

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

Romarey Lourdes Santos Ruiz,
Instituto Universitario de Microelectrónica Aplicada
(IUMA)
Universidad de Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, España
rsantos@iuma.ulpgc.es

Pedro Pérez Carballo
Instituto Universitario de Microelectrónica Aplicada
(IUMA)
Universidad de Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, España
carballo@iuma.ulpgc.es

Abstract— The present work provides an analysis of the RISC-V Instruction Set Architecture (ISA), highlighting its open, modular, and royalty-free nature. It's presented as an alternative to proprietary architecture, fostering innovation, reducing costs, and enhancing security. The main part of the work is the study and experimentation of several RISC-V cores on FPGA devices, which provides a practical evaluation of their performance and efficiency. Specifically, the work evaluated NEORV32, AIRISC, CVA5 and CVA6 cores. NEORV32 presents ease of use and modularity. It can integrate external modules via the AXI4 bus, including the addition of a DMA controller for efficient data transfers. AIRISC is optimized for embedded applications, presents stable performance at 100 MHz with a low impact on the FPGA. It has a low resource occupancy, with a 5-stage pipeline architecture that uses AHB-Lite interfaces. CVA5 showed low resource utilization, operating at 100 MHz. Finally, CVA6 is a RV64 architecture that was implemented on a Genesys 2 prototyping board that includes an AMD Kintex 7 series FPGA and its logic synthesis was performed on an ASIC.

Keywords – RISC-V; NEORV32; AIRISC; CVA5; CVA6; FPGA; RISC-V toolchain; ISA extensions.

I. INTRODUCTION

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 [1] 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.

RISC-V is an open, modular and flexible emerging ISA that allows extensions to be included depending on design needs. This makes it easy to use in a wide range of designs, from very

simple IoT-oriented to more complex ones for high-performance computing.

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.

The instruction set is organized into three specifications: RV32I, RV64I, and RV128I, which define instruction sets of 32, 64, and 128, respectively, and vector sets of RV32V and RV64V.

In addition, RISC-V has a set of standardized extensions such as the M extension that supports the multiplication and division of whole numbers into, the C extension that includes the compressed instructions, the F and D extension for single and double precision floating-point instructions and the V extension for vector support.

RISC-V also 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.).

II. OBJECTIVES

The main objective of this work is to study the RISC-V architecture for different FPGA devices, including the following operational objectives:

1. Know the advantages, limitations and main characteristics of the ISA RISC-V, as well as define the main areas of application.
2. Know the ecosystem of software solutions necessary for application development.
3. Select different existing implementations and study the efficiency, occupancy, and power consumption suitability of such architecture in advanced FPGA devices.

- Obtain results and make comparisons of the platforms implemented under different optimization conditions.

III. BASIC RISC-V RV32I ARCHITECTURE

RV32I, has 40 instructions. Implement the operations needed to achieve basic functionality with 32-bit integers. It is encoded in a 32-bit instruction format. It includes instructions for addition, subtraction, bit-level logical operations, *load/store*, hops, and *branches*.

Fig. 1 shows the types of statement formats used and the fields into which they are organized:

- Type R: Instruction format for instructions that operate with registers
- Type I: format for immediate operations and load for loading data from memory to registers
- Type S: store format, which loads log data to memory
- SB Type: Branch Format
- Type U: Immediately superior format
- Type J: Non-conditional jump format

The fields of the instruction formats are as follows:

- Opcode: These are the bits that specify the instruction to be executed
- RD (Target Record): Specify the target record that will contain the result of the operation
- RS1 (source record 1): Specifying the first source record
- RS2 (Source 2/Immediate Record): Specifies the second source record or an immediate value
- func (function): these are different specific fields (available for some statements) and specify a variant of the statement

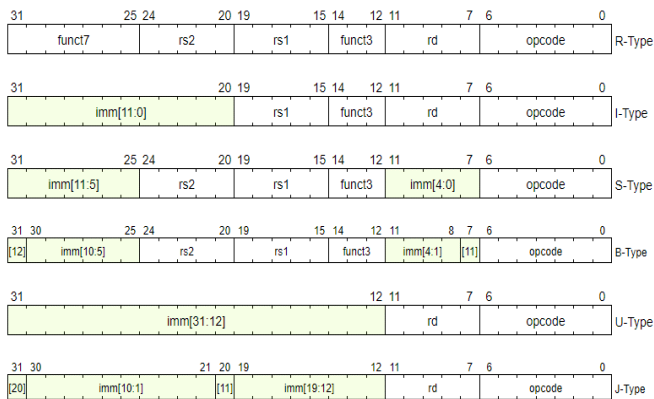


Figure 1. RISC-V instruction formats

RISC-V supports the following modes of operation for different privileges:

- User/Application Mode:** for user processes intended for the use of normal applications and operating systems.
- Supervisor Mode:** for kernel and/or device drivers, offer support for operating systems.
- Machine Mode:** For bootloader and/or *firmware*, it executes the most reliable code, has the highest

privileges, and is the only mandatory privilege level for a *RISC-V* hardware platform.

Fig. 2 shows an example of a program that sorts a set of data in RISC-V assembler.

```
# 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
```

Figure 2. Example of RISC-V program using RISC-V ABI

IV. RISC-V CORE STUDY

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 [2], and CVA5:** These are small cores optimized for low power consumption and with integrated peripherals, especially suited for FPGAs.
- CVA6 and Sargantana:** high-performance 64-bit cores, capable of supporting a full operating system.

Also, some commercial cores have been considered, including processors from SiFive, Synopsys ARC-V core and MicroBlaze-V from AMD. The study also takes into account some open-source platforms like X-HEEP and Cheshire that

support the integration of the RISC-V core in a complete SoC, in some cases with multiprocessing capabilities.

V. RISC-V CORE IMPLEMENTATIONS

To get a clear view of the capabilities offered by different RISC-V cores (RV32 and RV64 versions), 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.

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 (Fig. 3).

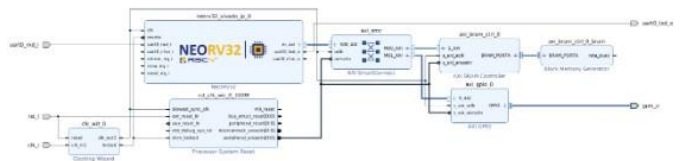


Figure 3. NEORV32 platform

The AIRISC [3] 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 (Fig. 4).

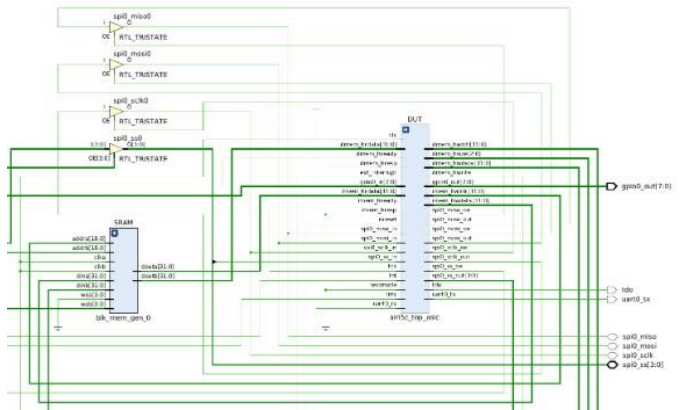


Figure 4. AIRISC core with memory program

The CVA5 [4] 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 so extensive as in the case of NEORV32, which puts it at a disadvantage for use in embedded systems (Fig. 5).

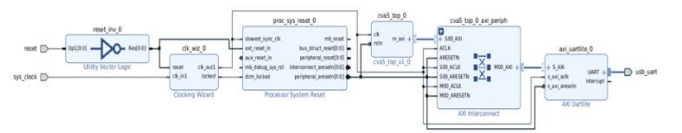


Figure 5. CVA5 basic platform

CVA6 [5] was implemented on a Genesys 2 prototyping board that includes a Kintex 7 series FPGA from AMD (Fig. 6). 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.

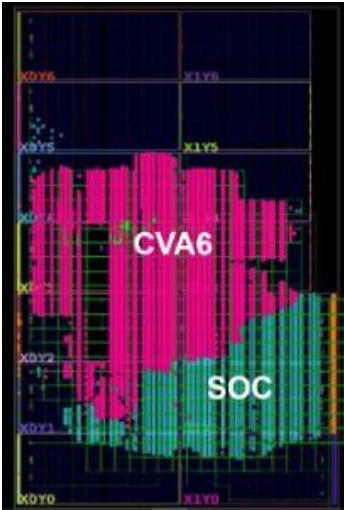


Figure 6. Layout of CVA6 SoC implemented in a Kintex device

Table I summarizes the resources used during the deployment of the various cores on AMD 7 Series FPGA devices. Data from MicroBlaze V provided by AMD has also been included as a comparison with the cores evaluated.

TABLE I. RESOURCES USED BY THE RISC-V CORES					
	LUTs	FFs	BRAM	DSP	Notes
NeoRV32	2.291	2.132	9.5	-	Artix
AIRISC	12.877	6.193	365	4	Artix
CVA5	3.625	1.929	3	5	Artix
CVA6	73.426	46.550	50	27	Kintex
MicroBlaze V: Microcontroller Preset	2.236	1.138	0	4	Artix*
MicroBlaze V: Real-time Preset	4.106	2.685	6	4	Artix*
MicroBlaze V: Application Preset	8.623	5.325	16	6	Artix*
MicroBlaze V: Maximum Frequency	1.227	826	0	0	Artix*
*Information for MicroBlaze V from AMD datasheet					

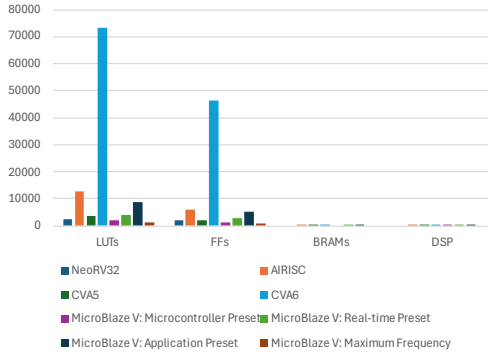


Figure 7. Global Resources utilization for the RISC-V cores

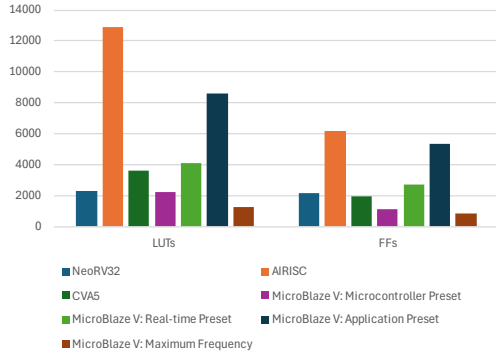


Figure 8. LUTs and FF used by the RISC-V cores (only RV32 cores)

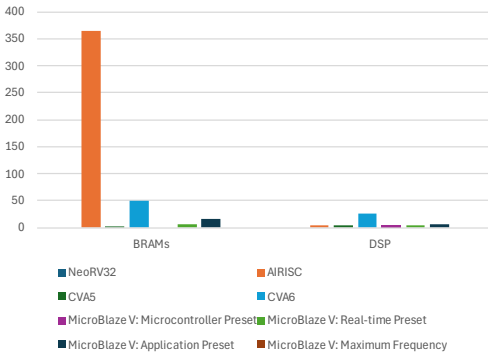


Figure 9. BRAMs and DSP blocks used by the RISC-V cores

Fig. 7 shows the big difference between 32-bit and 64-bit implementations in terms of LUTs and FFs. Fig. 8 considers only 32-bit implementations, showing that the AIRISC core requires more resources for its implementation. Finally, Fig. 9 highlights the strategy of using BRAM blocks for the implementation of local memory in the AIRISC processor. Fig. 10 and Table III shows the power of the cores. All cores are implemented using a clock of 100 MHz (Table II).

TABLE II. MAX. FREQUENCY OF THE CORES

	FPGA	Freq. (MHz)	Period (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

TABLE III. POWER OF THE EVALUATED CORES

	Total	Dynamic	Static	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

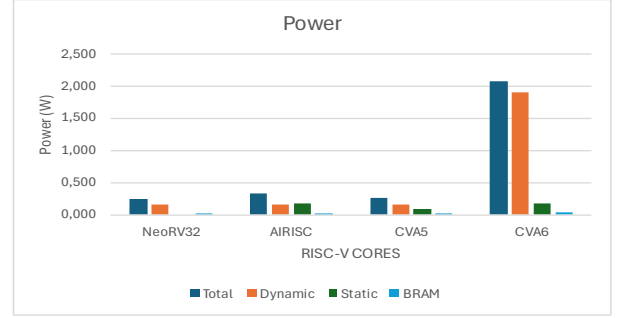


Figure 10. Power of the RISC-V Cores

VI. CONCLUSIONS AND FUTURE WORK

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 ones. 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..

VII. REFERENCES

- [1] RISC-V, «The RISC-V Instruction Set Manual Volume I: User-Level ISA V2.1», vol. I, 2016.
- [2] 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]
- [3] 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]
- [4] OpenHW Group, «CVA5: Application-Class 5-Stage RISC-V CPU», *GitHub*, 2025. [En línea]. Disponible: <https://github.com/openhwgroup/cva5>. [Acceso: 28-jul-2025]
- [5] 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]

