# Creating a RISC-V heterogeneous CPU architecture

Progress report

**Matthew Knight**

Department of Computer Science

University of Warwick

# 1 Introduction

The aim of this project is to design a heterogeneous SoC using the RISC-V open source core designs. Once designed, the SoC should be implementable on an FPGA and be able to execute bare-metal C code and assembly with SMP, demonstrating that all cores in the SoC can be used at the same time.

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. ARM released the big.LITTLE in 2011[2], a mobile heterogeneous architecture. The architecture provides the low power usage needed in mobile devices the majority of the time when idle, only running tasks like checking for new messages. It also provides the high performance that can be demanded from phones when used to browse the internet, play mobile games, etc. Most big.LITTLE designs use HMP (heterogeneous multiprocessing) where all cores are available to have processes assigned to them at all times. The alternatives to this are: clustered switching, where either the big cores or the LITTLE cores are in use, and in-kernel switching, where big and LITTLE cores are paired to form a virtual core and only one performs the tasks assigned to the virtual core at any one time.

## 1.1 Related work

### 1.1.1 A RISC-V Heterogeneous SoC for Embedded Devices

A 64-bit RISC-V host core that offloads tasks to a PMCA (Programmable Many-Core Accelerator) made from 8 32-bit RISC-V cores. Implements two types of RISC-V core in a single SoC, but one is used as an accelerator for the other as opposed to them being of equal standing.

### 1.1.2 Muntjac multicore RV64 processor

Homogeneous RV64GC multicore. Linux-capable base to enable specialised designs. Implemented on Xilinx Kintex 7. Aims to be easily understood and extendable - a great starting point for development of specialised designs. Limited to only one type of core in base repository, but contains interconnect for other designs.

### 1.1.3 BlackParrot[1]

Homogeneous RV64G multicore. Linux-capable general purpose processor. Implemented on an IC at 12nm. A general purpose RISC-V multicore. Implicitly allows heterogeneous CPU by hiding the multicore components beneath an ISA layer and allows easy extensions by keeping the design tiny, modular and friendly. Heterogeneous designs are limited to those that share the exact same instruction set, so wouldn't allow RV32 and RV64 in a single processor.

## 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 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 instructions on both cores when implemented in an SoC on an FPGA (Must).

3. Measure the performance of the SoC for comparison against single-core designs. (Should)

4. Run embedded Linux on the SoC and connect to it via serial or SSH (Could).

5. Allow processes to execute on both cores inside of embedded Linux (Could).

6. Intelligently select which core the process will run on, depending on factors such as process priority and resource usage (Won't).

# 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.

# 3 Progress

Summarise the progress you have made so far. You can cross-reference other sections (Section 2).

# 4 Project management

Include a timetable (in 2 week chunks) for the remainder of the academic year, up until the submission deadline.

# References

[1] M. W. D. C. J. S. D. P. G. C. Z. Z. A. S. C. B. V. T. G. A. J. J. M. O. M. B. T. D. Petrisko, F. Gilani. Blackparrot: An agile open source risc-v multicore for accelerator socs. *IEEE Micro*, 2020.

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