# Customized RISC-V In a Simple Game Console

Zdenek Prikryl, Pavel Snobl

Codasip

## Abstract

*RISC-V processors have found their way into products already. The openness of the RISC-V ISA, the strong ecosystem, and, more importantly, the ability to innovate through custom extensions have made this possible. Custom extensions can target any domain including game consoles. This paper will focus on the NES emulator and how it can be accelerated on RISC-V processors. Firstly, we will create a virtual platform with the standard RISC-V processor in a RV32IMCBZc configuration. Then, we will explore different instruction set extensions that help with performance. The virtual platform and the design space exploration will be done using Codasip Studio, EDA tools focused on (not only) RISC-V customization. The process contains software profiling, looking at the instruction set sequences, creating new instructions in the processor description language that allows regeneration of the programming and simulation tools, and implementing the processor in RTL. Once the performance of the virtual platform is at an acceptable level, we will perform PPA analysis to understand the impact of the added instructions in processor implementation*.

## Introduction

The idea of domain-specific processors or accelerators is not new, it has been here for a while. The process of development of such a processor can be a bit problematic if no or limited automation is used. In other words, if designers need to develop a new domain specific processor, they need a virtual platform, programming toolchain, RTL, and other tools to validate the ideas and impact of the accelerator. These items can be developed separately, but it is a time-consuming task that requires experienced teams. On top of that, they need to be carefully synchronized, and some testing or verification approach must be in place to check that all teams are doing the same thing, and that they interpret the specification correctly. It is obvious that the design space exploration is not as fast as it should be since all changes are made manually. Therefore, tools that automate the process of design space exploration are desperately needed. Codasip Studio [1] is designed to help with the design space exploration, automatic programming toolchain generation, virtual platform creation, and processor implementation, verification, or other tasks needed for the development of domain-specific processors. It uses architecture description language [2] called CodAL that describes the processor including its instruction set and microarchitecture at high-level abstraction level.

In this paper, we explore how adding custom instructions to a RISC-V processor [3] can improve the performance of an NES emulator application. Firstly, the off-the-shelf RISC-V processor and standard programming toolchain are used to see the starting point for the NES emulator. Then, a virtual platform and profiler is used to find hotspots, and new instructions are identified. These instructions are added to the vanilla RISC-V processor using Codasip Studio. Lastly, the whole platform, including the customized processor, is put into FPGA.

## NES and its Emulator

NES or Nintendo Entertainment System is an 8-bit video game console originally released by Nintendo in 1983 in Japan and in 1985 in the United States. It is built around the Ricoh 2A03/2A07 (NTSC/PAL versions) microprocessor, running at 1.79 MHz and 1.66 MHz respectively. The CPU is a variant of the MOS Technology 6502 with the BCD (Binary Coded Decimal) mode disabled, working with 2 KB of RAM. The graphics of the console are handled by a specialized PPU (Picture Processing Unit) chip with 2 KB of video RAM and a screen resolution of 256x240 pixels with up to 25 colors displayed at the same time. The sound is produced by another specialized chip, the APU (Audio Processing Unit), capable of outputting sound using a triangle wave channel, a white noise channel, a DPCM channel, and two pulse wave channels.

The MOS Technology 6502 [5] was designed as an 8-bit processor with a 16-bit memory address range and six registers available to the programmer. There are five 8-bit data and index registers. There is also a 16-bit program counter, and a 16-bit cycle count register (*CLK*), not visible to the programmer. It has 56 instructions and 13 addressing modes, determining where the operands of the instructions are located.

The memory space is divided into so-called pages, each 256 bytes long. The first two pages are special, serving as the so called *zero mode*, accessible via shorter instructions, and the *stack*, used for things like subroutine calls and accessible only using special push/pop and call/return instructions.

As the NES emulator, Infones [4] is used. It is one of the many available emulators out there. It runs on the standard x86 platform, and it is extendable to other platforms. The emulator is written in C++ and licensed under the Apache

Licence 2.0, enabling the modifications described in this paper.

## NES Accelerator

For this project, we chose the Codasip RISC-V processor L110, a member of the 100 family. It is a highly configurable and customizable embedded processor at the instruction set level and on memory subsystem. The configuration selected to start with is RV32IMCBZc with TCMs in which the code and data will be stored. In other words, the software is completely stored in TCMs to have as fast access to data and code as possible.

The processor is not enough to get everything working. We need to create a platform that besides the processor contains also peripherals and other important blocks, such as display or input controller. Such a platform was created in Codasip Studio, so it generates not only RTL, but also a virtual prototype that can be used to identify new instructions.

So, the first step is to profile the overall performance on the virtual platform and find hot stops. Codasip profiler was used to get the data. A homegrown game, Nova the squirrel [6], that is licensed under GNU General Public License, is used to measure the data. The outcome showed that there are three functions that take most of the execution time per frame. The functions are *InfoNES_LoadFrame* that takes 27% of the execution time, *InfoNES_DrawLine* that takes about 7%, and then the *K6502_Step* that takes 52% of the execution time. The *InfoNES_LoadFrame* is responsible for writing the data to the graphical RAM on TFT display, so there is nothing much to optimize, since is it given by the hardware that is used. The *InfoNES_DrawLine* is responsible for line rendering and accessing different tables with sprites, background, etc. It tightly cooperates with PPU and APU emulators. Since the time spent is not significant, this function is not optimized. The *K6502_Step* is responsible for execution of individual instructions of the game and takes a significant amount of time. Therefore, it is the best candidate for optimization.

*K6520_Step* is a simple switch that depends on the opcode of an instruction and emulates the instruction. During this it accesses the processor registers modelled as global variables and manipulates data stored in RAM, PPU, and APU. The RISC-V extension that was developed implements all the instructions present in the switch. Some additional instructions were added as well that decouple the CPU, PPU and APU, since PPU and APU are still emulated in software.

## Results

The graph in Fig. 1. shows the result in terms of performance improvements per frame, energy consumption per frame, and area addition. The comparison is done relative to the L110 without customization. L110X marks the enhanced L110. As we can see, the performance gain per frame is 28%, while the energy consumption per frame dropped by 13% even though the design is 18% bigger. The reason why the energy consumption is lower is the fact that we need less cycles to do the computation and clock gating that Codasip Studio inserts. Most of the area increase is in the register files that are used for *stack* and *zero mode* pages data storage.

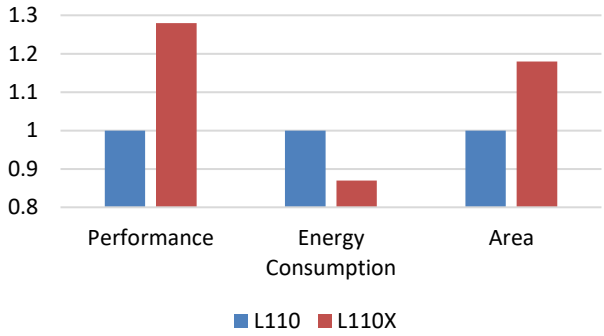


Fig. 1. Results

## Summary

This paper shows the results of the RISC-V extensions for a game console emulator. It uses Codasip RISC-V processor as the base for the extension. Then several custom instructions are added that first emulate some parts of the MOS Technology 6502 processor, and second, they help with the overall execution of the emulator. The RISC-V extensions were designed in the CodAL language, serving as an input to Codasip Studio. Codasip Studio generates all needed outputs including but not limited to an LLVM based C/C++ compiler that is fully aware of the new instructions, profiler and debugger, and RTL that can be used for synthesis or FPGA. The customized RISC-V processor is then put first into a virtual platform. Then, the whole platform is moved to FPGA. The extensions bring a 28% performance boost for each frame, and energy consumption is reduced by 13% per frame even though the area is increased by 18%. One of the most important metrics is the time needed for the extension, which was five days only to get from the idea to a working prototype on FPGA.

## References

[1] Codasip website: https://www.codasip.com

[2] Prabhat Mishra, Nikil Dutt, “Processor Description Languages”, 2011, ISBN: 9780080558370

[3] RISC-V website: https://riscv.org/

[4] NES emulator website: https://github.com/jay-kumogata/InfoNES

[5] 6502 Reference: https://www.nesdev.org/obelisk-6502-guide/reference.html

[6] Nova the squirrel game webpage: https://github.com/NovaSquirrel/NovaTheSquirrel