Creating a RISC-V heterogeneous SoC

Matthew Knight
Department of Computer Science
University of Warwick

28 March 2023

Abstract

This project aims to design and analyse heterogeneous RISC-V SoCs implemented in FPGAs. Many devices have varying performance needs during use, and always aim to minimise power usage. Heterogeneous architectures aim to provide a solution that delivers high performance and high efficiency by combining highly efficient CPU cores (S cores) and highly powerful CPU cores (B cores) in a single processor. When performing background, low priority tasks the S cores are used and the B cores are idle/off, resulting in low power usage. When there are demanding, high priority tasks the B and S cores are used, resulting in higher performance than if just the S cores existed.

The CPU cores implement the RISC-V ISA, an open-standard ISA with support for Linux. There is minimal research into general purpose RISC-V processors implementing a heterogeneous architecture, so the project will perform some novel research. The S and B cores will be designed using the RocketChip generator, that generates HDL for a configurable RISC-V SoC.

The designed heterogeneous SoC has been be benchmarked and compared to homogenous designs in terms of size, power draw and performance. Based on these metrics, heterogeneous RISC-V SoCs implemented in FPGAs have limited benefits over homogenous, with a B and S processor providing small decreases in power and area usage for significantly less performance compared to a two B processor, but giving much increased performance for small power and area increases compared to a two S processor.

Contents

Chapter 1

Introduction

The aim of this project is to design a heterogeneous system on a chip (SoC) using the RISC-V open standard ISA[8][3]. Once designed, the SoC should be implementable on an FPGA and be able to execute bare-metal C code and assembly with SMP, allowing all cores to execute code at the same time.

1.1 Motivation

Designing a processor is a complex challenge. They need to be power efficient to reduce the cost of running the system, as well as providing adequate processing power when required. Often one is sacrificed to improve the other: for instance, energy efficiency is often reduced due in designs of large processors containing multiple cores. Due to the increased size and complexity, more power has to be supplied to achieve the same performance as a smaller processor.

A heterogeneous CPU attempts to solve this issue by combining dissimilar core designs; often a powerful core, or p-core, and an efficient core, or e-core. When only light processing is required, the p-core can be effectively shutdown and the e-core will do all processing, resulting in less power used. For heavy processing, the p-core is then used to increase peak performance. Depending on the exact implementation, the p-core can be used either individually or in

tandem with the e-core, but both result in greater performance than just the e-core.

This has been used extensively in many mobile devices, and increasingly in larger devices like laptops. The Apple M2 processor[1] is one such example, implementing 4 high-performance cores and 4 energy-efficient cores. These are arranged in a hybrid configuration similar to ARM DynamIQ and big.LITTLE[5], allowing cores to dynamically be assigned work that best fits them. In Apple's design, both cores are capable of executing the same code, with the performance core benefitting from much increased cache and other proprietary changes to increase it's speed over the efficiency core.

1.2 Related work

1.2.1 A RISC-V Heterogeneous SoC for Embedded Devices[7]

This project is ongoing, and presents work designing a RV64 (RISC-V 64-bit) host core that offloads tasks to a PMCA (Programmable Many Core Accelerator) made from RV32 (RISC-V 32-bit) cores, which implement extensions for machine learning and discrete signal processing. The suggested use-case for the SoC is in IoT applications and programmable embedded devices. The host core is Linux compatible, and offers a full OS that acts as a platform for programs that run on the PMCA. The use of a large RV64 core to allow a full Linux OS to run on the SoC provides a huge amount of flexibility to the programmer, as the OS implements features like CLI, memory virtualisation, networking and more that allow programs to be written much more generally than embedded software running without an OS. However, this usage of a full Linux OS could be considered excessive for the use-case. An embedded Linux OS would have a lower overhead due to the reduced services it offers, which is very beneficial in an embedded environment where efficiency is highly important. Unfortunately, no data is provided about the processing power or energy usage of the design.

1.2.2 Muntjac multicore RV64 processor[4]

Muntjac is an SoC generator comprising of multiple components, that can be used to produce a Linux capable SoC. There is only one type of core in the system, RV64, but the SoC can be multicore. The project report is dedicated to the design of the core and cache as opposed to usage in any devices, as the purpose is to provide an easily understood and extensible platform for specialised designs. This is excellently done - the project is very well presented and uses multiple open-standards like TileLink[6] to increase the ease of working with it. It would be possible for this project to be extended for a heterogeneous SoC design, but it appears that this has not yet been done in any public works that extend the project.

1.3 Objectives

The overall aim is design a heterogeneous SoC containing two types of RISC-V core that can execute bare-metal C code and assembly with SMP. Objectives have been labelled according to the MoSCoW method[2] to indicate their importance and expected completion.

1. Design a heterogeneous RISC-V SoC containing 2 dissimilar cores, each capable of executing at least the RV64 instruction set (Must).
2. Execute assembly instructions on both cores when implemented in an SoC on an FPGA (Must).
3. Execute bare-metal code on both cores when implemented in an SoC on an FPGA (Should).
4. Measure the performance of the SoC for comparison against single-core designs. (Should)
5. Run embedded Linux on the SoC and connect to it via serial or SSH (Could).
6. Allow processes to execute on both cores inside of embedded Linux (Could).
7. Intelligently select which core the process will run on, depending on factors such as process priority and resource usage (Won't).

Figure 1.1: Objectives for the project

Chapter 2

Background

Your literature review goes here. **?**] discusses the advantages of functional programming by exploring *e.g.* lists, trees, and noughts and crosses in a functional programming language.

Chapter 3

Design

In this chapter, we describe the overall design of our solution to the problem identified in Chapter 1, building on work described in Chapter 2.

Chapter 4

Implementation

In this chapter, we describe the implementation of the design we described in Chapter 3. You should **not** describe every line of code in your implementation. Instead, you should focus on the interesting aspects of the implementation: that is, the most challenging parts that would not be obvious to an average Computer Scientist. Include diagrams, short code snippets, etc. for illustration.

Chapter 5

Evaluation

Describe the approaches you have used to evaluate that the solution you have designed in Chapter 3 and executed in Chapter 4 actually solves the problem identified in Chapter 1.

While you can discuss unit testing etc. you have carried here a little bit, that is the minimum. You should present data here and discuss that. This might include *e.g.* performance data you have obtained from benchmarks, survey results, or application telemetry / analytics. Tables and graphs displaying this data are good.

Chapter 6

Conclusions

The project is a success. Summarise what you have done and accomplished.

6.1 Future work

Suggest what projects might follow up on this. The suggestions here should **not** be small improvements to what you have done, but more substantial work that can now be done thanks to the work you have done or research questions that have resulted from your work.

Bibliography

[1] Apple, . Apple unveils m2. https://www.apple.com/uk/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/, 2022.

[2] Dai Clegg, Richard Barker. *Case Method Fast-Track: A RAD Approach*. Addison-Wesley, 1994. ISBN 978-0-201-62432-8.

[3] Editors Andrew Waterman, Krste Asanovic & RISC-V InternationalJohn Hauser, December 2021. The risc-v instruction set manual, volume ii: Privileged architecture, document version 20211203. https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf.

[4] Guo, Xuan & Bates, Daniel & Mullins, Robert & Bradbury, Alex. Muntjac multicore RV64 processor: introduction and microarchitectural guide. Technical Report UCAM-CL-TR-972, University of Cambridge, Computer Laboratory, June 2022. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-972.pdf.

[5] Ltd., ARM. big.little processing. https://web.archive.org/web/20121022055646/http://www.arm.com/products/processors/technologies/bigLITTLEprocessing.php, 2011.

[6] SiFive, . Sifive tilelink specification. https://sifive.cdn.prismic.io/sifive/7bef6f5c-ed3a-4712-866a-1a2e0c6b7b13_tilelink_spec_1.8.1.pdf.

[7] Valente, Luca & Sinigaglia, Mattia & Tortorella, Yvan & Rossi, Davide & Benini, Luca. A risc-v heterogeneous soc for embedded devices. https://open-src-soc.org/2022-05/media/posters/4th-RISC-V-Meeting-2022-05-03-Luca-Valente-poster-abstract.pdf.

[8] Waterman, Editors Andrew & RISC-V FoundationKrste Asanovic, December 2019. The risc-v instruction set manual, volume i: User-level isa, document version 20191213. https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf.