Design and Analysis of a Heterogeneous RISC-V SoC on FPGAs

Matthew Knight
Department of Computer Science
University of Warwick

Supervised by Dr Eduardo Wächter

15 April 2023

THE UNIVERSITY OF WARWICK

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 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 and Research

2.1 RISC-V

2.1.1 ISAs and RISC vs CISC

An ISA is an instruction set architecture, and is the formal definition for what an abstract model of a CPU. The ISA defines the instructions, registers, communication standards and other features of a CPU in order to allow a person with the ISA to design a physical CPU that would implement the abstract model. CPUs that implement the same ISA (or supersets of an ISA) are able to execute the same code, meaning programs written for a CPU can be run on a different CPU if they share the same ISA. This is how a single compiled program is able to be run on so many different systems, and is incredibly important for modern devices, where there is a huge range of processors available.

Many modern ISAs are based on older ISAs that have been extended to allow for new instructions, registers, communication protocols, etc, while maintaining support for previous ISAs. This is typically called CISC (Complex Instruction Set Computers), where there are a very large amount of instructions the processor can execute. The backwards compatibility can be very useful, and having explicit instructions for many tasks can mean the tasks are completed more efficiently than if multiple smaller instructions were executed. Having large instructions can also reduce program sizes, due to the increased operations per instruction, which is very beneficial, especially in small systems.

There are several issues with CISC however, such as increased complexity in the CPU as the amount of instructions increases, leading to rising costs of design and manufacture, and can force the physical size of the CPU to be larger to accommodate the extra logic space for larger instructions. More complex designs will also make future development harder if backwards compatibility is to be maintained.

The alternative to CISC is RISC (Reduced Instruction Set Computer). RISC aims to reduce the number of instructions the CPU requires in order to keep the design as simple as possible. This allows the design to be done much more easily, as well as allowing future extensions to be easier. Reduced complexity can also allow for greater optimisations in what instructions are in the ISA, with the aim of achieving greater performance than CISC by executing more instructions at much greater speed. There are drawbacks however, with increased program size and possible reduction in speed of execution for multiple operations that could've been completed in a single instruction using CISC.

2.1.2 The RISC-V ISA

RISC-V is a new, modern, open-standard ISA that follows RISC concepts. The ISA does not specify a single instruction set, but multiple by having three different width instruction sets with different base instructions. 32-bit RISC-V (RV32I) is focussed on embedded and personal systems, 64-bit RISC-V (RV64I) for personal and server systems and 128-bit RISC-V (RV128I) for server and high-performance compute systems. RV32 and RV64 have embedded versions, RV32E and RV64E, that have reduced registers and instructions.

Extensions

At base, these implement only integer addition/subtraction. The ISA contains extensions that add more functionality to the CPUs, such as support for multiplication/division, floating point operations IEEE-754, compressed instructions, atomics, etc. An instruction set implementing these extensions is referred to by adding the initial of the extension to the name, so a 64-bit CPU with integer addition/subtraction, multiplication/division and compressed instructions would be 'RV64IMC'. Custom extensions can also be written, allowing for a great deal of customisation in RISC-V CPU designs.

There are 30 official extensions to the base instruction sets. The most popular are listed below.

M Integer multiplication and division

A Atomic instructions

F Single-precision floating point

D Double-precision floating point

Zicsr Control and Status Register (CSR)

Zifencei Instruction-Fetch Fence

G Short for `IMAFDZicsrZifencei` as a 'general-purpose' CPU, e.g. RV64G

C Compressed instructions, 16-bit instead of 32/64

Others Quad-precision floating point, bit manipulation, vector, misaligned atomics, etc

2.2 Processor Design

Processor performance can be measured in multiple ways, with the main metrics being speed at executing tasks, cost of manufacturing, physical size and energy consumption. These are implicitly linked - a processor with larger physical size will likely draw a larger amount of energy, contain more logic and so be faster at executing tasks and cost more to manufacture. The aim of processor design is to maximise the speed of task execution, while minimising the rest. The balance between these is what gives rise to different processor designs, as a mobile device is much more constrained in power usage than a large server and will require a different processor.

2.2.1 Power reduction in processors

Maximising task execution while minimising energy consumption can be balanced in a multitude of ways. Power consumption in CPUs can be split into two sections: switching power and leakage power. Switching power is the power used by CMOS (transistor) gates as they change states, and varies on CPU activity. Leakage power is the power lost through slow leakage current flow through transistors in the 'off' state, and has increased as transistor sizes decrease and the boundary that must be crossed reduces.

Dynamic Voltage Frequency Scaling (DVFS)

CPUs contain clocks, which synchronises the components inside the CPU as they change state and execute instructions. Increasing the clock speed will result in an increase to the number of instructions executed per second, directly increasing the performance of the CPU. However, this directly increases the amount of switching power the CPU draws, given by equation 2.1.

$$P_{switching} = \alpha \cdot C \cdot V^2 \cdot f \alpha = activity factor, proportion of transistors that switch every cycle C = ca \tag{2.1}$$

In addition to this, increased clock speeds often require increased voltage in order to increase the current flow through transistors to activate them in the reduced time frame from shorter clock periods. Therefore, reducing the clock speed and the voltage will result in a quadratic reduction in switching power for a linear reduction in performance and if low performance is needed, the power usage can be significantly reduced. CPUs implementing dynamic voltage frequency scaling are able to adjust their voltage and frequency during runtime dependant on the current tasks being run. This allows them to dramatically reduce power consumption during periods of low CPU utilisation, while still being able to provide good performance with high CPU utilisation.

However, this makes some requirements of the CPU. There must be software that allows the CPU to estimate it's current utilisation, and this has to be run frequently in order to have a fast response and increase clock speed when a demanding task is run. This can take up valuable CPU time, and requires some form of scheduling to repeatedly switch to and from this task. The CPU must also implement hardware that allows it to change the clock speed and voltage, increasing the physical size of the CPU and increasing manufacturing costs.

Clock Gating

Clock gating is another technique for reducing power consumption in a CPU. When a section of the CPU is unused for a number of cycles, the clock signal to that section is removed. This disconnect reduces the switching power, as no transistor switching will occur in that section of the CPU without the clock, as well as reducing the capacitance seen by the clock generator.

2.2.2 Heterogeneous Designs

Another solution for power reduction in processors is heterogeneous designs. Multicore processors contain multiple CPUs in order to increase the potential performance in parallel computing tasks. A heterogeneous design contains multiple, different CPU designs instead of all CPUs being copies of each other as in homogeneous designs.

By having multiple different types of CPU, the processor can increase efficiency and reduce power consumption by intelligently selecting which CPU to run a task on. For example, a processor containing a small (S) core and a big (B) core could be running in a mobile phone. For the vast majority of the time, the mobile phone is not in use and only runs tasks such as checking for texts, emails, etc. These tasks can be run on the S core, with the B core fully disabled/clock gated, reducing the power consumption to that of the S core + B core leakage current. This power consumption should be less than that of the B core performing the tasks, in order for the processor to actually provide efficiency increases over a B core homogeneous design.

When the mobile device is actively in use, like video playback or mobile games, the B core would be used to provide greater performance than the S core, or both used for parallel tasks. This allows a heterogeneous S+B design to provide better energy efficiency and better performance than a homogeneous design with one B core.

There are some drawbacks to heterogeneous designs. In order to properly utilise the differences between the cores, a scheduler must be customised for the exact processor as other designs with different CPUs will need to switch which task is run where at different stages. A heterogeneous system with a medium core and a big core will be able to use the smaller core for more intensive tasks than a heterogeneous system with a small core and a big core, thus requiring a scheduler with different parameters.

The physical size and complexity of the processor will also be increased compared to a single core design. This increases the cost of design and manufacturing, another drawback compared to homogeneous designs.

Accelerator-style Heterogeneous Designs

Some heterogeneous designs do not attempt to pair types of general purpose CPUs, but instead have a host CPU type and accelerator CPU type. The host CPU explicitly schedules tasks for the accelerator CPU, which is typically optimised explicitly for a certain task to increase the performance and efficiency in completing it.

2.3 Related work

2.3.1 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.

2.3.2 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.

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`.