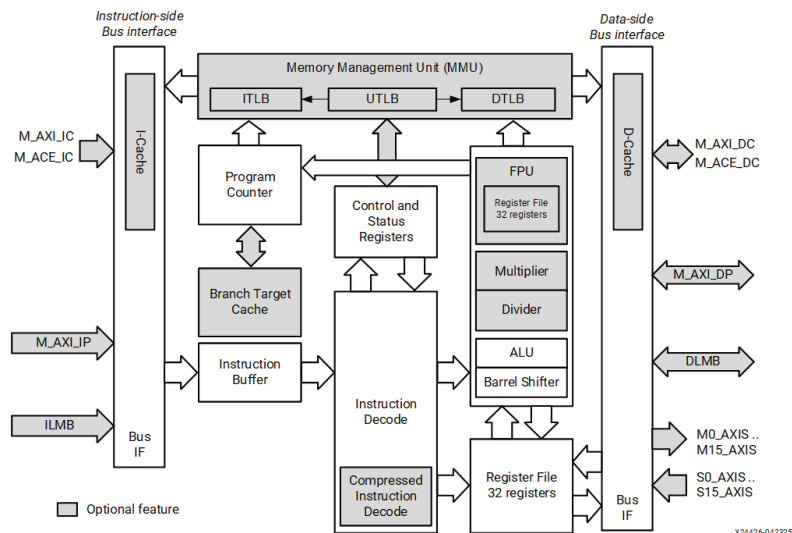


Figura 70. Diagrama de bloques del procesador MicroBlaze V [145]

Algunas de las principales características de MicroBlaze V se resumen a continuación:

- Arquitectura con 32 registros de propósito general.
- Contador de programa de 32 bits ampliable.
- Procesamiento segmentado en el que se emite una instrucción en cada ciclo de reloj. El número de etapas del *pipeline* es configurable.
- Unidad Aritmético Lógica.
- Desplazador combinacional (*barrel-shifter*).
- Instrucciones para el manejo del registro de estado (CSR) e instrucciones de control "Zicsr".
- "Zifencei" instruction-fetch fence.

Además de las características anteriores, el procesador MicroBlaze V permite la activación selectiva de funciones adicionales:

- Multiplicación y división (extensión "M").
- Instrucciones atómicas (extensión "A").
- Instrucciones en punto flotante (extensión "F").
- Comprensión del código (extensión "C").
- Manipulación de bits (extensiones "Zba", "Zbb", "Zbc", "Zbs").

También permite implementar distintas versiones en función de las necesidades de computación, definiendo *pipelines* con distinta profundidad (3, 4, 5 y 8 etapas de pipeline). Esta configuración afecta a la latencia (medida en ciclos de reloj) en la ejecución de las distintas instrucciones de la arquitectura. Por ejemplo, una instrucción JAL tarda 3 ciclos si se configura con 3, 4 y 5 etapas y 5 ciclos para una configuración de 8 etapas.

MicroBlaze-V, a diferencia de otros núcleos citados en este documento, está soportado únicamente en dispositivos FPGA y MPSoC de AMD/Xilinx. Las frecuencias máximas de funcionamiento que se alcanzan se resumen en la Tabla 13, mientras en la Tabla 14 se muestran los recursos utilizados para distintas configuraciones.

Tabla 13. Frecuencia de funcionamiento de MicroBlaze-V

Family	F _{max} (MHz)
AMD Virtex™ 7	322
AMD Kintex™ 7	327
AMD Artix™ 7	223
AMD Zynq™ 7000	224
AMD Spartan™ 7	196
AMD Virtex™ UltraScale™	393
AMD Kintex™ UltraScale™	384
AMD Virtex™ UltraScale+™	556
AMD Kintex™ UltraScale+™	557
AMD Zynq™ UltraScale+™	506
AMD Artix™ UltraScale+™	494
AMD Spartan™ UltraScale+™	413
AMD Versal™ Adaptive SoCs	460

Tabla 14. Recursos utilizados en dispositivos Artix 7

Device Utilization – Artix 7 FPGAs (XC7A200T fbg676-3)					
Configuration	Device Resources				
	LUTs	FFs	BRAMs (36K)	DSP48	F _{max} MHz
Microcontroller Preset	2236	1138	0	4	156
Real-time Preset	4106	2685	6	4	119
Application Preset	8623	5325	16	6	109
Maximum Frequency	1227	826	0	0	223

En la Figura 71 se muestra un ejemplo de plataforma basada en MicroBlaze-V.

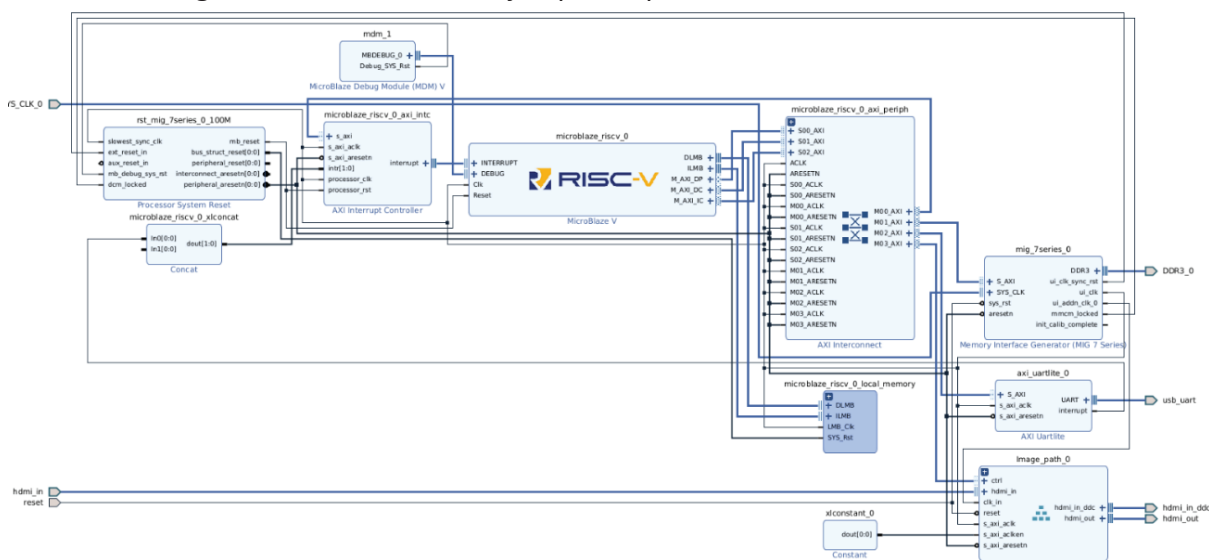


Figura 71. Ejemplo de plataforma basada en MicroBlaze V

5.4. Plataformas basadas en RISC-V

5.4.1. X-HEEP

X-HEEP (*eXtensible Heterogeneous Energy-Efficient Platform*) [146] [147] [148] es un proyecto de un microcontrolador RISC-V de código abierto descrito en *SystemVerilog*, diseñado para plataformas pequeñas y desarrollado en el Laboratorio de Sistemas Integrados (ESL) de la EPFL, en colaboración con el Centro EcoCloud de la EPFL. Este microcontrolador puede ampliarse con aceleradores de forma ágil para aplicaciones con mayor exigencia de rendimiento. Para ello, ofrecen un MCU simple y personalizable, que incluye CPU, periféricos comunes (UART, SPI, JTAG, etc) y memorias, permitiendo a los usuarios extenderlo sin la necesidad de modificar internamente el MCU, simplemente instanciando el diseño personalizado. De este modo, se hereda una IP capaz de arrancar un RTOS como FreeRTOS con toda la pila de *firmware*, controladores HAL y SDK, lo que permite desarrollar el *hardware* o aplicación específica compatible con el microcontrolador.

La plataforma admite simulación usando herramientas como *Verilator*, *Questasim*, etc y construcción de *firmware* vinculando CMake con gcc o clang y/o *SEGGER Embedded Studio*. Se puede implementar tanto en FPGA como en ASIC.

La arquitectura de X-HEEP está dividida en dominios de potencia con un color diferente como se puede observar en la Figura 72: uno para el CPU, para los bancos de memoria, para periféricos y para otros periféricos que siempre estén activos. El diseño maximiza el ahorro de energía reutilizando varias IP de proyectos abiertos como PULP [149], OpenHW [121] y lowRISC [150] además de IP personalizadas.

El subsistema de la CPU se basa en núcleos de 32 bits de bajo consumo y de código abierto, seleccionados por su madurez, por su extensibilidad y porque están verificados. El subsistema de memoria tiene múltiples bancos controlados individualmente por reloj o en modo de retención para optimización energética. El subsistema periférico incluye componentes de propósito general como temporizadores, controladores de interrupciones y GPIO, y puede apagarse cuando no se utiliza. Por último, el subsistema periférico siempre activo mantiene IP críticas funcionando sin control energético, incluyendo el DMA y controladores de arranque.

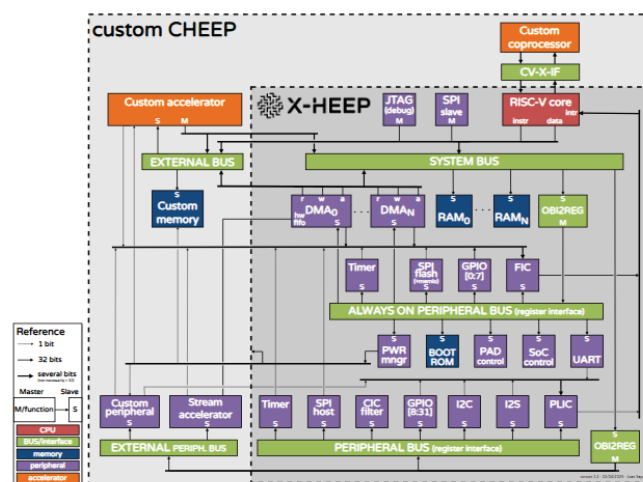


Figura 72. Arquitectura X-HEEP [148]

5.4.2. Cheshire

Cheshire [151] es una plataforma con capacidad para ejecutar Linux, que ha sido diseñada inicialmente como plataforma basadas en CVA6. Permite el desarrollo de SoC de distinta complejidad, desde sistemas empujados hasta aceleradores con multitud de núcleos, con la configurabilidad y el minimalismo como valores clave. Es posible utilizar configuraciones con hasta 31 núcleos CVA6 conectados entre sí. Está formado por distintos tipos de bloques:

- Núcleos CVA6,
- Periféricos multiplexados (UART, I2C, SPI, GPIO, Boot ROM, etc.),
- Módulo de DMA,
- Periféricos no multiplexados (JTAG Debug, Serial Link, VGA y USB 1.1) y
- El *crossbar* que une todos los elementos que conforman el sistema.

En la Figura 73 se muestra la arquitectura general de la plataforma Cheshire.

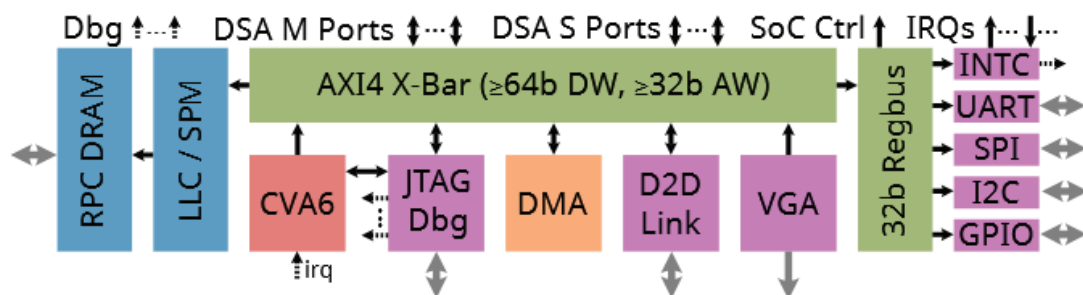


Figura 73. Arquitectura de Cheshire [151]

5.4.3. Sonata System

Sonata system v1.0 [152][153] es una plataforma abierta de lowRISC y parte del proyecto Sunburst, diseñada para mejorar el desarrollo de sistemas embebidos seguros. Este proyecto de código abierto que envuelve un PCB, el RTL para un SoC y el *software* que corre sobre éstos combina una FPGA AMD Xilinx Artix-7 con un microcontrolador Raspberry Pi RP2040 sobre el núcleo Ibex.

Sonata abarca aspectos cruciales de CHERIoT para proteger contra errores, como desbordamiento de búfer, uso después de liberar, permisos de acceso a la memoria, límites que restringen accesos y creación de segmentos para aislar diferentes partes de un sistema. Cada segmento posee atributos únicos y opera en hilos que cumplen ciertos criterios lo que lo hace ideal para ejecutar *software* no confiable en entornos embebidos.

Ofrece varios bloques IP, como UART, I2C, SPI, GPIO, UART, USB, entre otros, además de un sistema eficaz de gestión de memoria e interrupciones. Asimismo, es compatible con CHERIoT RTOS y cuenta con un entorno de desarrollo Nix para simplificar las compilaciones. En la Figura 74, se muestra el diagrama de bloques de la placa.

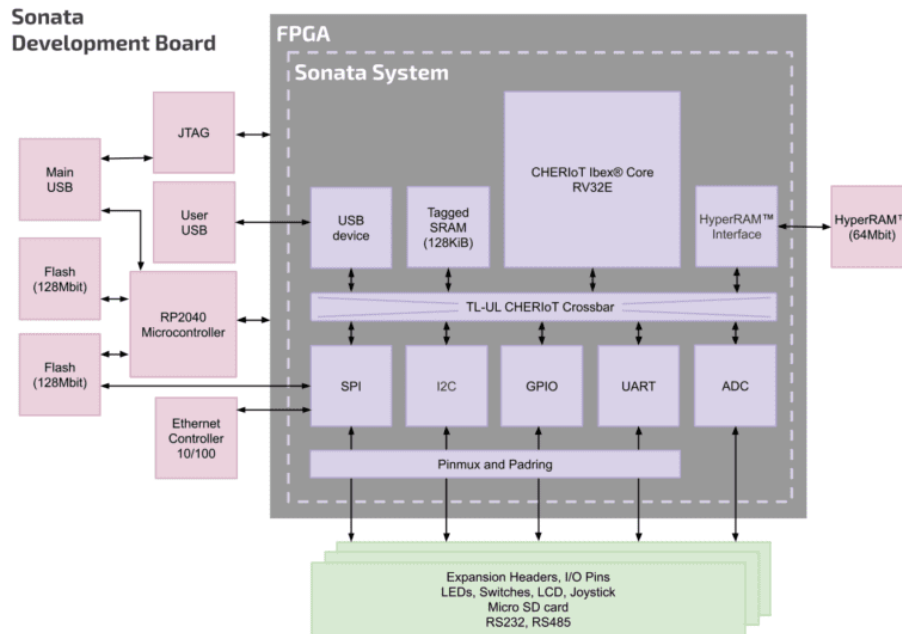


Figura 74. Diagrama de bloques de la placa de desarrollo Sonata[153]

5.4.4. PolarFire SoC FPGA

La plataforma PolarFire SoC FPGA Icicle Kit[154] de Microchip Technology[155] es una plataforma de evaluación de bajo consumo que ha sido diseñada para aplicaciones embebidas que requieren procesamiento heterogéneo, capacidades de lógica programable y procesadores RISC-V. Incluye un SoC con FPGA PolarFire MPFS250T en combinación con un núcleo E51[130] de SiFive (RV64IMAC) y cuatro núcleos U54 (RV64GC).

Con una arquitectura de ejecución paralela es capaz de operar sistemas en tiempo real junto con entornos Linux. El kit tiene 2 GB de memoria LPDDR4, 8 GB de almacenamiento eMMC, Flash SPI de 1 Gb, interfaces de comunicación como PCIe Gen2, USB 2.0 OTG, Ethernet Gigabit dual, UART, SPI, I2C y CAN, ranura para tarjeta SD y, además, conectores de expansión compatibles con Raspberry Pi y mikroBUS.

Su diseño es adecuado para aplicaciones que requieren computación segura y eficiente para sectores como inteligencia artificial, aprendizaje automático, internet de las cosas, defensa, automoción, sistemas aeroespaciales y redes. En la Figura 75, se muestra el diagrama de bloques con las funcionalidades incluidas en la plataforma.

En la literatura se muestran distintos tipos de aplicaciones, en ámbitos de IA, aplicaciones médicas y aplicaciones espaciales, entre otras. Por ejemplo, en [156] se aplican los avances en el aprendizaje automático y la inteligencia artificial para el análisis de datos médicos para el diagnóstico de enfermedades y los resultados del tratamiento, lo que ha cambiado el dominio de la atención médica. En este trabajo se implementa sobre Microchip PolarFire SoC FPGA un modelo de una red neuronal convolucional (CNN) entrenado para la clasificación de latidos con objeto de detectar.

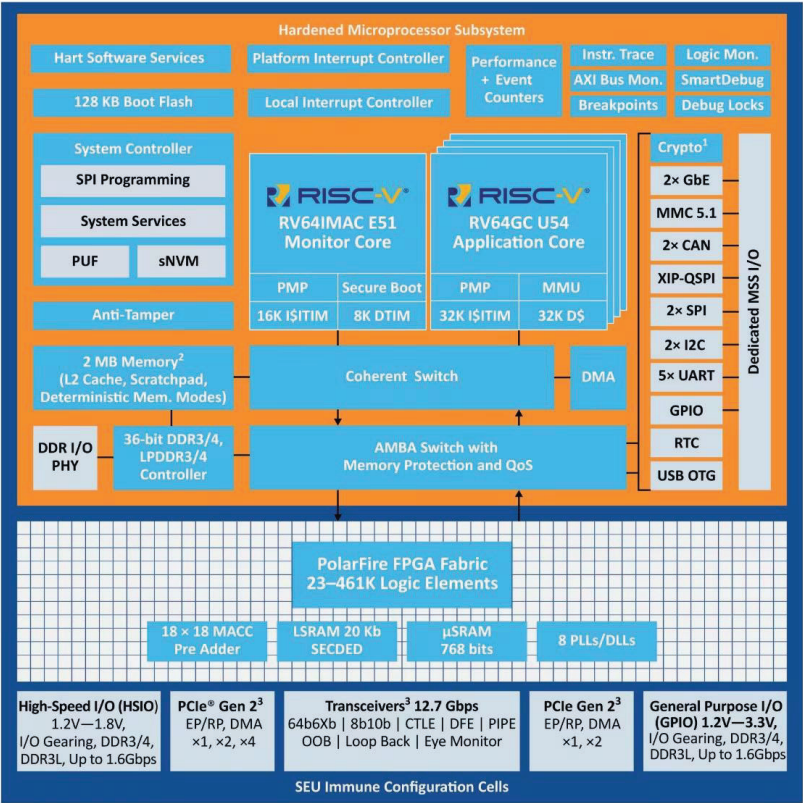


Figura 75. Diagrama de bloques PolarFire SoC Icicle Kit[157]

5.5. Conclusiones

En este capítulo central se ha hecho un amplio estudio de distintos *cores* disponibles, ya sean en el ámbito de la I+D, de *cores* disponibles en el ámbito comercial y de plataformas completas basadas en RISC-V. Se han presentado las características principales y otros detalles relativos a su implementación. Muchos de los *cores* presentados están soportados tanto para dispositivos FPGA como para su implementación ASIC.

Capítulo 6. Implementación de los núcleos RISC-V

En este capítulo se aborda el proceso de diseño e implementación de varios núcleos representativos: NEORV32 que destaca por su modularidad y enfoque educativo, AIRISC que se centra en aplicaciones industriales con soporte extendido de instrucciones. CVA5 está orientada a implementaciones FPGA y CVA6 se caracteriza por su alto rendimiento y compatibilidad con entornos de verificación avanzados. Para ello, se detallan las etapas de implementación de los *cores*, desde la preparación del entorno de desarrollo hasta la síntesis en FPGA, incluyendo herramientas, simulación y verificación.

La implementación de los núcleos NeoRV32, AIRISC, CVA5 y CVA6 sobre las placas Nexys 4 DDR, Nexys Video y Genesys 2 revela distintas estrategias de distribución lógica en la FPGA Artix-7 o en Kintex-7, evidenciadas mediante las visualizaciones obtenidas en Vivado. En la presentación de resultados de implementación física se muestra la estrategia de *placement/routing* utilizadas para cada caso, que finalmente resulta en la distribución espacial de los recursos utilizados. Ello permite comparar la eficiencia espacial, la escalabilidad y la organización interna de cada arquitectura RISC-V en entornos FPGA.

6.1. Implementación de NEORV32

6.1.1. Configuración de las herramientas *software*

El entorno *software* del procesador incluye *makefiles* de aplicaciones, bibliotecas *software* para todas las funciones de la CPU y el procesador, un cargador de arranque, un entorno de ejecución y varios de ejemplo, incluida una adaptación de CoreMark MCU *benchmark* y el conjunto de pruebas oficial de la arquitectura RISC-V. RISC-V. RISC-V GCC se utiliza como cadena de herramientas por defecto (también se proporcionan cadenas de herramientas precompiladas).

Antes de proceder a la ejecución de aplicaciones en el núcleo NEORV32, se debe configurar correctamente el entorno de desarrollo. Para compilar y depurar ejecutables compatibles con RISC-V, es necesario instalar la cadena de herramientas RISC-V GNU *toolchain* (Código 3), que contiene compilador, ensamblador, enlazador y depurador adaptados a la arquitectura RISC-V.

```
project$ git clone https://github.com/riscv-gnu-toolchain
project$ cd riscv-gnu-toolchain
riscv-gnu-toolchain$ ./configure --prefix=/opt/riscv --with-arch=rv32i --with-abi=ilp32
riscv-gnu-toolchain$ make
```

Código 3. Instalación de RISC-V GNU toolchain

Esta herramienta puede compilarse a partir del código fuente o instalarse desde distribuciones precompiladas. Es necesario asegurar que el compilador está configurado para soportar las extensiones que se van a usar en el *core*, como Zicntr o Zxcfu.

Para poder integrar el núcleo en una FPGA se crea un proyecto en Vivado y se hace uso de la placa Nexys4 DDR, se habilitan los pines, se escoge una frecuencia de 100 MHz y se configura el SoC mediante la activación de módulos en el archivo “neorv32_config.vhd”. Una vez configurado el entorno y como primera prueba funcional se compila el clásico programa “Hello World” que demuestra la correcta integración de núcleo con la FPGA y el programa de comunicación serie Cutecom (Figura 76).

```
1
2
3
4 << NEORV32 Bootloader >>
5
6 BLDV: Mar 15 2024
7 HWV: 0x01090906
8 CLK: 0x05f5e100
9 MISA: 0x40801104
10 XISA: 0x00000083
11 SOC: 0x0003800d
12 IMEM: 0x00004000
13 DMEM: 0x00002000
14
15 Autoboot in 8s. Press any key to abort.
16 Aborted.
17
18 Available CMDs:
19 h: Help
20 r: Restart
21 u: Upload
22 s: Store to flash
23 l: Load from flash
24 x: Boot from flash (XIP)
25 e: Execute
26 CMD:> u
27 Awaiting neorv32_exe.bin... OK
28 CMD:> e
29 Booting from 0x00000000...
30
31
32
33 ## ## ##### ##### ## ## ##### ##### ## ## ## ##
34 ##### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
35 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
36 ## ## ## ##### ## ## ##### ## ## ## ## ## ## ## ##
37 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
38 ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ##
39 ## ## ##### ## ## ## ## ## ##### ##### ## ## ## ##
40
41 Hello world! :)
```

Figura 76. Hello World ejecutado en NEORV32

Después de validar el funcionamiento básico, se procede a ejecutar un algoritmo más complejo: la ordenación por inserción (Código 4). Este ejemplo visto en el Capítulo 2, se cargó en el sistema a través del gestor de arranque incluido en la ROM del procesador y el programa de comunicación Cutecom usando una interfaz UART.

```
int main()
{
    int lista[TAM]=
    {12,10,5,6,1,3,13,0,15,16,111,33,22,20,4,8,1,3,77,2,12,10,5,6,1,3,13,0,15,16,111,
    33,22,20,4,8,1,3,77,2};
    int temp=0;
    int i,j;
    long Begin_Time, End_Time, User_Time;
    neorv32_rte_setup();
    neorv32_uart0_setup(BAUD_RATE, 0);
    neorv32_rte_print_logo();
    neorv32_uart0_printf("La lista DESORDENADA es: \n");

    Begin_Time = (long)neorv32_mtime_get_time();

    for (i=0;i<TAM;i++)
        neorv32_uart0_printf("%u ",(uint32_t)lista[i]);
    for (i=1;i<TAM;i++)
    {
        for (j=0;j<TAM-1;j++)
        {
            if (lista[j] > lista[j+1])
            {
                temp = lista[j];
                lista[j]=lista[j+1];
                lista[j+1]=temp;
            }
        }
    }
    neorv32_uart0_puts("\nLos valores ORDENADOS de lista son: \n");
    for(i=0;i<TAM;i++)
        neorv32_uart0_printf("%u ",(uint32_t)lista[i]);
    {
        End_Time = (long)neorv32_mtime_get_time();
    }
    neorv32_uart0_printf ("Execution ends\n");
    User_Time = End_Time - Begin_Time;
    neorv32_uart0_printf("NEORV32: Total cycles:      %u\n", (uint32_t)User_Time);
    return 0;
}
```

Código 4. Código fuente del programa de ordenación de datos adaptados a NEORV32

Para ello, se inicializó un array de 40 datos sin ordenar y se obtuvo una lista correctamente ordenada tras la ejecución del algoritmo. El tiempo de ejecución se midió usando la función `neorv32_mtime_get_time`, que accede al temporizador interno del sistema. Se obtuvo un resultado de **13980217 ciclos** de reloj (Figura 77).

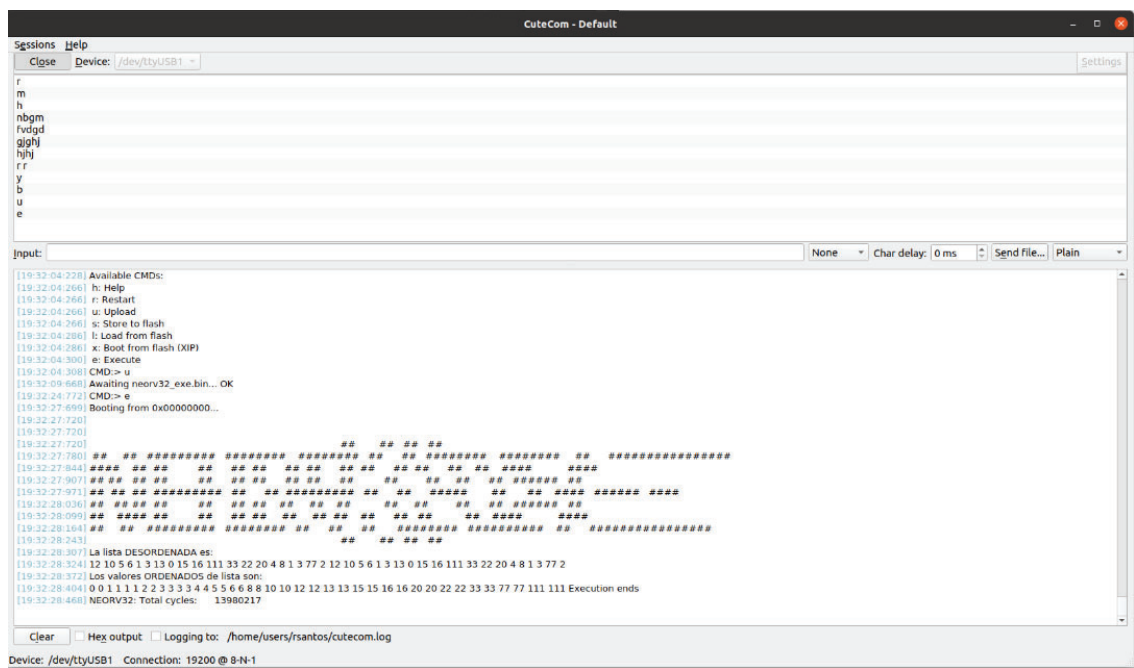


Figura 77. Resultados de la ejecución de ordenación por inserción

6.1.2. Desarrollo del *hardware*

NEORV32 es un procesador RISC-V de código abierto que está pensado como procesador auxiliar listo para usar dentro de un diseño de SoC más grande o como microcontrolador personalizado.

El sistema es altamente configurable y proporciona periféricos comunes opcionales como memorias integradas, temporizadores, interfaces serie, etc. integrados, temporizadores, interfaces serie, puertos de E/S de propósito general y una interfaz de bus externa para conectar IP personalizadas como memorias, NoC, etc. conectar IP personalizados como memorias, NoCs y otros periféricos. La depuración en línea y en el sistema está compatible con OpenOCD/gdb accesible a través de JTAG.

Se presta especial atención a la seguridad de ejecución para proporcionar un comportamiento definido y predecible en cualquier momento. La CPU asegura de que todos los accesos a la memoria se reconocen como válidas. Las instrucciones mal formadas o inválidas no se ejecutan. Cuando se produce una situación inesperada, se informa al código de aplicación mediante excepciones de *hardware*.

6.1.3. Integración de módulos externos en el diseño

Tras obtener los resultados del *core*, pasamos a evaluar la facilidad de integración de nuevas funcionalidades en el sistema usando el *core* NEORV32. Para ello, se diseña un nuevo proyecto en Vivado 2024.1 usando la metodología de diseño por bloques. En este entorno, el núcleo NEORV32 se encapsula como una IP personalizada y se integran algunos periféricos del catálogo de Vivado (Figura 78).

6.1.3.1. Integración de controladores genéricos (BRAM y GPIO)

El primer paso consiste en la integración mediante el bus AXI4 de controladores genéricos disponibles en el catálogo de Vivado. En concreto se ha añadido un bloque

6.1. Implementación de NEORV32

AXI GPIO, un controlador AXI BRAM y un bloque de memoria BRAM. Además, se han hecho externos los pines de la UART interna del NEORV32.

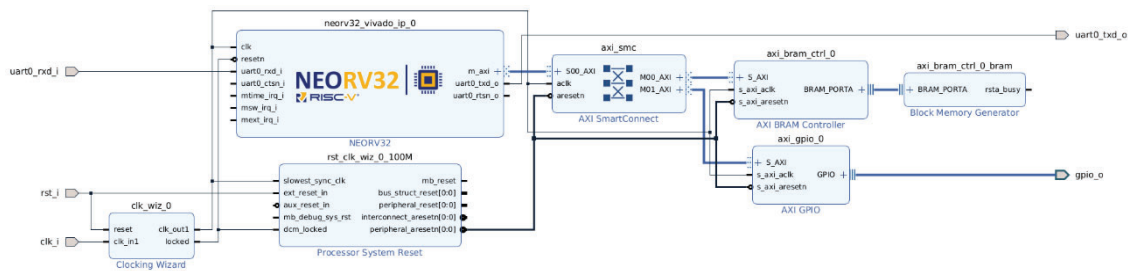


Figura 78. Diseño de bloques implementado en Vivado

El diseño se simula, sintetiza e implementa correctamente y se genera el correspondiente *bitstream* para su volcado en la FPGA. A continuación, se muestran los resultados de prestaciones obtenidos tras la implementación.

En la implementación hardware realizada para NEORV32 el nivel de utilización de recursos es muy bajo, por lo que queda espacio disponible para la integración del resto del sistema. En la Figura 79 se muestra la implementación de los módulos que conforman el núcleo NEORV32, incluyendo la unidad de procesamiento, periféricos integrados como UART, temporizadores y GPIO, así como bloques de memoria interna (BRAM).

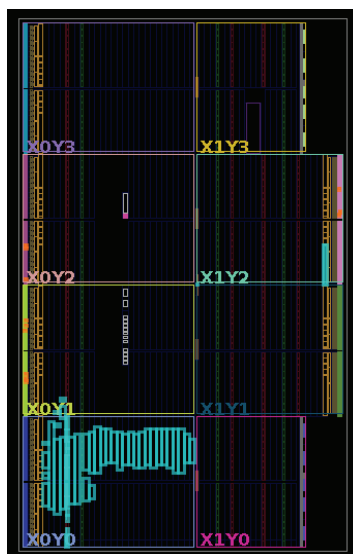


Figura 79. Implementación de NEORV32

En concreto, en la Figura 80, Figura 81 y Figura 82 se muestran detalles de las prestaciones obtenidas para la implementación del procesador. En la Figura 83 se resume la configuración de los IOB del diseño (detalles completos en la Figura 88). En las figuras siguientes (Figura 84, Figura 85, Figura 86 y Figura 87) se muestra la implementación, el análisis temporal realizado para un reloj de 100 MHz y el consumo de potencia total obtenido para el diseño completo. Como se puede apreciar, el impacto del procesador en el diseño completo es mínimo.

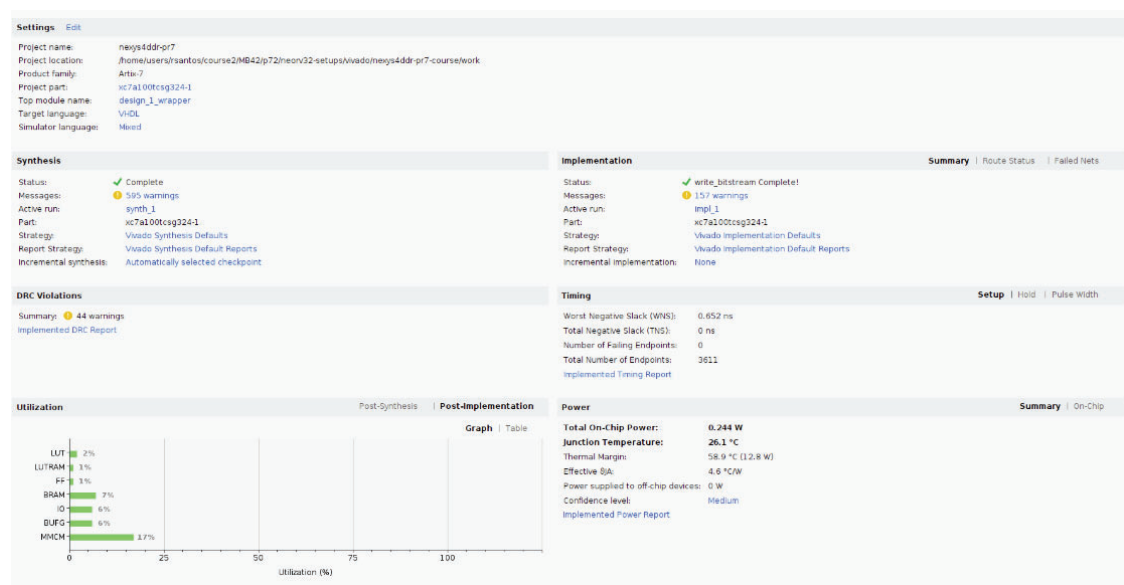


Figura 80. Prestaciones del bloque IP NEORV32

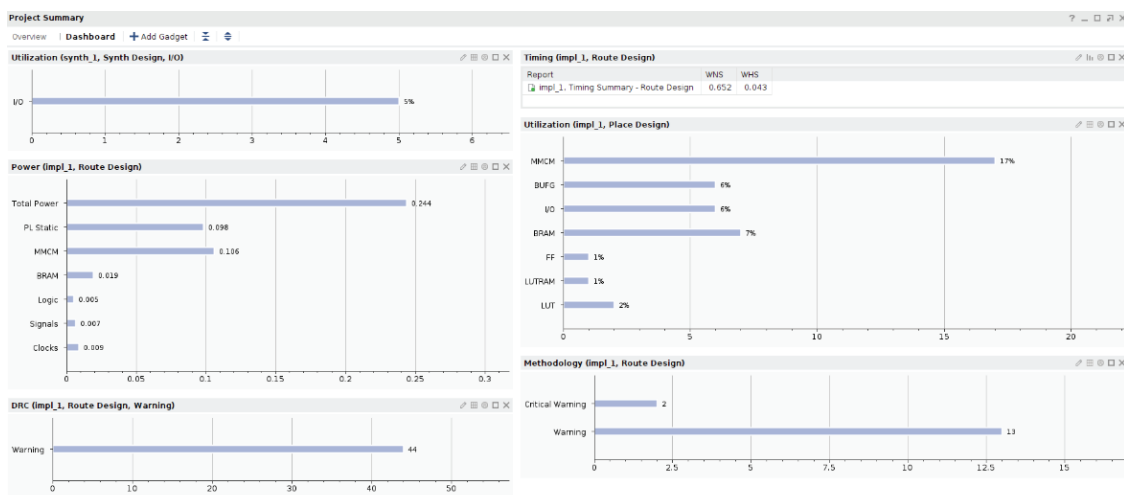


Figura 81. Detalles de los resultados de prestaciones de la implementación del bloque IP NEORV32

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
design_1_wrapper	1582	1532	2	9	12	2	1
design_1_i (design_1)	1582	1532	2	9	0	2	1

Figura 82. Impacto en la utilización de recursos del IP NEORV32

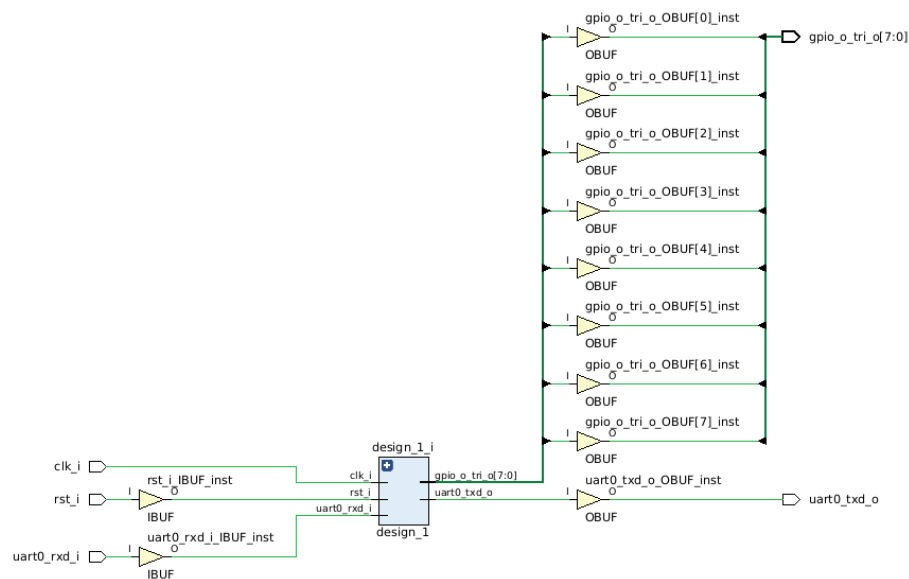


Figura 83. Terminales de E/S del diseño

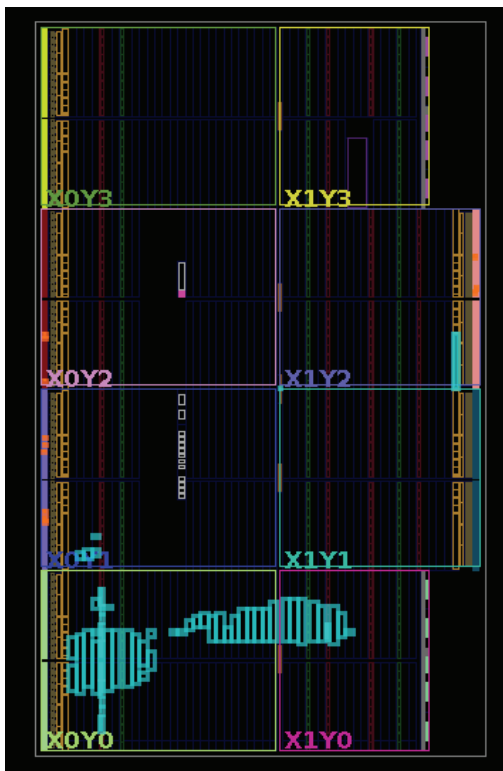


Figura 84. Detalles de la implementación final del diseño

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.579 ns	Worst Hold Slack (WHS):	0.030 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5530	Total Number of Endpoints:	5530	Total Number of Endpoints:	2204

Figura 85. Análisis temporal para una frecuencia de referencia de 100 MHz

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 0.252 W
Design Power Budget: Not Specified
Process: typical
Power Budget Margin: N/A
Junction Temperature: 26.1°C
 Thermal Margin: 58.9°C (12.8 W)
 Ambient Temperature: 25.0 °C
 Effective θ_{JA} : 4.6°C/W
 Power supplied to off-chip devices: 0 W
 Confidence level: Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

On-Chip Power

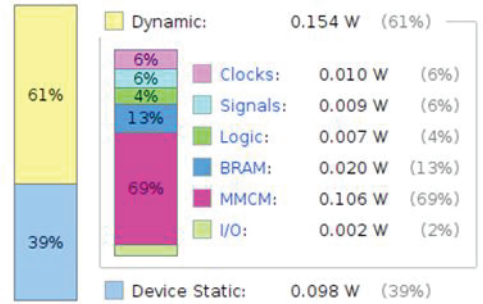


Figura 86. Análisis de potencia del diseño basado en NEORV32

Name	^1	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
design_1_wrapper		2291	2132	3	829	2264	27	9.5	12	2	1
design_1_i (design_1)		2291	2132	3	829	2264	27	9.5	0	2	1
axi_bram_ctrl_0 (desi		25	12	0	15	25	0	0	0	0	0
axi_bram_ctrl_0_bram		4	5	0	3	3	1	1	0	0	0
axi_gpio_0 (design_1		47	72	0	25	47	0	0	0	0	0
axi_smc (design_1_ax		312	388	0	146	311	1	0	0	0	0
clk_wiz_0 (design_1_c		0	0	0	0	0	0	0	0	2	1
neorv32_vivado_ip_0		1890	1622	3	672	1866	24	8.5	0	0	0
rst_clk_wiz_0_100M (c		16	33	0	10	15	1	0	0	0	0

Figura 87. Resultados de implementación final de NEORV32

Name	Port	I/O Std	Vcco	Slew	Drive Strength (...)	Off-Chip Termina...	Remaining Margin...
I/O Bank 0 (0)							
I/O Bank 14 (5)		LVC MOS33	3.30	SLOW	12	FP_VTT_50	
N14	gpio_o_tri_o[3]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	98.26
R18	gpio_o_tri_o[4]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	96.11
V17	gpio_o_tri_o[5]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	81.67
U17	gpio_o_tri_o[6]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	88.92
U16	gpio_o_tri_o[7]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	81.41
I/O Bank 15 (3)		LVC MOS33	3.30	SLOW	12	FP_VTT_50	
H17	gpio_o_tri_o[0]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	91.94
K15	gpio_o_tri_o[1]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	92.37
J13	gpio_o_tri_o[2]	LVC MOS33	3.30	SLOW	12	FP_VTT_50	89.27
I/O Bank 16 (0)							
I/O Bank 34 (0)							
I/O Bank 35 (1)		LVC MOS33	3.30	SLOW	12	FP_VTT_50	
D4	uart0_tx_d_o	LVC MOS33	3.30	SLOW	12	FP_VTT_50	99.64

Figura 88. Detalles de los IOB utilizados

Una vez cargada la configuración de referencia de la FPGA, se accede a la aplicación de comunicación serie Cutecom para cargar la aplicación binaria “demo_bram” utilizando el loader genérico que previamente se ha cargado desde la ROM de programas del procesador. Este programa escribe en 10 regiones consecutivas de la memoria BRAM, comenzando por la dirección 0xE0000000 y posteriormente transferir los datos leídos a la dirección 0x40000000, permitiendo comprobar la integridad de la transferencia de datos (Figura 89).

6.1. Implementación de NEORV32

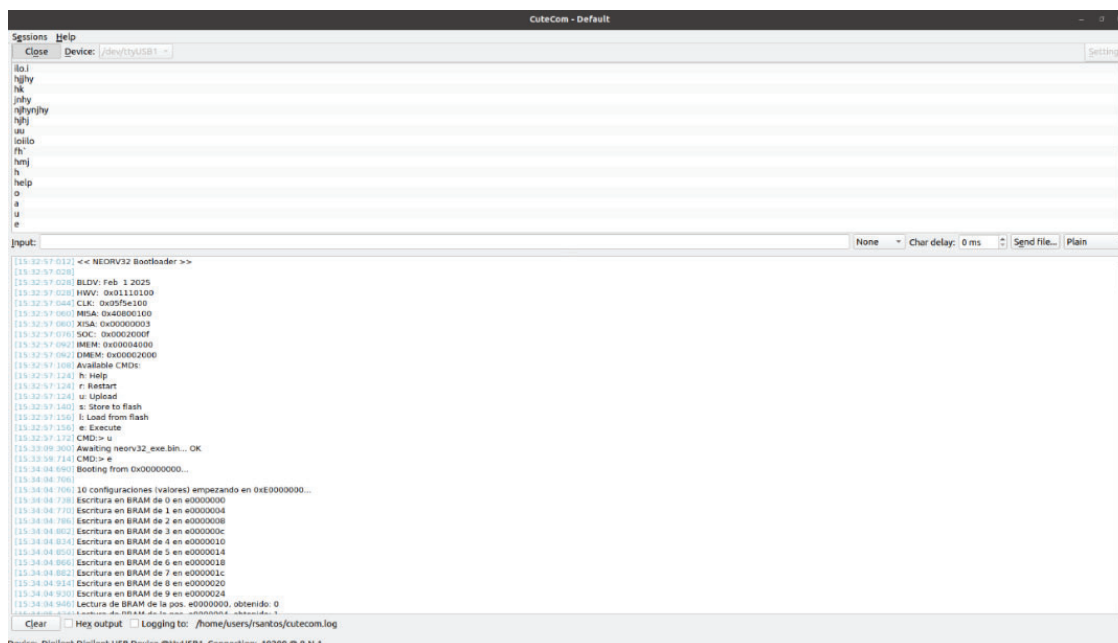


Figura 89. Resultado de la ejecución del ejemplo demo bram

6.1.3.2. Integración de transferencias DMA

En este caso se configura y utiliza el DMA de NEORV32 para realizar transferencias de datos entre regiones de memoria sin la intervención directa de la CPU, siguiendo el esquema de buses mostrado en la Figura 50. En la Figura 90 se puede observar el resultado de la ejecución de la activación del controlador DMA (*Direct Memory Acces*). En el ejemplo `demo_dma` se incluyen dos casos de test:

- Caso 1: Se transfiere byte a byte con conversión *endianness* un bloque de 16 bytes, se activa DMA_BSWAP para invertir el orden de los bytes y se confirma que los datos en destino coinciden con los esperados.
- Caso 2: Se transfiere palabra a palabra con valor fijo. Para ello se transfiere el mismo valor `dma_src[0]` a las cuatro posiciones de `dma_dst` y se configura para leer una palabra constante y escribir en direcciones incrementales.

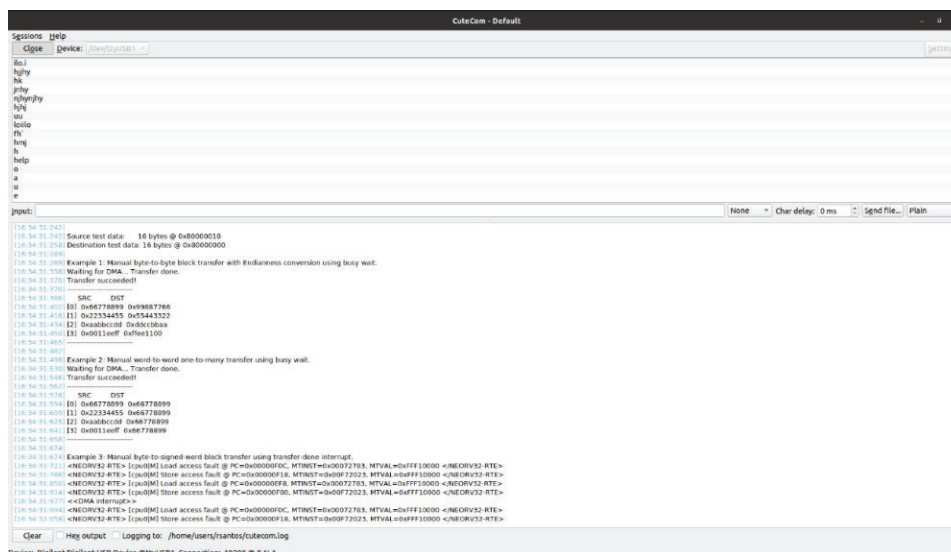


Figura 90. Resultado test demo dma

6.1.3.3. Extensión de instrucciones usando la CFU

Para ejecutar el siguiente ejemplo, ha sido necesario habilitar las siguientes extensiones del conjunto de instrucciones RISC-V en el bloque CPU NEORV32:

- Zicntr: proporciona contadores de rendimiento como *cycle*, *instret*, etc.
- Zxcfu: permite la implementación de instrucciones definidas por el usuario a través de la Unidad de Funciones Personalizadas (CFU).

Estas extensiones se activan antes de la síntesis mediante los parámetros de configuración del núcleo disponibles en el bloque de CPU de Vivado.

En este caso se integra un algoritmo de cifrado ligero por bloques XTEA para IoT. Se basa en una estructura de Feistel de 64 bits, ejecutando un total de 64 rondas con una clave de 128 bits. Es sensible a distintos ataques de criptoanálisis, pero representa un buen compromiso entre seguridad y complejidad, suficiente en algunas aplicaciones que no requieran un alto grado de seguridad.

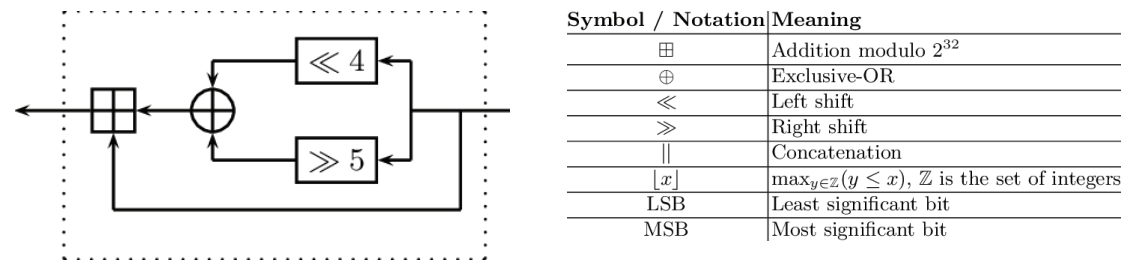


Figura 89. XTEA Feistel round [158]

En la Figura 91 se muestra la ejecución del algoritmo XTEA en NEORV32, usando tanto la integración de un acelerador CFU como implementado usando la CPU estándar. Como se puede observar, la inclusión de instrucciones hechas a medida, en este caso una instrucción que acelera la ejecución de la estructura de Feistel, permite una reducir en un factor de 6 el número de ciclos necesarios para completar el cifrado del texto completo.

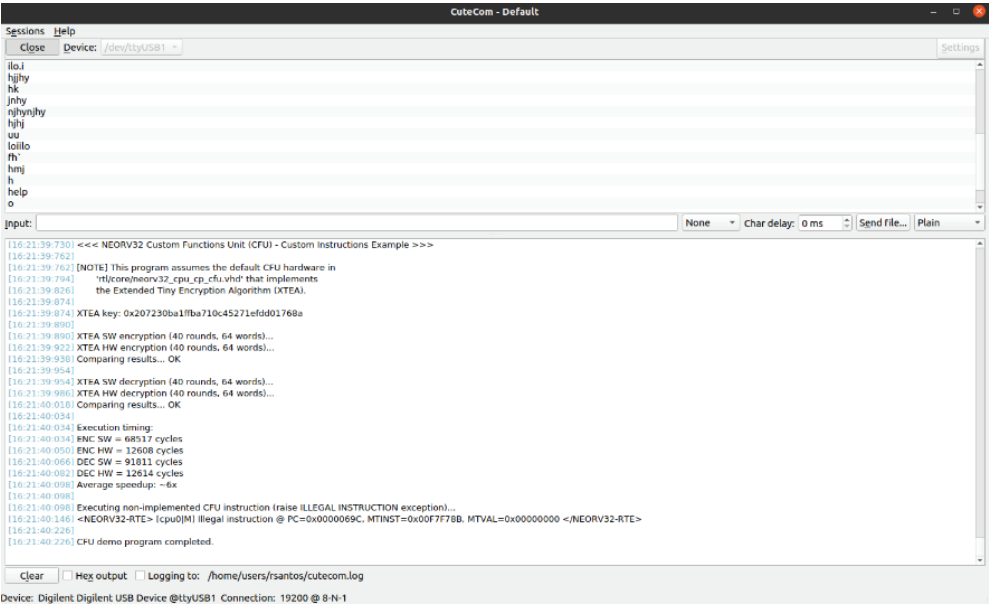


Figura 91. Integración del algoritmo de cifrado ligero XTEA

El sistema final incluye la utilización del DMA para la transferencia de datos y el cifrado y descifrado con el algoritmo XTEA usando instrucciones personalizadas CFU. Para ello se adaptan las funciones de cifrado `xtea_encrypt_dma_text` (Código 5) y descifrado `xtea_decrypt_dma_text` (Código 6) para nuestro sistema.

En primer lugar, se configuran las claves en registros CFU escribiendo 4 palabras de 32 bits que se cargan en registros CSR, seguidamente se recorre el texto fuente de 8 bytes (64 bits), se inicializa el módulo *hardware* de cifrado XTEA y se realiza un cifrado iterativo por rondas en `XTEA_ROUNDS` y finalmente el resultado se almacena en un buffer de salida `xtea_cypher_buffer`.

```
void xtea_encrypt_dma_text(void) {

    neorv32_uart0_printf("\nIniciando cifrado XTEA por CFU...\n");

    // Configuran las claves en CFU
    neorv32_cpu_csr_write(CSR_CFUREG0, key[0]);
    neorv32_cpu_csr_write(CSR_CFUREG1, key[1]);
    neorv32_cpu_csr_write(CSR_CFUREG2, key[2]);
    neorv32_cpu_csr_write(CSR_CFUREG3, key[3]);

    neorv32_uart0_printf("\nXTEA key: 0x%x%x%x%x\n\n",
                        neorv32_cpu_csr_read(CSR_CFUREG0),
                        neorv32_cpu_csr_read(CSR_CFUREG1),
                        neorv32_cpu_csr_read(CSR_CFUREG2),
                        neorv32_cpu_csr_read(CSR_CFUREG3));

    for (int i = 0; i < TEXTO_TAM; i+= 8) {
        uint32_t v0= *(uint32_t*)&dma_texto_src[i];
        uint32_t v1= *(uint32_t*)&dma_texto_src[i+4];

        xtea_hw_init(0);
        for (int r = 0; r < XTEA_ROUNDS; r++) {
            v0 = xtea_hw_enc_v0_step(v0, v1);
            v1 = xtea_hw_enc_v1_step(v0, v1);
        }

        xtea_cypher_buffer[i/4]=v0;
        xtea_cypher_buffer[i/4 + 1]=v1;
    }
    neorv32_uart0_printf("Cifrado completado.\n");
}
```

Código 5. Función `xtea_encrypt_dma_text`

Por otro lado, la función de descifrado `xtea_decrypt_dma_text` (Código 6) carga las 4 palabras, recorre el texto cifrado en bloques de 8 bytes (64 bits) y tras recorrer el texto se inicializa el *hardware* de descifrado con el valor de suma total (`XTEA_ROUNDS * xtea_delta`) que es el estado inicial requerido por XTEA para descifrar (el descifrado para este caso requiere recorrer las rondas en orden inverso, comenzando por el valor acumulado en la suma). Tras este paso, comienza el descifrado por rondas en orden inverso al cifrado y finalmente se almacenan secuencialmente en el buffer de destino `dma_texto_dst`.

```

void xtea_decrypt_dma_text(void) {
    neorv32_uart0_printf("\nIniciando descifrado XTEA por CFU...\n");
    // Se configuran las claves en CFU
    neorv32_cpu_csr_write(CSR_CFUREG0, key[0]);
    neorv32_cpu_csr_write(CSR_CFUREG1, key[1]);
    neorv32_cpu_csr_write(CSR_CFUREG2, key[2]);
    neorv32_cpu_csr_write(CSR_CFUREG3, key[3]);
    for (int i = 0; i < TEXTO_TAM; i += 8) {
        uint32_t v0 = xtea_cypher_buffer[i / 4];
        uint32_t v1 = xtea_cypher_buffer[i / 4 + 1];
        xtea_hw_init(XTEA_ROUNDS * xtea_delta); // inicializa con suma para
descifrado
        for (int r = 0; r < XTEA_ROUNDS; r++) {
            v1 = xtea_hw_dec_v1_step(v0, v1); // macros de descifrado
            v0 = xtea_hw_dec_v0_step(v0, v1);
        }
        dma_texto_dst[i]      = *((char*)&v0;
        dma_texto_dst[i + 1] = *((char*)&v0 + 1);
        dma_texto_dst[i + 2] = *((char*)&v0 + 2);
        dma_texto_dst[i + 3] = *((char*)&v0 + 3);
        dma_texto_dst[i + 4] = *((char*)&v1;
        dma_texto_dst[i + 5] = *((char*)&v1 + 1);
        dma_texto_dst[i + 6] = *((char*)&v1 + 2);
        dma_texto_dst[i + 7] = *((char*)&v1 + 3);
    }
    neorv32_uart0_printf("Descifrado completado.\n");
}

```

Código 6. Función `xtea_decrypt_dma_text`

En esta parte del Código 7 se configura el DMA para la transferencia de bloque a bloque con incremento de dirección y se transfiere el contenido cifrado `xtea_cypher_buffer` al `dma_texto_dst` y se espera a que el DMA finalice antes de continuar con `DMA_STATUS_IDLE`. Finalmente, se imprimen los primeros 200 bytes cifrados para verificación.

```

neorv32_uart0_printf("Transfiriendo datos cifrados por DMA...\n");
uint32_t dma_config_xtea =
    DMA_CMD_B2B |
    DMA_CMD_SRC_INC |
    DMA_CMD_DST_INC;
neorv32_dma_transfer(
    (uint32_t)&xtea_cypher_buffer[0],
    (uint32_t)&dma_texto_dst[0],
    TEXTO_TAM,
    dma_config_xtea);
while(neorv32_dma_status() != DMA_STATUS_IDLE);
neorv32_uart0_printf("Transferencia completada.\n");
//Se muestran los datos cifrados
neorv32_uart0_printf("Primeros bytes cifrados:\n");
for(int i=0; i<200; i++){
    neorv32_uart0_printf("%02x ", dma_texto_dst[i]);
}

```

Código 7. Transferencia de datos por DMA

En la Figura 92 se muestra el proceso en el que se comienza con la recepción de un bloque de texto plano de tamaño fijo (1024 caracteres) a través de la UART almacenando cada byte en el buffer `dma_texto_src`. Una vez completada la recepción del texto, se procede al cifrado mediante el algoritmo XTEA, ejecutando tanto en *hardware* como en *software* (con el objetivo de comparar el rendimiento de ambas implementaciones). Luego, los datos cifrados se transfieren mediante DMA a otro buffer, y se imprime una muestra para verificación. Posteriormente, se descifra el texto usando la misma CFU y se valida por UART. Finalmente, se comparan los métodos de transferencia con la medición de ciclos de CPU del contador `MCYCLE`, lo que permite demostrar la eficiencia de cada método y el desempeño del sistema.

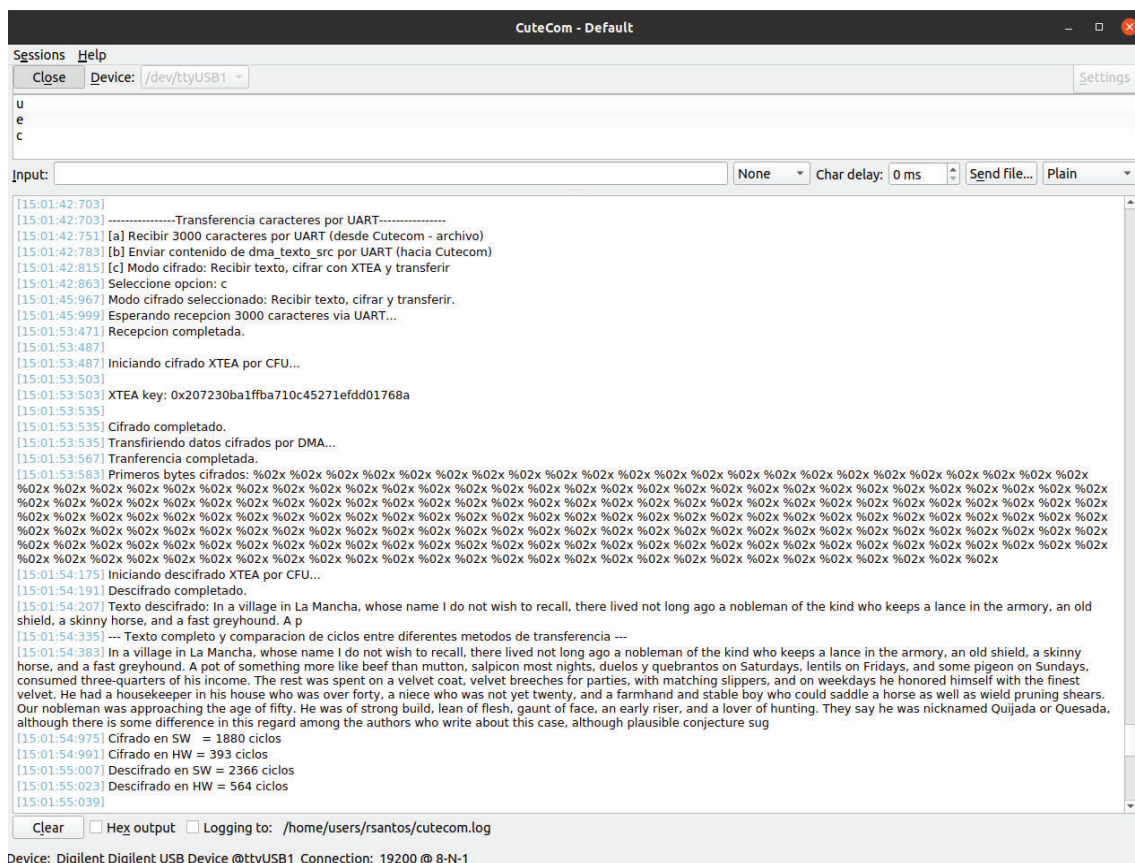


Figura 92. Aplicación XTEA usando transferencias DMA

6.2. Implementación de AIRISC

Como se ha indicado, el núcleo RISC-V AIRISC implementa una arquitectura Harvard de 32 bits con un pipeline de cinco etapas. Tiene dos interfaces AHB Lite para instrucciones y acceso a datos, y permite añadir extensiones mediante una interfaz de coprocesador. Además, incluye una unidad multiplicadora/divisora (MUL/DIV), instrucciones comprimidas (RV32IMC) y varios periféricos estándar como temporizador compatible con MTIME, una UART y un SPI. Además, el *core* incluye un bloque ICAP que permite la reconfiguración parcial de forma dinámica (*Dynamic Function Exchange*). Para ello se incluyen *bitstream* en las direcciones de memoria asignados al módulo ICAP.

6.2.1. Configuración del core AIRISC

Para poder implementar el núcleo AIRISC es necesario preparar el entorno inicialmente, instalando Cadence Incisive, OpenOCD y GDB. Además, para trabajar con AIRISC, también es necesario el *toolchain* RISC-V GCC dado que es compatible con el conjunto de instrucciones RV32IMAC, simulador Verilator que será útil para validar el diseño antes de la síntesis y Vivado para la generación del flujo de bits y la implementación en FPGA. EL diagrama general, incluyendo las interfaces con memoria se muestra en la Figura 93.

En esta implementación se ha clonado del repositorio [142] el proyecto AIRISC y en Vivado se ha creado un nuevo proyecto RTL en el que se selecciona la placa Nexys Video de la familia Artix 7. Seguidamente se importa el archivo de restricciones “constraints.xdc” y se añade el archivo toplevel “FPGA_Top.v” así como los archivos RTL en lenguaje Verilog. Se instancian los bloques IP necesarios y se selecciona una entrada de reloj de 100 MHz.

En la Figura 94, se muestra una parte del bloque de interconexión del núcleo dónde se destacan los bloques principales: DUT (*Device Under Test*), SRAM externa y señales de comunicación como UART, SPI y GPIO. En DUT se pueden observar las diferentes interfaces de memoria que sugieren el uso del bus AHB-Lite: imem_hrddata[31:0], inem_hready e inem_hresp (interfaces de lectura de instrucciones) y dmem_hrddata[31:0], dmem_hready y dmem (interfaces de lectura de datos); interfaces externas: gpio0_in[7:0], spi0_miso_in, spi0_mosi_in, spi0_sclk_in, spi0_ss_in, uart0_rx y ext_interrupt; señales de test: tck, tms, tdi y testmode.

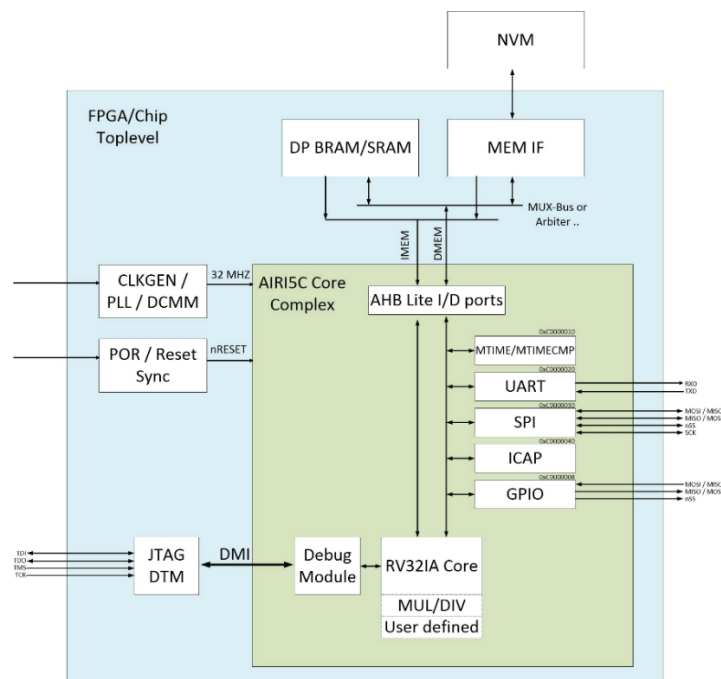


Figura 93. Diagrama de bloques de AIRISC

6.2. Implementación de AIRISC

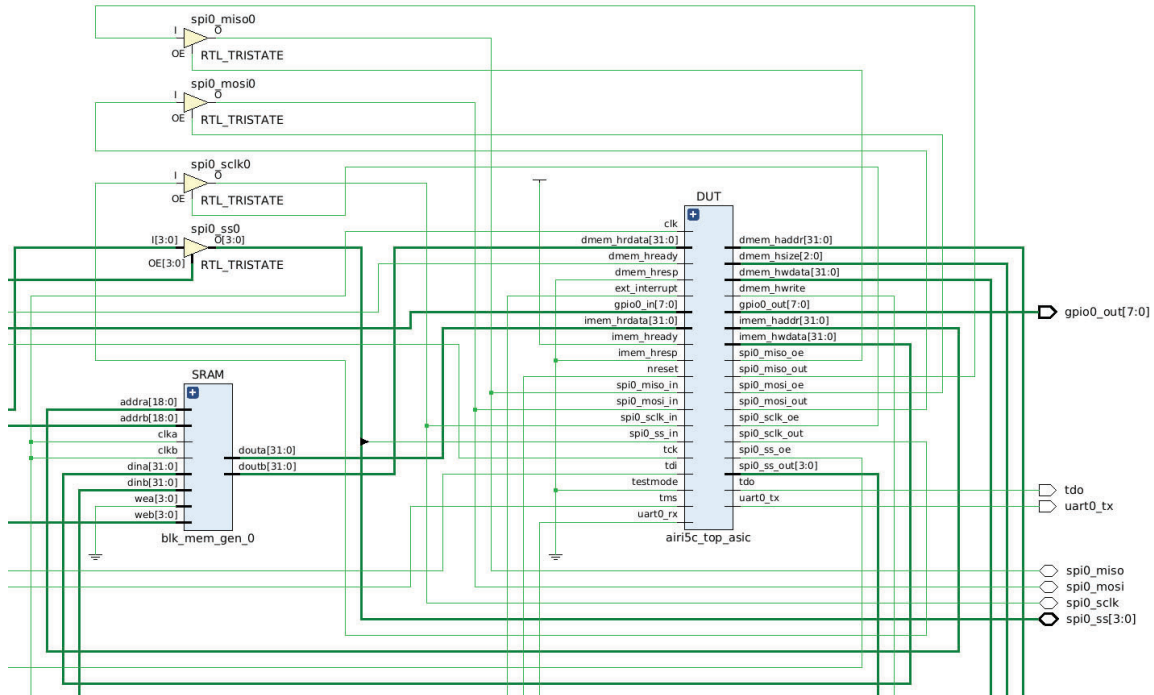


Figura 94. Esquema de interconexión AIRISC

En los resultados obtenidos de los *testbench* para las configuraciones de las instrucciones RV32IMFC (Figura 95, Figura 96 y Figura 97), se puede comprobar que el núcleo AIRISC ejecuta correctamente las instrucciones definidas, lo que permite avanzar hacia las fases de síntesis y verificación en *hardware*.

```
==== AIRISC TESTBENCH STARTUP ====
DUT: FPGA
=====
> Init and Basic tests
=====
Core configuration: RV32IMFC
Info: VDD: 0, RESET: 1, CLK: 0, CLKQSPI: 0, GPIO_I: GPIO_0, EXT_INT: 0, tms: 0, tdi: 0, tck: 0, sen: 0, sdi: 0
Warning: [neotrng WARNING] Implementing non-physical pseudo-RNG!
File: /home/users/vproject/airisc/airisc_core_complex/external/neotrng/rtl/neotrng.vhd
Info: 0 ps Iteration: 0 Process: /airisc_top_tb/DUT/trng/trng_core/entropy_source(2)/neotrng_cell_inst/sln_node_true/line_310
Warning: [neotrng WARNING] Implementing non-physical pseudo-RNG!
File: /home/users/vproject/airisc/airisc_core_complex/external/neotrng/rtl/neotrng.vhd
Info: 0 ps Iteration: 0 Process: /airisc_top_tb/DUT/trng/trng_core/entropy_source(1)/neotrng_cell_inst/sln_node_true/line_310
Warning: [neotrng WARNING] Implementing non-physical pseudo-RNG!
File: /home/users/vproject/airisc/airisc_core_complex/external/neotrng/rtl/neotrng.vhd
Info: 0 ps Iteration: 0 Process: /airisc_top_tb/DUT/trng/trng_core/entropy_source(0)/neotrng_cell_inst/sln_node_true/line_310
Note: [neotrng NOTE] << neotrng v3 - A tiny and Platform-Independent True Random Number Generator >>
Info: 0 ps Iteration: 0 Process: /airisc_top_tb/DUT/trng/trng_core/line_104 File: /home/users/vproject/airisc/airisc_core_complex/external/neotrng/rtl/neotrng.vhd
Info: VDD: 1, RESET: 1, CLK: taktet, CLKQSPI: taktet, GPIO_I: GPIO_0, EXT_INT: 0, tms: 0, tdi: 0, tck: 0, sen: 0, sdi: 0
Info: JTAG BYPASS test: reset..
Info: [USF-XSln-96] XSln completed. Design snapshot 'airisc_top_tb_behav' loaded.
Info: [USF-XSln-97] XSln simulation ran for 1000ms
Launch simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:13 . Memory (MB): peak = 2099.777 ; gain = 125.691 ; free physical = 15627 ; free virtual = 26075
# run -all
Info: JTAG BYPASS test: set to BYPASS and write AC65, LSB first, i.e. 1010011000110101
Info: JTAG BYPASS test: read back: 58cb, 0101100011001101Info: JTAG TAP: reset..
Info: JTAG TAP: send core halt request.
JTAG TAP: wait some cycles.
JTAG TAP: Read status of debug module... 00011043 ok.
```

Figura 95. Resultado testbench RV32IMFC

```
RV32M Instructions
=====
DIV : read mem file for testcase 0blk_mem_gen_v8_4_4 collision detected at time: 479791735000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
DIVU : read mem file for testcase 1blk_mem_gen_v8_4_4 collision detected at time: 48424945000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
MULM : read mem file for testcase 2blk_mem_gen_v8_4_4 collision detected at time: 488658135000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
MULHSU : read mem file for testcase 3blk_mem_gen_v8_4_4 collision detected at time: 508427355000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
MULMU : read mem file for testcase 4blk_mem_gen_v8_4_4 collision detected at time: 52819695000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
MUL : read mem file for testcase 5blk_mem_gen_v8_4_4 collision detected at time: 54796365000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
REM : read mem file for testcase 6blk_mem_gen_v8_4_4 collision detected at time: 556135135000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
REMU : read mem file for testcase 7blk_mem_gen_v8_4_4 collision detected at time: 572241935000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write ad
ress: 14
o.k., running..success.
RV32M Instruction Tests completed with errorcount = 0
```

Figura 96. Resultado testbench RV32M

```

RV32F Test
-----
PADD : read mem file for testcase 00bk_mem_gen_v8_4_4 collision detected at time: 633931395000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PCLAS : read mem file for testcase 10bk_mem_gen_v8_4_4 collision detected at time: 665797575000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PCMP : read mem file for testcase 20bk_mem_gen_v8_4_4 collision detected at time: 670230775000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PCVT : read mem file for testcase 30bk_mem_gen_v8_4_4 collision detected at time: 702388785000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PCVT_M : read mem file for testcase 40bk_mem_gen_v8_4_4 collision detected at time: 734254965000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PDIV : read mem file for testcase 50bk_mem_gen_v8_4_4 collision detected at time: 766631845000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PMADD : read mem file for testcase 60bk_mem_gen_v8_4_4 collision detected at time: 798498015000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
PMIN : read mem file for testcase 70bk_mem_gen_v8_4_4 collision detected at time: 830947865000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
LDST : read mem file for testcase 80bk_mem_gen_v8_4_4 collision detected at time: 863397725000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
MOVE : read mem file for testcase 90bk_mem_gen_v8_4_4 collision detected at time: 895263885000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.
RECORDING : read mem file for testcase 100bk_mem_gen_v8_4_4 collision detected at time: 901564875000, Instance: airisc_top_tb.DUT.SRAM.inst.native_mem_module.blk_mem_gen_v8_4_4_inst, A read address: 14, B write
address: 14
o.k., running..success.

RV32F Tests completed with errorcount =      0
cumulative errors / number of tests:      0 /      60
expected errors:      0
TB PASSED

```

Figura 97. Resultado testbench RV32F

6.2.2. Desarrollo del hardware

En [159] se presenta el mapa de memoria utilizado, que es posible modificar para cambiar la asignación de periféricos o añadir áreas de memoria adicionales como por ejemplo para una memoria no volátil. En la Figura 98, se muestra una sugerencia de distribución básica de las direcciones en el sistema, incluyendo una ROM de depuración, memoria principal (SRAM) y periféricos. La dirección 0x00000000 está dedicada a la depuración del sistema, en 0x80000000 comienza el inicio de la memoria principal y a partir de 0xC0000000 en adelante permite controlar y comunicar con los periféricos.

Al estar las direcciones perfectamente alineadas y espaciadas uniformemente, facilita el acceso desde *software* embebido, así como al estar separada la ROM de depuración de la memoria principal, permite mantener herramientas de diagnóstico activas incluso si el sistema falla.

En la Figura 99, se muestra el comportamiento de las señales del bus AHB-Lite durante los accesos de lectura y escritura a la memoria. Como se puede observar en la figura, la memoria tiene una latencia de un ciclo de reloj, lo que significa que no necesita ciclos de espera para las lecturas. Sin embargo, para el caso de los accesos a escrituras, el valor a escribir no está disponible hasta un ciclo después de haber aplicado la dirección (destacado en los flancos a y b). Además, los datos leídos en el procesador (hrdata) no se actualizan hasta que se produce una nueva transacción (flancos c y d).

Finalmente, en la Figura 100 se muestra el bloque AIRISC con la interfaz AHB.

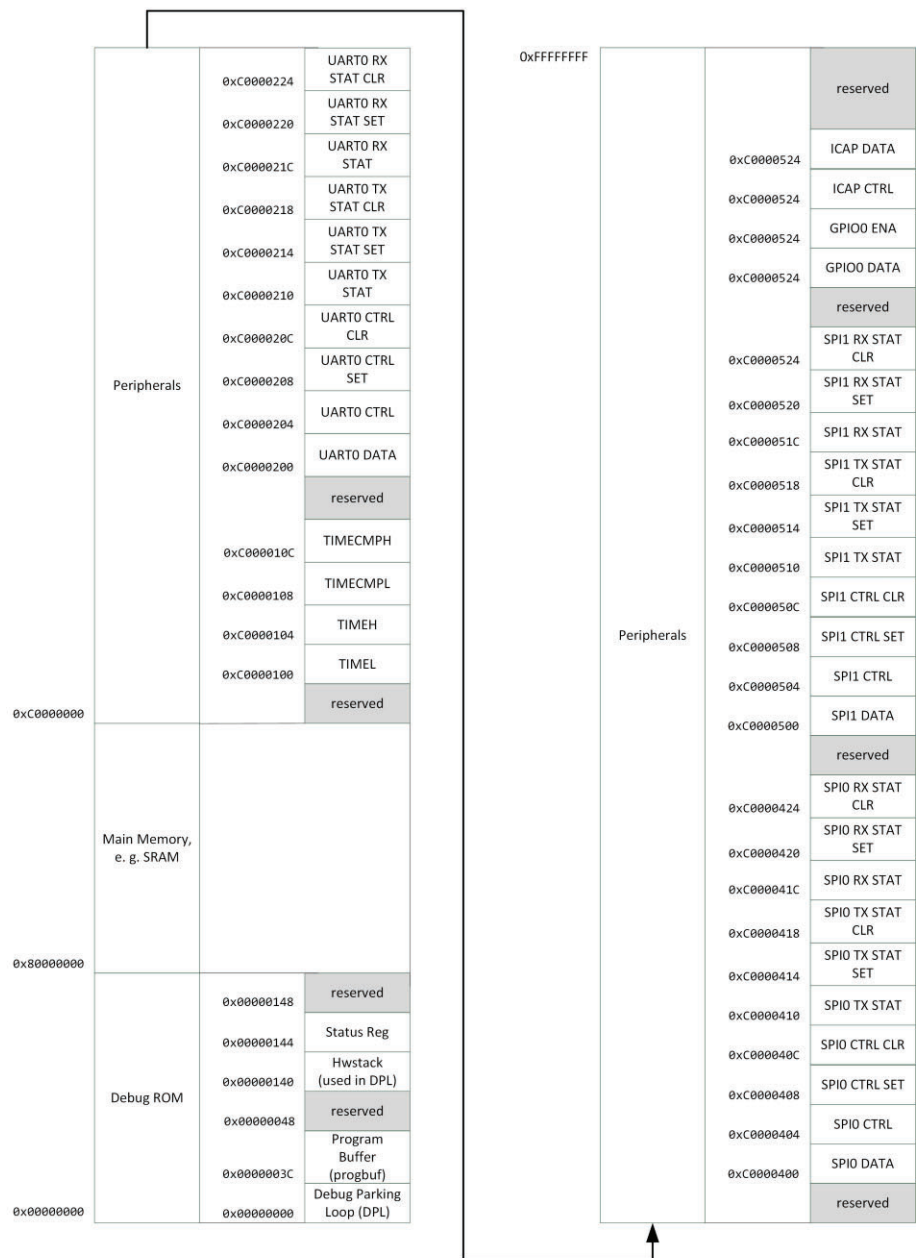


Figura 98. Mapa de memoria AIRISC [159]

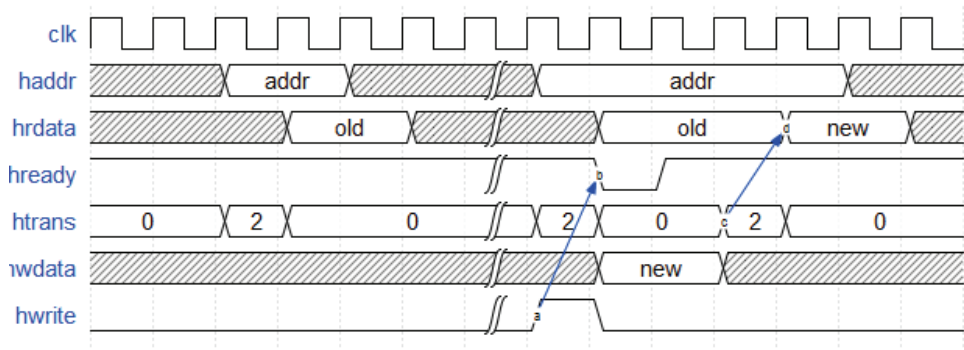


Figura 99. Señales del bus AHB-Lite [159]

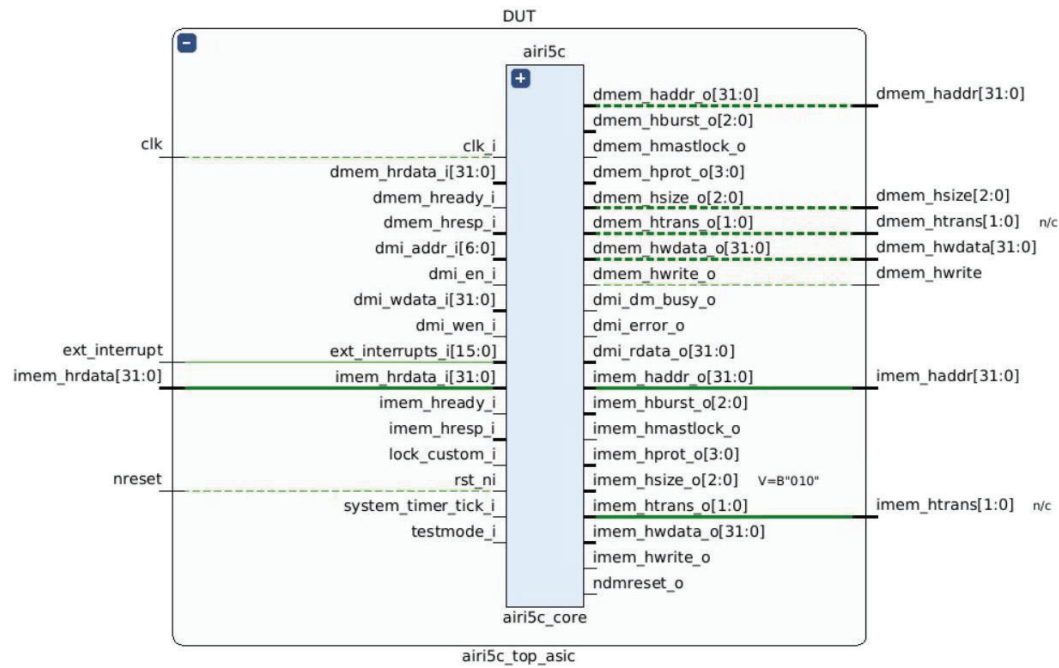


Figura 100. AIRISC con la interfaz AHB

6.2.3. Implementación y análisis de prestaciones

Se ha implementado la plataforma siguiendo los pasos del flujo de diseño de Vivado. En la Figura 101 se presenta una visión global de la implementación realizada en el dispositivo de referencia (Xilinx Artix xc7a200tsbg484-1). Como se puede apreciar, el impacto final en cuanto a la ocupación de recursos es baja, excepto en la utilización de los bloques de memoria BRAM, en la que se sigue la estrategia utilizarla para la memoria externa, dejando espacio para añadir nuevas funcionalidades al sistema. En la Figura 102 se presentan detalles de la implementación de la ruta de datos (*pipeline*) de la CPU, remarcando los bloques que la conforman.

En la Figura 103 se muestran los recursos utilizados, que, tal como se comentó anteriormente, son reducidos excepto para el caso de los bloques BRAM. La utilización de bloques DSP se realiza para implementar las operaciones de multiplicación. Los bloques de memoria permiten implementar el programa de aplicación y otros recursos de almacenamiento local.

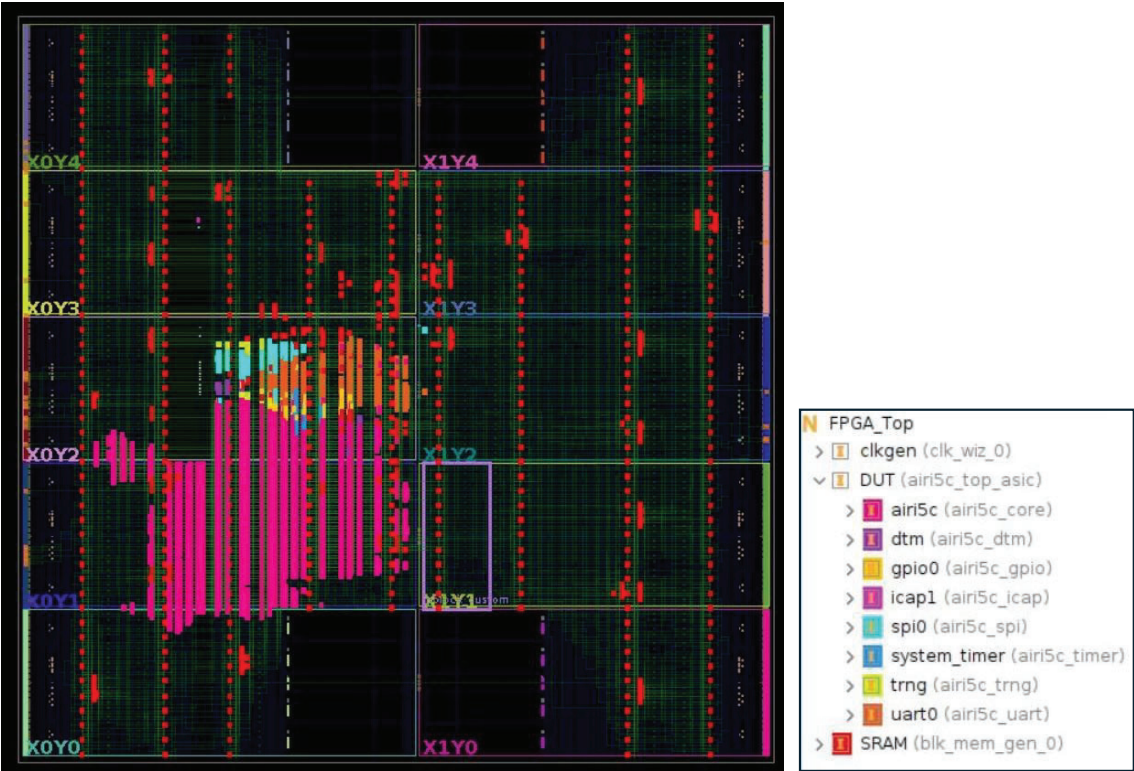


Figura 101. Layout del núcleo AIRISC

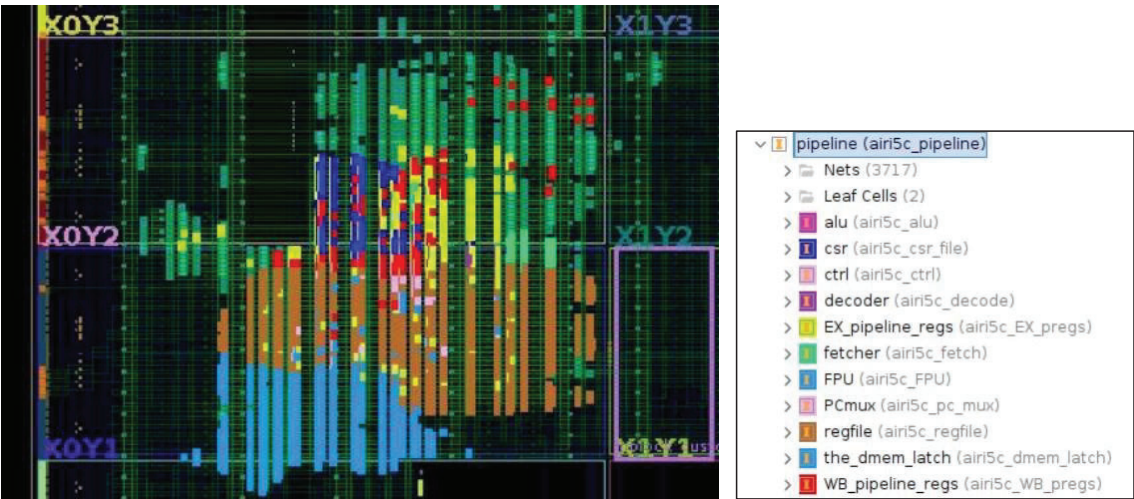


Figura 102. Layout del pipeline de AIRISC

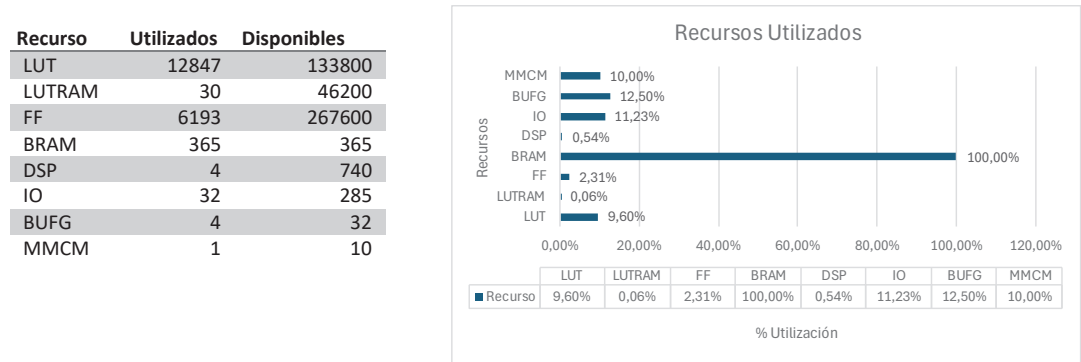


Figura 103. Utilización de recursos utilizados por la plataforma basada en AIRISC

En relación con las prestaciones temporales, el diseño cumple con las restricciones de reloj, para un periodo de 10 ns, suficiente para la mayoría de las aplicaciones empotradas. El histograma (Figura 104) muestra muy pocas rutas con *slack* próximo a cero lo que implica que la mayor parte del diseño tiene holgura para la optimización temporal. En el histograma se han seleccionado las rutas más lentas, en este caso únicamente con dos rutas identificadas.

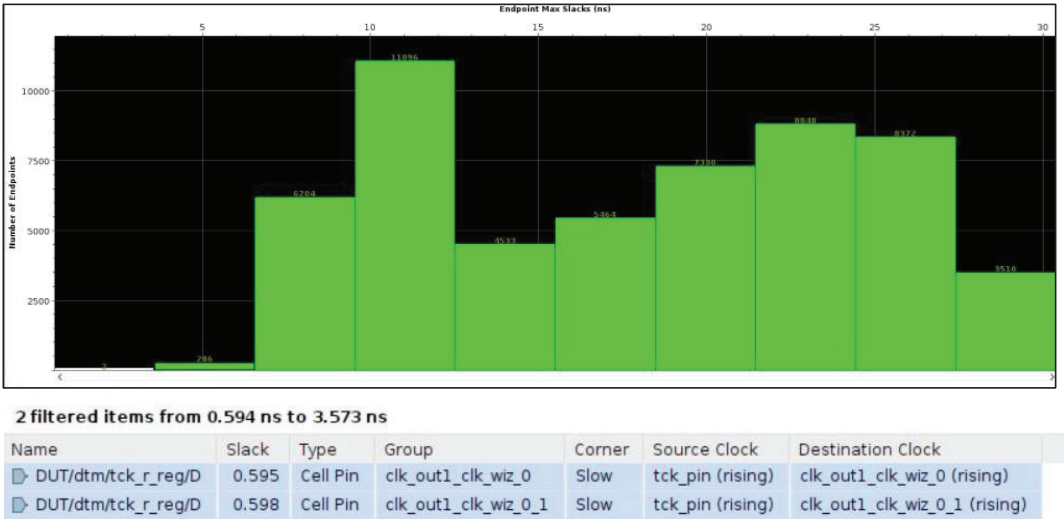


Figura 104. Histograma para el análisis temporal

Para el núcleo AIRISC se muestra en la Figura 105 que el diseño funciona correctamente a 100 MHz. Como se puede apreciar no existen errores en las comprobaciones de *Setup*, *Hold* y *Pulse Width*, obteniendo una *Worst Negative Slack* (WNS) de 0.595 ns y una *Worst Hold Slack* (WHS) 0.031 ns en las 30849 rutas analizadas y 3.000 ns para una *Worst Pulse Width Slack* (WPWS) en las 6953 rutas analizadas. Por tanto, el diseño está bien sincronizado y es seguro para su uso en aplicaciones. La potencia disipada por la plataforma mínima es de 324 mW (Figura 106).

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.595 ns	Worst Hold Slack (WHS): 0.031 ns	Worst Pulse Width Slack (WPWS): 3.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 30849	Total Number of Endpoints: 30849	Total Number of Endpoints: 6953

Figura 105. Análisis temporal de AIRISC

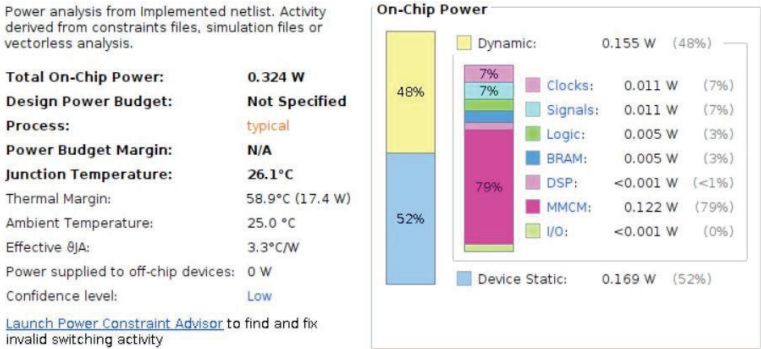


Figura 106. Potencia disipada por AIRISC

6.3. Implementación de CVA5

La implementación del núcleo CVA5 es una propuesta más avanzada con respecto a los núcleos anteriores. CVA5 es un procesador RISC-V de clase aplicación, diseñado específicamente para ofrecer alto rendimiento en aplicaciones que utilizan FPGAs. Con un diseño modular escrito en SystemVerilog y con una arquitectura de 5 etapas de pipeline, es capaz de soportar extensiones RV32IMAFD. La Figura 107 muestra el pipeline de procesador CVA5. El procesamiento de instrucciones se divide en tres etapas principales: *Fetch/Pre-Decode*, *Decode/Issue* y *Write Back*. En la primera etapa, se calcula la dirección de la siguiente instrucción, se traduce la dirección virtual a física y se accede a la instrucción en la caché. También se usan el predictor de saltos y la pila de direcciones de retorno para mejorar el flujo de control. En *Decode/Issue*, se descodifica la instrucción y se emite si hay recursos disponibles. En *Write Back*, se escriben los resultados en los registros o en memoria. Además, el diagrama incluye componentes como DTLB y DMMU, la caché de datos, buses de comunicación y memorias temporales. Finalmente, el *Arbiter* coordina el acceso a recursos compartidos.

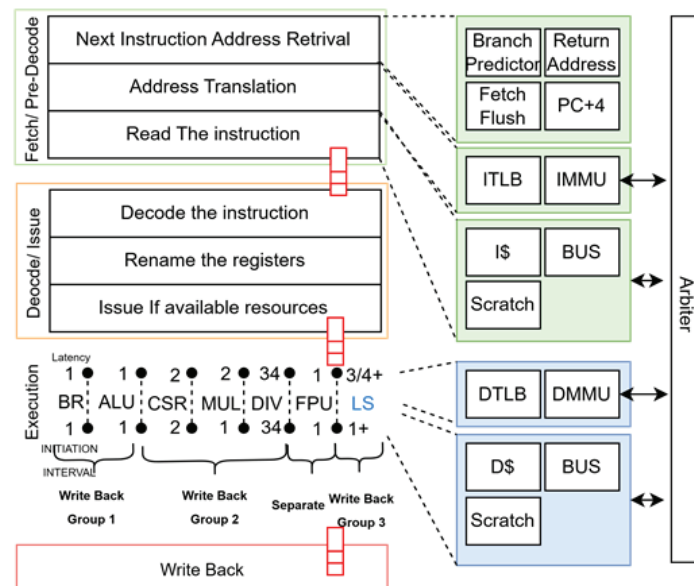


Figura 107. Pipeline 5 etapas CVA5[143]

6.3.1. Configuración de las herramientas y del bloque IP

De igual modo, se hace necesario preparar el entorno inicialmente con la instalación previa del *software* RISC-V GCC *toolchain*, Vivado y Verilator. Después de este paso se clona el repositorio de referencia [143].

En Vivado se ejecuta el script Tcl que permite crear la IP personalizada con el núcleo CVA5. A continuación, se crea un nuevo proyecto en el que se selecciona la placa de prototipado a utilizar, en este caso Nexys 4 DDR y se crea el archivo toplevel. Se añade el archivo de restricciones "constraints.xdc" para configurar los pines de la placa y se define el reloj de entrada a 100 MHz.

El bloque CVA5 contiene dos parámetros. Por una parte, el tamaño de la memoria de programa y por otra la ubicación del fichero de inicialización de dicha memoria (que contiene el programa a ejecutar) en formato MIF (Figura 108).

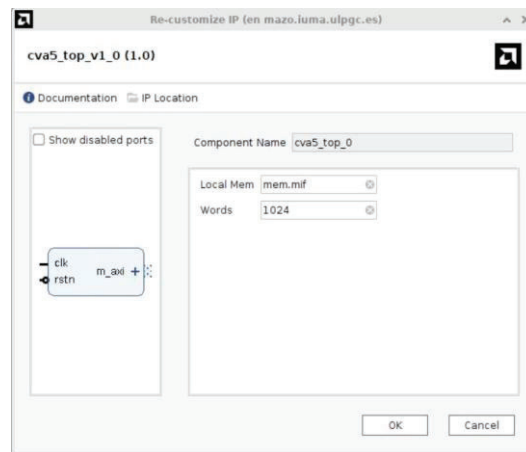


Figura 108. Configuración del bloque CVA5

6.3.2. Diseño de la plataforma

La Figura 109, muestra el diseño de bloques creado a partir del núcleo CVA5. Como se puede apreciar, el IP CVA5 ofrece una interfaz AXI4Lite maestra para comunicarse con el resto de la plataforma, además de la entrada del reloj y de la entrada de reset activa a nivel bajo. En la plataforma se ha incluido una UART conectada al bus AXI4.

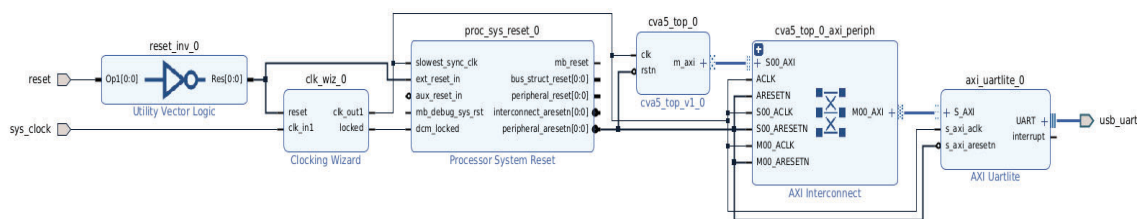


Figura 109. Diseño de bloques CVA5

6.3.3. Implementación y análisis de prestaciones

Se ha implementado la plataforma siguiendo los pasos del flujo de diseño de Vivado. En la Figura 110 (a) se muestra una visión completa de la implementación en el dispositivo de referencia (Xilinx Artix xc7a100tcs324-1). Como se puede apreciar, el impacto final en cuanto a la ocupación de recursos es baja, dejando espacio para añadir nuevas funcionalidades al sistema. En la Figura 110 (b) se presentan detalles de la implementación de la CPU, remarcando el fichero de registros, la ALU, la unidad *load/store* y la unidad de gestión de saltos (*branch unit*). En la Figura 111 se muestran los recursos utilizados, que, tal como se comentó anteriormente, son bastante reducidos. La utilización de bloques DSP se realiza para implementar las operaciones de multiplicación. Los bloques de memoria permiten implementar el programa de aplicación y otros recursos de almacenamiento local.

Con respecto a las prestaciones temporales, el diseño cumple con las restricciones de reloj, para un periodo de 10 ns, suficiente para la mayoría de las aplicaciones empotradas (Figura 112). La ruta crítica, en términos de *setup*, está en los registros de control y estado (Figura 113). La potencia disipada por la plataforma mínima es de 0,26 vatios (Figura 114).

6.3. Implementación de CVA5

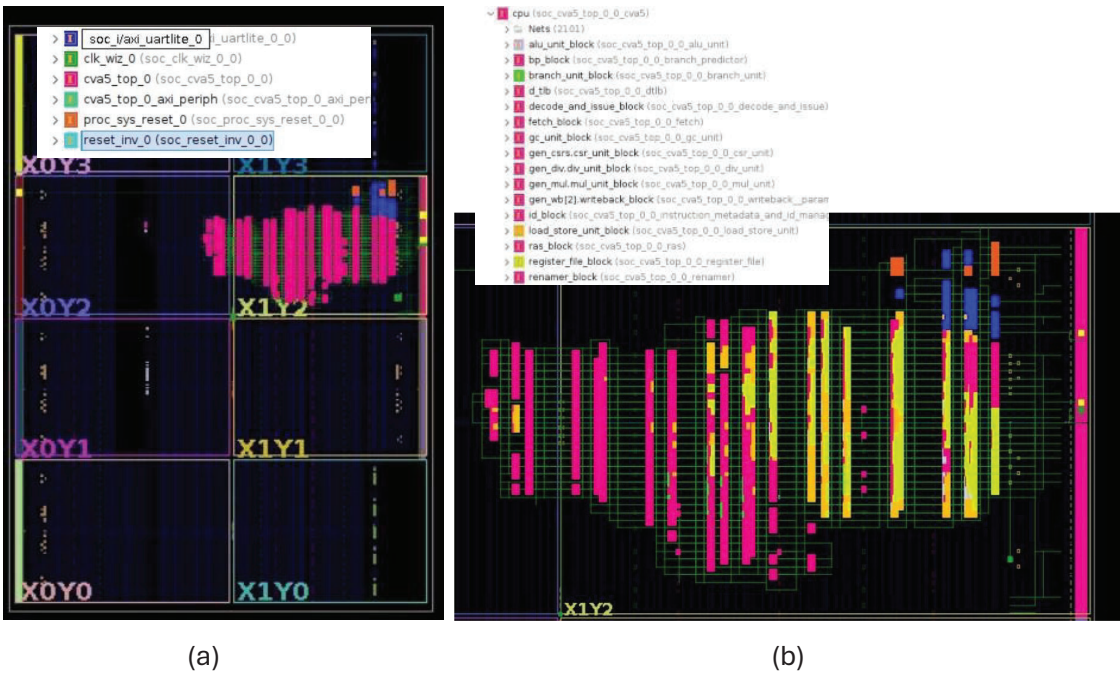


Figura 110. Layout del core CVA5 implementado

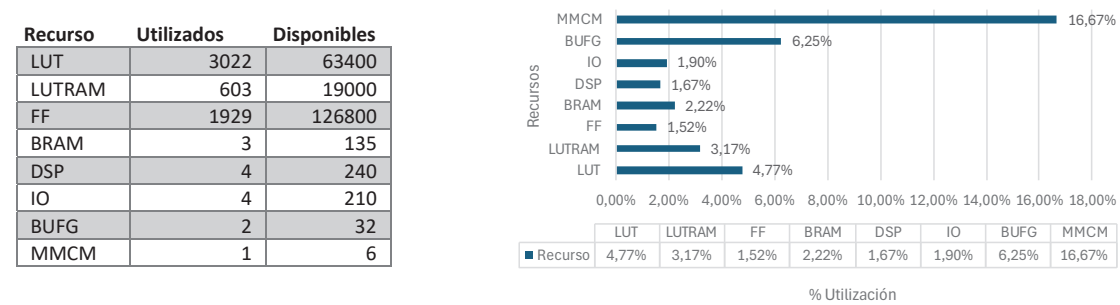


Figura 111. Utilización de recursos utilizados por la plataforma basada en CVA5

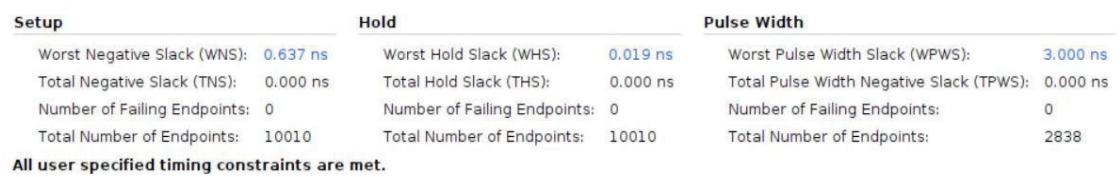


Figura 112. Requerimientos temporales del diseño (setup, hold y pulso)

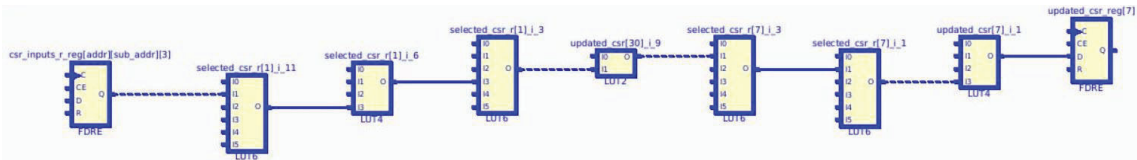


Figura 113. Ruta crítica del diseño (Control/Status Registers)

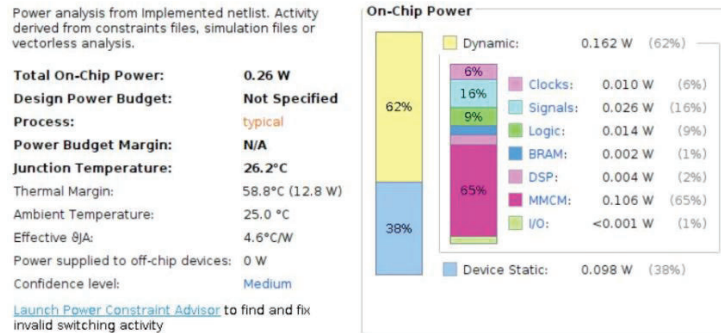


Figura 114. Análisis de potencia del diseño implementado

En la Figura 115 se muestra la implementación realizada para la memoria de programa (a) y su contenido inicializado en las primeras posiciones (b). El programa utilizado imprime el mensaje “Hello world” por la UART.

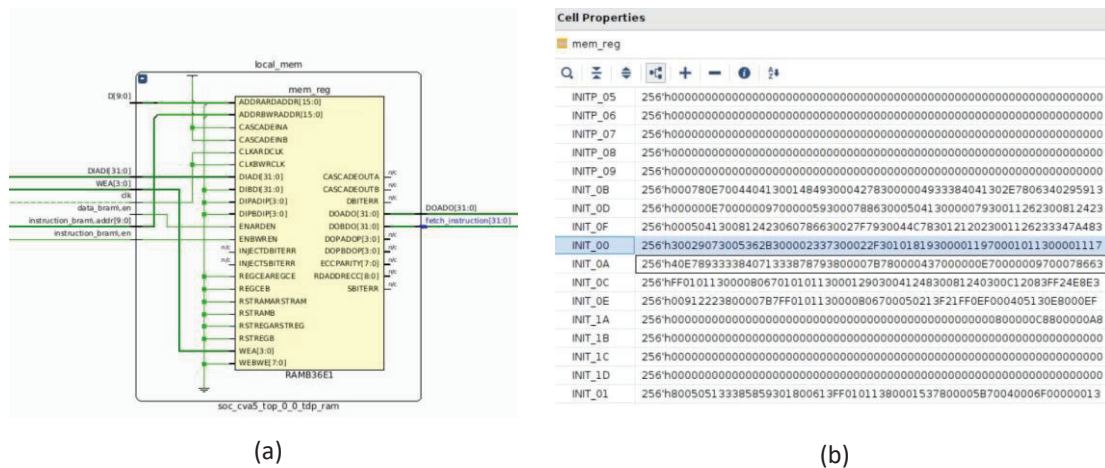


Figura 115. Memoria de program de CVA5

6.4. Implementación de CVA6

Como se ha indicado en el capítulo 5, CVA6 es un procesador RISC-V altamente configurable, forma parte de la familia de procesadores *CORE-V* del OpenHW Group. Implementa un pipeline en seis etapas: (1) Generación del PC y búsqueda de instrucciones, (2) decodificación de las instrucciones, (3) etapa de emisión, (4) etapa de ejecución y (5) fase de *commit*, según se puede ver en la Figura 58.

6.4.1. Implementación FPGA

La arquitectura CVA6 está integrada en un SoC que se puede implementar tanto en dispositivos FPGA y en tecnología ASIC. Se ha realizado su implementación FPGA para la placa de prototipado FPGA Genesys 2 [160]. Los resultados de las prestaciones obtenidos de su implementación FPGA se muestran en la Figura 116. El *layout* de la implementación se muestra en la Figura 117, en la que se ha diferenciado la implementación del CVA6 y del resto del SoC (MMU, buses del sistema, etc.).

6.4. Implementación de CVA6

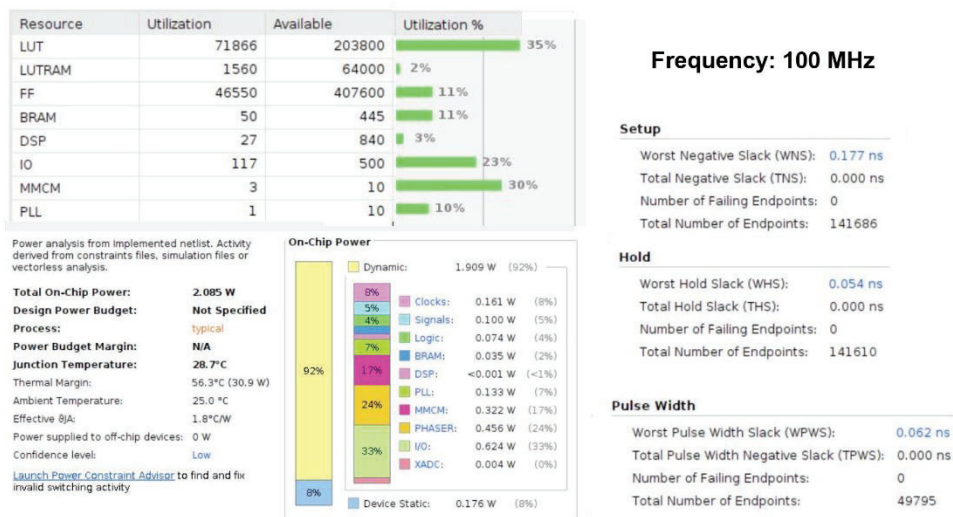


Figura 116. Prestaciones de la implementación de CVA6 para Genesys 2

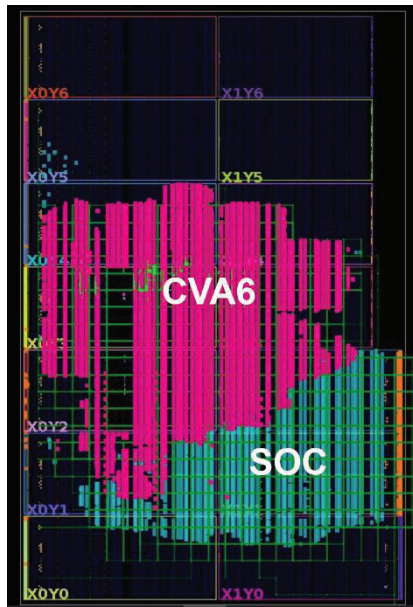


Figura 117. Layout de la implementación de CVA6 para la placa AMD Kintex Genesys2

Como se indicó, CVA6 soporta Linux, utilizando el proceso de arranque indicado, desde el *boot* en modo máquina hasta la ejecución de la aplicación de usuario, según se muestra en la Figura 118. En la Figura 119 se muestra la utilización de OpenSBI durante el arranque de la placa y la ejecución de aplicaciones de usuario en la Figura 120.

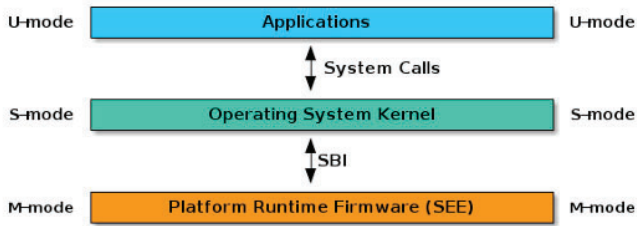


Figura 118. Modos de funcionamiento en CVA6 durante el proceso de arranque

```

copying boot image .... done!

OpenSBI v0.9

      /\
     /\
    /\
   /\
  /\
 /\
/\
_/_

Platform Name       : ARIANE RISC-V
Platform Features   : medeleg
Platform HART Count : 1
Platform IPI Device : aclint-mswi
Platform Timer Device : aclint-mtimer @ 1000000Hz
Platform Console Device : uart8250
Platform HSM Device : ---
Platform Reboot Device : ---
Platform Shutdown Device : ---
Firmware Base       : 0x80000000
Firmware Size       : 220 KB
Runtime SBI Version  : 0.3

Domain0 Name        : root
Domain0 Boot HART    : 0
Domain0 HARTs        : 0*
Domain0 Region00     : 0x0000000020080000-0x00000000200bffff (I)
  
```

Figura 119. OpenSBI arrancando en el microprocesador CVA6

```

# ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:18:3E:02:E3:7F
          BROADCAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          LOOPBACK MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

sit0      Link encap:UNSPEC HWaddr 00-00-00-00-00-00-A8-42-00-00-00-00-
          NOARP MTU:1480 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

```

# udhcpd
udhcpd: started, v1.33.1
[ 185.795852] Open device, request interrupt 11
[ 185.795852] Open device, request interrupt 11
udhcpd: sending discover
udhcpd: sending discover
[ 190.359881] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 190.359881] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpd: sending select for 10.13.24.43
udhcpd: lease of 10.13.24.43 obtained, lease time 10800
deleting routers
adding dns 193.145.138.100
adding dns 193.145.138.200
# ifconfig -a
eth0      Link encap:Ethernet HWaddr 00:18:3E:02:E3:7F
          inet addr:10.13.24.43 Bcast:10.13.24.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MTU:1500 Metric:1
          RX packets:227 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:88539 (86.4 KiB)  TX bytes:1434 (1.4 KiB)

```

Figura 120. Ejecución de aplicaciones para el arranque de DHCP en la interfaz Ethernet

6.4.2. Implementación ASIC

Para el caso de la implementación ASIC, se ha realizado la síntesis lógica en Synopsys Design Vision del diseño para la tecnología CMOS TSMC de 65 nm. En la Figura 121 se muestra la jerarquía principal de la implementación. La etapa de ejecución (*ex_stage_i*) que incluye las unidades de load/store Unit (*lsu_i*), ALU (*alu_i*), la unidad de punto flotante (*fpu_gen_fpu_i*) o la unidad de predicción de saltos (*branch_unit_i*). La etapa de issue (*issue_stage_i*) incluye la unidad de Scoreboard (*i_scoreboard*). Por último, se identifican las etapas de decodificación de la instrucción (*decoder_i*) y la etapa de frontend (*i_frontend*), que incluye entre otras unidades, la cola de instrucciones (*i_instr_queue*), el buffer de destino de saltos predichos (*btb_gen_i_btb*), el buffer de historia de saltos (*bht_gen_i_bht*). En la Figura 122 se muestra la interfaz de E/S del *core*. En la Figura 123 presenta un diagrama general del diseño sintetizado, incluyendo los bloques principales del diseño (primer nivel de jerarquía mostrada en la Figura 121).

En la Figura 124 se muestra el bloque que implementa los contadores de prestaciones de la arquitectura, incluyendo la información de la TLB, falta de información actualizada en las caches de instrucciones y datos, problemas de *stall* en la ruta de datos y otros indicadores de interés a la que la arquitectura puede acceder en modo privilegiado.

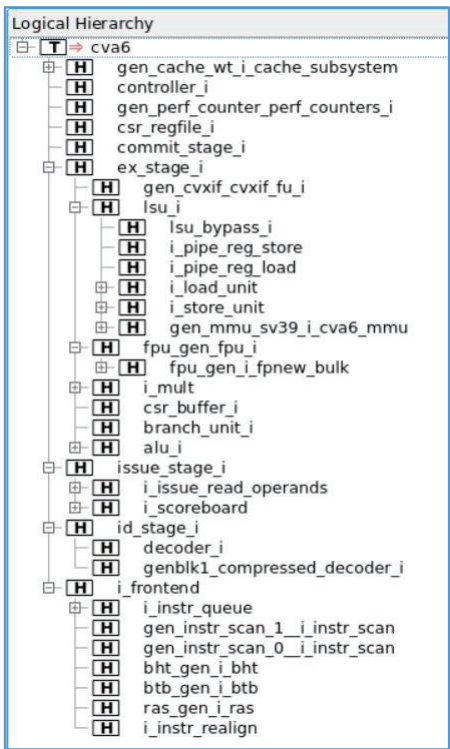


Figura 121. Jerarquía del diseño en Synopsys Design Vision

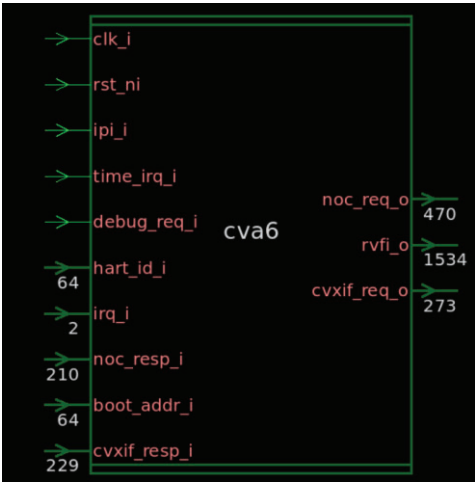


Figura 122. Interfaz de E/S de CVA6

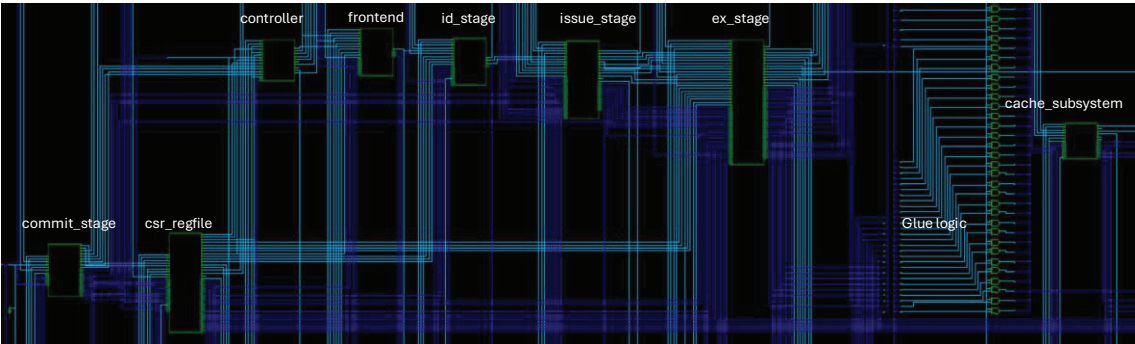


Figura 123. Bloques principales de CVA6 sintetizados en Design Compiler

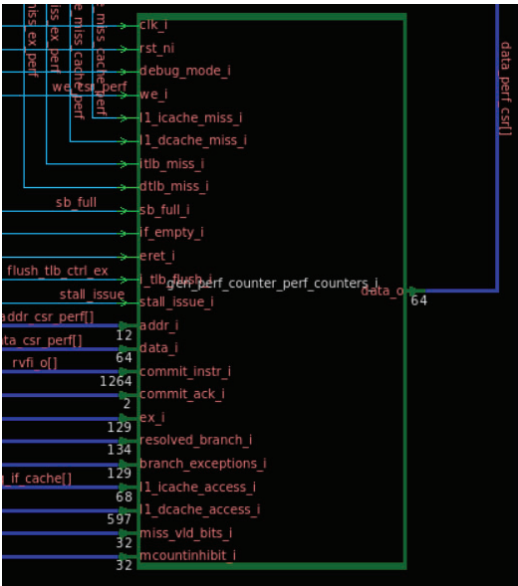


Figura 124. Bloque contadores de prestaciones de CVA6

En cuanto a las prestaciones del sistema, se ha sintetizado utilizando un reloj de 200 MHz, cumpliendo los requisitos de *slack*, según se muestra en la Figura 125. La ruta crítica del diseño se está ubicada en la actualización del contador de programa, una vez se ha accedido a la información del BHT (k) a partir del registro de dirección de la cache de instrucciones, tal como se muestra en la Figura 126.

El área estimada (antes de realizar el diseño físico) es de 616676,77 μm^2 , para un total de células estándar de 193.330. En la Figura 127 se muestra la distribución de células utilizadas (el diseño contiene un total de 347 bloques) y en la Figura 128 se muestra la distribución del área de cada uno de los tipos de celdas (lógica combinatorial, buffers, inversores y lógica secuencial).

La potencia estimada en la fase de síntesis lógica consumida por el circuito es de 51,6 mW para el procesador. Estos datos se verán incrementados por la generación de la red de distribución del reloj y el consumo de las interconexiones.

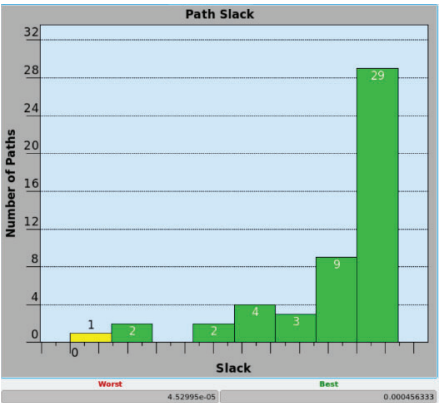


Figura 125. Histograma de Slack del diseño

6.4. Implementación de CVA6

Design : cva6		
Startpoint: i_frontend/icache_vaddr_q_reg_5_ (edge-triggered flip-flop clocked by main_clk)		
Endpoint: i_frontend/npc_q_reg_63_ (edge-triggered flip-flop clocked by main_clk)		
Path Group: main_clk (Frequency: 200 MHz)		
Path Type: max		
Point	Incr	Path

clock main_clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
i_frontend/icache_vaddr_q_reg_5_/CP (EDFCNQD1)	0.00	0.00 r
i_frontend/icache_vaddr_q_reg_5_/Q (EDFCNQD1)	0.18	0.18 f
...		
i_frontend/npc_q_reg_63_/D (DFCNQD1)	0.00	4.96 r
data arrival time		4.96
clock main_clk (rise edge)	5.00	5.00
clock network delay (ideal)	0.00	5.00
clock uncertainty	0.00	5.00
i_frontend/npc_q_reg_63_/CP	0.00	5.00 r
library setup time	-0.04	4.96
data required time		4.96

data required time		4.96
data arrival time		-4.96

slack (MET)		0.00

Figura 126. Ruta crítica (resumen)

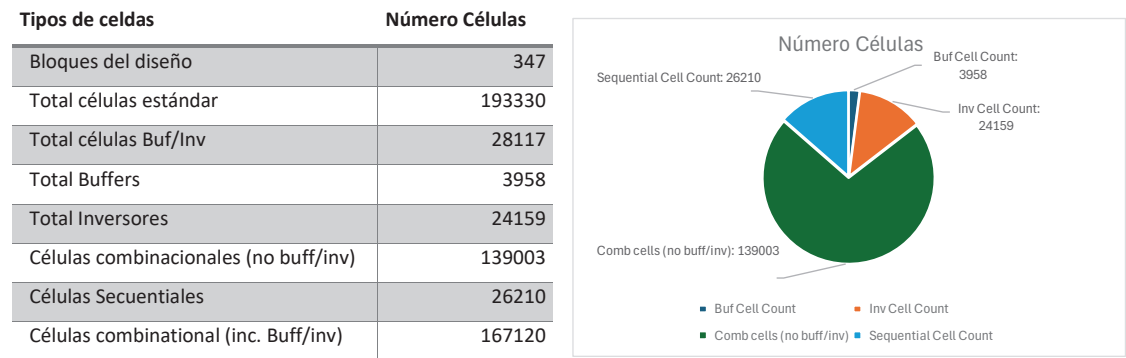


Figura 127. Distribución del tipo de células estándar utilizadas en el diseño

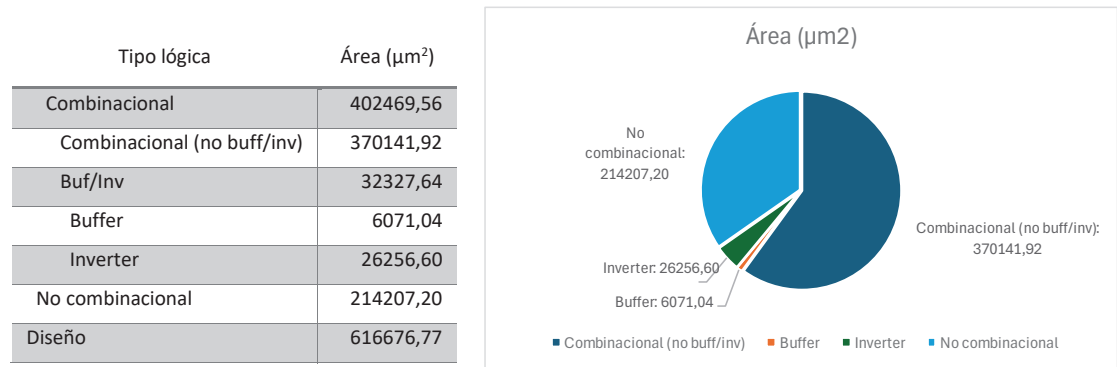


Figura 128. Distribución del área utilizada por las células estándar utilizadas en el diseño

Design : cva6

Startpoint: i_frontend/icache_vaddr_q_reg_5_
(edge-triggered flip-flop clocked by main_clk)

Endpoint: i_frontend/npc_q_reg_63_
(edge-triggered flip-flop clocked by main_clk)

Path Group: main_clk (Frequency: 200 MHz)

Path Type: max

Point	Incr	Path

clock main_clk (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
i_frontend/icache_vaddr_q_reg_5_/CP (EDFCNQD1)	0.00	0.00 r
i_frontend/icache_vaddr_q_reg_5_/Q (EDFCNQD1)	0.18	0.18 f
...		
i_frontend/npc_q_reg_63_/D (DFCNQD1)	0.00	4.96 r
data arrival time		4.96
clock main_clk (rise edge)	5.00	5.00
clock network delay (ideal)	0.00	5.00
clock uncertainty	0.00	5.00
i_frontend/npc_q_reg_63_/CP	0.00	5.00 r
library setup time	-0.04	4.96
data required time		4.96

data required time		4.96
data arrival time		-4.96

slack (MET)		0.00

Figura 129. Ruta crítica (resumen)

Frequency: 200 MHz

Global Operating Voltage = 1.2

Cell Internal Power = 50.8611 mW (99%)

Net Switching Power = 762.8559 uW (1%)

Total Dynamic Power = 51.6239 mW (100%)

Cell Leakage Power = 18.7646 uW

Power Group	Internal Power	Switching Power	Leakage Power	Total Power (%)

Total	50.8654 mW	0.7629 mW	1.8764e+04 nW	51.6470 mW

Figura 130. Distribución de la potencia estimada en la fase de síntesis

6.5. Análisis de los resultados

A continuación se resumen los *cores* implementados en términos de PPA (potencia, prestaciones y recursos) [161]. Estas variables se utilizan para decidir los criterios de optimización del diseño.

Generalmente el coste es función de las prestaciones y el área (o de los recursos en el caso de los dispositivos FPGA). Estos dos parámetros son los más importantes para el caso de circuitos sin limitación de potencia. Con la introducción de la electrónica para dispositivos móviles, y especialmente con las necesidades de computación para la Inteligencia Artificial, la potencia pasa a tomar un papel relevante en la medida de la

calidad del diseño final. La seguridad también está tomando un papel relevante en los sistemas actuales. En [162] se incorpora los aspectos de seguridad en un estudio relevante en el que compara distintas arquitecturas RISC-V.

RISC-V aporta ventajas significativas en estos indicadores. Por una parte, reduce los costes de implementación desde el diseño base, al adoptar el criterio de incluir en el diseño aquello que es relevante para la aplicación, evitando la presencia de silicio que no se va a utilizar. Ello afecta a los parámetros de potencia, área y, por tanto, a las prestaciones del sistema, evitando largas rutas combinacionales que impiden el cumplimiento de los criterios temporales.

6.5.1. Comparación de área/recursos

En la Tabla 15 se muestra la comparación del uso de recursos de los cuatro SoC implementados. Hay que tener en cuenta que en el caso del CVA6, se trata de un sistema en chip completo de 64 bits, frente a los anteriores que incluyen *cores* de 32 bits. No se han incluido en la comparación los datos de la implementación ASIC del CVA6 debido a la distinta granularidad de las células y tipos de implementación. La Figura 131 compara el uso de recursos entre los diferentes *cores*: (a) incluye todos los *cores*, (b) solamente *cores* de 32 bits y (c) incluye únicamente el uso de BRAM y DSPs para todos los *cores*.

Tabla 15. Comparación de resultados de implementación de distintos *cores*

	LUTs	FFs	BRAMs	DSP	Notas
NeoRV32	2.291	2.132	9.5	-	(1)
AIRISC	12.877	6.193	365	4	(1)
CVA5	3.625	1.929	3	5	(1)
CVA6	73.426	46.550	50	27	(2)
MicroBlaze V: Microcontroller Preset Conf.	2.236	1.138	0	4	(3)
MicroBlaze V: Real-time Preset Conf.	4.106	2.685	6	4	(3)
MicroBlaze V: Application Preset Conf.	8.623	5.325	16	6	(3)
MicroBlaze V: Maximum Frequency Conf.	1.227	826	0	0	(3)

Notas:

- (1) Implementado en dispositivo AMD Serie 7 Artix a 100 MHz
- (2) Implementado en dispositivo AMD Serie 7 Kintex a 100 MHz
- (3) Para dispositivos AMD Serie 7 Artix 7. Fuente: AMD

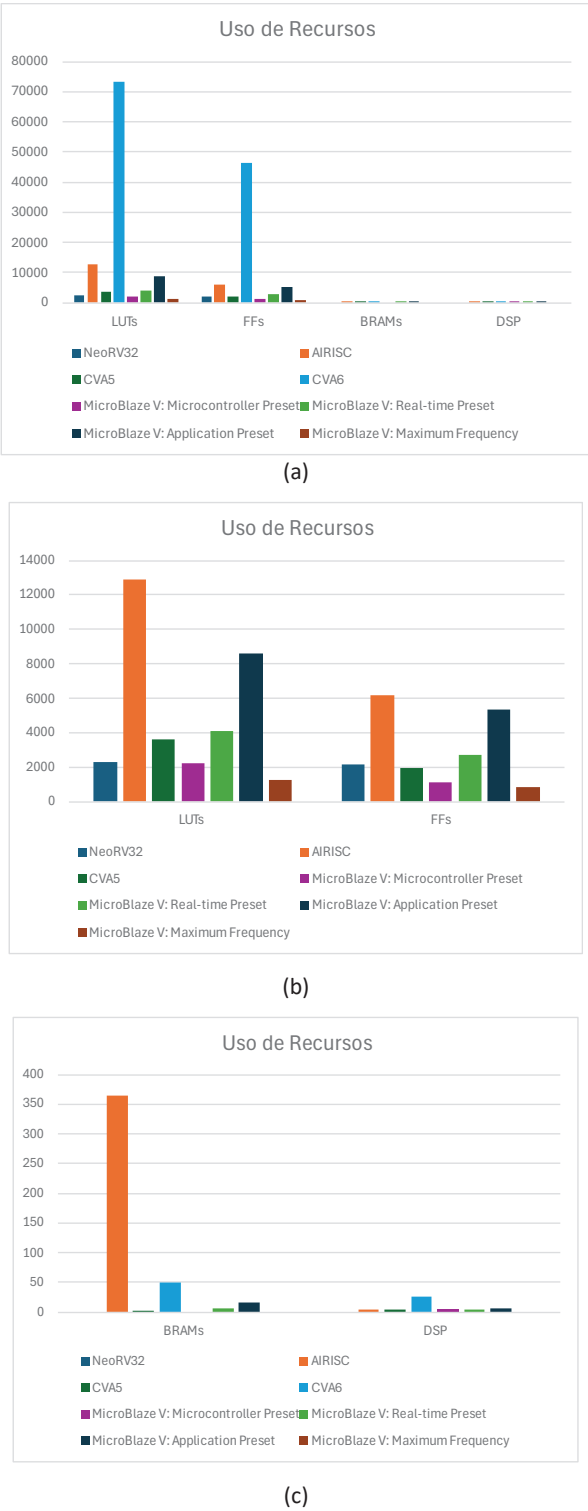


Figura 131. Utilización de recursos para distintos cores: (a) datos agregados, (b) detalle de LUTs y FFs y (c) detalles de BRAMs y DSPs.

6.5.2. Comparación de las prestaciones temporales

Los resultados del análisis temporal obtenidos mediante Vivado y trabajando a una frecuencia de 100 MHz, indica que el diseño cumple satisfactoriamente con los requisitos temporales en el análisis de *Setup*, *Hold* y *Pulse Width*.

Para el cálculo de la frecuencia máxima (FMAX) se utiliza el parámetro WNS y la frecuencia definida (F) o tu inverso, el periodo T, de tal forma que la frecuencia máxima se calcula mediante la expresión **$F_{MAX} = 1000 / (T - WNS)$** . En la Tabla 16 se muestran los cálculos realizados para los diseños presentados para la iteración de periodo de reloj de 10 ns.

Tabla 16. Frecuencias de funcionamiento de los cores RISC-V analizados

	Tecnología	Frecuencia (MHz)	Periodo (ns)	WNS (ns)	FMAX (MHz)
NEORV32	Artix	100	10	0,579	106,14
AIRISC	Artix	100	10	0,595	106,32
CVA5	Artix	100	10	0,637	106,80
CVA6	Kintex	100	10	0,177	101,80

6.5.3. Comparación de consumo de potencia

En el caso de NEORV32 el análisis de consumo de potencia muestra un consumo total de 0,252 W bajo condiciones de proceso. Además, en el gráfico de la Figura 86, la potencia dinámica representa un 61% del total, destacando el uso del bloque MMCM con 0,106 W y seguido del BRAM con 0,020 W. La potencia estática asciende a 0,090 W representado el 39% del total, lo que indica una contribución significativa del estado de reposo del dispositivo.

En cuanto a los resultados obtenidos en AIRISC se observa un consumo total de 0,324 W. La potencia dinámica representa el 48% del total (0,155 W) con una carga de MMCM que consumen el 79 % de la potencia dinámica. Otros componentes como buffer de reloj, señales, lógica, BRAM e interfaces representan una contribución menor. La potencia estática asciende a 0,169 W (52%).

En el núcleo CVA5 el análisis de potencia arroja un consumo total de 0,260 W. La potencia dinámica representa 0,162 W (62%) destacando el uso de MMCM y seguido de las señales con un gasto de 0,026 W. La potencia estática se mantiene en 0,098 W.

Finalmente, el núcleo CVA6 presenta un consumo de potencia superior, 2,085 W. La potencia dinámica representa 1,909 W (92%) y la potencia estática se mantiene en 0,176 W. Como se aprecia en la Figura 116, el consumo de los IOB representa el 33% del consumo total del sistema.

En la siguiente Tabla 17 se resumen los consumos de potencia estimados durante la fase de diseño para los distintos cores implementados. Se presentan datos de la potencia total, la potencia estática del dispositivo, la potencia dinámica que refleja el funcionamiento del diseño implementado y se añade el efecto producido por la inclusión de los bloques de memoria de tipo BRAM. Al igual que el caso de los recursos, la arquitectura CVA6 presenta un consumo de potencia significativo frente a las

implementaciones de 32 bits. Los *cores* de 32 bits están implementados utilizando una frecuencia objetivo de 100 MHz y en todos los casos están funcionando por debajo de 0,5 Watios. Para el caso de 64 bits, sigue teniendo un consumo limitado. La Figura 132 compara la potencia disipada por todos los *cores* y la Figura 133 solamente los *cores* de 32 bits.

Tabla 17. Potencia disipada por las implementaciones realizadas (en Watios)

	Potencia total	Potencia dinámica	Potencia estática	Potencia BRAM
NeoRV32	0,252	0,154	0,098	0,020
AIRISC	0,324	0,155	0,169	0,005
CVA5	0,260	0,162	0,098	0,002
CVA6	2,085	1,909	0,176	0,035

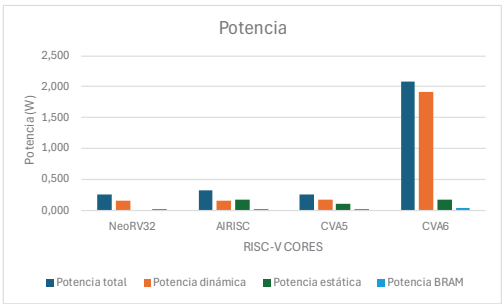


Figura 132. Potencia disipada (en Watios)

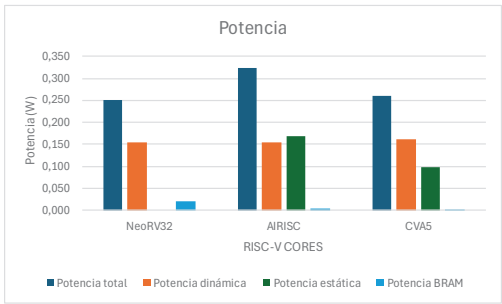


Figura 133. Potencia disipada de los cores de 32 bits

6.6. Conclusiones

En este capítulo se han presentado los resultados de los distintos experimentos realizados en este trabajo, para la implementación y pruebas de distintos núcleos de procesadores RISC-V de 32 y 64 bits sobre dispositivos FPGA. En los casos de procesadores de 32 bits, orientados a la ejecución de aplicaciones empujadas, la potencia disipada siempre se ha mantenido por debajo de 0,5 Watios, mientras que, en el caso de 64 bits, no ha alcanzado los 3 Watios, incluso en un procesador capaz de ejecutar un sistema operativo completos, como es el caso de Linux.

En cuanto a la utilización de recursos su impacto es mínimo, y solamente en el caso de AIRISC se ha optado por ocupar todos los bloques BRAM disponibles para disponer de memoria suficiente para evitar acceder al exterior. Esta estrategia no es la utilizada en el resto de los casos, cuando hay una jerarquía de memoria más compleja.

En todos los casos se ha alcanzado el funcionamiento a 100 MHz para dispositivos básicos de la Serie 7 de AMD (Artix/Kintex). Es posible alcanzar frecuencias de funcionamiento más altas, utilizando dispositivos más rápidos de AMD, tales como Virtex o incluso UltraScale+. Para muchas aplicaciones *baremetal*, la frecuencia objetivo definida en este trabajo es suficiente.

Capítulo 7. Conclusiones y trabajos futuros

Este capítulo final resumen las principales conclusiones obtenidas y las posibles líneas de trabajo que podrían ayudar a consolidar aún más el potencial de RISC-V en el ámbito académico y en la industria.

7.1. Conclusiones

El trabajo ha ido introduciendo de forma incremental las características más relevantes de RISC-V y la necesidad del uso de esta arquitectura. Se recogen las distintas arquitecturas ISA RISC-V disponibles, instrucciones base, nivel de privilegios, tipos de formato de instrucción y cómo se interpreta cada bit. A continuación, se ofrece una visión detallada del universo de extensiones de RISC-V, explicando desde las extensiones base y estándar hasta su uso en diferentes ámbitos.

El trabajo explora la metodología *software* asociada a RISC-V. Se abordarán los simuladores y emuladores que ofrecen evaluar y depurar programas sin necesidad de *hardware* físico, así como las herramientas de desarrollo, compiladores y entornos integrados que mejoran la experiencia del desarrollador. También se incluye el soporte de sistemas operativos, desde distribuciones Linux compatibles hasta sistemas en tiempo real para dispositivos integrados.

A lo largo del trabajo se estudian diferentes núcleos disponibles para la arquitectura RISC-V, así como distintas plataformas que soportan su integración.

En el trabajo se experimenta con algunos *cores* para analizar su facilidad de utilización, la capacidad de integración y las prestaciones obtenidas. Destaca la facilidad de trabajo con el *core* NEORV32, con la disponibilidad del código fuente tanto en VHDL como Verilog, su integración en distintos flujos de diseño y la calidad de la documentación. En concreto, se ha experimentado su integración en AMD Vivado, facilitando la creación de un IP configurable desde el entorno de diseño, la inclusión de nuevos bloques desde el propio catálogo de Vivado, facilitado por la compatibilidad de las interfaces de bus AMBA AXI4, así como la facilidad para incluir nuevas extensiones utilizando distintos mecanismos disponibles en el propio procesador. En cuanto al entorno *software* se ofrecen algunas facilidades para la interacción con periféricos estándar (UART, SPI, etc) y de soporte para RTOS (FreeRTOS y ZephyrOS). Sin embargo, no existe una amplia librería de componentes *software* como en otros núcleos más maduros que ralentizan su rápida adopción como *core* de referencia. A pesar de ello está recibiendo soporte industrial como es el caso de Nordic Semiconductor [163].

Si bien CVA5, no está tan desarrollado como NEORV32, su diseño orientado hace dispositivos FPGA y la baja utilización de recursos lo hace candidato a poder ser integrado en diseños sencillos.

El otro *core* de interés, en este caso para 64 bits es CVA6. Se trata de un *core* complejo, Linux *ready*, que está siendo utilizado como plataforma de desarrollo para aplicaciones tanto académicas como industriales. La utilización en distintos trabajos de grupo, tanto en este trabajo como en otros previos [164][165], ha demostrado su capacidad para utilizarlos como plataforma de referencia. En presentaciones recientes OpenHW Group ha mostrado su interés en mantener su desarrollo y en su la utilización en trabajos relacionados con proyectos europeos, como por ejemplo Tristan e Isolde [166].

7.2. Trabajos futuros

Los trabajos futuros que se pueden plantear en este ámbito son muy amplios al ser éste un campo de continua investigación debido al interés estratégico en diseño de chips.

Existe un conjunto de *cores* comerciales que deben ser evaluados para tener una visión más amplia del sector. Estos son los casos de empresas como Andes Technologies, Ventana Micro Systems o los *cores* de SemiDynamics.

Otros *cores* estudiados no han podido ser evaluados de forma experimental. Este es el caso de Sargantana, diseño que incluye un conjunto de recursos que están al nivel de CVA6. Si bien se ha hecho un *setup* inicial, se han producido actualizaciones recientes que no han sido evaluadas. Por tanto, uno de los primeros trabajos a abordar es el estudio detallado y experimentación de dicho *core* y su comparación con CVA6.

Con respecto a CVA6, es necesario analizar en profundidad algunos aspectos de su configuración *hardware*, en concreto el trabajo con las interfaces de integración de aceleradores *hardware* dedicados y su efecto en la generación de código. En este sentido, se profundizará en la experimentación con las plataformas Cheshire y X-HEEPS para evaluar la capacidad de integración de nuevos *cores*.

En cuanto a los procesadores de 32 bits, si bien ya hay un conocimiento bastante detallado de NEORV32, es necesario determinar el efecto de la integración de un RTOS desde el punto de vista de los requisitos *hardware*. Finalmente se ve necesario ampliar la experimentación realizada con respecto a CVA5. Para todos los casos, será necesario evaluar las prestaciones con aplicaciones reales y su comparación con *cores* de ARM, en su caso.

Referencias

- [1] ARM Ltd., «Architecting a Smarter World – Arm», *Arm.com*, 2019. [En línea]. Disponible: <https://www.arm.com/>. [Acceso: 19-abr-2020]
- [2] IBM, «IBM - España», *International Business Machines Corporation*, 2019. [En línea]. Disponible: <https://www.ibm.com/es-es>
- [3] Intel Corporation, «Intel: Soluciones del centro de datos, IoT e innovación en PC», *Intel.es*, 2018. [En línea]. Disponible: <https://www.intel.es/content/www/es/es/homepage.html>. [Acceso: 19-abr-2020]
- [4] A. Roell, S. Technical, y S. Professional, «OpenPOWER Foundation», 2014. .
- [5] K. Asanović y D. A. Patterson, «Instruction Sets Should Be Free: The Case For RISC-V», 2014.
- [6] European Processor Initiative (EPI), «EPI - European Processor Initiative», *European Processor Initiative*, 2020. [En línea]. Disponible: <https://www.european-processor-initiative.eu/project/epi/>. [Acceso: 19-abr-2020]
- [7] SPARC International Inc., «Documentos Técnicos», *SPARC International*, 2025. [En línea]. Disponible: <https://sparc.org/technical-documents/>. [Acceso: 07-jul-2025]
- [8] C. Price, «MIPS IV Instruction Set», 1995 [En línea]. Disponible: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf>
- [9] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer, y M. Linauer, *Open-Source RISC-V Processor IP Cores for FPGAs - Overview and Evaluation*. 2019.
- [10] OpenRISC Project, «OpenRISC - OpenRISC», *openrisc.io*, 2025. [En línea]. Disponible: <https://openrisc.io/>. [Acceso: 07-jul-2025]
- [11] Andes Technology Corporation, «Homepage - Andes Technology», *AndesTech.com*, 2025. [En línea]. Disponible: <https://www.andestech.com/en/>. [Acceso: 07-jul-2025]
- [12] SiFive Inc., «SiFive - Leading the RISC-V Revolution», *SiFive*, 2025. [En línea].

- Disponible: <https://www.sifive.com/>. [Acceso: 07-jul-2025]
- [13] Western Digital, «RISC-V», *Western Digital*, 2024. [En línea]. Disponible: <http://www.azothot.com/risc-v-8.html>. [Acceso: 19-may-2024]
 - [14] SiFive Inc., «HiFive1 Rev B», *SiFive Board*, 2024. [En línea]. Disponible: <https://www.sifive.com/boards/hifive1-rev-b>. [Acceso: 19-may-2024]
 - [15] PULP Platform, «PULP FAQs», *PULP Platform*, 2025. [En línea]. Disponible: <https://www.pulp-platform.org/>. [Acceso: 07-jul-2025]
 - [16] K. Asanovi Rimantas Avizienis Jonathan Bachrach Scott Beamer David Biancolini Christopher Celio Henry Cook Daniel Dabbelt John Hauser Adam Izraelevitz Sagar Karandikar Ben Keller Donggyu Kim John Koenig *et al.*, «The Rocket Chip Generator», 2016.
 - [17] A. Waterman y Asanovic Krste, «The RISC-V Instruction Set Architecture, Volume I: User-Level ISA», 2019 [En línea]. Disponible: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://csg.csail.mit.edu/6.375/6_375_2019_www/resources/riscv-spec.pdf
 - [18] RISC-V International, «RISC-V Foundation | Instruction Set Architecture (ISA)», *risc.org*, 2020. [En línea]. Disponible: <https://riscv.org/>. [Acceso: 19-abr-2020]
 - [19] C. Celio, D. Patterson, y K. Asanovic, «The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor», *EECS Department, University of California, Berkeley*, 2015. [En línea]. Disponible: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html>. [Acceso: 19-abr-2020]
 - [20] J. Miura, H. Miyazaki, y K. Kise, «A portable and Linux capable RISC-V computer system in Verilog HDL», *arXiv e-prints*, n.º xx, p. arXiv:2002.03576, feb. 2020 [En línea]. Disponible: <https://ui.adsabs.harvard.edu/abs/2020arXiv200203576M>
 - [21] Roa Logic, «RV12 RISC-V CPU Core», *GitHub*, 2020. [En línea]. Disponible: <https://roalogic.github.io/RV12/>. [Acceso: 19-abr-2020]
 - [22] SiFive Inc., «Freedom: Source files for SiFive's Freedom platforms», *GitHub*, 2020. [En línea]. Disponible: <https://github.com/sifive/freedom>. [Acceso: 20-may-2024]
 - [23] SiFive Inc., «SiFive GitHub Profile», *GitHub*, 2020. [En línea]. Disponible: <https://github.com/sifive/>. [Acceso: 20-may-2024]
 - [24] A. Krste Asanovic, Rimantas Avizienis, Jacob Bachmeyer, Christopher F. Batten *et al.*, «The RISC-V Instruction Set Manual v2.2», *2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings*, vol. I, pp. 1-32, 2017.
 - [25] RISC-V International, «¿Por qué RISC-V?», *RISC-V International*, 2020. .
 - [26] Amazon Web Services, «AWS presenta la compatibilidad con RISC-V en el kernel

- FreeRTOS», AWS, 2019. [En línea]. Disponible: <https://aws.amazon.com/es/about-aws/whats-new/2019/02/aws-announces-riscv-support-freertos-kernel/>. [Acceso: 19-may-2024]
- [27] D. Baños Expósito, «Google distribuirá las herramientas de Android para para Android en 2024», *Geektopia*, 2023. [En línea]. Disponible: <https://www.geektopia.es/es/technology/2023/11/01/noticias/google-distribuir-las-herramientas-de-android-para-para-android-en-2024.html>. [Acceso: 19-may-2024]
- [28] DigiTimes, «Beat Nvidia with RISC-V? Samsung expands its AI chip unit in Silicon Valley», *DigiTimes*, 2024. [En línea]. Disponible: <https://www.digitimes.com/news/a20240419PD212/samsung-ai-chip-nvidia-risc-v.html>. [Acceso: 19-may-2024]
- [29] DatacenterDynamics, «IBM ofrece en código abierto la arquitectura Power para reforzar las CPU de marcado», *DatacenterDynamics*, 2019. [En línea]. Disponible: <https://www.datacenterdynamics.com/es/noticias/ibm-ofrece-en-código-abierto-la-arquitectura-power-para-reforzar-las-cpu-de-marcado/>. [Acceso: 19-may-2024]
- [30] R. Garcia-Ramirez *et al.*, «Siwa: a RISC-V RV32I based Micro-Controller for Implantable Medical Applications», en *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, 2020, pp. 1-4 [En línea]. Disponible: <https://doi.org/10.1109/LASCAS45839.2020.9068952>
- [31] Rich Wawrzyniak, «RISC-V Market Report: Application Forecasts in a Heterogeneous World (abridged version)», n.º January, 2024.
- [32] E. Borin, «RISC-V Assembly Programming», 2021. .
- [33] RISC-V, «The RISC-V Instruction Set Manual Volume I: User-Level ISA V2.1», vol. I, 2016.
- [34] K. Cheng y J. Clarke, «RISC-V ABIs Specification, Document Version 1.0, Editors», nov. 2022 [En línea]. Disponible: <https://github.com/riscv-non-isa/riscv-elf-psabi-doc>. [Acceso: 25-ago-2025]
- [35] J. Manuel y M. Cuadros, «Processor architecture Introduction to computers II», 2022.
- [36] A. Waterman, Y. Lee, R. Avižienis, D. Patterson, K. Asanović, y J. Hauser, «The RISC-V Instruction Set Manual: Volume II. Privileged Architecture», vol. II, 2021.
- [37] The Linux Foundation, «Introduction to RISC-V», *edX*, 2024. [En línea]. Disponible: <https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD110x+1T2024/home>. [Acceso: 08-may-2024]
- [38] RVEmu Project, «Control and Status Registers - Writing a RISC-V Emulator in

- Rust», *RVEmu Book*, 2023. [En línea]. Disponible: <https://book.rvemu.app/hardware-components/03-csrs.html>. [Acceso: 25-ago-2025]
- [39] E. Cui, T. Li, y Q. Wei, «RISC-V Instruction Set Architecture Extensions: A Survey», *IEEE Access*, vol. 11, pp. 24696-24711, 2023.
- [40] R. Jones, «RISC-V extensions: what's available and how to find them», *Red Hat Research*, 2023. [En línea]. Disponible: <https://research.redhat.com/blog/article/risc-v-extensions-whats-available-and-how-to-find-it/>. [Acceso: 12-nov-2024]
- [41] RISC-V International, «RISC-V Ratified Specifications», *riscv.org*, 2024. [En línea]. Disponible: <https://riscv.org/technical/specifications/>. [Acceso: 18-nov-2024]
- [42] RISC-V International, «Ratified Extensions», *RISC-V Tech Hub*, 2024. [En línea]. Disponible: <https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154732/Ratified+Extensions>. [Acceso: 12-nov-2024]
- [43] X. Yu, Y. Sun, Y. Zhao, H. Kuang, y J. Han, «RVCE-FAL: A RISC-V Scalar-Vector Custom Extension for Faster FALCON Digital Signature», *Proceedings -Design, Automation and Test in Europe, DATE*, 2024.
- [44] Chisel Project/LF Projects LLC, «Chisel - Software-defined hardware», *chisel-lang.org*, 2025. [En línea]. Disponible: <https://www.chisel-lang.org/>. [Acceso: 25-ago-2025]
- [45] G. López-Paradís, A. Armejach, y M. Moretó, «Gem5 + rtl: A Framework to Enable RTL Models Inside a Full-System Simulator», *ACM International Conference Proceeding Series*, ago. 2021 [En línea]. Disponible: [/doi/pdf/10.1145/3472456.3472461?download=true](https://doi/pdf/10.1145/3472456.3472461?download=true). [Acceso: 25-ago-2025]
- [46] H. B. Amor, C. Bernier, y Z. Prikryl, «A RISC-V ISA Extension for Ultra-Low Power IoT Wireless Signal Processing», *IEEE Transactions on Computers*, vol. 71, n.º 4, pp. 766-778, abr. 2022.
- [47] P. P. Bernardo *et al.*, «A Scalable RISC-V Hardware Platform for Intelligent Sensor Processing», *Proceedings -Design, Automation and Test in Europe, DATE*, 2024.
- [48] K. K. Balasubramanian, M. Di Salvo, W. Rocchia, S. Decherchi, y M. Crepaldi, «Designing RISC-V Instruction Set Extensions for Artificial Neural Networks: An LLVM Compiler-Driven Perspective», *IEEE Access*, vol. 12, pp. 55925-55944, 2024.
- [49] M. S. Yu, C. Y. Yuan, T. L. Chen, y J. K. Lee, «Case Study: Optimization Methods With TVM Hybrid-OP on RISC-V Packed SIMD», *IEEE Access*, vol. 12, pp. 64193-64211, 2024.
- [50] S. Lee, J. Choi, S. Noh, J. Koo, y J. Kung, «DBPS: Dynamic Block Size and Precision

- Scaling for Efficient DNN Training Supported by RISC-V ISA Extensions», *Proceedings - Design Automation Conference*, vol. 2023-July, 2023.
- [51] X. Zou, Y. Peng, T. Li, L. Kong, Z. Pang, y L. Zhang, «Accelerating Blake3 in RISC-V», *2023 2nd International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT)*, pp. 296-301, ago. 2023.
- [52] Y. Zhao *et al.*, «Enhancing RISC-V Vector Extension for Efficient Application of Post-Quantum Cryptography», *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, vol. 2023-July, pp. 10-17, 2023.
- [53] E. N. Isman, C. Topal, L. Akcay, y B. Ors, «Instruction Extension of an Open Source RV32IMC Core for NTRU Cryptosystem», *ECCTD 2020 - 24th IEEE European Conference on Circuit Theory and Design*, sep. 2020.
- [54] H. Li, N. Mentens, y S. Picek, «Maximizing the Potential of Custom RISC-V Vector Extensions for Speeding up SHA-3 Hash Functions», *Proceedings -Design, Automation and Test in Europe, DATE*, vol. 2023-April, 2023.
- [55] K. Shi, S. Xu, Y. Diao, D. Boland, y Y. Bao, «ENCORE: Efficient Architecture Verification Framework with FPGA Acceleration», *FPGA 2023 - Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 209-219, feb. 2023 [En línea]. Disponible: <https://dl.acm.org/doi/10.1145/3543622.3573187>. [Acceso: 12-nov-2024]
- [56] D. Malathi, R. Sneha, M. Shanmugapriya, y S. Sethurajan, «Design of Risc-V Processing Unit Using Posit Number System», *ICSPP 2023 - 4th International Conference on Signal Processing and Communication*, pp. 427-431, 2023.
- [57] J. Huang, Q. Dou, y L. Shen, «Extending the RISC-V Instruction Set for High Performance Data Compression Hardware Acceleration», *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pp. 131-132, 2024.
- [58] Q. Ducasse, P. Cotret, y L. Lagadec, «JIT Compiler Security through Low-Cost RISC-V Extension», *2023 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2023*, pp. 125-128, 2023.
- [59] B. Marshall, D. Page, y T. H. Pham, «A lightweight ISE for ChaCha on RISC-V», *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, vol. 2021-text, pp. 25-32, jul. 2021.
- [60] A. Castellanos, «Jupiter: RISC-V Assembler and Runtime», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/andrescv/jupiter>. [Acceso: 07-jul-2025]
- [61] A. Castellanos, «Jupiter: RISC-V Assembler and Runtime Simulator», *GitHub*, 2023. [En línea]. Disponible: <https://github.com/andrescv/jupiter>. [Acceso: 29-ago-2025]

- [62] K. Vakil, «Venus: RISC-V Instruction Set Simulator Built for Education», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/kvakil/venus>
- [63] hm-riscv, «vscode-riscv-venus: VS Code extension with the Venus RISC-V simulator», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/hm-riscv/vscode-riscv-venus>
- [64] B. Landers, «RARS: RISC-V Assembler and Runtime Simulator», *GitHub*, 2023. [En línea]. Disponible: <https://github.com/TheThirdOne/rars>. [Acceso: 20-may-2024]
- [65] RISC-V Software, «Spike: RISC-V ISA Simulator», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/riscv-software-src/riscv-isa-sim>. [Acceso: 20-may-2024]
- [66] RISC-V Software, «Spike: a RISC-V ISA Simulator», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/riscv-software-src/riscv-isa-sim>. [Acceso: 27-ago-2025]
- [67] Tom Preston-Werner, «Semantic Versioning 2.0.0 | Semantic Versioning», *SemVer.org*, 2013. .
- [68] M. B. Petersen, «Ripes: A Visual Computer Architecture Simulator», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/mortbopet/Ripes>. [Acceso: 20-may-2024]
- [69] QEMU Project, «QEMU / QEMU · GitLab», *GitLab Repository*, 2025. [En línea]. Disponible: <https://gitlab.com/qemu-project/qemu>. [Acceso: 07-jul-2025]
- [70] GNU Project, «GCC, the GNU Compiler Collection», *GNU.org*, 2025. [En línea]. Disponible: <https://www.gnu.org/software/gcc/>. [Acceso: 07-jul-2025]
- [71] RISC-V Collaboration Group, «GNU toolchain for RISC-V, including GCC», *GitHub Repository*, 2025. [En línea]. Disponible: <https://github.com/riscv-collab/riscv-gnu-toolchain>. [Acceso: 07-jul-2025]
- [72] LLVM Project, «The LLVM Project: Modular Compiler and Toolchain Technologies», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/llvm/llvm-project>. [Acceso: 20-may-2024]
- [73] xPack Dev Tools, «The xPack GNU RISC-V Embedded GCC», *xPack Build Framework*, 2025. [En línea]. Disponible: <https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack>. [Acceso: 07-jul-2025]
- [74] ILG's archive projects, «GNU MCU Eclipse RISC-V Embedded GCC», *GitHub*, 2019. [En línea]. Disponible: <https://github.com/ilg-archived/riscv-none-gcc>. [Acceso: 20-may-2024]
- [75] SiFive Inc., «SiFive Software - Accelerate & Enhance your RISC-V Development», *SiFive*, 2025. [En línea]. Disponible: <https://www.sifive.com/software>. [Acceso: 07-jul-2025]

- [76] The Linux Foundation, «Decentralized innovation, built with trust», *LinuxFoundation.org*, 2025. [En línea]. Disponible: <https://www.linuxfoundation.org/>. [Acceso: 07-jul-2025]
- [77] The Linux Foundation, «Rise: RISC-V Software Ecosystem», *RiseProject.dev*, 2025. [En línea]. Disponible: <https://riseproject.dev/>. [Acceso: 27-ago-2025]
- [78] Fedora Project, «Architectures/RISC-V - Fedora Project Wiki», *Fedora Wiki*, 2025. [En línea]. Disponible: <https://fedoraproject.org/wiki/Architectures/RISC-V>. [Acceso: 07-jul-2025]
- [79] Debian Project, «RISC-V - Debian Wiki», *Debian Wiki*, 2025. [En línea]. Disponible: <https://wiki.debian.org/RISC-V>. [Acceso: 07-jul-2025]
- [80] OpenMandrivaProject, «Index of /cooker/repository/riscv64/», *rpmfind.net*, 2024. [En línea]. Disponible: <https://rpmfind.net/linux/RPM/openmandriva/cooker/riscv64/index.html>. [Acceso: 07-jul-2025]
- [81] openSUSE Project, «openSUSE:Factory:RISCV - RISC-V port of openSUSE Factory», *openSUSE Build Service*, 2025. [En línea]. Disponible: <https://build.opensuse.org/project/show/openSUSE:Factory:RISCV>. [Acceso: 07-jul-2025]
- [82] R. Dolbeau, «riscv-crossdev: A RISC-V Port of Gentoo's Crossdev», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/riscvarchive/riscv-crossdev>. [Acceso: 29-may-2024]
- [83] Parabola Project, «AbsLibre: Parabola package recipes (Arch Build System Libre)», *Parabola GNU/Linux-Libre*, 2025. [En línea]. Disponible: <https://git.parabola.nu/abslibre/abslibre.git/>. [Acceso: 07-jul-2025]
- [84] Ataraxia Linux Developers, «Ataraxia GNU/Linux», *GitHub Repository*, 2025. [En línea]. Disponible: <https://github.com/ataraxialinux/ataraxia>. [Acceso: 07-jul-2025]
- [85] Á. Figueroa, «RISCV port of the Slackware distribution», *GitHub Repository*, 2025. [En línea]. Disponible: https://github.com/fede2cr/slackware_riscv. [Acceso: 07-jul-2025]
- [86] Ubuntu Wiki Contributors, «RISC-V - Ubuntu Wiki», *Ubuntu Wiki*, 2025. [En línea]. Disponible: [https://wiki.ubuntu.com/RISC-V/Pine64 Star64](https://wiki.ubuntu.com/RISC-V/Pine64%20Star64). [Acceso: 07-jul-2025]
- [87] FreeRTOS Project, «FreeRTOS Real Time Kernel (RTOS)», *SourceForge.net*, 2024. [En línea]. Disponible: <https://sourceforge.net/projects/freertos/>. [Acceso: 07-jul-2025]
- [88] FreeRTOS Developers, «FreeRTOS for RISC-V RV32 and RV64», *FreeRTOS.org*, 2025. [En línea]. Disponible: <https://www.freertos.org/Using-FreeRTOS-on-RISC->

- V. [Acceso: 07-jul-2025]
- [89] The Linux Foundation, «Zephyr: Primary Git Repository for the Zephyr Project», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/zephyrproject-rtos/zephyr>
- [90] S. Kalapothas, M. Galetakis, G. Flamis, F. Plessas, y P. Kitsos, «A Survey on RISC-V-Based Machine Learning Ecosystem», *Information 2023, Vol. 14, Page 64*, vol. 14, n.º 2, p. 64, ene. 2023 [En línea]. Disponible: <https://www.mdpi.com/2078-2489/14/2/64/htm>. [Acceso: 08-may-2024]
- [91] SpinalHDL, «VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/SpinalHDL/VexRiscv>. [Acceso: 19-may-2024]
- [92] SpinalHDL, «SpinalHDL: Scala-based Hardware Description Language», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/SpinalHDL/SpinalHDL>. [Acceso: 19-may-2024]
- [93] R. Dolbeau, «VexRiscvBPluginGenerator: Plugin generatos for RISC-V B/K/P extensions», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/rdolbeau/VexRiscvBPluginGenerator/>. [Acceso: 19-may-2024]
- [94] J. Bachrach *et al.*, «Chisel: Constructing hardware in a Scala embedded language», *Proceedings - Design Automation Conference*, pp. 1216-1225, 2012 [En línea]. Disponible: <https://dl.acm.org/doi/10.1145/2228360.2228584>. [Acceso: 19-may-2024]
- [95] C. Celio, «RISC-V BOOM - RISC-V BOOM», *boom-core.org*, 2025. [En línea]. Disponible: <https://boom-core.org/>. [Acceso: 19-may-2024]
- [96] Berkeley Architecture Research Group, «BOOM: The Berkeley Out-of-Order RISC-V Processor», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/riscv-boom/riscv-boom>. [Acceso: 19-may-2024]
- [97] Yosys Headquarters, «PicoRV32: A Size-Optimized RISC-V CPU», *GitHub Repository*, 2024. [En línea]. Disponible: <https://github.com/YosysHQ/picorv32>. [Acceso: 19-may-2024]
- [98] D. Todd, «Tightly Coupling the PicoRV32 RISC-V Processor with Custom Logic Accelerators via a Generic Interface», *All Theses*, may 2021 [En línea]. Disponible: https://open.clemson.edu/all_theses/3552. [Acceso: 27-ago-2025]
- [99] S. Nolting, «NEORV32: Customizable MCU-Class 32-bit RISC-V Soft-Core CPU and SoC», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/stnolting/neorv32>. [Acceso: 08-may-2024]
- [100] The Linux Foundation, «A proven RTOS ecosystem, by developers, for

- developers», *ZephyrProject.org*, 2024. [En línea]. Disponible: <https://www.zephyrproject.org/>. [Acceso: 19-may-2024]
- [101] FreeRTOS Project, «FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions», *FreeRTOS.org*, 2024. [En línea]. Disponible: <https://www.freertos.org/index.html>. [Acceso: 19-may-2024]
- [102] S. Nolting, «Adding Custom Hardware Modules - NEORV32 User Guide», *GitHub Repository*, 2025. [En línea]. Disponible: https://github.com/stnolting/neorv32/blob/main/docs/userguide/adding_custom_hw_modules.adoc. [Acceso: 29-ago-2025]
- [103] SpinalHDL, «NaxRiscv: Superscalar Out-of-Order RISC-V CPU», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/SpinalHDL/NaxRiscv>. [Acceso: 19-may-2024]
- [104] F. Kermarrec, S. Bourdeauducq, J.-C. Le Lann, y H. Badier, «LiteX: an open-source SoC builder and library based on Migen Python DSL», n.º March, 2020 [En línea]. Disponible: <http://arxiv.org/abs/2005.02506>
- [105] FrontGrade Gailer, «NOEL-V RISC-V Processor», *Gaisler.com*, 2024. [En línea]. Disponible: <https://www.gaisler.com/index.php/products/processors/noel-v>. [Acceso: 19-may-2024]
- [106] N.-J. Wessman *et al.*, «De-RISC: A Complete RISC-V Based Space-Grade Platform», en *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 802-807.
- [107] K. Mohajerani, «ORCA RISC-V Core», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/kammoh/ORCA-risc-v>. [Acceso: 19-may-2024]
- [108] Microchip Technology Inc., «VectorBlox™ Accelerator SDK», *Microchip Technology*, 2025. [En línea]. Disponible: <https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/vectorblox>. [Acceso: 10-jun-2024]
- [109] O. Kindgren, «SERV: The SErial RISC-V CPU», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/olofk/serv>. [Acceso: 19-may-2024]
- [110] olofk, «Servant: FPGA Reference platform», *SERV documentation*, 2024. [En línea]. Disponible: <https://serv.readthedocs.io/en/latest/servant.html>. [Acceso: 19-may-2024]
- [111] olofk, «CoreScore», *CoreScore.store*, 2024. [En línea]. Disponible: <https://corescore.store/>. [Acceso: 19-may-2024]
- [112] O. Kindgren, «Observer: Configurable Sensor Aggregation Platform Using SERV RISC-V CPUs», *GitHub*, 2024. [En línea]. Disponible:

- <https://github.com/olofk/observer>. [Acceso: 19-may-2024]
- [113] O. Kindgren, «Subservient: Small SERV-based SoC Primarily for OpenMPW Tapeout», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/olofk/subservient/>. [Acceso: 19-may-2024]
- [114] P. Campbell, «VRoom! RISC-V CPU», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/MoonbaseOtago/vroom>. [Acceso: 19-may-2024]
- [115] lowRISC, «Ibex: 32-bit RISC-V Core for Embedded SoC Integration», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/lowRISC/ibex>. [Acceso: 09-may-2024]
- [116] P. D. Schiavone *et al.*, «Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for internet-of-things applications», *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS 2017*, vol. 2017-Janua, pp. 1-8, nov. 2017.
- [117] D. Rossi *et al.*, «PULP: A parallel ultra low power platform for next generation IoT applications», *2015 IEEE Hot Chips 27 Symposium, HCS 2015*, may 2015.
- [118] lowRISC C.I.C., «lowRISC: Collaborative open silicon engineering», *lowRISC.org*, 2024. [En línea]. Disponible: <https://lowrisc.org/>. [Acceso: 19-may-2024]
- [119] OpenHW Group, «CV32E20: An embedded 32-bit RISC-V CPU core — CV32E20 Documentation 0.1 documentation», *CV32E20 Documentation*, 2025. [En línea]. Disponible: <https://docs.openhwgroup.org/projects/cve2-user-manual/en/latest/>. [Acceso: 27-jul-2025]
- [120] OpenHW Group, «CORE-V CVE2: Small 32-bit RISC-V CPU core (RV32IMC/EMC)», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/openhwgroup/cve2>. [Acceso: 27-jul-2025]
- [121] OpenHW Group, «OpenHW Group GitHub Profile», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/openhwgroup>. [Acceso: 24-jul-2025]
- [122] tom01h, «zero-riscy CPU Core», *GitHub Repository*, 2025. [En línea]. Disponible: <https://github.com/tom01h/zero-riscy>. [Acceso: 28-jul-2025]
- [123] F. Zaruba y L. Benini, «The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology», *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, n.º 11, pp. 2629-2640, nov. 2019.
- [124] OpenHW Group, «CVA6: An application class RISC-V CPU core», *OpenHW Group Documentation*, 2024. [En línea]. Disponible: <https://docs.openhwgroup.org/projects/cva6-user-manual/>. [Acceso: 13-may-2024]
- [125] OpenHW Group, «CVA6: A 64-bit RISC-V CPU Core», *GitHub Repository*, 2024. [En línea]. Disponible: <https://github.com/openhwgroup/cva6>. [Acceso: 13-may-2024]

2024]

- [126] Barcelona Supercomputing Center, «Sargantana, la tercera generación de procesadores Lagarto», *Proyecto DRAC*, 2022. [En línea]. Disponible: <https://drac.bsc.es/es/sargantana2>. [Acceso: 13-may-2024]
- [127] V. Soria-Pardos *et al.*, «Sargantana: A 1 GHz+ In-Order RISC-V Processor with SIMD Vector Extensions in 22nm FD-SOI», *Proceedings - 2022 25th Euromicro Conference on Digital System Design, DSD 2022*, pp. 254-261, 2022.
- [128] S. Inc., «RISC-V Core IP Portfolio - SiFive», *SiFive*, 2024. [En línea]. Disponible: <https://www.sifive.com/risc-v-core-ip>. [Acceso: 19-may-2024]
- [129] SiFive Inc., «SiFive E31 Core Complex Manual», 2017.
- [130] SiFive, «SiFive E51 Core Complex Series Manual V1p2», 2017.
- [131] SiFive Inc., «SiFive Portfolio», 2024 [En línea]. Disponible: <https://www.sifive.com>. [Acceso: 20-jun-2025]
- [132] SiFive Inc., «SiFive Intelligence X280 as AI Compute Host: Google Datacenter Case Study», *SiFive Blog*, 2022. [En línea]. Disponible: <https://www.sifive.com/blog/sifive-intelligence-x280-as-ai-compute-host-google>. [Acceso: 13-may-2024]
- [133] C. Chen *et al.*, «Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension: Industrial Product», *Proceedings - International Symposium on Computer Architecture*, vol. 2020-May, pp. 52-64, may 2020.
- [134] T-Head Semiconductor (Alibaba Group), «XuanTie Product List», *XRVM.cn*, 2024. [En línea]. Disponible: <https://www.xrvm.cn/product/list/xuantie?spm=a2cl5.14290816.0.0.998ff6tCf6tC9S>. [Acceso: 13-may-2024]
- [135] L. T-Head Semiconductor Co., «OpenXuantie - OpenC910 Core», *GitHub*, 2024. [En línea]. Disponible: <https://github.com/T-head-Semi/openc910>. [Acceso: 20-may-2024]
- [136] GigaDevice Semiconductor Inc., «RISC-V Microcontrollers», *GigaDevice.com*, 2024. [En línea]. Disponible: <https://www.gigadevice.com/product/mcu/risc-v>. [Acceso: 20-may-2024]
- [137] I. Set y A. Manual, «Bumblebee Processor Core», 2019.
- [138] N. S. Technology, «N200 Series 32-Bit risc-v processor_Nuclei-Best RISC-V Processor IP», *Nuclei Product Page*, 2025. [En línea]. Disponible: <https://www.nucleisys.com/product/200.php>. [Acceso: 20-may-2024]
- [139] Synopsys, «Synopsys Processor Designer», *Technology*, pp. 1-16, 2009 [En línea].

- Disponible:
<http://www.synopsys.com/Tools/SLD/ProcessorDev/Pages/default.aspx%5Chttp://www.coware.com/PDF/products/ProcessorDesigner.pdf>
- [140] Synopsys Inc., «Synopsys ARC-V RPX-100 Processor IP», *Synopsys*, 2025. [En línea]. Disponible: <https://www.synopsys.com/dw/ipdir.php?ds=arc-v-rpx-100>. [Acceso: 24-jul-2025]
- [141] Fraunhofer IPMS, «Welcome - Fraunhofer IPMS - Fraunhofer IPMS», *Fraunhofer IPMS*, 2024. [En línea]. Disponible: <https://www.ipms.fraunhofer.de/en.html>. [Acceso: 20-may-2024]
- [142] Fraunhofer IMS, «AIRISC Core Complex», *GitHub*, 2024. [En línea]. Disponible: https://github.com/Fraunhofer-IMS/airisc_core_complex. [Acceso: 13-may-2024]
- [143] OpenHW Group, «CVA5: Application-Class 5-Stage RISC-V CPU», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/openhwgroup/cva5>. [Acceso: 28-jul-2025]
- [144] SFU-RCL (Simon Fraser University - Reconfigurable Computing Lab), «Taiga Project», *GitLab*, 2020. [En línea]. Disponible: <https://gitlab.com/sfu-rcl/taiga-project>. [Acceso: 28-jul-2025]
- [145] AMD Inc., *MicroBlaze V Processor Reference Guide (UG1629)*. AMD Documentation Portal, 2025 [En línea]. Disponible: <https://docs.amd.com/r/en-US/ug1629-microblaze-v-user-guide/MicroBlaze-V-Architecture>. [Acceso: 28-ago-2025]
- [146] E.-X. Team, «X-HEEP — X-HEEP 1.0 documentation», *ReadTheDocs*, 2025. [En línea]. Disponible: <https://x-heap.readthedocs.io/en/latest/index.html>. [Acceso: 24-jul-2025]
- [147] Embedded Systems Laboratory (ESL) with EPFL EcoCloud Center, «X-HEEP Web Server», *EPFL - X-HEEP Project*, 2025. [En línea]. Disponible: <https://x-heap.epfl.ch/>. [Acceso: 24-jul-2025]
- [148] Embedded Systems Lab (ESL) - EPFL, «X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/esl-epfl/x-heap?tab=readme-ov-file>. [Acceso: 24-jul-2025]
- [149] PULP Platform, «PULP Platform: Open-source RISC-V processors and SoCs», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/pulp-platform>. [Acceso: 24-jul-2025]
- [150] lowRISC C.I.C., «lowRISC - GitHub Profile», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/lowRISC>. [Acceso: 24-jul-2025]

- [151] A. Ottaviano, T. Benz, P. Scheffler, y L. Benini, «Cheshire: A Lightweight, Linux-Capable RISC-V Host Platform for Domain-Specific Accelerator Plug-In», *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, n.º 10, pp. 3777-3781, may 2023 [En línea]. Disponible: <https://arxiv.org/abs/2305.04760v2>. [Acceso: 28-ago-2025]
- [152] lowRISC, «Sonata™ v1.0 release», *lowRISC.org*, 2024. [En línea]. Disponible: <https://lowrisc.org/news/sonata-v1-0-release/>. [Acceso: 29-jul-2025]
- [153] lowRISC, «Sonata System: A Full Microcontroller System Utilizing the CHERIOT Ibex Core», *GitHub Repository*, 2025. [En línea]. Disponible: <https://github.com/lowRISC/sonata-system>. [Acceso: 29-jul-2025]
- [154] Microchip Technology Inc., «PolarFire® SoC FPGA Icycle Kit», *Microchip Technology*, 2025. [En línea]. Disponible: <https://www.microchip.com/en-us/development-tool/mpfs-icycle-kit-es>. [Acceso: 24-ago-2025]
- [155] Microchip Technology Inc., «Empowering Innovation», *Microchip Technology*, 2024. [En línea]. Disponible: <https://www.microchip.com/>. [Acceso: 24-ago-2025]
- [156] R. Barnwal, G. Noble, R. Kedia, S. Kala, y S. Nalesh, «Leveraging PolarFire SoC for Heartbeat Classification Using Convolutional Neural Network», en *2023 IEEE Asia Pacific Conference On Postgraduate Research In Microelectronics And Electronics (PRIMEAsia)*, 2023, pp. 54-55.
- [157] Microchip Technology Inc., «PolarFire® SoC FPGAs», *Microchip Technology*, 2024. [En línea]. Disponible: <https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas>. [Acceso: 24-ago-2025]
- [158] G. Sekar, N. Mouha, V. Velichkov, y B. Preneel, «Meet-in-the-Middle Attacks on Reduced-Round XTEA», 2011, pp. 250-267.
- [159] Fraunhofer IPMS, «AIRISC User Guide — AIRISC Base Core documentation», *Fraunhofer IMS GitHub Pages*, 2025. [En línea]. Disponible: https://fraunhofer-ims.github.io/airisc_core_complex/AIRISC_UserGuide_en.html#. [Acceso: 21-ago-2025]
- [160] Digilent Inc., «Genesys 2 Reference Manual - Digilent Reference», *Digilent Reference*, 2025. [En línea]. Disponible: <https://digilent.com/reference/programmable-logic/genesys-2/reference-manual>. [Acceso: 29-ago-2025]
- [161] Semiconductor Engineering, «PPA (Power, Performance, Area)», *Semiconductor Engineering Knowledge Center*, 2025. [En línea]. Disponible: https://semiengineering.com/knowledge_centers/eda-design/definitions/ppa/. [Acceso: 31-ago-2025]
- [162] E. Lazzeri, B. E. Forlin, G. Furano, M. Ottavi, y L. Cassano, «An Experimental

- Comparison of RISC-V Processors: Performance, Power, Area and Security - Special Session Paper», *Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT*, pp. 1-6, 2024.
- [163] Nordic Semiconductor ASA, «NEORV32», 2025. [En línea]. Disponible: <https://docs.nordicsemi.com/bundle/ncs-2.9.0/page/zephyr/boards/others/neorv32/doc/index.html>. [Acceso: 31-ago-2025]
- [164] F. D. El Ayachi, «Implementación ASIC de un microprocesador RISC-V para un SoC TSN», ULPGC, Las Palmas de Gran Canaria, 2023 [En línea]. Disponible: <http://hdl.handle.net/10553/120087>
- [165] R. Rodríguez Paredes, «Estudio e Implementación de extensiones dedicadas del juego de instrucciones RISC-V», Univ. Las Palmas de Gran Canaria, 2025.
- [166] C. B. Ciobanu *et al.*, «RISC-V Accelerators, Enablement and Applications for Automotive and Smart Home in the ISOLDE Project». Zenodo, 2025 [En línea]. Disponible: https://doi.org/10.1007/978-3-031-78380-7%5C_18